TEKNOFEST

HAVACILIK, UZAY VE TEKNOLOJİ FESTİVALİ

ÇİP TASARIM YARIŞMASI

2024

SAYISAL İŞLEMCİ TASARIM KATEGORİSİ

DETAY TASARIM RAPORU

TAKIM ADI

YILDIRIM MİKROTEK

PROJE ADI

Oniks Core

BAŞVURU ID

292814

İçindekiler

SEMBOLLER LİSTESİ

$	Önbellek
V$	Veri Önbelleği
B$	Buyruk Önbelleği

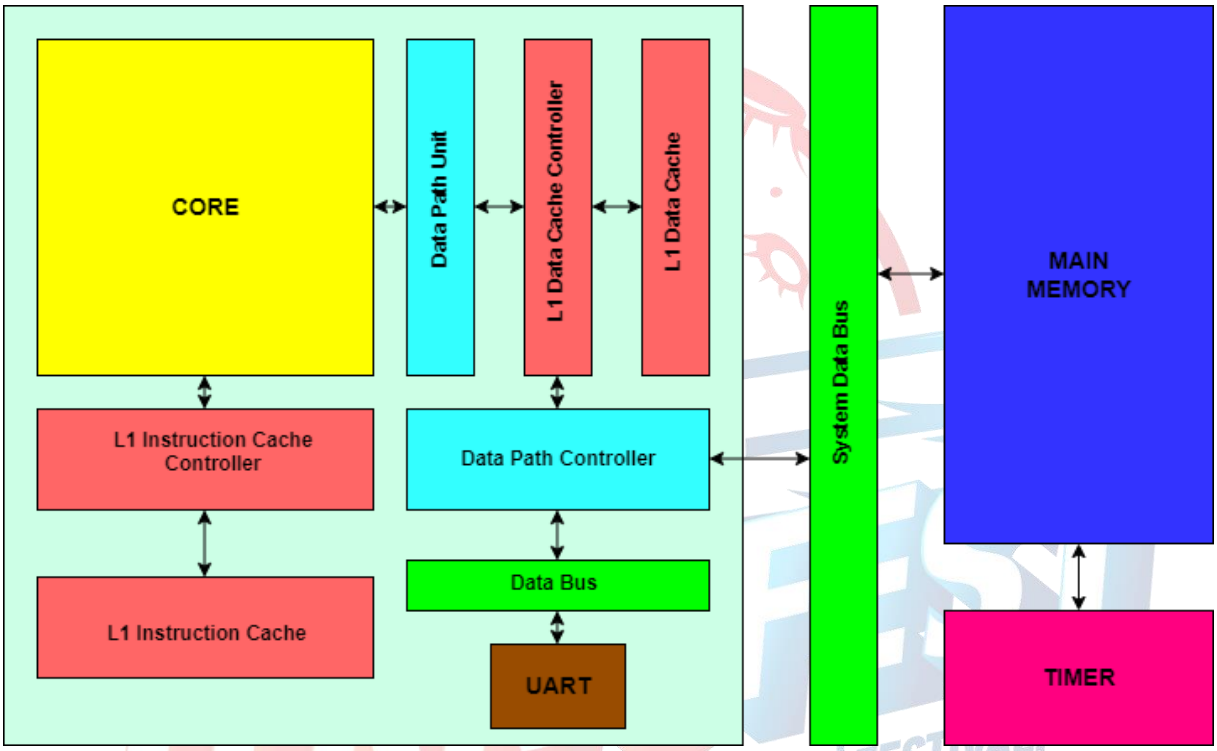
KISALTMALAR LİSTESİ

ASIC	Application Specific Integrated Circuit
FPGA	Field Programmable Gate Array
SRAM	Static Random Access Memory
UART	Universal Asynchronous Receiver-Transmitter
PDK	Process Design Kit
PC	Program Counter
DDB	Denetim Durum Birimi
CSR	Control and Status Register

1. TEMEL TASARIM ÖZETİ

Bu proje kapsamında, şartnamede verilen isterler doğrultusunda RV32IMAFB_Zicsr buyruk kümesi mimarisine sahip RISC-V tabanlı sıralı yürütüm (in-order) yapan bir işlemci tasarlanmış olup bu rapor, işlemcinin mimari tanımlamasını, tasarımını, çip tasarım akışını ve FPGA ortamındaki gerçekleme ve doğrulama akışını içermektedir. İşlemci tasarım, test ve doğrulamasına ait tüm kodlar ve dosyalar https://github.com/AYoldass/Yildirim-Mikrotek

adresinde bulunabilir. **Şekil 1**'de ise sistemin genel yapısı gösterilmiştir. Sistemin genel blok diagramı aşağıda verilmiştir.



Şekil 1

Gerçekleştirilen Çekirdek, (1) Getir1 (fetch1),(2) Getir2 (fetch2), (3) Çöz (decode), (4) Yürüt (execute), (5) Geri Yaz (write-back) olmak üzere 5 aşamalı boru hattından oluşmaktadır. Proje isterleri kapsamında bir seviyeli bir önbellek yapısı oluşturulmuştur. Bellek sisteminde veri ve buyruk için 1'e 3 oranında önbellekler kullanılmıştır. İşlemcinin çevre birimleri ile haberleşmesi ana bellek denetleyicisine ek olarak ön tasarım raporunda değişikliğe gidilerek AXI yerine Tilelink [1] arayüzü ile gerçekleştirilmiştir. İşlemci, proje isteri kapsamında UART protokolünü desteklemektedir.

ÖZELLİK	ONİKS CORE
BUYRUK KÜMESİ	RV32IMAFB_Zicsr
SENTEZLENEBİLİR	VERILOG RTL
FPGA	Zybo Z7 - Xilinx Basys3
VERİYOLU	Tilelink
BORU HATTI	5
VERİ ÖNBELLEĞİ	1KB SRAM
BUYRUK ÖNBELLEĞİ	3KB SRAM
ÇEVRE BİRİMLERİ	UART
CoreMark SKORU	427 iterasyon/saniye
CoreMark / MHz	3.24

Çipin serimini gerçekleştirilmesi için Synopsy[2] ile sağlanacaktır. Çip seriminde önbellekler SRAM[3] tabanlı bellek olarak gerçeklenecektir.

Sistemin işlevselliği, simülasyon ortamında çeşitli simülatörlerde farklı testler yürütülerek doğrulanmıştır. Bu testlere ek olarak çeşitli senaryoları test eden mikro programlar yazılmış ve çalıştırılmıştır.

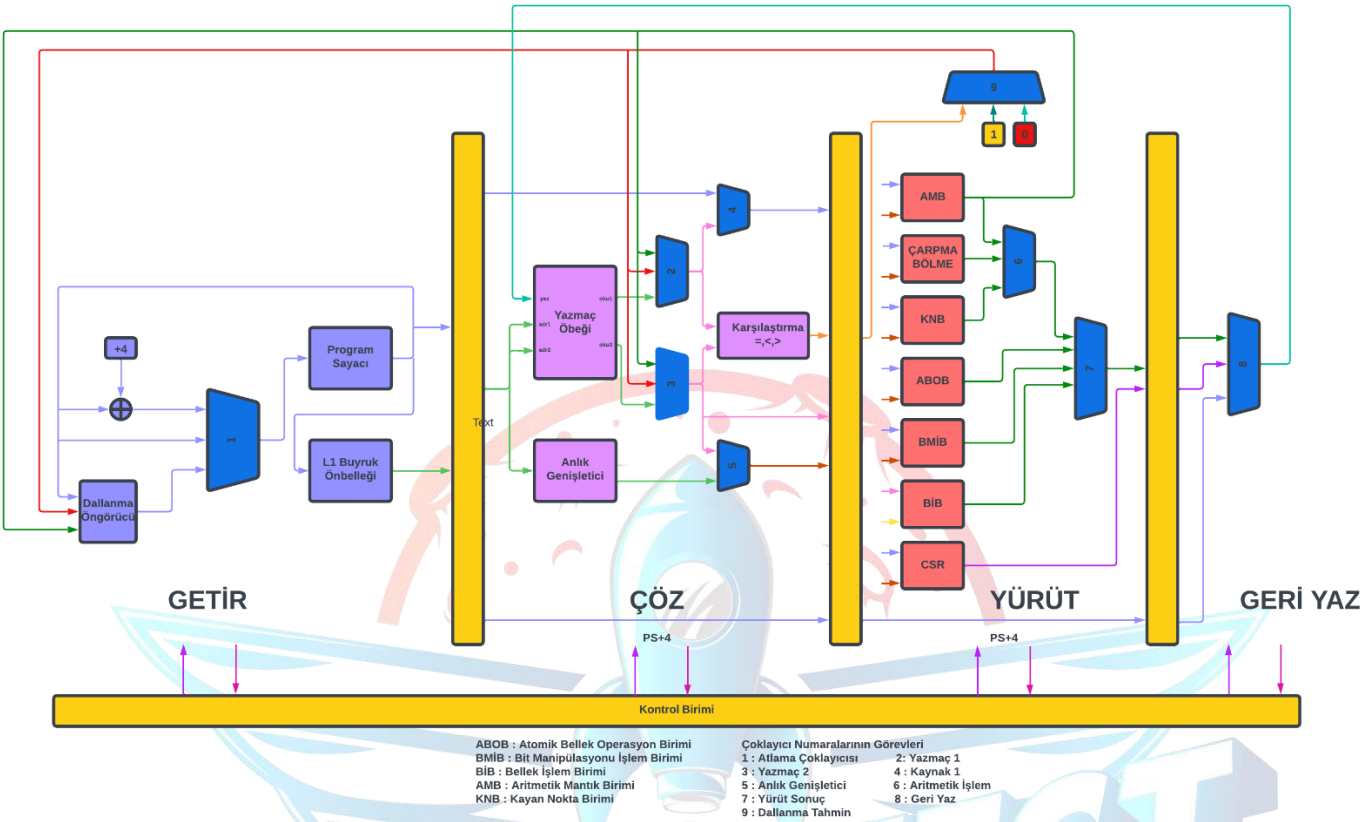
Verilen buyruk isterlerini gerçekleştirebilmek için çarpma işlemi açık kaynaklı multipler generator kullanılarak gerçekleştirilmiştir. Çarpma için kullanılan algoritma Modified Booth Dadda algoritması olup. Bölme işlemi için bit tabanlı yeniden kaydetmeli bölme algoritması kullanılmıştır ve 18 çevrimde gerçekleşmektedir. Performansın arttırılması için yürüt aşamasında dallanma öngörücü (-ing branch predictor) kullanılmıştır. Kayan nokta biriminde toplama çıkarma işlemler için Carry-Select Adder [4], Booth Çarpma Algoritması[5] ve Newton-Raphson [6] bölme yöntemleri tercih edilmiştir. Ayrıca işlemcimiz atomik bellek işlemleri(-ing Atomic Memory Operation), bit-manipülasyonu ve kontrol durum yazmaçları(-ing Control and Status Register) buyruk tiplerini de desteklemektedir.

2. PROJE MEVCUT DURUM DEĞERLENDİRMESİ

Proje kapsamında geliştirilen işlemci çekirdeğimzin son durumunda ön tasarım raporunda da belirtilen atomik bellek operasyonu buyrukları dışında bütün isterleri karşılamaktadır. Atomik bellek operasyonu buyruklarında bazı bölümler hatalı çalıştığı tespit edildi ve o kısım tekrar düzenlenmektedir. ÖTR'den farklı olarak boru hattı aşama sayısını 5 aşamaya çıkarma kararı alındı bunun sebebini Getir başlığı altında detaylı açıklanmıştır. Ayrıca önbelleklerin tasarımında modern işlemciler incelenmiş ve 3'e 1 oranı uygulanmıştır sonuç olarak önbelleklerde 3KB buyruk önbelleği, 1KB veri önbelleği olacak şekilde düzenlenmiştir. Ayrıca veri yolu olarak ÖTR'de AXI tasarlayacağımızı belirtmiştik ancak donanımda kazanç sağlama ve performansı arttırmak için Tilelink tercih edilmiştir bu kısımla alakalı daha detaylı bilgi çevresel birimler başlığı altında verilmiştir. RV32IMAFB_Zicsr buyruk kümesi mimarisini (instruction set architecture) destekleyen işlemcimiz atomik buyruklar dışında kalan bütün testleri başarıyla geçmiştir. Projenin ilerlemesi sırasında ön tasarım raporunda belirtilenden farklı şekilde uygulanan bazı optimizasyona yönelik tasarım kararları alınmıştır. Buyruk önbelleği ve veri önbelleği ise performans artışı için SRAM tabanlı gerçeklenmiştir. İşlemcimizde atomik bellek işlemleri buyruk tipleri iyileştirilecek ve Synopsys tamamlanacaktır.

3. PROJE DETAY TASARIMI

Şekil 2'de proje kapsamında gerçeklenen işlemcinin detaylı boru hattı şeması görülebilir.



Şekil 2. Getir-Çöz-Yürüt

3.1.ÇEKİRDEK TASARIMI

3.1.1 Boru Hattı

Proje kapsamında geliştirilen sistemin detaylı boru hattı şeması **Şekil 2**'de gösterilmektedir. Boru hattı ve sistem genelinde herhangi bir açık kaynak tasarım kullanılmamış ve çekirdek tamamen özgün bir şekilde gerçekleştirilmiştir. Sistemde 5 aşamalı sıralı yürütme yapan işlemci çekirdeği, işlemci çekirdeğinin Getir aşamasıyla haberleşen 3 KB boyutunda 1. seviye buyruk önbelleği (L1I$), yürüt aşamasındaki bellek işlem birimiyle haberleşen 1 KB boyutunda 1. seviye veri önbelleği (L1V$) bulunmaktadır, çekirdek dışı birimler (çevre birimleri ve bellek hiyerarşisi) ile çekirdeğin haberleşmesini yönlendirmek için Tilelink master eklenmiş ve çip dışıyla haberleşmeyi gerçekleştirmek için UART çevre birimi yarışma şartnamesinde istendiği şekilde özelleştirilerek sisteme entegre edilmiştir. Sistem mimarisinin yazmaç seviyesindeki kodlaması Verilog-2005 donanım tasarlama dili kullanılarak gerçekleştirilmiştir. İşlemci çekirdeği, RV32IMAFB_Zicsr özelleştirilmiş buyruk kümesi mimarisini desteklemektedir.

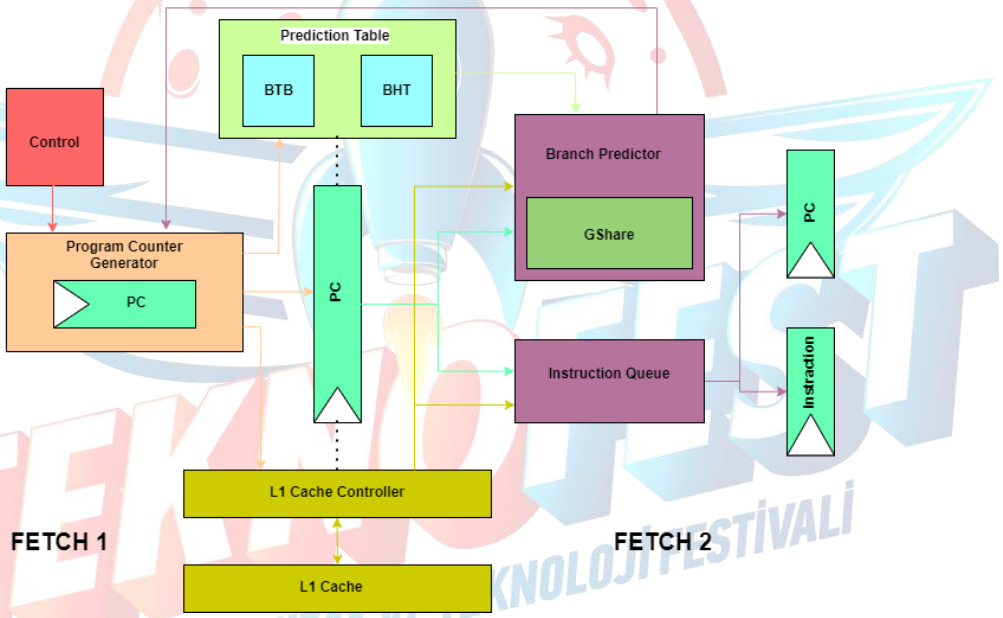
Çekirdekte toplama işlemi, toplama-sembolü yerine Sklansky[7] toplayıcısı ile gerçekleştirilmiştir. Çarpma işleminde çarpma sembolü yerine, değiştirilmiş Booth Dadda algoritmasıtercih edilmiştir. Değiştirilmiş Booth algoritması ve Dadda Tree yöntemlerinin bir araya getirilmesiyle oluşturulmuş bu

algoritma görece küçük bir alanda yüksek başarılı çarpma işlemlerinin yapılmasını sağlamaktadır. Bölme modülünde yeniden kaydetmeli bölme algoritması[8] kullanılmıştır. Karmaşık bölme algoritmalarına kıyasla çok küçük bir alanda ve çok basit işlemlerle bölme işlemini gerçekleştiren bu algoritma bir çevrimde iki adım ilerleyecek şekilde tasarlanmış ve 18 çevrimde sonuç vermektedir.

Kayan nokta biriminde toplama çıkarma işlemler için Carry-Select Adder, Booth Çarpma Algoritması ve Newton-Raphson bölme yöntemleri tercih edilmiştir. Ayrıca işlemcimiz atomik bellek işlemleri(-ing Atomic Memory Operation), bit-manipülasyonu ve kontrol durum yazmaçları(-ing Control and Status Register) buyruk tiplerini de desteklemektedir.

3.1.2 Getir(-ing Fetch)

Getir aşaması, buyruk önbelleği ile birlikte işlemcinin program akışına bağlı olarak değişen program sayacını üretmekten, ana bellekten işlemciye talimatları getirmeye ve dallanma tahmincisi ile performansı artırmaktan sorumludur. Alınan talimatlar program sayacı(-ing PC) tarafından belirtilen bellek adresinden alınmaktadır. Bu işlem önbellekler aracılığıyla gerçekleştirilmektedir.



Şekil 3

İşlemcimizde, getir aşaması kritik yolun azaltılması ve performansın artırılması amacıyla Getir1 ve Getir2 olmak üzere iki aşamaya bölünmüştür. Bu iki aşamalı yapı, işlemcinin her saat vuruşunda daha fazla talimat çekmesine ve boru hattının daha stabil yürütülmesine imkan vermektedir, böylece işlemcimiz her saat diliminde bir talimat işleyebilmektedir.

İşlemcimizde belleklerde ve öngörücü tablolarımızda SRAM kullandığımızdan en iyi durumda 1 saatlik okuma gecikmesi bulunmaktadır, bu durumu göz önüne aldığımızda Getir'i iki aşamaya bölmek işlemcimizde ekstra çevrim gecikmesine neden olmamaktadır ve her aşamanın gerçekleştirmesi gereken işlemlerin daha kısa sürede tamamlanmasını sağladığından işlemcinin daha yüksek saat hızında çalışmasına olanak tanımaktadır.

Bu tasarım değişimi, boru hattını da dengelememize olanak sağlayarak, boru hattının tıkanması olasılığını azaltarak boru hattı verimliliğini arttırdı. Aynı zamanda, dallanma tahmin biriminin daha verimli çalışmasına izin verdi; Getir1 aşamasında dallanma tahmini yapılır ve program sayacı üretilirken Getir2 aşamasında ise bu tahminin doğruluğu kontrol edilip gerekli düzeltmeler yapılabilir.

Getir aşamasını ikiye bölmek ayrıca bellek erişimlerinin daha verimli yönetilmesine olanak tanımaktadır. İlk aşamada talimatın önbelleğe alınıp alınmadığı kontrol edilebilir, ikinci aşamada ise bellekten alınabilir. Bu şekilde işlemcimizin genel performansı artmakta, saat hızı yükselmekte ve önbellek performansı daha verimli hale gelmiştir.

Getir aşaması şu şekilde çalışılmıştır:

I) Getir1 aşamasında PC üretiliyor

II) Öngörü tablolarında ve önbelleklere istekler yükleniyor

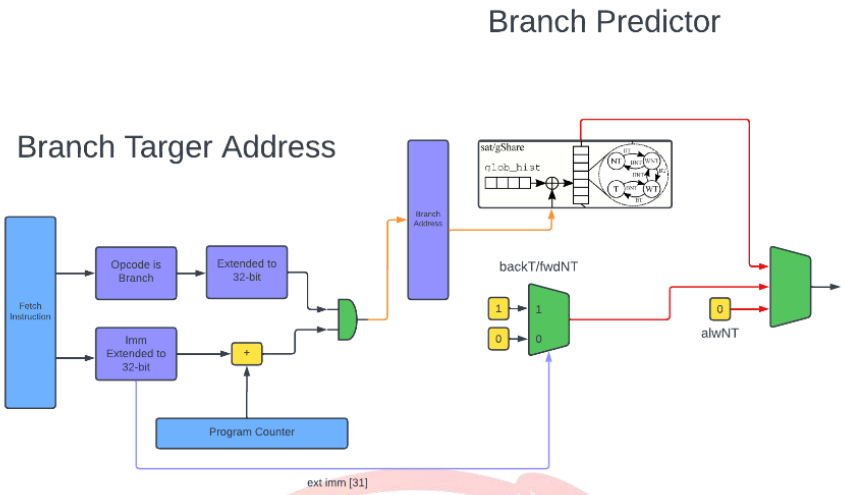
III) Bir sonraki çevrimde dallanma öngörüsü ve buyruk kuyruğunda sonuçlar değerlendiriliyor

IV) Eğer bir dallanma yapılacaksa bu buyruk üzerinde Getir1 aşamasında kombinasyonel olarak veri iletiliyor ve o çevrim içerisinde sinyalin güncellenmesi ve yeni isteğin yapılması sağlanıyor.

Getir2 aşamasında karşılaştığımız sorun henüz bellekten yanıta dönmemiş isteklerin boru hattındaki durumu oldu bu sorunu çözmek için yapılan istekler ve karşılanan istekler ile ilgili sayaç koyduk. Getir1'den yapılmış ancak Getir2'de cevaplanmamış önbellekteki veya veriyolundaki sayaçları ve istekleri sayıyor. Bunlar geldikçe sayıyı azaltıyoruz. Eğer DDB'den herhangi bir boşaltma sinyali gelirsede bu sayacı kontrol ederek Getir2 aşamasında o kadar isteği görmezden geliyoruz.

3.1.2.1 Dallanma Öngörücü

Dallanma öngörüsü, mikroişlemci performansının merkezi bir parçasıdır, zira doğru yapıldığında, işlemci boru hattını verimli bir şekilde doldurur ve bu da yüksek talimat yürütme oranlarına olanak tanır [9] .Dallanma öngörüsü mekanizması dallanma tarihi tablosu (*ing. BHT*) ve program sayacı (*ing. PC*) bilgilerini kullanarak gelecekteki dallanmaların sonuçlarını tahmin etmektedir. Bu tahminleme sürecinde başarımı arttırmak için GShare dallanma öngörücü kullanılmıştır [10]. GShare öngörücüsü, global dallanma tarihini ve Program Sayacı'nın düşük bitlerini birleştirerek dallanma tahminlerinde bulunmaktadır. Bu birleştirme, farklı Program Sayacı değerleri için ortaya çıkan benzersiz dallanma davranışlarının daha iyi eşleştirilmesine olanak tanımaktadır. GShare'in gücü, bir işlem içerisindeki farklı dallanmalar arasında ilişki kurabilme ve bu ilişkilerden yararlanarak daha hassas tahminler yapabilme yeteneğindedir. Özellikle, dallanmaların sık sık ve öngörülemez şekilde değiştiği büyük ve karmaşık işlem akışlarında, GShare dallanma öngörücüsü, daha geniş bir geçmiş yelpazesine dayanarak daha doğru tahminler sunmaktadır. GShare öngörücüsünün seçilmesinin bir diğer nedeni de koşulsuz(örn. jalr) ve koşullu(*ing. beq*) dallanmaların etkilerini daha iyi ayırt edebilmesidir. Koşullu dallanmaların sonucu öngörülebilir bir desene sahip olabilirken koşulsuz dallanmalar genellikle sabit bir davranış sergiler. GShare, bu iki tür dallanmanın geçmiş desenlerini birleştirerek ve bir 'Dallanma Geçmişi Tablosu' kullanarak daha akıllı tahminler yapabilmektedir. Bu, özellikle birden fazla dallanmanın sıkışık olarak gerçekleştiği durumlarda önemlidir çünkü ardışık dallanmalar birbirlerinin sonuçları üzerinde doğrudan bir etkiye sahip olabilir. Ayrıca GShare, boru hattının derinleştiği ve bellek erişim sürelerinin arttığı işlemcilerde dallanma tahminlerinin hızını ve doğruluğunu artırarak önemli bir performans kazancı sağlamaktadır. **Şekil 4**'te GShare Dallanma Öngörücü'nün mekanizması bulunmaktadır.



Şekil 4

3.1.3 Çöz

Çöz aşaması, işlemcinin ardışık düzeninde kritik bir rol oynar ve iki ana işlevi yerine getirir. İlk işlevi, işlemciye getirilen talimatları opcode'larına göre ayırmak ve işlevlerine uygun olarak işlemek için tasarladık. Bu ayrıştırma işlemi, işlemcinin talimatları doğru bir şekilde tanımlamasını ve bunları uygun işlem birimlerine yönlendirmesini sağlar. Talimat içindeki immediate değerleri gerektiğinde Immediate Expander birimi tarafından 32-bit değerlere dönüştürülmekte ve ardından Execute aşamasına aktarılmak üzere Immediate Value Decoder birimine yönlendirilmekte.

İkinci işlev, kaynak kayıt adreslerinin ayrıştırılmasını içerir. Execute aşamasına transfer edilmek üzere, talimat dizininde belirtilen dizindeki kaynak kayıt değerleri seri olarak ve mümkün olan en paralel şekilde çözülerek Execute aşamasına aktarılmakta. Bu ayrıştırma sürecinde, talimatların daha hızlı işlenmesi için seri kod çözücüler kullanılarak, işlemcinin genel performansını artırmayı hedefledik.

Gelen talimatları opcode'larına göre ayrıştırmakta ve gerekli birimlere yönlendirmekteyiz. mikro_islem_o sinyali ile aktif işlem birimini, sadece gerekli bitlere bakarak kontrol etmekteyiz. Kontrol sinyalleri (regwrite, memtoreg, memwrite, memread, branch, jump, op_imm, op_pcimm) ise yürütme aşamasında gerçekleştirilecek işlemleri belirlemektedir.

Ek olarak, istisnai durumlar (exception_o), geçersiz opcode'lar, yasadışı kaydırmalar veya özel durumlar (ECALL, EBREAK, MRET) tespit edildiğinde, işlemcinin bu durumları yönetebilmesi için bir istisna sinyali ile ayrı bir çıkış sağladık.

3.1.4 Yürüt

Yürüt aşaması, getirilen buyrukların işlenmesini içerir ve temel aritmetik işlemler (toplama, çıkarma, çarpma, bölme), bit düzeyinde mantıksal işlemler, dallanma kararları ve sistem çağrıları gibi bir dizi işlemi gerçekleştirir. Bu aşamada, Aritmetik Mantık Birimi (*ing. ALU*) toplama, çıkarma gibi temel aritmetik işlemleri yaparken, çarpma ve bölme işlemleri için ayrı birimler kullanılacaktır. "AFB" (*ing. Atomic, Float, Branch*) bileşeni, atomik işlemleri, kayar nokta işlemlerini ve dallanma işlemlerini yürütür. Bellek işlemleri için Yükle/Sakla birimi kullanılır ve bu birim, veri önbelleğine erişim için okuma veya yazma işlemlerini yönetir. Dallanma buyrukları, tahmin edilen sonuçlara göre boru hattı yönlendirmesini gerçekleştirir. İşlemler tamamlandıktan sonra, veri bağımlılıklarını çözmek ve

performans artışı sağlamak için "Çöz" aşamasına veri yönlendirmesi yapılır bu yaklaşım verimli ve yüksek performanslı bir işlem akışı sağlamak için kritik önem taşımaktadır.

3.1.4.1 Aritmetik Mantık Birimi(ing. ALU)

Yarışma koşulları göz önünde bulundurularak, toplama, çıkarma ve mantık işlemleri gibi temel aritmetik işlemlerin verimli bir şekilde gerçekleştirilmesi, bir işlemcinin performansında kritik bir rol oynar. İşlemcide Aritmetik Mantık Birimi, bu temel işlemleri yüksek hızda paralel işlem yürütebilmek için özel algoritmalar ve donanımlar incelenmiştir. Hem hız hem de tasarımın serimi dikkate alınarak, toplama işlemleri için Sklansky toplayıcılar kullanılmıştır bu sayede toplama süresi önemli ölçüde azaltılarak paralel hesaplama potansiyeli arttırılmıştır.

3.1.4.2 Çarpma/Bölme Birimi

Çarpma işlemleri toplama işlemlerine göre daha karmaşık ve kaynak yoğun olduğundan, özel algoritmalar incelenmiş alan ve performans dengesini optimize etmek için Modified Booth Dadda Algoritması tasarlanmıştır. Modified Booth Dadda Algoritması, hem sayıda hem de verimde önemli iyileştirmeler sunarak, çarpma işlemlerinde kullanılan alanı minimize eder ve toplama işlemlerinden elde edilen performansı maksimize etmemizi sağladı. Son olarak bölme işlemi, toplama ve çıkarma işlemleri kullanılarak gerçekleştirilmiştir. Bunun nedeni özel bölme algoritmaların fazla alan kaplaması oldu.

3.1.4.3 Kayan Nokta Birimi(ing. Floating Point Unit)

Yarışma kapsamında işlemci mimarisi için tasarlanacak olan kayar nokta biriminde (-ing. FPU), performans, donanım kullanımı ve enerji verimliliği arasında optimal bir denge sağlamak amacıyla özel algoritmalar kullanılmasına karar verdik. Carry-Select Adder, Booth Çarpma Algoritması ve Newton-Raphson bölme yöntemleri tercih edildi. Carry-Select Adder, modüler yapısı ile donanım maliyetini düşük tutarken işlem hızını maksimize etmemizi sağladı; Booth Algoritması, çarpma işlemlerini verimlileştirerek hem alan hem de enerji tüketimini optimize etmiştir; Newton-Raphson yöntemi ise, yüksek doğrulukta ve hızlı yakınsaklıkta bölme işlemlerini sabit bir donanım alanında gerçekleştirerek güç tüketimini öngörülebilir düzeyde tutar. Bu algoritmaların entegrasyonu, işlemcimizin yüksek performans sergilemesine olanak tanırken, maliyet ve enerji verimliliği açısından da faydalı olmuştur.

3.1.4.4 Atomik Bellek Operasyon Birimi (ing. Atomic Memory Operation Unit)

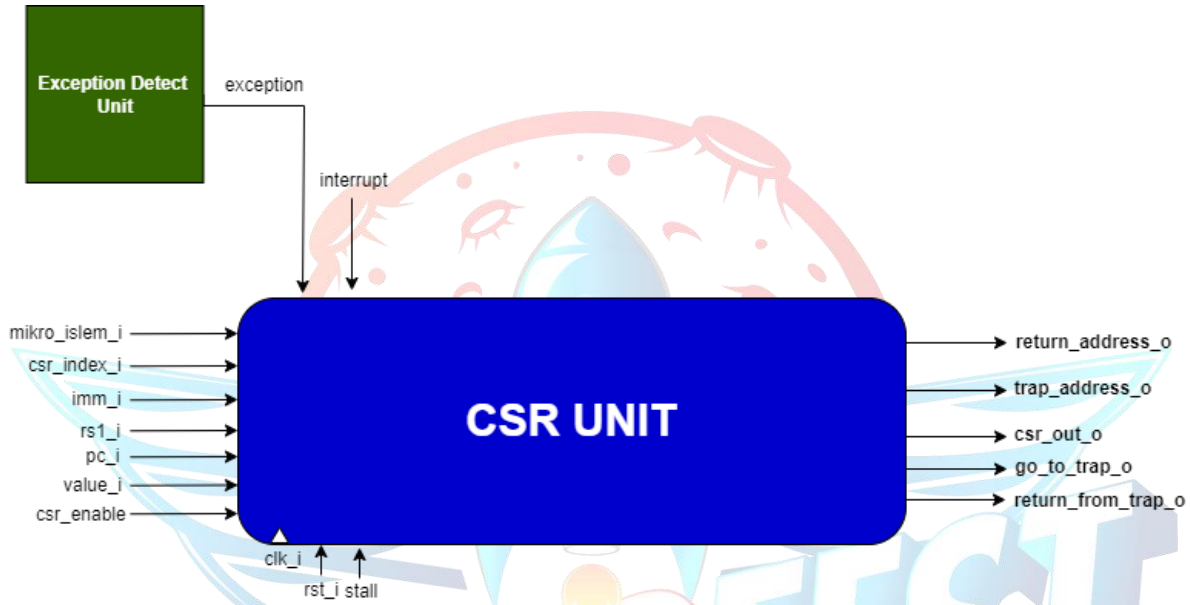
Yarışma kapsamında tasarlanacak Atomik Bellek Operasyon Birimi(ing. AMO) işlem birimi, çok iş parçacıklı ve eşzamanlı programlama modelleri için kritik önem taşıyan senkronizasyon mekanizmalarını verimli bir şekilde yürütebilmesi hedeflenmektedir [11] . Bu birim, 'Load-Reserved' (LR.W) ve 'Store-Conditional' (SC.W) işlemleri gibi kritik bölüm (ing. critical section) yönetimini sağlayarak, çoklu iş parçacıklı ortamlarda veri bütünlüğünü korumak için tasarlanmaya çalışılmıştır. Ayrıca, Atomik Bellek Operasyon Birimi talimatları - örneğin, AMOSWAP.W, AMOADD.W, vb. - veri üzerinde atomik değişiklikler yaparak hafıza erişim yarışlarını önleyecektir. Optimizasyon açısından, bu birim düşük gecikme(*ing. latency*) süreleri ve yüksek hızlı veri yolu erişimi ile tasarlanması, atomik işlemlerin verimli bir şekilde tamamlanması için gerekli olan hafıza ve işlemci kaynakları önceliklenmesi hedeflenmektedir. Ayrıca, bu işlemlerin boru hattındaki diğer talimatlarla olan etkileşimleri dikkate alınarak boru hattı engellemeleri azaltılmaya çalışılarak boru hattı verimliliği artırılması hedeflenmektedir.

3.1.4.5 Kontrol ve Durum Yazmacı Birimi (ing. Control and Status Register (CSR) Unit)

İşlemcimiz için tasarladığımız CSR birimi, sadece machine mode'u destekleyen CSR kayıtlarını içermektedir. Bu birimi, RISC-V mimarisinin Zicsr eklentisini destekleyecek şekilde oluşturduk. **Şekil**

5'te görülen CSR Modülü, işlemcinin yönetim ve kontrol işlevlerini etkin bir şekilde yerine getirmesini sağlar. Mikroişlem kontrol sinyallerini, CSR indekslerini, immediate değerlerini, register ve PC değerlerini giriş olarak alır. Çıkış olarak, tuzak adreslerini, CSR değerlerini ve tuzak sinyallerini sağlar.

Bu birim, mstatus, mie, mtvec, mscratch, mepc, mcause ve mtval gibi önemli CSR kayıtlarını içerir. Özellikle, Exception Detect Unit (İstisna Tespit Birimi) ile entegre çalışarak istisnalar ve interrupt sinyallerini yönetir. CSR birimimiz, Teknofest yarışması gereksinimlerine uygun olarak sadece machine mode'u destekleyecek şekilde tasarlandı ve bu sayede yüksek performans ve güvenilirlik sağlanacak. Bu tasarım, işlemcimizin RISC-V Zicsr eklentisi ile tam uyumlu çalışmasını ve çeşitli kontrol durumlarını verimli bir şekilde yönetmesini mümkün kılacaktır.



Şekil 5

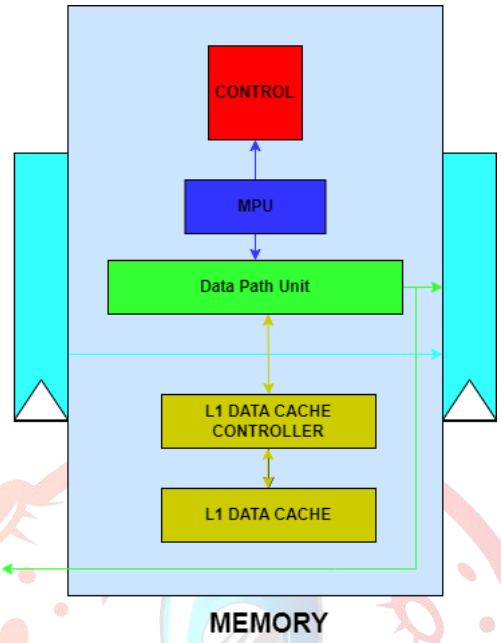
3.1.4.6 Bit-Manipülasyonu İşlem Birimi

Bu birimde, aritmetik işlemler için donanımsal olarak hızlandırılmış yollar sağlayarak, kayar nokta ve tam sayı işlemlerinde performansı iyileştirmesi hedeflenmiştir[12] . Örneğin, "Count Leading Zeros" veya "Carry-Less Multiply" gibi işlemler için özelleştirilmiş devreler incelenerek tasarlanmıştır. Bu devreler, genellikle çok sayıda işlem döngüsü gerektiren bit seviyesi işlemlerini tek bir döngüde tamamlayarak(örn. Barrel Shifter) işlem hızını artırmakta ve düşük güç tüketimi sağlamaktadır.

3.1.4.7 Bellek İşlem Birimi (İng. Memory Process Unit)

İşlemci mimarisinde bellek işlem birimi, işlemci çekirdeği ile bellek arasındaki veri alışverişini yönetmektedir ve bu süreç, hedef bellek adresinin hesaplanması, verinin belleğe yazılması veya bellekten okunması gibi adımları içermektedir. Bellek işlemlerinin işlemcinin diğer aşamalarıyla koordine edilmesi gerekir ve bu süreç boru hattında gecikmelere yol açabilmektedir. Bu nedenle, bellek işlem birimi işlem süresini en aza indirmek için önbellek hiyerarşisi ve bellek erişim stratejileri kullanılmaktadır. Bellek işlem biriminin performansını optimize etmek amacıyla önbellekleme, spekülatif yürütme, sıra dışı yürütme, engellemeyen önbellek tasarımı ve önbelleğe alma gibi teknikler uygulanarak boru hattı duraklamaları en aza indirilmeye çalışılmıştır. Bu birimde tasarım kararında güncellemeye giderek memory aşaması(ing. stage) oluşturularak ayrı bir bölüme de taşınması da düşünülmektedir. Bu aşama bellek işlem birimi ve veriyolu olmak üzere iki aşamadan oluşacaktır. Bellek işlem biriminde işlemlerin doğru hizalanması, bayt seçimleri ve oluşabilecek olağan dışı durumları

tespit etmektedir. Veriyolu ise adres haritasına göre önbelleklenen ve önbelleklenemeyen adresleri ayrıştırır ve hiyerarşi sağlar. **Şekil 6**'da detaylı olarak gösterilmiştir.



Şekil 6

3.1.4.8 Denetim Durum Birimi

Denetim durum birimi, boru hattının kontrolünü sağlar. Boru hattı aşamalarından gelen hazır sinyalleri ve yazmaç adreslerini dikkate alarak yönlendirme yapar veya boru hattını durdurur. Genellikle kombinasyonel devrelerden oluşan denetim durum birimi, çarpma biriminin yönlendirilemez olması ve başlangıçtaki boru hattındaki buyrukların geçersiz olması nedeniyle iki bitlik durmuş (stopped) ve boş_başla isimli ardışık (sequential) birimler bulundurur.

Veri Sorunu Denetimi

Veri sorunu, bir yazmaca yazma işlemi gerçekleşmeden önce aynı yazmaca bir okuma isteği geldiğinde ortaya çıkar.Çok aşamalı boru hatlarında, veri yönlendirmesi yazmaç öğesi okunduktan sonra Geri Yaz aşamasına kadar her aşamada yapılabilir. Ancak bizim tasarımımızda, yazmaç öğesinin okunduğu ve geri yazıldığı aşamalar arasında sadece bir aşama olduğundan (Yürüt), tek bir aşamadan veri yönlendirmesi yapılması yeterli olmuştur ancak bu aşama Bellek aşamasına taşıyarak optimizasyon yapabiliriz. Bu sayede, aynı yazmaca yazma işleminden sonra okuma yapılması boru hattını durdurmaz ve sadece veri yönlendirmesi yapılır.

Yapı Sorunu Denetimi

Yürüt işlemleri birimlerinin, istenen işlemleri birden fazla çevrimde tamamladığı bilinir. Bu durum, bu birimlerin buyruğu geldiğinde tüm boru hattını beklemeye zorlar. Bu tür buyruklar Yürüt aşamasına geldiğinde, Denetim Durum Birimine bildirilir ve tüm boru hattı buradan gelen sinyallerle durdurulur.

3.1.4.10 İSTİSNA ALGILAMA BİRİMİ

Bu Verilog modülünü, RISC-V işlemcisinde istisnaları tespit etmek için tasarladık. ECALL, EBREAK, MRET ve ILLEGAL istisnalarını decode aşamasında çözerek istisna çıkışını sağladık. Diğer bellek erişim hataları ve hizalama istisnalarını tespit etmek için ayrı bir modül tasarladık. ECALL, işletim sistemi çağrıları için; EBREAK, hata ayıklama kesmeleri için; MRET, makine modundan dönüş için; ILLEGAL, geçersiz komutlar için kullanılıyor. Ayrıca, yükleme ve depolama işlemlerinde hizalama veya erişim hatası olduğunda tetiklenen istisnaları yönetmek için gerekli modülleri geliştirdik. Bu modül ile işlemcideki istisna ve hata durumlarını etkin bir şekilde tespit ve yönetim sağladık.

```
`define MACHINE_SOFTWARE_INTERRUPT 3
`define MACHINE_TIMER_INTERRUPT 7
`define MACHINE_EXTERNAL_INTERRUPT 11
`define INSTRUCTION_ADDRESS_MISALIGNED 0
`define INSTRUCTION_ACCESS_FAULT 1
`define ILLEGAL_INSTRUCTION 2
`define EBREAK 3
`define LOAD_ADDRESS_MISALIGNED 4
`define LOAD_ACCESS_FAULT 5
`define STORE_ADDRESS_MISALIGNED 6
`define ECALL 11
```

Şekil 7

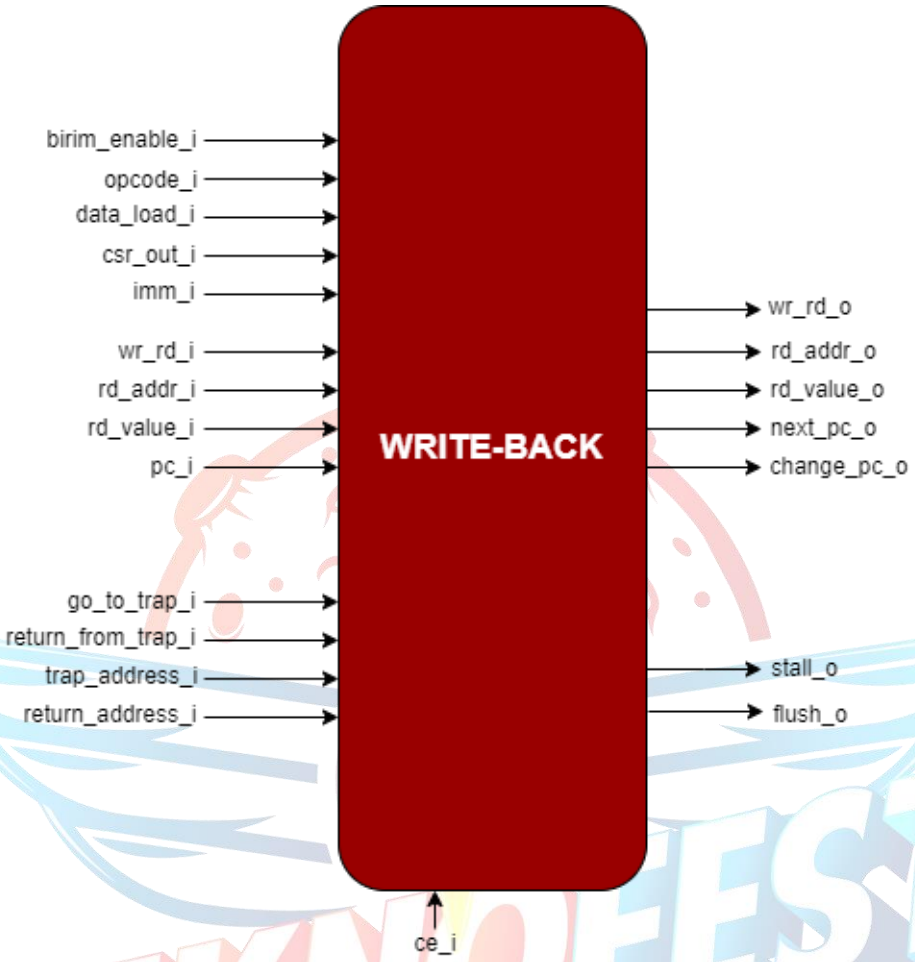
3.1.5 GERİ YAZ

Write-back aşaması, işlemcinin ardışık düzeninde son aşama olarak tasarlanır. Bu aşamada, yürütülen talimatların sonuçlarını belirledik ve kayıt dosyasına yazılmasını sağladık. ALU sonuçlarını yürütme aşamasında (EX) ürettik ve bellekten okunan verileri bellek erişim aşamasında (MEM) işleyerek yazma aşamasında (WB) kayıt dosyasına yazdık. jal ve jalr gibi talimatlarla bir sonraki talimatın adresini hesaplayarak kayıt dosyasına yazdık. CSR okuma/yazma işlemleri de sonuçlarını ilgili kayda yazdık.

Write-back aşamasında, hangi verinin kayıt dosyasına yazılacağını belirlemek için bir multiplexer (MUX) kullandık. Talimatın opcode ve funct3/funct7 alanlarına göre belirlenen hedef kayıt numarasını kullanarak, uygun kayda seçilen sonucu yazdık.

Girişler arasında yükleme verisi, CSR değeri, işlemci talimat kodu, immediate değeri, hedef kayıt yazma izni, hedef kayıt adresi, hedef kayda yazılacak veri, mevcut PC değeri, tuzak ve tuzaktan dönüş sinyalleri bulunmakta. Çıkışlar arasında yazma izni, hedef kayıt adresi, hedef kayda yazılacak veri, yeni PC değeri, PC'nin değişmesi gerektiğini belirten sinyal, ardışık düzen duraklatma ve temizleme sinyalleri bulunmakta. Başlangıçta tüm sinyalleri sıfırladık. go_to_trap_i veya return_from_trap_i sinyali aktifse, PC'yi uygun adrese yönlendirecek ve önceki aşamaları temizleyeceğiz. Normal operasyonlarda, veri

talimatlarına göre hedef kayda yazacağız ve gerektiğinde PC'yi güncelleyeceğiz. Bu modül, write-back aşamasında gerçekleşen tüm işlemleri ve gerekli kontrol sinyallerini yönetmek üzere tasarladık.



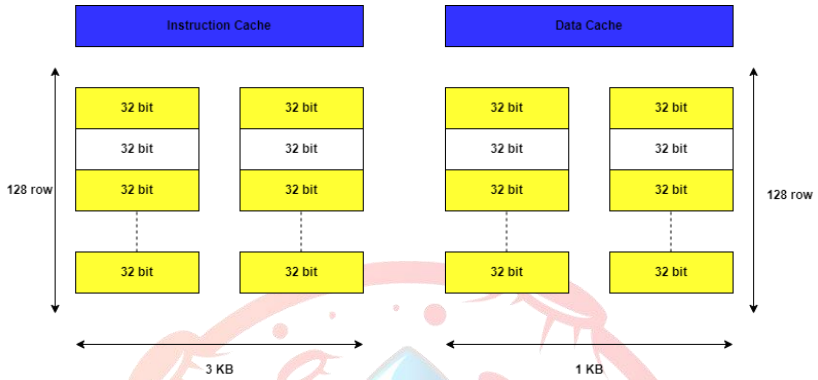
Şekil 8

3.2. Bellek Tasarımları

Proje kapsamında gerçekleştirilen bellek hiyerarşisi, Getir Aşamasındaki 1. seviye buyruk önbelleği denetleyicisine bağlı 3 KB boyutunda buyruk önbelleği ve buyruk önbelleği denetleyicisi, Yürüt Aşamasında bulunan 1. seviye veri önbelleği denetleyicisine bağlı 1 KB boyutunda Veri Önbelleği ve bu iki önbellek denetleyicisini ana belleğe bağlayan bir Ana Bellek Denetleyicisinden oluşmaktadır. Buyruk Önbelleği yalnızca program sayacının ana bellekten getirdiği buyrukları içerirken, Veri Önbelleği "Load" ve "Store" buyrukları tarafından getirilen verileri içerir[13]. Bir programın eriştiği buyruklar, erişilen verilere kıyasla alan ve zaman açısından daha fazla yerelliğe sahiptir.

Veri ve buyruk önbellek denetleyicileri "Çevreleyici Modül Ana Belleği" ile Ana Bellek Denetleyicisi aracılığıyla bağlıdır. Ana Bellek Denetleyicisi, Önbellek denetleyicilerinden biri Ana Bellek isteği oluşturduğunda bu isteği Teknofest Ana Belleğine aktarır. Aynı anda iki Önbellek Denetleyicisinin de istek oluşturduğu durumda Buyruk Önbelleği isteği önceliklendirilir. Buyruk Önbelleği, bir döngünün veya fonksiyon çağrısının işlenmesi durumunda zamanda yerellikten faydalanırken, Veri Önbelleği ise aynı verinin tekrar kullanıldığı durumlarda zamanda yerellikten faydalanır.

Önbellek ve önbellek denetleyicilerin satır sayısı, veri öbeği genişliği, yol sayısı parametrik olarak tasarlanmıştır. Buyruk önbelleği 6 yollu olarak 3 KB veri önbelleği ise 2 yollu olarak 1KB olarak tasarlanmıştır. Saklanan adreslere geri erişim için yaz ve tahsis et(ing. write and allocate) yöntemi kullanılmıştır. **Şekil 9**'de sırasıyla Buyruk ve Veri Önbelleklerinin tasarımları açıklanmaktadır.



Şekil 9

3.2.1 Önbellek Kontrol Mekanizmaları

Önbellek kontrol mekanizmaları, önbellek tutarlılığı ve verimliliği açısından kritiktir. Önbellek denetleyicileri, önbelleklerin ana bellek ile veri senkronizasyonunu yönetir ve önbellek güncellemelerini, geçersizliklerini ve yeniden doldurmalarını kontrol eder. Ayrıca, önbellek denetleyicileri, önbellek politikalarının performansını analiz eder ve önbellek vuruş ve kaçırma oranlarını izleyerek iyileştirmeler yapar. İşlemci mimarisinde, önbellek denetleyicileri "Çevreleyici Modül Ana Belleğine" (bellek yönetim birimi veya bellek denetleyicisi) sinyaller gönderir. Bu modül, işlemci tarafından istenen veri veya talimatların ana bellekten önbelleğe taşınmasını koordine eder. Eğer bir veri veya talimat önbellekte mevcut değilse ve ana bellekten alınması gerekiyorsa, bu denetleyici gerekli veri transfer sürecini başlatır.

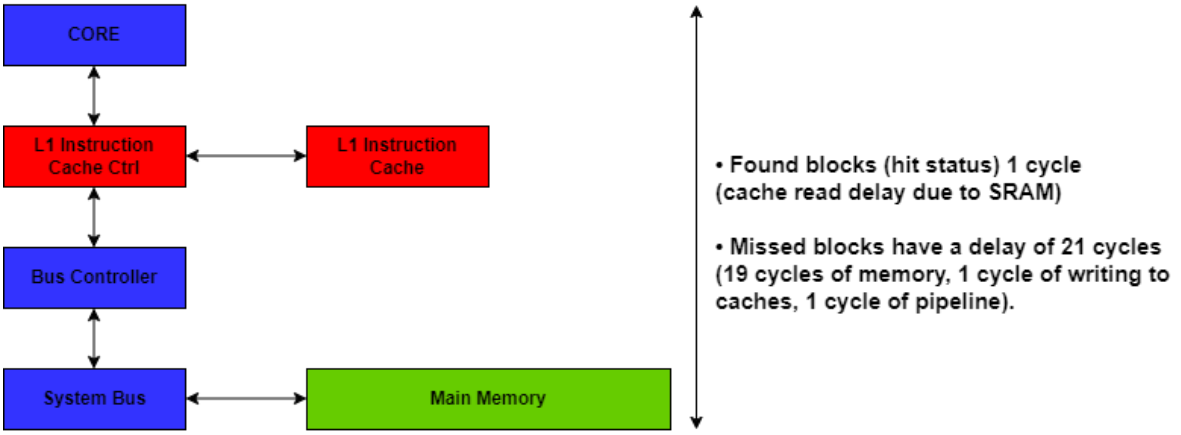
3.2.2 Buyruk ve Veri Önbelleği

Her iki önbellek de 32 bitlik veri bloklarından oluşur ve işlemcinin ana belleğe erişimini azaltmak için kullanılır.

Buyruk Önbelleği: Talimatları hızlı bir şekilde almak için kullanılır. Bu önbellek, işlemcinin talimatları daha hızlı işlemesine ve talimat akışının kesintisiz olmasına olanak tanır.

Buyruk Önbelleği Denetleyicisi: Talimatların getirilmesinden sorumlu denetleyicidir. Bu denetleyici, talimatların önbellekten mi yoksa ana bellekten mi getirileceğini belirler.

Şekil 10'da L1 buyruk önbelleği mekanizması ve gecikme süreleri verilmiştir:



Şekil 10

Tablo 1 gecikme durumları verilmiştir. Buyruk önbelleğimizde SRAM kullandığımız bulma(ing. hit) durumunda 1 çevrim okuma gecikmesine sahibiz. Bulamama(ing. miss) durumunda ise 19 çevrim bellek, 1 çevrim yazma, 1 çevrim boru hattında olmak üzere toplam 21 çevrim kaybediyoruz.

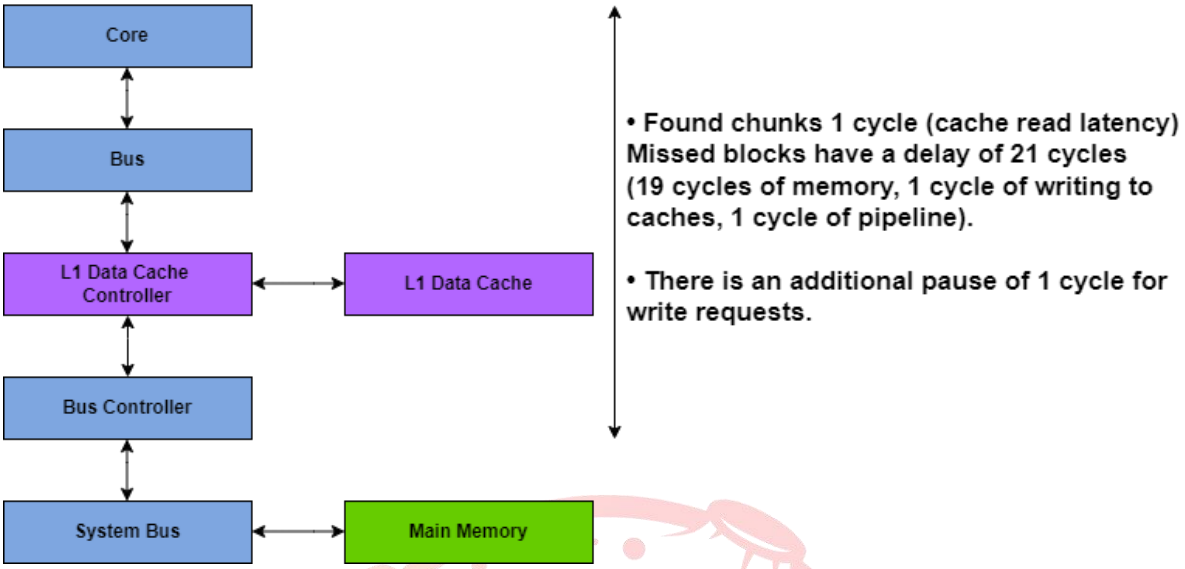
Tablo 1

Buyruk Belleği İsteği	Çevrim Sayısı
Okuma En İyi Durumda	1 çevrim
Okuma En Kötü Durumda	21 çevrim

Veri Önbelleği: Verileri hızlı bir şekilde almak ve depolamak için kullanılır. Bu önbellek, özellikle yükleme ve depolama işlemleri sırasında bellek erişim süresini azaltır.

Veri Önbelleği Denetleyicisi: Verilerin getirilmesi ve depolanmasından sorumlu denetleyicidir. Bu denetleyici, verilerin önbellekten mi yoksa ana bellekten mi getirileceğini belirler.

Şekil 11'de L1 veri önbelleği mekanizması ve gecikme süreleri verilmiştir:



Şekil 11

Tablo 2'de gecikme durumları verilmiştir. Buyruk önbelleğimizde SRAM kullandığımız bulma(ing. hit) durumunda 1 çevrim okuma gecikmesine sahibiz. Bulamama(ing. miss) durumunda ise 19 çevrim bellek, 1 çevrim yazma, 1 çevrim boru hattında olmak üzere toplam 21 çevrim kaybediyoruz. Yazma isteklerinde 1 çevrim ek duraklama olmaktadır. Önbelleksiz işlem gecikmeleri en az 3 çevrim (1 çevrim kaydetme, 2 çevrim veri yolu denetleyicisi) sonra iletilmektedir.

Tablo 2

Buyruk Belleği İsteği	Çevrim Sayısı
Okuma En İyi Durumda	1 çevrim
Okuma En Kötü Durumda	21 çevrim
Yazma En İyi Durumda	2 çevrim
Yazma En İyi Durumda	22 çevrim

Fetch 1 ve Fetch 2: Bunlar, talimatların getirilme aşamalarıdır. Fetch 1 aşamasında, talimat adresi belirlenir ve önbelleğe erişim başlatılır. Fetch 2 aşamasında ise talimatın önbellekten alınıp alınmadığı kontrol edilir.

Yükleme ve Depolama İşlemleri: Bunlar, verilerin yüklenmesi ve depolanması işlemleridir. Veri önbelleği, bu işlemler sırasında verilerin hızlı bir şekilde işlenmesini sağlar.

3.2.3 Önbellek Durum Mekanizmaları

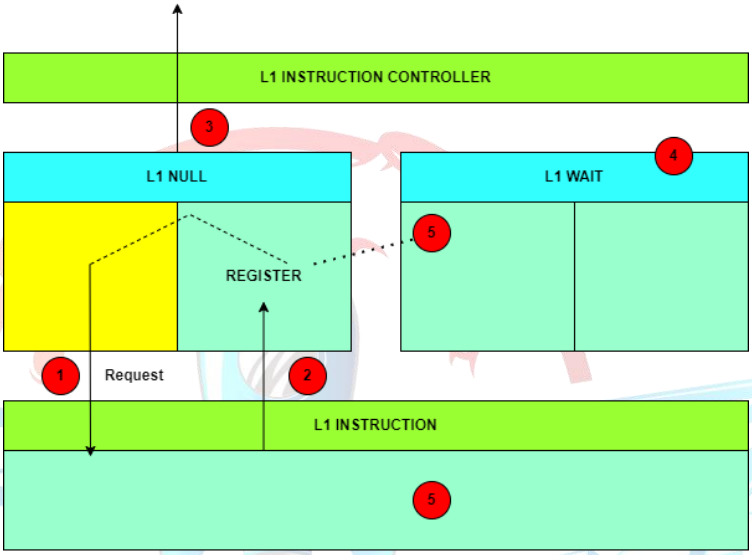
Önbellek durum diyagramı, önbellek kontrolünün nasıl çalıştığını gösterir. Bu diyagram, önbelleğin farklı durumlarını ve bu durumlar arasında nasıl geçiş yapılacağını açıklar.

L1 NULL: Önbelleğin başlangıç durumudur. Bu durumda, önbellekte geçerli veri bulunmamaktadır.

L1 BEKLEME: Önbellek ana bellekten veri bekliyor. Bu durumda, veri isteği ana belleğe iletilmiş ve yanıt beklenmektedir.

L1 TALİMAT: Veri önbellekte bulunur ve işlemciye iletilir. Bu durumda, veri doğrudan önbellekten getirilir ve Fetch 2 aşamasına iletilir.

KAYIT: Önbellek ana bellekten veriyi kaydeder. Bu durumda, ana bellekten gelen veri önbelleğe yazılır ve sonraki erişimler için saklanır.



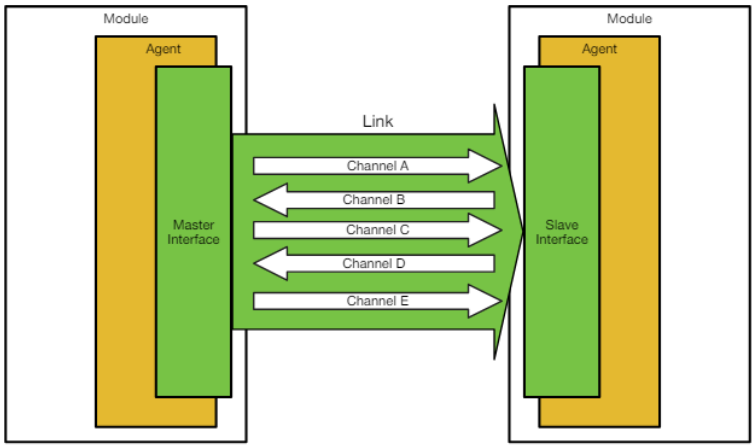
2	The discovery status is checked by comparing the line labels with the address.
3	If data is found, it is combinatorially directed to the Fetch 2 stage.
4	In case of a miss, the path to be removed is determined and a request is made to main memory and the response is waited.
5	When the response returns, data is written onto the selected path and the controller rewinds to state 2.

Şekil 12

3.3.Çevre Birimi Tasarımı

Çevre birimlerinin veri iletişiminin sağlanması için üzerinde çalıştığımız üç protokol yapısı: AXI, Wishbone ve TileLink prokolüdür. TileLink arayüzü bus kontrol ünitesi ile iletişim kurarak çevre birimlerinin çekirdekten gelen programa göre kontrol edilmesini sağlar. TileLink arayüzüne gidecek olan saklama ve yükleme komutları çekirdekte bulunan bellek işlem birimi tarafından seçilerek ilgili çevre birimini kontrol etmek üzere ilgili adrese gönderilir.

TileLink, belleğe ve diğer yardımcı cihazlara tutarlı bellek eşlemeli erişim ile birden fazla ana bilgisayara olanak sağlayan çip ölçeğinde bir ara bağlantı standardıdır. Tilelinkte 5 kanal olmasına rağmen sadece biz A ve D kanallarını kullandık.



Şekil 13

Kanal A : İşlemin hangi adreste gerçekleşeceğini belirtir.

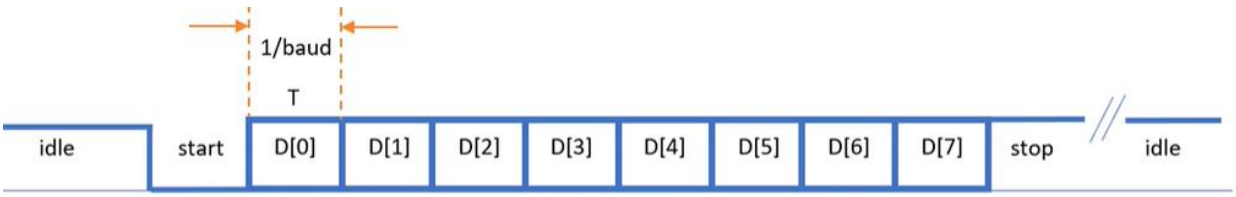
Kanal D: Talep edene bir veri yanıtı iletir.

Tilelink protokolünde sadece 2 kanal kullandığımız için en düşük gecikmeyle çalışmasını sağladık.

Axi'nin de 5 kanalı vardır ve yüksek miktarda sinyal işleme gerektirir. Sıralı yürütme yapıldığından ve her döngü 1 talimatı işleyebildiğinden, boru hattı hiçbir zaman aynı yazma veya okuma döngüsünü talep etmediğinden paralel kanallar bir avantaj sağlamaz. Paralel kanallar bir avantaj sağlamaz çünkü boru hattı hiçbir zaman aynı yazma veya okuma döngüsünü talep etmez. Bu yüzden Tilelink protokolünü tercih ettik.

3.3.1 UART

UART (Evrensel Asenkron Alıcı/Verici) iletişim protokolü, seri veri iletişimi için sıkça kullanılan bir yöntemdir. UART protokolü, iki cihaz arasında veri alışverişi sağlamak için bir seri veri yolu kullanır.Veri iletilen mesaj; başlangıç biti, 8 bit veri biti ve bir durdurma biti içerir. Başlangıç biti, veri iletiminin başladığını gösterir ve her zaman "0" durumunda beklemektedir. Veri bitleri, gönderilen bilginin kendisidir ve 8 bittir. Durdurma biti, iletimin sonlandırıldığını gösterir ve her zaman "1" olarak ayarlanır. Bu bitlerin sırası ve kontrolü, kuyruk tabanlı bir veri yolu ve sonlu durum makinesi (FSM) aracılığıyla gerçekleştirilmiştir. Alıcı (RX) ve verici (TX) bölümleri ayrı ayrı tasarlanmış ve TileLink Slave protokolünü kullanan bir modül altında birleştirilmiştir. UART protokolü, cihazlar arasında veri iletimi için farklı hızlarda kullanılabilir ve bu hızlar genellikle baud hızı olarak belirtilir. İstenen iletişim hızını (baudrate) sağlamak için basit bir sayaç (counter) kullanılmıştır. Bu sayaçtan çıkan sinyal, bir saat döngüsü boyunca aktif kalarak gerekli senkronizasyonu sağlar.



Şekil 14

4. ÇİP TASARIM AKIŞI (20 PUAN)

Çip tasarım akışında Synopsys kullanılarak çip tasarım akışının gerçekleştirilmesi planlanmaktadır. Bu süreçte, ilk olarak tasarım spesifikasyonlarına göre yapılmış olan RTL (Register Transfer Level) tasarımının simülasyonu Synopsys VCS veya Design Compiler ile yapılacaktır. Ardından, sentezleme işlemi Synopsys Design Compiler kullanılarak gerçekleştirilip kapı düzeyinde netlist elde edilecektir. Fonksiyonel doğrulama için Synopsys VCS kullanılacak ve ardından yerleşim ve yönlendirme işlemi Synopsys IC Compiler veya ICC2 ile yapılacaktır.

Zamanlama analizi Synopsys PrimeTime kullanılarak gerçekleştirilecek ve tasarımın zamanlama gereksinimlerine uygunluğu kontrol edilecektir. Güç analizi Synopsys PrimeTime PX ile yapılarak, tasarımın güç tüketimi değerlendirilecektir. Anten analizi ve düzeltmeleri Synopsys IC Validator ile yapılıp, tasarımın anten etkileri analiz edilecektir. Tasarımın üretim için GDSII formatına dönüştürülmesi Synopsys IC Compiler veya ICC2 ile sağlanacaktır. Son olarak, nihai doğrulama ve optimizasyonlar Synopsys DRC/LVS araçları ile yapılacak ve tasarımın doğruluğu ve üretilebilirliği sağlanacaktır.

Tasarım süreci boyunca sık karşılaşılan hatalar arasında zamanlama ihlalleri, güç tüketimi limitlerinin aşılması, hizalama hataları ve bellek erişim hataları bulunmaktadır. Bu tür hataların önüne geçmek için her adımda dikkatli analizler ve doğrulamalar yapılacaktır. Örneğin, zamanlama ihlallerini önlemek için Synopsys PrimeTime ile kapsamlı zamanlama analizleri yapılacak, güç tüketimi limitlerinin aşılmaması için güç optimizasyon teknikleri kullanılacak ve hizalama ve bellek erişim hataları için tasarım doğrulama araçları etkin şekilde kullanılacaktır. Son olarak, nihai doğrulama ve optimizasyonlar Synopsys DRC/LVS araçları ile yapılacak ve tasarımın doğruluğu ve üretilebilirliği sağlanacaktır.

5. TEST

Testler projede en çok üzerinde zaman harcadığımız kısımlardandır. Synopsys üzerinden çalışmaya geç başladığımız için özellikle RTL tasarımımız üzerinden doğrulama ve FPGA testlerine uzun bir süre ayrılmıştır. Tasarımın doğru çalıştığını doğrulamak için kullandığımız tüm test araçları sırasıyla; cocotb [14], verilator [15], icarus verilog [16], modelsim, Vivado, Quartus ve spike [17] şeklindedir.

Bu test araçları beraber veya yalnız kullanılarak aşağıda verilen testlerin birçoğu başarıyla geçirilmiştir. Atomik buyruklar için yaptığımız testlerde henüz tam bir başarım sağlanamamıştır.

Açık kaynak kullanılan testler: Riscv-tests, Riscv-arch-tests, Spike, Coremark.

Kendi Testlerimiz: UART, TIMER testi, önbellek testleri.

Testler simülasyonlar dışında FPGA'ler üzerinde de koşturulmuştur. İşlemcimiz, elde bulunan XC7Z020-1CLG400C Zybo Z7 125 MHz'de, Xilinx Basys3'de 50 MHz'de çalıştırılmıştır.

5.1 Çekirdek Testleri

Şartnamede göre RISC-V testleri C programları da linker scriptler ile derlenmiştir. Testler VIVADO'nun yanı sıra bir Python kütüphanesi olan cocotb'de yazılan testbenchler kullanılarak çalıştırılmıştır. Cocotb kullanılması testbenchten bağımsız olarak simülatör değiştirmemize olanak tanımaktadır. Bu sayede

verilog testbench desteklemeyen Verilatör gibi simülatörler de kullanılabilmiştir. Ayrıca python tabanlı testbenchler gerek test otomasyonu gerekse de zaman açısından çalışma verimimizi artırmıştır.

Geçen yıl yapılan yarışmayı referans alarak aapg testi de uygulanmış ve en çok hata bulmamızı sağlayan test olmuştur. Kapsama tabanlı rastgele testler üretebilen aapg sayesinde 100 milyarlarca istatistiksel olarak üretilmiş buyruklarla spike emülatörünü referans olarak kullanarak imza tabanlı (ing. signature based) ve iz tabanlı (ing. trace based) olarak hem de FPGA tasarımı üzerinde testler gerçekleştirilmiştir. Bu testlerin haricinde de işlemcimiz benchmarklar yürütülerek doğrulanmıştır. Bu testlere ve benchmarklara ek olarak C programlarıyla çeşitli uç senaryoları test eden mikro programlar yazılmış ve çalıştırılmıştır. Aşağıda Şekil 15 ’de bit manipülasyon işlemleri testi için örnek C kodu verilmiştir.

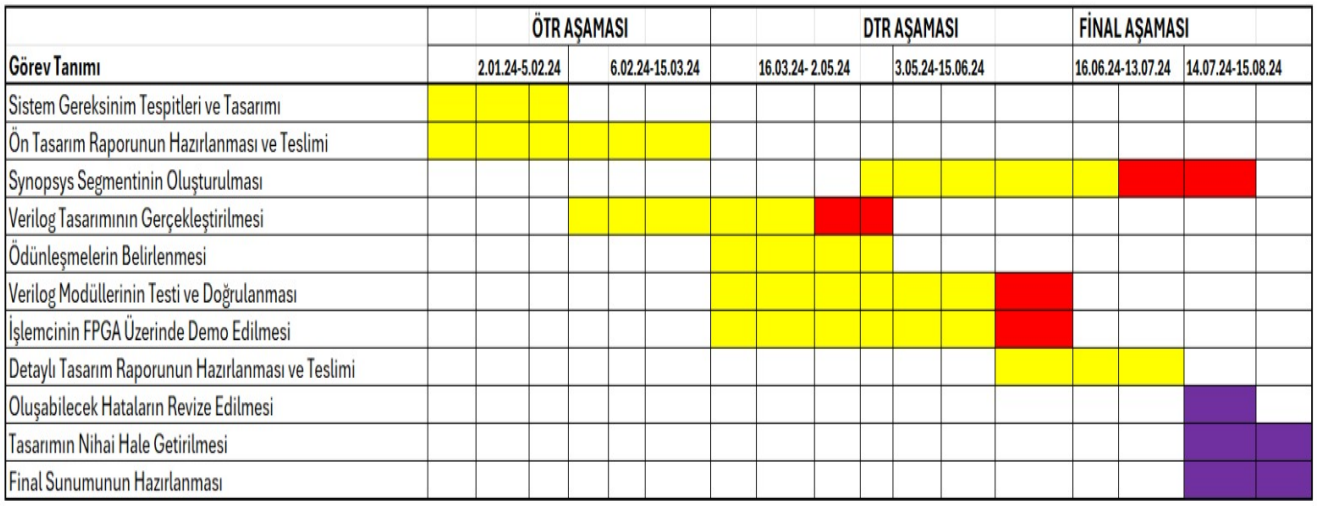
```
int main()
{
  FILE *f;
  for (int k = 0; k < 1; k++)
  {
    char filename[128];
    snprintf(filename, 128, "testdata_%d.hex", k);
    f = fopen(filename, "w");
    for (int i = 0; i < 10000; i++)
    {
      uint32_t din_insn;
      uint32_t din_rs1 = xorshift32();
      uint32_t din_rs2 = xorshift32();
      uint32_t dout_rd;
      switch (xorshift32() % 21)
      {
      case 3: // CLZ
        din_insn = 0x60001013;
        dout_rd = rv32b::clz(din_rs1);
        break;
      case 4: // CTZ
        din_insn = 0x60101013;
        dout_rd = rv32b::ctz(din_rs1);
        break;
      case 5: // PCNT
        din_insn = 0x60201013;
        dout_rd = rv32b::pcnt(din_rs1);
        break;
      case 6: // SLL
        din_insn = 0x00001033;
        dout_rd = rv32b::sll(din_rs1, din_rs2);
        break;
      case 7: // SRL
        din_insn = 0x00005033;
        dout_rd = rv32b::srl(din_rs1, din_rs2);
        break;
      case 8: // SRA
        din_insn = 0x40005033;
        dout_rd = rv32b::sra(din_rs1, din_rs2);
        break;
      case 9: // SLO
        din_insn = 0x20001033;
        dout_rd = rv32b::slo(din_rs1, din_rs2);
        break;
      case 10: // SRO
        din_insn = 0x20005033;
        dout_rd = rv32b::sro(din_rs1, din_rs2);
        break;
      case 11: // ROL
        din_insn = 0x60001033;
        dout_rd = rv32b::rol(din_rs1, din_rs2);
        break;
      case 12: // ROR
        din_insn = 0x60005033;
        dout_rd = rv32b::ror(din_rs1, din_rs2);
        break;
      case 13: // MIN
        din_insn = 0x0a004033;
        dout_rd = rv32b::min(din_rs1, din_rs2);
        break;
      case 14: // MAX
        din_insn = 0x0a005033;
        dout_rd = rv32b::max(din_rs1, din_rs2);
        break;
      case 15: // MINU
        din_insn = 0x0a006033;
        dout_rd = rv32b::minu(din_rs1, din_rs2);
        break;
      case 16: // MAXU
        din_insn = 0x0a007033;
        dout_rd = rv32b::maxu(din_rs1, din_rs2);
        break;
      case 17: // ANDN
        din_insn = 0x40007033;
        dout_rd = rv32b::andn(din_rs1, din_rs2);
        break;
      case 18: // ORN
        din_insn = 0x40006033;
        dout_rd = rv32b::orn(din_rs1, din_rs2);
        break;
      case 19: // XNOR
        din_insn = 0x40004033;
        dout_rd = rv32b::xnor(din_rs1, din_rs2);
        break;

      }
      fprintf(f, "%08llx_%08llx_%08llx_%08llx\n", (long long)din_insn,
      (long long)din_rs1, (long long)din_rs2, (long long)dout_rd);
      }
      fclose(f);
      }

      return 0;
      }
```

Şekil 15

5.2 Çevresel Birimler Testleri

UART birimi TileLink arayüzünden bağımsız olarak farklı bir Verilog modülü ve yarışma komitesi tarafından verilen test algoritması ile detaylı bir şekilde test edilmiştir. Ayrıca, TileLink arayüzünün bir parçası olarak üst blokta yer alacak şekilde UART birimi tekrar test edilmiştir. Bu test senaryoları, UART'ın kontrol ve durum yazmaçlarının çeşitli durumlarını, ana belleğe veri yazma ve okuma işlemlerini, hata durumlarının tespitini ve test edilmesini içermektedir.

6. İŞ PLANI



Şekil 16

Sarı Renk → Başarılı
Kırmızı → Tamamlanmamış (Bitmek Üzere)
Mor → Yapılması Planlanan

Not: Snopsy geç verilmesinden kaynaklı yetişmedi. Verilog tasarımında atomik bellek operasyonları birimlerinde hata oluştuğu için eksik kaldı. Bundan dolayı Test ve Doğrulanma bölümünde ve FPGA Üzerinde Demo Edilmesi bölümünde de eksiklik bulunuyor.

7. KAYNAKÇA

[1] Terpstra, W. W. (2017, November). TileLink: A Free And Open Source, High Performance Scalable Cache Coherent Fabric Designed for RISC-V. In Proc. 7th RISC-V Workshop.

[2] Kommuru, H. B., & Mahmoodi, H. (2009). ASIC design flow tutorial using synopsys tools. Nano-Electronics & Computing Research Lab, School of Engineering, San Francisco State University San Francisco, CA, Spring.

[3] de Oliveira, A. B., Tambara, L. A., Benevenuti, F., Benites, L. A., Added, N., Aguiar, V. A., ... & Kastensmidt, F. L. (2020). Evaluating soft core RISC-V processor in SRAM-based FPGA under radiation effects. *IEEE Transactions on Nuclear Science*, *67*(7), 1503-1510.

[4] Zhao, X. (2024). RISC-V-based Flexible Integer Unit Design and Evaluation.

[5] Sun, R., Liu, H., Zhang, R., & Qu, J. (2023, October). Design and Implementation of RISC-V Based Pipelined Multiplier. In *Journal of Physics: Conference Series* (Vol. 2625, No. 1, p. 012006). IOP Publishing.

[6] Chen, J. (2020). Hardware Acceleration for Elementary Functions and RISC-V Processor. McGill University (Canada).

[7] Gao, Z., Zhao, L., & Chen, H. (2022, June). A trigonometric function instruction set extension method based on RISC-V. In *2022 IEEE/ACIS 22nd International Conference on Computer and Information Science (ICIS)* (pp. 119-126). IEEE.

[8] Takagi, N., Kadowaki, S., & Takagi, K. (2005, June). A hardware algorithm for integer division. In 17th IEEE Symposium on Computer Arithmetic (ARITH'05) (pp. 140-146). IEEE.

[9] McFarling, S. (1993). *Combining branch predictors* (Vol. 49). Technical Report TN-36, Digital Western Research Laboratory.

[10] Kim, I., Jun, J., Na, Y., & Kim, S. W. (2015). Design of a G-Share branch predictor for EISC processor. *IEIE Transactions on Smart Processing and Computing*, *4*(5), 366-370.

[11] Asgharzadeh, A., Cebrian, J. M., Perais, A., Kaxiras, S., & Ros, A. (2022, June). Free atomics: hardware atomic operations without fences. In Proceedings of the 49th Annual International Symposium on Computer Architecture (pp. 14-26).

[12] Jain, V., Sharma, A., & Bezerra, E. A. (2020, April). Implementation and extension of bit manipulation instruction on RISC-V architecture using FPGA. In 2020 IEEE 9th International Conference on Communication Systems and Network Technologies (CSNT) (pp. 167-172). IEEE.

[13] Brehob, M., & Enbody, R. (1999). An analytical model of locality and caching. *Michigan State University, Department of Computer Science and Engineering MSU-CSE-99-31*.

[14] Gadde, D. N., Kumari, S., & Kumar, A. Effective Design Verification–Constrained Random with Python and Cocotb.

[15] Snyder, W. (2004, June). Verilator and systemperl. In North American SystemC Users' Group, Design Automation Conference (Vol. 79, pp. 122-148).

[16] "Icarus verilog," https://github.com/steveicarus/iverilog.

[17] "Spike, a risc-v isa simulator," https://github.com/riscv-software-src/riscv-isa-sim.