İSTANBUL TEKNİK ÜNİVERSİTESİ

ELEKTRİK-ELEKTRONİK FAKÜLTESİ

BİR SAYISAL SİSTEM TASARIMININ EVRENSEL DOĞRULAMA METODU İLE DOĞRULANMASI

BİTİRME TEZİ

GÜLER ÇOKTAŞ

040090368

Bölümü: Elektronik ve Haberleşme Mühendisliği Bölümü

Programı: Elektronik Mühendisliği

Danışmanı: Doç. Dr. Sıddıka Berna ÖRS YALÇIN

MAYIS 2014

ÖNSÖZ

Öncelikle lisans eğitimim boyunca sahip oldukları değerli bilgilerini paylaşmayı benden esirgemeyen tüm öğretim üyelerine ve bitirme projemin her aşamasında bana rehberlik eden danışmanım Doç. Dr. Sıddıka Berna ÖRS YALÇIN'a teşekkürü bir borç bilirim.

Çalışmalarımda bana yardımcı olan, Dialog Semiconductor Şirketi'nden Yük. Müh. Şeyda Aygın ve Yük. Müh. Funda Kutay'a ayrıca teşekkürlerimi sunarım.

Son olarak, her daim yanımda olup her konuda bana destek verdikleri gibi bitirme projemi tamamlarken de destek olan sevgili aileme teşekkür ediyorum.

MAYIS 2014 Güler ÇOKTAŞ

İÇİNDEKİLER

KISALTMALAR

UVM : Universal Verification Methodology
TET : Test Edilen Tasarım
IVB : Incisive Verification Builder
IES : Incisive Enterprise Simulator
RTL : Register Transfer Logic
OVM : Open Verification Methodology
BCLs : Base Class Libraries
IEEE : Institute of Electrical and Electronics Engineers
VCS : Verilog Compiler Simulator
DVE : Discovery Visual Environment
TLM : Transaction Level Modeling

TABLO LİSTESİ

ŞEKİL LİSTESİ

BİR SAYISAL SİSTEM TASARIMININ EVRENSEL DOĞRULAMA METODU İLE DOĞRULANMASI

ÖZET

Günümüzde sayısal sistem tasarımları oldukça karmaşık hale gelmiştir. Bu sayısal sistem tasarımlarının karmaşıklığı arttıkça, tasarım sonucunda elde edilen sistemin doğru çalıştığını test etmek de zorlaşmaktadır.

Sayısal sistem tasarımlarının doğru çalışıp çalışmadığını test etmek için yaygın olarak kullanılan tasarım araçları çeşitli benzetim araçları da sağlamış durumdalar. Bu benzetim araçları ile davranışsal benzetim yaparak devrenin her hangi bir giriş sekansına doğru çıkış cevabı verip vermediği kolaylıkla test edilebilir. Gecikmelerin de hesaba katıldığı yerleştirme ve bağlantı sonrası benzetimi yapılarak bu idealsizlikler sonucunda sistemin hala doğru çalışıp çalışmadığı elde edilen dalga formu ile gözlemlenebilir. Fakat tasarım çok karmaşık ve büyük bir sistem olduğunda, zaten dalga formlarını incelemek bile oldukça zorlaşmaktadır.

Bir sayısal sistemin güvenilir bir şekilde doğrulanabilmesi için çeşitli doğrulama metotları geliştirilmiştir. Bu bitirme projesinde, bu doğrulama metotlarından Evrensel Doğrulama Metodu (Universal Verification Methodology – UVM) kullanılarak bir sayısal sistem tasarımının doğrulanması gerçekleştirilmiştir. Test edilmek üzere tasarlanan sayısal sistem tasarımı ise ardışıl toplayıcı devresidir.

Öncelikle, ardışıl toplayıcı devresinin benzetim araçları kullanılarak davranışsal benzetimi yapılmış, seçilen giriş sekansına doğru çıkışı ürettiği görülmüştür. Bunun ardından rastgele giriş işareti üretilerek bu giriş işaretlerine de vermesi gerektiği çıkışı ürettiği gözlemlenmiştir. Daha sonra UVM ile onun sağladığı temel sınıf kütüphaneleri kullanılarak doğrulama için gerekli kod blokları yazılmıştır. Bu kod blokları ile bir test ortamı oluşturularak ardışıl toplayıcı devresinin doğrulanması gerçekleştirilmiştir.

VERIFICATION OF A DIGITAL SYSTEM DESIGN WITH UNIVERSAL VERIFICATION METHODOLOGY

SUMMARY

Nowadays the digital system designs became quite complicate. As the complication of this digital system designs increases, it gets more difficult to test if the system works accurately.

The design tools that are in use to test if the digital system designs work accurately, are providing various simulation tools. It may be easily tested if the circuit gives the right output response to any input sequence, by making behavioral simulations with these simulation tools. It may be observed with the wave form if the system still works right as the result of these lack of ideal, by making post place and route simulation which is added with delays. But when the design is a very complicated and large system, it already become very difficult to examine even the wave forms.

There are various verification methods developed in order to a digital system may be verified reliably. In this graduation project, a digital system design verification is materialized by using Universal Verification Methodology (UVM). And the digital system design which planned to be tested is sequential adder circuit.

Primarily, behavioral simulation of the sequential adder circuit has been made by using simulation tools, and it has been observed that it produces the right output to input sequence which is chosen. Subsequently, a random input signal is generated and it has been observed that it provides right outputs to these input signals. Then the code blocks -which are needed for verification- are written by using UVM and the base class libraries that it provides. The verification of sequential adder circuit is materialized by constituting a test environment with these code blocks.

1. GİRİŞ

Sayısal sistem tasarımlarının gün geçtikçe gelişmesi ve karmaşık tasarımların gerçekleştirilmesi, bu tasarımların doğrulama problemini beraberinde getirmiştir ref. Bu bitirme projesi kapsamında bir sayısal sistem tasarımını doğrulamak için uzun hali UVM kullanılması tercih edilmiştir ref. Bunun sebebi, UVM kullanarak doğrulama yaparken tasarımcıya sağlamış olduğu avantajlar, güvenilirliği ve bunların yanında tümleşik devre tasarımlarının doğrulanmasında kullanılan bir standart haline gelmesidir. Günümüzde, sayısal sistem tasarımlarının doğrulanmasında oldukça rağbet gören UVM'nin nasıl kullanıldığını öğrenmek ve bir sayısal sistem üzerinde kullanmak amaçlanmaktadır.

UVM ile doğrulama işlemini gerçekleştirmeden önce elimizde test edilmesi gereken bir sayısal sistem tasarımı olmalıdır. Test edilecek tasarım (TET) Verilog, VHDL yada SystemVerilog donanım tanımlama dillerinden biri kullanılarak tasarlanmış olmalıdır. UVM temel sınıf kütüphanelerinin ve doğrulama esnasında yazılacak kodların SystemVerilog dilinde olması, UVM ile doğrulama yaparken SystemVerilog dilinin öğrenilmesini gerektirmektedir. Bu nedenle, bitirme projesi kapsamında TET, SystemVerilog ref donanım tanımlama dili ile yazılmıştır.

Bitirme projesinin asıl amacı UVM'nin öğrenilmesidir. Bu nedenle test edilmek üzere basit bir sayısal sistem tasarımı tercih edilmiştir. Bu basit sayısal sistem tasarımı ardışıl toplayıcı devresidir.

SystemVerilog dili ile tasarlanan ardışıl toplayıcı devresi, istenilen giriş değerleri tanımlanarak bir benzetim aracında (Simvision) ref davranışsal olarak test edilmiştir. Yapılan test işlemini bir adım daha ileriye taşımak için test kodunda rastgele girişler üretilerek de test işlemi tekrarlanmıştır. Ardışıl toplayıcının tasarımı ve davranışsal benzetimine ait bu adımlar beşinci bölümün ilk alt başlığında ayrıntılı olarak anlatılmıştır.

Bir sonraki aşamada, UVM temel sınıf kütüphanelerinden yararlanarak Keskin Doğrulama Geliştirici (–Incisive Verification Builder – IVB) ref aracı ile otomatik olarak

doğrulama için gerekli kodlar ürettirilmiştir. Üretilen bu kodlar, doğrulama ortamı için gerekli bazı kodların sadece temel satırlarını içermektedir. Doğrulamayı gerçekleştirebilmek için üretilmiş olan kodlara tasarıma bağlı olarak ek satırlar

eklenmesi ve UVM standardı için gerekli diğer kodların da yazılması gerekmiştir. Böylece ardışıl toplayıcı devresi için doğrulama ortamı oluşturulmuş ve Keskin İşletme Simülatörü (Incisive Enterprise Simulator – IES) ref ile ortamda tanımlanan test koşturulmuştur. Doğrulama işleminin tüm adımlarına, ayrıntılı olarak, beşinci bölümün ikinci alt başlığında yer verilmiştir.

2. DOĞRULAMA

Doğrulama, yapılan tasarımın istenildiği gibi çalıştığından emin olmak için yapılan işlemler bütünüdür. Bu işlemler, bir sayısal sistem doğrulanıyorsa eğer, test kodları yazmak, test ortamı oluşturmak ve testi bir benzetim aracı ile çalıştırmak şeklinde gerçekleştirilir ref.

Doğrulamanın amacını "hataları bulmak" olarak tanımlarsak, kısmen doğru bir cevap vermiş oluruz [1]. Donanım tasarımının amacı, belirli bir işi gerçekleştiren bir cihaz tasarlamaktır. Doğrulama mühendisinin amacı da cihazın bu işi başarılı bir şekilde yaptığından emin olmaktır. Ne zaman bir uyuşmazlık olursa o zaman hata elde edilir.

Doğrulama süreci tasarım ortaya koyma süreci ile paraleldir. Tasarımcı insan dili tanımı ile yorumlanmış bir blok için donanım tarifi okur, genellikle Kayıt Aktarma Dili (Register Transfer Language – RTL) ile, makinanın anlayabileceği ikili karşılığını yaratır. Doğrulama mühendisi de her zaman donanım tarifini okur, doğrulama planı oluşturur ve RTL kodun doğru çalıştığını gösterecek testler inşa eder.

Bir sayısal sistemin tasarımı ne kadar önemliyse, bu tasarımın doğru bir biçimde çalıştığından emin olunması ve güvenilirliği de büyük bir önem arz etmektedir. Bu nedenle sayısal sistem tasarımlarının doğrulanmasına ihtiyaç duyulması kaçınılmazdır.

3. UVM

UVM, tümleşik devre tasarımlarının doğrulanmasında kullanılan bir standart haline gelmiştir [2]. Temel olarak, Açık Doğrulama Metodu (Open Verification Methodology – OVM)'ndan türetilmiştir. UVM'yi oluşturmak ve Cadence Design Systems ile Mentor Graphics tarafından 2007'de geliştirilen OVM üzerine temellendirmek fikri, Accellera şirketinin bir teknik alt komisyonu tarafından, 2009'da ortaya atılmıştır. 2011 yılında ise yine Accellera şirketi tarafından UVM'nin ilk sürümü ortaya çıkartılmıştır.

UVM'den önce, bir sayısal sistemin doğrulanması için Specman e, Vera, SystemVerilog dili gibi çeşitli doğrulama dilleri ile yazılan test kodları ve OVM gibi doğrulama metotları kullanılmakta idi [3]. Fakat tasarlanan sayısal sistemlerin karmaşıklığı arttıkça, bu metotlar ile sayısal sistemlerin doğrulanması gittikçe daha zor bir hal almıştır. UVM'nin sunduğu SystemVerilog dilinde yazılmış Temel Sınıf Kütüphaneleri (Base Class Libraries – BCLs) sayesinde, sayısal sistemlerin doğrulanması kullanıcılar açısından derli toplu bir hal almış ve kullanıcıların isteklerini karşılayabilecek bir niteliğe ulaşmıştır.

SystemVerilog dili kullanılarak istenildiği gibi doğrulama ortamı (verification environment) oluşturmak oldukça basittir. Fakat kullanıcılar bu doğrulama ortamı üzerinde değişiklikler yaparak tekrar kullanmak isterler. Bu nedenle yeniden kullanılabilirlik oldukça önemlidir. BCLs, kullanıcılara bir dizi ortak mimari ve bağlantı standartları sağlar. SystemVerilog ile doğrulama metodolojisi seçiminde kullanıcılar, seçmiş oldukları aracın satıcılarına bağlı kalmaktadır. Bu durum kullanıcıların kısıtlanması anlamına geldiği için sorun oluşturmaktadır.

UVM, tüm ileri gelen araç satıcılarının kullandığı açık standart bir metodoloji haline gelmiştir [4]. UVM'de var olan sınıf hiyerarşisi sayesinde tüm bileşenler ve onların alt bileşenlerine birer sorumluluk düşmektedir. Yazılan kodda bir değişim yapılması

gerektiğinde, tüm bileşenlerin tanımlanması ayrı ayrı yapıldığından, yalnızca ilgili bileşen değiştirilerek istenen sonuç kolaylıkla alınabilmektedir.

4. SYSTEMVERILOG DONANIM TANIMLAMA–DOĞRULAMA DİLİ VE TASARIM ARAÇLARI

4.1. SystemVerilog Donanım Tanımlama-Doğrulama Dili

SystemVerilog, yarı iletken ve elektronik tasarım endüstrisinde, Verilog donanım tanımlama dilinin gelişimi üzerine temellendirilmiştir ve donanım tanımlama dili ile donanım doğrulama dilinin birleşimi halindedir [5]. SystemVerilog, 2005'te Elektrik ve Elektronik Mühendisleri Enstitüsü (Institute of Electrical and Electronics Engineers – IEEE) tarafından benimsenmiştir. Daha sonra geliştirilmiştir ve şu an IEEE 1800-2012 standardı kullanılmaktadır. SystemVerilog dilini destekleyen tasarım araçları Cadence IES, Synopsys Verilog Derleyici Simülatör (Verilog Compiler Simulator – VCS) ve Mentor Graphics – QuestaSim araçlarıdır.

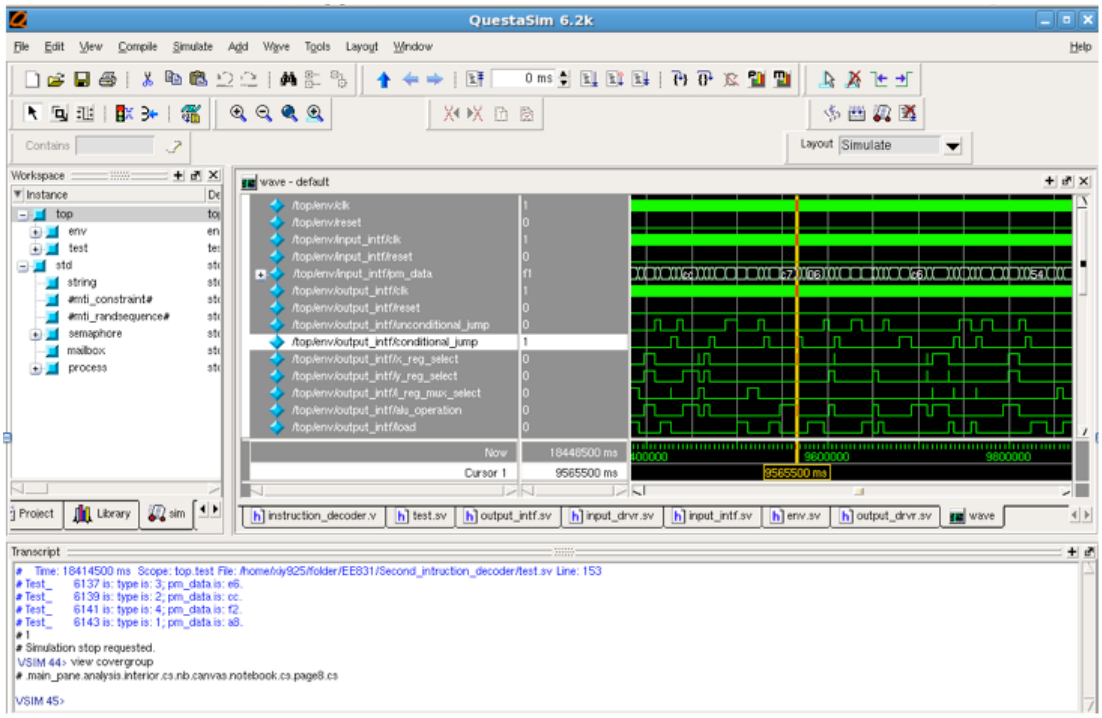
SystemVerilog donanım tanımlama – doğrulama dilinin ilk sürümü davranışsal ve RTL'den kapı seviyesine kadar donanım tasarımı içermekteydi [6]. Daha sonra tasarımcıların istedikleri ilave yapılar eklenmiştir. C programlama diline benzer data tipleri, döngüler ve operatörler eklenmiştir. Son olarak ortaya tüm tasarım ve doğrulama özelliklerini içeren tek bir birleşik dil ortaya çıkmıştır.

4.2. Tasarım Araçları

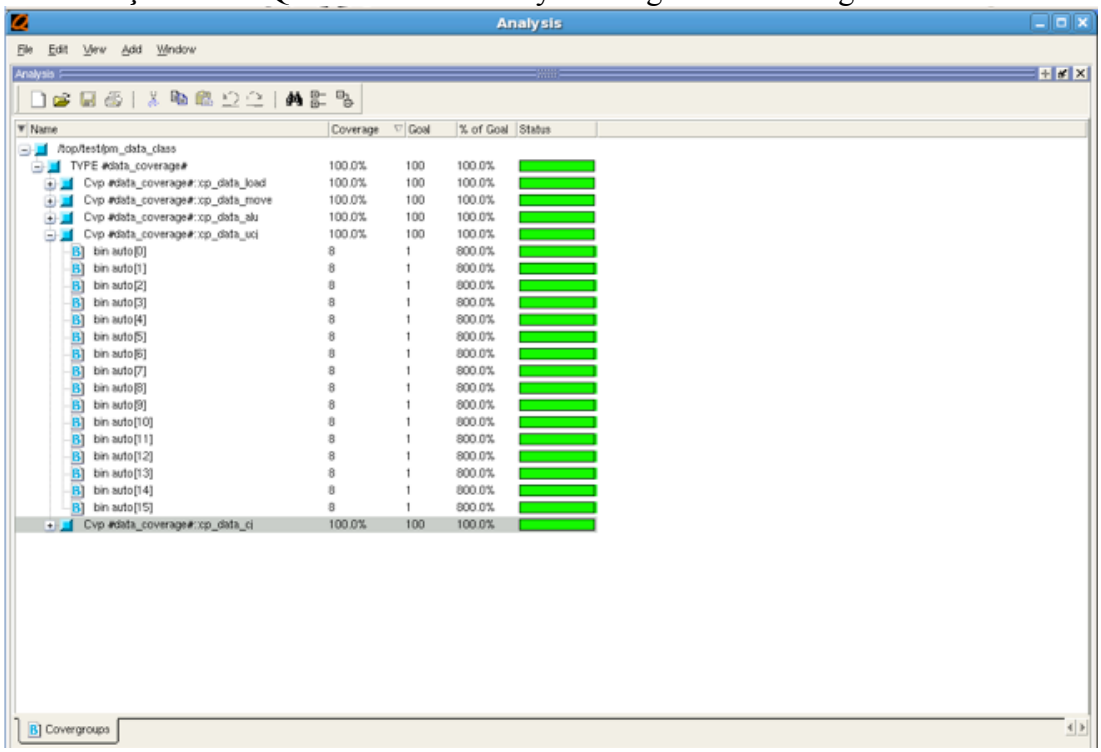
4.2.1. QuestaSim

Questa, Mentor Graphics'e ait gelişmiş fonksiyonel doğrulama aracıdır ve QuestaSim adında tümleşik bir platform içermektedir [7]. QuestaSim, büyük elektronik sistemlerin yüksek verimlilikle gelişmiş doğrulamasını gerçekleştirebilir, yapısında gömülü olarak yönetme ve hataları ayıklama olanakları barındırır. Yine Mentor Graphics'e ait olan ModelSim aracı üzerinde temellendirilmiştir. ModelSim ile QuestaSim arasındaki en önemli fark şudur: QuestaSim doğrulama yönetimi ile uyuşum, şekilsel tabanlı teknolojiler, düşük güçlü benzetim vs. gibi olanaklara sahip olan Questa platformunun benzetim motorudur[8]. Doğal olarak QuestaSim, UVM

için SystemVerilog donanım tanımlama – doğrulama dilini destekler, ModelSim ise desteklemez.



Şekil 4.1 : QuestaSim ile fonksiyonel doğrulama ekran görüntüsü.

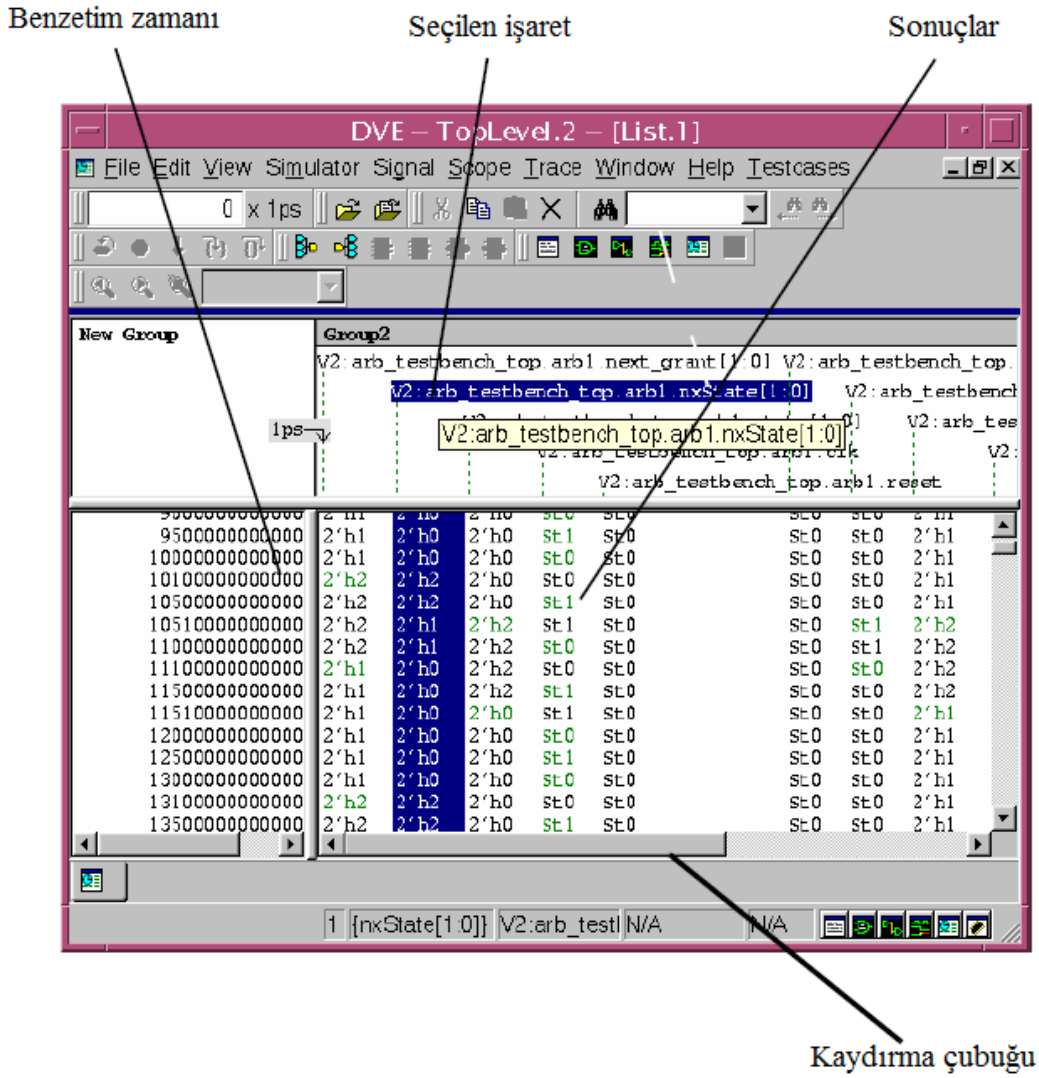


Şekil 4.2 : QuestaSim ile analiz ekran görüntüsü.

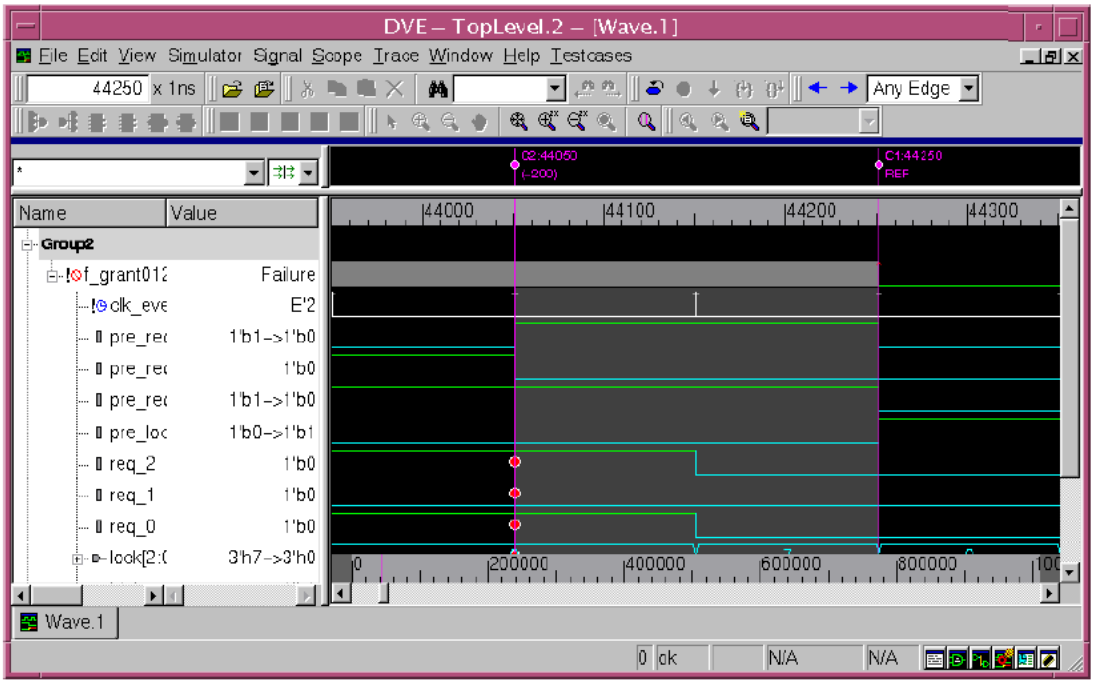
4.2.2. VCS

VCS, bugün sektöründe öncü tasarımcıların gelişmiş tasarımlarına ait doğrulama ortamları için fonksiyonel doğrulama çözümleri ürettikleri bir tasarım aracıdır[9]. Gömülü derleme ve görüntüleme ortamı sayesinde, popüler olan bütün tasarım ve doğrulama dillerini destekler. Bunlara UVM de dahildir.

VCS, Görsel Keşif Ortamı (Discovery Visual Environment – DVE) içerir. DVE yazılımı, VCS benzetimlerini destekleyen grafiksel bir doğrulama ortamıdır [10].



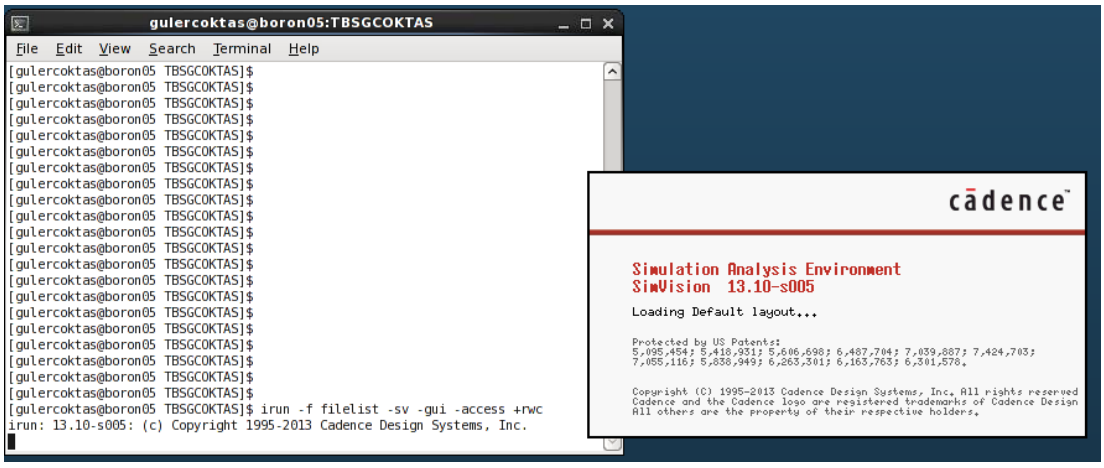
Şekil 4.3 : DVE Ortamı liste penceresi.



Şekil 4.4 : DVE Ortamında doğrulama ekran görüntüsü.

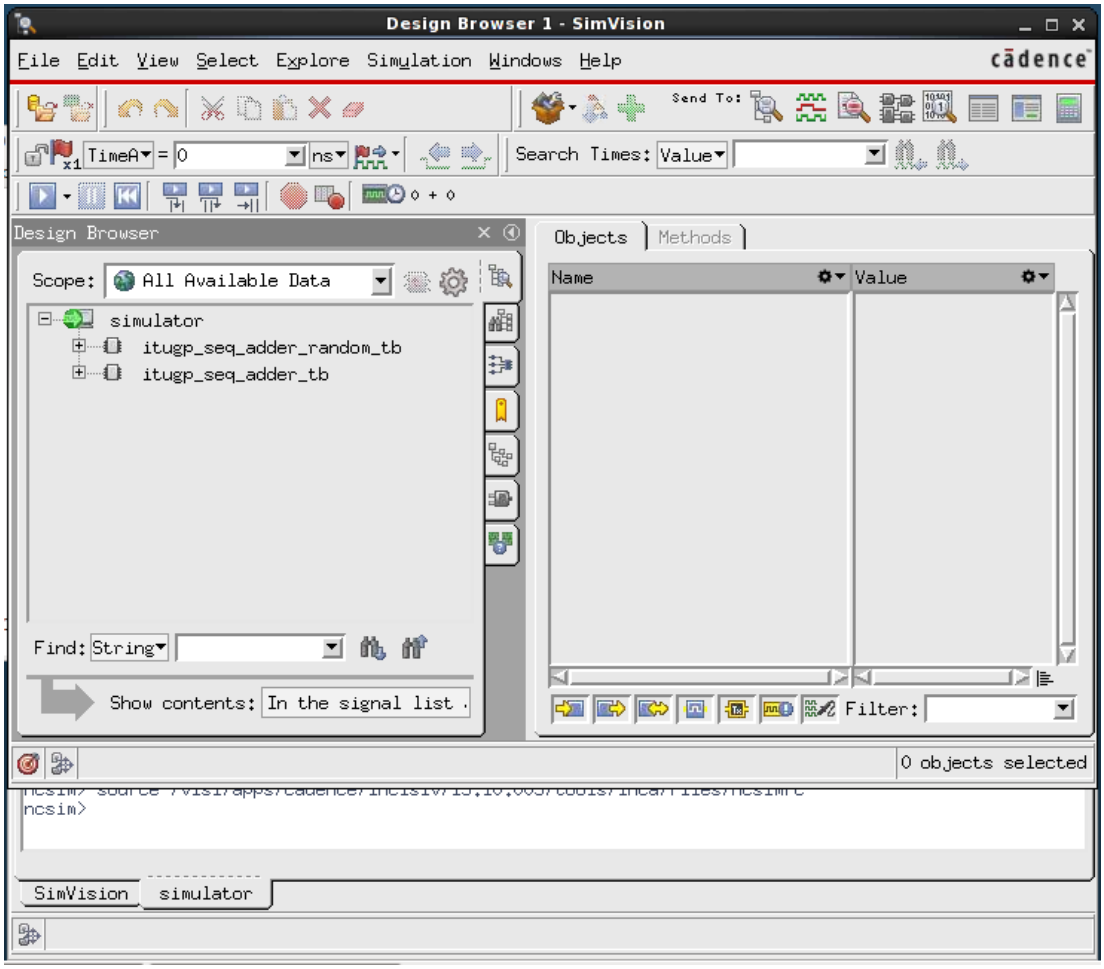
4.2.3. IES ve IVB

IES, Cadence firmasının bir tasarım aracıdır. Bugün IES, RTL yolu ile, sistem seviyesinden kapı seviyesine kadar olan tasarımları doğrulamak için analiz yapma, yeniden kullanma olanağı ve test düzeneği otomasyonu sağlar [11]. UVM de dahil olmak üzere birçok sınıf kütüphanesini ve bunun yanında birçok donanım tanımlama – doğrulama dillerini destekler. İşaret seviyesi ve işlem seviyesi akışları destekleyen, SimVision adında, birleşik grafiksel derleme ortamına sahiptir.



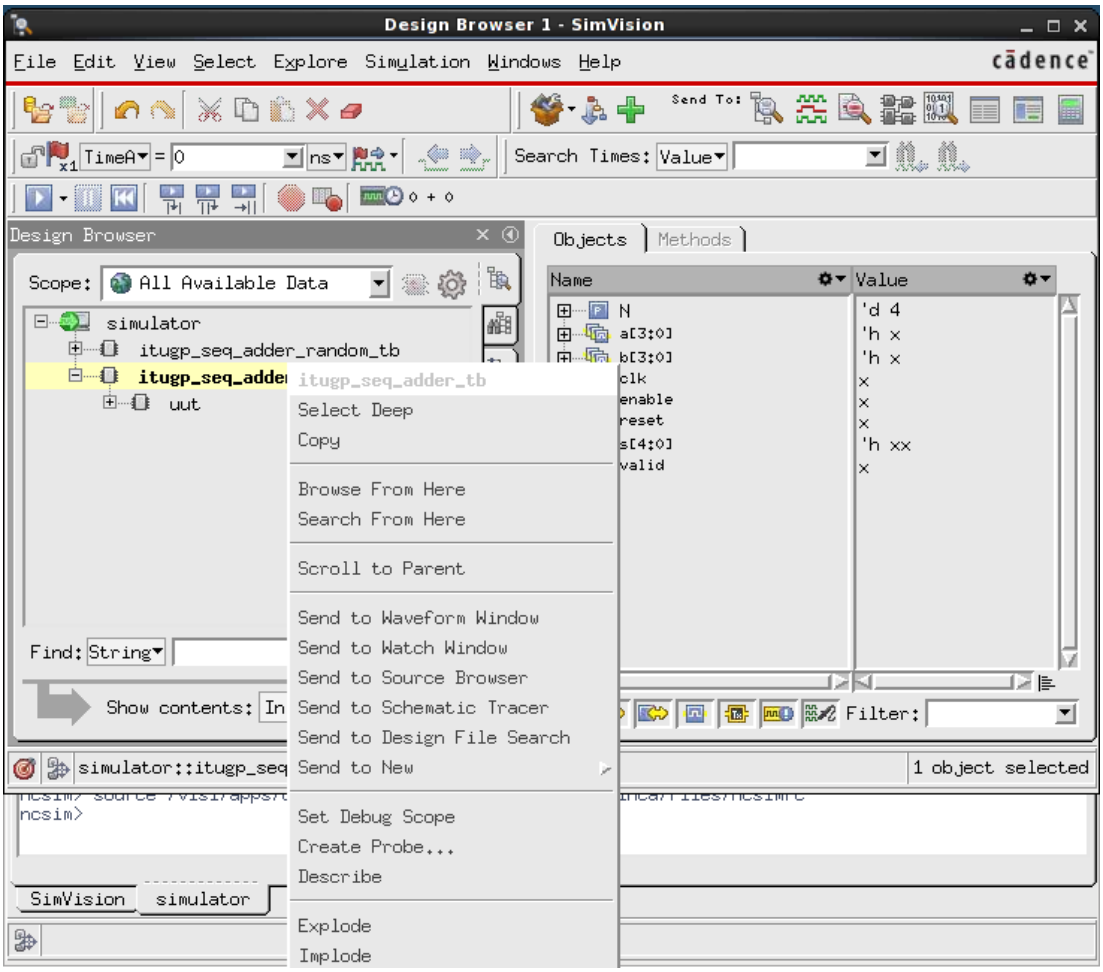
Şekil 4.5 : Terminalden SimVision Ortamına geçiş.

SimVision aracını çalıştırabilmek için önceden tasarıma ait kaynak kodu ve test kodu yazılır, kodlar bir dosyada listelenir. Bu liste, Şekil 4.5'da görüldüğü, gibi terminale yazılan komut ile çalıştırılır ve SimVision aracı açılır.

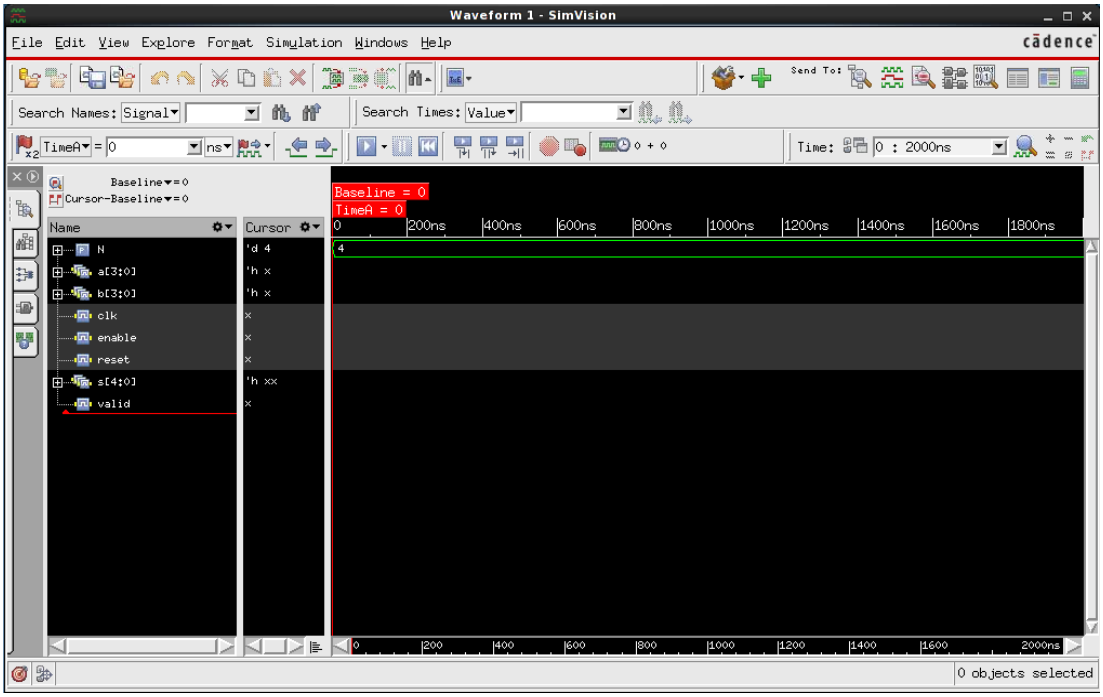


Şekil 4.6 : SimVision Ortamının ekran görüntüsü.

Yazılmış olan kaynak ve test kodları tasarım listesinde görülmektedir. Benzetimi yapılacak olan test kodu seçilerek dalga formu gözlemlenir.



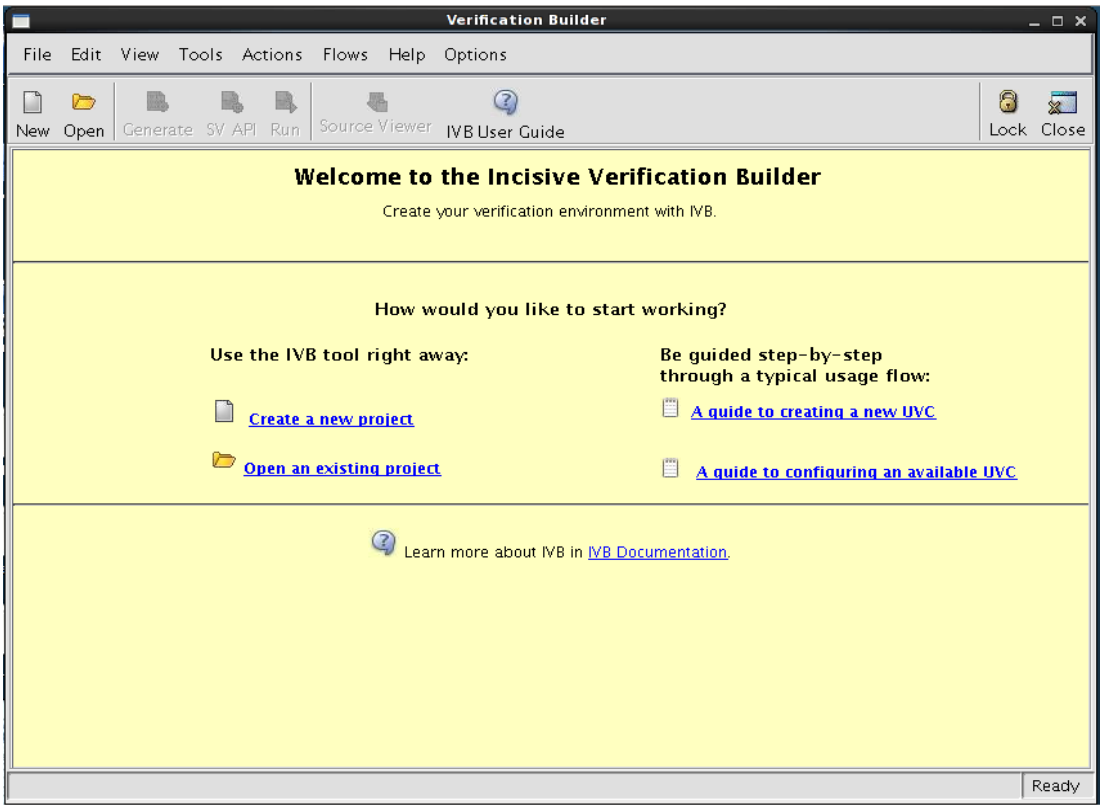
Şekil 4.7 : SimVision Ortamında benzetim adımları.



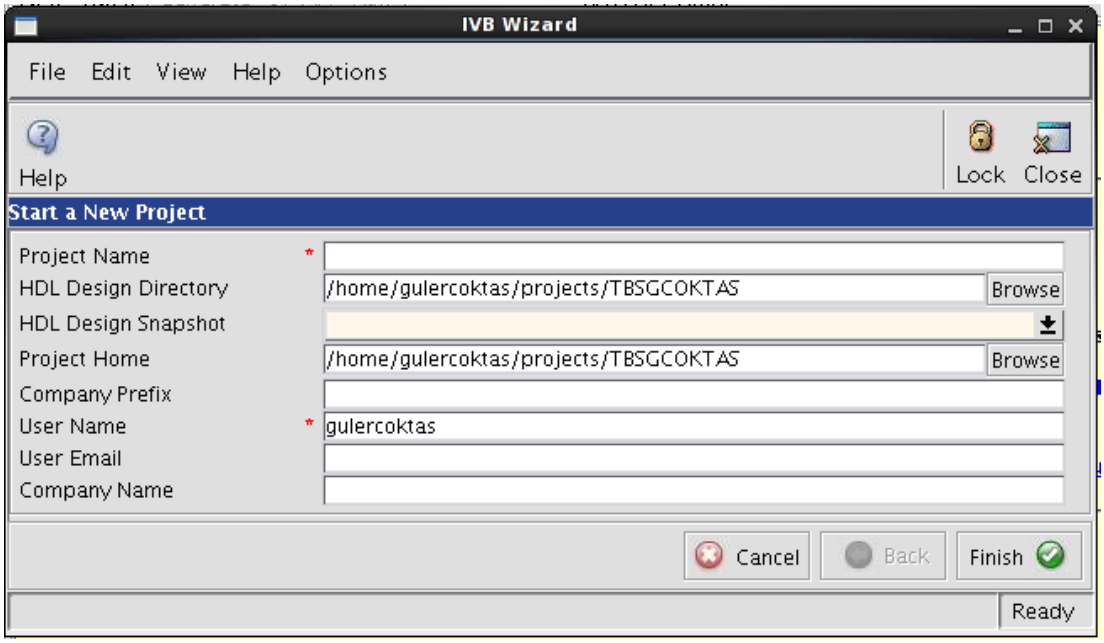
Şekil 4.8 : SimVision Ortamında benzetim dalga formu ekran görüntüsü.

Cadence, doğrulama ortamı oluşturmak için ayrıca bir Keskin Doğrulama İnşa Edici IVB tasarım aracını barındırır. Bu bitirme projesinde, SimVision ortamı ve IVB tasarım aracı kullanıldığından, bunlara ayrıntılı olarak yer verilmiştir.

IVB aracı yine terminalden verilen komutla açılır. Yeni bir proje oluşturulur.

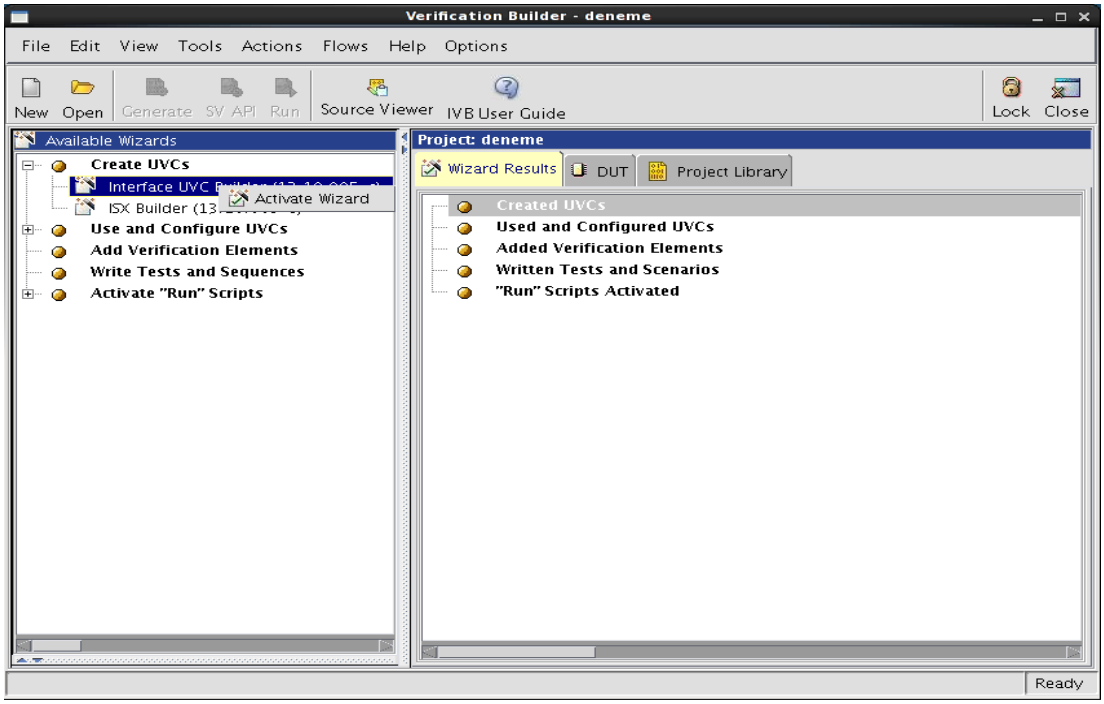


Şekil 4.9 : IVB Aracı başlangıç sayfası.



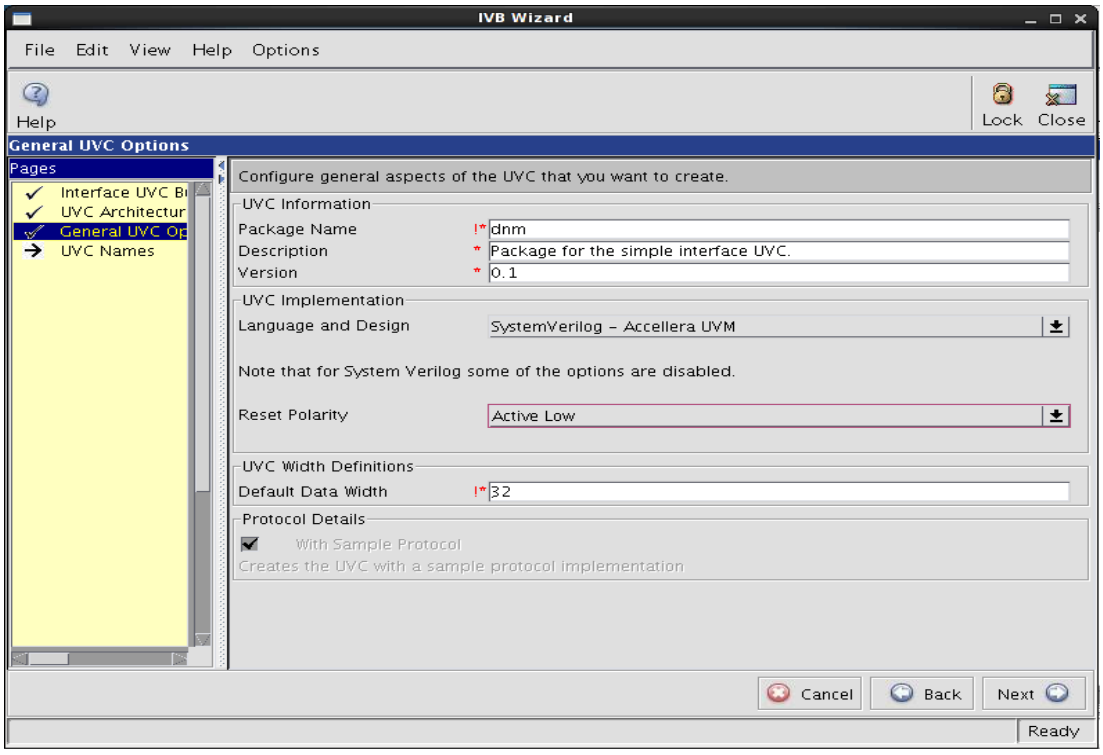
Şekil 4.10 : IVB Aracında yeni proje oluşturma.

Proje oluşturulduktan sonra, UVM doğrulama elemanları ile arayüz oluşturma sihirbazı aktif hale getirilir.



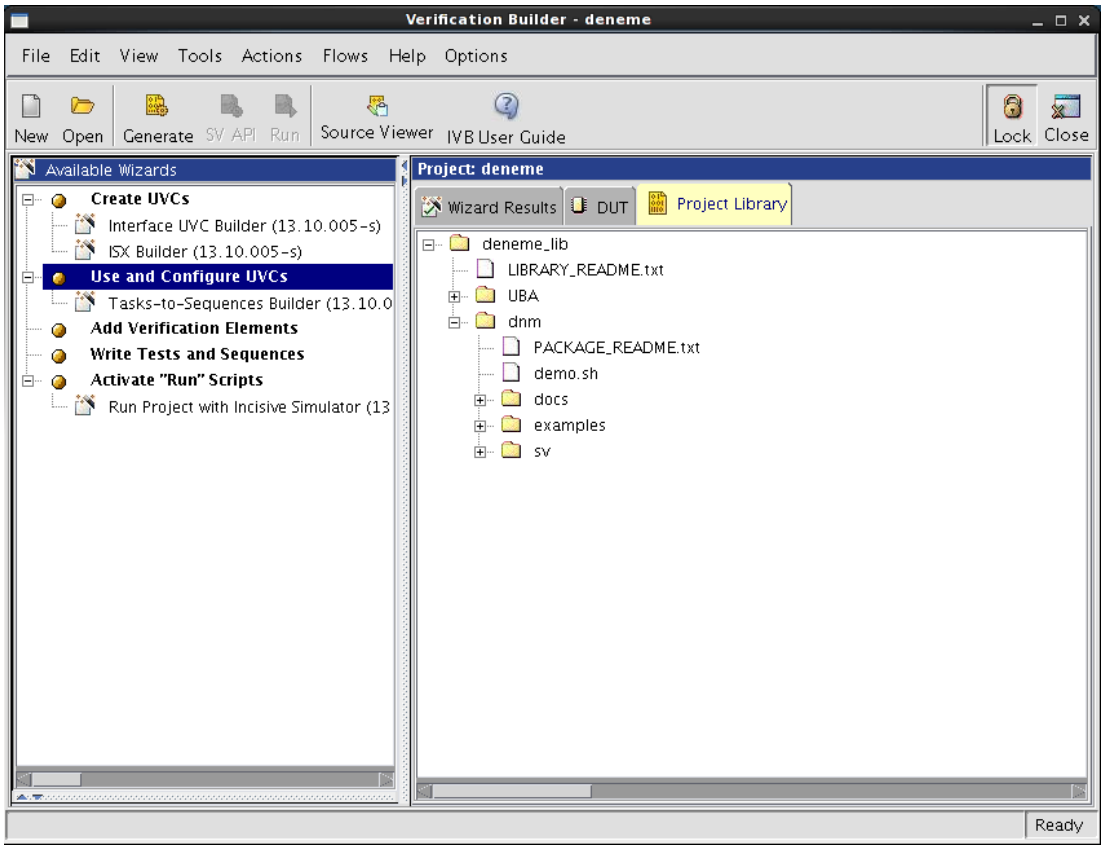
Şekil 4.11 : IVB Aracında doğrulama elemanlarını otomatik üretme adımı 1.

Doğrulama için dil ve tasarım seçimi yapılır. Bu proje için seçilen dil SystemVerilog ve kullanılacak kütüphane UVM olacaktır.



Şekil 4.12 : IVB Aracında doğrulama elemanlarını otomatik üretme adımı 2.

Taslak halinde üretilen kodlar bir dizin yapısı ile birlikte gelmektedir. Otomatik olarak IVB aracı tarafından üretilen bu kodlar içerisinde tüm UVM elemanları yer almamaktadır. Gereken elemanlar tasarımcı tarafından eklenmeli ve üretilen kodlar da tasarıma bağlı olarak düzenlenmelidir. IVB aracının oluşturduğu doğrulama dizininde, doğrulama ortamının dışında, benzetimi çalıştıracak bir komut dosyası ve örnek test düzeneği kodları da yer almaktadır.



Şekil 4.13 : IVB Aracının ürettiği doğrulama dizini.

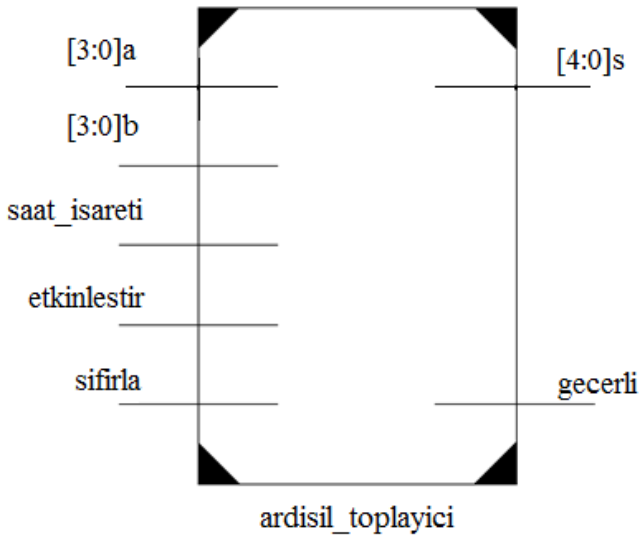
5. ARDIŞIL TOPLAYICININ DOĞRULANMASI

5.1. Ardışıl Toplayıcı

Bitirme projesinin asıl amacı UVM ile doğrulamayı öğrenmek olduğundan, doğrulanacak tasarımın basit bir sayısal sistem olması uygun görülmüştür. Bu nedenle ardışıl bir toplayıcı devresi tasarlanmıştır. Donanım tasarlama dili olarak SystemVerilog tercih edilmiştir.

5.1.1. Ardışıl Toplayıcı Tasarımı

Ardışıl toplayıcı devresi iki tane dört bitlik sayıyı toplama işlemini gerçekleştirir. Ardışıl devre olduğundan bir adet saat işareti girişi vardır ve devre bu saat işaretinin yükselen kenarında tetiklenmektedir. Devrede senkron sıfırlama yapılır. Sıfırla girişi aktif sıfır olarak çalışmaktadır. Yani sıfırla işareti lojik bir olunca, bir sonraki saat işareti geldiğinde devreyi sıfırlayacaktır. Bunlara ek olarak etkinleştir girişi vardır. Sıfırlama olmadığı durumda, etkinleştir lojik bir seviyesindeyse, saat işareti geldiğinde iki girişi toplayacaktır. Üç saat periyodu kadar gecikme verilmiştir devreye ve toplama sonucu beş bitlik çıkışa üç saat periyodu sonunda aktarılır.

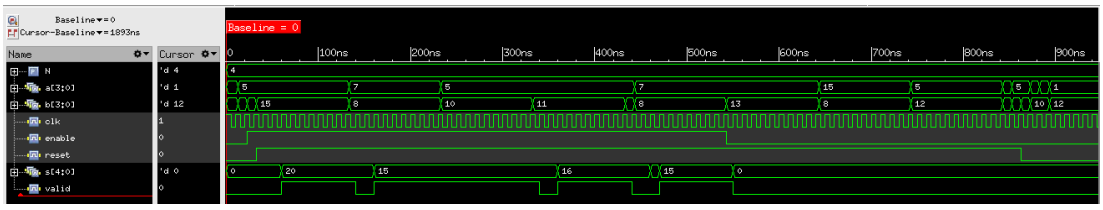


Şekil 5.1 : Ardışıl toplayıcı modülü.

Şekil 5.1'de yer alan ardışıl toplayıcı modülüne ait şematikte giriş ve çıkışlar görülmektedir. Geçerli çıkışı, çıkışın doğru olduğu anlarda lojik bir seviyesinde, çıkış yanlış bir değeri gösteriyorsa veya devre sıfırlama durumundaysa lojik sıfır seviyesinde olacak şekilde tanımlanmıştır. Çıkışın yanlış değeri göstermesi demek, giriş işaretlerinin toplam sonucu oluştuktan sonra, bu sonuç gecikme süresinin bitmesiyle çıkışa aktarılmadan giriş işaretlerinin değişmesi durumunu ifade etmektedir. Bu geçerli çıkışı doğrulama aşamasında gerekli olacağı için eklenmiştir. Ardışıl toplayıcı devresine ait kaynak kodu EK A.1'de yer almaktadır.

5.1.2. Ardışıl Toplayıcının Davranışsal Benzetimi

Ardışıl toplayıcı devresinin girişlerine istenilen değerlerin girildiği bir test kodu yazılarak, bu durumda devrenin doğru çıkışı üretip üretmediğini görmek amacıyla davranışsal benzetim yapılmıştır. Bu benzetim SimVision ortamında gerçekleştirilmiştir.

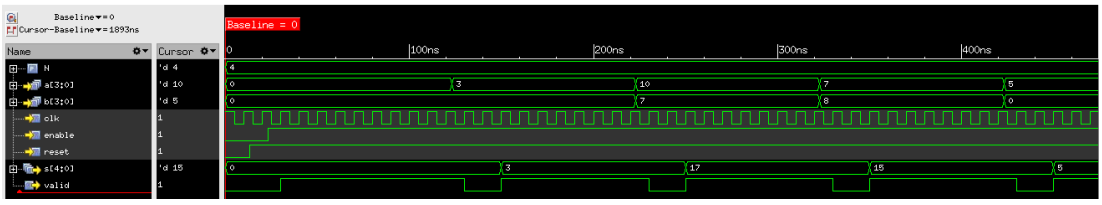


Şekil 5.2 : Ardışıl toplayıcının davranışsal benzetimi.

Şekil 5.2'de görüldüğü gibi, devre davranışsal olarak doğru çalışmaktadır. Bu benzetimi yaparken yazılan test kodu Ek A.2'de yer almaktadır.

5.1.3. Rastgele Giriş İşaretleri ile Benzetim

Yapılan davranışsal benzetimi bir adım ileriye götürmek amacıyla girişlerin tasarımcı tarafından tanımlanmadığı bir benzetim yapılmıştır. Bu benzetim için ayrı bir test kodu yazılarak girişler için rastgele değerler üretilmiştir. Rastgele girişler ile yapılan benzetim sonucu Şekil 5.3'te görüldüğü gibidir.



Şekil 5.3 : Rastgele giriş işaretleri ile benzetim.

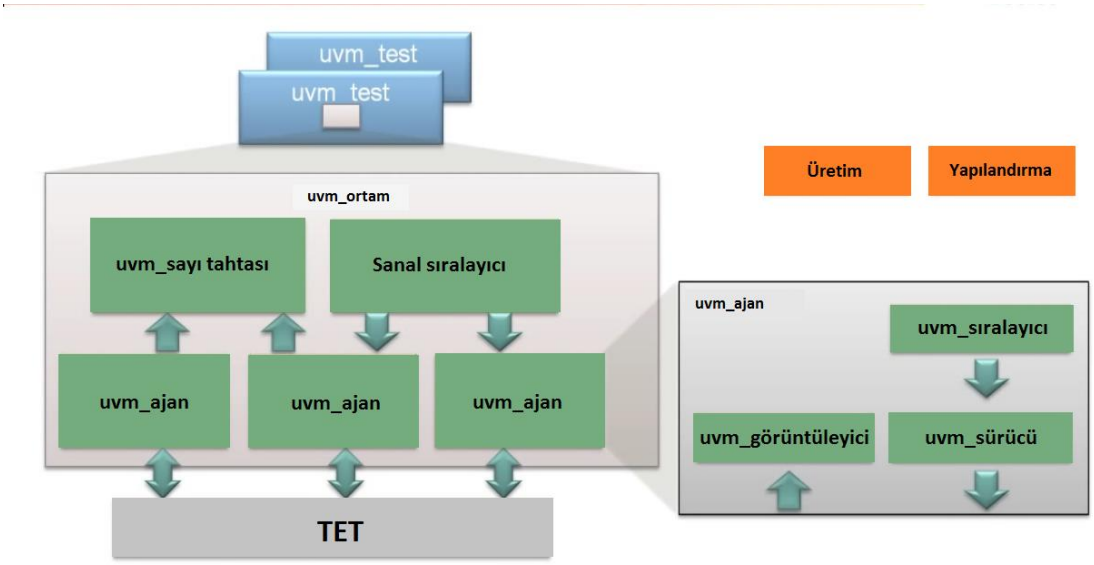
Rastgele giriş işaretlerini üretmek için bir döngü tanımlanarak döngünün her adımında a ve b girişlerine rastgele sayı atanması sağlanmıştır. Benzetim için yazılan koda Ek A.3'te yer verilmiştir.

5.2. Ardışıl Toplayıcının UVM ile Doğrulanması

5.2.1. Genel Bakış

UVM ile bir sayısal sistemin doğrulanabilmesi için bir doğrulama ortamı oluşturulur. Bir test düzeneği içerisinde, oluşturulan bu doğrulama ortamını barındırır. Doğrulama ortamı ile TET'in haberleşebilmesi için bir test üst modülüne bir de arayüze ihtiyaç vardır. Şekil 5.4'te yer alan UVM genel resminde TET, doğrulama ortamı ve test düzeneği görülmektedir. Test üst modülüne ve arayüze ileride değinilecektir.

Tasarımcının isteğine bağlı olarak test sayısı artırılabilir. Aynı doğrulama ortamı kullanılarak, TET farklı testler ile doğrulanabilir.



Şekil 5.4 : UVM genel resim [12].

UVM genel resminde yer alan bileşenler için Tablo 5.1'de tanımlamalar yapılmıştır.

Tablo 5.1 : UVM genel resme ait tanımlar.

İsim	Tanımlama
TET	Test edilecek tasarım.
uvm_test	Test düzeneği, tasarıma göre yapılandırılabilen kısımdır.
uvm_ortam	Sabit kısımdır, TET ile haberleşir.
uvm_ajan	Doğrulama elemanıdır, uvm ortam içinde yer alır.
uvm_sürücü	Uvm ajan içinde yer alır, TET'i sürer.
uvm_görüntüleyici	Uvm ajan içinde yer alır, TET'i görüntüler.
uvm_sıralayıcı	Uvm ajan içinde yer alır, uvm sürücü için sıra öğresi üretir.
sanal sıralayıcı	Doğrulama elemanlarının haberleşmesinde kullanılır.
uvm_sayı tahtası	TET'in doğru çalışıp çalışmadığını kontrol eder.
üretim	Uvm nesnelerini ve elemanlarını üretir.
yapılandırma	Uvm ortamını ve elemanlarını yapılandırır.

5.2.2. Sınıflar

5.2.2.1. Temel Sınıflar

UVM, içerisinde BCLs barındıran bir pakettir[13]. Bu paket, üç ana sınıf çeşidine sahiptir:

- UVM Nesne Sınıfı (uvm_nesne)
- UVM Eleman Sınıfı (uvm_eleman)
- UVM İşlem Sınıfı (uvm_işlem)

UVM nesne sınıfı, bütün UVM verileri ile hiyerarşik sınıflar için bir temel sınıftır [14]. Elemanlar ve işlemlerin tamamı uvm_nesne'den türetilmiştir. Uvm_nesne'nin başlıca görevi; oluştur, kopyala, karşılaştır, yazdır, kaydet gibi yaygın olan işlemler için bir takım metotlar tanımlamasıdır. UVM nesne sınıfından türeyen sınıflar;

oluştur (create) ve tür adı elde et (get_type_name) gibi tamamen sanal metotları uygulamak zorundadırlar.

UVM eleman sınıfı, sınıf tabanlı bir hiyerarşik test düzeneği yapısı kurmak için kullanılır. Tüm uvm_elemanların temelinde bu sınıf vardır. Elemanlar, benzetim süresince var olan yarı-statik nesnelerdir. Bu nedenle modül ve program blokları gibi hiyerarşik yapılar kurmalarına izin verilir. Her eleman, hiyerarşik yol ismi ile beraber, eşi olmayan bir adrese sahiptir. UVM eleman sınıfından doğrulama elemanları türetilir; uvm_ortam, uvm_ajan, uvm_görüntüleyici vs. gibi. UVM eleman sınıfının fazlara ayırma, yapılandırma, raporlama, üretim, işlem kaydı gibi avantajları da vardır.

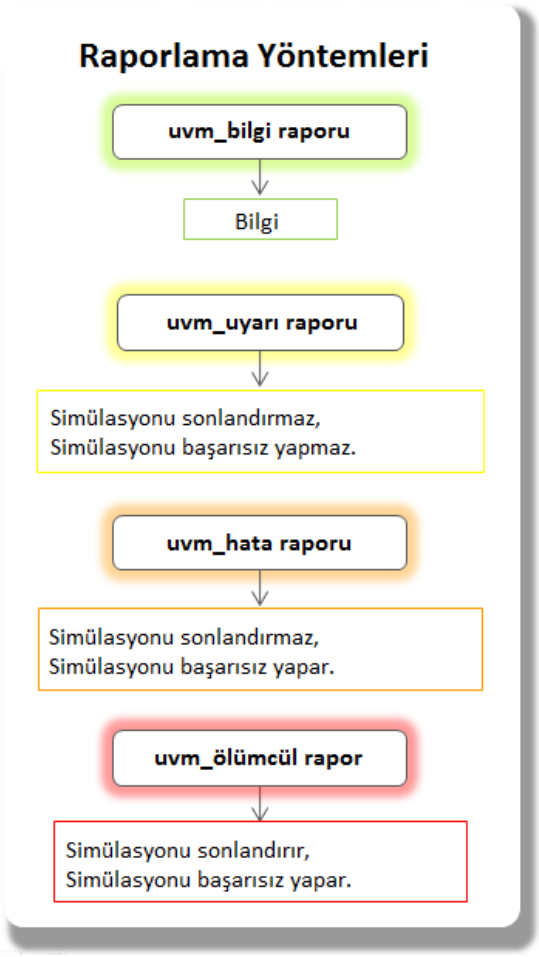
UVM işlem sınıfı, uyarıcı üretme ve analiz etmede kullanılır. uvm_elemanların aksine, uvm_işlemler doğal olarak geçicilerdir. Zamanlama ve kaydedici arayüzlerine sahiptirler. Basit işlemler doğrudan uvm_işlemler'den türetilir, fakat sıra-izinli işlemler uvm_sıra öğesinden türetilir.

5.2.2.2. Rapor Sınıfları

UVM rapor nesnesi, UVM raporlama olanakları ile bir arayüz oluşturur [14]. Bu arayüz sayesinde, benzetim süresince elemanlar çeşitli mesajlar verir.

UVM, tasarımcının yapılandırabilmesi için, uyumlu biçimlendirme ile gömülü olarak raporlama olanağı sağlar. Şekil 5.5'ten de anlaşılacağı üzere, raporlama yöntemleri, raporların önem derecesi ve gereksizliği ile alakalıdır.

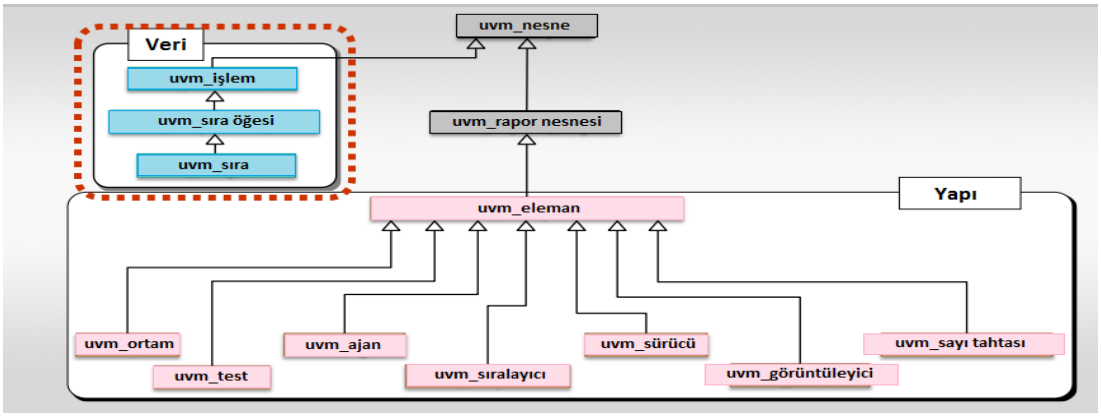
Doğrulama yaparken yazılan kodların uygun yerlerine uvm_bilgi_raporu eklenerek kolaylıkla derleme yapılabilir. Raporlama yöntemleri etkili bir biçimde kullanıldığı taktirde, kod yazma aşamasında hataları ayıklamak oldukça basit bir hal alır.



Şekil 5.5 : Raporlama yöntemleri [15].

5.2.2.3. Sınıf Hiyerarşisi

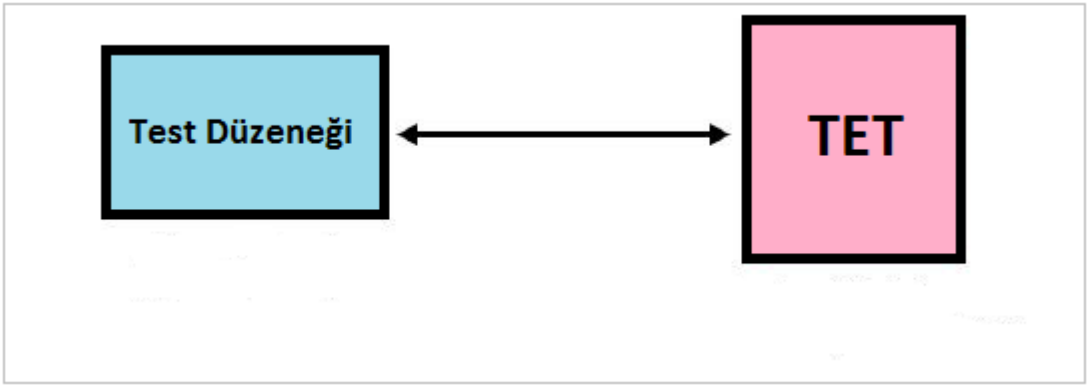
UVM'de var olan sınıf hiyerarşisinde uvm_ortam, uvm_ajan, uvm_sıralayıcı, uvm_sürücü, uvm_görüntüleyici ve uvm_sayı tahtası birer uvm_elemandır. Şekil 5.6'da yer alan UVM sınıf hiyerarşisinin şematiğinde görüldüğü gibi, uvm_eleman, uvm_rapor nesnesinden, o da uvm_nesneden miras kalmıştır.



Şekil 5.6 : UVM sınıf hiyerarşisi [16].

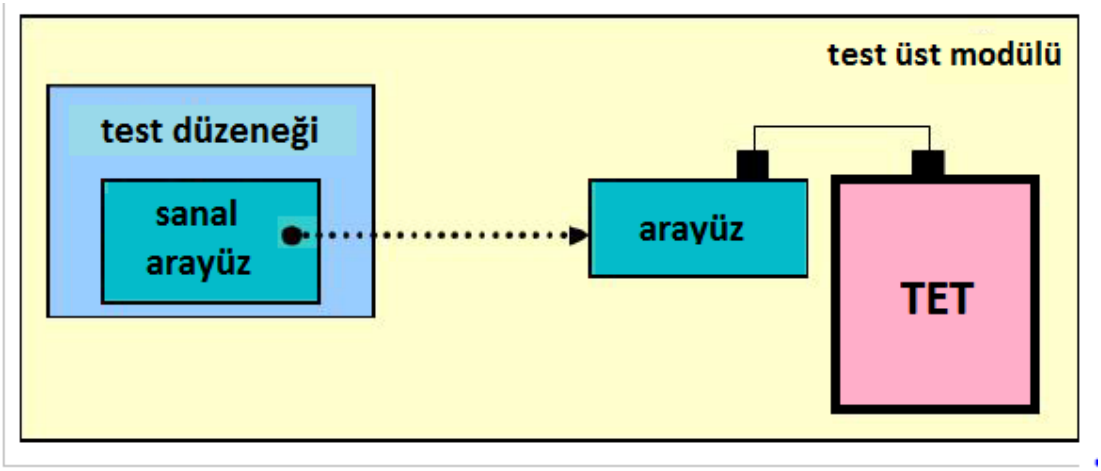
5.2.3. Test Üst Modülü

TET ile test düzeneği arasındaki ilişki en basit haliyle Şekil 5.7'de yer almaktadır.



Şekil 5.7 : Test düzeneği – TET ilişkisi [13].

Test düzeneği içerisinde yer alan doğrulama ortamı ile TET arasında bir arayüze ihtiyaç vardır. Tüm bunları içinde barındıran, Şekil 5.8'de görüldüğü gibi, bir de test üst modülü olmalıdır.



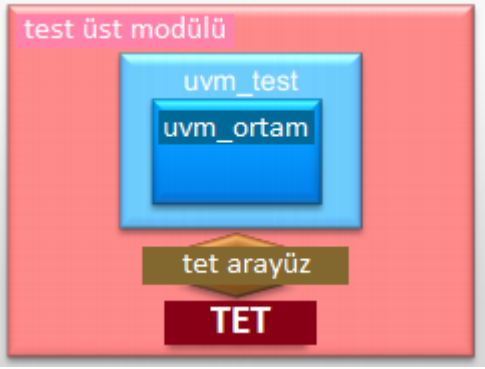
Şekil 5.8 : Test üst modülü[13].

Test üst modülünde TET, TET ile test düzeneği arasındaki arayüz ve test düzeneği yer almaktadır. Ardışıl toplayıcı devresinin test üst modülünde, Ek B.1'de görüldüğü gibi, bunlara ek olarak saat işareti ve sıfırla işaretleri üretilmektedir. Test üst modülü içerisinde test düzeneğinin de yer alması için, test üst modülüne ardışıl toplayıcı devresinin doğrulama paketi de aktarılmıştır. Ardışıl toplayıcı devresinin doğrulama paketi Ek B.2'de yer almaktadır.

5.2.4. Arayüz

TET ile doğrulama ortamının haberleşmesini sağlayan bir arayüz mevcuttur. Bu haberleşme, test üst modülünde gerçekleşmektedir. Şekil 5.9'da TET arayüzü görülmektedir. TET arayüzünde, TET giriş ve çıkışları tanımlanmaktadır.

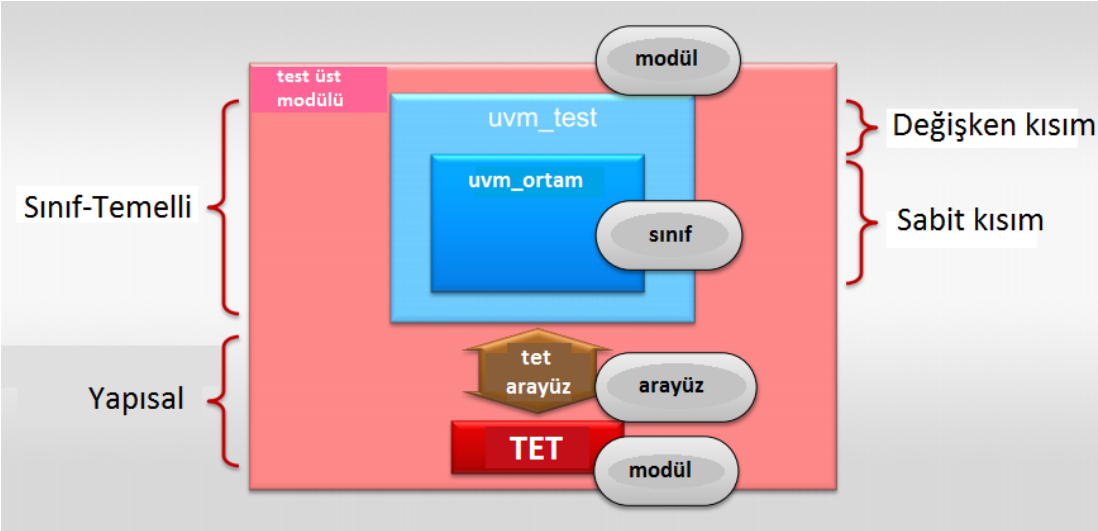
Ardışıl toplayıcı devresinin arayüzüne EK C'de yer verilmiştir.



Şekil 5.9 : TET arayüzü [17].

5.2.5. Elemanlar

5.2.5.1. UVM Ortam ve UVM Test Düzeneği



Şekil 5.10 : UVM test düzeneği ve uvm_ortam [18].

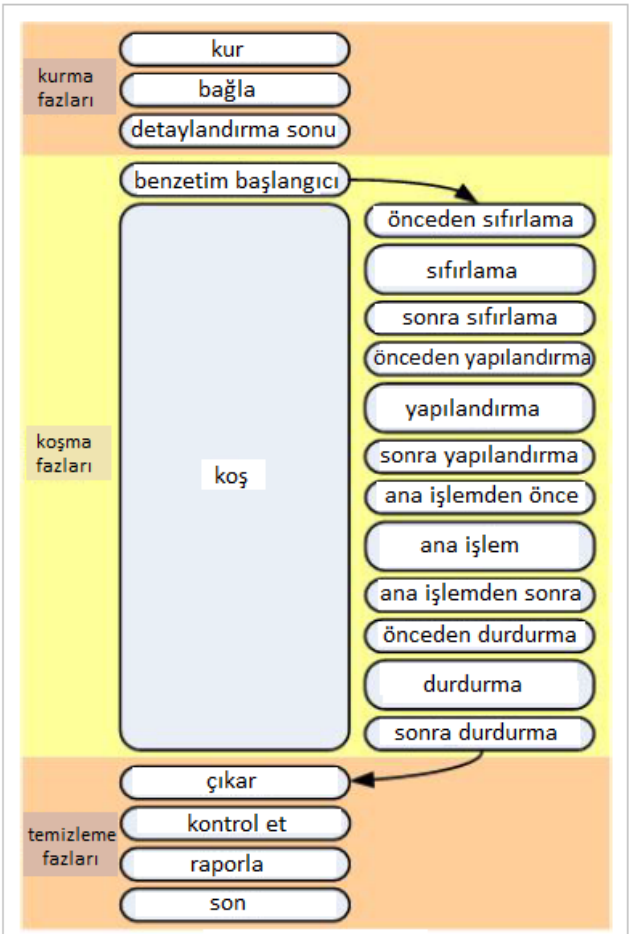
UVM test düzeneği, uvm_eleman temel sınıfından türemiş sınıflardan oluşur [13]. Test düzeneği hiyerarşisi, elemanlar içinde elemanlar tarafından tanımlanır. Başka bir deyişle, iç içe yer alan sınıf ilişkileri serisinden oluşur. UVM test düzeneği mimarisi, aynı projede daha yüksek düzeylere katılan (dikey yeniden kullanım) veya farklı projelerde yer alan (yatay yeniden kullanım) doğrulama elemanları gruplarının

yeniden kullanılma olanağına sahip olduğu modüllü bir yapıdır. Bu nedenle uvm_test, tasarıma göre yapılandırılabilir, yani değişken, kısım olmaktadır. Uvm_ortam ise sabit kısımdır. Diğer uvm_elemanlar ile gerekli bağlantıları yapıldıktan sonra değiştirilmeyecektir.

Tasarıma bağlı olarak birden fazla doğrulama ortamı kullanılabilir. Ardışıl toplayıcı devresinin doğrulanması gerçekleştirilirken iki adet doğrulama ortamı kullanılmıştır. Bunlardan ilki üst ortam, ikinde doğrulama elemanlarının ortamıdır.

Ardışıl toplayıcı devresinin test düzeneği ve doğrulama ortamlarına EK D'de yer verilmiştir.

UVM test düzeneği uyarıcıları uygulamaya başlamadan önce, UVM elemanların hiyerarşisi kurulmalı ve doğrulama elemanlarının birbiri arasındaki bağlantılar yapılmalıdır[13]. Bu kurma ve bağlantılar UVM fazları ile gerçekleştirilir. Ayrıca UVM fazları ile benzetim akışının işleyişi yönetilir. Şekil 5.11'de UVM fazları yer almaktadır.



Şekil 5.11 : UVM fazları[13].

Kurma fazları, test düzeneğinin yapılandırıldığı yerlerde yer alır. Koşma fazları ise test düzeneğinin zaman harcanılan kısımlarını yönetir. Temizleme fazları test durumunun sonuçlarını toplar ve raporlar.

Benzetim başlangıcı fazı ile çıkar fazı arasındaki koşma fazlarına ait tanımlar Tablo 5.2'de yer almaktadır.

Tablo 5.2 : Koşma Fazları[23].

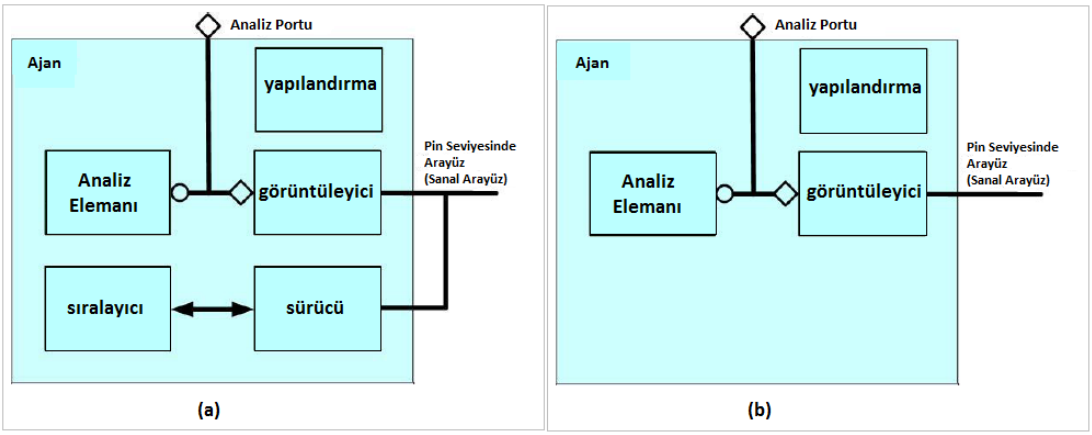
Faz Adı	Açıklaması
önceden sıfırlama	sıfırla işareti düşmeden önceki faz
sıfırlama	sıfırla işareti düştüğündeki faz
sonra sıfırlama	sıfırla işareti kalktığındaki faz
önceden yapılandırma	TET yapılandırılmadan önceki faz
yapılandırma	TET yapılandırıldığındaki faz
sonra yapılandırma	TET yapılandırıldıktan sonraki faz
ana işlemden önce	birincil test uyarıcısı başlamadan önceki faz
ana işlem	birincil test uyarıcısı başladığındaki faz
ana işlemden sonra	yeterince birincil test uyarıcısından sonraki faz
önceden durdurma	yerleşimden önceki faz
durdurma	yerleşimler bozulduğu andaki faz
sonra durdurma	yerleşme tamamlandıktan sonraki faz

5.2.5.2. UVM Ajanı

UVM ajan sınıfı, UVM elemanı sınıfından türetilmiştir ve doğrulama ortamı içinde yer alır. Ajan içinde ise görüntüleyici, sıralayıcı ve sürücü bağlantıları yapılmaktadır. Şekil 5.12'de görüldüğü gibi, UVM aktif iken bu üçü de ajan ile bağlantılıdır, pasif iken sadece görüntüleyici ile ajan bağlantı halindedir. Aynı zamanda sayı tahtası elemanı ve kapsayıcı elemanları ile de bağlantısı vardır. Bunlar ile bağlantısı analiz portu ile tanımlanır.

Tasarıma bağlı olarak bir doğrulama ortamında birden fazla ajan bulunabilir. Örneğin haberleşme protokollerinin doğrulanmasında alıcı ve verici için bir doğrulama

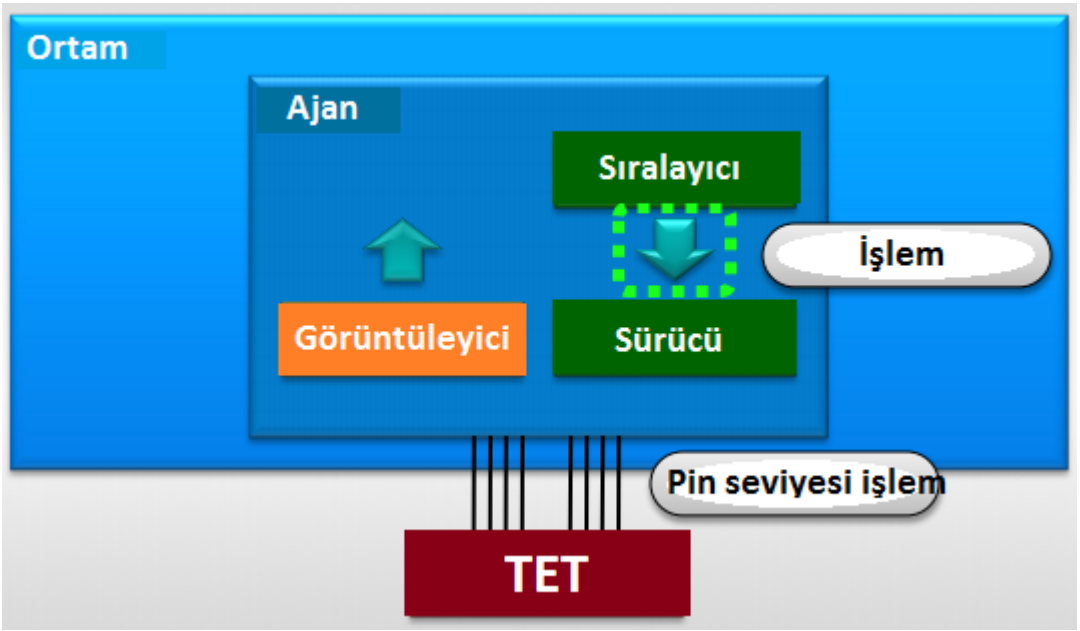
ortamında iki ayrı ajan bulunması gerekir. Fakat ardışıl toplayıcı için tek bir ajan yeterli olmaktadır. Ardışıl toplayıcı devresinin ajanına Ek E.1'de yer verilmiştir.



Şekil 5.12a : UVM aktif iken UVM ajanı [13].
Şekil 5.12b : UVM pasif iken UVM ajanı [13].

5.2.5.3. UVM Sıralayıcı

UVM sıralayıcı sınıfı, UVM eleman sınıfından türetilmiştir ve ajan içinde yer almaktadır. Sıralayıcı sınıfı, ürettiği işlemleri sürücüye aktarır. Bu işlemler en basit şekilde uvm_işlemler kullanılarak gerçekleştirilir. Fakat birden fazla değişkenin sürülmesi durumunda sıralayıcı sıra öğeleri üreterek bunları sürücüye aktarır. Sıra öğeleri ile ilgili ayrıntılara Sıra Öğeleri başlığı atında yer verilmiştir. UVM sıralayıcı Şekil 5.13'te yer almaktadır.



Şekil 5. 13 : UVM sıralayıcı [16].

UVM ile doğrulama yapılırken, UVM sınıflarını genişleten tasarıma ait sınıflar oluşturulur ve UVM paketi doğrulama paketine aktarılır. Fakat çoğu tasarımın doğrulanmasında sıralayıcı sınıfı için ayrı bir kaynak kodu yazılmamakta, doğrudan UVM sıralayıcı kullanılmaktadır.

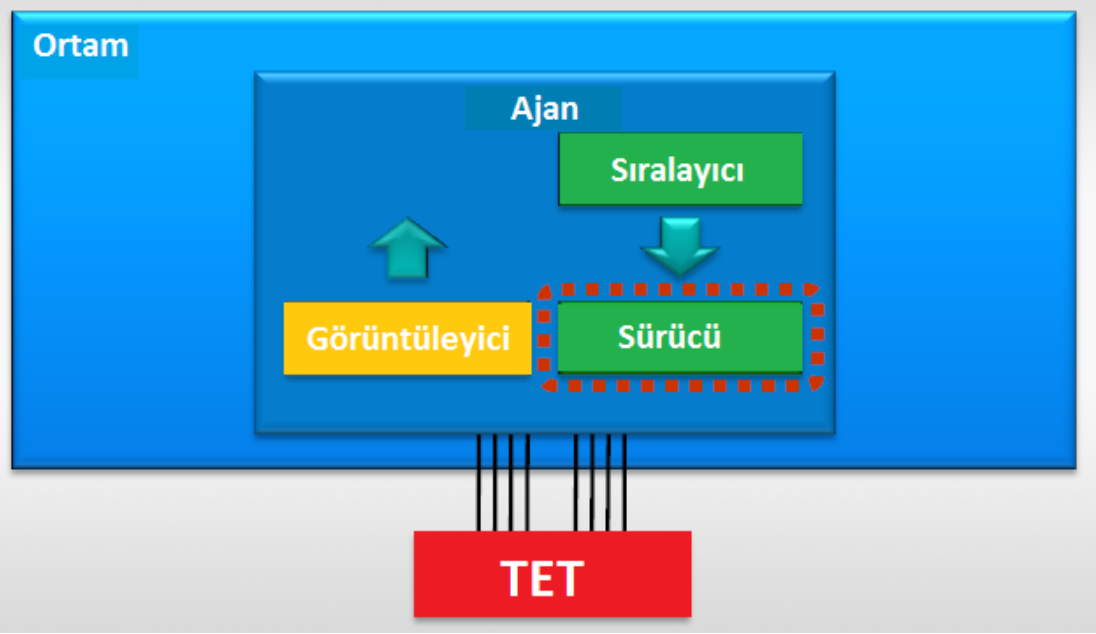
Ardışıl toplayıcı devresinin doğrulanmasında da böyle yapılmıştır. Sıralayıcı sınıfı, ajan içinde, doğrudan UVM sıralayıcı ile bağlanarak oluşturulmuştur.

Sıralayıcının görevi sadece ürettiği sıra öğelerini sürücüye aktarmaktır. Ne kadar sıra öğesi üreteceği ise bir sıra sınıfı ile gerçekleştirilir. Bu sıra sınıfı, ardışıl toplayıcı devresinde iç içe iki sıra sınıfı ile gerçekleştirilmiştir. Sürücüye iletilecek sıra öğelerinin sayısını bu sınıflar belirlemektedir. Bu sayı, sabit bir sayı olarak belirlenebileceği gibi rastgele bir değişken tanımlanarak da yapılabilir.

Ardışıl toplayıcı devresinde EK E.2'de görüldüğü gibi on adet sıra öğesi üretilmesi uygun görülmüştür.

5.2.5.4. UVM Sürücü

UVM sürücü de UVM elemandan türetilmiştir ve ajan içinde yer almaktadır. UVM sürücünün görevi, sıralayıcıdan aldığı sıra öğelerini TET girişlerine aktarmaktır. Şekil 5.14'te yer alan sürücünün sıralayıcı ile arasında sanal bir bağlantı vardır. Sürücünün TET ile olan bağlantısı ise pin seviyesinde bir bağlantıdır.

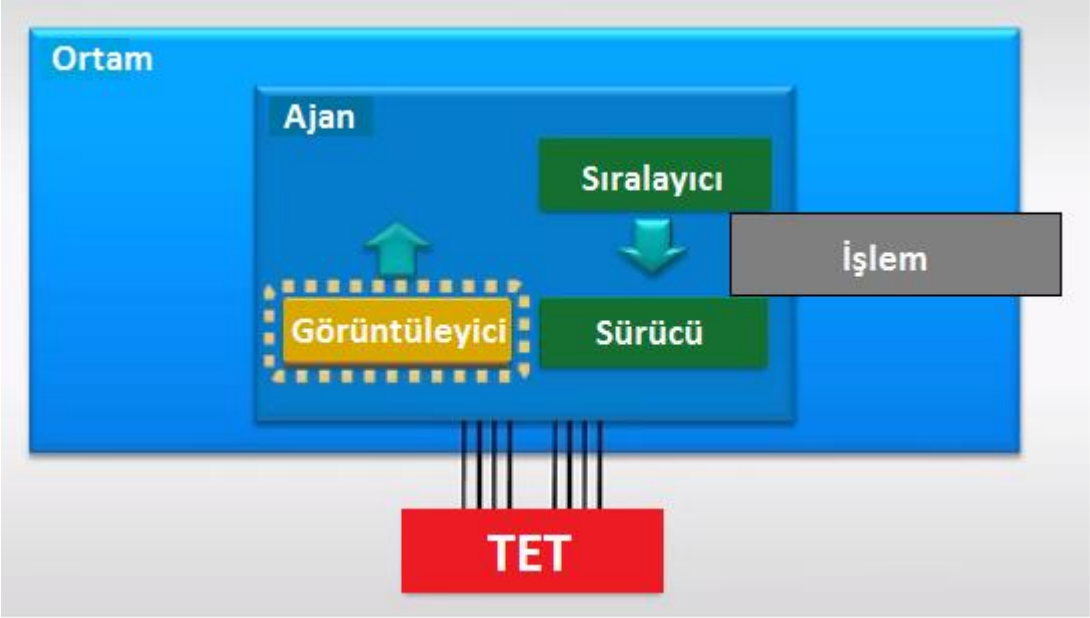


Şekil 5.14 : UVM sürücü [16].

Ardışıl toplayıcı devresinin sürücüsü, Ek E.3'te görüldüğü gibi, saat işareti ve sıfırla işareti dışındaki tüm TET girişlerini sürmektedir. Saat işareti ile sıfırla işareti ise test üst modülü tarafından belirlenir ve TET arayüzü ile ilgili TET girişlerine aktarılır. Sürücü kodunda, TET'in etkinleştir girişi sürülürken bir gecikme eklenmiştir. Bu gecikmenin sebebi, TET kodunda çıkışın üç saat işareti kadar gecikme sonrasında oluşmasıdır. Eğer sürücüye bu gecikme eklenmezse, çıkışta doğru olan değer görülemeden etkinleştir girişi sürücü tarafından lojik sıfıra çekilir ve TET anlamlı bir şekilde test edilmemiş olur.

5.2.5.5. UVM Görüntüleyici

UVM görüntüleyici sınıfı, UVM eleman sınıfından türetilmiştir ve ajan içinde yer almaktadır. TET sıra öğelerini giriş olarak alıp buna bir çıkış üretecektir. Görüntüleyicinin görevi, TET ile haberleşerek, onun hangi girişleri alıp hangi çıkışları ürettiğini görüntülemektir. Şekil 5.15'te görülen UVM görüntüleyici, UVM aktif değilken bile ajana bağlıdır.



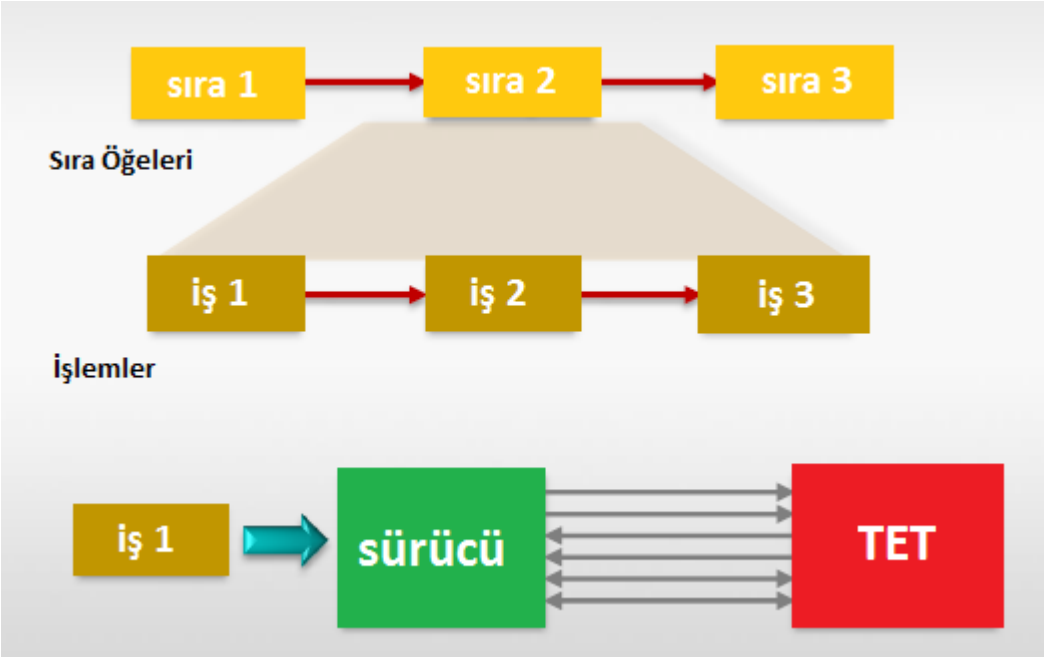
Şekil 5.15 : UVM görüntüleyici [19].

Görüntüleyici pasif bir elemandır, TET çıkışlarını görüntüleyerek analiz portuna aktarır. Bu analiz portunu ajan üzerinden kapsayıcılar okur. Ardışıl toplayıcı devresinin görüntüleyicisi, TET tarafından üretilen ve çıkışın geçerli olduğu anı bildiren, geçerli işareti lojik bir olduğu anda TET'i görüntüleyerek analiz portuna aktarır. Eğer görüntüleyiciden her an analiz portuna bilgi akışı gerçekleşseydi,

TET'in doğru çalışıp çalışmadığını belirlemek oldukça güçleşecekti. Ardışıl toplayıcı devresinin görüntüleyicisi Ek E.4'te yer almaktadır.

5.2.5.6. UVM Sıra Öğesi

UVM sıra öğesi sınıfı, UVM eleman sınıfından türetilmiştir. UVM Sıra öğesi TET için giriş değerlerinin üretildiği sınıftır. Sıralayıcı tarafından üretilen sıra öğeleri, Şekil 5.16'da görüldüğü gibi, sürücü üzerinden TET'e ulaşmaktadır.

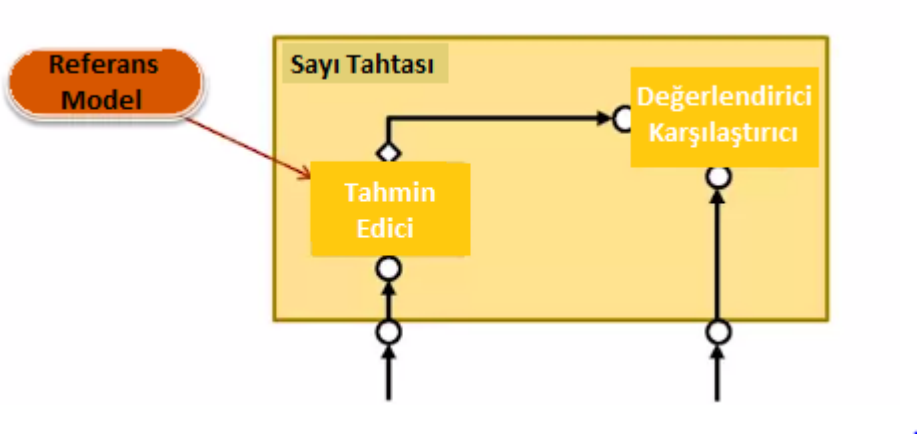


Şekil 5.16 : UVM sıra öğesi [20].

Sıra öğeleri rastgele işaretler olarak tanımlanır. Rastgele giriş işaretlerine, tasarıma yönelik kısıtlar girilerek doğrulamanın kesinliği artırılır. Rastgele sayılara nasıl kısıtlar girileceği Ek E.5'te, ardışıl toplayıcı devresinin sıra öğesinde, görülmektedir.

5.2.5.7. UVM Sayı Tahtası

UVM sayı tahtası sınıfı, UVM eleman sınıfından türetilmiştir ve bir analiz örneğidir. UVM sayı tahtasının asıl görevi, TET doğru çalışıyor mu çalışmıyor mu belirlemektir. Bunu belirleyebilmek için Şekil 5.17'de görüldüğü gibi bir referans modele ihtiyaç duyar.



Şekil 5.17 : UVM sayı tahtası [21].

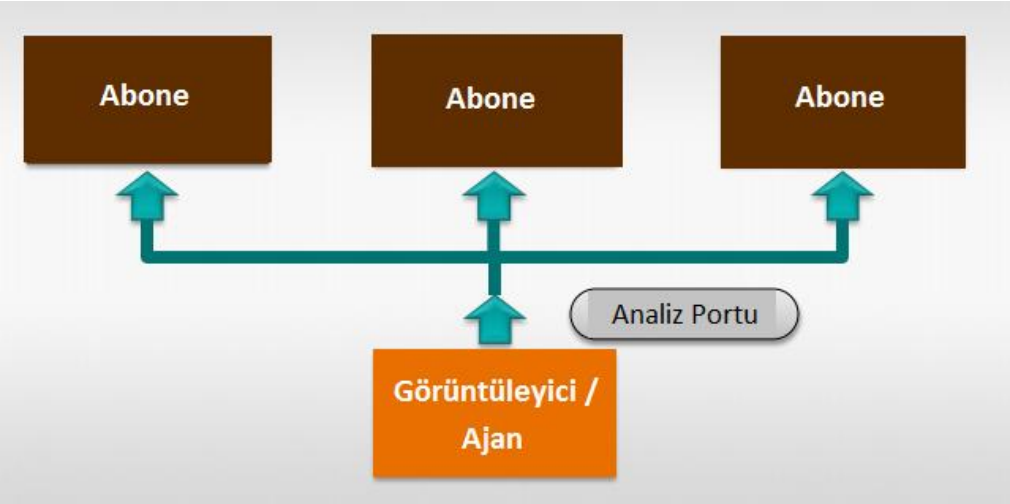
Referans model TET davranışını tahmin eder, tüm giriş kombinasyonları için çıkışın ne olacağını doğru bir şekilde hesaplar. Karmaşık sayısal sistem tasarımlarında C programlama dili ile bir model oluşturulup UVM ortamına aktarma olanağı vardır. Fakat ardışıl toplayıcı devresi basit bir tasarım olduğundan referans modeli SystemVerilog dili kullanarak oluşturmak da oldukça kolay olmuştur.

Sayı tahtası ajan ile bağlı olan bir analiz çıkış portuna sahiptir. Bu analiz çıkış portu sayesinde görüntüleyicinin analiz portuna gönderdiği TET davranışına ulaşır. Analiz çıkış portundan TET'in hangi girişlere hangi çıkışı ürettiği bilgisini alır. Referans model yardımıyla beklenen bir çıkış değeri hesaplar ve görüntüleyicinin elde ettiği gerçek çıkış değeri ile karşılaştırır. Raporlama tekniklerinden yararlanarak doğru çalışıp çalışmadığı anlaşılır.

Ardışıl toplayıcının sayı tahtası, bir abone ile iç içe tasarlanmıştır ve sayı tahtası sınıfı bu sayı tahtası abonesinin içinde yer almaktadır.

5.2.5.8. UVM Abonesi

UVM abonesi sınıfı, UVM eleman sınıfından türetilmiştir. Abone analiz portunun kullanımını basitleştirmeye yarar [22]. Şekil 5.18'de görüldüğü gibi birden fazla abone bir analiz portuna bağlandığında, yaz() (write()) fonksiyonu çağrıldığı zaman, her bir abone aynı sıra öğesinin imleyicisi olur.



Şekil 5.18 : UVM aboneleri [19].

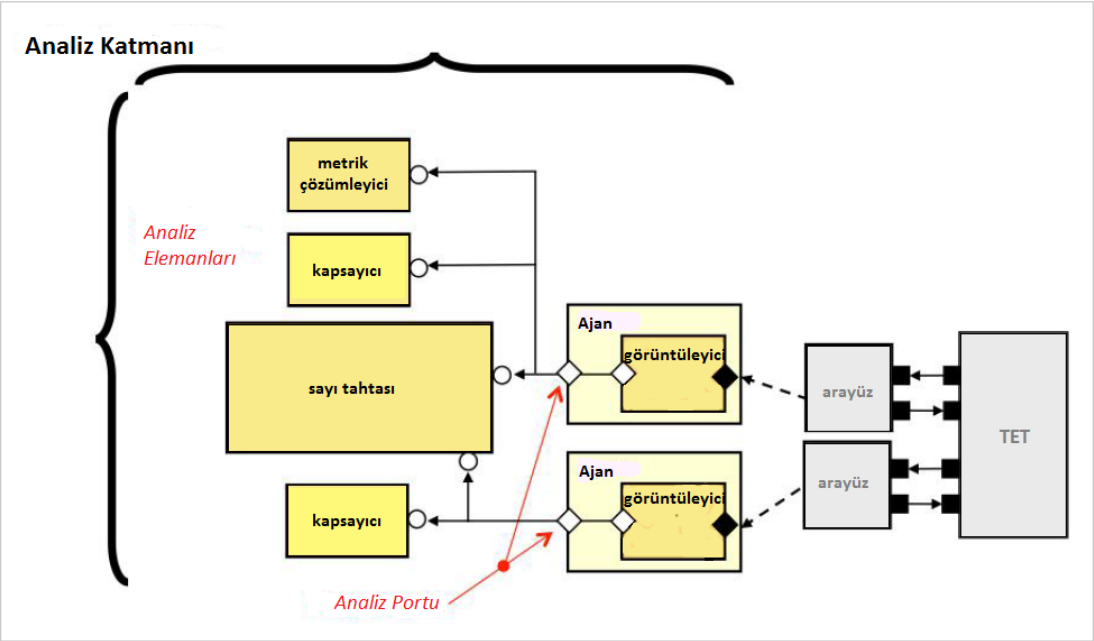
Her bir yaz() fonksiyonu, sıra öğesinin yerel bir kopyasını yapmalıdır ve bu kopyanın üzerinde çalışmalıdır. Böylece aynı sıra öğesinin başka bir imleyicisi olan abone tarafından, sıra öğesinin içeriğinin değiştirilmesi önlenmiş olur.

Ardışıl toplayıcı devresinde iki tane abone kullanılmıştır. Bunlardan biri kapsayıcı abone, diğeri de sayı tahtası abonesidir. Kapsayıcı; test tamamlandı mı, yeterince test yapıldı mı sorularının cevabı hakkında bilgi verir. Bunun bilgisini elde etmek için de kapsayıcının, ajanın analiz portuna bağlanması gerekir. Kapsayıcı abonesi, bu konuda kapsayıcıya yardımcı olmak için kullanılır. Sayı tahtası abonesi de sayı tahtasının analiz portuna bağlanmasını kolaylaştırmak için kullanılır.

Ardışıl toplayıcı devresinin doğrulanması sırasında kullanılan abonelere Ek E.6 ve Ek E.7'de yer verilmiştir.

5.2.6. Analiz Portu ve Analiz Çıkış Portu

Bir tasarımın doğrulanması iki ana başlığa ayrılır: Uyarıcı üretimi ve tasarım cevabının analizi[13]. Analiz bölümünün yargılama yaptığı bölüm, TET'in çıkışında uyarıcı sonrası bir cevap oluştuğu zaman başlar.

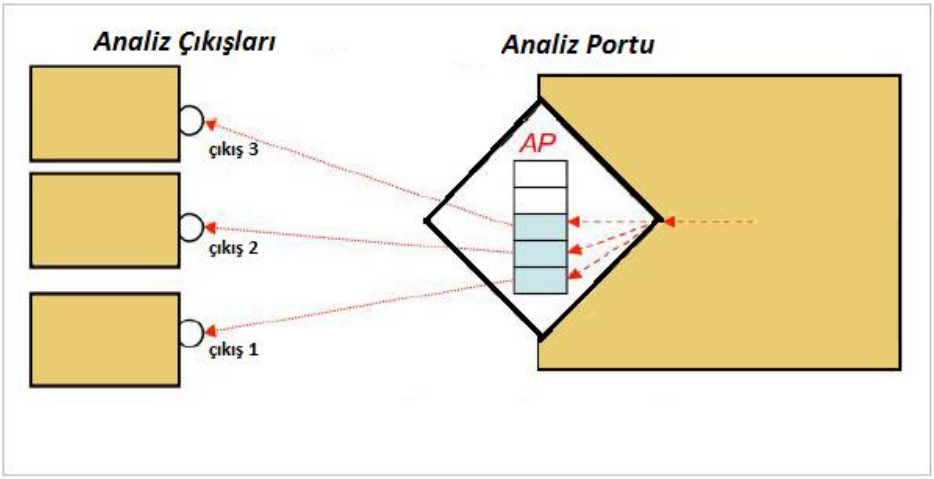


Şekil 5.19 : Analiz portu genel resmi[13].

TET üzerindeki, sanal arayüzler vasıtasıyla, işaret seviyesi hareketlerin gözlemlenmesi, Şekil 5.19'da görüldüğü gibi, birden fazla görüntüleyici kullanılarak da gerçekleştirilebilir. Görüntüleyiciler işaret seviyesi hareketleri İşlem Seviyesi Modelleme (Transaction Level Modeling – TLM) işlemlerine/sıra öğelerine çevirir. Bu işlemleri/sıra öğelerini, aboneler aracılığıyla analiz portlarını kullanan analiz elemanlarına yayınlar. Bu aboneler, işlemleri/sıra öğelerini ele geçirirler ve onların analizlerini gerçekleştirirler.

Şekil 5.19'daki analiz katmanında yer alan metrik çözümleyiciler; zamanlama, performans ve güç kullanımı gibi davranışsal olmayan fonksiyonları izler ve kaydederler. İşlem dizilerinin/sıra öğelerinin sayısına bağlı olarak UVM aboneleri olarak veya analiz çıkış portları ile uygulanabilirler.

Analiz portu ve analiz çıkış portu, standart UVM TLM anlamsalını takip ederler. Analiz portu, bağlanabilmek için yaz() uygulamasını gerektirir. Analiz çıkış portu ise yaz()fonksiyonunun uygulanmasını sağlar.



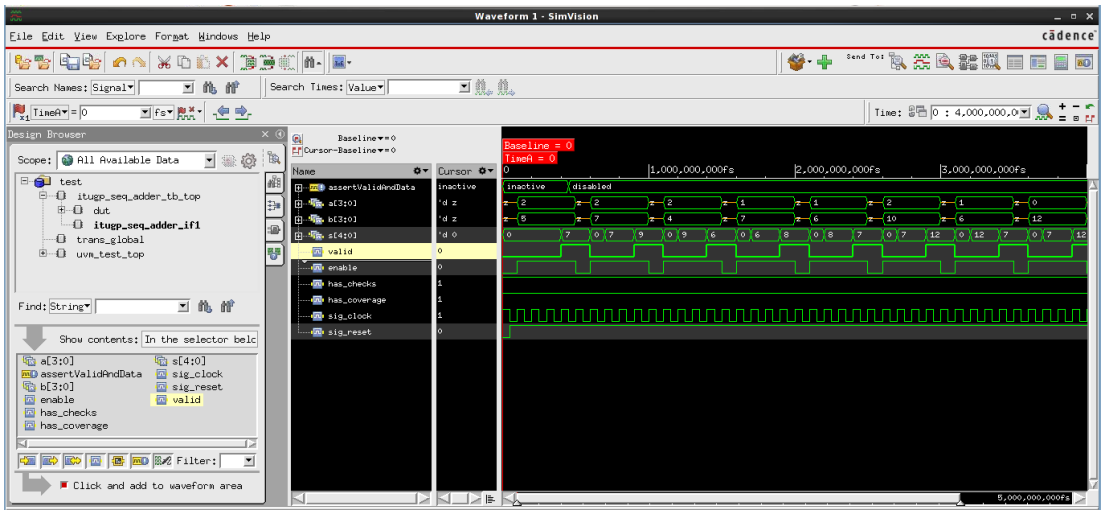
Şekil 5.20 : Analiz portu – analiz çıkış portları [13].

Şekil 5.20'de görüldüğü gibi analiz portu ile analiz çıkış portlarının birbirlerine bağlanabilmeleri için bağla() (connect()) fonksiyonunun çağırılması gerekir.

UVM abonesi ve UVM sayı tahtası sınıfları gömülü olarak analiz çıkış portu içermektedir. Bunların analiz çıkış portlarının ajan analiz portuna bağlanmaları gerekmektedir. Bağlantıların her iki tarafta da yapılması zorunludur. Port bağlantılarını incelemek için Ek E.1, E.6 ve E.7'ye birlikte bakılması gerekmektedir.

5.2.7. Çalıştırma Komutu

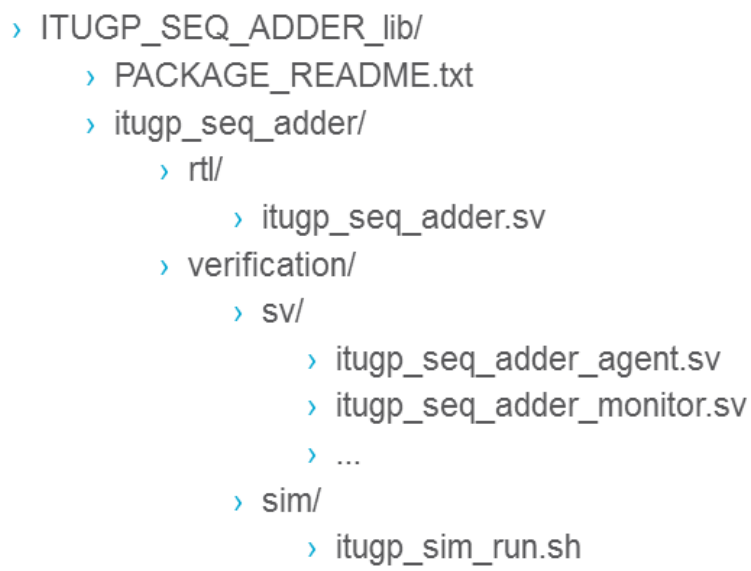
Oluşturulan UVM ortamını, IES tasarım aracında çalıştırmak için bir komut dosyasına ihtiyaç vardır. Terminalden bu komut dosyasının bulunduğu dizine gidilerek bu komut çalıştırılır. Komut dosyasının çağırılmasıyla beraber test düzeneği hiyerarşisi işlemeye başlar. UVM ortamında koşturulan test sonucunda UVM bilgi, uyarı ve hata raporları üretilir. Hatasız bir şekilde test tamamlandığında doğrulma işlemi bitmiş demektir. Bunun ardından dalga formunda işaretler gözlemlenmek istenirse, yine terminal kullanılarak, SimVision ortamına geçilir. Gerekli olan çalıştırma komutu dosyası EK.F'de yer almaktadır. Bu çalıştırma komutu ile ardışıl toplayıcı için kurulan doğrulama ortamında gerçekleştirilen test başarılı bir şekilde tamamlanmıştır. Bu testin ardından, elde edilen dalga formuna Şekil 5.21'de yer verilmiştir.



Şekil 5.21 : Ardışıl toplayıcının doğrulandıktan sonraki dalga formu.

5.2.8. Dizin Yapısı

UVM, tasarımcılara güvenilir bir doğrulama sağladığı gibi düzenli bir doğrulama ortamı da sağlamaktadır. Bu düzenli doğrulama ortamını düzgün bir biçimde gözden geçirebilmek için, yaygın olarak kullanılan bir dizin yapısına uyulmuştur. UVM ile doğrulama sırasında oluşturulan tüm dosyalar, bu dizine uyularak, Şekil 5.21'de görüldüğü gibi, yerleştirilmiştir.



Şekil 5.22 : Ardışıl toplayıcı devresinin doğrulanmasına ait dizin yapısı.

Tüm dosyalar bir kütüphane klasörü altında toplanmıştır. Bu klasör içerisinde dosyaların konumunu anlatan bir metin belgesi ile çalışma klasörü yer almaktadır. Çalışma klasörü ise doğrulama ve RTL olarak ikiye ayrılmıştır. RTL, TET kodunu içerir. Doğrulama klasöründe ise SystemVerilog ile yazılmış olan tüm doğrulama

kodlarını içinde barındıran bir sv klasörü ile benzetim klasörü olan sim klasörü yer alır. Bu klasör çalıştırma komutu dosyasını içermektedir ve bu komut bu klasör içinde çalıştırıldığından benzetim dosyaları ve rapor dosyaları da burada oluşturulur. Bir hata alındığında, düzeltmek amacıyla, bu raporlar incelenmelidir.

Tablo 5.3'te ardışıl toplayıcı devresini UVM ile doğrulamak için yazılan tüm kodların yerleşimi hakkında bilgiler yer almaktadır. Kaynak kodu adı boş bırakılan kod blokları, farklı isimde bir kaynak kodu içerisinde yer almaktadır ve bu kaynak kodu dosya yerleşiminden bulunur. Dosya yeri yer almıyor ise, UVM kütüphanesinde zaten var olan sınıf değiştirilmeden kullanılmış demektir. Ardışıl toplayıcı devresinde, sıralayıcı sınıfı bu şekilde kullanılmıştır.

Tablo 5.3 : Ardışıl toplayıcının doğrulanması sırasında oluşturulan dosyalar.

Açıklaması	Kaynak Kodu Adı	Modül/Sınıf Adı	Dosya Yeri
TET	itugp_seq_adder.sv	itugp_seq_adder	itugp_seq_adder/rtl/
üst ortam	itugp_seq_adder_top_env.sv	itugp_seq_adder_top_env	itugp_seq_adder/verification/sv/
doğrulama elemanı ortamı	itugp_seq_adder_vc_env.sv	itugp_seq_adder_vc_env	itugp_seq_adder/verification/sv/
test üst modülü	itugp_seq_adder_tb_top.sv	itugp_seq_adder_tb_top	itugp_seq_adder/verification/sv/
sıra öğesi sınıfı	itugp_seq_adder_seq_item.sv	itugp_seq_adder_seq_item	itugp_seq_adder/verification/sv/
sıralayıcı sınıfı		itugp_seq_adder_sequencer	
temel test sınıfı	itugp_seq_adder_test_lib.sv	itugp_seq_adder_base_test	itugp_seq_adder/verification/sv/
test sınıfı	itugp_seq_adder_test_lib.sv	itugp_seq_adder_first_test	itugp_seq_adder/verification/sv/
kapsayıcı abone sınıfı	itugp_seq_adder_fc_subscriber.sv	itugp_seq_adder_fc_subscriber	itugp_seq_adder/verification/sv/
sayı tahtası abone sınıfı	itugp_seq_adder_sb_subscriber.sv	itugp_seq_adder_sb_subscriber	itugp_seq_adder/verification/sv/
sayı tahtası		itugp_seq_adder_scoreboard	itugp_seq_adder/verification/sv/
ajan sınıfı	itugp_seq_adder_agent.sv	itugp_seq_adder_agent	itugp_seq_adder/verification/sv/
sürücü sınıfı	itugp_seq_adder_driver.sv	itugp_seq_adder_driver	itugp_seq_adder/verification/sv/
görüntüleyici sınıfı	itugp_seq_adder_monitor.sv	itugp_seq_adder_monitor	itugp_seq_adder/verification/sv/
birinci sıra sınıfı	itugp_seq_adder_seq_lib.sv	itugp_seq_adder_legal_transmit_seq	itugp_seq_adder/verification/sv/
ikinci sıra sınıfı	itugp_seq_adder_seq_lib.sv	itugp_seq_adder_illegal_transmit_seq	itugp_seq_adder/verification/sv/
paket	itugp_seq_adder_package.svh		itugp_seq_adder/verification/sv/
tanımlar	itugp_seq_adder_defines.sv		itugp_seq_adder/verification/sv/
komut dosyası	itugp_sim_run.sh		itugp_seq_adder/verification/sim/

6. SONUÇLAR

Bu bitirme projesinde, basit bir sayısal sistem tasarımının UVM ile doğrulanması ve bu sırada UVM ile doğrulama yönteminin öğrenilmesi amaçlanmıştır.

Basit bir sayısal sistem tasarımı olarak ardışıl bir toplayıcı devresi tasarlanmıştır. Ardışıl toplayıcı devresini doğrulamak için UVM kütüphanesinden yararlanılarak bir doğrulama ortamı oluşturulmuştur ve bu doğrulama ortamının ardışıl toplayıcı devresi ile bir arayüz vasıtasıyla haberleştirilmesi sağlanmıştır. Bu bağlantılar bir test üst modülünde yapılmıştır. Daha sonra ardışıl toplayıcı devresi için bir test düzeneği oluşturulmuştur. Bu test düzeneğinde devrenin girişlerinin rastgele sayılar ile sürülmesi, çıkışının görüntülenmesi, referans modelden elde edilen beklenen çıkış değeri ile görüntülenen gerçek çıkış değerinin karşılaştırılması ve devrenin doğru çalışıp çalışmadığının analiz edilmesi kurulan doğrulama ortamının elemanları ve test düzeneği sayesinde gerçekleştirilmiştir.

Projenin sonunda, ardışıl toplayıcı devresinin UVM ile doğrulanması sağlanmıştır. Basit bir sayısal sistemin bile UVM ile doğrulanmasının oldukça güç olduğunun farkına varılmıştır. UVM öğrenme aşamasında tasarımcıya zorluklar çıkarmış olsa da bunun yanında doğrulama hususunda çok büyük avantajlar da sağladığı görülmüştür.

UVM ile doğrulama adımları basit bir tasarım üzerinde öğrenildikten sonra, herhangi bir karmaşık tasarımın doğrulanmasını da yapabilecek yetiye sahip olunduğu düşünülmektedir.

Son yıllarda, sayısal sistem tasarımları oldukça geliştiğinden, bu tasarımların doğrulanması da gittikçe katlanan bir öneme sahip olmaktadır. Bu konudaki çalışmaların daha da artacağı ön görülmektedir.

KAYNAKLAR

[1] Spear C.(2008). *SystemVerilog for verification: A Guide to Learning the Testbench Language Features* (2nd Ed.). Marlboro, MA: Springer.

[2] Wikipedia, Universal Verification Methodology, [Alıntı Tarihi: 14 Aralık 2013], http://en.wikipedia.org/wiki/Universal_Verification_Methodology

[3] Yun Y. N., Kim J. B., Kim N. D., Min B. (2011), Beyond UVM for Practical SoC Verification, International SoC Design Conference, Korea.

[4] Bromley J., If System Verilog Is So Good, Why Do We Need the UVM? Sharing Responsibilities Between Libraries and the Core Language, Verilab Ltd, Edinburgh – Scotland.

[5] Wikipedia, SystemVerilog, [Alıntı Tarihi: 3 Mayıs 2014], http://en.wikipedia.org/wiki/SystemVerilog

[6] Sutherland, S., Davidmann, S., Flake, P., SystemVerilog for Design: A Guide to Using SystemVerilog for Hardware Design and Modeling, Kluwer Academic Publishers, Boston,MA, 2004, ISBN: 0-4020-7530-8

[7] Crypto Acceleration Using Asynchronous FPGAs, [Alıntı Tarihi: 11 Mayıs 2014],http://www.seminarsonly.com/Engineering-Projects/Computer/Crypto_Acceleration_Using_Asynchronous_FPGAs.php

[8] Wikipedia, Mentor Graphics, (2013). [Alıntı Tarihi: 15 Mayıs 2014], http://en.wikipedia.org/wiki/Mentor_Graphics

[9] Wikipedia, Synopsys, [Alıntı Tarihi: 16 Mayıs 2014], http://en.wikipedia.org/wiki/Synopsys

[10] Synopsys, (2005). Discovery Visual Environment User Guide, Version 2005.06

[11] Cadence Website, Incisive Enterprise Simulator, [Alıntı Tarihi: 10 Mayıs 2014], http://www.cadence.com/products/fv/enterprise_simulator/pages/default.aspx

[12] Aynsley, J., Doulos, First Steps with UVM Part 1, https://www.doulos.com/knowhow/video_gallery/#anchor34

[13] Mentor Graphics, Verification Academy UVM Cookbook, [Alıntı Tarihi: 10 Ocak 2014], https://verificationacademy.com/cookbook/uvm

[14] Accellera, (2011). Universal Verification Methodology (UVM) 1.1 Class Reference

[15] Kutay, F., (2012). Functional Verification UVM Methodology, EMDC Seminer Serisi

[16] Fitzpatrick, T., UVM Basics – Uvm Introducing Transactions, Verification Academy Website, [Alıntı Tarihi: 11 Kasım 2013], https://verificationacademy.com/sessions/uvm-introducing-transactions

[17] Fitzpatrick, T., UVM Basics – Uvm Hello World, Verification Academy Website, [Alıntı Tarihi: 11 Kasım 2013], https://verificationacademy.com/sessions/uvm-hello-world

[18] Fitzpatrick, T., UVM Basics – Uvm Connecting Env Dut, Verification Academy Website, [Alıntı Tarihi: 11 Kasım 2013], https://verificationacademy.com/sessions/uvm-connecting-env-dut

[19] Fitzpatrick, T., UVM Basics – Uvm Monitors and Subscribers, Verification Academy Website, [Alıntı Tarihi: 11 Kasım 2013], https://verificationacademy.com/sessions/uvm-monitors-and-subscribers

[20] Fitzpatrick, T., UVM Basics – Uvm Sequences and Tests, Verification Academy Website, [Alıntı Tarihi: 11 Kasım 2013], https://verificationacademy.com/sessions/uvm-sequences-and-tests

[21] Mentor, Verification Academy, Scoreboards and Results Predictors in UVM, https://verificationacademy.com/seminars/uvm-recipe-scoreboards-and-predictors-for-uvm

[22] Accellera, (2011). Universal Verification Methodology (UVM) 1.1 User's Guide

EKLER

EK A.1

EK A.2

EK A.3

EK B.1

EK B.2

EK C

EK D.1

EK D.2

EK D.3

EK E.1

EK E.2

EK E.3

EK E.4

EK E.5

EK E.6

EK E.7

EK F

ÖZGEÇMİŞ

Adı Soyadı: Güler ÇOKTAŞ

Doğum Yeri ve Tarihi: Sivas, 1991

Lise: İstanbul Atatürk Fen Lisesi; 2005-2009

Lisans: İstanbul Teknik Üniversitesi, Elektronik Mühendisliği; 2009-2014