



THE IMAGINATION UNIVERSITY PROGRAMME

RVfpga Deney 10

Dizisel Veri Yolları

1. GİRİŞ

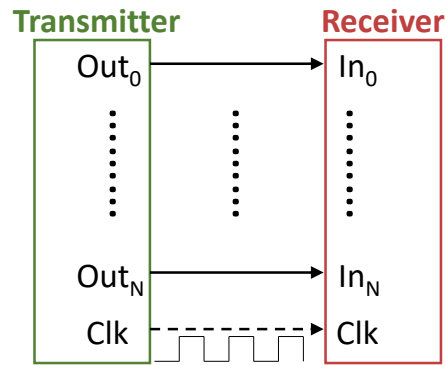
Bu deneyde ilk olarak dizeysel veri yollarının nasıl çalıştığını, günümüzde kullanılan en yaygın dizeysel veri yollarından SPI veri yolunun ana niteliklerini tanımlıyoruz (Bölüm 2). Ardından Nexys A7 kartındaki erişilebilir SPI ivmeölçerine odaklanıyoruz: bu çevre biriminin yüksek-düzeysel standardını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

2. DİZİSEL VERİ YOLLARI – SPI VERİ YOLU

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

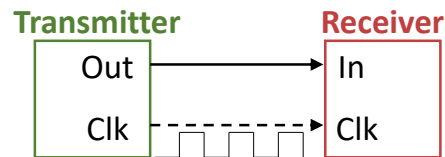
Önceki deneylerde gösterildiği gibi gömülü elektroniklerin ana amacı istenen işlevleri oluşturmak için işlemcilerle devreleri bağlamaktır. İşlemcilerle devrelerin bilgi paylaşması için ortak bir iletişim protokolleri olmalıdır. Bu veri alışverişini yapabilmek için yüzlerce iletişim protokolü tanımlanmıştır,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

Paralel arayüzler çok bitli 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 bitli 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 dizeysel 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 dizeysel arayüz örneğini gösterir. Yeni bir saat kenarında yeni bir veri biti aktarılır.



Figür 2. Dizeysel 1-bit veri yolu örneği.

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

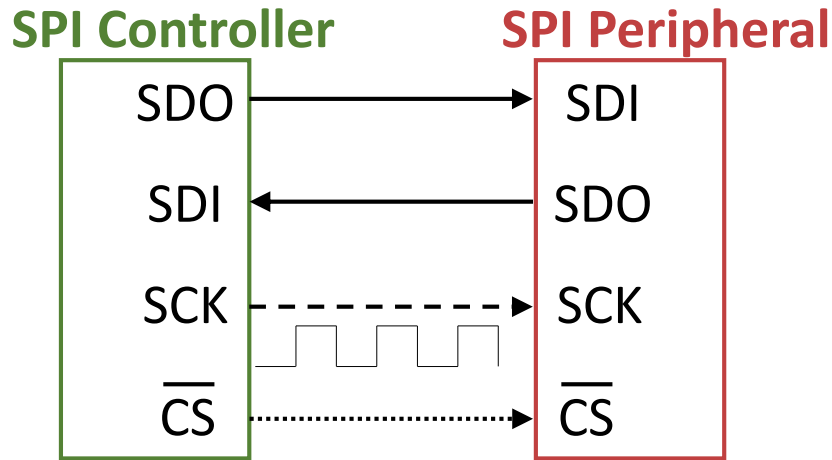
SPI Veri Yolu:

Dizeşel 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** – Dizeşel Veri Dışarı: Denetleyicinin çevre birimi aygıtı çıkışı
- **SDI** – Dizeşel Veri İçeri: Denetleyicinin çevre birimi aygıtından girdisi
- **SCK** – Dizeşel Saat: Denetleyiciden çevre birimi aygıtına 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.



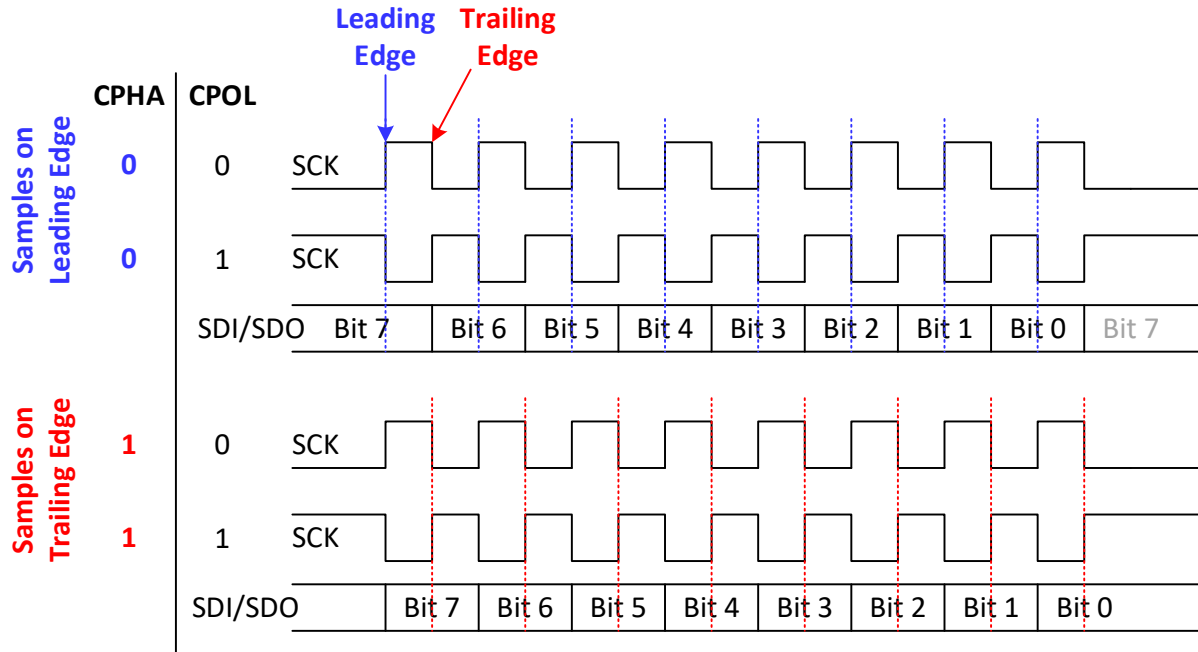
Figür 3. Bir SPI denetleyici ile bir SPI çevre birimi bir sistem örneği.

Dizeşel 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 olabilir. Birden çok çevre birimi bağlanınca erişilen çevre birimini seçmek için alçakta onaylanan çoklu yonga seçme sinyalleri (CSbar) kullanılır. SDO ile SDI dizeşel veri doğrularıdır: SDO (dizeşel veri dışarı) denetleyiciden çevre birimine çıktı veri iken SDI (dizeşel 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. Dizeşel saat (SCK) kenarı verinin örnekleme senkronize eder.

SPI arayüzü saatin boşa durumunu seçme, sinyali örnekleme fazı için CPOL ile CPHA ek sinyalleri de sağlar. Saat ucaylılık (CPOL) sinyali saat (SCK) 0'da boşayken 0, 1'de boşayken 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 boşa 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 önceki 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 boşta, 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 boşa olup yükselmeden önce SDO ile SDI sonraki veri baytının yüksek önemli bitini taşımalıdır.



Figür 4. CPHA/CPOL'ün veri örnekleme/yollamayla ilişkisi

3. 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 alıştırmayı da tanıtıyoruz.

A. 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 aralığıyla 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'nın SPI denetleyicisinin taban adresi **0x80001100**'dir. 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.

B. ADXL362 ivmeölçer spesifikasyonu

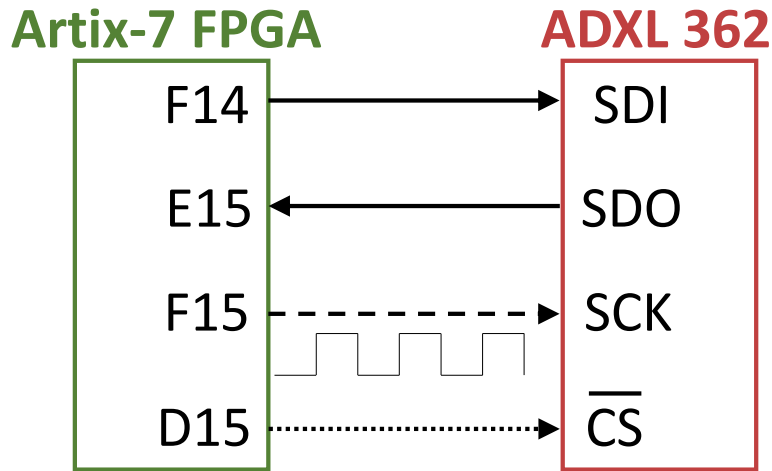
Nexys A7 kartı, bir Analog Devices ADXL362 ivmeölçeri içerir. Aygıtın bütün bilgilerini veri metninde şuradan bulabilirsiniz:

<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.

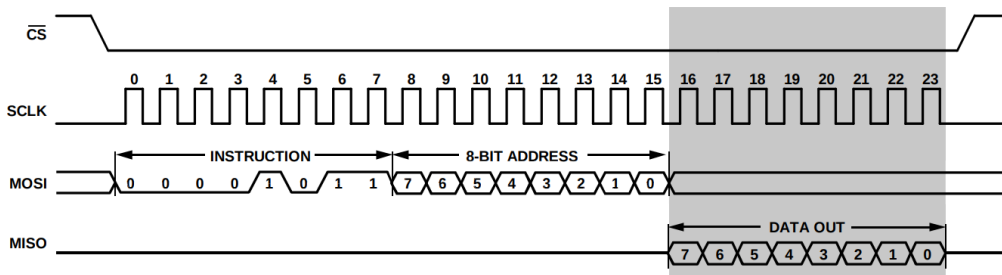


Figür 5. ADXL362 İvmeölçerinin Nexys A7 kartı ile arayüzü

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 gösterdiği bir çok baytlı yapı kullanır:

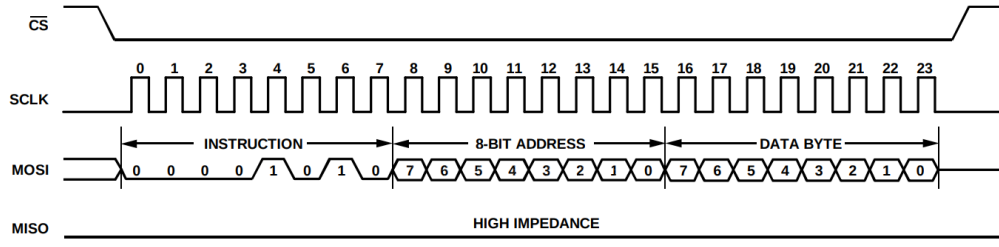
| | | | | |
|-----------|--------------------------|----------------|-------------|---------|
| <CS down> | <Write/Read (0x0A/0x0B)> | <address byte> | <data byte> | <CS up> |
|-----------|--------------------------|----------------|-------------|---------|

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. Yazma ça 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>)

| Reg | Name | Bits | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Reset | RW |
|------|----------------|-------|---------------|---------|-----------|---------|---------------------|-----------------|---------------------|------------|-------|----|
| 0x00 | DEVID_AD | [7:0] | | | | | DEVID_AD[7:0] | | | | 0xAD | R |
| 0x01 | DEVID_MST | [7:0] | | | | | DEVID_MST[7:0] | | | | 0x1D | R |
| 0x02 | PARTID | [7:0] | | | | | PARTID[7:0] | | | | 0xF2 | R |
| 0x03 | REVID | [7:0] | | | | | REVID[7:0] | | | | 0x01 | R |
| 0x08 | XDATA | [7:0] | | | | | XDATA[7:0] | | | | 0x00 | R |
| 0x09 | YDATA | [7:0] | | | | | YDATA[7:0] | | | | 0x00 | R |
| 0x0A | ZDATA | [7:0] | | | | | ZDATA[7:0] | | | | 0x00 | R |
| 0x0B | STATUS | [7:0] | ERR_USER_REGS | AWAKE | INACT | ACT | FIFO_OVER-RUN | FIFO_WATER-MARK | FIFO_READY | DATA_READY | 0x40 | R |
| 0x0C | FIFO_ENTRIES_L | [7:0] | | | | | FIFO_ENTRIES_L[7:0] | | | | 0x00 | R |
| 0x0D | FIFO_ENTRIES_H | [7:0] | | | | | UNUSED | | FIFO_ENTRIES_H[1:0] | | 0x00 | R |
| 0x0E | XDATA_L | [7:0] | | | | | XDATA_L[7:0] | | | | 0x00 | R |
| 0x0F | XDATA_H | [7:0] | SX | | | | | | XDATA_H[3:0] | | 0x00 | R |
| 0x10 | YDATA_L | [7:0] | | | | | YDATA_L[7:0] | | | | 0x00 | R |
| 0x11 | YDATA_H | [7:0] | SX | | | | | | YDATA_H[3:0] | | 0x00 | R |
| 0x12 | ZDATA_L | [7:0] | | | | | ZDATA_L[7:0] | | | | 0x00 | R |
| 0x13 | ZDATA_H | [7:0] | SX | | | | | | ZDATA_H[3:0] | | 0x00 | R |
| 0x14 | TEMP_L | [7:0] | | | | | TEMP_L[7:0] | | | | 0x00 | R |
| 0x15 | TEMP_H | [7:0] | SX | | | | | | TEMP_H[3:0] | | 0x00 | R |
| 0x16 | Reserved | [7:0] | | | | | Reserved[7:0] | | | | 0x00 | R |
| 0x17 | Reserved | [7:0] | | | | | Reserved[7:0] | | | | 0x00 | R |
| 0x1F | SOFT_RESET | [7:0] | | | | | SOFT_RESET[7:0] | | | | 0x00 | W |
| 0x20 | THRESH_ACT_L | [7:0] | | | | | THRESH_ACT_L[7:0] | | | | 0x00 | RW |
| 0x21 | THRESH_ACT_H | [7:0] | UNUSED | | | | | | THRESH_ACT_H[2:0] | | 0x00 | RW |
| 0x22 | TIME_ACT | [7:0] | | | | | TIME_ACT[7:0] | | | | 0x00 | RW |
| 0x23 | THRESH_INACT_L | [7:0] | | | | | THRESH_INACT_L[7:0] | | | | 0x00 | RW |
| 0x24 | THRESH_INACT_H | [7:0] | UNUSED | | | | | | THRESH_INACT_H[2:0] | | 0x00 | RW |
| 0x25 | TIME_INACT_L | [7:0] | | | | | TIME_INACT_L[7:0] | | | | 0x00 | RW |
| 0x26 | TIME_INACT_H | [7:0] | | | | | TIME_INACT_H[7:0] | | | | 0x00 | RW |
| 0x27 | ACT_INACT_CTL | [7:0] | RES | | LINKLOOP | | INACT_REF | INACT_EN | ACT_REF | ACT_EN | 0x00 | RW |
| 0x28 | FIFO_CONTROL | [7:0] | UNUSED | | | | AH | FIFO_TEMP | FIFO_MODE | | 0x00 | RW |
| 0x29 | FIFO_SAMPLES | [7:0] | | | | | FIFO_SAMPLES[7:0] | | | | 0x80 | RW |
| 0x2A | INTMAP1 | [7:0] | INT_LOW | AWAKE | INACT | ACT | FIFO_OVER-RUN | FIFO_WATER-MARK | FIFO_READY | DATA_READY | 0x00 | RW |
| 0x2B | INTMAP2 | [7:0] | INT_LOW | AWAKE | INACT | ACT | FIFO_OVER-RUN | FIFO_WATER-MARK | FIFO_READY | DATA_READY | 0x00 | RW |
| 0x2C | FILTER_CTL | [7:0] | RANGE | | RES | HALF_BW | EXT_SAMPLE | ODR | | | 0x13 | RW |
| 0x2D | POWER_CTL | [7:0] | RES | EXT_CLK | LOW_NOISE | | WAKEUP | AUTOSLEEP | MEASURE | | 0x00 | RW |
| 0x2E | SELF_TEST | [7:0] | UNUSED | | | | | | ST | | 0x00 | RW |

4. 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 oluşturun.

Yapılandırma, yazmaç bilgisi için Bölüm B'ye bak. RVfpga'nın SPI modülüne erişmek için şu alt yordamları kullan. Alt yordamları kullanmadan önce RVfpga'nın 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:
```

```

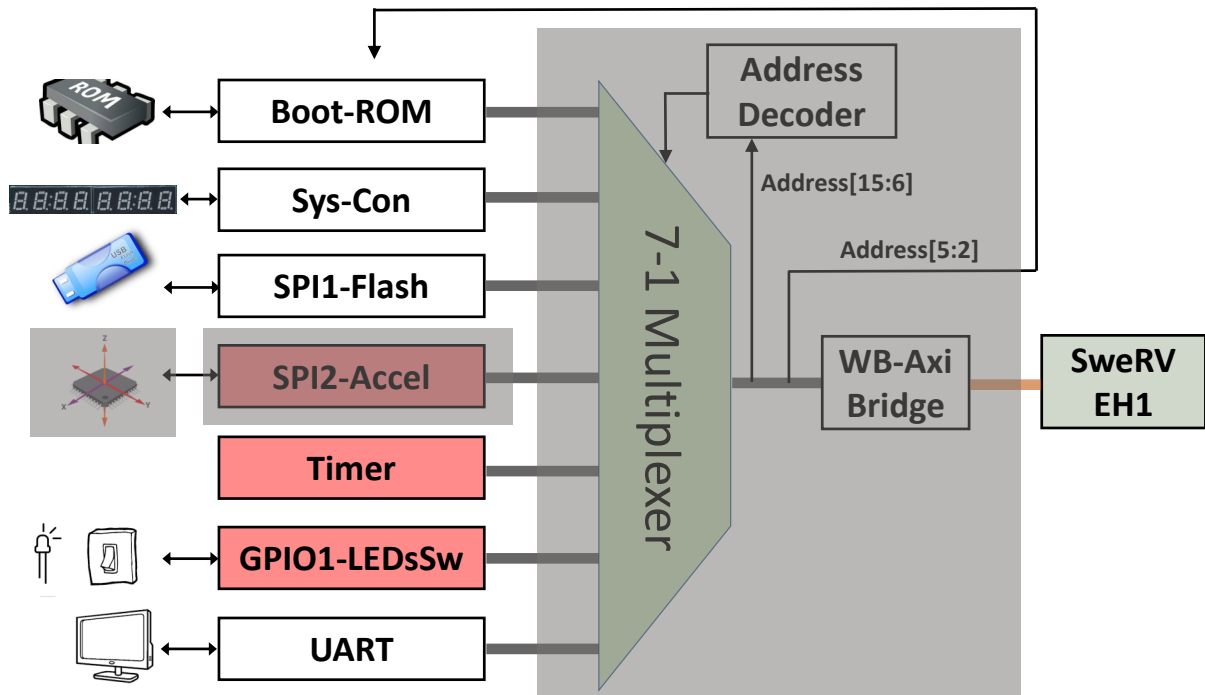
internalSpiClearIF: # internal clear interrupt flag
    li t1, SPSR # status register
    lb t0, 0(t1) # clear SPIF by writing a 1 to bit 7
    ori t0,t0,0x80
    sb t0, 0(t1)
internalSpiActualSend:
    li t0, SPDR # data register
    sb a0, 0(t0) # send the byte contained in a0 to spi
internalSpiTestIF:
    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
    
```

5. ALÇAK DÜZEYLİ GERÇEKLEŞTİRME

A. 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

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/çıkış SoC sinyalleriyle kart aygıtları 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.

```
78 ##Accelerometer
79 set_property -dict { PACKAGE_PIN E15 IOSTANDARD LVCMOS33 } [get_ports { i_accel_miso }]; #IO L11P_T1_SRCC_15 Sch=acl_miso
80 set_property -dict { PACKAGE_PIN F14 IOSTANDARD LVCMOS33 } [get_ports { o_accel_mosi }]; #IO L5N_T0_AD9N_15 Sch=acl_mosi
81 set_property -dict { PACKAGE_PIN F15 IOSTANDARD LVCMOS33 } [get_ports { accel_sclk }]; #IO L14P_T2_SRCC_15 Sch=acl_sclk
82 set_property -dict { PACKAGE_PIN D15 IOSTANDARD LVCMOS33 } [get_ports { o_accel_cs_n }];
```

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).

```
25 module rvfpga
26     #(parameter bootrom_file = "")
27     (input wire      clk,
28      input wire      rstn,
29      output wire [12:0] ddram_a,
30      output wire [2:0] ddram_ba,
31      output wire      ddram_ras_n,
32      output wire      ddram_cas_n,
33      output wire      ddram_we_n,
34      output wire      ddram_cs_n,
35      output wire [1:0] ddram_dm,
36      inout wire [15:0] ddram_dq,
37      inout wire [1:0] ddram_dqs_p,
38      inout wire [1:0] ddram_dqs_n,
39      output wire      ddram_clk_p,
40      output wire      ddram_clk_n,
41      output wire      ddram_cke,
42      output wire      ddram_odt,
43      output wire      o_flash_cs_n,
44      output wire      o_flash_mosi,
45      input wire      i_flash_miso,
46      input wire      i_uart_rx,
47      output wire      o_uart_tx,
48      inout wire [15:0] i_sw,
49      output reg [15:0] o_led,
50      output reg [7:0] AN,
51      output reg      CA, CB, CC, CD, CE, CF, CG,
52      output wire      o_accel_cs_n,
53      output wire      o_accel_mosi,
54      input wire      i_accel_miso,
55      output wire      accel_sclk);
56
248     .o_ram_bready (cpu.b_ready),
249     .i_ram_rid (cpu.r_id),
250     .i_ram_rdata (cpu.r_data),
251     .i_ram_rresp (cpu.r_resp),
252     .i_ram_rlast (cpu.r_last),
253     .i_ram_rvalid (cpu.r_valid),
254     .o_ram_rready (cpu.r_ready),
255     .i_ram_init_done (litedram_init_done),
256     .i_ram_init_error (litedram_init_error),
257     .io_data ((i_sw[15:0], gpio_out[15:0])),
258     .AN (AN),
259     .Digits_Bits ({CA, CB, CC, CD, CE, CF, CG}),
260     .o_accel_sclk (accel_sclk),
261     .o_accel_cs_n (o_accel_cs_n),
262     .o_accel_mosi (o_accel_mosi),
263     .i_accel_miso (i_accel_miso));
264
265     always @(posedge clk_core) begin
266         o_led[15:0] <= gpio_out[15:0];
267     end
268
269     assign o_uart_tx = 1'b0 ? litedram_tx : cpu_tx;
270
271 endmodule
```

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

2. 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).

```

382 // SPI for the Accelerometer
383 wire [7:0] spi2_rdt;
384 assign wb_s2m_spi_accel_dat = {24'd0, spi2_rdt};
385 wire spi2_irq;
386
387 simple_spi spi2
388 (// Wishbone slave interface
389 .clk_i (clk),
390 .rst_i (wb_rst),
391 .adr_i (wb_m2s_spi_accel_adr[2] ? 3'd0 : wb_m2s_spi_accel_adr[5:3]),
392 .dat_i (wb_m2s_spi_accel_dat[7:0]),
393 .we_i (wb_m2s_spi_accel_we),
394 .cyc_i (wb_m2s_spi_accel_cyc),
395 .stb_i (wb_m2s_spi_accel_stb),
396 .dat_o (spi2_rdt),
397 .ack_o (wb_s2m_spi_accel_ack),
398 .inta_o (spi2_irq),
399 // SPI interface
400 .sck_o (o_accel_sclk),
401 .ss_o (o_accel_cs_n),
402 .mosi_o (o_accel_mosi),
403 .miso_i (i_accel_miso));

```

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).

3. 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 gerçekleştirilir,

[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.

```

108 wb_mux
109 #(.num_slaves (7),
110 .MATCH_ADDR ({32'h00000000, 32'h00001000, 32'h00001040, 32'h00001100, 32'h00001200, 32'h00001400, 32'h00002000}),
111 .MATCH_MASK ({32'hffffff00, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0, 32'hffffffc0, 32'hffffff00}))
112 wb_mux io
113 (.wb_clk_i (wb_clk_i),
114 .wb_rst_i (wb_rst_i),
115 .wbm_adr_i (wb_io_adr_i),
116 .wbm_dat_i (wb_io_dat_i),
117 .wbm_sel_i (wb_io_sel_i),
118 .wbm_we_i (wb_io_we_i),
119 .wbm_cyc_i (wb_io_cyc_i),
120 .wbm_stb_i (wb_io_stb_i),
121 .wbm_cti_i (wb_io_cti_i),
122 .wbm_bte_i (wb_io_bte_i),
123 .wbm_dat_o (wb_io_dat_o),
124 .wbm_ack_o (wb_io_ack_o),
125 .wbm_err_o (wb_io_err_o),
126 .wbm_rty_o (wb_io_rty_o),
127 .wbs_adr_o (wb_rom_adr_o, wb_sys_adr_o, wb_spi_flash_adr_o, wb_spi_accel_adr_o, wb_ptc_adr_o, wb_gpio_adr_o, wb_uart_adr_o),
128 .wbs_dat_o (wb_rom_dat_o, wb_sys_dat_o, wb_spi_flash_dat_o, wb_spi_accel_dat_o, wb_ptc_dat_o, wb_gpio_dat_o, wb_uart_dat_o),
129 .wbs_sel_o (wb_rom_sel_o, wb_sys_sel_o, wb_spi_flash_sel_o, wb_spi_accel_sel_o, wb_ptc_sel_o, wb_gpio_sel_o, wb_uart_sel_o),
130 .wbs_we_o (wb_rom_we_o, wb_sys_we_o, wb_spi_flash_we_o, wb_spi_accel_we_o, wb_ptc_we_o, wb_gpio_we_o, wb_uart_we_o),
131 .wbs_cyc_o (wb_rom_cyc_o, wb_sys_cyc_o, wb_spi_flash_cyc_o, wb_spi_accel_cyc_o, wb_ptc_cyc_o, wb_gpio_cyc_o, wb_uart_cyc_o),
132 .wbs_stb_o (wb_rom_stb_o, wb_sys_stb_o, wb_spi_flash_stb_o, wb_spi_accel_stb_o, wb_ptc_stb_o, wb_gpio_stb_o, wb_uart_stb_o),
133 .wbs_cti_o (wb_rom_cti_o, wb_sys_cti_o, wb_spi_flash_cti_o, wb_spi_accel_cti_o, wb_ptc_cti_o, wb_gpio_cti_o, wb_uart_cti_o),
134 .wbs_bte_o (wb_rom_bte_o, wb_sys_bte_o, wb_spi_flash_bte_o, wb_spi_accel_bte_o, wb_ptc_bte_o, wb_gpio_bte_o, wb_uart_bte_o),
135 .wbs_dat_i (wb_rom_dat_i, wb_sys_dat_i, wb_spi_flash_dat_i, wb_spi_accel_dat_i, wb_ptc_dat_i, wb_gpio_dat_i, wb_uart_dat_i),
136 .wbs_ack_i (wb_rom_ack_i, wb_sys_ack_i, wb_spi_flash_ack_i, wb_spi_accel_ack_i, wb_ptc_ack_i, wb_gpio_ack_i, wb_uart_ack_i),
137 .wbs_err_i (wb_rom_err_i, wb_sys_err_i, wb_spi_flash_err_i, wb_spi_accel_err_i, wb_ptc_err_i, wb_gpio_err_i, wb_uart_err_i),
138 .wbs_rty_i (wb_rom_rty_i, wb_sys_rty_i, wb_spi_flash_rty_i, wb_spi_accel_rty_i, wb_ptc_rty_i, wb_gpio_rty_i, wb_uart_rty_i));
139
140 endmodule

```

CPU/Controller Signals

Peripheral Signals

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.

6. İLERİ DÜZEY ALIŞTIRMALAR

Alıştırma 2. Evrensel Eş Zamansız Alıcı Verici (UART) bir eş zamansız (asenكرون) iletişim protokolüdür. RVfpga yalın tasarımında bir UART modülü içerir (Figür 8), spesifikasyonunu şuradan bulabilirsiniz:

[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals/uart/docs/UART_spec.pdf

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

Ardından dizeysel 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 değerlendir.
- 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.