

**THE IMAGINATION UNIVERSITY PROGRAMME**

**RVfpga Deney 10**

**Dizisel Veri Yolları**

# GİRİŞ

Bu deneyde ilk olarak dizisel veri yollarının nasıl çalıştığını, günümüzde kullanılan en yaygın dizisel veri yollarından SPI veri yolunun ana niteliklerini tanımlıyoruz (Bölüm 2). Ardından Nexys A7 kartındaki erişilebilir SPI ivmeölçerine odaklanıyoruz: bu çevre biriminin yüksek-düzeyli standardını çözümleyip temel alıştırmalar önerip (Bölümler 3 ile 4) ardından alçak-düzeyli gerçekleştirmesini çözümleyip daha ileri düzey alıştırmalar öneriyoruz (Bölümler 5 ile 6).

# DİZİSEL VERİ YOLLARI – SPI VERİ YOLU

# Paralel veri yolları bir iletide birden çok bit yollarken dizisel veri yolları bir iletide bir bit yollar. İlk olarak bu iki iletişim şemasını karşılaştırıp ardından yaygın kullanılan dizisel veri yollarından biri olan SPI (dizisel çevre birimi arayüzü) protokülünü tanımlıyoruz. Bu önemli iletişim protokolüyle ilgili bilgini genişletmek için internette çokça bilgi bulabilirsin.

# Önceki deneylerde gösterildliği gibi gömülü elektroniklerin ana amacı istenen işlevleri oluşturmak için işlemcilerle devreleri bağlamaktır. İşlemcilerle devrelerin bilgi paylaşması için ortak bir iletişim protokolleri olmalıdır. Bu veri alışverişini yapabilmek için yüzlerce iletişim protokolü tanımlanmıştır, genel olarak iki ana kategoriye ayrılırlar: paralel ya da dizisel arayüzler.

Paralel arayüzler çok biti paralel olarak, bir diğer deyişle aynı anda, aktarır. Verinin veri yollarına (birden çok kablo) gerek duyarlar. Örneğin protokol bir kezde sekiz, onaltı, ya da daha çok biti iletebilir (Figür 1’e göz at). N veri bitinin yeni grupları aktarıma hazır olunca zamanlayabilmek için bir saate de gerek duyarlar.



Figür 1. Paralel 8-bit veri yolu örneği.

Paralel iletişimle karşılaştırıldığında dizisel arayüzler verilerinin akışını bir iletide bir bit olacak biçimde sağlarlar. Bu arayüzler yalnızca bir kabloyla bile çalışabilirken genelde dörtten çok kabloya yer verilmez. Figür 2 veri için bir kablo, saat için bir kablo barındıran bir dizisel arayüz örneğini gösterir. Yeni bir saat kenarında yeni bir veri biti aktarılır.



Figür 2. Dizisel 1-bit veri yolu örneği.

Paralel iletişimin hızlı olma, anlaşılırlık, göreceli kolay gerçekleştirme gibi artıları vardır. Ancak daha çok girdi/çıktı (I/O) doğrusu gerektirir. Dolayısıyla, uçlar kısıtlı olduğundan, gömülü sistemler genelde dizisel iletişimi seçer, böylelikle de potansiyel hız yerine az uç kullanma yoluna girerler.

**SPI Veri Yolu:**

Dizisel Çevre Birimi Arayüzü (SPI) protokolü sensörler, ADCler, DACler, kaydırma yazmaçları, SRAM, gibi çevre birimi ICleri ile mikrodenetleyici arasında en yaygın kullanılan arayüzlerden biridir. SPI, denetleyici-çevre birimi (eskiden usta-köle denirdi) iletişimi üzerine kurulu senkron, full duplex bir arayüzdür.

SPI veri yolu 4 port üzerinden iletişim kurar (Figür 3’e göz at):

* **SDO** – Dizisel Veri Dışarı: Denetleyicinin çevre birimi aygıta çıktısı
* **SDI** – Dizisel Veri İçeri: Denetleyicinin çevre birimi aygıttan girdisi
* **SCK** – Dizisel Saat: Denetleyiciden çevre birimi aygıta yollanır
* **CS** – Yonga Seçici: Etkin alçak sinyal; Denetleyici sinyali (çevre birimi seçiliyken 0) çevre birimine yollar

**Not:** eskiden SDO’ya MOSI de denirdi (usta veri dışarı, köle veri içeri), SDO’ya MISO da denirdi (usta veri içeri, köle veri dışarı). Bu kullanımlar güncel olmayıp hakaret gibidir, ancak yine de literatürle belgelendirmede günümüzde de bulunurlar.



Figür 3. Bir SPI denetleyici ile bir SPI çevre birimi bir sistem örneği.

Dizisel veri yolu yükselen ya da düşen saat kenarına senkronlanır. SPI bir full-duplex arayüzdür; denetleyici ile çevre birimi veriyi SDO ile SDI doğrularında aynı anda iletebilir. SPI arayüzlerinde yalnızca bir denetleyici vardır ancak birden çok çevre birimleri olabillir. Birden çok çevre birimi bağlanınca erişilen çevre birimini seçmek için alçakta onaylanan çoklu yonga seçme sinyalleri (CSbar) kullanılır. SDO ile SDI dizisel veri doğrularıdır: SDO (dizisel veri dışarı) denetleyiciden çevre birimine çıktı veri iken SDI (dizisel veri içeri) çevre biriminden denetleyiciye girdi veridir.

SPI iletişimini başlatmak için denetleyicinin çevre birimini seçip (CSbar sinyalini sağlayarak, bir diğer deyişle, CSbar = 0) ardından saat sinyalini çevre birimine yollamalıdır. SPI iletişimi sırasında aynı anda veri denetleyiciden SDO ile SDI sinyallerine, sinyallerden ise denetleyiciye iletilir. Dizisel saat (SCK) kenarı verinin örneklenmesini senkronize eder.

SPI arayüzü saatin boşta durumunu seçme, sinyali örnekleme fazı için CPOL ile CPHA ek sinyalleri de sağlar. Saat ucaylılık (CPOL) sinyali saat (SCK) 0’da boştayken 0, 1’de boştayken 1 olur. Saat fazı (CPHA) sinyali veriyi yollayıp örnekleme için saat fazını seçer. CPHA = 0 iken veri (SDI ya da SDO) öncü kenarda örneklenir (bir diğer deyişle SCK boşta olmayı bırakınca gelen ilk kenar ile sonrasındaki bütün dönümlerde); yani veri (SDI ile SDO) arka kenarda değişmeli, Figür 4’teki zamanlama diyagramlarındaki gibi. CPHA = 0 tam tersini yapar: veri arka kenarda örneklenip öncekü kenarda değişir, Figür 4’ün alt iki figüründe gösterildiği gibi. Yeni verinin iletildiği kenara dizisel iletişim genelde kaydırma yazmacı kullanılarak gerçekleştirildiği için *kayan kenar* da denir.

Bu deneyde kullandığımız SPI arayüzü CPHA = 0 ile CPOL = 0 olur, yani SCK düşük boştadır, denetleyiciyle çevre birimi veriyi yükselen kenarda örnekleyip yeni veriyi doğruya (SDO ya da SDI) düşen kenarlardan sonra kaydırır, Figür 4’ün üst zamanlama diyagramındaki gibi. Önemli olarak SCK boşta olup yükselmeden önce SDO ile SDI sonraki veri baytının yüksek önemli bitini taşımalıdır.



**Figür 4. CPHA/CPOL’ün veri örnekleme/yollamayla ilişkisi**

# SPI İVMEÖLÇER: YÜKSEK DÜZEYLİ SPESİFİKASYON

Çoğu çevre birimi bir SPI arayüzü içerir. Örneğin Nexys A7 kartındaki ivmeölçerin bir SPI arayüzü vardır. Bu bölümde RVfpga’in SPI denetleyicisinin yüksek düzeyli spesifikasyonunu tanımlayıp Nexys A7 kartındaki ADXL362 ivmeölçeri tanıtıyoruz. Ayrıca ivmeölçeri kullanan bir alıştırma da tanıtıyoruz.

1. **SPI denetleyici spesifikasyonu**

RVfpga’in SPI modülü OpenCores’tandır (<https://opencores.org/projects/simple_spi>). Pkaeti indirirsen modülün yüksek düzeyli spesifikasyonunu tanımlayan bir belge sağlanır. Bu belge şurada da sağlanır:

*[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals/spi/docs/simple\_spi.pdf*

SPI modülünün ana işlemiyle özelliklerini özetliyoruz; ancak ek bilgi için yukarıdaki belgeye göz at.

Bu modülün şu ana özellikleri vardır:

* Motorola’nın SPI spesifikasyonlarıyla uyumludur
* 8-bit WISHBONE RevB.3 klasik arayüzü kullanır
* Bir 4-entry read FIFO buffer ile bir 4-entry write FIFO buffer barındırır
* 1, 2, 3, ya da 4 aktarılmış bayttan sonra kesinti oluşumuna olanak sağlar
* Girdi saat sıklıklarının geniş bir aralğıyla işlem görebilir
* Bütünüyle sentezlenebilir

SPI çekirdek spesifikasyonunun Bölüm 3’ü SPI modülünün içerisindeki denetleme, durum yazmaçlarını tanımlar, ki bunların hepsi ayrı yazmaçlara atanmıştır (Tablo 1’e göz at). RVfpga’in SPI denetleyicisinin taban adresi **0x80001100**’dır. Bu yazmaçlar aşağıda detaylıca tanımlanır.

Tablo 1. SPI Yazmaçları

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Adı** | **Adresi** | **Genişliği** | **Erişimi** | **Tanımı** |
| SPCR | 0x80001100 | 8 | R/W | Control register |
| SPSR | 0x80001108 | 8 | R/W | Status register |
| SPDR | 0x80001110 | 8 | R/W | Data register |
| SPER | 0x80001118 | 8 | R/W | Extensions register |
| SPCS | 0x80001120 | 8 | R/W | CS register |

SPI Denetleme Yazmacı (SPCR) SPI modülünü denetler; Tablo 2 bitlerinin işlevlerini gösterir.

Tablo 2. SPCR bitleri

|  |  |  |
| --- | --- | --- |
| **Bit** | **Erişim** | **Adı & Tanımı** |
| 0:1 | R/W | **SPR**  SPI clock Rate: These bits select the SPI clock rate. |
| 2 | R/W | **CPHA**  Clock Phase: Determines the phase of sampling and sending data. When CPHA = 1, new data is shifted onto the wire at the leading edge and data is sampled on the trailing edge. When CPHA = 0, new data is shifted onto the wire at the trailing edge and sampled on the leading edge. |
| 3 | R/W | **CPOL**  Clock Polarity: Determines idle state of SPI clock (SCK). When CPOL = 0, SCK idles at 0, when CPOL = 1, SCK idles at 1. |
| 4 | R/W | **MSTR**  Mode Select: When MSTR = 1, the SPI core is a controller device. This is the only supported mode for this controller. |
| 6 | R/W | **SPE**  SPI Enable: When SPE = 1, the SPI core is enabled. When it is cleared (SPE = 0), the SPI core is disabled. |
| 7 | R/W | **SPIE**  SPI Interrupt Enable: When SPIE = 1, when the SPI Interrupt Flag in the status register is set, the host is interrupted. |

SPI Durum Yazmacı (SPSR) SPI modülünün durumunu sağlar; Tablo 3 bitlerinin işlevini gösterir.

Tablo 3. SPSR bitleri

|  |  |  |
| --- | --- | --- |
| **Bit** | **Access** | **Description** |
| 0 | R/W | **RFEMPTY**  Read FIFO Empty: If RFEMPTY = 1, the read FIFO is empty. |
| 1 | R/W | **RFFULL**  Read FIFO Full: If RFFULL = 1, the read FIFO is full. |
| 2 | R/W | **WFEMPTY**  Write FIFO Empty: IF WFEMPTY = 1, the write FIFO is empty. |
| 3 | R/W | **WFFULL**  Write FIFO Full: IF WFFULL = 1, the write FIFO is full. |
| 6 | R/W | **WCOL**  Write Collision flag: When WCOL = 1, the SPDATA register was written to while the Write FIFO was full. Writing a 1 to WCOL clears this bit. |
| 7 | R/W | **SPIF**  SPI Interrupt Flag: SPIF = 1 upon completion of a transfer block. If SPIF is asserted (‘1’) and SPIE is set, an interrupt is generated. Writing a 1 to SPIF clears it. |

SPI Veri Yazmacı (SPDR) okunacak ya da yazılacak veriyi sağlar. SPI denetleyicisi 4 x 8-bit Write Buffer ile bir 4x 8-bit Read Buffer içerir.

SPI Genişletilmiş Yazmacı (SPER) ek işlevsellik sağlar; Tablo 4 barındırdığı değişik alanları tanımla.

Tablo 4. SPER bitleri

|  |  |  |
| --- | --- | --- |
| **Bit** | **Erişim** | **Tanım** |
| 0:1 | R/W | **ESPR**  Extended SPI Clock Rate Select: Add two bits to the SPR (SPI Clock Rate Select). |
| 6:7 | R/W | **ICNT**  Interrupt Count: Determine the transfer block size. The SPIF bit is set after ICNT transfers. Thus, it is possible to reduce kernel overhead due to reduced interrupt service calls. |

Son olarak, SPI Yonga Seç (SPCS) yazmacı seçilecek çevre birimini seçer. Bu sinyalin genişliği SS\_WIDTH (SPI Seç Genişliği) parametresiyle yapılandırılabilir. RVfpga’de SPI arayüzü başına yalnızca bir çevre birimi vardır, yani SS\_WIDTH = 1.

**GÖREV:** SPCR, SPSR, SPDR, SPER, SPCS yazmaçlarının SPI modülündeki bildirileriyle adreslerinin tanımının yerini bul. SPI modülü *[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals/spi* klasörünün içindedir.

R

1. **ADXL362 ivmeölçer spesifikasyonu**

Nexys A7 kartı, bir Analog Devices ADXL362 ivmeölçeri içerir. Aygıtın bütün bilgilerini veri metninde şuradan bulabilirsin:

<https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL362.pdf>

ADXL362 bir 3-eksenli MEMS ivmeölçeridir, 100Hz veri çıktı oranında 2μA’den az tüketim yapar, hareket tetikli uyandırma modunda 270 nA tüketim yapar. 12-bit çıktı çözünürlüğü sağlar, ancak daha düşük çözünürlük de yeterli olduğunda daha verimli bir-bayt aktarımlar için 8-bit formatlı veri de sağlanır. ±2 g, ±4 g, ±8 g ölçüm aralıkları ±2 g aralığında 1 mg/LSB çözünürlüğünde erişilebilirdir. ADXL362 Ölçüm Modundayken ivme verisini sürekli olarak ölçüp X-veri, Y-veri, Z-veri yazmaçlarında depolar.

ADXL362 ivmeölçer kullanıcının yapılandırıp ivmeölçer verisini okumasını sağlayan birkaç yazmaç (Tablo 5) içerir. Aygıt denetleme yazmaçlarına yazılarak yapılandırılıp, ivmeölçer verisi aygıt yazmaçları okunarak bulunur. Aygıtla bütün iletişimler bir yazmaç adresi ile iletişimin okuma ya da yazma olduğunu belirten bir bayrak belirlemelidir. Veri aktarımı yazmaç adresiyle iletişim bayrağı aygıta gönderildikten sonra gerçekleşir.

Bu ivmeölçer bir SPI iletişim şeması kullanarak bir çevre birimi aygıtı olarak davranır. FPGA ile ivmeölçer arasındaki arayüz Figür 5’te gösterilmiştir.

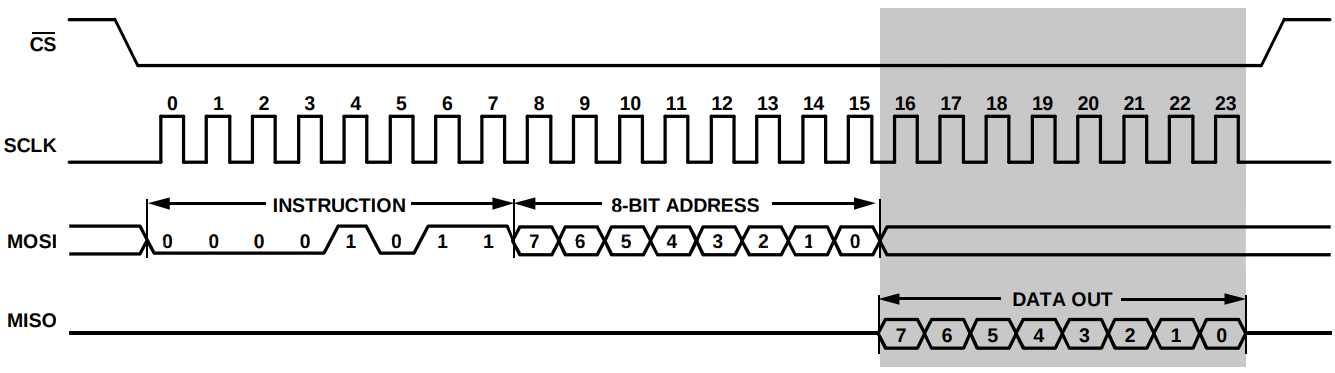
# 

Figür 5. ADXL362 İvmeölçerinin Nexys A7 kartı ile arayüzü

Önerilen SPI saat sıklığı 1-5 MHz aralığındadır. SPI, SPI mod 0’da işlem görür (CPOL = 0 ile CPHA = 0). SPI portu, ilk baytın iletişimin bir yazmaç okuma (0x0B) ya da yazmaç yazma (0x0A) olduğunu göstediği bir çok baytlı yapı kullanır:

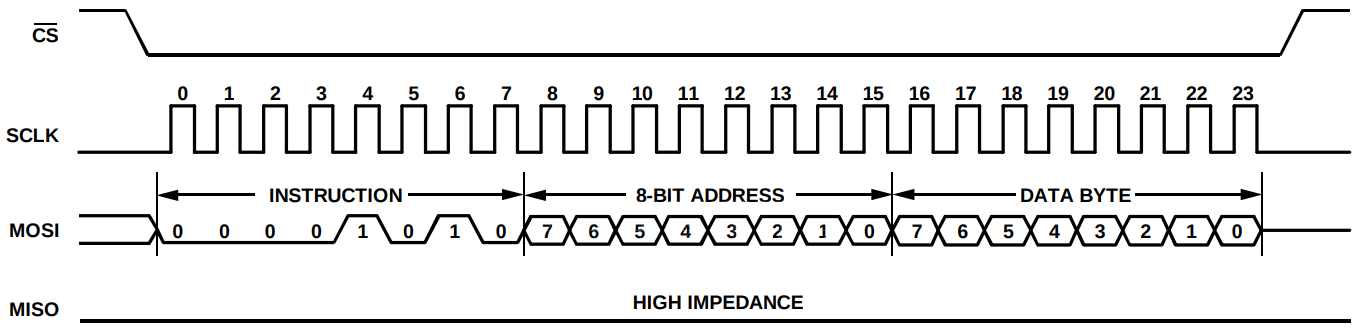
**<CS down> <Write/Read (0x0A/0x0B)> <address byte> <data byte> <CS up>**

Figür 6 ile Figür 7 SPI denetleyici (denetleyici) ile ivmeölçer (çevre birimi) arasındaki iletişimi iki örnekle görselleştirir: Figür 6 yazmaç okumayı gösterir, Figür 7 yazmaç yazmayı gösterir.



Figür 6. Yazmaç oku

(Figür şuradan <https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL362.pdf>)



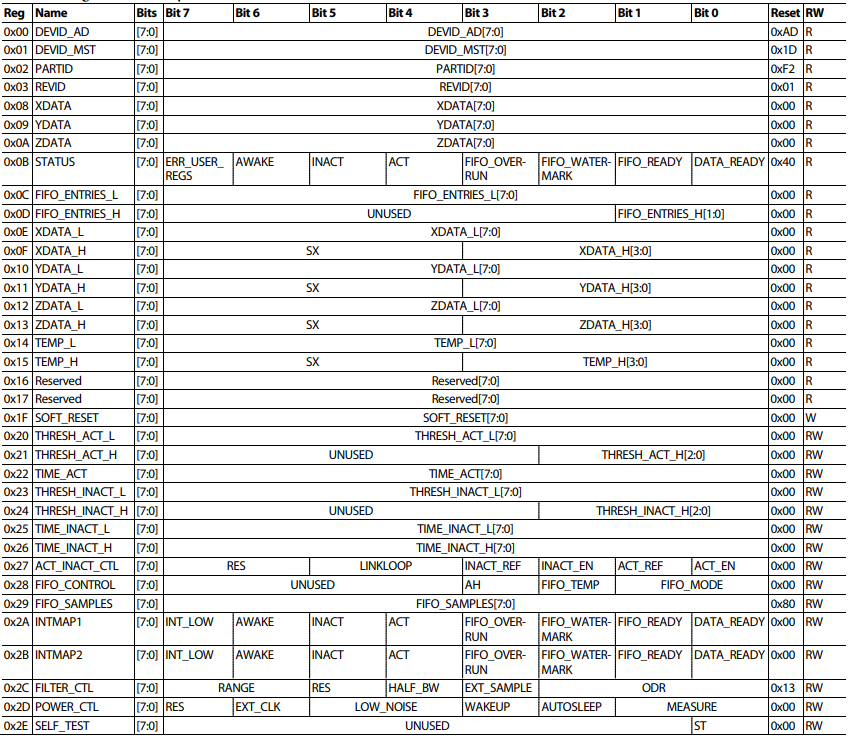
Figür 7. Yazmaça yazma

(Figür şuradan <https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL362.pdf>)

Tablo 5 ADXL362 ivmeölçerindeki yazmaçları gösterir. Yazmaçların eksiksiz tanımı için ADXL362 bilgi metnine bak: <https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL362.pdf>.

Tablo 5. ADXL362 ivmeölçer yazmaçları

(Tablo şuradan <https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL362.pdf>)



# TEMEL ALIŞTIRMALAR

**Alıştırma 1.** X-eksen, Y-eksen, Z-eksen ivme verisinin yüksek önemli sekiz bitini okuyup bu değerleri 8-sayı 7-Kesim Ekranlarda gösteren bir RISC-V çevirici programı oluştur. Yapılandırma, yazmaç bilgisi için Bölüm B’ye bak. RVfpga’in SPI modülüne erişmek için şu alt yordamları kullan. Alt yordamları kullanmadan önce RVfpga’in SPI modülüyle ilgili Bölüm A’daki bilgiler üzerine anlamaya çalış. Alt yordamların kısa özeti:

* + İşlev spiInit: SPI modülünü ilk değerlendirir.
  + İşlev spiCS: CS durumunu SPCS yazmacına gönder.
  + İşlev spiCSUp: CS Doğrusunu yükseğe indir, spiCS alt yordamını çalıştırarak.
  + İşlev spiCSDown: CS Doğrusunu düşüğe indir, spiCS alt yordamını çalıştırarak.
  + İşlev spiSendGetData: SPI’dan bayt yollayıp çevre birimi verisini al.

|  |
| --- |
| # Register addresses for SPI Peripheral  #define SPCR 0x80001100  #define SPSR 0x80001108  #define SPDR 0x80001110  #define SPER 0x80001118  #define SPCS 0x80001120 |

|  |
| --- |
| # Function: Initialize SPI peripheral  # call: by call ra, spiInit  # inputs: None  # outputs: None  # destroys: t0, t1  **spiInit**:  li t1, SPCR # control register  li t0, 0x53 # 01010011 no ints, core enabled, reserved, controller, cpol=0, cha=0, clock divisor 11 for 4096  sb t0, 0(t1)  li t1, SPER # extension register  li t0, 0x02 # int count 00 (7:6), clock divisor 10 (1:0) for 4096  sb t0, 0(t1)  ret |

|  |
| --- |
| # Function: Pull CS Line to either high or low - Provides quick calls spiCSUp and spiCSDown  # call: by call ra, spiCS  # inputs: CS status in a0 (0 is low, 1 is high)  # outputs: None  # destroys: t0  **spiCS**:  li t0, SPCS # CS register  sb a0, 0(t0) # Send CS status  ret  **spiCSUp**:  li a0, 0x00  j spiCS  **spiCSDown**:  li a0, 0xFF  j spiCS |

|  |
| --- |
| # Function: Send byte through SPI and get the peripheral data back  # call: by call ra, spiSendGetData  # inputs: data byte to send in a0  # outputs: received data byte in a1  # destroys: t0, t1  **spiSendGetData**:  internalSpiClearIF: # internal clear interrupt flag  li t1, SPSR # status register  lb t0, 0(t1) # clear SPIF by writing a 1 to bit 7  ori t0,t0,0x80  sb t0, 0(t1)  internalSpiActualSend:  li t0, SPDR # data register  sb a0, 0(t0) # send the byte contained in a0 to spi  internalSpiTestIF:  li t1, SPSR # status register  lb t0, 0(t1)  andi t0, t0, 0x80  li t1, 0x80  bne t0,t1,internalSpiTestIF # loop while SPSR.bit7 == 0. (transmission in progress)  internalSpiReadData:  li t0, SPDR # data register  lb a1, 0(t0) # read the message from SPI  ret |

# ALÇAK DÜZEYLİ GERÇEKLEŞTİRME

1. **SPI İvmeölçer alçak düzeyli gerçekleştirme**

Bu deneyin ilk bölümünde RVfpga’in SPI modüllerinin nasıl kullanılacağını gösterdik, bu son bölümde ise SPI modülünün RVfpga’de nasıl gerçekleştirildiğini tanımlayacağız. Önceki deneylerdeki formattaki gibi SPI denetleyicisinin çözümlemesini üç faza bölüyoruz:

1. SoC ile ivmeölçer arasındaki fiziksel bağlantı (Figür 8’de sol taralı bölge)
2. RVfpga Sistem Denetleyicisinde içerilen SPI denetleyicisinin entegrasyonu (Figür 8’de orta taralı bölge)
3. SPI denetleyicisi ile SweRV EH1 Çekirdeği arasındaki bağlantı (Figür 8’de sağ taralı bölge)

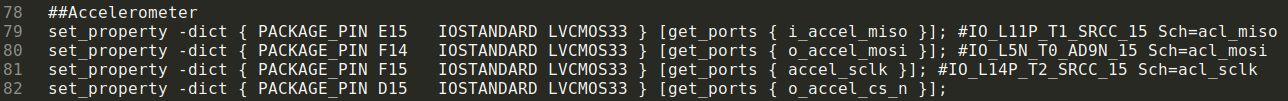


Figür 8. RVfpga’e entegre SPI denetleyicisi

INSTRUCT

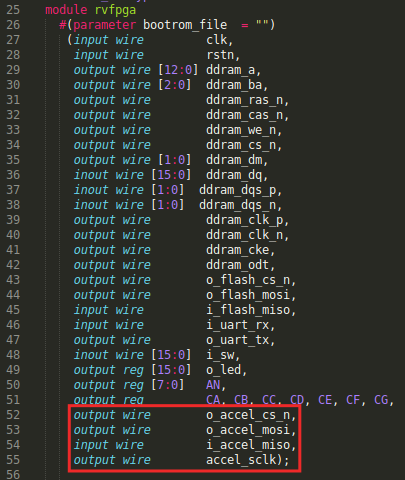
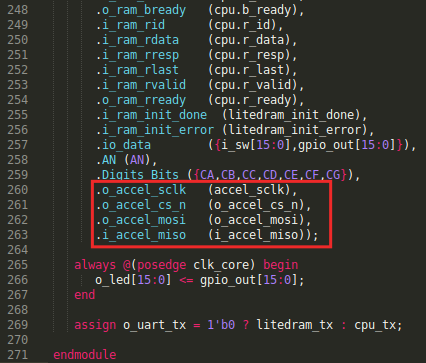
1. **İvmeölçer ile SoC’nin fiziksel bağlantısı**

Diğer çevre birimlerindeki gibi RVfpga kısıtlandırma dosyası ivmeölçere bağlantıları içermelidir. Projenin kısıtlandırma dosyaları (*[RVfpgaPath]/RVfpga/src/rvfpga.xdc*) girdi/çıktı SoC sinyalleriyle kart aygııtları arasındaki bağlantıyı tanımlar. İvmeölçerin dört ucunu SoC ile bağlayan sinyallerin adları şunlardır: *o\_accel\_cs\_n*, *o\_accel\_mosi* (SDO sinyaline denktir), *i\_accel\_miso* (SDI sinyaline denktir), *accel\_sclk*. Önemli olarak bu sinyaller çağ dışı adlar kullanır, ancak OpenCores’un SPI modülüyle tutarlı olmak için bu adları kullanıyoruz (bu modülün RVfpga’de somutlamasını Figür 11’de görebilirsin). Figür 9 bu 4 bağlantının tanımlandığı Verilog kodunu gösterir.



Figür 9. SoC ile ivmeölçerin bağlantısı (dosya *rvfpga.xdc*).

RVfpga sisteminin üst modülünün 52-55 satırlarında (bir diğer deyişle **RVfpga** modülü) SoC’ye bağlanan dört sinyali görebilirsin (Figür 10’un solu), o modülün sonunda ise **swervolf\_core** modülüyle bağlantılarını (Figür 10’un sağı).

Figür 10. İvmeölçerin üst modülle bağlantısı (dosya *rvfpga.sv*).

**GÖREVLER:** Bu dört sinyali kısıtlar dosyasından SweRVolf SoC modülüne izle (*o\_accel\_cs\_n*, *o\_accel\_mosi*, *i\_accel\_miso*, *accel\_sclk*). Şu dosyalara bakman gerekecek:

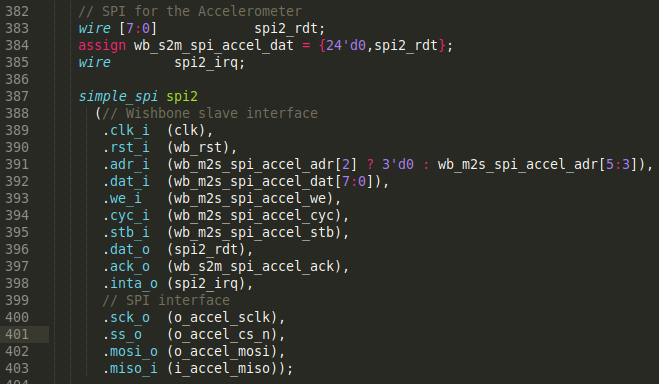
*[RVfpgaPath]/RVfpga/src/rvfpga.xdc*

*[RVfpgaPath]/RVfpga/src/rvfpga.sv*

*[RVfpgaPath]/RVfpga/src/SweRVolfSoC/swervolf\_core.v*I

1. **SPI2-İvmeölçer modülünün SoC’de entegrasyonu**

**swervolf\_core** (*[RVfpgaPath]/RVfpga/src/SweRVolfSoC/swervolf\_core.v*) modülünün 387-403 arası satırlarında ivmeölçer için SPI modülü somutlanır (Figür 11’e göz at).



Figür 11. Zamanlayıcı modülün entegrasyonu (dosya *swervolf\_core.v*).

Çevre birimlerinde hep olduğu gibi, modülün arayüzü iki bloğa bölünebilir: Wishbone sinyalleri (Tablo 6) ile dış I/O sinyalleri (Tablo 7). Wishbone sinyalleri SweRV EH1 çekirdeğinin ADC ile SPI protokolü kullanarak iletişim kurmasını sağlar.

Tablo 6. Wishbone Sinyalleri

|  |  |  |  |
| --- | --- | --- | --- |
| **Port** | **Genişlik** | **Yön** | **Tanım** |
| cyc\_i | 1 | Inputs | Indicates valid bus cycle (core select) |
| adr\_i | 15 | Inputs | Address inputs |
| dat\_i | 32 | Inputs | Data inputs |
| dat\_o | 32 | Outputs | Data outputs |
| sel\_i | 4 | Inputs | Indicates valid bytes on data bus (during valid cycle it must be 0xf) |
| ack\_o | 1 | Output | Acknowledgment output (indicates normal transaction termination) |
| err\_o | 1 | Output | Error acknowledgment output (indicates an abnormal transaction termination) |
| rty\_o | 1 | Output | Not used |
| we\_i | 1 | Input | Write transaction when asserted high |
| stb\_i | 1 | Input | Indicates valid data transfer cycle |
| inta\_o | 1 | Output | Interrupt output |

Tablo 7. Dış I/O Sinyalleri

|  |  |  |  |
| --- | --- | --- | --- |
| **Port** | **Genişlik** | **Yön** | **Tanım** |
| miso\_i | 1 | Input | Controller data Input - Peripheral data Output |
| mosi\_o | 1 | Output | Controller data Output - Peripheral data Input |
| ss\_o | 1 | Output | Chip Select |
| sck\_o | 1 | Output | System clock |

Figür 11’de gösterildiği gibi, Wishbone veri yolu sinyalinde çekirdeğin sağladığı adresin [5:2] bitleri (*wb\_m2s\_spi\_accel\_adr[5:2]*) erişilebilir 5 SPI yazmacı arasından birini seçmek için kullanılır (Tablo 1).

1. **SPI Denetleyici ile SweRV EH1 Çekirdek arasındaki bağlantı**

Önceki deneylerde açıklandığı gibi aygıt denetleyicileri SweRV EH1 çekirdeğine bir çoklayıcı ile köprü üzerinden bağlıdır (Figür 8). 7:1 çoklayıcı (Figür 12) *[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect/wb\_intercon.v* dosyasında gerçekleştirilip, *[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect/wb\_intercon.vh*dosyasının 104-205 satırlarında somutlaması yapılır. İkinci dosya **swervolf\_core** modülünün satır 168’inde içerilir, dosya şuradadır: *[RVfpgaPath]/RVfpga/src/SweRVolfSoC/swervolf\_core.v*.



Figür 12. 7-1 çoklayıcı CPU’ya bağlanacak çevre birimini seçer (*wb\_intercon.v*).

Çoklayıcı okunacak ya da yazılacak çevre birimini seçer, CPU’yu (*wb\_io\_\** sinyalleri –Figür 12’nin 115-126 satırları) bir çevre biriminin Wishbone Veri Yolu ile bağlar (Figür 12’nin 127-138 satırları), adresteki bilgiye göre (110-111 satırları). Örneğin, eğer CPU’nun oluşturduğu adres 0x80001100-0x8000113F aralığında ise, ivmeölçer modülü seçilip *wb\_io\_\** sinyalleri *wb\_spi\_accel\_\** sinyallerine bağlanır.

# İLERİ DÜZEY ALIŞTIRMALAR

**Alıştırma 2.** Evrensel Eş Zamansız Alıcı Verici (UART) bir eş zamansız (asenkron) iletişim protokolüdür. RVfpga yalın tasarımında bir UART modülü içerir (Figür 8), spesifikasyonunu şuradan bulabilirsin: *[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals/uart/docs/UART\_spec.pdf*

Bölüm A’da SPI İvmeölçer için yaptığımız gibi bu modülün alçak düzeyli gerçekleştirmesini çözümle.

Ardından dizisel port üzerinden komut istemcine bir mesaj yazdıran bir RISC-V çevirici programı oluştur. RVfpga’in UART modülüne erişmek için şu alt yordamları kullan. Alt yordamları kullanmadan önce anlamaya çalış. Alt yordamların kısa özetleri:

* + İşlev uartInit: UART modülünü ilk değerlendirir.
  + İşlev uartSendByte: UART üzerinden bayt yolla.
  + İşlev uartSendString: UART üzerinden karakter dizisi yolla.

|  |
| --- |
| # Register addresses for Uart Peripheral  # ----------------------------------------------------------------------------  #define CONSOLE\_ADDR 0x80001008  #define HALT\_ADDR 0x80001009  #define UART\_BASE 0x80002000  #define REG\_BRDL (4\*0x00) /\* Baud rate divisor (LSB) \*/  #define REG\_IER (4\*0x01) /\* Interrupt enable reg. \*/  #define REG\_FCR (4\*0x02) /\* FIFO control reg. \*/  #define REG\_LCR (4\*0x03) /\* Line control reg. \*/  #define REG\_LSR (4\*0x05) /\* Line status reg. \*/  #define LCR\_CS8 0x03 /\* 8 bits data size \*/  #define LCR\_1\_STB 0x00 /\* 1 stop bit \*/  #define LCR\_PDIS 0x00 /\* parity disable \*/  #define LSR\_THRE 0x20  #define FCR\_FIFO 0x01 /\* enable XMIT and RCVR FIFO \*/  #define FCR\_RCVRCLR 0x02 /\* clear RCVR FIFO \*/  #define FCR\_XMITCLR 0x04 /\* clear XMIT FIFO \*/  #define FCR\_MODE0 0x00 /\* set receiver in mode 0 \*/  #define FCR\_MODE1 0x08 /\* set receiver in mode 1 \*/  #define FCR\_FIFO\_8 0x80 /\* 8 bytes in RCVR FIFO \*/ |

|  |
| --- |
| .section .data  welcome:  .string "\nHELLO WORLD !!!\n" |

|  |
| --- |
| # Function: Initialize UART peripheral  # call: by call ra, uartInit  # inputs: None  # outputs: None  # destroys: t0, t1  # ----------------------------------------------------------------------------  uartInit:  li t0, UART\_BASE  /\* Set DLAB bit in LCR \*/  li t1, 0x80  sb t1, REG\_LCR(t0)  /\* Set divisor regs \*/  li t1, 27  sb t1, REG\_BRDL(t0)  /\* 8 data bits, 1 stop bit, no parity, clear DLAB \*/  li t1, LCR\_CS8 | LCR\_1\_STB | LCR\_PDIS  sb t1, REG\_LCR(t0)  li t1, FCR\_FIFO | FCR\_MODE0 | FCR\_FIFO\_8 | FCR\_RCVRCLR | FCR\_XMITCLR  sb t1, REG\_FCR(t0)  /\* disable interrupts \*/  sb zero, REG\_IER(t0)  ret |

|  |
| --- |
| # Function: Send byte through UART  # call: by call ra, uartSendByte  # inputs: a0, byte to be sent  # outputs: None  # destroys: t0, t1  # ----------------------------------------------------------------------------  uartSendByte:  li t1, UART\_BASE  /\* Check for space in UART FIFO \*/  lb t0, REG\_LSR(t1)  andi t0, t0, LSR\_THRE  beqz t0, uartSendByte  sb a0, 0(t1)    ret |

|  |
| --- |
| # Function: Send string through UART (terminated by \0)  # call: by call ra, uartSendString  # uses: uartSendByte  # inputs: a0, address of first character of string to be sent  # outputs: None  # destroys: t0, t1, t2  # ----------------------------------------------------------------------------  uartSendString:  li t1, UART\_BASE  add t2,zero,ra # save caller address  add a1,zero,a0 # use a1 as index  /\* Load first byte \*/  lb a0, 0(a1)  internalNextChar:  call ra, uartSendByte  addi a1, a1, 1  lb a0, 0(a1)  bne a0, zero, internalNextChar  add ra,zero,t2 # restore caller address  ret |

**Alıştırma 3.** Bir diğer yaygın iletişim protokolünün adı I2C’dir. Nexys A7 kartındaki sıcaklık sensörü bu protokolü kullanır. RVfpga’i bir I2C denetleyicisi içerecek biçimde genişletip Nexys A7 kartının ADT7420 sıcaklık sensörüyle bağla (<https://www.analog.com/media/en/technical-documentation/data-sheets/adt7420.pdf>). Ardından bu yeni çevre birimiyle iletişim kurup sıcaklığı 7-kesimli ekranlarda gösteren bir program yaz.