Tang Edu Board Kullanım Kılavuzu

Tang Nano 9K Tabanlı Eğitim Kartı

Beti Elektronik

Üniversite Laboratuvarları ve Öğrenciler

Sürüm: Taslak
Tarih: 2026

İçindekiler

Bölüm 1

Giriş

Tang Edu Board, Beti Elektronik tarafından Sipeed Tang Nano 9K FPGA modülü çevresinde tasarlanmış bir eğitim kartıdır. Amacı, dijital tasarım derslerinde görülen konuları ek kablolama gerektirmeden gerçek donanım üzerinde denenebilir hale getirmektir: LEDler, butonlar, slide switchler ve 7-segment display kartın üzerinde hazırdır.

1.1 Kılavuzun Amacı

Bu kılavuz bir ders kitabı değildir; dijital tasarım teorisini derste gördüğünüzü varsayar ve o bilgiyi kart üzerinde çalışan tasarımlara dönüştürmenin yolunu gösterir. Bölümler bir öğrenme sırası izler:

- Bölüm 2 kartı ve üzerindeki birimleri tanıtır,
- Bölüm 3 pin eşlemelerini ve constraint dosyalarını verir,
- Bölüm 4 Gowin IDE akışını ve temel VHDL yapısını anlatır,
- Bölüm 5 LED, buton ve switch ile ilk uygulamaları yaptırır,
- Bölüm 6 ders konularının kart üzerindeki karşılıklarını gösterir,
- Bölüm 7 ise 7-segment display'i, Tang Nano'nun HDMI çıkışını ve genişleme konnektörlerini toplar.

İlk kez FPGA kullanıyorsanız bölümleri sırayla izleyin. Daha önce FPGA görmüş bir kullanıcı için Bölüm 3 ve Bölüm 5 başlangıç için yeterlidir; gerisi ihtiyaç oldukça açılıp bakılacak şekilde yazılmıştır.

1.2 Hedef Kullanıcı

Kılavuzun birincil kullanıcısı, dijital tasarım dersini almakta olan ve ilk kez FPGA programlayacak üniversite öğrencisidir. Laboratuvar asistanları ve eğitmenler için de deney hazırlarken başvurulacak bir referans olarak düzenlenmiştir; özellikle Bölüm 3.2'deki aktif seviye tablosu ve Bölüm 6'daki deney fikirleri bu amaçla kullanılabilir.

Ön koşul olarak temel mantık devreleri bilgisi (kapılar, flip-flop kavramı) yeterlidir. VHDL deneyimi beklenmez; gereken asgari özet Bölüm 4.5'te verilmiştir.

Bölüm 2

Donanım Genel Özellikleri

Bu bölümde Tang Nano 9K ve Tang Edu Board donanımı, kartı güvenli ve doğru kullanabilmeniz için gerekli seviyede anlatılır; datasheet düzeyinde ayrıntıya girilmez.

2.1 Tang Nano 9K

Tang Nano 9K, Sipeed'in ürettiği küçük bir FPGA modülüdür ve Tang Edu Board'un beynidir. Üzerinde Gowin GW1NR-9 ailesinden `GW1NR-LV9QN88PC6/I5` FPGA'sı bulunur; 8640 LUT'luk bu FPGA, kılavuzdaki tüm örnekler ve çok daha kapsamlı projeler için yeterlidir.

Modül tek USB-C kablosuyla hem beslenir hem programlanır (üzerindeki USB-JTAG dönüştürücü sayesinde ek programlayıcı donanımı gerekmez). 27 MHz osilatörü tasarımların sistem clock'udur (Bölüm 6.2.4); bitstream, FPGA'nın dahili flash'ına yazıldığında kart her enerjilendiğinde tasarımınızla açılır.

Modülün kendi üzerinde de birkaç küçük LED ve buton bulunur; ancak bu kılavuzdaki örnekler, erişimi ve gözlemi çok daha rahat olan Edu Board birimlerini kullanır. Modülün HDMI konnektörü gibi ek olanakları Bölüm 7'de ele alınır.

2.2 Tang Edu Board

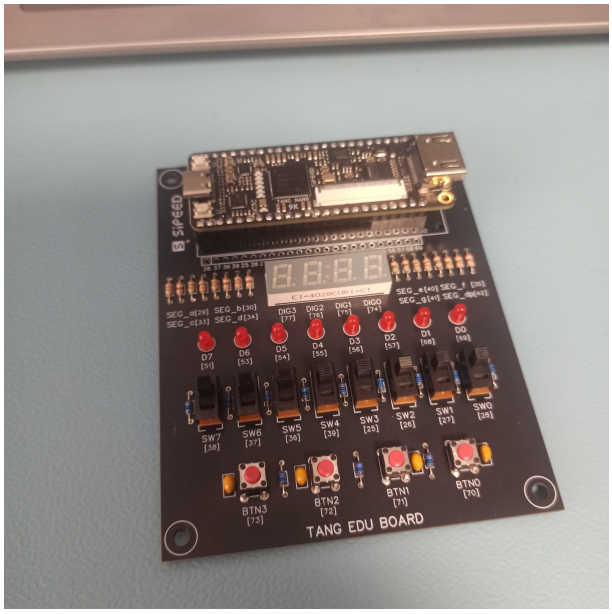
Tang Edu Board, Tang Nano 9K'nın pinlerini eğitimde en sık kullanılan giriş ve çıkış birimlerine dağıtan taşıyıcı karttır; modül, kart üzerindeki soketine takılır.

Kart üzerindeki birimler ve ayrıntılarının anlatıldığı bölümler:

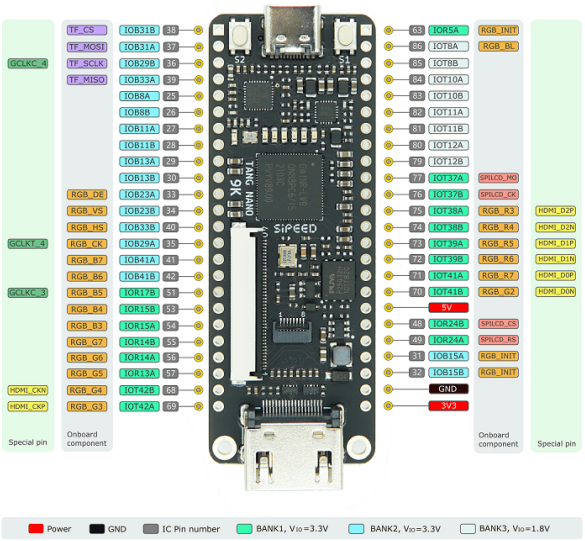
- **8 adet LED** — çıkışları gözlemlemenin en hızlı yolu (Bölüm 5.1),
- **4 adet buton** — donanım debounce'lu anlık girişler (Bölüm 5.2),
- **8 adet slide switch** — konumunu koruyan kalıcı girişler (Bölüm 5.3),
- **4 haneli 7-segment display** — sayısal değer gösterimi (Bölüm 7.1),
- **2 adet 24-pin genişleme konnektörü** — harici devre bağlantıları (Bölüm 7.3).

Tüm birimlerin FPGA pin eşlemeleri Bölüm 3'te, aktif seviye davranışları Bölüm 3.2'dedir. Temel uygulamalar için karta herhangi bir ek bağlantı yapmanız gerekmez.

Dikkat
Genişleme konnektörlerine harici devre bağlamadan önce Bölüm 7.3'deki gerilim uyarılarını mutlaka okuyun; bazı pinler 3.3 V değil 1.8 V seviyesindedir ve yanlış bağlantı FPGA'ya kalıcı zarar verebilir.



(a) Tang Edu Board — taşıyıcı kartın tamamı (üstte Tang Nano 9K modülü takılı).



(b) Yalnızca Tang Nano 9K modülünün pinout şeması.

Şekil 2.2.1: Tang Edu Board ve üzerine takılan Tang Nano 9K modülü.

Bölüm 3

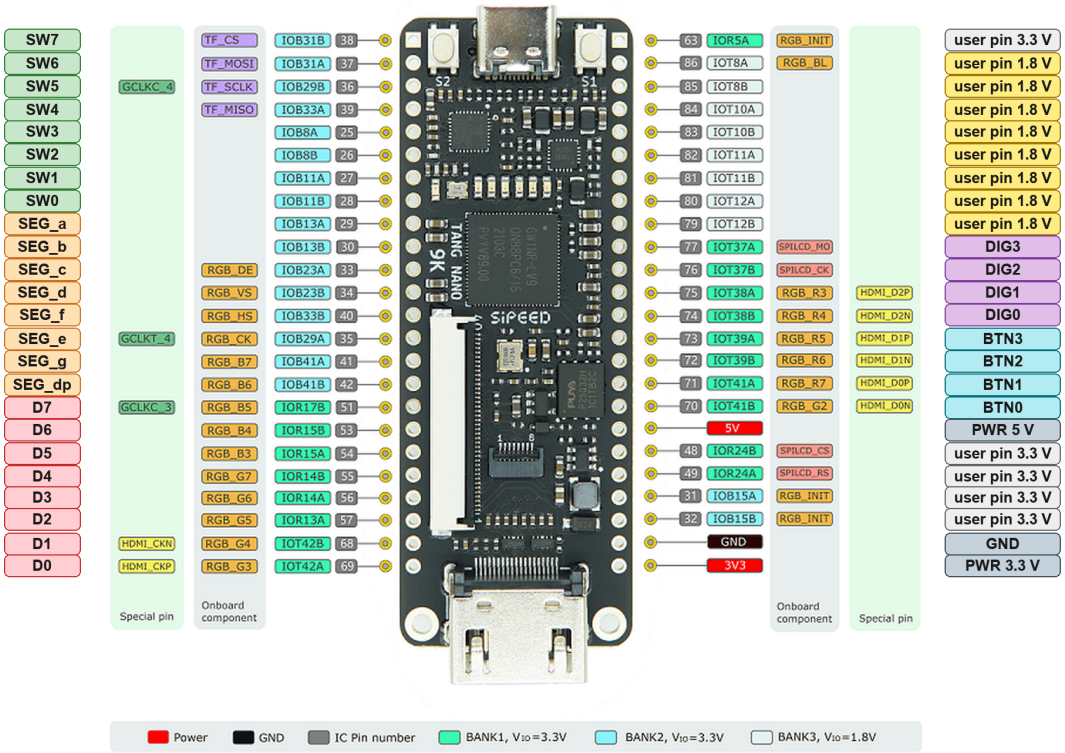
Pin Diyagramı ve Bağlantılar

Bu bölüm, Tang Edu Board üzerindeki çevre birimlerinin FPGA pinleriyle nasıl eşleştiğini açıklar.

Not
Pin bilgileri yazılırken ana kaynak `../CODE_PART/TANG_EDU_BOARD.cst` dosyası ve Beti schematic REV 1.0 olmalıdır. Silkscreen ile schematic çelişirse schematic esas alınır.

3.1 Pin Haritası

Şekil 3.1.1, Tang Edu Board üzerindeki her çevre biriminin Tang Nano 9K'nın hangi pinine bağlı olduğunu, pinlerin konnektörlerdeki *fiziksel sırasıyla* gösterir. Kutulardaki numaralar FPGA pin numaralarıdır ve constraint dosyasında (Bölüm 3.3) aynen kullanılır.



Şekil 3.1.1: Tang Edu Board çevre birimlerinin Tang Nano 9K pinlerine eşlenmesi (üstten görünüm, USB-C yukarıda). Dış sütunlardaki etiketler kartın üzerindeki yazılarla (silkscreen) eşleşir; VHDL port karşılıkları aşağıdaki tablodadır. User pinler konnektörlerden serbestçe kullanılabilir.

Aynı bilginin port indeksine göre sıralı özeti:

Birim	Port	FPGA pinleri (indeks sırasıyla)
LEDler (D0–D7)	`led[7:0]`	led0→led7: 69, 68, 57, 56, 55, 54, 53, 51
Slide switchler (SW0–SW7)	`sw[7:0]`	sw0→sw7: 28, 27, 26, 25, 39, 36, 37, 38
Butonlar (BTN0–BTN3)	`btn[3:0]`	btn0→btn3: 70, 71, 72, 73
7-seg segmentleri (SEG_a–dp)	`seg[7:0]`	a, b, c, d, e, f, g, dp: 29, 30, 33, 34, 35, 40, 41, 42
7-seg digit seçimi (DIG0–DIG3)	`dig[3:0]`	dig0→dig3 (sağ haneden sola): 74, 75, 76, 77
Clock	`clk`	52 (modül üzerindeki 27 MHz osilatör)

Şemadaki gri ve sarı etiketli *user pin*'ler hiçbir kart birimine bağlı değildir; iki adet 24-pin konnektör üzerinden harici devreler için doğrudan kullanılabilir. Sarı etiketli 79–86 pinleri FPGA'nın 1.8 V bankındadır: bu pinlere 3.3 V sinyal uygulamayın. User pinlerin güvenli kullanımı Bölüm 7.3'de ele alınır.

3.2 Aktif Seviye Bilgileri

Dijital tasarımda `'1'` yazmak her zaman "aç" anlamına gelmez; bir birimin hangi seviyede aktif olduğunu yazılım değil, kartın donanım bağlantısı belirler. Aşağıdaki tablo bu kılavuzun en sık dönüp bakacağınız tablosudur:

Birim	Aktif seviye	Pratik anlamı
LEDler	active-LOW	`'0'` yakar, `'1'` söndürür
Butonlar	active-LOW	basılıyken `'0'`, serbestken `'1'`
Slide switchler	—	bir konum `'0'`, diğer konum `'1'`
7-segment (segmentler)	active-HIGH	`'1'` segmenti yakar
7-segment (digit seçimi)	active-LOW	`'0'` yazılan hane aktif olur

Nedenleri donanımdadır: LEDlerin ortak ucu 3.3 V'a bağlıdır, bu yüzden pin `'0'` olduğunda akım akar ve LED yanar. Buton hatlarında pull-up direnci vardır; basmak hattı GND'ye çeker. Ayrıca her butonda RC devresiyle *donanım debounce* yapılmıştır, yazılımda ek debounce gerekmez. Switch hatlarındaki harici pull-up nedeniyle bir konum hattı GND'ye çeker (`'0'`), diğer konumda hat `'1'` okunur. 7-segment display *common cathode* yapıdadır: bir hanenin ortak katodu `'0'` ile seçilir, seçili hanenin segmentleri `'1'` ile yanar.

> **Not**
> Bölüm 5'teki örneklerde görülen `not` terslemelerinin tamamı bu tablodan türetilir. Bir birim "ters çalışıyor" gibi göründüğünde önce buraya dönün.

3.3 Constraint Dosyasının Kullanımı

Sentezleyici, tasarımınızdaki port isimlerini bilir ama bu portların FPGA'nın hangi fiziksel bacağına gideceğini bilemez. Bu eşlemeyi `.cst` uzantılı *physical constraint* dosyası yapar. Her port için iki satır yeterlidir:

Kod 3.3.1: Bir portun pine eşlenmesi

```
IO_LOC  "led[0]" 69;                 // hangi fiziksel pin
IO_PORT "led[0]" IO_TYPE=LVCMOS33;   // elektriksel standart (3.3 V)
```

`IO_LOC` portu pine bağlar; `IO_PORT` o pinin gerilim standardını ve sürücü ayarlarını belirler. Dikkat: VHDL'de `led(0)` diye yazılan vektör elemanı, constraint dosyasında `led[0]` diye köşeli parantezle yazılır. Dosya, Gowin projesine normal bir kaynak dosyası gibi eklenir (*Add Files*) ve Place & Route bu eşlemeye göre yapılır.

Pin eşlemesi tek başına yeterli değildir: araçların tasarımın *zamanında* çalıştığını denetleyebilmesi için clock'un frekansını da bilmesi gerekir. Bunu ikinci bir constraint dosyası, `.sdc` uzantılı *timing constraint* dosyası yapar ve bu karttaki projelerde tek satırdan oluşur:

Kod 3.3.2: Timing constraint: 27 MHz clock tanımı

```
// 27 MHz -> periyot = 37.037 ns
create_clock -name clk -period 37.037 -waveform {0 18.518} [get_ports {clk}]
```

`-period` periyodu ns cinsinden verir; `-waveform {0 18.518}` sinyalin 0. ns'de yükselip periyodun yarısında düştüğünü, yani %50 doluluk oranını tanımlar. Bu değerlerin fiziksel karşılığı — kare dalga, periyot ve yükselen kenar kavramları — Bölüm 6.2.4'te anlatılır. Araçlar bu tanımla, flip-floplar arasındaki her yolun 37 ns içinde tamamlandığını doğrular.

Tang Edu Board için her iki master dosya da (`TANG_EDU_BOARD.cst` ve `TANG_EDU_BOARD.sdc`) kılavuz ekinde verilmiştir. Master `.cst` kartın *tamamını* tanımlar ve olduğu gibi kopyalanmak için değildir: Gowin, constraint dosyasında geçen ama tasarımda bulunmayan bir port gördüğünde Place & Route aşamasında "net not found" benzeri bir hatayla durur. Doğru kullanım, projenizin portlarına karşılık gelen satırları master dosyadan seçip almaktır; `clk` satırına ise hemen her proje ihtiyaç duyar.

Dikkat

`led[3]`, `led[4]` ve `led[5]` (pin 56/55/54), GW1NR-9'un SSPI dual-purpose pinleridir. Bu pinleri kullanan her projede *Project* → *Configuration* → *Dual-Purpose Pin* altından *"Use SSPI as regular IO"* işaretlenmelidir; aksi halde Place & Route, PR2017/PR2028 hatasıyla durur.

Bölüm 4

Yazılım: Gowin IDE Kurulumu, Kullanımı ve Lisanslama

Bu bölüm, öğrencinin Tang Edu Board ile çalışmaya başlaması için gerekli Gowin IDE kurulum ve kullanım adımlarını açıklar.

4.1 Kurulum

Doldurulacak
Bu bölüm, yeni başlayan bir stajyerin temiz kurulumu sırasında çekilecek ekran görüntüleriyle yazılacak. İçerik kaynağı: `Gowin_EDA_Install_Guide.md`. Çekilecek ekranlar: (1) Gowin hesap/indirme sayfası, Education Edition seçili; (2) installer bileşen seçimi (IDE + Programmer, tümü işaretli); (3) USB/JTAG driver kurulum onayı; (4) kurulum sonu ve masaüstü kısayolu; (5) Sipeed programmer değişimi (indirme sayfası + klasör).

4.2 Lisanslama

Doldurulacak
Education Edition lisans gerektirmez — bölümün ana mesajı bu olacak, kısa tutulacak. Çekilecek ekranlar: (1) ilk açılışta License Manager penceresi; (2) Standard Edition'a geçilirse Float Lic ayarı (Sipeed sunucusu: `gowinlic.sipeed.com`, port 10559).

4.3 Yeni Proje Oluşturma

Doldurulacak
Çekilecek ekranlar: (1) File $\rightarrow$ New $\rightarrow$ FPGA Design Project; (2) cihaz seçimi ekranında `GW1NR-LV9QN88PC6/I5` (GW1NR-9C) seçili hali; (3) VHDL dosyası ekleme (Add Files); (4) .cst ve .sdc eklendikten sonra proje ağacı. Constraint içerikleri için Bölüm 3.3'e referans verilecek.

4.4 Derleme ve Programlama

Doldurulacak
Çekilecek ekranlar: (1) Process panelinde Synthesize ve Place & Route; (2) başarılı akış sonrası yeşil işaretler; (3) Programmer penceresi, kart algılanmış ve .fs dosyası seçili; (4) SRAM Program (geçici) ile embFlash (kalıcı) mod seçimi. SSPI dual-purpose ayarı gerektiren durum için Bölüm 3.3'teki uyarıya referans verilecek.

4.5 Temel VHDL Yapısı

Bu kılavuzdaki tüm uygulamalar VHDL ile yazılmıştır. Uygulama bölümlerinde her seferinde tekrar anlatmamak için, bir VHDL dosyasının temel yapı taşları burada kısaca özetlenir. Her VHDL tasarımı iki ana parçadan oluşur: dış dünyaya açılan kapıları tanımlayan **entity** ve bu kapılar arasındaki davranışı tanımlayan **architecture**.

Kod 4.5.1: Bir VHDL dosyasının temel yapısı

```
library ieee;                    -- standart kutuphane
use ieee.std_logic_1164.all;     -- std_logic tipleri icin gerekli

entity ornek_tasarim is
    port (
        sw  : in  std_logic;                    -- giris : tek bit
        btn : in  std_logic;                    -- giris : tek bit
        led : out std_logic_vector(1 downto 0)  -- cikis : 2 bitlik demet
    );
end entity ornek_tasarim;

architecture rtl of ornek_tasarim is
    signal ara_sinyal : std_logic;  -- ic baglanti, disari cikmaz
begin
    ara_sinyal <= sw and btn;       -- eszamanli atama
    led(0)     <= ara_sinyal;
    led(1)     <= not ara_sinyal;
end architecture rtl;
```

Kısaca:

- **entity** tasarımın port listesini tanımlar. Her portun bir yönü ve bir tipi vardır: butonlar ve switchler gibi girişler `in`, LEDler gibi çıkışlar `out` olur.
- **Tipler:** `std_logic` tek bir teli, `std_logic_vector` bir tel demetini temsil eder. `(7 downto 0)` yazımı 8 telli bir demet tanımlar; en soldaki bit 7 numaralı, en sağdaki 0 numaralı teldir.
- **architecture** içindeki `<=` işareti atama değil *bağlantı* olarak okunmalıdır. Satırlar bir program gibi sırayla değil, donanımda hepsi aynı anda çalışır; bu yüzden satırların sırası önemsizdir.
- **signal**, portlarda görünmeyen iç bağlantılar (ara teller) için kullanılır ve `architecture` ile `begin` arasında tanımlanır.

Eşzamanlı atamalar yalnızca "şu tel şuna bağlı" diyebilir; bir değerin *saklanması* gerektiğinde (sayaç, kaydırmalı register, durum makinesi) **process** kullanılır:

Kod 4.5.2: Clock ile çalışan process kalıbı

```
process(clk)                     -- duyarlilik listesi
begin
```

```
    if rising_edge(clk) then        -- clock'un yukselen kenarinda
        sayac < = sayac + 1;         -- deger saklanir ve guncellenir
    end if;
end process;
```

Parantez içindeki *duyarlılık listesi*, process'in hangi sinyal değiştiğinde değerlendirileceğini söyler. `rising_edge(clk)` koşulu sayesinde içerideki atamalar yalnızca clock'un yükselen kenarında gerçekleşir; bu kalıp donanımda flip-flop üretir. Bu kılavuzdaki hafıza gerektiren tüm örnekler bu kalıbı kullanır.

Not

Entity'deki port isimleri, constraint dosyasındaki isimlerle birebir aynı olmalıdır (bkz. Bölüm 3.3); aksi halde pinler eşleşmez. Process'in sayaç, shift register ve state machine gibi kullanımları Bölüm 6'da ilgili konularla birlikte gösterilir.

Bölüm 5

Temel Kart Uygulamaları

Bu bölümdeki uygulamalar, Tang Edu Board üzerindeki temel giriş ve çıkış birimlerini tek tek tanımanızı sağlar. Her uygulama aynı şablonla anlatılır: önce *amaç*, ardından *kullanılan pinler* ve *aktif seviye bilgisi*, sonra *VHDL mantığı* ve son olarak *kartta beklenen davranış*.

> **Not**
> Pin numaralarının tam listesi ve elektriksel ayrıntılar Bölüm 3'tedir. Proje oluşturma, derleme ve karta yükleme adımları her uygulamada tekrar edilmez; bu adımlar için Bölüm 4'e bakınız. Entity, architecture, port ve signal gibi temel VHDL kavramları için Bölüm 4.5'e bakınız.

5.1 LEDler

Amaç Kart üzerindeki sekiz LED'i sabit bir desenle yakmak. Bu, kartla yapacağınız ilk tasarımdır: tek bir çıkış portu ile sentez, yerleştirme ve programlama akışının tamamını uçtan uca denemiş olursunuz.

Kullanılan Pinler `led(7 downto 0)` portu; `led(0)`–`led(7)` sırasıyla 69, 68, 57, 56, 55, 54, 53 ve 51 numaralı FPGA pinlerine bağlıdır.

Aktif Seviye LEDler **active-LOW** çalışır: ortak uçları 3.3 V'a bağlıdır, bu yüzden bir LED'i yakmak için ilgili pine `'0'` yazılır. `'1'` yazılan LED söner. Bu, kart üzerindeki en sık unutulan ayrıntıdır; bu bölümdeki tüm örneklerde tekrar vurgulanır.

VHDL Mantığı Aşağıdaki tasarım, sekiz LED'e sabit bir desen yazar. Girişi olmayan, yalnızca çıkış portu bulunan en küçük tasarımdır:

Kod 5.1.1: LED deseni yazdırma

```
library ieee;                    -- standart kutuphane
use ieee.std_logic_1164.all;     -- std_logic tipleri icin

entity led_pattern is
    port (
        led : out std_logic_vector(7 downto 0)  -- 8 LED cikisi
    );
end entity led_pattern;

architecture rtl of led_pattern is
begin
```

```
    -- LED'ler active-LOW: '0' yazilan LED yanar.
    -- "10101010" -> led(0), led(2), led(4), led(6) yanar.
    led < = "10101010";           -- sabit desen; en sol bit = led(7)
end architecture rtl;
```

Desen vektöründeki en soldaki bit `led(7)`'ye, en sağdaki bit `led(0)`'a gider. Deseni değiştirip kartı yeniden programlayarak bit sırası ile kart üzerindeki LED sırası arasındaki ilişkiyi gözlemleyebilirsiniz.

Dikkat

`led(3)`, `led(4)` ve `led(5)` (pin 56/55/54) FPGA'nın SSPI dual-purpose pinleri üzerindedir. Bu LEDleri kullanan bir projede *Project* → *Configuration* → *Dual-Purpose Pin* altından *"Use SSPI as regular IO"* seçeneği işaretlenmelidir; aksi halde Place & Route, PR2017/PR2028 hatası verir. Ayrıntı için Bölüm 3'e bakınız.

Beklenen Davranış Kart programlandığında `led(0)`, `led(2)`, `led(4)` ve `led(6)` yanar; diğer dört LED sönük kalır. Desendeki `'0'` ve `'1'` değerlerini ters çevirirseniz yanan ve sönen LEDler yer değiştirir.

5.2 Butonlar

Amaç Karttan ilk kez giriş okumak: bir butonun durumunu bir LED'e aktarmak.

Kullanılan Pinler `btn(3 downto 0)` portu; `btn(0)`–`btn(3)` sırasıyla 70, 71, 72 ve 73 numaralı pinlere bağlıdır.

Aktif Seviye Butonlar **active-LOW** çalışır: basılı buton `'0'`, serbest buton (pull-up sayesinde) `'1'` okunur. Kart üzerinde her buton için RC donanım debounce devresi bulunduğundan, bu bölümdeki gibi basit uygulamalarda yazılımda ayrıca debounce yapmanız gerekmez.

VHDL Mantığı Buton ile LED'in ikisi de active-LOW olduğundan doğrudan bağlantı yeterlidir:

Kod 5.2.1: Buton ile LED kontrolü

```
-- btn ve led ikisi de active-LOW oldugundan
-- dogrudan baglanti dogru sonucu verir:
-- basili buton ('0') -> LED yanar ('0').
led(0) < = btn(0);
```

Buradaki "eksi ile eksinin birbirini götürmesi" önemlidir: buton basılıyken hatta `'0'` görülür, LED de `'0'` ile yandığı için sinyali terslemeye gerek kalmaz. Aktif seviyeleri kodda açıkça göstermek isterseniz aynı davranış iki adımda da yazılabilir:

Kod 5.2.2: Aynı davranışın aktif seviyeleri açıkça gösteren hali

```
btn_pressed < = not btn(0);      -- '1' = buton basili
led(0)      < = not btn_pressed; -- '0' = LED yanar
```

Beklenen Davranış Butona basılı tuttuğunuz sürece LED yanar, bıraktığınızda söner. Dört butonu dört ayrı LED'e bağlayarak aynı davranışı hepsinde gözlemleyebilirsiniz.

5.3 Slide Switchler

Amaç Kalıcı (basılı tutmayı gerektirmeyen) giriş kavramını göstermek: sekiz switch ile sekiz LED'i kontrol etmek.

Kullanılan Pinler `sw(7 downto 0)` portu; `sw(0)`–`sw(7)` sırasıyla 28, 27, 26, 25, 39, 36, 37 ve 38 numaralı pinlere bağlıdır.

Aktif Seviye Her switch hattında harici pull-up direnci bulunur. Switch aşağı konuma çekildiğinde hat GND'ye bağlanır ve FPGA pini `'0'` okur. Diğer konumda hat pull-up üzerinden `'1'` okunur. Buton gibi anlık değil, konumunu koruyan bir giriştir.

> **Not**
> Bu kılavuzda slide switch için `'0'` değeri switch'in aşağı/GND konumunu, `'1'` değeri ise pull-up konumunu ifade eder. Kod yazarken switch'in fiziksel yönü ile FPGA'nın okuduğu lojik değeri ayrı düşünmek gerekir.

VHDL Mantığı Sekiz switch'i sekiz LED'e bağlarken iki aktif seviye aynı anda düşünülmelidir: switch aşağı konumdayken FPGA `'0'` okur, LED ise `'0'` yazılınca yanar. Bu yüzden aşağı konumdaki switch'in LED'i yakmasını istiyorsak doğrudan bağlantı yeterlidir:

Kod 5.3.1: Switchler ile LED kontrolü

```
-- Switch asagi konumda GND'ye baglanir ve '0' okunur.
-- LED'ler de '0' yazilinca yanar.
-- Bu nedenle switch asagi -> LED yanar.
led < = sw;
```

Eğer uygulamada pull-up konumunu "aktif" kabul etmek ve switch `'1'` iken LED'in yanmasını isterseniz sinyali terslemeniz gerekir. Bu durumda `led < = not sw;` yazılır. Bu iki farklı kullanım, elektriksel seviye ile kullanıcıya gösterilen anlamın aynı şey olmadığını öğretir.

Beklenen Davranış Bu örnekte her switch kendi hizasındaki LED'i kontrol eder: switch aşağı konuma çekildiğinde LED yanar, diğer konumda söner. Switchler konumlarını koruduğu için LEDler siz değiştirene kadar aynı durumda kalır.

Sık Kontrol Edilecek Noktalar Bu bölümde hata ararken önce aktif seviyeleri kontrol edin. LEDler active-LOW olduğu için `'0'` ile yanar. Butonlar basılıyken `'0'` okunur. Slide switch aşağı konumdayken GND'ye bağlıdır ve `'0'` okunur. Ayrıca LED3, LED4 veya LED5 kullanılıyorsa SSPI pinlerinin regular IO olarak ayarlandığından emin olun.

Bölüm 6

Dijital Tasarım Konularının Kart Üzerindeki Karşılığı

Bu bölümün amacı dijital tasarım dersini baştan anlatmak değildir. Amaç, derste görülen her yapının Tang Edu Board üzerinde hangi giriş ve çıkışlarla denenebileceğini göstermektir. Her konu için kısa bir çalışma mantığı, VHDL'deki karşılığı ve **Kartta:** ile başlayan somut bir deneme fikri verilir; tam proje kodları verilmez.

6.1 Kombinasyonel Mantık

Kombinasyonel devrelerde çıkış yalnızca o andaki girişlere bağlıdır; devre hiçbir şey hatırlamaz. Bu yüzden bu bölümdeki tüm konular yalnızca eşzamanlı atamalarla, process kullanmadan yazılır.

6.1.1 Logic Gate'ler

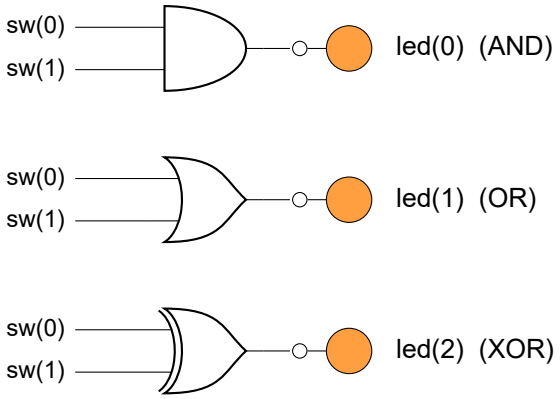
Temel kapıların tamamı VHDL'de hazır operatör olarak vardır: `and`, `or`, `not`, `nand`, `nor`, `xor` ve `xnor`. İki girişli kapıların davranışı tek tabloda özetlenebilir:

a	b	AND	OR	XOR	NAND	NOR	XNOR
0	0	0	0	0	1	1	1
0	1	0	1	1	1	0	0
1	0	0	1	1	1	0	0
1	1	1	1	0	0	0	1

Kartta: `sw(0)` ve `sw(1)` iki giriş olsun; farklı kapıları farklı LEDlere bağlayarak hepsini aynı anda gözlemleyebilirsiniz. Sonucun "1" olduğunu LED'in yanmasıyla göstermek için başa `not` eklenir (LEDler active-LOW):

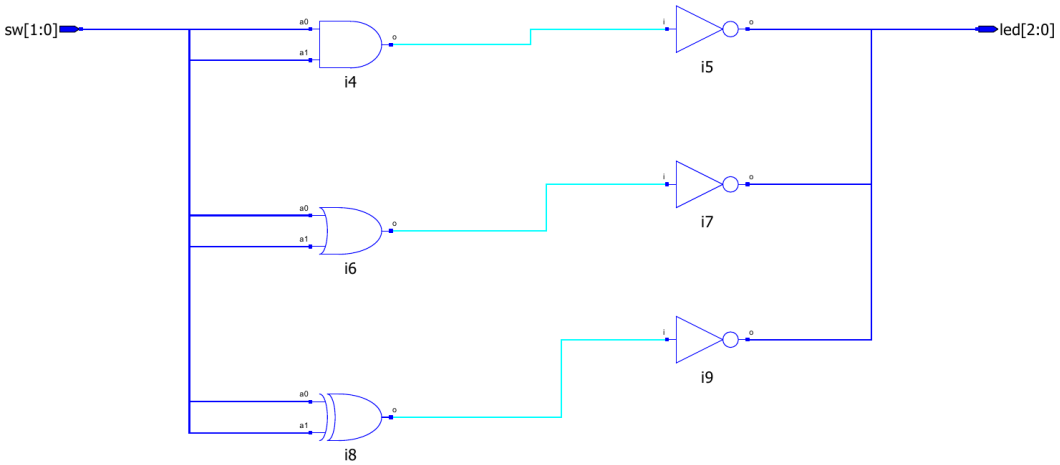
Kod 6.1.1.1: Ayni girisleri birden fazla kapiyla gozlemleme

```
led(0) <= not (sw(0) and sw(1));  -- AND sonucu
led(1) <= not (sw(0) or  sw(1));  -- OR  sonucu
led(2) <= not (sw(0) xor sw(1));  -- XOR sonucu
```



Şekil 6.1.1.1: Aynı iki giriş (`sw(0)`, `sw(1)`) farklı kapılara verilip sonuçları ayrı LED'lerde gözlenir. Kapı çıkışındaki küçük daire, LED'ler active-LOW olduğu için eklenen çeviricidir.

Aynı tasarımın Gowin'in ürettiği RTL şeması olarak nasıl göründüğü Şekil 6.1.1.2'de verilmiştir.



Şekil 6.1.1.2: 6.1.1 tasarımının Gowin RTL Design Viewer görünümü (sentez sonrası üretilen şema).

> **Not**
> Her deneyin RTL şeması, Gowin IDE'de tasarımı *Synthesize* ettikten sonra üst çubuktaki **Tools** sekmesinden **Schematic Viewer** → **RTL Design Viewer** yolu ile görülebilir.

6.1.2 Multiplexer ve Demultiplexer

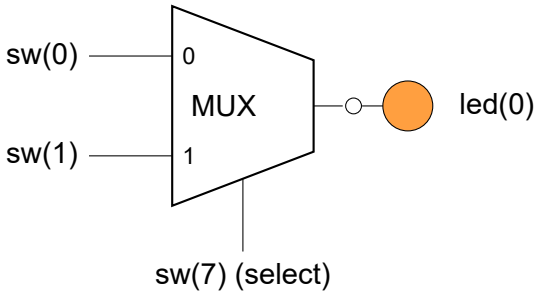
Multiplexer, select hattının değerine göre girişlerinden birini çıkışa aktaran bir seçicidir. VHDL'de bunun doğal karşılığı `when/else` koşullu atamasıdır:

Kod 6.1.2.1: 2'ye 1 multiplexer

```
mux_out <= sw(0) when sw(7) = '0' else sw(1);
led(0)  <= not mux_out;
```

Kartta: `sw(0)` ve `sw(1)` data, `sw(7)` select olsun. Select'i değiştirdiğinizde LED'in artık öbür switch'i takip ettiğini görürsünüz; iki data switch'ini farklı konumlara alırsanız geçiş anı netleşir.

Demultiplexer aynı fikrin tersidir: tek bir giriş, select'e göre çıkışlardan yalnızca birine yönlendirilir, seçilmeyen çıkışlar pasif değerde bekler. MUX örneğini yazdıktan sonra DEMUX'u kendiniz türetmek iyi bir alıştırmadır.



Şekil 6.1.2.1: 2'ye 1 multiplexer: `select` (`sw(7)`) 0 iken `sw(0)`, 1 iken `sw(1)` çıkışa aktarılır.

6.1.3 Adder, Subtracter ve Comparator

Aritmetik işlemler için `ieee.numeric_std` kütüphanesi ve `unsigned` tipi kullanılır; `std_logic_vector` doğrudan toplanamaz, önce dönüştürülür. Sekiz switch iki adet 4-bitlik sayı olarak okunabilir:

Kod 6.1.3.1: 4-bit toplayici; besinci bit carry

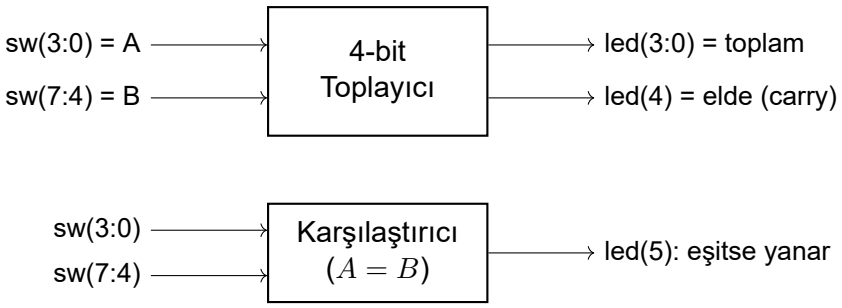
```
use ieee.numeric_std.all;            -- unsigned islemler icin

toplam < = unsigned('0' & sw(3 downto 0))
        + unsigned('0' & sw(7 downto 4));
led(4 downto 0) < = not std_logic_vector(toplam);
```

Kartta: sağdaki dört switch birinci sayı, soldaki dört switch ikinci sayı; sonuç beş LED'de binary okunur. `led(4)` carry bitidir: $9 + 9 = 18$ gibi bir toplamda yandığını görürsünüz. Çıkarma için `+` yerine `-` yazmak yeterlidir; bu durumda en üst bit borrow anlamına gelir. Karşılaştırma ise tek satırdır:

Kod 6.1.3.2: Esitlik karsilastirici

```
led(5) < = '0' when sw(3 downto 0) = sw(7 downto 4) else '1';
```

İki sayı eşit olduğunda LED yanar; `>` ve `<` çıkışları da aynı kalıpla ayrı LEDlere bağlanabilir (`unsigned` dönüşümüyle).



Şekil 6.1.3.1: Sekiz switch iki adet 4-bit sayı olarak okunur: toplayıcı bloğu toplamı ve eldeyi, karşılaştırıcı bloğu eşitliği üretir.

6.2 Sequential Logic

Sequential devrelerde çıkış, anlık girişlere ek olarak devrenin *hatırladığı* önceki duruma da bağlıdır. Değer saklamak için Bölüm 4.5'teki process kalıbı kullanılır; bu bölümdeki her konu o kalıbın bir varyasyonudur.

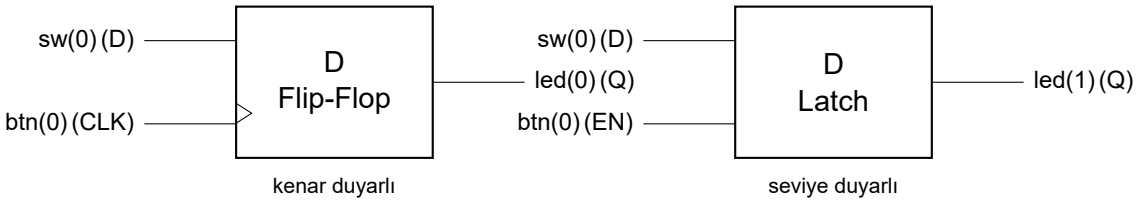
6.2.1 Latch ve Flip-Flop

İkisi de bir bit saklar; fark *ne zaman* güncelliklerindedir: latch, enable aktifken girişini sürekli geçirir (seviye duyarlı); flip-flop yalnızca clock kenarında örnek alır (kenar duyarlı). Fark, clock'u elle atarak en net görülür — `btn(0)` clock, `sw(0)` data olsun:

Kod 6.2.1.1: Elle clock'lanan D flip-flop

```
if falling_edge(btn(0)) then  -- basma ani (buton active-LOW)
    saklanan <= sw(0);        -- data yalnizca bu anda orneklenir
end if;
led(0) <= not saklanan;
```

Kartta: butona bastığınız *an* switch'in değeri saklanır; sonrasında switch'i oynatmak LED'i etkilemez — bir sonraki basışa kadar değer korunur. Bu bir latch olsaydı, buton basılı olduğu sürece LED switch'i anlık takip ederdi; kenar ile seviye duyarlılığı arasındaki fark tam olarak budur. FPGA tasarımında bilinçli olarak hep flip-flop kullanılır; kombinasyonel bir atamada bazı koşullara değer vermeyi unutursanız sentezleyici istemeden latch üretir ve Gowin IDE uyarı verir. Butonu clock yapmak da yalnızca bu tür deneyler içindir; gerçek tasarımlarda 27 MHz sistem clock'u kullanılır.



Şekil 6.2.1.1: D flip-flop yalnızca clock kenarında (`btn(0)` basma anı) örnek alır — saat girişindeki üçgen bunu gösterir; latch ise `btn(0)` basılı olduğu sürece `sw(0)`'ı çıkışa geçirir.

6.2.2 Shift Register

Shift register, sakladığı bitleri her clock kenarında bir pozisyon kaydırır ve boşalan yere yeni biti alır. VHDL'de bunun karşılığı birleştirme operatörü & ile tek satırdır:

Kod 6.2.2.1: 8-bit shift register cekirdegi

```
if falling_edge(btn(0)) then     -- her buton basisinda bir kez
    q <= q(6 downto 0) & sw(0);  -- sola kaydir, yeni bit sagdan
end if;
led <= not q;
```

Kartta: `sw(0)`'ı `'1'` yapıp butona birkaç kez basın: LEDlerde soldan yürüyen bir ışık dizisi oluşur. Switch'i `'0'`'a alıp basmaya devam ederseniz dizinin arkasından karanlık yürür. Kaydırma kavramını bundan daha somut gösteren bir deney yoktur.

6.2.3 Counter

Sayaç, her clock kenarında bir artan bir `unsigned` sinyalden ibarettir: `sayac <= sayac + 1`. Karttaki asıl ders sayacın kendisi değil, *hız* problemidir: 27 MHz ile artan bir sayacın alt bitleri saniyede milyonlarca kez değişir ve LED'de sürekli yanıyor gibi görünür. Çözüm, LED'lere sayacın *üst* bitlerini bağlamaktır:

Kod 6.2.3.1: Gozle izlenebilir sayac

```
signal sayac : unsigned(26 downto 0);
...
```

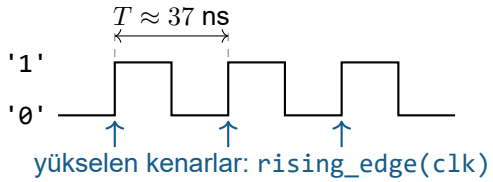
```
if rising_edge(clk) then
    sayac <= sayac + 1;  -- her yukselen kenarda 1 artar
end if;                  -- (saniyede 27 milyon kez!)
led <= not std_logic_vector(sayac(26 downto 19));
```

Kartta: en soldaki LED birkaç saniyede bir, sağa doğru her LED bir öncekinin iki katı hızda yanıp söner — binary saymayı gözle izlersiniz. Bu "üst bitleri kullan" fikri, ilerideki tüm zamanlama işlerinin (LED yakıp söndürme, 7-segment tarama) temelidir.

6.2.4 Clock Sinyali ve Dalga Formu

Bu bölümde hep kullandığımız `clk`, Tang Nano 9K üzerindeki 27 MHz kristal osilatörden gelir ve FPGA'nın 52 numaralı pinine bağlıdır. "Ayarlamak" için yapılacak tek şey, sinyali tasarıma almaktır: entity'ye `clk : in std_logic` portu eklenir ve constraint dosyasında pin 52'ye eşlenir (master dosyada bu satır hazırdır, bkz. Bölüm 3.3). Osilatör, kart beslendiği andan itibaren kesintisiz üretir; açıp kapatamazsınız.

Dalga formu %50 doluluk oranlı bir kare dalgadır: sinyal periyodun yarısında `'1'`, yarısında `'0'` seviyesindedir ve periyot $T = 1/27\,\text{MHz} \approx 37$ ns'dir:



Her periyotta bir yükselen kenar oluşur; yani `rising_edge(clk)` bloğu saniyede 27 milyon kez çalışır — Bölüm 6.2.3'teki hız probleminin kaynağı bu sayıdır. Tasarımdaki tüm flip-floplar aynı kenarı paylaştığı için hepsi aynı anda güncellenir; buna *senkron tasarım* denir ve bu kılavuzdaki tüm örnekler bu ilkeye uyar. Farklı bir frekans gerekirse Gowin'in PLL IP çekirdeği kullanılabilir, ancak çoğu uygulamada 6.2.3'teki gibi sayaçla yavaşlatmak yeterlidir.

6.3 State Machine

Durum makinesi, davranışı isimlendirilmiş durumlara bölerek karmaşayı yönetir: devre her an tek bir durumdadır ve girişler yalnızca *durumlar arası geçişleri* tetikler. VHDL'de durumlar için özel bir tip tanımlanır, geçişler `case` ile yazılır:

Kod 6.3.1: Uc durumlu makine iskeleti

```
type durum_tipi is (DUR, YURU, YANIP_SON);
signal durum : durum_tipi;
...
if falling_edge(btn(0)) then
    case durum is
        -- '=>' case dalini acar: "durum bu ise ..."
        -- '<=' ise siradaki durumu atar
        when DUR       => durum <= YURU;
        when YURU      => durum <= YANIP_SON;
        when YANIP_SON => durum <= DUR;
    end case;
end if;
```

Her durumda LED'lere hangi desenin sürüleceği ise ayrı bir `case` ile kombinasyonel olarak yazılır; böylece "nerede olduğum" ile "ne gösterdiğim" birbirinden ayrılır. Bu ayrım, durum makinesi tasarımının en önemli alışkanlığıdır.

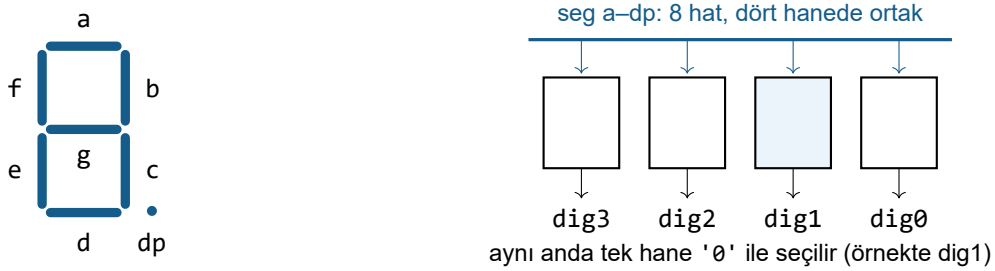
Bölüm 7

Kart Üzerindeki Diğer Uygulamalar

Bu bölüm, temel LED/buton/switch akışından sonra gelen kart özelliklerini ve Tang Nano 9K üzerindeki ek uygulama olanaklarını toplar.

7.1 7-Segment Kullanımı

Kart üzerindeki display, dört haneli ve *common cathode* yapıda bir 7-segment göstergedir. Her hane, harflerle adlandırılan yedi segment ve bir ondalık noktasından (dp) oluşur. Sürüş kuralı Bölüm 3.2'deki tabloda özetlenmişti: segmentler `'1'` ile yanar, bir hane ortak katoduna `'0'` yazılarak seçilir.



Şekil 7.1.1: Segment adlandırması (solda) ve dört hanenin ortak segment hatları (sağda)

Bir rakamı göstermek, o rakamın segment desenini `seg` çıkışına yazmaktan ibarettir. Desenler 7 bitlik vektörlerle, `seg(6)`=g … `seg(0)`=a sırasıyla kodlanır; ondalık noktası `seg(7)` ayrıca sürülür. 4-bit giriş kullanıldığı için 9'dan sonrası hex olarak devam eder — display'de A–F harflerini de gösterebilirsiniz:

Kod 7.1.1: 4-bit değerden segment desenine kod çözücü (hex 0–F)

```
case deger is                                  -- sira: g f e d c b a
    when "0000" => seg(6 downto 0) < = "0111111";  -- 0
    when "0001" => seg(6 downto 0) < = "0000110";  -- 1
    when "0010" => seg(6 downto 0) < = "1011011";  -- 2
    when "0011" => seg(6 downto 0) < = "1001111";  -- 3
    when "0100" => seg(6 downto 0) < = "1100110";  -- 4
    when "0101" => seg(6 downto 0) < = "1101101";  -- 5
    when "0110" => seg(6 downto 0) < = "1111101";  -- 6
    when "0111" => seg(6 downto 0) < = "0000111";  -- 7
    when "1000" => seg(6 downto 0) < = "1111111";  -- 8
    when "1001" => seg(6 downto 0) < = "1101111";  -- 9
    when "1010" => seg(6 downto 0) < = "1110111";  -- A
    when "1011" => seg(6 downto 0) < = "1111100";  -- b
```

```
    when "1100" => seg(6 downto 0) < = "0111001";  -- C
    when "1101" => seg(6 downto 0) < = "1011110";  -- d
    when "1110" => seg(6 downto 0) < = "1111001";  -- E
    when others => seg(6 downto 0) < = "1110001";  -- F
end case;
```

Asıl püf noktası dört hanenin *ortak* segment hatlarını paylaşmasıdır: aynı anda yalnızca tek bir haneye desen yazılabilir. Dört hanede farklı rakamlar göstermek için haneler sırayla ve hızla taranır — her an tek hane yanar, ama göz saniyede ∼50'den hızlı tekrarı sürekli ışık olarak algılar. Tarama hızı, Bölüm 6.2.3'teki "üst bitleri kullan" fikriyle sayaçtan türetilir:

Kod 7.1.2: Hane tarama: sayacın üst bitleriyle hane seçimi

```
hane < = sayac(16 downto 15);          -- ~800 Hz'de 0..3 dolasir
dig  < = "1110" when hane = "00" else  -- dig0 secili ('0' = aktif)
         "1101" when hane = "01" else  -- dig1
         "1011" when hane = "10" else  -- dig2
         "0111";                       -- dig3
```

Seçilen haneyle eş zamanlı olarak `seg` çıkışına o hanenin rakam deseni yazılır. Bu hızda her hane saniyede yaklaşık 200 kez tekrarlanır ve titreme görülmez; tarama çok yavaşlatılırsa (örneğin üst bitler seçilirse) hanelerin sırayla yandığını gözlemleyebilirsiniz — taramanın nasıl çalıştığını görmek için bu, bilerek yapılabilecek güzel bir deneydir.

Kartta: ilk deneme için `sw(3 downto 0)` değerini tek haneye yazın (tarama gerekmez, `dig < = "1110"` sabit kalabilir). Sonrasında Bölüm 6.1.3'teki toplayıcıyla birleştirip toplamı display'de göstermek, bu bölümle Bölüm 6'yı birleştiren iyi bir çalışmadır.

7.2 HDMI

Tang Nano 9K modülü üzerinde bir HDMI konnektörü bulunur. Bu konnektör, FPGA'den çıkan TMDS hatlarıyla harici bir monitöre görüntü üretmek için kullanılabilir. Bu bölümdeki HDMI örneği, yalnızca sabit renk göstermeyen; ekranda hareket, çarpışma ve switch kontrolü de içeren küçük bir görüntü uygulamasıdır.

Dikkat

HDMI TMDS hatları Tang Nano 9K üzerindeki paylaşımlı pinleri kullanır. Bu projede pin 68–75 aralığı HDMI için ayrıldığı için Tang Edu Board üzerindeki LED0/LED1, BTN0–BTN3 ve DIG0/DIG1 aynı anda kullanılmaz. HDMI denemesi yaparken bu pinlere bağlı diğer çevre birimlerinden davranış beklemeyin.

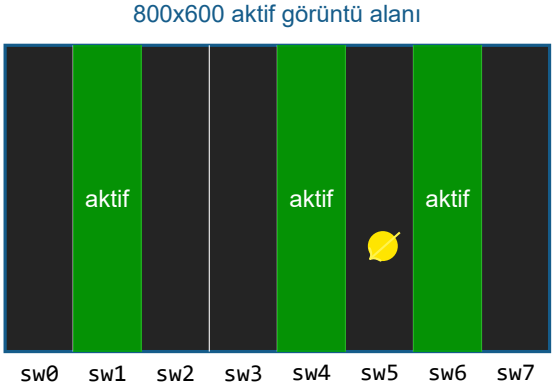
Bu bölümdeki örnek proje, manual kod projeleri altındaki `7_2` klasöründedir. Proje, 27 MHz kart saatinden PLL ve clock divider bloklarıyla yaklaşık 40 MHz pixel clock ve yaklaşık 199.8 MHz TMDS seri saat üretir. Çıkış formatı 800x600 @ 60 Hz VESA zamanlamasına göre ayarlanmıştır.

Not

Bu örnekte HDMI denemesi 800x600 @ 60 Hz olarak ayarlanmıştır. Bazı 1920x1080 paneller, özellikle scaler kartına bağlı paneller, 640x480 girişini ince bir şerit gibi gösterebilir. 720p60 daha yüksek TMDS hızına çıktığı için parazit veya pırıltı yapabilir. 800x600 @ 60 Hz bu kart ve kablo düzeninde daha dengeli bir deneme çözümüdür; 16:9 paneller görüntüyü yatayda esnetebilir.

Ekranda koyu renkli bir arka plan, sarı bir top ve switchlerle seçilen yeşil duvar bölgeleri görülür. Top başlangıçta sağ üst taraftan sola-aşağı yönde hareket eder; ekran kenarlarına veya aktif

yeşil duvarlara çarptığında yön değiştirerek seker. Görüntü alanı soldan sağa sekiz dikey bölgeye ayrılmıştır: `sw(0)` en soldaki, `sw(7)` en sağdaki bölgeyi kontrol eder. Slide switch'ler pull-up bağlı olduğu için switch yukarı konumdayken giriş `'1'` okunur ve ilgili bölge yeşil duvar olur; aşağı konumdaki switch'in bölgesi koyu arka plan renginde kalır.



Şekil 7.2.1: HDMI örneğinde yukarı alınan switchlerin etkinleştirdiği yeşil duvar bölgeleri (örnekte `sw1`, `sw4` ve `sw6` yukarıda)

Üst seviye modülün dışarıya açtığı hatlar sade tutulmuştur: kart saati, sekiz slide switch ve HDMI için diferansiyel TMDS çıkışları. Video zamanlaması, TMDS kodlama, paralel-seri dönüşüm ve LVDS çıkış buffer'ları aynı VHDL dosyasının içinde verilmiştir.

Kod 7.2.1: HDMI örneğinin üst seviye portları

```
entity hdmi_ball_walls is
    port (
        clk        : in  std_logic;                    -- 27 MHz kart saati
        sw         : in  std_logic_vector(7 downto 0); -- duvar secen switchler
        tmds_clk_p : out std_logic;                    -- HDMI clock, pozitif hat
        tmds_clk_n : out std_logic;                    -- HDMI clock, negatif hat
        tmds_d_p   : out std_logic_vector(2 downto 0); -- TMDS veri hatlari, pozitif
        tmds_d_n   : out std_logic_vector(2 downto 0)  -- TMDS veri hatlari, negatif
    );
end entity hdmi_ball_walls;
```

Duvarların aktif seviyesi, karttaki switch bağlantısıyla doğrudan ilişkilidir. Switch yukarı konumdayken giriş `'1'` okunduğu için, senkronlanan switch değeri ek bir tersleme yapılmadan doğrudan duvar etkinleştirme vektörü olarak kullanılır:

Kod 7.2.2: Switch yukarı konumunu aktif duvar olarak yorumlama

```
wall_active < = sw_sync; -- switch yukari: '1' okunur, duvar aktif olur
```

HDMI pinleri constraint dosyasında diferansiyel çift olarak verilir. Bu pinler Tang Nano 9K üzerindeki resmi HDMI pin yerleşimiyle uyumludur; aynı zamanda Tang Edu Board üzerinde bazı temel çevre birimleriyle paylaşıldıkları için bu örnek proje kendi `.cst` dosyasında yalnızca kullandığı portları tanımlar.

Kod 7.2.3: HDMI TMDS pinlerinin proje yerel constraint eşlemesi

```
// TMDS data lane 0: pozitif pin 71, negatif pin 70
IO_LOC  "tmds_d_p[0]" 71,70;

// TMDS data lane 1: pozitif pin 73, negatif pin 72
IO_LOC  "tmds_d_p[1]" 73,72;

// TMDS data lane 2: pozitif pin 75, negatif pin 74
IO_LOC  "tmds_d_p[2]" 75,74;

// TMDS clock lane: pozitif pin 69, negatif pin 68
IO_LOC  "tmds_clk_p" 69,68;
```

Kartta: HDMI kablosunu Tang Nano 9K üzerindeki HDMI konnektörüne ve monitöre bağlayın. Gowin'de `7_2.gprj` projesini açıp Synthesize, Place & Route ve Program adımlarını çalıştırın. Komut satırıyla denemek isterseniz proje klasöründe `open_project 7_2.gprj` ve ardından `run all` komutları kullanılabilir. Programlanan tasarımda switchleri yukarı kaldırarak ilgili dikey bölgenin yeşil duvara dönüştüğünü, topun da bu duvarlardan sektiğini gözlemleyin.

Sorun giderme: Monitörde görüntü yoksa önce doğru HDMI girişinin seçildiğini ve FPGA'nın programlandığını kontrol edin. Görüntü ince bir şerit gibi görünüyorsa panel/scaler seçilen çözünürlüğü beklenenden farklı yorumluyor olabilir; bu yüzden örnek 640x480 yerine 800x600 @ 60 Hz kullanır. Ekranda pırıltı veya rastgele noktalar görülürse kablo, bağlantı kalitesi veya TMDS hızı sınırda olabilir; bu kart üzerinde 720p60 gibi daha yüksek hızlı denemeler daha hassastır.

7.3 Genişleme Konnektörleri

Tang Nano 9K modülünün iki kenarındaki 24'er pin, Tang Edu Board üzerindeki iki adet 24-pin genişleme konnektörüne birebir aktarılır. Şekil 3.1.1'deki renkli etiketli pinler kart birimlerine bağlıdır; gri ve sarı etiketli *user pin*'ler ise hiçbir kart birimine bağlı değildir ve harici devreler için serbesttir:

Grup	FPGA pinleri	`IO_TYPE`
User pinler, 3.3 V	63, 48, 49, 31, 32	`LVCMOS33`
User pinler, 1.8 V	79, 80, 81, 82, 83, 84, 85, 86	`LVCMOS18`
Besleme	5 V, 3.3 V, GND	—

Dikkat

Sarı etiketli 79–86 pinleri FPGA'nın 1.8 V bankındadır (BANK3, $V_{IO} = 1.8$ V). Bu pinlere **asla 3.3 V sinyal uygulamayın**; FPGA'ya kalıcı zarar verebilir. Constraint dosyasında bu pinler için `IO_TYPE =LVCMOS18` kullanılmalıdır. 5 V hattı yalnızca harici devre beslemesi içindir; hiçbir FPGA pinine 5 V sinyal bağlanmaz.

User pinler, kart birimlerinde olduğu gibi `.cst` dosyasına eklenerek kullanılır (Bölüm 3.3). Master constraint dosyası yalnızca kart birimlerini içerdiği için user pin satırlarını kendiniz yazarsınız:

Kod 7.3.1: User pinlerin projeye eklenmesi

```
// 3.3 V bankindaki bir user pin
IO_LOC  "harici_giris" 63;
IO_PORT "harici_giris" IO_TYPE=LVCMOS33;

// 1.8 V bankindaki bir user pin -- LVCMOS18 zorunlu
IO_LOC  "harici_1v8" 79;
IO_PORT "harici_1v8" IO_TYPE=LVCMOS18;
```

Kart birimlerine bağlı pinler de konnektörlerde fiziksel olarak erişilebilir durumdadır; ancak bu pinleri harici devre için kullanırsanız ilgili kart birimi (LED, switch, buton veya display hattı) aynı sinyali paylaşmaya devam eder. Harici bağlantılar için önce yukarıdaki serbest user pinleri tercih edin.

Not

User pinlerin bir kısmı Tang Nano 9K modülünün kendi üzerindeki RGB/SPI LCD konnektör hatlarıyla ortaktır (örneğin 48/49 SPI LCD, 63/31/32 RGB hatları). Modülün kendi LCD konnektörlerine bir şey takılı olmadığı sürece bu pinler serbestçe kullanılabilir.

Güvenli kullanım için üç temel kural yeterlidir: harici devrenin GND'si mutlaka konnektördeki GND'ye bağlanır (ortak referans); bağlantılar kartın beslemesi kesilmişken yapılır; ve bağlanan sinyalin gerilim seviyesi ilgili pinin bank gerilimiyle (3.3 V veya 1.8 V) eşleşir.