ÇİP TASARIM YARIŞMASI

MİKRODENETLEYİCİ TASARIM KATEGORİSİ

ÖN TASARIM RAPORU

Takım Adı: Yeditepe Üniversitesi Sayısal Tasarım Topluluğu

TAKIM ID: 582202

Başvuru ID: 3062081

İçindekiler

Kısaltmalar:

ADR: Address Register
ARE: Auto-Reload Register
AXI4: Advanced eXtensible Interface 4
BRAM : Block Memory
CCR: Communication Configuration Register
CFG: Configuration Register
CNT: Counter Register
CPB: Clock-Per-Bit
DMA: Direct memory access
DOR: Dual Output Read
DR: Data Register
DTR: Detaylı Tasarım Raporu
EVC: Event Clear Register
EVN: Event Register
FPGA: Field-Programmable Gate Array
GPIO: General Purpose Input/Output
GUI: Graphical User Interface
HDL: Hardware Description Language
I2C: Inter-Integrated Circuit
IDR: Input Data Register
LDPC: Low Density Parity Check
ODR: Output Data Register
ÖTR: Ön Tasarım Raporu
PLIC: Platform-Level Interrupt Controller
PP: Page Program
PRE: Prescaler Register
QOR: Quad Output Read
QPP: Quad Page Program
QSPI: Quad Serial Peripheral Interface
RDR: Read Data Register
RDSR1: Read Status Register 1
ROM: Read-Only Memory
RX: Receive
SE: Sector Erase
SRAM: Static Random Access Memory
STA: Status Register
STP: Stop-bit
struct: Structure (C dilinde veri yapısı)
TCL: Tool Command Language
TDR: Transmit Data Register
TX: Transmit
WEL: Write Enable Latch
WRDI: Write Disable
WREN: Write Enable

1. Sistem Tanımı ve Temel Tasarım Özeti

Takım Tanıtımı ve Proje Hedefi

Takımımız, Yeditepe Üniversitesi Elektrik-Elektronik ve Bilgisayar Mühendisliği lisans öğrencilerinden oluşmaktadır. 7Yonga projesi kapsamında, açık kaynaklı bir RISC-V işlemci çekirdeği, çevre birimleri ve veri yolları kullanılarak özgün bir mikrodenetleyici tasarımı hedeflenmektedir.

Tasarım Bileşenleri ve Entegrasyon Süreci

Tasarımımız, PULP Platformu tarafından sağlanan AXI4 (Advanced eXtensible Interface 4) veri yolu kitaplığı ve OpenHW Group tarafından geliştirilen RV32E40P işlemci çekirdeği temel alınarak gerçekleştirilmektedir. Geliştirilen çevre birimleri, bellek bileşenleri (veri ve komut belleği) ile LDPC modülü, AXI4 üzerinden işlemci çekirdeği ile bütünleştirilerek mikrodenetleyici sistemi oluşturulmaktadır.

Geliştirme ve Test Süreci

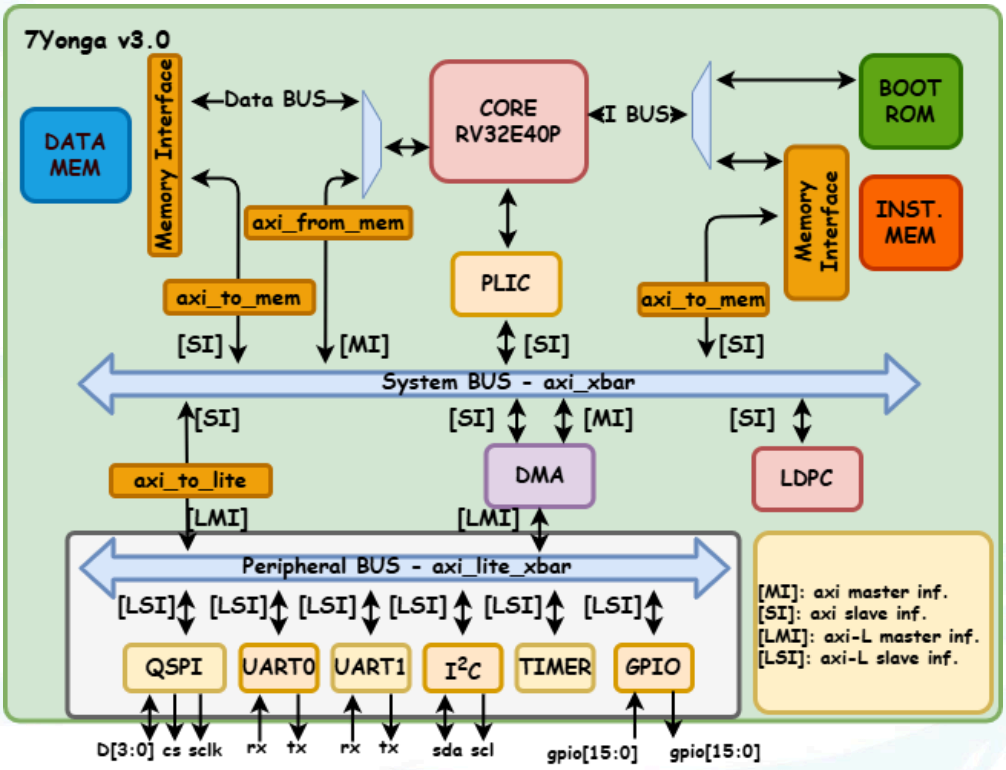
Ön Tasarım Raporu (ÖTR) sonrasında AXI4 ve RV32E40P kod depoları incelenmiş ve temel testleri tamamlanmıştır. Bellek yapıları ile UART çevre biriminin tasarımı başarıyla gerçekleştirilmiştir. Ayrıca, işlemcinin komut ve veri arayüzleri için gerekli kontrol birimleri tasarlanmış ve test edilmiştir.

Donanım Test Raporu (DTR) öncesinde FPGA üzerinde tüm sistemin test edilmesi planlansa da, sistemin tamamının bir araya getirilmesi sürecinde Vivado ile simülasyon aşamasında çeşitli hatalarla karşılaşılmıştır. Bu nedenle hâlihazırda tasarımın tüm bileşenleri birleştirilmiş olup, doğrulama testleri sürmektedir. Testler, DMA ve PLIC haricinde yazılması gereken tüm içerikler yazılmıştır. Kullanılacak HDL kod depoları kullanım için araştırdık ve projemize entegre etme işlemlerini gerçekleştirdik, örneğin: RV32E40P, AXI4, pulp common cell ve Coremark. DTR süreci sonrasında test ortamı (testbench) üzerinden yapılan doğrulama işlemleri tamamlandığında, FPGA üzerinde nihai sistem testleri gerçekleştirilecektir.

2. Proje Detay Tasarımı

2.1. Sistem Mimarisi

Görsel 2.1.1'de sistemin üs sistem tasarımı gösterilmiştir. Bu tasarım ile birlikte; yarışma tarafından istenen isterler aşağıda listelendiği şekilde sağlanmıştır.



Görsel 2.1.1 - Üst Sistemin Blok Gösterimi

2.1.1 Mikrodenetleyici Çekirdeği:

Tasarımımızda mikrodenetleyici çekirdeği olarak isterlerde bulunan OpenHW Group CORE-V CV32E40P RISC-V IP'si kullanılacaktır.

CV32E40P, 4 aşamalı işlem hattına sahip, küçük ve verimli bir 32-bit in-order RISC-V çekirdeğidir. RV32IM[F|Zfinx]C komut setini destekler ve PULP özel uzantıları sayesinde performans ile enerji verimliliğini artırır. Kompakt yapısı ve düşük güç tüketimi ile gömülü sistemler ve IoT uygulamaları için uygundur. Ancak, işlem hattının sığ olması nedeniyle yüksek saat frekanslarında sınırlı ölçeklenebilirliğe sahiptir.[1]

2.1.2 Veri Yolları:

Tüm üst sistem içerisinde modüller arasında veri alışverişini sağlayan veri yolu yapıları için PULP platformunun açık kaynaklı **AXI4 (Advanced eXtensible Interface 4)** deposu tercih edilmiştir. Bu kaynak

ile birlikte **AXI4** ve **AXI4-Lite** standartları için çok sayıda dönüştürücü modül, zengin arabirim (interface) tanımları ve bağlantı (interconnect) yapıları sağlanmaktadır.

Çekirdekten çıkan veri ve buyruk (instruction) sinyalleri, tasarladığımız bellek ve çevresel birim kontrolcülerine AXI4 modülleri aracılığıyla bağlanmaktadır. Bu noktada, AXI4 deposundan alınan adaptör ve bağlantı yapıları, tarafımızca geliştirilen seçim ve yönlendirme modülleri ile entegre edilerek, çekirdeğin veri belleği ile doğrudan bağlantısı sağlanmakta; çevresel birimlerle olan bağlantı ise AXI4 standardına uygun **crossbar** yapısı üzerinden gerçekleştirilmektedir. Aynı şekilde, buyruk getirme (instruction fetch) işlemi sırasında Boot ROM ve buyruk belleği (instruction memory) arasında yönlendirme yapılabilmesi amacıyla yine AXI4 deposunda yer alan modülleri içeren, tarafımızca geliştirilen alt birimler kullanılmaktadır.

2.1.3 Bellekler:

Proje şartnamesinde belirtildiği üzere, işlemcinin çalışması için iki temel bellek yapısı gereklidir: Genel amaçlı verilerin saklandığı **veri belleği (data memory)** ve çalıştırılacak programın bulunduğu **buyruk belleği (instruction memory)**. Simülasyon ortamında bu bellekler basit yazılım modelleri ile temsil edilirken, FPGA testlerinde **Block RAM (BRAM)** kaynakları, yongaya gömülü testlerde (tape-out aşaması) ise **SRAM blokları** kullanılacaktır. Ayrıca işlemcinin başlangıçta çalıştıracağı sabit kodları içeren **statik buyruk belleği**, simülasyonlarda bir ROM modeli ile temsil edilirken, FPGA testlerinde ve çip üretimi aşamalarında özel olarak sentezlenmiş donanım blokları kullanılacaktır. Buyruk belleği, gelen istekleri ilgili bellek hücrelerine yönlendirerek işlemcinin doğru kodlara erişmesini sağlamaktadır.

Kullanılan **RV32E40P** çekirdeği, bellek erişimlerinde hizasız erişimi donanımsal olarak desteklememektedir. Bu nedenle, buyruk belleğinde olduğu gibi, veri belleği kontrolünde de hizalı erişimi garanti eden benzer bellek kontrolcüleri kullanılmıştır. Bu kontrolcüler, çekirdeğin yalnızca hizalanmış (aligned) erişimlerde çalışacağı varsayımıyla tasarlanmış ve hizasız (unaligned) erişimlerde hata üretmeyecek şekilde yapılandırılmıştır. [2]

2.1.4 Çevre Birimleri:

Çevre birimleri daha anlaşılır bir tasarım için top modülün altında “soc_peripherals_top.sv” dosyasında listelenmiştir. Aynı zamanda dosya içeriğinde top modülden gelen AXI4 Master sinyalini AXI4-Lite Master sinyaline dönüştüren ve bu sinyali AXI4 crossbar ile çevre birimlerine dağıtan veri yolu bulunmaktadır. Bu sayede çekirdekten veya DMA’dan gelen isteklerin çevre birimlerine iletilmesi ve modüller arası arası haberleşmeyi mümkün kılar.

2.1.5 Kontrol ve Yardımcı Donanım Birimleri:

Doğrudan Bellek Erişimi (Direct Memory Access, *DMA*), mikrodenetleyicilerde çevre birimleri ile bellek arasında veri aktarımını işlemcinin doğrudan müdahalesi olmadan gerçekleştiren bir yöntemdir. Özellikle büyük veri bloklarının aktarımında, daha yüksek hız ve daha düşük güç tüketimi sağlanarak sistem

verimliliği artırılmaktadır. Ayrıca, işlemci bu aktarım sürecinde meşgul edilmediğinden, diğer görevleri kesintisiz bir şekilde yürütebilir.

Sistem içerisinde **UART**, **I2C**, **QSPI** ve **LDPC** birimlerinden gelen veri aktarım istekleri, DMA yapısında tanımlı ayrı kanallar üzerinden çekirdek tarafından konfigüre edilmiş öncelik sırasına göre seçilecektir. Bu istekler, sistem veri yolu aracılığıyla veri belleğine veya buyruk belleğine yönlendirilerek aktarım gerçekleştirilecektir.

DMA Modülünün Donanımda Uygulanması

Doğrudan Bellek Erişimi (DMA – Direct Memory Access) modülünün donanım üzerinde uygulanabilirliğini değerlendirmek amacıyla **STM32** geliştirme kartları dikkate alınmıştır. Bu kapsamda, DMA veri akışının genel işleyişi aşağıda adım adım açıklanmıştır:

1. Uygulama, sistem veri yolu (bus) üzerinden DMA modülüne erişerek ilgili DMA kanallarını yapılandırır. Bu yapılandırma sırasında örneğin kanalın önceliği gibi parametreler belirlenir.
2. Uygulama, işlem yapacak olan çevre birimini DMA ile çalışacak şekilde yapılandırır ve ardından bu çevre birimini çalıştırır.
3. Çevre birimi, çalışma esnasında belirlenen yapılandırma kurallarına göre DMA modülüne istek (request) ileterek veri aktarımı başlatılması için sinyal gönderir.
4. DMA modülü, aldığı istekleri öncelik sırasına göre değerlendirir ve doğrudan çevre birimiyle olan çevre birimi veri yolu bağlantısı üzerinden ilgili veri aktarım işlemini başlatır.
5. DMA'nın diğer bağlantısı, sistem veri yoluna bağlıdır. Bu sayede, veri aktarımının diğer ucu olan ana bellek ya da LDPC modülü gibi bileşenlere **master arayüz** üzerinden erişim sağlar.
6. Veri aktarımı sırasında bir hata oluşması, işlemin tamamlanması ya da yalnızca bir kısmının bitirilmesi gibi durumlar için, DMA yapılandırma kurallarına göre kesmeler (interrupt) üretir. Üretilen bu kesmeler, **Platform Düzeyi Kesme Denetleyicisi** (PLIC – Platform-Level Interrupt Controller) üzerinden sisteme bildirilir.

Tasarımımızda yer alan **Platform Düzeyinde Kesme Denetleyicisi** (Platform-Level Interrupt Controller, *PLIC*), çevre birimleri tarafından üretilen harici kesmeleri, çekirdek tarafından belirlenmiş öncelik seviyelerine göre sıralayarak yönetir. Bu kesmeler, çekirdeğe `irq_i[11]` portu üzerinden **Makine Düzeyi Harici Kesme** (Machine External Interrupt, *MEI*) olarak iletilir.

Çekirdek, kesmeleri işlerken PLIC'in **bellek adreslenebilir** (*memory-mapped*) yazmaçları üzerinden okuma işlemleri gerçekleştirir. Bu sayede, kesmenin kaynağı ve önceliği hakkında bilgi edinilebilir. Böylece hızlandırıcının mevcut durumu, çevre birimlerinin çalışmaları ve **Doğrudan Bellek Erişimi** (Direct Memory Access, *DMA*) yoluyla yürütülen işlemler hakkında çekirdeğe etkili bir şekilde bildirim sağlanmış olur.

PLIC tasarımı, PULP Platformu'nun GitHub sayfasında paylaşılan açık kaynaklı mimariden esinlenilerek oluşturulmuştur. Yarışma sürecinde de bu tasarım doğrudan kullanılacaktır.

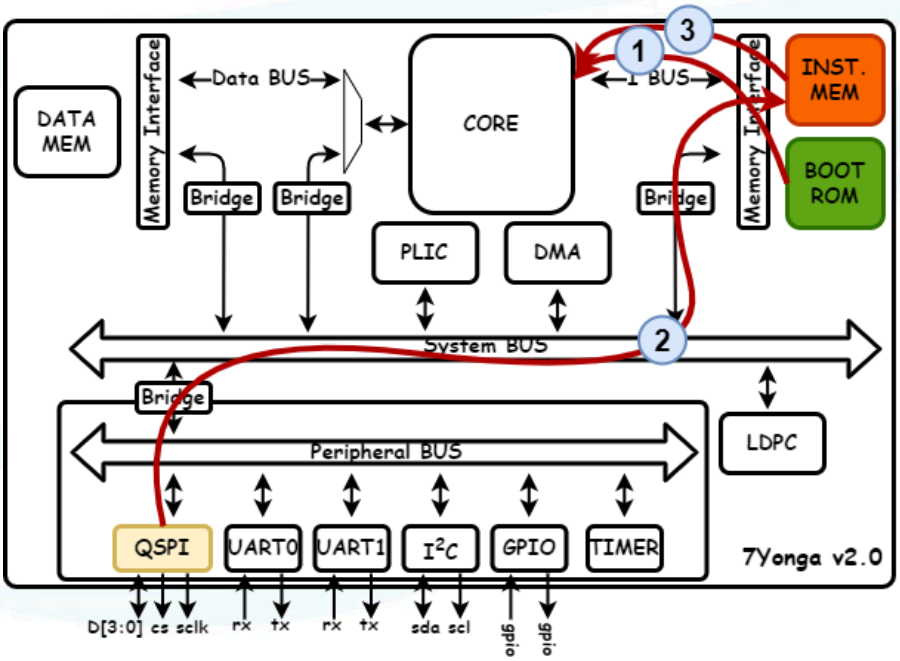
Detaylı Tasarım Raporu (DTR) süreci boyunca kesme (interrupt) ve DMA **Doğrudan Bellek Erişimi** mekanizmalarının entegrasyonu tarafımızca planlanmamıştır. Ancak, DTR aşamasının ardından, tasarımın daha test edilebilir hale gelmesi ve büyük ölçüde tamamlanması ile birlikte kesme yapısının ve DMA'nın sisteme entegre edilmesi planlanmaktadır.

2.1.6 Mikrodenetleyici Boot Süreci:

Boot süreci mikrodenetleyicinin başlatıldığında önce boot ROM'dan çalışmasının ardından Qspi modülü ile birlikte harici flash belleğinden programın okunması ve buyruk belleğine yazılması sürecidir.

Üç aşamadan oluşan ön yükleme süreci, sistemin açılmasıyla birlikte Görsel 2.1.6'de gösterildiği ve aşağıda madde madde açıklanan adımlar doğrultusunda gerçekleşecektir:

1. Önyükleme işlemi, çekirdeğin *boot* (başlangıç) adresi olarak tanımlanmış konumdan, **salt okunur bellek** (Read-Only Memory, *ROM*) içerisinde yer alan ön yükleyici kodunu okumasıyla başlayacaktır.
2. Ön yükleyici kod, **QSPI** ve diğer çevre birimi denetleyicilerini çalıştıracak, ardından mikrodenetleyiciye bağlı olan **FLASH** bellekten okuma işlemi gerçekleştirilecektir. Elde edilen veriler, belirlenen sıra ile **buyruk belleğine** (instruction memory) aktarılacaktır.
3. Okuma işlemi tamamlandığında, mikrodenetleyici çekirdeği, doğrudan buyruk belleğinde yer alan başlatma koduna yönlendirilerek (*branch*) asıl programın yürütülmesine başlanacaktır.



Görsel 2.1.6 - Boot Süreci Gösterimi

2.1.7 Adres Haritası ve Linker File:

Donanım ve yazılım bileşenlerinin uyumlu çalışabilmesi için adres haritaları, her iki tarafın da üzerinde uzlaştığı ortak bir yapıya dayanmaktadır. Bu kapsamda, hem donanım tanımlama dili (Verilog-HDL) ile adres tanımlamaları oluşturulmuş hem de yazılım tarafında bağlayıcı (linker) betikleri ve çevre birimi sürücüleri (peripheral drivers) gibi modüller içerisinde aynı adres tanımlarını içeren eşdeğer kodlar yazılmıştır.

Donanım tarafında, adres tanımlamaları ./7Yonga/gateware/inc dizini altında yer alan soc_addr_rules.svh dosyasında tanımlanmıştır. Bu dosya, sistemde yer alan bileşenlerin donanım seviyesindeki adres yapılandırmalarını içermektedir.

Aşağıdaki örnekte görülebileceği üzere, önce belirli bir adres değeri oluşturulmuş, ardından bu adresin ait olduğu aralık tanımlanarak bir adres kuralı yazılmıştır. Bu adres kuralları, tüm donanım modüllerinde ortak biçimde kullanılmış ve böylece HDL tasarımında daha sade, anlaşılır ve sürdürülebilir bir yapı elde edilmiştir.

```
typedef logic [31:0] addr_t;
typedef struct packed {
    addr_t       start_addr;
    addr_t       end_addr;
} addr_rule_t;
localparam addr_rule_t ROM_ADDR_RULE = '{
    32'h3000_0000,
    32'h3000_0400
};
```

Linker, derleme sürecinin son aşamasında çalışan ve birden fazla nesne dosyasını (object files) birleştirerek çalıştırılabilir bir dosya oluşturan araçtır. Özellikle gömülü sistemlerde, programın hangi belleğe yerleştirileceğini belirlemek için bellek haritalarının oluşturulmasında kritik rol oynar. Linker betikleri (linker script), .text, .data gibi farklı bellek bölümlerinin fiziksel adresler üzerindeki yerleşimini kontrol etmeye yarar. Bu sayede yazılım ve donanım bileşenleri arasında doğru bir bellek eşleşmesi sağlanır.

Aşağıda verilen örnek linker betiği, gömülü bir sistemde kodun RAM bölgesine yerleştirilmesi amacıyla kullanılmaktadır. _start etiketi, programın başlangıç adresini tanımlar. MEMORY bloğu ile sistemdeki kullanılabilir fiziksel bellek alanları tanımlanır. Bu örnekte ram bölgesi 0x30000000 adresinden başlamakta ve 1 KB uzunluğundadır.

SECTIONS bloğunda ise derlenen programın bölümleri (.text, .data) ilgili fiziksel adres bölgelerine yerleştirilir. .text bölümü genellikle çalıştırılabilir kodları içerirken, .data bölümü başlatılmış küresel değişkenleri barındırır. ALIGN(4) ifadesiyle 4 bayt hizalama sağlanmış ve _end etiketiyle bellek yerleşimi sonlandırılmıştır.

```
ENTRY(_start)
MEMORY
{
  /* Define the memory regions */
  ram (rwx) : ORIGIN = 0x30000000, LENGTH = 1K
}

SECTIONS
{
  /* Define the sections of the output file */
  .text : {
    *(.text)
  } > ram

  .data : {
    *(.data)
  } > ram

  . = ALIGN(4);
  _end = . ;
}
```

2.2. Tasarım Detayları

2.2.1. İşlemci ve Bus Yapısının Tasarımı

2.2.1.1 Çekirdek Entegrasyonu:

"cv32e40p_top" modülü, işlemcinin en üst (top) düzey modülüdür. Bu modül, mikrodenetleyici sisteminin top modülünde çağrılarak sisteme entegre edilir [3].

"FPU", "FPU_ADDMUL_LAT", "FPU_OTHERS_LAT", "ZFINX", "COREV_PULP", "COREV_CLUSTER" ve "NUM_MHPMCOUNTERS" gibi parametre (ing. parameter) değerleri, erişimi kolaylaştırmak amacıyla "inc" klasörü altında bulunan "soc_configuration.svh" adlı dosyada tanımlanmıştır. Bu parametreler, mümkün olan en az çekirdek özelliğini içerecek şekilde yapılandırılmıştır. Bu tercihle, performans sayaçları (NUM_MHPMCOUNTERS) dışındaki tüm parametreler sıfır değeriyle kullanım dışı (ing. disable) bırakılmıştır. Performans sayaçları parametresi ise dokümantasyon belirtilen varsayılan "1" değeriyle yapılandırılmıştır.

Saat (clock), reset ve kontrol pinleri olan "rst_ni", "clk_i", "scan_cg_en_i", "fetch_enable_i", "pulp_clock_en_i" ve "core_sleep_o" sinyalleri, dokümantasyona uygun biçimde standart bağlantılar ile entegre edilmiştir. Kontrol sinyalleri varsayılan olarak kullanım dışı bırakılmıştır.

Çekirdeğe ait yapılandırma pinleri, statik değerlere sahip oldukları için parametreler gibi "soc_configuration.svh" dosyasında tanımlanmış ve kolay erişilebilir biçimde düzenlenmiştir. RISC-V standartları gereği her işlemci tasarımında en az bir adet 0 numaralı hart kimliğine (ing. hardware

thread) sahip bir hart bulunmalıdır. Bu nedenle “hart_id_i” pini için sıfır değeri atanmıştır. Çekirdeğin buyruk alma (fetch) işlemini başlatacağı adresi belirten “boot_addr_i” pini, sabit buyrukları içeren ROM adresine bağlanmıştır. Exception ve trap gibi özel durumlar için başlangıç adresi olarak yine boot adresi kullanılmıştır; bu yapı ilk testler için yeterli bulunmuştur.

Çekirdeğin buyruk ve veri arayüzleri, geliştirilen buyruk ve veri arayüz kontrolcülerine bağlanmıştır. Bu arayüzler hakkında detaylı açıklamalar aşağıdaki başlıklarda verilmektedir.

Debug ve kesme (interrupt) arayüzleri, ilk aşamada kullanım dışı kalacak şekilde sabit sıfır değerine bağlanmıştır. İlerleyen aşamalarda debug ve kesme desteği eklendiğinde, interrupt arayüzü Harici Kesme Denetleyicisi (PLIC) ile ilişkilendirilecektir. PLIC üzerinden gelen kesme sinyalleri, bu arayüz üzerinden çekirdeğe iletilerek kesmelerin işlenmesi (ing. Interrupt handling) sağlanacaktır. Benzer şekilde, debug kontrol sinyalleri geliştirilecek debug modülü ile entegre edilecektir.

2.2.1.2 Veri Yolları Tasarımları ve Tercihleri

Tasarımımızda iki adet veri yolu bulunmaktadır.

Bunlardan ilki çekirdek, harici diğer modüller ve ikinci veri yoluna bağlantıyı sağlayan sistem veri yoludur. tasarımı için axi4 ve pulp platform reposunda yer alan axi_xbar modülü tercih edilmiştir. Bu yapı sayesinde DMA ve çekirdek gibi iki master sistem daha hızlı bir şekilde kendi isteklerini gerçekleştirilmektedir.

Ayrıca sistem veri yolundan çevre birimlerine gidecek isteklerin ayrıştırılıp ilgili birimlere iletilmesi için ikinci bir veri yolu olan çevre birimleri veri yolu tasarıma eklenmiştir. Pulp platform’un axi_lite_xbar modülü ile gerçeklenen bu veri yolu ile sistem veri yoluna karşın daha düşük güç tüketimi önceliklendirilmiştir.

2.2.1.3 Çekirdek Buyruk ve Veri Arayüzleri

Hazır olarak kullanılan RV32E40P çekirdeğinde implementation aşamasında buyruk ve bellek olmak üzere iki arayüz bulunmaktadır. Buyruk arayüzü ile çekirdek fetch aşamasında buyruk alırken, veri arayüzü ile lw ve sw gibi buyruklar ile bellek işlemlerinin gerçekleştirir.

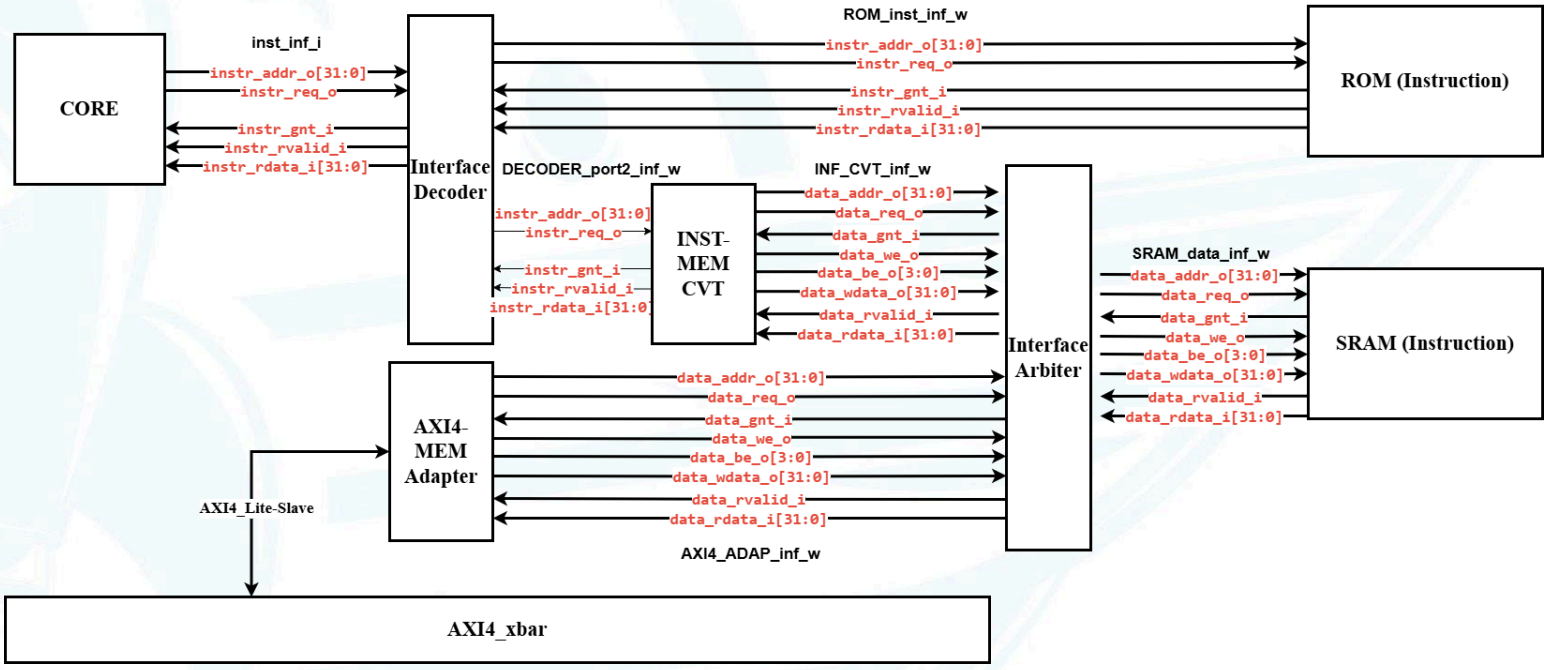
Öncelikli olarak bu arayüzlerin doğrudan sistem veri yoluna bağlanmasını düşündük. Bu yöntem ile çekirdekten çıkan buyruk ve veri arayüz sinyalleri hazır adaptörler ile doğrudan sistem veri yoluna bağlanması planlanmıştı. Bu sayede tasarım dah sade ve güç tüketimi yönünden verimli olacaktı. Lakin fetch ve veri belleği işlemleri için performansın oldukça önemli bir özellik olması ve bu yöntem ile isteklerin geç cevaplanacağı endişesi bizi daha farklı çözümlere yönelendirmiştir. Örnek projeler ve yayınlarda da yaptığımız araştırmalar ile Pulpino projesinin de tercih ettiği tasarım bizim için alan/güç ve performans anlamında daha uygun geldi. Bu yapı ile birlikte çekirdek ile buyruk ve veri bellekleri arasında ayrı iletişim bağlantılarının olması, çekirdek ve diğer birimler arasında sistem veri yolu bulunması planlanmıştır.

Çekirdek Buyruk Arayüzü

Bu bölüm, işlemci çekirdeğinin belirli bir bellek adresinden buyruk (ing. instruction) okumasını sağlayan yapıdır. İşlem, çekirdeğin buyruk arayüzü (ing. instruction interface) üzerinden gönderdiği sinyaller ile başlar. Bu sinyaller şunlardır:

- instr_addr_o: Buyruk adres çıkışı (instruction address output)
- instr_req_o: Buyruk isteği çıkışı (instruction request output)
- instr_gnt_i: Buyruk izni girişi (instruction grant input)
- instr_rvalid_i: Buyruk verisinin geçerli olduğunu belirten giriş (instruction read valid input)
- instr_rdata_i: Buyruk veri girişi (instruction read data input)

Görsel 2.2.1.3.a'da, çekirdekten çıkan buyruk getirme (instruction fetch) sinyallerinin ve sistem veri yolundan (system bus) gelen bellek işlemlerinin, belleğe yönlendirilmeden önce nasıl seçildiği ve ilgili modüllere nasıl yönlendirildiği ayrıntılı şekilde gösterilmiştir.



Görsel 2.2.1.3.a - Mikrodenetleyici Buyruk Arayüzü Tasarımı

İşlemci çekirdeği tarafından üretilen adres, bir adres çözücü (decoder) birimi aracılığıyla ROM ya da buyruk belleği (ing. instruction memory) olarak ayrıştırılır. Bu işlem, PULP platformunun ortak hücre (common cell) deposunda yer alan addr_decode modülü kullanılarak gerçekleştirilmiştir. Böylece adreslerin, tüm tasarımda yer alan bir yapı (struct) formatına uygunluğu korunmuş olur.

Adres çözücünün ürettiği sinyal paketi, doğrudan ROM'a bağlanarak bir buyruk okuma işlemini başlatabilir. Ancak adresin, buyruk verisine değil de genel buyruk belleğine (instruction memory)

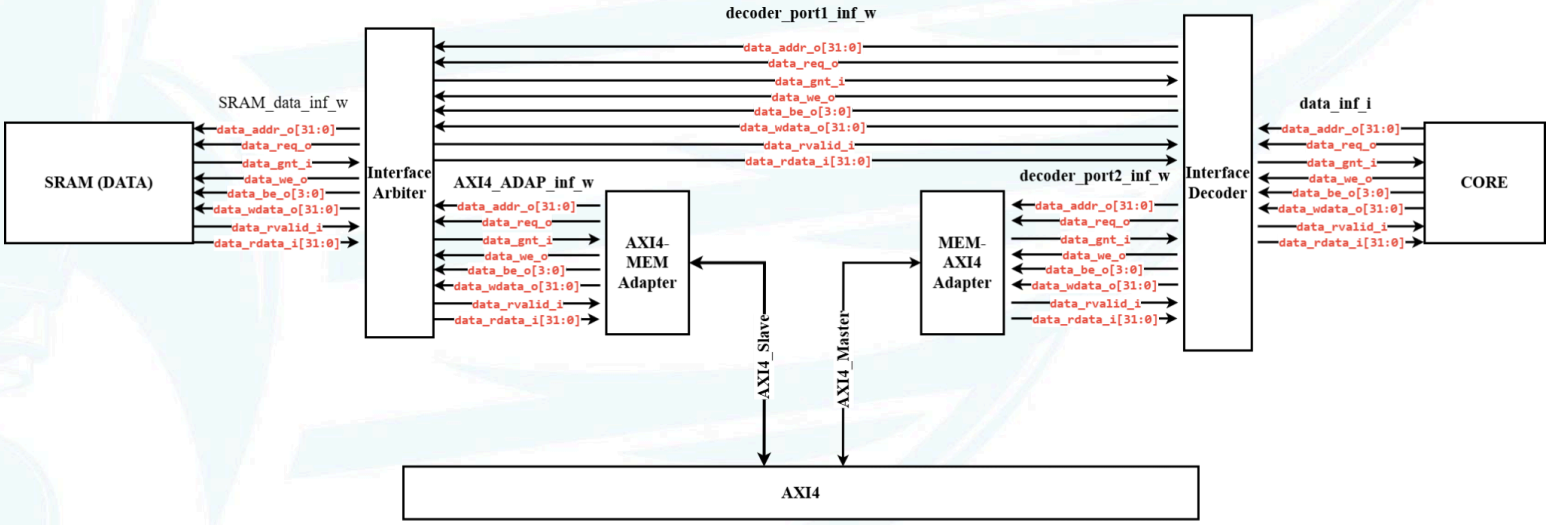
yönlendirilmesi durumunda, sinyal paketi genişletilerek buyruk arayüzünden veri arayüzüne dönüştürülür.

Bu dönüştürme işlemi sonrası, başka bir veri yazma isteği sistem veri yolu üzerinden axi4_xbar modülü aracılığıyla gelebilmektedir. Bu buyruk arayüz birimi, okuma işlemleri ile birlikte yazma isteklerini de aynı anda alabildiği için, kaynaklar arasında rekabet (contention) oluşabilir.

Bu rekabetin adil biçimde yönetilmesi amacıyla Round-Robin algoritması ile çalışan bir Seçici (Arbiter) modülü tasarlanmıştır. Bu Seçici, aynı anda gelen istekler arasında sırayla seçim yaparak hangi isteğin belleğe iletileceğine karar verir. Seçilen sinyal paketi doğrudan buyruk belleğine yönlendirilir. İletişim sürecince mikrodenetleyici çekirdeği instr_req_o sinyalini açık tutarak instr_rdata_i ve instr_gnt_i sinyali el sıkışmasını bekler bu süre içerisinde bu modülde iletişim açık tutulur.

Çekirdek Veri Arayüzü

İşlemci çekirdeğinden gelen veri okuma/yazma (read/write) istekleri, buyruk arayüzünde (instruction interface) olduğu gibi bir adres çözücü (address decoder) ile ayrıştırılır. Bu sayede gelen isteklerin hangi bellek alanına (örneğin veri belleği veya çevresel birim) yönlendirileceği belirlenir. Görsel 2.2.1.3.b'de ise, tasarımımızda çekirdekten ve sistem veri yolundan gelen bellek işlemlerinin veri arayüzü denetleyicisi (data interface controller) içerisinde nasıl işlendiği, kontrol mantığının hangi yapılarla gerçekleştirildiği ve bellek erişim sıralamasının nasıl belirlendiği açıklanmıştır.



Görsel 2.2.1.3.b - Mikrodenetleyici Veri Arayüzü Tasarımı

Çekirdek Veri erişiminde kullandığı sinyaller şunlardır:

- data_addr_o[31:0] (**Çıkış / Output**): Belleğe erişilecek adresi belirten sinyaldir (memory address).
- data_req_o (**Çıkış / Output**): Geçerli bir erişim isteği olduğunu belirtir (request valid). Bu sinyal, data_gnt_i bir çevrimliğine (cycle) yüksek olana kadar yüksek kalır.

- data_gnt_i (**Giriş / Input**): Bellek arayüzü isteği kabul ettiğinde bir çevrimliğine yüksek olur (grant). Bu sinyal alındıktan sonra data_addr_o bir sonraki döngüde değişebilir.
- data_we_o (**Çıkış / Output**): Yazma etkinleştirme sinyali (write enable). Yüksek (high) olduğunda yazma, düşük (low) olduğunda okuma işlemi yapılır. data_req_o ile birlikte gönderilir.
- data_be_o[3:0] (**Çıkış / Output**): Bayt etkinleştirme sinyali (byte enable). Okunacak veya yazılacak baytları belirtir. data_req_o ile birlikte gönderilir.
- data_wdata_o[31:0] (**Çıkış / Output**): Belleğe yazılacak veriyi içerir. data_req_o ile eşzamanlı gönderilir.
- data_rvalid_i (**Giriş / Input**): Bellek erişim yanıtının tamamlandığını belirtir (response valid). Okuma ve yazma işlemleri için bu sinyal tam olarak bir çevrim süresince yüksek olur. Okuma işlemlerinde bu sinyal yüksekken data_rdata_i geçerli veri taşır.
- data_rdata_i[31:0] (**Giriş / Input**): Bellekten okunan geçerli veridir (read data).

Veri belleğine yapılacak erişimler, sistem veri yolu (system data bus) olan axi4_xbar aracılığıyla gerçekleştirilir. Bu noktada, çekirdekten gelen bellek erişim sinyalleri, PULP platformunun axi4 deposunda bulunan axi_from_mem modülü yardımıyla AXI4 ana (master) arayüz sinyallerine dönüştürülür.

Veri belleğine gönderilecek sinyaller, aynı sistem veri yolunu kullanan diğer okuma/yazma istekleriyle karşılaştırılır ve bu sinyaller arasından uygun olanı seçilir. Bu seçim işlemi, buyruk tarafında da kullanılan ve Round-Robin algoritması ile çalışan seçici (arbiter) modülü tarafından yapılır. Bu yapının amacı, gelen istekler arasında adil (fair) bir seçim gerçekleştirmektir.

Seçilen sinyal paketi, veri belleği üzerinde doğrudan okuma ya da yazma işlemini gerçekleştirir ve çekirdeğin isteği yerine getirilmiş olur.

Öte yandan, axi4_xbar üzerinden gelen okuma/yazma sinyalleri de buyruk arayüzündekine benzer şekilde, axi4 deposunda yer alan axi_to_mem modülü aracılığıyla belleğe uygun hale getirilerek bellek arayüzüne yönlendirilir.

2.2.1.4 Birincil Harici Modüller

LDPC, DMA, ve PLIC gibi birimlerin önceliğinin diğer çevre birimlerine göre fazla olmasından dolayı sistem veri yoluna konumlandırılmalarına karar verilmiştir. Bu sayede daha yüksek hızlarda veri iletimleri gerçekleştirebilmektedirler. axi_xbar temelli sistem veri yolundan gelen axi4 slave arayüzleri tasarlayacağımız LDPC, DMA ve PLIC gibi modüllerin kontrol ve veri yazmaçlarına bağlanacaktır.

2.2.1.5 Çevre Birimi Bağlantıları

Çevre birimlerinin, sistem veri yoluna bağlı axi_lite_xbar modülü temelli çevre birimleri veri yoluna bağlı olacak şekilde tasarım gerçekleştirilmiştir. Çekirdeğin gerçekleştireceği bir yazma isteği sırası ile önce veri arayüzünde yer alan adres çözücüden geçer, ardından adresin veri yolu adres aralığına gelmesi ile istek axi_from_mem ile axi master arayüzüne dönüştürülür. Axi master isteği, axi4_xbar yapısının

slave arayüzle ile birlikte sistem veri yoluna aktarılır. içerde yer alan yapılar yardımı ve adres aralığının çevre birimleri veri yolunun adres aralığına düşmesi ile birlikte çevre birimleri veri yolunun master portuna aktarılır. Buradaki sinyal önce axi4'ten axi4 lite sinyaline dönüştürür ve axi4 slave arayüzünden çevre birimleri veri yoluna bağlanır. Bellek isteğinin tekrardan adres değeri ile ilgili çevre biriminin slave arayüzüne ulaşır. Aynı yol üzerinden response sinyali iletilerek veri işleminin tamamlanması sağlanır.

Bu tasarımımızda karşılaştığımız başlıca unsur hazır olarak alınan açık kaynak modüller olmuştur. Bu kaynak kodların modüler ve büyük çalışmalar olmasından dolayı kodun anlaşılması ve implemente edilmesinde bir takım uyum sorunları yaşanmıştır. Elimizde bulunan Sentez ve Simülasyon araçlarının bu projeler için daha basit olması kaynak kodun sınırlı kullanılmasına veya projenin kaynak koda göre ilerlemesine sebep olmuştur. Örneğin çekirdek çekirdek doğrulama işlemlerinde daha detaylı anlatıldığı şekilde; Pulp Platform'un çekirdek doğrulaması için geliştirilmiş olduğu test ortamı, Verilator ile tam yeterlilikte çalıştırılamamaktadır. Bu yüzden çekirdek doğrulaması Vivado ile Xsim de gerçekleştirilmiştir. Genel olarak kaynak kodların projeye implemente edilmesi sorunlarında ise kaynak kodun test modüllerinde nasıl çağrıldığı ve test edildiği izlenerek doğrudan teknik bilgi edilmiştir.

2.2.2. Çevre Birim Tasarım Detayları

Şartnamede tanımlanan 6 çevre biriminden 4'ü (GPIO, Timer, UART, QSPI Master) tamamlanmış ve test bench ile test edilmiştir. I2C Master modülü geliştirilme aşamasındadır. Tüm modüller AXI4-Lite arayüzü ile sistem bus'a bağlanmaktadır.

2.2.2.1. GPIO

Tasarım Detayları:

- Şartname EK-2'ye uygun olarak 16 adet giriş, 16 adet çıkış pini (sabit konfigürasyon)
- GPIO_IDR (0x00): Input data register - üst 16 bit her zaman '0'
- GPIO_ODR (0x04): Output data register - sadece alt 16 bit kullanılır
- AXI4-Lite slave arayüzü ile sistem bus'a bağlantı

Driver Durumu:

- C driver header (gpio_driver.h) ve implementasyon tamamlandı
- Pin-level manipülasyon fonksiyonları: gpio_set_output_pin(), gpio_clear_output_pin(), gpio_toggle_output_pin()
- Toplu okuma/yazma: gpio_read_input_pins(), gpio_write_output_pins()
- Masked write desteği ile seçici pin güncelleme

2.2.2.2. Timer

Tasarım Detayları:

- Şartname EK-2'deki tüm registerlar implementte edildi:
  - TIM_PRE (0x00): Prescaler - sistem saatini bölme
  - TIM_ARE (0x04): Auto-reload değeri
  - TIM_CNT (0x14): 32-bit sayaç (read-only)
  - TIM_EVN (0x18): Event counter - auto-reload olaylarını sayar
- Yukarı/aşağı sayma modları (TIM_MOD register)
- Clear ve enable kontrolleri

Özel Davranışlar:

- Aşağı sayma modunda ilk yüklemede event üretilmez
- Prescaler=0 iken her saat darbesi, prescaler=0xFFFFFFFF iken saniyede 1 sayım

Driver Durumu:

- Mikrosaniye, milisaniye ve saniye bazlı konfigürasyon fonksiyonları
- timer_delay_us(), timer_delay_ms() blocking delay fonksiyonları
- Non-blocking delay kontrolleri
- Event counter yönetimi

2.2.2.3. UART

Tasarım Detayları:

- Şartname EK-2'ye tam uyumlu register seti:
  - UART_CPB (0x00): Baud rate = (sistem saat frekansı)/UART_CPB
  - UART_STP (0x04): Stop bit kontrolü (1, 1.5, 2 bit)
  - UART_CFG (0x10): TX enable, RX/TX durum bayrakları
- 1 Mbps'ye kadar programlanabilir baud rate
- Hardware tarafından set edilen, software tarafından temizlenen durum bayrakları

State Machine Yapısı:

- 4 durumlu TX/RX state machine'ler (IDLE, START, DATA, STOP)
- Start bit doğrulaması ile false start detection
- Framing error kontrolü

Driver Durumu:

- Temel fonksiyonlar: uart_send_byte(), uart_receive_byte()
- String gönderimi: uart_send_string()
- Printf-style yardımcılar: uart_print_int(), uart_print_hex()

- Non-blocking receive: uart_receive_byte_nonblocking()

2.2.2.4. QSPI Master

Tasarım Detayları:

- Şartname EK-2'deki tüm komutlar desteklenir:
  - READ, DOR, QOR (okuma komutları)
  - PP, QPP (yazma komutları)
  - SE, WRR, WREN, WRDI (kontrol komutları)
- QSPI_CCR register yapısı:
  - Instruction (7:0), Data mode (9:8), R/W bit (10)
  - Dummy cycles (15:11), Data length (24:16), Prescaler (30:25)
- 8 adet 32-bit data register (DR0-DR7) ile 256 byte page desteği
- SDR mode, 3-byte addressing

State Machine:

- 6 durumlu kontrol: IDLE → SEND_INSTRUCTION → SEND_ADDRESS → DUMMY_CYCLES → TRANSFER_DATA → WAIT_COMPLETE
- Dinamik I/O yön kontrolü (x1/x2/x4 modları için)

Driver Durumu:

- Flash ID okuma: qspi_flash_read_id()
- Page program: qspi_flash_page_program() (x1 ve x4 desteği)
- Sector erase: qspi_flash_sector_erase()
- Bootloader fonksiyonu: qspi_boot_load_program()

2.2.2.5. I2C Master

Planlanan Tasarım:

- Şartname EK-2'ye uygun 400 kHz sabit SCL frekansı
- 7-bit adres modu (I2C_ADR[6:0])
- 1-4 byte veri transferi (I2C_NBY register kontrolü)
- I2C_RDR/TDR registerları ile veri alışverişi

Mevcut Durum:

- Register tanımlamaları tamamlandı
- Driver header dosyası (i2c_driver.h) hazır
- RTL implementasyonu devam ediyor

Karşılaşılan Zorluklar ve Çözümler

AXI4-Lite Protokol Uyumluluğu:

- Write address ve data kanallarının eş zamanlı yönetimi için aw_en flag mekanizması eklendi.

UART Timing Hassasiyeti:

- Yüksek baud rate'lerde doğru timing için clock-per-bit hesaplamasında dikkatli tasarım uygulandı.

QSPI Çok Modlu Veri Transferi:

- x1/x2/x4 modları arasında geçiş için qspi_data_oen sinyali ile dinamik I/O kontrolü sağlandı.

2.2.3. LDPC Hızlandırıcı Tasarım Detayları

Bu aşamada LDPC kodu yazılmamıştır. LDPC üzerine yapılan çalışmalar devam etmektedir. Araştırma sonucunda LDPC tasarımı hakkında ilgili bilgiler edinilmiştir.

LDPC, Temel olarak **Parite Denetim Matrisi'dir (H)** LDPC kodları, seyrek (low-density) bir H matrisiyle tanımlanır. Bu matris, kodun yapısını belirler. **Generator Matrix (G)** Kodlayıcıda, sistematik LDPC için G matrisi ile bilgi bitleri parite bitleriyle birlikte çıktı olarak üretilir. **Düzenli LDPC** Tüm satır ve sütunlar aynı sayıda "1" içerir. Donanım uygulamaları için daha öngörülebilir ve paralelleştirilebilir yapı sunar.

LDPC tasarımı aşağıdaki gibi planlanmıştır:

1. **Veri Alınacak:** Giriş verileri sistem veri yolundan 32-bit'lik parçalarda alınacak.
2. **Bilgi Bitleri Kaydedilecek:** Belleğe veya kayıt setine yazılacak.
3. **Parite Üretilecek:** Bilgi bitleri, G matrisi ile çarpılarak parite bitleri hesaplanacak.
4. **Çıktı Üretilecek:** Sistematik formatta (önce bilgi bitleri, sonra parite bitleri) kodlanmış veri çıkışa verilecek.

Bu tasarımın Python'da yazılması planlanmaktadır. Python kodu yazıldıktan sonra, RTL implementasyon aşamasında aşağıdaki sürecin izleneceğine karar kılınmıştır.

Modüler Tasarım yapılacak, Python'daki her blok, donanımda bir modüle dönüştürülecek. RTL Kodlama yapılacak, yani encoder core, input/output buffer, controller vs. bileşenler yazılacak. Sentez ve simülasyon yapılacak, sentezlenip ModelSim/Vivado'da test edilecek. Son olarak Performans Optimizasyonu yapılarak gecikme azaltılacak, pipelining ve paralellik artırılacak.

2.3. Boot

Boot işlemlerinin gerçekleştirilebilmesi amacıyla birkaç adet **Infineon** marka **128 MB kapasiteli NOR tipi SPI Flash Bellek** (S25FL128SAGMFI000, 16-pin SOIC) temin edilmiştir. Bu belleklerin devre kartı üzerinde test amacıyla kullanılabilmesi için, harici bağlantılara uygun dönüştürücü modüller de ayrıca edinilmiştir.

Sistem açılışında çalışacak boot (ön yükleme) kodu, tasarım içerisinde **salt okunur bellek** (ROM - Read Only Memory) yapısında tutulmaktadır. Simülasyon ve FPGA üzerinde yapılan testlerde, standart bir ROM modelinin donanımsal olarak gerçeklenmesi kullanılmaktadır. Boot amacıyla yazılan programın bu modele otomatik olarak yerleştirilmesini sağlayan bir görev tanımlanmış ve bu görev için gerekli betikler (script) ./7Yonga/tasks/boot_steps dizininde hazırlanmıştır. Bu betikler, boot sürecinde kullanılacak olan programdan bir **boot ROM modeli** oluşturarak test ortamına entegre edilmesini otomatize bir metod ile sağlamaktadır.

Mevcut durumda, sistemin tam anlamıyla boot gerçekleştirmesinden ziyade, ROM üzerinden çalıştırılmak üzere çeşitli test kodları kullanılmaktadır. Detaylı tasarım sürecinin tamamlanmasının ardından, yukarıda açıklanan boot işlem adımlarını gerçekleştirecek nihai bir boot programının geliştirilmesi ve ROM içerisine yerleştirilmesi planlanmaktadır.

2.4. FPGA Prototipleme

2.4.1 Driver Library Yapısı

Driver dosyaları 7Yonga/firmware/peripheral_drivers at main · mhfuzun/7Yonga dizini altında bulunabilecektir. I2C hariç, tüm driver dosyalarının header dosyaları hazırdır. GPIO modülünün kaynak kodları da mevcuttur. Aşağıda modül driver yapılarının örnek kullanımları ve genel fonksiyonları mevcuttur.

2.4.1.1 UART Driver Struct Yapısı

UART çevre birimi için hazırlanan driver, donanım yazmaçlarına erişimi kolaylaştıran struct yapısı kullanır:

```
typedef struct {
    uint32_t base_addr;      // UART çevre biriminin baz adresi
    uart_regs_t *regs;       // Register yapısına pointer
    uint32_t system_clock;   // Sistem saat frekansı (Hz)
} uart_driver_t;
typedef struct {
    volatile uint32_t CPB;   // 0x00 - Clock-per-bit register
    volatile uint32_t STP;   // 0x04 - Stop-bit register
```

```
    volatile uint32_t RDR;    // 0x08 - Read data register
    volatile uint32_t TDR;    // 0x0C - Transmit data register
    volatile uint32_t CFG;    // 0x10 - Configuration register
} uart_regs_t;
```

Örnek kullanım, baud rate ayarlama:

```
uart_driver_t uart;
 uart_init(&uart, 0x40002000, 50000000);
 uart_set_baud_rate(&uart, 115200); // CPB register'a 434 yazılır
```

2.4.1.2 GPIO Driver Struct Yapısı

```
typedef struct {
 uint32_t base_addr; gpio_regs_t *regs;
 } gpio_driver_t;
```

2.4.1.3 Timer Driver Struct Yapısı

```
typedef struct {
    uint32_t base_addr;      // Timer çevre biriminin baz adresi
    timer_regs_t *regs;      // Register yapısına pointer
    uint32_t system_clock;    // Sistem saat frekansı (Hz)
} timer_driver_t;


typedef struct {
    volatile uint32_t PRE;    // 0x00 - Prescaler register
    volatile uint32_t ARE;    // 0x04 - Auto-reload register
    volatile uint32_t CLR;    // 0x08 - Clear register
    volatile uint32_t ENA;    // 0x0C - Enable register
    volatile uint32_t MOD;    // 0x10 - Mode register
    volatile uint32_t CNT;    // 0x14 - Counter register (RO)
    volatile uint32_t EVN;    // 0x18 - Event register (RO)
    volatile uint32_t EVC;    // 0x1C - Event clear register
```

```
} timer_regs_t;
```

Timer konfigürasyonu:

```
timer_driver_t timer;

timer_init(&timer, 0x40003000, 50000000);  // 50MHz

timer_set_prescaler(&timer, 49999);       // PRE=49999, 1ms tick

timer_set_auto_reload(&timer, 1000);      // ARE=1000, 1 saniye periyot

timer_set_mode(&timer, TIMER_MODE_UP);    // MOD=1, yukarı sayma

timer_enable(&timer);                     // ENA=1
```

2.4.1.4 QSPI Driver Struct Yapısı

```
typedef struct {
   uint32_t base_addr;      // QSPI çevre biriminin baz adresi
   qspi_regs_t *regs;       // Register yapısına pointer
   uint32_t system_clock;   // Sistem saat frekansı (Hz)
} qspi_driver_t;


typedef struct {
   volatile uint32_t CCR;   // 0x00 - Communication configuration register
   volatile uint32_t ADR;   // 0x04 - Address register
   volatile uint32_t DR0;   // 0x08 - Data register 0
   volatile uint32_t DR1;   // 0x0C - Data register 1
   volatile uint32_t DR2;   // 0x10 - Data register 2
   volatile uint32_t DR3;   // 0x14 - Data register 3
   volatile uint32_t DR4;   // 0x18 - Data register 4
   volatile uint32_t DR5;   // 0x1C - Data register 5
   volatile uint32_t DR6;   // 0x20 - Data register 6
   volatile uint32_t DR7;   // 0x24 - Data register 7
   volatile uint32_t STA;   // 0x28 - Status register
} qspi_regs_t;
```

QSPI Flash Okuma Örneği:

```
qspi_driver_t qspi;

qspi_init(&qspi, 0x40004000, 50000000);

qspi_set_prescaler(&qspi, 1);  // SCLK = 25MHz


// Flash'tan 256 byte okuma (x4 mode)

uint8_t buffer[256];

qspi_flash_read(&qspi, 0x100000, buffer, 256, QSPI_DATA_MODE_4X);
```

2.4.2 Simülasyon Ortamı Testleri

Simülasyon ortamı için test scriptleri 7Yonga/gateware/peripherals/tests at main · mhfuzun/7Yonga dizinindedir. Testlerden çıkan sonuçlar aşağıdaki gibidir.

2.4.2.1 UART Modül Testi

UART modülü, SystemVerilog testbench'te UART alıcı modellemesi ile test edilmiştir:

- **Loopback Testi**: TX çıkışı RX girişine bağlanarak veri gönderimi ve alımı doğrulandı
- **Baud Rate Testi**: Farklı CPB değerleri (10, 50, 434) ile farklı hızlarda iletişim test edildi
- **Stop Bit Testi**: 1, 1.5 ve 2 stop bit konfigürasyonları doğrulandı

2.4.2.2 Timer Modül Testi

Timer modülü up/down counting modları ve auto-reload fonksiyonları ile test edildi:

- **Prescaler Testi**: Prescaler=2 ile her 3 clock cycle'da bir sayım
- **Auto-reload**: Counter ARE değerine ulaştığında sıfırlanma ve event counter artışı
- **Clear İşlemi**: CLR=1 ile counter sıfırlama, disabled durumda da çalışması

2.4.2.3 QSPI Master Testi

QSPI modülü, basit bir flash memory modeli ile test edildi:

- **Komut Gönderimi**: WREN, RDSR1, READ, PP komutlarının CCR register üzerinden gönderimi
- **Multi-mode Transfer**: x1, x2, x4 modlarında veri transferi

- **Page Program**: 256 byte'a kadar veri yazma, DR0-DR7 register'ları kullanımı

2.4.2.4 GPIO Modül Testi

GPIO modülü input/output işlemleri için test edildi:

- **Output Testi**: ODR register'a yazılan değerin gpio_out pinlerine yansıması
- **Input Testi**: gpio_in pinlerindeki değerin IDR register'da okunması
- **Read-Only Koruması**: IDR register'a yazma denemelerinin etkisiz olması

3. Çip Tasarım Akışı

EDA araçlarına ulaşım imkanı olmadığı için akışın aşamaları koşulamamıştır. Bu aşamada ÖTR sonrası akış hakkında bilgi akış hakkında daha detaylı bilgi toplanarak akış süreci için hazırlık yapılmıştır. Planlama, süreç detayları ve alınacak önlemler aşağıda belirtilmiştir.

Lisansların temin edildikten sonra doğrulamalar yapılacaktır. Tasarım nihai hale geldikten sonra çevre birimleri farklı saatlerde çalıştığı görülürse, CDC (Clock Domain Crossing) yapıları planlanacaktır.

Sentez aşamasında isterlere bağlı olarak teknoloji kütüphanesi belirlenecek ve sentez aracı ile Verilog/SystemVerilog dosyaları, zamanlama kısıtlamaları ve timing kitaplıkları kullanılarak sentez yapılacak ve gate-level netlist üretilecektir. Sentezden elde edilen alan ve timing raporları incelenecek, negative slack gibi durumlarla karşılaşılır veya alan istenenden büyük ise .sdc yeniden gözden geçirilecek ve RTL tasarımda sadeleştirmeye gidilecektir.

Floorplanning aşamasında Fiziksel alan bölünecek, makro bloklar yerleştirilecek ve I/O pin tanımları yapılacaktır. Die alanı belirlenecek, I/O pad yerleşimi yapılacak ve güç dağıtım planı yapılacaktır. Bu aşamada .lef dosyaları güncellenecektir. Bu aşamada eğer modüller beklenenden büyük olursa ve sıkışmaya yol açarsa block-level floorplanning yapılması planlanmıştır.

Placement aşamasında standard hücrelerin yerleşimi yapılacaktır. Standard hücrelerin yerleştirilmesi sonrasında utilization, timing değerleri kontrol edilecektir. Bu aşamada yine CTS(Clock-Tree Synthesis) yapılacaktır. Ana clock sinyali tree haline getirilerek bufferlanacaktır. Clock gecikmesi gibi durumlar ile karşılaşılırsa clock bölgeleri ayrılacak veya CTS topolojisi(örneğin H-tree) değiştirilecektir. Ayrıca örneğin Illegal placement ile karşılaşılırsa duruma bağlı olarak die border constraints tanımı düzenlenecek veya legalization tool çalıştırılacak, eğer CTS hazırlığında hatalar ile karşılaşılırsa clock constraintler düzenlenecek, Pre-CTS optimization çalıştırılacak.

Routing aşamasında hücreler arası bağlantılar çizilecektir. Öncelikle Global Routing ile katman planlaması yapılması, sonrasında Detailed Routing yapılarak tüm bağlantıların tamamlanması hedeflenmektedir. Bu aşamada Routing sıkışması gibi bir durumla karşılaşılırsa Route Layer Planning yapılması veya cell'lerin yeniden yerleştirilmesi yapılacaktır, bunula birlikte DRC'de karşılaşılacak hatalarda Routing rule'lar yeniden tanımlanabilir(Min Width Violation), Via türleri güncellenebilir(Via Enclosure Violation).

Signoff aşamasında DRC(Design Rule Check) ile katman kuralları kontrol edilecek, LVS(Layout versus Schematic) ile şematik ile layout eşleşmesi doğrulanacak, LEC(Logical equivalence check) ile RTL ile GDS arasındaki eşdeğerlik kontrol edilecek, RC Extraction ile .spef dosyası kullanılacak ve timing STA(static timing analysis ile) tekrar analiz edilecek. Bu aşamada karşılaştığımız hatalarda, mesela DRC'de, örneğin metal spacing violation veya minimum fill violation gibi hatalarda karşılaştığımızda Detailed Routing tekrar yapılacak ve Dummy fill eklenecek. LVS'te karşılaşılabilecek Port Mistmatch durumunda Clean-up sonrası yeniden extract yapılacak, Power/Ground Errors hatası ile karşılılırsa Power net assignmentları yeniden yapılacak. LEC'de Constant Propogation Differences hatası ile karşılaşılırsa RTL'de define ve parametreler yeniden kontrol edilecek, RC Extraction'da Setup/Hold violations ile karşılaşılırsa buffer eklenebilir, clock tree optimize edilir veya lojik gecikme azaltılır ve yol kısaltımı amaçlanır.

Son aşamada GDSII dosyası ve yanında .vcd, .sdf, .lef dosyaları (simülasyon amaçlı) üretilmesi ve tasarımın başarılı bir şekilde sonuçlanması hedeflenmektedir.

3. Test

LDPC AXI arayüzü için UVM/SV ile yapılan doğrulama çalışmaları doğrultusunda AXI doğrulaması için sağlanacak VIP(Verification Intellectual Property) ve UVM/SV doğrulaması için ticari araçlar sağlanmadığı için doğrulamalar yapılmamıştır. Açık kaynak VIP'lerin kullanılmasının eksik puan alınmasına sebep olacağı belirtilmiştir, bu doğrultuda ilgili doğrulama faaliyetleri tekrar edilmemek adına final aşamasına saklanmıştır. Bunun yerine AXI arayüzü ile bir dummy monitor oluşturmuş, final aşamasında doğrulama süreci planlanmıştır. Bu aşamada AXI Slave interface'e bağlanacak şekilde pasif VIP bağlantı altyapısı hazırlanmıştır. ve sistemin AXI Master'ı üzerinden LDPC'ye veri aktarımı planlanmış ve bu veri yolundaki trafiğin protokol uyumluluğu izlenecektir.

AXI interface modülünde AXI sinyal ve datalar belirtilmiştir. Sinyal yönü belirleme ve doğru clock kenarlarında örnekleme için clocking blokları kullanılmıştır. Ana ister için tanımlanan pasif clocking yanında yapılacak testlerde kullanılma ihtimaline karşı aktif clocking bloğu tanımlanmıştır.

AXI Lite arayüz trafiğinin pasif gözlemlenmesi amacıyla axi_lite_dummy_monitor sınıfı tanımlanmıştır. Bu sınıf, uvm_monitor sınıfından türetilmiştir ve hem veri gözlemi hem de UVM analiz portu üzerinden transaction iletimi işlevlerini yerine getirir. Monitör, protokol uyumluluğunu izleyerek AXI Lite haberleşmesinin doğrulanmasını sağlar. Pasif bağlantı için interface'e clocking bloğu ile erişim sağlanır. Bu sayede, posedge pclk kenarında AXI Lite sinyalleri stabil iken gözlemlenir. Bu, AXI protokolüne uygun örnekleme zamanlamasının elde edilmesini garanti eder. Monitor sınıfında, build_phase içerisinde vif atanması yapılır. Veri gözlemi run_phase içerisinde, AXI Lite protokolü el sıkışma kurallarına uygun olarak sıralı şekilde yapılır. Örneğin, write işlemleri önce AWVALID && AWREADY, ardından WVALID && WREADY, en sonunda da BVALID kontrolü ile tamamlanır. trans_observed() fonksiyonu, gözlemlenen transaction'ların kontrolü, loglama veya coverage işlemleri için kullanılır.

Dummy monitöre eşlik etmesi amacıyla arayüzünün yanında transfer bilgilerini tutmak için axi_rw sınıfı ve izlenen işlemleri analiz etmek, kontrol etmek ve coverage yapmak için callback sınıfı eklenmiştir.

axi_rw sınıfı, hem okuma (READ) hem de yazma (WRITE) türündeki AXI işlemlerini temsil etmek amacıyla tanımlanmıştır. addr, data ve kind üyeleri ile bir AXI Lite transferinin temel bileşenlerini içerir.

İkinci yapı olan axi_lite_dummy_monitor_cbs sınıfı ise, uvm_callback tabanlı bir mekanizma ile axi_lite_dummy_monitor tarafından gözlemlenen transferlerin ek analizini yapar. Bu sınıf, doğrulama sürecinde kullanılan bir callback mekanizmasıdır ve AXI Lite protokolüne uygunluk kontrolü, covergroup, hata ve uyarı üretimi gibi işlemleri yapar. İlgili dosyalar ./7Yonga/tasks/verif dizininde yer almaktadır.

Mikrodenetleyici Çekirdeğinin Doğrulanması:

Mikrodenetleyici çekirdeğinin doğrulama sürecinde ilk olarak açık kaynaklı **Verilator** simülasyon aracı tercih edilmiştir. RV32E40P çekirdeğine ait donanım tanımlama kodlarının (HDL - Hardware Description Language) simülasyonu başarıyla derlenmiş olsa da, çekirdeği çevreleyen ve yürütme kayıtlarını izleyen bazı modüllerin **SystemVerilog** dilinin ileri düzey özelliklerini içermesi nedeniyle Verilator çeşitli hatalar üretmiştir.

Bu yapısal uyumsuzluklar nedeniyle, doğrulama sürecinde **Vivado** geliştirme ortamının sunduğu **Xsim** simülasyon aracı kullanılmasına karar verilmiştir. Doğrulama için gerekli olan HDL kodları ve ilgili betikler ./7Yonga/tasks/core_verification dizininde yer almaktadır. Bu dizinde, test kodlarının derlenmesini sağlayan ve Vivado Xsim aracını çalıştıran komut dosyaları hem **Windows Batch** hem de **TCL (Tool Command Language)** formatında hazırlanmıştır. Bu betikler sayesinde GUI-grafik arayüzü olmadan otomatize bir yöntem ile doğrulamanın çalıştırılabilmesi sağlanmaktadır.

Test kodlarının doğrulama amacıyla Spike simülatörü ile yürütülmesi için **Linux** ortamında bir **Makefile** betiği kullanılmaktadır. Spike çıktıları ile Xsim çıktılarının karşılaştırılması amacıyla, **C dili** ile geliştirilmekte olan özel bir karşılaştırıcı program bulunmaktadır. Mevcut durumda bu karşılaştırıcı, her iki yürütmenin kayıt çıktıları ile program sonundaki başarılı/başarısız sinyallerine odaklanmaktadır.

Bu doğrultuda, RV32E40P çekirdeğinin hem Vivado Xsim aracı hem de kendi geliştirdiğimiz test programları kullanılarak temel doğrulamaları gerçekleştirilmiştir. İlerleyen aşamalarda, C dili ile geliştirilen karşılaştırıcı programın yetenekleri artırılarak daha ayrıntılı ve sistematik doğrulamalar yapılması hedeflenmektedir.

4. Takım Organizasyonu

4.1. Takım Tanımı

7Yonga projesi için takımımız 3 kişiden oluşmaktadır. Tablo 5.1'de üyelerimizin öğrenim ve görevleri listelenmiştir.

Üye	Öğrenim
Muhammet Furkan UZUN	Yeditepe Üniversitesi, Elektrik-Elektronik Mühendisliği 2. Sınıf öğrencisi.
Hasan GÜZELŞEMME	Yeditepe Üniversitesi, Bilgisayar Mühendisliği 3. Sınıf öğrencisi.
Erhan ÖNALDI	Yeditepe Üniversitesi, Bilgisayar Mühendisliği 4. Sınıf öğrencisi.

Tablo 5.1 - Üyeler ve Öğrenim Bilgileri

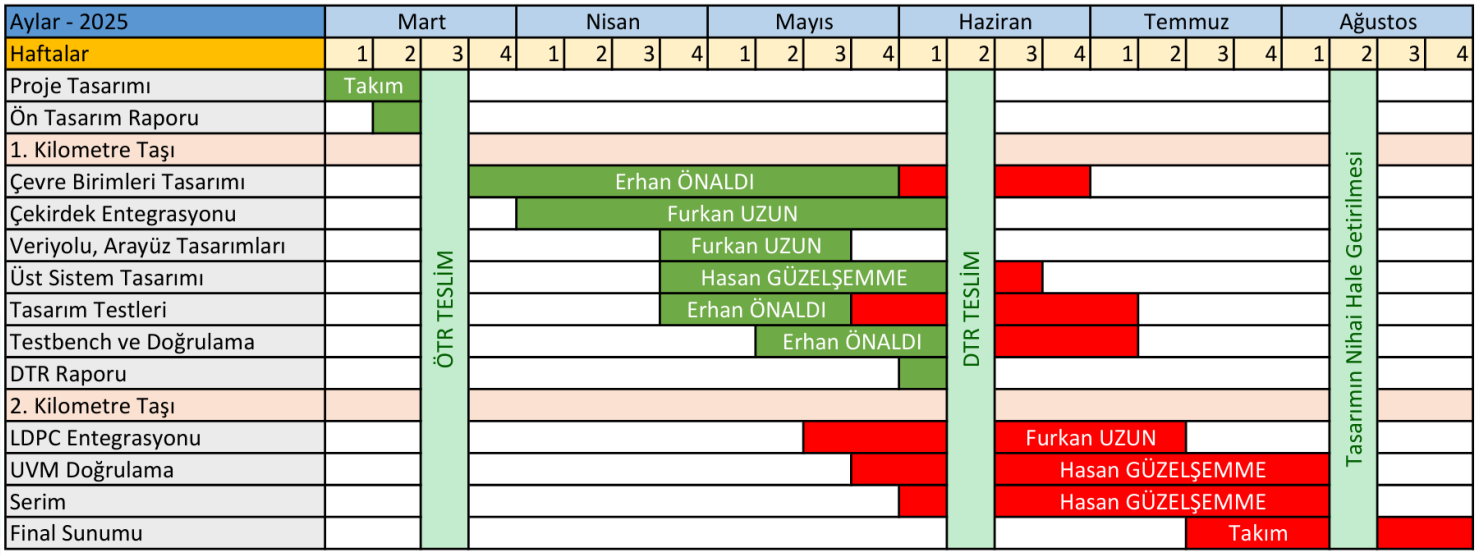
4.2. Görev Dağılımı

ÖTR sonrasında yapılan detaylı değişiklikler bulunmamaktadır. Bunun haricinde Ön Tasarım Raporunda tam anlamıyla bahsedilmeyen DMA (Doğrudan Bellek Erişimi) modülünün uygulaması detaylandırılmıştır.

Üye	Görev
Muhammet Furkan UZUN	Çekirdeğin, belleklerin, veri yollarının implementasyonu.
Hasan GÜZELŞEMME	Çevre birimleri tasarımı, Devre serimi, LDPC implementasyonu.
Erhan ÖNALDI	Çevre birimleri tasarımı, testler, Bare-Metal test ortamı geliştirme.

5. İş Planı ve Risk Planlaması

Projede birçok kod bileşeninin entegrasyonu başarıyla tamamlanmış olsa da, test süreçlerinin tamamlanamaması nedeniyle FPGA üzerinde yapılması planlanan doğrulama çalışmaları aksamış ve bu durum proje takviminde gecikmelere yol açmıştır. DTR sonrası projenin tamamlanarak nihai hale getirilmesi planlanmaktadır.



6. Kaynakça

1. https://docs.openhwgroup.org/projects/cv32e40p-user-manual/en/cv32e40p_v1.8.3/intro.html
2. https://docs.openhwgroup.org/projects/cv32e40p-user-manual/en/cv32e40p_v1.8.3/load_store_unit.html
3. https://docs.openhwgroup.org/projects/cv32e40p-user-manual/en/cv32e40p_v1.8.3/integration.html