



Tang Edu Board Kullanım Kılavuzu

Tang Nano 9K Tabanlı Eğitim Kartı

Sürüm 1.0

Üniversite laboratuvarları ve öğrencileri için Beti® Elektronik tarafından üretilmiştir.

Hazırlayan: **Osman Bedirhan Hakyemez**

obhakyemez@gmail.com

İçindekiler

1	Giriş	1
1.1	Kılavuzun Amacı	1
1.2	Hedef Kullanıcı	1
2	Donanım Genel Özellikleri	2
2.1	Tang Nano 9K	2
2.2	Tang Edu Board	2
3	Pin Diyagramı ve Bağlantılar	4
3.1	Pin Haritası	4
3.2	Aktif Seviye Bilgileri	5
3.3	Constraint Dosyasının Kullanımı	5
4	Yazılım: Gowin IDE Kurulumu ve Kullanımı	7
4.1	Kurulum	7
4.1.1	İndirme ve Hesap Oluşturma	7
4.1.2	Windows’a Kurulum	9
4.2	Yeni Proje Oluşturma	11
4.3	Derleme ve Programlama	13
4.4	Sık Alınan Hatalar	14
4.4.1	PR2017/PR2028: SSPI pinleri (led[3]–led[5])	14
4.4.2	“net not found”: constraint dosyasında fazla satır	15
4.4.3	Programmer kartı görmüyor veya yazma başarısız oluyor	15
4.5	Temel VHDL Yapısı	15
5	Temel Kart Uygulamaları	17
5.1	LED’ler	17
5.2	Butonlar	18
5.3	Slide Switchler	18
6	Dijital Tasarım Konularının Kart Üzerindeki Karşılığı	20
6.1	Kombinasyonel Mantık	20
6.1.1	Logic Gate’ler	20
6.1.2	Multiplexer ve Demultiplexer	21
6.1.3	Adder, Subtractor ve Comparator	22
6.2	Sequential Logic	22
6.2.1	Latch ve Flip-Flop	23
6.2.2	Shift Register	23
6.2.3	Counter	23
6.2.4	Clock Sinyali ve Dalga Formu	24
6.3	State Machine	24

7	Kart Üzerindeki Diğer Uygulamalar	26
7.1	7-Segment Kullanımı	26
7.2	HDMI	27
7.3	Genişleme Konnektörleri	30

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: LED'ler, butonlar, slide switchler ve 7-segment display kartın üzerinde hazırdr.

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 konektörü 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.

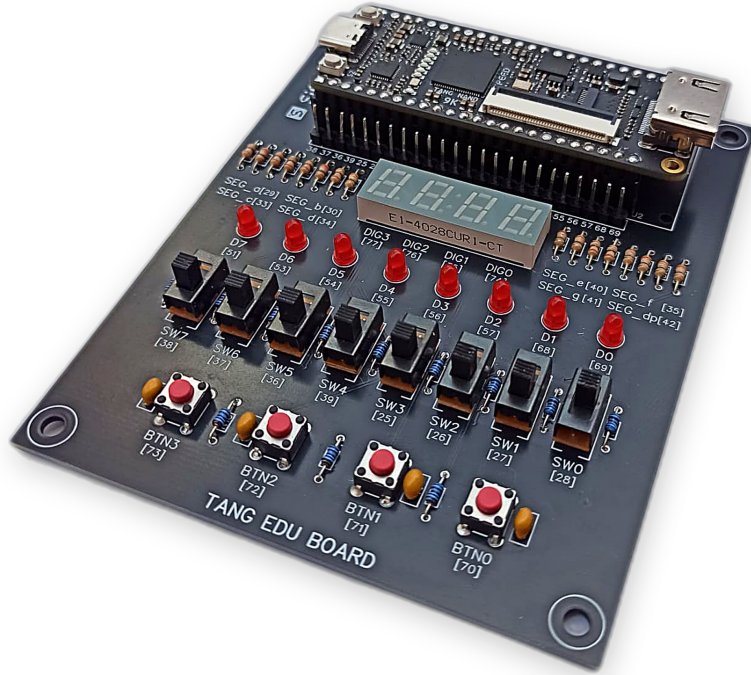
Kart üzerindeki birimler ve ayrıntılarının anlatıldığı bölümler:

- **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 konektörü** — harici devre bağlantıları (Bölüm 7.3).

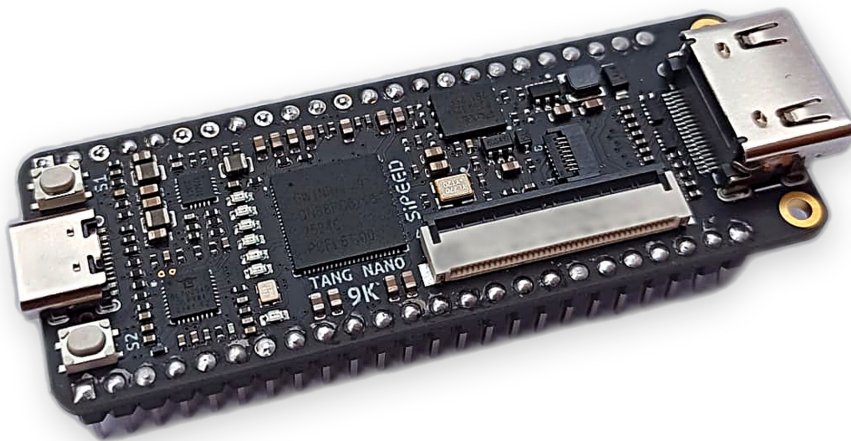
Tüm birimlerin FPGA pin eşlemeleri Bölüm 3'te, aktif seviye davranışları Bölüm 3.2'dedir. Temel uygulamalar için karta herhangi bir ek bağlantı yapmanız gerekmez.

Dikkat

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



Şekil 2.2.1: Tang Edu Board — taşıyıcı kartın tamamı (üstte Tang Nano 9K modülü takılı).



Şekil 2.2.2: Tang Nano 9K modülü — tek başına (karta takılmadan).

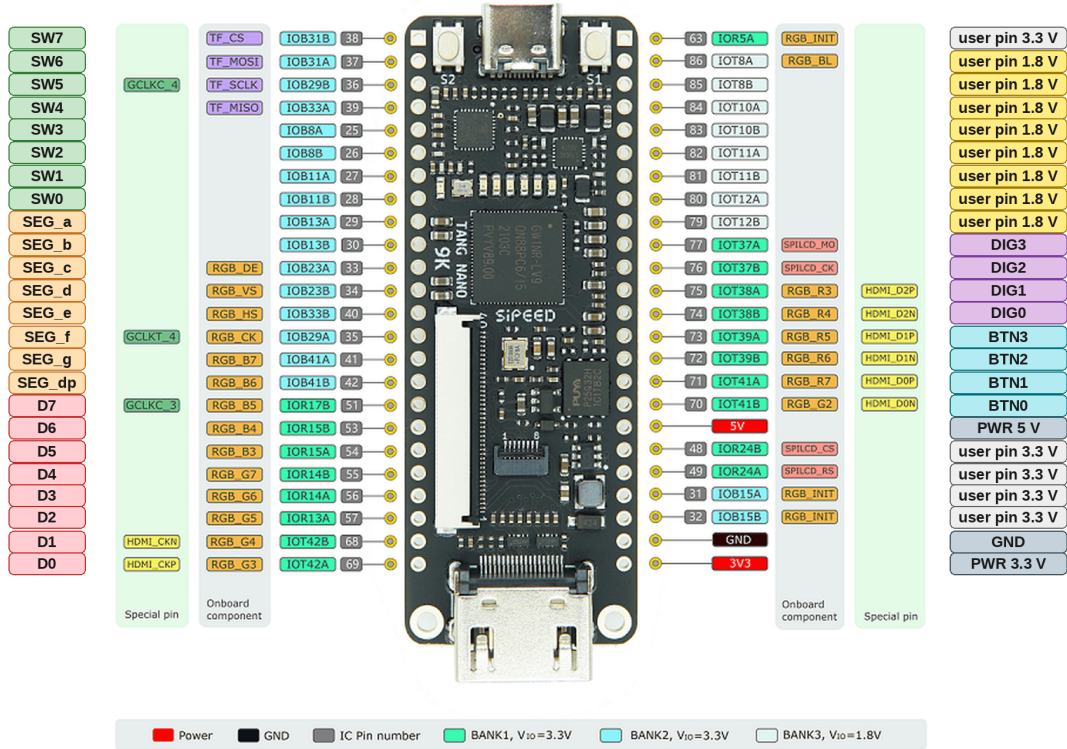
Bölüm 3

Pin Diyagramı ve Bağlantılar

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

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)
LED'ler (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→dp: 29, 30, 33, 34, 40, 35, 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 konektör 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'te 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
LED'ler	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: LED'lerin 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. Şekil 3.2.1 bu bağlantıların üçünü şematik olarak gösterir.

Not

Bölüm 5'teki örneklerde görülen **not** terslemelerinin tamamı bu tablodan türetilir. Bir birim “ters çalışıyor” gibi görüldüğünde önce buraya dönün.

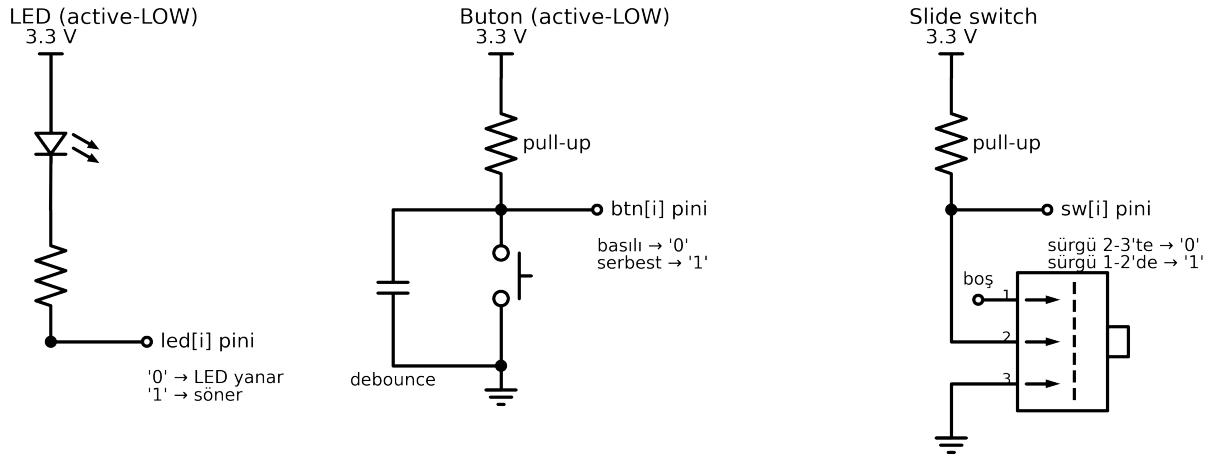
3.3 Constraint Dosyasının Kullanımı

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

Kod 3.3.1: Bir portun pine eşlenmesi

```
1 IO_LOC "led[0]" 69; // hangi fiziksel pin
2 IO_PORT "led[0]" IO_TYPE=LVCMS33; // 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



Şekil 3.2.1: Aktif seviyelerin donanım karşılığı (kavramsal şema). Solda LED: ortak uç 3.3 V'ta olduğundan pin '0' iken akım akar ve LED yanar. Ortada buton: pull-up direnci hattı normalde '1' de tutar, basmak hattı GND'ye çeker; kondansatör donanım debounce içindir. Sağda üç uçlu slide switch: orta uç (2) FPGA pinine gider; sürgü 2-3 konumundayken hat '0', 1-2 konumundayken (1 numaralı uç boşta olduğundan) pull-up nedeniyle '1' okunur.

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. Bu kılavuzdaki temel, tek clock'lu projelerde tek bir *create_clock* satırı yeterlidir:

Kod 3.3.2: Timing constraint: 27 MHz clock tanımı

```
1 // 27 MHz -> periyot = 37.037 ns
2 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, bu clock'a bağlı register'lar arasındaki yolların 37.037 ns'lik periyot (setup/hold) gereksinimini karşılayıp karşılamadığını analiz eder. PLL ile yeni clock üretilen veya birden fazla clock bulunan projelerde ise türetilen her clock için ek *create_clock* satırları gerekir; örneğin Bölüm 7.2'deki HDMI projesinin *.sdc* dosyası üç clock tanımı içerir.

Tang Edu Board için her iki master dosya da (*TANG_EDU_BOARD.cst* ve *TANG_EDU_BOARD.sdc*), örnek projelerle birlikte Beti Elektronik ürün sayfasından edinilebilir. 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 ve Kullanımı

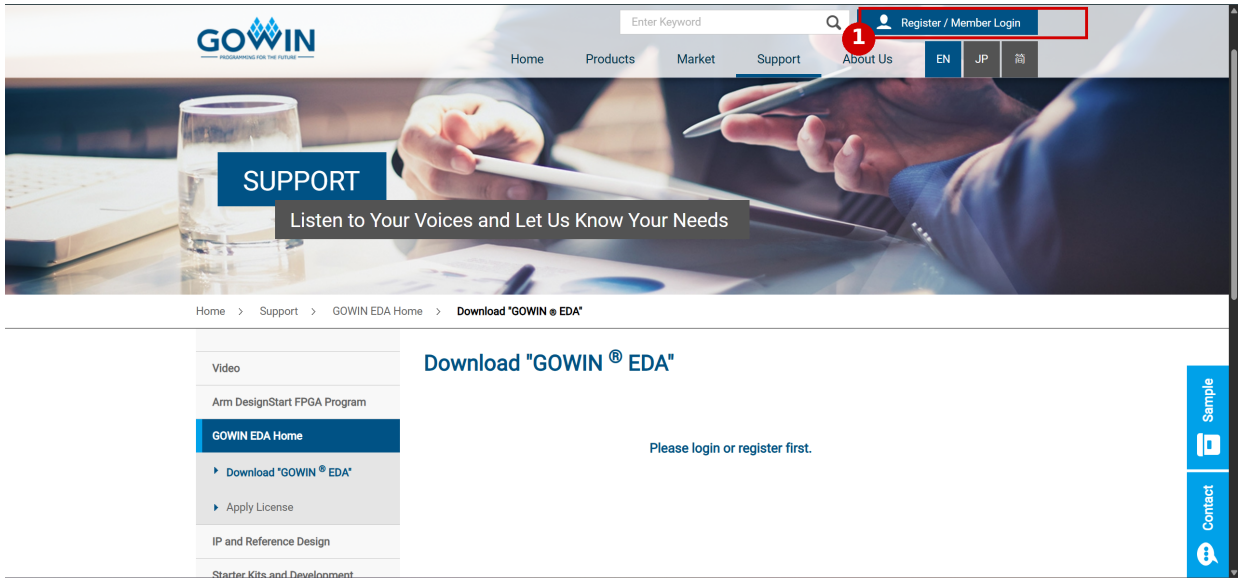
Bu bölüm, öğrencinin Tang Edu Board ile çalışmaya başlaması için gerekli Gowin IDE kurulum ve kullanım adımlarını açıklar.

4.1 Kurulum

Tang Edu Board için tüm geliştirme akışı — HDL yazma, sentez, Place & Route, bitstream üretimi ve kartın programlanması — Gowin EDA yazılımıyla yürütülür. Bu kılavuzda yazılımın **Education Edition** sürümü kullanılır: lisans dosyası gerektirmez ve karttaki GW1NR-9C yongasını tam olarak destekler. Standard Edition yalnızca Education'ın kapsamadığı yonga ve IP'ler için gereklidir ve ayrıca lisans başvurusu ister; bu kart için gerekli değildir.

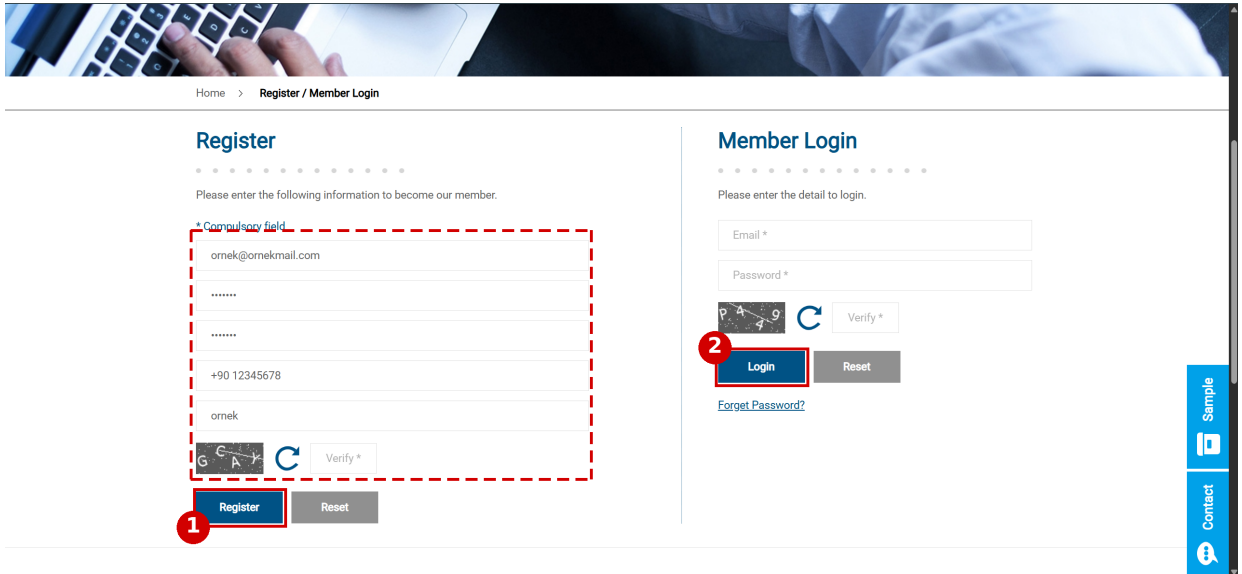
4.1.1 İndirme ve Hesap Oluşturma

1. Tarayıcıdan https://www.gowinsemi.com/en/support/download_eda/ adresine gidin (ana sayfadan *Support* → *GOWIN EDA Home* → *Download "GOWIN EDA"* yolunu izleyerek de ulaşılır). İndirme bağlantıları üyelik olmadan gösterilmez; sayfada yalnızca *"Please login or register first."* uyarısı görünür. Sağ üstteki **Register** / **Member Login** düğmesine tıklayın (Şekil 4.1.1.1, kutu 1).



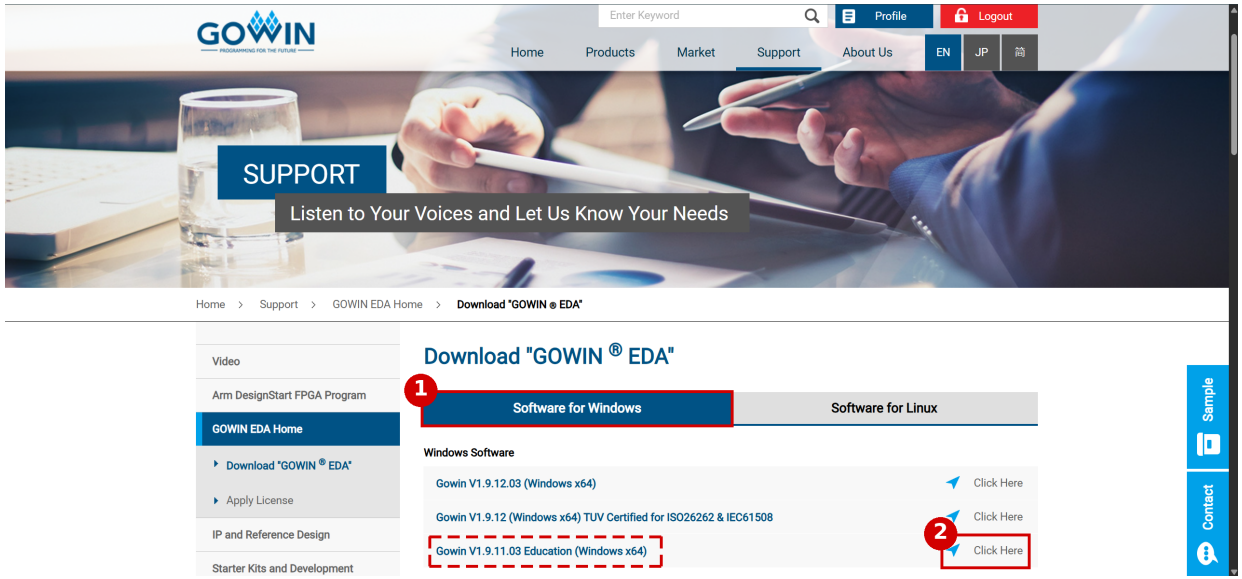
Şekil 4.1.1.1: Gowin EDA indirme sayfası. Üye girişi yapılmadığı için indirme listesi boştur; kutu 1 ile işaretli *Register / Member Login* düğmesine tıklanır.

2. Açılan sayfada hesabınız yoksa soldaki *Register* formunu doldurun: e-posta, şifre (iki kez), telefon, kullanıcı adı ve güvenlik kodu (Şekil 4.1.1.2, kesikli çerçeve). Ardından **Register** düğmesine tıklayın (kutu 1). Hesabınız zaten varsa sağdaki *Member Login* bölümüne e-posta, şifre ve güvenlik kodunu girip **Login** düğmesiyle giriş yapın (kutu 2).



Şekil 4.1.1.2: Kayıt / giriş sayfası. Kesikli çerçeve: yeni üyelikte doldurulacak alanlar; kutu 1: *Register* düğmesi; kutu 2: hesabı olanlar için *Login* düğmesi.

3. Giriş yaptıktan sonra indirme sayfasına dönüldüğünde yazılım listesi görünür. **Software for Windows** sekmesinin seçili olduğundan emin olun (Şekil 4.1.1.3, kutu 1) ve adında *Education* geçen sürümün satırındaki (kesikli çerçeve) **Click Here** bağlantısına tıklayarak kurulum dosyasını indirin (kutu 2). Sürüm numarası zamanla değişir; *Education* ibareli en güncel Windows sürümünü seçin (ekran görüntüsünde V1.9.11.03).



Şekil 4.1.1.3: Giriş sonrası indirme listesi. Kutu 1: *Software for Windows* sekmesi; kesikli çerçeve: indirilecek *Education* sürümü; kutu 2: bu satırın *Click Here* bağlantısı.

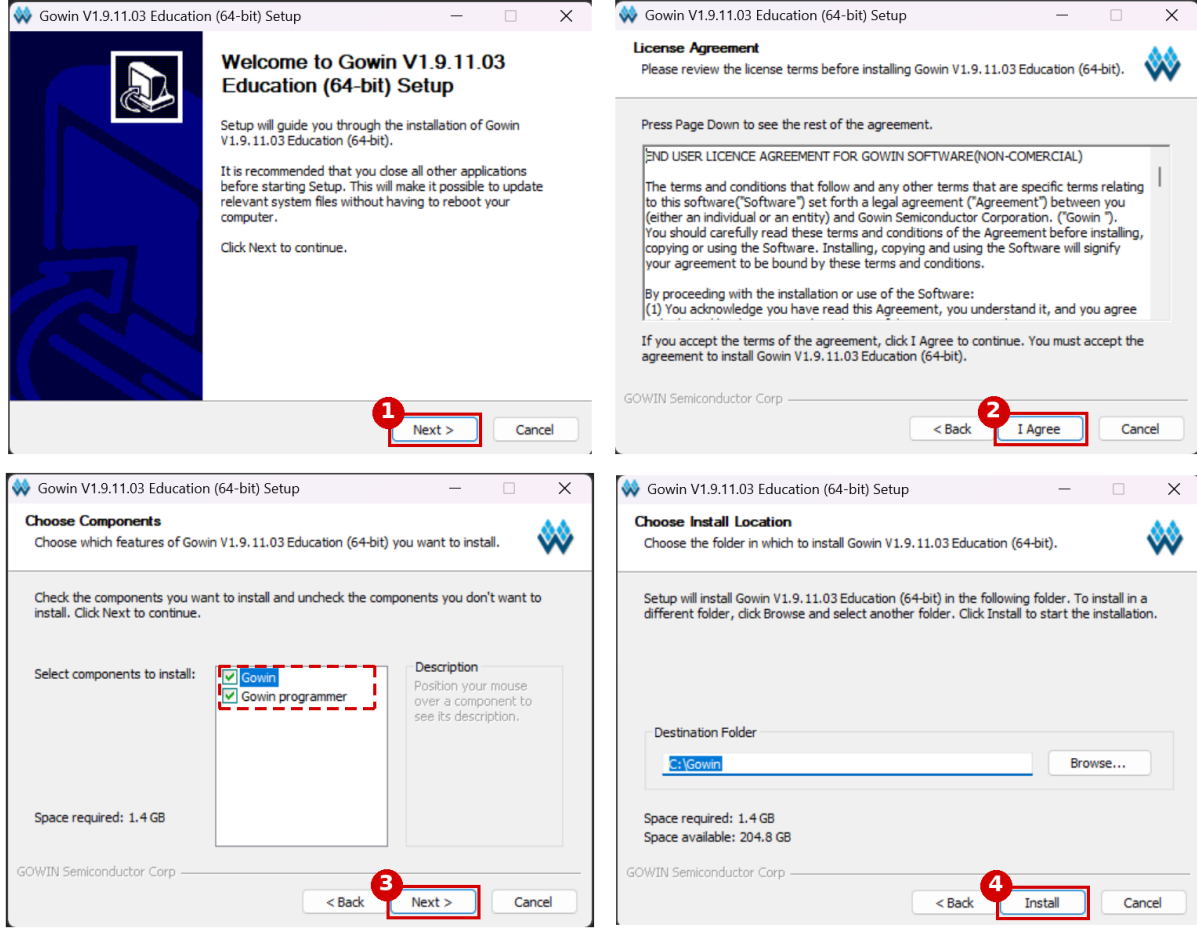
Not

Education Edition lisans gerektirmez: kurulum bittikten sonra IDE hiçbir lisans adımı sorulmadan doğrudan açılır. Bu nedenle kılavuzda ayrı bir lisanslama bölümü yoktur.

4.1.2 Windows'a Kurulum

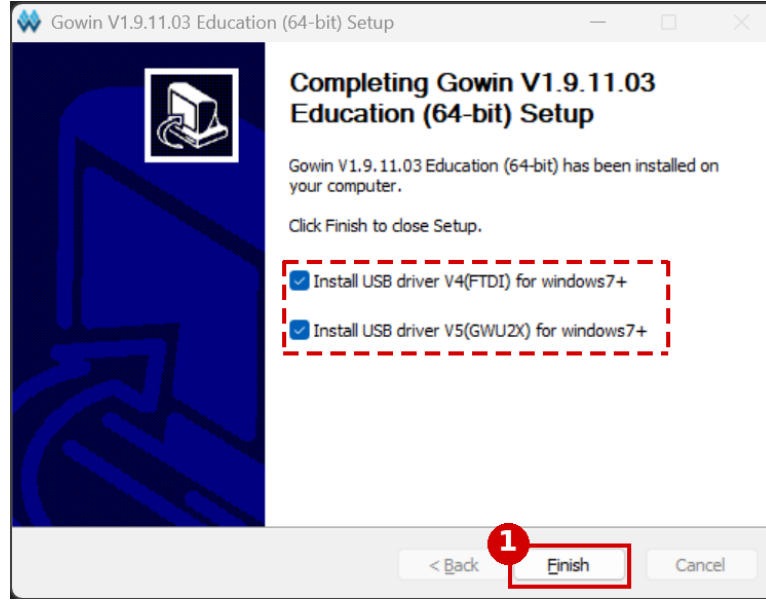
İndirilen kurulum dosyasını çalıştırın; sihirbazın dört adımı Şekil 4.1.2.1'de verilmiştir:

1. Karşılama ekranında **Next**'e tıklayın (kutu 1).
2. Lisans metnini onaylamak için **I Agree**'ye tıklayın (kutu 2).
3. Bileşen seçimi ekranında **Gowin** (IDE) ve **Gowin programmer** bileşenlerinin ikisini de işaretli bırakın (kesikli çerçeve) ve **Next**'e tıklayın (kutu 3).
4. Kurulum klasörünü (varsayılan C:\Gowin) değiştirmeden **Install**'a tıklayın (kutu 4) ve kurulumun bitmesini bekleyin.



Şekil 4.1.2.1: Gowin EDA kurulum sihirbazı. Kutu 1–4 sırasıyla tıklanacak düğmeleri, kesikli çerçeve işaretli kalması gereken bileşenleri gösterir.

Kurulumun son ekranında, kartın USB üzerinden tanınmasını sağlayan iki sürücü paketi önerilir: *Install USB driver V4 (FTDI)* ve *Install USB driver V5 (GWU2X)*. Her iki kutuyu da işaretli bırakın (Şekil 4.1.2.2, kesikli çerçeve) ve **Finish**'e tıklayın (kutu 1). Finish'ten sonra bu iki sürücünün kendi kurulum pencereleri sırayla açılır; açılan onay pencerelerini kabul etmeniz yeterlidir. Kurulum tamamlandığında masaüstünde Gowin IDE kısayolu oluşur.

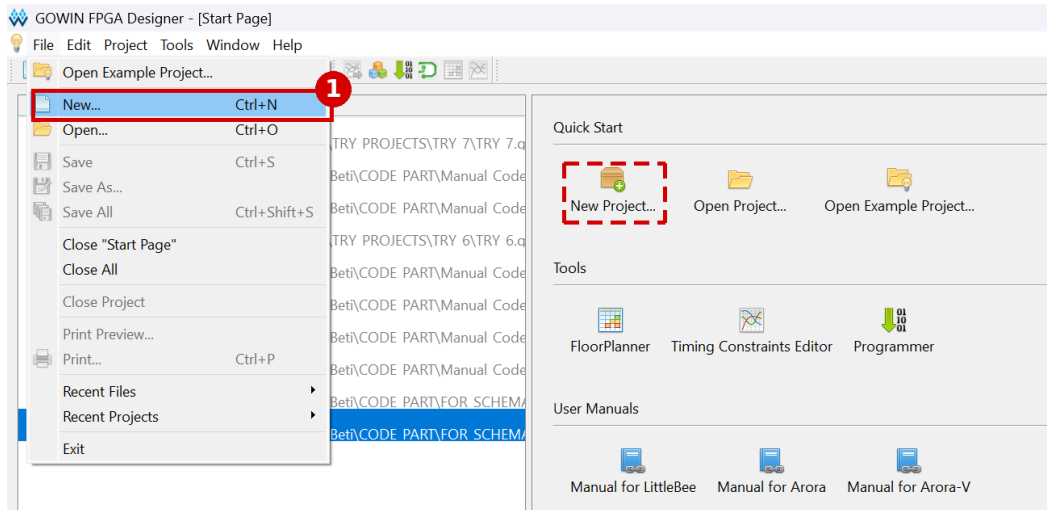


Şekil 4.1.2.2: Kurulumun son ekranı. Kesikli çerçeve: işaretli kalması gereken iki USB sürücüsü (FTDI ve GWU2X); kutu 1: *Finish* düğmesi.

4.2 Yeni Proje Oluşturma

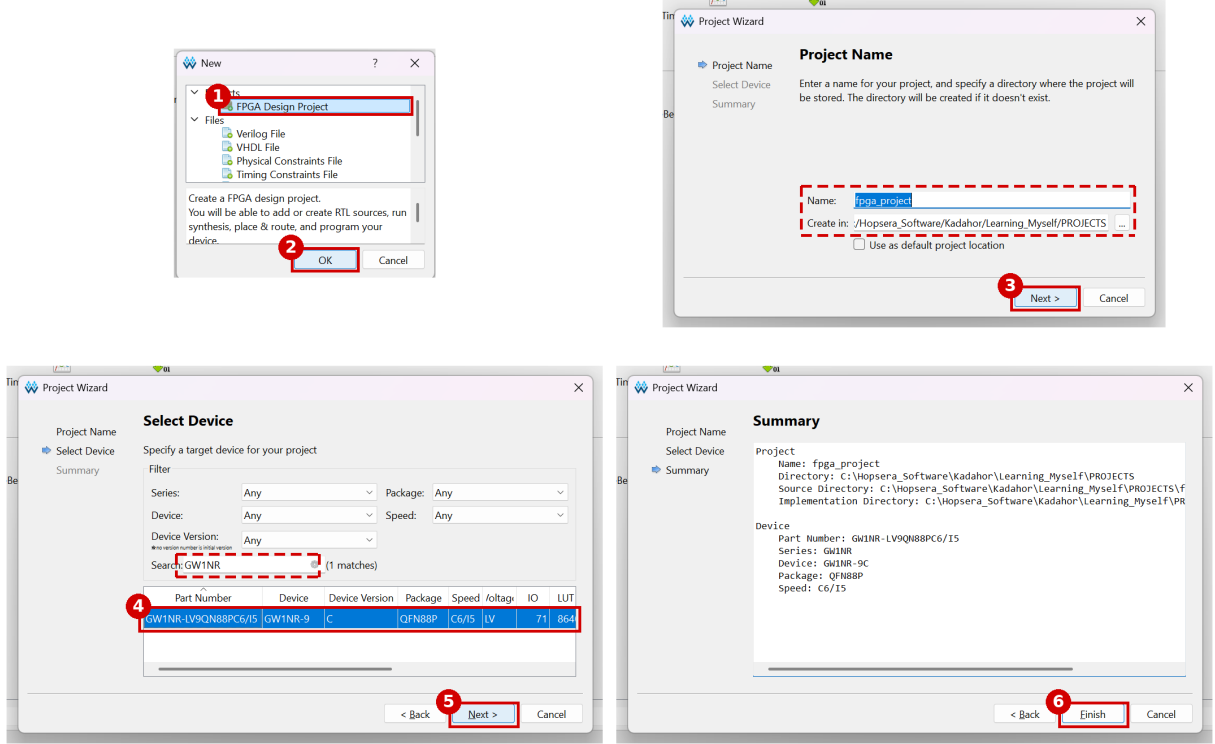
Gowin IDE’de yeni bir proje, Project Wizard ile birkaç adımda oluşturulur:

1. *File* → *New* menüsüne tıklayın (Şekil 4.2.1, kutu 1; kısayolu **Ctrl+N**). Start Page’deki *New Project...* simgesi de (kesikli çerçeve) aynı pencereyi açar.



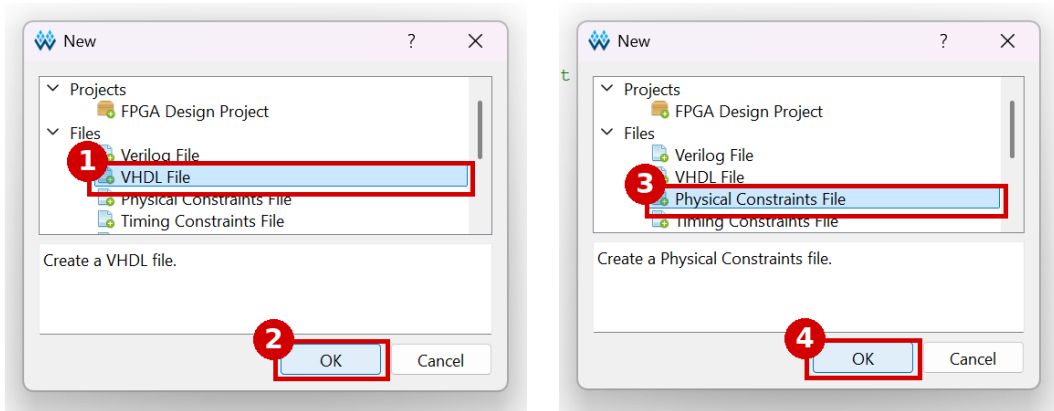
Şekil 4.2.1: Yeni proje oluşturma için iki yol: *File* → *New* menüsü (kutu 1) veya Start Page’deki *New Project...* kısayolu (kesikli çerçeve).

2. Sihirbaz adımları Şekil 4.2.2’de verilmiştir: açılan *New* penceresinde **FPGA Design Project**’i seçin (kutu 1) ve **OK**’a tıklayın (kutu 2). *Project Name* ekranında projenin adını ve kaydedileceği klasörü girin (kesikli çerçeve) ve **Next**’e tıklayın (kutu 3). *Select Device* ekranında *Search* kutusuna **GW1NR** yazın (kesikli çerçeve), filtrelenen **GW1NR-LV9QN88PC6/I5** satırına tıklayın (kutu 4) ve **Next**’e geçin (kutu 5). *Summary* ekranında seçimleri kontrol edip **Finish**’e tıklayın (kutu 6).



Şekil 4.2.2: Project Wizard adımları: proje türü ve onay (kutu 1–2), proje adı/konumu (kesikli) ve Next (kutu 3), cihaz araması (kesikli), GW1NR-LV9QN88PC6/I5 seçimi (kutu 4) ve Next (kutu 5), özet ekranında Finish (kutu 6).

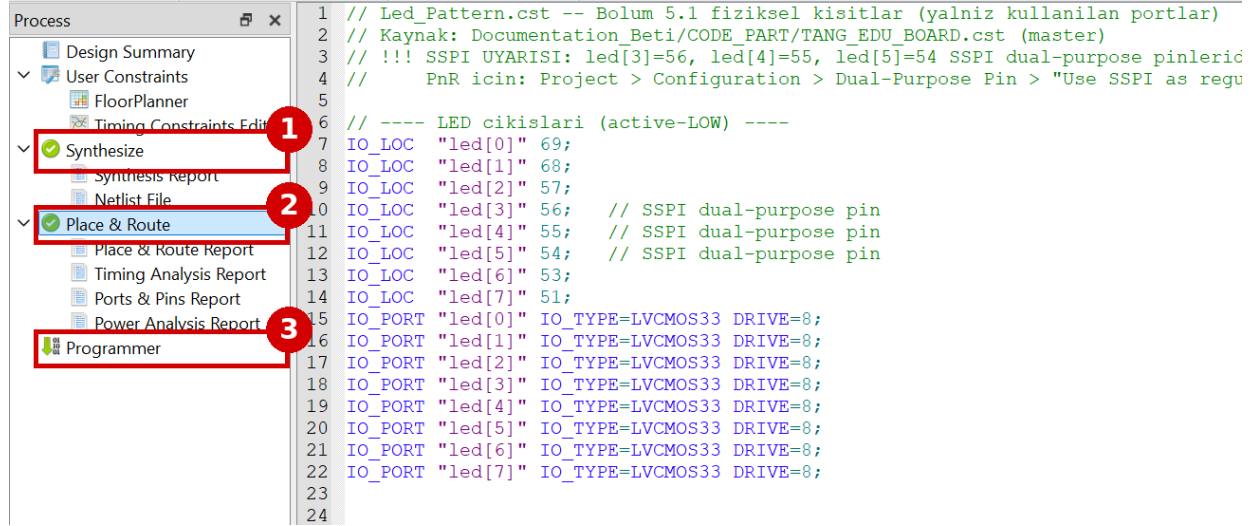
3. Proje açıldıktan sonra kaynak dosyalar eklenir: *File* → *New* penceresinden **VHDL File** ile .vhd (Şekil 4.2.3, kutu 1–2), **Physical Constraints File** ile .cst dosyası (kutu 3–4) oluşturulur; timing constraint (.sdc) için aynı penceredeki *Timing Constraints File* kullanılır. Bilgisayarda hazır duran dosyaları eklemek içinse Design ağacında proje adına sağ tıklayıp *Add Files...* seçilir. Constraint dosyalarının içeriği için Bölüm 3.3'e bakın.



Şekil 4.2.3: Projeye yeni dosya ekleme: *VHDL File* (kutu 1–2) ve *Physical Constraints File* (kutu 3–4).

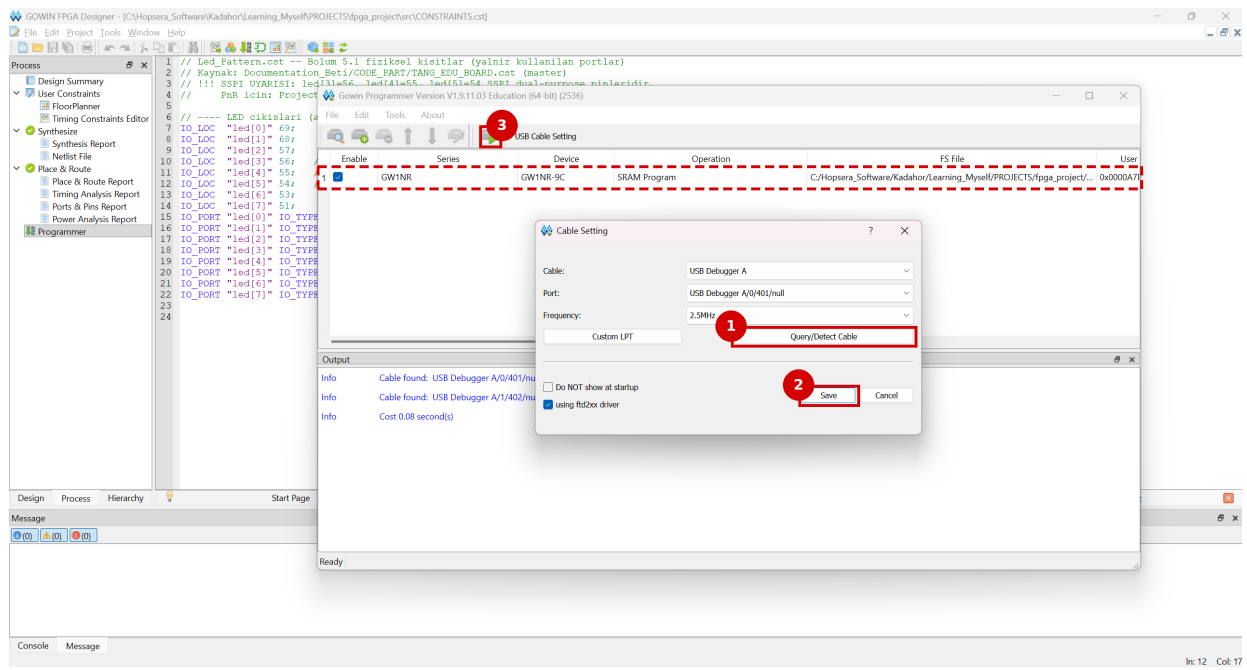
4.3 Derleme ve Programlama

Derleme, *Process* sekmesinden yürütülür (Şekil 4.3.1): önce **Synthesize**'a (kutu 1), ardından **Place & Route**'a (kutu 2) çift tıklayın. Başarıyla biten adımların yanında yeşil onay işareti belirir; bir adım hata verirse mesajlar alttaki *Message* panelinde listelenir (sık karşılaşılan hatalar için Bölüm 4.4).



Şekil 4.3.1: Process sekmesindeki akış: *Synthesize* (kutu 1) ve *Place & Route* (kutu 2) çift tıklanır; yeşil işaretler adımın başarıyla bittiğini gösterir. Karta yazmak için *Programmer* açılır (kutu 3).

Kartı programlamak için kartı USB ile bilgisayara bağlayın ve **Programmer**'ı açın (Şekil 4.3.1, kutu 3). Kart algılandığında cihaz satırında **GW1NR-9C** görünür ve *FS File* sütununda üretilen bitstream (.fs) otomatik seçilir (Şekil 4.3.2, kesikli çerçeve). Kablo ilk kez ayarlanıyorsa *USB Cable Setting* penceresinde **Query/Detect Cable** ile kabloyu buldurun (kutu 1) ve **Save** ile kaydedin (kutu 2). Yazma işlemini araç çubuğundaki yeşil *Program/Configure* düğmesi başlatır (kutu 3).



Şekil 4.3.2: Gowin Programmer. Kesikli çerçeve: algılanan kart (GW1NR-9C) ve seçili .fs dosyası; kutu 1-2: kablo ayarı (*Query/Detect Cable* → *Save*); kutu 3: programlamayı başlatan düğme.

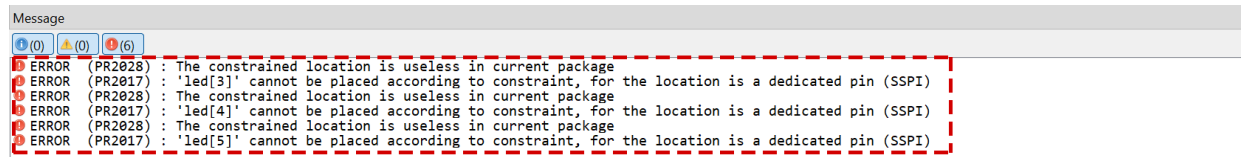
Operation sütunundaki **SRAM Program** modu bitstream'i geçici olarak yükler; kartın gücü kesildiğinde tasarım silinir. Kalıcı yükleme için *Operation* hücresine çift tıklanıp **embFlash** modlarından biri seçilir.

4.4 Sık Alınan Hatalar

Bu bölüm, kartla çalışırken en sık karşılaşılan hataları ve çözümlerini toplar. Hata mesajları Gowin IDE'nin alt kısmındaki *Console / Message* panelinde görünür.

4.4.1 PR2017/PR2028: SSPI pinleri (led[3]–led[5])

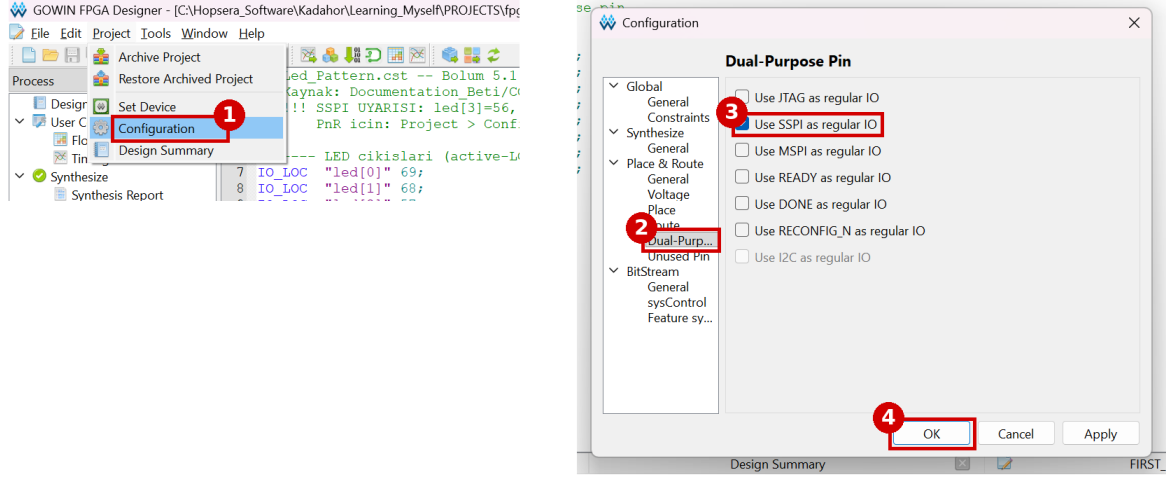
Belirti: Place & Route, PR2017 veya PR2028 kodlu bir hatayla durur; mesajda ilgili pinin SSPI'ya ayrıldığı belirtilir (Şekil 4.4.1.1).



Şekil 4.4.1.1: PR2028/PR2017 hatalarının Message panelindeki görünümü (led[3]–led[5] SSPI pinlerinde).

Neden: Pin 56/55/54 (led[3], led[4], led[5]) GW1NR-9'un SSPI dual-purpose pinleridir ve varsayılan ayarda konfigürasyon arayüzüne ayrılmıştır (bkz. Bölüm 3.3).

Çözüm: *Project* → *Configuration* → *Dual-Purpose Pin* altından “Use SSPI as regular IO” seçeneğini işaretleyin ve Place & Route'u yeniden çalıştırın (Şekil 4.4.1.2). Bu ayar proje bazlıdır; bu pinleri kullanan her yeni projede tekrarlanmalıdır.



Şekil 4.4.1.2: Çözüm adımları: *Project* → *Configuration* (kutu 1), *Place & Route* altındaki *Dual-Purpose Pin* (kutu 2), “*Use SSPI as regular IO*” işaretlenir (kutu 3) ve *OK* ile onaylanır (kutu 4).

4.4.2 “net not found”: constraint dosyasında fazla satır

Belirti: Place & Route, constraint dosyasındaki bir port için “*net not found*” benzeri bir hatayla durur.

Neden: Master TANG_EDU_BOARD.cst dosyası olduğu gibi kopyalanmış, ancak tasarımın entity’inde bu portların tümü yoktur.

Çözüm: Master dosyadan yalnızca entity’de bulunan portlara karşılık gelen satırları alın (bkz. Bölüm 3.3). Port isimlerinin entity ile birebir aynı olduğunu da kontrol edin.

4.4.3 Programmer kartı görmüyor veya yazma başarısız oluyor

Belirti: Programmer penceresinde cihaz listesi boş kalır ya da yazma sırasında hata alınır.

Çözüm: Sırasıyla deneyin:

1. USB kablosunu ve portunu değiştirin; yalnızca şarj amaçlı (veri hattı olmayan) kablolar kartı göstermez.
2. Kurulumda USB/JTAG sürücülerinin yüklendiğini doğrulayın (bkz. Bölüm 4.1).
3. Sorun sürüyorsa IDE ile gelen Programmer, Tang Nano’nun USB-JTAG donanımıyla uyumsuz olabilir. Sipeed’in dağıttığı Programmer sürümünü indirip onu kullanın: <https://dl.sipeed.com/shareURL/TANG/programmer>

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 davranış tanımlayan **architecture**.

Kod 4.5.1: Bir VHDL dosyasının temel yapısı

```

1 library ieee;                -- standart kutuphane
2 use ieee.std_logic_1164.all; -- std_logic tipleri için gerekli
3
4 entity ornek_tasarim is
5     port (

```

```

6      sw : in  std_logic;           -- giris : tek bit
7      btn : in  std_logic;         -- giris : tek bit
8      led : out std_logic_vector(1 downto 0) -- cikis : 2 bitlik demet
9  );
10 end entity ornek_tasarim;
11
12 architecture rtl of ornek_tasarim is
13     signal ara_sinyal : std_logic; -- ic baglanti, disari cikmaz
14 begin
15     ara_sinyal <= sw and btn;      -- eszamanli atama
16     led(0)    <= ara_sinyal;
17     led(1)    <= not ara_sinyal;
18 end architecture rtl;

```

Kısaca:

- **entity** tasarımın port listesini tanımlar. Her portun bir yönü ve bir tipi vardır: butonlar ve switchler gibi girişler *in*, LED’ler gibi çıkışlar *out* olur. Port satırları noktalı virgülle *ayrılır*; bu yüzden listedeki son porttan sonra noktalı virgül yazılmaz.
- **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 değer *saklanması* gerektiğinde (sayaç, kaydırmalı register, durum makinesi) **process** kullanılır:

Kod 4.5.2: Clock ile çalışan process kalıbı

```

1 process(clk)           -- duyarlılık listesi
2 begin
3     if rising_edge(clk) then -- clock'un yükselen kenarında
4         sayac <= sayac + 1;   -- deger saklanır ve guncellenir
5     end if;
6 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 LED'ler

Amaç Kart üzerindeki sekiz LED'i sabit bir desenle yakmak. Bu, kartla yapacağınız ilk tasarımıdır: 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 LED'ler **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 unutulmuş 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 tasarımıdır:

Kod 5.1.1: LED deseni yazdırma

```

1 library ieee;                      -- standart kutuphane
2 use ieee.std_logic_1164.all;       -- std_logic tipleri için
3
4 entity led_pattern is
5     port (
6         led : out std_logic_vector(7 downto 0) -- 8 LED çıkışı
7     );
8 end entity led_pattern;
9
10 architecture rtl of led_pattern is
11 begin
12     -- LED'ler active-LOW: '0' yazılan LED yanar.
13     -- "10101010" -> led(0), led(2), led(4), led(6) yanar.

```

```

14 led <= "10101010";          -- sabit desen; en sol bit = led(7)
15 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 LED'leri 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 sönmük kalır. Desendeki '0' ve '1' değerlerini ters çevirirseniz yanan ve sönen LED'ler 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

```

1 -- btn ve led ikisi de active-LOW olduğundan
2 -- doğrudan bağlantı doğru sonucu verir:
3 -- basılı buton ('0') -> LED yanar ('0').
4 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

```

1 btn_pressed <= not btn(0);          -- '1' = buton basılı
2 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 davranış 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 yukarı/pull-up konumunu ifade eder. Tüm örneklerde *yukarı konum ('1')* “aktif” kabul edilir; 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: yukarı konumdaki switch '1' okunur, LED ise '0' yazılınca yanar. Yukarı alınan switch'in LED'i yakması için sinyal terslenir:

Kod 5.3.1: Switchler ile LED kontrolü

```
1 -- Switch yukari konumda pull-up ile '1' okunur.
2 -- LED'ler '0' yazilınca yanar.
3 -- Tersleme sayesinde: switch yukari -> LED yanar.
4 led <= not sw;
```

Doğrudan `led <= sw;` bağlantısı da geçerli bir tasarımdır; bu durumda ilişki tersine döner ve aşağı konum ('0') LED'i yakar. Bu iki farklı kullanım, elektriksel seviye ile kullanıcıya gösterilen anlamın aynı şey olmadığını öğretir; kılavuzdaki örnekler tutarlılık için hep “yukarı = aktif” kuralını izler.

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

Sık Kontrol Edilecek Noktalar Bu bölümde hata ararken önce aktif seviyeleri kontrol edin. LED'ler active-LOW olduğu için '0' ile yanar. Butonlar basılıyken '0' okunur. Slide switch aşağı konumdayken GND'ye bağlıdır ve '0' okunur; örneklerde yukarı konum ('1') aktif kabul edilir. Ayrıca LED3, LED4 ve LED5 (pin 56/55/54) FPGA'nın SSPI dual-purpose pinleri üzerindedir; bu LED'leri kullanan projelerde “*Use SSPI as regular IO*” ayarını işaretlemeyi unutmayın (bkz. Bölüm 4.4).

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

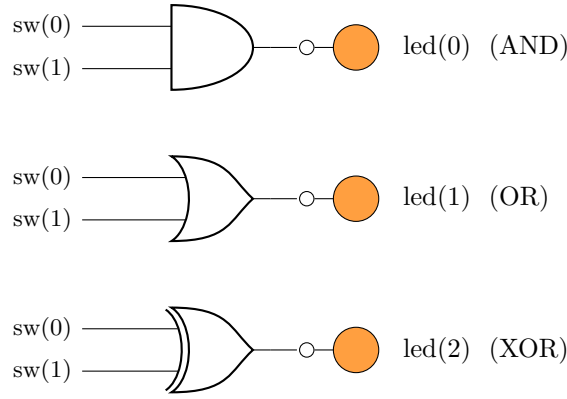
Kartta: `sw(0)` ve `sw(1)` iki giriş olsun; farklı kapıları farklı LED'lere bağlayarak hepsini aynı anda gözlemleyebilirsiniz. Sonucun "1" olduğunu LED'in yanmasıyla göstermek için başa `not` eklenir (LED'ler active-LOW):

Kod 6.1.1.1: Aynı girişleri birden fazla kapıyla gözlemleme

```

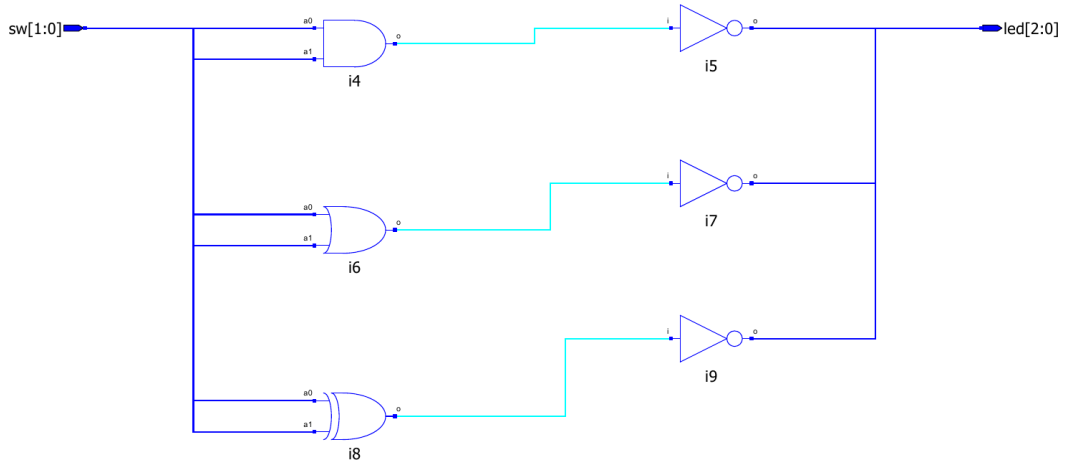
1 led(0) <= not (sw(0) and sw(1)); -- AND sonucu
2 led(1) <= not (sw(0) or sw(1));  -- OR sonucu
3 led(2) <= not (sw(0) xor sw(1)); -- XOR sonucu

```



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 görüldüğü Şekil 6.1.1.2'de verilmiştir.



Şekil 6.1.1.2: 6.1.1 tasarımının Gowin RTL Design Viewer görünümü.

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

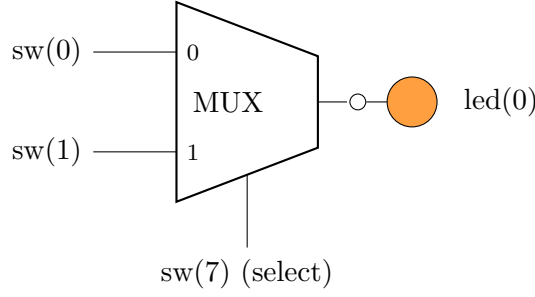
Multiplexer, select hattının değerine göre girişlerinden birini çıkışa aktaran bir seçicidir. VHDL'de bunun doğal karşılığı *when/else* koşullu atamasıdır:

Kod 6.1.2.1: 2'ye 1 multiplexer

```
1 mux_out <= sw(0) when sw(7) = '0' else sw(1);
2 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, Subtractor 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 toplayıcı; beşinci bit carry

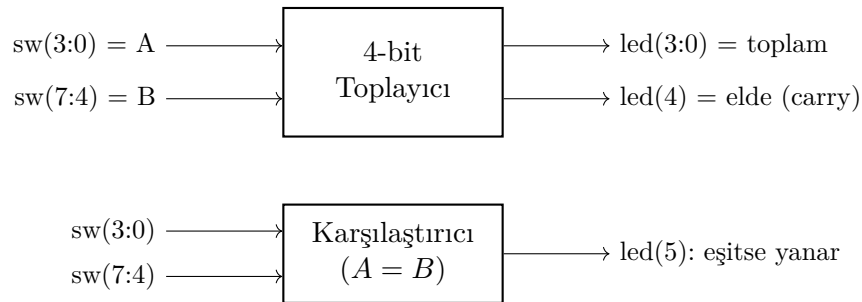
```
1 use ieee.numeric_std.all;           -- unsigned işlemler için
2
3 toplam <= unsigned('0' & sw(3 downto 0))
4       + unsigned('0' & sw(7 downto 4));
5 led(4 downto 0) <= not std_logic_vector(toplam);
```

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: Eşitlik karşılaştırıcı

```
1 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 LED'lere 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.

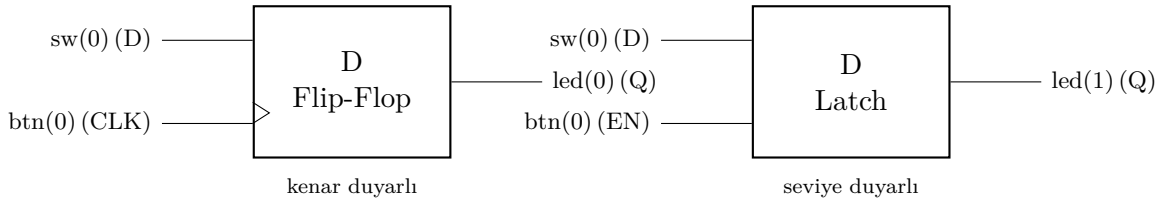
6.2.1 Latch ve Flip-Flop

İkisi de bir bit saklar; fark *ne zaman* güncellediklerindedir: 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

```
1 if falling_edge(btn(0)) then -- basma ani (buton active-LOW)
2   saklanan <= sw(0);         -- data yalnızca bu anda orneklenir
3 end if;
4 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 çekirdeği

```
1 if falling_edge(btn(0)) then -- her buton basısında bir kez
2   q <= q(6 downto 0) & sw(0); -- sola kaydır, yeni bit sağdan
3 end if;
4 led <= not q;
```

Kartta: `sw(0)`'ı '1' yapıp butona birkaç kez basın: LED'lerde 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: Gözle izlenebilir sayaç

```
1 signal sayac : unsigned(26 downto 0) := (others => '0');
2 ...
3 if rising_edge(clk) then
```

```

4   sayac <= sayac + 1; -- her yükselen kenarda 1 artar
5 end if; -- (saniyede 27 milyon kez!)
6 led <= not std_logic_vector(sayac(26 downto 19));

```

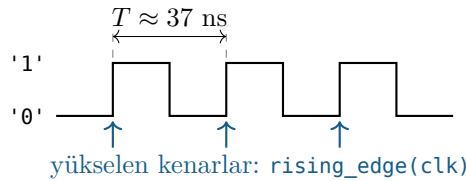
Sinyal tanımındaki `:= (others => '0')` başlangıç değeri, sayacın hem simülasyonda tanımsız (U) başlamasını önler hem de FPGA açılışında sıfırdan saymaya başlamasını garanti eder; bu kalıbı değer saklayan tüm sinyallerde kullanmak iyi bir alışkanlıktır.

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 $T = 1/27 \text{ MHz} \approx 37 \text{ ns}$ ’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. Bu kılavuzda yalnızca butonla clock’lanan öğretim deneyleri (Bölüm 6.2’deki flip-flop ve shift register denemeleri ile Bölüm 6.3’teki durum makinesi) bu ilkenin bilinçli istisnalarıdır; gerçek tasarımlarda tüm flip-floplar sistem clock’u ile sürülür. 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: Üç durumlu makine iskeleti

```

1 type durum_tipi is (DUR, YURU, YANIP_SON);
2 signal durum : durum_tipi := DUR; -- acilista DUR durumu
3 ...
4 if falling_edge(btn(0)) then
5     case durum is
6         -- '>' case dalini acar: "durum bu ise ..."
7         -- '<=' ise siradaki durumu atar
8         when DUR      => durum <= YURU;
9         when YURU     => durum <= YANIP_SON;
10        when YANIP_SON => durum <= DUR;
11    end case;
12 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.

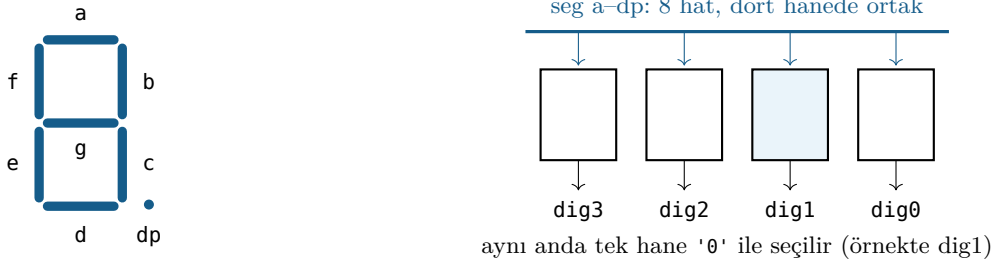
Bölüm 7

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 — bu bölümdeki örneklerde kullanılmadığından **seg(7) <= '0'**; ile sönmük tutulmalıdır. 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)

```

1 case deger is
2   when "0000" => seg(6 downto 0) <= "0111111"; -- 0
3   when "0001" => seg(6 downto 0) <= "0000110"; -- 1
4   when "0010" => seg(6 downto 0) <= "1011011"; -- 2
5   when "0011" => seg(6 downto 0) <= "1001111"; -- 3
6   when "0100" => seg(6 downto 0) <= "1100110"; -- 4
7   when "0101" => seg(6 downto 0) <= "1101101"; -- 5
8   when "0110" => seg(6 downto 0) <= "1111101"; -- 6
9   when "0111" => seg(6 downto 0) <= "0000111"; -- 7
10  when "1000" => seg(6 downto 0) <= "1111111"; -- 8
11  when "1001" => seg(6 downto 0) <= "1101111"; -- 9
12  when "1010" => seg(6 downto 0) <= "1110111"; -- A
13  when "1011" => seg(6 downto 0) <= "1111100"; -- b
14  when "1100" => seg(6 downto 0) <= "0111001"; -- C

```

```

15  when "1101" => seg(6 downto 0) <= "1011110"; -- d
16  when "1110" => seg(6 downto 0) <= "1111001"; -- E
17  when others => seg(6 downto 0) <= "1110001"; -- F
18  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 hızlı 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

```

1  hane <= sayac(16 downto 15); -- ~824 Hz'de ilerler,
2                                -- 0..3 tam turu ~206 Hz
3  dig <= "1110" when hane = "00" else -- dig0 secili ('0' = aktif)
4        "1101" when hane = "01" else -- dig1
5        "1011" when hane = "10" else -- dig2
6        "0111"; -- dig3

```

Seçilen haneyle eş zamanlı olarak *seg* çıkışına o hanenin rakam deseni yazılır. Bu hızda her hane saniyede yaklaşık 206 kez yenilenir 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

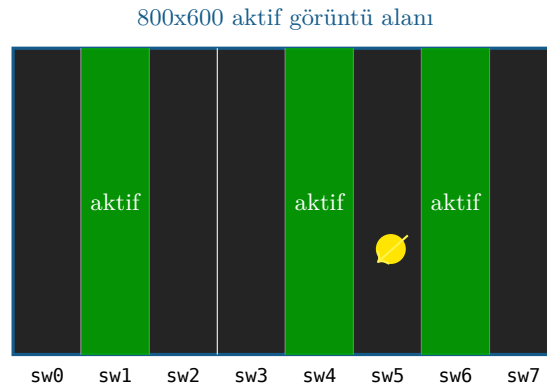
HDMI TMDS hatları Tang Nano 9K üzerindeki paylaşımlı pinleri kullanır. Bu projede pin 68–75 aralığı HDMI için ayrıldığı için Tang Edu Board üzerindeki LED0/LED1, BTN0–BTN3 ve DIG0/DIG1 aynı anda kullanılmaz. HDMI denemesi yaparken bu pinlere bağlı diğer çevre birimlerinden davranış beklemeyin. Ayrıca bu birimler (LED dirençleri, buton pull-up/RC devreleri, digit sürücüleri) aynı pinlere fiziksel olarak bağlı kaldığından yüksek hızlı TMDS hatlarına ek elektriksel yük getirir; bazı monitör ve kablo kombinasyonlarında görüntü kararlılığı bundan etkilenebilir. Görüntü kararsızsa Tang Nano 9K modülünü taşıyıcı karttan ayırarak denemek ayrıştırıcı bir testtir.

Bu bölümdeki örnek proje, Beti Elektronik ürün sayfasından edinilebilen kılavuz kod projeleri arşivindeki 7_2 klasöründedir. Proje, 27 MHz kart saatinden PLL ve clock divider bloklarıyla yaklaşık 40 MHz pixel clock ve bunun 5 katı olan yaklaşık 199.8 MHz serializer clock’u üretir. TMDS veri hatları bu saatin her iki kenarını da kullanarak (DDR) hat başına yaklaşık 400 Mbit/s taşır; HDMI clock kanalının frekansı ise pixel clock ile aynıdır (yaklaşık 40 MHz). 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 pırlıltı yapabilir. 800x600 @ 60 Hz bu kart ve kablo düzeninde daha dengeli bir deneme çözümüdür; 16:9 paneller görüntüyü yatayda esnetebilir.

Ekranı 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

```

1 entity hdmi_ball_walls is
2   port (
3     clk      : in  std_logic;           -- 27 MHz kart saati
4     sw       : in  std_logic_vector(7 downto 0); -- duvar secen switchler
5     tmds_clk_p : out std_logic;         -- HDMI clock, pozitif hat
6     tmds_clk_n : out std_logic;         -- HDMI clock, negatif hat
7     tmds_d_p  : out std_logic_vector(2 downto 0); -- TMDS veri hatlari, pozitif
8     tmds_d_n  : out std_logic_vector(2 downto 0); -- TMDS veri hatlari, negatif
9   );
10 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

```

1 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

```

1 // TMDS data lane 0: pozitif pin 71, negatif pin 70
2 IO_LOC "tmds_d_p[0]" 71,70;
3
4 // TMDS data lane 1: pozitif pin 73, negatif pin 72
5 IO_LOC "tmds_d_p[1]" 73,72;
6
7 // TMDS data lane 2: pozitif pin 75, negatif pin 74
8 IO_LOC "tmds_d_p[2]" 75,74;
9
10 // TMDS clock lane: pozitif pin 69, negatif pin 68
11 IO_LOC "tmds_clk_p" 69,68;

```

Not

Negatif portlar (`tmds*_n`) .cst dosyasında ayrıca adlandırılmaz: `IO_LOC` satırında `_p` port adına verilen pin çiftinin ikinci numarası (örneğin 71,70 çiftindeki 70) diferansiyel çiftin negatif ucudur. Çiftin iki ucunu birlikte süren eleman, VHDL kodundaki `ELVDS_OBUF` diferansiyel çıkış buffer'ıdır; `_n` portları bu buffer'ın `OB` çıkışına bağlanır.

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 sınırdadır; 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	LVCMS33
User pinler, 1.8 V	79, 80, 81, 82, 83, 84, 85, 86	LVCMS18
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=LVCMS18` 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

```
1 // 3.3 V bankındaki bir user pin
2 IO_LOC "harici_giris" 63;
3 IO_PORT "harici_giris" IO_TYPE=LVCMS33;
4
5 // 1.8 V bankındaki bir user pin -- LVCMS18 zorunlu
6 IO_LOC "harici_lv8" 79;
7 IO_PORT "harici_lv8" IO_TYPE=LVCMS18;
```

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.