TEKNOFEST

HAVACILIK, UZAY VE TEKNOLOJİ FESTİVALİ

ÇİP TASARIM YARIŞMASI

SAYISAL TASARIM KATEGORİSİ

DETAY TASARIM RAPORU

TAKIM ADI

NONLIMITED TEAM

PROJE ADI

NONLIMITED-V

BAŞVURU ID

405906

İçindekiler

1. TEMEL TASARIM ÖZETİ 4
2. PROJE MEVCUT DURUM DEĞERLENDİRMESİ 4
3. PROJE DETAY TASARIMI (45 PUAN) 5
3.1. Sistem Mimarisi 5
3.2. Tasarım Detayı 6
3.2.1. Çekirdek Tasarımı 6
3.2.1.1. Yazmaç Öbeği 6
3.2.1.2. Kodlayıcılar (Encoders) 6
3.2.1.3. Anlık Değer Ayrıştırıcısı (Immediate Value Extractor) 6
3.2.1.4. Aritmetik Mantık Birimi (Arithmetic Logic Unit) 7
3.2.1.5. Sıkıştırılmış Kod Çözücü 8
3.2.1.6. ROM 9
3.2.1.7. RAM 9
3.2.1.8. İşlemci Performansını Arttıracak Tasarımlar 9
3.2.1.9. Boru Hattı Tasarımı 9
3.2.1.10. Harvard Mimarisi Tasarımı 10
3.2.2. Bellek Tasarımları 11
3.2.2.1. L2 Önbelleğin Matematiksel Hesaplaması 13
3.2.2.2. L1 Önbelleklerin Matematiksel Hesaplaması 13
3.2.2.3. Yazma Politikaları 13
3.2.3. Çevre Birimleri Tasarımları 13
4. ÇİP TASARIM AKIŞI 14
5. TEST 15
5.1. RTL Tasarımı Test ve Doğrulaması 15
5.1.1. Çekirdek Testleri 15
5.1.2. Bellek Organizasyonu Testi 15
5.1.3. Karıştırma Testleri 16
5.1.4. Çevre Birimleri Testleri 16
5.2. Sentez Sonrası Test 16
5.3. Serim Sonrası Test 16
6. TAKIM ORGANİZASYONU (5 PUAN) 17
6.1. Takım Organizasyonu 17
6.2. Görev Dağılımı 17
7. İŞ PLANI ve RİSK PLANLAMASI (5 PUAN) 17
8. KAYNAKÇA (5 PUAN) 19

Figür Listesi

Şekil 1: Tasarımın Genel Blok Şeması 5
Şekil 2: Ayrıntılı Çekirdek Şematiği 6
Şekil 3: Bölme Algoritması Örnek Gösterimi 7
Şekil 4: Hamming Mesafe Algoritma Şematiği 8
Şekil 5: Bellek Organizasyonu Şematiği 11
Şekil 6: Bellek Haritası 12
Şekil 7: Çevresel Birim Veri Yolu 14
Şekil 8: OpenLane Akış Diyagramı 14
Şekil 9: Takım Organizasyonu 17
Şekil 10: İş Planı 17

1. TEMEL TASARIM ÖZETİ

Yapılan araştırmalar sonrasında şartnamedeki isterlerin tümünü sağlayacak; Harvard Mimarili, çok vuruşlu (multi-cycle), 5 aşamalı boru hattı sistemli, boru hattı sorunlarını çözümlüyen, yürüt (execution) adımında hızlı&efektif algoritmalar ve sistemlerin bulunduğu, doğrudan ilişkili (direct memory cache) ve kümeli ilişkili (set associative) bellek hiyeraşisi barındıran, çevresel birimler ile çekirdek arasındaki iletişimin kendi tasarladığımız veri iletişimi yolu ile sağlanacak olan işlemci tasarımımızın blok şemasını tasarladık. Tasarımımızı gerçeklemeye, görev tanımlarımıza göre RTL tasarımlarını yaparak devam ettik. RTL tasarımımızı hiyeraşik olarak tasarladık ve her modülün sentez sonrası simulasyonlarını gerçekleştirdik. Statik zaman analizlerimizi de gerçekleştirdikten sonra tasarladığımız modüller, belirlediğimiz kriterlerin altında olduğu zaman tekrardan RTL tasarımımıza dönüp düzenlemelerimizi gerçekleştirdik. RTL tasarımımızı tamamladıktan sonra OpenLane Tasarım Akışı baz alınarak, tasarım isterlerini sağlayacak olan ASIC tasarımımızı gerçekleştirmeye başladık.

2. PROJE MEVCUT DURUM DEĞERLENDİRMESİ

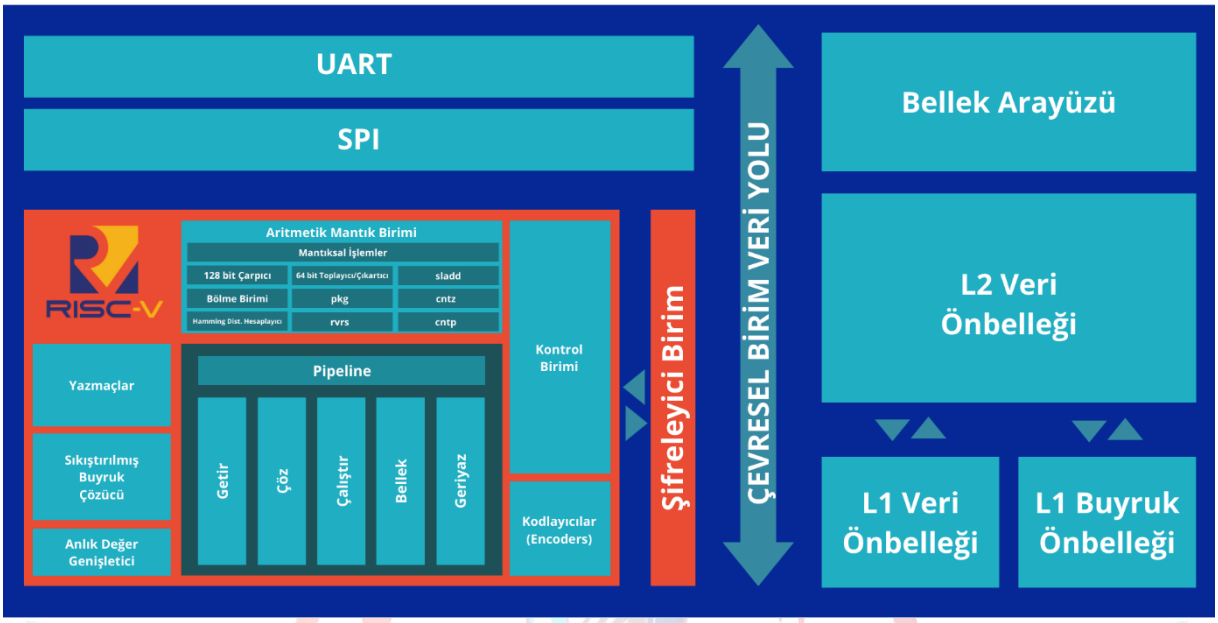
ÖTR'de bahsi geçen hedef tasarımımıza bağlı kalmaya çalışarak projemizi tasarlasakta proje tasarımı esnasında karşımıza çıkan veya öngördüğümüz sorunlardan dolayı bazı yerlerde değişikliklere gittik. Tasarımda yaptığımız değişiklikler şunlardır:

- Proje tasarımı esnasında bellek tasarımında kaynak tüketimi/utilization değerlerinin çok yüksek çıkması sonucu L1 önbelleklerde Kümeli İlişkili Önbellek (Set Associative Cache) kullanmak yerine Doğrudan Eşleşmeli Önbellek (Direct Memory Cache) tasarımını tercih etmeye karar kıldık.
- ÖTR aşamasında ki veri akışını anlatmak için sunulan hatalı şematik değiştirilerek şifreleyici birimi, L1-L2 bellekleri ve ana bellek arasındaki veri akışı Tübitak mentörlerinden alınan bilgilere ve yapılan araştırmalara istinaden yeniden düzenlendi.
- ÖTR'de sunduğumuz çekirdek mimarisi tasarımı diyagramının karışık olduğu geri dönütü üzerine, çekirdek mimarisi tasarımımızın diyagramını sade ve anlaşılır şekilde düzenledik.
- Wrapper dosyası paylaşıldıktan sonra ÖTR'de sunulan Genel Tasarım Blok Şeması'nda bulunan Bellek Arayüzü ve Buyruk Arayüzü blokları şematikten çıkartıldı.
- ÖTR'de "Çip Tasarım Akışı" bölümünde OpenLane Akışı'nda eksiklerimizi gidermek için OpenLane üzerine daha yoğun çalışıldı ve tasarımımızın fiziksel/backend tarafı tasarlanmaya başlandı.

Ek olarak ÖTR boru hattı şematiğinde gösterdiğimiz AMB, kontrol birimi gibi çekirdek içi birimler birden fazla modülü kapsayacak şekilde tasarlandı. RV64IMC çekirdek tasarımı için gerekli olan Sıkıştırılmış Buyruk Çözücü (Compressed Decoder) eklendi. CPU tarafından kontrol edilecek (AMB, Anlık Genişletme Birimi, Mux'lar vb. gibi) birimlerin sinyalleri kod çözücüler (encoders) yardımıyla sağlandı.

3. PROJE DETAY TASARIMI (45 PUAN)

3.1. Sistem Mimarisi

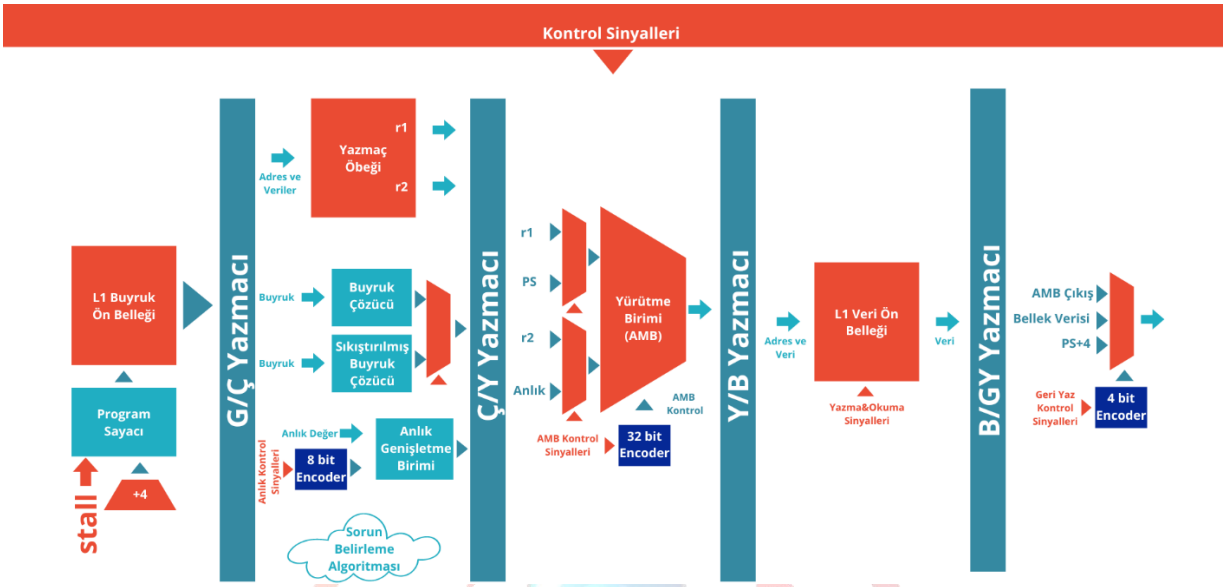


Şekil 1: Tasarımın Genel Blok Şeması

Şartname'de istenilen RV64IMC çekirdeği; boru hattı yöntemi kullanarak, çok vuruşlu (multi-cycle) ve I-M-C uzantılarına uygun olarak tasarlanmıştır. Çekirdek içerisinde RISC-V'ın standart yapısına uygun olarak CPU, Yaçmaç Öbeği (Register File) ve Anlık Genişletme Birimine ek olarak sıkıştırılmış buyrukların çözülmesi için Sıkıştırılmış Buyruk Çözücü ve kontrol sinyallerinin düzenlenmesi için kodlayıcılar (encoders) tasarlanmıştır. AMB'de çarpma işleminde Wallace Tree Yöntemi, bölme işleminde Geliştirilmiş Booth's Divider Algoritması ve 64 bit girdi içersinde bit sayma yürütme işlemlerinde iterasyonlu efektif toplama yöntemi kullanılmıştır. Bellekler, isterlere uygun olarak boyutladırılıp ve haritalandırılıp L1 önbellekler için Doğrudan İlişkili Bellek (Direct Memory Cache), L2 önbellek için ise 4 yollu Kümeli İlişkili Bellek (4 Way Set Associative) olarak tasarlanmıştır. Şifreleyici Birim L1 önbellekleri ile çekirdek arasında konumlandırılmıştır. UART ve SPI Şartname'deki isterlere uygun olarak tasarlanmış bellekte ayrılan adreslere yazmaçları haritalanmıştır. Sistem mimarisi hiyeraşik olarak tasarlanmış olup sentezlenebilir bir tasarım gerçekleştirilmiştir. Sistem mimarimizde ASIC tasarım akışına uygun olarak tasarlanmaya özen gösterilmiş ve kompleks tasarım metodlarından uzak durulmuştur.

3.2. Tasarım Detayı

3.2.1. Çekirdek Tasarımı



Şekil 2: Ayrıntılı Çekirdek Şematiği

3.2.1.1. Yazmaç Öbeği

"Register_File.v" modülü işlemcinin yazmaçlarının tutulduğu, bu yazmaçlara okuma ve yazma işlemlerinin gerçekleşmesini sağlayan modüldür. R1 ve R2 kaynak yazmaçları içeriği okunacak yazmacın, yazmaç öbeğindeki yerini belirten 5 bitlik sayılardır. RD kaynak yazmacı ise yazılacak yazmacın hangi yazmaç olduğunu belirten 5 bitlik bir sayıdır. RV64 mimarili bir işlemci olduğu için Yazmaç Öbeği'ndeki yazçmaçların boyutu 64 bittir. Okuma işlemi için gelen buyruğun tipine göre, R-Tip, S-Tip ve B-Tip buyruklarda okunacak iki yazmaçın (R1 ve R2), I-Tip buyruklarda ise okunacak tek yazmacın (R1), hangi yazmaç olduğu bilgisi bulunur. U-Tip ve UJ-Tip buyruklar yazmaçlardan okuma yapmazlar. Yazma işleminde ise gelen buyruğun tipine göre, R-Tip, I-Tip, U-Tip ve UJ-Tip buyruklarda hangi yazmaca (RD) yazılacağı bilgisi bulunur. S-Tip ve B-Tip buyruklar yazmaçlara yazma yapmazlar. Özel buyruklar olan "hmdst", "pkg", "sladd", "cntz" ve "cntp" R-Tip buyruklardır ve ek olarak "rvrs" buyruğu I-Tip buyruktur.

3.2.1.2. Kodlayıcılar (Encoders)

Kodlayıcılar girdilerindeki değeri "1" olan telin sırasının ikilik sistemde karşılığını veren devrelerdir. Öncelikli Kodlayıcılar (Priority Encoder) ise birden fazla girişin 1 olması durumunda yüksek öncelikli girişi dikkate alıp diğer giriş değerlerini önemsemeyen kodlayıcılardır. CPU'nun iç tasarımında 4-2'lik, 8-3'lük ve 32-5'lik kodlayıcılar kullanılmıştır. Bu kodlayıcıların tamamı Öncelikli Kodlayıcılardır. Kodlayıcılar AMB (32-5 Kodlayıcı), Anlık Genişletme Birimi (8-3 Kodlayıcı) modülleri ve Geri Yaz (4-2 Kodlayıcı) aşamasındaki kontrol sinyallerinin belirlenmesi için kullanılmıştır.

3.2.1.3. Anlık Değer Ayrıştırıcısı (Immediate Value Extractor)

RISC-V buyruk kümesinde anlık tipteki buyrukların sahip olduğu sayısal değerler, buyruğun tipine göre farklı şekillerde dağıtılmıştır. Bu sebeple anlık değerin buyruk içinden düzgünce çıkartılması gerekir. Anlık değerler buyruk içinde hep aynı format ve sıra ile bulunmaz ve buyruk tiplerine göre ayrıştırılmaları gerekir. Buyruk tiplerine göre verilen buyruktaki anlık değeri ayrıştırıp işlemlerde kullanmak üzere 64 bitlik bir sayı haline getirmek için "Immediate_Extractor.v" modülü tasarlanmıştır. Anlık değerler en yüksek değerli bitine göre genişletilmiştir (sign extended). Örneğin 5 bitle 01000

şeklinde gösterilen 8 sayısı 000...001000 olarak 64 bite genişletilirken 5 bitle 11000 şeklinde gösterilen -8 sayısı en yüksek bitteki değerle genişletilerek 111...111000 şeklinde 64 bitlik bir sayıya çevirilmiştir. Böylece sayının işareti korunmuştur.

3.2.1.4. Aritmetik Mantık Birimi (Arithmetic Logic Unit)

Aritmetik Mantık Birimi (AMB) X ve Y şeklinde 64 bitlik iki girdi alıp seçilen işlem türüne göre bir çıktı üreten modüldür. Bu işlemci tasarımında RV64IMCX çekirdek tasarımındaki buyrukların kullanacağı hesaplamaları karşılayacak şekilde 19 adet operasyon tipi belirlenmiştir. AMB sonuç (RESULT) yazmacının 128 bit olma sebebi 64 bitlik iki sayıyla yapılan çarpma işleminin sonucunun 128 bitlik bir sayı olabilmesi ve "mulh" buyruğunun bu sonucun 127-64 bitlerini, mul buyruğunun ise bu sonucun 63-0 bitlerini almasıdır. Diğer tüm buyruklarda 65. bite taşma ihtimali olmadığı için sadece mulh buyruğu geldiğinde 127-64 bit arası değerler çıkışa aktarılmıştır. Şartnamede bahsi geçen özel buyrukların modülleri AMB'nin alt modülleri olarak hiyeraşik bir şekilde tasarlanmıştır.

a) **Bölme Birimi:** Donanım sistemlerinde bölme işleminin temeli genellikle çıkarma işlemine dayanmaktadır. Bölme işleminde birçok yöntem vardır. Bu yöntemler için gerekli olan saat çevrim sayısı genellikle bölünen bitlerinin sayısına eşittir. Bunun için ana odak noktamız kullanılan kaynak değerleri olmuştur. Birçok yöntemde birden fazla kaydırıcı varken, bizim tasarımımızda tek kaydırmalı yazmaç bulunmaktadır. Yöntemimiz şu şekildedir:

7 (0111) / 2 (0010) = 3 (0011) R 1 (0001)

Step	Remainder	Divisor	Rem-Div
Initial	0000 0111	0010	Always < 0
Shift	0000 1110	0010	< 0
1	0001 1100	0010	< 0
2	0011 1000	0010	0011-0010 > 0
2	0001 1000	0010	
3	0011 0001	0010	0011-0010 > 0
3	0001 0001	0010	
4	0010 0011	0010	
Final	R1	3	

Kalan: 0001 Bölüm

Şekil 3: Bölme Algoritması Örnek Gösterimi

1- Kalanın sol yarısını (MSB tarafı) 0'lanır ve bölünen sayı kaydıraçlı yazmacın sağ yarısına (LSB) konur. Ardından bir bit sola kaydırma yapılır.

2- Kaydıraçlı yazmacın sol yarısından (MSB tarafı) böleni çıkarılır. Bu sonuç 0'dan küçükse kalanı sola kaydırılır ve bir sonraki adıma geçilir.

3- Bu sonuç sıfırdan büyük veya sıfıra eşitse, kaydıraçlı yazmacın sol yarısına, çıkartma işleminin sonucu yazılır ve kalan yazmacı sola bir bit kaydırılır. Ardından kalanın en sağdaki biti 1'e ayarlanır.

4- Son tekrara gelindiğinde (N=Bölünen Bit Genişliği+1 olduğunda) kaydıraçlı yazmacın sağ yarısı (LSB) alınır ve bölüm çıkışına aktarılır. Kaydıraçlı yazmacın sol yarısı (MSB) sağa kaydırılır ve bu değer kalan sonucuna aktarılır.

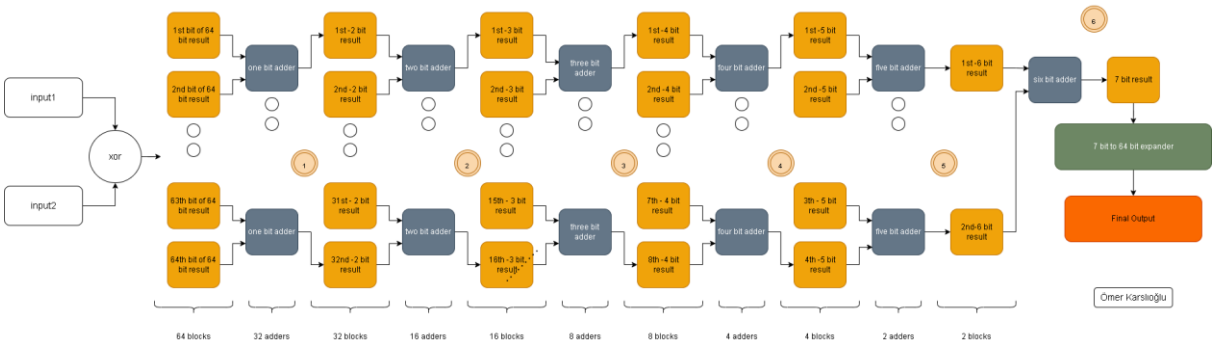
b) **128 Bit Çarpıcı:** Mantıksal sistemlerde çarpma işlemi için birçok yöntem geliştirilmiştir. Bunlardan bazıları şunlardır: *Wallace Tree Yöntemi, Dadda Yöntemi, Klasik Yöntem, Booth'un Algoritması.*

Booth'un Algoritması, 64 bit sayı x 64 bit sayı işlemi gerçekleştirirken 64 saat çevrimi ile birden çok yinelemeyi işleyeceği için tercih edilmez. Diğer yöntemler birbirleri ile kıyaslandığında[1] [2]. Wallace

Tree Yöntemi [3] [4] [5] güç tüketimi, kaynak kullanımı ve hız konusunda daha dengeli ve efektif olduğuna karar kılarak Wallace Tree Yöntemi'ni kullanarak çarpma modülümüzü hazırladık.

c) **Toplama-Çıkarma Birimi:** Bir bitlik tam toplayıcılar (full adders)'dan oluşturduğumuz toplama ve çıkarma işlemlerinin seçme biti (Sel_i)'ye göre seçilerek gerçekleşmesini sağlayan tasarımımızdır.

d) **Hamming Mesafesi Hesaplama Birimi:** Bu birimde girilen iki sayı XOR işlemine sokulduktan sonra çıkan 64 bitlik sonucun tüm bitleri toplanıp sonuca atarılır. Bu süreçte hız değerlerini artırmak için toplama işlemi 6 iterasyonda gerçekleştirilmiş ve aynı zamanda kaynak tüketimi düşük tutulmuştur.



Şekil 4: Hamming Mesafe Algoritma Şematiği

XOR işlemi sonucunda elde edilen çıktının bitleri bir bitlik toplayıcı sayesinde toplanır, ardından çıkan iki bitlik (cary biti + sonuç biti) sonuçlar iki bitlik toplayıcılarda toplanır. Bu sistem bu şekilde son iterasyona kadar devam eder. Son iterasyonda altı bitlik toplayıcıdan çıkan 7 bitlik sonuç 64 bite genişletilir ve sonuca aktarılır. Böylelikle altı iterasyonda işlem tamamlanmış olur.

e) **Diğer Özel Buyrukların Yürütme Birimleri Hakkında:** "pkg-rvrs-sladd" özel buyruklarının yürütme birimleri ise isterlere uygun bir şekilde tasarlanmıştır.
"cntz" özel buyruğunun yürütme birimi lookup table olarak tasarlanmıştır.
"cntp" özel buyruğunun yürütme birimi, işlenmek üzere giren sayının ikilik sistemdeki karşılığında 1 olan bitlerini Hamming Mesafesi Hesaplama Birimi'nde uygulanan iterasyonlu toplama sistemi ile aynı tasarlanmıştır.

3.2.1.5. Sıkıştırılmış Kod Çözücü

Bu modül buyruk getir aşamasında alındıktan sonra buyruğun sıkıştırılmış mı yoksa normal buyruk mu olduğuna karar veren modüldür. Klasik olarak 32-bit buyrukların ilk iki biti "11"dir sıkıştırılmış buyruklar ise "00", "01" veya "10" bitleri ile başlar. Buyruk eğer sıkıştırılmış bir buyruksa bu buyruğu RISC-V talimatlarına uygun bir şekilde 32-bit karşılığına genişleterek çıkışına verir. Bu sayede sıkıştırılmış buyruğu normal uzunlukta ki bir buyruğa dönüştürmüş oluruz. Sıkıştırılmış buyruklar bizim için önem taşır. Öncelikli olarak kod boyutunu azaltması ve kaynak tüketimi yönünden daha etkili olmaları gibi birçok faydaları bulunmaktadır [3].

3.2.1.6. ROM

ROM modülü, işlemcinin adresini belirterek istediği program buyruklarını işlemciye getirecek(fetch) olan modüldür. Bu bellek işlemci tarafından salt okunurdur.

3.2.1.7. RAM

RAM modülü programın çalışması esnasında geçici bellekte veri saklama veya okuma işlemlerinde kullanılacak modüldür.

Not: RAM ve ROM modüllerinin tasarımına "Bellek Tasarımı" başlığı altında daha detaylı değinilecektir.

3.2.1.8. İşlemci Performansını Arttıracak Tasarımlar

İşlemcide kullanılmak üzere iki adet tasarım modeli kurgulanıp uygulanmıştır. Bu modeller; Boru Hattı ve Harvard mimarisidir. Boru Hattı yöntemi aynı vuruşta birden fazla işlem yapılmasına olanak sağlar. Örnek olarak 1. buyruk geri-yaz aşamasına geldiğinde 2. buyruk bellek aşamasında, 3. buyruk yürüt, 4. buyruk çöz, 5. buyruk getir aşamasında olur. Bu yöntemle klasik tek vuruşluk işlemcide tek vuruşta bir buyruk işleme alınırken, çok vuruşluk işlemcide birden fazla buyruk farklı aşamalar(getir-çöz-yürüt-bellek-geri yaz) tarafından işlenir. Harvard mimarisi ise işlemcinin bilgisayar organizasyonu ile alakalı olup işlemcinin diğer dış modüllerle (ROM ve RAM gibi) çalışma biçimini tanımlar.

3.2.1.9. Boru Hattı Tasarımı

Boru Hattı yöntemi uygulanmış işlemciler yaptıkları işlemleri parçalara ayırarak bir çevrimde birden fazla işin aynı anda yapılmasını sağlar. Örneğin bir saat vuruşunda hem sıradaki buyruk getirilip hem de önceden getirilmiş bir buyruk ALU'da işleme tabi tutulabilir. Böylece bir çevrimin tamamlanması için beklenilmesi gereken asgari süre azaltılmış olmaktadır. Bunun sonucunda da daha yüksek buyruk başına çevrim hızlarına çıkılabilmektedir.

Buyruk başına çevrim değeri yazılan algoritmaya göre değişikiklik arz etmektedir. Örneğin toplama işlemi bir saat çevrimi gerçekleşen bir işlemken, bölme işleminin gerçekleşmesi 65 saat çevrimine ihtiyaç duymaktadır. Boru hattında yürütme işlemi meşgulken başka bir yürütme işlemi gerçekleşemediğinden, yürütme aşamasında birden fazla saat çevrimine ihtiyaç duyan buyruklarda boru hattı durdurulur. Bundan dolayı performans kaybı oluşabilir ve buyruk başına çevrim artabilir. Tasarımımızda bir saat çevriminde en fazla 5 adet buyruk işlenebilir.

<u>İşlemcideki Boru Hattı Aşamaları:</u>

Tasarladığımız işlemci 5 aşamalı boru hattına sahiptir. Bu aşamalar ise:

a. Getir (Fetch): İşlemci ROM bellekten bir sonraki buyruğu getirip bir yazmaca kaydeder. Bu buyruk boru hattının sonuna kadar her aşamaya tek tek aktarılır.
b. Çöz (Decode): Getirilen buyruk çözümlenir ve parametrelerine ayırılır.
c. Yürüt (Execute): Çözümlenen buyruğun yürütülmesi yani sonucun hesaplanmasının gerçekleştiği aşamadır.
d. Bellek (Memory): Bellekten veri okunan veya belleğe veri yazılan aşamadır.
e. Geri Yaz (Writeback): Yürüt veya Bellek aşamalarından elde edilen sonuçlar eğer bir yazmaca kaydedilecekse bu aşamada kaydedilir.

Boru Hattı Yöntemi Sorunları (Pipeline Hazards):
Boru hattı ile beraber şu 3 adet sorunla karşılaşırız:

a. Veri Bağımlılığı (Data Hazard) : Bu sorun bir buyruğun girdi olarak kullanacağı yazmaç verisinin bir veya iki önceki buyrukta değiştirilmiş olmasıdır. İşlenen buyruğun sonucunun ilgili yazmaca yazılması için üç saat vuruş geçip Geri Yaz (Writeback) aşamasının bitmiş olması gerekir. Veri bağımlılığı sorununda bir veya iki sonraki buyruk geldiği zaman henüz geri yazılmamış bu veriye ihtiyaç duyar. Bu sorunu gidermek için iki yöntem uygulanabilir. Boru hattına üç çevrim boyunca bir buyruk almayıp Bellek ve Geri Yaz aşamalarını ilerletip diğer aşamaları bekletmek, veya Yürüt işleminden elde edilen ve bir sonraki aşamaya aktarılacak olan sonucun aynı zamanda Yürüt aşamasına tekrar geri verilmesidir. İşlemcimizin tasarımında Veri Bağımlılığı sorununun giderilmesi için ihtiyaç olan verinin ilgili aşamalardan direkt olarak alınıp kullanılması yöntemi uygulanmıştır.

b. Kontrol Sorunu (Control Hazard): Bu sorun dallanma (branch) buyruklarında dallanmaya karar verilme aşamasının Yürütme yani üçüncü aşama olmasından kaynaklanan bir sorundur. Dallanma buyruğunda dallanmanın yapılıp yapılmayacağına Yürütme aşamasında karar verilir ve Program Sayacının yeni değeri elde edilmiş olur. Fakat bu aşamaya gelene kadar boru hattına iki adet yürütülmemesi gereken buyruk dahil olmuş olabilir.

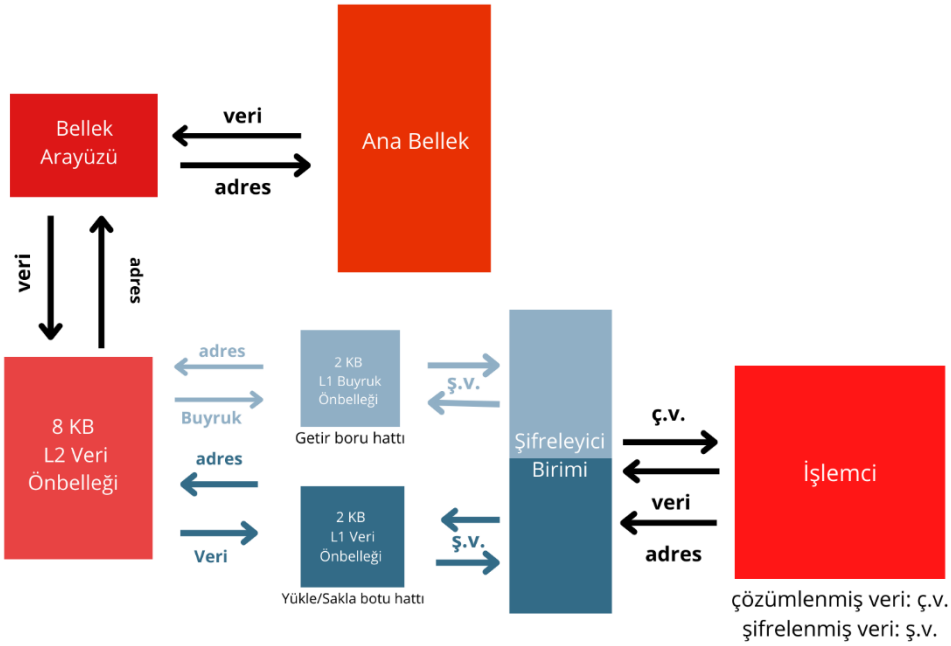
Bu sorunu gidermek için iki yöntem uygulanabilir. Birincisi Boru Hattına bir dallanma buyruğu girdiğinde dallanma kararı belli olana kadar, yani iki çevrim boyunca boru hattını bekletmek; diğeri ise Dallanma Tahmini (Branch Prediction) yaparak dallanma kararını önceden verilmiş olan kararlara bakarak verip, örneğin iki aşamalı bir sonlu durum makinesi kullanarak, dallanma buyruğu daha Yürüt aşamasına gelmeden Program Sayacını ilgili yere çekmektir. Bu yöntem beraberinde yanlış tahmin etme sorununu oluşturur. Bu sorunun çözümü için ise yanlış tahmin edildiği anlaşılan buyruktan sonra işlenen buyrukları geri alma işlemi uygulanmalıdır. Yanlış tahminler işlemcinin performansını olumsuz etkileyeceğinden işlemcimizin tasarımında Kontrol Sorununun giderilmesi için dallanma tahmini yapılmamış, boru hattını iki çevrim boyunca bekletme (stall) yöntemi uygulanmıştır.

c. Bellek Yükleme Gecikmesi (Load Stall) : Bu sorun yükleme buyruğundan sonra gelen buyruğun bellekten yüklenecek olan veriye bağımlı olma sorunudur. Bu sorun da aynı Veri Bağımlılığı sorunu gibi bekleyerek ya da ara tel çekilerek çözülebilir. Fakat Veri Bağımlılığına ek olarak Bellek aşamasından Yürüt aşamasına direkt olarak tel çekilse bile yükleme buyruğunun Bellek aşamasına gelmesi için bir çevrim gecikme kaçınılmazdır.İşlemcimizin tasarımında Bellek Yükleme Gecikmesi sorununun çözümü için hem Bellek aşamasından tel çekilmiş, hem de boru hattı bir çevrim bekletilmiştir.

3.2.1.10. Harvard Mimarisi Tasarımı

Von Neumann Mimarisi'nde hem buyruklar hem de veriler aynı bellekte tutulur. İşlemci RAM bellekteki buyrukları sırasıyla okuyup işler. İşlemci gerektiğinde RAM 'e veri yazar ve okur. Eğer program buyruklarının okunacağı bellek, verilerin depolandığı bellekten ayrı tutulacaksa bu mimariye de Harvard Mimarisi adı verilir. Mimarimiz Harvard Mimarisi'ne göre tasarlanmıştır, yani program buyruklarının okunacağı ROM ve verilerin saklanıp okunacağı RAM şeklinde iki ayrı belleğimiz olacaktır. Bu sayede buyruk okuma işlemi ile veri okuma-yazma işlemleri bağımsız olup, işlemci bellek okuma-yazma işlemlerinde daha efektif bir performans sergileyecektir.

3.2.2. Bellek Tasarımları



Şekil 5: Bellek Organizasyonu Şematiği

Detay tasarım raporunda önbellek, şifreleme birimi, bellek arayüzü ve ana bellek hakkında bizden beklenen isterler göz önüne alınarak hazırladığımız bellek organizasyonu görseldeki gibidir.

Bu bölümde öncelikle, tercih ettiğimiz bellek organizasyonuna göre veri akışının nasıl gerçekleştiğine değinilecektir. Daha sonra tasarımda tercih ettiğimiz seçimlere ve politikalara değinilecektir.

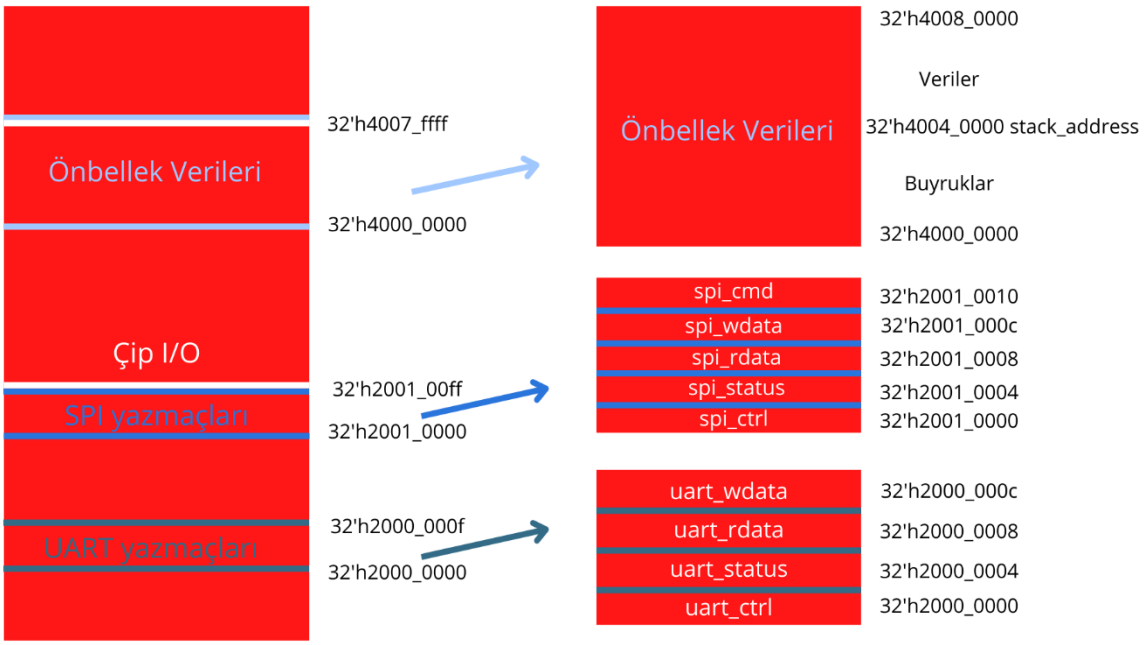
İşlemci ile bellekler arasında iki türlü iletişim vardır: İşlemci belleklere ya veri yazar ya da işlemci belleklerden veri alır. Öncelikle işlemcinin belleklerden veri istediği durumdaki bellek akışını inceleyelim:

İşlemci talep ettiği verinin ana bellekte kaydolduğu adres değerini istek olarak gönderir. Veri önbelleklerde; adresten elde edilen L1 ve L2 önbellekler için belirlenen index bitleri ve tag bitlerine göre aranır. Index ve tag bitlerine göre, veri L1 önbelleklerden başlamak üzere belleklerde aranır. L1 önbelleklerde veri bulunamazsa, L2 veri önbelleğine çıkılır. Veri L2 önbelleğinde de bulunamazsa arayüz vasıtasıyla ana belleğe çıkılır ve veri işlemciye getirilir.

İşlemciden önbelleklere veri kaydedileceği zaman kaydedilecek veri ve adres işlemciden sinyal olarak çıkar. Adresin L1 önbellek index bitlerinin işaret ettiği L1 setinin kirli biti (dirty bit) 0 ise veri L1 önbelleğe yazılır. Eğer kirli biti 1 ise ve kirli veri için L2 önbellekte belirlenen kısımda boş yer var ise; L1 önbelleğindeki kirli veri L2'ye kaydedilir, L1 önbellekteki kirli bit 0 olarak güncellenir ve güncel veri L1'e yazılır. L2 önbellekte kirli verinin haritalandığı bloklarda boşluk yoksa; önce L2'deki eski verilerden en az kullanılanı ana belleğe çıkarılır, daha sonra kirli veri L2'ye kaydedilir, L1 kirli biti 0 olarak değiştirilir ve güncel veri L1'e kaydedilir.

Veri işlemciden L1 önbelleklere giderken S-box ile şifrelenir. L1 önbelleklerden işlemciye giren veriler ise ters S-box ile çözümlenir.

Bellek organizasyonu hakkında verdiğimiz bu kısa bilgiden sonra biraz daha detaya inelim ve tercihlerimiz üzerine konuşalım.



Şekil 6: Bellek Haritası

UART, SPI, önbellek isterlerini ve Çip I/O'larını baz alarak hazırladığımız ana bellek adreslemeleri şekil 6'daki gibidir.

Bu bölümde son olarak ana bellek ve önbellekler arasındaki haritalama (mapping) hesaplamasına ve türlerine değineceğiz. Ana bellek ve önbellekler arasında haritalama yapmak ve bunun nasıl yapılacağını belirlemek bellek organizasyonunun en kritik aşamalarından birisidir. Haritalama yaparak ana bellekteki veriler ile önbelleklerdeki veriler arasında ilişki kurulmuş olunur. Ana belleğin 32'h4000_0000 adresinden 32'h4008_0000 adresine kadar olan kısmının önbelleklere haritalanması bize ister olarak verilmişti. Bu iki adres aralığında toplam tutulabilecek veri boyutunu şu şekilde hesapladık:

1- Bu iki adres değeri arasında toplamda 2^{19} array mevcut. Tam sayı olarak ifade edersek: 524.288.
2- Her bir array de 8 bit yani 1 bayt veri tutulmaktadır. Dolayısıyla ana belleğin hacmi 524.288 bayttır.
3- 524.288'i 1024'e bölerek ana belleğin ön bellekler ile haritalanacak kısmının hacmini 512 KB bulduk.

Sonuç olarak bizden 512KB'lık ana bellek kısmı ile 2 tane 2 KB'lık L1 önbellekleri ve 1 tane 8 KB'lık L2 önbelleğinin map edilmesi isteniyor. Öncelikle 512 KB ana bellek kısmı 8 KB'lık L2 önbelleğe haritalanır. Bu durumda fazla sayıda veri aynı sete haritalanacağından L2 önbelleğini 4 Yollu Kümeli İlişkili (4 Way Set Associative) olarak tasarlamaya karar verdik. 4 Yollu Kümeli İlişki ortalama olarak miss oranını yüzde dörtlerin altına düşürmektedir[6]. Böylelikle aynı sete 5 tane veri yükleme gerekmediği takdirde ana belleğe çıkmak gerekmeyecektir.

3.2.2.1. L2 Önbelleğin Matematiksel Hesaplaması

İsterlere göre L2 önbelleği 8 KB (8192 bayt) olarak belirlenmiştir. 4 Yollu Kümeli İlişkili bellek tipi tercih ettiğimiz için her bir sette (4*64) / 8= 32 bayt data tutulur. L2 önbellek derinliği ise toplam hacmimizi her bir sette tutulacak veri baytına bölerek bulabiliriz:

8192 B / 32 B = 256. Sonuç olarak L2 önbelleği şu şekilde tanımladık: [3:0][63:0] L2cache [255:0]

L1 önbelleklerini ise Doğrudan İlişkili Haritalama (Direct Mapping) mantığını kullanarak haritalamaya karar verdik. L1 önbellek boyutu küçük olduğu için ve birçok open-source uygulamalarda tasarım sadeliğini sağlamak amacıyla L1 önbellekler haritalanırken Doğrudan İlişkili Haritalama uygulandığını gördük. Biz de bu yüzden bu tercihte bulunduk.

3.2.2.2. L1 Önbelleklerin Matematiksel Hesaplaması

İsterlere göre L1 önbellekleri 2 KB (2048 bayt) olarak belirlenmiştir. Doğrudan İlişkili Haritalama tercih ettiğimiz için her bir sette 64 bit (8 bayt) veri bulunacaktır. Önbellek derinliğinin kaç set olduğunu bulmak için toplam hacmimizi her bir sette tutulacak bayt miktarına bölebiliriz:

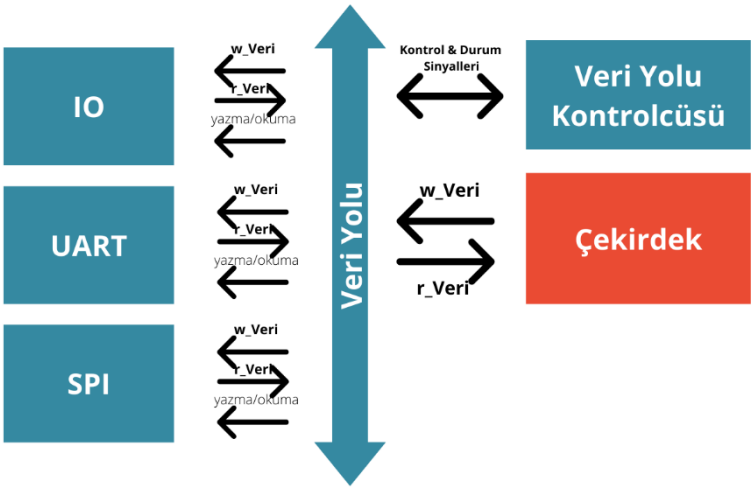
2048 B / 8 B = 256. Sonuç olarak L1 önbelleklerimizi şu şekilde tanımladık: [63:0] L1_DataCache [255:0], [63:0] L1_InstructionCache [255:0].

3.2.2.3. Yazma Politikaları

Daha önceki kısımlarda haritalama yaparken kirli bit kullanacağımızdan bahsetmiştik. Veriyi doğrudan ana belleğe yazmak, bellek veri akışını bir hayli yavaşlatır. Özellikle bu durumu göz önünde bulundurarak yazma sırasında hit durumu oluşursa write-back politikasını kullanmaya karar verdik. Okuma sırasında miss durumu oluşur ise no-write allocate politikasını uygulamayı düşündük. Çünkü eski veriyi ana belleğe çıkarmak ve ana bellekteki veriyi önbelleğe yazmanın karışık bir tasarım oluşturacağından ve getirilen verinin bir daha kullanılmama ihtimali bulunduğunu göze alarak bu kararı verdik. Birtakım verilerin önbelleklere getirilmesi yani write allocate yapılması makul bir uygulama olurdu. Ancak bu ayrımı gerçeklemenin bellek organizasyonunda karışıklığa sebep olacağını düşündüğümüzden write no-allocate politikasını uygulamakta karar kıldık. [7] [8]

3.2.3. Çevre Birimleri Tasarımları

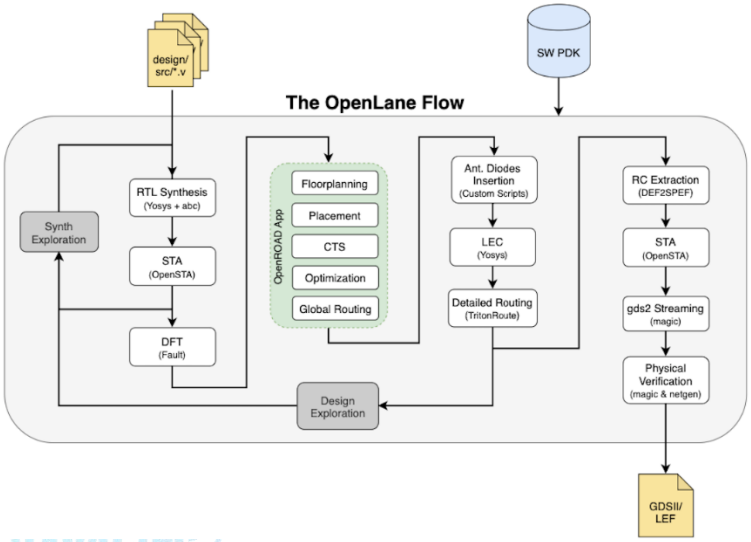
L1 önbellekleri çekirdeğe doğrudan paralel olarak bağlanmıştır. Çevre birimlerinin veri iletişiminin sağlanması için üzerinde çalıştığımız üç protokol yapısı: AXI, Wishbone ve kendi veri yolu prokolümüzdür. Wishbone, özellikle de AXI gibi bus yapıları tasarlandığı takdirde kontrolcülerinin bir wrapper gibi yazılıp sistem tasarımımızda kompleksliğe yol açacağını ön gördüğümüzden kendi veri yolu iletişimimizi sağlamaya karar verdik. Gelen adreslerin yazma-okuma işlemi belirli olduktan sonra ilgili adresteki verinin çekirdeğe ulaşması veya çekirdekten çevresel birimlere aktarmasını sağlayacak kendi iletişim protokolümüzün tasarımına başladık.



Şekil 7: Çevresel Birim Veri Yolu

Bu veri yolu tasarımı, "Bellek Tasarımları" başlığı altında sunmuş olduğumuz bellek haritasına göre gelen adres ilgili çevresel biriminin yaçmaçlarının adres aralığında ise yazılacak veya okunacak olan veri bu veri yolu içerisinden çekirdeğe veya çevresel birimlere ulaşmaktadır.

4. ÇİP TASARIM AKIŞI



Şekil 8: OpenLane Akış Diyagramı

Çip tasarım akışında OpenLane Akış Diyagramı referans olarak kullanılmıştır. Burada Verilog dosyalarının RTL sentezi, Vivado kullanılarak yazılmış ve simüle edilmiştir. OpenLane Akışı ise şu şekilde gerçeklenmektedir: Sentez, yerleşim planı, güç dağıtım ağı, silikonun yerleştirilmesi, saat ağacının sentezi (CTS), yol atama, GDSII dosyasının sentezi ve kontrollerin yapılması şeklindedir.

OpenLane Tasarım Akışı'nda; Magic, Klayout, Netgen, Ngspice ve Xschem gibi araçlar, tasarım ve doğrulama için kullanılmaktadır. Tasarım oluşturulmaya çalışılırken Klayout aracından oldukça faydalanılmış olup, oluşan GDSII dosyaları gözlemlenmiştir. Bunun yanında Ngspice aracındanda simülasyonlar oluşturulmaya çalışılmış fakat hatalar sonucunda, Vivado üzerinde yapılmış olan simülasyonlar tercih edilmiştir. Ayrıca saat ağacının sentezinde TritonCTS aracı kullanılmaya ve openSTA kullanılarak ideal saatin oluşturulması ve zamanlama analizi yapılmaya çalışılmıştır.

Projenin serimi şu aşamada gerçeklenememiş olsa da tasarımlarda ve programı kullanırken karşılaşılan problemler şu şekildedir: Öncelikle “docker: invalid reference format: repository name must be lowercase.” hatasıyla oldukça fazla karşılaşıldı. Bu hata her seferinde “docker ${args_to_docker} run ${args_to_run} image_ref ${cmd_to_exec}” komutu yazılarak çözüldü. Bunun yanında “docker permission denied.” hatası alındığında ise “sudo chmod 666 /var/run/docker.sock” komutu yazılarak çözülmüştür.

Çip tasarım akışı boyunca, OpenLane aracının OpenRoad Github hesabından paylaşılan bilgiler, Berkeley üniversitesinin “Intro to OpenLane and Skywater 130” lab dokümanından, Matt Venn’in “Zero to ASIC Course” ve videolarından, ayrıca Gitter üzerinde kurulan OpenROAD topluluğundan faydalınılmıştır. Bunun yanında Efabless projesinde bulunan Caravel tasarımı da incelenerek fikir oluşturması hedeflendi.

5. TEST

5.1. RTL Tasarımı Test ve Doğrulaması

RTL kodu yazıldıktan sonra sentezden önce beklendiği gibi çalışıp çalışmadığı (fonksiyonel doğrulama) kontrol edilir. Doğrulama işlemleri, tasarımla paralel gitmektedir. Temel olarak her modül/sistem için ayrı bir doğrulama planı ve testbench dosyaları oluşturulmuştur.

5.1.1. Çekirdek Testleri

İlk önce tasarlanan tüm buyruklar kontrol edilir ve sonuçları gözlenir. Oluşabilecek olan diagnostik durumlara göre kod dizilimleri oluşturulur. Oluşturulan algoritma sonucu boru hattı içerisindeki buyrukların etki ettiği donanımların girdi ve çıktıları incelenmiştir. Kendi yazdığımız algoritmaların/buyruk dizilimlerinin sonuçları doğrulandıktan sonra RISC-V topluluğunun RV64IMC için sunmuş olduğu test algoritmaları [4] ile işlemci doğrulanır. Test algoritmalarına göre buyruk başına çevrim hesaplanır. Hedeflenen performans sağlanmıyorsa RTL tasarım aşamasına tekrar dönülür. Çekirdek içerisinde bulunan AMB, Yazmaç Öbeği, Anlık Genişletme Birimi, Kod Çözücüler ve Sıkıştırılmış Buyruk Çözücüler RTL tasarımı esnasında girdileri ve çıktıları kontrol edilerek test edilmiştir.

5.1.2. Bellek Organizasyonu Testi

“rw” sinyali 1 iken adres ile gelen veri index değerine göre eşleştiği ve kirli bit değeri 0 olan L1 önbellek bloğuna yazılır. Kirli bit değeri 1 yapılır. Aynı indexe sahip bir başka veri önbelleğe yazılmak istenirse, öncelikle önbelleğe daha önce yazılmış olan veri “L1_previous_data”ya yüklenir. Ve bellek hiyerarşisinin üst kısımlarına çıkarılır. Eğer L2 önbelleği uygunsa “set_data_to_L2cache_flag” aktif edilir ve bir sonraki saat çevriminde veri L2 önbelleğine yüklenir. Aynı saat çevriminde L1 veri önbelleğinin kirli bit değeri 0 yapılır. Bir sonraki saat çevriminde veri L1’e yüklenir. Eğer L2 önbelleği yükleme yapmak için müsait değilse, L2 önbelleğindeki eski veri “L2_previous_data”ya yüklenir ve ana belleğe çıkartılır. L2 kirli bit değeri 0 yapılır. “ L1_previous_data”, L2’ye yüklenir aynı saat çevriminde L1 kirli biti 0’a çekilir. Bir sonraki saat çevriminde güncel veri L1’e yazılır. Böylece, en kötü senaryo olan verinin L2 önbellekten ana belleğe yüklenmesi gerektiği durumda, bu aşama n saat çevriminde tamamlandığı varsayıldığında güncel verinin L1’e yüklenmesi için gereken saat çevrimi 4+n olarak hesaplanmış olur.

“rw” sinyali 0 iken gelen adresin index ve tag bilgilerine göre veri öncelikle önbelleklerde aranır, bulunursa “data_out” sinyaline değer yazılır; bulunamazsa adres değerinin offset bitleri hariç tüm

bitleri ile ana bellekteki denk gelen adresteki veri okunur ve “data_out”a verilir. Bu veri akışı hazırlanan testbench ile simule edilmiştir.

5.1.3. Karıştırma Testleri

Verilen karıştırma algoritmasına göre bizden; verilerin ve buyrukların her 1 baytının, üst dört ve alt dört bitlerinin kombinasyonlarına göre şifrelenmesi istenmiştir. 8’h00 ’dan başlamak üzere 8’hff’ye kadar olan değerler verilen S-box’a göre karıştırıldı. Örnek olarak 8’h00 baytını ele alalım. Bu veri karıştırıldığı zaman 8’h63 baytına çevrilmektedir. Ters s-box uygularken ise 8h’63 değeri 8’h00 haline çevrilmektedir. Hazırlanan lookup tabloları girilen verilere göre simule edilerek sonuçları gözlemlenmiştir.

5.1.4. Çevre Birimleri Testleri

Tasarım esnasında yapılan testlerde gönderilen ve alınan verilerin FIFO’lardaki durumu, verilerin SPI biriminde işlenmesi ve master-slave ilişkisi incelenir. Aynı zamanda Winbond W25Q256, Micron MT25QL128 Serial NOR FLASH'ların Verilog modelleri test altına SPI modülüne bağlanarak ve yarışma komitesinin sunmuş olduğu yazılım ile test edilir.

UART birimi test için hazırlanan alıcı/verici rolündeki farklı bir Verilog modülü ile yarışma komitesi tarafından sağlanan test algoritması ile test edilir.

5.2. Sentez Sonrası Test

İşlemcinin test edileceği FPGA’e göre Vivado ortamında statik zaman analizleri ve kaynak tüketimi analizleri gerçekleştirilir. Çıkan sonuçlar hedeflenen başarım/performans ölçütlerini olumsuz etkilediğinde RTL tasarım aşamasına geri dönülür.

5.3. Serim Sonrası Test

DRC ve LVS çıktıları, magic ve netgen kullanılarak uygun sınırlar içerisinde olup olmadığı kontrol edilir.

Tarama zincirleri olarak bilinen teknik ile de 1 veya 0 arasında kalmış, durumu bilinmeyen devreleri kontrol etmek ve optimize etmek için kullanılır. Giriş için test vektörleri oluşturularak, çıkıştaki değişimlerin uygun olup olmadığı izlenir.

Aynı zamanda rastgele durumlar girilerek, farklı sinyallerin tetiklenip tetiklenmediği, çipin düzgün bir şekilde çalışıp çalışmadığı kontrol edilir.

6. TAKIM ORGANİZASYONU (5 PUAN)

6.1. Takım Organizasyonu

Danışman Bilgisi: Doç. Dr. Enver Çavuş, Ankara Yıldırım Beyazıt Üni., Elektrik-Elektronik Müh.

Takım Bilgisi: Ankara Yıldırım Beyazıt Üni. Elektrik-Elektonik Müh. 4. sınıf öğrencileri Abdulkadir Arslan, Birkan Çetinkaya, Kerim Turak, Ömer Karslıoğlu'ndan oluşmaktadır.

6.2. Görev Dağılımı

Şekil 9: Takım Organizasyonu

7. İŞ PLANI ve RİSK PLANLAMASI (5 PUAN)

Şekil 10: İş Planı

Şekil 10'da iş planımız paylaşılmıştır. ÖTR aşamasındaki son teslim tarihinin ertelenmesine rağmen tanımladığımız görevleri yerine getirerek raporumuzu tam tarihinde teslim etmiştik. DTR aşamasının

üniversite final haftaları ile çakışmasından dolayı yapılan erteleme sonrasında iş planında tarih aralıklarında değişikliklere gittik. Kritik gördüğümüz 8 göreve tabloda yer verdik. DTR aşaması için tanımladığımız kritik görevlerden tamamladıklarımız tik işareti konularak gösterilmiştir. DTR aşaması için planladığımız; henüz tam olarak bitmemiş görevlerimiz ise şunlardır: “Belleklerin, bellek hiyerarşisinin ve bellek arayüzünün implementasyonu” ve “Çevre birim yöneticisi & UART-SPI çevre birimlerinin implementasyonu”. Bellek organizasyonunun RTL kodlarını yazmış bulunmaktayız ancak sentez aşamasında yaşadığımız sorundan dolayı bellek organizasyonunun implementasyonunu DTR tesliminden sonra 5 gün içerisinde tamamlamayı hedeflemekteyiz. İsterlere uygun olarak yazmakta bulunduğumuz SPI ve UART çevre birimlerinin tamamlanmasını ve entegrasyonunu DTR aşamasından 1 hafta sonra tamamlamayı öngörüyoruz.

Eksiklerimizi tamamladıktan sonra 17.07.2022 tarihinde verilog kodlarımızı FPGA üzerinde koşturarak final sunum çalışmalarımıza başlangıç yapacağız. Bu tarihten sonra 12 gün içerisinde sistem tasarımımızda tüm eksiklerimizi giderip geri kalan 12 gün boyunca ise sadece gds çıktısı almak için çalışarak çalışmalarımızı nihayetlendirmeyi amaçlıyoruz.

TEKNOFEST
HAVACILIK, UZAY VE TEKNOLOJİ FESTİVALİ

8. KAYNAKÇA

[1] Savita Nair, Ajit Saraf, “A Review Paper on Comparison of Multipliers based on Performance Parameters”, In: 2014, International Journal of Computer Application.

[2] Sumit Vaidya, Deepak Dandekar, “Delay-Power Performance Comparison of Multipliers in VLSI Circuit Design” In: July, 2010.

[3] Timsifive, “riscv-tests”, https://github.com/riscv-software-src/riscv-tests

[4] Himanshu Bansal, K. G. Sharma*, Tripti Sharma, “Wallace Tree Multiplier Designs: A Performance Comparison Review”, In 2014, International Institute for Science, Technology and Education: E-Journals.

[5] rcetin, “booth_wallace_multiplier”, https://github.com/rcetin/booth_wallace_multiplier

[6] Digital Design and Computer Architecture RISC-V edition, Sarah L Harris, David Money Harris, Chapter 8.3, p. 517. In: 2022.

[7] Mansi Jhamb, R.K. Sharma, A.K. Gupta, “A high level implementation and performance evaluation of level-I asynchronous cache on FPGA” In: 2017, Journal of King Saud University.

[8] Norman P. Jouppi, “Cache Write Policies and Performance”, In: December, 1991, WRL Research Report.

[9] Computer Organization and Design RISC-V Edition The Hardware Software Interface, Second Edition, David A. Patterson, John L. Hennessy. In: 31 Dec. 2020.

[10] The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 2.2", Editors Andrew Waterman and Krste Asanovic, RISC-V Foundation, May 2017.

[11] Yavuz Selim Tozlu, Yasin Yılmaz "Design and Implementation of a 32 bit RISC-V Core". In: June 2021.

[12] n-singha, adv_PhysicalDesign_Wrkshp, https://github.com/n-singha/adv_PhysicalDesign_Wrkshp

[13] OpenROAD Team, 09.07.2022, https://openroad.readthedocs.io/_/downloads/en/latest/pdf/

[14] Steven Ho, Nick Riasanovsky, "Great Ideas in Computer Architecture (Machine Structures) UC Berkeley EECS". In: 2018, last view: 01.04.2022, https://inst.eecs.berkeley.edu/~cs61c

[15] OpenRoad, https://theopenroadproject.org/resources/