

**ÇİP TASARIM YARIŞMASI**

**2024**

**MİKRODENETLEYİCİ TASARIM KATEGORİSİ**

**DETAY TASARIM RAPORU**

**ArchOzU**

**……………………………….**

**2007098**

**………………………………**

İçindekiler

[1. SİSTEM TANIMI VE TEMEL TASARIM ÖZETİ (5 PUAN) 2](#_Toc163043486)

[2. PROJE DETAY TASARIMI (50 PUAN) 2](#_Toc163043487)

[2.1. Sistem Mimarisi (20) 2](#_Toc163043488)

[2.2. Tasarım Detayı 2](#_Toc163043489)

[2.2.1. İşlemci ve Bus Yapısının Tasarımı (10) 2](#_Toc163043490)

[2.2.2. Peripheral Tasarım Detayları (10) 2](#_Toc163043491)

[2.2.3. FPGA Prototipleme Detayları (10) 3](#_Toc163043492)

[3. ÇİP TASARIM AKIŞI (15 PUAN) 3](#_Toc163043493)

[4. TEST (20 PUAN) 3](#_Toc163043494)

[5. TAKIM ORGANİZASYONU (3 PUAN) 3](#_Toc163043495)

[5.1. Takım Organizasyonu 3](#_Toc163043496)

[5.2. Görev Dağılımı 4](#_Toc163043497)

[6. İŞ PLANI ve RİSK PLANLAMASI (5 PUAN) 4](#_Toc163043498)

[7. KAYNAKÇA (2 PUAN) 4](#_Toc163043499)

# SİSTEM TANIMI VE TEMEL TASARIM ÖZETİ (5 PUAN)

{Bu kısımda sistemin basitçe genel tanımı ve proje kapsamında yürütülen faaliyetlerle ilgili olarak özet bilgiler sunulur. Örneğin RTL aşaması, simülasyon aşaması, çip fiziksel tasarım aşaması vs. tamamlanma oranları nelerdir?}

Bu proje OpenHW tarafından geliştirilen 4 aşamalı CV32E40P RISCV çekirdeğine çeşitli çevre birimleri ekleyerek mikrokontrolcü yapmayı hedeflemektedir. Bu çevre birimleri şu şekildedir:

* UART
* I2C Master
* QSPI Master
* Timer
* GPIO
* JTAG
* USB

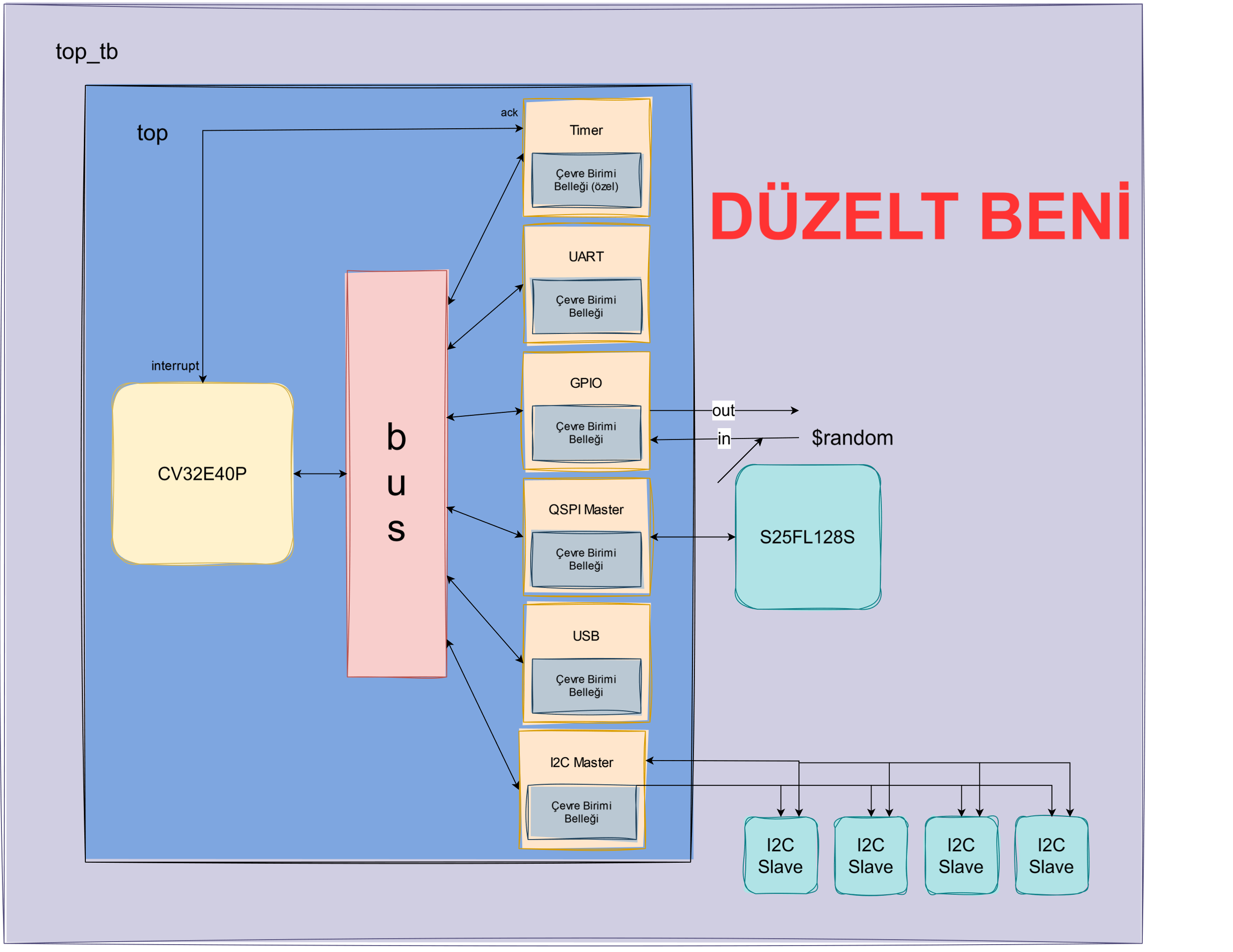
Bu çevre birimlerinin çekirdek ile bağlantısının kurulabilmesi için bus ve çevre birimi belleği modülleri kullanılmıştır. Hepsinin detayları ileriki bölümlerde anlatılacaktır. JTAG dışındaki tüm modüllerin RTL kodu tamamlanmıştır. JTAG, DTR sonrasına bırakılmıştır. USB ve JTAG dışındaki tüm modeller simüle edilmiştir. Çip fiziksel tasarımı tamamlanmıştır.[TAMAMLANDI DİMİ]

# PROJE DETAY TASARIMI (50 PUAN)

## Sistem Mimarisi (20)

{Bu kısımda projenin nihai tasarımının blok şeması verilir ve yarışma şartnamesi ile tasarlanan sistemin uyumluluklarına değinilir.

Projenin tamamlanan kısmının blok şeması aşağıdaki şekildedir, dolayısıyla JTAG, henüz dahil değildir:



Alt bloklardan ve görevlerinden bahsedilir. Bus yapısı, boot sekansı, peripheralların bağlantı detayları, yazılım ile ilgili yapılan çalışmalar, linker script, projenin FPGA ve ASIC implementasyon aşamalarındaki farklar vs}

## Tasarım Detayı

### İşlemci ve Bus Yapısının Tasarımı (10)

{CV32E40P işlemcisinin portları mikrodenetleyici içerisinde nasıl bağlanıyor?

Hangi bus yapısı seçildi, işlemci ve peripherallar bu bus’a nasıl bağlandırlar?

Çok fazla peripheral olmadığı için bus yapısını kendimiz hazırladık.

Seçilen yöntem ve metotların diğerlerine göre artı eksileri? Karşılaşılan zorluklar ve çözümleri?}

Bus yapısını biz hazırladığımız için kontrol tamamen bizde, rahatça değişiklik yapabiliriz. Ayrıca bu bus’ın nasıl çalıştığını çok daha iyi anlamamızı sağlıyor. Dahası, APB+AHB kombinasyonuna göre çok daha az yer kaplıyor.[KANITLA] Sıfırdan bus yapısı yazmak çok fazla hata yapmamıza sebep oldu.

### Peripheral Tasarım Detayları (10)

{Şartnamede tanımlanan peripheralların tasarım detayları hakkında bilgi verilmesi.

Bizim yazdığımız modül(ler):

UART

I2C

QSPI  
Timer

GPIO

Perip mem

Bus

Hazır aldığımız/alacağımız modül(ler):

JTAG

USB

### Çevre Birimi Belleği (perip\_mem.v)

Bus’ı tasarlarken önce veri belleğini 8kB’tan daha büyük yapıp tüm çevre birimlerine erişim hakkı vermeyi düşündük lakin bu sistemi inanılmaz karmaşık yapıyordu. Daha sonra çevre birimlerinin içerisine yazmaç olarak tutmayı denedik. Bu sistem çalışıyordu fakat hata yapmaya çok müsaitti, debug etmesi çok zordu ve aynı anda tüm yazmaçları okumuyorduk. Dolayısıyla bir verimsizlik vardı. Ek olarak tüm çevre birimleri için benzer adımları tekrar etmek gerekiyordu. Tüm bunları çözen bir modül yazmaya karar verdik. Bu modüle “Çevre Birimi Belleği” dedik. Çevre Birimi Belleği, çekirdek ile çevre birimi arasında aracılık yapmaktadır. İçerisinde şartnamede belirtilen yazmaçları tutar. Hem çekirdek hem de çevre birimi yazma/okuma işlemleri yapabilir. Modülün blok şeması aşağıdaki gibidir:



Çevre birimlerinde sık sık rastlanan bir örüntü gördük: Okunmak istenen ile yazmak istenen yazmaçların her zaman aynı olmaması. Bu yüzden çevre birimlerine çekirdeğin bağlantılarına ek olarak okuma adresi koyduk. Çevre birimi belleğinin büyük bir dezavantajı olduğunu fark ettik: Aynı adrese her iki taraftan da yazma isteği gelirse ne olacak?

Böyle bir durum söz konusu değil çünkü tüm çevre birimlerindeki yazmaçların fonksiyonları o kadar iyi belirlenmiş ki kullanım talimatlarına uyan bir yazılımcı çekirdekten çevre birimiyle çakışacak yazma isteği gönderemez. Çevre birimlerinde işlemin tamamlandığını söyleyen bir yazmaç var. Eğer işlem tamamlanmışsa/başlamamışsa, söz konusu çevre biriminin hiçbir yazmaca yazma emri vermeyeceğinin garantisini veriyoruz. Öteki taraftan, eğer işlem devam etmekte ise yazılımcı bize bunun garantisini vermelidir. Yazılımcıdan işlemi başlattıktan sonra işlemin devam etmekte olduğunu gösteren yazmacı okumadan diğer yazmaçlara yazma emri vermemesini istiyoruz. Bu anlaşma sayesinde, aradaki uyumsuzluk giderilir.

Bazı yazmaçlar çekirdek tarafından yalnızca okunabilir. Hangi yazmaçların yazılabileceği ve yazmaçların toplamının büyüklüğü çevre birimi belleğine parametre olarak verilir.

### UART (UART.v)

[BENİ DÜZELT]

### I2C Master (I2C\_master.v)

SDA ve SCL adında iki tel kullanılır. SDA çift yönlüdür. SCL, tek yönlüdür. Normalde SCL de çift yönlüdür fakat sistemde tek bir master olduğu için SCL teline yazan sadece bir tane modül vardır. I2C protokolünü özetlemek gerekirse:

* SCL 1 iken SDA 0 olursa başlangıç anlamına gelir.
* SCL 0 iken SDA 1 olursa bitiş anlamına gelir.
* Başlangıç ve bitiş arasında SCL 1 iken SDA değişemez.
* Adresler 7 bittir.
* Veriler 8 bittir.
* Sırasıyla “başlangıç + adres + okuma + ack + veri + ack/nack + bitiş” şeklindedir. 8 bitten daha fazla veri göndermek istenirse “veri + ack/nack” tekrarlanır.

Bu çevre biriminin içerisinde 8 state’den oluşan FSM bulunmaktadır. Stateler sırasıyla şu şekildedir:

1. IDLE: I2C\_CFG registerında tamamlanmamış yazma ya da okuma emri varsa START state’ine geç. Yalnızca okuma emri geldiği zaman oku (Yazma öncelikli).
2. START: 3 bitlik sayacı sıfırla. SCL teline master clock’tan oluşturulmuş senkron frekansı düşük clock sinyalini ver. Start condition oluşması için SDA teline 0 yaz. ADDR state’ine geç.
3. ADDR: I2C\_ADR registerındaki adresi SDA teli üzerinden gönder. Sayaç tamamlandığında ACK0 state’ine geç.
4. ACK0: Sayacı sıfırla. SDA teline yazmayı bırak. SDA teli 0’lanmışsa (ACK) bir sonraki state’e geç. Bir sonraki state yazma yapılacaksa WDATA yoksa RDATA’dır.
5. WDATA: I2C\_TDR registerından bir byte’lık veri gönder. Sayaç tamamlandığında ACK1 state’ine geç.
6. RDATA: SDA telinden I2C\_RDR registerına bir byte’lık veri yaz. Sayaç tamamlandığında ACK1 state’ine geç.
7. ACK1: Sayacı sıfırla. I2C\_NBY registerındaki sayı kadar (1-4 aralığına göre düzenlenmiş şekilde) önceki state’i tekrarla. Yazma modundaysak ve SDA telinden ACK sinyali gelmemişse tekrar göndermeyi dene (işlemin komple iptal edilemesine tercih edilmiştir, ileride değişiklik yapılabilir). Tekrarlar tamamlandıysa STOP state’ine geç. Okuma işlemi yapıyorsak son byte hariç ACK gönder. Son byte’ta slave’in daha fazla istemediğimizi anlaması için NACK gönder. Yazma işlemi yapıyorsak SDA teline yazmayı bırak.
8. STOP: STOP condition oluşması için SDA teline 1 yaz. SCL teline de gecikmeli 1 yaz. İşlemin tamamlandığını belirtmek için I2C\_CFG registerını düzelt. IDLE state’ine dön.

### QSPI Master (QSPI\_master.v)

QSPI’da SCLK telinden saat sinyali, CSn telinden select sinyali ve io tellerinden (4 tane) veri aktarılır. Şartname açısından QSPI protokolünü özetlemek gerekirse io tellerine yazılanlar sırasıyla şu şekildedir:

1. 8 bitlik komut gönderilir.
2. 24 bitlik adres gönderilir. [Opsiyonel]
3. Dummy cycle(’lar) gönderilir. [Opsiyonel]
4. Veri gönderilir/alınır. [Opsiyonel]

QSPI Master modülünün yükünü hafifletmek için oldukça esnek bir şekilde yazdık. QSPI Master modülünün konfigürasyonunu tamamen yazılıma bıraktık. İçerisinde 4 state’den oluşan FSM bulunmaktadır. State’ler sırasıyla şu şekildedir:

1. IDLE: Sayacı sıfırla. QSPI\_STA registerına işlemin bittiğini belirten biti yaz. io tellerinden sadece 1.sine (boş) veri yazılabilsin.
2. CMD: QSPI\_STA registerına meşgul olduğunu belirten biti yaz. X0 modunda isek IDLE state’ine geç(bitir).Dummy cycle istendiyse DUMMY state’ine geç, yoksa EXECUTE state’ine geç.
3. DUMMY: QSPI\_ADR registerındaki adresi oku. Okunan bitleri sırasıyla io bağlantısındaki 1. tele yaz. Eğer istenilen dummy cycle adresin uzunluğundan fazlaysa başa dön ve tekrar oku. İstenilen cycle kadar beklendiğinde EXECUTE state’ine geç.
4. EXECUTE: Sayaç sıfırlanınca IDLE state’ine dön.
   1. Okuma modu:
      1. x1: 4 bitlik io bağlantısının 2. telinden okuduğun veriyi sayacı göz önünde bulundurarak ilgili bölüme yaz. Sayacı 1 azalt.
      2. x2: 4 bitlik io bağlantısının ilk 2 telinden okuduğun veriyi sayacı göz önünde bulundurarak ilgili bölüme yaz. Sayacı 2 azalt.
      3. x4: 4 bitlik io bağlantısının tüm tellerinden okuduğun veriyi sayacı göz önünde bulundurarak ilgili bölüme yaz. Sayacı 4 azalt.
   2. Yazma modu:
      1. x1: 4 bitlik io bağlantısının 1. teline QSPI\_DRx registerından okuduğun veriyi sayacı göz önünde bulundurarak yaz. Sayacı 1 azalt.
      2. x2: 4 bitlik io bağlantısının ilk 2 teline QSPI\_DRx registerından okuduğun veriyi sayacı göz önünde bulundurarak yaz. Sayacı 2 azalt.
      3. x4: 4 bitlik io bağlantısının tüm tellerine QSPI\_DRx registerından okuduğun veriyi sayacı göz önünde bulundurarak yaz. Sayacı 4 azalt.

QSPI Master’da adresin yazımıyla dummy cycle’ın yazımını aynı state’de yapıyoruz. Bunun sebebi, dummy cycle’da io bağlantısının 1. teline yazılan verinin önemsiz olmasıdır. Böylece hem adresi göndermeyi hem de dummy cycle yazmayı aynı devre ile yaparak alandan tasarruf ediyoruz. Yazdığımız bu modülün içerisine komutları gömmediğimiz için yalnızca şartnamede belirtilen dasdasdadadas[BENİ DÜZELT] flash belleği ile değil, birçok bellekle uyumlu bir şekilde çalışabiliyor. Bu anlatılanlar göz önünde bulundurulduğunda QSPI\_CCR registerına atamalar şu şekilde yapılıyor:

1. QSPI\_CCR[9:8] bölümüne veri modu yazılıyor. Veri modu EXECUTE state’inde kaç tane tel kullanılacağına göre belirtiliyor. Eğer sadece komut gönderilecekse 0 yazılması gerekiyor.
2. QSPI\_CCR[15:11] bölümünü önce şartnamedeki gibi içeride yazan sayı kadar cycle diye baz aldık ama sonra fark ettik ki bu mümkün değil. Çünkü 5 bitlik yazılabilecek en yüksek sayı 31 ve dasdasdadadas[BENİ DÜZELT] flash belleğinde 24b adres + 8 dummy cycle = 32 cycle DUMMY(bizim FSM cinsinden) gerekiyor. Adres 24b ya da 32b. İki sayı da 8’e bölünüyor. Kullanıcının 8’in katı olan bir sayı kadar dummy cycle isteyeceğini baz alıp yazılan sayının 8 katı kadar DUMMY state’inde kalıyoruz. Dolayısıyla CMD ve EXECUTE state’leri arasında minimum 0, maksimum 256 cycle bekleyebilirsiniz.
3. QSPI\_CCR registerının geri kalan kısmına ek bir yorumumuz yok.

Bu açıklamalar göz önünde bulundurulduğunda şöyle bir tablo çıkıyor karşımıza:

A screenshot of a computer screen

Description automatically generated

C driver’ları (en azından header dosyaları) hazırlandı mı? Header ve Verilog kodları için github hesabınızdaki ilgili dosya pathi link verilebilir ama DTR’de peripheral tasarımı detaylıandırılmalıdır, sadece koda link verilip geçilmemelidir. Bütün peripherallar için tasarım ve doğrulama hangi aşamada, tamamlandı, şu şu fonksiyonlar eksik vs. bilgileri. Seçilen yöntem ve metotların diğerlerine göre artı eksileri? Karşılaşılan zorluklar ve çözümleri?}

### FPGA Prototipleme Detayları (10)

{FPGA prototipleme detayları. FPGA’da işlemci üzerinde yazılım koşturuldu mu?

Elimizde S25FL128S model Flash bellek olmadığı için FPGA’e gömeriz diye düşündük. Fakat Vivado boyutu çok büyük olduğu için sentezde hata verdi. Bu yüzden “defines.vh” headerına “`define NO\_FLASH” diye bir satır ekledik.

Bu satır FPGA implementasyonunda Flash belleğe önceden yazılmış olacağını varsaydığımız programı komut belleğine yazıyor. Aslında bir nevi QSPI protokolünü baypas ediyoruz. Bootloader kodu da sadece komut belleğine dallanmak için programlanıyor.

FPGA olarak Basys3 (A7-35T) kartını seçtik. Bağlantıları şu şekilde yaptık:

FPGA kaynaklarının kullanım miktarları şu şekilde:

Yazılımlarla peripherallar test edildi mi?

Timer’ı konfigüre eden ve sonrasında periyodik olarak GPIO’nun giriş portlarından veri okuyup bunu işleyerek (toplama, çıkarma, kaydırma komutları vb. şekilde yorumlayarak) GPIO’nun çıkış portlarına yazan bir program yazdık. Programı derledik ve sentez sırasında çıkan kritik uyarıları çözdük. Sonrasında implementasyon ve bitstream üretim aşamalarını başarılı bir şekilde tamamladık. Bu test sayesinde çekirdek, bus, bootloader kodunu (dallanma yapıp yapmamasını), timer modülünü, işlemcinin kesme isteklerine yanıt verip vermediğini, GPIO modülünü, linker scripti, program header’ı ve yazdığımız C kodunu doğrulamış olduk.

I2C modülünü Arduino UNO’yu slave olarak bağladıktan sonra test ederken ACK sinyali ile ilgili bir sorun yaşadık. Sorunun FPGA – Arduino uyumsuzluğu olduğunu anladık. DTR’den sonraya bıraktık.

C kodları için github hesabınızdaki ilgili dosya pathi link verilebilir ama FPGA üzerinde veya simülasyon ortamındaki çalışmalar DTR’de detaylandırılmalıdır. Alınan sonuçlar görsellerle belirtilmelidir. Karşılaşılan zorluklar ve çözümleri?}

# ÇİP TASARIM AKIŞI (15 PUAN)

{Bütün bir sistemin olmasa da peripheralların biri veya birkaçı ya da işlemci çekirdeği için çip fiziksel tasarım akışının tamamlanması beklenmektedir. Mümkünse tasarımın hepsi, tasarımın hepsi tamamlanmadıysa da tamamlanmış alt modüller üzerinde synthesis, placement ve routing gibi aşamalar sonrasında oluşan zamanlama analizi (timing analysis), kaynak kullanım (utilization) gibi dosyaların çıktılarından yararlanıp yorumlarda bulunulmalıdır. GDSII layout resimleri eklenmelidir. Güç tüketimi, 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 P & R programı tasarımı route edemediği şeklinde bir hata vermiştir, yarışmacılar da bu durumu bir yöntemle çözmüştür veya akış sırasında bir hata alınmıştır, bunu çözmek için başka bir commit’teki versiyona geçilmiştir şeklinde). Bu çözümler esnasında kullanılan teorik-genelgeçer bilgilere yer verilmelidir (Örneğin bu kaynaktan edinilen bilgiye göre utilization değeri bu değerin altında tutulmuştur). Açık kaynak programlar kullanıldıysa bahsedilen genelgeçer bilgilerle elde edilen sonuçlar uyumsuzluklar görüldüyse bunların sebebi açıklanmalıdır. Çip tasarım akışı boyunca faydalanılan topluluklar, Slack kanalları gibi yerler varsa bunların hangilerinden ne yönde yararlanıldığından bahsedilmelidir. Çip akışında geçirilen aşamaların hangilerinin daha kolay hangilerinin daha zor bulunduğuna, hangilerinde ne kadar zaman geçirildiğine kısaca değinilmelidir.}

# TEST (20 PUAN)

{Bu kısımda tasarımın RTL seviyesinde fonksiyonel simülasyonlar, CV32E40P işlemcinin doğrulanması, UART peripheral için UVM/SV ile yapılan doğrulama çalışmaları ve static timing analysis ile max frekans hesaplamaları detaylandırılmalıdır. Tüm tasarım tamamlanmasa bile hangi alt modüllerde simülasyonların nasıl gerçekleştirildiği, verification ortamı (FPGA ve çip tasarım doğrulaması için hangi verification araçları kullanıldıysa) ve testbench kodları ile bilgiler detaylıca verilmelidir.}

# TAKIM ORGANİZASYONU (3 PUAN)

## Takım Organizasyonu

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

Cengiz Emre Dedeağaç: Takımın danışmanıdır. Özyeğin Üniversitesi’nde Bilgisayar Mühendisliği bölümünde doktora yapmaktadır.

Kutay Bulun: Özyeğin Üniversitesi’nde Elektrik-Elektronik Mühendisliği bölümünde lisans öğrencisidir.

Metin Arda Köker: Özyeğin Üniversitesi’nde Bilgisayar Mühendisliği bölümünde yüksek lisans yapmaktadır.

Taha Gemici: Özyeğin Üniversitesi’nde Elektrik-Elektronik Mühendisliği bölümünde lisans öğrencisidir.

Ahmet Utku Erşahin: Özyeğin Üniversitesi’nde Elektrik-Elektronik Mühendisliği bölümünde yüksek lisans yapmaktadır.

## Görev Dağılımı

{Bu kısımda görev dağılımı ve ekip organizasyonu hakkında bilgi verilir. Kim hangi kısımlardan sorumlu?}

Kutay Bulun:

* UART modülünün UVM standardında test edilmesi

Metin Arda Köker:

* I2C Master modülünün test edilmesi
* Projenin GDSII çıktısının OpenLane’den alınması

Taha Gemici:

* Bus yapısının RTL kodunun yazılması
* I2C Master modülünün RTL kodunun yazılması
* QSPI Master modülünün RTL kodunun yazılması
* Timer modülünün RTL kodunun yazılması
* GPIO modülünün RTL kodunun yazılması
* ASIC/FPGA Clock Generator modülünün RTL kodunun yazılması
* Çevre Birimi Belleğinin RTL kodunun yazılması
* QSPI Master modülünün test edilmesi
* Timer modülünün test edilmesi
* GPIO modülünün test edilmesi
* ASIC/FPGA Clock Generator modülünün test edilmesi
* Çevre Birimi Belleğinin test edilmesi
* FPGA implementasyonunun yapılması

Ahmet Utku Erşahin:

* UART modülünün RTL kodunun yazılması
* JTAG modülünün RTL kodunun yazılması
* Açık-kaynak USB kodu bulmak

Bu görevlerden hangilerinin tamamlandığı zamana bağlı olarak değişmektedir. Şu linkten güncel olarak takip edebilirsiniz: [BENİ DÜZELT]

# İŞ PLANI ve RİSK PLANLAMASI (5 PUAN)

{Bu kısımda projenin FPGA ve çip 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.}

# KAYNAKÇA (2 PUAN)

{Bu bölümde raporda kullanılan kaynaklar yer almalıdır. Kaynaklar rapor içerisinde refere edilmelidir.}

|  |
| --- |
| **RAPOR TASLAKLARI İLE İLGİLİ NOT:** |
| **Bu belge Çip Tasarım Yarışması Mikrodenetleyici Kategorisi Detay Tasarım Raporu (DTR) şablonu olarak hazırlanmıştır.**  **Rapor, Çip Tasarım Yarışması Şartnamesi Bölüm 3.2.2’de belirtilen kurallara uygun olarak hazırlanmalıdır.**  **Rapor dili anlaşılırlığına, yazım düzenine, rapor içeriğine ve rapor bütünlüğüne dikkat edilmelidir.**  **Grafikler, resimler, devre şemaları vb. okunaklı olmalıdır. Kullanılan resimler mümkün olduğunca yüksek kaliteli olarak eklenmelidir.**  **Raporlar, A4 formatında, 11 punto, Calibri fontunda, satır aralığı 1.15 ve alt-üst ve yan kenarları 2.5 cm olacak şekilde hazırlanmalıdır. Detay tasarım raporu en fazla 30 sayfa olmalıdır (kapak sayfası, resimler, tablolar, referanslar dâhil).**  **Minimum rapor içeriği yukarıdaki gibi olmalıdır, başlıklar mümkün olduğunca değiştirilmemelidir. Gerek görülürse eklemeler yapılabilir.**  **Gereksiz uzun ifadeler yerine kısa ve anlaşılır ifadeler tercih edilmelidir.**  **“If you can’t explain something in simple terms, you don’t understand it” Feynman**  **Toplam sayfa sayısı 30’u geçmemelidir. Sayfa sınırı aşıldığı takdirde ilk 30 sayfa dikkate alınacaktır.** |