ÇİP TASARIM YARIŞMASI

MİKRODENETLEYİCİ TASARIM KATEGORİSİ

ÖN TASARIM RAPORU

Takım Adı: Yeditepe Üniversitesi Sayısal Tasarım Topluluğu

TAKIM ID: 582202

Başvuru ID: 3062081

İçindekiler

Kısaltmalar:
AXI4: Advanced eXtensible Interface 4
BRAM : Block Memory
DMA: Direct memory access
DTR: Detaylı Tasarım Raporu
FPGA: Field-Programmable Gate Array
GUI: Graphical User Interface
HDL - Hardware Description Language
LDPC: Low Density Parity Check
PLIC: Platform-Level Interrupt Controller
ROM: Read-Only Memory
ÖTR: Ön Tasarım Raporu
SRAM: Static Random Access Memory
TCL: Tool Command Language

1. Sistem Tanımı ve Temel Tasarım Özeti (5p)

{**Bu kısımda sistemin basitçe genel tanımı ve proje kapsamında yürütülen faaliyetlerle ilgili olarak özet bilgiler sunulur. Örneğin RTL tasarımı, RTL doğrulaması, çip akışı faaliyetleri vb. tamamlanma oranları nelerdir?**}

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 hedeflemekteyiz.

Tasarımımızda PULP Platform'un geliştirmiş olduğu AXI4 veri yolu kitaplığı ve OpenHW Group'un RV32E40P çekirdeğini temel almaktadır. Geiştirdiğimiz çevrebirimlerini, veri-buyruk belleklerini ve LDPC modulümüzü çekirdek ve AXI4 ile bir araya getirerek mikrodenetleyicimizi oluşturmaktayız.

ÖTR sonrası süreçte AXI4 ve RV32E40P reposunun incelenmesi ve testleri tamamlanmıştır. Belleklerin ve çevrebirimlerinden UART'ın tasarımı tamamlanmıştır. Ayrıca çekirdeğin buyruk ve veri arayüzleri için gerekli kontrol birimleri de tamamlanmıştır. DTR öncesi FPGA testleri tamamlanmak istense de bütün sistemin bir araya getirilmesi sürecinde yaşanan Vivado-Simulasyon hatalarından dolayı şuan için tasarımın bütününün testleri aşamasınd bulunmaktayız. DTR süreci sonrasında testbenchler vasıtasıyla tasarım doğrulandıktan sonra FPGA testleri gerçekleştirilecektir.

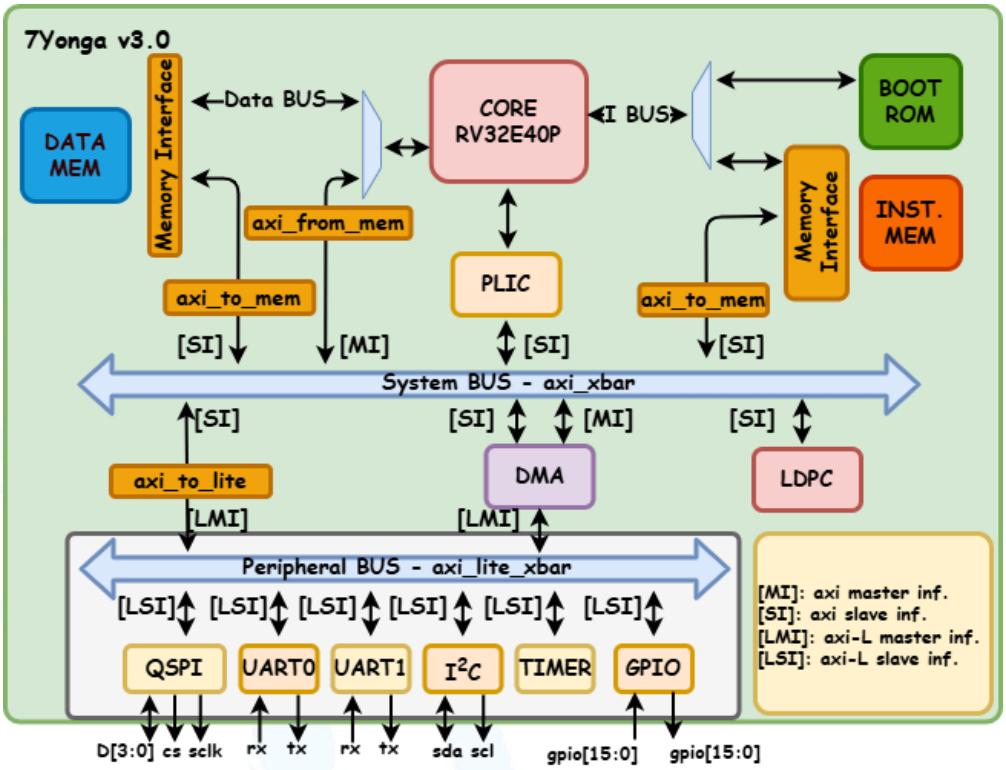
*** Çip akışını eklemeleyiz, ÖTR sonrası neler yapıldı. mesela bir flow koştuk gibi kısaca bahsedilmeli.

2. Proje Detay Tasarımı (55p)

2.1. Sistem Mimarisi (15p)

{**Bu kısımda projenin nihai tasarımının blok şeması verilir ve yarışma şartnamesi ile tasarlanan sistemin uyumluluklarına değinilir. Alt bloklardan ve görevlerinden bahsedilir. Bus yapısı, boot sekansı, çevre birimlerinin bağlantı detayları, yazılım ile ilgili yapılan çalışmalar, linker script, projenin FPGA ve ASIC implementasyon aşamalarındaki farklar vb. açıklanır.**}

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

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. [https://docs.openhwgroup.org/projects/cv32e40p-user-manual/en/cv32e40p_v1.8.3/intro.html]

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.

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. [https://docs.openhwgroup.org/projects/cv32e40p-user-manual/en/cv32e40p_v1.8.3/load_store_unit.html]

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 AXI4Lite 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.

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.

LDPC; hasanguzelsemme@gmail.com bu bölüm için de kısa bir özet gerekmektedir.

Boot Süreci;

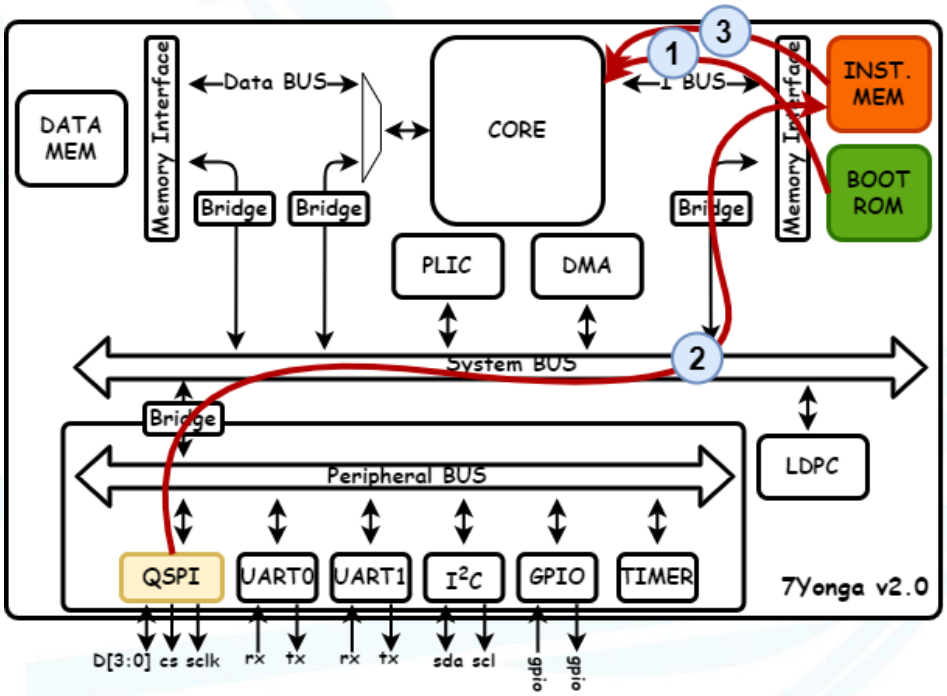
Boot süreci mikrodenetleyicinin başlatıldığında önce boot ROM'dan çalışmasının ardından Qspi modülü ile birlikte harici flash belleğinden programın okunması ve buyruk belleğine yazılması sürecidir.

Üç aşamadan oluşan ön yükleme süreci, sistemin açılmasıyla birlikte **Şekil x.x.x**'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 gerçekleştirecektir. 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 x.x.x - Boot Süreci Gösterimi

2.2. Tasarım Detayları (25p)

2.2.1. İşlemci ve Bus Yapısının Tasarımı (10p)

{**CV32E40P işlemcisinin portları mikrodenetleyici içerisinde nasıl bağlanıyor? Hangi bus yapısı kullanıldı, işlemci ve çevre birimleri bu bus'a nasıl bağlandılar? Seçilen yöntem ve metotların diğerlerine göre artı eksileri nelerdir? Karşılaşılan zorluklar ve çözümleri ne oldu? Bu aşamada bu hedefin başarılmış olması beklenmektedir.**}

*** M. Furkan UZUN, görseller ile veri-buyruk kontrolcülerinden bahset. çekirdeğin portlarından bahset. Aradaki axi_to_mem, axi_from_mem, axi_to_lite, addr_decoder, axi_xbar, axi_lite_xbar modüllerinden bahset. sram, bram, soft_ram switch olayından bahset.

Çekirdek Entegrasyonu;

“cv32e40p_top” modülü, işlemcinin en üst (top) düzey modülüdür. Bu modül, mikrodenetleyici sisteminin top modülünde çağrılarak sisteme entegre edilir. [https://docs.openhwgroup.org/projects/cv32e40p-user-manual/en/cv32e40p_v1.8.3/integration.html]

“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.

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.

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:

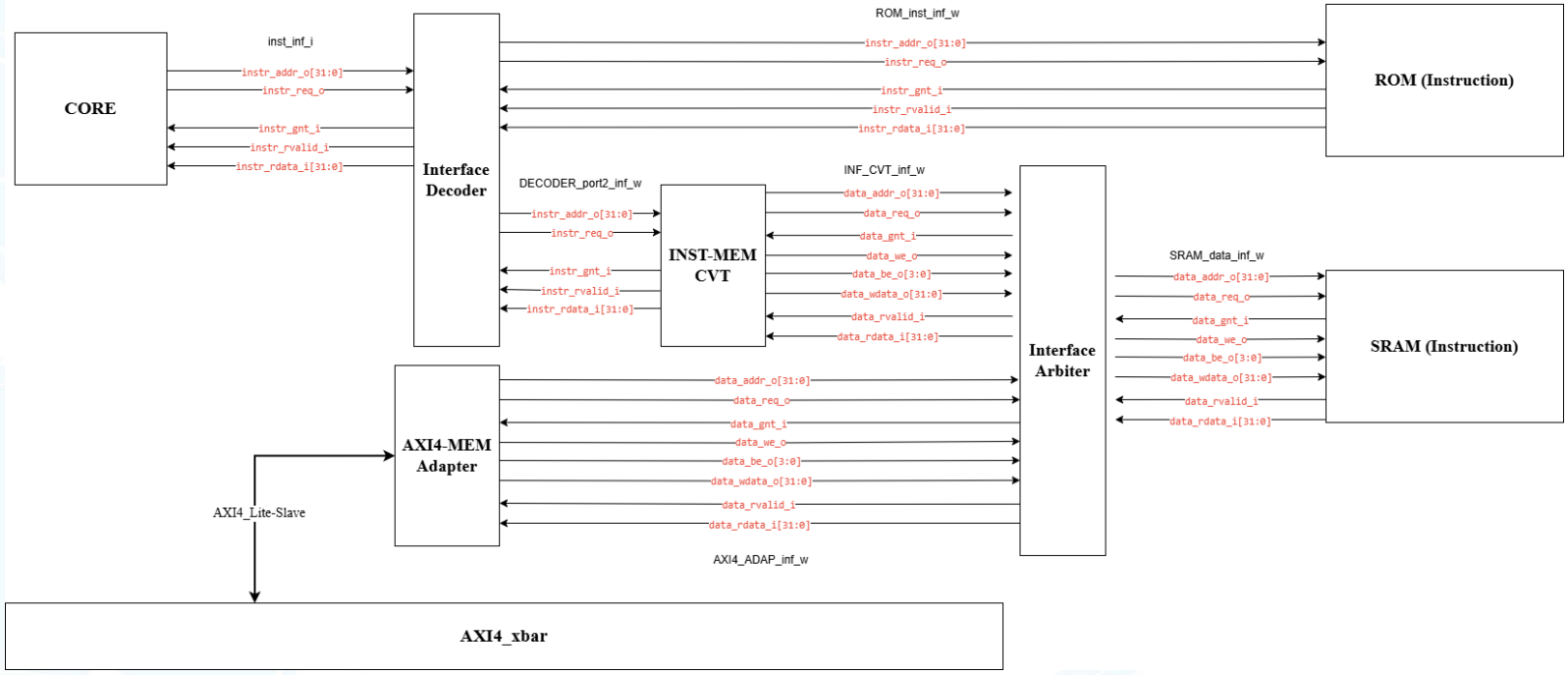
- `instr_addr_o`: Buyruk adres çıkışı (instruction address output)
- `instr_req_o`: Buyruk isteği çıkışı (instruction request output)
- `instr_gnt_i`: Buyruk izni girişi (instruction grant input)
- `instr_rvalid_i`: Buyruk verisinin geçerli olduğunu belirten giriş (instruction read valid input)
- `instr_rdata_i`: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

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.

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.

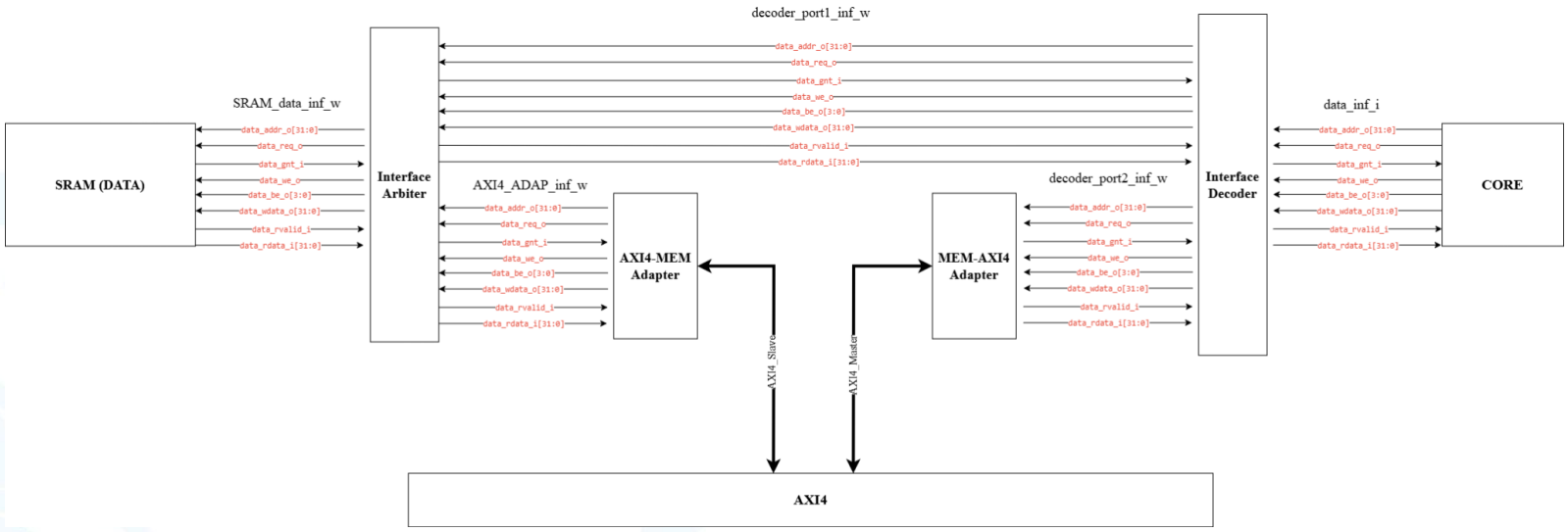
- `data_gnt_i` (**Giriş / Input**): Bellek arayüzü isteği kabul ettiğinde bir çevrimliğine yüksek olur (grant). Bu sinyal alındıktan sonra `data_addr_o` bir sonraki döngüde değişebilir.
- `data_we_o` (**Çıkış / Output**): Yazma etkinleştirme sinyali (write enable). Yüksek (high) olduğunda yazma, düşük (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.



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.

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 (8p)

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

1. **AXI4-Lite Protokol Uyumluluğu:**
   - Write address ve data kanallarının eş zamanlı yönetimi için aw_en flag mekanizması
2. **UART Timing Hassasiyeti:**
   - Yüksek baud rate'lerde doğru timing için clock-per-bit hesaplamasında dikkatli tasarım
3. **QSPI Çok Modlu Veri Transferi:**
   - x1/x2/x4 modları arasında geçiş için qspi_data_oen sinyali ile dinamik I/O kontrolü

2.2.3. LDPC Hızlandırıcı Tasarım Detayları (7p)

{**Şartnamede tanımlanan LDPC enkoder hızlandırıcının tasarım detayları hakkında bilgi verilmesi. Yazılım modellemesi aktiviteleri, modelden RTL gerçekleme akışı, tasarımın veri akış şeması nelerdir? İlgili model ve RTL kodları için GitHub hesabınızdaki ilgili dosya yoluna link verilebilir ama DTR'de LDPC hızlandırıcı tasarımı detaylandırılmalıdır, sadece koda link verilip geçilmemelidir. Karşılaşılan zorluklar**

ve çözümleri ne oldu? Bu aşamada en azından yazılım modellemesi faaliyetlerinin tamamlanmış olması beklenmektedir.}

*** hasanguzelsemme@gmail.com

2.3. Boot (7p)

{**Yazılan C/assembly kodunun derlenip simülasyon ortamında boot memory veya flash memory'e (bu aşamada opsiyonel) yüklenmesine ve çekirdek tarafından çalıştırılmasına dair detaylar. Bu aşamada en azından belirtilen düzeyde boot faaliyetlerinin tamamlanmış olması beklenmektedir.**}

*** M. Furkan UZUN C kodlarından bahset, crt0.S'den bahset.

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 (8p)

{**FPGA prototipleme detayları. FPGA'da işlemci üzerinde yazılım koşturuldu mu? Yazılımlarla çevre birimleri test edildi mi? C kodları için GitHub hesabınızdaki ilgili dosya yoluna link verilebilir ama FPGA üzerinde veya simülasyon ortamındaki çalışmalar DTR'de detaylandırılmalıdır. Alınan sonuçlar görsellerle belirtilmelidir. Karşılaşılan zorluklar ve çözümleri ne oldu?**}

***erhan.onaldi@std.yeditepe.edu.tr

*** C kodları yazlıdıktan sonra "struct" yapılarının görseli ve anlatımı bulunmalı en azından UART için yapılmış olsun. Örneğin; örnek kodlardaki uart_drive gibi bir library hazırlanmalı.

FPGA prototipleme henüz gerçekleştirilememiş olmakla birlikte, çevre birimleri için C driver'lar hazırlanmış ve simülasyon ortamında kapsamlı testler yapılmıştır.

2.4.1 Driver Library Yapısı

2.4.1.1 UART Driver Struct Yapısı

UART çevre birimi için hazırlanan driver, donanım register'larına 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;
```

Baud rate ayarlaması:

```
int uart_set_baud_rate(uart_driver_t *driver, uint32_t baud_rate);
```

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;
```

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;
```

2.4.2 Simülasyon Ortamı Testleri

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 (10p)

{**Bütün bir sistemin olmasa da öncül çalışma olarak çevre birimlerin biri veya birkaçı ya da işlemci çekirdeği için çip fiziksel tasarım akışının sentez ve statik zamanlama analizi gerçeklenmiş olması beklenmektedir. Mümkünse tasarımın hepsi, tasarımın hepsi tamamlanmadıysa da tamamlanmış alt modüller üzerinde zamanlama analizi (timing analysis), kaynak kullanımı (utilization) gibi dosyaların çıktılarından yararlanılarak yorumlarda bulunulmalıdır. Performans ve alan tüketimini iyileştirmek adına akışta yapılan özelleştirmelere değinilmelidir. Tasarım sürecinde gerek yazılımsal olarak gerek tasarımsal olarak karşılaşılan sorunlara ve bu sorunların nasıl çözüldüğüne değinilmelidir (Örneğin sentez programı tasarımı sentezleyemediği yönünde bir hata vermiştir, yarışmacılar da bu durumu bir yöntemle çözmüştür vb.). Bu çözümler esnasında kullanılan teorik-genelgeçer bilgilere yer verilmelidir (Örneğin bu kaynaktan edinilen bilgiye göre utilization değeri bu değerin altında tutulmuştur). Çip tasarım akışı boyunca faydalanılan topluluklar, Slack kanalları gibi yerler varsa bunların hangilerinden ne yönde yararlanıldığından bahsedilmelidir. Çip akışında geçilen aşamaların hangilerinin daha kolay hangilerinin daha zor bulunduğuna, hangilerinde ne kadar zaman geçirildiğine kısaca değinilmelidir.**}

*** hasanguzelsemme@gmail.com

4. Test (20p)

{**Bu kısımda tasarım için hazırlanan test planı ve gerçeklenmiş test raporu, RTL seviyesinde fonksiyonel simülasyonlar, CV32E40P işlemcinin doğrulanması, LDPC AXI arayüzü için UVM/SV ile yapılan doğrulama çalışmaları, varsa coverage raporları ve static timing analysis ile maksimum frekans hesaplamaları detaylandırılmalıdır. Tüm tasarım tamamlanmasa bile hangi alt modüllerde ve alt sistemlerde simülasyonların nasıl gerçekleştirildiği, doğrulama ortamı (FPGA ve simülasyonlar için hangi araçlar kullanıldıysa) ve testbench kodları ile bilgiler detaylıca verilmelidir. Bu aşamada en azından çekirdek + bir çevre birimiyle alt sistem düzeyi bir test senaryosu simülasyonda başarıyla koşabilmelidir.**}

*** hasanguzelsemme@gmail.com UVM notları.

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.

5. Takım Organizasyonu (3p)

5.1. Takım Tanımı (1p)

{**Bu kısımda takım üyeleri ve varsa danışman hakkında bilgi verilir. (İsim, soyisim, okul, bölüm, sınıf)**}

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

Tablo 5.1 - Üyeler ve Öğrenim Bilgileri

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.

5.2. Görev Dağılımı (2p)

{**Bu kısımda görev dağılımı ve ekip organizasyonu hakkında bilgi verilir. Kim hangi kısımlardan sorumlu? ÖTR planlamasına kıyasla değişiklikler varsa nedenleri.**}

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.

6. İş Planı ve Risk Planlaması (5p)

{**Bu kısımda projenin FPGA prototipleme ve çip akışı aşamaları için tasarım, doğrulama, sentez, fiziksel gerçekleme ve test süreçlerini içeren bir zaman planlaması ve risk planlaması yapılır. Zaman akış çizelgesi üzerinde iş paketlerinin ne kadarının tamamlandığı ne kadarının henüz tamamlanmadığı, takvimde gecikme olup olmadığı açık bir şekilde gösterilmelidir.**}

7. Kaynakça (2p)

{**Bu bölümde raporda kullanılan kaynaklar, tutarlı bir kaynakça üslubuyla yer almalıdır. Kaynaklar rapor içerisinde referanslanmalıdır.**}

DTR ile İlgili Notlar (Bu tablo, yapılacak DTR teslimatlarından çıkartılmalıdır)

- Tüm raporlar "İçindekiler" bölümü içermelidir.
- Her rapor bir kapak sayfası içermelidir.
- "Kapak", "İçindekiler" ve "Kaynakça" bölümleri, azami rapor uzunluğu olan 30 sayfanın dışında değerlendirilecektir.
- Rapor sayfaları birbirini takip edecek şekilde numaralandırılmalıdır.
- Yazı tipi: Calibri, Başlık boyutu: 14-11 punto, Yazı boyutu: 11 punto, Satır aralıkları: 1.15.
- Rapor teslim tarihinde tamamlandığı yazılan aktivitelerin yinelenebilirliği adına gerekli betikler ve OKUBENİ dosyaları, repo'larınız içerisinde bulunmalıdır (FPGA aktiviteleri hariç; simülasyonlar sağlanacak simülatörde, sentezler sağlanacak sentez aracında vb. çalışabilmelidir).
- Raporlarda bütün aktivitelerin tamamlanmış olması beklenmemektedir. Ancak net bir şekilde biten aktivitelerin detayları ve bitmemiş aktivitelerin detaylı planlarına raporların ilgili bölümlerinde yer verilmelidir. Tamamlanması zorunlu aktiviteler ilgili başlıklar altında belirtilmiştir.
- Şablonda sağlanan başlıklar sabit olmakla birlikte yarışmacılar, uygun gördükleri noktalarda ek alt başlıklar yaratabilirler.
- Eğer bir görev, "Bu aşamada en azından X tamamlanmış olmalıdır" ve benzeri bir şekilde tanımlanmışsa puanlandırmadaki etkisini vurgulamak amacıyladır. Bu şekilde tanımlanmamış görevler, örneğin ASIC akışında P&R aktivitesi için yalnızca öncül çalışmaların ve yeterince-iyi planların yazılması puanlandırma için yeterlidir. Ancak zorunlu olarak tanımlanmamış ve planlanması yeterli olan görevlerin tamamlanması durumu için puan havuzundan kısmi bir alan ayrılmıştır.