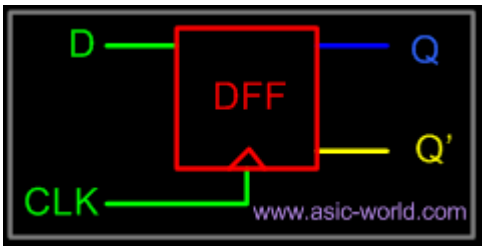
Giriş

Giriş

Verilog bir donanım tanımlama dilidir(**HARDWARE DESCRIPTION LANGUAGE (HDL)**). Bir donanım tanımlama dili dijital sistemleri tanımlamak için kullanılan bir dildir: örneğin, bir ağ anahtarı, bir mikroişlemci veya bir bellek veya basit bir flip-flop. Bunun anlamı, HDL kullanan biri herhangi seviyedeki herhangi bir (dijital-sayısal)donanımı tanımlayabilir.




```
 1 // D flip-flop Code
 2 module d_ff ( d, clk, q, q_bar);
 3 input d ,clk;
 4 output q, q_bar;
 5 wire d ,clk;
 6 reg q, q_bar;
 7
 8 always @ (posedge clk)
 9 begin
10   q <= d;
11   q_bar <=  ! d;
12 end
13
14 endmodule
```

Herhangi biri basit bir Flip flobu şekildek gösterildiği gibi tanımlayabilir, aynı şekilde 1 milyon kapısı olan karmaşık tasarımları da tanımlayabilir. Verilog, sanayide donanım tasarımı için kullanılan HDL dillerinden biridir. Bize, Davranışsal Seviyede(Behavior Level), Yazmaç Transfer Seviyesinde(Register Transfer Level (RTL)), Kapı Seviyesinde(Gate Level) ve anahtarlama seviyesinde(switch level) dijital bir tasarım yapmamıza izin verir. Verilog donanım tasarımcılarına tasarımlarını davranışsal yapıyla belirtmesine izin verir, gerçekleştirme(implementasyon) ayrıntılarını son tasarımdaki bir sonraki adıma ertelemesini sağlar.

Bu dili öğrenmek isteyen birçok mühendis, sıklıkla şu soruyu sormaktadır, Verilog öğrenmek ne kadar zaman alır? İyi haber, benim onlara cevabım, **eğer daha önce bir programlama dili biliyorsanız bir haftadan fazla sürmeyeceğidir.**

Tasarım Şekilleri

Verilog birçok donanım tanımlama dilindeki gibi aşağıdan yukarıya(Bottom-up) veya Yukarıdan aşağıya(Top-down) metodolojiye izin vermektedir.

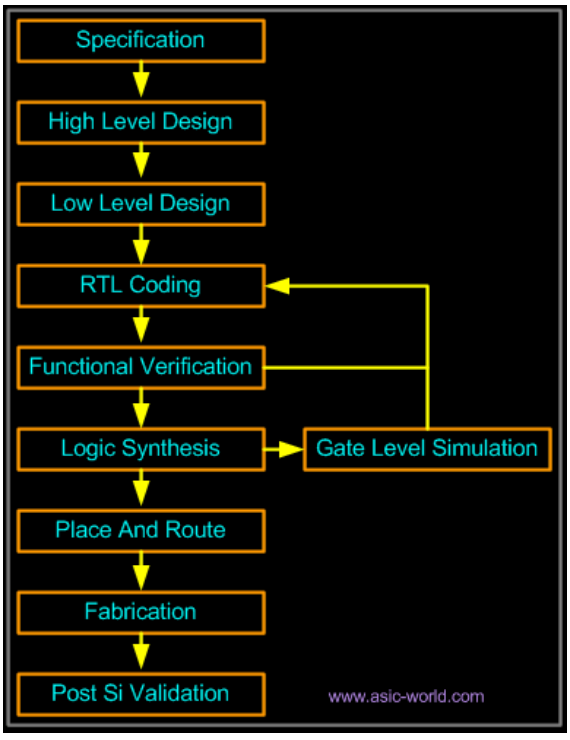
Aşağıdan Yukarıya Tasarım (Bottom-Up Design)

Elektornik tasarımdaki geleneksel metod ağaşıdan yukarıyadır. Herbir tasarım standart kapılar kullanılarak kapı-seviyesinde(gate-level) gerçekleşmektedir. Yeni tasarımlardaki karmaşıklık arttıkça bu metodun uygunlanmasını neredeyse imkansız kılmıştır. Yeni sistemler ASIC veya binlerce transistör içeren mikroişlemcilerden oluşmaktadır. Bu geleneksel aşağıdan yukarıya tasarım yerini yeni yapısal hiyerarşik tasarım metodlara bırakmak zorundadır. Bu yeni uygulamalar olmadan yeni karmaşıklıktaki tasarımların üstesinden gelmek imkansız olacaktır.

Yukarıdan Aşağıya Tasarım (Top-Down Design)

Her tasarımcının istediği tasarım şekli yukarıdan aşağıya olanıdır. Gerçk bir yukarıdan-aşağıya tasarım erken teste, farklı teknolojilerin kolay değişimine, yapısal sistem tasarımlarına ve birçok diğer öneri avantajlarına izin verir. Ancak tam bir yukarıdan aşağıya tasarımı takip etmek çok zordur. Bu nedenlerden dolayı, birçok tasarım her iki metodun karışımı şeklinde, herbir tasarım stilinin önemli elemanları gerçeklenerek tasarlanmaktadır.

Aşağıdaki şekil Yukarıdan-Aşağıya tasarım uygulamasını göstermektedir.



Verilog Soyutlama Seviyeleri (Abstraction Levels)

Verilog birçok farklı seviyedeki soyutlamayı desteklemektedir. Bunlardan üçü çok önemlidir:

- Davranışsal seviye (Behavioral level)
- Yazmaç Transfer Seviyesi(Register-Transfer Level-RTL)
- Kapı Seviyesi (Gate Level)

Davranışsal seviye(Behavioral level)

Bu seviye, koşutzamanlı(aynı zamanda olan) algoritmalar(Davranışsal) ile bir sistemi tanımlar. Herbir algoritmanın kendisi sıralıdır, bunun anlamı birbiri ardına gerçeklenen komutlar kümesini içermektedir. Fonksiyonlar(Function), Görevler(Task), ve Herzaman(Always) blokları temel elemanlarıdır. Tasarımın yapısal gerçeklemesiyle alakalı değildir.

Yazmaç Transfer Seviyesi (Register-Transfer Level)

Yazmaç Transfer Seviyesi kullanarak tasarlama, bir devrenin işlemlerini ve yazmaçlar arasındaki verilerin transferinin karakteristiğini belirtir. Harici bir saat kullanılır. RTL tasarım kesin zaman kısıtları içerir: işlemler kesin bir zamanda gerçekleştirilecek şekilde planlanmıştır. Modern RTL kodun tanımı,"Sentezlenebilen herhangi bir koda RTL kodu denir".

Kapı Seviyesi (Gate Level)

Lojik seviyesi içerisinde bir sistemin karakteristiği mantıksal bağlantılarla ve onların zamanlama özellikleriyle tanımlanmıştır. Tüm sinyaller ayrık sinyallerdir. Sadece belirli mantıksal(lojik) değerlere (`0', `1', `X', `Z`) sahip olabilirler. Kullanılabilir işlemler ilkel lojikle (AND, OR, NOT vb kapılar) önceden tanımlanmıştır. Kapı seviyesi modelleme kullanma, lojik tasarımın herhangi bir seviyesinde iyi bir fikir değildir. Kapı seviyesi kod, sentez araçlarına benzer araçlarla üretilirler ve netlist kapı seviyesi simülasyon ve arka uç için kullanılmaktadır.

Bir Günde Verilog

Giriş

Her bir yeni kullanıcı Verilog'u bir günde öğrenmeyi düşler,bunun kullanmak için yeterli olmasını ister. Benim ilerde belirteceğim birkaç sayfa sizin bu hayalinizi gerçekleştirecek. Burada bazı teori ve egzersizleri takip eden örnekler olacaktır. Bu eğitsel çalışma size nasıl programlama yapılacağını öğretmez; bazı programlama deneyimleriyle tasarlanmıştır. Bununla birlikte Verilog farklı kod bloklarını koşutzamanlı olarak gerçekler birçok programlama dilindeki sıralı gerçeklemesinin tersine, hâla bazı paralellikler vardır. Bazı dijital tasarımdaki deneyimleri elbette faydalıdır.

Verilogdan önceki yaşam şematikle dolu bir yaşamdı. Herbir tasarım, karmaşıklığına bakılmadan , şematik üzerinden tasarlanırdı. Bunlarını doğrulanması ve hata eğilimi, uzun sürede sonuç vermekteydi, sıkıcı geliştirme süreci, doğrulama … tasarım,doğrulama… tasarım,doğrulama… şeklinde sürüp gitmekteydi.

Verilog ortaya çıktığında, lojik devreler hakkında farklı düşünce yollarına sahip olmaya başladık. Verilog tasarım döngüsü geleneksel programlama dillerindeki gibiydi, tıpkı burada bahsedeceğimiz gibi:

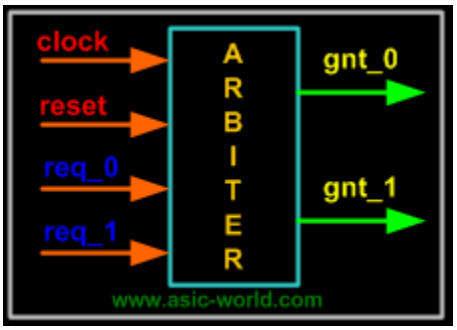
- Belirtimler (Specifications (specs))
- Yüksek seviyeli tasarım (High level design)
- Düşük seviyeli tasarım(Low level (micro) design)
- RTL kodlama
- Doğrulama (Verification)
- Sentez (Synthesis).

Listedeki ilk şey **belirtimler**(specifications)- tasarımımızdaki kısıtlamalar gereklilikler nelerdir? Ne inşa etmeye çalışıyoruz? Bu eğitsel için, iki etken arabulucusu(two agent arbiter) inşa edeceğiz: birbirine baskın çıkmaya çalışan iki etkeni seçen aygıt. Aşağıda bazı belirtimler bulunmaktadır.

- İki etken arabulucusu (Two agent arbiter).
- Aktif yüksek asenkron sıfırlama(Active high asynchronous reset).
- Sabit öncelik, 0 'ın 1 üzerindeki önceliği
- Onaylama istek kabul edildiği zaman kabul edilir.

Eğer belirtimlerimiz varsa, blok diyagramı çizebiliriz, bu da temelde sistemdeki veri akışının soyutlamasıdır(bloğa ne gelecek ne gidecek). Aşağıdaki örnekte bunun basit şeklidir. Henüz büyülü kara kutunun içinde ne olduğu konusunda endişe duymamaktayız.

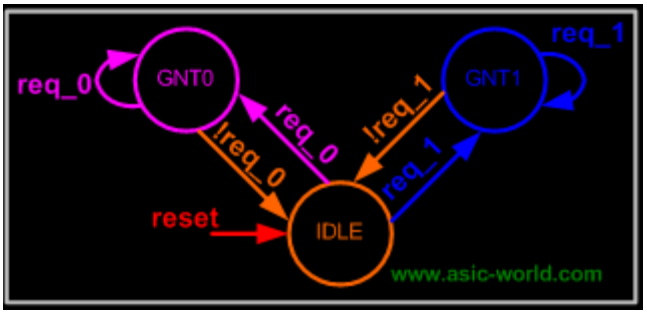
Ara bulucunun blok diyagramı



Eğer bir makinenin tasarımını Verilog olmadan yapıyorsak, standart prosedür bizi durum makinesi çizmeye zorlamaktadır. Bundan sonra herbir flipflop için durum geçişlerini gösteren doğruluk tablosu yapılmalıdır. Daha sonra da Karnaugh haritası çizerek, ve K-haritasından elde edilmiş optimize devre elde ederiz. Bu method küçük tasarımlar için gayet iyidir, ancak büyük tasarımlar için bu akış komplike ve hataya meyilli bir hal alır. İşte burada Verilog devreye girmektedir ve başka bir yol göstermektedir.

Düşük seviyeli tasarım

Verilog'un bizim arabulucumuzun tasarımında nasıl yardımcı olduğunu görmek için, bizim durum makinemize gidelim-şimdi düşük seviye tasarımın içindeyiz ve önceki diyagramdaki girişlerimizin makineyi nasıl etkilediğini görmek için kara kutuyu açalım.



Herbir halka makinedeki bir **durumu** temsil etmektedir. Herbir durum bir çıkışı bildirmektedir. Durumlar arasındaki oklar durum geçişleridir, geçişe neden olan olayla etiketlendirilmiştir. Örneğin, en soldaki turuncu okun anlamı; eğer makine GNT0(GNT0 uygun çıkış sinyali) durumundaysa ve eğer !req_0 giriş sinyalini aldıysa, makine IDLE durumuna geçecektir ve buna uygun çıkış sinyali verecektir. Bu durum makinesi ihtiyaç duyduğunuz sistemin tüm mantığını tanımlamaktadır. Bir sonraki adım bunun tamamını Verilog'a koymaktır.

Modüller

Bunu yapmak için biraz geriye dönmek gerekmektedir. Eğer ilk resimdeki arabulucu bloğa bakarsanız, isminin arabulucu("arbiter") olduğunu ve giriş/çıkış portlarına(req_0, req_1, gnt_0, ve gnt_1) sahip olduğunu görürüz.

Verilog bir HDL olduğundan, bunlara ihtiyaç duyacaktır. Verilogda, bizim kara kutularımıza **modül(module)** adı vereceğiz. Bu program içerisinde girişleri, çıkışları ve iç lojik işlerle ilgili şeyleri gösteren korunmuş bir sözcüktür; buna kabaca diğer programlama dillerindeki geri dönüş tiplerine(return) denktir diyebiliriz.

Arabulucu("arbiter") modül kodu

Eğer arabulucu(arbiter) bloğuna yakından bakacak olursak burada ok yönlerinin olduğunu görürüz(giren ok giriş için, çıkan ok çıkış için). Verilogda, modül ismini ve port isimlerini bildirdikten sonra, herbir port un yönünü tanımlayabiliriz.(versiyon notu:Verilog 2001 de portları ve port yönlerini aynı anda tanımlayabiliriz)Aşağıda bu kod gösterilmiştir.

```
 1 module arbiter (
 2 // Yanyana iki tire açıklama için kullanılır.
 3 clock      , // clock
 4 reset      , // Active high, syn reset
 5 req_0      , // Request 0
 6 req_1      , // Request 1
 7 gnt_0      , // Grant 0
 8 gnt_1        // Grant 1
 9 );
10 //------------Çıkış Portları------------------------------
11 // Not : tüm komutlar noktalı virgül ile bitirilir
12 input           clock               ;
13 input           reset               ;
14 input           req_0               ;
15 input           req_1               ;
16 //------------Çıkış Portları------------------------------
17 output        gnt_0                 ;
18 output        gnt_1                 ;
```

You could download file one_day1.v here

Burada sadece iki port vardır giriş ve çıkış portları. Gerçek hayatta çift yönlü portlarımız vardır. Verilog'da çift yönlü portları tanımlamak için "inout" kullanılır.

Çift Yönlü(Bi-Directional) Port Örneği-

inout read_enable; // read_enable portu çift yönlüdür.

Sinyal vektörlerini(bir bit'den fazla birleşik sinyal dizisi) nasıl tanımlarsınız? Verilog bunları tanımlamanın basit bir yolunu destekler.

Sinyal Vektörleri Örneği-
inout [7:0] address; // "address" portu çift yönlüdür

[7:0]'ın anlamı önemli bayt en solda olacak(little-endian) şekilde dönüştürülür- vektöre başlamak için en sağdaki bit 0'ı ifade edecek şekilde başlanır ve sola doğru gidilir. Eğer [0:7] şeklinde yaparsak bunun anlamı (big-endian)en önemli bayt en sağda olacak demektir. Sonlanma verinin hangi yönde okunacağını gösterir, fakat bu sistemlere göre farklılık gösterir, bu yüzden doğru sonlandırmayı kullanmak oldukça önemlidir. Analog olarak, bazı dilleri düşünün(İngilizce gibi) soldan sağa yazılır bazı dillerden(Arapça) farklı bunlar sağdan sola yazılırlar. Dilin hangi yönde aktığını bilmek onu okumak için önemlidir.

Özet

- Verilog'da bir blok/modülün nasıl tanımlandığını öğrendik.
- Port ve port yönlerinin nasıl tanımlandığını öğrendik.
- Vektör/sayısal portların nasıl tanımlandığını öğrendik.

Veri Tipleri

Veri tipleri donanımla ne yapmalıdırlar? Aslında hiç birşey. İnsanlar sadece içerisinde veri tipleri olan birden çok dil yazmak istemiştirler. Bu tamamen gereksizdir;

Fakat bekleyin… donanımın iki çeşit sürücüsü(driver) vardır.
(Sürücüler? Bunlar nelerdir?)

Bir sürücü, yüklemeyi süren bir veri tipidir. Temelde, fiziksel bir devrede, bir sürücü içerisinde elektronların gidip geldiği herhangi bir şey olabilir.

- Sürücü bir değer saklayabilir (örneğin: flip-flop).
- Ya da sürücüler bir değer saklayamaz, fakat iki noktayı birbirine bağlar (örneğin: tel).

İlk tipteki sürücüye Verilog'da kısaca reg("register"- yazmaç) denir. İkinci tip veri tipine ise tel("wire"-hımm buna da kısaca tel) denir. Bunu daha iyi anlamak için kolay lokma bölümleri tercih edebilirsiniz.

Bunların dışında birçok veri tipi vardır,örneğin, yazmaçlar işaretli(signed) yada işaretsiz(unsigned), kayan noktalı(floating point)… yeni biri olarak şimdilik bunlar için endişelenmeyin.

Örnekler :

wire and_gate_output; // "and_gate_output" sadece çıkış bir teldir(wire)

reg d_flip_flop_output; // "d_flip_flop_output" bir yazmaçtır(register); bir değer kaydeder ve bir değer üretir

reg [7:0] address_bus; // "address_bus" en sağdaki en yüksek 8bit yazmaçtır.

Özet

- Tel(Wire) veri tipi iki noktayı birleştirmek için kullanılır.
- Reg(yazmaç) veri tipi değer kaydetmek için kullanılır.
- Diğer tipler çok yaşasın. Bunları daha sonra göreceksiniz.

Operatörler(Operator)

Operatörler, diğer programlama dillerindeki gibidir. İki değer alırlar ve karşılaştırırlar(veya diğer bir şekilde bunlar üzerinde işlem yaparlar) bununla bir üçüncü sonuç elde ederler- en sık kullanılan örnek toplama, eşitlik, lojik-VE… Hayatı bizim için daha kolay hale getirmek için, neredeyse tüm operatörler eş programlama dili C ile aynıdır.

Operatör Tipi	Operatör Sembolü	Yaptığı İşlem
Aritmetik(Arithmetic)	*	Çarpım
	/	Bölme
	+	Toplama
	-	Çıkarma
	%	Modu
	+	Tekli artış(Unary plus)
	-	Tekli azalış(Unary minus)
Lojik(Logical)	!	Lojik tersi
	&&	Lojik VE
	||	Lojik VEYA
İlişkisel(Relational)	>	Büyüktür
	<	Küçüktür
	>=	Büyük eşit
	<=	Küçük eşit
Eşitlik(Equality)	==	Eşitlik-eşit mi
	!=	Eşitsizlik-eşit değil mi
İndirgeme(Reduction)	~	Bit tersi
	~&	VEDEĞİL
	|	VEYA
	~|	VEYADEĞİL
	^	DIŞLAMALIYADA(xor)
	^~	DIŞLAMALIYADA DEĞİL(xnor)
	~^	DIŞLAMALIYADA DEĞİL(xnor)
Kaydırma(Shift)	>>	Sağa kaydırma
	<<	Sola kaydırma
Bağlama(Concatenation)	{ }	Birbirine bağlama(Concatenation)
Koşul(Conditional)	?	Koşul(conditional)

Örnek -

- a = b + c ; // Bu çok kolaydı
- a = 1 << 5; // Bir düşünelim,hımm, buldum '1' i 5 kez sola kaydır.
- a = !b ; //peki b yi terse mi çeviriyor???
- a = ~b ; // Kaç kez 'a' 'ya atama yapacaksın, birçok sürücüye neden olabilir.

Özet-

- Hala C dilini denemeye devam ediyoruz, neredeyse C ile aynı.

Kontrol İfadeleri

Bekle bu da nesi? **if, else, repeat, while, for, case-** Verilog, C gibi görünüyor(büyük ihtimalle kullandığınız birçok programlama dili gibi)! Fonksiyonalite C diliyle aynıymış gibi görünse de Verilog bir HDL'dir, bu nedenle tanımlamalar donanıma dönüştürülmelidir. Bunun anlamı, kontrol ifadelerini kullanırken dikkatli olmalısınız(diğer türlü tasarımınız donanımda gerçeklenemez).

If-else

If-else ifadesi bir parça kodun yürütülüp yürütülmeyeceğinin koşulunu kontrol eder. Eğer bir koşul sağlanıyorsa, kod yürütülür. Değilse, kodun diğer kısmı koşulur.

```
 1 // begin ve end C/C++ 'daki süslü parantez '{' / '}' gibi davranır.
 2 if (enable == 1'b1) begin
 3   data = 10; // onluk atama
 4   address = 16'hDEAD; // onaltılık
 5   wr_enable = 1'b1; // ikili
 6 end else begin
 7   data = 32'b0;
 8   wr_enable = 1'b0;
 9   address = address + 1;
10 end
```

You could download file one_day2.v here

Herhangi biri koşul kontrolündeki herhangi bir operatörü kullanabilir, C dilindeki gibi. Eğer gerekliyse iç içe if else ifadeleri kullanılabilir; else olmadan ifadelerde sorun yoktur ancak bu onların kendi sorunudur kombinasyonel lojik modelleme yaparken bir tutucuda sonuçlanabilir (bu herzaman doğru değildir).

Case

Case ifadesi, eğer bir değişkenin birden çok değer için kontrol edilmeye ihtiyacı varsa kullanılmaktadır. Adres çözücü gibi, girişin bir adres olduğunda,alabileceği tüm değerler kontrol edilmelidir. Birçok iç içe if-else ifadesi yerine baktığımız her değer için tek bir case ifadesi kullanabiliriz: bu C++ daki switch ifadesiş gibidir.

Case ifadesi, korunmuş sözcük **case** ile başlar ve yine korunmuş sözcük **endcase** ile biter(Verilog kod bloklarını ayırmak için parantez kullanmaz). Durumlar bir ikinokta ile takip edilir ve yürütmek istediğiniz ifade, bu iki ayıraç arasında listelenir. Varsayılan(**default**) duruma sahip olmak güzel bir fikirdir. Ancak sonlu durum makinelerinde, eğer Verilog makinesi daha önce belirtilmemiş bir ifade girerse, makine asılı kalır. İfadeyi return ile varsayılan duruma getirmek bizi güvende tutar.

```
1 case(address)
2   0 : $display ("It is 11:40PM");
3   1 : $display ("I am feeling sleepy");
4   2 : $display  ("Let me skip this tutorial");
5   default : $display  ("Need to complete");
6 endcase
```

You could download file one_day3.v here

Bu gösteriyor ki adres değeri 3'tü ve bu benim hala bu eğitimi yazdığımı gösteriyor.

Not: if-else ve case ifadeleri için sık kullanılan bir şey, eğer tüm durumları kapsamıyorsa(If-else ifadesinde 'else' yoksa veya case ifadesinde 'default' yoksa), ve kombinasyonel bir ifade yazacaksan, sentez aracı tutucuyu(Latch) tercih eder.

 **While** normalde gerçek hayattaki modellemelerde kullanılmaz, fakat testbençlerde kullanılır. Diğer ifade bloklarında olduğu gibi begin ve end ile sınırlandırılırlar.

```
1 while (free_time) begin
2  $display ("Continue with webpage development");
3 end
```

You could download file one_day4.v here

"free_time" değişkeni kurulu olduğu sürece(1 olduğu sürece) begin ve end arasındaki kod gerçekleştirilecektir, burada ekrana sürekli **"Continue with webpage development"** yazılacaktır. Şimdi de farklı bir örneğe bakalım, bu birçok Verilog yapısında sıkça kullanılır. Evet doğru duydunuz. Verilog'un VHDL'den daha az kısıtlı kelimeleri vardır(anahtar sözcükleri) ve bu nedenle gerçek kodlamada daha azı kullanılır. Bu iyi olduğu kadar da doğrudur.

```
 1 module counter (clk,rst,enable,count);
 2 input clk, rst, enable;
 3 output [3:0] count;
 4 reg [3:0] count;
 5
 6 always @ (posedge clk or posedge rst)
 7 if (rst) begin
 8   count <= 0;
 9 end else begin : COUNT
10   while (enable) begin
11     count <= count + 1;
12     disable COUNT;
13   end
14 end
15
16 endmodule
```

You could download file one_day5.v here

Örnekte Verilog'un birçok yapısı kullanılmaktadır. Yeni bir blokla karşılaşabilirsiniz "always"- bu Verilog'un başka bir anahtar özelliğini örneklemektedir. Daha önce belirttiğimiz birçok yazılım dilleri, sıralı olarak gerçeklenir-buna ifade ifade denir. Diğer taraftan Verilogda birçok ifade paralel olarak gerçeklenir. Tüm bloklar belirtilen bir veya birden fazla koşul yerine getirildiği sürece herzaman -eşzamanlı olarak- çalışacak şekilde işaretlenmiştir.

Örnekte belirtildiği üzere, **always** bloğu, rst veya clk bir pozitif kenara ulaştığında,yani değerleri 0'dan 1'e geldiğinde koşulur. Bir programda iki veya daha fazla **always** bloğu aynı anda olabilir(burada gösterilmese de sıkça kullanılır).

Bir blok kodu ayırtılmış kelimeyi kullanarak ulaşılamaz hale getirebiliriz. Yukarıdaki örnekte, herbir counter(sayıcı) artışından sonra, COUNT kod bloğu ulaşılamaz hale geitirilir(bu örnekte gösterilmemiştir).

For döngüsü

Verilog'daki for döngüsü neredeyse C ve C++'daki gibidir. Tek fark Verilog ++ ve – operatörlerinin desteklemez. C'deki i++ yazmak yerine, bunun tam işlemsel karşılığını i=i+1 yazmanız gerekmektedir.

```
1      for (i = 0; i < 16; i = i +1) begin
2              $display ("Current value of i is %d", i);
3      end
```

You could download file one_day6.v here

Bu kodda 0'dan 15'e sayılar sırayla yazılırlar. RTL'de for döngüsü kullanırken dikkatli olun ve kodunuzun donanımdaki gibi gerçekleştirilebildiğine emin olun…ve döngünüz sonsuz döngü olmamalı.

Repeat (Tekrar)

Repeat daha önce belirttiğimiz for döngüsüne benzerdir. Bunun dışında dışarıdan bir değişken belirtilir ve arttırılır, döngüyü bildirdiğimizde kaç kez kodun çalışacağını bildiririz, ve değişken arttırılmaz(örnekteki gibi pek kullanılmaz ama)

```
1 repeat (16) begin
2   $display ("Current value of i is %d", i);
3   i = i + 1;
4 end
```

Çıktı bir önceki örnekteki gibidir.For döngüsüne benzer şekilde gerçek donanım gerçeklemesinde repeat kullanımı çok nadirdir.

Summary

- While, if-else, case(switch) ifaedeleri C dilindeki gibidir.
- If-else ve case ifadeleri kombinasyonel lojik için tüm durumların kapsanmasına ihtiyaç duyar.
- For-döngüsü C'deki gibidir ancak ++ ve – operatörleri yoktur.
- Repeat for-döngüsüyle aynıdır fakat artış değişkenleri yoktur.

Değişken Ataması

Dijital olarak iki tip eleman vardır, kombinasyonel ve sıralı. Elbette bunları biliyoruz. Fakat soru şu ki "Bunu Verilog'da nasıl modelleriz?". Güzel Verilog, iki yönlü kombinasyonel lojik ve tek yönlü sıralı lojik modeli destekler.

- Kombinasyonel elemanlar atama ve herzaman ifadeleriyle modellenebilirler.
- Sıralı elemanlar sadece herzaman ifadeleriyle modellenebilirler.
- Üçüncü bir blok da sadece testbenç için kullanılır: buna başlangıç değeri ifadesi denir.

Başlangıç Bloğu(Initial Block)

Bir başlangıç bloğu, adından da anlaşıldığı üzere sadece simülasyon başladığında gerçekleştirilir. Eğer birden fazla başlangıç bloğumuz varsa, simülasyonun başında hepsi birden gerçekleştirilir.

Örnek

```
1 initial begin
2             clk = 0;
3             reset = 0;
4             req_0 = 0;
5             req_1 = 0;
6 end
```

You could download file one_day8.v here

Verilen örnekte, simülasyonun başında,(zaman=0 iken) begin ve end bloğunun arasındaki tüm değişkenler sıfır olarak sürülmektedir.

Herzaman(Always) Bloğu

İsminden de anlaşıldığı üzere, bir always(herzaman) bloğu herzaman gerçekleştirilir, başlangıç bloğundaki gibi sadece bir kere(simülasyonun başında) değil. Diğer bir fark ise bir herzaman(always) bloğunun bir hassasiyet listesi olabilir veya onunla ilgili bir gecikme(delay) olabilir.

Hassasiyet listesi(sensitive list), always bloğuna kodun ne zaman gerçekleştirileceğini söyler, aşağıdaki örnekte gösterildiği gibi. ' always' korunmuş sözcüğünden sonra @sembolü bloğun ne zaman tetikleneceğini parantez içinde belirtir.

Always(herzaman) bloğu hakkında önemli bir not: tel(wire) veri tipinde kullanılamaz, fakat reg(yazmaç) ve integer(tamsayı) veri tiplerinde kullanılabilir.

```
1 always  @ (a or b or sel)
2 begin
3   y = 0;
4   if (sel == 0) begin
5     y = a;
6   end else begin
7     y = b;
8   end
9 end
```

Belirtilen örnek 2:1 ‘lik bir çoklayıcı(mux-multiplexer) örneğidir,giriş değerleri “a” ve “b” ile; “sel” girişi seçmek için ve “y” de çıkış için kullanılmıştır. Herhangi bir kombinasyonel lojikte, çıkış giriş değiştiğinde değişir. Bu teori always bloğuna uygulandığında bunun anlamı always bloğunun içindeki kod giriş değişkenleri(veya çıkış kontrol değişkenleri) değiştiğinde gerçekleştirilir. Bu değişkenler hassasiyet listesine eklenmiştir ve a,b ve sel olarak isimlendirilmiştir.

İki tip hassasiyet listesi vardır: seviye hassas (kombinasyonel devreler için) ve kenar hassas (flip-floplar için). Aşağıdaki kodda 2:1 Çoklayıcıdır fakat y çıkışı artık bir flip flop çıkışıdır.

```
1 always  @ (posedge clk )
2 if (reset == 0) begin
3   y <= 0;
4 end else if (sel == 0) begin
5   y <= a;
6 end else begin
7   y <= b;
8 end
```

You could download file one_day10.v here

Normalde flip flopları sıfırlamak(reset) zorundayız, bu nedenle saat 0’dan 1’e geçtiğinde(posedge-positive edge-pozitif kenarda), sıfırlama gerçekleşmiş mi(senkron sıfırlama-reset), sonra normal şekilde devam ederiz. Eğer yakından incelersek kombinasyonel lojikte bizim atama için “=” operatörümüz vardır, ancak sıralı blokda ise "<=" operatörümüz vardı. Güzel, "=" bloklama ataması ve "<=" bloklamama ataması. "=" kodu begin/end arasında sıralı olarak gerçekleştirirken, "<=" paralel olarak gerçekleştirmektedir.

Bir herzaman bloğuna(always) hassasiyet listesi olmadan da sahip olabiliriz, bu durumda aşağıdaki kodda görüldüğü gibi bir gecikmeye(delay) ihtiyacımız vardır.

```
1 always  begin
2    #5  clk = ~clk;
3 end
```

İfadenin önündeki #5, ifadenin gerçekleştirilmesini 5 zaman birimi kadar geciktirmektedir.

Atama(assign) İfadesi

Bir atama ifadesi sadece kombinasyonel lojiği modeller ve devamlı gerçekleştirilir. Bu nedenle atama ifadesi sürekli atama ifadesi('continuous assignment statement') olarak adlandırılır ve hassasiyet listesi yoktur.

```
1 assign out = (enable) ? data : 1'bz;
```

You could download file one_day12.v here

Verilen örnek bir üç-durumlu arabelleği(tri-state buffer) örneğidir. “enable” 1 olduğunda, veri çıkışa sürülmektedir değilse çıkış yüksek empedansa çekilmektedir. Çoklayıcı(mux), şifreleyici(encoder),şifre çözücü(decoder) oluşturmak için içiçe koşullara ihtiyacımız vardır.

```
1 assign out = data;
```

Bu örnek basit bir arabellek örneğidir.

Görev(Task) ve Fonksiyon(Function)

Eğer daha önce kullandığımız şeyleri tekrar tekrar kullanıyorsak, Verilog diğer programlama dillerindeki gibi adres tekrarlı kullanılmış kod yani görev(task) ve fonksiyonları(function) destekler.

Aşağıdaki kod çift pariteyi(even parity) hesaplamak için kullanılmıştır.

```
 1 function parity;
 2 input [31:0] data;
 3 integer i;
 4 begin
 5   parity = 0;
 6   for (i= 0; i < 32; i = i + 1) begin
 7     parity = parity ^ data[i];
 8   end
 9 end
10 endfunction
```

Fonksiyonlar ve görevler aynı söz dizime sahiptir; tek fark görevlerin(task) gecikmeleri(delay) olabilir ancak fonksiyonların gecikmesi olamaz. Bunun anlamı fonksiyonlar kombinasyonel lojiği modellemek için kullanılır.

İkinci bir fark ise fonksiyonlar bir değer döndürebilir ancak görevler döndüremez.

Test Benç(Test Benche)

Tamam tasarım dokümanına göre kodu daha önce yakzık, şimdi ne yapacağız peki?

Güzel şimdi kodun belirtimlere uygun çalışıp çalışmadığını test edeceğiz. Genelde bu bizim dijital labında geçirdiğimiz zamanlara benzer şekilde gerçekleşmektedir: girişler sürülür, çıkışların beklenen değerlerle uyuşup uyuşmadığı kontrol edilir. Şimdi arabulucu testbence bakalım.

```
 1 module arbiter (
 2 clock,
 3 reset,
 4 req_0,
 5 req_1,
 6 gnt_0,
 7 gnt_1
 8 );
 9
10 input clock, reset, req_0, req_1;
11 output gnt_0, gnt_1;
12
13 reg gnt_0, gnt_1;
14
15 always @ (posedge clock or posedge reset)
```

```
16 if (reset) begin
17  gnt_0 <= 0;
18  gnt_1 <= 0;
19 end else if (req_0) begin
20   gnt_0 <= 1;
21   gnt_1 <= 0;
22 end else if (req_1) begin
23   gnt_0 <= 0;
24   gnt_1 <= 1;
25 end
26
27 endmodule
28 // Testbench Code Goes here
29 module arbiter_tb;
30
31 reg clock, reset, req0,req1;
32 wire gnt0,gnt1;
33
34 initial begin
35   $monitor ("req0=%b,req1=%b,gnt0=%b,gnt1=%b", req0,req1,gnt0,gnt1);
36   clock = 0;
37   reset = 0;
38   req0 = 0;
39   req1 = 0;
40    #5  reset = 1;
41    #15  reset = 0;
42    #10  req0 = 1;
43    #10  req0 = 0;
44    #10  req1 = 1;
45    #10  req1 = 0;
46    #10  {req0,req1} = 2'b11;
47    #10  {req0,req1} = 2'b00;
48    #10  $finish;
49 end
50
51 always begin
52   #5  clock =  ! clock;
53 end
54
55 arbiter U0 (
56 .clock (clock),
57 .reset (reset),
58 .req_0 (req0),
59 .req_1 (req1),
60 .gnt_0 (gnt0),
61 .gnt_1 (gnt1)
62 );
63
64 endmodule
```

You could download file arbiter.v here

Daha önce bildirdiğimiz tüm arabulucu girişleri yazmaç olarak ve çıkışlar da tel olarak görünüyor, güzel, bu doğru. Bu testbenç'i yapmamızın nedeni sürülen giriş değerleri ile bunlara karşılık gelen çıkış değerlerinin gözlemlenmesidir.

Tüm gerekli değişkenleri bildirdikten sonra, bilinen durumlar için girişlere başlangıç değeri verilir: bunu initial block içerisinde yaparız. Başlangıç değeri verildikten sonra, sırayla arabulucuda test edeceğimiz reset, req0, req1 bildirilir. Saat bir herzaman bloğuyla üretilir.

Testi bitirdikten sonra, simülatörü durdurmamız gerekmektedir. Bu nedenle “$finish” simülaysyonu sonlandırmak için kullanılır. “$monitor” sinyal listesindeki değişiklikleri gözlemlemek ve istediğimiz formatta yazdırmak için kullanılır.

```
req0=0,req1=0,gnt0=x,gnt1=x
req0=0,req1=0,gnt0=0,gnt1=0
req0=1,req1=0,gnt0=0,gnt1=0
req0=1,req1=0,gnt0=1,gnt1=0
req0=0,req1=0,gnt0=1,gnt1=0
req0=0,req1=1,gnt0=1,gnt1=0
req0=0,req1=1,gnt0=0,gnt1=1
req0=0,req1=0,gnt0=0,gnt1=1
req0=1,req1=1,gnt0=0,gnt1=1
req0=1,req1=1,gnt0=1,gnt1=0
req0=0,req1=0,gnt0=1,gnt1=0
```

Ben bu çıkışı elde etmek için Icarus Verilog simülatörünü kullandım.

Verilog'un Tarihçesi

Verilog'un Tarihçesi

Verilog ilk olarak 1984'lerde Gateway Design Automation Şirketi tarafından donanım modelleme dili olarak başlatılmıştır. Söylentilere göre orijinal dil özellikleri zamanın en popüler HDL dilinden ve geleneksel bilgisayar dillerinden C 'den almıştır. Bu esnada Verilog henüz standartlaştırılmamıştı ve dil kendisini 1984'den 1990'a kadar olan aralıkta geliştirmiştir.

Verilog simülatörü 1985'lerin başında ilk olarak kullanılmaya başlanmıştır ve 1987'nin sonlarında büyük ölçüde gelişmiştir. Verilog simülatörünün gerçekleştirilmesi Gateway tarafından satılmaktaydı. En büyük ek gelişme Verilog-XL idi, bu bir dizi özellik ve adı kötüye çıkmış XL algoritmasını kapı seviyesi simülasyon için en etkili biçime getircek şekilde gerçekleştirmiştir.

Tarih 1990'ların sonlarını gösteriyordu. Cadence Tasarım Sistemi, bunların ilk üretimi İnce film süreç simülatörü eklendi, Cadence artık Verilog dilinin sahibi oldu, ve Verilog'u hem bir dil olarak hemde bir simülatör olarak pazarlamaya devam etti. Aynı zamanda Synopsys en-yüksek tasarım metodolojisini Verilog kullanarak pazarlıyordu. Bu çok etkinb bir kombinasyondu.

1990'da, Cadence eğer Verilog kapalı bir dil olarak kalırsa sanayinin VHDL'i tercih etmesine neden olacaktı. Bunun üzerine Cadence 1991'de Open Verilog International (OVI) 'yi hazırladı ve Verilog Donanım Tanımlama dili olarak belgeledi. Bu dilin herkese açıldığı olaydır.

OVI gerçektende Dil Başvuru Kitabı(Language Reference Manual (LRM)) için gözardı edilemeyecek büyüklükte bir işi başardı, herşeye açıklık getirdi ve dilin belirtimlerini pazarlayıcısından olabildiğinde bağımsız hale getirdi.

Yakın zamanda Verilog için pazarda birçok şirketin olduğu ortaya çıktı , potansiyel olarak herkes Gateway'in şimdiye kadar yaptığı yapmak istiyordu ve kendi ihtiyaçlarına göre dili değiştiriyordu. Bu herkese yaymanın temel amacını aşmıştı. Bunun sonucu olarak 1994'te IEEE 1364 çalışma grubu OVI LRM'u bir IEEE standardına dönüştürmeye başladılar. Bu çabanın sonucu olarak başarılı bir beğeni ile 1995'de sonuçlandı, ve Verilog 1995 Aralığında bir IEEE standardı haline geldi.

Cadence OVI(LRM)'yi verdiğinde, birçok firme Verilog simülatörünü kullanmaya başlamıştı. 1992'de ilk simülatör duyurulmuştu ve 1993'te Cadence dışındaki birçok firmanın hazırladığı Verilog simülatörleri bulunmaktaydı. Bunların en başarılısı VCS(Verilog Compiled Simülatör- Verilog Derlenmiş Simülatör) Chronologic Simulation tarafınadan hazırlanmıştı. Bu bir yorumlayıcının aksine gerçek bir derleyiciydi, bu Verilog-XL ne olduğuydu. Sonuç olarak derleme zamanı çok tatmin ediciydi fakat simülasyon gerçekleme hızı daha hızlıydı.

Bu arada Verilog ve PLI'ın popülaritesi üssel olarak artmaktaydı. Verilog bir HDL olarak iyi biçimlendirilmiş ve resmi kurumlarca yatırım yapılan VHDL'den daha çok takdir edilirdi. OVI ortaya çıkmadan önce daha genel kabul görmüş standartlara ihtiyaç duyulmuştu. OVI'nin yönetim kurulu IEEE çalışma komitesinden Verilog'u bir IEEE standardına getirmelerini istemiştir Çalışma komitesi 1364, 1993'ün ortalarında oluşturulmuş ve 14 Ekim 1993'de ilk toplantısını gerçekleştirmiştir.

Standart Verilog sözdizimi ile PLI'ı tek bir alanda toplamıştır ve Mayıs 1995'de IEEE 1364-1995 standardını oluşturmuştur.

Aradan geçen zamanda, Verilog'a birçok özellik eklenmiştir, ve yeni versiyon Verilog 2001 olarak isimlendirilmiştir. Bu versiyon Verilog 1995'in sahip olduğu birçok hatayı düzeltmiş görünmektedir. Bu versiyona 1364-2001 adı verilmiştir.

Tasarım ve Araç Akışı

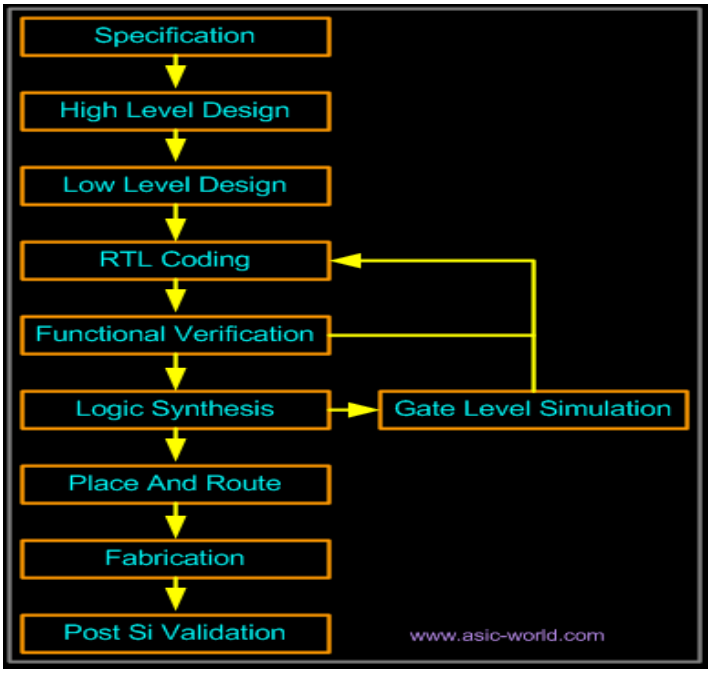
Giriş

Verilog'a yeni biri olarak bazı örnekleri denemek ve yeni birşeyleri tasarlamak isteyebilirsiniz. Bu amacınızı gerçekleştirmek için gerekli araç akışını listeledim. Bunu şahsen kendim denedim çalışıyor. Burada sadece baştan sonra tasarım bölümünü ve araç akışının FPGA tasarımının bitlerini aldım, bunu araçlar için çok fazla para harcamadan yaptım.

ASIC/FPGA'in çeşitli adımları

- **Belirtim(Specification) :** Kelime işlemci;Word, Kwriter, AbiWord, Open Office gibi.
- **Yüksek Seviyeli Tasarım(High Level Design):**Word, Kwriter, AbiWord gibi kelime işlemci, dalga formunu çizmek için waveformer veya testbencher veya Word, Open gibi araçlar kullanır.
- **Mikrı Tasarım/Düşük seviyeli tasarım(Micro Design/Low level design):** Word, Kwriter, AbiWord gibi kelime işlemci,dalga formunu çizmek için waveformer veya testbencher veya Word gibi araçlar kullanılır.
- **RTL kodlama(RTL Coding):** Vim, Emacs, conTEXT, HDL TurboWriter
- **Simülasyon(Simulation):** Modelsim, VCS, Verilog-XL, Veriwell, Finsim, Icarus.
- **Sentez(Synthesis):** Design Compiler, FPGA Compiler, Synplify, Leonardo Spectrum. Bunları Altera ve Xilinx gibi FPGA satıcılarından bedava indirebilirsiniz.
- **Alan& Rota(Place & Route):** FPGA için FPGA satıcılarından P&R araçları. ASIC araçları ise Apollo gibi pahalı P&R araçlarına ihtiyaç duyar. Öğrenciler LASI,Magic kullanabilirler.
- **Silikon Sonrası Onaylama(Post Si Validation):** ASIC ve FPGA için, çipin gerçek bir ortamda test edilmesi gerekmektedir. Çevrim kartı(board) tasarımı, araç sürücüsünün yerleştirilmiş olması gerekmektedir.

Figure : Typical Design flow

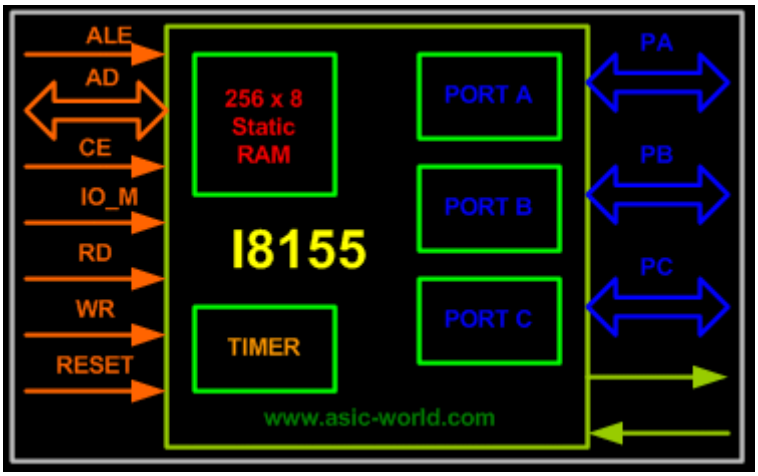


Belirtim (Specification)

Bu sizin tasarlamayı düşündüğünüz sistem/tasarımın önemli parametrelerinin tanımlandığı adımdır. Basit bir örnek bir sayıcı tasarımı olabilir; 4bit genişliğinde, senkron yeniden başlatmalı(reset), aktif yüksek etkinlenebilir; eğer yeniden başlatma(reset) aktifse sayıcının çıkışı "0" 'a gitmeli.

Yüksek Seviyeli Tasarım(High Level Design)

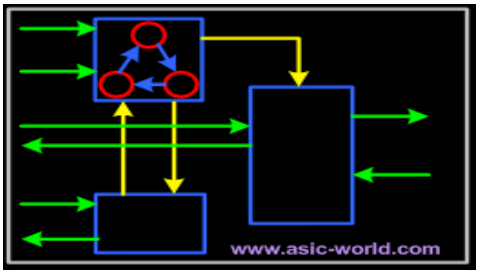
Bu bölümde birçok bloğu ve bunlar arasındaki haberleşme tanımlanmaktadır. Bir mikroişlemci tasarlayacağımızı varsayalım: yüksek seviyeli tasarımın anlamı tasarımı blokların fonksiyonlarına göre ayırmak; bizim durumumuzda bloklar yazmaçlar, ALU(aritmetik mantık birimi), talimat çözme(Instruction Decode), bellek arayüzü(Memory Interface),vb.



Şekil : I8155 Yüksek Seviyeli Blok Diyagram

Mikro Tasarım/Düşük Seviyeli Tasarım(Micro Design/Low level design)

Düşük seviyeli tasarım veya mikro tasarım adımında tasarımcının herbir bloğu uygulandığında nasıl tanımlayacağını belirtir. Durum makinesi, sayıcılar, çoklayıcılar, şifre çözücüler, iç yazmaçlar hakkında ayrıntı içerir. Herzaman, farklı arayüzlerde dalgaformu çizmek iyi bir fikirdir. Bu adım bizim çok zaman harcadığımız bölümlerden biridir.



Şekil : Örnek Düşük seviye tasarım

RTL Kodlama(RTL Coding)

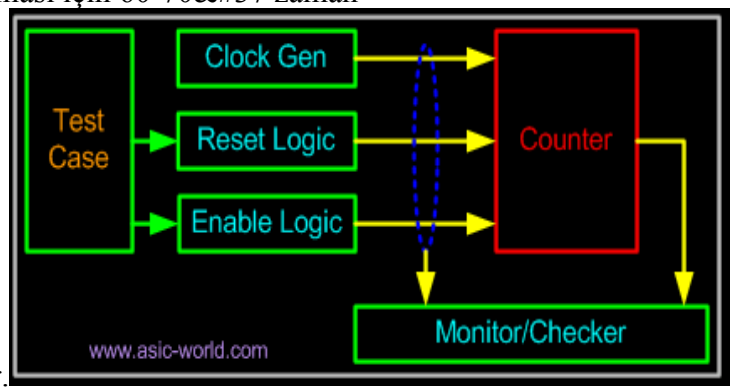
RTL kodlamada, Mikro tasarım dilin sentezlenebilir yapıları kullanılarak Verilog/VHDL koda dönüştürülür. Normalde doğrulama ve sentezlemeye geçmeden önce kodun yaralarını sarmayı severiz.

```
 1 module addbit (
 2 a      , // ilk giriş
 3 b      , // ikinci giriş
 4 ci     , // elde girişi(Carry input)
 5 sum    , // toplama çıkışı
 6 co       // elde çıkışı
 7 );
 8 //Giriş bildirimi
 9 input a;
10 input b;
11 input ci;
12 //Çıkış bildirimi
13 output sum;
14 output co;
15 //Port veri tipleri
16 wire  a;
17 wire  b;
18 wire  ci;
19 wire  sum;
20 wire  co;
21 //Kod buradan itibaran başlamaktadır
22 assign {co,sum} = a + b + ci;
23
24 endmodule // addbit modülünün sonu
```

You could download file addbit.v here

Simülasyon(**Simulation)**

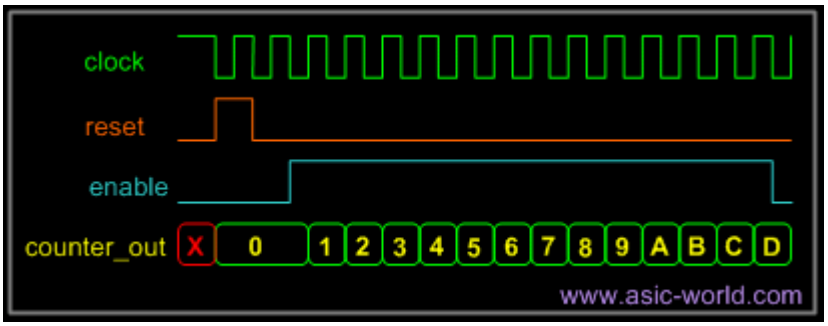
Simülasyon, herhangi bir seviyedeki soyutlamanın fonksiyonel karakteristiğini doğrulama sürecine denir. Simülatörleri Donanım modellerini simüle edebilmek için kullanıyoruz. RTL kodun belirtilen fonksiyonel gereklilikleri karşılayıp karşılamadığını test etmek için tüm RTL bloklarının fonksiyonel olarak doğruluğunu kontrol etmeliyiz. Bunu yapmak için bir clk(saat), reset(yeniden başlatma) ve gerekli test vektörlerini üretecek bir testbenç yazmamız gerekmektedir .Bir sayıcı için örnek bir testbenç aşağıda gösterilmiştir. Normalde tasarım doğrulaması için 60-70&#37 zaman harcanır.



Şekil : Örnek Testbenç Ortamı

Test altındaki aracın (DUT (Device Under Test)) fonksiyonel olarak doğru çalışıp çalışmadığını anlamak için simülatörden çıkan dalgaformu kullanırız. Birçok simülatör bir dalgaformu gösterici ile birlikte gelir. Tasarım karmaşıklaştıkça biz kendi kendini kontrol eden testbençler yazdık, nerede testbenç test vektörünü uygular,sonra da DUT'ın çıkışı beklenen değerleri karşılaştırır.

Burada başka türlü bir simülasyon daha vardır, buna **zamanlı simülasyon(timing simulation)** denir, sentez veya Alan ve Rota(P&R-Place and Route)'dan sonra yapılır. Burada kapı gecikmelerini ve tel gecikmelerini ekledik ve test altındaki aracın beklenen saat hızında çalışıp çalışmadığını inceledik. Buna aynı zamanda **SDF simülasyon** veya **kapı seviyesi simülasyon** denir.

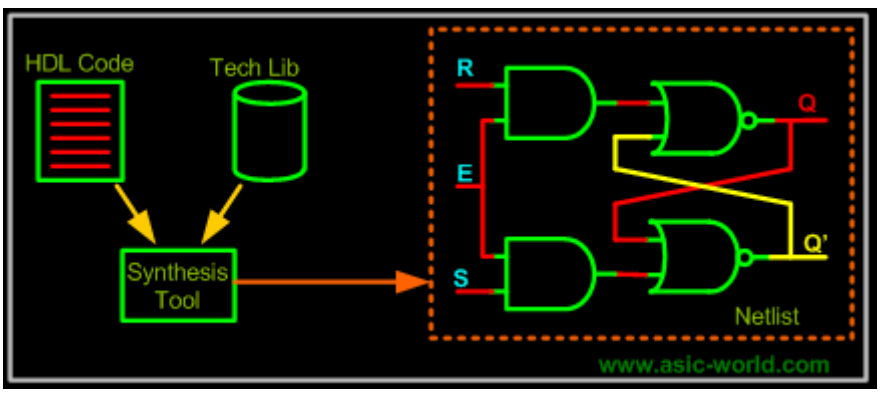


Şekil : 4 bitlik Yukarı Sayıcı Dalgaformu

Sentez(Synthesis)

Sentez, tasarım derleyici veya Synplify gibi sentez araçlarının Verilog veya VHDL 'deki RTL'den aldıkları, hedef teknolojiyi , girişlere göre kısıtlamak ve RTL'i hedef teknolojinin kısıtlarıyla eşleştirmektir. Sentez araçları, RTL'i kapılarla eşleştirdikten sonra eşleştirilimiş tasarımın zamanlama ihtiyacına uyup uymadığını en düşük miktarda zamanlama analizi ile yapmalıdır. (Dikkat edilmesi gereken önemli şey ise sentez araçları tel gecikmesiyle ilgilenmez sadece kapı gecikmelerini önemser). Sentezden sonra netlistde son şeye (Alan ve Rota) geçmeden önce elimizde normal olarak yapılmış bir dizi şey olmalıdır.

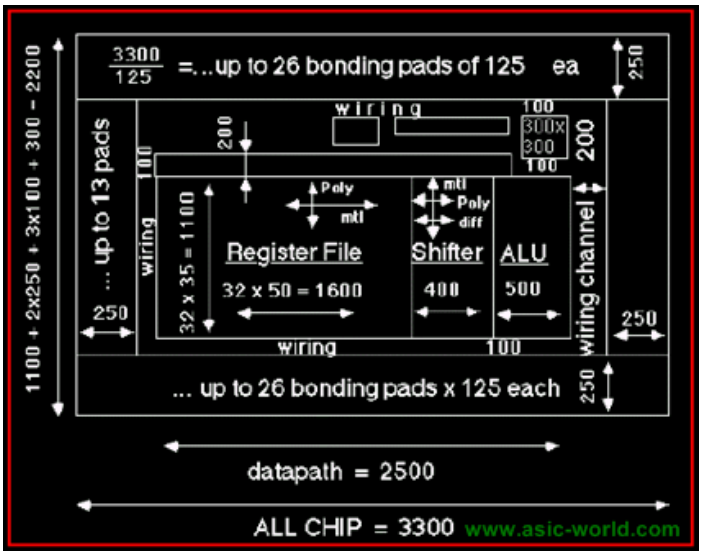
- **Biçimsel Doğrulama(Formal Verification) :** RTL ile kapı eşleşmesi doğrumu kontrol et.
- **Tarama ekleme(Scan insertion) :** ASIC durumunda tarama zincirini ekle.



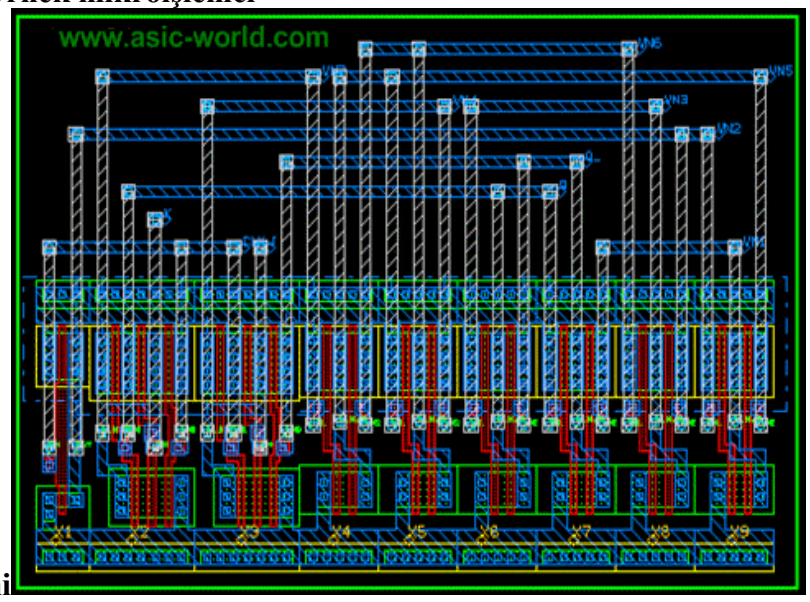
Şekil : Sentez Akışı

Alan & Rota (Place & Route)

Sentez araçlarından gelen kapıseviyesi netlis alınır ve Alan ve Rota(place and route) aracına Verilog netlist biçimine uygun şekilde içe aktarılır. Tüm kapılar ve flip-floplar yerleştirilir; saat ağacı sentezi(clock tree synthesis) ve yeniden başlatma(reset) yönlendirilir. Bundan sonra herbir blok yönlendirilir. P&R araçlarının çıktısı bir GDS dosyasıdır, ASIC üretmek için dökümhane tarafından kullanılır. Arka uç takımı normalde SPEF (standart parazit değiştirme formatı-standard parasitic exchange format) /RSPF (azaltılmış parazit değiştirme formatı-reduced parasitic exchange format)/DSPF (detaylı parazit değiştirme formatı-detailed parasitic exchange format)'i ASTRO gibi yerleşim araçları ile temizleyip ön uç takımına atar, bunlarda, Prime Time gibi araçlardan okunmuş_prazit(read_parasitic) komutları kullanarak SDF(standar gecikme biçimi-standard delay format)'i kapı seviyesi simülasyonu için tam olarak hazır hale getirir.



Şekil : Örnek mikroişlemci yerleşimi



Şekil : J-K Flip-Flop

Silikon Sonrası Onaylama(Post Silicon Validation)

Chip(silikon) bir kere fabrikaya geldiğinde, gerçek ortalama koyulmasına ve pazara çıkmadan önce test edilmesi gerekmektedir. RTL ile gelen simülasyon hızı(saniyedeki saat darbe sayısı) çok yavaştır, bu nedenle herzaman silikon sonrası onaylamada bir hata bulunma olasılığı çok yüksektir.

Benim İlk Verilog Programım

Giriş

Eğer herhangi bir programlama kitabına bakacak olursanız herzaman bir "Merhaba Dünya" programıyla başladığını göreceksiniz; bunu yazdığınızda bu dilde birşeyler yapıpı yapamayacağınızdan emin olabilirsiniz.

Bu nedenle şimdi size bir merhaba dünya programının nasıl yazılacağını Verilogda bir sayıcı tasarımıyla göstereceğim.

Merhaba Dünya Programı

```
 1 //------------------------------------------------------
 2 // Bu benim ilk Verilog programım
 3 // Tasarım ismi     : hello_world(merhaba dünya)
 4 // Dosya ismi       : hello_world.v
 5 // Fonksiyon        : Bu program ekrana 'hello world' yazdıracak
 6 // Kodlayan         : Deepak
 7 //------------------------------------------------------
 8 module hello_world ;
 9
10 initial begin
11   $display ("Hello World by Deepak");
12    #10  $finish;
13 end
14
15 endmodule // hello_world modülünün sonu
```

You could download file hello_world.v here

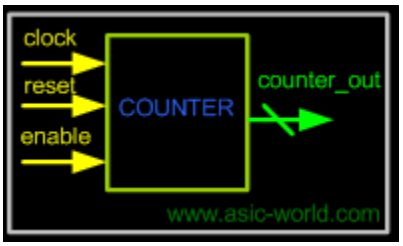
Yeşil renkteki sözcükler açıklamalar(comment), mavi renktekiler korunmuş sözcüklerdir. Verilog'da herhangi bir program korunmuş bir sözcük '**module**'<modül_ismi> ile başlar. Yukarıdaki örnekte satır 8 hello_word isimli modülü içerir.(NOT: Bazı derleyiciler modül bildiriminden önce `include', `define' gibi ön işlemci ifadeleri kullanabiliyor.)

Satır 10 başlangıç(initial) bloğunu içerir: bu blok simülasyon başlatıldıktın hemen sonra, zaman=0(0ns), yalnızca bir kez yürütülür. Satır 10 da 'begin' ile başlayıp başlayıp satır 13 de 'end' ile sonlanan bu blok iki ifade içerir. Verilogda bir blokda eğer birden fazla satır varsa begin ve end kullanmanız gerekmektedir. Modül satır 15 deki 'endmodule' korunmuş sözcüğü ile sonlandırılmıştır.

Merhaba Dünya Programı Çıkışı

```
Hello World by Deepak
```

Sayaç Tasarım Bloğu



Sayaç Tasarım Berlirtimi

- 4-bit senkron yukarı sayaç.
- Üst seviyede aktif, senkron sıfırlama(active high, synchronous reset).
- Üst seviyede aktif etkin(active high enable).

Sayaç Tasarımı

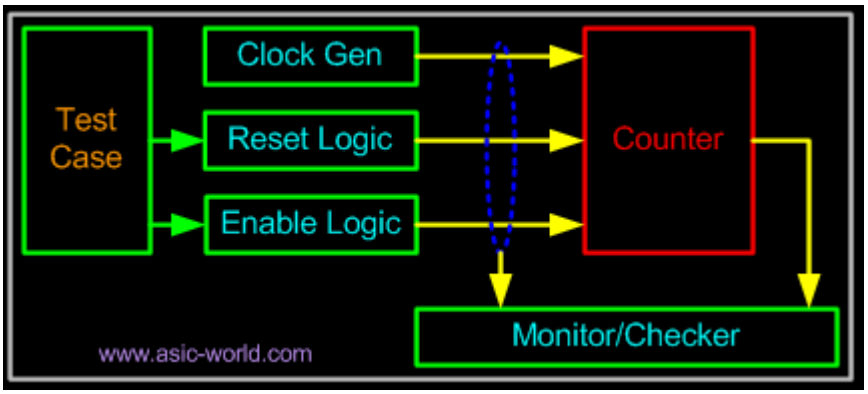
```
 1 //-------------------------------------------------
 2 // Bu benim ikinci Verilog tasarımım
 3 // Tasarım İsmi    : first_counter
 4 // Dosya İsmi      : first_counter.v
 5 // Fonksiyonu      : Senkron üst seviyede
 6 // aktif sıfırlama ve
 7 // etkinleştirmeli 4 bit yukarı sayaç
 8 //-------------------------------------------------
 9 module first_counter (
10 clock , // Tasarımın saat girişi
11 reset , // aktif yüksek senkron Reset girişi
12 enable , // Sayaç için yukarı seviyede aktif etkinleştirme
13 counter_out // sayacın 4 bit vektör çıkışı
14 ); // port listesinin sonu
15 //------------Giriş Portları------------------------------
16 input clock ;
17 input reset ;
18 input enable ;
19 //------------Çıkış Portları------------------------------
20 output [3:0] counter_out ;
21 //------------Giriş Portları Veri Tipleri-----------------
22 // Kurala göre tüm giriş portları tel(wire) olmalı
23 wire clock ;
24 wire reset ;
25 wire enable ;
26 //------------Çıkış Portları Veri Tipleri-----------------
27 // Çıkış portları bellek elemanı(reg-yazmaç) veya bir tel olabilir
28 reg [3:0] counter_out ;
29
30 //------------Kod Burada Başlamaktadır--------------------
31 // Bu sayaç yükselen kenar tetiklemeli olduğundan,
32 // Bu bloğu saatin
33 // artan kenara göre tetikleyeceğiz.
34 always @ (posedge clock)
35 begin : COUNTER // Block Name
```

```
 36   // Saatin herbir yükselen kenarında sıfırlama(reset) aktifmi diye
kontrol edeceğiz
 37   // Eğer aktifse sayacın çıkışına 4'b0000 yükleyeceğiz
 38   if (reset == 1'b1) begin
 39     counter_out <=  #1  4'b0000;
 40   end
 41   // Eğer etkin aktifse, sayacı arttıracağız
 42   else if (enable == 1'b1) begin
 43     counter_out <=  #1  counter_out + 1;
 44   end
 45 end // COUNTER bloğunun sonu
 46
 47 endmodule // counter modülünün sonu
```

You could download file first_counter.v here

Sayaç Test Benç

Herhangi bir sayısal(dijital) devre, ne kadar karmaşık olursa olsun test edilmesi gerekmektedir. Sayaç mantığı için bizim saat(clock) ve sıfırlama(reset) lojiğini sağlamamız gerekmektedir. Eğer sayaç sıfırlama(reset) durumu dışında ise, sayacın girişini aktif hale getiririz, ve dalgaformuna bakarız böylece doğru sayıp saymadığını anlarız. Bu Verilogda yapılmıştır.



Sayaç testbenci saat üretici(clock generator), sıfırlama(reset) kontrolü ve monitör/denetçi (monitor/checker) lojiğini içerir. Aşağıdaki basit kod monitör/denetçi lojiği olmayan testbençtir.

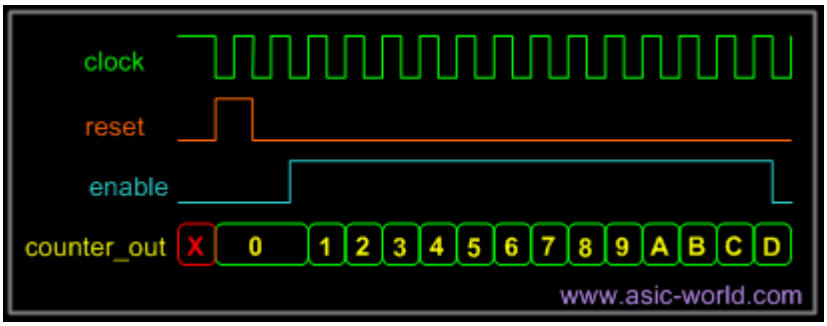
```
  1 `include "first_counter.v"
  2 module first_counter_tb();
  3 // Girişlerin yazmaçlar(regs) ve çıkışların tel(wire) olarak
bildirilmesi
  4 reg clock, reset, enable;
  5 wire [3:0] counter_out;
  6
  7 // Tüm değişkenlerin başlangıç değerleri
  8 initial begin
  9   $display ("time\t clk reset enable counter");
 10   $monitor ("%g\t %b   %b     %b      %b",
 11       $time, clock, reset, enable, counter_out);
 12   clock = 1;       // saat(clock) ilk değeri
 13   reset = 0;       // sıfırlama(reset) ilk değeri
 14   enable = 0;      // etkin(enable)ilk değeri
 15    #5  reset = 1;    // reset belirtimi
```

```
16    #10  reset = 0;   // reset' den çıkış durumu
17    #10  enable = 1;  // enable durumu
18    #100  enable = 0; // enable'dan çıkış durumu
19    #5  $finish;      // Simülasyonu sonlandır
20 end
21
22 // Saat üretici(Clock generator)
23 always begin
24    #5  clock = ~clock; // Her 5 saat darbesinde durum değişimi
25 end
26
27 // Testbenç'den DUT'a bağlantı
28 first_counter U_counter (
29 clock,
30 reset,
31 enable,
32 counter_out
33 );
34
35 endmodule
```

You could download file first_counter_tb.v here

Zaman	Saat	Sıfırlam	Etkin	Sayaç
time	clk	reset	enable	counter
0	1	0	0	xxxx
5	0	1	0	xxxx
10	1	1	0	xxxx
11	1	1	0	0000
15	0	0	0	0000
20	1	0	0	0000
25	0	0	1	0000
30	1	0	1	0000
31	1	0	1	0001
35	0	0	1	0001
40	1	0	1	0001
41	1	0	1	0010
45	0	0	1	0010
50	1	0	1	0010
51	1	0	1	0011
55	0	0	1	0011
60	1	0	1	0011
61	1	0	1	0100
65	0	0	1	0100
70	1	0	1	0100
71	1	0	1	0101
75	0	0	1	0101
80	1	0	1	0101
81	1	0	1	0110
85	0	0	1	0110
90	1	0	1	0110
91	1	0	1	0111
95	0	0	1	0111
100	1	0	1	0111
101	1	0	1	1000
105	0	0	1	1000
110	1	0	1	1000
111	1	0	1	1001
115	0	0	1	1001
120	1	0	1	1001
121	1	0	1	1010
125	0	0	0	1010

Sayaç Dalgaformu



Verilog HDL Sözdizimi(Syntax) ve Anlambilimi(Semantic)

Sözcüksel Düzen

Verilog HDL yarafından kullanılan temel sözcüksel düzen C programlama diliyle benzerdir. Verilog HDL büyük küçük harf duyarlı(case-sensitive) bir dildir. Tüm anahtar sözcükler küçük harfle yazılır.

Boşluk(White Space)

Boşluk, silme , sekmeler, yenisatır, ve form beslemesi içerebilir. Bu karakterler diğer kavramları bölmediği sürece göz ardı edilirler. Ancak, silme ve sekmeler dizgiler(string) aynı şekildedir.

Boşluk karakterleri:

- Silme boşluğu
- Sekmeler
- Biçimlendirme
- Yeni satır
- Form-besleme

Boşluk(White Spaces) Örneği

Fonksiyonel Olarak Eşit Kodlar

Kötü kod: Kodunuzu asla böyle yazmayın.

```
1 module addbit(a,b,ci,sum,co);
2 input a,b,ci;output sum co;
3 wire a,b,ci,sum,co;endmodule
```

You could download file bad_code.v here

İyi Kod: Kod yazmanın iyi yolu.

```
 1       module addbit (
 2       a,
 3       b,
 4       ci,
 5       sum,
 6       co);
 7       input           a;
 8       input           b;
 9       input           ci;
10       output         sum;
11       output         co;
12       wire            a;
13       wire            b;
14       wire            ci;
15       wire            sum;
16       wire            co;
17
18       endmodule
```

You could download file good_code.v here

Açıklamalar(Comment)

Yorum-açıklamaları belirtmenin iki biçimi vardır.

- Tek satırda açıklama yapılacaksa // ile başlar ve satır başıyla biter
- Birden fazla satır açıklama /* ile başlayıp */ ile bitirilmelidir.

Açıklama örneği

```
 1 /* Bu bir
 2   çok satırlı açıklama
 3   örneğidir */
 4 module addbit (
 5 a,
 6 b,
 7 ci,
 8 sum,
 9 co);
10
11 // Giriş Portları Tek satır açıklama
12 input           a;
13 input           b;
14 input           ci;
15 // Çıkış portları
16 output         sum;
17 output         co;
18 // Veri tipleri
19 wire            a;
20 wire            b;
21 wire            ci;
22 wire            sum;
23 wire            co;
24
25 endmodule
```

You could download file comment.v here

Büyük Küçük Harf Duyarlı(Case Sensitivity)

Verilog HDL büyük küçük harfe duyarlıdır.

- Küçük harfli kelimeler büyük harfli kelimelerden farklıdır
- Tüm verilog anahtar sözcükleri küçük harfle yazılır

Benzersiz isimlere örnek

```
1 input                    // bir Verilog Anahtarsözcüğü
2 wire                     // bir Verilog Anahtarsözcüğü
3 WIRE                   // bir benzersiz isim( bir anahtar sözcük değil)
4 Wire                    //bir benzersiz isim(bir anahtar sözcük değil)
```

You could download file unique_names.v here

NOTE : Verilog anahtarsözcüklerini asla bir benzersiz isim olarak kullanmayın, büyük küçük harf farklı olarak kullanılabilir.

Tanıtıcılar(Identifiers)

Bir objeye verilen isimler olarak kullanılır tanıtıcılar, bir yazmaca veya bir fonksiyona veya bir modüle, tanımlamanın başka yerlerinde referans edilebilir bir isimdir.

- Tanıtıcılar bir alfabetik karakterle veya altçizgi ile başlamalıdır (**a-z A-Z _**)
- Tanıtıcılar alfabetik karakterler, sayısal karakterler, altçizgi ve dolar işareti içeriebilir (**a-z A-Z 0-9 _ $**)
- Tanıcılar 1024 karakter uzunluğuna kadar izin verilir.

Kullanılabilir tanıtıcı örnekleri

data_input mu
clk_input my$clk
i386 A

Kaçak Tanıtıcılar(Escaped Identifiers)

Verilog HDl herhangi bir karakterin kaçak tanıtıcı ile bir tanıtıcıda kullanılmasını sağlar. Kaçak tanıtıcların sağladığı anlam yazılabilir ASCII karakterlerin herhangib bir tanıtıcıda kullanılmasıdır(33'den 126'ya kadar olan ondalık sayılar, veya 21'den 7E kadar olan onaltılık sayılar).

- Kaçak tanıtıcılar ters taksim(\) işareti ile başlar
- Normal tanıtıcılar ile kaçak tanıtıcılar birbirinden farkı ters taksimle anlaşılır.
- Kaçak tanıtıcılar boşluk ile sonlandırılmalı(virgül, parantez, noktalı virgül karakterleri boşluktan önce kullanılırsa kaçak tanıtıcıların bir parçası oalbilir)
- Kaçak tanıtıcılardan sonra boşluk bırakılmazsa tanıtıcıdan sonra gelen karakterlerde tanıtıcının bir parçası olarak kabul edilir.

Kaçak tanıtıcılara örnek

Verilog sayısal bir karakterle başlayan tanıcılara izin vermemektedir. Bu nedenle eğer sayısal bir değerle başlayan bir tanıtıcı kullanmak istiyorsanız bir kaçak tanıtıcı kullanamanız gerekmektedir bu da aşağıda gösterilmiştir.

```
 1 // kaçak karakter olarak kullanılan dizgi(string)den sonra
 2 // mutlaka boşluk olmalı
 3 module \1dff (
 4 q,      // Q çıkışı
 5 \q~ ,   // Q_out çıkışı
 6 d,      // D girişi
 7 cl$k,   // CLOCK girişi
 8 \reset* // Reset girişi
 9 );
10
11 input d, cl$k, \reset* ;
12 output q, \q~ ;
13
14 endmodule
```

You could download file escape_id.v here

Verilog'da Sayılar

Sabit sayıları onluk(decimal), onaltılık(hexadecimal), sekizlik(octal), veya ikilik(binary) biçimde tanımlayabilirsiniz. Negatif sayılar 2'ye tümleyen(2's complement) şekilde gösterilir. Soru işareti(?) karakteri sayılar için kullanıldığında Verilogda z karakterinin alternatifidir. Altçizgi(_) karakteri ilk karakter olmadığı sürece sayılar için heryerde kullanılabilir, ve bu kullanıldığı yerde yok sayılır.

Tamsayılar(Integer)

Verilog HDL'de tamsayıların aşağıdaki şekilde belirtilmesine izin verilir:

- Ölçeklendirilmiş yada ölçeklendirilmemiş sayılar (boyutlandırılmamış büyüklük 32 bit'dir.)
- İkili(binary), sekizli(octal), onlu(decimal), veya onaltılı(hexadecimal) tabanda olabilir.
- Taban ve onaltılı basamaklar(a,b,c,d,e,f) büyük küçük harf duyarlı değildir.
- Boşluk boyut, taban ve değer için kullanılabilir

Söz dizim: <size>'<radix> <value>

Tamsayı örnekleri

Tamsayı	Saklanma şekli
1	00000000000000000000000000000001
8'hAA	10101010
6'b10_0011	100011
'hF	00000000000000000000000000001111

Verilog değer<value> artışı belirtilen boyutta<size> sağdan sola doğru doldurulur.

- Eğer boyut<size> değerden<value> küçükse değerin en soldaki bit'leri atılır
- Eğer boyut<size> değerden<value> büyükse en soldaki bitler değerin en soldaki değerine göre doldurulur.
- ->En soldaki '0' veya '1' ise '0' ile doldurulur.
- ->En soldaki 'Z' ise 'Z' ile doldurulur.
- ->En soldaki'X' ise 'X' ile doldurulur.

Not : X bilinmeyen Z ise yüksek empedans belirtir, 1 lojik yüksek veya 1 ,ve 0 lojik alçak veya 0 belirtir.

Tamsayı örnekleri

Tamsayı	Saklanma şekli
6'hCA	001010
6'hA	001010
16'bZ	ZZZZZZZZZZZZZZZZ
8'bx	xxxxxxxx

Gerçel-Reel Sayılar

- Verilog reel sabitleri ve değişkenleri desteklemektedir
- Verilog reel sayıları tamsayılara yuvarlayarak dönüştürmektedir
- Reel sayılar 'Z' ve 'X' içeremez
- Reel sayılar onluk veya bilimsel şekilde gösterilebilir
- <değer>.<değer>(< value >.< value >)
- < ondalık kısım>E< üst kısmı> (< mantissa >E< exponent >)
- Reel sayılar en yakın tamsayıya göre yuvarlanır.

Gerçek sayılara örnek

Reel Sayı	Ondalık Gösterimi
1.2	1.2
0.6	0.6
3.5E6	3,500000.0

İşaretli ve İşaretsiz Sayılar(Signed and Unsigned Numbers)

Verilog her iki tipteki sayıları da destekler ancak kesin kısıtları vardır. C dilindeki gibi işaretli ve işaretsiz tam sayıları belirtmek için bizim int ve unint tiplerimiz yoktur.

Negatif işaret öneki olmayan herhangi bir sayı pozitiftir veya dolaylı yoldan işaretsizdir(unsigned).

Negatif sayılar, sabit sayılar için boyutu belirtilmeden önce eksi işareti koyulur, böylece işaretli sayı olurlar. Verilog içeride sayıyı 2'ye tümleyeni şeklinde gösterir. Seçmeli bir işaret belirteci işaretli artimetik için eklenmiştir.

Örnekler

Sayı	Açıklama
32'hDEAD_BEEF	İşaretsiz veya işaretli pozitif sayı
-14'h1234	İşaretli negatif sayı

Örnek dosya Verilogun işaretli ve işaretsiz sayılara nasıl davrandığını göstermektedir.

```
 1 module signed_number;
 2
 3 reg [31:0]  a;
 4
 5 initial begin
 6   a = 14'h1234;
 7   $display ("Current Value of a = %h", a);
 8   a = -14'h1234;
 9   $display ("Current Value of a = %h", a);
10   a = 32'hDEAD_BEEF;
11   $display ("Current Value of a = %h", a);
12   a = -32'hDEAD_BEEF;
13   $display ("Current Value of a = %h", a);
14    #10  $finish;
15 end
16
17 endmodule
```

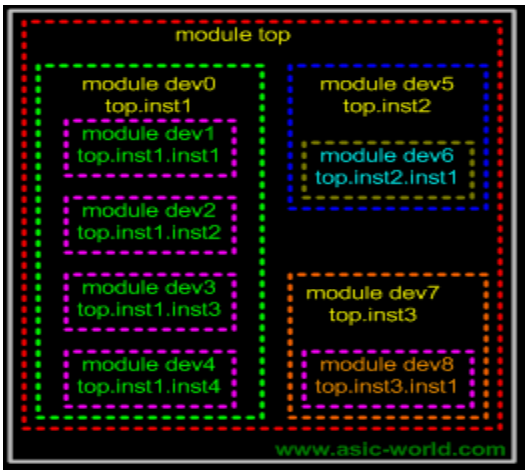
You could download file signed_number.v here

Çıktısı:

```
Current Value of a = 00001234
Current Value of a = ffffedcc
Current Value of a = deadbeef
Current Value of a = 21524111
```

Modüller(Modules)

- Modüller Verilog tasarımının inşaatıdır.
- Tasarım hiyerarşisini diğer modüllerdeki modülleri örnekleyerek oluşturabilirsiniz.
- Başka bir modüldeki,daha üst seviyedeki modüldeki, modülü kullandığınızda onu örneklemiş olursunuz.



Portlar(Port)

- Portlar bir modül ile onun çevresi arasındaki iletişimi sağlar.
- Bir hiyerarşide en üst seviyedeki modül hariç tüm modüllerin portları vardır.
- Portlar sıralarına veya isimlerine göre ilişkilendirilebilir.

Portları giriş(input),çıkış(output) yada girişçıkış(inout) olarak bildirebilirsiniz. Port bildiriminin sözdizimi aşağıdaki gibidir:

input [değer_aralığı:değişken_aralığı] tanıtıcı_listesi; (input [range_val:range_var] list_of_identifiers;)

output [değer_aralığı:değişken_aralığı] tanıtıcı_listesi; (output [range_val:range_var] list_of_identifiers;)

inout [değer_aralığı:değişken_aralığı] tanıtıcı_listesi; (inout [range_val:range_var] list_of_identifiers;)

NOT : İyi bir kod pratiği için, her satırda yalnızca bir port belirteci olmalı, aşağıda olduğu gibi

Port Bildirim Örneği

```
1 input              clk       ; // saat girişi
2 input  [15:0]      data_in   ; // 16 bit veri giriş yolu
3 output [7:0]       count     ; // 8 bit sayaç çıkışı
4 inout              data_bi   ; // çift yönlü veri yolu
```

You could download file port_declare.v here

Verilogda tam bir örnek

```
 1 module addbit (
 2 a      , // ilk giriş
 3 b      , // ikinci giriş
 4 ci     , // elde(carry) girişi
 5 sum    , // toplam çıkışı
 6 co       // elde(carry)çıkışı
 7 );
 8 //Giriş bildirimi(Input declaration)
 9 input a;
10 input b;
11 input ci;
12 //Çıkış bildirimi(Ouput declaration)
13 output sum;
14 output co;
15 //Port Veri tipleri
16 wire  a;
17 wire  b;
18 wire  ci;
19 wire  sum;
20 wire  co;
21 //Kod buradan itibaren başlamaktadır
22 assign {co,sum} = a + b + ci;
23
24 endmodule // addbit modülünün sonu
```

You could download file addbit.v here

Modüller birbirine (dolaylı olarak) port sıralarına göre bağlanmıştır

Burada sıralama doğru olarak eşleşmelidir. Normalde portları birbirine dolaylı olarak bağlamak iyi bir fikir değildir. Herhangi bir port eklendiğinde veya silindiğinde, hata ayıklama(debug) sırasında soruna neden olabilir(örneğin: portların yerleştirilmesi derleme hatasına neden olabilir).

```
 1 //------------------------------------------------
 2 // Bu basit bir toplayıcı programıdır
 3 // Tasarım Adı     : adder_implicit
 4 // Dosya ismi      : adder_implicit.v
 5 // Fonksiyonu      : Bu program dolaylı port bağlantısı
 6 //                  nasıl yapılır onu gösterecek
 7 // Kodlayan        : Deepak Kumar Tala
 8 //------------------------------------------------
 9 module adder_implicit (
10 result        , // Toplayıcının çıkışı
11 carry         , // Toplayıcının elde çıkışı
12 r1            , // ilk giriş
13 r2            , // ikinci giriş
14 ci              // elde girişi
15 );
16
17 // Giriş Port Bildirimi (Input Port Declaration)
18 input    [3:0]   r1         ;
19 input    [3:0]   r2         ;
20 input            ci         ;
21
22 // Çıkış Port Bildirimi (Output Port Declaration)
23 output   [3:0]  result      ;
24 output          carry       ;
25
26 // Port Telleri(Wires)
27 wire     [3:0]    r1        ;
28 wire     [3:0]    r2        ;
29 wire              ci        ;
30 wire     [3:0]    result    ;
31 wire              carry     ;
32
33 // İç değişkenler(Internal variable)
34 wire              c1        ;
35 wire              c2        ;
36 wire              c3        ;
37
38 // Kod buradan başlamaktadır
39 addbit u0 (
40 r1[0]           ,
41 r2[0]           ,
42 ci              ,
43 result[0]       ,
44 c1
45 );
46
47 addbit u1 (
48 r1[1]          ,
49 r2[1]          ,
50 c1             ,
51 result[1]      ,
52 c2
53 );
```

```
54
55 addbit u2 (
56 r1[2]          ,
57 r2[2]          ,
58 c2             ,
59 result[2]      ,
60 c3
61 );
62
63 addbit u3 (
64 r1[3]          ,
65 r2[3]          ,
66 c3             ,
67 result[3]      ,
68 carry
69 );
70
71 endmodule // adder(toplayıcı) modülünün sonu
```

You could download file adder_implicit.v here

İsimleriyle bağlantılı modüller

Hiyerarşiyi bir ağaç olarak düşünürsek, burada isim yaprak modülüyle eşleşmelidir, sıralama önemli değildir.

```
 1 //----------------------------------------------------
 2 // Bu basit bir toplama programıdır
 3 // Tasarım İsmi     : adder_explicit
 4 // Dosya İsmi       : adder_explicit.v
 5 // Fonksiyon        : ismi yaprak Modülü ile eşleşmelidir
 6 // sıralma önemli değildir.
 7 // Tasarımcı        : Deepak Kumar Tala
 8 //----------------------------------------------------
 9 module adder_explicit (
10 result        , // toplayıcının çıkışı sonucu
11 carry         , // Elde çıkışı
12 r1            , // ilk giriş
13 r2            , // ikinci giriş
14 ci              // elde girişi
15 );
16
17 // Giriş Portu Bildirimi
18 input    [3:0]   r1         ;
19 input    [3:0]   r2         ;
20 input            ci         ;
21
22 // Çıkış Portu Bildirimi
23 output   [3:0]  result      ;
24 output          carry       ;
25
26 // Port Telleri
27 wire     [3:0]    r1        ;
28 wire     [3:0]    r2        ;
29 wire              ci        ;
30 wire     [3:0]    result    ;
31 wire              carry     ;
32
33 // İç Değişkenler
34 wire              c1        ;
35 wire              c2        ;
```

```
36 wire              c3        ;
37
38 // Kod Buradan İtibaren Başlamaktadır
39 addbit u0 (
40 .a            (r1[0])        ,
41 .b            (r2[0])        ,
42 .ci           (ci)           ,
43 .sum          (result[0])    ,
44 .co           (c1)
45 );
46
47 addbit u1 (
48 .a            (r1[1])        ,
49 .b            (r2[1])        ,
50 .ci           (c1)           ,
51 .sum          (result[1])    ,
52 .co           (c2)
53 );
54
55 addbit u2 (
56 .a            (r1[2])        ,
57 .b            (r2[2])        ,
58 .ci           (c2)           ,
59 .sum          (result[2])    ,
60 .co           (c3)
61 );
62
63 addbit u3 (
64 .a            (r1[3])        ,
65 .b            (r2[3])        ,
66 .ci           (c3)           ,
67 .sum          (result[3])    ,
68 .co           (carry)
69 );
70
71 endmodule // Toplayıcı(adder) modülü sonu
```

You could download file adder_explicit.v here

Bir modülün başlatılması(Instantiating)

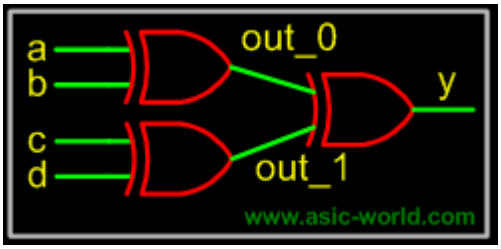
```
 1 //----------------------------------------------------
 2 // Basit bir eşlik(parity) programı
 3 // Tasarım ismi     : parity
 4 // Dosya İsmi       : parity.v
 5 // Fonksiyon        : Bu program basit-ilkel bir modül port balantısının
 6 //                   nasıl yapılacağını göstermektedir
 7 // Kodlayan         : Deepak
 8 //----------------------------------------------------
 9 module parity (
10 a      , // ilk giriş
11 b      , // ikinci giriş
12 c      , // Üçüncü giriş
13 d      , // dördüncü giriş
14 y        // Eşlik(Parity) çıkışı
15 );
16
17 // Giriş Bildirimi
18 input       a       ;
```

```
19 input       b       ;
20 input       c       ;
21 input       d       ;
22 // Çıkış Bildirimi
23 output      y      ;
24 // port veri tipleri
25 wire        a        ;
26 wire        b        ;
27 wire        c        ;
28 wire        d        ;
29 wire        y        ;
30 // İç değişkenler
31 wire        out_0 ;
32 wire        out_1 ;
33
34 // Kod buradan itibaren başlamaktadır
35 xor u0 (out_0,a,b);
36
37 xor u1 (out_1,c,d);
38
39 xor u2 (y,out_0,out_1);
40
41 endmodule // eşlik(parity) modülü sonu
```

You could download file parity.v here

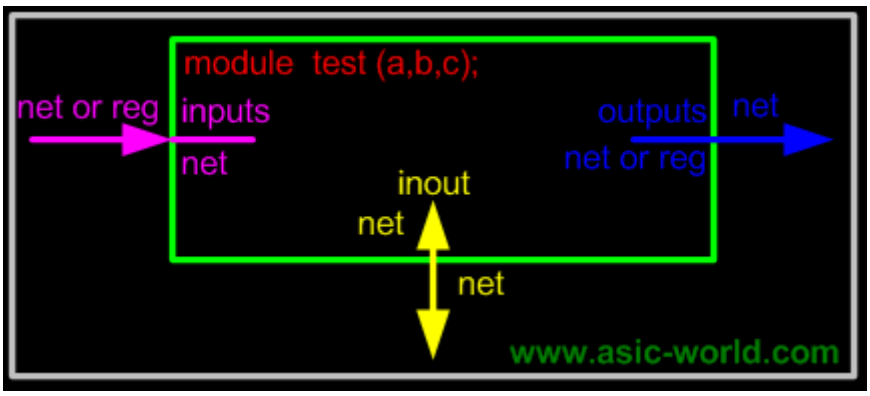
Soru: toplayı modülündeki u0 ile eşlik modülündeki u0 arasındaki fark nedir?

Şematik(Schematic)



Port Bağlantı Kuralları

- Girişler(Inputs) : içten olanlar muhakkak net(ağ) tipinde olmalıdır, harici olanlar reg veya net tipindeki değişkenlerle bağlantılı olmalıdır.
- Çıkışlar(Outputs) : içten net veya reg tipinde olabilir, dıştan çıkışlar net tipindeki bir değişkenle bağlantılı olmalı.
- GirişÇıkışlar(Inouts) : içten veya dıştan herzaman net tipinde olmalı, sadece net tipindeki bir değişkenle bağlantılı olabilir.



- Genişlik eşleşmesi: farklı büyüklükteki içten ve dıştan portları birbirine bağlamak mümkündür. Fakat sentez araçlarının problem rapor edebilir buna dikkat edin.
- Bağımsız portlar(Unconnected ports) : bağımsız portlara bir "," kullanılarak izin verilir.
- net veri tipleri yapıları birbirine bağlamak için kullanılır.
- Bir net veri tipi, eğer bir sinyal yapısal bir bağlantıda kullanılmışsa gereklidir.

Örnek- Dolaylı Bağımsız Port(Implicit Unconnected Port)

```
 1 module implicit();
 2 reg clk,d,rst,pre;
 3 wire q;
 4
 5 // Burada ikinci port bağlanmamıştır
 6 dff u0 ( q,,clk,d,rst,pre);
 7
 8 endmodule
 9
10 // D flip-flop
11 module dff (q, q_bar, clk, d, rst, pre);
12 input clk, d, rst, pre;
13 output q, q_bar;
14 reg q;
15
16 assign q_bar = ~q;
17
18 always @ (posedge clk)
19 if (rst == 1'b1) begin
20   q <= 0;
21 end else if (pre == 1'b1) begin
22   q <= 1;
23 end else begin
24   q <= d;
25 end
26
27 endmodule
```

You could download file implicit.v here

Örnek – Açıkça Bağlantısız Port(Explicit Unconnected Port)

```
 1 module explicit();
 2 reg clk,d,rst,pre;
 3 wire q;
 4
 5 // Burada q_bar bağlanmamıştır
 6 // Portları herhangi bir ısrada bağlayabiliriz
 7 dff u0 (
 8 .q         (q),
 9 .d  (d),
10 .clk       (clk),
11 .q_bar     (),
12 .rst       (rst),
13 .pre       (pre)
14 );
15
16 endmodule
17
18 // D flip-flop
19 module dff (q, q_bar, clk, d, rst, pre);
20 input clk, d, rst, pre;
21 output q, q_bar;
22 reg q;
23
24 assign q_bar = ~q;
25
26 always @ (posedge clk)
27 if (rst == 1'b1) begin
28   q <= 0;
29 end else if (pre == 1'b1) begin
30   q <= 1;
31 end else begin
32   q <= d;
33 end
34
35 endmodule
```

You could download file explicit.v here

Hiyerarşik Tanıtıcılar

Hiyerarşik yol isimleri, en üstteki modül tanıtıcı modül anlık tanıtıcılar temellidir, periyotlara bölünmüştür.

Bu temelde kullanışlıdır, daha düşük bir modüldeki sinyalin içini görmek istediğimizde veya bir iç modülün içindeki değeri zorluyorsak. Aşağıdaki örnekte iç modül sinyalinin nasıl izlendiği gösterilmiştir.

Örnek

```
1 //-------------------------------------------------
2 // Basit bir toplayıcı program
3 // Tasarım İsmi : adder_hier
4 // Dosya İsmi   : adder_hier.v
5 // Fonksiyon : Bu program verilogun hiyerarşik yol işini göstermektedir
6 // Kodlayan     : Deepak
7 //-------------------------------------------------
```

```
 8 `include "addbit.v"
 9 module adder_hier (
10 result        , // Toplayıcının çıkışı
11 carry         , // toplayıcının elde çıkışı
12 r1            , // ilk giriş
13 r2            , // ikinci giriş
14 ci              // elde girişi
15 );
16
17 // Giriş Portları Bildirimi
18 input    [3:0]   r1         ;
19 input    [3:0]   r2         ;
20 input            ci         ;
21
22 // Çıkış Portları Bildirimi
23 output   [3:0]  result      ;
24 output          carry       ;
25
26 // Port Telleri
27 wire     [3:0]    r1        ;
28 wire     [3:0]    r2        ;
29 wire              ci        ;
30 wire     [3:0]    result    ;
31 wire              carry     ;
32
33 // İç değişkenler
34 wire              c1        ;
35 wire              c2        ;
36 wire              c3        ;
37
38 // Kod buradan itibaren başlamaktadır
39 addbit u0 (r1[0],r2[0],ci,result[0],c1);
40 addbit u1 (r1[1],r2[1],c1,result[1],c2);
41 addbit u2 (r1[2],r2[2],c2,result[2],c3);
42 addbit u3 (r1[3],r2[3],c3,result[3],carry);
43
44 endmodule // adder modülünün sonu
45
46 module tb();
47
48 reg [3:0] r1,r2;
49 reg  ci;
50 wire [3:0] result;
51 wire  carry;
52
53 // girişlerin kullanılması
54 initial begin
55   r1 = 0;
56   r2 = 0;
57   ci = 0;
58    #10  r1 = 10;
59    #10  r2 = 2;
60    #10  ci = 1;
61    #10  $display("+-------------------------------------------------
----+");
62   $finish;
63 end
64
65 // Bir alt modülle bağlantı
66 adder_hier U (result,carry,r1,r2,ci);
67
```

```
 68 // Hier demo burada
 69 initial begin
 70   $display("+-----------------------------------------------------
+");
 71   $display("|  r1  |  r2  |  ci  | u0.sum | u1.sum | u2.sum | u3.sum
|");
 72   $display("+-----------------------------------------------------
+");
 73   $monitor("|  %h   |  %h   |  %h   |    %h    |   %h   |   %h    |
%h    |",
 74   r1,r2,ci, tb.U.u0.sum, tb.U.u1.sum, tb.U.u2.sum, tb.U.u3.sum);
 75 end
 76
 77 endmodule
```

You could download file adder_hier.v here

```
+-----------------------------------------------------+
|  r1  |  r2  |  ci  | u0.sum | u1.sum | u2.sum | u3.sum |
+-----------------------------------------------------+
|  0   |  0   |  0   |    0   |   0    |   0    |   0    |
|  a   |  0   |  0   |    0   |   1    |   0    |   1    |
|  a   |  2   |  0   |    0   |   0    |   1    |   1    |
|  a   |  2   |  1   |    1   |   0    |   1    |   1    |
+-----------------------------------------------------+
```

Veri Tipleri

Verilog Dili iki temel veri tipine sahiptir:

- **Net** – bileşenler arasındaki yapısal bağlantıları gösterir.
- **Register** – veri saklamak için kullanılan değişkenleri gösrerir.

Herbir sinyalin onunla ilişkili bir veri tipi vardır:

- **Açıkça bildirilmiş**-Verilog kodunda açıkça bildirim yaparak**.**
- **Dolaylı bildirilmiş**- bildirim olmaksızın kodunuzun yapısal inşa bloğunda bağlantı için kullanılır. Dolaylı bildirim herzaman bir net tipinde “wire” ve bir bit genişliğindedir.

Net Tipleri

Herbir net tipi farklı tipteki donanımları(PMOS, NMOS, CMOS, vb) modellemek için kullanılmış bir fonksiyonaliteye sahiptir.

Net Veri Tipi	Fonksiyonalitesi
wire, tri	Bağlantılı olan tel(wire)-özel bir çözünürlük fonksiyonu yoktur
wor, trior	Telli(Wired) çıkışlar OR(VEYA) ile birlikte(ECL ‘i modeller)
wand, triand	Telli(Wired) çıkışlar AND(VE) ile birlikte (açık-kollektör)
tri0, tri1	Net kullanılmadığı zaman aşağı yukarı çekmek için
supply0, supply1	Net sabit lojik 0 veya lojik 1(güç sağlama)’e sahiptir
Trireg	Z ile sürüldüğünde(üçdurumlu-tristate) son değeri tutar

Not : Tüm net veri tipleri arasında en sık kullanılanı teldir(wire).

Örnek - wor

```
 1 module test_wor();
 2
 3 wor a;
 4 reg b, c;
 5
 6 assign a =  b;
 7 assign a =  c;
 8
 9 initial begin
10   $monitor("%g a = %b b = %b c = %b", $time, a, b, c);
11    #1  b  = 0;
12    #1  c  = 0;
13    #1  b = 1;
14    #1  b = 0;
15    #1  c = 1;
16    #1  b = 1;
17    #1  b = 0;
18    #1  $finish;
19 end
20
21 endmodule
```

You could download file test_wor.v here

Simülatör Çıktısı

```
0 a = x b = x c = x
1 a = x b = 0 c = x
2 a = 0 b = 0 c = 0
3 a = 1 b = 1 c = 0
4 a = 0 b = 0 c = 0
5 a = 1 b = 0 c = 1
6 a = 1 b = 1 c = 1
7 a = 1 b = 0 c = 1
```

Örnek- wand

```
 1 module test_wand();
 2
 3 wand a;
 4 reg b, c;
 5
 6 assign a =  b;
 7 assign a =  c;
 8
 9 initial begin
10   $monitor("%g a = %b b = %b c = %b", $time, a, b, c);
11    #1  b  = 0;
12    #1  c  = 0;
13    #1  b = 1;
14    #1  b = 0;
15    #1  c = 1;
16    #1  b = 1;
17    #1  b = 0;
18    #1  $finish;
19 end
20
21 endmodule
```

You could download file test_wand.v here

Simülatör Çıktısı

```
 0 a = x b = x c = x
 1 a = 0 b = 0 c = x
 2 a = 0 b = 0 c = 0
 3 a = 0 b = 1 c = 0
 4 a = 0 b = 0 c = 0
 5 a = 0 b = 0 c = 1
 6 a = 1 b = 1 c = 1
 7 a = 0 b = 0 c = 1
```

Örnek- tri

```
 1 module test_tri();
 2
 3 tri a;
 4 reg b, c;
 5
 6 assign a = (b) ? c : 1'bz;
 7
 8 initial begin
 9   $monitor("%g a = %b b = %b c = %b", $time, a, b, c);
10   b  = 0;
11   c  = 0;
12    #1  b = 1;
13    #1  b = 0;
14    #1  c = 1;
15    #1  b = 1;
16    #1  b = 0;
17    #1  $finish;
18 end
19
20 endmodule
```

You could download file test_tri.v here

Simülatör Çıktısı

```
 0 a = z b = 0 c = 0
 1 a = 0 b = 1 c = 0
 2 a = z b = 0 c = 0
 3 a = z b = 0 c = 1
 4 a = 1 b = 1 c = 1
 5 a = z b = 0 c = 1
```

Örnek- trireg

```
 1 module test_trireg();
 2
 3 trireg a;
 4 reg b, c;
 5
 6 assign a = (b) ? c : 1'bz;
 7
 8 initial begin
 9   $monitor("%g a = %b b = %b c = %b", $time, a, b, c);
10   b  = 0;
11   c  = 0;
12    #1  b = 1;
13    #1  b = 0;
14    #1  c = 1;
15    #1  b = 1;
```

```
16    #1  b = 0;
17    #1  $finish;
18 end
19
20 endmodule
```

You could download file test_trireg.v here

Simülatör Çıktısı

```
0 a = x b = 0 c = 0
1 a = 0 b = 1 c = 0
2 a = 0 b = 0 c = 0
3 a = 0 b = 0 c = 1
4 a = 1 b = 1 c = 1
5 a = 1 b = 0 c = 1
```

Yazmaç(Register) Veri Tipleri

- Yazmaçlar(Registers) başka bir atama olana kadar kendilerine atanmış son değeri saklarlar.
- Yazmaçlar veri saklama yapıları olarak ifade edilir.
- Regs dizisi ile hafıza(memory) adı verilen yapılar oluşturabilirsiniz.
- Prosedürel bloklarda(procedural block) yazmaç veri tipi değişken olarak kullanılır.
- Bir prosedürel blokta(procedural block) eğer bir sinyale bir değer atandıysa bir yazmaç veri tipi gereklidir.
- Prosedürel bloklar (Procedural blocks) **initial** ve **always** anahtar sözcükleri ile başlar.

Veri Tipleri	Fonksiyonalite
reg	İşaretsiz değişken(Unsigned variable)
integer	İşaretli değilken(Signed variable) - 32 bit'lik
time	İşaretsiz tamsayı (Unsigned integer) - 64 bit'lik
real	Çift duyarlıklı kayan noktalı değişkenler(Double precision floating point variable)

Not : Tüm yazmaç veri tiplerinden reg en genel kullanılanıdır.

Dizgiler(Strings)

Bir dizgi çift tırnak arasında sınırlandırılmasıyla ve hepsinin tek bir satırı içerdiği bir dizi karakterdir. Dizgiler ifadelerde işlemler olarak kullanılır ve atamalar sekiz-bitlik ASCII değerler dizisi gibi davranır, herbir sekiz-bitlik ASCII değer bir karakteri gösterir. Bir dizgiye kaydedilecek bir değişkeni bildirmek için, maksimum sayıdaki karakteri taşıyabilecek boyuttaki değişken bir yazmaç bildirimelidir. Sonlandırma karakteri için herhangi bir ekstra bit gerekmemektedir; Verilog'un dizgi sonlandırma karakteri yoktur. Dizgiler üstesinden standart operatörlerle gelinebilir.

Eğer bir değişkene atanabileceğinden daha küçük bir değer verildiyse, atamadan sonra Verilog içeriğin solunu sıfırlarla doldurmaktadır. Bu dizgi olmayan değerlerin atanamasında da geçerlidir.

Kesin karakterler(certain character) dizgilerde sadece çıkış karakteri ile birlikte kullanılabilir. Aşağıdaki tabloda gösterilmiştir.

Dizgilerde(String) Özel Karakterler

Karakter	Tanımlama
\n	Yeni satır karakteri(New line character)
\t	Tab karakteri
\\	Ters taksim(Backslash) (\) karakteri
\"	Çift tırnak(") karakteri
\ddd	1-3 sekizlik basamakta belirtilmiş bir karakter için (0 <= d <= 7)
%%	Oran (%) karakteri

Örnek

```
 1 //------------------------------------------------------
 2 // Tasarım İsmi    : strings
 3 // Dosya İsmi      : strings.v
 4 // Fonksiyonu      : Bu program dizgilerin(string)
 5 //         yazmaçlarda(reg) nasıl tutulduğunu gösterecek
 6 // Kodlayan        : Deepak Kumar Tala
 7 //------------------------------------------------------
 8 module strings();
 9 // 21 byte'lık bir yazmaç(register) değişkeni bildirimi
10 reg [8*21:0] string ;
11
12 initial begin
13   string = "This is sample string";
14   $display ("%s \n", string);
15 end
16
17 endmodule
```

You could download file strings.v here

```
This is sample string
```

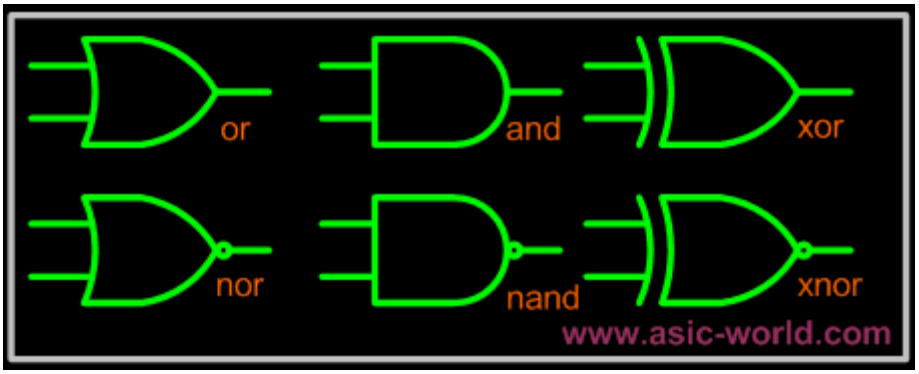
Kapı Seviyesi Modelleme(Gate Level Modeling)

Giriş

Verilog kapılar, iletim kapıları, ve anahtarlar gibi basitlikler üzerine kurulmuştur. Bunlar tasarımda çok nadir kullanılır(RTL kodlamada), fakat ASIC/FPGA hücrelerini modellemek için sentez sonrası dünyada kullanılır; bu hücreler daha sonra kapı seviyesi simülasyon, veya SDF simülasyon için kullanılır. Ayrıca sentez aracından gelen çıkış netlist formatı alan ve rota aracının içine aktarılır, bu da Verilog kapı seviyesi temelleridir.

Not : RTL mühendisleri hala kapı seviyesi temelleri kullanırlar veya RTL'de IO hücreleri kullanırken ASIC kütüphane hücresi kullanırlar, bunlar çapraz uzay senkronizasyon hücreleridir.

Kapı Temelleri(Gate Primitives)



Kapılar bir skaler çıkış ve birden çok skaler girişe sahiptir. Kapı terminalleri listesindeki ilk terminal çıkış ve diğer terminaller girişlerdir.

Kapı	Tanımlama
and	N-girişli AND kapısı
nand	N-girişli NAND kapısı
or	N- girişli OR kapısı
nor	N- girişli NOR kapısı
xor	N- girişli XOR kapısı
xnor	N- girişli XNOR kapısı

Örnekler

```
 1 module gates();
 2
 3 wire out0;
 4 wire out1;
 5 wire out2;
 6 reg  in1,in2,in3,in4;
 7
 8 not U1(out0,in1);
 9 and U2(out1,in1,in2,in3,in4);
10 xor U3(out2,in1,in2,in3);
11
12 initial begin
13   $monitor(
14   "in1=%b in2=%b in3=%b in4=%b out0=%b out1=%b out2=%b",
15    in1,in2,in3,in4,out0,out1,out2);
16   in1 = 0;
17   in2 = 0;
18   in3 = 0;
19   in4 = 0;
20    #1  in1 = 1;
21    #1  in2 = 1;
22    #1  in3 = 1;
23    #1  in4 = 1;
24    #1  $finish;
25 end
26 endmodule
```

You could download file gates.v here

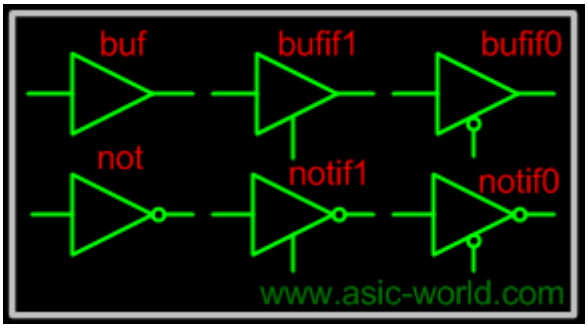
```
in1 = 0 in2 = 0 in3 = 0 in4 = 0 out0 = 1 out1 = 0 out2 = 0
in1 = 1 in2 = 0 in3 = 0 in4 = 0 out0 = 0 out1 = 0 out2 = 1
in1 = 1 in2 = 1 in3 = 0 in4 = 0 out0 = 0 out1 = 0 out2 = 0
in1 = 1 in2 = 1 in3 = 1 in4 = 0 out0 = 0 out1 = 0 out2 = 1
in1 = 1 in2 = 1 in3 = 1 in4 = 1 out0 = 0 out1 = 1 out2 = 1
```

İletim Kapı Temelleri(Transmission Gate Primitives)

İletim kapıları iki yönlü kapılardır ve dirençli(resistive) ve dirençsiz(non- resistive) olabilir.

Sözdizim: anahtarsözcük benzersiz_isim(girişçıkış1, girişçıkış2, kontrol);

(keyword unique_name (inout1, inout2, control);)



Kapı	Tanımlama
not	N-çıkışlı evireç(inverter)
buf	N- çıkışlı ara bellek(buffer).
bufif0	Üç-durumlı(Tri-state) buffer, alt seviyede aktif
bufif1	Üç-durumlı(Tri-state) buffer, üst seviyede aktif.
notif0	Üç-durumlı(Tri-state) evireç(inverter), düşükken etkin.
notif1	Üç-durumlı(Tri-state) evireç(inverter), yüksekken etkin.

İletim kapıları(Transmission gates) tran ve rtran sürekli açıktır ve bir kontrol hattı yoktur. Tran, iki teli bölünmüş şekilde sürülmesi için arayüz olarak kullanılır, ve rtran ise sinyalleri zayıflatmak için kullanılır.

Örnek

```
1 module transmission_gates();
2
3 reg data_enable_low, in;
4 wire data_bus, out1, out2;
5
6 bufif0 U1(data_bus,in, data_enable_low);
7 buf  U2(out1,in);
8 not U3(out2,in);
9
```

```
10 initial begin
11   $monitor(
12     "@%g in=%b data_enable_low=%b out1=%b out2= b data_bus=%b",
13     $time, in, data_enable_low, out1, out2, data_bus);
14   data_enable_low = 0;
15   in = 0;
16    #4  data_enable_low = 1;
17    #8  $finish;
18 end
19
20 always  #2  in = ~in;
21
22 endmodule
```

You could download file transmission_gates.v here

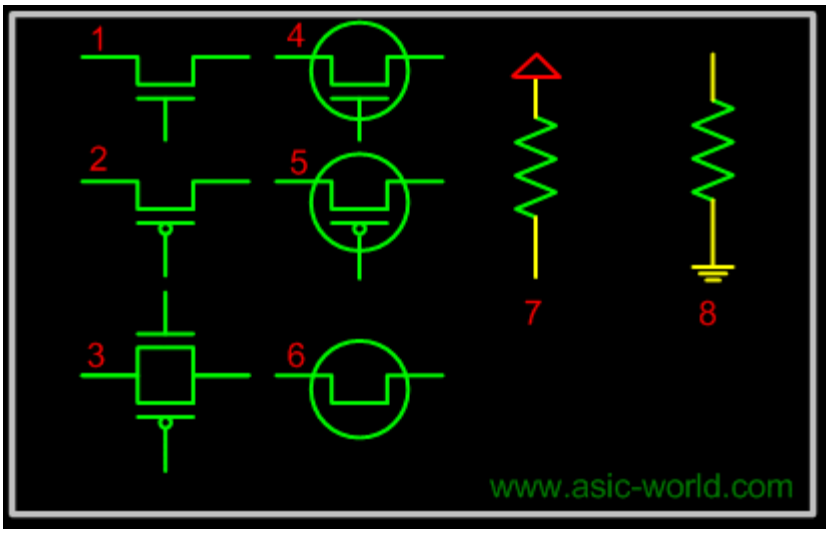
```
@0 in = 0 data_enable_low = 0 out1 = 0 out2 = 1 data_bus = 0
@2 in = 1 data_enable_low = 0 out1 = 1 out2 = 0 data_bus = 1
@4 in = 0 data_enable_low = 1 out1 = 0 out2 = 1 data_bus = z
@6 in = 1 data_enable_low = 1 out1 = 1 out2 = 0 data_bus = z
@8 in = 0 data_enable_low = 1 out1 = 0 out2 = 1 data_bus = z
@10 in = 1 data_enable_low = 1 out1 = 1 out2 = 0 data_bus = z
```

Anahtar(Switch) Temelleri

Verilogda altı farklı anahtar temeli(transistor modelleri) kullanılmaktadır, nmos, pmos ve cmos, ve bunlarla ilişkili üç dirneçsel versiyonu vardır rnmos, rpmos ve rcmos. Cmos tipi anahtarların iki kapısı ve bu nedenle iki kontrol sinyali vardır.

Sözdizimi: anahtarsözcük benzersiz_isim(akaç,kaynak, kapı)

keyword unique_name (drain. source, gate)



Kapı	Tanımlaması
1. pmos	Tek-yönlü PMOS anahtarlama
1. rpmos	Dirençli PMOS anahtarlama
2. nmos	Tek-yönlü NMOS anahtarlama
2. rnmos	Dirençli NMOS anahtarlama
3. cmos	Tek-yönlü CMOS anahtarlama
3. rcmos	Dirençli CMOS anahtarlama
4. tranif1	Çift-yönlü transistör (Yüksek)
4. tranif0	Çift-yönlü transistör (Düşük)
5. rtranif1	Dirençli transistör (Yüksek)
5. rtranif0	Dirençli transistör (Düşük)
6. tran	Çift-yön geçişli transistör
6. rtran	Dirençli geçişli transistör
7. pullup	Yukarı çekme(pull up) direnci(resistor)
8. pulldown	Aşağı çekme(pull down) direnci(resistor)

İletim kapıları çift-yönlüdürler ve dirençli veya dirençsiz olabilir. Direçli araçlar sinyal bir seviye görülen çıkış gücünü azaltır. Tüm anahtarlar sinyalleri sadece kaynaktan(source) akaca(drain) iletirler, araçların yanlış bağlanması yüksek empedans çıkışının oluşmasıyla sonuçlanır.

Örnek

```
1 module switch_primitives();
2
3 wire  net1, net2, net3;
4 wire  net4, net5, net6;
5
6 tranif0 my_gate1 (net1, net2, net3);
7 rtranif1 my_gate2 (net4, net5, net6);
8
9 endmodule
```

You could download file switch_primitives.v here

İletim kapıları(Transmission gates) tran ve rtran sürekli açıktır ve bir kontrol hattı yoktur. Than iki tel ayrı sürücüleri göstermek için, rtan ise sinyalleri zayıflatmak için kullanılır. Dirençli araçlar sinyal gücünü azaltır bu da bir seviyedeki çıkışta kendini gösterir. Tüm anahtarlar sinyalleri sadece kaynaktan(source) akaca(drain) iletirler, araçların yanlış bağlanması yüksek empedans çıkışının oluşmasıyla sonuçlanır.

Lojik Değerler ve Sinyal Güçleri

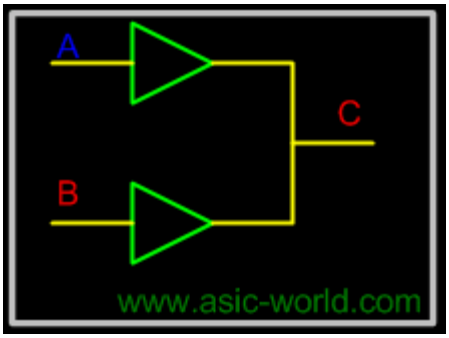
Verilog HDL'in dört lojik değeri vardır.

Lojik Değer	Tanımı
0	Sıfır(zero), düşük(low), yanlış(false)
1	Bir(one), yüksek(high), doğru(true)
z veya **Z**	Yüksek empedans(high impedance), dalgalanan(floating)
x veya **X**	Bilinmeyen(unknown), henüz başlangıç değeri olmayan(uninitialized), çekişme(contention)

Verilog Güç Seviyeleri

Güç Seviyesi	Kullanılan Anahtar Sözcük
7 Besleme Sürücüsü(Supply Drive)	supply0 supply1
6 Güçlü Çekim(Strong Pull)	strong0 strong1
5 Çekem Sürücüsü(Pull Drive)	pull0 pull1
4 Yüksek Direnç(Large Capacitance)	large
3 Zayıf Sürücü(Weak Drive)	weak0 weak1
2 Orta Direnç(Medium Capacitance)	medium
1 Küçük Direnç(Small Capacitance)	small
0 Yüksek Empedans(Hi Impedance)	highz0 highz1

Örnek: Güçlü Seviye(Strength Level)



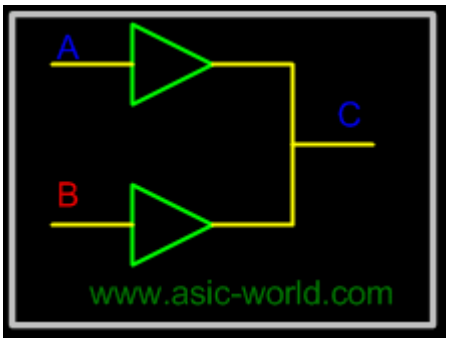
İki arabelleğinde çıkışı var

A : Pull 1 (1'e Çekme)

B : Supply 0 (0'la Besleme)

Supply 0 pull 1'den daha güçlü olduğu sürece C Çıkışı B değerini alır.

Örnek 2 : Güç Seviyesi



İki arabelleğinde çıkışı var

A : Supply 1 (1'le Besleme)

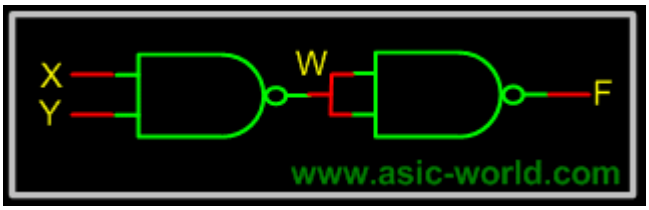
B : Large 1(Büyük 1)

Supply 1 Large 1'den güçlü olduğu sürece, C Çıkışı A değerini alır.

Temelleri Kullanarak Tasarlama(Designing Using Primitives)

Temelleri kullanarak tasarlama sadece kütüphane geliştirmede kullanılır, ASIC sağlayıcısı ASIC kütüphanesi Verilog betimlemeyi, Verilog temelleri ve kullanıcı tanımlı temeller(user defined primitives-(UDP)) kullanılarak destekler.

NAND(VEDEĞİL) kapısından AND(VE) Kapısı Elde Etme



Kodu

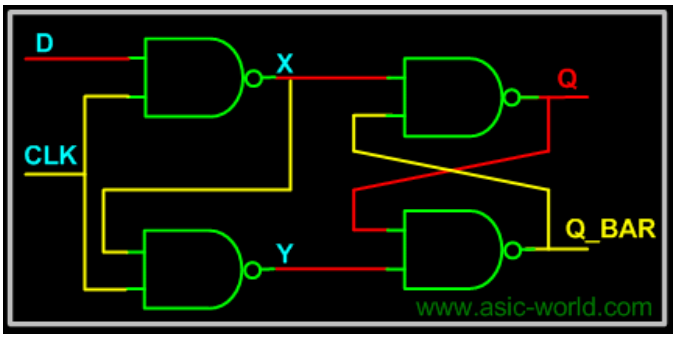
```
  1 //NAND(VEDEĞİL) kapılarından AND(VE) kapısının oluşturulmasının yapısal
modeli
  2 module and_from_nand();
  3
  4 reg X, Y;
  5 wire F, W;
  6 // NAND(VEDEĞİL) modülünün iki örneklemi
  7 nand U1(W,X, Y);
  8 nand U2(F, W, W);
  9
```

```
10 // Testbenç Kodu
11 initial begin
12   $monitor ("X = %b Y = %b F = %b", X, Y, F);
13   X = 0;
14   Y = 0;
15    #1  X = 1;
16    #1  Y = 1;
17    #1  X = 0;
18    #1  $finish;
19 end
20
21 endmodule
```

You could download file and_from_nand.v here

```
X = 0 Y = 0 F = 0
X = 1 Y = 0 F = 0
X = 1 Y = 1 F = 1
X = 0 Y = 1 F = 0
```

NAND(VEDEĞİL) kapılarından D-Flip elde edilmesi



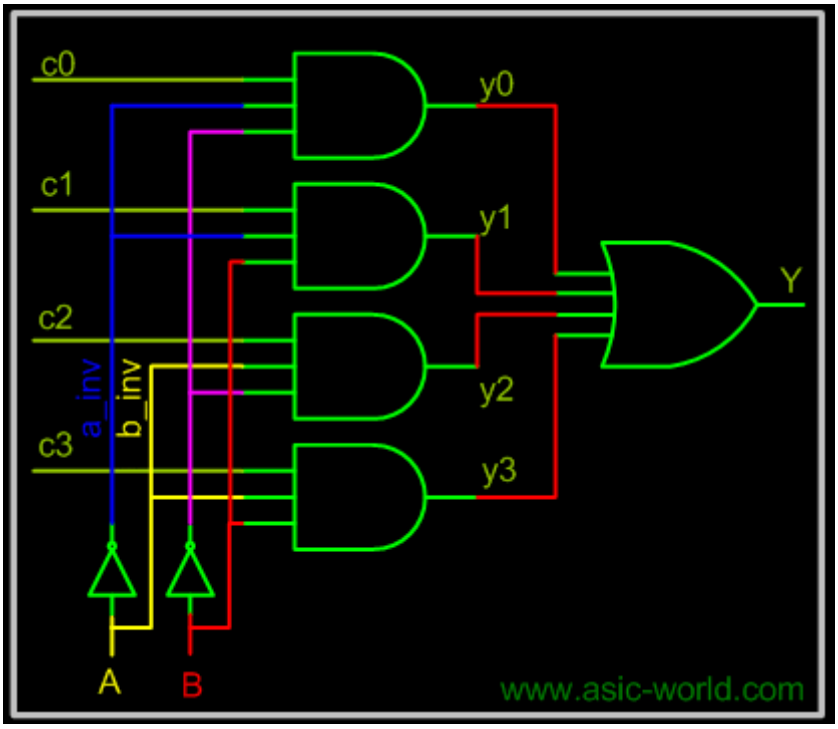
Verilog Kodu

```
 1 module dff_from_nand();
 2 wire Q,Q_BAR;
 3 reg D,CLK;
 4
 5 nand U1 (X,D,CLK) ;
 6 nand U2 (Y,X,CLK) ;
 7 nand U3 (Q,Q_BAR,X);
 8 nand U4 (Q_BAR,Q,Y);
 9
10 // Belirtilen kodun testbenci
11 initial begin
12   $monitor("CLK = %b D = %b Q = %b Q_BAR = %b",CLK, D, Q, Q_BAR);
13   CLK = 0;
14   D = 0;
15    #3  D = 1;
16    #3  D = 0;
17    #3  $finish;
18 end
19
20 always  #2  CLK = ~CLK;
21
22 endmodule
```

You could download file dff_from_nand.v here

```
CLK = 0 D = 0 Q = x Q_BAR = x
CLK = 1 D = 0 Q = 0 Q_BAR = 1
CLK = 1 D = 1 Q = 1 Q_BAR = 0
CLK = 0 D = 1 Q = 1 Q_BAR = 0
CLK = 1 D = 0 Q = 0 Q_BAR = 1
CLK = 0 D = 0 Q = 0 Q_BAR = 1
```

Temellerden Çoklayıcı(Multiplexer) Elde Edilmesi



Verilog Kodu

```
 1 module mux_from_gates ();
 2 reg c0,c1,c2,c3,A,B;
 3 wire Y;
 4 //Invert the sel signals
 5 not (a_inv, A);
 6 not (b_inv, B);
 7 // 3-input AND gate
 8 and (y0,c0,a_inv,b_inv);
 9 and (y1,c1,a_inv,B);
10 and (y2,c2,A,b_inv);
11 and (y3,c3,A,B);
12 // 4-input OR gate
13 or (Y, y0,y1,y2,y3);
14
15 // Testbench Code goes here
16 initial begin
17   $monitor (
18    "c0 = %b c1 = %b c2 = %b c3 = %b A = %b B = %b Y = %b",
19    c0, c1, c2, c3, A, B, Y);
20   c0 = 0;
21   c1 = 0;
22   c2 = 0;
23   c3 = 0;
24   A = 0;
```

```
25   B = 0;
26    #1  A  = 1;
27    #2  B  = 1;
28    #4  A  = 0;
29    #8  $finish;
30 end
31
32 always  #1  c0 = ~c0;
33 always  #2  c1 = ~c1;
34 always  #3  c2 = ~c2;
35 always  #4  c3 = ~c3;
36
37 endmodule
```

You could download file mux_from_gates.v here

```
c0 = 0 c1 = 0 c2 = 0 c3 = 0 A = 0 B = 0 Y = 0
c0 = 1 c1 = 0 c2 = 0 c3 = 0 A = 1 B = 0 Y = 0
c0 = 0 c1 = 1 c2 = 0 c3 = 0 A = 1 B = 0 Y = 0
c0 = 1 c1 = 1 c2 = 1 c3 = 0 A = 1 B = 1 Y = 0
c0 = 0 c1 = 0 c2 = 1 c3 = 1 A = 1 B = 1 Y = 1
c0 = 1 c1 = 0 c2 = 1 c3 = 1 A = 1 B = 1 Y = 1
c0 = 0 c1 = 1 c2 = 0 c3 = 1 A = 1 B = 1 Y = 1
c0 = 1 c1 = 1 c2 = 0 c3 = 1 A = 0 B = 1 Y = 1
c0 = 0 c1 = 0 c2 = 0 c3 = 0 A = 0 B = 1 Y = 0
c0 = 1 c1 = 0 c2 = 1 c3 = 0 A = 0 B = 1 Y = 0
c0 = 0 c1 = 1 c2 = 1 c3 = 0 A = 0 B = 1 Y = 1
c0 = 1 c1 = 1 c2 = 1 c3 = 0 A = 0 B = 1 Y = 1
c0 = 0 c1 = 0 c2 = 0 c3 = 1 A = 0 B = 1 Y = 0
c0 = 1 c1 = 0 c2 = 0 c3 = 1 A = 0 B = 1 Y = 0
c0 = 0 c1 = 1 c2 = 0 c3 = 1 A = 0 B = 1 Y = 1
```

Kapı ve Anahtar(Switch) Gecikmesi(delay)

Gerçek devrelerde, lojik kapılar onlarla ilişkili olan gecikmelere sahiptir. Verilog kapılarla ilişkilendirilmiş gecikme mekanızmasını sağlar.

- Yükseliş(Rise), Düşüş(Fall) and Kapama(Turn-off) gecikmeleri.
- En düşük(Minimal), Tipik(Typical), ve En fazla(Maximum)gecikmeler.

Verilogda gecikmeler(delay) aşağıdaki örnekte de gösterildiği gibi #'sayı' ile belirtilir, # gecikmeyi belirtmek için özel bir karakteri ifade eder, ve sayı da simülatörün bu ifadeye geldiğinde tıklama sayısını belirtir.

- #1 a = b : 1 birim geciktir, yada 1 tikden sonra gerçekleştir
- #2 not (a,b) : Tüm atama 2 birim gecikerek yapılır.

Gerçek transistörlerin giriş ve çıkış arasında çözünürlük gecikmesi(resolution delay) vardır. Bu Verilogda yükselen(rise), düşen(fall), kapama(turn-off) zamanı virgülle bölünerek bir veya birden fazla gecikme belirtilerek modellenir.

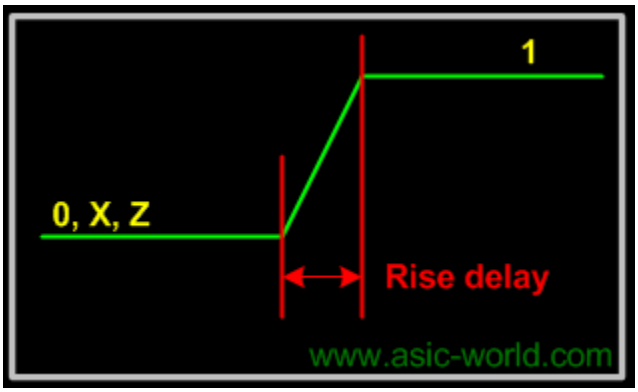
Sözdizim: anahtar sözcük #(gecikme{ler}) benzersiz_isim(düğüm berlirtimi);

keyword #(delay{s}) unique_name (node specifications);

Anahtar Elemanı	Gecikme Sayısı	Belirlenmiş gecikmeler
Switch	1	Yükselen(Rise), düşen(fall) ve kapama(turn-off) zamanları eşit uzunluktadır
	2	Yükselen (Rise) ve düşen(fall) zamanları
	3	Yükselen(Rise), düşen(fall) ve kapama(turn-off)
(r)tranif0, (r)tranif1	1	Hem açma(turn on) hem de kapama(turn off)
	2	açma(turn on), kapama(turn off)
(r)tran	0	İzin verilmez

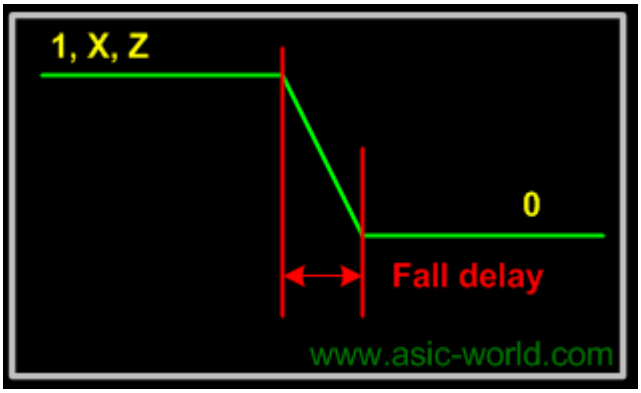
Yükselen Gecikme(Rise Delay)

Yükselen gecikme bir kapı çıkışının başka bir değerden(0,x,z) 1'e geçişidir.



Düşen Gecikme(Fall Delay)

Düşen gecikme bir kapı çıkışının başka bir değerden(1,x,z) 0'a geçişidir.



Kapama Gecikmesi(Turn-off Delay)

Kapama gecikmesi bir kapı çıkışının başka bir değerden(0,1,x) z'ye geçişidir.

En Küçük Değer (Min Value)

En küçük değer bir kapının beklenen en az gecikme değeridir.

Tipik Değer (Typ Value)

Typ değer bir kapının beklenen tipik gecikme değeridir.

En Büyük Değer (Max Value)

Max değer bir kapının beklenen en büyük gecikme değeridir.

Örnek

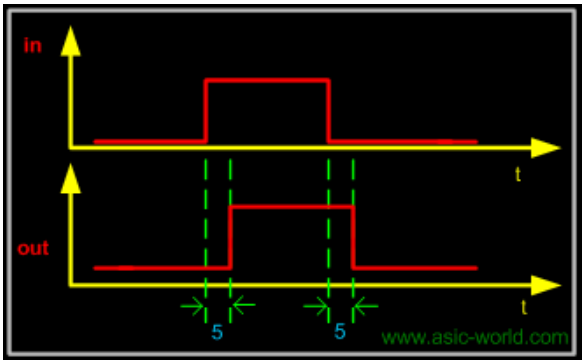
Aşağıda gecikmeni kullnımıyla ilgili bazı örnekler vardır.

Örnek – Tekli Gecikme(Single Delay)

```
 1 module  buf_gate ();
 2 reg in;
 3 wire out;
 4
 5 buf #(5) (out,in);
 6
 7 initial begin
 8   $monitor ("Time = %g in = %b out=%b", $time, in, out);
 9   in = 0;
10    #10  in = 1;
11    #10  in = 0;
12    #10  $finish;
13 end
14
15 endmodule
```

You could download file buf_gate.v here

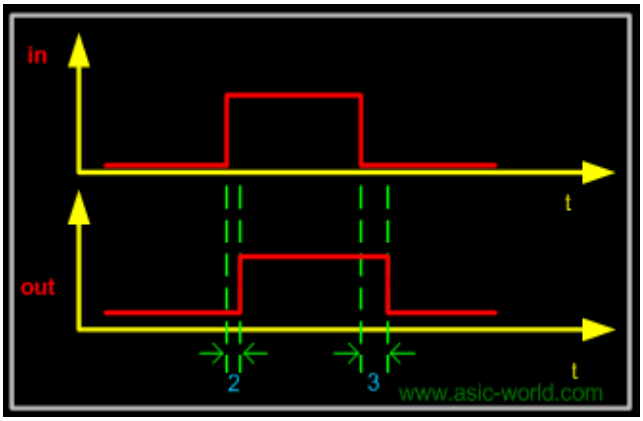
```
Time = 0 in = 0 out=x
Time = 5 in = 0 out=0
Time = 10 in = 1 out=0
Time = 15 in = 1 out=1
Time = 20 in = 0 out=1
Time = 25 in = 0 out=0
```



Örnek- İkili Gecikme(Two Delays)

```
 1 module  buf_gate1 ();
 2 reg in;
 3 wire out;
 4
 5 buf #(2,3) (out,in);
 6
 7 initial begin
 8   $monitor ("Time = %g in = %b out=%b", $time, in, out);
 9   in = 0;
10    #10  in = 1;
11    #10  in = 0;
12    #10  $finish;
13 end
14
15 endmodule
```

You could download file buf_gate1.v here



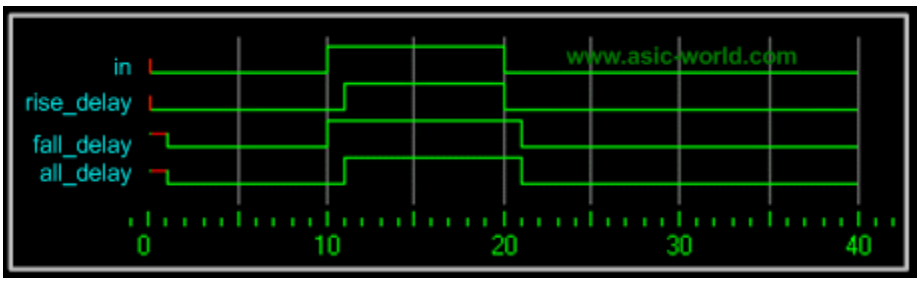

```
Time = 0 in = 0 out=x
Time = 3 in = 0 out=0
Time = 10 in = 1 out=0
Time = 12 in = 1 out=1
Time = 20 in = 0 out=1
Time = 23 in = 0 out=0
```

Örnek- Tüm Gecikmeleler (All Delays)

```
 1 module delay();
 2  reg in;
 3  wire rise_delay, fall_delay, all_delay;
 4
 5  initial begin
 6   $monitor (
 7     "Time=%g in=%b rise_delay=%b fall_delay=%b all_delay=%b",
 8     $time, in, rise_delay, fall_delay, all_delay);
 9   in = 0;
10    #10  in = 1;
11    #10  in = 0;
12    #20  $finish;
13  end
14
15  buf #(1,0)U_rise (rise_delay,in);
16  buf #(0,1)U_fall (fall_delay,in);
17  buf  #1  U_all (all_delay,in);
18
19 endmodule
```

You could download file delay.v here

```
Time = 0 in = 0 rise_delay = 0 fall_delay = x all_delay = x
Time = 1 in = 0 rise_delay = 0 fall_delay = 0 all_delay = 0
Time = 10 in = 1 rise_delay = 0 fall_delay = 1 all_delay = 0
Time = 11 in = 1 rise_delay = 1 fall_delay = 1 all_delay = 1
Time = 20 in = 0 rise_delay = 0 fall_delay = 1 all_delay = 1
Time = 21 in = 0 rise_delay = 0 fall_delay = 0 all_delay = 0
```



Örnek – Karma Örnek

```
 1 module delay_example();
 2
 3 wire out1,out2,out3,out4,out5,out6;
 4 reg b,c;
 5
 6 // Tüm geçişler için gecikme
 7 or      #5                    u_or     (out1,b,c);
 8 // Artan ve azalan gecikme
 9 and    #(1,2)                 u_and    (out2,b,c);
10 // Artan, azalan ve kapama gecikmesi
11 nor    #(1,2,3)               u_nor    (out3,b,c);
12 //Tek Gecikme, min, typ ve max
13 nand   #(1:2:3)               u_nand   (out4,b,c);
14 //İkili gecikmeler, min,typ ve max
15 buf    #(1:4:8,4:5:6)         u_buf    (out5,b);
16 //Üçlü gecikmeler, min, typ, ve max
17 notif1 #(1:2:3,4:5:6,7:8:9) u_notif1 (out6,b,c);
18
19 //Testbenç kodu
20 initial begin
21   $monitor (
22   "Time=%g b=%b c=%b  out1=%b out2=%b out3=%b out4=%b out5=%b out6=%b",
23     $time, b, c , out1, out2, out3, out4, out5, out6);
24   b = 0;
25   c = 0;
26    #10  b = 1;
27    #10  c = 1;
28    #10  b = 0;
29    #10  $finish;
30 end
31
32 endmodule
```

You could download file delay_example.v here

```
Time = 0 b = 0 c=0  out1=x out2=x out3=x out4=x out5=x out6=x
Time = 1 b = 0 c=0  out1=x out2=x out3=1 out4=x out5=x out6=x
Time = 2 b = 0 c=0  out1=x out2=0 out3=1 out4=1 out5=x out6=z
Time = 5 b = 0 c=0  out1=0 out2=0 out3=1 out4=1 out5=0 out6=z
Time = 8 b = 0 c=0  out1=0 out2=0 out3=1 out4=1 out5=0 out6=z
Time = 10 b = 1 c=0  out1=0 out2=0 out3=1 out4=1 out5=0 out6=z
Time = 12 b = 1 c=0  out1=0 out2=0 out3=0 out4=1 out5=0 out6=z
Time = 14 b = 1 c=0  out1=0 out2=0 out3=0 out4=1 out5=1 out6=z
Time = 15 b = 1 c=0  out1=1 out2=0 out3=0 out4=1 out5=1 out6=z
Time = 20 b = 1 c=1  out1=1 out2=0 out3=0 out4=1 out5=1 out6=z
Time = 21 b = 1 c=1  out1=1 out2=1 out3=0 out4=1 out5=1 out6=z
Time = 22 b = 1 c=1  out1=1 out2=1 out3=0 out4=0 out5=1 out6=z
```

```
Time = 25 b = 1 c=1  out1=1 out2=1 out3=0 out4=0 out5=1 out6=0
Time = 30 b = 0 c=1  out1=1 out2=1 out3=0 out4=0 out5=1 out6=0
Time = 32 b = 0 c=1  out1=1 out2=0 out3=0 out4=1 out5=1 out6=1
Time = 35 b = 0 c=1  out1=1 out2=0 out3=0 out4=1 out5=0 out6=1
```

N-Girişli Temeller

AND(VE), NAND(VEDEĞİL), OR(VEYA), NOR(VEYADEĞİL), XOR(DIŞLAMALI YADA), XNOR (DIŞLAMALI YADA DEĞİL) temellerinin bir çıkışı birden çok girişi vardır.

- Tek çıkış ilk terminaldir.
- Tüm diğer terminaller girişlerdir.

Örnek

```
 1 module n_in_primitive();
 2
 3 wire out1,out2,out3;
 4 reg in1,in2,in3,in4;
 5
 6 // Two input AND gate
 7 and u_and1 (out1, in1, in2);
 8 // four input AND gate
 9 and u_and2 (out2, in1, in2, in3, in4);
10 // three input XNOR gate
11 xnor u_xnor1 (out3, in1, in2, in3);
12
13 //Testbench Code
14 initial begin
15   $monitor (
16   "in1 = %b in2 = %b in3 = %b in4 = %b out1 = %b out2 = %b out3 = %b",
17   in1, in2, in3, in4, out1, out2, out3);
18   in1 = 0;
19   in2 = 0;
20   in3 = 0;
21   in4 = 0;
22    #1  in1 = 1;
23    #1  in2 = 1;
24    #1  in3 = 1;
25    #1  in4 = 1;
26    #1  $finish;
27 end
28
29 endmodule
```

You could download file n_in_primitive.v here

```
in1 = 0 in2 = 0 in3 = 0 in4 = 0 out1 = 0 out2 = 0 out3 = 1
in1 = 1 in2 = 0 in3 = 0 in4 = 0 out1 = 0 out2 = 0 out3 = 0
in1 = 1 in2 = 1 in3 = 0 in4 = 0 out1 = 1 out2 = 0 out3 = 1
in1 = 1 in2 = 1 in3 = 1 in4 = 0 out1 = 1 out2 = 0 out3 = 0
in1 = 1 in2 = 1 in3 = 1 in4 = 1 out1 = 1 out2 = 1 out3 = 0
```

N-Çıkışlı Temeller

Buf(buffer-arabellek) ve temel olmayanların(not primitives) birden çok sayıda çıkışı ve bir girişi vardır.

- Çıkışlar listelenen ilk terminallerdir.
- Tek çıkış ise son terminaldir.

Örnek

```
 1 module n_out_primitive();
 2
 3 wire out,out_0,out_1,out_2,out_3,out_a,out_b,out_c;
 4 wire in;
 5
 6 // tek çıkışlı Buffer(arabellek)kapısı
 7 buf u_buf0 (out,in);
 8 // dört çıkışlı Buffer(arabellek)kapısı
 9 buf u_buf1 (out_0, out_1, out_2, out_3, in);
10 // üç çıkışlı Invertor(evirici) kapısı
11 not u_not0 (out_a, out_b, out_c, in);
12
13 endmodule
```

You could download file n_out_primitive.v here

Kullanıcı Tanımlı Temeller(User Defined Primitives-UDP)

Giriş

Verilog kapılar, iletim kapıları ve anahtarlar gibi temeller üzerine kurulmuştur. Bunlar çok az sayıda temel yapılardır; eğer daha çok kompleks temellere ihtiyacımız varsa, Verilog UDP yi, ya da basitçe Kullanıcı Tanımlı Temelleri(User Defined Primitives) destekler. UDP kullanrak şunları modelleyebiliriz:

- Kombinasyonel-Bileşimli Lojik(Combinational Logic)
- Ardışıl-Sıralı Lojik(Sequential Logic)

Bu UDP'lere tam ASIC kütüphanesi modelleri modellemek için zamanlama bilgilerini de ekleyebiliriz.

Sözdizimi(Syntax)

UDP tahsis edilmiş kelime **primitive** ile başlar ve **endprimitive** ile biter. Bunları portlar/terminaller takip eder. Bu bizim modül tnaımlamak için yaptığımızın aynısıdır. UDP'ler **module** ve **endmodule** bloğunun dışında da tanımlanabilir.

```
 1 //Bu kod giriş/çıkış(input/output) portlarının
 2 // ve temellerin(primitve) nasıl bildirildiğini göstermektedir
 3 primitive udp_syntax (
 4 a, // Port a
 5 b, // Port b
 6 c, // Port c
 7 d  // Port d
 8 );
 9 output a;
10 input b,c,d;
11
12 // UDP fonksiyon kodu buradan itibaren başlamaktadır
13
14 endprimitive
```

You could download file udp_syntax.v here

Yukarıdaki kodda, udp_syntax temel ismidir, ve port a,b,c,d 'yi içerir.

UDP tanımlamanın resmi sözdizimi şöyle devam eder:

```
<UDP>
   ::= primitive <name_of_UDP> ( <output_terminal_name>,
      <input_terminal_name> <,<input_terminal_name>>* ) ;
   <UDP_declaration>+
   <UDP_initial_statement>?
   <table_definition>
   endprimitive

<name_of_UDP>
   ::= <IDENTIFIER>

<UDP_declaration>
   ::= <UDP_output_declaration>
   ||= <reg_declaration>
   ||= <UDP_input_declaration>

<UDP_output_declaration>
   ::= output <output_terminal _name>;
<reg_declaration>
   ::=   reg <output_terminal_name> ;

<UDP_input_declaration>
   ::= input <input_terminal_name> <,<input_terminal_name>>* ;

<UDP_initial_statement>
   ::= initial <output_terminal_name> = <init_val> ;

<init_val>
   ::= 1'b0
   ||= 1'b1
   ||= 1'bx
   ||= 1
```

```
   ||= 0

<table_definition>
   ::= table
         <table_entries>
      endtable

<table_entries>
   ::= <combinational_entry>+
   ||= <sequential_entry>+

<combinational_entry>
   ::= <level_input_list> : <OUTPUT_SYMBOL> ;

<sequential_entry>
   ::= <input_list> : <state> : <next_state> ;

<input_list>
   ::= <level_input_list>
   ||= <edge_input_list>

<level_input_list>
   ::= <LEVEL_SYMBOL>+

<edge_input_list>
   ::= <LEVEL_SYMBOL>* <edge> <LEVEL_SYMBOL>*

<edge>
   ::= ( <LEVEL_SYMBOL> <LEVEL_SYMBOL> )
   ||= <EDGE_SYMBOL>

<state>
   ::= <LEVEL_SYMBOL>

<next_state>
   ::= <OUTPUT_SYMBOL>
   ||= -
```

UDP port kuralları

- Bir UDP bir çıkış ve 10 taneye kadar arttırılabilir giriş içerebilir.
- Çıkış portları ilk port olmalıdır bunları bir veya birden fazla giriş portu takip edebilir.
- Tüm UDP portları skaler(sayıl) büyüklüklerdir, Vektör portlarına izin verilmez.
- UDP'ler çiftyönlü portlara sahip olamaz.
- Bir ardışıl UDP'nin çıkış terminalinin ek bir **reg** tipi bildirime ihtiyacı vardır.
- Kombinasyonel bir UDP'nin çıkış terminali için bir **reg**(yazmaç) tanımlamak yasaktır.

Gövde(Body)

Hem kombinasyonel hem sıralı(ardışıl) temellerin fonksiyonalitesi bir tablonun içinde tanımlıdır ve kodda olduğu gibi ayrılmış kelime '**endtable**' ile biter. Ardışıl UDP için, çıkışa bir başlangıç değeri vermek için **initial**(ilk değer) kullanabiliriz.

```
1 // Bu kod UDP gövdesinin nasıl göründüğünü göstermek içindir
2 primitive udp_body (
3 a, // Port a
4 b, // Port b
```

```
 5 c  // Port c
 6 );
 7 output a;
 8 input b,c;
 9
10 // UDP fonksiyonu buradan başlar
11 // A = B | C;
12 table
13  // B  C    : A
14     ?  1    : 1;
15     1  ?    : 1;
16     0  0    : 0;
17 endtable
18
19 endprimitive
```

You could download file udp_body.v here

Not: Bir UDP 'z' ‘yi giriş tablosunda kullanamaz

Verilen UDP için Testbenç

```
 1 `include "udp_body.v"
 2 module udp_body_tb();
 3
 4 reg b,c;
 5 wire a;
 6
 7 udp_body udp (a,b,c);
 8
 9 initial begin
10   $monitor(" B = %b C = %b  A = %b",b,c,a);
11   b = 0;
12   c = 0;
13    #1  b = 1;
14    #1  b = 0;
15    #1  c = 1;
16    #1  b = 1'bx;
17    #1  c = 0;
18    #1  b = 1;
19    #1  c = 1'bx;
20    #1  b = 0;
21    #1  $finish;
22 end
23
24 endmodule
```

You could download file udp_body_tb.v here

Simülatör Çıktısı

```
B = 0 C = 0  A = 0
B = 1 C = 0  A = 1
B = 0 C = 0  A = 0
B = 0 C = 1  A = 1
B = x C = 1  A = 1
B = x C = 0  A = x
B = 1 C = 0  A = 1
B = 1 C = x  A = 1
B = 0 C = x  A = x
```

Tablo(Table)

Tablo UDP'nin fonksiyonunu tanımlamak için kullanılır. Verilog ayrılmış kelimesi **table** tablonun başlangıcını belirtir ve **endtable** 'da tablonun sonunu belirtir.

Tablo içindeki herbir satır bir koşuldur; bir giriş değiştiğinde, giriş koşulu ile eşleştirilir ve çıkış, girişteki bu yeni değişikliğe tepki vermek için girişi ölçer.

Başlangıç Değeri(Initial)

Başlangıç(initial) ifadesi ardışıl UDP'leri başlatmak için kullanılır. Bu ifade anahtar sözcük '**initial**' ile başlar. Bu ifadeyi takiben tek bit gerçek değeri çıkış terminalindeki reg 'e bir atama ifadesi olmalıdır.

```
 1 primitive udp_initial (a,b,c);
 2 output a;
 3 input b,c;
 4 reg a;
 5 // a has value of 1 at start of sim
 6 initial a = 1'b1;
 7
 8 table
 9 // udp_initial behaviour
10 endtable
11
12 endprimitive
```

You could download file udp_initial.v here

Semboller

UDP fonksiyonları tanımlamak için özel semboller kullanır. Aşağıdaki tablo UDP'de kullanılan sembolleri göstermektedir:

Sembol	Çevirisi	Açıklama
?	0 veya 1 veya X	? anlamı değişken 0 veya 1 veya x olabilir
b	0 veya 1	? ile aynıdır, fakat x dahil değildir
f	(10)	Bir giriş üzerindeki düşen-azalan kenar
r	(01)	Bir giriş üzerindeki artan-yükselen kenar
p	(01) veya (0x) veya (x1) veya (1z) veya (z1)	x ve z 'yi içeren yükselen kenar
n	(10) veya (1x) veya (x0) veya (0z) veya (z0)	x ve z 'yi içeren azalan kenar
*	(??)	Tüm geçişler
-	Değişim yoktur	Değişim Yoktur

Kombinasyonel UDP'ler

Kombinasyonel UDP'de, çıkışa, mevcut girişin bir fonksiyonu olarak karar verilir. Bir girişin değeri değiştiğinde, UDP durum tablosunda eşleşen bir satır değerlendirilir. Çıkış durumu bu satırın gösterdiği değerle atanır. Bu koşul ifadesiyle aynıdır: tablodaki herbir satır bir koşuldur.

Kombinasyonel UDP'lerde giriş için bir alan ve çıkış bir alan vardır. Giriş alanları ve çıkış alanları sütunlarla birbirinden ayrılmıştır. Tablodaki herbir satır bir noktalıvirgül ile sonlandırılır. Örneğin, gösterilen bu durum tablosu girişi eğer girişlerin hepsi 0 ise çıkışın sıfır olacağını belirtmektedir.

```
1 primitive udp_combo (.....);
2
3 table
4 0 0 0 : 0;
5 ...
6 endtable
7
8 endprimitive
```

You could download file udp_combo.v here

Durum tablosundaki girişlerin sırası UDP tanımlama başlığındaki port listesindeki girişlerin sırasına uygun olmalıdır. Bunun giriş bildiriminin sırasıyla bir alakası yoktur.

Tablodaki her bir satır belirli bir giriş durumları kombinasyonu için bir çıkışı tanımlar. Eğer tüm girişler x olarak bildirildiyse, çıkış da x olarak belirtilmeli. Tabloda belirtilmemiş tüm kombinasyonlar için varsayılan çıkış durumu x ile sonuçlanır.

Örnek

Aşağıdaki örnekte giriş, bir önemsiz koşuluyla ? gösterilmektedir. Bu sembol 1,0 ve x 'in tekrarlı çıkarımını belirtmektedir. Tablo girişi bunu şöyle bildirir, girişler 0 ve 1 ise çıkış 1'dir mevcut durumun ne olduğunun bir önemi yoktur.

Tüm olası giriş kombinasyonlarını açıkça belirtmeniz zorunluluğu yoktur. Tüm kombinasyonların açıkça belirtilmemesi varsayılan çıkış durumu x ile sonuçlarnır.

Aynı kombinasyondaki girişler için farklı çıkışların belirtilmesi yasaktır.

```
 1 // Bu kod UDP gövdesinin nasıl göründüğünü göstermektedir
 2 primitive udp_body (
 3 a, // Port a
 4 b, // Port b
 5 c  // Port c
 6 );
 7 output a;
 8 input b,c;
 9
10 // UDP fonksiyonu buradan itibaren başlamaktadır
11 // A = B | C;
12 table
13  // B  C    : A
```

```
14      ?  1    : 1;
15      1  ?    : 1;
16      0  0    : 0;
17  endtable
18
19  endprimitive
```

You could download file udp_body.v here

Üstteki UDP'yi kontrol etmek için TestBenç

```
 1  `include "udp_body.v"
 2  module udp_body_tb();
 3
 4  reg b,c;
 5  wire a;
 6
 7  udp_body udp (a,b,c);
 8
 9  initial begin
10    $monitor(" B = %b C = %b  A = %b",b,c,a);
11    b = 0;
12    c = 0;
13     #1  b = 1;
14     #1  b = 0;
15     #1  c = 1;
16     #1  b = 1'bx;
17     #1  c = 0;
18     #1  b = 1;
19     #1  c = 1'bx;
20     #1  b = 0;
21     #1  $finish;
22  end
23
24  endmodule
```

You could download file udp_body_tb.v here

Simülatör Çıktısı

```
B = 0 C = 0  A = 0
B = 1 C = 0  A = 1
B = 0 C = 0  A = 0
B = 0 C = 1  A = 1
B = x C = 1  A = 1
B = x C = 0  A = x
B = 1 C = 0  A = 1
B = 1 C = x  A = 1
B = 0 C = x  A = x
```

Seviye Hassasiyetli Ardışıl UDP(Level Sensitive Sequential UDP)

Seviye-hassasiyetli ardışıl davranış kombinasyonel davranışla aynı yolla gösterilir, yalnızca çıkış reg tipinde bidirilir, ve herbir tablo girişinde ek bir alan vardır. Bu yeni alan UDP'nin mevcut durumunu gösterir.

- Çıkış, burada bir iç durum olduğundan reg tipinde bildirilir. UDP'nin çıkış değeri herzaman iç durumla aynıdır.
- Şuanki-mevcut durum için ek bir alan eklenmiştir. Bu alan girişlerden ve çıkıştan iki nokta üst üste ile(:) ayrılmıştır.

Ardışıl-sıralı UDP'ler giriş alanları ile çıkış alanı arasına kombinasyonel UDP'yi karşılaştırmak için ek bir alan yerleştirilir. Bu ek alan UDP'nin mevcut durumunu gösterir ve mevcut çıkış değeriyle aynı olduğu düşünülmüştür. İki nokta üst üste ile sınırlandırılmıştır.

```
1 primitive udp_seq (.....);
2
3 table
4 0 0 0 : 0 : 0;
5 ...
6 endtable
7
8 endprimitive
```

You could download file udp_seq.v here

Örnek

```
 1 primitive udp_latch(q, clk, d) ;
 2 output q;
 3 input clk, d;
 4
 5 reg q;
 6
 7 table
 8   //clk d    q     q+
 9   0     1  : ? :   1   ;
10   0     0  : ? :   0   ;
11   1     ?  : ? :   -   ;
12 endtable
13
14 endprimitive
```

You could download file udp_latch.v here

Kenar Hassasiyetli UDP'ler(Edge-Sensitive UDPs)

Seviye hassasiyetli davranışta, girişlerin değerleri ve şuanki değer çıkış değerine karar vermek için yeterliydi. Kenar hassasiyetli davranış farkı ise çıkışdaki değişikli, girişlerin belirli değişimleriyle tetiklenmektedir.

Kombinasyonel ve seviye hassasiyetlide girişlerde olduğu gibi, bir ? 0,1 ve x değerlerini sırayla belirtmektedir. Çıkış kolonundaki bir tire(-) değer değişiminin olmadığını belirtmektedir.

Tüm belirtilmemiş geçişlerin varsayılan çıkış değeri x'dir. Bu nedenle, daha önceki örnekte olduğu gibi, veri 0'a eşitken ve mevcut durum 1 iken saatin 0'dan x'e geçişi q çıkışının x'e gitmesiyle sonuçlanmaktadır.

Tüm geçişler özellikle belirtilmediği sürece çıkışı etkilemeyebilir. Yoksa, bunlar çıkış değerinin x'e değişmesine neden olacaktır. Eğer UDP herhangi bir girişin kenarına hassas ise, arzu edilen çıkış durumu herbir girişin tüm çıkışları için belirtilmek zorundadır.

Örnek

```
 1 primitive udp_sequential(q, clk, d);
 2 output q;
 3 input clk, d;
 4
 5 reg q;
 6
 7 table
 8 // saatin yükselen kenarındaki elde edilen çıkış
 9 // clk(saat)    d       q       q+
10    (01)         0   :   ?   :   0   ;
11    (01)         1   :   ?   :   1   ;
12    (0?)         1   :   1   :   1   ;
13    (0?)         0   :   0   :   0   ;
14 // saatin negatif kenarını önemsemeyin
15    (?0)         ?   :   ?   :   -   ;
16 // sabit durumlu saat için d değişikliklerini önemsemeyin
17    ?      (??)      :   ?   :   -   ;
18  endtable
19
20 endprimitive
```

You could download file udp_sequential.v here

UDP'ye ilk değer verilerek örnek

```
 1 primitive udp_sequential_initial(q, clk, d);
 2 output q;
 3 input clk, d;
 4
 5 reg q;
 6
 7 initial begin
 8   q = 0;
 9 end
10
11 table
12 // saatin yükselen kenarındaki elde edilen çıkış
13 // clk         d        q       q+
14    (01)         0   :   ?   :   0   ;
15    (01)         1   :   ?   :   1   ;
16    (0?)         1   :   1   :   1   ;
17    (0?)         0   :   0   :   0   ;
18 // saatin negatif kenarını önemsemeyin
19    (?0)         ?   :   ?   :   -   ;
20 // sabit durumlu saat için d değişikliklerini önemsemeyin
21    ?      (??)      :   ?   :   -   ;
22  endtable
23
24 endprimitive
```

You could download file udp_sequential_initial.v here

Verilog Operatörleri

Aritmetik Operatörler(Arithmetic Operators)

- İkili: +, -, *, /, % (modül(mod) operatörü)
- Tekli: +, - (işareti belirtmek için kullanılır)
- Tamsayıyı bölmede virgüllü kısım atılır
- Modül-mod işleminin sonucu(modulus operation) ilk operandın işaretini alır
- Eğer herhangi bir operand bit değeri bilinmeyen değer x ise, bunun sonucundaki alacak değer de x'dir.
- Yazmaç(Register) veri tipi işaretsiz değerler olarak kullanılır(negatif sayılar 2'ye tümleyen şeklinde tutulur)

Örnek

```
 1 module arithmetic_operators();
 2
 3 initial begin
 4   $display (" 5  +  10 = %d", 5  + 10);
 5   $display (" 5  -  10 = %d", 5  - 10);
 6   $display (" 10 -  5  = %d", 10 - 5);
 7   $display (" 10 *  5  = %d", 10 * 5);
 8   $display (" 10 /  5  = %d", 10 / 5);
 9   $display (" 10 /  -5 = %d", 10 / -5);
10   $display (" 10 %s  3  = %d","%", 10 % 3);
11   $display (" +5       = %d", +5);
12   $display (" -5       = %d", -5);
13    #10  $finish;
14 end
15
16 endmodule
```

You could download file arithmetic_operators.v here

```
5  +  10 =  15
5  -  10 =  -5
10 -  5  =   5
10 *  5  =  50
10 /  5  =  2
10 /  -5 = -2
10 %  3  =   1
+5       =  5
-5       =  -5
```

İlişkisel Operatörler

Operatör	Açıklama
a < b	a küçüktür b
a > b	a büyüktür b
a <= b	a küçük veya eşittir b
a >= b	a büyük veya eşittir b

- Sonuç skaler bir değerdir(örneğin a < b için)
- Eğer ilişki yanlışsa 0 (a büyüktür b ise)
- Eğer ilişki doğruysa 1 (a küçüktür b ise)
- Eğer bir operandın bilinemyen x bitleri varsa sonuç x'dir (eğer a veya b X içeriyorsa)

Not: Eğer bir operand x veya z ise bu testin sonucu yanlışmış(0) gibi ele alınır.

Örnek

```
 1 module relational_operators();
 2
 3 initial begin
 4   $display (" 5     <=  10 = %b", (5     <= 10));
 5   $display (" 5     >=  10 = %b", (5     >= 10));
 6   $display (" 1'bx  <=  10 = %b", (1'bx  <= 10));
 7   $display (" 1'bz  <=  10 = %b", (1'bz  <= 10));
 8    #10  $finish;
 9 end
10
11 endmodule
```

You could download file relational_operators.v here

```
 5     <=  10 = 1
 5     >=  10 = 0
1'bx   <=  10 = x
1'bz   <=  10 = x
```

Eşitlik Operatörleri

İki türlü eşitlik operatörü vardır. Durum eşitliği(case equality) ve lojik-mantıksal eşitlik(logical equality).

Operatör	Açıklama
a === b	a b'ye eşittir, x ve z'yi de içerir (durum eşitliği(Case equality))
a !== b	a b'ye eşit değildir, x ve z'yi de içerir (durum eşitliği(Case equality))
a == b	a b 'ye eşittir, sonuç bilinmeyen olabilir(mantıksal eşitlik(logical equality))
a != b	a b 'ye eşit değildir, sonuç bilinmeyen olabilir(mantıksal eşitlik(logical equality))

- Operandlar bit bir karşılaştırılır, eğer iki operand farklı uzunluktaysa sıfır ile doldurularak uzunlukları eşitlenir
- Sonuç ya 0 (yanlış-false) ya da 1 (doğru-true) 'dir.
- == ve != operatörleri için, eğer operandlar x veya bir z içeriyorsa sonu da x'dir.
- === ve !== operatörleri için, bitler x ve z karşılaştırmaya eklenir ve doğru olabilmesi için birbiriyle eşleşmeli.

Not : Sonuç her zaman 0 yada 1'dir.

Örnek

```
 1 module equality_operators();
 2
 3 initial begin
 4   // Durum eşitliği(Case Equality)
 5   $display (" 4'bx001 ===  4'bx001 = %b", (4'bx001 ===  4'bx001));
 6   $display (" 4'bx0x1 ===  4'bx001 = %b", (4'bx0x1 ===  4'bx001));
 7   $display (" 4'bz0x1 ===  4'bz0x1 = %b", (4'bz0x1 ===  4'bz0x1));
 8   $display (" 4'bz0x1 ===  4'bz001 = %b", (4'bz0x1 ===  4'bz001));
 9   // Durum eşitsizliği (Case Inequality)
10   $display (" 4'bx0x1 !==  4'bx001 = %b", (4'bx0x1  ! ==  4'bx001));
11   $display (" 4'bz0x1 !==  4'bz001 = %b", (4'bz0x1  ! ==  4'bz001));
12   // Lojik eşitlik(Logical Equality)
13   $display (" 5       ==   10      = %b", (5       ==   10));
14   $display (" 5       ==   5       = %b", (5       ==   5));
15   // Lojik eşitsizlik(Logical Inequality)
16   $display (" 5       !=   5       = %b", (5        ! =   5));
17   $display (" 5       !=   6       = %b", (5        ! =   6));
18    #10  $finish;
19 end
20
21 endmodule
```

You could download file equality_operators.v here

```
4'bx001 ===  4'bx001 = 1
4'bx0x1 ===  4'bx001 = 0
4'bz0x1 ===  4'bz0x1 = 1
4'bz0x1 ===  4'bz001 = 0
4'bx0x1 !==  4'bx001 = 1
4'bz0x1 !==  4'bz001 = 1
5       ==   10      = 0
5       ==   5       = 1
5       !=   5       = 0
5       !=   6       = 1
```

Lojik-Mantıksal Operatörler (**Logical Operators**)

Operatör	Açıklama
!	Lojik olumsuzlama (logic negation)
&&	Lojik-mantıksal VE(and)
||	Lojik-mantıksal VEYA(or)

- && ve || ile bağlantılı ifadeler soldan sağa değerlendirilir.
- Değerlendirme sonuç bilinene kadar devam eder
- Sonuç skaler bir değerdir:
  - eğer ilişki yanlışsa -> 0
  - eğer ilişki doğruysa -> 1
  - eğer operandlar x(bilinmeyen-unknown) bit değerlerine sahipse -> x

Örnek

```
 1 module logical_operators();
 2
 3 initial begin
 4   // Lojik AND(VE)
 5   $display ("1'b1 && 1'b1 = %b", (1'b1 && 1'b1));
 6   $display ("1'b1 && 1'b0 = %b", (1'b1 && 1'b0));
 7   $display ("1'b1 && 1'bx = %b", (1'b1 && 1'bx));
 8   // Lojik OR(VEYA)
 9   $display ("1'b1 || 1'b0 = %b", (1'b1 || 1'b0));
10   $display ("1'b0 || 1'b0 = %b", (1'b0 || 1'b0));
11   $display ("1'b0 || 1'bx = %b", (1'b0 || 1'bx));
12   // Lojik O>lumsuzlama(Negation)
13   $display ("! 1'b1       = %b", ( !   1'b1));
14   $display ("! 1'b0       = %b", ( !   1'b0));
15    #10  $finish;
16 end
17
18 endmodule
```

You could download file logical_operators.v here

```
1'b1 && 1'b1 = 1
1'b1 && 1'b0 = 0
1'b1 && 1'bx = x
1'b1 || 1'b0 = 1
1'b0 || 1'b0 = 0
1'b0 || 1'bx = x
! 1'b1       = 0
! 1'b0       = 1
```

Bit Bit Operatörler(Bit-wise Operators)

Bit bit(Bitwise) operatörler iki operand arasında bit bit yol alır. Bir operanddaki herbir bit için diğer operanddaki uygun bit ile işlem yapar. Eğer bir operand diğerinden kısaysa, kısa olan operandın soluna sıfır eklenerek boyları eşitlenir.

Operatör	Açıklama
~	Olumsuzlama-Tersini alma(Negation)
&	Ve(And)
|	Kapsamlı veya(inclusive or)
^	Dışlayan veya(exclusive or-xor)
^~ or ~^	Dışlayan veya değil (exclusive nor (eşitlik))

- Bilinmeyen bitler içeren hesaplamalar aşağıdaki şekilde yapılır:
- -> ~x = x
- -> 0&x = 0
- -> 1&x = x&x = x
- -> 1|x = 1
- -> 0|x = x|x = x
- -> 0^x = 1^x = x^x = x
- -> 0^~x = 1^~x = x^~x = x

- Eğer operandların bit uzunlukları birbirine eşit değilse, daha kısa olanın soluna sıfır eklenir.

Örnek

```
 1 module bitwise_operators();
 2
 3 initial begin
 4   // Bit Wise Negation
 5   $display (" ~4'b0001           = %b", (~4'b0001));
 6   $display (" ~4'bx001           = %b", (~4'bx001));
 7   $display (" ~4'bz001           = %b", (~4'bz001));
 8   // Bit Wise AND
 9   $display (" 4'b0001 &  4'b1001 = %b", (4'b0001 &  4'b1001));
10   $display (" 4'b1001 &  4'bx001 = %b", (4'b1001 &  4'bx001));
11   $display (" 4'b1001 &  4'bz001 = %b", (4'b1001 &  4'bz001));
12   // Bit Wise OR
13   $display (" 4'b0001 |  4'b1001 = %b", (4'b0001 |  4'b1001));
14   $display (" 4'b0001 |  4'bx001 = %b", (4'b0001 |  4'bx001));
15   $display (" 4'b0001 |  4'bz001 = %b", (4'b0001 |  4'bz001));
16   // Bit Wise XOR
17   $display (" 4'b0001 ^  4'b1001 = %b", (4'b0001 ^  4'b1001));
18   $display (" 4'b0001 ^  4'bx001 = %b", (4'b0001 ^  4'bx001));
19   $display (" 4'b0001 ^  4'bz001 = %b", (4'b0001 ^  4'bz001));
20   // Bit Wise XNOR
21   $display (" 4'b0001 ~^ 4'b1001 = %b", (4'b0001 ~^ 4'b1001));
22   $display (" 4'b0001 ~^ 4'bx001 = %b", (4'b0001 ~^ 4'bx001));
23   $display (" 4'b0001 ~^ 4'bz001 = %b", (4'b0001 ~^ 4'bz001));
24    #10  $finish;
25 end
26
27 endmodule
```

You could download file bitwise_operators.v here

```
 ~4'b0001           = 1110
 ~4'bx001           = x110
 ~4'bz001           = x110
 4'b0001 &  4'b1001 = 0001
 4'b1001 &  4'bx001 = x001
 4'b1001 &  4'bz001 = x001
 4'b0001 |  4'b1001 = 1001
 4'b0001 |  4'bx001 = x001
 4'b0001 |  4'bz001 = x001
 4'b0001 ^  4'b1001 = 1000
 4'b0001 ^  4'bx001 = x000
 4'b0001 ^  4'bz001 = x000
 4'b0001 ~^ 4'b1001 = 0111
 4'b0001 ~^ 4'bx001 = x111
 4'b0001 ~^ 4'bz001 = x111
```

İndirgeme Operatörleri(Reduction Operators)

Operatör	Açıklama
&	Ve(and)
~&	Vedeğil(nand)
|	Veya(or)
~|	Veyadeğil(nor)
^	Dışlayan veya(xor)
^~ or ~^	Dışlayan veya değil(xnor)

- İndirgeme operatörleri teklidir.
- Bir operand üzerinde bit bit(bit-wise) işlem yapar ve tek bitlik sonuç üretir.
- Tekli indirgeme NAND veNOR operatörleri AND ve OR ile aynı şekilde işlem yapar ancak sonucun tersini alır.
- -> Bilinmeyen bitler daha önce belirtildiği gibi işlem görür.

Örnek

```
 1 module reduction_operators();
 2
 3 initial begin
 4   // Bit Wise AND reduction
 5   $display (" &  4'b1001 = %b", (&  4'b1001));
 6   $display (" &  4'bx111 = %b", (&  4'bx111));
 7   $display (" &  4'bz111 = %b", (&  4'bz111));
 8   // Bit Wise NAND reduction
 9   $display (" ~& 4'b1001 = %b", (~& 4'b1001));
10   $display (" ~& 4'bx001 = %b", (~& 4'bx001));
11   $display (" ~& 4'bz001 = %b", (~& 4'bz001));
12   // Bit Wise OR reduction
13   $display (" |  4'b1001 = %b", (|  4'b1001));
14   $display (" |  4'bx000 = %b", (|  4'bx000));
15   $display (" |  4'bz000 = %b", (|  4'bz000));
16   // Bit Wise OR reduction
17   $display (" ~| 4'b1001 = %b", (~| 4'b1001));
18   $display (" ~| 4'bx001 = %b", (~| 4'bx001));
19   $display (" ~| 4'bz001 = %b", (~| 4'bz001));
20   // Bit Wise XOR reduction
21   $display (" ^  4'b1001 = %b", (^  4'b1001));
22   $display (" ^  4'bx001 = %b", (^  4'bx001));
23   $display (" ^  4'bz001 = %b", (^  4'bz001));
24   // Bit Wise XNOR
25   $display (" ~^ 4'b1001 = %b", (~^ 4'b1001));
26   $display (" ~^ 4'bx001 = %b", (~^ 4'bx001));
27   $display (" ~^ 4'bz001 = %b", (~^ 4'bz001));
28    #10  $finish;
29 end
30
31 endmodule
```

You could download file reduction_operators.v here

```
&  4'b1001 = 0
&  4'bx111 = x
&  4'bz111 = x
~& 4'b1001 = 1
~& 4'bx001 = 1
~& 4'bz001 = 1
|  4'b1001 = 1
|  4'bx000 = x
|  4'bz000 = x
~| 4'b1001 = 0
~| 4'bx001 = 0
~| 4'bz001 = 0
^  4'b1001 = 0
^  4'bx001 = x
^  4'bz001 = x
~^ 4'b1001 = 1
~^ 4'bx001 = x
~^ 4'bz001 = x
```

Kaydırma Operatörleri (Shift Operators)

Operatör	Açıklama
<<	Sola kaydırma
>>	Sağa kaydırma

- Soldaki operand sağdaki operandda belirtilen sayıda bit pozisyonunda kaydırılır.
- Boşalan bit pozisyonları sıfır ile doldurulur.

Örnek

```
 1 module shift_operators();
 2
 3 initial begin
 4   // Sola kaydırma
 5   $display (" 4'b1001 << 1 = %b", (4'b1001 << 1));
 6   $display (" 4'b10x1 << 1 = %b", (4'b10x1 << 1));
 7   $display (" 4'b10z1 << 1 = %b", (4'b10z1 << 1));
 8   // Sağa kaydırma
 9   $display (" 4'b1001 >> 1 = %b", (4'b1001 >> 1));
10   $display (" 4'b10x1 >> 1 = %b", (4'b10x1 >> 1));
11   $display (" 4'b10z1 >> 1 = %b", (4'b10z1 >> 1));
12    #10  $finish;
13 end
14
15 endmodule
```

You could download file shift_operators.v here

```
4'b1001 << 1 = 0010
4'b10x1 << 1 = 0x10
4'b10z1 << 1 = 0z10
4'b1001 >> 1 = 0100
4'b10x1 >> 1 = 010x
4'b10z1 >> 1 = 010z
```

Birbirine Bağlama-Birleştirme Operatörü (Concatenation Operator)

- Birleştirme ifadesi küme parantezleri arasında { ve }belirtilir ve ifadeler virgül ”,” ile birbirinden ayrılır.
- ->Örnek: + {a, b[3:0], c, 4'b1001} // eğer a ve c 8-bit ise sonuç 24 bittir.
- Boyutlandırılmamış sabit sayıların birlşetirilmesine izin verilmez.

Örnek

```
1 module concatenation_operator();
2
3 initial begin
4   // birleştirme(concatenation)
5   $display (" {4'b1001,4'b10x1}  = %b", {4'b1001,4'b10x1});
6    #10  $finish;
7 end
8
9 endmodule
```

You could download file concatenation_operator.v here

```
{4'b1001,4'b10x1}  = 100110x1
```

Çoğaltma-Kopyalama (Replication) Operatörü

Kopyalama operatörü, bir grup biti n kez kopyalamak için kullanılır. Örneğin 4 bitlik bir değişkeniniz var ve siz bımı 4 kez kopyalayıp 16 bitlik bir değişken oluşturmak istiyorsunuz, işte bu durumda kopyalama(replication) operatörünü kullanabiliriz.

Operatör	Açıklama
{n{m}}	m değerini, n kez kopyala

- Tekrarlama çarpanı (sabit olmalı) şu şekilde kullanılabilir:
- -> {3{a}} // bu {a, a, a} ya eşittir
- İç içe kopyalama ve bağlama operatörleri mümkündür:
- -> {b, {3{c, d}}} // bu {b, c, d, c, d, c, d}’ye eşittir.

Örnek

```
1 module replication_operator();
2
3 initial begin
4   // replication
5   $display (" {4{4'b1001}}      = %b", {4{4'b1001}});
6   // replication and concatenation
7   $display (" {4{4'b1001,1'bz}} = %b", {4{4'b1001,1'bz}});
8    #10  $finish;
9 end
10
11 endmodule
```

You could download file replication_operator.v here

```
{4{4'b1001}       = 1001100110011001
{4{4'b1001,1'bz}  = 1001z1001z1001z1001z
```

Koşullu Operatörler(Conditional Operators)

- Koşul operatörü aşağıdaki C-benzeri formattadır:
- ->koşul_ifadesi ? doğru_ifade : yanlış_ifade
- Doğru_ifade veya yanlış_ifade koşul_ifadesinin sonucuna göre(doğru yada yanlış olmasına göre) gerçekleştirilir.

Örnek

```
 1 module conditional_operator();
 2
 3 wire out;
 4 reg enable,data;
 5 // Üç durumlu arabellek
 6 assign out = (enable) ? data : 1'bz;
 7
 8 initial begin
 9   $display ("time\t enable data out");
10   $monitor ("%g\t %b      %b    %b",$time,enable,data,out);
11   enable = 0;
12   data = 0;
13    #1  data = 1;
14    #1  data = 0;
15    #1  enable = 1;
16    #1  data = 1;
17    #1  data = 0;
18    #1  enable = 0;
19    #10  $finish;
20 end
21
22 endmodule
```

You could download file conditional_operator.v here

```
time    enable data out
0       0      0    z
1       0      1    z
2       0      0    z
3       1      0    0
4       1      1    1
5       1      0    0
6       0      0    z
```

Operatör Önceliği

Operator	Semboller
Birli(Unary), Çarpım(Multiply), Bölüm(Divide), Modülü(Modulus)	!, ~, *, /, %
Toplama(Add), Çıkarma(Subtract), Kaydırma(Shift)	+, - , <<, >>
İlişli(Relation), Eşitlik(Equality)	<,>,<=,>=,==,!=,===,!==
İndirgeme(Reduction)	&, !&,^,^~,|,~|
Lojik-mantık(Logic)	&&, ||
Koşul(Conditional)	? :

Verilog Davranışsal Modelleme(Behavioral Modeling)

Verilog HDL Soyutlama Seviyeleri(Abstraction Levels)

- Davranışsal Modeller(Behavioral Models) :lojiğin davranışının modellendiği daha yüksek seviyeli modellemedir.
- RTL Modeller: Lojik yazmaç(register) seviyesibde modellenir.
- Yapısal Modeller(Structural Models) : Lojik hem yazmaç seviyesinde hem de kapı seviyesinde modellenir.

Prosedürel Bloklar

Verilog davranışsal kodu prosedür bloğunun içindedir, fakat bir istisna vardır: bazı davranışsal kodlar ayrıca prosedür bloğunun dışında da olabilir. Bunu işlemi gerçekleştirirken daha detaylı göreceğiz.

Verilog'da iki tip prosedürel blok vardır:

- **Başlangıç(initial) :** başlangıç bloğu sadece sıfır zamanında gerçekleştirilir.(başlangıç gerçekleştirmesi sıfır zamanında olur)
- **Herzaman(always) :** herzaman(always) bloğu döngüsü tekrar tekrar gerçekleştirilebilir; diğer bir şekilde ifade edecek olursak, herzaman gerçekleştirilir.

Örnek – başlangıç(initial)

```
 1 module initial_example();
 2 reg clk,reset,enable,data;
 3
 4 initial begin
 5  clk = 0;
 6  reset = 0;
 7  enable = 0;
 8  data = 0;
 9 end
10
11 endmodule
```

You could download file initial_example.v here

Yukarıdaki başlangıç bloğu gerçekleştirilmesi ve herzaman(always) bloğu gerçekleştirilmesi sıfır zamanında başlar. Always bloğu burada saatin pozitif kenarı olayını bekler ancak initial bloğu beklemeksizin begin ve end ifadeleri arasındaki tüm ifadeleri gerçekleştirilir.

Example – herzaman(always)

```
 1 module always_example();
 2 reg clk,reset,enable,q_in,data;
 3
 4 always @ (posedge clk)
 5 if (reset)  begin
 6    data <= 0;
 7 end else if (enable) begin
 8    data <= q_in;
 9 end
```

```
10
11 endmodule
```

You could download file always_example.v here

Bir always bloğunda tetikleyici olay gerçekleştiğinde begin ve end arasındaki kod gerçekleştirilir; daha sonra always bloğu bir sonraki tetikleme olayı gerçekleşene kadar bekler. Bu bekleme süreci ve olay esnasındaki gerçekleştirme simülasyon sonlandırılana kadar tekrarlanır.

Prosedürel Atama İfadeleri

- Prosedürel atama ifadeleri değerleri yazmaçlara(reg-register),tamsayı(integer), real(reel sayı) yada zaman değişkenlerine atar ve net(wire(tel) veri tiplerine)'lere atama yapamaz.
- Bir yazmaca(reg veri tipine), bir net'in(wire-tel), sabitin(constant), başka bir yazmacın veya özel bir değerin değerini atayabilirsiniz.

Örnek – Kötü prosedürel atama örneği

```
 1 module initial_bad();
 2 reg clk,reset;
 3 wire enable,data;
 4
 5 initial begin
 6  clk = 0;
 7  reset = 0;
 8  enable = 0;
 9  data = 0;
10 end
11
12 endmodule
```

You could download file initial_bad.v here

Örnek- İyi prosedürel atama

```
 1 module initial_good();
 2 reg clk,reset,enable,data;
 3
 4 initial begin
 5  clk = 0;
 6  reset = 0;
 7  enable = 0;
 8  data = 0;
 9 end
10
11 endmodule
```

You could download file initial_good.v here

Prosedürel Atama Grupları

Eğer bir prosedür bloğu bir ifadeden fazla ifade içeriyorsa, bu ifadeler şunların içine koyulmalı:

- Sıralı **begin - end** bloğu
- Paralel **fork - join** bloğu

Begin-end kullanılırken buna grup adını verebiliriz. Buna **isimlendirilmiş bloklar**(**named blocks**) denir.

Örnek - "begin-end"

```
 1 module initial_begin_end();
 2 reg clk,reset,enable,data;
 3
 4 initial begin
 5  $monitor(
 6    "%g clk=%b reset=%b enable=%b data=%b",
 7    $time, clk, reset, enable, data);
 8   #1   clk = 0;
 9   #10  reset = 0;
10   #5   enable = 0;
11   #3   data = 0;
12   #1  $finish;
13 end
14
15 endmodule
```

You could download file initial_begin_end.v here

Begin : clk(saat) bir birim zamandan sonra 0 değerini alır, reset(sıfırlama) 11 birim zamandan sonra 0 değerini alır, enable(etkin) 16 birim zamandan sonra, data(veri) 19 birimden sonra 0 değerini alır. Tüm ifadeler sıralı-ardışıl olarak gerçekleştirilir.

Simülatör Çıktısı

```
0 clk=x reset=x enable=x data=x
1 clk=0 reset=x enable=x data=x
11 clk=0 reset=0 enable=x data=x
16 clk=0 reset=0 enable=0 data=x
19 clk=0 reset=0 enable=0 data=0
```

Örnek-"fork-join"

```
 1 module initial_fork_join();
 2 reg clk,reset,enable,data;
 3
 4 initial begin
 5  $monitor("%g clk=%b reset=%b enable=%b data=%b",
 6    $time, clk, reset, enable, data);
 7  fork
 8     #1   clk = 0;
 9     #10  reset = 0;
10     #5   enable = 0;
11     #3   data = 0;
12  join
```

```
13   #1  $display ("%g Terminating simulation", $time);
14  $finish;
15 end
16
17 endmodule
```

You could download file initial_fork_join.v here

Fork : (Çatallama) clk 1 birim zamandan sonra, reset 10 birim zamandan sonra, enable 5 birim zamandan sonra, data 3 birim zamandan sonra değerini alır. Tüm ifadeler paralel olarak gerçekleştirilir.

Simülatör Çıktısı

```
0 clk=x reset=x enable=x data=x
1 clk=0 reset=x enable=x data=x
3 clk=0 reset=x enable=x data=0
5 clk=0 reset=x enable=0 data=0
10 clk=0 reset=0 enable=0 data=0
11 Terminating simulation
```

Sıralı İfade Grupları

begin - end anahtar sözcükleri:

- Birçok ifadeyi biraraya toplar.
- İfadeler sıralı-ardışıl olarak gerçekleştirilir.
- -> ardışıl gruplar içindeki herhangi bir zamanlama bir önceki ifadeye aittir(ilişkilidir).
- -> Gecikmeler sırada biriktirilir(herbir gecikme bir önceki gecikmeye eklenir)
- -> Blok, bloktaki son ifade bittiğinde biter.

Örnek- ardışıl(sequential)

```
 1 module sequential();
 2
 3 reg a;
 4
 5 initial begin
 6   $monitor ("%g a = %b", $time, a);
 7    #10  a = 0;
 8    #11  a = 1;
 9    #12  a = 0;
10    #13  a = 1;
11    #14  $finish;
12 end
13
14 endmodule
```

You could download file sequential.v here

Simülatör Çıktısı

```
0 a = x
10 a = 0
21 a = 1
33 a = 0
46 a = 1
```

Paralel İfade Grupları

fork - join anahtar sözcükleri:

- Birçok ifadeyi biraraya toplar.Group several statements together.
- Tüm ifadeler paralel olarak(hepsi aynı zamanda) gerçekleştirilir.
- -> Pararlel grup içindeki zamanlama grubun başında salttır.
- -> Blok, son ifade tamamlandığında biter.(en büyük gecikme ifadesi, bloktaki ilk ifade olabilir).

Örnek - Paralel

```
 1 module parallel();
 2
 3 reg a;
 4
 5 initial
 6 fork
 7   $monitor ("%g a = %b", $time, a);
 8    #10  a = 0;
 9    #11  a = 1;
10    #12  a = 0;
11    #13  a = 1;
12    #14  $finish;
13 join
14
15 endmodule
```

You could download file parallel.v here

Simülatör Çıktısı

```
0 a = x
10 a = 0
11 a = 1
12 a = 0
13 a = 1
```

Örnek- "begin-end" ve "fork - join" karışık kullanımı

```
 1 module fork_join();
 2
 3 reg clk,reset,enable,data;
 4
 5 initial  begin
 6   $display ("Starting simulation");
 7   $monitor("%g clk=%b reset=%b enable=%b data=%b",
 8     $time, clk, reset, enable, data);
 9   fork : FORK_VAL
10      #1  clk = 0;
11      #5  reset = 0;
12      #5  enable = 0;
13      #2  data = 0;
14   join
15    #10  $display ("%g Terminating simulation", $time);
16   $finish;
17 end
18
```

```
19 endmodule
```

You could download file fork_join.v here

Simülatör Çıktısı

```
0 clk=x reset=x enable=x data=x
1 clk=0 reset=x enable=x data=x
2 clk=0 reset=x enable=x data=0
5 clk=0 reset=0 enable=0 data=0
15 Terminating simulation
```

Bloklayan ve Bloklamayan(blocking and nonblocking) Atamalar

Bloklayan(tıkayan) atamalar kodlandığı sırada gerçekleştirilirler, bundan dolayı bunlar ardışıldır. Şuanki ifade gerçekleştirilene kadar bir sonraki ifade tıkanır, bunlara bloklayan ifadeler denir. Atamalar "=" sembolüyle yapılır. Örneğin a=b;

Bloklamayan ifadeler paralel olarak gerçekleştirilir. Bir sonraki ifadenin gerçekleştirilmesi mevcut ifadenin gerçeklemesini beklemez yani mevcut ifade bir sonraki ifadeyi tıkamaz, buna bloklamayan ifade denir. Atama "<=" sembolüyle yapılır. Örneğin a <= b;

Örnek– bloklayan ve bloklamayan atamalar

```
 1 module blocking_nonblocking();
 2
 3 reg a,b,c,d;
 4 // bloklayan atamalar
 5 initial begin
 6    #10  a = 0;
 7    #11  a = 1;
 8    #12  a = 0;
 9    #13  a = 1;
10 end
11
12 initial begin
13    #10  b <= 0;
14    #11  b <= 1;
15    #12  b <= 0;
16    #13  b <= 1;
17 end
18
19 initial begin
20    c = #10 0;
21    c = #11 1;
22    c = #12 0;
23    c = #13 1;
24 end
25
26 initial begin
27    d <=  #10  0;
28    d <=  #11  1;
29    d <=  #12  0;
30    d <=  #13  1;
31 end
32
33 initial begin
34   $monitor("TIME = %g A = %b B = %b C = %b D = %b",$time, a, b, c, d);
```

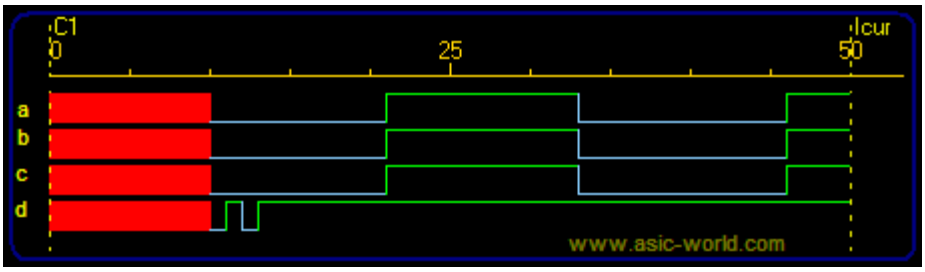
```
35    #50  $finish;
36 end
37
38 endmodule
```

You could download file blocking_nonblocking.v here

Simülatör Çıktısı

```
TIME = 0  A = x B = x C = x D = x
TIME = 10 A = 0 B = 0 C = 0 D = 0
TIME = 11 A = 0 B = 0 C = 0 D = 1
TIME = 12 A = 0 B = 0 C = 0 D = 0
TIME = 13 A = 0 B = 0 C = 0 D = 1
TIME = 21 A = 1 B = 1 C = 1 D = 1
TIME = 33 A = 0 B = 0 C = 0 D = 1
TIME = 46 A = 1 B = 1 C = 1 D = 1
```

Dalgaformu(Waveform)



assign(atama) ve deassign

Assign ve deassign prosedürel atama ifadeleri, istenen aralıklarla kontrollü olarak yazmaçlara sürekli atamaya izin vermektedir. Assign prosedürel ifadesi bir yazmacın üzerine prosedürel olarak yazar. Deassign prosedürel ifadesi bir yazmacın sürekli olarak atanmasını sonlandırır.

Örnek- assign ve deassign

```
 1 module assign_deassign ();
 2
 3 reg clk,rst,d,preset;
 4 wire q;
 5
 6 initial begin
 7   $monitor("@%g clk %b rst %b preset %b d %b q %b",
 8     $time, clk, rst, preset, d, q);
 9   clk = 0;
10   rst = 0;
11   d  = 0;
12   preset = 0;
13    #10  rst = 1;
14    #10  rst = 0;
15   repeat (10) begin
16     @ (posedge clk);
17     d <= $random;
18     @ (negedge clk) ;
19     preset <= ~preset;
20   end
```

```
21    #1  $finish;
22 end
23 // Saat üreteci
24 always  #1  clk = ~clk;
25
26 // q'nun assign ve deassign flip flop modülü
27 always @(preset)
28 if (preset) begin
29   assign U.q = 1; // assign prosedürel ifadesi
30 end else begin
31  deassign U.q;    // deassign prosedürel ifadesi
32 end
33
34 d_ff U (clk,rst,d,q);
35
36 endmodule
37
38 // D Flip-Flop modeli
39 module d_ff (clk,rst,d,q);
40 input clk,rst,d;
41 output q;
42 reg q;
43
44 always @ (posedge clk)
45 if (rst) begin
46   q <= 0;
47 end else begin
48   q <= d;
49 end
50
51 endmodule
```

You could download file assign_deassign.v here

Simülatör Çıktısı

```
@0  clk 0 rst 0 preset 0 d 0 q x
@1  clk 1 rst 0 preset 0 d 0 q 0
@2  clk 0 rst 0 preset 0 d 0 q 0
@3  clk 1 rst 0 preset 0 d 0 q 0
@4  clk 0 rst 0 preset 0 d 0 q 0
@5  clk 1 rst 0 preset 0 d 0 q 0
@6  clk 0 rst 0 preset 0 d 0 q 0
@7  clk 1 rst 0 preset 0 d 0 q 0
@8  clk 0 rst 0 preset 0 d 0 q 0
@9  clk 1 rst 0 preset 0 d 0 q 0
@10 clk 0 rst 1 preset 0 d 0 q 0
@11 clk 1 rst 1 preset 0 d 0 q 0
@12 clk 0 rst 1 preset 0 d 0 q 0
@13 clk 1 rst 1 preset 0 d 0 q 0
@14 clk 0 rst 1 preset 0 d 0 q 0
@15 clk 1 rst 1 preset 0 d 0 q 0
@16 clk 0 rst 1 preset 0 d 0 q 0
@17 clk 1 rst 1 preset 0 d 0 q 0
@18 clk 0 rst 1 preset 0 d 0 q 0
@19 clk 1 rst 1 preset 0 d 0 q 0
@20 clk 0 rst 0 preset 0 d 0 q 0
@21 clk 1 rst 0 preset 0 d 0 q 0
@22 clk 0 rst 0 preset 1 d 0 q 1
@23 clk 1 rst 0 preset 1 d 1 q 1
@24 clk 0 rst 0 preset 0 d 1 q 1
```

```
@25 clk 1 rst 0 preset 0 d 1 q 1
@26 clk 0 rst 0 preset 1 d 1 q 1
@27 clk 1 rst 0 preset 1 d 1 q 1
@28 clk 0 rst 0 preset 0 d 1 q 1
@29 clk 1 rst 0 preset 0 d 1 q 1
@30 clk 0 rst 0 preset 1 d 1 q 1
@31 clk 1 rst 0 preset 1 d 1 q 1
@32 clk 0 rst 0 preset 0 d 1 q 1
@33 clk 1 rst 0 preset 0 d 1 q 1
@34 clk 0 rst 0 preset 1 d 1 q 1
@35 clk 1 rst 0 preset 1 d 0 q 1
@36 clk 0 rst 0 preset 0 d 0 q 1
@37 clk 1 rst 0 preset 0 d 1 q 0
@38 clk 0 rst 0 preset 1 d 1 q 1
@39 clk 1 rst 0 preset 1 d 1 q 1
@40 clk 0 rst 0 preset 0 d 1 q 1
```

force(zorlama) ve release(bırakma)

Prosedürel sürekli atama ifadesinin başkabir biçimi ise force ve release prosedürel ifadeleriyle sağlanır. Bu ifadeler assign-deassign çiftiyle benzer etkiye sahiptir, ancak bir force yazmaçlardaki gibi net'lere uygulanabilir.

Force ve release kullanıldığında, kapı seviyesi simülasyon yaparken reset(sıfırlama) bağlanırlık sorunlarında etkili olmaktadır. Böylece bellekten veri okuma işlemi sırasındaki hatalar için tek ve çift bit eklenerek kullanılabilinir.

Örnek- force ve release

```
 1 module force_release ();
 2
 3 reg clk,rst,d,preset;
 4 wire q;
 5
 6 initial begin
 7   $monitor("@%g clk %b rst %b preset %b d %b q %b",
 8     $time, clk, rst, preset, d, q);
 9   clk = 0;
10   rst = 0;
11   d  = 0;
12   preset = 0;
13    #10  rst = 1;
14    #10  rst = 0;
15   repeat (10) begin
16     @ (posedge clk);
17     d <= $random;
18     @ (negedge clk) ;
19     preset <= ~preset;
20   end
21    #1  $finish;
22 end
23 // Clock generator
24 always  #1  clk = ~clk;
25
26 // flip flop modülün force ve release ifadesi
27 always @(preset)
28 if (preset) begin
29   force U.q = preset; // force prosedürel ifadesi
```

```
30 end else begin
31   release U.q;    // release prosedürel ifadesi
32 end
33
34 d_ff U (clk,rst,d,q);
35
36 endmodule
37
38 // D Flip-Flop modeli
39 module d_ff (clk,rst,d,q);
40 input clk,rst,d;
41 output q;
42 wire q;
43 reg q_reg;
44
45 assign q = q_reg;
46
47 always @ (posedge clk)
48 if (rst) begin
49   q_reg <= 0;
50 end else begin
51   q_reg <= d;
52 end
53
54 endmodule
```

You could download file force_release.v here

Simülatör Çıktısı

```
@0  clk 0 rst 0 preset 0 d 0 q x
@1  clk 1 rst 0 preset 0 d 0 q 0
@2  clk 0 rst 0 preset 0 d 0 q 0
@3  clk 1 rst 0 preset 0 d 0 q 0
@4  clk 0 rst 0 preset 0 d 0 q 0
@5  clk 1 rst 0 preset 0 d 0 q 0
@6  clk 0 rst 0 preset 0 d 0 q 0
@7  clk 1 rst 0 preset 0 d 0 q 0
@8  clk 0 rst 0 preset 0 d 0 q 0
@9  clk 1 rst 0 preset 0 d 0 q 0
@10 clk 0 rst 1 preset 0 d 0 q 0
@11 clk 1 rst 1 preset 0 d 0 q 0
@12 clk 0 rst 1 preset 0 d 0 q 0
@13 clk 1 rst 1 preset 0 d 0 q 0
@14 clk 0 rst 1 preset 0 d 0 q 0
@15 clk 1 rst 1 preset 0 d 0 q 0
@16 clk 0 rst 1 preset 0 d 0 q 0
@17 clk 1 rst 1 preset 0 d 0 q 0
@18 clk 0 rst 1 preset 0 d 0 q 0
@19 clk 1 rst 1 preset 0 d 0 q 0
@20 clk 0 rst 0 preset 0 d 0 q 0
@21 clk 1 rst 0 preset 0 d 0 q 0
@22 clk 0 rst 0 preset 1 d 0 q 1
@23 clk 1 rst 0 preset 1 d 1 q 1
@24 clk 0 rst 0 preset 0 d 1 q 0
@25 clk 1 rst 0 preset 0 d 1 q 1
@26 clk 0 rst 0 preset 1 d 1 q 1
@27 clk 1 rst 0 preset 1 d 1 q 1
@28 clk 0 rst 0 preset 0 d 1 q 1
@29 clk 1 rst 0 preset 0 d 1 q 1
@30 clk 0 rst 0 preset 1 d 1 q 1
```

```
@31 clk 1 rst 0 preset 1 d 1 q 1
@32 clk 0 rst 0 preset 0 d 1 q 1
@33 clk 1 rst 0 preset 0 d 1 q 1
@34 clk 0 rst 0 preset 1 d 1 q 1
@35 clk 1 rst 0 preset 1 d 0 q 1
@36 clk 0 rst 0 preset 0 d 0 q 1
@37 clk 1 rst 0 preset 0 d 1 q 0
@38 clk 0 rst 0 preset 1 d 1 q 1
@39 clk 1 rst 0 preset 1 d 1 q 1
@40 clk 0 rst 0 preset 0 d 1 q 1
```

Koşullu İfade(Conditional Statement) if-else

if - else ifadesi diğer ifadelerin gerçekleştirilmesini kontrol eder. C gibi bir programlama dilinde olduğu gibi, if-else programın akışını kontrol eder. Bir if(eğer) koşulu için eğer bir ifadeden fazla ifadenin gerçekleştirilmesi gerekiyorsa, begin ve end kullanmamız gerekir daha önceki örneklerde olduğu gibi.

Sözdizim: if

if (koşul)

ifadeler;

Sözdizimi : if-else

if (koşul)

ifadeler;

else

ifadeler;

Sözdizim: iç içe if-else-if

if (koşul)

ifadeler;

else if (koşul)

ifadeler;

.....

.....

else

statements;

Örnek- basit bir if ifadesi

```
 1 module simple_if();
 2
 3 reg latch;
 4 wire enable,din;
 5
 6 always @ (enable or din)
 7 if (enable) begin
 8   latch <= din;
 9 end
10
11 endmodule
```

You could download file simple_if.v here

Örnek- if-else

```
 1 module if_else();
 2
 3 reg dff;
 4 wire clk,din,reset;
 5
 6 always @ (posedge clk)
 7 if (reset) begin
 8   dff <= 0;
 9 end else  begin
10   dff <= din;
11 end
12
13 endmodule
```

You could download file if_else.v here

Örnek- iç içe - if-else-if

```
 1 module nested_if();
 2
 3 reg [3:0] counter;
 4 reg clk,reset,enable, up_en, down_en;
 5
 6 always @ (posedge clk)
 7 // Eğer reset gerçekleştiyse
 8 if (reset == 1'b0) begin
 9    counter <= 4'b0000;
10 // Eğer counter(sayaç) aktifse ve up count(yukarı sayaç) seçiliyse
11 end else if (enable == 1'b1 && up_en == 1'b1) begin
12    counter <= counter + 1'b1;
13 // Eğer counter(sayaç) aktifse ve down count(aşağı sayaç) seçiliyse
14 end else if (enable == 1'b1 && down_en == 1'b1) begin
15    counter <= counter - 1'b1;
16 // Eğer sayma aktif değilse
17 end else begin
18    counter <= counter; // Artık kod(Redundant code)
19 end
20
21 // Testbenç Kodu
22 initial begin
23   $monitor ("@%0dns reset=%b enable=%b up=%b down=%b count=%b",
24              $time, reset, enable, up_en, down_en,counter);
25   $display("@%0dns Driving all inputs to know state",$time);
```

```
26   clk = 0;
27   reset = 0;
28   enable = 0;
29   up_en = 0;
30   down_en = 0;
31    #3  reset = 1;
32   $display("@%0dns De-Asserting reset",$time);
33    #4  enable = 1;
34   $display("@%0dns De-Asserting reset",$time);
35    #4  up_en = 1;
36   $display("@%0dns Putting counter in up count mode",$time);
37    #10  up_en = 0;
38   down_en = 1;
39   $display("@%0dns Putting counter in down count mode",$time);
40    #8  $finish;
41 end
42
43 always  #1  clk = ~clk;
44
45 endmodule
```

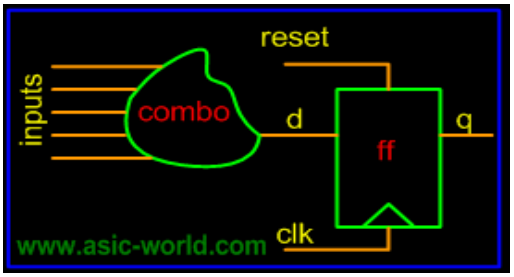
You could download file nested_if.v here

Simülasyon Kaydı –iç içe - if-else-if

```
@0ns Driving all inputs to know state
@0ns reset=0 enable=0 up=0 down=0 count=xxxx
@1ns reset=0 enable=0 up=0 down=0 count=0000
@3ns De-Asserting reset
@3ns reset=1 enable=0 up=0 down=0 count=0000
@7ns De-Asserting reset
@7ns reset=1 enable=1 up=0 down=0 count=0000
@11ns Putting counter in up count mode
@11ns reset=1 enable=1 up=1 down=0 count=0001
@13ns reset=1 enable=1 up=1 down=0 count=0010
@15ns reset=1 enable=1 up=1 down=0 count=0011
@17ns reset=1 enable=1 up=1 down=0 count=0100
@19ns reset=1 enable=1 up=1 down=0 count=0101
@21ns Putting counter in down count mode
@21ns reset=1 enable=1 up=0 down=1 count=0100
@23ns reset=1 enable=1 up=0 down=1 count=0011
@25ns reset=1 enable=1 up=0 down=1 count=0010
@27ns reset=1 enable=1 up=0 down=1 count=0001
```

Paralel if-else

Yukarıdaki örnekte, (enable == 1'b1 && up_en == 1'b1) en yüksek öncelikli olarak verilmiştir ve (enable == 1'b1 && down_en == 1'b1) koşulu en düşük öncelikli olarak verilmiştir. Normalde biz reset kontrolünü öncelik sırasına koymayız aşağıdaki şekilde gösterildiği gibi.



Sonuç olarak, eğer öncelik mantığına ihtiyacımız varsa iç içe il-else ifadelerini kullanmalıyız. Diğer taraftan eğer bir öncelik mantığı gerçeklemek istemiyorsak, sadece bir girişin aynı zamanda aktif olduğunu biliyorsak(eğer tüm girişler birbirini dışlayansa) kodu gösterildiği gibi yazabiliriz.

Şu bilinen bir gerçek ki öncelik gerçekleme paralel gerçeklemeden daha çok lojik alır. Bu yüzden eğer girişlerin tümünün birbirini dışladığını biliyorsanız, paralel lojik olarak kodlayabilirsiniz.

```
 1 module parallel_if();
 2
 3 reg [3:0] counter;
 4 wire clk,reset,enable, up_en, down_en;
 5
 6 always @ (posedge clk)
 7 //Eğer reset gerçekleştiyse
 8 if (reset == 1'b0) begin
 9    counter <= 4'b0000;
10 end else begin
11   // Eğer counter(sayaç) aktifse ve up count(yukarı sayaç) modundaysa
12   if (enable == 1'b1 && up_en == 1'b1) begin
13     counter <= counter + 1'b1;
14   end
15   // Eğer counter(sayaç) aktifse ve down count(aşağı sayaç) modundaysa
16   if (enable == 1'b1 && down_en == 1'b1) begin
17     counter <= counter - 1'b1;
18   end
19 end
20
21 endmodule
```

You could download file parallel_if.v here

Case İfadesi

Case ifadesi bir ifadeyi bir seri durumla karşılaştırır ve ifadeyi gerçekleştirir ya da ilk eşleşen durumla ilgili olan bir grup ifadeyi gerçekleştirir:

- Case ifadesi tekli veya çoklu ifadeleri destekler.
- Çoklu ifadelerin gruplanması begin ve end anahtar sözcüğüyle sağlanır.

Bir case ifadesinin söz dizimi aşağıda gösterildiği gibidir.

case ()

< case1 > : < ifade >

< case2 > : < ifade >

.....

default : < ifade >

endcase

Normal Case

Örnek- case

```
 1 module mux (a,b,c,d,sel,y);
 2 input a, b, c, d;
 3 input [1:0] sel;
 4 output y;
 5
 6 reg y;
 7
 8 always @ (a or b or c or d or sel)
 9 case (sel)
10   0 : y = a;
11   1 : y = b;
12   2 : y = c;
13   3 : y = d;
14   default : $display("Error in SEL");
15 endcase
16
17 endmodule
```

You could download file mux.v here

Örnek- case 'in default'suz kullanımı

```
 1 module mux_without_default (a,b,c,d,sel,y);
 2 input a, b, c, d;
 3 input [1:0] sel;
 4 output y;
 5
 6 reg y;
 7
 8 always @ (a or b or c or d or sel)
 9 case (sel)
10   0 : y = a;
11   1 : y = b;
12   2 : y = c;
13   3 : y = d;
14   2'bxx,2'bx0,2'bx1,2'b0x,2'b1x,
15   2'bzz,2'bz0,2'bz1,2'b0z,2'b1z : $display("Error in SEL");
16 endcase
17
18 endmodule
```

You could download file mux_without_default.v here

Yukarıdaki örnekte çoklu durum(case) elemanlarının nasıl tek durum elemanıymış gibi bildirileceği gösterilmiştir.

Verilog case ifadesi bir kimlik karşılaştırması yapar(=== operatöründe olduğu gibi); case ifadesi aşağıdaki örnekte de gösterildiği gibi x ve z değerlirinin kontrolü için de kullanılabilir.

Örnek- case'in x ve z ile kullanımı

```
 1 module case_xz(enable);
 2 input enable;
 3
 4 always @ (enable)
```

```
 5 case(enable)
 6   1'bz : $display ("enable is floating");
 7   1'bx : $display ("enable is unknown");
 8   default : $display ("enable is %b",enable);
 9 endcase
10
11 endmodule
```

You could download file case_xz.v here

casez ve casex ifadeleri

Case ifadelerinin x ve z lojiğinin önemsiz durum olarak kullanımına izin verdiği özel versiyonudur :

- casez : z ye önemsiz durummuş gibi davranır.
- casex : x ve z 'ye önemsiz durummuş gibi davranır.

Örnek- casez

```
 1 module casez_example();
 2 reg [3:0] opcode;
 3 reg [1:0] a,b,c;
 4 reg [1:0] out;
 5
 6 always @ (opcode or a or b or c)
 7 casez(opcode)
 8   4'b1zzx : begin //daha düşük 2:1 bit önemsizdir, bit 0 x'le eşleşmeli
 9               out = a;
10               $display("@%0dns 4'b1zzx is selected, opcode
%b",$time,opcode);
11             end
12   4'b01?? : begin
13               out = b; // bit 1:0 önemsiz(don't care)
14               $display("@%0dns 4'b01?? is selected, opcode
%b",$time,opcode);
15             end
16   4'b001? : begin  //bit 0 önemsiz(don't care)
17               out = c;
18               $display("@%0dns 4'b001? is selected, opcode
%b",$time,opcode);
19             end
20   default : begin
21               $display("@%0dns default is selected, opcode
%b",$time,opcode);
22             end
23 endcase
24
25 // Testbenç kodu buradan itibaren başlamaktadır.
26 always  #2  a = $random;
27 always  #2  b = $random;
28 always  #2  c = $random;
29
30 initial begin
31   opcode = 0;
32    #2  opcode = 4'b101x;
33    #2  opcode = 4'b0101;
34    #2  opcode = 4'b0010;
35    #2  opcode = 4'b0000;
```

```
 36    #2  $finish;
 37 end
 38
 39 endmodule
```

You could download file casez_example.v here

Simülasyon çıktısı - casez

```
@0ns default is selected, opcode 0000
@2ns 4'b1zzx is selected, opcode 101x
@4ns 4'b01?? is selected, opcode 0101
@6ns 4'b001? is selected, opcode 0010
@8ns default is selected, opcode 0000
```

Örnek- casex

```
  1 module casex_example();
  2 reg [3:0] opcode;
  3 reg [1:0] a,b,c;
  4 reg [1:0] out;
  5
  6 always @ (opcode or a or b or c)
  7 casex(opcode)
  8   4'b1zzx : begin // Önemsiz 2:0 bitleri
  9               out = a;
 10               $display("@%0dns 4'b1zzx is selected, opcode
%b",$time,opcode);
 11             end
 12   4'b01?? : begin // bit 1:0 önemsiz(don't care)
 13               out = b;
 14               $display("@%0dns 4'b01?? is selected, opcode
%b",$time,opcode);
 15             end
 16   4'b001? : begin // bit 0 önemsiz(don't care)
 17               out = c;
 18               $display("@%0dns 4'b001? is selected, opcode
%b",$time,opcode);
 19             end
 20   default : begin
 21               $display("@%0dns default is selected, opcode
%b",$time,opcode);
 22             end
 23 endcase
 24
 25 // Testbenç kodu buradan itibaren başlamaktadır
 26 always  #2  a = $random;
 27 always  #2  b = $random;
 28 always  #2  c = $random;
 29
 30 initial begin
 31   opcode = 0;
 32    #2  opcode = 4'b101x;
 33    #2  opcode = 4'b0101;
 34    #2  opcode = 4'b0010;
 35    #2  opcode = 4'b0000;
 36    #2  $finish;
 37 end
 38
 39 endmodule
```

You could download file casex_example.v here

Simülasyon Çıktısı- casex

```
@0ns default is selected, opcode 0000
@2ns 4'b1zzx is selected, opcode 101x
@4ns 4'b01?? is selected, opcode 0101
@6ns 4'b001? is selected, opcode 0010
@8ns default is selected, opcode 0000
```

Örnek- case, casex, casez karşılaştırılması

```
 1 module case_compare;
 2
 3 reg sel;
 4
 5 initial begin
 6    #1  $display ("\n     Driving 0");
 7   sel = 0;
 8    #1  $display ("\n     Driving 1");
 9   sel = 1;
10    #1  $display ("\n     Driving x");
11   sel = 1'bx;
12    #1  $display ("\n     Driving z");
13   sel = 1'bz;
14    #1  $finish;
15 end
16
17 always @ (sel)
18 case (sel)
19   1'b0 : $display("Normal : Logic 0 on sel");
20   1'b1 : $display("Normal : Logic 1 on sel");
21   1'bx : $display("Normal : Logic x on sel");
22   1'bz : $display("Normal : Logic z on sel");
23 endcase
24
25 always @ (sel)
26 casex (sel)
27   1'b0 : $display("CASEX  : Logic 0 on sel");
28   1'b1 : $display("CASEX  : Logic 1 on sel");
29   1'bx : $display("CASEX  : Logic x on sel");
30   1'bz : $display("CASEX  : Logic z on sel");
31 endcase
32
33 always @ (sel)
34 casez (sel)
35   1'b0 : $display("CASEZ  : Logic 0 on sel");
36   1'b1 : $display("CASEZ  : Logic 1 on sel");
37   1'bx : $display("CASEZ  : Logic x on sel");
38   1'bz : $display("CASEZ  : Logic z on sel");
39 endcase
40
41 endmodule
```

You could download file case_compare.v here

Simülasyon Çıktısı

```
     Driving 0
Normal : Logic 0 on sel
CASEX  : Logic 0 on sel
CASEZ  : Logic 0 on sel
```

```
     Driving 1
Normal : Logic 1 on sel
CASEX  : Logic 1 on sel
CASEZ  : Logic 1 on sel

     Driving x
Normal : Logic x on sel
CASEX  : Logic 0 on sel
CASEZ  : Logic x on sel

     Driving z
Normal : Logic z on sel
CASEX  : Logic 0 on sel
CASEZ  : Logic 0 on sel
```

Döngüleme İfadeleri

Döngüleme ifadeleri prosedürel bloğun içinde görünür sadece; Verilog'un dört döngüleme ifadesi vardır diğer programlama dillerindekilere benzer şekilde.

- forever
- repeat
- while
- for

forever(daima-sonsuz) ifadesi

Forever(daima) döngüsü sürekli olarak gerçekleştirilir, döngü asla sonlanmaz. Normalde biz forever ifadesini başlangıç bloğunda kullanılır.

sözdizim : forever < ifade >

forever ifadesi kullanılırken çok dikkatli olunmalı: eğer forevet ifadesinde bir zamanlama yapısı varsa, simülasyon askıda kalabilir. Aşağıdaki kod bir uygulamadır ve zamanlama yapısı forever ifadesine eklenmiştir.

Örnek – Serbest hareketli saat üreteci(Free running clock generator)

```
 1 module forever_example ();
 2
 3 reg clk;
 4
 5 initial begin
 6    #1  clk = 0;
 7   forever begin
 8      #5  clk =  ! clk;
 9   end
10 end
11
12 initial begin
13   $monitor ("Time = %d  clk = %b",$time, clk);
14    #100  $finish;
15 end
16
```

```
17 endmodule
```

You could download file forever_example.v here

repeat ifadesi

repeat döngüsü <ifade> belirli bir <sayıda> gerçekleştiren döngüdür.

sözdizim: repeat (< sayı >) < ifade>

Örnek- repeat

```
 1 module repeat_example();
 2 reg  [3:0] opcode;
 3 reg  [15:0] data;
 4 reg        temp;
 5
 6 always @ (opcode or data)
 7 begin
 8   if (opcode == 10) begin
 9     // Döndürek gerçekleştirme
10     repeat (8) begin
11        #1  temp = data[15];
12       data = data << 1;
13       data[0] = temp;
14     end
15   end
16 end
17 // Basit test kodu
18 initial begin
19    $display (" TEMP  DATA");
20    $monitor (" %b     %b ",temp, data);
21     #1  data = 18'hF0;
22     #1  opcode = 10;
23     #10  opcode = 0;
24     #1  $finish;
25 end
26
27 endmodule
```

You could download file repeat_example.v here

while döngü ifadesi

While döngüsü <söylem> doğru olduğu sürece gerçekleştirilir. Bu diğer programlama dilleriyle aynıdır.

sözdizim : while (< söylem >) < ifade >

Örnek- while

```
 1 module while_example();
 2
 3 reg [5:0] loc;
 4 reg [7:0] data;
 5
 6 always @ (data or loc)
 7 begin
 8   loc = 0;
```

```
 9   // Eğer veri 0 ise,loc 32'dir(geçersiz değer)
10   if (data == 0) begin
11     loc = 32;
12   end else begin
13     while (data[0] == 0) begin
14       loc = loc + 1;
15       data = data >> 1;
16     end
17   end
18   $display ("DATA = %b   LOCATION = %d",data,loc);
19 end
20
21 initial begin
22    #1  data = 8'b11;
23    #1  data = 8'b100;
24    #1  data = 8'b1000;
25    #1  data = 8'b1000_0000;
26    #1  data = 8'b0;
27    #1  $finish;
28 end
29
30 endmodule
```

You could download file while_example.v here

for döngüsü ifadesi

For döngüsü diğer dillerde kullanılan for döngüsü aynı şekildedir.

- Bir <başlangıç ataması> döngünün başlangıcında birkez gerçekleştirilir.
- <deyim> doğru olduğu sürece döngü gerçekleştirlir.
- Döngünün herbir turunun sonunda bir <adım ataması> yapar.

sözdizim: for (< başlangıç ataması>; < deyim >, < adım ataması>) < ifade >

Not : verilog'un C de olduğu gibi ++ veya -- operatörleri yoktur.

Örnek - For

```
 1 module for_example();
 2
 3 integer i;
 4 reg [7:0] ram [0:255];
 5
 6 initial begin
 7   for (i = 0; i < 256; i = i + 1) begin
 8      #1  $display(" Address = %g  Data = %h",i,ram[i]);
 9     ram[i] <= 0; // RAM ilk değer olarak 0 verilir
10      #1  $display(" Address = %g  Data = %h",i,ram[i]);
11   end
12    #1  $finish;
13 end
14
15 endmodule
```

You could download file for_example.v here

Sürekli Atama İfadeleri

Sürekli atama ifadeleri net'leri(wire veri tipini) sürer. Yapısal bağlantıları gösterir.

- Üç durumlu(Tri-State) arabellekleri modellemek için kullanılır.
- Kombinasyonel lojiği modellemek için kulanılabilir.
- Prosedürel bloğun(always ve initial bloklarının) dışındadır.
- Sürekli atama herhangi bir prosedürel atamanın üstündedir.
- Sürekli atamanın sol-tarafı mutlaka bir net veri tipinde olmalı.

sözdizim : atama (direnç, direnç) **#(delay)** net = deyim;

Örnek-Tek bit toplayıcı

```
 1 module adder_using_assign ();
 2 reg a, b;
 3 wire sum, carry;
 4
 5 assign  #5  {carry,sum} = a+b;
 6
 7 initial begin
 8   $monitor (" A = %b  B = %b CARRY = %b SUM = %b",a,b,carry,sum);
 9    #10  a = 0;
10   b = 0;
11    #10   a = 1;
12    #10   b = 1;
13    #10   a = 0;
14    #10   b = 0;
15    #10  $finish;
16 end
17
18 endmodule
```

You could download file adder_using_assign.v here

Örnek - Üç durumlu(Tri-state) arabellek(buffer)

```
 1 module tri_buf_using_assign();
 2 reg data_in, enable;
 3 wire pad;
 4
 5 assign pad = (enable) ? data_in : 1'bz;
 6
 7 initial begin
 8   $monitor ("TIME = %g ENABLE = %b DATA : %b PAD %b",
 9     $time, enable, data_in, pad);
10    #1  enable = 0;
11    #1  data_in = 1;
12    #1  enable = 1;
13    #1  data_in = 0;
14    #1  enable = 0;
15    #1  $finish;
16 end
17
18 endmodule
```

You could download file tri_buf_using_assign.v here

Yayılım Gecikmesi (Propagation Delay)

Sürekli atamaların belirtilmiş bir gecikmeye sahiptir; sadece tüm geçişler için gecikmeler belirtilmiş olmalıdır. Bir minimum:tipik:maksimum gecikme aralığı belirtilmelidir.

Örnek – Üç-durumlu arabellek(Tri-state buffer)

```
 1 module tri_buf_using_assign_delays();
 2 reg data_in, enable;
 3 wire pad;
 4
 5 assign #(1:2:3) pad = (enable) ? data_in : 1'bz;
 6
 7 initial begin
 8   $monitor ("ENABLE = %b DATA : %b PAD %b",enable, data_in,pad);
 9    #10  enable = 0;
10    #10  data_in = 1;
11    #10  enable = 1;
12    #10  data_in = 0;
13    #10  enable = 0;
14    #10  $finish;
15 end
16
17 endmodule
```

You could download file tri_buf_using_assign_delays.v here

Prosedürel Blok Kontrolü

Prosedürel blok simülasyon zamanının sıfır anında aktif olacaktır. Seviye hassasiyetli olay kontrolü kullanma bir prosedürün gerçekleştirilmesini kontrol eder.

```
 1 module dlatch_using_always();
 2 reg q;
 3
 4 reg d, enable;
 5
 6 always @ (d or enable)
 7 if (enable) begin
 8   q = d;
 9 end
10
11 initial begin
12   $monitor (" ENABLE = %b D = %b  Q = %b",enable,d,q);
13    #1  enable = 0;
14    #1  d = 1;
15    #1  enable = 1;
16    #1  d = 0;
17    #1  d = 1;
18    #1  d = 0;
19    #1  enable = 0;
20    #10  $finish;
21 end
22
23 endmodule
```

You could download file dlatch_using_always.v here

d veya enable ‘daki herhangi bir değişiklik olay kontrolü ve prosedür ifadesinin gerçekleştirilmesini sağlamaktadır.

Prosedürel Kodlama kullanılarak Çoklu Lojik (Combo Logic)

Kombinasyonel lojiği modellemek için, bir prosedür bloğu girişteki herhangi bir değişikliğe hassas olmalı. Bir diğer önemli kuralsa kombinasyonel lojik modellerken takip edilmelidir. Eğer koşullu kontrolü “if” kullanarak yapıyorsanız, “else” bölümünü kullanmak zorundasınız. Elsin unutulması bir tutucu(latch) ile sonuçlanır. Eğer else bölümünü yazmayı sevmiyorsanız, çoklu bloğun(combo block) tüm değişkenlerine ilk değer vermelisiniz.

Örnek – Bir bitlik Toplayıcı

```
 1 module adder_using_always ();
 2 reg a, b;
 3 reg sum, carry;
 4
 5 always @ (a or b)
 6 begin
 7   {carry,sum} = a + b;
 8 end
 9
10 initial begin
11   $monitor (" A = %b B = %b CARRY = %b SUM = %b",a,b,carry,sum);
12    #10  a = 0;
13    b = 0;
14     #10  a = 1;
15     #10  b = 1;
16     #10  a = 0;
17     #10  b = 0;
18     #10  $finish;
19 end
20
21 endmodule
```

You could download file adder_using_always.v here

Prosedürel bloğun içindeki ifade tüm vektörlerle aynı anda çalışır.

Örnek- 4-bit Toplayıcı

```
 1 module adder_4_bit_using_always ();
 2 reg[3:0] a, b;
 3 reg [3:0] sum;
 4 reg carry;
 5
 6 always @ (a or b)
 7 begin
 8   {carry,sum} = a + b;
 9 end
10
11 initial begin
12   $monitor (" A = %b B = %b CARRY = %b SUM = %b",a,b,carry,sum);
13    #10  a = 8;
14    b = 7;
15     #10  a = 10;
16     #10  b = 15;
17     #10  a = 0;
```

```
18      #10  b = 0;
19      #10  $finish;
20  end
21
22  endmodule
```

You could download file adder_4_bit_using_always.v here

Örnek – Tutucuyu önlemenin yolları – Tüm koşulların belirtilmesi

```
 1  module avoid_latch_else ();
 2
 3  reg q;
 4  reg enable, d;
 5
 6  always @ (enable or d)
 7  if (enable) begin
 8    q = d;
 9  end else begin
10    q = 0;
11  end
12
13  initial begin
14    $monitor (" ENABLE = %b  D = %b Q = %b",enable,d,q);
15     #1  enable = 0;
16     #1  d = 0;
17     #1  enable = 1;
18     #1  d = 1;
19     #1  d = 0;
20     #1  d = 1;
21     #1  d = 0;
22     #1  d = 1;
23     #1  enable = 0;
24     #1  $finish;
25  end
26
27  endmodule
```

You could download file avoid_latch_else.v here

Örnek - Tutucuyu önlemenin yolları – Hataya meyilli değişkenlerin sıfır yapılması

```
 1  module avoid_latch_init ();
 2
 3  reg q;
 4  reg enable, d;
 5
 6  always @ (enable or d)
 7  begin
 8    q = 0;
 9    if (enable) begin
10      q = d;
11    end
12  end
13
14  initial begin
15    $monitor (" ENABLE = %b  D = %b Q = %b",enable,d,q);
16     #1  enable = 0;
17     #1  d = 0;
18     #1  enable = 1;
19     #1  d = 1;
20     #1  d = 0;
```

```
21    #1  d = 1;
22    #1  d = 0;
23    #1  d = 1;
24    #1  enable = 0;
25    #1  $finish;
26 end
27
28 endmodule
```

You could download file avoid_latch_init.v here

Prosedürel Kodlama kullanılarak Ardışıl Lojik

Ardışıl lojiği modellemek için, bir prosedür bloğu saatin pozitif veya negatif kenarına hassas olmalı. Asenkron sıfırlama(reset) modellemek için, prosedür bloğu hem saate hem de reset'e hassas olmalı. Ardışıl lojikteki tüm atamalar birbirini tıkamayan-engellemeyen(nonblocking) şekilde yapılmalıdır.

Bazen hassasiyet listesinde çoklu kenar tetiklemeli değişkenlere sahip olmak isteyebilirz: bu simülasyon için iyidir. Fakat sentez için bu gerçek hayattaki kadar hassas değildir, flip-flop'un sadece bir saati, bir reset(sıfırlama) ve bir preset(önayarlı) olabilir.

Yeni başlayanların sık olarak karşılaştıkları bir hata ise saati(clock) flip-flop'un enable(ektin) girişine bağlanmasıdır. Bu simülasyon için iyidir ancak sentez için bu doğru değildir.

Örnek – Kötü kodlama – İki saat kullanarak

```
 1 module wrong_seq();
 2
 3 reg q;
 4 reg clk1, clk2, d1, d2;
 5
 6 always @ (posedge clk1 or posedge clk2)
 7 if (clk1) begin
 8   q <= d1;
 9 end else if (clk2) begin
10   q <= d2;
11 end
12
13 initial begin
14   $monitor ("CLK1 = %b CLK2 = %b D1 = %b D2 %b Q = %b",
15     clk1, clk2, d1, d2, q);
16   clk1 = 0;
17   clk2 = 0;
18   d1 = 0;
19   d2 = 1;
20    #10  $finish;
21 end
22
23 always
24   #1  clk1 = ~clk1;
25
26 always
27  #1.9 clk2 = ~clk2;
28
29 endmodule
```

You could download file wrong_seq.v here

Örnek – asenkron reset ve asenkron preset 'li D Flip-flopu

```
 1 module dff_async_reset_async_preset();
 2
 3 reg clk,reset,preset,d;
 4 reg  q;
 5
 6 always @ (posedge clk or posedge reset or posedge preset)
 7 if (reset) begin
 8   q <= 0;
 9 end else if (preset) begin
10   q <= 1;
11 end else begin
12   q <= d;
13 end
14
15 // Testbenç kodu buradan itibaren başlar
16 initial begin
17   $monitor("CLK = %b RESET = %b PRESET = %b D = %b Q = %b",
18     clk,reset,preset,d,q);
19   clk    = 0;
20    #1  reset  = 0;
21   preset = 0;
22   d      = 0;
23    #1  reset = 1;
24    #2  reset = 0;
25    #2  preset = 1;
26    #2  preset = 0;
27   repeat (4) begin
28      #2  d      = ~d;
29   end
30    #2  $finish;
31 end
32
33 always
34   #1  clk = ~clk;
35
36 endmodule
```

You could download file dff_async_reset_async_preset.v here

Örnek – senkron reset ve senkron preset'li D Flip-flopu

```
 1 module dff_sync_reset_sync_preset();
 2
 3 reg clk,reset,preset,d;
 4 reg  q;
 5
 6 always @ (posedge clk)
 7 if (reset) begin
 8   q <= 0;
 9 end else if (preset) begin
10   q <= 1;
11 end else begin
12   q <= d;
13 end
14
15 // Testbenç kodu buradan itibaren başlar
16 initial begin
17   $monitor("CLK = %b RESET = %b PRESET = %b D = %b Q = %b",
18     clk,reset,preset,d,q);
```

```
19   clk    = 0;
20    #1  reset  = 0;
21   preset = 0;
22   d      = 0;
23    #1  reset = 1;
24    #2  reset = 0;
25    #2  preset = 1;
26    #2  preset = 0;
27   repeat (4) begin
28      #2  d      = ~d;
29   end
30    #2  $finish;
31 end
32
33 always
34   #1  clk = ~clk;
35
36 endmodule
```

You could download file dff_sync_reset_sync_preset.v here

Bir prosedür kendi kendini tetikleyemez

Değer atanmasını veya sürülmesini engelleyen bir değişken bloğu prosedürü tetikleyemez.

```
 1 module trigger_itself();
 2
 3 reg clk;
 4
 5 always @ (clk)
 6    #5  clk =  ! clk;
 7
 8 // Testbenç kodu buradan itibaren başlar
 9 initial begin
10   $monitor("TIME = %d  CLK = %b",$time,clk);
11   clk = 0;
12    #500  $display("TIME = %d  CLK = %b",$time,clk);
13   $finish;
14 end
15
16 endmodule
```

You could download file trigger_itself.v here

Prosedürel Blok Uyumluluğu

Eğer bir modülün içinde birden çok always bloğu varsa, tüm bloklar(always ve initial blokları) 0 zamanında gerçekleştirilmeye başlanacaktır ve eşzamanlı olarak çalışmaya devam edecektir. Eğer kodlama uygun şekilde yapılmamışsa bazen bu bir yarış durumuna neden olmaktadır.

```
 1 module multiple_blocks ();
 2 reg a,b;
 3 reg c,d;
 4 reg clk,reset;
 5 // Çoklu Lojik(Combo Logic)
 6 always @ ( c)
 7 begin
 8   a = c;
```

```
 9 end
10 // Ardışıl Lojik(Seq Logic)
11 always @ (posedge clk)
12 if (reset) begin
13   b <= 0;
14 end else begin
15   b <= a & d;
16 end
17
18 // Testbenç kodu buradan itibaren başlar
19 initial begin
20   $monitor("TIME = %d CLK = %b C = %b D = %b A = %b B = %b",
21     $time, clk,c,d,a,b);
22   clk = 0;
23   reset = 0;
24   c = 0;
25   d = 0;
26    #2  reset = 1;
27    #2  reset = 0;
28    #2  c = 1;
29    #2  d = 1;
30    #2  c = 0;
31    #5  $finish;
32 end
33 // Saat üreteci
34 always
35   #1  clk = ~clk;
36
37 endmodule
```

You could download file multiple_blocks.v here

Yarış durumu(Race condition)

```
 1 module race_condition();
 2 reg b;
 3
 4 initial begin
 5   b = 0;
 6 end
 7
 8 initial begin
 9   b = 1;
10 end
11
12 endmodule
```

You could download file race_condition.v here

Yukarıdaki kodda, her iki blok da aynı zamanda gerçekleştiğinden b'nin değerinin ne olduğunu söylemek zordur. Verilog'da eğer önemsenmediyse bir yarış durumu sık olan birşeydir.

İsimlendirilmiş Bloklar

Bloklara isim verilebilir, bu anahtar sözcük begin'den sonra : blok_ismi eklenerek yapılır. İsimlendirilmiş bloklar 'disable' ifadesi ile etkisizleştirilebilir(disable).

Örnek – İsimlendirilmiş Bloklar

```
 1 // Bu kod bit'in kurulu en düşük biti bulur
 2 module named_block_disable();
 3
 4 reg [31:0] bit_detect;
 5 reg [5:0]  bit_position;
 6 integer i;
 7
 8 always @ (bit_detect)
 9 begin : BIT_DETECT
10   for (i = 0; i < 32 ; i = i + 1) begin
11      // Eğer bit kurulu ise, bit pozisyonunu al
12      // Bloğun gerçekleştirilmesini etkisizleştir
13      if (bit_detect[i] == 1) begin
14         bit_position = i;
15         disable BIT_DETECT;
16      end  else begin
17         bit_position = 32;
18      end
19   end
20 end
21
22 // Testbenc kodu buradan başlar
23 initial begin
24   $monitor(" INPUT = %b  MIN_POSITION = %d", bit_detect, bit_position);
25    #1  bit_detect = 32'h1000_1000;
26    #1  bit_detect = 32'h1100_0000;
27    #1  bit_detect = 32'h1000_1010;
28    #10  $finish;
29 end
30
31 endmodule
```

You could download file named_block_disable.v here

Yukarıdaki örnekte BIT_DETECT isimlendirilmiş bloktur ve bitin pozisyonu bulunduğunda etkisizleştirilmiştir.

Procedural Timing Control

Prosesürel Blok ve zamanlama kontrolleri

- Gecikme(Delay) kontrolleri.
- Kenar-Hassasiyetli Olay(Edge-Sensitive Event) kontrolleri.
- Seviye-Hasssiyetli Olay(Level-Sensitive Event) kontrolleri-Bekleme(Wait) ifadeleri.
- İsimlendirilmiş Olaylar.

Gecikme Kontrolleri(Delay Controls)

Prosedürel ifadelerin gerçekleştirilmesinin belirli simülasyon zamanı tarafında gecikmesi;

#< zaman > < ifade>;

Örnek – saat üreteci (clk_gen)

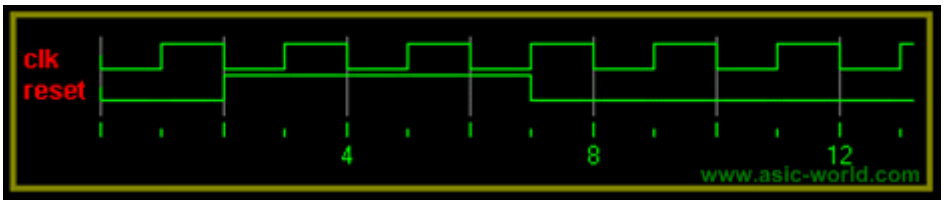
```
 1 module clk_gen ();
 2
 3 reg clk, reset;
 4
 5 initial begin
 6   $monitor ("TIME = %g RESET = %b CLOCK = %b", $time, reset, clk);
 7   clk = 0;
 8   reset = 0;
 9    #2  reset = 1;
10    #5  reset = 0;
11    #10  $finish;
12 end
13
14 always
15    #1  clk =  ! clk;
16
17 endmodule
```

You could download file clk_gen.v here

Simülasyon Çıktısı

```
TIME = 0  RESET = 0 CLOCK = 0
TIME = 1  RESET = 0 CLOCK = 1
TIME = 2  RESET = 1 CLOCK = 0
TIME = 3  RESET = 1 CLOCK = 1
TIME = 4  RESET = 1 CLOCK = 0
TIME = 5  RESET = 1 CLOCK = 1
TIME = 6  RESET = 1 CLOCK = 0
TIME = 7  RESET = 0 CLOCK = 1
TIME = 8  RESET = 0 CLOCK = 0
TIME = 9  RESET = 0 CLOCK = 1
TIME = 10 RESET = 0 CLOCK = 0
TIME = 11 RESET = 0 CLOCK = 1
TIME = 12 RESET = 0 CLOCK = 0
TIME = 13 RESET = 0 CLOCK = 1
TIME = 14 RESET = 0 CLOCK = 0
TIME = 15 RESET = 0 CLOCK = 1
TIME = 16 RESET = 0 CLOCK = 0
```

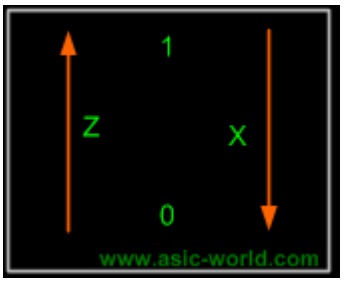
Dalgaformu(Waveform)



Kenar hassasiyetli Olay Kontrolleri(Edge sensitive Event Controls)

Gecikmelerin bir sonraki ifade için gerçekleştirilmesi sinyal üzerinde belirtilen geçiş olana kadardır.

sözdizim : @ (< artankenar >|< azalankenar > sinyal) < ifade>;



Örnek- Kenar Beklemesi(Edge Wait)

```
 1 module edge_wait_example();
 2
 3 reg enable, clk, trigger;
 4
 5 always @ (posedge enable)
 6 begin
 7   trigger = 0;
 8   // 5 saat döngüsü kadar bekle
 9   repeat (5) begin
10     @ (posedge clk) ;
11   end
12   trigger = 1;
13 end
14
15 //Testbenç kodu buradan başlar
16 initial begin
17   $monitor ("TIME : %g CLK : %b ENABLE : %b TRIGGER : %b",
18     $time, clk,enable,trigger);
19   clk = 0;
20   enable = 0;
21    #5   enable = 1;
22    #1   enable = 0;
23    #10  enable = 1;
24    #1   enable = 0;
25    #10  $finish;
26 end
27
28 always
```

```
29   #1  clk = ~clk;
30
31 endmodule
```

You could download file edge_wait_example.v here

Simülatör Çıktısı

```
TIME : 0 CLK : 0 ENABLE : 0 TRIGGER : x
TIME : 1 CLK : 1 ENABLE : 0 TRIGGER : x
TIME : 2 CLK : 0 ENABLE : 0 TRIGGER : x
TIME : 3 CLK : 1 ENABLE : 0 TRIGGER : x
TIME : 4 CLK : 0 ENABLE : 0 TRIGGER : x
TIME : 5 CLK : 1 ENABLE : 1 TRIGGER : 0
TIME : 6 CLK : 0 ENABLE : 0 TRIGGER : 0
TIME : 7 CLK : 1 ENABLE : 0 TRIGGER : 0
TIME : 8 CLK : 0 ENABLE : 0 TRIGGER : 0
TIME : 9 CLK : 1 ENABLE : 0 TRIGGER : 0
TIME : 10 CLK : 0 ENABLE : 0 TRIGGER : 0
TIME : 11 CLK : 1 ENABLE : 0 TRIGGER : 0
TIME : 12 CLK : 0 ENABLE : 0 TRIGGER : 0
TIME : 13 CLK : 1 ENABLE : 0 TRIGGER : 0
TIME : 14 CLK : 0 ENABLE : 0 TRIGGER : 0
TIME : 15 CLK : 1 ENABLE : 0 TRIGGER : 1
TIME : 16 CLK : 0 ENABLE : 1 TRIGGER : 0
TIME : 17 CLK : 1 ENABLE : 0 TRIGGER : 0
TIME : 18 CLK : 0 ENABLE : 0 TRIGGER : 0
TIME : 19 CLK : 1 ENABLE : 0 TRIGGER : 0
TIME : 20 CLK : 0 ENABLE : 0 TRIGGER : 0
TIME : 21 CLK : 1 ENABLE : 0 TRIGGER : 0
TIME : 22 CLK : 0 ENABLE : 0 TRIGGER : 0
TIME : 23 CLK : 1 ENABLE : 0 TRIGGER : 0
TIME : 24 CLK : 0 ENABLE : 0 TRIGGER : 0
TIME : 25 CLK : 1 ENABLE : 0 TRIGGER : 1
TIME : 26 CLK : 0 ENABLE : 0 TRIGGER : 1
```

Seviye Hassasiyetli Olay Kontrolü (Wait ifadeleri)

Gecikmenin bir sonraki ifadenin gerçekleştirilmesi <deyim> doğru olana kadardır

sözdizim: wait (<deyim>) < ifade>;

Örnek – Seviye Beklemesi(Level Wait)

```
 1 module wait_example();
 2
 3 reg mem_read, data_ready;
 4 reg [7:0] data_bus, data;
 5
 6 always @ (mem_read or data_bus or data_ready)
 7 begin
 8   data = 0;
 9   while (mem_read == 1'b1) begin
10     // #1 sonsuz döngüyü önlemek için çok önemlidir
11     wait (data_ready == 1)  #1  data = data_bus;
12   end
13 end
14
15 // Testbenç kodu buradan başlar
```

```
16 initial begin
17  $monitor ("TIME = %g READ = %b READY = %b DATA = %b",
18    $time, mem_read, data_ready, data);
19  data_bus = 0;
20  mem_read = 0;
21  data_ready = 0;
22   #10  data_bus = 8'hDE;
23   #10  mem_read = 1;
24   #20  data_ready = 1;
25   #1   mem_read = 1;
26   #1   data_ready = 0;
27   #10  data_bus = 8'hAD;
28   #10  mem_read = 1;
29   #20  data_ready = 1;
30   #1   mem_read = 1;
31   #1   data_ready = 0;
32   #10  $finish;
33 end
34
35 endmodule
```

You could download file wait_example.v here

Simülatör Çıktısı

```
TIME = 0  READ = 0 READY = 0 DATA = 00000000
TIME = 20 READ = 1 READY = 0 DATA = 00000000
TIME = 40 READ = 1 READY = 1 DATA = 00000000
TIME = 41 READ = 1 READY = 1 DATA = 11011110
TIME = 42 READ = 1 READY = 0 DATA = 11011110
TIME = 82 READ = 1 READY = 1 DATA = 11011110
TIME = 83 READ = 1 READY = 1 DATA = 10101101
TIME = 84 READ = 1 READY = 0 DATA = 10101101
```

İçe Atama Zaman Kontrolleri(Intra-Assignment Timing Controls)

İçe atama(Intra-assignment) kontrolleri sağ taraftaki ifadeyi herzaman hazır hale getirir ve gecikme veya olay kontrolünden sonra sonucu atar.

İçe olmayan atamalarda(non-intra-assignment)kontroller(sol taraftaki gecikme veya olay kontolü), sağ taraftaki ifade gecikme ve olay kontrollerinden sonra hazır hale getirilir.

Örnek – İçe Atama (Intra-Assignment)

```
 1 module intra_assign();
 2
 3 reg a, b;
 4
 5 initial begin
 6   $monitor("TIME = %g  A = %b  B = %b",$time, a , b);
 7   a = 1;
 8   b = 0;
 9   a = #10 0;
10   b = a;
11    #20  $display("TIME = %g  A = %b  B = %b",$time, a , b);
12   $finish;
13 end
14
```

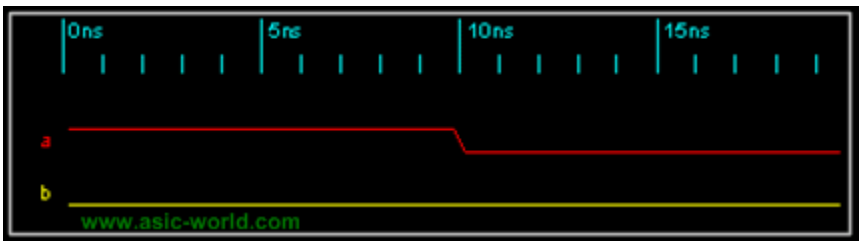
```
15 endmodule
```

You could download file intra_assign.v here

Simülasyon Çıktısı

```
TIME = 0   A = 1  B = 0
TIME = 10  A = 0  B = 0
TIME = 30  A = 0  B = 0
```

Dalgaformu



Sürekli Atamalar(Continuous Assignment) ile Çoklu Mantık(Combo Logic) Modelleme

Sağ taraftaki herhangi bir sinyal değiştiğinde, bütün sağ taraf yeniden değerlendirilir ve sonuç sol tarafa atanır.

Örnek – Üç-durumlu Arabellek(Tri-state Buffer)

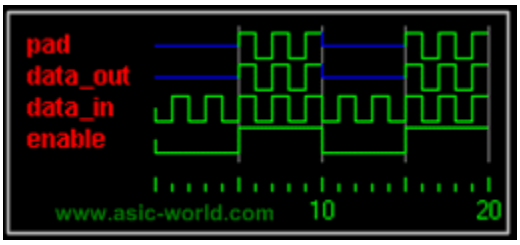
```
 1 module tri_buf_using_assign();
 2 reg data_in, enable;
 3 wire pad;
 4
 5 assign pad = (enable) ? data_in : 1'bz;
 6
 7 initial begin
 8   $monitor ("TIME = %g ENABLE = %b DATA : %b PAD %b",
 9     $time, enable, data_in, pad);
10    #1  enable = 0;
11    #1  data_in = 1;
12    #1  enable = 1;
13    #1  data_in = 0;
14    #1  enable = 0;
15    #1  $finish;
16 end
17
18 endmodule
```

You could download file tri_buf_using_assign.v here

Simülasyon Çıktısı

```
TIME = 0 ENABLE = x DATA : x PAD x
TIME = 1 ENABLE = 0 DATA : x PAD z
TIME = 2 ENABLE = 0 DATA : 1 PAD z
TIME = 3 ENABLE = 1 DATA : 1 PAD 1
TIME = 4 ENABLE = 1 DATA : 0 PAD 0
TIME = 5 ENABLE = 0 DATA : 0 PAD z
```

Dalgaformu(**Waveform)**



Örnek- Çoklayıcı(Mux)

```
 1 module mux_using_assign();
 2 reg data_in_0, data_in_1;
 3 wire data_out;
 4 reg  sel;
 5
 6 assign data_out = (sel) ? data_in_1 : data_in_0;
 7
 8 // Testbenç kodu buradan başlar
 9 initial begin
10   $monitor("TIME = %g SEL = %b DATA0 = %b DATA1 = %b OUT = %b",
11     $time,sel,data_in_0,data_in_1,data_out);
12   data_in_0 = 0;
13   data_in_1 = 0;
14   sel = 0;
15    #10  sel = 1;
16    #10  $finish;
17 end
18
19 // Toggle(çöğünmek) data_in_0 #1
20 always
21   #1  data_in_0 = ~data_in_0;
22
23 // Toggle(çöğünmek) data_in_1 #2
24 always
25   #2  data_in_1 = ~data_in_1;
26
27 endmodule
```

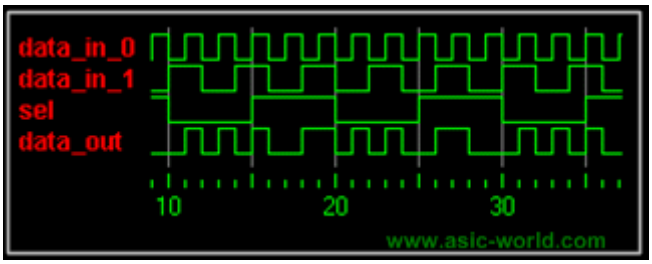
You could download file mux_using_assign.v here

Simülasyon Çıktısı

```
TIME = 0 SEL = 0 DATA0 = 0 DATA1 = 0 OUT = 0
TIME = 1 SEL = 0 DATA0 = 1 DATA1 = 0 OUT = 1
TIME = 2 SEL = 0 DATA0 = 0 DATA1 = 1 OUT = 0
TIME = 3 SEL = 0 DATA0 = 1 DATA1 = 1 OUT = 1
TIME = 4 SEL = 0 DATA0 = 0 DATA1 = 0 OUT = 0
TIME = 5 SEL = 0 DATA0 = 1 DATA1 = 0 OUT = 1
TIME = 6 SEL = 0 DATA0 = 0 DATA1 = 1 OUT = 0
TIME = 7 SEL = 0 DATA0 = 1 DATA1 = 1 OUT = 1
TIME = 8 SEL = 0 DATA0 = 0 DATA1 = 0 OUT = 0
TIME = 9 SEL = 0 DATA0 = 1 DATA1 = 0 OUT = 1
TIME = 10 SEL = 1 DATA0 = 0 DATA1 = 1 OUT = 1
TIME = 11 SEL = 1 DATA0 = 1 DATA1 = 1 OUT = 1
TIME = 12 SEL = 1 DATA0 = 0 DATA1 = 0 OUT = 0
TIME = 13 SEL = 1 DATA0 = 1 DATA1 = 0 OUT = 0
TIME = 14 SEL = 1 DATA0 = 0 DATA1 = 1 OUT = 1
TIME = 15 SEL = 1 DATA0 = 1 DATA1 = 1 OUT = 1
```

```
TIME = 16 SEL = 1 DATA0 = 0 DATA1 = 0 OUT = 0
TIME = 17 SEL = 1 DATA0 = 1 DATA1 = 0 OUT = 0
TIME = 18 SEL = 1 DATA0 = 0 DATA1 = 1 OUT = 1
TIME = 19 SEL = 1 DATA0 = 1 DATA1 = 1 OUT = 1
```

Dalgaformu (Waveform)



Görev (Task) ve Fonksiyonlar(Functions)

Görev (Task)

Görevler tüm programlama dillerinde kullanılırlar, genellikle prosedür veya altprogram (subroutine) olarak bilinirler. Kod satırları **task…end task** arasına alınırlar. Veri göreve gönderilir, işlem bittikten sonra, sonuç döndürülür. Bunlar özel olarak veri girişleri(data ins) veri çıkışları(data outs) yada netlistdeki tel(wire) olarak isimlendirilmelidir. Kodun ana gövdesine eklenmiştir, birden çok kez çağırılabilinir, böylece kod tekrarlarından kaçınılmış olunur.

- Görevler(tasks) modülde kullanıldığı yerde tanımlanmıştır. Bir görevi farklı dosyalarda da tanımlamak mümkündür ve görevin olduğu derleme yönergesine göre eklenebilir.
- Görevler, posedge, negedge, # delay ve wait gibi zamanlama gecikmelerini(timing delays), içerebilir.
- Görevler birçok sayıda giriş veya çıkışlara sahip olabilirler.
- Değişkenler görevlerin içinde bildirilirler ve sadece bu görev içinde görülebilir yani yerel değişkendir. Görev içindeki bildirim sırası görev çağırıldığında değişkenlerin göreve nasıl gönderileceğini tanımlamaktadır.
- Görevler yerel değişkenler kullanılmadığı zaman global değişkenleri alabilir, kullanabilir ve kaynak olarak kullanabilir. Yerel değişkenler kullanıldığında, temelde çıkış sadece görev gerçekleştirilmesinin sounda atanır.
- Görevler başka görevleri veya fonksiyonları çağırabilir.
- Görevler kombinasyonel ve ardışıl lojik modelleme için kullanılabilir.
- Bir görev özellikle bir ifade ile çağırılmalıdır, fonksiyonda olduğu gibi bir ifadenin içinde kullanılamaz.

Sözdizim

- Bir görev **task** anahtar sözcüğüyle başlar ve **endtask** anahtar sözcüğüyle biter.
- Girişler ve çıkışlar task anahtar sözcüğünden sonra bildirilirler.
- Yerel değişkenler giriş ve çıkışlar bildiriminde sonra bildirilirler.

Örnek – Basit bir Görev

```
 1 module simple_task();
 2
 3 task convert;
 4 input [7:0] temp_in;
 5 output [7:0] temp_out;
 6 begin
 7   temp_out = (9/5) *( temp_in + 32)
 8 end
 9 endtask
10
11 endmodule
```

You could download file simple_task.v here

Örnek – Global değişkenler kullanılan görevler

```
 1 module task_global();
 2
 3 reg [7:0] temp_out;
 4 reg [7:0] temp_in;
 5
 6 task convert;
 7 begin
 8   temp_out = (9/5) *( temp_in + 32);
 9 end
10 endtask
11
12 endmodule
```

You could download file task_global.v here

Bir Görevin Çağrılması (Calling a Task)

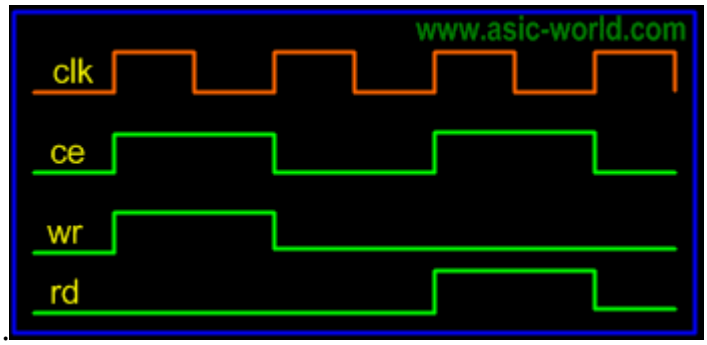
Varsayalım ki örnek 1 deki görev mytask.v adlı dosyada tutulsun. Bir görevin ayrı bir dosyada kodlanmasının avantajı bu bu görevin birçok modülde kullanılabilmesidir.

```
 1 module  task_calling (temp_a, temp_b, temp_c, temp_d);
 2 input [7:0] temp_a, temp_c;
 3 output [7:0] temp_b, temp_d;
 4 reg [7:0] temp_b, temp_d;
 5 `include "mytask.v"
 6
 7 always @ (temp_a)
 8 begin
 9   convert (temp_a, temp_b);
10 end
11
12 always @ (temp_c)
13 begin
14   convert (temp_c, temp_d);
15 end
16
17 endmodule
```

You could download file task_calling.v here

Örnek- CPU Yazma/Okuma Görevi (Write / Read Task)

Aşağıda belleğe yazma ve bellekten okuma için kullanılan dalgaformu vardır. Şimdi bu read/write(okuma/yazma)'ı görev olarak yazacağız




```
 1 module bus_wr_rd_task();
 2
 3 reg clk,rd,wr,ce;
 4 reg [7:0]  addr,data_wr,data_rd;
 5 reg [7:0]  read_data;
 6
 7 initial begin
 8   clk = 0;
 9   read_data = 0;
10   rd = 0;
11   wr = 0;
12   ce = 0;
13   addr = 0;
14   data_wr = 0;
15   data_rd = 0;
16   // yazma(write) ve okuma(read) görevleri burada çağırılmaktadır
17    #1  cpu_write(8'h11,8'hAA);
18    #1  cpu_read(8'h11,read_data);
19    #1  cpu_write(8'h12,8'hAB);
20    #1  cpu_read(8'h12,read_data);
21    #1  cpu_write(8'h13,8'h0A);
22    #1  cpu_read(8'h13,read_data);
23    #100  $finish;
24 end
25 // Saat üreteci (Clock Generator)
26 always
27    #1  clk = ~clk;
28 // CPU Okuma(Read) Görevi
29 task cpu_read;
30   input [7:0]  address;
31   output [7:0] data;
32   begin
33     $display ("%g CPU Read  task with address : %h", $time, address);
34     $display ("%g  -> Driving CE, RD and ADDRESS on to bus", $time);
35     @ (posedge clk);
36     addr = address;
37     ce = 1;
38     rd = 1;
39     @ (negedge clk);
40     data = data_rd;
41     @ (posedge clk);
42     addr = 0;
```

```
43      ce = 0;
44      rd = 0;
45      $display ("%g CPU Read  data              : %h", $time, data);
46      $display ("======================");
47    end
48  endtask
49  // CPU Yazma(Write) Görevi
50  task cpu_write;
51    input [7:0]  address;
52    input [7:0] data;
53    begin
54      $display ("%g CPU Write task with address : %h Data : %h",
55        $time, address,data);
56      $display ("%g  -> Driving CE, WR, WR data and ADDRESS on to bus",
57        $time);
58      @ (posedge clk);
59      addr = address;
60      ce = 1;
61      wr = 1;
62      data_wr = data;
63      @ (posedge clk);
64      addr = 0;
65      ce = 0;
66      wr = 0;
67      $display ("======================");
68    end
69  endtask
70
71  // Görevleri test etmek için bellek modeli
72  reg [7:0] mem [0:255];
73
74  always @ (addr or ce or rd or wr or data_wr)
75  if (ce) begin
76    if (wr) begin
77      mem[addr] = data_wr;
78    end
79    if (rd) begin
80      data_rd = mem[addr];
81    end
82  end
83
84  endmodule
```

You could download file bus_wr_rd_task.v here

Simülasyon Çıktısı

```
1 CPU Write task with address : 11 Data : aa
1  -> Driving CE, WR, WR data and ADDRESS on to bus
======================
4 CPU Read  task with address : 11
4  -> Driving CE, RD and ADDRESS on to bus
7 CPU Read  data              : aa
======================
8 CPU Write task with address : 12 Data : ab
8  -> Driving CE, WR, WR data and ADDRESS on to bus
======================
12 CPU Read  task with address : 12
12  -> Driving CE, RD and ADDRESS on to bus
15 CPU Read  data              : ab
======================
```

```
16 CPU Write task with address : 13 Data : 0a
16  -> Driving CE, WR, WR data and ADDRESS on to bus
======================
20 CPU Read  task with address : 13
20  -> Driving CE, RD and ADDRESS on to bus
23 CPU Read  data              : 0a
======================
```

Fonksiyon(Function)

Bir Verilog HDL fonksiyonu bir görevle(task) aynıdır, çok ufak farklılıklar vardır, örneğin örneğin fonksiyon birden fazla çıkış süremez, gecikme içeremez.

- Fonksiyonlar kullanılacakları modülde tanımlanırlar. Fonksiyonlar ayrı dosyalarda tanımlanabilir ve derleme yönergesine göre görevler örneklenerek istenilen fonksiyon eklenir.
- Fonksiyonlar **zamanlama gecikmelerini içeremez,** posedge,negedge, #delay 'de olduğu gibi, bunun anlamı fonksiyonlar "sıfır" zaman gecikmesiyle gerçekleştirilir.
- Fonksiyonlar istediği sayıda girişe sahip olabilir ancak sadece bir tane çıkış içerebilir.
- Fonksiyon içerisinde bildirilen değişkenler sadece bu fonksiyon içinde görülebilir yerel değişkenlerdir. Fonksiyon içindeki bildirim sırası değişkenlerin kullanan tarafından nasıl gönderileceğini tanımlar.
- Fonksiyonlar yerel değişkenler kullanılmadığı zaman global değişkenler alınabilir, kullanılabilir ve kaynak olarak kullanılabilir. Yerel değişkenler kullanıldığı zaman temelde çıkış sadece fonksiyon gerçekleştirildikten sonra atanır.
- Fonksiyonlar **kombinasyonel lojiği modellemek** için kullanılabilir.
- Fonksiyonlar diğer **fonksiyonları çağırabilir ancak görevleri çağıramazlar.**

Sözdizim

- Bir fonksiyon **function** anahtar sözcüğü ile başlamaktadır ve **endfunction** fonksiyonu ile sonlandırılmaktadır.
- **Girişler(inputs)** function anahtar sözcüğünden sonra bildirilmektedir.

Örnek- Basit bir fonksiyon

```
 1 module simple_function();
 2
 3 function  myfunction;
 4 input a, b, c, d;
 5 begin
 6   myfunction = ((a+b) + (c-d));
 7 end
 8 endfunction
 9
10 endmodule
```

You could download file simple_function.v here

Örnek- Bir fonksiyonun çağırılması

```
 1 module  function_calling(a, b, c, d, e, f);
 2
 3 input a, b, c, d, e ;
 4 output f;
 5 wire f;
 6 `include "myfunction.v"
 7
 8 assign f =  (myfunction (a,b,c,d)) ? e :0;
 9
10 endmodule
```

You could download file function_calling.v here

Sistem Görev ve Fonksiyonları (System Task and Function)

Giriş

Simülasyon sırasında giriş ve çıkış üretmek için görevler(tasks) ve fonksiyonlar(functions) vardır. Bunların isimleri dolar işaretiyle ($) başlar. Sentez araçları sistem fonksiyonlarını ayrıştırır ve gözardı eder, ve bundan dolayı sentezlenebilir modellere bile eklenebilir.

$display, $strobe, $monitor

Bu komutlar aynı sözdizimine sahiptir ve simülasyon esnasında metni ekranda gösterir. GTKWave, Undertow veya Debussy dalgaformu gösterme aracından daha az kullanışlıdır. $display ve $strobe gerçekleştirildi her seferinde bir kez ekranda gösterir ancak $monitor parametrelerinden birinin değiştiği her seferde ekranda gösterir. $display ile $strobe arasındaki fark $strobe gerçekleştiği andan ziyade mevcut simülsyon zaman birimin en sonunda parametrelerini gösterir. Dizgi(string) formatı C/C++'dakinin aynıdır, ve biçimleme karakterlerini içerebilir. Biçimlendirme karakterleri %d (onluk), %h (onaltılık), %b (ikili), %c (karakter), %s (dizgi-string) and %t (zaman), %m (hiyerarşi seviyesi)'ni içerir. %5d, %5b, vb 5 karakterlik boşluklara yazar. Sona eklenen b,h,o görev isimleri olarak varsayılan biçimi binary(ikili), octal(sekizli) veya hexadecimal(onaltılı) biçime dönüştürür.

Sözdizimi

- $display ("format_string", par_1, par_2, ...);
- $strobe ("format_string", par_1, par_2, ...);
- $monitor ("format_string", par_1, par_2, ...);
- $displayb (as above but defaults to binary..);
- $strobeh (as above but defaults to hex..);
- $monitoro (as above but defaults to octal..);

$time, $stime, $realtime

Bunlar mevcut simülasyon zamanlarını sırasıyla 64-bit'lik tamsayı, bir 32-bit'lik tamsayı ve bir reelsayı olarak döndürür.

$reset, $stop, $finish

$reset simülasyon zamanını 0'a döndürerek sıfırlar; $stop simülatörü durdurur ve etkileşimli moda geçirerek kullanıcıya komut girmesi sağlanır; $finish simülatörden çıkarak işletim sistemine dönmeyi sağlar.

$scope, $showscope

$scope(hiyerarşi_adı) mevcut hiyerarşik kapsamı hiyerarşi_adı olarak atar. $showscopes(n) mevcut kapsamda tüm modülleri, görevleri ve blok isimlerini listeler.

$random

$random çağırıldığı her anda rasgele bir tamsayı üretir. Eğer sıra tekrarlanabilirse, ilk olarak bir sayısal argüman (seed) verilir. Diğer türlü "seed" bilgisayarın saatinden türetilir.

$dumpfile, $dumpvar, $dumpon, $dumpoff, $dumpall

Debussy gibi simülasyon izleyicisine değişkenlerin değişimini döker. Döküm dosyaları bir simülasyondaki tüm değişkenleri dökebilir durumdadır. Bu hata ayıklama(debugging) için kullanışlıdır ancak çok yavaş olabilir.

Sözdizimi

- $dumpfile("dosyaismi.vcd")
- $dumpvar tasarımdaki tüm değişkenleri(variable) döker
- $dumpvar(1, top) en üst ve bir altındaki modüldeki tüm değişkenleri döker, fakat en üstteki örneklenmiş modüllerin dökümü yapılmaz.
- $dumpvar(2, top) en üstteki ve bir seviye altındaki modülün değişkenlerin dökümü yapılır.
- $dumpvar(n, top) en üstteki ve n-1 seviye alttaki modülün değişkenlerin dökümü yapılır.
- $dumpvar(0, top) en üstteki modül ve altındaki tüm seviyelerin değişkenlerin dökümü yapılır.
- $dumpon dökümü başlatır.
- $dumpoff dökümü durdurur.

$fopen, $fdisplay, $fstrobe $fmonitor and $fwrite

Bu komutlar daha seçici olarak dosyalara yazar.

- $fopen bir çıkış dosyası açar ve bu açık dosyaya diğer komutlar tarafından ulaşmaya imkan sağlar.
- $fclose dosyayı kapatır böylece diğer programlar bu dosyaya ulaşabilir.
- $fdisplay ve $fwrite bir dosyaya biçimli yazmayı sağlar.Birbiriyle aynıdır fakat $fdisplay herbir gerçekleştirildiğinde yeni bir satır eklerken, $write eklemez.
- $strobe da gerçekleştirildiğinde dosyaya yazar fakat yazmadan önce zaman adımındaki tüm işlemlerin tamamlanmasını bekler. Bu nedenle başlangıçta #1 a=1; b=0; $fstrobe(hand1, a,b); b=1; dosyaya a ve b değerleri yazılacaksa 1 1 yazar.
- $monitor ‘ün argümanlarından biri değiştiğinde dosyaya yazar.

Sözdizimi

- handle1=$fopen("dosyaadı1.uzantısı ")
- handle2=$fopen("dosyaadı2.uzantısı ")
- $fstrobe(handle1, biçim, değişken listesi) // dosyaadı1.uzantısı ‘daki veriyi işaretler(strobe)
- $fdisplay(handle2, biçim, değişken listesi) //veriyi dosyaadı2.uzantısı dosyasına yazar
- $fwrite(handle2, biçim , değişken listesi) // veriyi dosyaadı2.uzantısı dosyasına hepsi bir satırda olacak şekilde yazar. Yeni bir satır gerektiğinde biçim dizgisini koyun.

TestBenç Yazma Sanatı

Giriş

Bir testbenç yazmak RTL kodun kendisini yazmak kadar karmaşıktır. Bu günlerde ASIC’ler gün geçtikçe daha da karmaşık bir hal almakta ve bu nedenle bu karmaşık ASIC bir meydan okumaya döndü. Tipik olarak 60-70%’lik zaman ASIC’de doğrulama(verification) /geçerleme(validation)/test(testing) için harcanır. Bu birçok ASIC mühendisi tarafından iyi bilinmesine rağmen , hala mühendisler geçerleme mutlu ettiğini söyleyemezler.

Bazı örnekleri VLSI sınıfından 1999-2001 arasında aldım. Lütfen aşağıdaki eğitselin sizi nasıl geliştireceği konusunda kendinizi özgür hissedin.

Başlamadan Önce

Testbenç yazmak için önemli şey DUT("design under test"- test altında tasarım yapmak) tasarım belirtimine sahip olmaktır. Belirtimler açıkça anlaşılmış olmalı ve test benç mimarisinin ve test senaryosunun(test durumlarını) temel dokümanlarının detaylı olarak yapılması gerekmektedir.

Örnek – Sayaç(Counter)

Varsayalım ki basit bir 4-bit'lik yukarı sayaç yapacağız, enable(etkin) yüksek olduğu sürece sayacak ve reset(sıfırlama) yüksek olduğunda sayaç sıfırlanacak yani sıfıra gelecek. Reset saatle senkron şekilde çalışmaktadır.

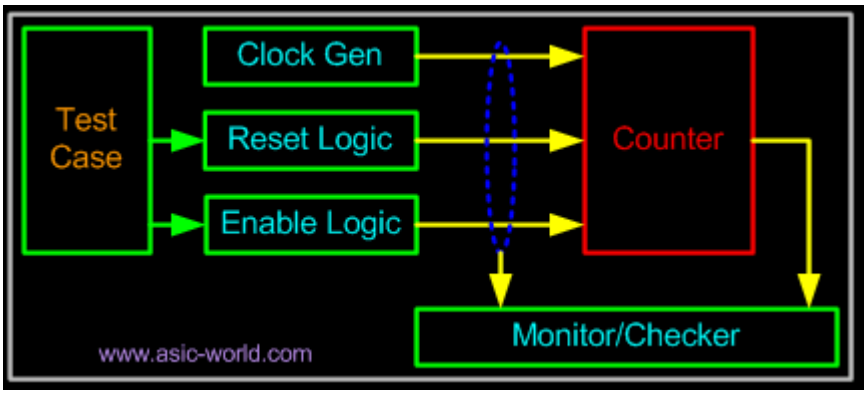
Code for Counter

```
 1 //----------------------------------------------------
 2 // Tasarım İsmi: counter(sayaç)
 3 // Dosya İsmi  : counter.v
 4 // Fonksiyon   : 4 bit yukarı doğru sayaç(up counter)
 5 // Kodlayan    : Deepak
 6 //----------------------------------------------------
 7 module counter (clk, reset, enable, count);
 8 input clk, reset, enable;
 9 output [3:0] count;
10 reg [3:0] count;
11
12 always @ (posedge clk)
13 if (reset == 1'b1) begin
14   count <= 0;
15 end else if ( enable == 1'b1) begin
16   count <= count + 1;
17 end
18
19 endmodule
```

You could download file counter.v here

Test Planı

Kendi kendini test eden bir testbenç yazacağız fakat bunu sizin anlamanıza yardımcı olsun diye adım adım yapacağız böylece otomatik test benç yazım şeklini anlamış olacaksınız. Bizim testbenç ortamı aşağıdaki şekildeki gibi görünmektedir.



DUT testbençte örneklenir ve testbenç bir saat üreteci,reset(sıfırlama) üreteci,enable(etkin) lojik üreteci ve temel olarak sayacın beklenen değerini hesaplayan ve bunu sayacın çıkışındakiyle karşılaştıran bir karşılaştırma lojiği içerir.

Test Durumları

- Reset(sıfırlama) Testi : reset(sıfırlama-yeniden başlatma) olmadan başlanır daha sonra birkaç saat darbesi boyunca resetlenir ve reset bırakılır. Daha sonra çıkışın sıfır atanıp atanmadığına bakılır.
- Enable(etkinlin) Testi : Sıfırlamadan sonra etkinkleştirme yapılır ve iptal edilir.
- Son olarak rasgele şekilde enable ve reset aktifleştirilir ve iptal edilir.

Bundan fazla test durumları ekleyebiliriz fakat burada sadece sayacı test etmiyoruz test benç nasıl yazılır onu öğreniyoruz.

Bir Testbenç Yazma

Herhangi bir testbenç yaratmanın ilk adımı saçma da olsa DUT'a girişleri reg olarak bildiren ve DUT'dan çıkışları bir wire olarak alan bir şablon oluşturmaktır, daha sonra aşağıda gösterildiği gibi DUT örneklenmelidir. Not,testbençde port listesi yoktur.

Test Benç

```
 1 module counter_tb;
 2   reg clk, reset, enable;
 3   wire [3:0] count;
 4
 5   counter U0 (
 6   .clk    (clk),
 7   .reset  (reset),
 8   .enable (enable),
 9   .count  (count)
10   );
11
12 endmodule
```

You could download file counter_tb1.v here

Bir sonraki adım saat üreteci lojiği eklemek olabilir: bu dosdoğru yapılır, ki biz bir saat üreteci nasıl yapılır biliyoruz. Bir saat üreteci eklemeden önce DUT'ın tüm girişlerini aşağıdaki kodda olduğu gibi bilinen bir durumda sürmeliyiz.

Saat Üreteçli Test Benç

```
 1 module counter_tb;
 2   reg clk, reset, enable;
 3   wire [3:0] count;
 4
 5   counter U0 (
 6   .clk    (clk),
 7   .reset  (reset),
 8   .enable (enable),
 9   .count  (count)
10   );
11
12   initial
13   begin
14     clk = 0;
15     reset = 0;
```

```
16      enable = 0;
17    end
18
19    always
20       #5  clk =  ! clk;
21
22  endmodule
```

You could download file counter_tb2.v here

Bir başlangıç(initial) bloğu Verilogda sadece birkez gerçekleştirilir, böylece simülatör clk, reset ve enable değerlerini 0 olarak atar; sayaç koduna bakacak olursak(elbette DUT belirtimine uygun olması tercih edilir) hepsinin 0 olarak sürülmesi tüm bu sinyallerin etkisiz kılar.

Bir sat üretecini üretmenin birçok yolu vardır: bunlardan biri de aşağıdaki koda alternatif olarak kullanılabilecek başlangıç bloğunun içinde forever(herzaman) döngüsü kullanmaktır. Bir parametre ekleyebilirsiniz ya da saat frekansını kontrol etmek için tanım kullanabilirsiniz. Karmaşık bir saat üreteci yazabilirsiniz, buna PPM(parts per million-milyonlarca parçaya bölme,kaydırarak saatleme) deriz, daha sonra da görev döngüsünü(duty cycle) kontrol edersiniz. Yukarıda belirtilenlerin hepsi DUT'daki belirtime ve Test Benç Tasarımcısının yaratıcılığına bağlıdır.

Bu noktada, testbencinizin saati doğru üretip üretmediğini test etmek isteyebilirsiniz: güzel bunu herhangi bir Verilog simülatöründe derleyebilirsiniz. Komut satırı seçeneklerini aşağıda gösterildiği şekilde vermelisiniz.

C:\www.asic-world.com\veridos counter.v counter_tb.v

Elbette modül ismiyle dosya ismini aynı tutmak güzel bir fikirdir. Peki, derlemeye geri dönelim, göreceksiniz ki simülatör ekranda herhangi bir şey yazabilir yada ekrana bir herhangi bir dalgaformu dökebilir. Bu nedenle aşağıdaki kodda da gösterildiği gibi destek eklememiz gerekmektedir.

Test Benç'e devam…

```
 1 module counter_tb;
 2   reg clk, reset, enable;
 3   wire [3:0] count;
 4
 5   counter U0 (
 6   .clk    (clk),
 7   .reset  (reset),
 8   .enable (enable),
 9   .count  (count)
10   );
11
12   initial begin
13     clk = 0;
14     reset = 0;
15     enable = 0;
16   end
17
18   always
19      #5  clk =  ! clk;
20
```

```
21   initial  begin
22     $dumpfile ("counter.vcd");
23     $dumpvars;
24   end
25
26   initial  begin
27     $display("\t\ttime,\tclk,\treset,\tenable,\tcount");
28     $monitor("%d,\t%b,\t%b,\t%b,\t%d",$time, clk,reset,enable,count);
29   end
30
31   initial
32    #100  $finish;
33
34   //Testbençin geri kalanı bu satırdan sonra gelecek
35
36 endmodule
```

You could download file counter_tb3.v here

$dumpfile simülatörün daha sonra kaydetmek için kullanacağı dalgaformunu belirtecek dosya için kullanılmıştır, bu dalgaformu daha sonra bir dalgaformu göstericisi tarafından kullanılabilir. (Lütfen araçlar bölümündeki ücretsiz gösterici araçlarını tercih ediniz.) $dumpvars temelde Verilog derleyicisine "counter.vcd"tüm sinyalleri boşaltma talimatı verir.

$display metin veya değişkenleri stdout(ekranda) yazdırmak için kullanılır, \t boşluk eklemek için kullanılır. Sözdizim C dilindeki printf ile aynıdır. İkinci satırdaki $monitor birazcık farklıdır: $monitor listelenen(clk,reset,enable,count) değişkenlerin değişimlerinin izlerini tutar. Bunlardan herhangi biri değiştiğinde, bunların değerlerini belirtilen kök yapısına uyarak yazar.

$finish simülasyonu #100 zaman biriminde sonlandırmak için kullanılır(not: tüm başlangıç(initial) ve always(herzaman) blokları 0 zamanında gerçekleştirilmeye başlar).

Şimdi temel iskeleyi yazmış bulunmaktayız, bunu derleyelim(compile) ve ne kodladığımızı görelim. Simülatörün çıktısı aşağıdaki gibi olmalıdır.

```
C:\www.asic-world.com>veridos counter.v counter_tb.v
VeriWell for Win32 HDL Version 2.1.4 Fri Jan 17 21:33:25 2003

This is a free version of the VeriWell for Win32 Simulator
Distribute this freely; call 1-800-VERIWELL for ordering information
See the file "!readme.1st" for more information

Copyright (c) 1993-97 Wellspring Solutions, Inc.
All rights reserved


Memory Available: 0
Entering Phase I...
Compiling source file : counter.v
Compiling source file : counter_tb.v
The size of this model is [2%, 5%] of the capacity of the free version

Entering Phase II...
Entering Phase III...
No errors in compilation
Top-level modules:
counter_tb
```

```
    time      clk,    reset,  enable, count
    0,                0,      0,      0,      x
    5,                1,      0,      0,      x
    10,       0,      0,      0,      x
    15,       1,      0,      0,      x
    20,       0,      0,      0,      x
    25,       1,      0,      0,      x
    30,       0,      0,      0,      x
    35,       1,      0,      0,      x
    40,       0,      0,      0,      x
    45,       1,      0,      0,      x
    50,       0,      0,      0,      x
    55,       1,      0,      0,      x
    60,       0,      0,      0,      x
    65,       1,      0,      0,      x
    70,       0,      0,      0,      x
    75,       1,      0,      0,      x
    80,       0,      0,      0,      x
    85,       1,      0,      0,      x
    90,       0,      0,      0,      x
    95,       1,      0,      0,      x

Exiting VeriWell for Win32 at time 100
0 Errors, 0 Warnings, Memory Used: 0
Compile time = 0.0 Load time = 0.0 Simulation time = 0.1

Normal exit
Thank you for using VeriWell for Win32
```

Reset(Sıfırlama)Lojiğinin Eklenmesi

Bir kere bize testbencimizin ne yapacağını gösteren temel lojiğe sahipsek, daha sonra reset (sıfırlama) lojiğini ekleyebiliriz. Eğer test durumlarına bakarsak, simülasyon esnasında herhangi bir zamanda reset'i aktif hale getirebileceğimiz kısıtını eklediğimizi görürüz. Bunu sağlamak için birçok yaklaşımımız vardır, fakat ben size uzun süre gidecek bir şey öğretmek istiyorum. Verilog'da "events"(olaylar) adı verilen birşeyler vardır: olaylar tetiklenebilir ve ayrıca da gözlemlenebilir, bunu görmemiz için olayın gerçekleşmesi gerekir.

Şimdi bizim reset lojiğimizi belirttiğimiz şekilde tetikleyici olayı bekleyecek şekilde "reset_trigger" kodlayalım: bu olay gerçekleştiğinde reset lojiği saatin negatif kenarında olduğu varsayılır ve bir sonraki negatif kenarda deaktif duruma geçtiğini aşağıdaki kodda da görebilirsiniz. Ayrıca reset'i aktif olmayan duruma getirdikten sonra, reset lojiği başka bir olayı "reset_done_trigger" çağırır. Bu tetikleme olayı testbençteki sync up için kullanılabilir.

Reset lojiği için kod

```
1 event reset_trigger;
2   event  reset_done_trigger;
3
4   initial begin
5     forever begin
6       @ (reset_trigger);
7       @ (negedge clk);
8       reset = 1;
9       @ (negedge clk);
```

```
10        reset = 0;
11        -> reset_done_trigger;
12      end
13    end
```

You could download file counter_tb4.v here

Test durumu lojiğinin eklenmesi

İleri giderek, test durumlarını üretmek için lojik ekleyelim, tamam bu eğitimin ilk bölümünde bizim üç tane testdurumumuz var. Hadi şimdi bunları yeniden listeleyelim ☺

- Reset(sıfırlama) Testi : reset(sıfırlama-yeniden başlatma) olmadan başlanır daha sonra birkaç saat darbesi boyunca resetlenir ve reset bırakılır. Daha sonra çıkışın sıfır atanıp atanmadığına bakılır.
- Enable(etkinlin) Testi : Sıfırlamadan sonra etkinkleştirme yapılır ve iptal edilir.
- Son olarak rasgele şekilde enable ve reset aktifleştirilir ve iptal edilir.

Yeniden tekrarlayacak olursak: bir test durumunu test etmek için birçok yol olabilir, bu tamamen Test benç tasarımcının yaratıcılığına kalmıştır. Hadi şimdi basit bir uygulamayı ele alalım ve bunu yavaşça temelini oluşturalım.

Test Durumu 1- Asserting/ De-asserting reset(etkinleştirilebilir/etkinleştirilemez sıfırlama)

Bu test durumunda, biz sadece reset_trigger olayını 10 simülasyon biriminden sonra tetikleyeceğiz.

```
1 initial
2   begin: TEST_CASE
3      #10  -> reset_trigger;
4   end
```

You could download file counter_tb5.v here

Test Durumu 2 - Assert/ De-assert enable after reset is applied.

Bu test durumunda, yeniden başlatma(reset) lojiğini tetikleriz ve biz enable sinyaline lojik 1 vermeye başlamadan önce reset lojik işleminin tamamlaması beklenir.

```
 1  initial
 2   begin: TEST_CASE
 3      #10  -> reset_trigger;
 4     @ (reset_done_trigger);
 5     @ (negedge clk);
 6     enable = 1;
 7     repeat (10) begin
 8       @ (negedge clk);
 9     end
10     enable = 0;
11   end
```

You could download file counter_tb6.v here

Test Durumu 3- enable ve reset'in rasgele etkinleştirilmesi/etkinleştirilmemesi.

Bu test durumunda reset 'i etkinleştiririz ve daha sonra enable ve reset sinyallerini rasgele sürebiliriz.

```
 1  initial
 2   begin : TEST_CASE
 3      #10  -> reset_trigger;
 4     @ (reset_done_trigger);
 5     fork
 6       repeat (10) begin
 7          @ (negedge clk);
 8         enable = $random;
 9       end
10       repeat (10) begin
11         @ (negedge clk);
12         reset = $random;
13       end
14     join
15   end
```

You could download file counter_tb7.v here

Şimdi siz bana bu üç test durumununda aynı dosyada olması mı gerekir diye sormalısınız? Cevap hayır. Eğer biz bu üç test durumunu da tek bir dosyaya koyarsak, üç initial bloğu reset ve enable sinyallerini sürdüğünden bir yarış koşuluna düşmüş oluruz. Bu nedenle normalde, bir test benç kodlama yapıldığında, test durumları ayrı ayrı kodlanır ve testbencin içine `include aşağıda gösterildiği gibi bir yönergeyle koyulur.(Bunu yapmak için daha iyi yollar var ancak bunu nasıl yapmak istediğiniz düşünmek zorundasınız.)

Eğer tüm üç test durumuna bakacak olursanız, test durumlarının yürütmesi tamamlanmadıysa simülasyon sonlanacaktır. Daha iyi bir kontrol mekanizması için, bir "terminate_sim"(simülasyonun sonlanması) benzeri bir olayı ekleyebiliriz ve bu olay tetiklendiğinde $finish yürütülür. Bu olayı test durumunun yürütülmesinin sonunda tetikleyebiliriz. $finish için kodlama aşağıda gösterilmiştir.

```
1   event terminate_sim;
2   initial begin
3   @ (terminate_sim);
4      #5  $finish;
5   end
```

You could download file counter_tb8.v here

Test durumu #2 'nin değiştirilmiş-düzenlenmiş hali aşağıdaki gibidir:

```
1 initial
2   begin: TEST_CASE
3      #10  -> reset_trigger;
4     @ (reset_done_trigger);
5     @ (negedge clk);
6     enable = 1;
7     repeat (10) begin
8       @ (negedge clk);
9     end
```

```
10      enable = 0;
11       #5  -> terminate_sim;
12    end
13
```

You could download file counter_tb9.v here

Yaklaşımdaki ikinci problem, şimdiye kadar dalgaformunu elle kontrol etmemiz gerekiyordu ve simülatörün ekran çıktısı eğer DUT doğru şekilde çalışıyorsa görülebiliryordu. Dördüncü bölüm bunu nasıl otomatik şekilde yaparız onu gösterecek.

Karşılaştırma Lojiğinin Eklenmesi

Bir testbenci kendi kendini kontrol eden/otomatikleşmiş yapmak için, ilk olarak DUT'ı fonksiyonalite olarak taklit edecek bir model geliştirmemiz lazım. Bizim örneğimizde, bu çok kolay olacak, fakat eğer DUT karmaşıksa, onu taklit etmek çok karmaşık olabilir ve birçok yenilikçi tekniğe ihtiyaç duyabiliriz bu işi kendikenine kontrol edecek hale getirmek için.

```
1 reg [3:0] count_compare;
2
3 always @ (posedge clk)
4 if (reset == 1'b1) begin
5   count_compare <= 0;
6 end else if ( enable == 1'b1) begin
7   count_compare <= count_compare + 1;
8 end
```

You could download file counter_tb10.v here

Eğer DUT fonksiyonalitesini taklit edecek bir lojiğe sahipsek, denetçi(checker) lojiği eklememiz gerekmektedir, bu verilen herhangi bir noktada beklenen değerle mevcut değeri denetiminde tutar. Eğer bir hata varsa, beklenen değeri ve mevcut değeri yazar, ve "terminate_sim" olayı tetiklenerek simülasyon sonlandırılır.

```
1 always @ (posedge clk)
2   if (count_compare  ! = count) begin
3     $display ("DUT Error at time %d", $time);
4     $display (" Expected value %d, Got Value %d", count_compare, count);
5      #5  -> terminate_sim;
6   end
```

You could download file counter_tb11.v here

Şimdi tüm lojik elimizde mevcut, böylece $display ve $monitor 'ü silebiliriz, böylece bizim testbencimiz tam otomatik hale gelmiştir, ve DUT giriş ve çıkışlarını elle doğrulamamıza gerek kalmamıştır. count_compare'i count_compare= count_compare +2 olarak değiştirmeye çalışın ve karşılaştırma lojiğinin nasıl çalıştışını görün. Bu bizim testbencimizin kararlı olup olmadığını anlamamızın başka bir yoludur.

Bizim test ortamımızı daha kolay anlaşılır bir hale getirmek için bazı süslü püslü yazdırma yapabiliriz aşağıdaki şekilde olduğu gibi.

```
C:\Download\work>veridos counter.v counter_tb.v
VeriWell for Win32 HDL  Sat Jan 18 20:10:35 2003

This is a free version of the VeriWell for Win32 Simulator
Distribute this freely; call 1-800-VERIWELL for ordering information
```

```
See the file "!readme.1st" for more information

Copyright (c) 1993-97 Wellspring Solutions, Inc.
All rights reserved


Memory Available: 0
Entering Phase I...
Compiling source file : counter.v
Compiling source file : counter_tb.v
The size of this model is [5%, 6%] of the capacity of the free version

Entering Phase II...
Entering Phase III...
No errors in compilation
Top-level modules:
counter_tb

###########################################
Applying reset
Came out of Reset
Terminating simulation
Simulation Result : PASSED
###########################################
Exiting VeriWell for Win32 at time 96
0 Errors, 0 Warnings, Memory Used: 0
Compile time = 0.0, Load time = 0.0, Simulation time = 0.0

Normal exit
Thank you for using VeriWell for Win32
```

Biliyorum ki, bu çıkışı elde etmek için kullandığın test benç kodunu görmek istiyorsunuz, test benç kodunu buradan ve sayaç kodunu da buradan bulabilirsiniz.

Bahsetmediğim birçok şey var; zaman bulduğumda bunları da detaylı şekilde bu konuya ekleyebilirim

Bellek ve FSM Modelleme

Bellek Modelleme

Belleği modellemeye yardımcı olmak için, Verilog iki boyutlu diziler için destek sağlar. Belleklerin davranışsal modelleri yazmaç değişkenlerinin dizisiyle bildirilerek modellenir; dizideki herhangi bir sözcüğe dizinin indisiyle-diziniyle eşilebilir. Dizideki ayrık bir bit'e ulaşmak için geçici bir değişken gereklidir.

Sözdizim

reg [kelimeboyutu:0] dizi_ismi [0:diziboyutu]
(reg [wordsize:0] array_name [0:arraysize])

Örnekler

Bildirim(Declaration)

reg [7:0] my_memory [0:255];

Burada [7:0] belleğin genişliği ve [0:255] de belleğin derinliğidir ve parametreleri şöyledir:

- Genişlik(Width) : 8 bit, en yüksek bit en soldakidir(little endian)
- Derinlik(Depth) : 256, 0ncı adres dizideki 0ncı bölgeyi belirtir.

Değerlerin Saklanması

Benim_bellegim[adresi]=atanacak_veri;
(my_memory[adress] = data_in;)

Değerlerin Okunması

cikan_veri_benim_bellegim[adresi];
(data_out = my_memory[address];)

Bit Okuma(Read)

Bazen sadece bşr bit'in okunmasına gerek duyulabilir. Ne yazık ki Verilog bir bitlik okumaya ya da yazmaya izin vermemektedir: bu problem için incelikle bulunan çözüm aşağıdaki belirtilmiştir.

cikan_veri_benim_bellegim[adresi];
(data_out = my_memory[address];)

cikan_verinin_0nci_biti=cikan_veri[0];
(data_out_it_0 = data_out[0];)

Belleği Parafe Etmek(Initializing Memories)

Bir bellek dizisi, diskteki bellek örüntü dosyasından okuyarak ve onu bellek dizisinde saklayarak parafe edilebilir(başlatılabilir). Bunu yapmak için sistem görevleri $readmemb ve $readmemh 'ı kullanacağız. $readmemb belleğin içeriğinin ikili gösterimi için kullanılır, $readmemh ise hex-16'lık gösterimi için kullanılır.

Sözdizim

$readmemh("dosya_ismi",bellek_dizisi, başlangıç_adresi,bitiş_adresi);
($readmemh("file_name",mem_array,start_addr,stop_addr);)

Not : başlangıç ve bitiş adresi isteğe bağlıdır(opsiyoneldir).

Örnek – Basit bellek

```
1 module  memory();
2 reg [7:0] my_memory [0:255];
3
4 initial begin
5  $readmemh("memory.list", my_memory);
6 end
7 endmodule
```

You could download file memory.v here

Örnek- Memory.list dosyası

```
1 //Açıklamalara(Comments) izin verilir
2 1100_1100   // Bu ilk adrestir :8'h00
3 1010_1010   // Bu ikinci adrestir :8'h01
4 @ 55        // Yeni bir adres atla(jump):8'h55
5 0101_1010   // Bu adres :8'h55
6 0110_1001   // Bu adres :8'h56
```

You could download file memory.list here

$readmemh sistem görevleri testbenç vektörlerini okumak için de kullanılabilir. Bunu zaman bulduğumda testbenç bölümünde daha ayrıntılı şekilde açıklayacağım.

Farklı tipteki bellek örnekleri için örnekler bölümüne bakınız.

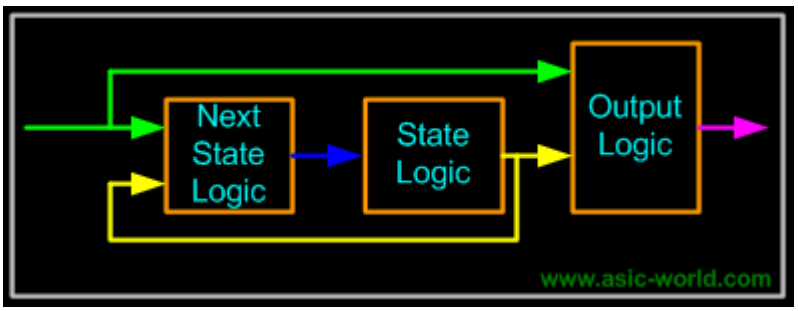
FSM'ye Giriş

Durum makineleri veya FSM(sonlu durum makineleri) dijital tasarımın kalbidir; elbette bir sayaç FSM'nin basit bir biçimidir. Ben Verilog'u öğrenirken, "FSM 'yi Verilogda nasıl kodladığıma" hayret etmiştim ve "bunu kodlamanın en iyi yolu neydi". Ben önce bu sorunun ilkini aşağıda cevaplayacağım ve ikinci bölümü en güzel bölümde bulabilirsiniz.

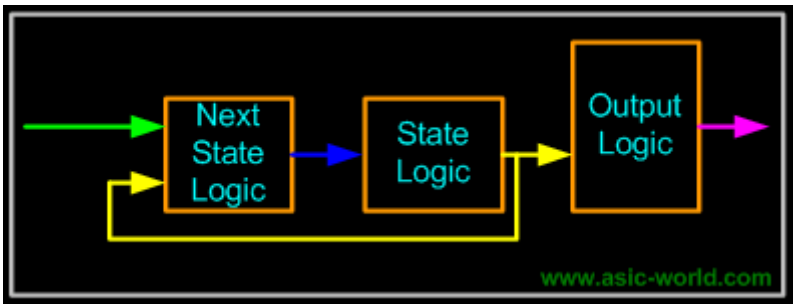
Durum Makine Tipleri

Herbirinden üretilen çıkışlara göre sınıflandırılmış iki tip durum makinesi bulunmaktadır. İlki Moore Durum Makinesidir, çıkışlar sadece şuanki durumun bir fonksiyonudur, ikincisi ise Mealy Durum Makinesidir, bunda ise çıkışlardan biri yada birkaçı şuanki duırumun be bir yada birçak girişin fonksiyonudur.

Mealy Modeli



Moore Modeli



Durum makineleri durumların şifrelenemesine göre de sınıflandırılabilir. Şifreleme(encoding) stili kritik bir faktördür bu FSM'nin hızına ve kapı karmaşıklığına karar verir. İkili(binary), gri(gray), sıcak olan(one hot), soğuk olan(one cold), ve neredeyse sıcak olan(almost one hot) FSM için kullanılan farklı tipteki şifreleme stilleridir.

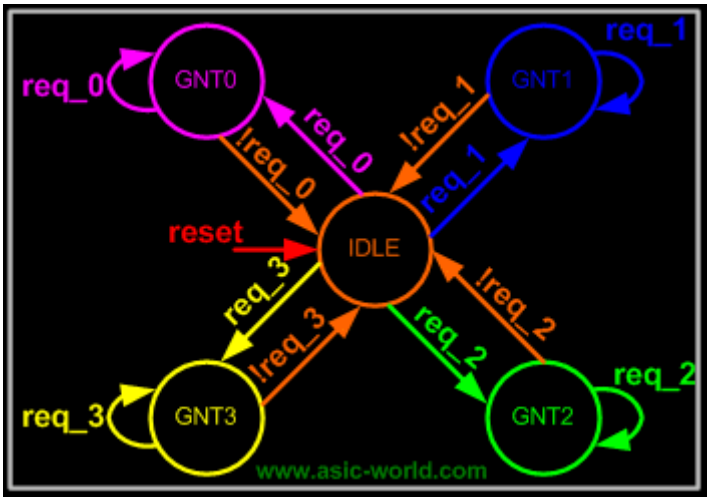
Durum makinelerinin modellenmesi

FSM'yi kodlarken aklınızda tutmanız gereken şeylerden biri kombinasyonel lojik ve ardışıl lojik iki farklı always(herzaman) bloğunda olabilir. Yukarıda belirttiğim iki şekilde, next state logic(bir sonraki durum lojiği) herzaman kombinasyonel lojiktir. State Registers(Durum Yazmaçları) ve Output logic(Çıkış lojiği) ardışıl lojiktir. Bu çok önemlidir ki next state logic için asenkron sinyaller FSM'yi beslemeden önce senkronlaştırılabilir. Herzaman FSM'yi ayrı bir Verilog dosyasında tutmaya çalışın.

Parametreler gibi sabit bildirimi kullanmak ya da 'define(tanımlamak) ile FSM'nin durumlarını tanımlamak kodu daha okunabilir ve daha kolay yönetilebilir yapmaktadır.

Örnek- Arabulucu(Arbiter)

Verilog'daki FSM kodlama şekillerini çalışmak için arbiter(arabulucu) kullanacağız.



Verilog Kodu

FSM ckodu üç bölüm içerir:

- Şifreleme(Encoding) şekli.
- Kombinasyonel(Combinational) bölüm.
- Ardışıl(Sequential) bölüm.

Şifreleme(Encoding) Şekli

Birçok şifreleme şekli bulunmaktadır, bunlardan birkaçı aşağıdadır:

- Binary(İkili) Şifreleme
- Bir Sıcak(One Hot) Şifreleme
- Bir Soğuk(One Cold) Şifreleme
- Neredeyse Biri Sıcak(Almost One Hot) Şifreleme
- Neredeyse Biri Soğuk(Almost One Cold) Şifreleme
- Gri(Gray) Şifreleme

Yukarıdaki tüm tiplerde biz normalde bir sıcak(one hot) ve ikili(binary) şifreleme kullanırız.

Bir Sıcak Şifreleme(One Hot Encoding)

```
1 parameter  [4:0]  IDLE  = 5'b0_0001;
2 parameter  [4:0]  GNT0  = 5'b0_0010;
3 parameter  [4:0]  GNT1  = 5'b0_0100;
4 parameter  [4:0]  GNT2  = 5'b0_1000;
5 parameter  [4:0]  GNT3  = 5'b1_0000;
```

You could download file fsm_one_hot_params.v here

İkili Şifreleme(Binary Encoding)

```
1 parameter  [2:0]  IDLE  = 3'b000;
2 parameter  [2:0]  GNT0  = 3'b001;
3 parameter  [2:0]  GNT1  = 3'b010;
4 parameter  [2:0]  GNT2  = 3'b011;
5 parameter  [2:0]  GNT3  = 3'b100;
```

You could download file fsm_binary_params.v here

Gri Şifreleme(Gray Encoding)

```
1 parameter  [2:0]  IDLE  = 3'b000;
2 parameter  [2:0]  GNT0  = 3'b001;
3 parameter  [2:0]  GNT1  = 3'b011;
4 parameter  [2:0]  GNT2  = 3'b010;
5 parameter  [2:0]  GNT3  = 3'b110;
```

You could download file fsm_gray_params.v here

Kombinasyonel Bölüm(Combinational Section)

Bu bölüm fonksiyonlarlai atama ifadeleriyle veya bir case ifadesi ile always(herzaman) bloğu kullanarak modellenebilir. Bu kez biz always blok versiyonunu göreceğiz.

```
 1 always @ (state or req_0 or req_1 or req_2 or req_3)
 2 begin
 3   next_state = 0;
 4   case(state)
 5     IDLE : if (req_0 == 1'b1) begin
 6          next_state = GNT0;
 7            end else if (req_1 == 1'b1) begin
 8          next_state= GNT1;
 9            end else if (req_2 == 1'b1) begin
10          next_state= GNT2;
11            end else if (req_3 == 1'b1) begin
12          next_state= GNT3;
13        end else begin
14          next_state = IDLE;
15            end
16     GNT0 : if (req_0 == 1'b0) begin
17          next_state = IDLE;
18            end else begin
19          next_state = GNT0;
20       end
21     GNT1 : if (req_1 == 1'b0) begin
22          next_state = IDLE;
23            end else begin
24          next_state = GNT1;
25       end
26     GNT2 : if (req_2 == 1'b0) begin
27          next_state = IDLE;
28            end else begin
29          next_state = GNT2;
30       end
31     GNT3 : if (req_3 == 1'b0) begin
32          next_state = IDLE;
33            end else begin
34          next_state = GNT3;
35       end
36    default : next_state = IDLE;
37   endcase
38 end
```

You could download file fsm_combo.v here

Ardışıl Bölüm(Sequential Section)

Bu bölümde modelleme sadece kenar hassasiyetli lojik olan posedege veya negedge saatli always bloğu kullanılarak yapılmak zorundadır.

```
always @ (posedge clock)
begin : OUTPUT_LOGIC
  if (reset == 1'b1) begin
    gnt_0 <=  #1  1'b0;
    gnt_1 <=  #1  1'b0;
    gnt_2 <=  #1  1'b0;
    gnt_3 <=  #1  1'b0;
    state <=  #1  IDLE;
  end else begin
    state <=  #1  next_state;
    case(state)
       IDLE : begin
                gnt_0 <=  #1  1'b0;
                gnt_1 <=  #1  1'b0;
                gnt_2 <=  #1  1'b0;
                gnt_3 <=  #1  1'b0;
           end
    GNT0 : begin
             gnt_0 <=  #1  1'b1;
           end
        GNT1 : begin
                 gnt_1 <=  #1  1'b1;
               end
        GNT2 : begin
                 gnt_2 <=  #1  1'b1;
               end
        GNT3 : begin
                 gnt_3 <=  #1  1'b1;
               end
     default : begin
                 state <=  #1  IDLE;
               end
    endcase
  end
end
```

You could download file fsm_seq.v here

İkili Şifrelemen Kullanmanın Tam Kodu

```
module fsm_full(
clock , // Saat(Clock)
reset , // Aktif yüksek sıfırlama(Active high reset)
req_0 , // 0ncı etkenden(agent)aktif yüksek istek(Active high request)
req_1 , // 1nci etkenden(agent)aktif yüksek istek(Active high request)
req_2 , // 2nci etkenden(agent)aktif yüksek istek(Active high request)
req_3 , // 3ncı etkenden(agent)aktif yüksek istek(Active high request)
gnt_0 , // 0ncı etkenden(agent)aktif yüksek onay(Active high grant)
gnt_1 , // 1nci etkenden(agent)aktif yüksek onay(Active high grant)
gnt_2 , // 2nci etkenden(agent)aktif yüksek onay(Active high grant)
gnt_3   // 3ncı etkenden(agent)aktif yüksek onay(Active high grant)
);
// Port bildirimi burada yapılmaktadır
input clock ; // Saat(Clock)
```

```
15 input reset ; // Active high reset
16 input req_0 ; // Active high request from agent 0
17 input req_1 ; // Active high request from agent 1
18 input req_2 ; // Active high request from agent 2
19 input req_3 ; // Active high request from agent 3
20 output gnt_0 ; // Active high grant to agent 0
21 output gnt_1 ; // Active high grant to agent 1
22 output gnt_2 ; // Active high grant to agent 2
23 output gnt_3 ; // Active high grant to agent
24
25 // İç Değişkenler (Internal Variables)
26 reg    gnt_0 ; // Active high grant to agent 0
27 reg    gnt_1 ; // Active high grant to agent 1
28 reg    gnt_2 ; // Active high grant to agent 2
29 reg    gnt_3 ; // Active high grant to agent 3
30
31 parameter  [2:0]  IDLE  = 3'b000;
32 parameter  [2:0]  GNT0  = 3'b001;
33 parameter  [2:0]  GNT1  = 3'b010;
34 parameter  [2:0]  GNT2  = 3'b011;
35 parameter  [2:0]  GNT3  = 3'b100;
36
37 reg [2:0] state, next_state;
38
39 always @ (state or req_0 or req_1 or req_2 or req_3)
40 begin
41   next_state = 0;
42   case(state)
43     IDLE : if (req_0 == 1'b1) begin
44         next_state = GNT0;
45            end else if (req_1 == 1'b1) begin
46         next_state= GNT1;
47            end else if (req_2 == 1'b1) begin
48         next_state= GNT2;
49            end else if (req_3 == 1'b1) begin
50         next_state= GNT3;
51       end else begin
52         next_state = IDLE;
53            end
54     GNT0 : if (req_0 == 1'b0) begin
55         next_state = IDLE;
56            end else begin
57         next_state = GNT0;
58      end
59     GNT1 : if (req_1 == 1'b0) begin
60         next_state = IDLE;
61            end else begin
62         next_state = GNT1;
63      end
64     GNT2 : if (req_2 == 1'b0) begin
65         next_state = IDLE;
66            end else begin
67         next_state = GNT2;
68      end
69     GNT3 : if (req_3 == 1'b0) begin
70         next_state = IDLE;
71            end else begin
72         next_state = GNT3;
73      end
74    default : next_state = IDLE;
75   endcase
```

```
 76 end
 77
 78 always @ (posedge clock)
 79 begin : OUTPUT_LOGIC
 80   if (reset) begin
 81     gnt_0 <=  #1  1'b0;
 82     gnt_1 <=  #1  1'b0;
 83     gnt_2 <=  #1  1'b0;
 84     gnt_3 <=  #1  1'b0;
 85     state <=  #1  IDLE;
 86   end else begin
 87     state <=  #1  next_state;
 88     case(state)
 89    IDLE : begin
 90                 gnt_0 <=  #1  1'b0;
 91                 gnt_1 <=  #1  1'b0;
 92                 gnt_2 <=  #1  1'b0;
 93                 gnt_3 <=  #1  1'b0;
 94           end
 95    GNT0 : begin
 96             gnt_0 <=  #1  1'b1;
 97           end
 98         GNT1 : begin
 99                  gnt_1 <=  #1  1'b1;
100                end
101         GNT2 : begin
102                  gnt_2 <=  #1  1'b1;
103                end
104         GNT3 : begin
105                  gnt_3 <=  #1  1'b1;
106                end
107      default : begin
108                  state <=  #1  IDLE;
109                end
110     endcase
111   end
112 end
113
114 endmodule
```

You could download file fsm_full.v here

Testbenç

```
 1 `include "fsm_full.v"
 2
 3 module fsm_full_tb();
 4 reg clock , reset ;
 5 reg req_0 , req_1 ,  req_2 , req_3;
 6 wire gnt_0 , gnt_1 , gnt_2 , gnt_3 ;
 7
 8 initial begin
 9   $display("Time\t    R0 R1 R2 R3 G0 G1 G2 G3");
10   $monitor("%g\t    %b  %b  %b  %b  %b  %b  %b  %b",
11     $time, req_0, req_1, req_2, req_3, gnt_0, gnt_1, gnt_2, gnt_3);
12   clock = 0;
13   reset = 0;
14   req_0 = 0;
15   req_1 = 0;
16   req_2 = 0;
17   req_3 = 0;
```

```
18    #10  reset = 1;
19    #10  reset = 0;
20    #10  req_0 = 1;
21    #20  req_0 = 0;
22    #10  req_1 = 1;
23    #20  req_1 = 0;
24    #10  req_2 = 1;
25    #20  req_2 = 0;
26    #10  req_3 = 1;
27    #20  req_3 = 0;
28    #10  $finish;
29 end
30
31 always
32   #2  clock = ~clock;
33
34
35 fsm_full U_fsm_full(
36 clock , // Clock
37 reset , // Active high reset
38 req_0 , // Active high request from agent 0
39 req_1 , // Active high request from agent 1
40 req_2 , // Active high request from agent 2
41 req_3 , // Active high request from agent 3
42 gnt_0 , // Active high grant to agent 0
43 gnt_1 , // Active high grant to agent 1
44 gnt_2 , // Active high grant to agent 2
45 gnt_3   // Active high grant to agent 3
46 );
47
48
49
50 endmodule
```

You could download file fsm_full_tb.v here

Simülatör Çıktısı

```
Time        R0 R1 R2 R3 G0 G1 G2 G3
0           0  0  0  0  x  x  x  x
7           0  0  0  0  0  0  0  0
30          1  0  0  0  0  0  0  0
35          1  0  0  0  1  0  0  0
50          0  0  0  0  1  0  0  0
55          0  0  0  0  0  0  0  0
60          0  1  0  0  0  0  0  0
67          0  1  0  0  0  1  0  0
80          0  0  0  0  0  1  0  0
87          0  0  0  0  0  0  0  0
90          0  0  1  0  0  0  0  0
95          0  0  1  0  0  0  1  0
110         0  0  0  0  0  0  1  0
115         0  0  0  0  0  0  0  0
120         0  0  0  1  0  0  0  0
127         0  0  0  1  0  0  0  1
140         0  0  0  0  0  0  0  1
147         0  0  0  0  0  0  0  0
```

Parametreleştirilmiş Modüller(Parameterized Modules)

Giriş

Varsayalım ki bizim farklı derinliklerde fakat aynı fonksiyonaliteye sahip sayaçlara ihtiyacımız olan bir tasarımımız var. Başka bir şekilde birçok farklı derinlikteki ve RAM'lerdeki gibi aynı fonksiyonaliteye sahip farklı genişlikteki örneklemlere ihtiyacı olan bir tasarımımız olduğunu varsayalım. Normalde bunu farklı genişlikteki sayaçlar yaratarak ve daha sonrada bunları kullanarak sağlarız. Aynı kuralı daha önce bahsettiğimiz RAM içinde kullanabiliriz.

Fakat Verilog bu problemin üstesinden gelebilmek için güçlü bir yöntem sağlar: parametre adında bir şey destekler; bu parametreler belirli modüllerdeki yerel değişkenlere benzer.

Varsayılan değerleri, defparam kullanarak veya örneklem esnasında yeni bir parametre kümesi ile geçersiz kılabiliriz. Buna parametrelerin üstüne yazma(parameter overriding) denir.

Parametreler (Parameters)

Bir parametre Verilog tarafından modül yapısı içerisinde sabit bir değer olarak bildirilerek tanımlanmıştır. Değer, modülün davranışını fiziksel gösteriminde karakterize eden bir dizi özellik kümesi olarak kullanılmıştır.

- Bir modülün içinde tanımlanmıştır.
- Yerel kapsamlıdır(Local scope).
- Örneklem(instantiation) zamanında üstüne yazılabilir(overridde).
- -> Eğer çoklu parametreler tanımlandıysa, bunların tanımlandığı sırayla üstüne yazılmalıdır. Eğer bir üstüne yazma değeri belirtilmemişse varsayılan parametre bildirimi değerleri kullanılır.
- Defparam ifadesi ile değiştirilmiş olabilir.

Defparam Kullanarak Parametrenin Üstüne Yazmak(Parameter Override)

```
 1 module secret_number;
 2 parameter my_secret = 0;
 3
 4 initial begin
 5   $display("My secret number is %d", my_secret);
 6 end
 7
 8 endmodule
 9
10 module defparam_example();
11
12 defparam U0.my_secret = 11;
13 defparam U1.my_secret = 22;
14
15 secret_number U0();
16 secret_number U1();
17
18 endmodule
```

You could download file defparam_example.v here

Örneklem(İnstantiating) Sırasında Parametrenin Üstüne Yazmak (Parameter Override)

```
 1 module secret_number;
 2 parameter my_secret = 0;
 3
 4 initial begin
 5   $display("My secret number in module is %d", my_secret);
 6 end
 7
 8 endmodule
 9
10 module param_overide_instance_example();
11
12 secret_number #(11) U0();
13 secret_number #(22) U1();
14
15 endmodule
```

You could download file param_overide_instance_example.v here

Bir değerden fazla parametre gönderme

```
 1 module  ram_sp_sr_sw (
 2 clk          , // Saat Girişi(Clock Input)
 3 address      , // Adres Girişi(Address Input)
 4 data         , // İki yönlü veri(Data bi-directional)
 5 cs           , // Chip Seçimi(Chip Select)
 6 we           , // Yazma Etkinleştirme/Okuma Etkinleştirme(Write
Enable/Read Enable)
 7 oe             // Çıkış Etkinleştirme(Output Enable)
 8 );
 9
10 parameter DATA_WIDTH = 8 ;
11 parameter ADDR_WIDTH = 8 ;
12 parameter RAM_DEPTH = 1 << ADDR_WIDTH;
13 // Actual code of RAM here
14
15 endmodule
```

You could download file param_more_then_one.v here

Bir parametreden fazla parametre örneklenirken, parametre değerleri alt mdülde bildirildikleri sırayla gönderilmelidir.

```
1 module  ram_controller ();//Bazı portlar
2
3 // Denetleyici Kodu (Controller Code)
4
5 ram_sp_sr_sw #(16,8,256)  ram(clk,address,data,cs,we,oe);
6
7 endmodule
```

You could download file param_more_then_one1.v here

Verilog 2001

Verilog 2001'de şimdiye kadar verdiğim kodlar çalışacaktır ancak yeni özellikler kodu daha okunabilir hale getirir ve hataları ortadan kaldırır.

```
1 module  ram_controller ();//Bazı portlar
2
3 ram_sp_sr_sw #(
4      .DATA_WIDTH(16),
5      .ADDR_WIDTH(8),
6      .RAM_DEPTH(256))  ram(clk,address,data,cs,we,oe);
7
8 endmodule
```

You could download file param_more_then_one2.v here

Sizce bu kod VHDL 'den mi kopyalandı?

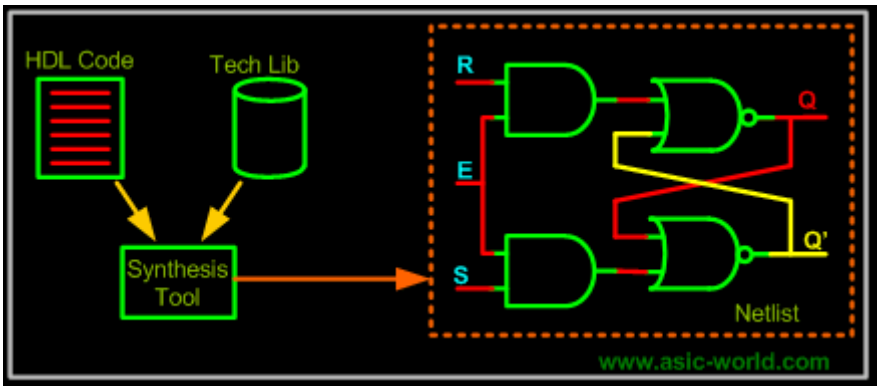
Verilog Sentez Eğitseli(Tutorial)

Lojik sentez nedir?

Lojik sentez, yüksek seviyeli tanımlama tasarımını optimize edilmiş kapı-seviyesi gösterime dönüştürme işlemidir. Lojik sentez, standart hücre kütüphanesini(cell library) kullanır, standart hücre kütüphanesi basit hücrelere ki bunlara örnek basit lojik kapılar ve,veya,veya değil; veya makro hücrelere ki bunlar toplayıcı(adder),çoklayıcı(mux), bellek ve flipfloplara sahiptir. Standart hücrelerin bir araya getirilmiş haline teknoloji kütüphanesi(technology library) denir. Normalde teknoloji kütüphanesi transistör büyüklüğü(0.18u, 90nm) ile bilinir.

Bir devre tanımlaması Verilog gibi Donanım Tanımlama Dili(Hardware Description Language (HDL)) tarafından yazılır. Daha sonra bu zamanlama, alan, test edilebilirliği ve gücü gibi tasarım kısıtlarını içerir.

Verilog eğitselimizin son bölümünde büyük bir örneğin tipik tasarım akışını göreceğiz.



HDL(Lojik Sentez) öncesi hayat

Daha önce okuldada öğrendiğiniz gibi, herşey(tüm sayısal devreler) elle tasarlanırdı. Karnough diyagramları çizilir, lojik optimize edilirdi, şematik çizilirdi. Bu eski günlerde mühendislerin dijiyal lojik devrelerinin nasıl tasarlanma şekliydi. Bu tasarım birkaç yüz kapı içerdiği sürece sorun değildi.

HDL'in etkisi ve Lojik sentez.

Yüksek-seviyeli tasarımda insan kaynaklı hatalara meyil daha azdır çünkü daha yüksek seviyede soyutlamayla tanımlanmıştır. Yüksek seviyeli tasarım tasarım kısıtlarına önemli bir ilgi olmadan yapılır. Yüksek seviyeli tasarımdan kapı seviyesine geçiş sentez araçlarıyla olur, tasarımı bütün halinde optimize edebilmek için çeşitli algoritmalar kullanılır. Bu çeşitli tasarımcıların tasarımdaki farklı blok stillerinin oluşmasını ve idealin altında(suboptimal) tasarımlardan kaynaklanan problemleri ortadan kaldırır. Lojik sentez araçları teknoloji bağımsız tasarıma izin verir. Tasarımın yeniden kullanılması teknoloji bağımsız tanımlamalarla mümkündür.

Burada neyi tartışıyoruz ?

Verilog'a geldiğinde, sentez akışı diğer dillerdekiyle aynıydı. Aşağıdaki birkaç sayfada bizim odaklandığımız kod parçalarının nasıl kapılara dönüştürüldüğüdür. Daha önceki bölümleri okurken merak ettiğiniz, bu Donanımda nasıl gösterilir? Bir örnek gecikmeler"delays" olabilir. Gecikmeleri sentezlememizin herhangi bir yolu yoktur, fakat elbette belirli sinyallere arabellek ekleyerek gecikmeleri ekleyebiliriz. Fakat bu durum sentez hedef teknolojiye çok bağımlı olmaktadır.(VLSI bölümünde daha derinlemesine inceleyeceğiz)

İlk olarak sentez araçları tarafından desteklenmeyen yapıları inceleyeceğiz; aşağıdaki tablo sentez araçları tarafından desteklenmeyen yapıları göstermektedir.

Sentezde Desteklenemeyen Yapılar

Yapının Tipi	Notlar
initial(başlangıç)	Sadece test benç de kullanılır.
events (olaylar)	Olaylar test benç bileşenlerinin senkronizasyonunu daha hassas yapar.
real(reel)	Reel veri tipi desteklenmez.
time (zaman)	Zaman(Time) veri tipi desteklenmez.
force(zorlama) ve release (bırakma)	Force ve release veri tipi desteklenmez.
assign ve deassign	assign ve deassign reg veri tipi desteklenmez. Fakat wire veri tipindeki assign(atama) desteklenir.
fork join	Aynı etkiyi elde etmek için bloklamayan(nonblocking) atama kullanın.
primitives (temeller)	Sadece kapı seviyesi temeller desteklenir.
table (tablo)	UDP ve tablolar desteklenme.

Sentezlenemeyen Verilog yapılarına örnek

Yukarıda belirtilen yapıları içeren herhangi bir kod sentezlenemez, fakat senezlenebilir yapılar içerisindeki kötü kodlama da sentezlenememeye neden olabilir. Mühendislerin bir hassasiyet listesinde flip-flop'u saatin hem posedge(pozitif kenar) hem de negedge(negatif kenar)'de kodladığını gördüm.

Diğer genel tipteki bir kod, bir reg(yazmaç) değişkeninin birden çok always(herzaman) bloğu tarafından sürülebildiğidir. Ancak şuna emin olabilirsiniz ki bu kesinlikle sentez hatasına neden olmaktadır.

Örnek- Initial(Başlangıç) İfadesi

```
 1 module synthesis_initial(
 2 clk,q,d);
 3 input clk,d;
 4 output q;
 5 reg q;
 6
 7 initial begin
 8  q <= 0;
 9 end
10
11 always @ (posedge clk)
12 begin
13  q <= d;
14 end
15
16 endmodule
```

You could download file synthesis_initial.v here

Gecikmeler(Delays)

a = #10 b; Bu kod sadece simülasyon amacıyla faydalıdır.

Sentez aracı normalde bu tip yapıları reddeder, ve sadece yukarıdaki ifadede #10 olmadığını varsayarak koda a = b; gibi davranır.

X ve Z 'nin herzaman reddedilmesinin karşılaştırılması

```
 1 module synthesis_compare_xz (a,b);
 2 output a;
 3 input b;
 4 reg a;
 5
 6 always @ (b)
 7 begin
 8   if ((b == 1'bz) || (b == 1'bx)) begin
 9     a = 1;
10   end else begin
11     a = 0;
12   end
13 end
14
15 endmodule
```

You could download file synthesis_compare_xz.v here

Donanıma yabancı olan tüm tasarım mühendislerinin karşılaştığı problemlerden biridir. Onlar normal olarak X ve Z değişkenlerini karşılaştırma eğilimindedirler. Pratikte bu yapılacak en kötü şeydir, bu yüzden lütfen X ve Z yi karşılaştırmaktan kaçının. Tasarımınızı iki durumla, 0 ve 1 ile sınırlandırın. Sadece chip'in IO pad seviyesinde üç-durumlu(tri-state) kullanın. Bunu daha sonraki birkaç sayfada örnek olarak göreceğiz.

Sentez Tarafından Desteklenen Yapılar

Verilog basit bir dildir; siz kolay anlaşılan ve eşleştirmesi kolay kapı kodları yazabilirsiniz. If ve case ifadelerinin kullanıldığı kodlar basittir ancak sentez araçlarında küçük başağrılarına neden olur. Ancak eğer süslü kodlamayı seviyorsanız ve zorlukları seviyorsanız, tamam korkmayın, Verilogda biraz deneyim kazandıktan sonra bunları kullanabilirsiniz. Yüksek seviyeli yapıları kullanmak büyük zevktir ve zaman kazandırır.

Herhangi bir lojiği tanımlamanın en genel yolu atama(assign) ifadeleri ya da herzaman(always) bloğudur. Bir atama(assign) ifadesi kombinasyonel lojiği modellemek için kullanılırken; always (herzaman) bloğu hem kombinasyonel hemde ardışıl(sequential) bloğu modellemek için kullanılabilir.

Yapı Tipi	Anahtar Sözcük veya Açıklama	Not
ports (portlar)	input, inout, output	inout 'u sadece IO seviyesinde kullanın.
Parameters (parametreler)	parameter	Bu tasarımı daha genel yapar
module definition (modül tanmlama)	module	
signals(sinyaller) ve variables(değişkenler)	wire, reg, tri	Vektörlere izin verilir
instantiation (örnekleme)	Modül örneklemeleri/ temel kapı örneklemeleri	ÖR.- (vedeğil)nand (out,a,b), RTL'i bu şekilde kodlama kötü bir fikir.
Function(fonksiyon) ve tasks (görevler)	function , task	Zamanlama yapıları reddedilir
Procedural(prosedürel)	always, if, else, case, casex, casez	initial(başlangıç) desteklenmemiştir
procedural blocks (prosedürel bloklar)	begin, end, named blocks, disable	İsimlendirilmiş blokların etkisizleştirilmesine izin verilir
data flow(veri akışı)	assign (atama)	Gecikme bilgisi ihmal edilir
named Blocks (isimlendirilmiş bloklar)	disable (etkisizleştirmek)	İsimlendirilmiş blokların etkisizleştirilmesi desteklenmiştir.
loops (döngüler)	for, while, forever	While ve forever döngüleri @(posedge clk) veya @(negedge clk) içermelidir

Operatörler ve bunların Etkileri

En sık karşılaşılan problemlerden biri mantıksal ve indirgeme-azaltma operatörlerinin bocalamasından kaynaklanmaktadır. Bu nedenle dikkatli olun.

Operatör Tipi	Operatör Sembolü	Gerçekleştirilen İşlem
Aritmetik	*	Çarpma (Multiply)
	/	Bölme(Division)
	+	Toplama(Add)
	-	Çıkarma(Subtract)
	%	Modu(Modulus)
	+	Birli artı(Unary plus)
	-	Birli eksi(Unary minus)
Lojik	!	Lojik tersi(Logical negation)
	&&	Lojik ve(Logical and)
	||	Lojik veya(Logical or)
İlişkisel	>	Büyüktür(Greater than)
	<	Küçüktür(Less than)
	>=	Büyük eşittir(Greater than or equal)
	<=	Küçük eşittir(Less than or equal)
Eşitlik	==	Eşitlik(Equality)
	!=	Eşitsizlik(inequality)
İndirgeme(Reduction)	&	Bit bit olumsuzlama(Bitwise negation)
	~&	Vedeğil(nand)
	|	Veya(or)
	~|	Veyadeğil(nor)
	^	Dışlamalı yada(xor)
	^~ ~^	Dışlamalı yada değil(xnor)
Kaydırma(Shift)	>>	Sağa kaydırma(Right shift)
	<<	Sola kaydırma(Left shift)
Birbirine bağlama(Concatenation)	{ }	Concatenation
Koşul(Conditional)	?	Koşul(conditional)

Lojik Devre Modelleme(Logic Circuit Modeling)

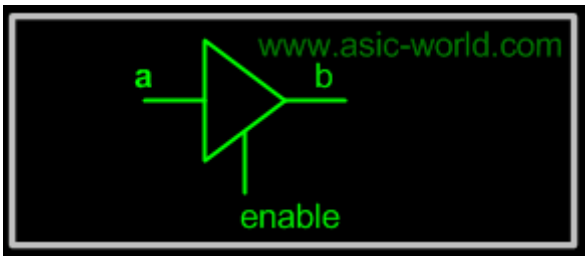
Daha önce sayısal-dijital tasarımda öğrendiğimiz gibi iki tip sayısal devre olabilir. Birincisi kombinasyonel devreler(combinational circuits) ve ikincisi de ardışıl devreler(sequential circuits). İyi bir sentez çıkışı almak ve sürprizlerden kaçınmak için takip edilmesi gereken çok az kural vardır.

Atama Kullanarak Kombinasyonel Devre Modelleme

Verilogda kombinasyonel devre modelleme atamayla yada always(herzaman) bloğuyla yapılır. Verilogda basit bir kombinasyonel devre yazma çok açıktır, aşağıdaki örnekteki gibi

assign y = (a&b) | (c^d);

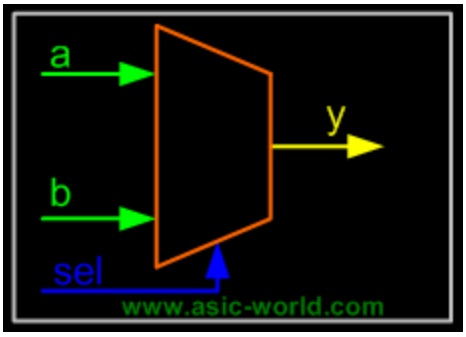
Üç-durumlu arabellek(Tri-state buffer)




```
1 module tri_buf (a,b,enable);
2  input a;
3  output b;
4  input enable;
5  wire b;
6
7 assign b = (enable) ? a : 1'bz;
8
9 endmodule
```

You could download file tri_buf.v here

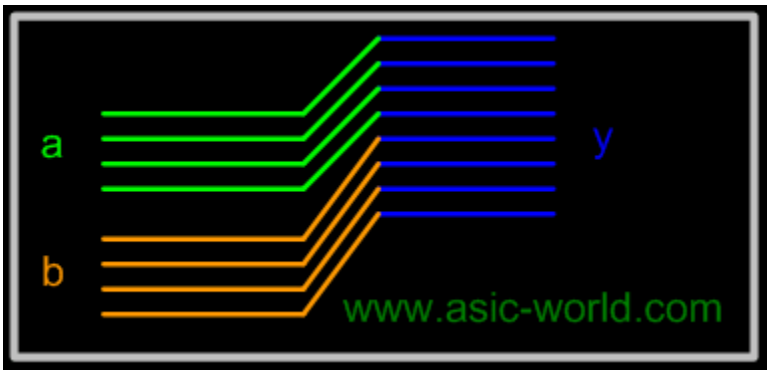
Çoklayıcı(Mux –Multiplexer)




```
1 module mux_21 (a,b,sel,y);
2              input a, b;
3              output y;
4              input sel;
5              wire y;
6
7              assign y = (sel) ? b : a;
8
9 endmodule
```

You could download file mux_21.v here

Basit Bir Birbirine Bağlama(Simple Concatenation)




```
1 module bus_con (a,b);
2              input [3:0] a, b;
3              output [7:0] y;
4              wire [7:0] y;
5
6              assign y = {a,b};
7
8 endmodule
```

You could download file bus_con.v here

Elde Bit'li 1 Bitlik Toplayıcı(1 Bit Adder With Carry)

```
 1 module addbit (
 2 a      , // ilk giriş(first input)
 3 b      , // İkinci giriş(Second input)
 4 ci     , // Elde girişi (Carry input)
 5 sum    , // toplam çıkışı(sum output)
 6 co       // elde çıkışı(carry output)
 7 );
 8 //Giriş bildirimi
 9 input a;
10 input b;
11 input ci;
12 //Çıkış bildirimi
13 output sum;
14 output co;
15 //Port Veri Tipleri
16 wire  a;
17 wire  b;
18 wire  ci;
19 wire  sum;
20 wire  co;
21 //Kod buradan başlar
22 assign {co,sum} = a + b + ci;
23
24 endmodule // addbit modülünün sonu
```

You could download file addbit.v here

2 İle Çarpma (Multiply by 2)

```
1 module muliply (a,product);
2             input [3:0] a;
3             output [4:0] product;
4             wire [4:0] product;
5
6             assign product  = a << 1;
7
8 endmodule
```

You could download file multiply.v here

3 'den 8'e Kod Çözücü (3 is to 8 decoder)

```
 1 module decoder (in,out);
 2 input [2:0] in;
 3 output [7:0] out;
 4 wire [7:0] out;
 5 assign out  =      (in == 3'b000 ) ? 8'b0000_0001 :
 6 (in == 3'b001 ) ? 8'b0000_0010 :
 7 (in == 3'b010 ) ? 8'b0000_0100 :
 8 (in == 3'b011 ) ? 8'b0000_1000 :
 9 (in == 3'b100 ) ? 8'b0001_0000 :
10 (in == 3'b101 ) ? 8'b0010_0000 :
11 (in == 3'b110 ) ? 8'b0100_0000 :
12 (in == 3'b111 ) ? 8'b1000_0000 : 8'h00;
13
14 endmodule
```

You could download file decoder.v here

Always(Herzaman) Kullanarak Kombinasyonel Devre Modelleme

Always ifadesi kullanarak modelleme yaparken, gereken önem verilmediyse sentezden sonra bir tutucu(latch) alma şansı vardır. (Kimse tasarımında tutucuları seviyor gözükmez, ancak bunlar daha hızlıdır, ve daha az transistör kullanır. Bu nedenle zamanlama analiz araçları her zaman tutucularla sorun yaşarlar; etkinleştirme(enable) pinindeki geçici performans düşmesi (glitch)ise bir başka problemdir).

Aşağıdaki kodda da görüldüğü gibi always ifadesindeki tutucuları önlemenin basit bir yolu always kodunun başındaki LHS değişkenine herzaman 0 sürmektir.

always kullanarak 3'e 8'lik kod çözücü(3 is to 8 decoder using always)

```
 1 module decoder_always (in,out);
 2 input [2:0] in;
 3 output [7:0] out;
 4 reg [7:0] out;
 5
 6 always @ (in)
 7 begin
 8   out = 0;
 9   case (in)
10     3'b001 : out = 8'b0000_0001;
11     3'b010 : out = 8'b0000_0010;
12     3'b011 : out = 8'b0000_0100;
13     3'b100 : out = 8'b0000_1000;
14     3'b101 : out = 8'b0001_0000;
```

```
15     3'b110 : out = 8'b0100_0000;
16     3'b111 : out = 8'b1000_0000;
17   endcase
18 end
19
20 endmodule
```

You could download file decoder_always.v here

Ardışıl Devre Modelleme (Sequential Circuit Modeling)

Ardışıl lojik devre modelleme herzaman(always) bloğundaki hassasiyet listesindeki kenar hassasiyetli elemanlar kullanarak yapılır. Ardışıl lojik sadece always bloğu kullanılarak modellenir. Normalde biz ardışıl devreler için bloklamayan(nonblocking) atamalar kullanırız.

Basit Flip-Flop

```
 1 module flif_flop (clk,reset, q, d);
 2 input clk, reset, d;
 3 output q;
 4 reg q;
 5
 6 always @ (posedge clk )
 7 begin
 8   if (reset == 1) begin
 9     q <= 0;
10   end else begin
11     q <= d;
12   end
13 end
14
15 endmodule
```

You could download file flip_flop.v here

Verilog Kodlama Stili

Şimdiye kadar gösterdiğim kodlara bakacak olursanız, size bir kodlama stili empoze etmeye çalıştığımı göreceksiniz. Tüm firmaların kendilerine ait kodlama ilkeleri ve bu kodlama ilkelerini kontrol eden linters benzeri araçları vardır. Aşağıda küçük bir ilkeler listesi vardır.

- Sinyal ve değişkenler için anlamlı isimler kullanın
- Aynı always bloğunda seviye ve kenar hassasiyetli elemanları karıştırmayın
- Pozitif ve negatif kenar tetiklemeli flip-flopları karıştırmaktan kaçının
- Lojik yapıları optimize etmek için parantez kullanın
- Basit çoklu lojik(combo logic) için sürekli atama ifadeleri kullanın
- Ardışıl için tıkamayan(nonblocking) ve çoklu lojik için tıkayan-bloklayan kullanın
- Aynı always bloğunda tıkayan(blocking) ve tıkamayan(nonblocking) atamaları karıştırmayın
- Aynı değişkende çoklu atama olup olmadığına dikkat edin
- if-else veya case ifadelerini açıkça tanımlayın

Verilog PLI(Programming Language Interface-Programlama Dili Arayüzü) Eğitseli

Giriş

Verilog PLI(Programming Language Interface-Programlama Dili Arayüzü) C veya C++ fonksiyonlarını Verilog kodundan çağıran bir mekanizmadır.

Verilogda fonksiyon çağırma sistem çağrısı(system call) olarak adlandırılır. Yerleşik bir sistem çağrısı örneği $display, $stop, $random 'dur. PLI kullanıcıya kendi sistem çağrılarını oluşturmaya izin verir, yani Verilog sözdiziminin bizim yapmamıza izin vermediklerini. Bunlardan bazıları:

- Güç analizi(Power analysis).
- Kod kaplama araçları(Code coverage tools).
- Verilog simülasyon veri yapısını daha etkin gecikmelerle değiştirmek (modifying the Verilog simulation data structure - more accurate delays).
- Özel Çıkış görüntüleri(Custom output displays).
- Birlikte-simülasyasyon (Co-simulation).
- Tasarım hata ayıklama araçları(Design debug utilities).
- Simülasyon analizi(Simulation analysis).
- Simülasyonu hızlandırmak için C-modeli arayüz (C-model interface to accelerate simulation).
- Testbenç modelleme(Testbench modeling).

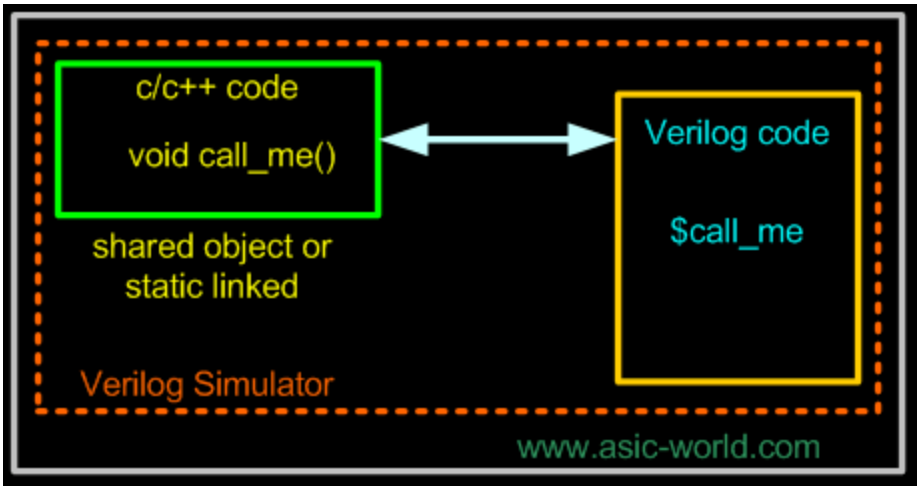
PLI'ın bu uygulamalarını gerçekleştirebilmek için, C kodunun Verilog simülatörünün iç veri yapısına erişebilmelidir. Bunu sağlamak için Verilog PLI acc rutinleri veya basiyçe erişim rutinleri adı verilen şeyi desteklemelidir.

İkinci bir rutin kümesine de tf veya basitçe basit görev(task) ve fonksiyon rutinleri adı verilmiştir. tf ve acc PLI 1.0 rutinleridir ve çok büyük ve eskidir. Son rutin kümesi ise en son sürüm Verilog 2001 ile sunulmaktadır ve buna vpi rutinleri denir. Bunlar küçüktür ve kristal berraklığında PLI rutinleridir ve bu nedenle bu yeni versiyon PLI 2.0'dır.

Verilog 2001 LRM ve PLI 1.0 IEEE dokümanlarında herbir ve tüm desteklenen fonksiyonlar hakıında bilgi bulabilirsiniz. Verilog IEEE LRM'leri donanım altyapısı olan herkesin anlayabileceği şekilde yazılmıştır. Eğer IEEE dokümanlarını elde edemiyorsanız kitaplar bölümünde listelenen PLI kitaplarını satın alabilirsiniz.

Nasıl Çalışır

- Fonksiyonları C/C++ kodu şeklinde yazın.
- Onları paylaşılmış kütüphaneleri üretmek (Windowsda *.DLL ve UNIXde *.so) için derleyin. VCS benzeri simülatörler statik bağlamaya izin vermektedir.
- Bu fonksiyonları Verilog kodunda kullanın(Genellkile Verilog Testbencinde).
- Temelde simülatör, Verilog Kodunun derlenmesi sürecinde C/C++ fonksiyonlarının detayları simülatöre gönderilir.(Buna bağlama(linking) denir ve bunun nasıl yapıldığını anlamak için kullanıcı klavuzuna bakmanız gerekir).
- Bir kere bağladıktan sınra simülatörü Verilog simülasyonundaki gibi çalıştırın.



Verilog kodu simülatör tarafından gerçekleştirilirken, simülatör kullanıcı tanımlı sistem görevleriyle karşılaştığında($ ile başlayanlar), yürütme kontrolü PLI rutinine (C/C++) fonksiyonuna geçer.

Örnek – Merhaba Dünya (Hello World)

Bir hello fonksiyonu tanımlayacağız, çağırıldığında ekrana "Hello Deepak" yazdıracak. Bu örnek hiçbir standart PLI fonksiyonlarını(ACC, TF ve VPI) kullanmamaktadır. Tam bağlama detayları için lütfen simülatör kılavuzuna bakınız. Herbir simülatör C/C++ fonksiyonlarından simülatöre bağlamayı kendi yollarıyla gerçekleştirir.

C Kodu

```
1 #include <stdio.h>
2
3 void hello () {
4   printf ("\nHello Deepak\n");
5 }
```

You could download file hello.c here

Verilog Kodu

```
1 module hello_pli ();
2
3 initial begin
4   $hello;
5    #10  $finish;
6 end
7
8 endmodule
```

You could download file hello_pli.v here

Simülasyonun Çalıştırılması

Bağlama tamamlandıkta sonra, simülasyon normal bir simülasyon gibi çalıştırılır, komut satırı seçeneklerindeki ufak değişiklikler ile: sadece simülatöre PLI kullandığımızı söylememiz gerekir.(Modelsim komut satırına hangi paylaşımlı objelerin yükleneceğini bilme ihtiyacı duyar).

PLI Uygulaması Yazma

Daha önce gösterdiğim örnek çok basit ve pratik amaç için iyi değil. Bizim kötü şöhretli sayaç örneğimizi göz önüne alalım ve DUT referans modelini ve C Kontrolörünü yazalım ve Verilog Testbencine bağlayalım. İlk olarak PLI kullanarak bir C modeli yazmak için gereksinimleri listeleyelim.

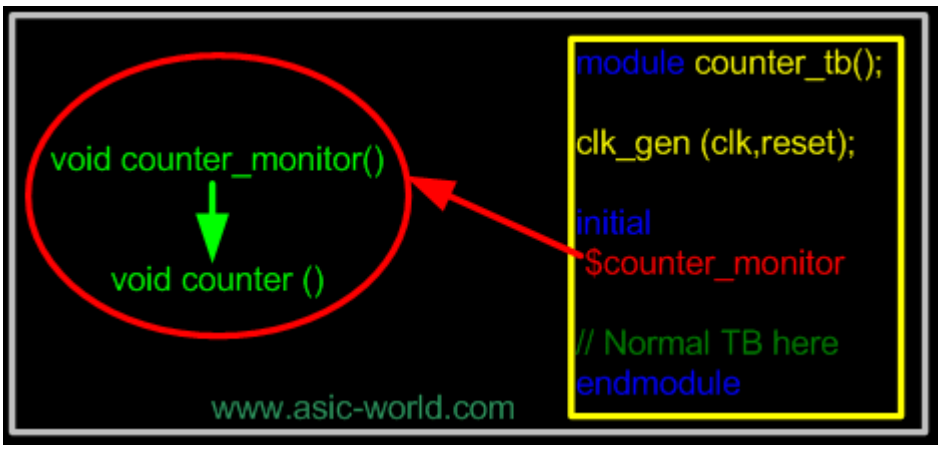
- C modeli çağırmak demek, giriş sinyallerinde değişiklik var demektir(wire veya reg tiplerinde).
- Almak demek, değişen sinyal(veya başka bir sinyal) değerinin Verilog kodundan C koduna dönüştürülmesi.
- Sürmek demek C kodundan Verilog koduna herhangi bir sinyal değerinin alınmasıdır.

Burada Verilog PLI'ın yukarıdaki gereksinimleri destekleyen rutinler(fonksiyonlar) kümesi vardır.

PLI Uygulama Belirtimi(Specification)

PLI kullanarak bizim kötü şöhretli sayaç testbencimiz için gereksinimleri tanımlayalım. Bizim PLI fonksiyonumuzu $counter_monitor olarak çağıracağız.

- Sayaç(counter) lojiğini C 'de gerçekle(implement)
- Kontrolör(Checker) lojiğini C 'de gerçekle(implement)
- Kontrolör başarısız olduğunda simülasyonu sonlandır.



C fonksiyonunun Çağırılması

C'de sayaç yazmak çok kolaydır, ancak ne zaman sayaç değerini arttıracağız? Güzel saat sinyalindeki değişimi izlemeliyiz. (Not: Bu yolla, reset ve clock sinyallerini Verilog kodundan sürmek iyi bir fikirdir.) Saat değiştiğinde, sayaç fonksiyonunun gerçekleştirilmesi gerekmektedir. Bu aşağıdaki rutin kullanılarak meydana getirilir.

- acc_vcl_add rutinini kullan, bunun sözdizimi Verilog PLI LRM'de bulunabilir.

acc_vcl_add rutini temel olarak bize sinyaller listesini izlememizi sağlar ve izlenen sinyallerden biri değiştiğinde kullanıcı tanımlı fonksiyonu (alıcı(consumer) C rutini adı verilen) çağırır. VLC rutinini dört tane argümanı vardır:

- izlenen objeyi işleyin
- Alıcı(Consumer C) rutinini objenin değeri değiştiğinde çağırın
- Alucı(Consumer) C rutinine dizgi gönderilmiş olabilir.
- Öntanımlı VCL bayrakları(flags): vcl_verilog_logic lojik izleme için, vcl_verilog_strength güç izleme içindir

acc_vcl_add(net, display_net, netname, vcl_verilog_logic);

Detaylara dalmadan önce aşağıdaki koda bakalım.

C Kodu - Temel

Counter_monitor bizim C fonksiyonumuzdur, Verilog Testbenç tarafından çağırılabilir. Diğer C kodu için, geliştirdiğimiz uygulamaya has başlık dosyalarını eklememiz gerekmektedir. Bizim durumumuzda acc rutinlerini ekleme dosyalarını eklememiz gerekmektedir.

Erişim rutini acc_initialize erişim rutinleri için ortamı ilk kullanıma hazırlar ve program başka erişim rutinleri tarafından çağırılmadan önce sizin C-dili uygulama programınız tarafından çağırılmış olmalıdır. Erişim rutinini çağıran bir C-dili uygulamadan çıkmadan önce programın sonunda acc_close çağırılarak erişim rutin ortamından da çıkılması gerekmektedir.

```
 1 #include "acc_user.h"
 2
 3 handle clk ;
 4 handle reset ;
 5 handle enable ;
 6 handle dut_count ;
 7 void counter ();
 8
 9 void counter_monitor() {
10   acc_initialize();
11   clk = acc_handle_tfarg(1);
12   reset = acc_handle_tfarg(2);
13   enable = acc_handle_tfarg(3);
14   dut_count = acc_handle_tfarg(4);
15   acc_vcl_add(clk,counter,null,vcl_verilog_logic);
16   acc_close();
17 }
18
19 void counter () {
20   io_printf("Clock changed state\n");
21 }
```

You could download file counter.c here

Verilog objelerine erişmek için bir tanıtıcı(handle) kullanırız. Bir tanıtıcı(handle), tasarım hiyerarşisindeki belirli bir obje için işaretçi(pointer) olan ön tanımlı bir veri tipidir. Herbir tanıtıcı(handle) erişilebilir objenin tek bir örneğinin erişim rutinleri hakkında bilgi taşır; bilgi objenin tipi, artı nasıl ve nerede objeyle ilgili bilgi bulunur hakkındadır. Fakat objeye özgü bilgileri nasıl tanıtıcıya göndeririz? Güzel bunu yapmanın birçok yolu vardır, fakat şimdilik, bunu Verilog'dan parametre olarak $counter_monitor'e göndereceğiz: bu parametrelere C-programın içinden acc_handle_tfarg() ile erişilebilir. Argümanların sayısı koddaki gibidir.

Yani, clk = acc_handle_tfarg(1) temelde clk'u ilk paramatre gönderiminin tanıtıcısı yapar; benzer şekilde tüm tanıtıcıları(handles) atayabiliriz. Şimdi clk'u sinyal listesine acc_vcl_add(clk,counter,null,vcl_verilog_logic) rutiniyle ekleyip izleyebiliriz. Burada clk bir tanıtıcıdır(handle) ve counter da clk değiştiğinde gerçekleştirilecek kullanıcı fonksiyonudur.

Counter() fonksiyonunun herhangi bir açıklamaya ihtiyacı yoktur, basit bir "Hello world"-tipi bir koddur.

 **Verilog Kodu**

Altta sayaç örneği için basit testbenç kodu vardır. Gösterilen sözdiziminda C-fonksiyonunu çağırdık. Eğer gönderilen obje bir örnekse(instant),çift tırnak arasında gönderilir. Eğer tüm objelerimiz net veya wire(tel) ise, bunları çift tırnak içinde göndermeye gerek yoktur.

```
 1 module counter_tb();
 2  reg enable;
 3  reg reset;
 4  reg clk_reg;
 5  wire clk;
 6  wire [3:0] count;
 7
 8 initial begin
 9   enable = 0;
10   clk_reg = 0;
```

```
11   reset = 0;
12   $display("%g , Asserting reset", $time);
13    #10  reset = 1;
14    #10  reset = 0;
15   $display ("%g, Asserting Enable", $time);
16    #10  enable = 1;
17    #55  enable = 0;
18   $display ("%g, Deasserting Enable", $time);
19    #1  $display ("%g, Terminating Simulator", $time);
20    #1  $finish;
21 end
22
23 always begin
24    #5  clk_reg =  ! clk_reg;
25 end
26
27 assign clk = clk_reg;
28
29 initial begin
30   $counter_monitor (counter_tb.clk, counter_tb.reset,
31     counter_tb.enable, counter_tb.count);
32 end
33
34 counter U(
35 .clk (clk),
36 .reset (reset),
37 .enable (enable),
38 .count (count)
39 );
40
41 endmodule
```

You could download file pli_counter_tb.v here

Kullandığımız simülatöre bağlı olarak, derleme ve yürütme adımları değişebilir. Yukarıdaki kodu C kodu ile çalıştırdığınızda aşağıdaki çıktıyı alacaksınız:

```
0 , Asserting reset
Clock changed state
Clock changed state
Clock changed state
20, Asserting Enable
Clock changed state
Clock changed state
Clock changed state
Clock changed state
Clock changed state
Clock changed state
Clock changed state
Clock changed state
Clock changed state
Clock changed state
Clock changed state
Clock changed state
Clock changed state
Clock changed state
85, Deasserting Enable
Clock changed state
86, Terminating Simulator
```

C Kodu- Tamamı

Şimdi bizim fonksiyonumuz saatte bir değişiklik olduğunda çağırılabiliyor, yani sayaç kodumuzu yazabiliriz. Ancak,bir problem var: her zaman sayaç fonksiyonundan çıkışdığından yerel değişken(local variable) değerini kaybediyor. Değişkenlerin durumlarını korumanın birkaç yolu vardır.

- Sayaç değişkenin global olarak bildirilmesi
- tf_setworkarea() ve tf_getworkarea() rutinleri kullanılarak yerel değişkenlerin değerleri kaydedilebilir ve alınabilir.

Sadece bir tane değişkenimiz olduğundan ilk çözümü kullanacağız, sayacı global olarak bildireceğiz.

Sayaç için eşdeğer bir model yazmak için, saat(clock),sıfırlama(reset), etkin(enable) sinyal girişleri DUT için gereklidir ve DUT 'ın çıkışı count(sayaç) kod denetçisi için gereklidir. Verilog kodundaki değerleri okumak için PLI rutinimiz vardır

acc_fetch_value(tanıtıcı,"biçim")
(acc_fetch_value(handle,"format"))

Fakat dönen değer bir dizgidir(string), yani eğer çoklu-bit vektör sinyali bu rutin kullanılarak okunduysa dizgiyi(string) tamsayıya(integer) dönüştürmemiz gerekmektedir. pli_conv bu dönüşümü yapan bir fonksiyondur. tf_dofinish() rutini DUT ve TB sayaç değerleri eşleşmediği zaman diğer bir deyimle simülasyon uyumsuzluk gösterdiğinde simülasyonu sonlandırır.

```
 1 #include "acc_user.h"
 2
 3 typedef char * string;
 4 handle clk ;
 5 handle reset ;
 6 handle enable ;
 7 handle dut_count ;
 8 int count ;
 9 int sim_time;
10 string high = "1";
11 void counter ();
12 int pli_conv (string in_string, int no_bits);
13
14 void counter_monitor() {
15   acc_initialize();
16   clk       = acc_handle_tfarg(1);
17   reset     = acc_handle_tfarg(2);
18   enable    = acc_handle_tfarg(3);
19   dut_count = acc_handle_tfarg(4);
20   acc_vcl_add(clk,counter,null,vcl_verilog_logic);
21   acc_close();
22 }
23
24 void counter () {
25   p_acc_value value;
26   sim_time = tf_gettime();
27   string i_reset = acc_fetch_value(reset,"%b",value);
28   string i_enable = acc_fetch_value(enable,"%b",value);
```

```
29   string i_count = acc_fetch_value(dut_count,"%b",value);
30   string i_clk = acc_fetch_value(clk, "%b",value);
31   int size_in_bits= acc_fetch_size (dut_count);
32   int tb_count = 0;
33   // Sayaç fonksiyonu burdan devam eder
34   if (*i_reset == *high) {
35     count = 0;
36     io_printf("%d, dut_info : Counter is reset\n", sim_time);
37   }
38   else if ((*i_enable == *high) && (*i_clk == *high)) {
39     if ( count == 15 ) {
40       count = 0;
41     } else {
42       count = count + 1;
43     }
44   }
45   // Sayaç kontrolörü fonksiyonu kontrolör lojiği burdan başlar
46   if ((*i_clk  ! = *high) && (*i_reset  ! = *high)) {
47     tb_count = pli_conv(i_count,size_in_bits);
48     if (tb_count  ! = count) {
49         io_printf("%d, dut_error : Expect value %d, Got value %d\n",
50             sim_time, count, tb_count);
51    tf_dofinish();
52     } else {
53         io_printf("%d, dut_info  : Expect value %d, Got value %d\n",
54             sim_time, count, tb_count);
55     }
56   }
57 }
58
59 // Çok bitli vektör için tamsayıya dönüştürme
60 int pli_conv (string in_string, int no_bits) {
61   int conv = 0;
62   int i = 0;
63   int j = 0;
64   int bin = 0;
65   for ( i = no_bits-1; i >= 0; i = i - 1) {
66     if (*(in_string + i) == 49) {
67       bin = 1;
68     } else if (*(in_string + i) == 120) {
69       io_printf ("%d, Warning : X detected\n", sim_time);
70       bin = 0;
71     } else if (*(in_string + i) == 122) {
72       io_printf ("%d, Warning : Z detected\n", sim_time);
73       bin = 0;
74     } else {
75       bin = 0;
76     }
77     conv = conv + (1 << j)*bin;
78     j ++;
79   }
80   return conv;
81 }
82
```

You could download file pli_full_example.c here

Yukarıdaki kodu simülatörle derleyebilir ve çalıştırabilirsiniz.

Simülatör ile Bağlama(Linking)

Daha önce gördüğümüz sayaç örneğini aşaıdaki simülatörler ile bağlayacağız. Eğer diğer simülatörlerle nasıl bağlanacağını görmek için, bana bildirin.

- VCS
- Modelsim

Ben eğitsellerimde Linux kullanıyorum, bu yüzden eğer windows veya solariste bağlama nasıl yapılır öğrenmek için simülatörün kullanım kılavuzuna bakınız.

VCS

VCS simülatörü ile bir tab(sekme) dosyası yaratmalısınız. Bizim örneğimizde tab(sekme) dosyası aşağıdakine benzer şekilde görünmektedir:

$counter_monitor call=counter_monitor acc=rw:*

Burada $counter_monitor Verilog kodun kullanacağı kullanıcı tanımlı fonksiyonun ismidir, call=counter_monitor ise $counter_monitor Verilogda çağırıldığı zaman çağırılacak C fonksiyonudur. Acc=rw:* ise bize erişim(access) rutinlerine read(okuma) ve write(yazma) ile simülatörün iç verisine erişiyoruz. "**:***" anlamı ise tasarımdaki tüm modüllere uygulanabilir demektedir.

Kodu derlemek için komut satırı seçenekleri :

vcs -R -P pli_counter.tab pli_counter_tb.v counter.v pli_full_example.c -CFLAGS "-g -I$VCS_HOME/`vcs -platform`/lib" +acc+3

Geri çağırma(callback) kullandığımız için +acc+3; kullanmalıyız geri kalan seçenekler oldukça kolay, VCS kullanım klavuzuna bakabilirsiniz.

Modelsim

VCS'de olduğu gibi Modelsim simülatörü de PLI ile haberleşmenin kendi yollarını belirlemiştir. Verilogda kullanacağımız tüm kullanıcı tanımlı fonksiyonların ve bunlara uygun çağırılacak C fonksiyonlarının fonksiyon listesini oluşturmamız gerekmektedir. VCS ile farklı şekilde, bunu aşağıda gösterdiğimiz şekilde bir C dosyasının içinde yapmamız gerekmektedir.

```
1 #include "veriuser.h"
2 #include "pli_full_example.c"
3
4 s_tfcell veriusertfs[] = {
5   {usertask, 0, 0, 0, counter_monitor, 0, "$counter_monitor"},
6   {0}  // last entry must be 0
7 };
```

You could download file pli_full_example_modelsim.c here

vlib work

vlog pli_counter_tb.v counter.v

gcc -c -g -I$MODEL/include pli_full_example_modelsim.c

ld -shared -E -o pli_full_example.sl pli_full_example_modelsim.o

vsim -c counter_tb -pli pli_full_example.sl

vsim komut satırında, simülasyonu başlatmak için "run -all" yazın.

Windowsda nasıl derleme ve bağlama yapacağınızı öğrenmek için Modelsim kullanım klavuzuna bakınız.

Sayaç Simülasyon Çıktısı

```
0 , Asserting reset
10, dut_info : Counter is reset
15, dut_info : Counter is reset
20, Asserting Enable
20, dut_info  : Expect value 0, Got value 0
30, dut_info  : Expect value 0, Got value 0
40, dut_info  : Expect value 1, Got value 1
50, dut_info  : Expect value 2, Got value 2
60, dut_info  : Expect value 3, Got value 3
70, dut_info  : Expect value 4, Got value 4
80, dut_info  : Expect value 5, Got value 5
85, Deasserting Enable
86, Terminating Simulator
```

PLI Rutinleri

PLI 1.0 iki tip rutin sağlar, bunlar

- Erişim rutinleri (access routines)
- Görev ve fonksiyon rutinleri (task and function routines).

PLI 2.0 ‘da erişim rutinleri ile görev ve fonksiyon rutinlerini VPI rutinleri için birleştiriyor, ve ayrıca PLI 1.0 daki kargaşayı açıklığa kavuşturmaktadır.

Erişim Rutinleri (Access Routines)

Erişim rutinleri, Verilog-HDL içindeki bilgiye prosedürel erişimi sağlayan C programlama dili rutinleridir. Erişim rutinleri aşağıdaki iki işlemden birini gerçekleştirir:

Okuma(Read) İşlemi: sizin devre tasarımınızdaki belirli objelerin doğrudan veri yapılarının içinden gelen bilgilerin okunmasıdır. Erişim rutinleri aşağıda belirtilen objeler ile ilgili bilgileri okuyabilir:

- Modül örnekleri(instances)
- Modül portları

- Modül yolları(paths)
- Modül-içi yollar(Inter-module paths)
- En üst seviye modüller(Top-level modules)
- Temel örnekler(Primitive instances)
- Temel terminaller (Primitive terminals)
- Net
- Yazmaçlar(Registers)
- Parametreler(Parameters)
- Belirtim parametreleri(Specparams)
- Zamanlama kontrolleri(Timing checks)
- İsimlendirilmiş olaylar(Named events)
- Integer, real ve time değişkenleri

Yazma(Write) İşlemi: sizin devre tasarumınızdaki objelerin iç veri yapılarına yeni bilgilerin yazılmasıdır. Erişim rutinleri aşağıdaki objelere yazabilir:

- Modül-içi yollar (Inter-module paths).
- Modül yolları(Module paths).
- Temel örnekler(Primitive instances).
- Zamanlama kontrolleri(Timing checks).
- Yazmaç lojik değerleri(Register logic values).
- Ardışıl UDP lojik değerleri(Sequential UDP logic values).

Erişim rutinleri tarafından gerçekleştirilen işlemlere göre aşağıdaki gibi 6 kategoride sınıflanır.

- Getirme(Fetch): Bu rutinler dizayn hiyerarşisindeki farklı objeler hakkında çeşitli bilgileri döndürür.
- Tanıtıcı(Handle): Bu rutinler dizayn hiyerarşisindeki farklı objeler hakkında çeşitli tanıtıcıları döndürür.
- Değiştirme(Modify) : Bu rutinler dizayn hiyerarşisindeki farklı objelerin çeşitli değerleri değiştirir.
- Sonraki(Next): Bir döngü yapısının içinde kullanıldığında, next rutini tasarım hiyerarşisi içinde belirli referans objelerle ilgili verilen tipteki herbir objeyi bulur.
- Yardımcı(Utility): Bu rutinler erişim rutin ortamını düzenleme ve ilk değer verme gibi çeşitli işlemleri gerçekleştirir.
- Vcl : Değer Değişimi Bağlama(The Value Change Link (VCL)) bir PLI uygulamasına seçilen objenin değer değişiminin izlenmesine izin verir.

Erişim Rutinleri Referansı

Rutin	Açıklama
acc_handle_scope()	Bu fonksiyon bi objenin kapsamının tanıtıcısını döndürür. Kapsam, modül,görev,fonksiyon, isimlendirilimiş paralel blok veya isimlendirilmiş ardışıl blokdan biri olabilir.
acc_handle_by_name()	Bu rutin açıkça belitilmiş isim ve kapsam(scope) temelli Verilog-HDL objesinin tanıtıcısını döndürür.
acc_handle_parent()	Bu fonksiyon bir objenin temel örneğini veya modül örneğinin üstünü(parent) tanıtıcısını döndürür
acc_handle_port()	Bu fonksiyon modül port'u için tanıtıcı döndürür
acc_handle_hiconn()	Bu fonksiyon hiyerarşik olarak daha üst net bağlantısını skaler modül portu veya vektör portunun bir biti için döndürür
acc_handle_loconn()	Bu fonksiyon hiyerarşik olaraka daha alt net bağlantıyı skaler bir modül portu veya vektör portunun bir bit için döndürür
acc_handle_path()	Bu fonksiyon modül içi yolu yani çıkış portundan giriş portuna bağlantıyı gösteren yol için bir tanıtıcı döndürür
acc_handle_modpath()	Bu fonksiyon bir modülün yolu için tanıtıcı döndürür
acc_handle_datapath()	Bu fonksiyon bir modül örneği için belirtilmiş kenar-hassasiyetli modül yolu için bir veri yolu tanıtıcısı döndürür
acc_handle_pathin()	Bu fonksiyon bir modül yolu kaynağına ilk net bağlantısı için tanıtıcıyı döndürür
acc_handle_pathout()	Bu fonksiyon bir modül yolu varışına ilk net bağlantısı için tanıtıcı döndürür
acc_handle_condition()	Bu fonksiyon belitilmiş yol için koşul ifadesi tanıtıcısını döndürür
acc_handle_tchk()	Bu fonksiyon bir modülün(veya hücrenin) belirtilmiş zamanlama kontrolü için tanıtıcıyı döndürür
acc_handle_tchkarg1()	Bu fonksiyon zamanlama kontrolünün ilk argümanına bağlı net için tanıttıcı döndürür
acc_handle_tchkarg2()	Bu fonksiyon zamanlama kontrolünün ikinci argümanına bağlı net için tanıtıcı döndürür
acc_handle_simulated_net()	Bu fonksiyon bir argümandan geçen daraltılmış net ile ilgili simüle edilmiş net'i döndürür
acc_handle_terminal()	Bu fonksiyon temel(primitive) terminal için tanıtıcı döndürür
acc_handle_conn()	Bu fonksiyon temel terminale bağlı net için tanıtıcı döndürür
acc_handle_tfarg()	Bu fonksiyon C-dili rutini ilgili sistem görev veya fonksiyonunun belirtilmiş argümanı için tanıtıcı döndürür

acc_fetch_attribute()	Bu fonksiyon bir parametrenin değerini ya da kaynak tanımınızdaki özel bir parametreyi isimlendirilmiş olarak bir öznitelik(attribute) olarak döndürür
acc_fetch_paramtype()	Bu fonksiyon bir parametrenin veri tipini üç öntanımlı tamsayı sabitlerinden biri olarak döndürür.
acc_fetch_paramval()	Bu fonksiyon bir parametrenin değerini veya özelparametreyi(specparam) döndürür
acc_fetch_defname()	Bu fonksiyon bir modül örnekleminin veya temel örneklemin tanımlayıcı ismini bir işaretçi(pointer) olarak döndürür
acc_fetch_fullname()	Bu fonksiyon herhangi isimlendirilmiş obje veya modül yolunun tam hiyerarşik ismini işaretçi(pointer) olarak döndürür
acc_fetch_name()	Bu fonksiyon isimlendirilmiş herhangi obje veya modül yolu için örneklem ismini işaretçi(pointer) olarak döndürür
acc_fetch_delays()	Bu fonksiyon farklı objeler için farklı gecikme değerlerini getirir(fetch)
acc_fetch_size()	Bu fonksiyon bir net,register veya port'un bit boyutunu döndürür.
acc_fetch_range()	Bu fonksiyon bir vektörün bit değer kümesini(range) bildirir
acc_fetch_tfarg()	Bu fonksiyon sizin C-dili rutininizle ilgili belirtilmiş sistem görev ve fonksiyon argümanının değerini döndürür
acc_fetch_direction()	Bu fonksiyon bir port veya terminalin yönünü öntanımlı tamsayı sabitlerinden biri olarak döndürür
acc_fetch_index()	Bu fonksiyon bir port veya terminal için sıfır tabanlı tamsayı indis döndürür
acc_fetch_edge()	Bu fonksiyon giriş yolu veya çıkış terminalini öntanımlı tamsayı sabitleri olarak kenar (belirteci)tipi olarak döndürür.
acc_set_value()	Bu fonksiyon lojik bildiren veya bir net,yazmaç veya değişkenin güç değerinin karakter dizgisi için bir işaretçi(pointer) döndürür.
acc_initialize()	Bu fonksiyon erişim rutinleri için ortamı başlatmaktadır(initialize)
acc_close()	Bu fonksiyon erişim rutinleri tarafından kullanılan iç belleği serbest bırakır; tüm konfigürasyon parametrelerini varsayılan değerlerine sıfırlar.
acc_configure()	Bu fonksiyon çeşitli erişim rutinlerini kontrol eden işlemlerin parametrelerini atar
acc_product_version()	Bu fonksiyon Verilog simülatörünün hangi versiyonunun erişim rutinine bağlandığını bildiren karakter dizgisinin(character string) işaretçisini(pointer) döndürür.

acc_version()	Bu fonksiyon sizin erişim rutin yazılımınızın versiyon numarasını belirten karakter dizgisi işaretçesini (pointer) döndürür.
acc_count()	Bu fonksiyon belirli bir referan objesiyle ilişkili obje sayısını tamsayı olarak döndürür
acc_collect()	Bu fonksiyon belirli bir referans objesi ile ilişkili tüm objelerin tanıtıcılarını içeren bir dizi için işaretçi(pointer) döndürür
acc_free()	Bu fonksiyon acc_collect ile ayrılan belleği serbest bırakır
acc_compare_handles()	Bu fonksiyon eğer iki giriş tanıtıcısı aynı objeye başvuruyorsa true(doğru) dönderiyor
acc_object_in_typelist()	Bu fonksiyon bir objenin tipe veya tamtipe uyup uymadığını veya bir giriş dizisinde belirtilmiş bir özelliğe uyup uymadığına karar verir.
acc_object_of_type()	Bu fonksiyon bir objenin tipe veya tamtipe uyup uymadığını veya belirtilmiş bir özelliğe uyup uymadığına karar verir.
acc_next_cell()	Bu fonksiyon belirtilen hiyerarşi altındaki modüldeki bölgenin içindeki bir sonraki hücre örneklemini döndürür
acc_next_child()	Bu fonksiyon bir modülün birsonraki çocuğunu(child) döndürür
acc_next_modpath()	Bu fonksiyon bir modülün bir sonraki yolunu döndürür
acc_next_net()	Bu fonksiyon bir modülün bir sonraki net'ini döndürür
acc_next_parameter()	Bu fonksiyon bir modül içindeki bir sonraki parametreyi döndürür
acc_next_port()	Bu fonksiyon port listesinde belirtilen sıraya göre bir modülün bir sonraki giriş,çıkış veya girişçıkış portlarını döndürür
acc_next_portout()	Bu fonksiyon port listesinde belirtilen sıraya göre bir modülün bir sonraki çıkış veya girişçıkış portunu döndürür
acc_next_primitive()	Bu fonksiyon bir modülün içindeki bir sonraki kullanıcı tanımlı temel(UDP) ,kapı veya anahtarı döndürür
acc_next_specparam()	Bu fonksiyon bir modülün içindeki bir sonraki özelparametreyi(specparam) döndürür
acc_next_tchk()	Bu fonksiyon bir modülün içindeki bir sonraki zamanlama kontrolünü döndürür
acc_next_terminal()	Bu fonksiyon bir kapının,anahtarın ya da kullanıcı tanımlı temellerin(UDP)bir sonraki terminalini döndürür
acc_next()	Bu fonksiyon bir kapsamdaki, obje tipi dizisindeki belirtilen herbir tip için bir sonraki objeyi döndürür
acc_next_topmod()	Fonksiyon,bir sonraki en üst seviye modülü döndürür

acc_next_cell_load()	Bu fonksiyon bir hücre içindeki bir sonraki net yüklemesini(load) döndürür
acc_next_load()	Bu fonksiyon bir net tarafından sürülen bir sonraki temel terminali(primitive terminal) döndürür
acc_next_driver()	Bu fonksiyon bir neti süren bir sonraki temel terminali döndürür
acc_next_hiconn()	Bu fonksiyon bir modülün portu için hiyerarşik olarak bir yüksek seviyedeki net bağlantısını döndürür
acc_next_loconn()	Bu fonksiyon bir modülün portu için hiyerarşik olarak bir alt seviyedeki net bağlantısını döndürür
acc_next_bit()	Bu fonksiyon bir genişletilmiş vektör portu veya genişletilmiş vektör neti içindeki herbir bit'in tanıtıcısını döndürür
acc_next_input()	Bu fonksiyon belirtilmiş modül yolunun veya veri yolunun bir sonraki giriş yolu terminalinin tanıtıcısını döndürür
acc_next_output()	Bu fonksiyon belirtilmiş modül yolunun veya veri yolunun bir sonraki çıkış yolu terminalinin tanıtıcısını döndürür

Erişim Rutinleri Kullanarak Program Akışı

Daha önceki örneklerde de görüldüğü üzere, bir kullanıcı uygulaması yazmadan önce gerçekleştirilmesi gereken bazı adımlar vardır. Bunlar aşağıdaki programda gösterilmiştir.

```
1  #include <acc_user.h>
2
3 void pli_func() {
4   acc_initialize();
5   // Ana Gövde (Main body): kullanıcı uygulama kodunu buraya yerleştirin
6   acc_close();
7 }
```

You could download file acc_flow.c here

- acc_user.h : erişim rutinleriyle ilgili tüm veri tipleri
- acc_initialize() : değişkenlere ilk değer atanması(initialize) ve ortamın kurulması
- main body : kullanıcı tanımlı uygulama
- acc_close() : acc_initialize() fonksiyonu ile gerçekleştirilen olayların geri alınması

Objeler için Tanıtıcı(Handle)

Handle(tanıtıcı) öntanımlı bir veri tipidir: C deki pointer(işaretçi) benzeri bir yapısı vardır, tasarım veritabanındaki herhangi bir türdeki objeyi işaret etmek için kullanılabilir. Handle (tanıtıcı) erişim rutin metodolojisinin ve en önemlisi bu bölümde belirtilen PLI 2.0 yeni konseptinin omurgasını oluşturmaktadır.

Bildirimler(Declarations)

- handle my_handle;
- handle clock;
- handle reset;

Değişen Değer Bağlaması (Value change link(VCL))

VCL, bir PLI uygulamasının seçilmiş objeler için değer değişiminin izlenmesine olanak vermektedir. VCL aşağıdaki objeler için değer değişimlerini görüntüleyebilir:

- Olaylar(Events)
- Skaler(Scalar) ve vektör yazmaçlar (vector registers)
- Skaler net'ler
- Genişletilmiş net vektörlerinin bir seçimleri(Bit-selects of expanded vector nets)
- Genişletilmemiş net vektörleri(Unexpanded vector nets)

VCL aşağıda belirtilen objeler hakkında bilgi veremez:

- Genişletilmemiş net veya yazmaç vektörlerinin bit seçimleri
- Bölüm seçimleri (Part-selects)
- Bellekler (Memories)
- İfadeler (Expressions)

Yardımcı Rutinler (Utility Routines)

Verilog sistemi ve kullanıcıların rutinleri arasındaki etkileşim Verilog sistemi tarafından destekelenen bir rutinler kümesi tarafından halledilmektedir. PLI 1.0 da tanımlanan kütüphane fonksiyonlarında parametre gönderilerek geniş bir yelpazede işlemler gerçekleştirilmektedir. Sistem çağrısı bir simülasyon senkronizasyonu yapmak veya koşullu program kesme noktası gerçekleştirmek için kullanılmıştır.

Bu rutinler ayrıca Utility(yardımcı) rutinler olarak adlandırılır. Bunların birçoğu iki biçimdedir: biri mevcut çağrı veya örneklemin üstesinden gelirken; diğeri mevcut olanı dışındaki örneklemin üstesinden gelir ve örnek işaretçi(pointer) ile başvurulur.

Yardımcı Rutinlerin(Utility Routines) Sınıflandırılması

Rutin	Açıklama
tf_getp()	Mevcut örneklem için bir görev(task) veya fonksiyon(function) argümanının(parametrenin) tamsayı değerini alır
tf_putp()	Mevcut görev veya fonksiyonun bir argümanına 32-bitlik tamsayı değeri gönderir.
tf_getrealp()	Mevcut örneklem için bir görev veya fonksiyon argümanının reel değerini alır.
tf_putrealp()	Mevcut görev veya fonksiyonun bir argümanına 64-bit kayan noktalı değer gönderir.
tf_getlongp()	Mevcut örneklem için (64 bitlik) uzun argüman değerini alır.

tf_putlongp()	Bir görev veya fonksiyonun mevcut örneklemine 64-bitlik tamsayı argüman değeri gönderir.
tf_strgetp()	Biçimlendirilmiş argüman değerini alır.
tf_getcstringp()	Bir parametrenin değerini karakter dizgisi olarak alır.
tf_strdelputp()	Yazma değeri bir karakter dizgisi olarak belirtildiğinde belirtilen gecikme ile bir argümana değer yazılır.
tf_strlongdelputp()	Yazma karakteri bir karakter dizgisi olarak belirtildiğinde belirtilen (64-bitlik)gecikme ile bir argümana değer yazılır.
tf_strrealdelputp()	Yazma karakteri bir karakter dizgisi olarak belirtildiğinde belirtilen reel gecikme değeri ile bir argümana değer yazılır.
tf_copypvc_flag()	Parametre değerini kopyala flag(bayrağı) değiştir.
tf_movepvc_flag()	Argüman değerini taşı flag(bayrağı) değiştir.
tf_testpvc_flag()	Mevcut görev/fonksiyon örneklemi için argüman değeri değişim bayrağını test et.
tf_getpchange()	Mevcut görev veya fonksiyon örnekleminde değişen değerlerin bir sonraki argüman sayısını alır.
tf_gettime()	Mevcut simülasyon zamanını alır.
tf_getlongtime()	Görev veya fonksiyon çağırma zaman aralığında, mevcut simülasyon zamanını 64-bitlik tamsayı değeri olarak alır.
tf_getrealtime()	Mevcut simülasyon zamanını çağırılan modülün zamanölçeğiyle ölçekli olarak alır.
tf_strgettime()	Mevcut simülasyon zamanını karakter dizgisi olarak alır.
tf_gettimeunit()	Mevcut görev çağrısını içeren modül için zaman ölçeği birimini alır.
tf_gettimeprecision()	Mevcut görev çağrısını içeren modül için zaman ölçeği hassasiyetini alır.
tf_synchronize()	Mevcut sistem görev veya fonksiyon argümanlarının işlemlerini senkronize eder.
tf_rosynchronize()	Mevcut siste görev veya fonksiyon argümanlarının işlemlerini mevcut zaman slotunu olay üretimi ile bastırarak senkronize eder.
tf_getnextlongtime()	Bir sonraki tarifeli(belirlenmiş) simülasyon olayının zamanını alır.
tf_setdelay()	Belirli bir tamsayı simülasyon zamanında bir kullanıcı görevini yeniden aktive eder.
tf_setlongdelay()	Belirli bir 64-bit tamsayı ile belirtilmiş simülasyon zamanında bir kullanıcı görevini yeniden aktive eder.
tf_setrealdelay()	Belirli bir simülasyon zamanındaki bir kullanıcı görevini aktive eder.
tf_clearalldelays()	Tüm tarfeli-belirlenmiş yeniden aktifleştirme gecikmelerini siler
io_printf	Biçimlendirilmiş bir mesajı yazılım üretiminin çıkış kanalına yazdırır bu PLI uygulamalarında ve bir log dosyası için çağırılır.
io_mcdprintf()	Biçimlendirilmiş mesajı bir veya daha fazla dosyaya yazar.
tf_warning()	Bir uyarı rapor eder.

tf_error()	Bir hata rapor eder.
tf_text()	Hata bilgilerini, tf message ile daha sonra tektardan göstermek için kaydeder.
tf_message()	Kullanıcı tarafından üretilmiş hata mesajını standart hata mesajı sözdizimi ile rapor eder.
tf_getinstance()	Mevcut örneklem işaretçisini(pointer) alır.
tf_mipname()	Mevcut modül örnekleminin hiyerarşik yol ismini alır.
tf_spname()	Bir kapsamın hiyerarşik yol adını dizgi olarak alır.
tf_setworkarea()	Bir hücredeki mevcut kullanıcı görev/fonksiyon örneklemi için bir çalışma alanı işaretçisi saklar.
tf_getworkarea()	tf_setworkarea() rutini ile kaydedilmiş çalışma alanı işaretçisini alır.
tf_nump()	Mevcut görev veya fonksiyon örneklemi argüman listesindeki argüman sayısını alır.
tf_typep()	Mevcut görev veya fonksiyonun bir argümanının veri tipini alır.
tf_sizep()	Mevcut görev veya fonksiyon örnekleminin bir argümanının boyutunu alır.
tf_dostop()	Etkileşimli modu etkinleştir.
tf_dofinish()	Simülasyonu sonlandır.
mc_scan_plusargs()	Komut satırı artı(+) seçenekleri al.
tf_compare_long()	İki 64-bitlik tamsayı değerini karşılaştır.
tf_add_long()	İki 64-bit tamsayı değerini topla.
tf_subtract_long()	Bir 64-bit tamsayı değerini başka birinden çıkart.
tf_multiply_long()	İki 64-bit tamsayı değerlerini çarp.
tf_divide_long()	ki 64-bit tamsayı değerlerini böl.
tf_long_to_real()	Long integer'ı real sayıya çevir.
tf_longtime_tostr()	64-bit tamsayı zaman değerini dizgiye(string) çevir.
tf_real_tf_long()	Bir reel(real) sayıyı long(64-bit) tamsayıya çevir.
tf_write_save()	Kayıt dosyasının sonuna bir veri bloğunu ekle.
tf_read_restart()	Daha önce yazılmış bir kayıt dosyasından bir blok veriyi alır.

 **Diğer Kullanışlı Fonksiyonlar**

acc_* ve tf_* rutinlerinden başka, simülatörler aşağıdaki listedeki fonksiyonları destekler.

- veriusertfs
- endofcompile_routines
- err_intercept

veriusertfs

VCS'den başka tüm simülatörler için veriusertfs desteklenir. Veriusertfs dizisi, PLI'ın kullandığı kullanıcı tarafından yazılmış Verilog Simülatör ile ilişkilendirilmiş sistem görev ve fonksiyonlarını uygulamaların bir tablosudur. Aşağıdaki tabloda her bir yeni sistem görevi ve fonksiyon yaratmak istediğinizde dizide doldurduğunuz alanları göstermektedir.Bu alanlar dizide gösterildiği sırayla bulunur.

Alan	Açıklama
type (tip)	Bu anahtar sözcük, rutinin bir sistem görevi yada fonksiyonu olduğunu belirtir; usertask(kullanıcıgörevi) bir sistem görevini tanımlar veya userfunction(kullanıcıfonksiyonu) kullanıcı tanımlı değer döndüren veya userrealfunction gerçek bir değer döndüren bir sistem fonksiyonunu tanımlar.
data (veri)	Veri argümanı(0 bir veri argümanı gönderilmeyecek demektir).
checktf	Sistem görev veya fonksiyonunu kontrol eden bir kullanıcı destekli opsiyonel bir rutin için işaretçidir(pointer) (0 checkf rutini yoktur demektir).
sizetf	Bir sistem fonksiyonu tarafından döndürülmüş değerin bit büyüklüğünü bir kullanıcı destekli rutin için işaretçidir(pointer) (0 sizetf rutini olmadığını anlamına gelir).
calltf	Veritool sistem görev veya fonksiyonlarını simülasyon esnasında gerçekleştirirken çağırılan ana rutin yani kullanıcı destekli uygulama için işaretçidir(pointer) (0 calltf rutini yoktur demektir).
misctf	Opsiyonel çokyönlü rutin(optional miscellaneous routine) için işaretçidir(pointer) (0 misctf rutini yoktur demektir).
$tfname	Tırnak içine alınmış tam dizgi sistem görev veya fonksiyonunun ismini tanımlar;$ ilk karakter olmalıdır ve bunu herhangi bir harf,sayı veya altçizgi(_) karakteri kombinasyonu takip etmelidir.

endofcompile_routines

Bu veri yapısı, adından da anlaşıldığı üzere, simülasyonun sonunda çağırılacak fonksiyonu bildirmek üzere kullanılır. Mevcut örneğin buna ihtiyacı yoktur. Yine de aşağıdaki gibi bir varsayılan tanımlama olmak zorundadır.

```
int (*endofcompile_routines[])() ={0};
```

err_intercept

Bu fonksiyonu, doğru(true) yada yanlış(false)(bir bool tipi) döndürür, kodun hata tesbiti mekanizmasını geliştirmek için kullanılır ve küçük uygulamalar için gözardı edilebilir. Tipik bir örnek aşağıda gösterildiği gibi olabilir.

bool err_intercept(level, facility,code)

int level; char * facility; char *code;

{ return (true); }

PLI Örneği

Eğitselin geri kalanında olduğu gibi, sayaç(counter) örneğimizi test vektör üretimi, monitor(gözleyici),checker(kontrolör):herşey C kodunda oluşturulmuş olacak şekilde, doğrulayalım. Birçok PLI fonksiyonu kendi kullanımlarını göstermek için kullanılmıştır.

Testbenç bu bileşenlere sahip olmalıdır:

- C'de yazılmış saat üretimi(clock generation)
- Saat üreteci(clock generator) HDL sarıcı(wrapper)
- Test üretimi(Test generation) C'de yazılmış
- Test üreteci(Test generator) HDL sarıcı(wrapper)
- Gözleyici(Monitor) ve Kontrolör(Checker) C'de yazılmış
- Gözleyici(Monitor) Kontrolör(Checker) HDL Sarıcı(Wrapper)
- DUT/Monitor/Clock/Test Üretimi(Generation) Verilog'da örneği(instance)

C fonksiyonlarının çağıracağı HDL sarıcı(wrapper) yazmak iyi bir fikirdir.

Saat Üreteci(Clock Generator)

Normalde saat üretecini PLI'da sahip olmak istemeyiz, onu Verilog'un içine koymak herzaman daha iyidir.

```
 1 #include "acc_user.h"
 2 #include "veriuser.h"
 3
 4 // Saatin açılma(ON) ve kapanma(OFF) zamanını tanımla
 5 #define PERIOD  5
 6
 7 // Veri yapısı
 8 struct clkData {
 9   int clk;
10   int clkCnt;
11 };
12
13 // Saati toggle(çögünen)ana rutin
14 void clkGen () {
15   // Kaydedilmiş workarea(çalışmaalanını) al
16   struct clkData *data = ( struct clkData *
)tf_igetworkarea(tf_getinstance());
```

```
 17   if (data->clkCnt == PERIOD) {
 18     data->clk = (data->clk == 0) ? 1 : 0;
 19     data->clkCnt = 0;
 20     //io_printf("%d Current clk = %d\n",tf_gettime(), data->clk);
 21   } else {
 22     data->clkCnt ++;
 23   }
 24   // saat sinyalini HDL'de sür
 25   tf_putp (1, data->clk);
 26 }
 27
 28 // checktf() rutini
 29 // Bu fonksiyon objelere ilk değerlerini verir ve ayrıca workarea 'daki
objeleri kaydeder
 30 void clkInit() {
 31   struct clkData *data = ( struct clkData * )malloc( sizeof( struct
clkData ) );
 32   data->clkCnt = 0;
 33   data->clk    = 0;
 34   tf_setworkarea(data);
 35 }
 36
 37 // misctf() rutini
 38 // Bu rutin 1 saat darbesinden(tick) sonra çağırılır
 39 void clkReactive (int data, int reason, int paramvc) {
 40   // eğer callback nedeni reaktif ise clkGen fonksiyonunu çağır
 41   if (reason == reason_reactivate) {
 42     clkGen();
 43   }
 44   // callback  gecikmesini 1 tick olarak ata
 45   tf_setdelay(1);
 46 }
```

You could download file clkGen.c here

Saat Üreteci HDL Sarıcı(Clock Generator HDL Wrapper)

```
1 module clkGen(clk);
2 output clk;
3 reg clk;
4
5 initial $clkGen(clk);
6
7 endmodule
```

You could download file clkGen.v here

Sayaç Gözlemleyici(Counter Monitor)

```
  1 #include "acc_user.h"
  2 #include "veriuser.h"
  3 #include <malloc.h>
  4 #include <string.h>
  5
  6 struct myCounter {
  7   handle count;
  8   handle enable;
  9   handle reset;
 10   handle clk;
 11   char     *count_value;
 12   char     *enable_value;
 13   char     *reset_value;
```

```
  14   char     *clk_value;
  15   char   *clk_last_value;
  16   int    checker_count;
  17   int    count_width;
  18   int    error;
  19   int    error_time;
  20 };
  21
  22 // Çoklu-bit vektör'den tamsayıya dönüştürme.
  23 int pliConv (char *in_string, int no_bits, int sim_time) {
  24   int conv = 0;
  25   int i = 0;
  26   int j = 0;
  27   int bin = 0;
  28   for ( i = no_bits-1; i >= 0; i = i - 1) {
  29     if (*(in_string + i) == 49) {
  30       bin = 1;
  31     } else if (*(in_string + i) == 120) {
  32       io_printf ("%d counterMonitor : WARNING : X detected\n",
sim_time);
  33       bin = 0;
  34     } else if (*(in_string + i) == 122) {
  35       io_printf ("%d counterMonitor : WARNING : Z detected\n",
sim_time);
  36       bin = 0;
  37     } else {
  38       bin = 0;
  39     }
  40     conv = conv + (1 << j)*bin;
  41     j ++;
  42   }
  43   return conv;
  44 }
  45
  46 void counterModel (struct myCounter *counter) {
  47   int current_value ;
  48   int time = tf_gettime();
  49   // Bizim modelimiz sadece posedge(pozitif kenarda)kontrol eder
  50   if ((strcmp(counter->clk_value,"1") == 0)
  51       && (strcmp(counter->clk_last_value,"0") == 0))  {
  52     // Mevcut saat değerini dönüştürür
  53     current_value  =
  54       pliConv(counter->count_value,counter->count_width,time);
  55     // Giriş kontrol sinyalinin virgüllü yada bilinmeyen olmasını
kontrol et
  56     if (strcmp(counter->reset_value,"x") == 0) {
  57       io_printf("%d counterMonitor : WARNING : reset is x\n", time);
  58     }
  59     if (strcmp(counter->reset_value,"z") == 0) {
  60       io_printf("%d counterMonitor : WARNING : reset is z\n", time);
  61     }
  62     if (strcmp(counter->enable_value,"x") == 0) {
  63       io_printf("%d counterMonitor : WARNING : enable is x\n", time);
  64     }
  65     if (strcmp(counter->enable_value,"z") == 0) {
  66       io_printf("%d counterMonitor : WARNING : enable is z\n", time);
  67     }
  68     // Monitor sayacını arttır ve sadece etkin(enable) 1 ise ve
  69     // sıfırlama(reset) aktif değilse karşılaştır
  70     if (strcmp(counter->enable_value,"1") == 0
  71         && strcmp(counter->reset_value,"0") == 0) {
```

```
  72       if (counter->checker_count  ! = current_value) {
  73         io_printf
  74           ("%d counterMonitor : ERROR : Current value of monitor is %d
dut is %d\n",
  75           time, counter->checker_count, current_value);
  76         counter->error ++;
  77         if (counter->error == 1) counter->error_time = time;
  78       } else {
  79         io_printf
  80           ("%d counterMonitor : INFO  : Current value of monitor is %d
dut is %d\n",
  81           time, counter->checker_count, current_value);
  82       }
  83       counter->checker_count =
  84         (counter->checker_count == 15) ? 0 : counter->checker_count +
1;
  85     // Eğer reset aktifse gözlemleyiciyi sıfırla(reset)
  86     } else if (strcmp(counter->reset_value,"1") == 0) {
  87       io_printf("%d counterMonitor : INFO  : Reset is asserted\n",
time);
  88       counter->checker_count = 0;
  89     }
  90   }
  91   // Saat durumunu güncelle(update)
  92   strcpy(counter->clk_last_value,counter->clk_value);
  93 }
  94
  95 // misctf
  96 void counterMonitor(int data, int reason, int paramvc) {
  97   struct myCounter *counter =
  98     (struct myCounter *) tf_igetworkarea(tf_getinstance());
  99   if ((reason == reason_paramvc) || (reason == reason_paramdrc)) {
 100     tf_synchronize( );
 101   } else if (reason == reason_synch) {
 102     counter->clk    = acc_handle_tfarg(1);
 103     counter->reset  = acc_handle_tfarg(2);
 104     counter->enable = acc_handle_tfarg(3);
 105     counter->count  = acc_handle_tfarg(4);
 106     // Get the values
 107     counter->clk_value    = acc_fetch_value(counter->clk, "%b", 0);
 108     counter->reset_value  = acc_fetch_value(counter->reset, "%b", 0);
 109     counter->enable_value = acc_fetch_value(counter->enable, "%b", 0);
 110     counter->count_value  = acc_fetch_value(counter->count, "%b", 0);
 111     counter->count_width  = acc_fetch_size (counter->count);
 112     // Call the counter model
 113     counterModel (counter);
 114   }
 115   // $finish çağırıldığında simülasyon statüsünü yazdır
 116   if (reason == reason_finish) {
 117     io_printf("========================================\n");
 118     if (counter->error  ! = 0) {
 119       io_printf (" Simulation : FAILED\n");
 120       io_printf ("   Mismatched %d\n",counter->error);
 121       io_printf ("   First Mismatch at time %d\n", counter-
>error_time);
 122     } else {
 123       io_printf (" Simulation : PASSED\n");
 124     }
 125     io_printf("========================================\n");
 126   }
 127 }
```

```
128
129 // calltf()
130 void initCounter(int data, int reason) {
131   struct myCounter       *counter;
132   // modelin tek bir önekleminin
133   // tüm değişkenler için bellek ayır.
134   counter = (struct myCounter *) malloc (sizeof(struct myCounter));
135   // model örneklemi için ilk değer ata(Initialize=.
136   counter->clk    = acc_handle_tfarg(1);
137   counter->reset  = acc_handle_tfarg(2);
138   counter->enable = acc_handle_tfarg(3);
139   counter->count  = acc_handle_tfarg(4);
140   // Mevcut clk değerinin bir kopyasını sakla.
141   counter->clk_last_value  =  acc_fetch_value(counter->clk, "%b", 0);
142   // `$counter_monitor` 'ün herhangi bir argümanı değiştiğinde
143   // `counter_monitor`'ün callback 'ini etkinleştir.
144   tf_asynchon();
145   // Başlangıç(initial) sayaç değerine 0 ata.
146   counter->checker_count = 0;
147   counter->error = 0;
148   counter-> error_time = 0;
149   // `$counterMonitor` 'ün örneklemin model verisini kaydet.
150   tf_setworkarea((char *)counter);
151 }
```

You could download file counterMonitor.c here

Sayaç Gözlemleyici HDL Sarıcı (Counter Monitor HDL Wrapper)

```
1 module counterMonitor (clk, reset, enable, count);
2 input clk, reset, enable;
3 input [3:0] count;
4
5 wire clk, reset, enable;
6 wire [3:0] count;
7
8 initial $counterMonitor(clk,reset,enable,count);
9
10 endmodule
```

You could download file counterMonitor.v here

Sataç TestÜreteci(Counter TestGen)

Test dosyası için sözdizimi

gecikme: Komut = Değer
(delay : Command = Value)

Gecikme (Delay) : saat tick 'i olarak

Komut(Command) : reset(sıfırlama) veya enable(etkin)

Değer(Value) : 0 veya 1

```
  1 #include "acc_user.h"
  2 #include "veriuser.h"
  3 #include "string.h"
  4 #include "stdio.h"
  5
  6 #define IDLE      0
  7 #define INCR      1
  8 #define WAIT      2
  9 #define DRIVE     3
 10 #define DONE      4
 11
 12 struct testGenObject {
 13   char* testFile;
 14   int   debug;
 15   char  cmdArray[100] [100];
 16   int   cmdSize;
 17   int   CmdPointer;
 18   char* command;
 19   int   wait;
 20   int   value;
 21   int   clkCnt;
 22   int   state;
 23   handle count;
 24   handle enable;
 25   handle reset;
 26   handle clk;
 27   char* clk_value;
 28   char  *clk_last_value;
 29 };
 30
 31 static struct testGenObject *object;
 32
 33 // Sayacın arttırımı(Increment counter)
 34 void waitTicks () {
 35   object->clkCnt = object->clkCnt + 1;
 36 }
 37
 38 // Bu fonksiyon test dosyasının içeriğini
 39 // obje komut dizisine yükler
 40 void loadTest() {
 41   FILE *testFile;
 42   char currentLine [100];
 43   object->cmdSize = 0;
 44   if((testFile = fopen(object->testFile, "r")) == NULL) {
 45      printf("Error Opening File.\n");
 46   }
 47   while (fgets(currentLine, sizeof(currentLine), testFile)  ! = NULL )
{
 48     // cmdArray satırını kaydeder
 49     strcpy(object->cmdArray[object->cmdSize], currentLine);
 50     // satır numarasını ve veriyi yazdırır
 51     if (object->debug)
 52       printf("Line %d: %s\n", object->cmdSize,
 53           object->cmdArray[object->cmdSize]);
 54     // Test dosyasındaki herbir satırı alır file
 55     object->cmdSize ++;
 56   }
 57   // Test dosyasını kapatır
 58   fclose(testFile);
 59 }
 60
```

```
  61 // Bu fonksiyon komut satırı seçeneklerini işler
  62 void processCmdOptions () {
  63   // Hata ayıklama(debug) seçeneklerini alır
  64   if (mc_scan_plusargs("plidebug")  ! = NULL) {
  65      object->debug = 1;
  66   } else {
  67      object->debug = 0;
  68   }
  69   // Test dosyasının ismini alır
  70   if (mc_scan_plusargs("test=") == NULL) {
  71     printf("ERROR : No test file option passed, use
+test=testfile\n");
  72   } else {
  73     object->testFile =  mc_scan_plusargs("test=");
  74     if (object->debug) printf("Test file name %s\n",object->testFile);
  75   }
  76 }
  77
  78 void doTest() {
  79   char* ptoks;
  80   char* tcmd;
  81   s_setval_delay delay_s;
  82   s_setval_value value_s;
  83   // Mevcut saat değerini alır
  84   object->clk_value =   acc_fetch_value(object->clk, "%b", 0);
  85   // BFM sadece saatin yükselen kenarında(rising edge)sürülür
  86   if ( ! strcmp(object->clk_last_value,"1") &&  ! strcmp(object-
>clk_value,"0")) {
  87     switch (object->state) {
  88       case IDLE  :
  89         if (object->debug) printf("%d Current state is IDLE\n",
tf_gettime());
  90         if (object->CmdPointer < object->cmdSize) {
  91           tcmd = object->cmdArray[object->CmdPointer];
  92           if (object->debug) printf("Test line %d current command-%s",
  93               object->CmdPointer, tcmd);
  94           ptoks = strtok(tcmd, ":");
  95           int lcnt = 0;
  96           while(ptoks  ! = NULL) {
  97             if (*ptoks  ! = '=') {
  98               if (lcnt == 0) {
  99                 object->wait = atoi(ptoks);
 100                 if (object->debug) printf("Wait    : %d\n", object-
>wait);
 101               } else if (lcnt == 1) {
 102                 object->command = ptoks;
 103                 if (object->debug) printf("Command : %s\n", ptoks);
 104               } else {
 105                 object->value = atoi(ptoks);
 106                 if (object->debug) printf("Value   : %d\n", object-
>value);
 107               }
 108               lcnt ++;
 109             }
 110             ptoks = strtok(NULL, " ");
 111           }
 112           object->CmdPointer ++ ;
 113           if (object->wait == 0) {
 114             if (object->debug) printf("%d Next State DRIVE\n",
tf_gettime());
 115             object->state = DRIVE;
```

```
 116             doTest();
 117           } else {
 118             if (object->debug) printf("%d Next State WAIT\n",
tf_gettime());
 119             object->state = WAIT;
 120           }
 121         } else {
 122           if (object->debug) printf("%d Next State DONE\n",
tf_gettime());
 123           object->state = DONE;
 124         }
 125         break;
 126       case WAIT  :
 127           if (object->debug) printf("%d Current state is WAIT : %d\n",
 128               tf_gettime(), object->clkCnt);
 129           if ((object->clkCnt + 1) >= object->wait) {
 130             object->wait = 0;
 131             object->clkCnt = 0;
 132             if (object->debug) printf("%d Next State DRIVE\n",
tf_gettime());
 133             object->state = DRIVE;
 134             doTest();
 135           } else {
 136             waitTicks();
 137           }
 138           break;
 139       case DRIVE :
 140          if (object->debug) printf("%d Current state is DRIVE\n",
tf_gettime());
 141          value_s.format    = accIntVal;
 142          delay_s.model     = accNoDelay;
 143          delay_s.time.type = accTime;
 144          delay_s.time.low  = 0;
 145          delay_s.time.high = 0;
 146          if ( ! strcmp(object->command,"reset")) {
 147            value_s.value.integer =  object->value;
 148            acc_set_value(object->reset,&value_s,&delay_s);
 149          } else if ( ! strcmp(object->command,"enable")) {
 150            value_s.value.integer =  object->value;
 151            acc_set_value(object->enable,&value_s,&delay_s);
 152          } else {
 153            if (object->debug) printf("ERROR : What command do you
want\n");
 154          }
 155          if (object->debug) printf("%d Next State IDLE\n",
tf_gettime());
 156          object->state = IDLE;
 157          break;
 158       case DONE  :
 159          if (object->debug) printf("%d Current state is DONE\n",
tf_gettime());
 160          tf_dofinish();
 161          break;
 162       default    : object->state = IDLE;
 163                    break;
 164     }
 165   }
 166   object->clk_last_value =   acc_fetch_value(object->clk, "%b", 0);
 167 }
 168
 169 void initCounterTestGen () {
```

```
 170   //acc_initialize( );
 171   //acc_configure( accDisplayErrors, "false" );
 172   object = (struct testGenObject *) malloc (sizeof(struct
testGenObject));
 173   // ilk ve varsayılan değerleri yükler
 174   object->testFile = "simple.tst";
 175   object->cmdSize = 0;
 176   object->CmdPointer = 0;
 177   object->clkCnt = 0;
 178   object->state = IDLE;
 179   // Modelin bu örneklemini ilkdeğer ata(initialize).
 180   object->clk    = acc_handle_tfarg(1);
 181   object->reset  = acc_handle_tfarg(2);
 182   object->enable = acc_handle_tfarg(3);
 183   object->count  = acc_handle_tfarg(4);
 184   // Tüm girişlerde aktif olmayan sinyalleri sürer
 185   tf_putp (2, 0);
 186   tf_putp (3, 0);
 187   // Mevcut clk değerinin bir kopyasını kaydeder.
 188   object->clk_last_value =   acc_fetch_value(object->clk, "%b", 0);
 189   // Komut satırından testdosyası ismini ve hata ayıklama
seçeneklerini alır
 190   processCmdOptions();
 191   // Test dosyasını açar ve komut dizisini yapar
 192   loadTest(object);
 193   // saat değiştiğinde callback ekler
 194   acc_vcl_add( object->clk, doTest, object->clk_value,
vcl_verilog_logic );
 195   // Tüm acc rutinleri buna sahip olmalı
 196   acc_close();
 197 }
```

You could download file counterTestGen.c here

Sayaç TestÜreteci HDL Sarıcı(Counter TestGen HDL Wrapper)

```
 1 module counterTestGen (clk, reset, enable, count);
 2 input clk;
 3 output reset, enable;
 4 input [3:0] count;
 5
 6 wire clk;
 7 reg reset, enable;
 8 wire [3:0] count;
 9
10 initial $counterTestGen(clk,reset,enable,count);
11
12 endmodule
```

You could download file counterTestGen.v here

HDL TestBenç Top

```
 1 module top();
 2 wire clk;
 3 wire [3:0] count;
 4 wire enable;
 5 wire reset;
 6
 7 // clk(saat) üreteci ile bağlantı kur
 8 clkGen clkGen(.clk (clk));
 9
```

```
10 // DUT ile bağlantı kur
11 counter dut(
12 .clk    (clk),
13 .reset  (reset),
14 .enable (enable),
15 .count  (count)
16 );
17
18 // Monitor/Checker 'a bağlan
19 counterMonitor monitor(
20 .clk    (clk),
21 .reset  (reset),
22 .enable (enable),
23 .count  (count)
24 );
25
26 // Test Generator'ına bağlan
27 counterTestGen test(
28 .clk    (clk),
29 .reset  (reset),
30 .enable (enable),
31 .count  (count)
32 );
33
34 endmodule
```

You could download file top.v here

Örnek: Test File(Test Dosyası)

```
1 : reset = 1
10 : reset = 0
5 : enable = 1
20 : enable = 0
```

Örnek Çıktı

```
5 counterMonitor : WARNING : X detected
5 counterMonitor : WARNING : X detected
5 counterMonitor : WARNING : X detected
5 counterMonitor : WARNING : X detected
17 counterMonitor : WARNING : X detected
17 counterMonitor : WARNING : X detected
17 counterMonitor : WARNING : X detected
17 counterMonitor : WARNING : X detected
29 counterMonitor : WARNING : X detected
29 counterMonitor : WARNING : X detected
29 counterMonitor : WARNING : X detected
29 counterMonitor : WARNING : X detected
29 counterMonitor : INFO  : Reset is asserted
41 counterMonitor : INFO  : Reset is asserted
53 counterMonitor : INFO  : Reset is asserted
65 counterMonitor : INFO  : Reset is asserted
77 counterMonitor : INFO  : Reset is asserted
89 counterMonitor : INFO  : Reset is asserted
101 counterMonitor : INFO  : Reset is asserted
113 counterMonitor : INFO  : Reset is asserted
125 counterMonitor : INFO  : Reset is asserted
137 counterMonitor : INFO  : Reset is asserted
149 counterMonitor : INFO  : Reset is asserted
233 counterMonitor : INFO  : Current value of monitor is 0 dut is 0
```

```
245 counterMonitor : INFO  : Current value of monitor is 1 dut is 1
257 counterMonitor : INFO  : Current value of monitor is 2 dut is 2
269 counterMonitor : INFO  : Current value of monitor is 3 dut is 3
281 counterMonitor : INFO  : Current value of monitor is 4 dut is 4
293 counterMonitor : INFO  : Current value of monitor is 5 dut is 5
305 counterMonitor : INFO  : Current value of monitor is 6 dut is 6
317 counterMonitor : INFO  : Current value of monitor is 7 dut is 7
329 counterMonitor : INFO  : Current value of monitor is 8 dut is 8
341 counterMonitor : INFO  : Current value of monitor is 9 dut is 9
353 counterMonitor : INFO  : Current value of monitor is 10 dut is 10
365 counterMonitor : INFO  : Current value of monitor is 11 dut is 11
377 counterMonitor : INFO  : Current value of monitor is 12 dut is 12
389 counterMonitor : INFO  : Current value of monitor is 13 dut is 13
401 counterMonitor : INFO  : Current value of monitor is 14 dut is 14
413 counterMonitor : INFO  : Current value of monitor is 15 dut is 15
425 counterMonitor : INFO  : Current value of monitor is 0 dut is 0
437 counterMonitor : INFO  : Current value of monitor is 1 dut is 1
449 counterMonitor : INFO  : Current value of monitor is 2 dut is 2
461 counterMonitor : INFO  : Current value of monitor is 3 dut is 3
473 counterMonitor : INFO  : Current value of monitor is 4 dut is 4
=======================================
 Simulation : PASSED
=======================================
```

Verilog Prosedürel Arayüz(Verilog Procedural Interface (VPI))

Verilog Prosedürel Arayüzü(VPI), Verilog Donanım Tanımlama Dili için bir C-programlama arayüzüdür.VPI Verilog HDL için üçüncü nesil prosedürel arayüzdür. VPI, eksiksiz Verilog HDL'e tutarlı, nesneye-dayalı erişim sağlar.

VPI erişim kümesi ve standart C programlama dili fonksiyonlarından çağırdığınız yardımcı rutinlerle tutarlıdır. Bu rutinler Verilog HDL tasarımınızda bulunan örneklenmiş simülasyon objeleriyle etkileşim içerisindedir.

VPI ,Verilog PLI 1.0 ve PLI 2.0 ile aynıdır, bu nedenler eğer birşeyler PLI 2.0 'da çalışıyorsa kolayca VPI 'a taşınabilir. VPI daha önceki PLI 2.0'dan daha temizdir. Tüm fonksiyon ve görevler vpi_* ile başlar. Örneğin vpi_printf().

Adımlar : VPI Kullanarak Uygulama Yazmak(Writing Application Using VPI)

Aşağıdaki adımları bir C uygulaması yazmak ve bunu Verilog simülatörü ile arayüzlendirmek için gerçekleştirmek gerekir.

- Bir C fonksiyonu yazmak(Writing a C function)
- C Fonksiyonunu Yeni Sistem Görevi ile İlişkilendirmek(Associating C Functions with a New System Task)
- Yeni Sistem Görevlerini Kayda Geçirmek (Registering New System Tasks)
- Sistem Görevlerini Çağırmak (Invoking System Tasks)

Bir C fonksiyonu yazmak (Write a C function)

C fonksiyonu/rutini yazmak PLI 2.0 ile aynıdır; tek fark acc_user.h ve veriuser.h yerine vpi_user.h eklememiz gerekmektedir. Ayrıca Verilog simülatörüne erişim ve değiştirme için vpi_* fonksiyonlarını kullanırız.

Aşağıda basit bir C fonksiyon kodu örneği bulunmaktadır:

```
1 #include "vpi_user.h"
2
3 void hello() {
4   vpi_printf("\n\nHello Deepak\n\n");
5 }
```

You could download file hello_vpi.c here

C Fonksiyonunu Yeni Sistem Görevi ile İlişkilendirmek (Associating C Functions with a New System Task)

C fonksiyonunuzu bir sistem görevi(task) ile ilişkikendirmek için, s_vpi_systf_data tipinde bir veri tipi ve bu yapı için bir işaretçi(pointer) oluşturun. vpi_systf_data veri tipi vpi_user.h ekleme dosyasının içinde tanımlanmıştır.Aşağıda s_vpi_systf_data veri yapısı vardır.

```
1 typedef struct t_vpi_systf_data {
2   PLI_INT32 type;     // vpiSysTask, vpiSysFunc
3   PLI_INT32 sysfunctype; // vpiSysTask, vpi[Int,Real,Time,Sized,
SizedSigned]Func
4   PLI_BYTE8 *tfname;  // İlk karakter `$' olmalı
5   PLI_INT32 (*calltf)(PLI_BYTE8 *);
6   PLI_INT32 (*compiletf)(PLI_BYTE8 *);
7   PLI_INT32 (*sizetf)(PLI_BYTE8 *); // Sadece boyutlandırılmış
fonksiyonal geri çağırılabilir(callback)
8   PLI_BYTE8 *user_data;
9 } s_vpi_systf_data, *p_vpi_systf_data;
```

You could download file s_vpi_systf_data.h here

Alan Adı	Alan Açıklaması
type	task – değer döndürmez; function- değer döndürebilir.
sysfunctype	Eğer tip bir fonksiyonsa, sysfunctype, calltf fonksiyonunun döndüreceği değerin tipini belirtir.
tfname	Tırnak arasında belirtilmiş dizgi sistem görev veya fonksiyonun adını tanımlar. İlk karakter $ olmalı.
calltf	Bu alan sizin uygulama rutinini için bir işaretçidir(pointer).
compiletf	Bu alan simülatörün her seferinde görev ve fonksiyon örneklemini derlerken çağırdığı rutin için işaretçidir(pointer). Eğer böyle bir rutin yoksa NULL girin.
sizetf	Sistem görev ve fonksiyonunun döndürdüğü değerin boyutlarını bit olarak dönderen rutin için bir işaretçidir(pointer).
user_data	Bu alan opsiyonel veri için bir işaretçidir. Bu veriye vpi_get_systf_info() rutinini çağırarak erişebilirsiniz.

Hello rutini için örnek

```
 1 #include "hello_vpi.c"
 2
 3 // C fonksiyonlarının yeni bir sistem göreviyle ilişkilendirilmesi
 4 void registerHelloSystfs() {
 5   s_vpi_systf_data task_data_s;
 6   p_vpi_systf_data task_data_p = &task_data_s;
 7   task_data_p->type = vpiSysTask;
 8   task_data_p->tfname = "$hello";
 9   task_data_p->calltf = hello;
10   task_data_p->compiletf = 0;
11
12   vpi_register_systf(task_data_p);
13 }
14
15 // Yeni sistem görevini burada kayda geçirin(Register)
16 void (*vlog_startup_routines[ ] ) () = {
17    registerHelloSystfs,
18    0  // son varlık-giriş 0 olmalı
19 };
```

You could download file hello_vpi_modelsim.c here

Yeni Sistem Görevinin Kayda Geçirilmesi (Registering New System Tasks)

s_vpi_systf_data veri yapısını ilklendirdikten sonra, yeni sistem görevinizi kayıt altına almalısınız, böylece simülatör onu yürütebilecektir. Daha önce belirttiğim hello rutini örneğinde, 14 ile 17 satırlar arasında bu yapılmaktadır.

Sistem Görevlerininin Çağrılması (Invoking System Tasks)

Yeni sistem görevlerinizi başlangıç(initial) bloğunuzda veya herzaman(always) bloğunuzda aşağıda gösterildiği gibi çağırabilirsiniz.

```
1 module hello_pli ();
2
3 initial begin
4   $hello;
5    #10  $finish;
6 end
7
8 endmodule
```

You could download file hello_pli.v here

Simülatör ile Bağlama(Linking with Simulator)

PLI 1.0 ve PLI 2.0 ‘da olduğu gibi, simülatörün, VPI rutinlerini simülatöre bağlayan kendi yolları vardır.Bu simülatörleri örnekler olarak göreceğiz:

- VCS
- Modelsim
- NCSim

VCS

VCS simülatörü ile bir tab(atlama) dosyası yaratmalısınız. Örneğin aşağıdaki gibi görünebilir tab(atlama) dosyanız:

$hello call=hello

Burada $hello Verilog kodunun kullanacağı kullanıcı tanımlı fonksiyonun ismidir; call=hello ise bir C fonksiyonudur, $hello Verilogda çağırıldığında çağırılacaktır.

Kodu derlemek için komut satırı seçenekleri :

vcs -R -P hello.tab hello_vpi.c hello_pli.v +vpi -CFLAGS "-I$VCS_HOME/`vcs -platform`/lib"

VPI ve PLI komut satırı arasındaki tek fark +vpi seçeneğidir.

Modelsim

VCS ‘de olduğu gibi modelsim simülatörünün de PLI ile haberleşme için kendi yolları vardır. Verilog tarafından kullanılan tüm kullanıcı tanımlı fonksiyonlar ve bunlara uygun çağırılan C fonksiyonlarının fonksiyon listesini oluşturmamız gerekmektedir. VCS’den farklı olarak bunu aşağıda belirtildiği gibi bir C dosyasının içinde yapmamız gerekmektedir. Bunu daha önce C fonksiyonu ile yeni bir sistem görevi ilişkilendirilmesinde**"Associating C Functions with a New System Task"** belirtmiştik.

```
 1 #include "hello_vpi.c"
 2
 3 // C fonksiyonu ile yeni bir sistem görevi ilişkilendirmesi
 4 void registerHelloSystfs() {
 5   s_vpi_systf_data task_data_s;
 6   p_vpi_systf_data task_data_p = &task_data_s;
 7   task_data_p->type = vpiSysTask;
 8   task_data_p->tfname = "$hello";
 9   task_data_p->calltf = hello;
10   task_data_p->compiletf = 0;
11
12   vpi_register_systf(task_data_p);
13 }
14
15 // yeni sistem görevi burada kayda geçirilir
16 void (*vlog_startup_routines[ ] ) () = {
17    registerHelloSystfs,
18    0  // last entry must be 0
19 };
```

You could download file hello_vpi_modelsim.c here

vlib work

vlog hello_pli.v

gcc -c -g -I$MODEL/include hello_vpi_modelsim.c

ld -shared -E -o hello_vpi.sl hello_vpi_modelsim.o

vsim -c hello_pli -pli hello_vpi.sl

Simülasyonu başlatmak için vsim komut satırında "run -all" yazın.

Detaylı bilgi için modelsim kullanım klavuzuna bakın.

NCSim

Ncsim için derleme ve bağlama modelsime göre birmiktar daha kapalıdır. Modelsimden farklı olarak NCSim birçok bağlama yolunu destekler. Aşağıda gerçekleştrilimesi gereken adımların listesi vardır.

gcc -c -g -I$CDS_INST_DIR/tools/include hello_vpi_modelsim.c

ld -shared -E -o libvpi.so hello_vpi_modelsim.o

ncverilog hello_pli.v

VPI 'da bağlama için detaylı bilgi için NCSim veya NCVerilog kullanım klavuzuna bakınız.

VPI Rutinleri

Rutin	Açıklama
vpi_chk_error()	Bir önceki VPI çağrısı bir hataya neden olmuşmu bakar ve hata hakkında bilgiyi alıp getirir.
vpi_compare_objects()	İki obje tanıtıcısını(handle) karşılaştırı ve aynı objeyle ilgili olup olmadığına karar verir.
vpi_control()	Kullanıcı kodundan simülatöre pigi gönderir.
vpi_flush()	Simülatör çıkış kanalı ve log dosyası çıkış arabelleğindeki veriyi boşaltır.
vpi_free_object()	VPI objeleri için ayrılmış belleği serbest bırakır.
vpi_get()	Bir objenin bool özelliğini veya tamsayı değerine erişir.
vpi_get_cb_info()	Simülasyonla ilişkili geriçağrılarla(callback) ilgili bilgiye erişir.
vpi_get_data()	Bir gerçekleştirmenin kaydetme/yeniden başlatma bölgesinden bilgiyi alır.

vpi_get_delays()	Bir objenin gecikmeleri veya zamanlama limitlerine erişir.
vpi_get_str()	Bir objenin dizgi özelliği değerine erişir.
vpi_get_systf_info()	Kullanıcı tanımlı görev veya fonksiyon hakkındaki bilgiye erişir.
vpi_get_time()	Objenin zamanaralığını kullanarak mevcut simülasyon zamanına erişir.
vpi_get_userdata()	Bir gerçeklemenin sistem görev/fonksiyonlarının kaydedilme bölgesindan kullanıcı veri değerine erişir.
vpi_get_value()	Bir objenin simülasyon değerine erişir.
vpi_handle()	Bir objeyle birebir ilişkisi olan objenin tanıtıcısını döndürür.
vpi_handle_by_index()	Bir objenin tanıtıcısını,bir üst objenin içindeki indis sayısını kullanrak, döndürür.
vpi_handle_by_name()	Bir objenin tanıtıcısını özel ismiyle döndürür.
vpi_handle_multi()	Çoktan-bire ilişkisi olan objenin tanıtıcısını döndürür.
vpi_iterate()	Birden-çoğa ilişkideki objeye erişmek için kullanılan iteratör(iterator) tanıtıcısını döndürür.
vpi_mcd_open()	Bir dosyayı yazma için açar.
vpi_mcd_close()	vpi_mcd_open() rutini ile açılmış bir veya birden çok dosyayı kapatır.
vpi_mcd_flush()	Verilen MCD çıkış arabelleğindeki veriyi boşaltmak için bu fonksiyon kullanılır.
vpi_mcd_name()	Bir kanal açıklayıcısı olarak belirtilmiş dosyanın ismini döndürür.
vpi_mcd_printf()	vpi_mcd_open() ile açılmış bir veya birden çok dosyaya yazar.
vpi_printf()	VPI uygulamasından ve simülatör log dosyasından çağırılmış simülatörün çıkış kanalına yazar.
vpi_put_data()	Bir gerçeklemenin kaydetme/yeniden başlatma bölgesine veri koyar.
vpi_put_delays()	Bir objenin gecikme veya zamanlama limitlerini kurar.
vpi_put_userdata()	Bir gerçeklemenin sistem görev/fonksiyon kaydetme bölgesine kullanıcı veri değeri koyar.
vpi_put_value()	Bir objeye bir değer verir.
vpi_register_cb()	Simülasyon ilişkili geriçağrılarını(callback) kayda geçirir.
vpi_register_systf()	Kullanıcı tanımlı sistem görev veya fonksiyon geri çağrılarını kayda geçirir.
vpi_remove_cb()	vpi_register_cb() ile kayda geçirilmiş bir simülasyon geri çağrısını siler.
vpi_scan()	Verilog HDL hiyerarşisini birden-çoğa ilişkideki objeler için tarar.
vpi_vprintf()	VPI uygulamalarında çağırılmış ve simülatör log dosyasında varargs kullanılarak daha önce başlatılmı simülatörün çıkış kanalına yazar.

Verilog 2001'da Yeni Ne Var

Giriş

Verilog 2001 deki değişikliklerin birçoğu diğer dillerden alınmıştır, generate(üretmek), configuration(konfigürasyon), file operation(dosya işlemleri) VHDL'dendir. Ben sadece en sık kullanılan Verilog 2001'in değişikliklerini bir liste olarak ekledim. Örnekleri test edebilmek için Verilog 2001 desteği olan bir simülatöre ihtiyacınız vardır.

Virgül hassasiyet listesinde kullanılır (sensitive list)

Verilog'un önceki versiyonunda, birden çok hassasiyet listesi elemanı için 'or' kullanırdık. Ancak Verilog 2001 de, virgülü aşağıdaki örnekteki gibi kullanabiliriz.

```
 1 module comma_example();
 2
 3 reg a, b, c, d, e;
 4 reg [2:0] sum, sum95;
 5
 6 // virgül kullanımı için Verilog 2k örneği
 7 always @ (a, b, c, d, e)
 8 begin : SUM_V2K
 9   sum = a + b + c + d + e;
10 end
11
12 // yukarıdaki kod için Verilog 95 örneği
13 always @ (a or b or c or d or e)
14 begin : SUM_V95
15   sum95 = a + b + c + d + e;
16 end
17
18 initial begin
19   $monitor ("%g a=%b b=%b c=%b d=%b e=%b sum=%b sum95=%b",
20     $time, a, b, c, d, e, sum, sum95);
21    #1  a = 1;
22    #1  b = 1;
23    #1  c = 1;
24    #1  d = 1;
25    #1  e = 1;
26    #1  $finish;
27 end
28
29 endmodule
```

You could download file comma_example.v here

Aynısını kenar hassasiyetli kod için de kullanabiliriz, kullanımı aşağıdaki kodda gösterilmiştir.

```
 1 module comma_edge_example();
 2
 3 reg clk, reset, d;
 4 reg q, q95;
 5
 6 // virgül kullanımı için Verilog 2k örneği
 7 always @ (posedge clk, posedge reset)
```

```
 8 begin : V2K
 9   if (reset) q <= 0;
10   else q <= d;
11 end
12
13 // yukarıdaki kod için Verilog 95 örneği
14 always @ (posedge clk or posedge reset)
15 begin : V95
16   if (reset) q95 <= 0;
17   else q95 <= d;
18 end
19
20 initial begin
21   $monitor ("%g clk=%b reset=%b d=%b q=%b q95=%b",
22     $time, clk, reset, d, q, q95);
23   clk = 0;
24   reset = 0;
25   d = 0;
26    #4  reset = 1;
27    #4  reset = 0;
28    #1  d = 1;
29    #10  $finish;
30 end
31
32 initial  #1  forever clk = #1 ~clk;
33
34 endmodule
```

You could download file comma_edge_example.v here

Kombinasyonel lojik hassasiyet listesi(Combinational logic sensitive list)

Verilog 2001 RHS deki tüm değişkenleri listelemek için yıldız kullanımına izin vermektedir. Bu tip hatalarını yoketmekte ve böylece simülasyon ve sentez uyumsuzluğunu önlemektedir.

```
 1 module star_example();
 2
 3 reg a, b, c, d, e;
 4 reg [2:0] sum, sum95;
 5
 6 // çoklu lojik için yıldız kullanımı Verilog 2k örneği
 7 always @ (*)
 8 begin : SUM_V2K
 9   sum = a + b + c + d + e;
10 end
11
12 // yukarıdaki kod için Verilog 95 örneği
13 always @ (a or b or c or d or e)
14 begin : SUM_V95
15   sum95 = a + b + c + d + e;
16 end
17
18 initial begin
19   $monitor ("%g a=%b b=%b c=%b d=%b e=%b sum=%b sum95=%b",
20     $time, a, b, c, d, e, sum, sum95);
21    #1  a = 1;
22    #1  b = 1;
23    #1  c = 1;
24    #1  d = 1;
25    #1  e = 1;
```

```
26    #1  $finish;
27 end
28
29 endmodule
```

You could download file star_example.v here

Wire(Tel) Veri tipi

Verilog 1995'de, varsayılan veri tipi net ve bunun genişliği herzaman 1 bit idi. Verilog 2001 de ise genişlik otomatik olarak düzenlenmektedir.

Verilog 2001'de, varsayılan veri tipini `default net_type none ile etkisizleştirebiliriz. Bu temelde bildirilmemiş telleri(wire) yakalamamıza yardımcı olur.

Register(Yazmaç) Veri tipi

Yazmaç(register) veri tipine değişken(variable) denir, bu yeni başlayanlar için birçok kafa karmaşıklığı yaratır. Ayrıca, register(yazmaç)/variable(değişken) veri tipi için ilk değer(initial) belirtmek mümkündür. Reg veri tipi işaretli(signed) olarak da bildirilebilinir.

```
 1 module v2k_reg();
 2
 3 // v2k değişkenlere ilk değer verilmesine izin verir
 4 reg a = 0;
 5 // Burada Sadece son değişkene 0 değeri atanır. Yani d = 0
 6 // Geri kalan b, c 'ye x atanır.
 7 reg b, c, d = 0;
 8 // reg veri tipi v2k'da işaretli olabilir.
 9 // İşaretli sabitler atayabiliriz.
10 reg signed [7:0] data = 8'shF0;
11
12 // Fonksiyon işaretli değer döndürebilir
13 // Bunun portları işaretli portlar içerebilir
14 function signed [7:0] adder;
15   input a_in;
16   input b_in;
17   input c_in;
18   input signed [7:0] data_in;
19   begin
20     adder = a_in + b_in + c_in + data_in;
21   end
22 endfunction
23
24 endmodule
```

You could download file v2k_reg.v here

Yeni operatörler

Verilog 2001 iki yeni operatör belirtilmiştir, bunlar tasarımcıların ilgisini çekmektedir. Bunlar:

- <<< : Sola kaydırma, işaretli veri tipleri için kullanılır
- >>> : Sağa kaydırma, işaretli veri tipleri için kullanılır
- ** : üstel kuvvet operatörü.

İşaretli kaydırma(Signed shift) operatörü

```
 1 module signed_shift();
 2
 3 reg [7:0] unsigned_shift;
 4 reg [7:0] signed_shift;
 5 reg signed [7:0] data = 8'hAA;
 6
 7 initial begin
 8   unsigned_shift = data >> 4;
 9   $display ("unsigned shift = %b", unsigned_shift);
10   signed_shift = data >>> 4;
11   $display ("signed   shift = %b", signed_shift);
12   unsigned_shift = data << 1;
13   $display ("unsigned shift = %b", unsigned_shift);
14   signed_shift = data <<< 1;
15   $display ("signed   shift = %b", signed_shift);
16   $finish;
17 end
18
19 endmodule
```

You could download file signed_shift.v here

Kuvvet(Power) operatörü

```
 1 module power_operator();
 2
 3 reg [3:0] base = 2;
 4 reg [5:0] exponent = 1;
 5 reg [31:0] result = 0;
 6
 7 initial begin
 8   $monitor ("base = %d exponent = %d result = %d", base, exponent,
result);
 9   repeat (10) begin
10      #1  exponent = exponent + 1;
11   end
12    #1  $finish;
13 end
14
15 always @ (*)
16 begin
17   result = base ** exponent;
18 end
19
20 endmodule
```

You could download file power_operator.v here

Port Bildirimi(Declaration)

Verilog 2001, port yönlendirme ve modüldeki port listesindeki veri tipine izin verir, aşağıdaki örnekte gösterildiği gibi.

```
 1 module ansiport_example();
 2
 3 reg read,write = 0;
 4 reg [7:0] data_in  = 0;
 5 reg [3:0] addr = 0;
```

```
  6 wire [7:0] data_v95, data_notype, data_ansi;
  7
  8 initial begin
  9   $monitor (
 10   "%g rd=%b wr=%b addr=%b data_in=%h data_v95=%h data_notype=%h
data_ansi=%h"
 11   , $time, read, write, addr, data_in, data_v95, data_notype,
data_ansi);
 12    #1  read = 0; // why only for read ?
 13    #3  repeat (16) begin
 14     data_in = $random;
 15     write = 1;
 16      #1  addr = addr + 1;
 17   end
 18   write = 0;
 19   addr = 0;
 20    #3  repeat (16) begin
 21     read = 1;
 22      #1  addr = addr + 1;
 23   end
 24   read = 0;
 25    #1  $finish;
 26 end
 27
 28 memory_v95         U (read, write, data_in, addr, data_v95);
 29 memory_ansi_notype W (read, write, data_in, addr, data_notype);
 30 memory_ansi        V (read, write, data_in, addr, data_ansi);
 31
 32 endmodule
 33 // Verilog 95 kodu
 34 module memory_v95 ( read, write, data_in, addr, data_out);
 35 input  read;
 36 input  write;
 37 input  [7:0] data_in;
 38 input  [3:0] addr;
 39 output  [7:0] data_out;
 40
 41 reg [7:0] data_out;
 42 reg [7:0] mem [0:15];
 43
 44 always @ (*)
 45 begin
 46   if (write) mem[addr] = data_in;
 47 end
 48
 49 always @ (read, addr)
 50 begin
 51   if (read) data_out = mem[addr];
 52 end
 53
 54 endmodule
 55
 56 // Verilog 2k port listesinde tip olmadan
 57 module memory_ansi_notype (
 58   input  read,
 59   input  write,
 60   input  [7:0] data_in,
 61   input  [3:0] addr,
 62   output  [7:0] data_out
 63 );
 64 reg [7:0] mem [0:15];
```

```
65
66 always @ (*)
67 begin
68   if (write) mem[addr] = data_in;
69 end
70
71 assign data_out = (read) ? mem[addr] : 0;
72
73 endmodule
74
75 // Verilog 2k genişlik ve veri tipi listelenmiştir type listed
76 module memory_ansi (
77   input wire read,
78   input wire write,
79   input wire [7:0] data_in,
80   input wire [3:0] addr,
81   output reg [7:0] data_out
82 );
83
84 reg [7:0] mem [0:15];
85
86 always @ (*)
87 begin
88   if (write) mem[addr] = data_in;
89 end
90
91 always @ (read, addr)
92 begin
93   if (read) data_out = mem[addr];
94 end
95
96 endmodule
```

You could download file ansiport_example.v here

Generate(Üretme) Blokları

Bu özellik virkaç değişiklik ile VHDL'den alınmıştır. Döngülerdeki birbirinin taklidi çoklu örneklemler için kullanmak mümkündür. Aşağıda Verilog 2001 de generate ifadesinin kullanımına dair bir örnek vardır.

```
 1 module generate_example();
 2
 3 reg read,write = 0;
 4 reg [31:0] data_in  = 0;
 5 reg [3:0] address = 0;
 6 wire [31:0] data_out;
 7
 8 initial begin
 9   $monitor ("%g read=%b write=%b address=%b data_in=%h data_out=%h",
10     $time, read, write, address, data_in, data_out);
11    #1  read = 0; // neden sadece okuma için
12    #3  repeat (16) begin
13     data_in = $random;
14     write = 1;
15      #1  address = address + 1;
16   end
17   write = 0;
18   address = 0;
```

```
19    #3  repeat (16) begin
20     read = 1;
21      #1  address = address + 1;
22   end
23   read = 0;
24    #1  $finish;
25 end
26
27 genvar i;
28
29 generate
30   for (i=0; i < 4; i=i+1) begin : MEM
31      memory U (read, write,
32                data_in[(i*8)+7:(i*8)],
33                address,data_out[(i*8)+7:(i*8)]);
34   end
35 endgenerate
36
37 endmodule
38
39 // Çoklu zamanlarda bağlanmış olunacak bir alt modül
40 module memory (
41   input wire read,
42   input wire write,
43   input wire [7:0] data_in,
44   input wire [3:0] address,
45   output reg [7:0] data_out
46 );
47
48 reg [7:0] mem [0:15];
49
50 always @ (*)
51 begin
52   if (write) mem[address] = data_in;
53 end
54
55 always @ (read, address)
56 begin
57   if (read) data_out = mem[address];
58 end
59
60 endmodule
```

You could download file generate_example.v here

Çok Boyutlu Dizi (Multi-Dimension Array)

Verilog 1995 değişkenlerin tek boyutlu dizilerine izin veriyordu. Verilog 2001 iki boyutludan fazla boyutlu değişken ve net veri tipi dizilerine izin vermektedir. Dizi atamalarında, Verilog 2001 bölüm seçimine ve bölüm seçiminde kullanılacak değişkenlere izin vermektedir.Aşağıdaki örnekte bu gösterilmiştir.

```
1 module multi_array();
2
3 reg read_v95, read_multi, read_bit;
4
5 // Verilog 1995 ve Verilog 2001
6 // 1 boyutlu dizilere izin verir
7 reg [7:0] address;
8 reg [7:0] memory [0:255];
```

```
 9 wire [7:0] data_out;
10
11 assign data_out = (read_v95) ? memory[address] : 0;
12
13 // Verilog 2001 çok boyutlu dizilere izin verir
14 reg [7:0] address1, address2;
15 reg [15:0] array [0:255][0:255];
16 wire [7:0] data_array = (read_multi) ? array[address1][address2] : 0;
17
18 // Verilog 2001 dizilerin içindeki bit ve bölüm seçimine izin
vermektedir
19 wire [7:0] data_bit = (read_bit) ? array[1][200][12:5] : 8'b0;
20
21 // Verilog 2001 indekslenmiş vektör bölümü seçimine izin vermektedir
22 reg  [31:0] double_word;
23 reg  [2:0] byte_no;
24 wire [7:0] pos_offset = double_word[byte_no*8 +:8];
25 wire [7:0] neg_offset = double_word[byte_no*8 -:8];
26
27 initial begin
28   // Dizi içindeki Kontrol biti(Check bit) ve bölüm seçimi
29    #1  array[1][200] = 16'h1234;
30    #1  read_bit  = 1;
31    #1  array[1][200] = 16'hAAAA;
32   // indekslenmiş vektör bölüm seçimlerini kontrol et
33   double_word = 32'hDEAD_BEEF;
34    #1  byte_no = 2;
35    #1  byte_no = 1;
36    #1  $finish;
37 end
38
39 always @ (*)
40   $display (
41    "double_word : %h byte_no : %d pos_offset : %h neg_offset : %h",
42    double_word, byte_no, pos_offset, neg_offset);
43
44 always @ (*)
45   $display ("array[1][200] : %h read_bit : %b data_bit : %h",
46    array[1][200],read_bit, data_bit);
47
48 endmodule
```

You could download file multi_array.v here

 **Yeniden-girişli(Re-entrant) görevler(task) ve özyineli fonksiyonlar(recursive functions)**

Verilog 2001 yeni bir anahtar sözcük: automatic(otomatik) ekler. Bu anahtar sözcük bir göreve eklendiğinde, görec yeniden-girişli(re-entrant) yapar. Automatic ile bildirilmiş tüm görevler herbir koşutzamanlı giriş için dinamik olarak tahsis edilir.

Automatic anahtar sözcüğü ile eklenmiş bir fonksiyona özyineli(recursive) denir.

Örnek : Task(Görev)

```
 1 module re_entrant_task();
 2
 3 task automatic print_value;
 4   input [7:0] value;
 5   input [7:0] delay;
 6   begin
 7     #(delay) $display("%g Passed Value %d Delay %d", $time, value,
delay);
 8   end
 9 endtask
10
11 initial begin
12   fork
13      #1  print_value (10,7);
14      #1  print_value (8,5);
15      #1  print_value (4,2);
16   join
17    #1  $finish;
18 end
19
20 endmodule
```

You could download file re_entrant_task.v here

Örnek: Fonksiyon

```
 1 module recursuve_function();
 2
 3 function automatic [31:0] calFactorial;
 4   input [7:0] number;
 5   begin
 6     if (number == 1) begin
 7       calFactorial = 1;
 8     end else begin
 9       calFactorial =  number * (calFactorial(number - 1));
10     end
11   end
12 endfunction
13
14 initial begin
15  $display ("Factorial of 1 : %d", calFactorial(1));
16  $display ("Factorial of 4 : %d", calFactorial(4));
17  $display ("Factorial of 8 : %d", calFactorial(8));
18  $display ("Factorial of 16 : %d", calFactorial(16));
19  $display ("Factorial of 32 : %d", calFactorial(32));
20   #1  $finish;
21 end
22
23 endmodule
```

You could download file recursive_function.v here

Satıra isim ile parametre gönderme(In-line parameter passing by name)

Verilog 1995 bir modülde bildirilmiş parametreleri 2 yolda geçersiz kılabilir(override):

- defparam
- # kullanarak

Verilog 2001 problemin çözümü için bir yol eklemiştir buna # kullanımı denir. Bu işlem port bağlantısı isimle yapılarak yapılır, parametreler pozisyonları yerine isimleriyle geçersiz kılınır.

```
 1 module parameter_v2k();
 2 parameter D_WIDTH = 4;
 3 parameter A_WIDTH = 9;
 4
 5 reg  [A_WIDTH-1:0] address = 0;
 6 reg  [D_WIDTH-1:0] data_in = 0;
 7 wire [D_WIDTH-1:0] data_out;
 8 reg  rd,wr;
 9
10 initial begin
11   $monitor ("%g addr %d din %h dout %d read %b write %b",
12     $time, address, data_in, data_out, rd, wr);
13   rd = 0;
14   wr = 0;
15    #1  repeat (10) begin
16     wr = 1;
17     data_in = $random;
18      #1  address = address + 1;
19   end
20   wr = 0;
21   address  = 0;
22   data_in = 0;
23    #1  repeat (10) begin
24     rd = 1;
25      #1  address = address + 1;
26   end
27   rd = 0;
28    #1  $finish;
29 end
30
31 memory #(.AWIDTH(A_WIDTH),.DWIDTH(D_WIDTH)) U (
32   address, data_in, data_out, rd, wr);
33
34 endmodule
35
36 // Bellek modeli(Memory model)
37 module memory (address, data_in, data_out, rd, wr);
38 parameter DWIDTH = 8;
39 parameter AWIDTH = 8;
40 parameter DEPTH  = 1 << AWIDTH;
41
42 input  [AWIDTH-1:0] address;
43 input  [DWIDTH-1:0] data_in;
44 output [DWIDTH-1:0] data_out;
45 input  rd,wr;
46
47 reg [DWIDTH-1:0] mem [0:DEPTH-1];
```

```
48
49 always @ (*)
50   if (wr) mem[address] = data_in;
51
52 assign data_out = (rd) ? mem[address] : 0;
53
54 endmodule
```

You could download file parameter_v2k.v here

Rasgele(Random) Üreteci(Generator)

Verilog 1995'de her simülatör kendi $random versiyonunu gerçeklemekteydi. Verilog 2001'de $random standartlaştırıldı, böylece simülasyonlar tüm simülatörlerde hiçbir uyumsuzluk olmadan çalışabilmektedir.

Dosya Giriş/Çıkış (FileIO- Input/Output)

Bu Verilog 2001 'e eklenmiş iyi özelliklerden biridir. Verilog 1995'de dosya giriş/çıkışı(file IO) bellek dizisindeki hex dosyaları readmemh ile okunmasıyla sınırlandırılmıştır; ve dosyaya yazma $display ve $monitor ile sınırlandırılmıştır.

Fakat Verilog 2001'de, aşağıdaki işlemler gerçekleştirilebilir.

- C veya C++ tipi dosya işlemleri (dosyanın sonunun kontrolü gibi).
- Dosyanın karışık yerindeki karakterlerin okunması.
- Dosyadaki biçimlendirilmiş satırın okunması.
- Bir dosyaya biçimlendirilmiş şekilde yazmak.

Bir dosyanın açılması

Bir dosya okuma(reading) veya yazma(writing) için açılabilir, ve sözdizimi aşağıdaki gibidir.

file = $fopen("dosyaismi",r); // Okumak (reading) için

file = $fopen("dosyaismi",w); // Yazmak(writing) için

Aşağıdaki tabloda olası tüm $fopen modları bulunmaktadır.

"r" veya"rb"	Okumak için dosya aç
"w" or "wb"	Sıfır uzunluğuna kırp(truncate) veya yazmak için dosya aç
"a" or "ab"	Append (sonuna eklemek)(Dosyanın sonunu yazmak için aç)
"r+", "r+b", or "rb+"	Güncellemek için aç (okuma ve yazma)
"w+", "w+b", or "wb+"	Kırp(Truncate) veya güncellemek için yarat
"a+", "a+b", or "ab+"	Sonuna ekle(Append); dosyanın sonunda güncelleme için aç veya yarat

Bir dosyanın kapatılması(Closing a file)

Bir dosya aşağıdaki gibi kapatılabilir.

$fclose(file); // Burada file $fopen ile atanmış tanıtıcıdır

Bir dosyadan verinin okunması(Reading data from a file)

Verilog 2001 FileIO aşağıdaki yollardan bir dosyayı okumayı destekler.

- $fgetc ile bir karakter okuma.
- $fgets ile bir satır okuma.
- $fscanf ile biçimlendirilmiş veri okuma. $fscanf fonksiyonu dosya açıklayıcısı ile belirtilmiş dosyadan karakterleri okur ve bunları biçime göre yorumlar ve argümanlarına göre sonuçları kaydeder.
- $fread ile ikili veri okuma. $fread fonksiyonu bir yazmaçtaki veya bir bellekteki dosya açıklayıcısı ile belirtilmiş dosyadan ikili veriyi okur.

Herbir fonksiyon için detaylı açıklama için lütfen Verilog 2001 LRM 'e bakınız.

Aşağıdaki kodda doğrulama ortamında Verilog fileio nasıldır onu göreceğiz.

Örnek : Verilog FileIO

```
 1 module fileio;
 2
 3 integer in,out,mon;
 4 reg clk;
 5
 6 reg  enable;
 7 wire valid;
 8 reg [31:0] din;
 9 reg [31:0] exp;
10 wire [31:0] dout;
11 integer statusI,statusO;
12
13 dut dut (clk,enable,din,dout,valid);
14
15 initial begin
16   clk = 0;
17   enable  = 0;
18   din = 0;
19   exp = 0;
20   in  = $fopen("input.txt","r");
21   out = $fopen("output.txt","r");
22   mon = $fopen("monitor.tXt","w");
23 end
24
25 always  #  1 clk = ~clk;
26
27 // DUT giriş sürücü kodu (input driver code)
28 initial begin
29     repeat (10) @ (posedge clk);
30     while ( ! $feof(in)) begin
31       @ (negedge clk);
32       enable = 1;
```

```
 33       statusI = $fscanf(in,"%h %h\n",din[31:16],din[15:0]);
 34       @ (negedge clk);
 35       enable = 0;
 36     end
 37     repeat (10) @ (posedge clk);
 38     $fclose(in);
 39     $fclose(out);
 40     $fclose(mon);
 41      #100  $finish;
 42 end
 43
 44 // DUT çıkış izleme ve karşılaştırma lojiği (output monitor and compare
logic)
 45 always @ (posedge clk)
 46  if (valid) begin
 47    $fwrite(mon,"%h %h\n",dout[31:16],dout[15:0]);
 48    statusO = $fscanf(out,"%h %h\n",exp[31:16],exp[15:0]);
 49    if (dout  ! == exp) begin
 50      $display("%0dns Error : input and output does not match",$time);
 51      $display("       Got  %h",dout);
 52      $display("       Exp  %h",exp);
 53    end else begin
 54      $display("%0dns Match : input and output match",$time);
 55      $display("       Got  %h",dout);
 56      $display("       Exp  %h",exp);
 57    end
 58  end
 59
 60 endmodule
 61
 62 // DUT modeli
 63 module dut(
 64   input wire clk,enable,
 65   input wire [31:0] din,
 66   output reg [31:0] dout,
 67   output reg       valid
 68 );
 69
 70 always @ (posedge clk)
 71   begin
 72     dout  <= din + 1;
 73     valid  <= enable;
 74   end
 75
 76 endmodule
```

You could download file compare.v here

✦Giriş Dosyası(Input File)

```
0456 08a0
0457 08a1
0458 08a2
0459 08a3
045a 08a4
045b 08a5
045c 08a6
045d 08a7
045e 08a8
045f 08a9
0460 08aa
0461 08ab
0462 08ac
```

```
0463 08ad
0464 08ae
0465 08af
0466 08b0
0467 08b1
0468 08b2
0469 08b3
046a 08b4
046b 08b5
046c 08b6
046d 08b7
046e 08b8
046f 08b9
0470 08ba
0471 08bb
0472 08bc
0473 08bd
```

✦Beklenen Çıkış Dosyası (Output File)

```
0456 08a1
0457 08a2
0458 08a3
0459 08a4
045a 08a5
045b 08a6
045c 08a7
045d 08a8
045e 08a9
045f 08aa
0460 08ab
0461 08ac
0462 08ad
0463 08ae
0464 08af
0465 08b0
0466 08b1
0467 08b2
0468 08b3
0469 08b4
046a 08b5
046b 08b6
046c 08b7
046d 08b8
046e 08b9
046f 08ba
0470 08bb
0471 08bc
0472 08bd
0473 08be
```

Verilog'da İspat(Assertions)

Giriş

İspat(Assertion) ile Doğrulama(Verification) tasarımın beklenen davranışı gösterip göstermediğini belirtmek için ispat dili ile kullanılır, ve ispatı ölçen araçlar doğrulama altında tasarıma(design under verification) aittir.

İspat tabanlı doğrulama(Assertion-based verification- ABV) en çok dijital blok ve sistemlerin RTL tasarımlarından sorumlu tasarım ve doğrulama mühendisleri için kullanışlıdır. ABV tasarım mühendilerinin tasarım esnasında doğrulama bilgilerini yakalamasını sağlar. Ayrıca iç durum(internal state), veriyolu(datapath), ve hata önkoşul kaplam analizini etkinleştirir.

Basit bir ispat örneği FIFO olabilir; FIFO dolu olduğunda ve yazma gerçekleşmesi geçersizdir. Bu yüzden bir FIFO tasarımcısı bu koşulun gerçekleyip gerçeklemediğini kontrol eden bir ispat yazar.

İspat Dilleri (Assertions Languages)

Şu anda ispat yazmak için birçok yol vardır, bunlar aşağıda gösterilmiştir.

- Açık Doğrulama Kütüphanesi(Open Verification Library (OVL)).
- Formal Property Language Sugar
- SystemVerilog Assertions

Birçok ispat HDL de yazılabilir, fakat HDL ispatları uzunca ve karmaşık olabilir. Bu ispatın amacını yokeder ki amacı tasarımın doğruluğundan emin olmaktır. Uzunca ve karmaşık olan HDL ispatlarının yaratılması zordur ve kendileri hataya konu olabilir.

Bu eğitselde verilog tabanlı ispat (verilog based assertion (OVL)) ve PSL (sugar-şeker) göreceğiz.

İspat kullanmanın avantajları

- Tasarımın iç noktaları test edilebiliyor böylece tasarımın gözlemlenebilirliği artıyor.
- İspat monitörü kontrol edilerek bir hatanın gerçekleştirilmesi kısıtlanmıştır böylece hatanın tanısı(diagnosis) ve sezimi(detection) kolaylaştırılmıştır.
- Tasarımcılara aynı ispatı hem simülasyon hemde resmi doğrulama için kullanmalarına olanak verir.

İspat görüntüleyicinin gerçeklenmesi(Implementing assertion monitors)

İspat görüntüleyicileri tasarım doğrulamayı gösterir ve tasarım güvenini arttırmak için takip edilebilir.

- Tasarımın kapsamını arttırmak için ispat görüntüleyicisini birleştir(örneğin arayüz devrelerinde köşe durumlarında).
- Bir modülün dış arayüzü varsa ispat monitörünü ekle. Bu durumda doğru girişler için varsayımlar ve çıkış davranışı garanti altına alınmış ve doğrulanmıştır.
- Üçüncü parti modüllerle arayüzlendiğinde ispat görüntüleyicisini ekleyin, çünkü tasarımcı modül tanımına aşina olmayabilir veya modülü tamamen anlamamış olabilir. Bu durumlarda, modülü ispat görüntüleyicisi ile korumak modülün yanlış kullanılmasını önleyebilir.

Normalde ispatler tasarımcılar tarafından tasarımlarını korumak için gerçeklenir, böylece ispatu RTL içinde kodlarlar. Geleneksel olarak doğrulama mühendisleri ispatları doğrulama ortamında ispatlama yaptığını bilmeden kullanırlar. Doğrulama için basit bir ispat uygulaması protokolü kontrol edebilir.

Sonraki birkaç sayfada Open Verification Library ve PSL kullanarak ispat yapımının basit örneklerini göreceğiz.

Neye İhtiyacınız Var?

Open Verification Library örneklerini kullanmak için Accellera'dan indirmeniz gerekmektedir. PSL örneklerini çalıştırmak için PSL destekleyen bir simülatöre ihityacınız vardır.

Daha sonra da İspatla ilgili detayları öğrenmek ve daha çok örnek denemek için biraz sabra ihtiyacınız vardır.

FIFO 'nun Doğrulanması

Bizim ilk örneğimiz senkron bir FIFO'nun doğrulanmasıydı. Burada FIFO modelinin etrafında bir testbenç kuracağız ve basit ispatın basit temel protokolleri kontrol etmek için nasıl kullanıldığını göstereceğiz. Eğer daha iyi bir öneriniz varsa lütfen bana iletin.

FIFO Modeli

Aşağıda örnekler dizinindeki orijinal kodu bulabilirsiniz.

```
1 //-----------------------------------------------------
2 // Tasarım İsmi : syn_fifo
3 // Dosya İsmi   : syn_fifo.v
4 // Fonksiyonu   : Synchronous (single clock) FIFO
5 // Kodlayan     : Deepak Kumar Tala
6 //-----------------------------------------------------
7 module syn_fifo (
8 clk      , // Saat girişi(Clock input)
9 rst      , // Aktif yüksek sıfırlama(Active high reset)
```

```
10 wr_cs    , // Yazma chip seçimi (Write chip select)
11 rd_cs    , // Okuma chip seçimi(Read chip select)
12 data_in  , // Veri girişi(Data input)
13 rd_en    , // Okuma etkin (Read enable)
14 wr_en    , // Yazma Etkin (Write Enable)
15 data_out , // Veri Çıkışı (Data Output)
16 empty    , // FIFO boş(empty)
17 full       // FIFO dolu(full)
18 );
19
20 // FIFO sabitleri (constants)
21 parameter DATA_WIDTH = 8;
22 parameter ADDR_WIDTH = 8;
23 parameter RAM_DEPTH = (1 << ADDR_WIDTH);
24 // Port Bildirimleri (Declarations)
25 input clk ;
26 input rst ;
27 input wr_cs ;
28 input rd_cs ;
29 input rd_en ;
30 input wr_en ;
31 input [DATA_WIDTH-1:0] data_in ;
32 output full ;
33 output empty ;
34 output [DATA_WIDTH-1:0] data_out ;
35
36 //-----------İç Değişkenler-------------------
37 reg [ADDR_WIDTH-1:0] wr_pointer;
38 reg [ADDR_WIDTH-1:0] rd_pointer;
39 reg [ADDR_WIDTH :0] status_cnt;
40 reg [DATA_WIDTH-1:0] data_out ;
41 wire [DATA_WIDTH-1:0] data_ram ;
42
43 //-----------Değişken atamaları---------------
44 assign full = (status_cnt == (RAM_DEPTH-1));
45 assign empty = (status_cnt == 0);
46
47 //-----------Kod Başlangıcı---------------------------
48 always @ (posedge clk or posedge rst)
49 begin : WRITE_POINTER
50   if (rst) begin
51     wr_pointer <= 0;
52   end else if (wr_cs && wr_en ) begin
53     wr_pointer <= wr_pointer + 1;
54   end
55 end
56
57 always @ (posedge clk or posedge rst)
58 begin : READ_POINTER
59   if (rst) begin
60     rd_pointer <= 0;
61   end else if (rd_cs && rd_en ) begin
62     rd_pointer <= rd_pointer + 1;
63   end
64 end
65
66 always  @ (posedge clk or posedge rst)
67 begin : READ_DATA
68   if (rst) begin
69     data_out <= 0;
70   end else if (rd_cs && rd_en ) begin
```

```
    data_out <= data_ram;
  end
end

always @ (posedge clk or posedge rst)
begin : STATUS_COUNTER
  if (rst) begin
    status_cnt <= 0;
  // Okuma fakat yazma yok(Read but no write).
  end else if ((rd_cs && rd_en) &&  ! (wr_cs && wr_en)
                && (status_cnt  ! = 0)) begin
    status_cnt <= status_cnt - 1;
  // Yazma fakat okuma yok(Write but no read).
  end else if ((wr_cs && wr_en) &&  ! (rd_cs && rd_en)
              && (status_cnt  ! = RAM_DEPTH)) begin
    status_cnt <= status_cnt + 1;
  end
end

ram_dp_ar_aw #(DATA_WIDTH,ADDR_WIDTH)DP_RAM (
.address_0 (wr_pointer) , // address_0 input
.data_0    (data_in)    , // data_0 bi-directional
.cs_0      (wr_cs)      , // chip select
.we_0      (wr_en)      , // write enable
.oe_0      (1'b0)       , // output enable
.address_1 (rd_pointer) , // address_q input
.data_1    (data_ram)   , // data_1 bi-directional
.cs_1      (rd_cs)      , // chip select
.we_1      (1'b0)       , // Read enable
.oe_1      (rd_en)        // output enable
);

endmodule
```

You could download file here

Ram(Rasgele Erişim Hafıza) Modeli

```
//-----------------------------------------------------
// Tasarım İsmi : ram_dp_ar_aw
// Dosya İsmi   : ram_dp_ar_aw.v
// Fonksiyonu   : Asynchronous read write RAM
// Kodlayan     : Deepak Kumar Tala
//-----------------------------------------------------
module ram_dp_ar_aw (
address_0 , // address_0 Girişi(Input)
data_0    , // data_0 çift-yönlü (bi-directional)
cs_0      , // Chip Seaçimi(Select)
we_0      , // Yazma Etkin/Okuma Etkin (Write Enable/Read Enable)
oe_0      , // Çıkış Etkin (Output Enable)
address_1 , // address_1 Girişi(Input)
data_1    , // data_1 çift-yönlü(bi-directional)
cs_1      , // Chip Seçimi(Select)
we_1      , // Yazma Etkin/Okuma Etkin(Write Enable/Read Enable)
oe_1        // Çıkış Etkin(Output Enable)
);

parameter DATA_WIDTH = 8 ;
parameter ADDR_WIDTH = 8 ;
parameter RAM_DEPTH = 1 << ADDR_WIDTH;

```

```
24 //--------------Giriş Portları-----------------------
25 input [ADDR_WIDTH-1:0] address_0 ;
26 input cs_0 ;
27 input we_0 ;
28 input oe_0 ;
29 input [ADDR_WIDTH-1:0] address_1 ;
30 input cs_1 ;
31 input we_1 ;
32 input oe_1 ;
33
34 //--------------GirişÇıkış Portları-----------------------
35 inout [DATA_WIDTH-1:0] data_0 ;
36 inout [DATA_WIDTH-1:0] data_1 ;
37
38 //--------------İç değişkenler----------------
39 reg [DATA_WIDTH-1:0] data_0_out ;
40 reg [DATA_WIDTH-1:0] data_1_out ;
41 reg [DATA_WIDTH-1:0] mem [0:RAM_DEPTH-1];
42
43 //--------------Kod Buradan Başlar-----------------
44 // Memory Write Block
45 // Write Operation : When we_0 = 1, cs_0 = 1
46 always @ (address_0 or cs_0 or we_0 or data_0
47 or address_1 or cs_1 or we_1 or data_1)
48 begin : MEM_WRITE
49   if ( cs_0 && we_0 ) begin
50      mem[address_0] <= data_0;
51   end else if  (cs_1 && we_1) begin
52      mem[address_1] <= data_1;
53   end
54 end
55
56 // Üç-durumlu(Tri-State) Arabellek(Buffer) kontorlü
57 // çıkış(output) : we_0 = 0, oe_0 = 1, cs_0 = 1 olduğunda
58 assign data_0 = (cs_0 && oe_0 &&  ! we_0) ? data_0_out : 8'bz;
59
60 // Bellek Okuma Bloğu(Memory Read Block)
61 // Okuma İşlemi(Read Operation): we_0 = 0, oe_0 = 1, cs_0 = 1 olduğunda
62 always @ (address_0 or cs_0 or we_1 or oe_0)
63 begin : MEM_READ_0
64   if (cs_0 &&  ! we_0 && oe_0) begin
65     data_0_out <= mem[address_0];
66   end else begin
67     data_0_out <= 0;
68   end
69 end
70
71 //RAM'in ikinci Port'u
72 // Üç-durumlu Arabellek kontrolü(Tri-State Buffer control)
73 // Çıkış(output) : we_0 = 0, oe_0 = 1, cs_0 = 1 olduğunda
74 assign data_1 = (cs_1 && oe_1 &&  ! we_1) ? data_1_out : 8'bz;
75 // Bellek Okuma Bloğu (Memory Read Block) 1
76 // Okuma İşlemi(Read Operation):we_1 = 0, oe_1 = 1, cs_1 = 1 olduğunda
77 always @ (address_1 or cs_1 or we_1 or oe_1)
78 begin : MEM_READ_1
79   if (cs_1 &&  ! we_1 && oe_1) begin
80     data_1_out <= mem[address_1];
81   end else begin
82     data_1_out <= 0;
83   end
84 end
```

```
85
86 endmodule // ram_dp_ar_aw Modülünün Sonu
```

You could download file here

Testbenç Kodu

Aşağıdaki testbenç kodunda, taşma(overflow) ve bomboşalma(underflow) durumlarına neden olabilir. Benim söylemek istediğim şey FIFO derinliği 8'dir, böylece FIFO dan okumadan 8 yazma işlemi yapabiliriz. Eğer 9 yazma işlemi yaparsak 9. veri FIFO'nun içeriğine yazar.

Benzer şekilde,FIFO boş olduğunda FIFO'dan okursak bu bomboşalmaya(underflow) neden olur. Bu tür şeyler kod arayüz bloğu hatalıysa gerçekleşir. İspatlama kodunu RTL'de veya testbençde yapabiliriz. Bizim örneğimizdeki gibi ispatlamayı, RTL tasarımcısının kodunun içerisinde kendi yapması daha iyidir.

```
 1 module fifo_tb ();
 2 parameter DATA_WIDTH = 8;
 3 // derinliği 8 ile sınırla (Limit depth to 8)
 4 parameter ADDR_WIDTH = 3;
 5
 6 reg clk, rst, rd_en, wr_en;
 7 reg [DATA_WIDTH-1:0] data_in ;
 8 wire [DATA_WIDTH-1:0] data_out ;
 9 wire empty, full;
10 integer i;
11
12 initial begin
13   $monitor ("%g wr:%h wr_data:%h rd:%h rd_data:%h",
14     $time, wr_en, data_in,  rd_en, data_out);
15   clk = 0;
16   rst = 0;
17   rd_en = 0;
18   wr_en = 0;
19   data_in = 0;
20    #5  rst = 1;
21    #5  rst = 0;
22   @ (negedge clk);
23   wr_en = 1;
24   // Taşmaya neden oluyoruz(We are causing over flow)
25   for (i = 0 ; i < 10; i = i + 1) begin
26      data_in  = i;
27      @ (negedge clk);
28   end
29   wr_en  = 0;
30   @ (negedge clk);
31   rd_en = 1;
32   // Bomboşalmaya neden oluyoruz(We are causing under flow)
33   for (i = 0 ; i < 10; i = i + 1) begin
34      @ (negedge clk);
35   end
36   rd_en = 0;
37    #100  $finish;
38 end
39
40 always  #1  clk =  ! clk;
41
42 syn_fifo #(DATA_WIDTH,ADDR_WIDTH) fifo(
43 .clk      (clk)     , // Clock input
```

```
44 .rst      (rst)      , // Aktif yüksek sıfırlama(Active high reset)
45 .wr_cs    (1'b1)     , // Chip seçimine yazma(Write chip select)
46 .rd_cs    (1'b1)     , // Chip seçimini okuma (Read chip select)
47 .data_in  (data_in)  , // Veri girişi(Data input)
48 .rd_en    (rd_en)    , // Okuma etkin(Read enable)
49 .wr_en    (wr_en)    , // Yazma Etkin (Write Enable)
50 .data_out (data_out), // Veri Çıkış(Data Output)
51 .empty    (empty)    , // FIFO boş(empty)
52 .full     (full)       // FIFO dolu(full)
53 );
54
55 endmodule
```

You could download file fifo_tb.v here

OVL ile İspatlama(Assertion)

Şimdiye kadar FIFO ve testbenç kodunu gördük, şimdi de OVL kullanarak FIFO için ispat oluşturma örneğini göreceğiz.OVL kullanmak için öncelikle OVL paketini yüklemeliyiz. Daha sonra da kullanmamız gereken ispat dosyasını eklemeliyiz. Bizim örneğimizde assert_fifo_index.vlib kullanmaktayız, sentez araçlarının **synopsys translate_off** ve **synopsys translate_on** içindeki kodu okumalarını önlemek için snopsys translate_off kullanıyoruz. Bunu yapmamız gerekiyor çünkü bu sadece bir simülasyon kodu sentez için değil. Daha sonra `define OVL_ASSERT_ON ile ispatı etkinleştirmemiz gerekmektedir. OVL ispatlarını kontrol etmek için kullanabileceğimiz birçok farklı tanım vardır; herbir seçenek için detay OVL manual'dan bulunabilir.

RTL'de İspat(Assertion)

Aşağıdaki kodda, OVL ispatlarını hissetemek için kullanıyoruz.

- **assert_fifo_index :** Taşma(overflow) veya bomboşalma(underflow) hataları olduğunda hataları yazdırır.
- **assert_always :** Fifo dolu bayrağı varken yazma gerçekleştiğinde hatayı yazdırır.
- **assert_never :** Fifo boş bayrağı varken okuma olduğunda hataları yazdırır.
- **assert_increment :** Yazma işaretçisi (write pointer) >1 ile arttırıldığında hataları yazdırır.

Birçok parametreyi, ispatl istenilen fonksiyonu almak için kullanabilir; assert_fifo_index için, kurulması gereken parametreler (diğerleri için OVL dokümanlarına bakınız):

- önem seviyesi(severity_level) : `OVL_ERROR, Bu ölümcül hata,uyarı,vb için kurulabilir.
- FIFO'nun derinliği(depth of FIFO): FIFO'nun derinliği için kurulur.
- msg : Yazdırmak istediğimi mesajdır(message).

Bu ispat için detaylı bilgiyi OVL manual'ında bulabilirsiniz. Şimdi derlemek için +incdir yolundan VLIB yükleme dizinine geçmemiz gerekmektedir.

Örnek: varsayalım ki vlib home dizinine vlib adıyla yüklenmiş (/home/deepak) ve biz verilog XL kullanıyoruz derlemek için.

verilog +incdir+/home/deepak/vlib syn_fifo_assert.v fifo_tb.v ram_dp_ar_aw.v

```
 1 //=============================================
 2 // Function  : Synchronous (single clock) FIFO
 3 //             With Assertion
 4 // Coder     : Deepak Kumar Tala
 5 // Date      : 1-Nov-2005
 6 //=============================================
 7 // synopsys translate_off
 8 `define OVL_ASSERT_ON
 9 `define OVL_INIT_MSG
10 `include "assert_fifo_index.vlib"
11 `include "assert_always.vlib"
12 `include "assert_never.vlib"
13 `include "assert_increment.vlib"
14 // synopsys translate_on
15 module syn_fifo (
16 clk      , // Clock input
17 rst      , // Active high reset
18 wr_cs    , // Write chip select
19 rd_cs    , // Read chipe select
20 data_in  , // Data input
21 rd_en    , // Read enable
22 wr_en    , // Write Enable
23 data_out , // Data Output
24 empty    , // FIFO empty
25 full       // FIFO full
26 );
27
28 // FIFO sabitleri(constants)
29 parameter DATA_WIDTH = 8;
30 parameter ADDR_WIDTH = 8;
31 parameter RAM_DEPTH = (1 << ADDR_WIDTH);
32 // Port Bildirimleri(Declarations)
33 input clk ;
34 input rst ;
35 input wr_cs ;
36 input rd_cs ;
37 input rd_en ;
38 input wr_en ;
39 input [DATA_WIDTH-1:0] data_in ;
40 output full ;
41 output empty ;
42 output [DATA_WIDTH-1:0] data_out ;
43
44 //-----------İç değişkenler(Internal variables)-------------------
45 reg [ADDR_WIDTH-1:0] wr_pointer;
46 reg [ADDR_WIDTH-1:0] rd_pointer;
47 reg [ADDR_WIDTH :0] status_cnt;
48 reg [DATA_WIDTH-1:0] data_out ;
49 wire [DATA_WIDTH-1:0] data_ram ;
50
51 //-----------Değişken Atamaları(Variable assignments)--------------
52 assign full = (status_cnt == (RAM_DEPTH-1));
53 assign empty = (status_cnt == 0);
54
```

```
 55 //-----------Kod BAşlangıcı---------------------------
 56 always @ (posedge clk or posedge rst)
 57 begin : WRITE_POINTER
 58   if (rst) begin
 59    wr_pointer <= 0;
 60   end else if (wr_cs && wr_en ) begin
 61    wr_pointer <= wr_pointer + 1;
 62   end
 63 end
 64
 65 always @ (posedge clk or posedge rst)
 66 begin : READ_POINTER
 67   if (rst) begin
 68     rd_pointer <= 0;
 69   end else if (rd_cs && rd_en ) begin
 70     rd_pointer <= rd_pointer + 1;
 71   end
 72 end
 73
 74 always  @ (posedge clk or posedge rst)
 75 begin : READ_DATA
 76   if (rst) begin
 77     data_out <= 0;
 78   end else if (rd_cs && rd_en ) begin
 79     data_out <= data_ram;
 80   end
 81 end
 82
 83 always @ (posedge clk or posedge rst)
 84 begin : STATUS_COUNTER
 85   if (rst) begin
 86     status_cnt <= 0;
 87   // Oku fakat yazma(Read but no write).
 88   end else if ((rd_cs && rd_en) &&  ! (wr_cs && wr_en)
 89                 && (status_cnt  ! = 0)) begin
 90     status_cnt <= status_cnt - 1;
 91   // Yaz fakat okuma(Write but no read).
 92   end else if ((wr_cs && wr_en) &&  ! (rd_cs && rd_en)
 93                && (status_cnt  ! = RAM_DEPTH)) begin
 94     status_cnt <= status_cnt + 1;
 95   end
 96 end
 97
 98 ram_dp_ar_aw #(DATA_WIDTH,ADDR_WIDTH) DP_RAM (
 99 .address_0 (wr_pointer) , // address_0 input
100 .data_0    (data_in)    , // data_0 bi-directional
101 .cs_0      (wr_cs)      , // chip select
102 .we_0      (wr_en)      , // write enable
103 .oe_0      (1'b0)       , // output enable
104 .address_1 (rd_pointer) , // address_q input
105 .data_1    (data_ram)   , // data_1 bi-directional
106 .cs_1      (rd_cs)      , // chip select
107 .we_1      (1'b0)       , // Read enable
108 .oe_1      (rd_en)        // output enable
109 );
110
111 // İspatı buraya ekle
112 // synopsys translate_off
113 // taşma(overflow) ve bomboşalma(underflow) kontorolü için İspat
(Assertion)
114 assert_fifo_index #(
```

```
115 `OVL_ERROR      , // severity_level(önem seviyesi)
116 (RAM_DEPTH-1)   , // depth(derinlik)
117 1               , // push width(ekleme genişliği)
118 1               , // pop width(çıkarma genişliği)
119 `OVL_ASSERT     , // property type(özellik tipi)
120 "my_module_err" , // msg
121 `OVL_COVER_NONE , //coverage_level(kapsama seviyesi)
122 1) no_over_under_flow (
123 .clk     (clk),           // Saat(Clock)
124 .reset_n (~rst),          // Aktif düşük sıfırlama(Active low reset)
125 .pop     (rd_cs & rd_en), // FIFO Yaz(Write)
126 .push    (wr_cs & wr_en)  // FIFO Oku(Read)
127 );
128
129 // Dolu ve yazma kontrolü için ispat
130 assert_always #(
131 `OVL_ERROR       , // severity_level
132 `OVL_ASSERT      , // property_type
133 "fifo_full_write", // msg
134 `OVL_COVER_NONE    // coverage_level
135 ) no_full_write (
136 .clk       (clk),
137 .reset_n   (~rst),
138 .test_expr ( ! (full && wr_cs && wr_en))
139 );
140
141 // Boş ve okuma kontrolü için ispat
142 assert_never #(
143 `OVL_ERROR       , // severity_level
144 `OVL_ASSERT      , // property_type
145 "fifo_empty_read", // msg
146 `OVL_COVER_NONE    // coverage_level
147 ) no_empty_read (
148 .clk       (clk),
149 .reset_n   (~rst),
150 .test_expr ((empty && rd_cs && rd_en))
151 );
152
153 // Yazma işaretçisi sadece bir tane arttırılmışsa diye kontrol ispatı
154 assert_increment #(
155 `OVL_ERROR            , // severity_level
156 ADDR_WIDTH            , // width
157 1                     , // value
158 `OVL_ASSERT           , // property_typ
159 "Write_Pointer_Error" , // msg
160 `OVL_COVER_NONE         // coverage_level
161 ) write_count (
162 .clk         (clk),
163 .reset_n     (~rst),
164 .test_expr   (wr_pointer)
165 );
166 // synopsys translate_on
167 endmodule
```

You could download file syn_fifo_assert.v here

Simülatör Çıktısı

İlk OVL mesajı başlangıç mesajı, bu OVL kütüphanesinin eklenmiş ve hazır olup olmadığını kontrol etmek için kullanılır. my_module_err kullanıcı tanımlı mesajdır ki OVL bunu yazdırma ifadelerinde kullanır.

Gerikalan mesajlar Taşma (OVERFLOW) ve Bomboşalma(UNDERFLOW) mesajlarıdır.

```
 OVL_NOTE: `OVL_VERSION: ASSERT_FIFO_INDEX initialized
 @ fifo_tb.fifo.no_over_under_flow.ovl_init_msg_t Severity: 1, Message:
my_module_err
 OVL_NOTE: `OVL_VERSION: ASSERT_ALWAYS initialized
 @ fifo_tb.fifo.no_full_write.ovl_init_msg_t Severity: 1, Message:
fifo_full_write
 OVL_NOTE: `OVL_VERSION: ASSERT_NEVER initialized
 @ fifo_tb.fifo.no_empty_read.ovl_init_msg_t Severity: 1, Message:
fifo_empty_read
 0 wr:0 wr_data:00 rd:0 rd_data:xx
 5 wr:0 wr_data:00 rd:0 rd_data:00
 10 wr:1 wr_data:00 rd:0 rd_data:00
 12 wr:1 wr_data:01 rd:0 rd_data:00
 14 wr:1 wr_data:02 rd:0 rd_data:00
 16 wr:1 wr_data:03 rd:0 rd_data:00
 18 wr:1 wr_data:04 rd:0 rd_data:00
 20 wr:1 wr_data:05 rd:0 rd_data:00
 22 wr:1 wr_data:06 rd:0 rd_data:00
 24 wr:1 wr_data:07 rd:0 rd_data:00
        OVL_ERROR : ASSERT_FIFO_INDEX : my_module_err : OVERFLOW : severity
1 :
        time 25 : fifo_tb.fifo.no_over_under_flow.ovl_error_t
        OVL_ERROR : ASSERT_ALWAYS : fifo_full_write :  : severity 1 : time
25 :
        fifo_tb.fifo.no_full_write.ovl_error_t
 26 wr:1 wr_data:08 rd:0 rd_data:00
        OVL_ERROR : ASSERT_FIFO_INDEX : my_module_err : OVERFLOW : severity
1 :
        time 27 : fifo_tb.fifo.no_over_under_flow.ovl_error_t
 28 wr:1 wr_data:09 rd:0 rd_data:00
        OVL_ERROR : ASSERT_FIFO_INDEX : my_module_err : OVERFLOW : severity
1 :
        time 29 : fifo_tb.fifo.no_over_under_flow.ovl_error_t
 30 wr:0 wr_data:09 rd:0 rd_data:00
 32 wr:0 wr_data:09 rd:1 rd_data:00
 33 wr:0 wr_data:09 rd:1 rd_data:08
 35 wr:0 wr_data:09 rd:1 rd_data:09
 39 wr:0 wr_data:09 rd:1 rd_data:03
 41 wr:0 wr_data:09 rd:1 rd_data:04
 43 wr:0 wr_data:09 rd:1 rd_data:05
 45 wr:0 wr_data:09 rd:1 rd_data:06
        OVL_ERROR : ASSERT_FIFO_INDEX : my_module_err : UNDERFLOW :
severity 1 :
        time 47 : fifo_tb.fifo.no_over_under_flow.ovl_error_t
 47 wr:0 wr_data:09 rd:1 rd_data:07
        OVL_ERROR : ASSERT_FIFO_INDEX : my_module_err : UNDERFLOW :
severity 1 :
        time 49 : fifo_tb.fifo.no_over_under_flow.ovl_error_t
        OVL_ERROR : ASSERT_NEVER : fifo_empty_read :  : severity 1 : time
49 :
        fifo_tb.fifo.no_empty_read.ovl_error_t
```

```
 49 wr:0 wr_data:09 rd:1 rd_data:08
        OVL_ERROR : ASSERT_FIFO_INDEX : my_module_err : UNDERFLOW :
severity 1 :
        time 51 : fifo_tb.fifo.no_over_under_flow.ovl_error_t
        OVL_ERROR : ASSERT_NEVER : fifo_empty_read :  : severity 1 : time
51 :
        fifo_tb.fifo.no_empty_read.ovl_error_t
 51 wr:0 wr_data:09 rd:1 rd_data:09
 52 wr:0 wr_data:09 rd:0 rd_data:09
```

OVL İspat(Assertion) Listesi

Aşağıda sık olarak kullanılan OVL ispat listesi bulunmaktadır; herzaman için OVL kütüphanesinden gelen dokümanları daha fazla bilgi elde etmek ve ispatın tam listesi için tercih edebilirsiniz.

İspat(Assertion)	Açıklama
assert_always	Tek bitlik test_expr ifadesinin saatin herbir yükselen kenarında doğru(TRUE) sonucu verip vermediğini doğrular.
assert_always_on_edge	Belirli tipteki geçiş için tek bitlik örnekleyen olay ifadesini kontrol eder.
assert_change	start_event ifadesini saatin herbir yükselen kenarında test_expr değerinde bir değişiklik varmı diye kontrol eder.Eğer start_event TRUE(doğru) olarak örneklendiyse, kontrolör test_expr 'ını gerçekleştirir ve test_expr ını takip eden herbir num_cks yükselen saat kenarında yeniden gerçekleştirir. test_expr örneklenmediyse num_cks döngüsünün sonunda başlangıç değerine getirilir, ispat başarısız olur.
assert_cycle_sequence	event_sequence olayını saatin yükselen kenarında, event_sequence daki bitlerle tanımlanıp tanımlanamayacağını saatin yükselen kenarında ardışıl olarak başarılı ispatlayıp ispatlayamayacağını kontrol eder.
assert_decrement	test_expr ifadesininin değerinin yükselen saat kenarında bir önceki yükselen saat kenarından itibaren değişip değişmediğini karar verir. Eğer değişmişse, kontrolör doğrulayarak eski değerine azaltır. Eğer toplam değişiklik azalış değerine eşitse, kontrolör test_expr ifadesinin değerinin sarılmasına izin verir.
assert_even_parity	The assert_even_parity assertion checker checks the expression test_expr at each rising edge of clk to verify the expression evaluates to a value that has even parity. A value has even parity if it is 0 or if the number of bits set to 1 is even.
assert_fifo_index	FIFO'ya kaç kez yazıldığını(push-write) ve okunduğunu(pop-read) sayısını takip eder. Bu kontrolör, aynı saat döngüsünde eşzamanlı yazma/okumaya izin verir. FIFO'nun asla taşmamasını veya bomboşalmamasını sağlar.

assert_increment	test_expr ifadesininin değerinin yükselen saat kenarında bir önceki yükselen saat kenarından itibaren değişip değişmediğini karar verir. Eğer değişmişse, kontrolör doğrulayarak eski değerine yükseltir. Eğer toplam değişiklik artış değerine eşitse, kontrolör test_expr ifadesinin değerinin sarılmasına izin verir.
assert_never	Tek bitlik test_expr ifadesinin saatin herbir yükselen kenarında doğru(TRUE) **olmayan** sonucu verip vermediğini doğrular.
assert_one_hot	Test_expr ifadesini saatin yükselen kenarında bir-sıcak(one-hot) değere gerçekleyip gerçeklemediğinin doğruluğunu kontrol eder. Bir-sıcak(one-hot value) değerde yalnızca ve yalnızca bir bit 1 olmak zorundadır.
assert_range	test_expr ifadesini saatin yükselen kenarında ifadenin min'den max'a düşüp düşmeyeceğini kontrol eder. test_expr< min veya max < test_expr ise ispat başarısız olur.
assert_one_cold	Test_expr ifadesini saatin yükselen kenarında bir-soğuk(one-cold) veya aktifolmayan durum değerine gerçekleyip gerçeklemediğinin doğruluğunu kontrol eder. Bir-soğuk(one-hot value) değerde yalnızca ve yalnızca bir bit 0 olmak zorundadır.

PSL ile İspat(Assertion)

Biraz önce OVL kullanarak FIFO ispat örneğini görmüştük, şimdide de PSL kullanarak FIFO için ispat nasıl yapılır onu göreceğiz. PSL ispat iki yolla kodlanır.

- Sıralı(inline) Kodlama: Bu metodda, psl ispatları verilog kodunda açıklama olarak kodlanır.
- Harici Dosya(External File) : Bu metodda, psl ispatları ayrı dosyalarda vunit ismiyle kodlanır.

Sıralı(inline) Kodlama

- Tüm ispatlar içeriğe uygun şekilde ardışık seri açıklamalar şeklinde görülür.
- İlk ispat ifades satırı psl anahtar sözcüğü ile başlamalıdır.
- Psl anahtar sözcüğü ve bunu takip eden anahtar sözcük aynı satırda olmalıdır.
- İki nokta üst üste ile bir etiken belirtilebilir.
- assert veya assume anahtar sözcükleri ile davranışın ispatı özellikte betimlenir.
- Tasarımın davranışını betimler.
- İspatlar Verilog görev, fonksiyon veya UDP'lerine gömülemezler.
- Örnek: // psl etiket: assert davranış;

Harici Dosya(External File)

- İspatlarınızı varolan tasarımınızı değiştirmeden,IP mirası ile, ekleyebilirsiniz.
- Kaynak koda gömmeden önce ispatı denemek için.
- Bir takımda çalışırken ispatlar HDL yazarı tarafından oluşturulmadığı yerlerde.

```
vunit verification_unit_name (module_name) {
  default clock = clock_edge;
  property_name: assert behavior;
  property_name: cover {behavior};
}
```

Daha detaylı bilgi için simülatör ile birlikte gelen PSL kullanım klavuzuna bakınız.

RTL'de İspat (Assertion)

Aşağıdaki kodda, psl ispatı ile FIFO doluyken nasıl yazma yapılmadığını ve ayrıca FIFO boşken nasıl okuma yapılmadığını göreceğiz.

PSL ispatını kod ile sıralı(inline) // veya /**/ ile kodlayabiliriz. Herhangi bir ispat yazmadan önce, örnekteki gibi saati bildirmemiz gerekmektedir.

ncverilog +assert verilog_file1.v verilog_file2.v

```
 1 //============================================
 2 // Fonksiyonu: Senkron(tek saatli) FIFO
 3 //             ile İspat(Assertion)
 4 // Kodlayan  : Deepak Kumar Tala
 5 // Tarih     : 31-October-2002
 6 //============================================
 7 module syn_fifo (
 8 clk      , // Clock input
 9 rst      , // Active high reset
10 wr_cs    , // Write chip select
11 rd_cs    , // Read chipe select
12 data_in  , // Data input
13 rd_en    , // Read enable
14 wr_en    , // Write Enable
15 data_out , // Data Output
16 empty    , // FIFO empty
17 full       // FIFO full
18 );
19
20 // FIFO constants
21 parameter DATA_WIDTH = 8;
22 parameter ADDR_WIDTH = 8;
23 parameter RAM_DEPTH = (1 << ADDR_WIDTH);
24 // Port Declarations
25 input clk ;
26 input rst ;
27 input wr_cs ;
28 input rd_cs ;
29 input rd_en ;
30 input wr_en ;
31 input [DATA_WIDTH-1:0] data_in ;
32 output full ;
33 output empty ;
34 output [DATA_WIDTH-1:0] data_out ;
35
36 //-----------Internal variables-------------------
37 reg [ADDR_WIDTH-1:0] wr_pointer;
```

```
38 reg [ADDR_WIDTH-1:0] rd_pointer;
39 reg [ADDR_WIDTH :0] status_cnt;
40 reg [DATA_WIDTH-1:0] data_out ;
41 wire [DATA_WIDTH-1:0] data_ram ;
42
43 //-----------Variable assignments--------------
44 assign full = (status_cnt == (RAM_DEPTH-1));
45 assign empty = (status_cnt == 0);
46
47 //-----------Code Start-------------------------
48 always @ (posedge clk or posedge rst)
49 begin : WRITE_POINTER
50   if (rst) begin
51    wr_pointer <= 0;
52   end else if (wr_cs && wr_en ) begin
53    wr_pointer <= wr_pointer + 1;
54   end
55 end
56
57 always @ (posedge clk or posedge rst)
58 begin : READ_POINTER
59   if (rst) begin
60     rd_pointer <= 0;
61     data_out <= 0;
62   end else if (rd_cs && rd_en ) begin
63     rd_pointer <= rd_pointer + 1;
64     data_out <= data_ram;
65   end
66 end
67
68 always  @ (posedge clk or posedge rst)
69 begin : READ_DATA
70   if (rst) begin
71     data_out <= 0;
72   end else if (rd_cs && rd_en ) begin
73     data_out <= data_ram;
74   end
75 end
76
77 always @ (posedge clk or posedge rst)
78 begin : STATUS_COUNTER
79   if (rst) begin
80     status_cnt <= 0;
81   // Read but no write.
82   end else if ((rd_cs && rd_en) &&  ! (wr_cs && wr_en)
83                 && (status_cnt  ! = 0)) begin
84     status_cnt <= status_cnt - 1;
85   // Write but no read.
86   end else if ((wr_cs && wr_en) &&  ! (rd_cs && rd_en)
87                && (status_cnt  ! = RAM_DEPTH)) begin
88     status_cnt <= status_cnt + 1;
89   end
90 end
91
92 ram_dp_ar_aw #(DATA_WIDTH,ADDR_WIDTH) DP_RAM (
93 .address_0 (wr_pointer) , // address_0 input
94 .data_0    (data_in)    , // data_0 bi-directional
95 .cs_0      (wr_cs)      , // chip select
96 .we_0      (wr_en)      , // write enable
97 .oe_0      (1'b0)       , // output enable
98 .address_1 (rd_pointer) , // address_q input
```

```
 99 .data_1     (data_ram)    , // data_1 bi-directional
100 .cs_1       (rd_cs)       , // chip select
101 .we_1       (1'b0)        , // Read enable
102 .oe_1       (rd_en)         // output enable
103 );
104
105 // Add assertion here
106 // psl default clock = (posedge clk);
107 // psl ERRORwritefull: assert never {full && wr_en && wr_cs};
108 // psl ERRORreadempty: assert never {empty && rd_en && rd_cs};
109
110 endmodule
```

You could download file syn_fifo_psl.v here

Simülatör Çıktısı

Eğer bir uyuşmazlık olursa, bir ispat ekrana yazdırılıyor.

```
0 wr:0 wr_data:00 rd:0 rd_data:xx
5 wr:0 wr_data:00 rd:0 rd_data:00
10 wr:1 wr_data:00 rd:0 rd_data:00
12 wr:1 wr_data:01 rd:0 rd_data:00
14 wr:1 wr_data:02 rd:0 rd_data:00
16 wr:1 wr_data:03 rd:0 rd_data:00
18 wr:1 wr_data:04 rd:0 rd_data:00
20 wr:1 wr_data:05 rd:0 rd_data:00
22 wr:1 wr_data:06 rd:0 rd_data:00
24 wr:1 wr_data:07 rd:0 rd_data:00
ncsim: *E,ASRTST (syn_fifo_psl.v,107): (time 25 NS)
Assertion fifo_tb.fifo.ERRORwritefull has failed
26 wr:1 wr_data:08 rd:0 rd_data:00
28 wr:1 wr_data:09 rd:0 rd_data:00
30 wr:0 wr_data:09 rd:0 rd_data:00
32 wr:0 wr_data:09 rd:1 rd_data:00
33 wr:0 wr_data:09 rd:1 rd_data:08
35 wr:0 wr_data:09 rd:1 rd_data:09
39 wr:0 wr_data:09 rd:1 rd_data:03
41 wr:0 wr_data:09 rd:1 rd_data:04
43 wr:0 wr_data:09 rd:1 rd_data:05
45 wr:0 wr_data:09 rd:1 rd_data:06
47 wr:0 wr_data:09 rd:1 rd_data:07
ncsim: *E,ASRTST (syn_fifo_psl.v,108): (time 49 NS)
Assertion fifo_tb.fifo.ERRORreadempty has failed
49 wr:0 wr_data:09 rd:1 rd_data:08
ncsim: *E,ASRTST (syn_fifo_psl.v,108): (time 51 NS)
Assertion fifo_tb.fifo.ERRORreadempty has failed
51 wr:0 wr_data:09 rd:1 rd_data:09
52 wr:0 wr_data:09 rd:0 rd_data:09
Simulation complete via $finish(1) at time 152 NS + 0
fifo_tb.v:36   #100 $finish;
```

İşlem Sonrası (Post Processing)

Diğer simülasyonlarla benzer şekilde, simülatör tarafından simülasyonun başarılı yada başarısız olduğunu yazan mesajları işleyen işlem sonrası betiklere sahip olması gerekmektedir.

Derleyici Yönergeleri(Compiler Directives)

Giriş

Bir derleyici yönergesi, bir Verilog betimlemesinin derlenmesini kontrol etmek için kullanılabilir. Aksan işareti (`) bir derleyici yönergesini bildirir. Bir yönerge, bildirildiği noktadan başka bir yönerge üstüne gelene kadar etkindir, birde dosya sınırları dahilinde. Derleyici yönergeleri kaynak betimlemesinin herhangi bir yerinde görülebilir, fakat bunun bir modül bildiriminin dışında olması gerekmektedir. Bu ekde bulunan yönergeler IEEE-1364 'ün bir parçasıdır.

Herhangi bir dilde olduğu gibi, herbir derleyicinin komut satırı seçeneklerini işlemek için kendi yolu vardır ve koddaki derleyici yönergeleri ile desteklenmiştir. Belirli bir derleyici yönergesi için lütfen simülatörün kullanım klavuzuna bakınız.

`include (eklemek)

`include derleyici yönergesi Verilog derlemesi esnasında belirtilmiş kaynak dosyanın içeriğini başka bir dosyaya ekler. Derleme eklenen kaynak dosyanın içeriğinin eklenmesini gösteren `include komutuna kadar devam eder. `include derleyici yönergesini global veya sıkça kullanılan tanımlamalar ve görevleri(task) modül sınırları içerisinde tekrarlayan kodu kısaltmadan kullanabilirsiniz.

`define (tanımlama)

Bu derleyici yönergesi metin MACROS(Makroları) tanımlamak için kullanılır; bu normalde "isim.vh"verilog dosyasının içinde tanımlanmıştır; "isim" kodladığınız modül olabilir. Yani `define birçok dosya üzerinde kullanılabilen bir derleyici yönergesidir.

`undef (tanımı sil)

`undef derleyici yönergesi `define ve +define+ komut satırı artı seçenekler ile tanımlanmış metin makroları siler. `undef derleyici yönergesini birden fazla dosyadan kullandığın bir metin makrosunun tanımın kaldırmak için kullanabilirsiniz.

`ifdef (eğer tanımlıysa)

Opsiyonel olarak derleme sırasında kaynak koda eklenir. `ifdef derleyici yönergesi bir makronun daha önce tanımlanıp tanımlanmadığını kontrol eder, eğer tanımlandıysa takip eden kodu derler. Eğer makro tanımlanmadıysa opsiyonel olan `else yönergesindeki kodu derler. Hangi kodun metin makrosunun derleneceğini `define veya +define+ 'dan biri ile kontrol edebilirsiniz. `endif yönergesi koşullu kodun sonunu belirtir.

Örnek : ifdef

```
 1 module ifdef ();
 2
 3 initial begin
 4 `ifdef FIRST
 5     $display("First code is compiled");
 6 `else
 7   `ifdef SECOND
 8     $display("Second code is compiled");
 9   `else
10     $display("Default code is compiled");
11   `endif
12 `endif
13   $finish;
14 end
15
16 endmodule
```

You could download file ifdef.v here

`timescale (zaman ölçeği)

`timescale derleyici yönergesi zaman birimini ve yönergeyi takip eden modülün hassasiyetini belirler. Zaman birimi zaman değerini ölçen birimdir, simülasyon zamanı ve gecikme değerleri gibi. Zama hassasiyeti (time precision) simülatörün zaman değerlerini nasıl yuvarlayacağını belirtir. Simülatörün kullandığı yuvarlanmış zaman değerleri, siizn belirttiğimiz zaman hassasiyeti zaman birimi içerisinde kesinleştirmek içindir.Belirtilmiş en küçük zaman hassasiyeti simülatörün çalışacağı kesinliğe karar verir ve bu nedenle hassasiyet simülasyon performansını ve bellek tüketimini etkiler.

Dizgi(String)	Birim(Unit)
s	Saniye (Seconds)
ms	Milisaniye (Milliseconds)
us	Mikrosaniye(Microseconds)
ns	Nanosaniye(Nanoseconds)
ps	Pikosaniye (Picoseconds)
fs	Femtosaniye(Femtoseconds)

`resetall (hepsini sıfırla)

`resetall yönergesi tüm derleyici yönergelerini varsayılan değerlerine atar.

`defaultnettype (varsayılan net tipi)

`defaultnettype yönergesi dolaylı olarak bildirilmiş net'lerin alışılmış varsayılan tipini(wire-tel) geçersiz kılınmasına izin verir. Bu modülde yönergeden sonra bildirilmiş tüm net'lerin varsayılan tipini belirler.

 `nounconnected_drive ve `unconnected_drive

`unconnected_drive ve `nounconnected_drive yönergeleri modülün bağlanmamış tüm giriş portlarını `unconnected_drive yönergesinin argümanına göre yukarı veya aşağı çeker(pull up-pull down). İzin verilen argümanlar pull0 ve pull1 ‘dır.

Verilog Hızlı Kaynağı (Quick Reference)

Verilog Hızlı Kaynağı

Bu hala ilk adımdır, daha fazlasını eklemek için zamana ihtiyaç vardır.

MODÜL(MODULE)

module MODID[({PORTID,})];

[input | output | inout [range] {PORTID,};]

[{declaration}]

[{parallel_statement}]

[specify_block]

endmodule

range ::= [constexpr : constexpr]

BİLDİRİMLER(DECLARATIONS)

parameter {PARID = constexpr,};

wire | wand | wor [range] {WIRID,};

reg [range] {REGID [range],};

integer {INTID [range],};

time {TIMID [range],};

real {REALID,};

realtime {REALTIMID,};

event {EVTID,};

task TASKID;

[{input | output | inout [range] {ARGID,};}]

[{declaration}]

begin

[{sequential_statement}]

end

endtask

function [range] FCTID;

{input [range] {ARGID,};}

[{declaration}]

begin

[{sequential_statement}]

end

endfunction

 **PARALEL İFADELER(PARALLEL STATEMENTS)**

assign [(strength1, strength0)] WIRID = expr;

initial sequential_statement

always sequential_statement

MODID [#({expr,})] INSTID

([{expr,} | {.PORTID(expr),}]);

GATEID [(strength1, strength0)] [#delay]

[INSTID] ({expr,});

defparam {HIERID = constexpr,};

strength ::= supply | strong | pull | weak | highz

delay ::= number | PARID | (expr [, expr [, expr]])

KAPI TEMELLERİ(GATE PRIMITIVES)

and (out, in1, ..., inN);

nand (out, in1, ..., inN);

or (out, in1, ..., inN);

nor (out, in1, ..., inN);

xor (out, in1, ..., inN);

xnor (out, in1, ..., inN);

buf (out1, ..., outN, in);

not (out1, ..., outN, in);

bufif1 (out, in, ctl);

bufif0 (out, in, ctl);

notif1 (out, in, ctl);

notif0 (out, in, ctl);

pullup (out);

pulldown (out);

[r]pmos (out, in, ctl);

[r]nmos (out, in, ctl);

[r]cmos (out, in, nctl, pctl);

[r]tran (inout, inout);

[r]tranif1 (inout, inout, ctl);

[r]tranif0 (inout, inout, ctl);

ARDIŞIL İFADELER(SEQUENTIAL STATEMENTS)

;

begin[: BLKID

[{declaration}]]

[{sequential_statement}]

end

if (expr) sequential_statement

[else sequential_statement]

case | casex | casez (expr)

[{{expr,}: sequential_statement}]

[default: sequential_statement]

endcase

forever sequential_statement

repeat (expr) sequential_statement

while (expr) sequential_statement

for (lvalue = expr; expr; lvalue = expr)

sequential_statement

#(number | (expr)) sequential_statement

@ (event [{or event}]) sequential_statement

lvalue [<]= [#(number | (expr))] expr;

lvalue [<]= [@ (event [{or event}])] expr;wait (expr) sequential_statement

-> EVENTID;

fork[: BLKID

[{declaration}]]

[{sequential_statement}]

join

TASKID[({expr,})];

disable BLKID | TASKID;

assign lvalue = expr;

deassign lvalue;

lvalue ::=

ID[range] | ID[expr] | {{lvalue,}}

event ::= [posedge | negedge] expr

BELİRTME BLOĞU (SPECIFY BLOCK)

specify_block ::= specify

{specify_statement}

endspecify

BELİRTME BLOK İFADESİ (SPECIFY BLOCK STATEMENTS)

specparam {ID = constexpr,};

(terminal => terminal) = path_delay;

((terminal,} *> {terminal,}) = path_delay;

if (expr) (terminal [+|-]=> terminal) = path_delay;

if (expr) ({terminal,} [+|-]*> {terminal,}) =

path_delay;

[if (expr)] ([posedge|negedge] terminal =>

(terminal [+|-]: expr)) = path_delay;

[if (expr)] ([posedge|negedge] terminal *>

({terminal,} [+|-]: expr)) = path_delay;

$setup(tevent, tevent, expr [, ID]);

$hold(tevent, tevent, expr [, ID]);

$setuphold(tevent, tevent, expr, expr [, ID]);

$period(tevent, expr [, ID]);

$width(tevent, expr, constexpr [, ID]);

$skew(tevent, tevent, expr [, ID]);

$recovery(tevent, tevent, expr [, ID]);

tevent ::= [posedge | negedge] terminal

[&&& scalar_expr]

path_delay ::=

expr | (expr, expr [, expr [, expr, expr, expr]])

terminal ::= ID[range] | ID[expr]

İFADELER (EXPRESSIONS)

primary

unop primary

expr binop expr

expr ? expr : expr

primary ::=

literal | lvalue | FCTID({expr,}) | (expr)

TEKLİ OPERATÖRLER (UNARY OPERATORS)

+, - Positive, Negative

! Logical negation

~ Bitwise negation

&, ~& Bitwise and, nand

|, ~| Bitwise or, nor

^, ~^, ^~ Bitwise xor, xnor

İKİLİ OPERATÖRLER (BINARY OPERATORS)

Artan öncelikle:
?: if/else

|| Logical or

&& Logical and

| Bitwise or

^, ^~ Bitwise xor, xnor

& Bitwise and

==, != , ===, !== Equality

<, <=, >, >= Inequality

<<, >> Logical shift

+, - Addition, Subtraction

*, /, % Multiply, Divide, Modulo

İFADELERİN BOYUTLARI (SIZES OF EXPRESSIONS)

unsized constant 32
sized constant as specified

i op j +,-,*,/,%,&,|,^,^~ max(L(i), L(j))

op i +, -, ~ L(i)

i op j ===, !==, ==, !=

&&, ||, >, >=, <, <= 1

op i &, ~&, |, ~|, ^, ~^ 1

i op j >>, << L(i)

i ? j : k max(L(j), L(k))

{i,...,j} L(i) + ... + L(j)

{i{j,...k}} i * (L(j)+...+L(k))

i = j L(i)

SİSTEM GÖREVLERİ(SYSTEM TASKS)

* görevin IEEE standardında olmadığını belirtir fakat öğretici ekde belirtilmiştir.

GİRİŞ (INPUT)

$readmemb("fname", ID [, startadd [, stopadd]]);

$readmemh("fname", ID [, startadd [, stopadd]]);

$sreadmemb(ID, startadd, stopadd {, string});

$sreadmemh(ID, startadd, stopadd {, string});

ÇIKIŞ (OUTPUT)

$display[defbase]([fmtstr,] {expr,});

$write[defbase] ([fmtstr,] {expr,});

$strobe[defbase] ([fmtstr,] {expr,});

$monitor[defbase] ([fmtstr,] {expr,});

$fdisplay[defbase] (fileno, [fmtstr,] {expr,});

$fwrite[defbase] (fileno, [fmtstr,] {expr,});

$fstrobe(fileno, [fmtstr,] {expr,});

$fmonitor(fileno, [fmtstr,] {expr,});

fileno = $fopen("filename");

$fclose(fileno);

defbase ::= h | b | o

ZAMAN (TIME)

$time "now" as TIME

$stime "now" as INTEGER

$realtime "now" as REAL

$scale(hierid) Scale "foreign" time value

$printtimescale[(path)] Display time unit & precision

$timeformat(unit#, prec#, "unit", minwidth)

Set time %t display format

SİMÜLASYON KONTROLÜ (SIMULATION CONTROL)

$stop Interrupt

$finish Terminate

$save("fn") Save current simulation

$incsave("fn") Delta-save since last save

$restart("fn") Restart with saved simulation

$input("fn") Read commands from file

$log[("fn")] Enable output logging to file

$nolog Disable output logging

$key[("fn")] Enable input logging to file

$nokey Disable input logging

$scope(hiername) Set scope to hierarchy

$showscopes Scopes at current scope

$showscopes(1) All scopes at & below scope

$showvars Info on all variables in scope

$showvars(ID) Info on specified variable

$countdrivers(net)>1 driver predicate

$list[(ID)] List source of [named] block

$monitoron Enable $monitor task

$monitoroff Disable $monitor task

$dumpon Enable val change dumping

$dumpoff Disable val change dumping

$dumpfile("fn") Name of dump file

$dumplimit(size) Max size of dump file

$dumpflush Flush dump file buffer

$dumpvars(levels [{, MODID | VARID}])

Variables to dump

$dumpall Force a dump now

$reset[(0)] Reset simulation to time 0

$reset(1) Reset and run again

$reset(0|1, expr) Reset with reset_value*$reset_value Reset_value of last $reset

$reset_count # of times $reset was used

KARIŞIK (MISCELLANEOUS)

$random[(ID)]

$getpattern(mem) Assign mem content

$rtoi(expr) Convert real to integer

$itor(expr) Convert integer to real

$realtobits(expr) Convert real to 64-bit vector

$bitstoreal(expr) Convert 64-bit vector to real

BİÇİM DİZGİLERİNDE ÇIKIŞ BÖLÜMÜ (ESCAPE SEQUENCES IN FORMAT STRINGS)

\n, \t, \\, \" newline, TAB, "\", """

\xxx character as octal value

%% character "%"

%[w.d]e, %[w.d]E display real in scientific form

%[w.d]f, %[w.d]F display real in decimal form

%[w.d]g, %[w.d]G display real in shortest form

%[0]h, %[0]H display in hexadecimal

%[0]d, %[0]D display in decimal

%[0]o, %[0]O display in octal

%[0]b, %[0]B display in binary

%[0]c, %[0]C display as ASCII character

%[0]v, %[0]V display net signal strength

%[0]s, %[0]S display as string

%[0]t, %[0]T display in current time format

%[0]m, %[0]M display hierarchical name

 **SÖZLÜKSEL ELEMANLAR (LEXICAL ELEMENTS)**

hierarchical identifier ::= {INSTID .} identifier

identifier ::= letter | _ { alphanumeric | $ | _}

escaped identifer ::= \ {nonwhite}

decimal literal ::=

[+|-]integer [. integer] [E|e[+|-] integer]

based literal ::= integer " base {hexdigit | x | z}

base ::= b | o | d | h

comment ::= // comment newline

comment block ::= /* comment */