



**THE IMAGINATION UNIVERSITY PROGRAMME**

# **RVfpga Deney 6**

## **Girdi/Çıktıya Giriş**

## 1. GİRİŞ

6-10 arası Deneylerde RVfpga'nın Girdi/Çıktı (I/O) sistemini RISC-V işlemcisinin çevre birimleriyle etkileşimini etkinleştirmek için nasıl kullanıp genişleteceğini öğreneceksin. Aşağıdaki bu deneylerde kapsanan konuların tanıtımıdır:

- **Deney 6:** Nexys A7 kartındaki LEDlere, anahtarlara, düğmelere bağlı genel amaçlı girdi/çıkı (GPIO) uçlarının nasıl kullanılacağını öğren
- **Deney 7:** Karttaki 7-kesimli ekranların nasıl kullanılacağını öğren
- **Deney 8:** Zamanlayıcıları nasıl kullanacağını öğren
- **Deney 9:** Dış aygıtlarla arayüz yapmak için kesintileri nasıl kullanacağını öğren
- **Deney 10:** RVfpga'i kart üzerindeki SPI ivmeölçerle nasıl arayüz yapacağını öğren

Bu deneyde ilk olarak genel amaçlı bir I/O sisteminin ana özellikleriyle RVfpga'de kullanılanı tanımlıyoruz (Bölüm 2). Ardından basitleştirilmiş teorik bir genel yapıyı GPIO denetleyicisini tanımlıyoruz (Bölüm 3). Son olarak RVfpga'de kullanılan GPIO denetleyicisine odaklanıyoruz: ilk olarak yüksek düzeyli spesifikasyonunu çözümleyip ardından temel alıştırmalara geçiyoruz (Bölümler 4 ile 5). Deneyi alçak düzeyli gerçekleştirmesini çözümleyip, RVfpga'i Verilator'da simülasyonunu yapıp, ileri düzey alıştırmalara geçiyoruz (Bölümler 6 ile 7).

7-10 arası Deneylerde de bu genel yapıyı kullanıyoruz. Başlangıç bölümlerinde I/O denetleyicisinin yüksek düzeyli spesifikasyonunu (ana özellikleri, yazmaçları, işlevi, bellek eşlemesi) tanımlayıp ardından çevre birimini kullanmada pratik için temel alıştırmalar tanıtıyoruz. İleri düzey bölümlerde denetleyicinin alçak düzeyli gerçekleştirmesini tanımlıyoruz, sonrasında ise alçak düzeyli gerçekleştirmeyi değiştirip, değişikliği test eden programlar yazma için alıştırmalar sağlıyoruz.

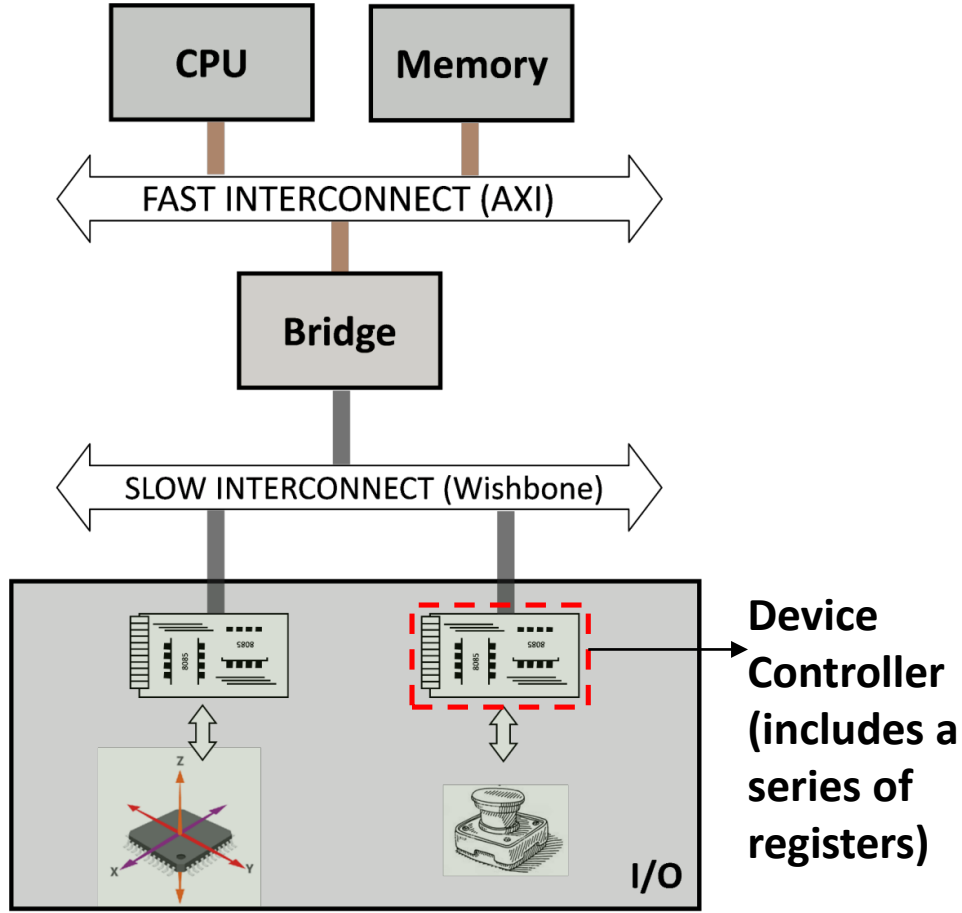
**Eğitmenlere not:** kurs düzeyine göre alıştırma karmaşıklığını seçebilirsin. Örneğin ilk/ikinci yıl kursu için (Computer Fundamentals ya da Computer Organization gibi), temel alıştırmalar – bu deneyde Bölüm 5 – uygun olacaktır. Ancak daha ileri düzey bir kursta (Computer Architecture ya da Embedded System Design gibi), temel de ileri düzey de – bu deneyde bölümler 5 ile 7 – kullanılabilir.

## 2. GİRDİ/ÇIKTI MİMARİSİ

Figür 1 üç ana parçadan oluşan Von Neumann Mimarisini görselleştirir: CPU, Bellek, I/O Sistemi. 6-10 arası Deneylerde CPU'nun girdi/çıkı (I/O) aygıtları ile etkileşimine odaklanıyoruz. I/O aygıtlarına çevre birimleri ya da basitçe aygıtlar da denir. Burada ana ünitelerin rolünü tanıtıyoruz:

- **CPU:** CPU bütün I/O işlemlerinin başlatıcısıdır. Bütün I/O hareketlerinin denetleyicisidir (eskiden “usta” denirdi ancak artık kullanım dışı bir terimdir). Bir direkt bellek erişimi (DMA) denetleyicisi (DMAC) da denetleyici olarak davranabilir ancak bu deneyde içerilmez.
- **Aygıt Denetleyicisi:** Aygıt denetleyicisi bir eylemi gerçekleştirmek için bir denetleyiciden okuma/yazma isteklerini bekler. Aygıt denetleyicileri I/O sistemi içerisinde çevre birimi olarak davranır (eskiden “köle” denirdi ancak artık kullanım dışı bir terimdir). Konsept olarak bir aygıt denetleyicisi denetleyiciden erişilebilen yazmaçlar dizisinden oluşur. Bu yazmaçların değerleri ilgili çevre birimine ne yapacağını söyler.
- **Ara bağlantı** (veri yolu, çaprazlayıcı, gibi) denetleyiciyle çevre birimleri arasında bir yol belirler. Ara bağlantı genelde belirli aygıtların bütün sistemi yavaşlatmasını

engellemek için bir köprüyle bağlı birkaç katmanla gerçekleştirilir.

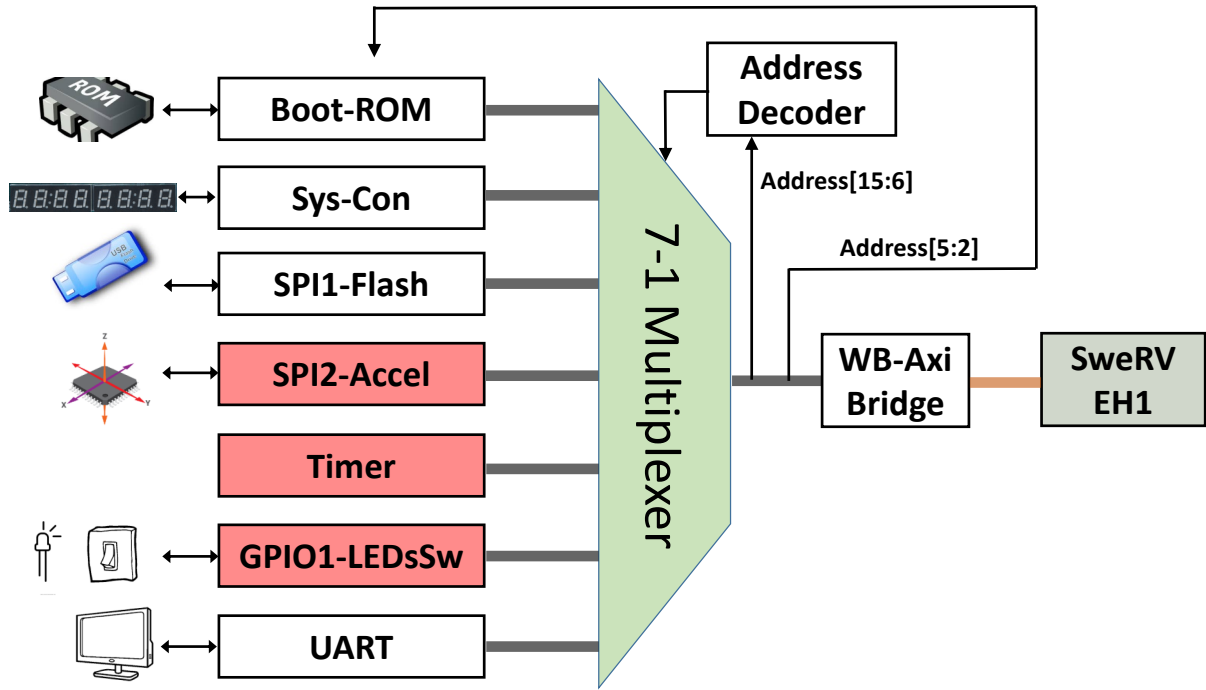


**Figür 1 Genel Yapılı Hesaplama Sistemi**

Figür 2, RVfpga'nın I/O sistemini gösterir. Yedi çevre birimi içerir:

- LEDler, Anahtarlar (bir çevre birimi sayılır), GPIO1 modülüne bağlı
- 7-kesimli ekranlar, Sistem Denetleyicisi modülüne bağlı
- Flash Bellek, SPI1 modülüne bağlı
- İvmeölçer, SPI2 modülüne bağlı
- Zamanlayıcı
- UART
- Boot (Önyükleme) ROM

Bir çoklayıcı yedi çevre birimi arasından birini seçip CPU'ya başlar. Önemli olarak çevre birimleri Wishbone veri yolu (gri), SweRV EH1 Çekirdeği AXI köprü (turuncu) kullandığından bir Wishbone'dan AXI'a Köprüsü gereklidir.



Figür 2. RVfpga'de I/O Sistemi

**GÖREV:** Figür 2'deki öğeleri SoC'de bul. Şu dosyalarla klasörleri çözümlemen gerekecek:

```
[RVfpgaPath]/RVfpga/src/SweRVolfSoC/swervolf_core.v.  
[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals  
[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect  
[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals/SystemController/swervolf_syscon.v  
[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect/wb_intercon.v  
[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect/wb_intercon.vh
```

RVfpga İlk Kullanım Kılavuzunda tanımlandığı gibi, orijinal SweRVolf (<https://github.com/chipsalliance/Cores-SweRVolf>) Figür 2'de gösterilen çevre birimlerinden yalnızca birkaçını içerir: Boot ROM, Sistem Denetleyicisi (7-Kesimli Ekranlar olmadan), SPI Flash Bellek, UART (Figür 2'de beyazla gösterilmiş). RVfpga, orijinal SweRVolf SoC'yi yeni çevre birimleriyle genişletir: bir SPI İvmeölçer, bir Zamanlayıcı, bir GPIO modülü (Figür 2'de kırmızıyla gösterilmiş), bir 7-kesimli ekran denetleyicisi (SweRVolf'un kendi Sistem Denetleyicisini genişleten).

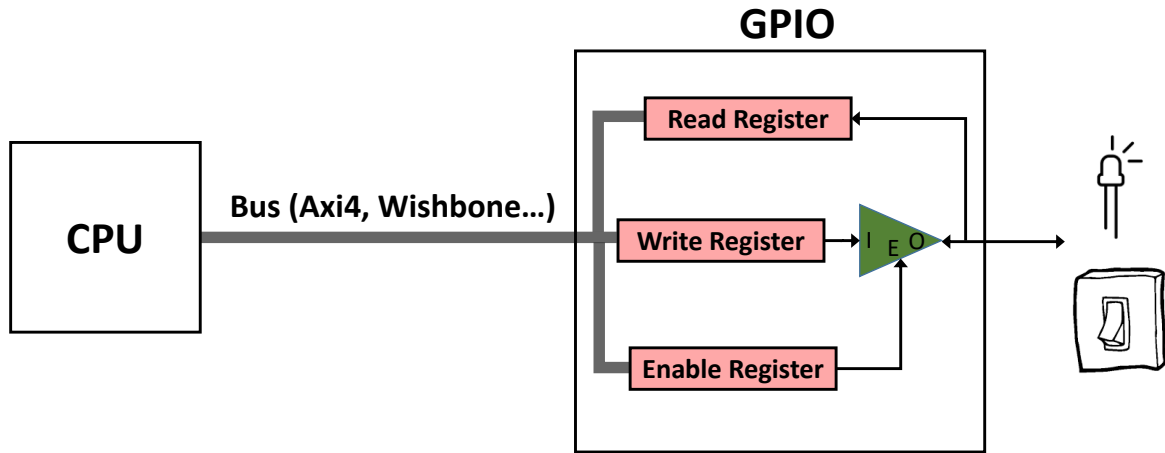
Bütün çevre birimleri işlemciden değerler alır ve/ya işlemciye değerler yollar. Bellek adresleri I/O değerleri için ayrılır, bunlara yazmaçlar, bellek eşlenmiş I/O yazmaçları, ya da aygıt denetleyici yazmaçları denir. Çevre birimine değer yollamak için CPU belirlenen adrese (memory-mapped register) değer depolar. Çevre biriminden değer olmak için CPU belirli bellek adresinden değer yükler. Dolayısıyla CPU'dan basit bir yükleme/depolama işlemi bir aygıtı yapılandırabilir, durumunu denetleyebilir, ondan veri okuyup, ona veri yazabilir.

Figür 2'deki çoklayıcı Address[15:6] kullanarak istenen aygıtı seçer. Aygıt denetleyicileri Address[5:2]'yi aygıtı denetlemek için yazmaç seçmek için kullanır.

### 3. GENEL AMAÇLI GİRİDİ/ÇIKTI (GPIO)

Bir genel amaçlı I/O (GPIO) denetleyicisi dış dijital uçları programlamacıya açar. Programın istenilen yerinde bu uçlar girdi ya da çıktı olarak yapılandırılabilir. Yapılandırma uç başına ayrı olup program süresince isteğe bağlı değişebilir. GPIO uçları LEDler, anahtarlar, düğmeler gibi dış aygıtlara bağlanabilir.

Figür 3 bir dış ucu CPUya bağlayan bir genel yapıları GPIO modülü için basitleştirilmiş bir diyagramı görselleştirir. Uç istenilen LED, anahtar gibi bir girdi/çıkıta aygıtına bağlanabilir. Uç bir üç-durumlu arabelleğe bağlıdır, figürde yeşille gösterilmiştir. Bu arabellek programlamacının ucu girdi ya da çıktı olarak yapılandırmasına olanak sağlar. Eğer üç-durumlu arabellek etkinse uç çıktı olarak davranır (örneğin bir LEDi ayarlamak için). Eğer üç-durumlu arabellek etkin değilse uç bir girdi olarak davranır (örneğin anahtarlardan değer okumak için).



**Figür 3. GPIO basitleştirilmiş devre**

Üç durumlu bir arabellek, bir olağan arabellek olarak (etkinken) ya da yüzer çıktı (etkin değilken) davranabilir. Üç durumlu arabellenin iki girdisi, E (etkinleştir) ile I (girdi), bir çıkışı, O, vardır, doğruluk tablosu Tablo 1’de gösterilmiştir. E 1 iken üç durumlu, çıktı (O) ile girdinin (I) aynı olduğu bir olağan arabellek olarak davranır. E 0 iken girdiyle çıktı arasında bağlantı olmayıp, çıktı (O) güdümlü değildir; O yüzerdir. Figür 3’te bir ucu çıktı olarak yapılandırmak için E, 1 olur, bu ucun CPU güdümlü olmasını sağlar. Bir ucu girdi olarak yapılandırmak için E, 0 olur, bu ucun CPU güdümlü olmasını engelleyip çevre birimini güdümlü olmasını sağlar.

**Tablo 1. Üç-durumlu doğruluk tablosu**

E	I	O
0	0	Hi-Z
0	1	Hi-Z
1	0	0
1	1	1

RVfpga bu yazmaçlarda depolanan değerleri okuma/yazma için bellek eşlenmiş I/O kullanır. Örneğin Figür 3’teki ucun bir anahtara bağlı olduğunu, GPIO’daki üç yazmacın şöyle eşlendiğini varsay:

- Okuma Yazmacı = Adres 0x80001400

- Yazma Yazmacı = Adres 0x80001404
- Etkinleştirme Yazmacı = Adres 0x80001408

Anahtarın durumunu okumak için şunu yaparız:

1. Ucu Etkinleştirme Yazmacına 0 yazarak girdi olarak yapılandırırız (bir diğer deyişle 0x80001408 adresine 0 *depolamayı* yürüterek).
2. 0x80001400 adresine *yükleme* yönergesi yürüterek Okuma Yazmacını okuruz.

#### 4. GPIO YÜKSEK DÜZEYLİ SPESİFİKASYON

Bu bölümde ilk olarak RVfpga'in GPIO'sunun yüksek düzeyli spesifikasyonunu çözümleyip ardından bu çevre birimini kullanan bir alıştırma öneriyoruz.

##### A. GPIO yüksek düzeyli spesifikasyon

RVfpga'de kullanılan GPIO modülü OpenCores'dandır (<https://opencores.org/projects/gpio>). OpenCores'un GPIO modül indirmesiyle sağlanan gpio\_spec.pdf belgesi modülün yüksek düzeyli spesifikasyonunu tanımlar. Şuradan erişilebilir:

[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals/gpio/docs/gpio\_spec.pdf. GPIO modülünün ana işlemiyle özelliklerini bu deneyde özetliyoruz. Ancak bütün spesifikasyonları gpio\_spec.pdf içerisinde elde edebilirsiniz.

GPIO modülünde şu ana özellikler vardır:

- Bir Wishbone Ara Bağlantısı kullanır.
- Yalnızca bir çevre birimi olarak işlem yapar.
- Kullanıcı, 1-32 GPIO ucu kullanabilir.
- Birden çok GPIO modülü (GPIO çekirdeği de denir), 32'den çok GPIO ucuna erişmek için paralel olarak kullanılabilir.
- Bütün GPIO uçları:
  - iki yönlü (bu durumda dış iki yönlü I/O hücreleri gerekir) olabilir.
  - üç durumlu ya da açık boşaltma etkin (open-drain enabled) (bu durumda dış üç durumlu ya da açık boşaltma I/O hücreleri gereklidir) olabilir.
- Girdi olarak programlanan GPIO uçları:
  - yazmaç olarak kullanılabilir.
  - CPU'ya kesinti isteği iletebilir.

GPIO çekirdek spesifikasyonunun Bölüm 4'ü GPIO modülü içerisinde erişilebilir denetleme, durum yazmaçlarını tanımlar. Bu yazmaçların hepsi ayrı adreslere, Tablo 2'de gösterildiği gibi, atanmıştır. GPIO yazmaçlarının taban adresi **0x80001400**'dir.

**Tablo 2. GPIO Yazmaçları**

Adı	Adresi	Genişlik	Erişim	Tanım
RGPIO_IN	0x80001400	1-32	R	GPIO input data
RGPIO_OUT	0x80001404	1-32	R/W	GPIO output data
RGPIO_OE	0x80001408	1-32	R/W	GPIO output driver enable
RGPIO_INTE	0x8000140C	1-32	R/W	Interrupt enable
RGPIO_PTRIG	0x80001410	1-32	R/W	Type of event that triggers an interrupt
RGPIO_AUX	0x80001414	1-32	R/W	Multiplex auxiliary inputs to GPIO outputs
RGPIO_CTRL	0x80001418	2	R/W	Control register
RGPIO_INTS	0x8000141C	1-32	R/W	Interrupt status
RGPIO_ECLK	0x80001420	1-32	R/W	Enable gpio_eclk to latch RGPIO_IN

RGPIO_NEC	0x80001424	1-32	R/W	Select active edge of gpio_eclk
-----------	------------	------	-----	---------------------------------

OpenCores'un GPIO modülü Figür 3'te görselleştirilen basitleştirilmiş sürümden daha karmaşık olsa da Figür 3'teki üç yazmacı yine tanıyabiliriz: Okuma (girdi), Yazma (çıkıtı), Etkinleştir. OpenCores'un GPIO modülünde bu yazmaçlar sırasıyla şöyle adlandırılır: RGPIO\_IN, RGPIO\_OUT, RGPIO\_OE, sırasıyla şu adreslere eşlenmiştir 0x80001400, 0x80001404, 0x80001408.

**GÖREV:** GPIO modülünde RGPIO\_IN, RGPIO\_OUT, RGPIO\_OE yazmaçlarının bildirisiyle adreslerinin tanımını bul. GPIO modülü şuradadır:  
*[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals/gpio/gpio\_top.v.*

RGPIO\_IN yazmacı genel amaçlı girdilere sürgüdür. RGPIO\_OUT yazmacı genel amaçlı çıktıları sürer. RGPIO\_OE, I/O uçlarını girdi ya da çıktı olarak yapılandırır. Etkinleştirme biti (RGPIO\_OE içerisinde) ayarlıyken karşılık gelen genel amaçlı çıktı sürücüsü etkinleştirilir, böylelikle uç bir çıktı çevre birimine bağlanabilir, LED gibi. Etkinleştirme biti sıfırlanınca çıktı sürücüsü açık-boşalmada işlem yapar, buna üç durumlu ya da yüksek direnç de denir, dolayısıyla uç bir girdi çevre birimine bağlıdır, anahtar ya da düğme gibi.

RVfpga'de GPIO modülünün ilk 16 GPIO ucu, 15:0, Nexys A7 kartındaki 16 LEDe bağlıdır. GPIO denetleyicisinin son 16 GPIO ucu, 31:16, karttaki 16 anahtara bağlıdır.

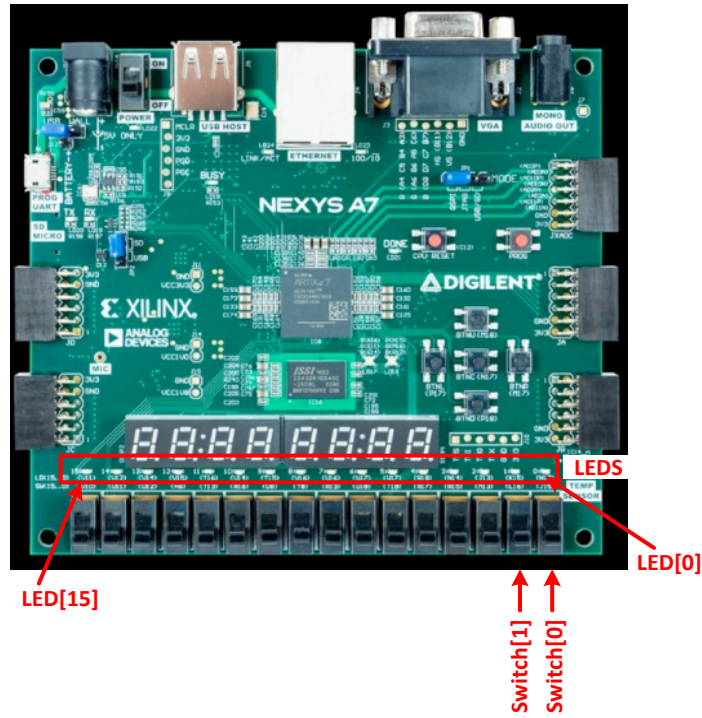
## 5. TEMEL ALIŞTIRMALAR

**Alıştırma 1.** Karttaki 16 LED'in bir yanından diğer yanına sürekli oynayan dört yakılmış LED bloğu gösteren bir RISC-V çeviricisi programıyla bir C programı yaz. Ayrıca hızla yönü denetleyen iki anahtar da içeriyoruz. Switch[0] hızı, Switch[1] yönü şöyle değiştirir:

- Switch[0] etkinse (yüksek), yakılmış LEDler hızlıca oynamalı. Öteki türlü yakılmış LEDler yavaşça oynamalı. "hızlıca" ile "yavaşça" sözcüklerinin ne düzey belirttiğini kendin tanımlayabilirsin ancak bu iki hız düzeyi gözle ayırt edilebilmeli.
- Switch[1] etkinse (yüksek), yakılmış LEDler tekrarlayarak sağdan sola oynamalı (en sola ulaşınca sağdan başlarlar). Öteki türlü yakılmış LEDler tekrarlayarak soldan sağa oynamalı.

Aşağıdaki Figür 4 Nexys A7 kartında LEDleri, anahtarları gösterir.





Figür 4. Nexys A7 FPGA Kartı: LEDler, Anahtarlar

**İpucu:** Anahtarların bellek-eşlenmiş I/O yazmaçlarının 31:16 uçlarına bağlı olduğunu anımsa. Yani, Switch[0]'ı okumak için, RGPIO\_OE[16]'ya 0 yazıp ardından RGPIO\_IN[16] değerini okuman gerekir. Diğer LEDlerle anahtarlara erişmek için RGPIO\_OE'yi uygun olarak yapılandırman gerekir.

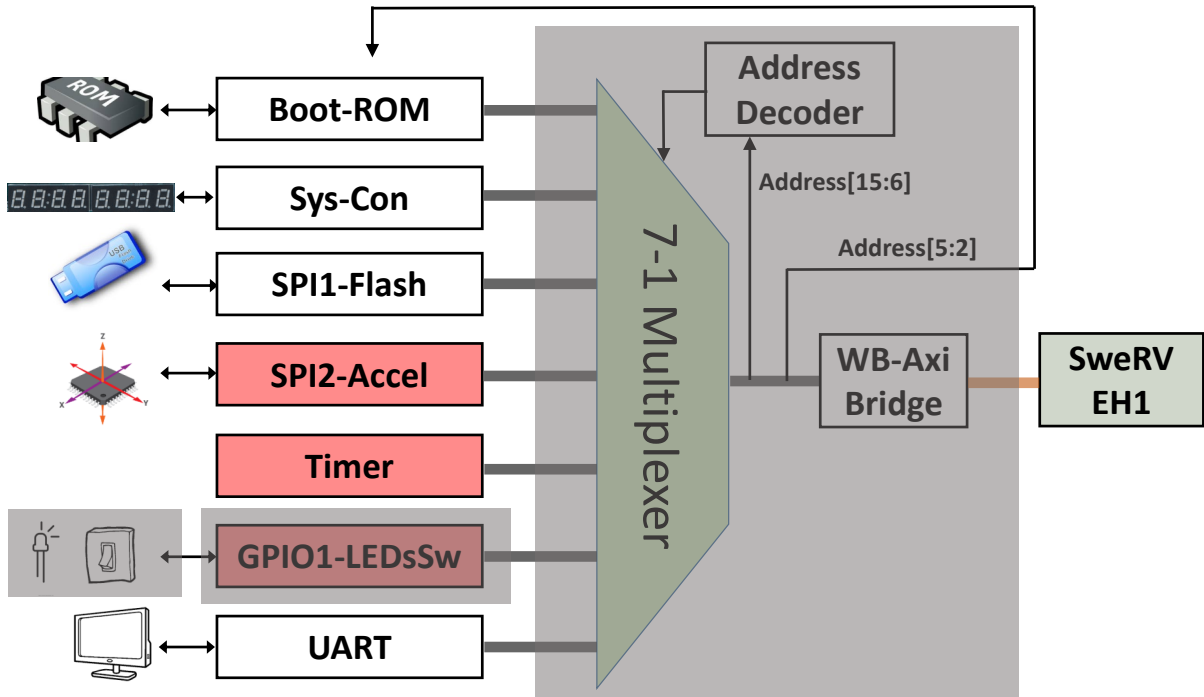
## 6. GPIO ALÇAK DÜZEYLİ GERÇEKLEŞTİRME, SİMÜLASYON

Bu bölümde RVfpga'de kullanılan GPIO'nun alçak düzeyli detaylarını tanımlıyoruz. Ardından RVfpgaSIM'i değiştirip basit bir çevirici örneği için Verilator'de bir örnek simülasyon çalıştırıyoruz. Son olarak ilk RVfpga'nın simülasyonunu yapacağın, ardından yeni GPIO çevre birimi ekleyip son olarak bu yeni çevre birimini kullanacağın alıştırma öneriyoruz.

### A. GPIO alçak düzeyli gerçekleştirme

Bellek eşlenmiş I/O kullanarak GPIO uçlarına erişmeyi deneyimlediğine göre şimdi GPIO'nun alçak düzeyli detaylarına girelim. GPIO Figür 5'te gösterildiği gibi üç ana parçaya bölünebilir: (1) RVfpga'nın karttaki LEDlerle/Anahtarlara dış bağlantısı (Figür 5'te sol taralı bölge); (2) GPIO modülünün RVfpga'e entegrasyonu (Figür 5'te orta taralı bölge); (3) GPIO ile SweRV EH1 Çekirdeği arasındaki bağlantı (Figür 5'te sağ taralı bölge).





Figür 5. 3 fazda GPIO çözümlemesi

#### i. LEDlerin/Anahtarların SoC ile bağlantısı

Projenin kısıtlandırma dosyası (*[RVfpgaPath]/RVfpga/src/rvfpga.xdc*) girdi/çıktı SoC sinyalleri ile kart aygıtları arasındaki bağlantıyı tanımlar. Bütün kart aygıtları bir FPGA ucuyla ilişkilendirilmiştir. Örneğin, Switch[0], kartın en sağındaki anahtar, bir baskılı devre kartı (PCB) izi üzerinden FPGA ucu J15'e bağlıdır.

Nexys A7 kartı 16 LED, 16 Anahtar içerir. 16 LEDi SoC'nin üst modülüyle bağlayan sinyal (SoC'ye RVfpga denir, şuradadır *[RVfpgaPath]/RVfpga/src/rvfpga.sv*) *o\_led[15:0]* adındadır, 16 Anahtarı üst modülle bağlayan sinyal ise *i\_sw[15:0]* adındadır. Figür 6 Xilinx tasarım kısıtlandırma (xdc) dosyasının, *rvfpga.xdc* (*[RVfpgaPath]/RVfpga/src* içerisinde) sinyalle FPGA uçları arasındaki 32 bağlantının tanımlandığı bölümünü gösterir.

```

26 set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { i_sw[0] }]
27 set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [get_ports { i_sw[1] }]
28 set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMOS33 } [get_ports { i_sw[2] }]
29 set_property -dict { PACKAGE_PIN R15 IOSTANDARD LVCMOS33 } [get_ports { i_sw[3] }]
30 set_property -dict { PACKAGE_PIN R17 IOSTANDARD LVCMOS33 } [get_ports { i_sw[4] }]
31 set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVCMOS33 } [get_ports { i_sw[5] }]
32 set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVCMOS33 } [get_ports { i_sw[6] }]
33 set_property -dict { PACKAGE_PIN R13 IOSTANDARD LVCMOS33 } [get_ports { i_sw[7] }]
34 set_property -dict { PACKAGE_PIN T8 IOSTANDARD LVCMOS18 } [get_ports { i_sw[8] }]
35 set_property -dict { PACKAGE_PIN U8 IOSTANDARD LVCMOS18 } [get_ports { i_sw[9] }]
36 set_property -dict { PACKAGE_PIN R16 IOSTANDARD LVCMOS33 } [get_ports { i_sw[10] }]
37 set_property -dict { PACKAGE_PIN T13 IOSTANDARD LVCMOS33 } [get_ports { i_sw[11] }]
38 set_property -dict { PACKAGE_PIN H6 IOSTANDARD LVCMOS33 } [get_ports { i_sw[12] }]
39 set_property -dict { PACKAGE_PIN U12 IOSTANDARD LVCMOS33 } [get_ports { i_sw[13] }]
40 set_property -dict { PACKAGE_PIN U11 IOSTANDARD LVCMOS33 } [get_ports { i_sw[14] }]
41 set_property -dict { PACKAGE_PIN V10 IOSTANDARD LVCMOS33 } [get_ports { i_sw[15] }]
42
43 set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33 } [get_ports { o_led[0] }]
44 set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVCMOS33 } [get_ports { o_led[1] }]
45 set_property -dict { PACKAGE_PIN J13 IOSTANDARD LVCMOS33 } [get_ports { o_led[2] }]
46 set_property -dict { PACKAGE_PIN N14 IOSTANDARD LVCMOS33 } [get_ports { o_led[3] }]
47 set_property -dict { PACKAGE_PIN R18 IOSTANDARD LVCMOS33 } [get_ports { o_led[4] }]
48 set_property -dict { PACKAGE_PIN V17 IOSTANDARD LVCMOS33 } [get_ports { o_led[5] }]
49 set_property -dict { PACKAGE_PIN U17 IOSTANDARD LVCMOS33 } [get_ports { o_led[6] }]
50 set_property -dict { PACKAGE_PIN U16 IOSTANDARD LVCMOS33 } [get_ports { o_led[7] }]
51 set_property -dict { PACKAGE_PIN V16 IOSTANDARD LVCMOS33 } [get_ports { o_led[8] }]
52 set_property -dict { PACKAGE_PIN T15 IOSTANDARD LVCMOS33 } [get_ports { o_led[9] }]
53 set_property -dict { PACKAGE_PIN U14 IOSTANDARD LVCMOS33 } [get_ports { o_led[10] }]
54 set_property -dict { PACKAGE_PIN T16 IOSTANDARD LVCMOS33 } [get_ports { o_led[11] }]
55 set_property -dict { PACKAGE_PIN V15 IOSTANDARD LVCMOS33 } [get_ports { o_led[12] }]
56 set_property -dict { PACKAGE_PIN V14 IOSTANDARD LVCMOS33 } [get_ports { o_led[13] }]
57 set_property -dict { PACKAGE_PIN V12 IOSTANDARD LVCMOS33 } [get_ports { o_led[14] }]
58 set_property -dict { PACKAGE_PIN V11 IOSTANDARD LVCMOS33 } [get_ports { o_led[15] }]

```

**Figür 6.** *i\_sw[15:0]*'ın karttaki anahtarlarla, *o\_led[15:0]*'ın karttaki LEDlerle bağlantısı (dosya *rvfpga.xdc*).

Üst modülün (*rvfpga*) 48-49 satırları SoC'ye bağlı bu iki sinyali gösterir (Figür 7'nin sol kısmı), modülün sonu ise **swervolf\_core** modülüyle bağlantılarını gösterir (Figür 7'nin sağ kısmı). Önemli olarak *i\_sw* ile *o\_led* sinyalleri *io\_data* (satır 257), **swervolf\_core** modülünde GPIO'ya bağlı 32-bitlik girdi/çıkış sinyali (Figür 8'de sonra gösterileceği gibi), sinyalinde birleşir. Dahası *o\_led* sinyali aracı bir sinyal üzerinde sürgü (latch) edilir, *gpio\_out* (satır 266).

```

25 module rvfpga
26     #(parameter bootrom_file = "bootloader.vh")
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,

```

```

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

```

**Figür 7.** LEDlerle Anahtarların üst modülle bağlantısı (*rvfpga.sv*)

**GÖREVLER:** Bu iki sinyali (*i\_sw*, *o\_led*) kısıtlar dosyasından başlayıp SweRVolf SoC modülüne (*io\_data*'da birleştirildikleri yerde) kadar izle. Şu dosyaları incelemen gerekecek:

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

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

Önceki bölümde RVfpga’de ilk 16 GPIO ucunun (15’ten 0’a) karttaki 16 LEDe bağlı olduğunu söyledik, bununla birlikte son 16 GPIO ucu (31’den 16’ya) 16 karttaki anahtara bağlıdır. Bu bilgi, bu bölümle Figür 8’de tanımlanan gerçekleştirmeyle uyumlu mudur?

## ii. GPIO modülünün SoC’de entegrasyonu

**swervolf\_core** modülünün 299-354 satırlarında

([RVfpgaPath]/RVfpga/src/SweRVolfSoC/swervolf\_core.v), GPIO modülünün somutlaması yapıp SoC’ye entegre edilir (Figür 8).

```

299 // GPIO - Leds and Switches
300 wire [31:0] en_gpio;
301 wire        gpio_irq;
302 wire [31:0] i_gpio;
303 wire [31:0] o_gpio;
304
305 bidirec gpio0 (.oe(en_gpio[0]), .inp(o_gpio[0]), .outp(i_gpio[0]), .bidir(io_data[0]));
306 bidirec gpio1 (.oe(en_gpio[1]), .inp(o_gpio[1]), .outp(i_gpio[1]), .bidir(io_data[1]));
307 bidirec gpio2 (.oe(en_gpio[2]), .inp(o_gpio[2]), .outp(i_gpio[2]), .bidir(io_data[2]));
308 bidirec gpio3 (.oe(en_gpio[3]), .inp(o_gpio[3]), .outp(i_gpio[3]), .bidir(io_data[3]));
309 bidirec gpio4 (.oe(en_gpio[4]), .inp(o_gpio[4]), .outp(i_gpio[4]), .bidir(io_data[4]));
310 bidirec gpio5 (.oe(en_gpio[5]), .inp(o_gpio[5]), .outp(i_gpio[5]), .bidir(io_data[5]));
311 bidirec gpio6 (.oe(en_gpio[6]), .inp(o_gpio[6]), .outp(i_gpio[6]), .bidir(io_data[6]));
312 bidirec gpio7 (.oe(en_gpio[7]), .inp(o_gpio[7]), .outp(i_gpio[7]), .bidir(io_data[7]));
313 bidirec gpio8 (.oe(en_gpio[8]), .inp(o_gpio[8]), .outp(i_gpio[8]), .bidir(io_data[8]));
314 bidirec gpio9 (.oe(en_gpio[9]), .inp(o_gpio[9]), .outp(i_gpio[9]), .bidir(io_data[9]));
315 bidirec gpio10 (.oe(en_gpio[10]), .inp(o_gpio[10]), .outp(i_gpio[10]), .bidir(io_data[10]));
316 bidirec gpio11 (.oe(en_gpio[11]), .inp(o_gpio[11]), .outp(i_gpio[11]), .bidir(io_data[11]));
317 bidirec gpio12 (.oe(en_gpio[12]), .inp(o_gpio[12]), .outp(i_gpio[12]), .bidir(io_data[12]));
318 bidirec gpio13 (.oe(en_gpio[13]), .inp(o_gpio[13]), .outp(i_gpio[13]), .bidir(io_data[13]));
319 bidirec gpio14 (.oe(en_gpio[14]), .inp(o_gpio[14]), .outp(i_gpio[14]), .bidir(io_data[14]));
320 bidirec gpio15 (.oe(en_gpio[15]), .inp(o_gpio[15]), .outp(i_gpio[15]), .bidir(io_data[15]));
321 bidirec gpio16 (.oe(en_gpio[16]), .inp(o_gpio[16]), .outp(i_gpio[16]), .bidir(io_data[16]));
322 bidirec gpio17 (.oe(en_gpio[17]), .inp(o_gpio[17]), .outp(i_gpio[17]), .bidir(io_data[17]));
323 bidirec gpio18 (.oe(en_gpio[18]), .inp(o_gpio[18]), .outp(i_gpio[18]), .bidir(io_data[18]));
324 bidirec gpio19 (.oe(en_gpio[19]), .inp(o_gpio[19]), .outp(i_gpio[19]), .bidir(io_data[19]));
325 bidirec gpio20 (.oe(en_gpio[20]), .inp(o_gpio[20]), .outp(i_gpio[20]), .bidir(io_data[20]));
326 bidirec gpio21 (.oe(en_gpio[21]), .inp(o_gpio[21]), .outp(i_gpio[21]), .bidir(io_data[21]));
327 bidirec gpio22 (.oe(en_gpio[22]), .inp(o_gpio[22]), .outp(i_gpio[22]), .bidir(io_data[22]));
328 bidirec gpio23 (.oe(en_gpio[23]), .inp(o_gpio[23]), .outp(i_gpio[23]), .bidir(io_data[23]));
329 bidirec gpio24 (.oe(en_gpio[24]), .inp(o_gpio[24]), .outp(i_gpio[24]), .bidir(io_data[24]));
330 bidirec gpio25 (.oe(en_gpio[25]), .inp(o_gpio[25]), .outp(i_gpio[25]), .bidir(io_data[25]));
331 bidirec gpio26 (.oe(en_gpio[26]), .inp(o_gpio[26]), .outp(i_gpio[26]), .bidir(io_data[26]));
332 bidirec gpio27 (.oe(en_gpio[27]), .inp(o_gpio[27]), .outp(i_gpio[27]), .bidir(io_data[27]));
333 bidirec gpio28 (.oe(en_gpio[28]), .inp(o_gpio[28]), .outp(i_gpio[28]), .bidir(io_data[28]));
334 bidirec gpio29 (.oe(en_gpio[29]), .inp(o_gpio[29]), .outp(i_gpio[29]), .bidir(io_data[29]));
335 bidirec gpio30 (.oe(en_gpio[30]), .inp(o_gpio[30]), .outp(i_gpio[30]), .bidir(io_data[30]));
336 bidirec gpio31 (.oe(en_gpio[31]), .inp(o_gpio[31]), .outp(i_gpio[31]), .bidir(io_data[31]));
337
338 gpio_top gpio_module(
339     .wb_clk_i      (clk),
340     .wb_rst_i      (wb_rst),
341     .wb_cyc_i      (wb_m2s_gpio_cyc),
342     .wb_adr_i      ({2'b0,wb_m2s_gpio_adr[5:2],2'b0}),
343     .wb_dat_i      (wb_m2s_gpio_dat),
344     .wb_sel_i      (4'b1111),
345     .wb_we_i       (wb_m2s_gpio_we),
346     .wb_stb_i      (wb_m2s_gpio_stb),
347     .wb_dat_o      (wb_s2m_gpio_dat),
348     .wb_ack_o      (wb_s2m_gpio_ack),
349     .wb_err_o      (wb_s2m_gpio_err),
350     .wb_inta_o     (gpio_irq),
351     // External GPIO Interface
352     .ext_pad_i     (i_gpio[31:0]),
353     .ext_pad_o     (o_gpio[31:0]),
354     .ext_padoe_o   (en_gpio));
355

```

**Figür 8. GPIO modülünün entegrasyonu (dosya swervolf\_core.v).**

Modülün arayüzü iki bloğa bölünebilir: Wishbone sinyalleri (Tablo 3), ki b SweRV EH1 Çekirdeğinin bir denetleyici/çevre birimi modülü kullanarak GPIO’yla iletişim kurmasını sağlar, dış I/O sinyalleri (Tablo 4).

**Tablo 3. Wishbone Sinyalleri**

Port	Genişlik	Yön	Tanım
wb_cyc_i	1	Inputs	Indicates valid bus cycle (core select)
wb_adr_i	15	Inputs	Address inputs
wb_dat_i	32	Inputs	Data inputs
wb_dat_o	32	Outputs	Data outputs
wb_sel_i	4	Inputs	Indicates valid bytes on data bus (during valid cycle it must be 0xf)
wb_ack_o	1	Output	Acknowledgment output (indicates normal transaction termination)
wb_err_o	1	Output	Error acknowledgment output (indicates an abnormal transaction termination)
wb_rty_o	1	Output	Not used
wb_we_i	1	Input	Write transaction when asserted high
wb_stb_i	1	Input	Indicates valid data transfer cycle
wb_inta_o	1	Output	Interrupt output

**Tablo 4. Dış I/O Sinyalleri**

Port	Genişlik	Yön	Tanım
in_pad_i	1-32	Inputs	GPIO inputs
out_pad_o	1-32	Outputs	GPIO outputs
oen_padoen_o	1-32	Outputs	GPIO output drivers enables (for three-state or open-drain drivers)

Figür 8'in satır 342'sinde gösterildiği gibi, *wb\_m2s\_gpio\_adr[5:2]* Wishbone veri yolu sinyalinde çekirdeğin sağladığı adresin 5:2 bitleri, 10 erişilebilir bellek eşlenmiş yazmaç arasından seçmek için kullanılır. Bu dört bit GPIO Çekirdeğine *wb\_adr\_i* sinyali üzerinden sağlanır (Figür 8'de gösterildiği gibi).

Girdi *ext\_pad\_i* direkt olarak GPIO Okuma Yazmacına bağlanır (RGPIO\_IN). Yine, çıktı *ext\_pad\_o* direkt olarak GPIO Yazma Yazmacına bağlanır (RGPIO\_OUT). Bu iki sinyal LEDlerle Anahtarlara (*i\_gpio*, *o\_gpio*, *io\_data*) 32 üç durumlu ara bellek modülü üzerinden bağlanır (Figür 8, 305-336 satırları). Böylelikle 32 ucun hepsi girdi ya da çıktı olarak yapılandırılabilir. Bizim durumumuzda alt 16 pin, 15:0, LEDlere bağlı, yani çıktı olarak yapılandırılmalı (Figür 7); üst 16 pin, 31:16, anahtarlara bağlı, yani girdi olarak yapılandırılmalı (Figür 7). Bu 32 üç durumlu ara belleği şu modülü içererek yapılandırıyoruz:

```
module bidirec (input wire oe, input wire inp, output wire outp, inout wire bidir);
    assign bidir = oe ? inp : 1'bZ ;
    assign outp = bidir;
endmodule
```

**GÖREVLER:** GPIO uçları (*io\_data*) GPIO modülüne üç durumlu ara bellekler ile bağlıdır (Figür 8). Üç durumlu belleği etkinleştirme sinyalinin iki durumu için çözümle (*oe=0*, *oe=1*).

GPIO modülüyle karttaki LEDlerin/Anahtarların bağlantısını göz önünde bulundurarak, programlamacı *en\_gpio*'ya hangi değerleri atamalıdır?

### iii. GPIO ile SweRV EH1 Çekirdeği arasındaki bağlantı

As shown in Figür 2, the device controllers are connected to the SweRV EH1 Core through a multiplexer and a bridge. The multiplexer selects one among the N possible peripherals (in our case, N=7), depending on the address generated by the CPU. The bridge translates the Wishbone signals used by the device controllers to the AXI4 signals used by the SweRV Core and vice versa (implemented in file `[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/AxiToWb/axi2wb.v`).

The 7:1 multiplexer (Figür 9) is implemented in file

`[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect/wb_intercon.v`, which is instantiated in lines 104-205 of file

`[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect/wb_intercon.vh`. This latter file is included in line 168 of the **swervolf\_core** module located here:

`[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'hfffff000, 32'hfffffc0, 32'hfffffc0, 32'hfffffc0, 32'hfffffc0, 32'hfffffc0, 32'hffff000}))
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 9. 7-1 çoklayıcısı CPU'ya bağlanacak çevre birimini seçer (`wb_intercon.v`).**

Çoklayıcı okunacak ya da yazılacak çevre birimini seçip, CPU'yu (`wb_io_*` sinyalleri –115-126 satırları, Figür 9) adrese bağlı olarak (110-111 satırları) bir çevre biriminin Wishbone Veri Yoluyla bağlar (127-138 satırları, Figür 9). Örneğin CPU'nun oluşturduğu adres 0x80001400-0x8000143F aralığında ise GPIO çevre birimi seçilir, dolayısıyla `wb_io_*` sinyalleri `wb_gpio_*` sinyalleriyle bağlanır.

Figür 10 çoklayıcının Verilog gerçekleştirmesini gösterir (şu dosyadan erişilebilir:

`[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect/wb_intercon_1.2.2/wb_mux.v`).

**GÖREV:** Çoklayıcının gerçekleştirmesini detaylıca çözümü. GPIO ilgili sinyallere (`wb_gpio_*`) odaklanabilirsin. Şu dosyaları incelemen gerekecek:

`[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals/SystemController/swervolf_syscon.v`

`[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect/wb_intercon.v`

`[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect/wb_intercon.vh`

`[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Interconnect/WishboneInterconnect/wb_intercon_1.2.2/wb_mux.v`

SoC'nin bu parçasını anlamak yalnızca bu deney değil, sonraki deneyler için de önemlidir. Sonraki bölümde gerçekleştirilen simülasyonlar eğer çoklayıcıyla ilgili yeni sinyaller ekleyerek genişletirsen anlama sürecinde yardımcı olabilir.



```

82 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
83 // Master/slave connection
84 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
85
86 //Use parameter instead of localparam to work around a bug in Xilinx ISE
87 parameter slave_sel_bits = num_slaves > 1 ? $clog2(num_slaves) : 1;
88
89 reg wbm_err;
90 wire [slave_sel_bits-1:0] slave_sel;
91 wire [num_slaves-1:0] match;
92
93 genvar idx;
94
95 generate
96   for(idx=0; idx<num_slaves ; idx=idx+1) begin : addr_match
97     assign match[idx] = (wbm_adr_i & MATCH_MASK[idx*aw+:aw]) == MATCH_ADDR[idx*aw+:aw];
98   end
99 endgenerate
100
101 //
102 // Find First 1 - Start from MSB and count downwards, returns 0 when no bit set
103 //
104 function [slave_sel_bits-1:0] ffl;
105   input [num_slaves-1:0] in;
106   integer i;
107
108   begin
109     ffl = 0;
110     for (i = num_slaves-1; i >= 0; i=i-1) begin
111       if (in[i])
112         /* verilator lint_off WIDTH */
113         ffl = i;
114       /* verilator lint_on WIDTH */
115     end
116   end
117 endfunction
118
119 assign slave_sel = ffl(match);
120
121 always @(posedge wb_clk_i)
122   wbm_err <= wbm_cyc_i & !(match);
123
124 assign wbs_adr_o = {num_slaves{wbm_adr_i}};
125 assign wbs_dat_o = {num_slaves{wbm_dat_i}};
126 assign wbs_sel_o = {num_slaves{wbm_sel_i}};
127 assign wbs_we_o = {num_slaves{wbm_we_i}};
128 /* verilator lint_off WIDTH */
129
130 assign wbs_cyc_o = match & (wbm_cyc_i << slave_sel);
131 /* verilator lint_on WIDTH */
132 assign wbs_stb_o = {num_slaves{wbm_stb_i}};
133
134 assign wbs_cti_o = {num_slaves{wbm_cti_i}};
135 assign wbs_bte_o = {num_slaves{wbm_bte_i}};
136
137 assign wbm_dat_o = wbs_dat_i[slave_sel*dw+:dw];
138 assign wbm_ack_o = wbs_ack_i[slave_sel];
139 assign wbm_err_o = wbs_err_i[slave_sel] | wbm_err;
140 assign wbm_rty_o = wbs_rty_i[slave_sel];
141
142 endmodule

```

**Figür 10. Wishbone çoklayıcısı (dosya *wb\_mux.v*).**

## B. Verilator Simülasyonu

Bu bölümde ilk olarak RVfpgaSIM simülatörünü yeni bir girdi sinyali ekleyerek değiştiriyoruz. Ardından RVfpgaSIM'i Verilator'le yeniden derleyip, simülatör basit bir program çalıştırırken bu yeni sinyali çözümlüyoruz.

### i. RVfpgaSIM'i Değiştirip Yeniden Derle

Simülasyonda gerçek LEDlerimiz ya da anahtarlarımız yok. Dolayısıyla, testbench'te (*[RVfpgaPath]/RVfpga/src/rvfpgasim.v*), anahtarları değiştirmeyi sinyale (*i\_sw*) 0xFE34 sabitini atayarak yapıyoruz (Figür 11'in solu). Ardından anahtarlar SweRVolf Çekirdeğine girdi olarak sağlanır (Figür 11'in sağı).

```
80 wire [15:0] i_sw;
81 assign i_sw = 16'hFE34; 248 .io_data ({i_sw,16'bz});
```

Figür 11. `i_sw` sinyali atanıp SweRVolf Çekirdeğine verilir, `rvfpgasim.v`.

İlk Kullanım Kılavuzundan anımsayacağın üzere testbench (`rvfpgasim.v`) girdi sinyallerini RVfpgaSIM için alıp (`clk`, `rst`, gibi gibi) (Figür 12'nin solu) `swervolf_core` modülünün somutlamasını yapar (Figür 12'nin sağı).

```
28 (input wire clk,
29 input wire rst,
30 input wire i_jtag_tck,
31 input wire i_jtag_tms,
32 input wire i_jtag_tdi,
33 input wire i_jtag_trst_n,
34 output wire o_jtag_tdo,
35 output wire o_uart_tx,
36 output wire o_gpio)

190 swervolf_core
191 #(.bootrom_file (bootrom_file))
192 swervolf
193 (.clk (clk),
194 .rstn (!rst),
195 .dmi_reg_rdata (dmi_reg_rdata),
```

Figür 12. RVfpgaSIM ile SweRVolf somutlaması için girdi sinyalleri (dosya `rvfpgasim.v`).

Kimi durumlarda simülatöre yeni girdi/çıkış sinyalleri eklemek isteyebilirsiniz. Örneğin, sonra RVfpgaSIM'e `i_sw0` adında bir sinyali nasıl içereceğini gösteriyoruz, ki bu en sağdaki anahtar için değer sağlar.

Sonraki adımları izle:

1. `[RVfpgaPath]/RVfpga/src/rvfpgasim.v` dosyasını değiştir:

a. `i_sw0` adında 1-bitlik yeni girdi sinyali içere. Figür 13'e göz at.

```
28 (input wire clk,
29 input wire rst,
30 input wire i_jtag_tck,
31 input wire i_jtag_tms,
32 input wire i_jtag_tdi,
33 input wire i_jtag_trst_n,
34 output wire o_jtag_tdo,
35 output wire o_uart_tx,
36 output wire o_gpio,
37 input wire i_sw0)
```

Figür 13. Yeni `i_sw0` girdi sinyali.

b. Bu sinyali en sağdaki anahtara sağla. Kalan anahtar değerlerini 0xFE34 olarak ata (önceki gibi bit 0 dışındakiler). Figür 14'e göz at.

```
82 wire [15:0] i_sw;
83 assign i_sw = {15'b111111100011010,i_sw0};
```

Figür 14. `i_sw0`'i en sağ anahtar olarak sağla.

2. `[RVfpgaPath]/RVfpga/src/OtherSources/tb.cpp` dosyasını değiştir: bu Verilator için ana C++ dosyasıdır. Bu dosyanın sonunda while döngüsünü bulabilirsin (Figür 15'te gösterilir), içerisinde bir yineleme bir saat darbesi oluşturur. Önemli olarak SoC için saat



sinyali bu döngüde ikili değeri tersine dönüştürerek oluşturulur (satır 178) (1→0 ya da 0→1). Ek olarak simülasyon süresi `main_time` (satır 180) değişkeninde nanosaniye biriminde ölçülür (saat dönümğ 20 ns'dir, dolayısıyla saat darbesi 10 ns). Son olarak simülasyon, simülasyon süreci `timeout` (173-176 satırlarında kırmızı kutu) çokluğuna erişince durur.

```

136 top->clk = 1;
137 top->rst = 1;
138
139 while (!(done || Verilated::gotFinish())) {
140     if (main_time == 100) {
141         printf("Releasing reset\n");
142         top->rst = 0;
143     }
144     if (main_time == 200)
145         top->i_jtag_trst_n = true;
146
147     top->eval();
148     if (tfp)
149         tfp->dump(main_time);
150     if (baud_rate) do_uart(&uart_context, top->o_uart_tx);
151     if (jtag && (main_time > 300)) {
152         int ret = jtag->doJTAG(main_time/20, //doJtag requires t to only increment by one
153             &top->i_jtag_tms,
154             &top->i_jtag_tdi,
155             &top->i_jtag_tck,
156             top->o_jtag_tdo);
157         if (ret != VerilatorJtagServer::SUCCESS) {
158             if (ret == VerilatorJtagServer::CLIENT_DISCONNECTED) {
159                 printf("Ending simulation. Reason: jtag_vpi client disconnected.\n");
160                 done = true;
161             }
162             else {
163                 printf("Ending simulation. Reason: jtag_vpi error encountered.\n");
164                 done = true;
165             }
166         }
167     }
168     if (gpio0 != top->o_gpio) {
169         printf("%lu: gpio0 is %s\n", main_time, top->o_gpio ? "on" : "off");
170         gpio0 = top->o_gpio;
171     }
172
173     if (timeout && (main_time >= timeout)) {
174         printf("Timeout: Exiting at time %lu\n", main_time);
175         done = true;
176     }
177
178     top->clk = !top->clk;
179
180     main_time+=10;
181 }

```

**Figür 15. Simülasyon için while döngüsü.**

Yeni `i_sw0` sinyaline döngüye girmeden önce ikili 0 değerini ata (Figür 16 solu), döngüde 30 us anında 1'e dönüştür (Figür 16 sağ).

<pre> 136 top-&gt;clk = 1; 137 top-&gt;rst = 1; 138 139 top-&gt;i_sw0 = 0; 140 141 while (!(done    Verilated::gotFinish())) { </pre>	<pre> 179 top-&gt;clk = !top-&gt;clk; 180 181 main_time+=10; 182 183 if (main_time == 30000) { 184     top-&gt;i_sw0 = 1; 185 } 186 </pre>
---	--

**Figür 16. `i_sw0` sinyaline değer ata.**

3. Bu değişikliklerin hepsini yapınca şu komutlarla RVfpgaSIM'i yeniden derle (bu GSG'de açıklanmıştı):

```

cd [RVfpgaPath]/RVfpga/verilatorSIM
make clean

```

make

Yeni dosya olarak *Vrvfpgasim* (RVfpga simulation binary)  
[RVfpgaPath]/RVfpga/verilatorSIM dizini içerisinde oluşturulmuş olmalı.

**WINDOWS:** Son adımı (adım 4) Cygwin terminalinde yapman gerek terminal (detaylı yönergeler için İlk Kullanım Kılavuzunda Bölüm 6 ile Ek C'ye bak). Önemli olarak Windows C: klasörü Cygwin'de şuradadır: */cygdrive/c*.

**MacOS:** Detaylı yönergeler için İlk Kullanım Kılavuzunda Ek D'ye bak.

## ii. *LedsSwitches.S* programının simülasyonunu çözümü

Bu bölümde RVfpga İlk Kullanım Kılavuzundaki *LedsSwitches.S* örnek programının simülasyonunu yapıyoruz (Figür 17). Bu program Nexys A7 kartındaki anahtarlardaki değerleri okuyup LEDlere yazar. Önemli olarak etkinleştirme yazmacını yapılandırmamız gerek, ki 32 girdi/çıkı ucu girdi ya da çıkı olarak bağlantılarına uygun yapılandırılın. Alt 16 uç GPIO'dan LEDlere bağlıdır, dolayısıyla bunlar CPU'ya göre çıkı uçlarıdır (Enable=1). Üst 16 uç GPIO'dan anahtarlara bağlıdır, dolayısıyla bunlar CPU'ya göre girdi uçlarıdır (Enable=0). Anahtarlar okuma yazmacının üst 16 bitini kapladığı için değerleri LEDe yazılmadan önce sağa kaydırılmalıdır.

```
#define GPIO_SWS    0x80001400
#define GPIO_LEDS   0x80001404
#define GPIO_INOUT  0x80001408

.globl main
main:

li x28, 0xFFFF
li x29, GPIO_INOUT
sw x28, 0(x29)                # Write the Enable Register

next:
    li a1, GPIO_SWS          # Read the Switches
    lw t0, 0(a1)

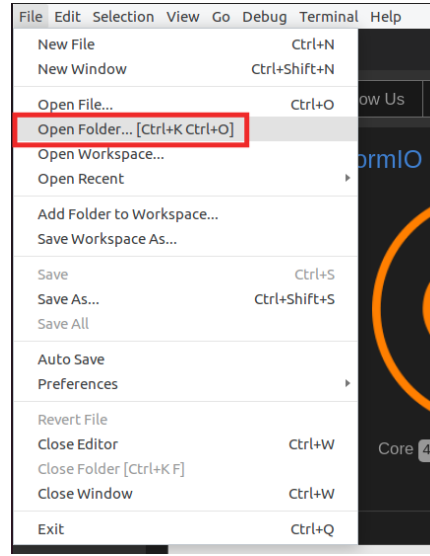
    li a0, GPIO_LEDS
    srl t0, t0, 16
    sw t0, 0(a0)              # Write the LEDs

    beq zero, zero, next
.end
```

**Figür 17. Genişletilmiş SweRVolf Çekirdeğinde çalıştırmak için LedsSwitches.s**

Simülasyonu çalıştırmak için sonraki adımları izle.

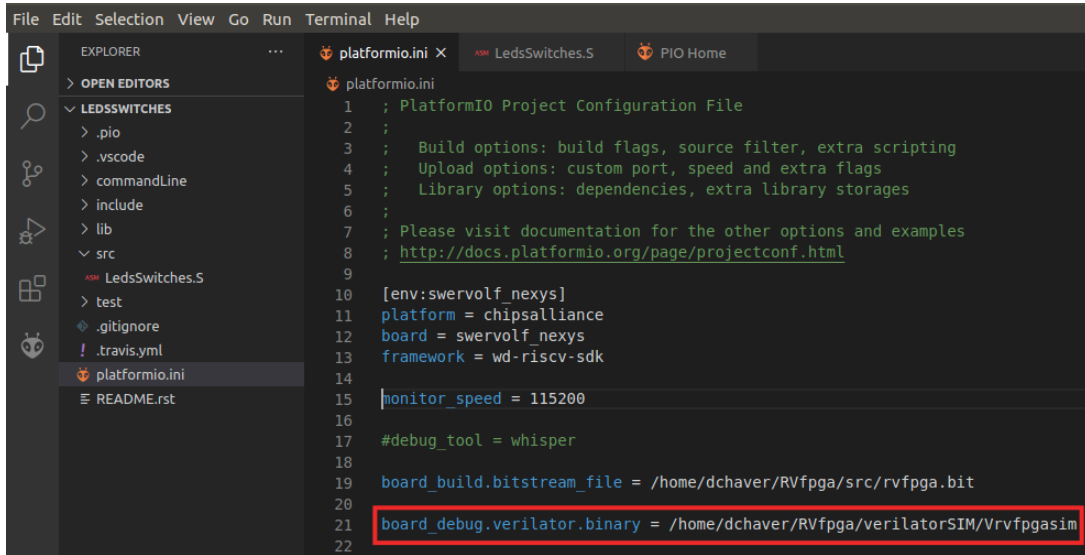
1. Bilgisayarında VSCode/PlatformIO aç.
2. Üst çubukta *File (Dosya)* → *Open Folder (Klasörü Aç)*... üzerine tıkla (Figür 18),  
[RVfpgaPath]/RVfpga/examples/ dizinine git.



**Figür 18. LedsSwitches.S örneğini aç**

3. *LedsSwitches* (açma, yalnızca seç) dizinini seçip OK'a tıkla. Örnek PlatformIO'da açılacaktır.
4. *platformio.ini* dosyasını açıp RVfpga simülasyon ikili dosyasına giden yolun doğru olup olmadığını denetle (Figür 19) (önceki bölümde adım 3). GSG'den anımsayacağın üzere şöyle görünmelidir:


```
board_debug.verilator.binary =  
[RVfpgaPath]/RVfpga/verilatorSIM/Vrvfpgasim
```



**Figür 19. Platformio ilk değerlendirme dosyası: platformio.ini**

**Windows:** RVfpga simülasyon yürütülebilirine *Vrvfpgasim.exe* denir. Dolayısıyla:

```
board_debug.verilator.binary = [RVfpgaPath]\RVfpga\verilatorSIM\Vrvfpgasim.exe
```

5. Sol menü şeridindeki PlatformIO ikonuna tıklayarak simülasyonu çalıştır , ardından Project Tasks (Proje Görevleri) → env:swervolf\_nexys → Platform genişletip Generate Trace (İz Oluştur) üzerine tıkla.

trace.vcd dosyası

[RVfpgaPath]/RVfpga/examples/LedsSwitches/.pio/build/swervolf\_nexys içinde oluşturulmuş olmalıdır, PlatformIO terminaline aşağıdaki komutu yazarak GTKWave ile açabilirsiniz.

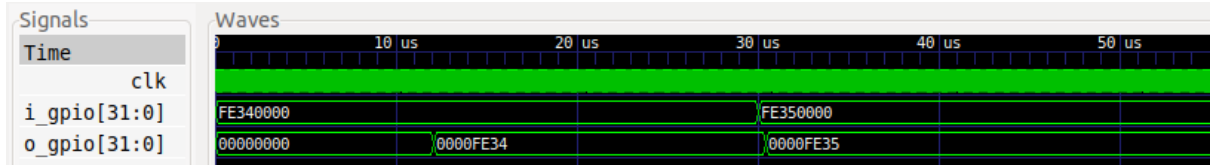
```
gtkwave [RVfpgaPath]/RVfpga/examples/LedsSwitches/.pio/build/swervolf_nexys/trace.vcd
```

**WINDOWS:** indirdiğin gtkwave64 klasörü, bin klasörünün içinde gtkwave.exe adlı bir uygulama içerir. Bu uygulamaya çift tıklayarak GTKWave'i başlat. Uygulamanın üst kısmında Dosya - Yeni Sekme Aç'a tıkla, [RVfpgaPath]/RVfpga/examples/LedsSwitches/.pio/build/swervolf\_nexys klasöründe oluşturulan trace.vcd dosyasını aç.

6. İzde şu sinyalleri içer (bu sinyalleri bulmak için *rvfpgasim-swervolf* modülünün içerisine git):

- Saat sinyalini ekle: **clk**
- GPIO girdi sinyalini ekle: **i\_gpio**
- GPIO çıktı sinyalini ekle: **o\_gpio**

Grafikte (Figür 20) 16 anahtarın değerinin (**i\_gpio** sinyalinin 16 yüksek önemli biti) is 16 LEDlere (**o\_gpio** sinyalinin 16 düşük önemli biti) biraz gecikmeyle kopyalandığını göreceksin. Sonra, en sağdaki anahtar (0→1) 30us anında değişir, bu en sağdaki LEDin biraz gecikmeyle değişmesine neden olur.



**Figür 20. LedsSwitches programının simülasyonu**

## 7. İLERİ DÜZEY ALIŞTIRMALAR

**Alıştırma 2.** Önceki bölümdeki simülasyonu detaylıca çözümü. Figür 21.elf *LedsSwitches* programının tersine çeviri sürümünü gösterir (Figür 17), GPIO yazmaçlarına (Etkinleştirme, Okuma, Yazma) erişen üç yönerge parlatılmıştır. İlk Kullanım Kılavuzundan anımsayacağın üzere derleme süresinde şu klasörde oluşturulan *firmware.dis* dosyasını açarak PlatformIO'da kolayca .elf programının tersine çeviri sürümünü görüntüleyebilirsiniz: [RVfpgaPath]/RVfpga/examples/LedsSwitches/.pio/build/swervolf-nexys/ (Figür 21).

```

> PSP
> src
  ≡ .sconsign36.dblite
  ≡ firmware.bin
  ≡ firmware.dis
  ≡ firmware.elf
  ≡ LedsSwitches.map
  ≡ libBoardBSP.a
  ≡ libPSP.a
  ≡ project.checksum
> libdeps
> .vscode
> commandLine
> include
> lib
> src
65 Disassembly of section .text:
66
67 00000090 <main>:
68 90: 00010e37      lui t3,0x10
69 94: fffe0e13      addi t3,t3,-1 # ffff <_sp+0xcebf>
70 98: 80001eb7      lui t4,0x80001
71 9c: 408e8e93      addi t4,t4,1032 # 80001408 <OVERLAY_END_OF_OVERLAYS+0xa0001408>
72 a0: 01cea023      sw t3,0(t4)
73
74 000000a4 <next>:
75 a4: 800015b7      lui a1,0x80001
76 a8: 40058593      addi a1,a1,1024 # 80001400 <OVERLAY_END_OF_OVERLAYS+0xa0001400>
77 ac: 0005a283      lw t0,0(a1)
78 b0: 80001537      lui a0,0x80001
79 b4: 40450513      addi a0,a0,1028 # 80001404 <OVERLAY_END_OF_OVERLAYS+0xa0001404>
80 b8: 0102d293      srli t0,t0,0x10
81 bc: 00552023      sw t0,0(a0)
82 c0: fe0002e3      beqz zero,a4 <next>

```

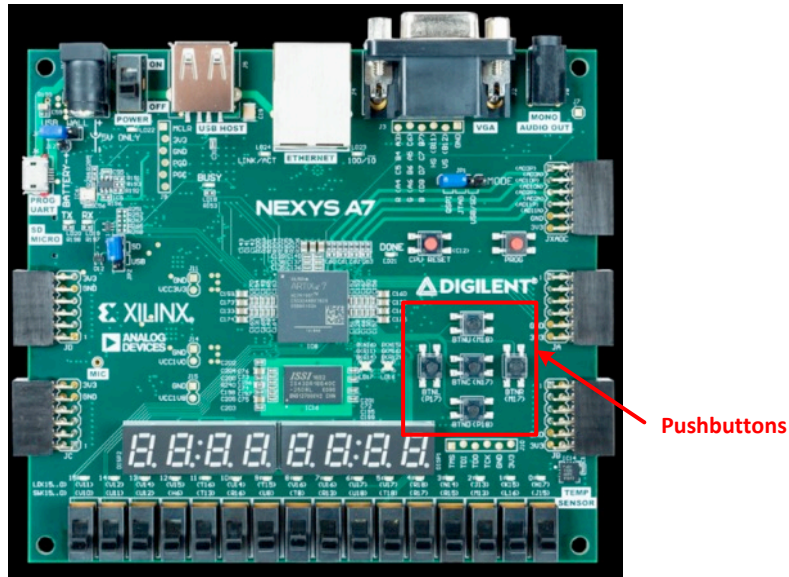
**Figür 21. LedsSwitches.S programının tersine çeviri sürümü**

Bu programın simülasyonunu RVfpgaSIM’de simülasyonunu yapıp Figür 21’de kırmızıyla gösterilen üç bellek yönergesinin yürütmesi sırasında GPIO sinyallerini çözümle (*sw*, *lw*, *sw*). Bu Bölüm A’da açıklanan GPIO alçak-düzeyleli gerçekleştirmesini anlamanda yardımcı olacaktır.

Bölüm B’deki simülasyondan başlayıp şu sinyaller için değerleri ekleyip çözümleyebilirsin (sinyallerin yerini bulmak için adı geçen modüllere git):

- rvfpgasim → swervolf → swerv\_eh1 → swerv → ifu
  - Saat: **clk**.
  - Getirilen yönergeler: **ifu\_i0\_instr** and **ifu\_i1\_instr**.
- rvfpgasim – swervolf
  - 32-bit input/output pins: **i\_gpio** and **o\_gpio**.
  - Address provided by the CPU: **wb\_m2s\_io\_adr**.
- rvfpgasim – swervolf – gpio\_module
  - GPIO External Interface: **ext\_pad\_i**, **ext\_pad\_o** and **ext\_padoe\_o**.
- rvfpgasim – swervolf – wb\_intercon0
  - Output address and data signals for the multiplexer of Figür 2: **wb\_io\_adr\_i**, **wb\_io\_dat\_i**, **wb\_io\_dat\_o**.
  - Input GPIO data signals for the multiplexer of Figür 2: **wb\_gpio\_adr\_i**, **wb\_gpio\_dat\_i**, **wb\_gpio\_dat\_o**.
  - Selection signals for the multiplexer of Figür 2: **wb\_\*\_cyc\_o**.
- rvfpgasim – swervolf – wb\_intercon0 – wb\_mux\_io
  - Match signal for the multiplexer of Figür 2: **match**.
- rvfpgasim – swervolf – rvtop – swerv – dec – arf – gpr\_banks(0) – gpr(5) – gprff
  - Register value for t0: **dout**.

**Alıştırma 3.** Beş düğmeyi desteklemek için RVfpga’i genişlet. Düğmeler Figür 22’de gösterilmektedir. Beş düğme konumlarına göre adlandırılır: yukarı, aşağı, sol, sağ, orta - BTNU, BTND, BTNL, BTNR, BTNC.



**Figür 22. Nexys A7 FPGA Kartındaki düğmeler**

- Kullandığımız GPIO modülünün (`gpio_top`) en yükseği 32 olduğundan, ki bu bizdeki I/O uçlarının sayısıdır (16 LEDs + 16 Anahtarlar), RVfpga'de bir diğer GPIO modülünün, 5 yeni üç durumlu ara bellekle gerekli sinyallerinin somutlamasını içermen gerekecek.
- 0x80001800'de (erişilebilir adresler) başlayan adresleri yeni GPIO denetleyicisinin açığa çıkardığı yazmaçları eşlemek için kullan. Önemli olarak çoklayıcıyı yeni çevre birimini içermek üzere değiştirmen gerek (Figür 9).
- Kısıtlandırma dosyasını şu FPGA uçlarına bağlı beş düğmeyi göz önüne alacak biçimde değiştirmen de gerek:
  - BTNC, PIN N17'ye bağlı
  - BTNU, PIN M18'e bağlı
  - BTNL, PIN P17'ye bağlı
  - BTNR, PIN M17'ye bağlı
  - BTND, PIN P18'e bağlı

**Alıştırma 4.** Karttaki beş düğme için **RVfpga**'de yeni bir denetleyici tasarla.

- Alıştırma 3'e kıyasla bu kez Figür 3'teki şemaya uygun olarak kendi GPIO denetleyicini Verilog ya da SystemVerilog'da gerçekleştirmen gerek. Devreyi basitleştirip yalnızca bir **Okuma Yazmacı** içerebilirsin (bir diğer deyişle üç durumlu ara bellekler ya da **Yazma Yazmacı** gerekmez).
- Önceki alıştırmadaki denetleyiciyi silmen gerekmez, düğmeler o GPIO denetleyicisinin kullanmadığı adreslere eşlenebilir.
- Sistem Denetleyicisi çevre biriminde yeni bir denetleyici içer. Kullanılmayan 0x80001018-0x8000101F adres aralığını kullanabilirsin. Önemli olarak Sistem Denetleyicisinde içerilen yazmaçlar CPU'ya direkt olarak Wishbone Veri Yolunun veri sinyaline CPU'nun oluşturduğu adrese (`i_wb_adr`) uygun bağlanarak okunur (`o_wb_rdt`). Nasıl ilerleyeceğini anlamak için **swervolf\_syscon** modülünün 234-266 satırlarını incele (`[RVfpgaPath]/RVfpga/src/SweRVolfSoC/Peripherals/SystemController/swervolf_syscon.v`).

**Alıştırma 5.** 1'den başlayarak gittikçe artan iki taban sayıyı LEDlerde gösteren bir RISC-V çevirici programıyla C programı yaz. Değeri arttırmadan önce geciktirmek için boş bir döngü içer ki değerler insan gözüyle görülebilsin. BTNC'yi Alıştırma 3'te gerçekleştirilen OpenCores çevre birimiyle okuyup, sayma işleminin hızını değiştirmek için kullan, BTNU'yu Alıştırma 4'te gerçekleştirilen geçici çevre birimiyle okuyup basıldığında sayacı yeniden başlatmak için kullan.