TEKNOFEST
HAVACILIK, UZAY VE TEKNOLOJİ FESTİVALİ
ÇİP TASARIM YARIŞMASI
2023
SAYISAL İŞLEMCİ TASARIM KATEGORİSİ
DETAY TASARIM RAPORU

TAKIM ADI

KASIRGA - KIZIL

PROJE ADI

KIZIL İŞLEMCİ PROJESİ

BAŞVURU ID

901285

16.03.2023

İÇİNDEKİLER

SEMBOLLER LİSTESİ

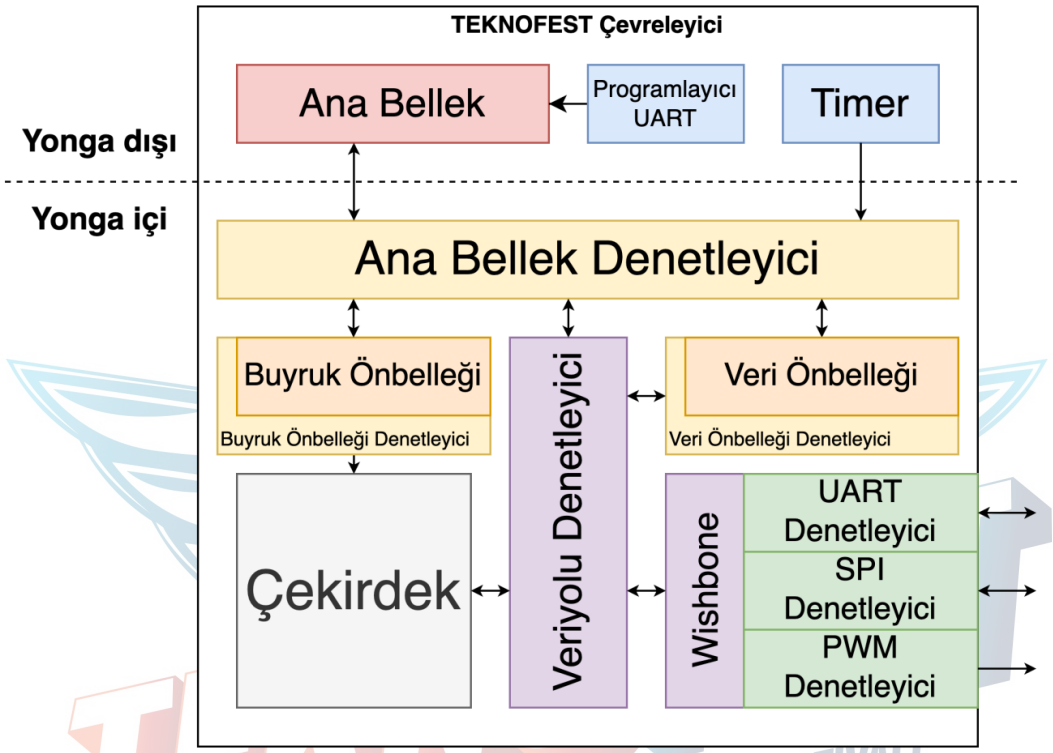
$	Önbellek
V$	Veri Önbelleği
B$	Buyruk Önbelleği

KISALTMALAR LİSTESİ

ASIC	Application Specific Integrated Circuit
FPGA	Field Programmable Gate Array
SRAM	Static Random Access Memory
UART	Universal Asynchronous Receiver-Transmitter
SPI	Serial Peripheral Interface
PWM	Pulse Width Modulation
PDK	Process Design Kit

1. TEMEL TASARIM ÖZETİ

Bu proje kapsamında, şartnamede verilen isterler doğrultusunda RV32IMCX 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/KASIRGA-KIZIL/tekno-kizil adresinde bulunabilir. Şekil 1.1'de ise sistemin genel yapısı gösterilmiştir.



Şekil 1.1. Sistemin Genel Blok Şeması

Gerçekleştirilen Çekirdek, (1) Getir (fetch), (2) Çöz (decode), (3) Yürüt (execute), (4) Geri Yaz (write-back) olmak üzere 4 boru hattı aşamasından oluşmaktadır. Proje isterleri kapsamında bir seviyeli bir önbellek yapısı oluşturulmuştur. Bellek sisteminde veri ve buyruk için eşit boyutlu önbellekler kullanılmıştır.

İşlemcinin çevre birimleri ile haberleşmesi ana bellek denetleyicisine ek olarak wishbone arayüzü ile gerçekleştirilmiştir. İşlemci, proje isterleri kapsamında UART, PWM ve SPI protokollerini desteklemektedir.

Çipin serimini gerçekleştirilmesi için OpenLane [1] akışı ile sağlanmıştır. Bu akışta her bir bileşen (çekirdek, çevre birimleri, önbellek denetleyicileri, vb.) ayrı olarak makrolaştırılmıştır. Çip seriminde önbellekler hibrit olarak hem yazmaç tabanlı hem de SRAM tabanlı bellek olarak gerçeklenmiştir. SRAM tabanlı bölümleri için çip akışında makrolaştırmak için OpenRAM [2] aracı kullanılmıştır. Elde edilen makroların, genel bir birim altında bağlanması ile işlemcinin serimi elde edilmiştir.

Sistemin davranışsal işlevselliği, hem. 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 [3] kullanılarak gerçekleştirilmiştir. Çarpma için kullanılan algoritma Radix-4 Booth Dadda algoritması olup. Convolution buyruğu icin gereken biriktirme (accumulate) işlemi Kogge-Stone algoritması ile sağlanmıştır. Bölme işlemi için bit tabanlı yeniden kaydetmeli bölme algoritması kullanılmıştır ve 18 çevrimde gerçekleşmektedir. Performansın arttırılması için yürüt aşamasında dallanma öngörücü (branch predictor), adres dönüş yığıtı (return address stack) ve dallanma hedef yazmacı (branch target buffer) kullanılmıştır.

Tablo 1.1. KIZIL İşlemci Özellikleri

Özellik	KIZIL İşlemci
Buyruk Kümesi	RISC-V (RV32IMCX)
Sentezlenebilir	Verilog RTL √
FPGA Boardları	AMD Virtex Ultrascale Vcu108 (125 MHz) √ Xilinx Basys3 (50 MHz)√
Wishbone	√
Veri Yönlendirmesi	√
Boru Hattı Aşama Sayısı	4
Veri Önbelleği	Doğrudan Eşlemeli 2KB SRAM
Buyruk Önbelleği	Doğrudan Eşlemeli 2KB DFFRAM
Çevre Birimleri	UART, SPI, PWM
CoreMark Skoru	214 iterasyon/saniye (@125MHz, 3000 iterasyon)
CoreMark/MHz	1,71
Dhrystone Skoru	349232 Dhrystones per Second, 2 microseconds for one run Dhrystone (@125MHz, 3milyon iterasyon)

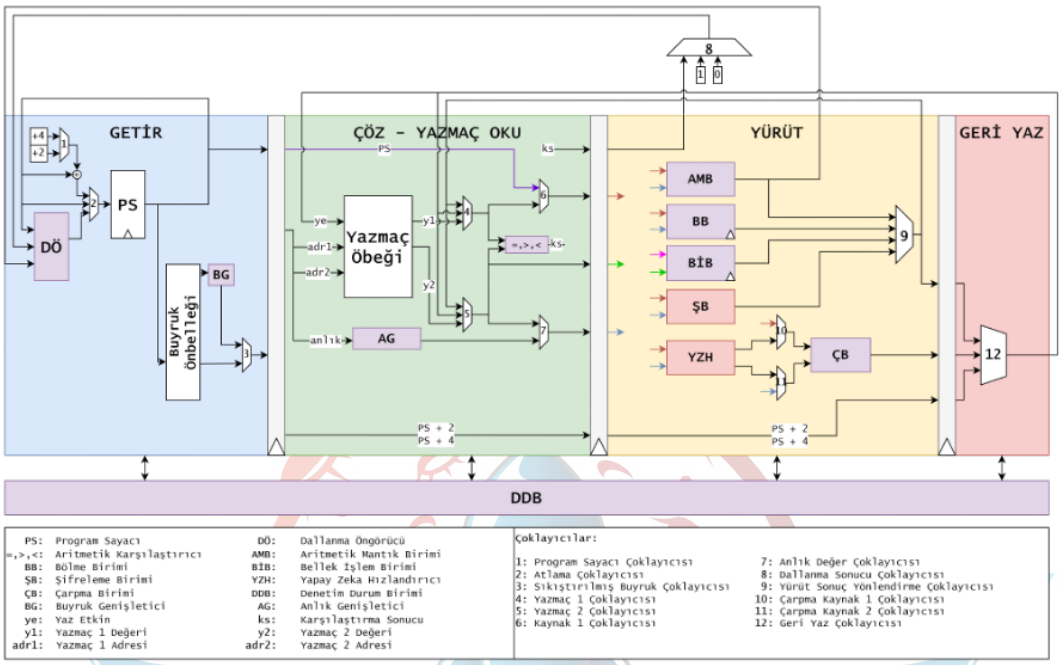
2. PROJE MEVCUT DURUM DEĞERLENDİRMESİ

Proje kapsamında geliştirilen işlemci çekirdeği ön tasarım raporunda da belirtilen bütün isterleri karşılamaktadır. Boru hattı aşama sayısı ve diğer büyük tasarım kararları (önbelleklerin 2KB-2KB, toplam 4KB buyruk ve veri olarak bölünmesi vb.) ÖTR'den bu yana değişmemiştir. RV32IMCX buyruk kümesi mimarisini (instruction set architecture) destekleyen işlemci gerekli bütün testleri başarıyla geçmiştir. Projenin ilerlemesi sırasında ön tasarım raporunda belirtilenden farklı şekilde uygulanan bazı optimizasyona yönelik tasarım kararları alınmıştır. Buyruk önbelleği performans göz önüne alınarak tamamen d flip-flop ve latch tabanlı [4] gerçeklenmiştir. Veri önbellegi ise yine performans artışı için hibrit olarak hem kapı hem de SRAM tabanlı gerçeklenmiştir.

Proje genel olarak her aşamasıyla bitmiş nihai tasarıma kadar optimizasyonlar yapılacak ve Openlane iyileştirmeleri yapılacaktır. Proje son durumunda, işlemci Openlane akışından 50 MHz'de geçirilmiştir fakat routing kolay olsun ve geçsin diye alan kocamandır ve fanout violation gibi akışı etkilemeyen değerler çözülmemiştir.

3. PROJE DETAY TASARIMI

Şekil 3.1'de proje kapsamında gerçeklenen işlemcinin detaylı boru hattı şeması görülebilir.



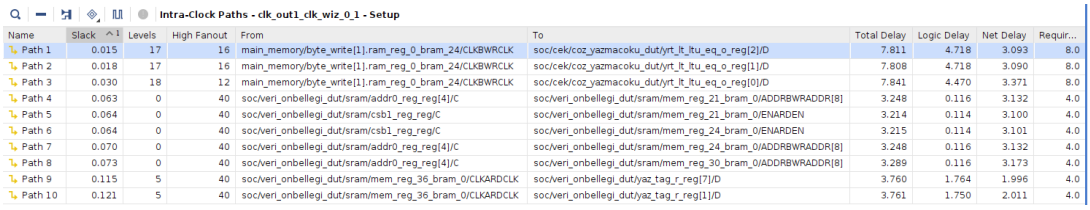
Şekil 3.1. Tasarlanmış Çekirdek Detaylı Boru Hattı Şeması

Kızıl İşlemci, UltraScale VCU108 ve Basys3 FPGA'lerinde çalıştırılmıştır ve çevreleyicide bulunan programlayıcı uart ile programlanarak derlenen hex kodları FPGA'lerde koşturulmuştur. Şartnamede belirtilen Nexys A7-100T FPGA elde bulunmadığından sadece Vivado'da implementasyonu denenmiştir. Tablo 3.1'de FPGA maksimum frekans, güç ve utilization değerleri verilmiştir. (Değerler Vivado 2022'de Flow_RunPostRoutePhysOpt implementasyonuyla alınmıştır.) (Basys3'e sığmadığı için 20480 satır ram, diğerlerinde 131072 satır ram kullanılmıştır.)

Tablo 3.1. FPGA Board'ları Performans ve Utilization Değerleri

FPGA Board	Maks. Frekans (MHz)	LUT	FF	BRAM	Toplam On-chip Güç (mW)
VCU108 (xcvu095-ffva2104-2-e)	125	9746	5931	148	1258
BASYS3 (xc7a35tcpg236-1)	50	10467	6005	32	307
NEXYS A7-100T (xc7a100tcsg324-1)	65	11219	6296	128	429

Şekil 3.2'de ise VCU108 FPGA'inde Kızıl İşlemci Kritik Path'ler gösterilmiştir. Görüleceği üzere main memory kritik path'i kısıtlamaktadır.



Intra-Clock Paths - clk_out1_clk_wiz_0_1 - Setup

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requir...
Path 1	0.015	17	16	main_memory/byte_write[1].ram_reg_0_bram_24/CLKBWRCLK	soc/cek/coz_yazmacoku_dut/yrt_lt_ltu_eq_o_reg[2]/D	7.811	4.718	3.093	8.0
Path 2	0.018	17	16	main_memory/byte_write[1].ram_reg_0_bram_24/CLKBWRCLK	soc/cek/coz_yazmacoku_dut/yrt_lt_ltu_eq_o_reg[1]/D	7.808	4.718	3.090	8.0
Path 3	0.030	18	12	main_memory/byte_write[1].ram_reg_0_bram_24/CLKBWRCLK	soc/cek/coz_yazmacoku_dut/yrt_lt_ltu_eq_o_reg[0]/D	7.841	4.470	3.371	8.0
Path 4	0.063	0	40	soc/veri_onbellegi_dut/sram/addr0_reg_reg[4]/C	soc/veri_onbellegi_dut/sram/mem_reg_21_bram_0/ADDRBWRADDR[8]	3.248	0.116	3.132	4.0
Path 5	0.064	0	40	soc/veri_onbellegi_dut/sram/csb1_reg_reg/C	soc/veri_onbellegi_dut/sram/mem_reg_21_bram_0/ENARDEN	3.214	0.114	3.100	4.0
Path 6	0.064	0	40	soc/veri_onbellegi_dut/sram/csb1_reg_reg/C	soc/veri_onbellegi_dut/sram/mem_reg_24_bram_0/ENARDEN	3.215	0.114	3.101	4.0
Path 7	0.070	0	40	soc/veri_onbellegi_dut/sram/addr0_reg_reg[4]/C	soc/veri_onbellegi_dut/sram/mem_reg_24_bram_0/ADDRBWRADDR[8]	3.248	0.116	3.132	4.0
Path 8	0.073	0	40	soc/veri_onbellegi_dut/sram/addr0_reg_reg[4]/C	soc/veri_onbellegi_dut/sram/mem_reg_30_bram_0/ADDRBWRADDR[8]	3.289	0.116	3.173	4.0
Path 9	0.115	5	40	soc/veri_onbellegi_dut/sram/mem_reg_36_bram_0/CLKARDCLK	soc/veri_onbellegi_dut/yaz_tag_r_reg[7]/D	3.760	1.764	1.996	4.0
Path 10	0.121	5	40	soc/veri_onbellegi_dut/sram/mem_reg_36_bram_0/CLKARDCLK	soc/veri_onbellegi_dut/yaz_tag_r_reg[1]/D	3.761	1.750	2.011	4.0

Şekil 3.2. VCU108 FPGA'inde Kızıl İşlemci Kritik Path'ler

3.1. Çekirdek Tasarımı

3.1.1. Boru Hattı

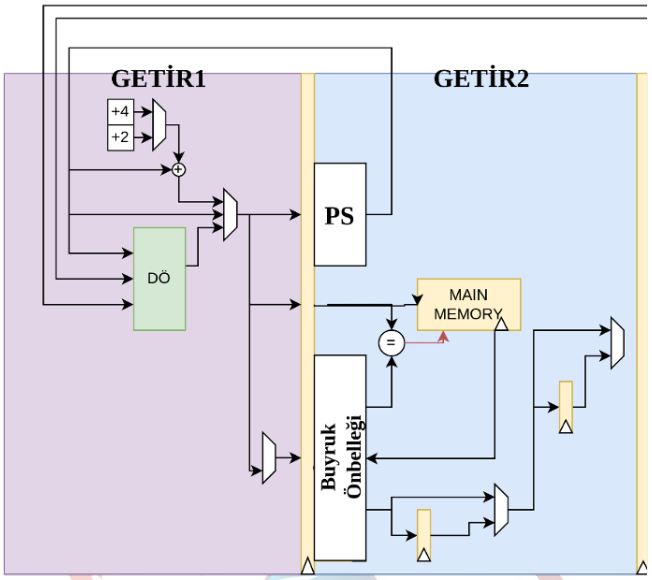
Proje kapsamında geliştirilen sistemin detaylı boru hattı şeması Şekil 3.1'de gösterilmektedir. Boru hattı ve sistem genelinde herhangi bir açık kaynak tasarım kullanılmamış ve çekirdek tamamen özgün bir şekilde gerçeklenmiştir. Sistemde 4 aşamalı sıralı yürütme yapan işlemci çekirdeği, işlemci çekirdeğinin Getir aşamasıyla haberleşen 2 KB boyutunda 1. seviye buyruk önbelleği (L1B$), yürüt aşamasındaki bellek işlem birimiyle haberleşen 2 KB boyutunda 1. seviye veri önbelleği (L1V$) bulunmaktadir, çekirdek dışı birimler (çevre birimleri ve bellek hiyerarşisi) ile çekirdeğin haberleşmesini yönlendirmek için wishbone master eklenmiş ve çip dışıyla haberleşmeyi gerçekleştirmek için UART, PWM ve SPI çevre birimleri yarışma şartnamesinde istenildiği şekilde özelleştirilerek sisteme entegre edilmiştir. Sistem mimarisinin yazmaç seviyesindeki kodlaması Verilog-2005 donanım tasarlama dili kullanarak gerçeklenmiştir. İşlemci çekirdeği, RV32IMCX özelleştirilmiş buyruk kümesi mimarisini desteklemektedir.

Çekirdekte toplama işlemi, toplama-sembolü yerine carry lookahead toplayıcısı ile gerçekleştirilmiştir. Çarpma işleminde çarpma sembolü yerine, değiştirilmiş Booth Dadda algoritması [5] tercih edilmiştir. Değiştirilmiş Booth algoritması ve Dadda Tree yöntemlerinin bir araya getirilmesiyle oluşturulmuş bu algoritma görece küçük bir alanda yüksek başarımlı çarpma işlemlerinin yapılmasını sağlamaktadır.

Bölme modülünde yeniden kaydetmeli bölme algoritması [6] kullanılmıştır. Karmaşık bölme algoritmalarına kıyasla çok küçük bir alanda ve çok basit işlemlerle bölme işlemini gerçekleştiren bu algoritma normalde 33 çevirimde sonuç hesaplamaktadır ama yapılan testler sonucunda kritik yolun bu modül üzerinde olmadığı görülmüş ve ortalama bir programda çok fazla sayıda olan bölme işlemlerinde çevirimden tasarruf sağlamak için bir çevirimde iki adım ilerleyecek şekilde güncellenmiştir. Bu sayede modül işlemcinin saat vuruş sıklığını arttırmadan 18 çevirimde sonuç verecek hale getirilmiş ve başarımda artış sağlanmıştır.

•Getir

Getir Modülü buyruk önbelleği ile birlikte, program akışına bağlı olarak değişen program sayacının üretilmesinden, buyrukların ana bellekten işlemciye getirilmesinden ve dallanma ve atlama buyruklarının tahminini yaparak performans artışı sağlayan dallanma öngörücüden sorumludur [7]. Yarışma kapsamında istenilen "Compressed Buyruk Eklentisi" [8] sıkça kullanılan bazı 32 bitlik buyrukların derleyici tarafından 16 bit buyruklara çevrilmesi ile kod boyutunun kısalmasını sağlamayı hedefler [9]. Bu eklentinin desteklenmesi sonucunda bütün buyruklar, bellekte 32-bit hizalı olmak yerine 16-bit hizalı tutulur. Bu durumda performans kaybı yaşanmaması için hizasız erişilen 32 bit buyrukların tek çevrimde getirilebilmesi gerekir. Ayrıca 16 bit buyrukların ana bellekten 32 bit olarak getir modülüne getirilmesi alanda yerelliğe (Spatial Locality) örnek olarak verilebilir ve yerellikten faydalanılması performansı arttırır. Bu durumların desteklenmesi getir modülünün kontrol mekanizmasını karmaşık hale getirmektedir. Açıklanan bu durumların doğru bir şekilde kontrol edilmesi için iki ayrı getir modülü tasarlanmış olup, yapılan testlerde (CoreMark vb.) daha iyi başarıma sahip olduğu tespit edilen tek getir aşamalı tasarım projenin ileriki aşamalarında kullanılmıştır. İki ayrı getir aşamasına sahip olan tasarımın neden tercih edilmediği aşağıda açıklanmaktadır. İki aşamalı getir tasarımının yüzeysel bir şematiği Şekil 3.3 ile görülebilir.



Şekil 3.3. 2 Aşamalı Getir Şeması

Bu tasarımda;

i-) OpenRAM ile sentezlenmiş buyruk önbelleği kullanılmıştır. OpenRAM karakteristiği nedeni ile okuma istekleri 1 çevrim gecikmektedir ve bu durumda çevrim kaybedilmemesi (bubble) için getir 2 aşamaya bölünmüştür.

ii-) Dallanma buyruğu geldiği zaman arada 1 çevrim fazladan kaybedilmektedir. Ayrıca, önbellekte bulunamama (cache miss) durumunda da aynı şekilde 1 çevrim kaybedilmektedir.

iii-) Kritik yolun getir aşamasında bulunmaması nedeniyle saat vuruş sıklığını arttırmamıştır.

Bu nedenlerden dolayı iki aşamalı getir tasarımından vazgeçilmiştir. Oluşturulan tek aşamalı getir modülünde buyruk önbelleğinin aynı çevrimde okunabilmesi için buyruk önbelleği, d flip-flop ve latch hücreleri ile [4] sentezlenmiştir. Bu tasarım kararı sonucunda buyruk önbelleği alanının SRAM'lere kıyasla yaklaşık 2 kat arttığı görülürken çevrim sayısı azaldığı için yapılan testlerde (CoreMark vb.) başarımı yaklaşık 2 katına çıkmıştır.

Ayrıca compressed buyrukların desteklenmesinden dolayı ortaya çıkan sorunlar, Buyruk Önbelleğine hizasız erişim yapılabilmesi ile çözülmüştür. Buyruk Önbelleğinin tasarımı bellek tasarımları bölümünde detaylı açıklanmıştır. Getir modülünün bir diğer görevi ise 16 bit compressed buyrukları 32 bit eşleniklerine çevirmektir.

Program sayacının bir sonraki çevrim için değeri, şu anki çevrimde işlenen buyruğa ve Yürüt aşamasından gelen "dallanma kontrol sinyallerine" bağlıdır. Şu anki çevrimde getir ve Yürüt modüllerinde işlenen buyrukların kontrol akışını değiştirmeyen buyruklar olması durumunda RISC-V mimarisinde byte adresleme kullanıldığından program sayacı 2 (compressed buyruklar için) veya 4 arttırılır. (Verilog tasarımında bit sayısından tasarruf ederek optimizasyon sağlamak için 1 veya 2 olarak arttırılmıştır, yorumlarken 2 veya 4 olarak yorumlanmaktadır.) Yürütteki buyruğun dallanmanın yanlış atladığını bildirmesi durumunda Yürütte hesaplanan program sayacı değeri, getir aşamasında program sayacının hesaplanmasında kullanılır. Dördüncü durum ise getir aşamasındaki buyruğun kontrol akış buyruğu olduğu durumdur. Bu dallanma ve atlama buyruklarının sonuçları Yürüt aşamasına kadar bilinmemek-

tedir. Dallanma öngörü birimi, dallanmaların yönünü tahmin etmek için kullanılır. Proje kapsamında tasarlanan dallanma öngörücü, aşağıda açıklanmaktadır.

Dallanma Öngörücü: Bellek gecikmelerinin ve boru hattı sayısının artması dallanma buyruklarının penaltısını arttırdığı için bu buyrukların sonuçlarının getir aşamasında tahmin edilebilmesi, işlemci başarımı için hayati öneme sahiptir [10]. Dallanma öngörücü algoritmalarının alan ve başarım ödünleşmeleri [11] göz önüne alındığında, işlemcide GShare öngörücü [12]kullanılmasına karar verilmiştir. GShare algoritması, program sayacının belirli bitlerini ve dallanma geçmişini tutan genel geçmiş yazmacını XOR'layarak bir indeks oluşturur. Bu indeks, çift kutuplu sayaç tablosundaki bir sayaca erişilmesi için kullanılır ve indeks farklı dallanma örüntülerinin ve buyruklarının aynı satıra erişmesini (aliasing) engellemeyi hedefler. Oluşturulan GShare modülü 5 bit genişliğinde bir genel geçmiş yazmacı içerir ve bu bitleri program sayacının 5'ten 1'e kadar olan bitleri ile XOR'lar ve 32 bitlik sayaç tablosuna erişir. GShare algoritmasının doğru çalışabilmesi için, bir dallanma buyruğu, kendi öngörüsünü okuduğu sayaç tablo satırını güncellemelidir. Fakat art arda birden fazla dallanma buyruğu gelmesi durumunda, en önden giden dallanma buyruğu, genel geçmiş tablosunu Yürüt aşamasında güncelleyeceği için arkadan giden dallanma buyrukları kendi öngörülerini okudukları satırdan başka satırlara erişir [13]. Bu durum genel geçmiş tablosunun spekülatif güncellenmesi ile çözülmüştür. Yanlış öngörü durumunda ise genel geçmiş tablosu, boru hattındaki dallanma buyruğu sayısı kadar geri sarılır.

Dallanmaların doğru tahmin edilebilmesi için davranışlarının yanı sıra atladıkları adresler de bilinmelidir. Dallanma Hedef Yazmacı (branch target buffer veya BTB) atlanan adreslerin saklanması için kullanılır. Oluşturulan dallanma hedef yazmacı 32 satıra sahiptir GShare indeksi ile erişilir. Buyruk adres genişliği göz önünde bulundurularak (0x40000000, 0x40080000) veri genişliğinin 18 bit olması yeterli olmuştur.

Dallanma öngörücüde bulunan bir diğer birim de adres dönüş yığıtıdır (return address stack veya RAS). Bir prosedür (function call) kodda farklı konumlardan çağırılabilir. Bu durumda BTB'de dönüş buyruğunun (return) indeksi daha öncesinden BTB tablosuna erişmediğinden dolayı doğru adrese ulaşamaz. Programın geri dönmesi gerektiğinde RAS'tan en son giren değer çekilir, bu değer dönüş adresi için bir tahmindir. Oluşturulan RAS iç içe 4 dönüş adresi tutabilir ve bu adreslerin her biri 18 bittir.

•Çöz-Yazmaç Oku

Çöz-Yazmaç Oku aşaması iki göreve sahiptir. Bunlardan birincisi getirilen buyrukları iş kodlarına (İng. opcode), işlev kodlarına (İng. function) göre parçalara ayırarak kaynak yazmaç adreslerinin, anlık değerlerin ayrıştırılmasıdır. Ayrıştırılan anlık değerler Anlık Genişletici biriminden geçerek 32 bitlik değere dönüştürülür ve Yürüt aşamasına aktarılmak üzere Anlık Değer Çoklayıcısına yönlendirilir. İkinci görev ise ilk görevde ayrıştırılan kaynak yazmaç adreslerinin gösterdiği dizinde bulunan kaynak yazmaç değerlerini okuyarak Yürüte aktarmak üzere Yazmaç 1 ve Yazmaç 2 Çoklayıcılarına yönlendirmektir. Buyruklar çözülürken seri çoklayıcılar kullanmaktan kaçınılmış, mümkün olduğunca paralel şekilde çözme işlemi yürütülmeye çalışılmıştır. Çözme işlemi tamamlandığında çözülen buyruklar 28 bitlik mikroişlemlere ayrılmıştır. Mikroişlem sayesinde Yürüt aşamasında buyruk hakkında bilinmesi gereken tüm bilgi aktarılabilmektedir. Çöz işlemi yalnızca buyruğun 14 bitine bakılarak yapılmaktadır ve işlem yapılırken tablo yöntemi kullanılmıştır. Bu sayede çözüm sırasında buyruğun ne yaptığı önceden bilinerek çöz aşamasında tekrar karşılaştırma yapılmamaktadır. Açık kaynaklı RISC-V Boom Core'daki [14] çözme aşamasından esinlenilmiştir. Aşağıda daha iyi anlaşılması için örnek bir mikroişlem kodu verilmiştir.

```
`define ADD_MI  {`YZH_YOK, `SIFRELEME_YOK, `BIB_YOK, `CARPMA_YOK,
                 `BOLME_YOK, `AMB_TOPLAMA, `DAL_YOK, `BIRIM_AMB,
                 `OPERAND_REG,  `YAZMAC_YAZ, `GERIYAZ_KAYNAK_YURUT}
```

•Yürüt:

Çöz-Yazmaç Oku aşamasında çözülen buyrukların buyruğa göre yazmaç değerleri veya anlık değerleri bu aşamaya iletilir. İletilen değerler yine Çöz-Yazmaç Oku'dan iletilen mikro işlem koduna göre alt birimlere yönlendirilir. Bu aşamada veri bağımlılığı bulunması ihtimaline karşın veri yönlendirmesi bulunmaktadır. Bu sayede yazma sonrasında okuma işlemi yapıldığında boru hattının bekletilmesi engellenerek bağımlı veri yürüt aşamasından Çöz-Yazmaç Oku aşamasına yönlendirilir. Ayrıca bu aşamada dallanma buyruklarının doğru atlayıp atlamadığı anlaşılmış olup Dallanma Öngörücü birimine elde edilen veri iletilir.

Aritmetik Mantık Birimi: Aritmetik mantık birimi RV32IMC buyruk kümesindeki mantıksal ve aritmetik buyrukları gerçekleştirir. Toplama ve çıkarma işlemleri için carry look ahead toplayıcısı kullanılmıştır. Dallanma buyrukları için gerekli olan toplama işlemi de yine bu bölümde gerçekleştirilmektedir.

Çarpma Birimi: Çarpma birimi [3] kullanılarak gerçekleştirilmiştir. 66 bitlik bir çarpıcı ve toplayıcı (Multiplier Accümülatör) oluşturulmuş olup işaretsiz çarpma işlemleri 32 bitlik yazmaç değerlerini 33 bite sıfır ile genişletilmesi ile gerçekleştirilir. Bu sayede harici bir çarpma birimi kullanılmasına gerek kalmaz. Çarpma işlemi bir çevrimde gerçekleşir fakat gecikmesi yüksek bir yol olduğu için çöz'e yönlendirilmez.

Yapay Zeka Hızlandırıcısı: Yapay zeka hızlandırıcı veri ve katsayı yazmaçlarından oluşan bir birimdir. Bu yazmaçlar Çöz-Yazmaç Oku bölümünde kullanılan yazmaç önbelleğine oldukça benzerdir. conv_run komutu haricindeki bütün komutlar bu yazmaçlara yazma veya okuma yaparak bu bölümde sonlandırılır. conv_run komutu ise yine yapay zeka hızlandırıcısı tarafından kontrol edilen çoklayıcılar ile çarpma biriminde bağlanır ve var olan çarpma birimini kullanarak konvolüsyon işlemini gerçekleştirir. Konvolüsyon işlemi 16 çevrim sürer.

Şifreleme Birimi: Şifreleme birimi isterlerde belirtilen her şifreleme buyruğuna ayrı bir birim oluşturularak gerçekleştirlmiştir. Sıfır sayıcı buyruğu (cntz) haricindeki diğer bütün buyrukları gerçekleştirmesi oldukça kolaydır. Fakat, sıfır sayıcı için [15]'de belirtilen şema kullanılmıştır. Bunun sebebi Çip Akışı bölümünde detaylı anlatılmakla beraber Yosys'in kullandığı sentezleme yönteminden kaynaklanmaktadır.

Bölme Birimi: Bölme işlemi 18 çevrimde yeniden kaydetmeli bölme algoritması ile gerçekleştirilir. Toplama ve çıkarma işlemleri kullanılır. Bu süre zarfında boru hattı durdurulur.

Bellek İşlem Birimi: Bellek işlem birimi RV32IMC buyruk kümesindeki LOAD ve STORE buyruklarının çekirdek dışına iletilmesinden sorumludur. Bellek İşlem Birimi birden fazla çevrim sürdüğü için bu süre zarfında boru hattı durdurulur.

3.1.2. Denetim Durum Birimi

Denetim durum birimi boru hattının kontrolünden sorumludur. Denetim durum birimi boru hattı aşamalarından gelen hazır sinyallerini ve yazmaç adreslerini göz önüne alarak yönlendirme yapar veya boru hattını durdurur. Genel olarak kombinasyonel devrelerden oluşan denetim durum birimi, çarpma biriminin yönlendirilemez olması nedeniyle ve başlangıçtaki boru hattında bulunan buyrukların geçersiz

olması nedeniyle 2 bitlik durmuş(stopped) ve bos_basla isimli ardışık (sequential) birimler bulundurmaktadır.

•Veri Sorunu Denetimi

Veri sorunu, bir yazmaca yazma isteğinin gelmesinin ardından henüz geri yaz aşamasına ulaşmayan buyruğun yazmaç öbeğini güncellemeden aynı yazmaca bir okuma isteği gelmesiyle oluşur. Böyle bir durumda KIZIL İşlemci'nin iyi yönleri, az aşamalı boru hattından oluşması ve veri yönlendirmesine sahip olmasıdır. Çok aşamalı boru hatlarında veri yönlendirmesi yazmaç öbeğinin okunduğu aşamanın ardından Geri Yaz aşamasına varıncaya kadar bütün aşamalarda yapılabilir. Ancak bizim tasarımımızda yazmaç öbeğinin okunduğu ve yazmaç öbeğine geri yazıldığı aşamalar arasında yalnızca bir aşama bulunduğundan (Yürüt) tek bir aşamadan veri yönlendirmesi yapılması yeterli olmuştur. Bu sayede aynı yazmaca yazma ardından okuma yapılması boru hattını duraklatmayıp sadece veri yönlendirmesini tetiklemektedir.

•Yapı Sorunu Denetimi

Çarpma, bölme, bellek ve yapay zeka işlemleri birimleri istenen işlemi birden fazla çevrimde tamamladığı bilinmektedir. Bu durum bahsi geçen işlem birimlerinin buyruğu geldiğinde tüm boru hattını bu işlemi beklemek zorunda bırakır. Bu buyruklar Yürüt aşamasına geldiğinde Denetim Durum Birimine bildirilir ve tüm boru hattı buradan yayılan sinyalle duraklatılır.

3.2. Bellek Tasarımları

Proje kapsamında gerçekleştirilen bellek hiyerarşisi, Getir Aşamasındaki 1. seviye buyruk önbelleği denetleyicisine bağlı 2 KB boyutunda Buyruk Belleği, Yürüt aşamasında bulunan 1. seviye veri önbelleği denetleyicisine bağlı 2 KB boyutunda Veri Önbelleği ve bu iki önbellek denetleyicilerini ana belleğe bağlayan bir Ana bellek denetleyiciden oluşmaktadır. Bellek hiyerarşisi şekil 1.1 ile görülebilir. Bellek sisteminde Buyruk Önbelleği ve Veri Önbelleğinin boyutlarının aynı olmasına rağmen bellek denetleyicileri ve tasarım karakteristikleri tamamen farklıdır. Buyruk önbelleği yalnızca program sayacının ana bellekten getirdiği buyrukları içerirken Veri Önbelleği "Load" ve "Store" buyrukları tarafından getirilen verileri içerir. Bir programın eriştiği buyruklar, erişilen verilere kıyasla alanda ve zamanda daha fazla yerelliğe sahiptir [16]. Veri ve buyruk önbellek denetleyicileri "Çevreleyici Modül Ana Belleğine" Ana Bellek denetleyici ile bağlıdır. Ana Bellek denetleyici, Önbellek denetleyicilerden biri Ana Bellek isteği oluşturduğunda bu isteği Teknofest Ana Belleğine aktarır. Aynı anda iki Önbellek Denetleyicinin de istek oluşturduğu durumda Buyruk Önbelleği isteği önceliklendirilir. Teknofest ekibi tarafından oluşturulan Ana bellek arayüzü 32 bit veriyolu genişliğine sahiptir, bu durum oluşturulan Önbelleklerin "word" erişimleri için alanda yerellikten faydalanamayacağı anlamına gelmektedir. Buyruk Önbelleği, bir döngünün veya fonksiyon çağrı işlenilmesi durumunda zamanda yerellikten faydalanırken Veri Önbelleği ise aynı verinin tekrar kullanıldığı durumlarda zamanda yerellikten faydalanır.

Tasarım Şartnamesinde belirtilen Veri adres aralığı göz önünde alındığında (0x40000000 , 0x40080000) her adrese 2^8 farklı ana bellek elemanı erişebileceği için ve etiketlerin karşılaştırılması için 1 adet 8 bitlik karşılaştırıcı yeterlidir. Buyruk ve veri önbellekleri 2KB, 2KB toplam 4KB olacak şekilde bölünmüştür.

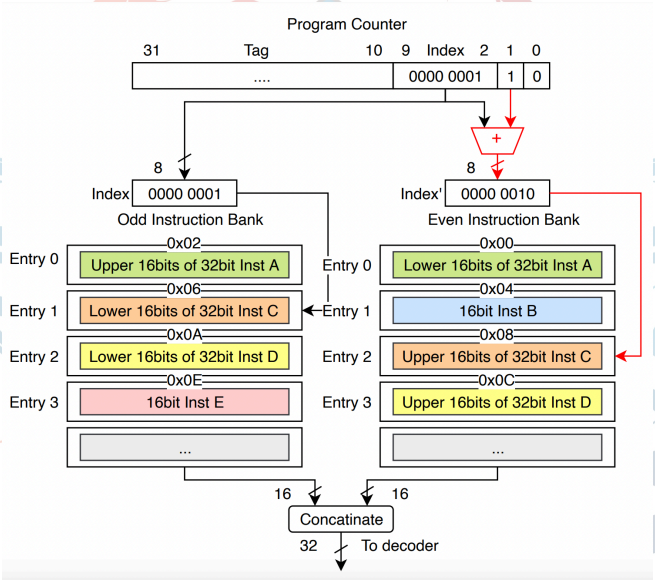
Bu bağlamda genel tasarım Harvard mimarisi olarak seçilmiştir. Bunun nedeni ise örneğin 1KB ve 3KB gibi önbellek boyutları atandığında adreslemenin verimsiz olmasıdır.

Buyruk ve Veri Önbelleklerinin tasarımları gerçekleştirilirken farklı karakteristikleri göz önünde bulun-

durulmuştur, aşağıda sırasıyla Buyruk ve Veri Önbelleklerinin tasarımları açıklanmaktadır.

3.2.1. Buyruk Önbelleği

Buyruk önbelleği tasarımı doğrudan eşlemeli yapıya sahiptir. Sıkıştırılmış buyruk eklentisinin gerektirdiği erişim sorunlarının çözülmesi için özel bir tasarıma sahiptir. Sıkıştırılmış buyrukları desteklerken dikkat edilmesi gereken en önemli nokta 32 ve 16 bitlik buyrukların bellekte boşluk bırakmadan bir arada bulunduğu gerçeğidir. Örneğin, Şekil 3.4 de bulunan C buyruğu 0x06 ve 0x08 adresleri arasında hizasız bir şekilde bulunmaktadır. Bu buyruğun boru hattına getirilebilmesi için 32 bit hizalı bir bellek sisteminde iki defa erişimde bulunulması gerekir. Genelde, çoğu getir mekanizmaları, bu hizasız erişim durumunu geçmiş veriyi bir tamponda tutarak çözmeye çalışırlar. Fakat, bu tamponun 32 bitlik ve bellekte hizasız tutulan bir buyruğa dallanma durumunda temizlenmesi gerekir. Bu yüzden, buyruğun alt ve üst 16 bitlik kısımları 2 farklı çevrimde getirilmek zorundadır. Bu sorunun çözülmesi için Şekil 3.4'de gösterilen önbellek tasarımı [9] kullanmaktadır.



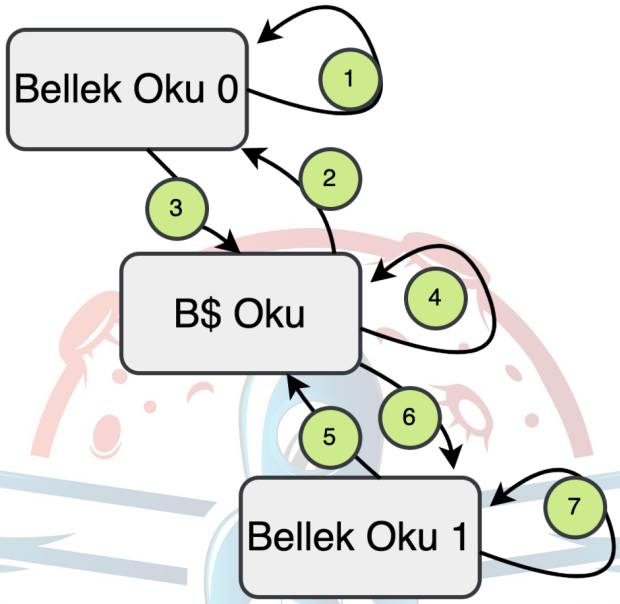
Şekil 3.4. Buyruk Önbelleği Esinlenilen Erişim Yapısı

Buyruk önbelleğile aynı çevrimde erişilebilmesi için buyruk önbelleğinin gerçekleştirilmesinde OpenRAM [2] yerine DFFRAM [4] kullanılmasına karar verilmiştir. Buyruk önbelleğinin boyutu yaklaşık 2 katına çıkarken performansta ortalam 2 kat artış görülmüştür. Bu konuda yapılan denemeler Getir başlığı altında detaylı anlatılmaktadır.

Hizasız erişimler Konomari'nin [9] önbellek yapısına uygun olarak iki tane 512x16 bitlik DFFRAM modülünün bir araya getirilmesiyle gerçekleştirilmiştir. Hizasiz adreslere yapilan erişimler etiket ve geçerli bitlerinin erişilmesi icin iki okuma girişli bir RAM gerektirmektedir. Fakat, bu RAM gereken konfigürasyonlarda DFFRAM üzerinde sentezlenememektedir. Bu nedenle, hizasız erişimlerin ardışık bellek adreslerine yapıldığı gerçeğinden faydalanılarak iki okuma girişli bir bellek modülü oluşturulmuştur. Bu modül iki tane 256x8 bitlik DFFRAM modülünün bir araya gelmesinden oluşur. Tek adreslere gelen istekler bir modüle çift adreslere gelen istekler ise öteki modüle yönlendirilmektedir. Bu sayede aynı çevrim içerisinde ardışık adreslerin etiket ve geçerli bitlerine erişilebilmiştir.

Tablo 3.2. Buyruk Önbelleği Gecikme ve Durum Geçişleri Tablosu

Buyruk Belleği İsteği	Çevrim Sayısı	Durum Makinesi İzlenilen Yol
Okuma (En İyi Durumda)	0	4
Okuma (En Kötü Durumda)	36	2 –>1 –>3 –>6 –>7 –>5



Şekil 3.5. Buyruk Önbellek Denetleyici Durum Makinesi

Buyruk önbelleği denetleyici istenilen verinin veri önbelleğinde bulunmaması durumunda boru hattını durdurarak ana bellekten getirir. Bu durumu kontrol eden durum makinesi Şekil 3.5'de görünmektedir. Bellek Oku 0 durumu Şekil 3.4'de gösterilen çift adreslerin getirilmesinden sorumludur. Bellek Oku 1 durumu ise yine aynı şemada gösterilen tek adreslerin getirilmesinden sorumludur. Buyruk önbelleği istenilen verileri içerdiği takdirde B$ Oku durumunda bekler.

3.2.2. Veri Önbelleği

Veri önbelleği tasarımı, her satırında 8 bit genişliğinde etiket ve 32 bit genişliğinde veri içeren 512 satırlık Sram Dizisi barındırmaktadır. Önbellek doğrudan eşlemeli bir tasarıma sahiptir ve her adres satırının erişebileceği yalnızca bir satır mevcuttur. Bu tasarım başlangıç için basit olması nedeniyle seçilmiştir ve alt paragrafta tasarımın nasıl güncellenebileceği açıklanmaktadır.

2 yollu kümeli eşlemeli 2 KB boyuta sahip bir önbellek aynı boyuta sahip doğrudan eşlemeli bir önbelleğe göre %20 daha fazla bulma (hit) oranına sahiptir [17] ve yol sayısının önbelleklerde kümeli eşleme sayısının iki katına çıkarılması boyutunun iki katına çıkarılmasına yakın bir başarım artışı sağlar [18] [19]. Ayrıca makale çıktıları ve yapılan ölçümler karşılaştırıldığında oluşturulan doğrudan eşlemeli veri önbelleğinin beklendiği şekilde performans sağladığı görülmektedir. Bu bilgiler ışığında başarım artışı sağlanabilmesi için önbellekler 2-yollu kümeli eşlemeli olarak güncellemiştir [20]. Güncellenmiş önbellekler RISC-V testleri ile test edilmiş olup, başarım ölçümleri yapılmadığı için rapora eklenmemiştir.

Veri önbelleği tasarımında, başarımı belirleyen bir diğer faktör de Önbellek ve Ana bellek etkileşiminin nasıl sağlanacağıdır. Oluşturulan Önbellek Sisteminde Önbellek ve Ana bellek arasındaki trafiğin azaltılması hedeflendiği için "Write-back" ve "Write Allocate" olarak gerçekleştirilmiştir. Bu iki terim, Ana

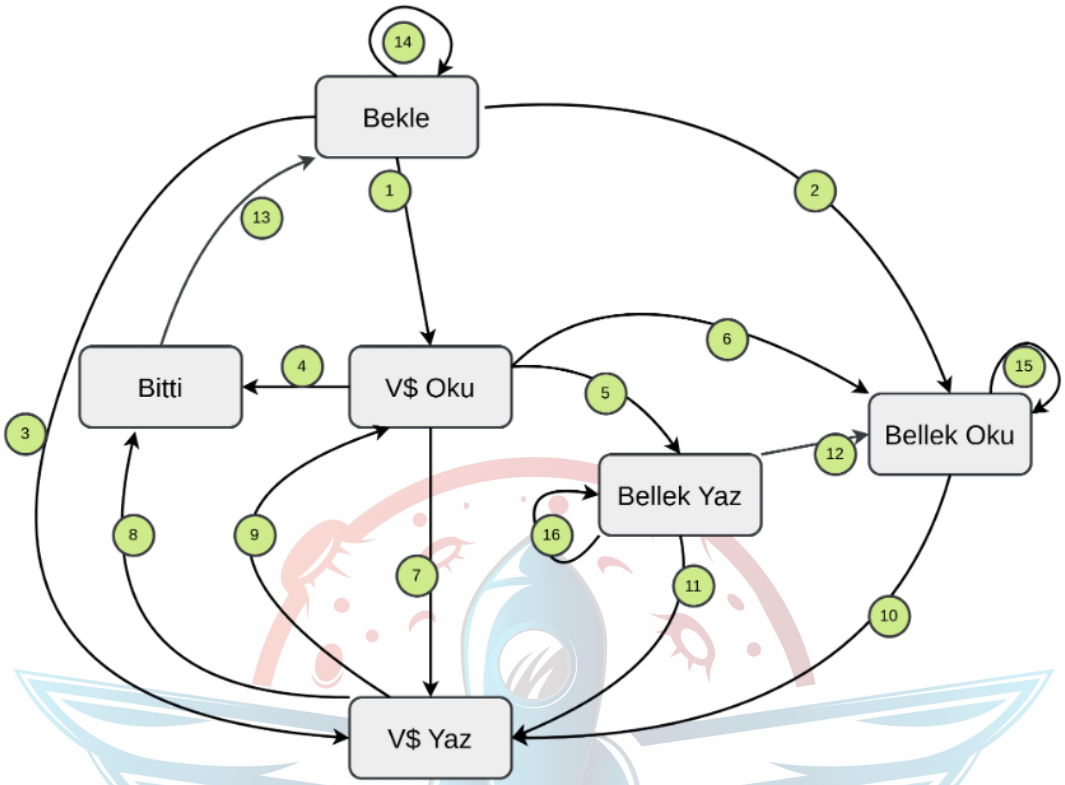
belleğe yazma işlemi yapılırken önbelleğin davranışını tanımlar. "Write-back" yazma işlemi yapılan verinin önbellekte bulunması sonucunda, verinin sadece önbelleğe yazılmasıdır. Verinin önbelleğe yazılacağı tek durum önbellek bloğunun yerine başka bir veri getirilmesi durumudur ve bu sayede ana belleğe daha az yazma işlemi yapılması mümkün olur. "Write Allocate" ise yazma işlemi yapılırken verinin önbellekte bulunmaması durumunda, ana belleğe yazılan verinin önbelleğe getirilmesi durumudur. Bu durum bellek sistemi 32 bit genişliğinde veri yoluna sahip olduğudan dolayı, sadece "Byte" ve "Half word" yazma işlemlerinde ortaya çıkmaktadır. 32 bitlik veri yazıldığında ise, bu verinin önbellekte bulunup bulunmamasından (hit-miss) bağımsız olarak veri sadece önbelleğe yazılır ve bu sayede ana belleğe daha az erişilir. Tablo 3.3 ile veri önbelleğinde yapılan okuma ve yazma isteklerini en düşük ve en yüksek gecikmeleri çevrim cinsinden gerçekleştirme süreleri verilmiştir. Çevrim sayılarının azaltılması için önbellek satırlarına ait geçerli (valid) ve kullanılmış (dirty) bitleri aynı çevrimde erişilebilen yazmaçlarda, 8 bitlik etiketler ve 32-bit veriler ise erişilmesi 1 çevrim süren SRAM dizilerinde saklanmaktadır. Bu sayede en sık karşılaşılan okumanın önbellekte bulunması (read hit) durumunun çevrimi azaltılmıştır. Şekil 3.6 ile ise veri önbelleği denetleyicinin durum makinesi görülmektedir, Önbelleklerin sakla (store) ve yükle (load) işlemlerindeki davranışları aşağıda yer almaktadır.

1- Yükle İsteği
i- Denetleyici, bir okuma isteği aldığında aynı çevrimde erişilen geçerli (valid) yazmaçlarını okur. Verinin okuduğu satır geçerli değilse veri bellekten okunur. Eğer geçerli ise etiket (Tag) karşılaştırılması için Önbellek satırı okunur.
ii- Eğer Önbellek Etiketi eşleşiyorsa işlem tamamlanmıştır. Diğer durumlarda ise veri Bellekten getirilir ve Önbelleğe yazılır. Önbelleğe yazılan veri Önbellekten tekrar okunur ve bu sayede işlem tamamlanır. Takip edilen yol Tablo 3.3 ile görülebilir.

2- Sakla İsteği
i- Denetleyici, bir yazma isteği aldığında aynı çevrimde erişilen geçerli yazmaçlarını okur ve işlem sonlanır. Verinin yazılacağı satır geçerli veya kirli değilse Önbelleğe yazma işlemi yapılır. Eğer Önbellek satırı kirli ise etiketin okunması için Önbellek okuma işlemi yapılır. Eğer yazma isteği word yazmıyorsa, veri bloğunun okunması için Ana bellek okuma işlemi yapıldıktan sonra Önbelleğe yazma işlemi yapılır.
ii- Önbelleklerde Etiketin eriştiği durumda Önbelleğe yazma işlemi yapılır. Eğer etiket eşleşmiyorsa önbellekten çıkarılacak veri ana belleğe yazıldıktan sonra Önbelleğe veri yazılabilir.

Tablo 3.3. Veri Önbelleği Gecikme ve Durum Geçişleri Tablosu

Veri Belleği İsteği	**Çevrim Sayısı**	**Durum Makinesi İzlenilen Yol**
Yazma (En İyi Durumda)	3	3 –>8 –>13
Yazma (En Kötü Durumda)	36	1 –>5 –>12 –>10 –>8 –>13
Okuma (En İyi Durumda)	3	1 –>4 –>13
Okuma (En Kötü Durumda)	37	1 –>5 –>12 –>10 –>9 –>4 –>13



Şekil 3.6. Veri Önbellek Denetleyici Durum Makinesi

3.2.3. Ana Bellek

Teknofest çevreleyici (wrapper) modülü tarafından sağlanmaktadır. Ana bellek boyutu maksimum 131072 satır (0x20000 satır) yani 0x80000 bayttır. Bu da yaklaşık 512 KB boyutuna denk gelmektedir.

3.3. Çevre Birimleri Tasarımı

Bütün çevre birimleri için Wishbone bus yapısı kullanılmış ve Wishbone B4 Classic Slave standardı [21] kullanılarak gerçekleştirilmiştir. Wishbone arayüzü, veriyolu denetleyici birimiyle iletişim sağlayarak çevre birimlerinin çekirdekten gelen programa göre kontrolünü sağlar. Wishbone arayüzüne gidecek olan yükleme (store) ve saklama (load) buyrukları çekirdekteki bellek işlem birimi tarafından seçilir ve ilgili çevre birimini kontrol etmek için ilgili adrese gönderilir.

3.3.1. UART

UART (Universal Asenkron Alıcı/Verici) iletişim protokolü, seri iletişim için yaygın olarak kullanılan bir protokoldür. UART protokolü, iki cihaz arasındaki iletişim için bir veri yolu sağlar. Veri yolu, verileri seri olarak gönderir ve alır. Her bir veri iletimi, başlangıç biti, veri bitleri, ve durdurma biti içerir. Başlangıç biti, veri iletiminin başlangıcını belirtir ve her zaman ”0” olarak ayarlanır. Veri bitleri, gönderilen verinin kendisidir ve 8 bittir. Durdurma biti, veri iletiminin sonunu belirtir ve her zaman ”1” olarak ayarlanır. Bu bitlerin sirasi ve kontrolu sıra (queue) tabanlı bir veriyolu (datapath) ve sonlu durum makinesi (Finite State Machine) kullanılarak gerçekleştirilmiştir. Alıcı (RX) ve verici (TX) bölümleri ayrı olarak oluşturulmuş ve Wishbone Slave protokolü kullanan bir modül altında birleştirilmiştir. UART protokolü, cihazlar arasında veri alışverişi yapmak için birçok farklı hızda kullanılabilir ve bu hızlar genellikle baud hızı olarak ifade edilir. Gerekli olan iletişim hızını (baudrate) elde etmek için de basit bir sayaç (counter) kullanılmıştır. Bu sayaçtan çıkan sinyal bir saat periyodu boyunca aktif kalarak gerekli olan senkronizasyonu sağlamaktadır.

3.3.2. SPI

SPI denetleyici çekirdek tarafından Wishbone arayüzü aracılığıyla kontrol edilmektedir. Denetleyicinin içerisine gömülü şekilde çalışan Wishbone çırak (İng. slave) ile gönderilen istekler denetleyicide erişmek istediği adrese göre işlevini gerçekleştirir. SPI protokolü için oluşturulan denetleyicinin tasarımında açık kaynaklı kodlar incelenmiştir. SPI protokolü ile haberleşen birçok cihazın veri sayfası (datasheet) [22] incelenerek hatalar ayıklanmıştır. Denetleyicinin tasarımında SCK sinyalinin kontrolü içerde tutulan bir sayaç sayesinde yapılmıştır. Kullanıcıdan gönderilecek olan CPHA ve CPOL değerleri üzerine bu sayaç örnekleme (sample) yaptığı kenarı belirlemektedir. Denetleyici içerisinde 8x32 Bitlik 3 tampon (buffer) bulunmaktadır. Bunlar MOSI, MISO ve buyruk (command) tamponlarıdır.

MOSI tamponu ilk giren ilk çıkar (first in first out (FIFO)) düzenine sahiptir. Şartnamede belirtilen SPI_WDATA adresine (0x2001000C) yazma isteği gönderilerek doldurulur. Gönderilen isteğin taşıdığı veri bu tamponun kuyruğunun (tail) bulunduğu dizine yazılır. ve kuyruk bir artırılır. MISO tamponu da bir önceki gibi ilk giren ilk çıkar düzenine sahiptir. SPI denetleyicide miso_en sinyali aktifleştirildikten sonra miso pinindeki voltaj örneklenerek tamponun seçili dizini doldurulur. Aktarım tamamlandığında ya da 32 bitlik veri öbeği tamamlandığında kuyruk bir artırılarak sıradaki dizine geçilir. SPI_RDATA adresine (0x20010008) okuma isteği gönderilerek ise daha önceden doldurulmuş tamponun başındaki değer okunur. Okuma sonrasında bütün tampon kaydırılır. MOSI pininden dışarı gönderilen ve MISO pininden okunan veriler küçüğü başta olacak şekilde tamponlara yerleştirilir. İçerisindeki buyruk tamponu sayesinde ise birden fazla buyruk isteği gönderildiğinde bu işlemleri sıraya koyarak ardı ardına gerçekleştirir. Nihai tasarıma ulaşıldığında Arduino UNO içerisine slave kodu yazılarak okuma ve yazma testleri yapılmış ve verilerin aktarımı gerçek zamanlı olarak gözlenmiştir. Gözlemler sonucu modülün istenen şekilde çalıştığı görülmüştür.

3.3.3. PWM

PWM, Pulse Width Modulation (Darbe Genişlik Modülasyonu) anlamına gelir. PWM denetleyicisi, giriş sinyalini işleyerek, belirli bir yükü çalıştırmak için bir çıkış sinyali üretir. Bu yük genellikle bir motor, bir ışık veya bir ısıtıcı gibi bir elektromekanik cihazdır. PWM denetleyicisi, bir yükü belirli bir hızda ve güçte çalıştırmak için kullanılır. PWM denetleyicisi, giriş sinyalini işleyerek belirli bir duty cycle (iş döngüsü) değeriyle bir çıkış sinyali üretir. Duty cycle, çıkış sinyalinin yüksek seviye süresinin giriş sinyalinin periyoduna oranıdır. Duty cycle, yüzde cinsinden ifade edilir. Örneğin, bir duty cycle değeri %50 ise, çıkış sinyalinin yüksek seviye süresi, giriş sinyalinin periyodunun yarısı kadardır. PWM denetleyicileri, analog sinyalleri dijital sinyallere dönüştürür ve bu dijital sinyalleri işleyerek yükün belirli bir güç ve hızda çalışmasını sağlar. Bu nedenle PWM denetleyicileri, genellikle mikrodenetleyicilerle birlikte kullanılır. PWM denetleyicileri, çeşitli alanlarda kullanılır. Örneğin, motor sürücüleri, güneş panelleri, LED aydınlatmalar, ısıtıcılar, fanlar ve piller gibi birçok uygulamada kullanılabilirler. Kalp Atışı Modunda darbe genişliği sabit kalır ve darbe periyodu değişir. Bu nedenle, çıkış sinyalinin frekansı değişir. Bu mod, batarya ömrünü uzatmak için kullanılır.

Standart Modunda, darbe periyodu sabit kalır ve darbe genişliği değişir. Bu nedenle, çıkış sinyalinin frekansı sabit kalır. Bu mod, motor hızı veya parlaklık gibi sabit bir çıkış gerektiren uygulamalar için kullanılır. PWM denetleyici tasarımında iç birimler açık kaynaklı [23] olarak kullanılmış olup, takımımız tarafından wishbone arayüzüne sahip ve PWM modlarını kontrol eden bir dış bir denetleyici modülü yazılmıştır.

3.3.4. TIMER

Teknofest çevreleyici (wrapper) modülü tarafından sağlanan ve sürekli sayan zamanlayıcı yazmacıdır. 0x30000000 adresi ile Timer'in alt 32 biti 0x30000004 adresi ile ise Timer'in üst 32 biti okunabilmektedir.

4. ÇİP TASARIM AKIŞI

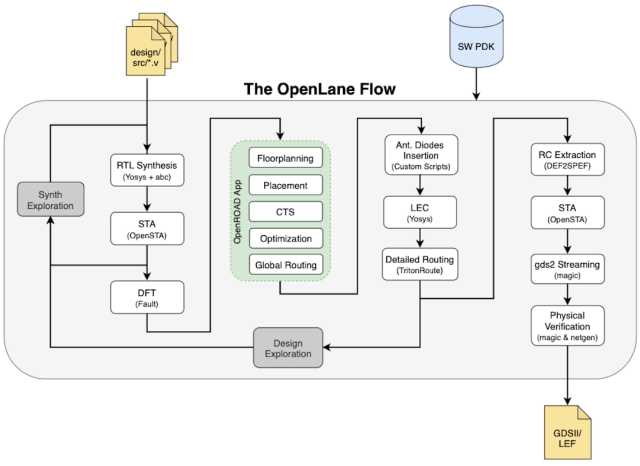
Çip Tasarım Akışında OpenLane [1], DFFRAM [4] ve OpenRAM [2] kullanılmıştır. Tasarım çeşitli makrolar halinde oluşturulup en son OpenLane altında bir araya getirilmiştir. En son geçirilen OpenLane akışının alan, frekans ve güç tüketimi değerleri Tablo 4.1'da görülmektedir.

Tablo 4.1. Openlane Makroları ve Performansları

Openlane Makrosu	Alan (um x um)	Frekans (MHz)	Güç (mW)
Çekirdek	1250 x 450	50	48,6
Veriyolu	850 x 450	50	10,4
DFFRAM256 (x 8bit)	246 x 446	95,3288	7,14
DFFRAM512 (x 16bit)	435 x 878	89,206	27

Güç tüketim değerleri ilgili makronun rpt dosyasından, frekans ve alan değerleri ise yine aynı klasör yapısında bulunan csv dosyasından alınmıştır.

Makroların kullanım amaçları ve nasıl sentezlendikleri ilgili bölümlerde açıklanmıştır. Makroların teker teker oluşturulması 1 hafta gibi bir sürede gerçekleştirilip. 1 hafta da bütün makroların bir araya getirilmesi için harcanmıştır. Makrolar nominal köşe değerler haricinde Hot ve Cold köşeler için de sorunsuz geçirilmiştir. Frekans ve güç tüketimi haricinde diğer sentez değerlerinde köşelere bağlı kayda değer bir değişim olmamaktadır.



Şekil 4.1. Openlane Çip Tasarım Akışı

Openlane: Birçok açık kaynaklı projenin bir araya gelmesinden oluşan bir VLSI akışı projesidir. OpenLane'in RTL sentez aşamasından Yosys sorumludur. Yosys verilen RTL girdisini SKY130 PDK [24] kütüphanesi kullanarak kapı seviyesine sentezler. Bu aşamada gerek modüller arası gerekse de modül içi sadeleştirmeler yapılır. Bu aşamada şifreleme buyrukları için gerekli olan sıfır sayıcı bölümünü 32 bit'lik 'case' kullanarak gerçekleştirilmeye çalışıldığı için ilk seferinde sorunlarla karşılaşılmıştır. Yosys 'case' yapılarını hash map olarak gerçekleştirdiği için ve 32 bitlik joker karakter (wildcard) kullanılarak yazılan bir 'case' yapısının Python dilinin hashmap'inin tutabileceği anahtar (key) sayısından fazla olmasından

dolayı değişiklikler gerçekleştirmiştir. Bu sorun sıfır sayıcının daha resmi bir algoritmayla [15] gerçeklenmesi sayesinde çözülmüştür. Sentezleme işleminden sonra OpenSTA tarafından kapı seviyesinde statik zamanlama doğrulaması gerçekleştirilir. Bu aşamada çeşitli parametreler aracılığı ile tasarım isterler doğrultusunda optimize edililir. Global Routing'den FastRoute sorumludur. Global Placement kısmında varsa makrolar ve bunlar harici bulunan kapılar genel olarak belirtilen alan veya otomatik yerleştirme (BASIC MACRO PLACEMENT) kullanılarak genel olarak dağıtılır. BASIC MACRO PLACEMENT maalesef iki taneden fazla makro olması durumunda fazla uzun sürdüğü için makrolar el ile yerleştirilmiştir. Sonrasında gelen Detailed Placement kısmından Floorplan, TritonMacroPlacer ve RePlAce programları sorumludur. Bunlar Global Placement'ta genel olarak yerleştirilen kapıları detaylı bir şekilde bağlar. Bu aşamada tıkanıklık (congestion) sorunlarıyla karşılaşılmış olup PL_TARGET_DENSITY ve DIE_AREA değerlerinin değiştirilmesi ile çözülmüştür. Ayrıca tıkanıklık olan bölümün anlaşılması ve makroların pozisyonlarının tespiti için OpenRoadGUI kullanılmıştır. OpenLane akışının güç dağıtım ve saat ağacı ağından ise sırasıyla PDN ve TritonCTS sorumludur. Detailed routing çıktısı yine Yosys tarafından gerçekleştirilecek olan mantıksal eşlik testine tabi tutulur. Bu noktada elde edilen çıktı tekrardan akış döngüsüne sokularak çeşitli isterler doğrultusunda optimize edilebilir. Aksi takdirde OpenLane akışının kalan kısımları elde edilen fiziksel çıktının doğrulanması üzerine odaklanır. DRC, LVS, XOR eşliği ve Anten testleri gerçekleştirilir ve tasarım akışı elde edilen çıktının GDSII dosyasına çevirilmesiyle sona erer. Openlane akışı sonucunda çeşitli hatalar konusunda raporlar oluşturulur. Bunlardan en çok karşılaşılan max capacitance ve slew hatalarıdır. Bu hataların çözülmesinde başlangıçta oldukça zorlanılsa da aşağıda bulunan konfigürasyon değişkenleri sayesinde, bütün makrolar slew ve kapasitans hataları olmadan geçirilmiştir. Bu değişken değerleri deneme yanılma yöntemiyle bulunmuş olup vakit kazanılması amacıyla değerlere fazlaca pay verilmiştir.

```
"PL_RESIZER_MAX_CAP_MARGIN": 50,
"GLB_RESIZER_MAX_CAP_MARGIN": 50,
"GLB_RESIZER_MAX_SLEW_MARGIN": 50,
"PL_RESIZER_HOLD_MAX_BUFFER_PERCENT": 60,
"PL_RESIZER_SETUP_MAX_BUFFER_PERCENT": 60,
"GLB_RESIZER_HOLD_SLACK_MARGIN": 0.1,
"GLB_RESIZER_SETUP_SLACK_MARGIN": 0.1
```

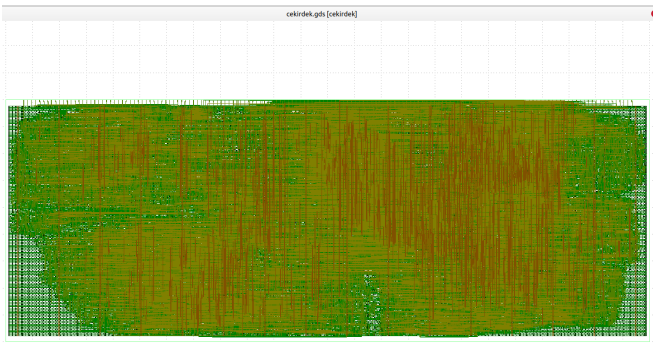
Max kapasitans ve slew değerleri haricinde setup ve hold değerleri, anten ve fanout violationları da kontrol edilmiştir. Anten violationları çipin üretiminde kullanılan yüklü iyonları yarı iletkeni etkileyebileceği yerleri göstermektedir. Fazla olması bir silikon diskinden (wafer) elde edilen çalışan çip sayısını azaltır. Bu hataları azaltmak için DIODE_INSERTION_STRATEGY değişkeni kullanılmıştır. Fakat, anten violationları sıfırlanamamıştır. Fanout violations ise slew ve kapasitans değerlerine doğrudan bağlı olan bir uyarıdır. Fanout'un fazla olması kapının belirlenen limitten fazla kapıya doğrudan bağlı olduğu anlamına gelir ve bu limiti aşan kapılar genelde kapasitans hatası da verir. Eğer kapasıtans ve slew hatası yoksa bu değer umursanmayabilir. Fakat bu değerin azaltılması genel olarak tasarımın kapasitans, slew ve frekans değerlerinin iyileşmesine yol açacaktır.

DFFRAM: DFFRAM açık kaynaklı bir RAM sentez programı olup flip-flop ve latch tabanlı RAM'ler sentezleyebilmektedir. Resmi olarak 32 bitlik RAM'ler sentezleyebilmesine rağmen FORCE_ACCEPT_SIZE değişkeni kullanılarak 8 ve 16 bitlik RAM'ler de sentezlenebilir. Buyruk önbelleğinde kullanmak amacıyla 256x8 ve 512x16 boyutlarında iki tane latch tabanlı tek girişli RAM sentezlenmiştir. DFFRAM placement

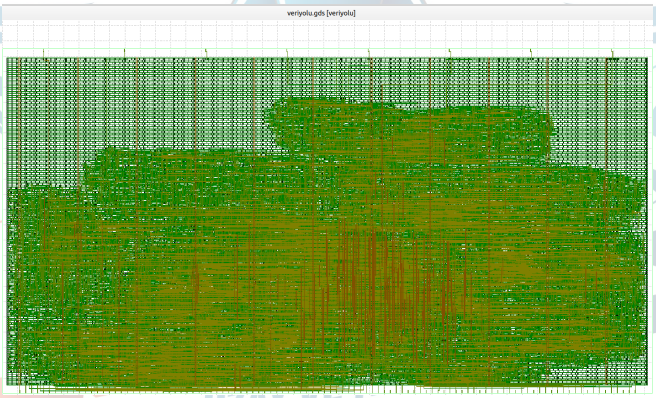
haricindeki aşamalarda OpenLane kullandığı için anten, kapasitans ve slew gibi raporlar OpenLane ile aynıdır. Elde edilen makrolar kapı seviyesi (gate-level) testlerle doğrulanmıştır.

OpenRAM: OpenRAM açık kaynaklı bir RAM sentezleme programı olup SRAM tabanlı RAM'ler sentezleyebilir. Veri onbelleğinde kullanmak amacıyla 8 bit etiket ve 32 bit verinin beraber olduğu 40 bitlik bir SRAM sentezlenmiştir.

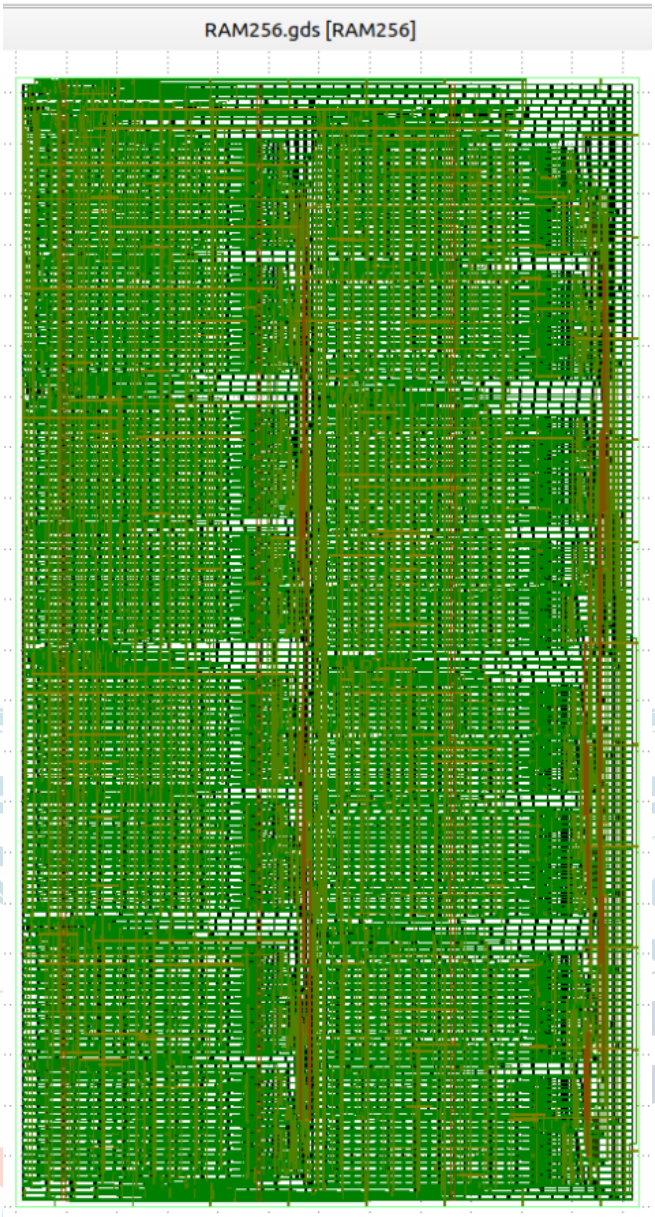
Openlane makroları aşağıdaki şekillerde görülebilir.



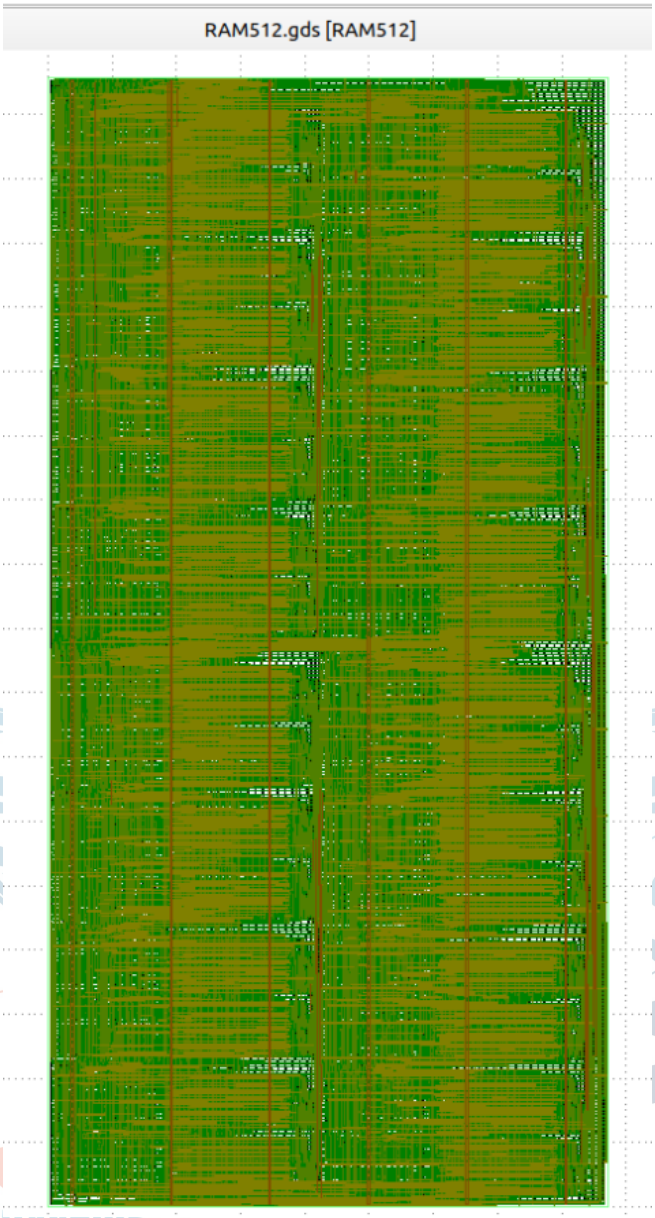
Şekil 4.2. Openlane Çekirdek Makrosu



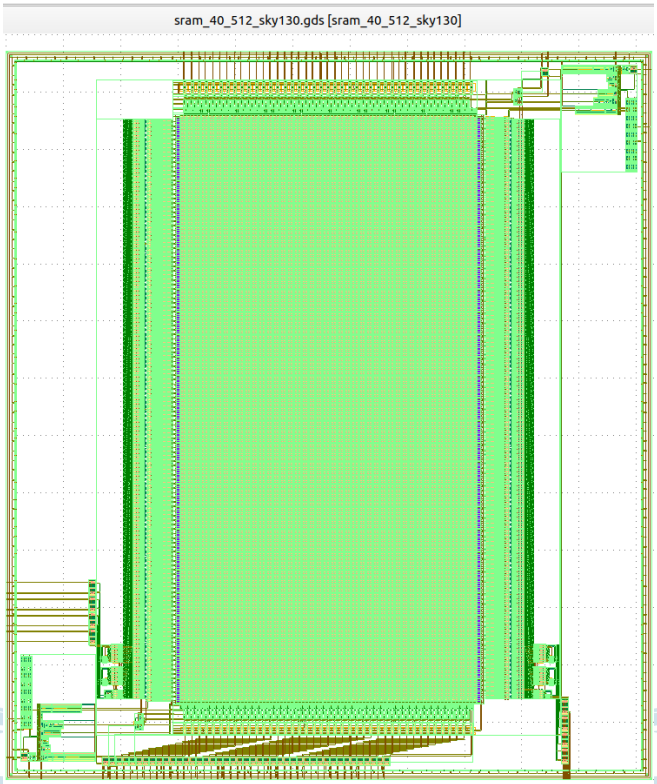
Şekil 4.3. Openlane Veriyolu Makrosu



Şekil 4.4. DFFRAM 256x8 Buyruk Önbelleği Tag Makrosu



Şekil 4.5. DFFRAM 512x16 Buyruk Önbelleği Veri Makrosu



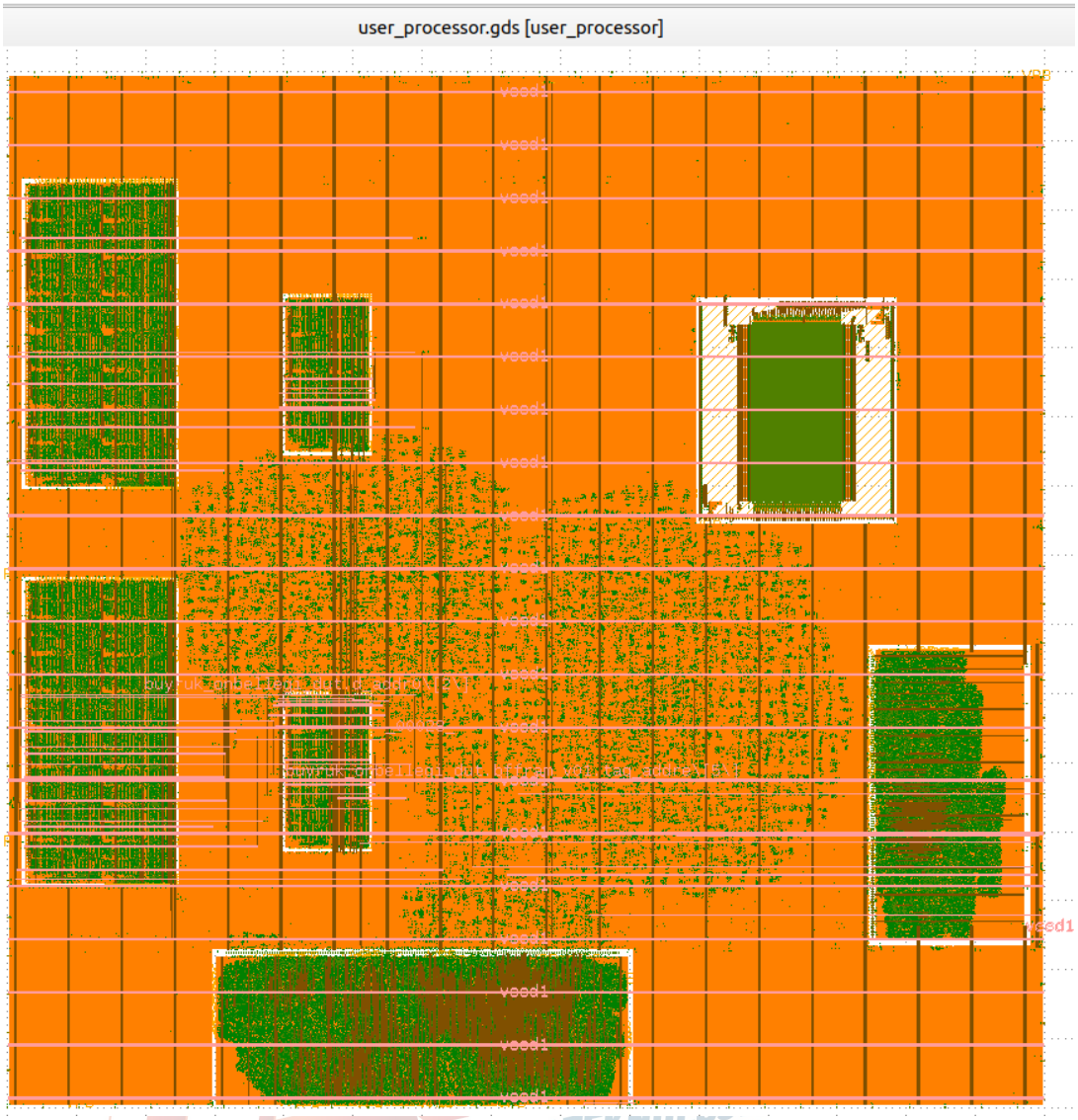
Şekil 4.6. SRAM 40x512 Veri Önbelleği Makrosu

Bu Openlane makrolarının performansları Tablo 4.2'de görülebilir.

Tablo 4.2. Openlane Makro Performansları

Openlane Makrosu	**Alan (um x um)**	**Frekans (MHz)**	**Güç (mW)**
Çekirdek	1250 x 450	50	48,6
Veriyolu	850 x 450	50	10,4
DFFRAM256 (x 8bit)	246 x 446	95,3288	7,14
DFFRAM512 (x 16bit)	435 x 878	89,206	27

Öncelikle Openlane'den makro olarak kullanma amacıyla geçirilen Şekil 4.2, 4.2, 4.4, 4.5, 4.6'daki modüller daha sonra user_processor (işlemci) modülünü sentezlemek için kullanılarak tekrar Openlane'den geçirilmiştir ve Şekil 4.7'de görülebilir. Openlane'den Geçmesi İçin alan çok büyük tutularak (3000um x 3000um) routing rahatlığı sağlanmıştır.



Şekil 4.7. İşlemcinin Openlane'den Geçmiş Hali

5. TEST

Testler projede en çok üzerinde zaman harcadığımız kısımdır. Tasarımın doğru çalıştığını doğrulamak için kullandığımız tüm test araçları sırasıyla; cocotb [25], verilator [26], icarus verilog [27], modelsim, Vivado, Quartus, Openlane, Yosys [28], spike [29] ve Rars [30] şeklindedir.

Bu test araçları beraber veya yalnız kullanılarak aşağıda verilen testlerin tamamı başarıyla geçirilmiştir. Açık kaynak testler: Riscv-tests, Riscv-arch-tests, Appg [31] ve Spike, Coremark, Dhyrstone, Kendi Testlerimiz: UART, SPI, PWM testleri, TIMER testi, önbellek testleri.

Çip tasarım akışı sonrasında elde edilen makrolar kapı seviyesi simülasyonlar aracılığı ile test edilmiştir.

Testler simülasyonlar dışında FPGA'ler üzerinde de koşturulmuştur. İşlemci, elde bulunan AMD Virtex Ultrascale VCU108'de 125 MHz'de, Xilinx Basys3'de 50 MHz'de çalıştırılmıştır. Şartnamede belirtilen Nexys A7 XC7A100T-1CSG324C FPGA elde bulunmadığından denenememiştir fakat Vivado'da 50 MHz'de implementasyonun geçtiği görülmüştür.

5.1. Çekirdek Testleri

Şartnamede CSR denetim buyrukları istenmediği için çekirdeğe eklenmemiştir, bundan dolayı da RISC-V testleri direkt test satırlarına atlayacak şekilde modifiye edilmiş, genel olarak tüm c programları da CSR buyrukları olmayan linker scriptler ile derlenmiştir.

Testler 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 arttırmıştır.

Bu testlerden en çok hata bulmamızı sağlayan test aapg olmuştur. Kapsama tabanlı rastgele testler üretebilen aapg sayesinde 100 milyarlarca istatistiksel olarak üretilmiş buyruklarla spike emülatörünü referans olarak kullanarak imza tabanlı(signature based) ve iz tabanlı (trace based) olarak hem de FPGA tasarımı üzerinde testler gerçekleştirdik. Bu testlerin haricinde de işlemicimiz Coremark ve Dhyrstone gibi benchmarklar yürütülerek doğrulanmıştır. Bu testlere ve benchmarklara ek olarak çeşitli uç senaryoları test eden mikro programlar yazılmış ve çalıştırılmıştır.

5.2. Çevresel Birim Testleri

UART, SPI ve PWM modülleri için aşağıdaki C kodları yazılmış ve her çevresel birime özel testler yapılmıştır;

•uart_demo.c –> UART alıcı, verici değişik baudratelerde (örneğin 115200) test edilmiştir. Şartnamede belirtilen 1 MBps hızı desteklediği görülmüştür.

•spi_demo.c –> SPI alıcı ve verici test edilmiştir. SPI modları arasında geçiş yapılarak örneklemelerin doğru zamanda ve doğru şekilde yapılması doğrulanmıştır. Slave olarak Arduino UNO'da slave SPI kodu yazılarak ve FPGA ile Arduino UNO arası kablo bağlantıları yapılarak denenmiştir.

•pwm_demo.c –> L298N [32] motor sürücü modülü kullanılıp 2 adet DC motor bağlanarak standard ve kalp atışı modları farklı süre, adım ve eşik frekansı değerleriyle test edilmiş ve beklenen çıkışlar fiziksel olarak gözlemlenmiştir, motorların istenilen şekillerde döndüğü doğrulanmıştır.

•uart_timer_demo.c –> Teknofest çevreleyici (wrapper) modülde bulunan sayaç, 0x30000000 ve 0x30000004 adreslerinden alt ve üst 32 bitler okuma yapılarak test edilmiş ve UART üzerinden örneğin FPGA frekansına göre saniye saniye değerler basılmış ve hesaplanan sürenin doğru gelip gelmediğine bakılmıştır.

Çevresel birimler bu şekilde, olabildiğince fiziksel işlemlerle FPGA (AMD Virtex Ultrascale VCU108) üzerinde doğrulanmıştır.

5.3. Benchmark Programları

Baremetal çekirdeklerin performans ölçümleri için en popüler benchmark programları olan CoreMark ve Dhrystone programları AMD Virtex Ultrascale VCU108 FPGA'inde 125 MHz'de çalıştırılmıştır.

5.3.1. CoreMark

CoreMark'ın [33] işlemcinin UART'ına sonuç basması ve çalışırken işlemcide geçen zamanı Timer'dan ölçmesi için ee_printf ve time fonksiyonları portlanmıştır. CoreMark'ı derlemek için TÜBİTAK-TÜTEL tarafından verilen riscv-gnu-toolchain [34] kullanılmıştır. Linker script'te csr buyrukları kapatılıp stack ve global pointerlar bellekte doğru ve boş bir yeri gösterecek şekilde ayarlanmıştır. Ayrıca saat hızı

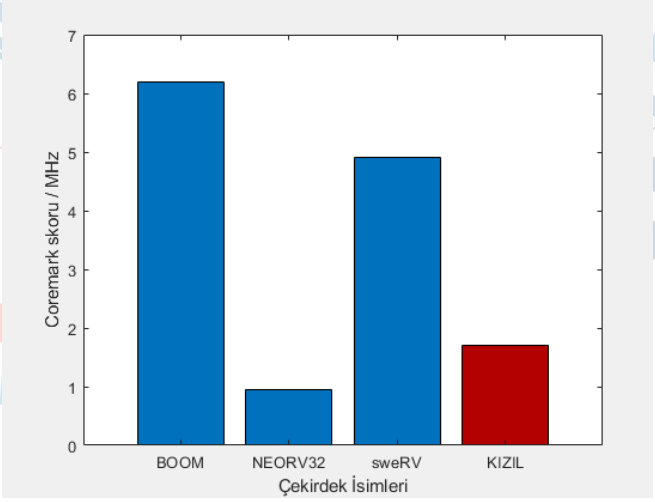
da işlemcinin FPGA'de (AMD Virtex Ultrascale VCU108) çalışma frekansına eşit olacak şekilde 125 MHz olarak ayarlanmıştır.

CoreMark 10 saniyenin altında güvenilir sonuç vermeyeceğini söylediğinden programın 10 saniyenin üzerinde çalışması için 3000 iterasyon verilmiştir. Kasırga Kızıl işlemcimiz geçen senenin teknofest kazananı olan Kasırga işlemcisinden Coremark testinde 4 kattan daha hızlıdır. [35].

```
2K performance run parameters for coremark.
CoreMark Size    : 666
Total ticks      : 1812232133
Total time (secs): 14
Iterations/Sec   : 214
Iterations       : 3000
Compiler version : GCC12.2.0
Compiler flags   : -O2 -mcmodel=medany -static -std=gnu99 -fno-common -nostdlib
-nostartfiles -fno-builtin -ffunction-sections -lm -lgcc -T riscv32-baremetal/li
nkdene.ld -DPERFORMANCE_RUN=1
Memory location  : STACK
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0xcc42
Correct operation validated. See README.md for run and reporting rules.
```

Şekil 5.1. 125 MHz ve 3000 İterasyonda CoreMark Çıktısı

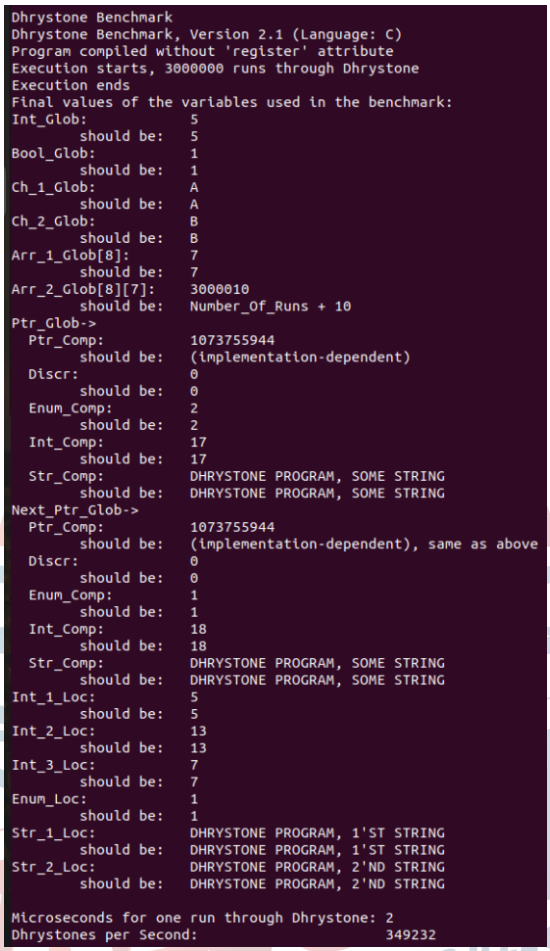
Şekil 5.3'de görüleceği üzere 125 MHz saat frekansındayken CoreMark saniyede 214 iterasyon sonucunu vermiştir. Yani CoreMark/MHz 1,71 değerine sahiptir ki bunu Şekil 5.2'te görüleceği üzere diğer açık kaynaklı RISC-V işlemci skorlarıyla karşılaştırabiliriz. Kızıl işlemcinin iyi bir performans gösterdiğini grafikten de görebiliyoruz.



Şekil 5.2. Açık kaynaklı RISC-V İşlemcileri ile Kızıl İşlemci Karşılaştırması

5.3.2. Dhrystone

CoreMark'a benzer şekilde Dhrystone [36] benchmark'ı da portlanmıştır ve Şekil 5.3'da görüleceği üzere Dhrystone'da da 349232 olarak bir hayli yüksek skor alınmıştır.



Şekil 5.3. 125 MHz ve 3000000 İterasyonda Dhrystone Çıktısı

5.4. LLVM/Clang Tabanlı Özelleştirilmiş Derleyici

Testleri derlemek için başlangıçta TÜBİTAK-TÜTEL'in paylaştığı GNU/GCC tabanlı riscv-gnu-toolchain kullanılmıştır fakat bu derleyici, özel buyruk işlemleri için sadece assembly yazdığınızda ya da C'de inline assembly [37] kod yazdığınızda özel buyruklara çevirebilmektedir. Özel buyrukları direkt C kodundan yakalamak için LLVM RISC-V backendine özel buyrukları ve örüntüleri eklenmiştir ve böylelikle C kodunda assembly yazmadan direkt olarak mantıksal işlemlerde ilgili özel buyruğun (şifreleme buyrukları) örüntüsü yakalanarak assembly kodlarına çevirme işlemleri başarılmıştır.

Örneğin aşağıda C kodunda yazılmış olan hamming_distance fonksiyonu LLVM RISC-V backendine yapılan eklemeler sayesinde yakalanıp hmdst şifreleme buyruğuna, popcount fonksiyonu ise cntp şifreleme buyruğuna çevrilebilmektedir:

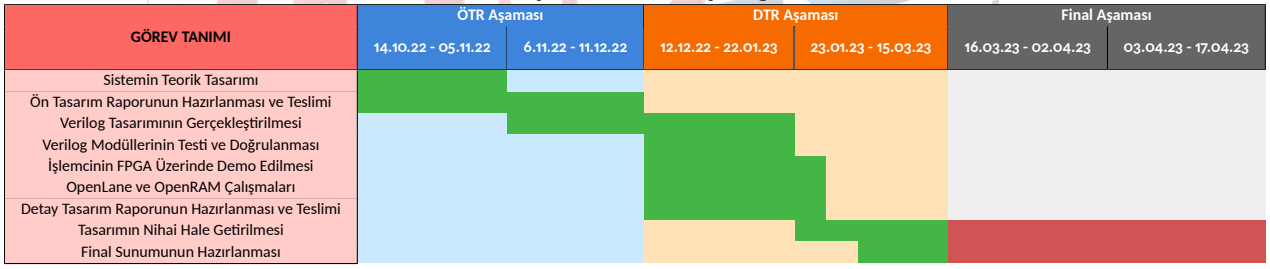
```
unsigned popcount(unsigned reg1){
    unsigned bir_sayisi = 0;
    int i;
    for(i = 0; i < 32; i++){
        bir_sayisi += (unsigned) 1 & (reg1 >> i);
    }
    return bir_sayisi;
}

unsigned hamming_distance(unsigned reg1, unsigned reg2){
    unsigned hamming_dist = 0;
    unsigned differences = reg1 ^ reg2;
    return popcount(differences);
}
```

Not: *Özel buyruklardan şifreleme buyrukları, basit aritmetik ve mantıksal işlemlere dayandıkları için örüntüleri kolayca yakalanmıştır fakat konvolüsyon buyrukları için bu durum söz konusu değildir ve çok özel boyutta matrisleri (1x1 ... 4x4) desteklediğinden bunları C kodundan direkt olarak yakalamaya çalışmanın gereksiz efor sarfetmek olacağı kanısına varılmıştır fakat konvolüsyon buyrukları yine TÜBİTAK-TÜTEL'in paylaştığı gcc derleyicisinde olduğu gibi assembly ya da C'de inline assembly olarak yazıldığında LLVM/Clang tabanlı derleyicimizde de yakalanabilmektedir.*

6. İŞ PLANI

Tablo 6.1'de iş planı zaman çizelgesi görülebilir.

Tablo 6.1. İş Planı Zaman Çizelgesi



Tablo 6.1'de görüldüğü üzere iş planı zaman çizelgesi yarışma tarihlerine uygun olarak ÖTR, DTR ve Final olmak üzere 3 ana fazdan oluşmakta ve her faz iki aşamaya ayrılmaktadır. Her fazın ilk aşaması o fazın şartlarını yerine getirmek için olan hazırlık aşamasını kapsamaktadır. ÖTR'den bu yana DTR ve Final aşama tarihleri değiştiğinden tabloda bu alanlar da değiştirilmiştir. DTR aşamasına kadar iş planına uygun şekilde gidilmiş, şartnamede istenen tüm tasarım ve doğrulama isterleri sağlanmış dolayısıyla bu aşamaya kadar olan iş paketleri tamamlanmıştır. DTR aşamasından sonra, Final aşamasında, tasarım nihai hale getirilene kadar işlemciyi hızlandırmak ya da alandan kazanmak gibi amaçlar doğrultusunda optimizasyon işlemleri yapılacak ve daha sonra da final sunumu hazırlanarak iş paketleri tamamlanacaktır.

Not: *İş planında tarih dışında herhangi bir değişiklik olmamasına rağmen not olarak düşmek gerekir ki KASIRGA - KIZIL Takım Kaptanı Seyyid Hikmet Çelik yarışmaya katıldığında TOBB ETÜ Elektrik-Elektronik*

Mühendisliği Lisans 4. Sınıf öğrencisiyken DTR aşamasında TOBB ETÜ Bilgisayar Mühendisliği Yüksek Lisans 1. Sınıf öğrencisi olmuştur. T3 KYS sisteminde bu bilgiler güncellenmiştir.

KAYNAKÇA

1. A. Ghazy and M. Shalan, "Openlane: The open-source digital asic implementation flow," in *Proc. Workshop on Open-Source EDA Technol.(WOSET)*, 2020.

2. M. R. Guthaus, J. E. Stine, S. Ataei, B. Chen, B. Wu, and M. Sarwar, "Openram: An open-source memory compiler," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2016, pp. 1–6.

3. M. Temel, A. Slobodova, and W. A. Hunt Jr, "Automated and scalable verification of integer multipliers," in *Computer Aided Verification: 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21–24, 2020, Proceedings, Part I*. Springer, 2020, pp. 485–507.

4. "Dffram: Standard cell library based memory compiler using ff/latch cells," https://github.com/AUCOHL/DFFRAM.

5. B. Millar, P. E. Madrid, and E. Swartzlander, "A fast hybrid multiplier combining booth and wallace/dadda algorithms," in *[1992] Proceedings of the 35th Midwest Symposium on Circuits and Systems*. IEEE, 1992, pp. 158–165.

6. N. Takagi, S. Kadowaki, and K. Takagi, "A hardware algorithm for integer division," in *17th IEEE Symposium on Computer Arithmetic (ARITH'05)*. IEEE, 2005, pp. 140–146.

7. R. S. Balpande and R. S. Keote, "Design of fpga based instruction fetch & decode module of 32-bit risc (mips) processor," in *2011 International Conference on Communication Systems and Network Technologies*. IEEE, 2011, pp. 409–413.

8. "C standard extension for compressed instructions," https://riscv.org/wp-content/uploads/2015/11/riscv-compressed-spec-v1.9.pdf.

9. T. Kanamori, H. Miyazaki, and K. Kise, "Rvcorep-32ic: A high-performance risc-v soft processor with an efficient fetch unit supporting the compressed instructions," *arXiv preprint arXiv:2011.11246*, 2020.

10. M. Evers and T.-Y. Yeh, "Understanding branches and designing branch predictors for high-performance microprocessors," *Proceedings of the IEEE*, vol. 89, no. 11, pp. 1610–1620, 2001.

11. S. Mittal, "A survey of techniques for dynamic branch prediction," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 1, p. e4666, 2019.

12. S. McFarling, "Combining branch predictors," Citeseer, Tech. Rep., 1993.

13. I. Kim, J. Jun, Y. Na, and S. W. Kim, "Design of a g-share branch predictor for eisc processor," *IEEE Transactions on Smart Processing and Computing*, vol. 4, no. 5, pp. 366–370, 2015.

14. K. Asanovic, D. A. Patterson, and C. Celio, "The berkeley out-of-order machine (boom): An industry-competitive, synthesizable, parameterized risc-v processor," University of California at Berkeley Berkeley United States, Tech. Rep., 2015.

15. G. Dimitrakopoulos, K. Galanopoulos, C. Mavrokefalidis, and D. Nikolos, "Low-power leading-zero counting and anticipation logic for high-speed floating point units," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 16, no. 7, pp. 837–850, 2008.

16. B. Moyer, *Real World Multicore Embedded Systems*. Newnes, 2013.

17. "Berkeley cache presentation," https://inst.eecs.berkeley.edu/~cs61c/su12/lec/12/12LecSu12Caches-2.pdf.

18. M. Brehob and R. Enbody, "An analytical model of locality and caching," *Michigan State University, Department of Computer Science and Engineering MSU-CSE-99-31*, 1999.

19. N. P. Jouppi, "Cache write policies and performance," *ACM SIGARCH Computer Architecture News*, vol. 21, no. 2, pp. 191–201, 1993.

20. "Kasırga kızıl github repository'si," https://github.com/KASIRGA-KIZIL/tekno-kizil/tree/two-way-cache.

21. A. P. I. Cores, "Wishbone b4," 2010.

22. "Spi interface specification," https://www.mouser.com/pdfdocs/tn15_spi_interface_specification.PDF.

23. "Pwm openmpw," https://github.com/SerdarUnal132/pwm_openmpw.

24. "Open source process design kit for usage with skywater technology foundry's 130nm node," https://github.com/google/skywater-pdk.

25. B. J. Rosser, "Cocotb: a python-based digital logic verification framework," in *Micro-electronics Section seminar*. CERN, Geneva, Switzerland, 2018.

26. W. Snyder, "Verilator and systemperl," in *North American SystemC Users' Group, Design Automation Conference*, 2004.

27. "Icarus verilog," https://github.com/steveicarus/iverilog.

28. "Yosys open synthesis suite," https://github.com/YosysHQ/yosys.

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

30. "Rars – risc-v assembler and runtime simulator," https://github.com/TheThirdOne/rars.

31. "Aapg - automated assembly program generator for the risc-v isa," https://gitlab.com/shaktiproject/tools/aapg.

32. "L298n datasheet," https://pdf1.alldatasheet.com/datasheet-pdf/view/22440/STMICROELECTRONICS/L298N.html.

33. S. Gal-On and M. Levy, "Exploring coremark a benchmark maximizing simplicity and efficacy," *The Embedded Microprocessor Benchmark Consortium*, 2012.

34. "Teknofest 2023 Çip tasarım yarışması github deposu," https://github.com/TUTEL-TUBITAK/TEKNOFEST_2023_Cip_Tasarim_Yarismasi/blob/main/baremetal-tekno-sw/README.md.

35. "2022 teknofest kazananı: Kasirga takımı," https://www.youtube.com/watch?v=v3x10U11ib4.

36. R. P. Weicker, "Dhrystone: a synthetic systems programming benchmark," *Communications of the ACM*, vol. 27, no. 10, pp. 1013–1030, 1984.

37. "Inline assembly," https://en.cppreference.com/w/c/language/asm.