



**ÇİP TASARIM YARIŞMASI**  
**MİKRODENETLEYİCİ TASARIM**  
**KATEGORİSİ**  
**ÖN TASARIM RAPORU**

Takım Adı: Yeditepe Üniversitesi Sayısal Tasarım Topluluğu

Başvuru ID: #3062081

## 1. Giriş

Takımımız, Yeditepe Üniversitesi Elektrik-Elektronik ve Bilgisayar Mühendisliği lisans öğrencilerinden oluşmaktadır. 7Yonga projesi kapsamında, açık kaynaklı bir RISC-V çekirdeği, çevre birimleri ve veri yolları ile özgün bir mikrodenetleyici tasarlamayı hedeflemektedir. Bu mikrodenetleyici, yüksek doğruluk ve düşük güç tüketimi öncelikleri doğrultusunda geliştirilecektir.

Bu raporda, Sistem Mimarisi bölümünde genel tasarım ilkeleri ve şemalar ele alınacak; Tasarım Detayları bölümünde sistemin teknik gereksinimleri, bu gereksinimlerin nasıl karşılanması, süreç boyunca karşılaşabilecek olası hatalar ve bunlara yönelik çözüm yaklaşımlarımız açıklanacaktır. Ayrıca, yüksek doğruluk ve düşük güç tüketimi hedeflerimizin nasıl sağlanacağı üzerinde durulacaktır. Takım Organizasyonu ve İş Planı bölümünde ise 7Yonga projesi ekibimizin görev dağılımı, süreç yönetimi ve iş planamasına ilişkin detaylar sunulacaktır.

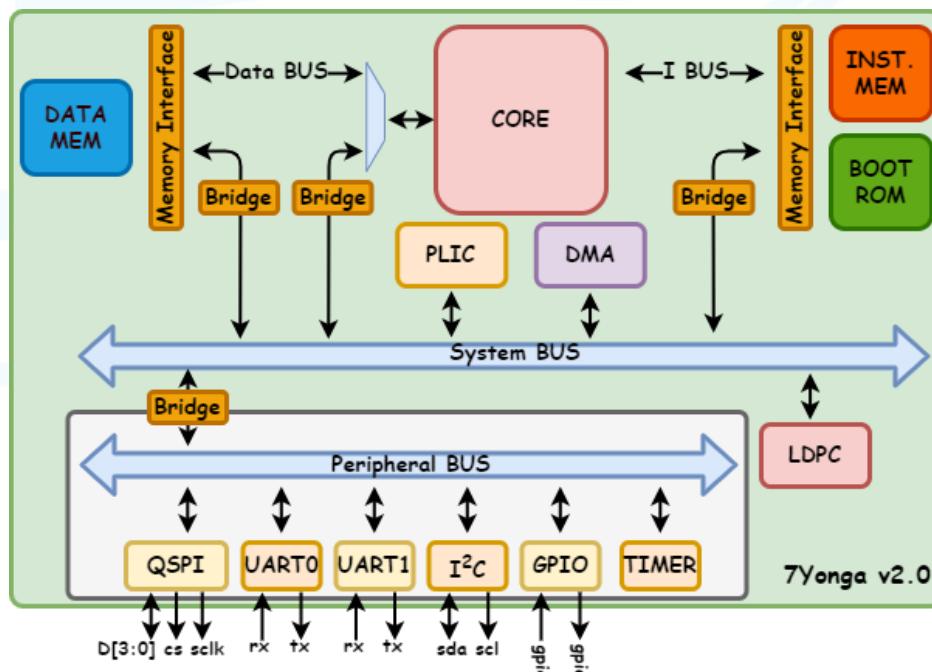
## 2. Sistem Mimarisi

Bu çalışmada, programların yürütülmesini sağlamak amacıyla bir işlemci çekirdeği, yürütülecek programların buyruk belleği ve veri belleği tasarlanmıştır. Önyükleme işlemlerini gerçekleştirmek için bir önyükleyici belleği bulunmakta olup, ayrıca LDPC (Low-Density Parity-Check Code) donanım hızlandırıcısı ve isterlerde belirtilen çevre birimleri sistemde yer almaktadır. Çekirdek, bellekler ve çevre birimlerini birbirine bağlamak üzere iki adet veri yolu kullanılmaktadır. Kesme yönetimi için PLIC (Platform-Level Interrupt Controller) kontrolcüsü, çevre birimlerinin veri işlemlerini hızlandırmak amacıyla ise DMA (Direct Memory Access) birimi tasarıma dahil edilmiştir.

Çevre birimlerinin tasarımında donanım hızlandırıcısı ve diğer bileşenlerin geliştirilmesi için SystemVerilog HDL kullanılacaktır. Testbench testleri ve FPGA testleri Vivado aracı ile gerçekleştirilecektir. Bunun yanı sıra, doğrulama süreçlerinde UVM (Universal Verification Methodology) ve Spike simülatörleri kullanılacaktır.

Son olarak, yarışma danışma ve değerlendirme kurulu (DDK) tarafından duyurulacak program ile tasarımın çip akışı yürütülecektir.

Kontrol birimleri, veri yolları ile bellekler [görsel 2.1](#)'de gösterildiği şekilde bir araya getirilecektir.



Görsel 2.1 – Mikrodenetleyici Tasarımı

### 3. Tasarım Detayları

#### 3.1 Mikrodenetleyici Tasarımı:

Bu bölümde mikrodenetleyici tasarımında bulunan modüllerin ve denetleyici birimlerin teknik detaylarından bahsedilecektir. Mikrodenetleyici tasarımımızda, STM32 Nucleo geliştirme kartları ve PULP Platform projesine ait PULPino tasarımından ilham alınarak gerçekleştirilmiştir. STM32 kartları ve PULPino, çevre birimlerinin entegrasyonu, veri yollarının düzenlenmesi ve bellek modüllerinin yerleştirilmesi gibi mimari tasarım kararlarında referans alınmış ve bağlantı yapıları bu örnekler doğrultusunda oluşturulmuştur [\[1\]](#)[\[2\]](#).

##### 3.1.1 Mikrodenetleyici Çekirdeği:

Tasarımızda mikrodenetleyici çekirdeği olarak OpenHW Group CORE-V CV32E40P RISC-V IP'si kullanılacaktır.

CV32E40P, küçük ve verimli bir 32-bit, in-order RISC-V çekirdeğidir. 4 aşamalı bir işlem hattına sahiptir ve RV32IM[F|Zfinx]C komut seti mimarisini destekler. Bunun yanı sıra, kod yoğunluğunu artırmak, performansı yükseltmek ve enerji verimliliğini artırmak için PULP özel uzantılarını içerir.

Dört aşamalı işlem hattına sahip bu işlemci, basit ve verimli bir yürütme süreci sunarken, RV32IM[F|Zfinx]C desteği sayesinde tamsayı (I), çarpma-bölme (M), sıkıştırılmış komutlar (C) ve isteğe bağlı kayan nokta (F/Zfinx) uzantılarını içermektedir. PULP özel uzantıları ile performans ve enerji verimliliği artırılmış olup, kompakt ve düşük güç tüketimli tasarımı sayesinde gömülü sistemler ve enerji verimliliği gerektiren uygulamalar için uygundur. Ancak, işlem hattı derinliğinin kısa olması nedeniyle yüksek saat frekanslarında ölçeklenme yeteneği daha düşüktür. Bu çekirdek, düşük güç tüketimi gerektiren gömülü sistemler, IoT cihazları ve yüksek verimlilik odaklı uygulamalar için uygun bir çözümüdür.

Kullanılacak çekirdeğin detaylı bir dokümantasyona sahip olması sayesinde, test süreçlerinde ve kendi mikrodenetleyicimizde kullanımı için gerekli tüm teknik bilgilere kolayca erişim sağlayabilmekteyiz. Örneğin, bellek erişim pinlerinin davranışları ve gereksinimleri, kesme ve hata ayıklama arayüzü gibi daha karmaşık bağınlılara ilişkin detaylar, dökümantasyonda kapsamlı bir şekilde açıklanmıştır. Bu sayede, sistemin entegrasyonu ve hata ayıklama süreçleri daha verimli şekilde gerçekleştirilebilmektedir.

##### 3.1.2 Veri Yolu Yapısı:

Tasarımızda iki farklı veri yolu bulunmaktadır.

İlk veri yolu, sistem veri yolu, birden fazla denetleyici (master) ve birden fazla istemci (slave) desteği sunmaktadır. Çekirdek ve doğrudan bellek erişimi (DMA) birimleri, bu veri yolu üzerinden okuma ve yazma isteklerinde bulunabilir. Gelen istekleri karşılayan istemci birimler ise buyruk belleği, veri belleği, donanım hızlandırıcılar, kesme kontrolcüsü (PLIC) ve DMA birimi olacaktır.

İkinci veri yolu, çevre birimi veri yolu, sistem veri yolu ile çevre birimleri arasındaki iletişimini sağlayan ve çevre birimlerinin bağlılığı ikincil bir veri yoludur.

Bu iki ayrı veri yolu sayesinde, mikrodenetleyici tasarımımızı kontrol ve çevre birimi olmak üzere iki temel bileşene ayırmaktayız. Bu ayırım, sistemin yönetimini kolaylaştırırken, farklı bileşenlerin güç tüketimlerinin bağımsız olarak kontrol edilmesine de olanak tanımaktadır.

##### 3.1.3 Bellekler:

Mikrodenetleyici tasarımımızda veri belleği, buyruk belleği ve ön yükleyici belleği olmak üzere üç farklı bellek bulunacaktır.

Çekirdeğin buyruk getir arayüzünden (ing. instruction interface) gelen istekleri karşılamak üzere bulunan, buyruk belleği ve önyükleyici belleği ile bağlı bir bellek arayüzüümüz bulunmaktadır. Bu bellek arayüzü ile birlikte çekirdektenden gelen istekler adrese göre buyruk belleğine veya ön

yükleyiciye yönlendirilecektir. Aynı zamanda çekirdeğin veya DMA yapısının bulunduğu sistem veri yolundan gelen isteklerin de buyruk belleğine iletilmesi sağlanmaktadır.

Cekirdeğin üzerine çalıştığı verileri saklayabilmesi için sistem veri yolu ile bağlantılı veri belleği bulunacaktır. Veri belleği, kullanılacak açık kaynak çekirdeğin bellek arayüzü dokümantasyonuna göre tasarlanacaktır. Çekirdekten, veri belleğine gerçekleştirilecek işlemler için Sistem Veri Yolunun kullanılması yerine doğrudan çekirdek ve veri belleğinin bağlantısı sağlanacaktır. Çekirdeğin diğer birimler ile gerçekleştirdiği işlemler için Sistem Veri Yolu ile ilgili birimlere erişim sağlanacaktır. Ayrıca Sistem Veri Yolundan gelecek erişimler için veri belleği arayüzünde iki işlem arayüzü bulunacaktır. Bu tasarım sayesinde mikrodenetleyici çekirdeği ile veri belleği işlemleri doğrudan bağlantı ile hızlı gerçekleştirilirken, aynı zamanda DMA biriminden gelecek erişimler de veri belleği mürasitliğine göre veri belleğinde işlenebilecektir.

Veri belleği ve buyruk belleği simülasyon ve FPGA testlerinde 8 kb boyutunda BRAM'ler ile gerçeklenirken, çip tasarım akışında 8 kb boyutunda Static Random Access Memory (SRAM) ile gerçeklenecektir. Ön yükleyici ise içinde statik olarak bulunan programın boyutuna göre Read-only Memory (ROM) yapısında tasarılanacaktır.

#### **3.1.4 Çevre Birimleri:**

Tasarımımızda GPIO, Timer, UART, I2C, QSPI gibi mikrodenetleyicinin dışarı ile iletişimini sağlayacak çevre birimleri bulunmaktadır. Çevre birimleri, çevre birimleri veri yolunun istemci portlarına bağlı bulunacaktır, tasarım sürecinde belirlediğimiz adresler ile çekirdeğin okuma yazma istekleri için erişim dahilinde bulunacaktır.

##### **3.1.4.a GPIO:**

GPIO, tasarımımızda, 16 giriş, 16 çıkış olarak sabitlenmiş olacak şekilde, toplamda 32 pinli, AXI4 arayüzü üzerinden CPU ile “base address + offset” yöntemiyle kontrol edilecek; giriş verileri GPIO\_IDR, çıkış verileri ise GPIO\_ODR register'ları ile sağlanacak şekilde kullanılacaktır.

##### **3.1.4.b TIMER:**

Timer, tasarımımızda, prescaler, auto-reload, clear, enable, mod, sayaç ve event kayıtlarını içeren bir timer modülü olarak AXI4 arayüzüyle CPU'ya sunulacak; sayaç değeri belirlenen eşik değere ulaştığında resetlenip event üretecek şekilde gerçekleştirilecektir.

##### **3.1.4.c UART:**

UART tasarımımızda hem veri gönderimi hem de veri alımını gerçekleştirecek. UART\_CPB, UART\_STP, UART\_RDR, UART\_TDR ve UART\_CFG gibi yazmaçlar üzerinden kontrol edilecek; bu sayede, örneğin genel kullanım veya LDPC veri akışı için uygun konfigürasyon ile, seri haberleşme işlemleri AXI4 arayüzüyle sağlanacaktır.

##### **3.1.4.d I2C:**

I2C, tasarımımızda görevini, sabit SCL frekansı kullanarak, I2C\_NBY, I2C\_ADR, I2C\_RDR, I2C\_TDR ve I2C\_CFG gibi yazmaçlar aracılığıyla hedef slave cihazla veri alışverişi yapacak şekilde gerçekleştirilecektir. Bu modül, kısa mesafe haberleşme uygulamaları için sensörler veya diğer çevre birimleri ile iletişimini sağlayacaktır.

##### **3.1.4.e QSPI:**

QSPI, tasarımımızda, CPU'nun boot işlemi sırasında non-volatile flash bellekten bootloader kodunun okunması gibi işlemleri gerçekleştirmek üzere, belirlenen yazmaçlar ve kontrol sinyalleri aracılığıyla, AXI arayüzüyle entegre edilecektir. Böylece, sistem başlangıcında gerekli verilerin flash bellekten çekilmesi ve çalıştırılması sağlanacaktır.

### 3.1.5 Kesmeler:

Kesme (ing. interrupt) mekanizması, mikrodenetleyicilerde verimlilik, güç yönetimi, zamanlama ve gerçek zamanlı çalışma açısından kritik bir rol oynar. Kesmeler olmadan, işlemcinin sürekli olarak çevre birimlerini kontrol etmesi gerekeceğinden zaman kaybı, gereksiz güç tüketimi ve düşük sistem performansı gibi sorunlar ortaya çıkabilir. Bu nedenle, kesmeler mikrodenetleyicilerin esneklik, hız ve enerji verimliliği sağlama sırasında temel bir bileşen olarak kabul edilir.

Tasarımızda yer alacak Platform-Level Interrupt Controller (PLIC), çevre birimleri tarafından üretilen harici kesmelerin, çekirdek tarafından belirlenen öncelik seviyelerine göre sıralanmasını ve çekirdeğin irq\_i[11] Machine External Interrupt (MEI) portu aracılığıyla çekirdeğe iletilmesini sağlayacaktır.

Cekirdek, kesmeleri işlerken PLIC'in memory-mapped yazmaçları üzerinden okuma işlemleri gerçekleştirerek kesmenin kaynağı ve önceliği hakkında bilgi edinebilecektir. Bu yapı sayesinde, hızlandırıcının çalışma durumu, çevre birimlerinin durumları ve DMA tarafından yürütülen işlemler hakkında çekirdeğe etkin bir şekilde bildirim sağlanacaktır.

PLIC tasarımları için PULP Platform'un Github sayfasında da paylaştığı tasarımdan ilham alınmış ve yarışma sürecinde de bu tasarım kullanılacaktır [\[3\]](#).

### 3.1.6 Doğrudan Bellek Erişimi (Direct Memory Access (DMA)):

Doğrudan Bellek Erişimi (DMA - Direct Memory Access), mikrodenetleyicilerde çevre birimleri ile bellek arasında veri aktarımını işlemci müdahalesi olmadan gerçekleştiren bir mekanizmadır. Özellikle büyük veri bloklarının transferinde daha yüksek hız ve daha düşük güç tüketimi sağlanırken, işlemci gereksiz yere mesgul edilmemişinden sistem verimliliği artar.

UART, I2C, QSPI ve LDPC birimlerinden gelen istekler ayrı kanallar üzerinden çekirdeğin konfigüre ettiği öncelik sırasına göre seçilerek sistem veri yolu üzerinden veri belleğine veya buyruk belleğine bağlantısı sağlanacaktır.

## 3.2 Çekirdek Doğrulaması:

Cekirdeğin doğru bir şekilde implemente edildiğini anlamak ve işlevselliğini doğrulamak amacıyla çeşitli testler gerçekleştirilecektir. Bu doğrulama süreci, çekirdek dokümantasyonunda belirtilen yöntemlere uygun olarak yürütülecek olup, kaynak kodlarda bulunan test ortamı ve "cv32e40p\_rvfi\_trace.sv" modülü kullanılarak gerçekleştirilecektir. Bu testler sonucunda, programın her bir adımına ilişkin detaylı bilgileri içeren log dosyaları elde edilecek ve bu veriler, Spike Buyruk Simülatörü tarafından üretilen çıktılar ile karşılaşılacaktır. Karşılaştırma sürecini otomatikleştirmek amacıyla özel bir Python betiği geliştirilecek ve test kodları, riscv-software-src adlı GitHub hesabında bulunan "riscv-tests" deposundan temin edilecektir. Bu kapsamlı test süreci, çekirdeğin tasarım gereksinimlerine uygunluğunu değerlendirmek ve doğrulamak için kritik bir adım olacaktır [\[4\]](#).

## 3.3 Donanım Hızlandırıcı (LDPC (Low-Density Parity-Check Code)):

Low-Density Parity-Check (LDPC) kodları, hata düzeltme yeteneği yüksek olan ileri yönlü hata düzeltme kodlarıdır ve özellikle yüksek veri hızlarında güvenilir iletişim sağlamak amacıyla kullanılır. LDPC kodları, düşük yoğunluklu bir eşlik denetim matrisi ile tanımlanır ve Shannon sınırına yakın performans göstermesi sayesinde modern haberleşme sistemlerinde, özellikle 5G, Wi-Fi, uydu haberleşmesi ve depolama sistemlerinde yaygın olarak kullanılmaktadır. LDPC kodlayıcı, veriyi belirli bir kod oranına göre genişleterek fazladan hata düzeltme bitleri ekler ve böylece alıcı tarafta hataların düzeltilmesini mümkün kılar.

LDPC kodlayıcısının gerçeklenmesi için öncelikle kullanılacak eşlik denetim matrisi (H matrisi) belirlenir ve sistemin ihtiyacına göre düzenlenir. Genellikle, donanımda verimli bir uygulama elde

etmek için H matrisi, daha küçük boyutlu bir temel matrisin (ing. base matrix) genişletilmesiyle oluşturulur. Bu genişletme işlemi, özellikle IEEE 802.11n/5G gibi standartlarda kullanılan prototip matrisi teknikleriyle gerçekleştirilerek kodlama ve çözme işlemlerinde donanım açısından daha verimli bir yapı sağlanır. Sistematiğ bir LDPC kodlamada, veri vektörü ve oluşturucu matris ( $G$  matrisi) çarpılarak kodlanmış çıktı (ing. codeword) elde edilir. Kodlanmış veri ve kodlanmış çıktı ilgili bellek alanına yazılır.

### 3.3.a Donanım Hızlandırıcı Yazılım Modellemesi:

Geliştirilecek LDPC modülünün, belirlenen gereksinimleri doğruluk ve performans açısından karşıladığı doğrulamak için bilgisayar üzerinde Python tabanlı bir simülasyon gerçekleştirilecektir. Simülasyonda kullanılan test verileri ve elde edilen test çıktıları, Verilog tabanlı doğrulama süreçlerinde ve FPGA üzerindeki donanım testlerinde de yeniden kullanılabilecek şekilde tasarlanacaktır.

### 3.4 UVM Testleri:

UVM ya da Universal Verification Methodology, Verilog, VHDL, SystemC'de yazılmış bir tasarıma testbench yazmak için kullanılan bir metodolojidir. UVM SystemVerilog kullanılarak yazılır.

UVM Constrained Random Verification'ı destekler. Constrained Random Verification, rastgele vektörlerin simülasyonda kullanılmasıdır. Bu sayede beklenmedik bugların bulunması sağlanabilir, bununla beraber sinyal üretimi otomatikleştirilir, bu sayede uyarıcı yazması için kullanıcıya gerek kalmaz.

AXI4 testini yaparken çeşitli bileşenler kullanılacaktır. Bunlardan bazıları: AXI Master/Slave davranışını taklit edecek olan UVM Agent, Test edilen AXI arayüzüne sinyalleri taşıyan UVM Driver, Test senaryolarını yönetecek olan UVM Sequencer... bu bileşenler aracılığıyla testbench oluşturulacak, protokole uygun veri paketleri tanımlanacaktır. Daha sonrasında protokole uygun sinyaller üretilicek ve bu sinyaller doğrultusunda oluşan veri trafiği kaydedilecektir. En nihayetinde oluşturulan ortamdan beklenen sonuçlar ile asıl elde edilen sonuçlar karşılaştırılacaktır. Test sonuçları doğrultusunda fonksiyonel doğruluk, el sıkışma mekanizmasının doğruluğu, adreslemenin doğru yapılip yapılmadığı, edge case'ler, kritik yol analizi yapılabilir ve bu analiz sonuçlarına bağlı olarak arayüzün doğru çalışıp çalışmadığını dair fikir edinilebilir.

### 3.5 Mikrodenetleyici Yazılım Testleri:

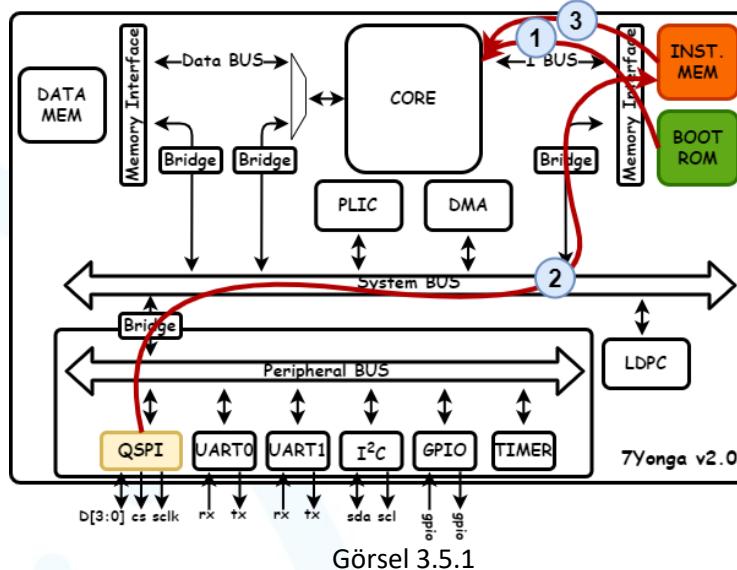
Geliştirdiğimiz mikrodenetleyicinin FPGA üzerinde test süreçlerini otomatikleştirmek ve standartlaştmak amacıyla kendi "Bare Metal" çalışma ortamımızı tasarlayacağız. Bu ortam, C dili ve RISC-V geliştirme araçları kullanılarak oluşturulacak ve betikler aracılığıyla test süreçlerinin otomatik olarak yürütülmesini sağlayan bir yapı içerecektir. Ayrıca, çevre birimlerinin ve hızlandırıcıların bellek adresleri, kontrol mekanizmaları ve UART tabanlı bir terminal arayüzü tasarlanarak, sistem bileşenlerinin etkin bir şekilde yönetilmesi sağlanacaktır. Bu sayede, mikrodenetleyicinin doğrulama süreci daha sistematik, güvenilir ve verimli bir hale getirilecektir.

#### 3.5.1 Önyükleme İşlemi:

Üç aşamadan oluşan ön yükleme işlemi, sistemin açılışı ile birlikte [görsel 3.5.1](#)'de gösterildiği ve aşağıda maddeler ile gösterildiği şekilde gerçekleşecektir.

- 1- Önyükleme işlemi çekirdeğin "boot" adresi olarak atanmış ön yükleyici kodunu ROM bellekten okuması ile başlayacaktır.
- 2- Ön Yükleyici kodu QSPI ve diğer çevre birimi araçlarını çalıştıracak ve mikrodenetleyiciye bağlı FLASH belleğinden okuma gerçekleştirecektir. Okunan veriler sırası ile buyruk belleğine yazılacaktır.

3- Okuma işleminin tamamlanması ile birlikte mikrodenetleyici çekirdeğinin, buyruk belleğine dallandırılması sağlanacaktır.



### 3.6 Çip Tasarım Akışı:

Çip tasarım akışı temel olarak iki aşamadan oluşur; Front End Tasarımı , Back End Tasarımı

#### 3.6.1 Front End Tasarımı

RTL tasarımcısı tasarım isterlerini kavrar ve bu isterleri karşılayan RTL kodunu yazar. Tasarımımızda hazır tasarımlar ve kendi yazdığımız tasarımlar System Verilog donanım tasarım dilinde olacaktır.

##### 3.6.1.2 Fonksiyonel Doğrulama

RTL kodu yazıldıktan sonra, yazılan kodun tasarım isterlerine uygun olarak çalışıp çalışmadığını kontrol etmek amacıyla fonksiyonalitesinin doğrulanması gerekmektedir. Bu işlem de RTL simülasyonu yapan bir araç ile yapılan simülasyonun sonuçlarının gözlenmesi ve fonksiyonalitede herhangi bir hata veya yazılan kodda herhangi bir bug ile karşılaşmadığından emin olunarak yapılır.

#### 3.6.2 Back End Tasarımı

##### 3.6.2.1 Mantık Sentezi

Eğer modülümüzün fonksiyonalitesi doğru ise, Mantık sentezi yapmak amacıyla sentez aracı aşamasına geçiş yaparız.

Sentez aşamasında, kodumuz bir “gate-level netlist”e dönüştürülür(davranışsal VHDL’den yapısal VHDL’e). Bu dönüşümü sağlayabilmek için bir “teknoloji kütüphanesine” bir de yazılan RTL koduna gereksinim duyuyoruz.

Gate-level netlist de yazılan RTL kod ile aynı fonksiyonalitede çalışması için içerisinde somutlaştırılmış standard cell’leri ve bu hücreler arası bağlantıları açıklayan bir VHDL dosyasıdır.

Teknoloji kütüphanesi içerisinde “standard cell” adı verilen hücreler barındırır. Bu hücreler mantık kapıları, mux’lar, macrolardır. Bu teknoloji kütüphanesi aracılığıyla verilen RTL kodumuz sağlanan standard cell’lere yerleştirilir ve sonrasında optimize edilir. Bununla birlikte **kütüphane** ile sağlanan gecikme bilgileri aracılığıyla Setup & Hold zamanlarının fark edilmesi ve RCB(Register-to-clock buffer) gecikmelerinin hesaplanması gibi gerekliliklerin yerine getirilmesini sağlar.

Gate-level netlist ile birlikte “.sdc” dosyası da elde edilir. Bu dosya dijital tasarım akışlarında tasarım sınırlamalarını belirlemekte sıkılıkla kullanılan dosya formatıdır. Bu sınırlamalar sentez ve implementasyon araçlarının tasarımını performans, alan ve güç olarak optimize etmesini sağlar.

### 3.6.2.2 Mantık Eşdeğerlik Kontrolü

Sentez sonucu oluşan gate-level netlist davranışsal özellikleri miras olarak alır ve bunun üzerine kütüphaneden elde ettiği standard cell'lerin kapı gecikme bilgilerini ekler. Bu gecikme bilgisi sayesinde sentez aracı kritik patikayı hesaplayabilir, dolayısıyla devredeki gecikme hakkında bir fikir sahibi olur.

Sentez sonrasında, sentez aracına verilen clock frekansları, I/O gecikmeleri, tasarım kuralları gibi sınırlamaların düzgün bir şekilde uygulandıklarından ve tasarım ile uyumlu olduklarından emin olmak için doğrulanırlar.

Örneğin, eğer kritik patika tarafından belirlenen clock periyodundan kısa bir clock periyodu kullanırsak, hata ile karşılaşırız ve Sentez aşamasını yinelememiz gereklidir.

### 3.6.2.3 Serim (PnR)

Serim aşaması tasarım akışındaki en önemli ve en karmaşık aşamadır.

Sentez sonrası elimizde net-list ve .sdc dosyaları bulunur. Bu aşamada Place and Route aşamalarına geçmeden önce Partitioning ve Floorplanning yapılır.

Partitioning kısmında eğer tasarım tekil değil ise, yani birbirine bağlı olan bloklar halinde ayrılabilirse ve bu ayırmının yapılması anlamlıysa, tasarım bloklara ayrılır ve yapılan ayrıştırma sayısı belirlenir. Buna örnek olarak bir mikroişlemcinin ALU, control unit ve memory unit olarak ayrıştırılması verilebilir.

Bu aşamadan sonra ise Floorplanning yapılır. Floorplanning'te ise hangi ASIC bölgesinin hangi bloğa ayrıldığı belirlenir. Partitioning yapılip yapılmaması ve buna bağlı olarak Floorplanning'in yapılması Serim aşamasını etkiler.

Serim(PnR) aşaması netlist ve .sdc dosyaları kullanılarak gerçekleştirilir. Bu dosyalar sayesinde inşa edilecek devrede kullanılacak standard cell'lerin sayısı, bu hücrelerin bağlantısı, gecikmeler ve tasarım kısıtlamaları hakkında bilgi sahibi olunur.

Placement aşamasında standard cell'ler ASIC içerisinde spesifik lokasyonlara yerleştirilir, sonrasında CTS(Clock Tree Synthesis) yapılarak clock dağılım ağı oluşturulur ki clock skew etkisi minimum düzeyde olsun ve timing bütün sequential elemanlarda tutarlı olsun. Son olarak, Routing'de de metal tabakalar aracılığıyla bu hücreler birbirlerine bağlanır.

Routing sonrası ilk LVS/DRC kontrolleri yapılabilir. Buradaki amaç, büyük tasarım hatalarının erken tespitiidir.

### 3.6.2.4 Signoff Aşaması

Routing sonrası tasarımın bütün zamanlama ve tasarım kuralı kontrollerinden geçmiş olması umulur ancak bu modern cihazlarda sağlanması zordur, dolayısıyla signoff aşamasına geçilir. Bu aşamada .sdc dosyasında belirtilen zamanlama, alan ve güç ile ilgili isterlerin sağlanıp sağlanmadığı kontrolü STA(Static Timing Analysis) ile yapılır. Bu işlem için Synopsys'in Primetime aracı kullanılabilir. Bütün kısıtlamalar sağlandığı zaman layout'un fizikal onaylanma aşamasına geçilir. Bu aşamada DRC, LVS Antenna Effect Check, ERC ve IR Drop gibi kontrollerin nihai değerlendirilmesi yapılır ve eğer herhangi bir hata yoksa tasarım GDSII formatına getirilir(hata durumunda Serim aşamasına geri dönülmesi gereklidir). Bu tasarımın son aşamasıdır ve GDSII dosyası fabrikaya üretim için yollanır.

## 4. Takım Organizasyonu ve İş Planı

7Yonga projesi için takımımız 3 kişiden oluşmaktadır. [Tablo 4.1](#)'de üyelerimizin öğrenim ve görevleri listelenmiştir.

[Görsel 4.1](#) proje takviminde, ekibin her bir üyesinin sorumluluklarını ve iş paketlerini, öngörülen sürelerle birlikte göstermektedir. Proje, Mart ayında planlama ve hazırlık süreçleriyle başlamış, Nisan ve Mayıs aylarında çevre birimleri tasarıımı ve test süreçlerine odaklanmıştır. Haziran ve Temmuz aylarında ise sistem entegrasyonu ve FPGA uygulamaları gerçekleştirilmiş, Ağustos ayında tasarımın nihai hale getirilmesi hedeflenmiştir. Takım üyeleri arasındaki görev paylaşımı belirlenmiş ve teslim tarihlerine uygun şekilde ilerleme planlanmıştır.

Üye	Öğrenim/Görev
<b>Muhammet Furkan UZUN</b>	Yeditepe Üniversitesi, Elektrik-Elektronik Mühendisliği 2. Sınıf öğrencisi. İş yükü: Çekirdeğin, belleklerin, veri yollarının implementasyonu, donanım hızlandırıcı
<b>Hasan GÜZELŞEMME</b>	Yeditepe Üniversitesi, Bilgisayar Mühendisliği 3. Sınıf öğrencisi. İş yükü: Çekirdek implementasyonu, çevre birimleri tasarımı, devre serimi
<b>Erhan ÖNALDI</b>	Yeditepe Üniversitesi, Bilgisayar Mühendisliği 4. Sınıf öğrencisi. İş yükü: Çevre Birimleri tasarımı, testler, donanım hızlandırıcı

Tablo 4.1 – Üyeler ve Görevleri



Görsel 4.1 – İş Planı

## 5. Kaynakça ve Ekler

- Traber, A., & Gautschi, M. (2017). PULPino: datasheet. *ETH Zurich, University of Bologna*, 36.
- Traber, A., Zaruba, F., Stucki, S., Pullini, A., Haugou, G., Flamand, E., ... & Benini, L. (2016, January). PULPino: A small single-core RISC-V SoC. In *3rd RISCV Workshop* (p. 15).
- pulp-platform. (2007). riscv-tests GitHub. Retrieved March 13, 2025, from [https://github.com/pulp-platform/rv\\_plic](https://github.com/pulp-platform/rv_plic)
- riscv-software-src. (?). RISC-V Platform-Level Interrupt Controller GitHub. Retrieved March 13, 2025, from <https://github.com/riscv-software-src/riscv-tests>