İSTANBUL TEKNİK ÜNİVERSİTESİ
ELEKTRİK – ELEKTRONİK FAKÜLTESİ

16 Bit Kayan Noktalı Aritmetik Mantık Birimi Tasarımı Ve Doğrulanması

BİTİRME ÖDEVİ

Sait Furkan Teke

040140728

Bölümü: Elektronik ve Haberleşme Mühendisliği

Programı: Elektronik ve Haberleşme Mühendisliği

Danışmanı: Doç. Dr. Sıddıka Berna ÖRS YALÇIN

MAYIS 2016

İSTANBUL TEKNİK ÜNİVERSİTESİ
ELEKTRİK – ELEKTRONİK FAKÜLTESİ

16 BİT KAYAN NOKTALI ARİTMETİK MANTIK BİRİMİ TASARIMI VE DOĞRULANMASI

BİTİRME ÖDEVİ

Sait Furkan TEKE

040140728

Bölümü: Elektronik ve Haberleşme Mühendisliği

Programı: Elektronik ve Haberleşme Mühendisliği

Danışmanı: Doç. Dr. Sıddıka Berna ÖRS YALÇIN

MAYIS 2016

ÖNSÖZ

Bitirme projesinin her aşamasında yardımlarını esirgemeyen, sabırla ve anlayışla doğru yolu göstererek hatalarımı düzeltmeme yardımcı olan değerli hocam Doç. Dr. Sıddıka Berna ÖRS YALÇIN'a teşekkürlerimi sunarım. Ayrıca bu zamana kadar bana ellerinden gelen her türlü desteği veren aileme teşekkür ederim.

MAYIS 2016 Sait Furkan Teke

İÇİNDEKİLER

KISALTMALAR

UVM : Universal Verification Methodology
FPGA : Field Programmable Gate Array
ALU : Arithmetic Logic Unit
IEEE : Institute of Electrical and Electronics Engineers
TET : Test Edilen Tasarım

ŞEKİL LİSTESİ

Sayfa

No

16 BİT KAYAN NOKTALI ARİTMETİK MANTIK BİRİMİ TASARIMI VE DOĞRULANMASI

ÖZET

Aritmetik mantık birimi toplama, çıkarma, çarpma, bölme, kaydırma gibi birçok farklı işlem özelliğine sahip olabilir. Aritmetik mantık biriminin sahip olduğu farklı özelliklere göre kullanım alanı da değişmektedir.

Bu projede 16 bit kayan noktalı sayılarla toplama ve çıkarma işlemi yapabilen bir aritmetik mantık biriminin tasarımı ve doğrulanması hedeflenmiştir. 16 bit kayan noktalı sayıların hesaplanması IEEE 754 standardı temel alınarak yapılmıştır. Sistemin tasarımı Verilog dili kullanılarak gerçekleştirilmiştir. Doğrulama aşamasında Evrensel Doğrulama Metodu (Universal Verification Methodology – UVM) kullanılmıştır ve kodlar SystemVerilog dili kullanılarak yazılmıştır.

16 BIT FLOATING POINT ARITHMETIC LOGIC UNIT DESIGN AND VERIFICATION

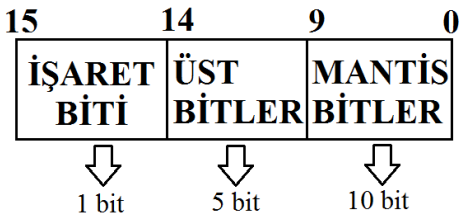
SUMMARY

Arithmetic logic unit can have different features such as addition subtraction, multiplication, division, shifting etc. The area of application of arithmetic logic unit can vary according to their features.

In this project, it is aimed to design and verify an arithmetic logic unit which can add and subtract with 16 bit floating point numbers. The calculation of 16 bit floating point numbers is performed based on IEEE 754 standard. The system is designed by using Verilog language. The system is verified by using Universal Verification Methodology (UVM). The code for verification phase is developed with SystemVerilog language.

1. GİRİŞ

Kayan noktalı sayılar, çok büyük ya da çok küçük sayılara ihtiyaç duyulduğunda kullanılmaktadır [1]. Sabit noktalı sayı gösterimi yerine kayan noktalı sayı gösterimini kullanmak doğruluk ve çözünürlük bakımından daha avantajlıdır. Kayan noktalı sayılar işaret biti, mantis bitleri ve üst bitlerin birleşiminden oluşan bit dizi ile gösterilirler. Bu gösterim standardına IEEE 754 adı verilmiştir [2]. Bu projede 16 bitlik kayan noktalı sayı gösterimi olan yarım duyarlıklı kayan noktalı sayı formatı kullanılmıştır [3]. Şekil 1.1'de yarım duyarlıklı kayan noktalı sayı formatı gösterilmiştir.



Şekil 1.1: IEEE 754 Yarım Duyarlıklı Kayan Noktalı Sayı Formatı Gösterimi

Yarım duyarlıklı kayan noktalı sayıların hesaplama işlemi aşağıdaki formül ile yapılır.

$$n = (-1)^s \, 2^{e-15}(1.f) \qquad (1)$$

Bu formülde s işaret bitini temsil eder ve 0,1 değerlerini alabilir. e üst değeri temsil eder ve 0-31 arası değer alabilir. f ise mantis bitlerin değerini temsil eder ve f, 0'a eşit olamaz.

Aritmetik mantık birimi (Arithmetic Logic Unit – ALU), aritmetik ve mantıksal işlemlerin yapıldığı modüldür. Merkezi işlem birimi (Central Processing Unit – CPU) içerisinde ALU önemli bir konuma sahiptir. Seçim bitlerine bağlı olarak ALU farklı işlemler ve farklı sonuçlar vermektedir [4].

Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Italic
Formatted: Font: Italic

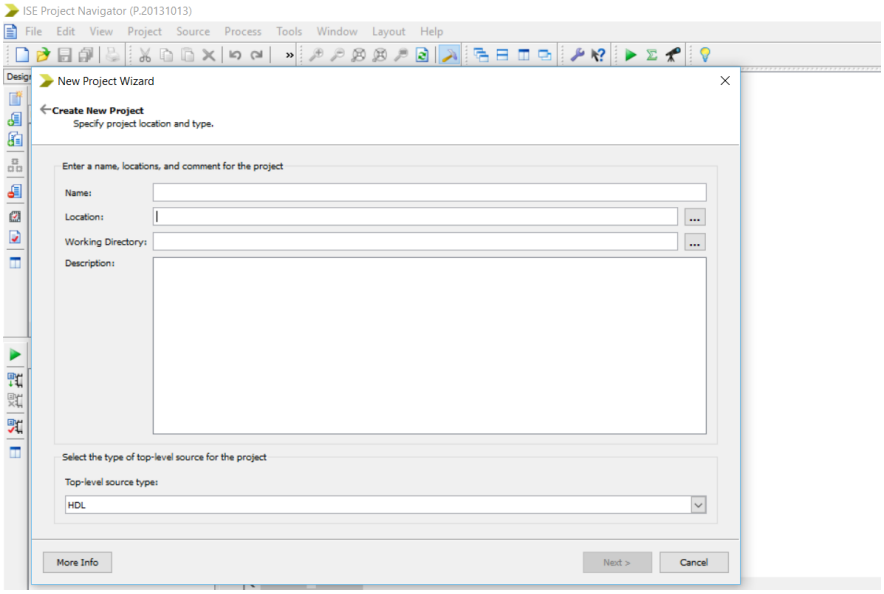
Bu projede 16 bit kayan noktalı sayılar kullanılarak toplama ve çıkarma işlemlerini yapabilen sayısal bir sistem tasarımı ve doğrulanması üzerine çalışılmıştır. Tasarım aşamasında Verilog [5], doğrulanma aşamasında ise System Verilog [6] dilleri kullanılmıştır. Doğrulama yapılırken Evrensel Doğrulama Yöntemi (Universal Verification Methodology – UVM) [7] takip edilmiştir. Kullanılan diller, araçlar ve diğer ayrıntılar hakkında bilgiler ilgili bölümün altında ayrıntılı olarak anlatılacaktır.

2. KULLANILAN TASARIM VE DOĞRULAMA ARAÇLARI

2.1 Tasarım Araçları

2.1.1 Xilinx ISE WebPACK Design Software

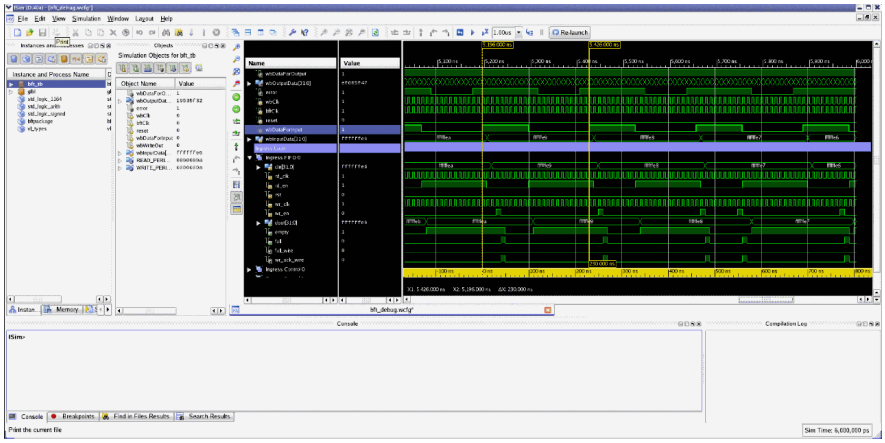
Xilinx firmasının geliştirmiş olduğu ISE WebPACK Design Software programı ücretsiz olarak dağıtılmaktadır. Bu program ile Alanda Programlanabilir Kapı Dizileri (Field Programmable Gate Array – FPGA) ref ve Karmaşık Programlanabilir Mantık Cihazı (Complex Programmable Logic Device – CPLD) ref tasarımlarının HDL sentezleri gerçekleştirilebilmektedir [8]. Bu projede Verilog dili kullanılarak tasarlanan sistemin sentezlenmesi bu program ile yapılmıştır.



Şekil 2.1: Xilinx ISE Design Suite Yeni Proje Oluşturma Ekranı

2.1.2 Xilinx ISE Simulator (ISim)

ISE WebPACK'in içerisinde bulunan ISim, HDL simülasyonlarını yapabilmektedir. Bu program sayesinde tasarlanan sistemi FPGA kartlarına yüklemeden test edebilme imkânı elde edilmiş olur [9].

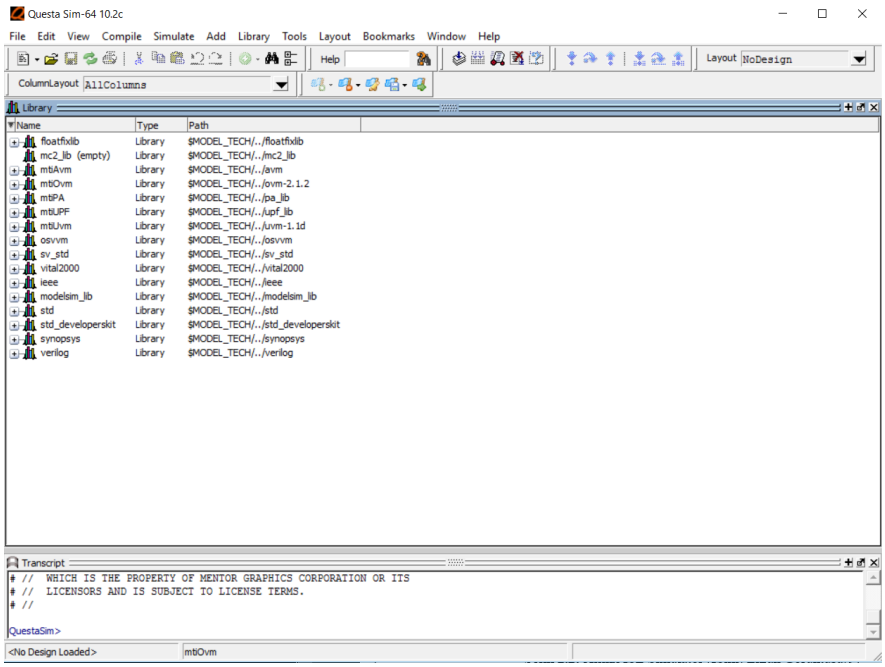


Şekil 2.2: Xilinx ISE Simulator (ISim) Ekran Görüntüsü [9]

2.2 Doğrulama Araçları

2.2.1 QuestaSim

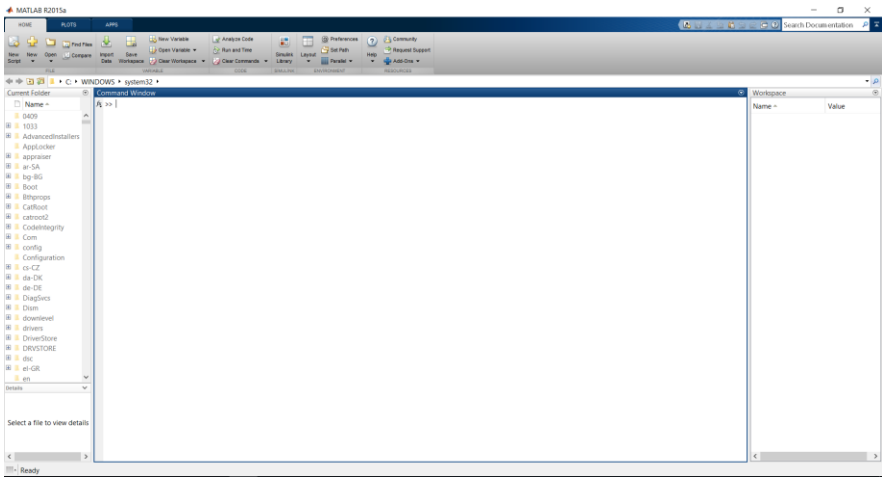
Mentor Graphics firmasına ait olan QuestaSim gelişmiş fonksiyonel doğrulama yapabilen bir programdır [10]. QuestaSim yüksek verimlilikle büyük elektronik sistemlerin doğrulama işlemlerini gerçekleştirebilmektedir. QuestaSim programı kullanılarak UVM için SystemVerilog dili ile doğrulama yapılabilir.



Şekil 2.3: QuestaSim Ekran Görüntüsü

2.2.2 MATLAB

MATLAB, MathWorks firmasına ait bir programdır ve bu program ile makine öğrenmesi, sinyal işleme, kontrol tasarımı, robotik gibi birçok alanda çalışma yapılabilmektedir [11]. Bu projenin doğrulama aşamasında tasarlanan sayısal sistemin sonuçları ile aynı işlemi yapan başka bir sistemin sonuçlarını karşılaştırmak için MATLAB programı kullanılmıştır.



Şekil 2.4: MATLAB Ekran Görüntüsü

3. KULLANILAN SAYISAL SİSTEMİN TASARIMI

3.1 Genel Bilgiler

16 bitlik kayan noktalı toplayıcı ve çıkarıcı devresini tasarlarken Verilog dili kullanıldı. Verilog sayısal sistem tasarlamak amacıyla kullanılan donanım tanımlama dilidir (Hardware Description Language – HDL) [5].

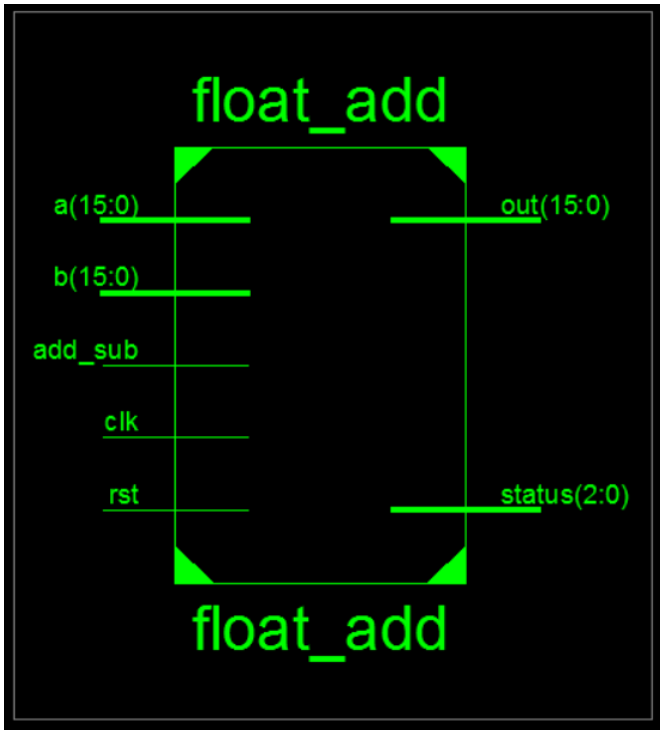
3.2 Tasarımın Gerçekleştirilmesi

16 bit kayan noktalı ALU devresinin tasarımı [4] numaralı referanstaki makaleyi takip ederek gerçekleştirilmiştir. Önceden de belirtildiği gibi bu projede sadece toplama ve çıkarma devresi tasarlanacaktır.

Toplama ve çıkarma devresi 6 farklı ana modülden oluşmaktadır: Paket çözücü (Unpack), karşılaştırıcı ve kaydırıcı (Comparator and shifter), Toplayıcı/Çıkarıcı (Add/sub),normalize edici (Normalizer), istisna kontrol edici (Exception checker) ve paketleyici (Packer). Devrede bu modüller arasındaki geçişler D flip floplar kullanılarak gerçekleştirilmiştir.

Devrenin 16 bitlik 2 farklı sayı girişi, toplama ya da çıkarma işlemlerinden hangisini yapacağını belirttiği 1 bitlik add_sub girişi, sistemin saat (clock) girişi olan clk ve sistemi sıfırlayan rst girişi ile birlikte toplam 5 girişi bulunmaktadır. Sistemin 2 adet çıkışı bulunmaktadır. Bunlar, çıkış sonucunu veren 16 bitlik out çıkışı ve çıkış durumu bildiren status çıkışıdır.

16 bitlik kayan noktalı toplayıcı çıkarıcı devresinin bütün alt modüllerini içeren üst modülün Verilog kodu EK A'da görülebilmektedir. Alt modüller arası veri geçişlerini sağlayan D flip flop devrelerinin Verilog kodları ise EK B'de görülebilmektedir.



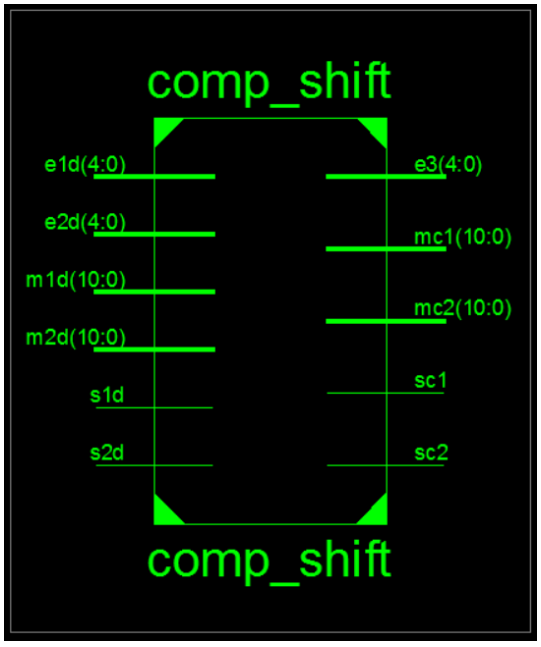
Şekil 3.1: 16 Bit Kayan Noktalı Toplayıcı Çıkarıcı Devresi Giriş ve Çıkışları

3.2.1 Paket Çözücü (Unpack) Modülü

Paket çözücü modülünde 16 bit halinde gelen sayı değerleri işaret, üst ve mantis bitlerine ayrıştırılır. Gelen sayının mantis bitlerinin en anlamlı kısmına IEEE 754 standardında olduğu gibi 1 eklenir. Bu işlem ile matis bitimiz 10 bit yerine 11 bit olmuş olur. Ayrıca bu modülde gelen değerlerin birbirine ya da sıfıra eşit olup olmadığı kontrol edilir. Sonuçlar D flip floplar aracılığıyla bir sonraki modüle aktarılır. Paket çözücü modülün Verilog kodları EK-C'de görülebilmektedir.

3.2.2 Karşılaştırıcı ve Kaydırıcı (Comparator and Shifter) Modülü

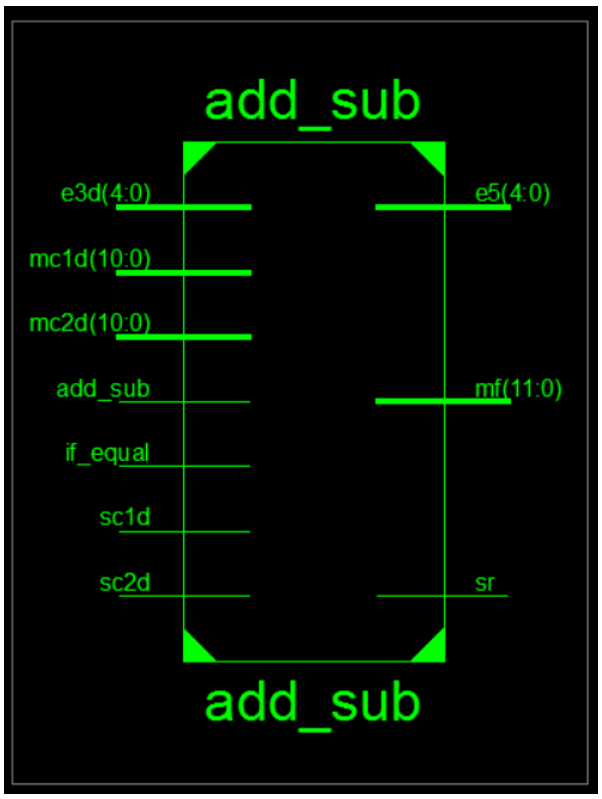
Karşılaştırıcı ve kaydırıcı modülünde gelen sayılar birbileri ile karşılaştırılır. Eğer sayıların üst (exponent) değerleri birbirinden farklıysa sayıların üstleri eşitlenir. Bu eşitleme işlemi üstü küçük olan sayının üst değeri her 1 arttırıldığında en anlamlı bit tarafından her arttırmada bir adet 0 eklenmesi ile sağa kaydırılması işlemidir. Eğer giren sayıların üstleri birbiri ile aynı ise herhangi bir değişiklik yapılmaz. Üstlerin eşitlenmesi ve kaydırma işlemi bittikten sonra çıkış değerleri bir sonraki modüle aktarılır. Karşılaştırıcı ve kaydırıcı modülünün Verilog kodları EK-D'de görülebilmektedir.



Şekil 3.2: Karşılaştırıcı ve Kaydırıcı Modülü

3.2.3 Toplayıcı/Çıkarıcı (Add/Sub) Modülü

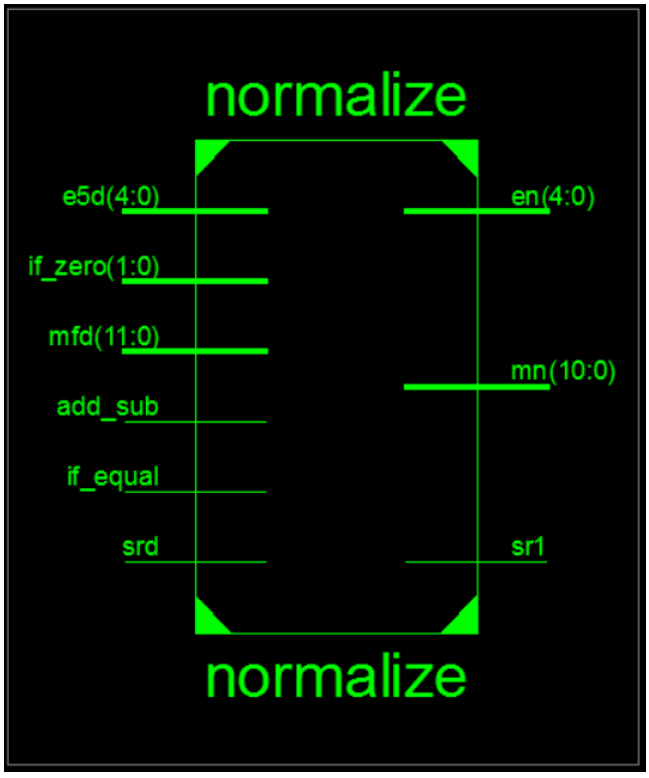
Toplayıcı ve çıkarıcı modülü gelen verilere göre toplama mı yoksa çıkarma mı yapılacağına karar verir ve bu işlemleri gerçekleştirir. Toplama ya da çıkarma işlemi gelen sayıların mantis bitleri üzerinde yapılır. Bir önceki modülde üst değerleri birbirine eşitlendiği için üst değer değişmeden aktarılır. Sistemin ilk girişinde alınan add_sub biti yapılacak işlemin toplama mı yoksa çıkarma mı olduğunu sisteme söylemektedir. Gelen bite göre modül işlemi gerçekleştirir. Toplama ve çıkarma işlemlerini alt modülü olan ripple-carry toplayıcı ile gerçekleştirir. Elde edilen sonuç ve diğer değerler bir sonraki modüle aktarılır. Toplayıcı çıkarıcı modülünün Verilog kodları EK-E'de görülebilmektedir.



Şekil 3.3: Toplayıcı/Çıkarıcı Modülü

3.2.4 Normalize Edici (Normalizer) Modül

Toplama ya da çıkarma işlemi gerçekleştirildikten sonra sonuç sayısı için işaret biti, üst bitleri ve mantis bitleri elde edilmiş oldu. IEEE 754 standardına göre paket çözücü modülünde mantis bitlerinin başına 1 biti eklenmişti. Bu eklemeyle birlikte mantis biti 10 bit yerine 11 bit boyutlu olmuştu. Toplama ya da çıkarma işleminden sonra elde edilen mantis bitlerinin en anlamlı biti yani 11. biti eğer 1 değilse bu biti 1 yapmak için kaydırma işlemi yapılır ve kaydırmanın yapıldığı yöne göre üst değeri arttırılır ya da azaltılır. Bu yapılan işlem ile sayı normalize edilmiş olur. Bu modülün çıkışında 5 bitlik üst değer, 11 bitlik mantis değeri ve 1 bitlik işaret değeri bir sonraki modüle aktarılır. Normalize edici modülün Verilog kodları EK-F'de görülebilmektedir



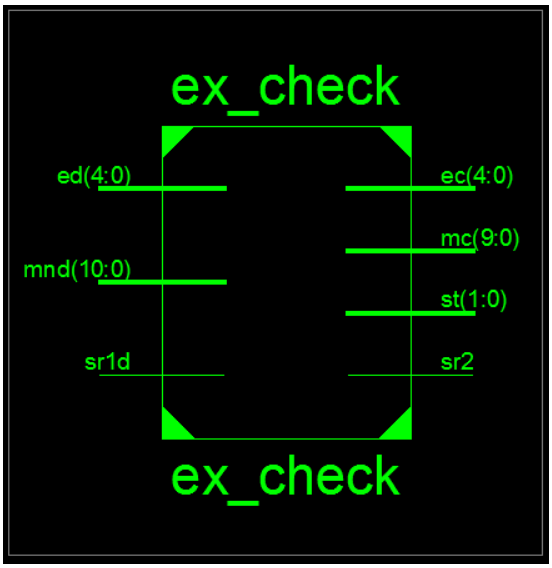
Şekil 3.4: Normalize Edici Modül

3.2.5 İstisna Kontrol Edici ~~(Exception Checker)~~ Modül

İstisna kontrol edici modülde çıkan sayı değerinin herhangi bir istisna durumuna sahip olup olmadığını kontrol eder ve durumu bit kodu halinde çıkışa verir. Bu sayede sistemde istisna bir durum var ise kolayca bulunabilir. İstisna kontrol edici 4 farklı durumu belirtebilmektedir. Taşma (overflow), yetersizlik (underflow), sonucun sıfır olması (result zero) ve normal işlem (normal operation). Bu modülde elde edilen veriler bir sonraki modüle çıkış olarak aktarılır. İstisna kontrol edici modülün Verilog kodları EK-G'de görülebilmektedir.

No.	Status bits[2:0]	Status
1	000	Result zero
2	001	Overflow
3	010	Underflow
4	011	Normal Operation

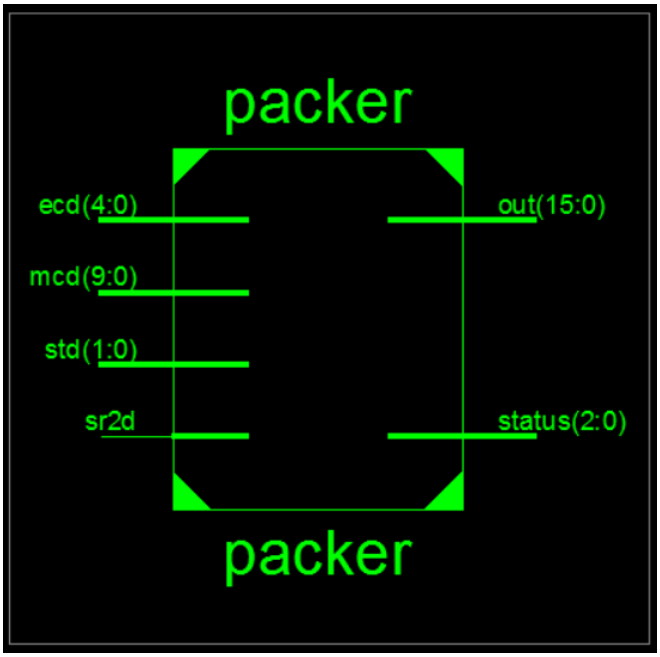
Şekil 3.5: İstisna Kontrol Edici Modülün Durum Çıkışları [4]



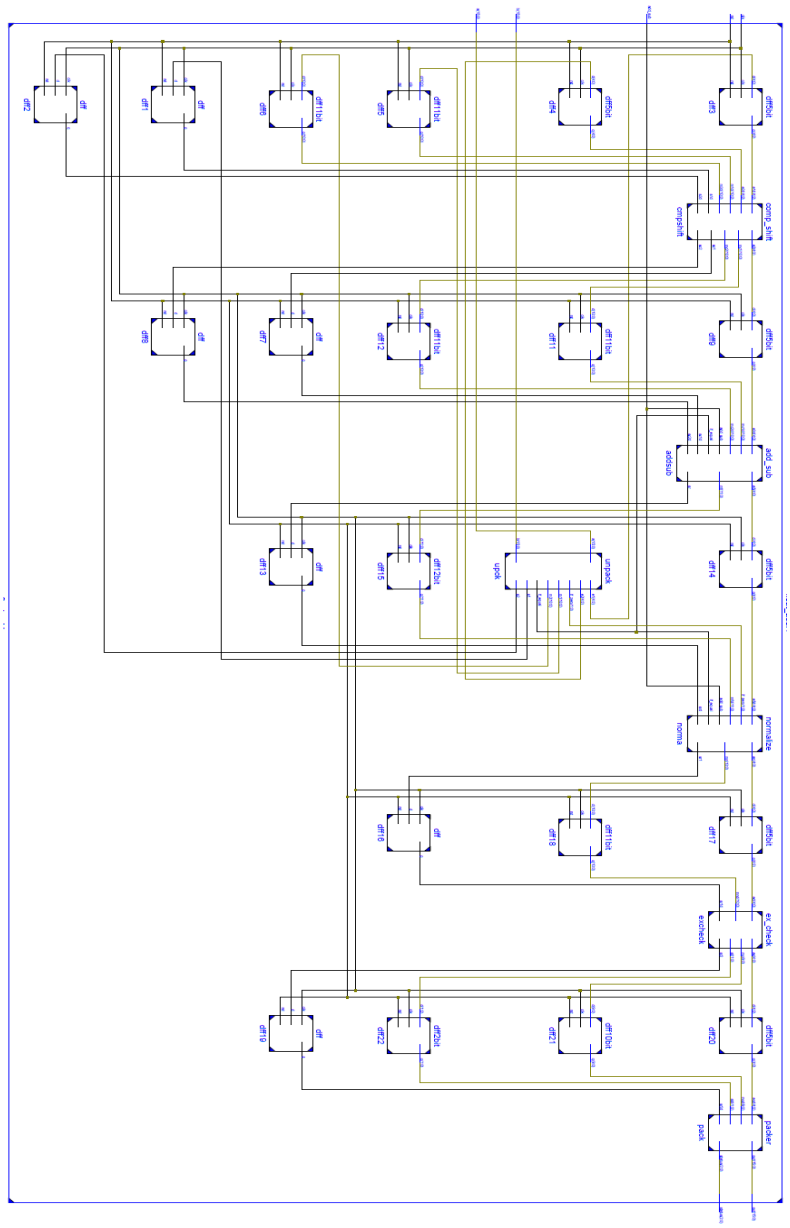
Şekil 3.6: İstisna Kontrol Edici Modül

3.2.6 Paketleyici (Packer) Modülü

Paketleyici modülü elde edilen işaret, üst ve mantis bitlerini birleştirerek 16 bitlik bir çıkış oluşturur. Aynı zamanda istisna kontrol edici modülden dönen durum bitini de çıkış olarak verir. Bu modülün çıkışları aynı zamanda ana sistemin de çıkışlarıdır. Paketleyici modülün Verilog kodları EK-H'de görülebilmektedir.



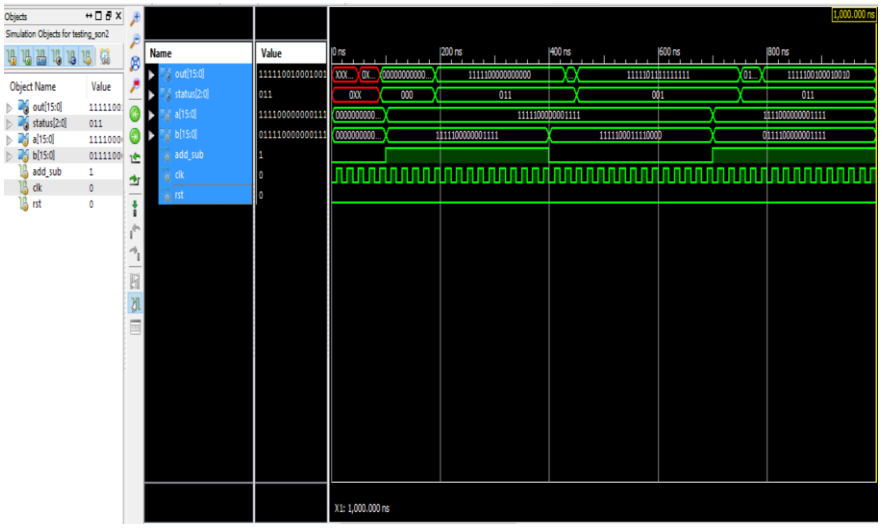
Şekil 3.7: Paketleyici Modülü



Şekil 3.8: 16 Bit Kayan Noktalı Toplayıcı Çıkarıcı Devresi Bütün Modüller

3.3 Tasarımın Simülasyonu

16 bit kayan noktalı toplayıcı çıkarıcı devresinin tasarımı bittikten sonra Xilinx ISE Simulator kullanılarak simülasyona belli giriş değerleri verildi. Bu giriş değerleri için olması gereken ve elde edilen çıkış değerleri karşılaştırıldı. Simülasyonda elde edilen sonuçlara göre tasarım düzgün bir şekilde çalışmaktadır.



Şekil 3.9: Xilinx ISE Simulator Ekran Görüntüsü

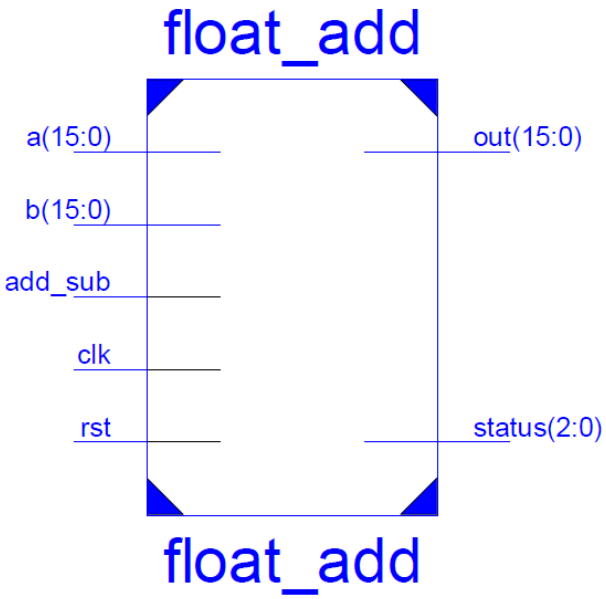
4. KULLANILAN SAYISAL SİSTEMİN DOĞRULANMASI

4.1 Genel Bilgiler

16 Bit Kayan Noktalı ALU devresinin doğrulama işlemi UVM yöntemi kullanılarak yapılacaktır. UVM, SystemVerilog dilini kullanarak doğrulama işlemini gerçekleştirir. Projenin doğrulama kısmında G. Çoktaş'ın bitirme tezinden [12] ve P. Araujo'nun Yeni Başlayanlar için UVM Rehberi [13] yazısından yararlanılmıştır.

4.2 Test Edilen Tasarım ~~(TET)~~

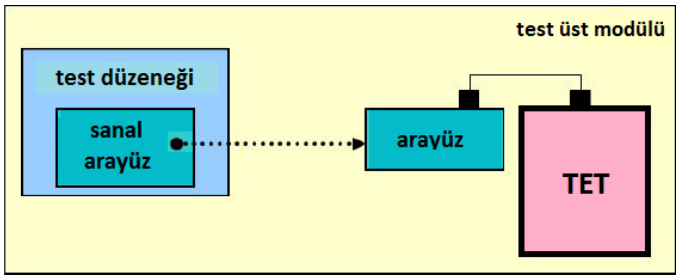
Doğrulama işleminde test edilen tasarım (TET) olarak önceki bölümde anlatılan 16 Bit Kayan Noktalı Toplayıcı Çıkarıcı ALU tasarımı kullanılacaktır. ALU'nun girişlerini ve çıkışlarını Şekil 4.1'de görebilirsiniz.



Şekil 4.1: TET'ın girişleri ve çıkışları

4.3 Test Üst Modülü ve Arayüz

Test üst modülünde TET ile test çevresi arasındaki bağlantıları sağlayan arayüz modülü çağırılır. Ayrıca diğer tüm gerekli alt bileşenleri içeren paket modülü de çağırılır. Arayüzde tanımlanan sistemin giriş ve çıkışları üst modülde de tanımlanır. Üst modülde giriş çıkış tanımlamaları yapıldıktan sonra TET ile arayüz arasındaki bağlantı kurulur. Test üst modülü EK-I'da, arayüz modülü EK-J'de ve paket modülü EK-K'de gösterilmiştir.



Şekil 4.2: Test üst modülü [12]

4.4 UVM Elemanları

UVM yöntemi ile doğrulama yapılırken belli elemanlar kullanılmaktadır. Elemanlar SystemVerilog dili kullanılarak yazılmıştır. Bu elemanlar alt başlıklarda ayrıntılı olarak incelenecektir.

4.4.1 UVM Çevre Elemanı ve Test Düzeneği

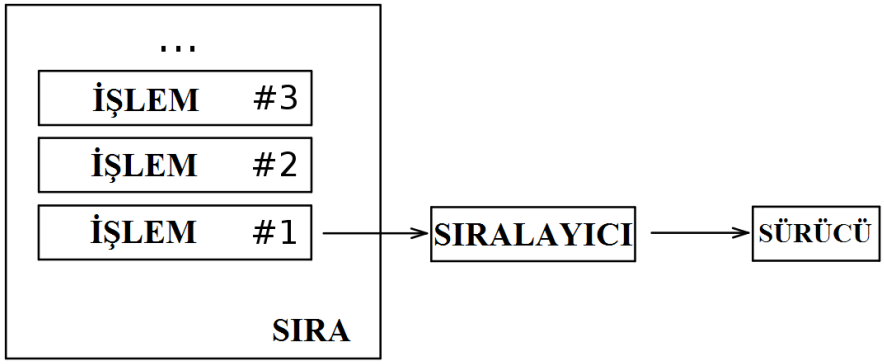
16 Bit Kayan Noktalı Toplayıcı Çıkarıcı ALU devresinin doğrulanması yapılırken iki adet çevre elemanı kullanılmıştır. Bunlardan biri üst çevredir, diğeri ise doğrulama elemanlarının çevresidir. Doğrulama elemanlarının çevresinde ajan ve sayı tahtası elemanları başlatılır ve birbirlerine bağlanır.

Test düzeneği uvm_test sınıfından türetilir ve genel olarak iki amacı vardır. Bunlardan birincisi çevre elemanını oluşturmak, ikincisi ise sıralayıcı elemanı ile sıra elemanını bağlamaktır.

UVM çevre elemanlarının ve test düzeneğinin SystemVerilog kodları EK-L'de görülebilir.

4.4.2– UVM Sıra Elemanı

UVM sıra elemanında TET'da bulunan girişlere değerler üretilir. Üretilen bu giriş değerleri rastgele değerlerdir. Rastgele giriş değerlerine göre sistem çıkış değerlerini hafızasına kaydeder. Sıra, sıralayıcı ve sürücü elemanları arasındaki ilişki Şekil 4.3'te görülebilir.

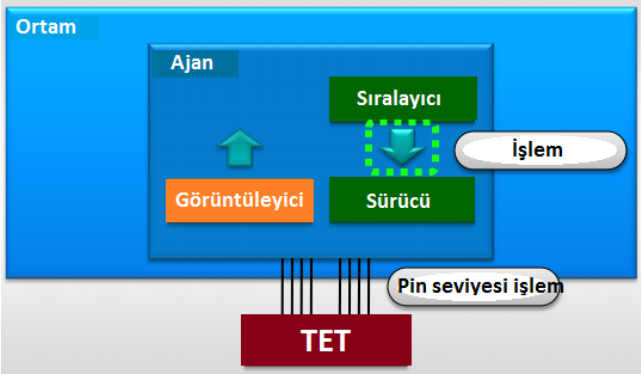


Şekil 4.3: Sıra, sıralayıcı ve sürücü arasındaki ilişki [13]

Sıra elamanının SystemVerilog kodları EK-M'de incelenebilir.

4.4.3– UVM Sıralayıcı Elemanı

UVM sıralayıcı elemanı uvm_sequencer sınıfından türetilmiştir. Sıralayıcının genel amacı sıra elemanından gelen verileri sürücüye iletmektir. Sıra ve sıralayıcı elemanları arasındaki bağlantı yukarıda bahsedilen test düzeneğinde yapılmaktadır. Şekil 4.4'te UVM sıralayıcısının görseli bulunmaktadır.

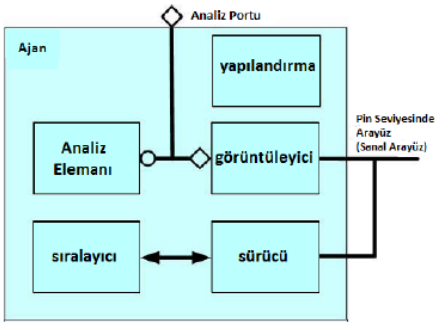


Şekil 4.4: Sıra, sıralayıcı ve sürücü arasındaki ilişki [12]

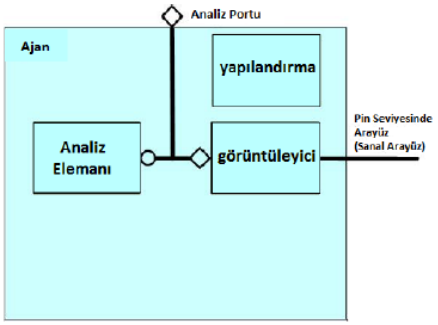
UVM sıralayıcı elemanının SystemVerilog kodları EK-N'de görülebilmektedir.

4.4.4– UVM Ajan Elemanı

Ajan elemanı uvm_agent sınıfından türetilir. UVM ajan elemanı, sıralayıcı, görüntüleyici ve sürücü elemanlarını birbirine bağlar. Görüntüleyici, sıralayıcı ve sürücü elemanları ajanın içinde çalışır hale getirilir. Şekil 4.5'te ve Şekil 4.6'da görülebileceği gibi UVM çalışırken ajan elemanı ve görüntüleyici, sıralayıcı ve sürücü elemanları birbirlerine bağlıdır. UVM çalışmadığı zaman ise sadece ajan ve görüntüleyici elemanları bağlantıdadır. UVM ajan elemanının SystemVerilog kodları EK-O'da görülebilir.



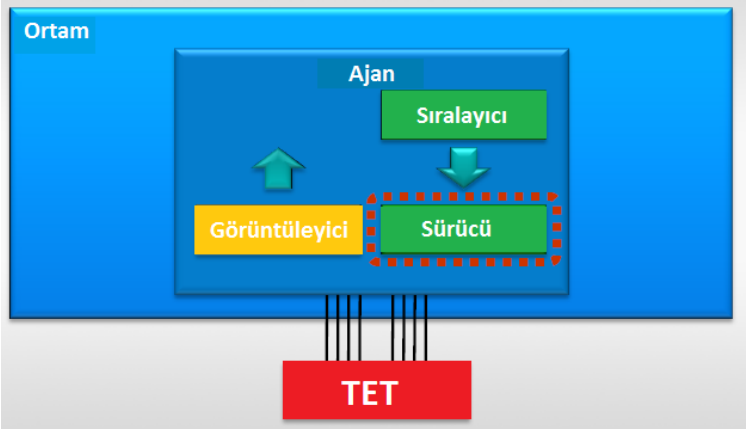
Şekil 4.5: UVM çalışırken ajan elemanı [12]



Şekil 4.6: UVM çalışmıyorken ajan elemanı [12]

4.4.5– UVM Sürücü Elemanı

UVM sürücü elemanı uvm_driver sınıfından türetilmiştir ve ajan elemanının altında bulunmaktadır. Sürücünün görevi TET ile iletişim halinde olmaktır. Sürücü elemanı, sıralayıcıdan aldığı verileri TET'ın girişlerine gönderir. Sürücü elemanı ve TET arasında pin seviyesinde bir bağlantı mevcuttur. UVM sürücü elemanı Şekil 4.7'de görülebilmektedir.

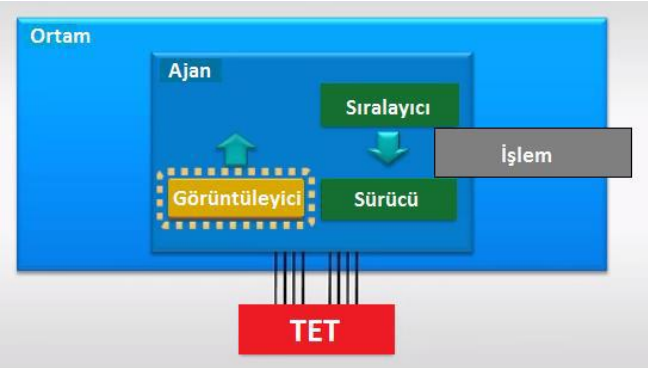


Şekil 4.7: UVM sürücü elemanı [12]

UVM sürücü elemanının SystemVerilog kodları EK-P'de görülebilmektedir.

4.4.6– UVM Görüntüleyici Elemanı

UVM görüntüleyici elemanı uvm_monitor sınıfından türetilmiştir ve ajan elemanının altında bulunmaktadır. Sürücünün sıralayıcıdan alıp TET'e gönderdiği giriş değerleri sonucunda çıkış değerleri üretilecektir. Görüntüleyici, TET'e giren giriş değerlerini ve girişlere göre çıkış değerlerini ekrana bastırır. Ayrıca protokollere aykırı bir durumda görüntüleyici hata mesajı da vermektedir. Şekil 4.8'de görüntüleyici elemanı görülebilmektedir.

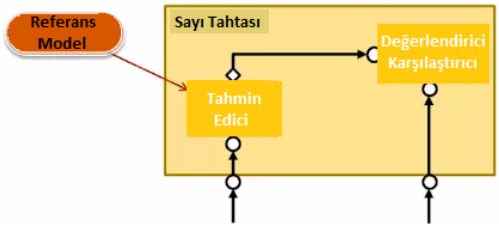


Şekil 4.8: UVM görüntüleyici elemanı [12]

Pasif bir eleman olan görüntüleyici, TET'e herhangi bir sinyal göndermez. Görüntüleyicinin amacı sadece sinyal bilgilerini alıp okumak ve bu bilgileri anlamlı hale getirerek görüntülemektir. UVM görüntüleyici elemanının SystemVerilog kodları EK-R'de görülebilmektedir.

4.4.7– UVM Sayı Tahtası Elemanı

UVM sayı tahtası elemanı uvm_scoreboard sınıfından türetilmiştir. TET'ın doğru bir şekilde çalışıp çalışmadığı bu elemanda kontrol edilir. TET'in doğru çalıştığını test edebilmek için referans modele ihtiyaç vardır. Bu projede 16 Bit Kayan Noktalı Toplayıcı Çıkarıcı Devresi için referans modeli MATLAB programı kullanılarak yapılmıştır. Tasarımda bulunan modüller MATLAB'da benzetilerek aynı aşamalar oluşturulmuş ve sayı tahtası elemanı içerisinde birbirine bağlanmıştır. UVM sayı tahtası elemanı Şekil 4.9'da görülebilmektedir.

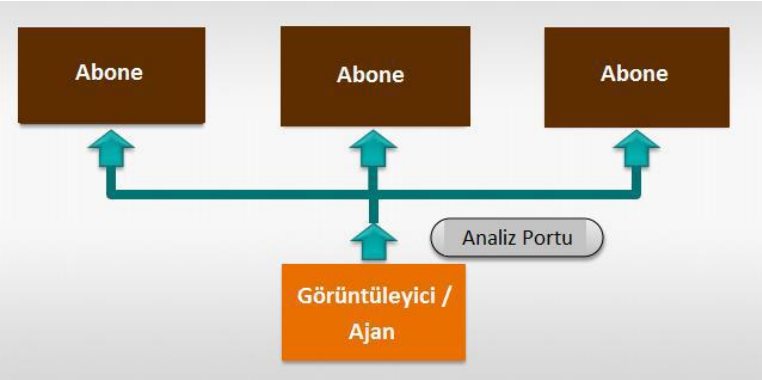


Şekil 4.9: UVM sayı tahtası elemanı [12]

Sayı tahtası ve ajan arasında analiz çıkış portu vardır. Bu port kullanılarak görüntüleyicinin sahip olduğu TET'in sinyallerine erişilir. TET'in hangi girişlere hangi çıkışları ürettiği bu analiz çıkış portunda görülür. MATLAB'da oluşturulan referans model kullanılarak olması beklenen çıkış değeri hesaplanır ve bu değer görüntüleyicide TET'ten elde edilen gerçek değer ile karşılaştırılır. Bu sayede sistemin doğru bir şekilde çalışıp çalışmadığı belirlenebilir. Sayı tahtası elemanı bir abone elemanının içinde bulunmaktadır. UVM sayı tahtası SystemVerilog kodları EK-S.1'de görülebilmektedir.

4.4.8– UVM Abonesi Elemanı

UVM abonesi elemanı uvm_subscriber sınıfından türetilmiştir. Abone elemanları analiz portunun daha kolay bir şekilde kullanılmasına olanak sağlamaktadır[14]. Şekil 4.10'da UVM abone elemanları görülebilmektedir.



Şekil 4.10: UVM abone elemanları [12]

Abone elemanlarının içerisinde yaz() (write()) fonksiyonu vardır. Her bir yaz() fonksiyonu sıra öğesinin yerel kopyasını oluşturmalıdır. Bu sayede aynı sıra öğesinin başka bir işaretleyicisi olan abone elemanının sıra öğesinin içeriğini değiştirmesi önlenmiş olunur [12].

Bu projede iki adet abone elemanı kullanılmıştır. Bu abone elemanlarından birisi sayı tahtası abonesi, diğeri ise kapsayıcı abonedir. Kapsayıcı, yeterli miktarda test yapılıp yapılmadığını, testin bitip bitmediğini inceler. Kapsayıcının bu bilgileri inceleyebilmesi için ajan elemanının analiz portuna bağlanması gerekmektedir ve bu bağlantıyı kapsayıcı abonesi ile sağlamaktadır. Aynı mantıkla sayı tahtası abonesi de sayı tahtasının analiz portuna kolay bir şekilde bağlanmasına yardımcı olur.

UVM abonesi elemanlarının SystemVerilog kodları EK S'de görülebilmektedir.

5. SONUÇLAR

Bitirme projesinde 16 bit kayan noktalı sayılarla toplama ve çıkarma yapabilen bir ALU tasarlanması ve doğrulanması hedeflenmiştir.

16 bit kayan noktalı sayılarla işlem yapabilmek için IEEE 754 standardında bulunan yarım duyarlıklı kayan noktalı sayı formatı takip edilmiştir. Tasarlanması hedeflenen sayısal sistem Verilog dilinde 6 alt modül halinde yazılmıştır. Yazılan modüller bir üst modül içerisinde birleştirilmiş ve alt modüller arasında veri iletimleri D flip floplar kullanılarak gerçekleştirilmiştir. Tasarım aşamasının sonunda simülasyon yapılmıştır. Bu simülasyonda sistemin verilen giriş değerlerinde istenen çıkış değerlerini ürettiği gözlenmiştir. Sistemin doğruluğunu test etmek için bu aşamadan sonra doğrulama aşamasına geçilmiştir.

Doğrulama aşamasında UVM kullanılmıştır. Doğrulama kodları SystemVerilog dilinde yazılmıştır. Doğrulama için tasarlanan sistemin yanında sonuçların karşılaştırılabileceği bir referans sistem gerekmektedir. Bu referans sistem MATLAB programı kullanılarak oluşturulmuştur. UVM elemanları ayrıntılı olarak incelenip yazıldıktan sonra doğrulama çalıştırma aşamasına geçilmiştir. Tasarlanan sistemin karışıklığı ve doğrulama programı hakkında yeterli bilgi sahibi olunamadığı için son işlem doğrulamanın çalıştırılması gerçekleştirilememiştir.

Sayısal sistemlerin doğrulanması yeniden üretim maliyetini azaltmada çok büyük bir etkendir. Üretime gitmeden önce sistemin istenen gereksinimler için test edilip doğrulanması, sistemde oluşacak hataları minimize etmektedir. Bu sayede hatalar üretimden sonra değil, üretilmeden önce test edilip düzeltilebilmektedir. Getirdiği avantajlardan ötürü bu konudaki çalışmaların daha fazla artacağı öngörülmektedir.

KAYNAKLAR

[1] **Rajit Ram Singh, Asish Tiwari, Vinay Kumar Singh, Geetam S Tomar**, 2011. VHDL environment for floating point Arithmetic Logic Unit -ALU design and simulation, *International Conference on Communication Systems and Network Technologies.*

[2]**IEEE 754-2008**, 2008. IEEE Standard for Floating-Point Arithmetic, *IEEE Computer Society.*

[3]**Wikipedia,** Half-precision floating-point format, [Alıntı Tarihi: 9 Mayıs 2016], https://en.wikipedia.org/wiki/Half-precision_floating-point_format.

[4]**Dave, O.E. and Aarthy, M**., 2014. ASIC Implementation of 32 and 64 bit Floating Point ALU using Pipelining, *International Journal of Computer Applications*, **94,** 27-35.

[5]**Wikipedia,** Verilog, [Alıntı Tarihi: 9 Mayıs 2016], https://en.wikipedia.org/wiki/Verilog.

[6]**Wikipedia,** Verilog, [Alıntı Tarihi: 16 Mayıs 2016], https://en.wikipedia.org/wiki/SystemVerilog.

[7]**Wikipedia,** Verilog, [Alıntı Tarihi: 16 Mayıs 2016], http://en.wikipedia.org/wiki/Universal_Verification_Methodology.

[8]**Xilinx Website**, ISE WebPACK Design Software, [Alıntı Tarihi: 9 Mayıs 2016], http://www.xilinx.com/products/design-tools/ise-design-suite/ise-webpack.html.

[9]**Xilinx Website**, ISE Simulator (ISim), [Alıntı Tarihi: 9 Mayıs 2016], http://www.xilinx.com/products/design-tools/isim.html.

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

[11]**MathWorks Website**, MATLAB, [Alıntı Tarihi: 18 Mayıs 2016], http://www.mathworks.com/products/matlab.

[12]**Çoktaş, G.**, 2014. Bir sayısal sistem tasarımının evrensel doğrulama metodu ile doğrulanması, *Lisans Tezi*, İ.T.Ü Elektronik ve Haberleşme Mühendisliği Bölümü, İstanbul

[13]**Pedro Araujo Website**, UVM Guide for Beginners, [Alıntı Tarihi: 18 Mayıs 2016], https://colorlesscube.com/uvm-guide-for-beginners/

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

EKLER

EK-A

```
`timescale 1ns / 1ps
module float_add(a,b,add_sub,clk,rst,out,status,valid);
    input[15:0] a;
    input[15:0] b;
    input add_sub;
    input clk;
    input rst;
    output[15:0] out;
    output[2:0] status;
    output valid;

    wire s1,s2;
    wire[4:0] e1,e2;
    wire[10:0] m1,m2;
    wire if_equal;
    wire[1:0] if_zero;

    unpack upck(a,b,s1,s2,e1,e2,m1,m2,if_equal,if_zero);

    wire s1d,sc1,sc1d;
    wire s2d,sc2,sc2d;
    wire[4:0] e1d,e3,e3d;
    wire[4:0] e2d;
    //wire[4:0] e4,e4d;
    wire[10:0] m1d,mc1,mc1d;
    wire[10:0] m2d,mc2,mc2d;

    wire sr,srd,sr1,sr1d,sr2,sr2d;
    wire[4:0] e5,e5d,en,ed,ec,ecd;
    wire[11:0] mf,mfd;
    wire[10:0] mn,mnd;

    wire[9:0] mc,mcd;

    wire[1:0] st,std;

    dff dff1(clk,rst,s1,s1d);
    dff dff2(clk,rst,s2,s2d);

    dff5bit dff3(clk,rst,e1,e1d);
    dff5bit dff4(clk,rst,e2,e2d);

    dff11bit dff5(clk,rst,m1,m1d);
    dff11bit dff6(clk,rst,m2,m2d);

    comp_shift cmpshift(s1d,s2d,e1d,e2d,m1d,m2d,sc1,sc2,e3,mc1,mc2);
```

```
    dff dff7(clk,rst,sc1,sc1d);
    dff dff8(clk,rst,sc2,sc2d);
    dff5bit dff9(clk,rst,e3,e3d);
    dff11bit dff11(clk,rst,mc1,mc1d);
    dff11bit dff12(clk,rst,mc2,mc2d);

    add_sub addsub(sc1d,sc2d,e3d,mc1d,mc2d,sr,e5,mf,add_sub,if_equal);

    dff dff13(clk,rst,sr,srd);
    dff5bit dff14(clk,rst,e5,e5d);
    dff12bit dff15(clk,rst,mf,mfd);

    normalize norma(srd,e5d,mfd,sr1,en,mn,if_equal,add_sub,if_zero);

    dff dff16(clk,rst,sr1,sr1d);
    dff5bit dff17(clk,rst,en,ed);
    dff11bit dff18(clk,rst,mn,mnd);

    ex_check excheck(sr1d,ed,mnd,sr2,ec,mc,st);

    dff dff19(clk,rst,sr2,sr2d);
    dff5bit dff20(clk,rst,ec,ecd);
    dff10bit dff21(clk,rst,mc,mcd);
    dff2bit dff22(clk,rst,st,std);

    packer pack(sr2d,ecd,mcd,std,out,status,valid);

endmodule
```

EK-B.1

```
`timescale 1ns / 1ps

module dff (clk,rst,d,q);
  input clk;
  input rst;
  input d;
  output reg q;

  always @(posedge clk)
        begin
          if(rst==1)
            begin
                q<=0;
            end
          else
            begin
                q<=d;
            end
        end

endmodule
```

EK-B.2

```
`timescale 1ns / 1ps

module dff2bit (clk,rst,d,q);
  input clk;
  input rst;
  input[1:0] d;
  output reg[1:0] q;

  always @(posedge clk)
        begin
          if(rst==1)
            begin
                q<=0;
            end
          else
            begin
                q<=d;
            end
        end

endmodule
```

EK-B.3

```
`timescale 1ns / 1ps

module dff5bit (clk,rst,d,q);
  input clk;
  input rst;
  input[4:0] d;
  output reg[4:0] q;

  always @(posedge clk)
        begin
          if(rst==1)
            begin
                q<=0;
            end
          else
            begin
                q<=d;
            end
        end

endmodule
```

EK-B.4

```
`timescale 1ns / 1ps

module dff10bit (clk,rst,d,q);
  input clk;
  input rst;
  input[9:0] d;
  output reg[9:0] q;

  always @(posedge clk)
        begin
          if(rst==1)
            begin
                q<=0;
            end
          else
            begin
                q<=d;
            end
        end

endmodule
```

EK-B.5

```
`timescale 1ns / 1ps

module dff11bit (clk,rst,d,q);
  input clk;
  input rst;
  input[10:0] d;
  output reg[10:0] q;

  always @(posedge clk)
        begin
          if(rst==1)
            begin
                q<=0;
            end
          else
            begin
                q<=d;
            end
        end

endmodule
```

EK-B.6

```
`timescale 1ns / 1ps

module dff12bit (clk,rst,d,q);
  input clk;
  input rst;
  input[11:0] d;
  output reg[11:0] q;

  always @(posedge clk)
        begin
          if(rst==1)
            begin
                q<=0;
            end
          else
            begin
                q<=d;
            end
        end

endmodule
```

EK-C

```
`timescale 1ns / 1ps

module unpack(a,b,s1,s2,e1,e2,m1,m2,if_equal,if_zero);
    input[15:0] a;
    input[15:0] b;
    output s1;
    output s2;
    output[4:0] e1;
    output[4:0] e2;
    output[10:0] m1;
    output[10:0] m2;
    output reg if_equal;
    output reg [1:0] if_zero;

    assign s1 = a[15];
    assign s2 = b[15];
    assign e1 = a[14:10];
    assign e2 = b[14:10];
    assign m1[10] = 1'b1;
    assign m1[9:0] = a[9:0];
    assign m2[10] = 1'b1;
    assign m2[9:0] = b[9:0];

    always @(a or b)
        begin
            if (a==b)
                begin
                    if_equal <= 1'b1;
                end
            else
                begin
                    if_equal <= 1'b0;
                end
        end


    always @(a or b)
        begin
            if (a[14:0]==0 && b[14:0]==0)
                begin
                    if_zero[0] <= 1'b1;
                    if_zero[1] <= 1'b1;
                end
            else if (a[14:0]==0)
                begin
                    if_zero[0] <= 1'b1;
                    if_zero[1] <= 1'b0;
                end
            else if (b[14:0]==0)
                begin
                    if_zero[0] <= 1'b0;
                    if_zero[1] <= 1'b1;
                end
            else
                begin
                    if_zero[0] <= 1'b0;
                    if_zero[1] <= 1'b0;
                end
        end


endmodule
```

EK-D

```
`timescale 1ns / 1ps
module comp_shift(s1d,s2d,e1d,e2d,m1d,m2d,sc1,sc2,e3,mc1,mc2);
    input s1d;
    input s2d;
    input[4:0] e1d;
    input[4:0] e2d;
    input[10:0] m1d;
    input[10:0] m2d;
    output sc1;
    output sc2;
    output reg[4:0] e3;
    output reg[10:0] mc1;
    output reg[10:0] mc2;

wire [5:0]temp_e1;
wire [5:0]temp_e2;
wire [5:0]temp_e1_e2;

assign sc1=s1d;
assign sc2=s2d;

assign temp_e1[5]=1'b0;
assign temp_e1[4:0]=e1d;

assign temp_e2[5]=1'b0;
assign temp_e2[4:0]=e2d;

assign temp_e1_e2=temp_e1+~temp_e2+1;


    always @(temp_e1_e2 or e1d or e2d)
        begin
            if (temp_e1_e2==0)
                begin
                    e3<=e1d;
                end
            else if(temp_e1_e2[5]==1)
                begin
                    e3<=e2d;
                end
            else
                begin
                    e3<=e1d;
                end
        end
```

```
    always @(temp_e1_e2 or m1d or m2d)
        begin
            if (temp_e1_e2[5]==0) begin
             case(temp_e1_e2)
             0 : begin mc1=m1d; mc2=m2d; end
             1 : begin mc1=m1d; mc2={1'b0,m2d[10:1]}; end
             2 : begin mc1=m1d; mc2={1'b0,1'b0,m2d[10:2]}; end
             3 : begin mc1=m1d; mc2={1'b0,1'b0,1'b0,m2d[10:3]}; end
             4 : begin mc1=m1d; mc2={1'b0,1'b0,1'b0,1'b0,m2d[10:4]}; end
             5 : begin mc1=m1d; mc2={1'b0,1'b0,1'b0,1'b0,1'b0,m2d[10:5]}; end
             6 : begin mc1=m1d; mc2={1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,m2d[10:6]}; end
             7 : begin mc1=m1d; mc2={1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,m2d[10:7]}; end
             8 : begin mc1=m1d; mc2={1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,m2d[10:8]}; end
             9 : begin mc1=m1d; mc2={1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,m2d[10:9]}; end
             10 : begin mc1=m1d; mc2={1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,m2d[10]}; end
             default : begin mc1=m1d; mc2=11'b00000000000; end
            endcase
             end
             else begin
             case(temp_e1_e2)
             0 : begin mc2=m2d; mc1=m1d; end
             63 : begin mc2=m2d; mc1={1'b0,m1d[10:1]}; end
             62 : begin mc2=m2d; mc1={1'b0,1'b0,m1d[10:2]}; end
             61 : begin mc2=m2d; mc1={1'b0,1'b0,1'b0,m1d[10:3]}; end
             60 : begin mc2=m2d; mc1={1'b0,1'b0,1'b0,1'b0,m1d[10:4]}; end
             59 : begin mc2=m2d; mc1={1'b0,1'b0,1'b0,1'b0,1'b0,m1d[10:5]}; end
             58 : begin mc2=m2d; mc1={1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,m1d[10:6]}; end
             57 : begin mc2=m2d; mc1={1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,m1d[10:7]}; end
             56 : begin mc2=m2d; mc1={1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,m1d[10:8]}; end
             55 : begin mc2=m2d; mc1={1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,m1d[10:9]}; end
             54 : begin mc2=m2d; mc1={1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,m1d[10]}; end
             default : begin mc2=m2d; mc1=11'b00000000000; end
            endcase
             end
        end

endmodule
```

EK-E.1

```
`timescale 1ns / 1ps

module add_sub(sc1d,sc2d,e3d,mc1d,mc2d,sr,e5,mf,add_sub,if_equal);
    input sc1d;
    input sc2d;
    input[4:0] e3d;
    //input[4:0] e4d;
    input[10:0] mc1d;
    input[10:0] mc2d;
    input add_sub;
   output reg sr;
    output [4:0] e5;
    output reg[11:0] mf;
    input if_equal;

   wire [12:0]temp_mc1d;
    wire [12:0]temp_mc2d;

    assign temp_mc1d[12:11]=2'b00;
    assign temp_mc2d[12:11]=2'b00;

    assign temp_mc1d[10:0]=mc1d;
    assign temp_mc2d[10:0]=mc2d;

   reg [12:0]temp_a;
    reg [12:0]temp_b;
    wire [12:0]temp_sum;
    wire [12:0]comp_temp_sum;

    always @(sc1d or temp_mc1d)
        begin
            if(sc1d==1)
                begin
                    temp_a= ~temp_mc1d+1;
                end
            else
              begin
                    temp_a= temp_mc1d;
            end
        end
```

```
    always @(sc2d or temp_mc2d)
        begin
            if(sc2d==1)
                begin
                  temp_b= ~temp_mc2d+1;
                end
            else
              begin
                  temp_b= temp_mc2d;
            end
        end

    ripple_add_sub addsub(temp_a,temp_b,temp_sum,add_sub);

     assign comp_temp_sum=~temp_sum[12:0]+1;

     assign e5=e3d;

     always @(temp_sum or comp_temp_sum or add_sub or if_equal or sc1d)
        begin
            if(add_sub==1 && if_equal==1)
                begin
                  sr=sc1d;
                  mf=12'b000000000000;
                end
            else if(temp_sum[12]==0)
                begin
                  sr=0;
                  mf=temp_sum[11:0];
                end
            else begin
                  sr=1;
                  mf=comp_temp_sum[11:0];
            end
        end

endmodule
```

EK-E.2

```
`timescale 1ns / 1ps

module ripple_add_sub(
    input [12:0] a,
    input [12:0] b,
    output [12:0] sum,
    input add_sub
    );

reg [12:0]b_twos_complement;


always @(add_sub or b)
        begin
            if(add_sub==1)
                begin
                  b_twos_complement = ~b;
                end
            else
              begin
                  b_twos_complement = b;
            end
        end


ripple_add RCAS(sum,a,b_twos_complement,add_sub);

endmodule
```

EK-E.3

```
`timescale 1ns / 1ps

module ripple_add(sum,a,b,cin);
    output [12:0] sum;
   input [12:0] a,b;
    input cin;
    wire [11:0] c;

    full_add FA0(sum[0],c[0],a[0],b[0],cin),
                FA1(sum[1],c[1],a[1],b[1],c[0]),
                FA2(sum[2],c[2],a[2],b[2],c[1]),
                FA3(sum[3],c[3],a[3],b[3],c[2]),
                FA4(sum[4],c[4],a[4],b[4],c[3]),
                FA5(sum[5],c[5],a[5],b[5],c[4]),
                FA6(sum[6],c[6],a[6],b[6],c[5]),
                FA7(sum[7],c[7],a[7],b[7],c[6]),
                FA8(sum[8],c[8],a[8],b[8],c[7]),
                FA9(sum[9],c[9],a[9],b[9],c[8]),
                FA10(sum[10],c[10],a[10],b[10],c[9]),
                FA11(sum[11],c[11],a[11],b[11],c[10]);

    assign sum[12] = a[12] ^ b[12] ^ c[11];

endmodule
```

EK-E.4

```
`timescale 1ns / 1ps

module full_add(sum,cout,a,b,cin);
    output sum,cout;
   input a,b,cin;

    assign sum = cin ^ a ^ b;
    assign cout = ~cin & a & b | cin & (a | b );

endmodule
```

EK-F

```
`timescale 1ns / 1ps

module normalize(srd,e5d,mfd,sr1,en,mn,if_equal,add_sub,if_zero);
    input srd;
    input[4:0] e5d;
    input[11:0] mfd;
    output  sr1;
    output reg [4:0] en;
    output reg [10:0] mn;
    input if_equal;
    input add_sub;
    input[1:0] if_zero;

    assign sr1 = srd;


    always @(mfd or e5d or if_equal or add_sub or if_zero)
        begin
            if(if_equal==1 && add_sub==1)
                begin
                    mn <= mfd[10:0];
                    en <= e5d;
                end
            else if(if_zero==2'b11 && add_sub ==0)
                begin
                    mn <= mfd[10:0];
                    en <= e5d;
                end
            else if(mfd[11]==1)
                begin
                  mn <= mfd[11:1];
                  en <= e5d+1;
                end
            else if(mfd[10]==1)
                begin
                  mn <= mfd[10:0];
                  en <= e5d;
                end
            else if(mfd[9]==1)
                begin
                  mn <= mfd[10:0]<<1;
                  en <= e5d-1;
                end
            else if(mfd[8]==1)
                begin
                  mn <= mfd[10:0]<<2;
                  en <= e5d-2;
                end
```

```
            else if(mfd[7]==1)
                begin
                  mn <= mfd[10:0]<<3;
                  en <= e5d-3;
                end
            else if(mfd[6]==1)
                begin
                  mn <= mfd[10:0]<<4;
                  en <= e5d-4;
                end
            else if(mfd[5]==1)
                begin
                  mn <= mfd[10:0]<<5;
                  en <= e5d-5;
                end
            else if(mfd[4]==1)
                begin
                  mn <= mfd[10:0]<<6;
                  en <= e5d-6;
                end
            else if(mfd[3]==1)
                begin
                  mn <= mfd[10:0]<<7;
                  en <= e5d-7;
                end
            else if(mfd[2]==1)
                begin
                  mn <= mfd[10:0]<<8;
                  en <= e5d-8;
                end
            else if(mfd[1]==1)
                begin
                  mn <= mfd[10:0]<<9;
                  en <= e5d-9;
                end
            else
                begin
                  mn <= mfd[10:0]<<10;
                  en <= e5d-10;
                end

        end


endmodule
```

EK-G

```
`timescale 1ns / 1ps

module ex_check(sr1d,ed,mnd,sr2,ec,mc,st);
    input sr1d;
    input[4:0] ed;
    input[10:0] mnd;
    output sr2;
    output reg[4:0] ec;
    output reg[9:0] mc;
    output reg[1:0] st;

    assign sr2 = sr1d;

    always @(ed or mnd)
        begin

            if(ed==31)
                begin
                    st<= 2'b01;
                    ec<=5'b11110;
                    mc<=10'b1111111111;
                end
            else if(ed==1)
                begin
                    st<=2'b10;
                    ec<=ed;
                    mc<=mnd[9:0];
                end
            else if(ed==0 && mnd==0)
                begin
                    st<=2'b00;
                    ec<=5'b00000;
                    mc<=10'b0000000000;
                end
            else
                begin
                    st<=2'b11; /
                    ec<=ed;
                    mc<=mnd[9:0];
                end

        end


endmodule
```

EK-H

```
`timescale 1ns / 1ps
module packer(sr2d,ecd,mcd,std,out,status,valid);
    input sr2d;
    input[4:0] ecd;
    input[9:0] mcd;
    input[1:0] std;
    output[15:0] out;
    output[2:0] status;
    output reg valid;

    assign out = {sr2d,ecd,mcd[9:0]};
    assign status[2] = 0;
    assign status[1:0] = std;

    always@*
    begin
        if ( out == {sr2d,ecd,mcd[9:0]})
        valid <= 1;
    else
        valid <= 0;
    end
    end

endmodule
```

EK-I

```
/*-----------------------------------------------------------------------------
 Description   : Top testbench module - Contains DUT,interface and test components
-----------------------------------------------------------------------------*/

`include "float_add.v"     // Import DUT
`include "FP_ALU_ADDER_if.sv"  // Import Interface
`include "FP_ALU_ADDER_package.svh"  // Import all other components which are
                                     // required as submodules

module FP_ALU_ADDER_tb_top;

  // Import UVM Package
  import uvm_pkg::*;

  // Import the FP_ALU_ADDER UVC Package
  import FP_ALU_ADDER_package::*;

  // Include the test library
  `include "FP_ALU_ADDER_test_lib.sv"

  // DUT I/O regs
  reg [15:0] a;
  reg [15:0] b;
  reg [15:0] out;
  reg [2:0] status;
  reg valid;
  reg clk;
  reg rst;
  reg add_sub;

  // Interface instance to be connected with DUT
  FP_ALU_ADDER_if dut_if();

  // Create the DUT, connect it with the interface
  FP_ALU_ADDER dut(
    .a(dut_if.a),
    .b(dut_if.b),
    .out(dut_if.out),
    .status(dut_if.status),
    .valid(dut_if.valid),
    .clk(dut_if.sig_clock),
    .rst(dut_if.sig_reset),
    .add_sub(dut_if.add_sub)
  );
```

```
  initial begin
    uvm_config_db #(virtual FP_ALU_ADDER_if)::set(null, "uvm_test_top", "vif", dut_if);
    run_test();
  end

  initial begin
    dut_if.sig_reset <= 1'b1;
    dut_if.sig_clock <= 1'b1;
    #21 dut_if.sig_reset <= 1'b0;
  end

  // Clock Generator
  always
    #20 dut_if.sig_clock = ~dut_if.sig_clock;

  initial
  begin
    $recordfile ("./test.trn");
    $recordvars ("depth=0", FP_ALU_ADDER_tb_top);
  end

endmodule
```

EK-J

```
/*-------------------------------------------------------------------------
Description  : Establishes connection between test environment and DUT
-------------------------------------------------------------------------*/

// Define interface
interface FP_ALU_ADDER_if ();

  // Import UVM macros and package
  import uvm_pkg::*;
  `include "uvm_macros.svh"

  // Interface I/O - DUT signals will be connected to these
    logic sig_clock;
    logic sig_reset;
    logic add_sub;
//  logic enable;
    logic valid;
    logic [15:0] a;
    logic [15:0] b;
    logic [15:0] out;
    logic [2:0] status;

  // Control flags
  bit has_checks = 1;
  bit has_coverage = 1;

  // SVA default clocking
  wire uvm_assert_clk = sig_clock && has_checks;
  default clocking master_clk @(negedge uvm_assert_clk);
  endclocking

  // SVA Default reset
  default disable iff (sig_reset);

  // Data must not be X or Z during Data Phase (when valid is raised)
  assertValidAndData: assert property (
    ($rose(valid) |=> !$isunknown(out)))
    else
   `uvm_error("ERR_DATA_XZ", "Data went to X or Z during Data Phase")

endinterface : FP_ALU_ADDER_if
```

EK-K

```
/*-----------------------------------------------------------------------------------------------
 Description: Package is where all the files that used in UVC (Universal Verificatin Component)
 are imported
-----------------------------------------------------------------------------------------------*/

// Import UVM macros
`include "uvm_macros.svh"
//Import Matlab DPI
`include "matlab_dpi_pkg.svh"

// Package Definition
package FP_ALU_ADDER_package;

    // UVM class library compiled in a package
    import uvm_pkg::*;

    `include "FP_ALU_ADDER_defines.sv"
    `include "FP_ALU_ADDER_seq_item.sv"
    `include "FP_ALU_ADDER_monitor.sv"
    `include "FP_ALU_ADDER_driver.sv"
    `include "FP_ALU_ADDER_agent.sv"
    `include "FP_ALU_ADDER_fc_subscriber.sv"
    `include "FP_ALU_ADDER_sb_subscriber.sv"

    `include "FP_ALU_ADDER_seq_lib.sv"
    `include "FP_ALU_ADDER_vc_env.sv"

endpackage : FP_ALU_ADDER_package
```

EK-L.1

```
/*-------------------------------------------------------------------------
 Description   : This file implements two kinds of test in the testbench.
    A test file verifies one or more cases in the test plan.
-------------------------------------------------------------------------*/

`include "FP_ALU_ADDER_top_env.sv"

class FP_ALU_ADDER_base_test extends uvm_test;

  `uvm_component_utils(FP_ALU_ADDER_base_test)

  FP_ALU_ADDER_top_env top_env0;
  uvm_table_printer printer;

  function new(string name = "FP_ALU_ADDER_base_test", uvm_component parent);
    super.new(name,parent);
    printer = new();
  endfunction : new

  // UVM build() phase ---------------------------------------------------
  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    // Enable transaction recording for everything
    set_config_int("*", "recording_detail", UVM_FULL);
    // Create the testbench
    top_env0 = FP_ALU_ADDER_top_env::type_id::create("top_env0", this);
  endfunction

  // UVM start_of_simulation() phase
  virtual function void start_of_simulation_phase(uvm_phase phase);
    super.start_of_simulation_phase(phase);
    printer.knobs.depth = 5;
    this.print(printer);
  endfunction

endclass : FP_ALU_ADDER_base_test
```

```
//------------------------------------------------------------------------------------------
//
// TEST: FP_ALU_ADDER_first_test - sets the first sequences
//
//------------------------------------------------------------------------------------------
class FP_ALU_ADDER_first_test extends FP_ALU_ADDER_base_test;
  `uvm_component_utils(FP_ALU_ADDER_first_test)

  function new(string name = "FP_ALU_ADDER_first_test", uvm_component parent);
    super.new(name,parent);
  endfunction

  // UVM build() phase
  virtual function void build_phase(uvm_phase phase);
    int unsigned itr;

    super.build_phase(phase);
  endfunction
  extern virtual task run_phase(uvm_phase phase);
endclass

// ------------------------------------------------------------------------------------------

task FP_ALU_ADDER_first_test::run_phase(uvm_phase phase);

   FP_ALU_ADDER_illegal_transmit_seq seq;

  `uvm_info(get_type_name(),"In build() function of FP_ALU_ADDER_first_test class", UVM_MEDIUM)
   seq = FP_ALU_ADDER_illegal_transmit_seq::type_id::create("seq");
   // -----------------
   assert(seq.randomize());
   // -----------------
   phase.raise_objection(this, "Starting the sequence.");
   // -----------------
   seq.start ( top_env0.FP_ALU_ADDER.agent_inst.sequencer );
   // -----------------
   phase.drop_objection(this, "Finishing the sequence.");

endtask : run_phase
```

EK-L.2

```
// Derive top environment from umv_env class
class FP_ALU_ADDER_top_env extends uvm_env;

  // Provide implementations of virtual methods such as get_type_name and create
  `uvm_component_utils(FP_ALU_ADDER_top_env)

  // FP_ALU_DIVIDER environment
  FP_ALU_ADDER_vc_env FP_ALU_ADDER;

  // Constructor - required syntax for UVM automation and utilities
  function new (string name, uvm_component parent);
    super.new(name, parent);
  endfunction

// Additional class methods
// UVM build() phase
function void build_phase(uvm_phase phase);
  super.build_phase(phase);

  // set vif property for child elements
  uvm_config_db#(virtual FP_ALU_ADDER_if)::set(this,"*","vif",FP_ALU_ADDER_tb_top.dut_if);
  FP_ALU_ADDER = FP_ALU_ADDER_vc_env::type_id::create("FP_ALU_ADDER", this);
endfunction

// UVM start_of_simulation() phase
function void start_of_simulation_phase(uvm_phase phase);
  super.start_of_simulation_phase(phase);
  uvm_test_done.set_drain_time(this, 1000);
endfunction
endclass
```

EK-L.3

```
// Derive UVC environment from umv_env class
class FP_ALU_ADDER_vc_env extends uvm_env;

  // Virtual Interface variable
  protected virtual interface FP_ALU_ADDER_if vif;

  // The following two bits are used to control whether checks and coverage are
  // done both in the bus monitor class and the interface.
  bit checks_enable = 1;
  bit coverage_enable = 1;

  // Components of the environment
   FP_ALU_ADDER_agent          agent_inst;
   FP_ALU_ADDER_fc_subscriber  fc_sub;
   FP_ALU_ADDER_scoreboard     scoreboard;

  /* Call macros to provide attributes for environment. uvm_component_utils_begin
     macro should be updated to get required utilities. */
  `uvm_component_utils_begin(FP_ALU_ADDER_vc_env)
    `uvm_field_int(checks_enable, UVM_DEFAULT)
    `uvm_field_int(coverage_enable, UVM_DEFAULT)
  `uvm_component_utils_end

  // Constructor - required syntax for UVM automation and utilities
  function new(string name, uvm_component parent);
    super.new(name, parent);
  endfunction : new

  // Additional class methods
  extern virtual function void build_phase(uvm_phase phase);
  extern virtual function void connect_phase(uvm_phase phase);
  extern protected task update_vif_enables();
  extern virtual task run_phase(uvm_phase phase);

endclass : FP_ALU_ADDER_vc_env
```

```
  // UVM build() phase
  function void FP_ALU_ADDER_vc_env::build_phase(uvm_phase phase);
    super.build();
    agent_inst = FP_ALU_ADDER_agent::type_id::create("agent_inst", this);
    fc_sub = FP_ALU_ADDER_fc_subscriber::type_id::create(.name("fc_sub"), .parent(this));
    scoreboard = FP_ALU_ADDER_scoreboard::type_id::create(.name("scoreboard"), .parent(this));

    if(!uvm_config_db#(virtual FP_ALU_ADDER_if)::get(this,"","vif",vif))
    `uvm_error("NOVIF","virtual if not configured");
  endfunction

 // UVM connect() phase
   function void FP_ALU_ADDER_vc_env::connect_phase(uvm_phase phase);
      super.connect();
      agent_inst.FP_ALU_ADDER_ap.connect(fc_sub.analysis_export);
      agent_inst.FP_ALU_ADDER_ap.connect(scoreboard.FP_ALU_ADDER_analysis_export);
   endfunction

  // ---------------------------------------------------------------------------------------------
  // TASK -> update_vif_enables() -> Function to assign the checks and coverage bits
  // ---------------------------------------------------------------------------------------------
  task FP_ALU_ADDER_vc_env::update_vif_enables();
    // Make assignments at time zero based upon config
    vif.has_checks <= checks_enable;
    vif.has_coverage <= coverage_enable;
    forever begin
      // Make assignments whenever enables change after time zero
      @(checks_enable || coverage_enable);
      vif.has_checks <= checks_enable;
      vif.has_coverage <= coverage_enable;
    end
  endtask : update_vif_enables

  // UVM run() phase
  task FP_ALU_ADDER_vc_env::run_phase(uvm_phase phase);
    super.run_phase(phase);
    fork
      update_vif_enables();
    join_none
  endtask
```

EK-M

```
/*-----------------------------------------------------------------
Description   :  Specifies the sequence item. Declare constraints
                 depending the verification strategy
-----------------------------------------------------------------*/

class FP_ALU_ADDER_seq_item extends uvm_sequence_item;

rand bit [15:0] a;
rand bit [15:0] b;
rand bit add_sub;
bit      [15:0] out;
bit      [2:0] status;
bit      [1:0] output_timing; // 00 --> LEGAL_OUTPUT_TIMING, 01 --> MISSING_OUTPUT, 10 --> EARLY_OUTPUT_TIMING, 11 --> LATE_OUTPUT_TIMING

`uvm_object_utils_begin(FP_ALU_ADDER_seq_item)
   `uvm_field_int(a, UVM_DEFAULT)
   `uvm_field_int(b, UVM_DEFAULT)
   `uvm_field_int(out, UVM_DEFAULT)
   `uvm_field_int(status, UVM_DEFAULT)
`uvm_object_utils_end


  // Constructor - required syntax for UVM automation and utilities
  function new (string name = "FP_ALU_ADDER_seq_item");
    super.new(name);
  endfunction : new

endclass : FP_ALU_ADDER_seq_item
```

EK-N

```
/********************** legal transmit sequence ****************************/

// Extend sequence from uvm_sequence class - using seq_item
class FP_ALU_ADDER_legal_transmit_seq extends uvm_sequence #(FP_ALU_ADDER_seq_item);

  `uvm_object_utils(FP_ALU_ADDER_legal_transmit_seq )

  // Constructor
  function new(string name="FP_ALU_ADDER_legal_transmit_seq ");
    super.new(name);
  endfunction

  // Sequence body definition
  virtual task body();
    begin
      `uvm_info(get_type_name(), "Executing...", UVM_MEDIUM)
        `uvm_do_with(req, { a <= 5; b >= 4;} )
    end
  endtask

endclass

/********************** illegal transmit sequence ****************************/

class FP_ALU_ADDER_illegal_transmit_seq  extends uvm_sequence #(FP_ALU_ADDER_seq_item);
  `uvm_object_utils(FP_ALU_ADDER_illegal_transmit_seq)

  // Sequence that will be called in this sequence
  FP_ALU_ADDER_legal_transmit_seq FP_ALU_ADDER_seq;

  // Parameter for this sequence

  // Constructor
  function new(string name="FP_ALU_ADDER_illegal_transmit_seq");
    super.new(name);
  endfunction

  // Sequence body definition
  virtual task body();
    uvm_component parent = get_sequencer();
    begin
      for(int i = 0; i < 10; i++) begin
        `uvm_do(FP_ALU_ADDER_seq)
      end
    end
  endtask

endclass
```

EK-O

```
// Derive an agent from uvm_agent class
class FP_ALU_ADDER_agent extends uvm_agent;

  FP_ALU_ADDER_monitor  monitor;
  FP_ALU_ADDER_driver   driver;
  /* Derive and define a sequencer from uvm_sequencer class that uses seq_item,
     called FP_ALU_DIVIDER_sequencer, then create an instance */
  typedef uvm_sequencer #(FP_ALU_ADDER_seq_item) FP_ALU_ADDER_sequencer;
  FP_ALU_ADDER_sequencer sequencer;

  // Creating analysis port
  uvm_analysis_port #(FP_ALU_ADDER_seq_item) FP_ALU_ADDER_ap;

  /* Call macros to provide attributes for agent. uvm_component_utils_begin macro
     should be updated to get required utilities. */
  `uvm_component_utils_begin(FP_ALU_ADDER_agent)
    `uvm_field_enum(uvm_active_passive_enum, is_active, UVM_DEFAULT)
  `uvm_component_utils_end

  // Constructor - required syntax for UVM automation and utilities
  function new (string name, uvm_component parent);
    super.new(name, parent);
    FP_ALU_ADDER_ap = new("FP_ALU_ADDER_ap",this);
  endfunction : new

  // UVM build() phase
  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    monitor = FP_ALU_ADDER_monitor::type_id::create("monitor", this);
    if(is_active == UVM_ACTIVE) begin
      sequencer = FP_ALU_ADDER_sequencer::type_id::create("sequencer", this);
      driver    = FP_ALU_ADDER_driver::type_id::create("driver", this);
    end
  endfunction

  // UVM connect() phase
  function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);
    monitor.item_collected_port.connect(FP_ALU_ADDER_ap); //connect monitor to agent via analysis port
    if(is_active == UVM_ACTIVE) begin
      // Binds the driver to the sequencer using consumer-producer interface
      driver.seq_item_port.connect(sequencer.seq_item_export);
    end
  endfunction

endclass : FP_ALU_ADDER_agent
```

EK-P

```
/*-----------------------------------------------------------------
Description   : This files implements the driver functionality.
-----------------------------------------------------------------*/

class FP_ALU_ADDER_driver extends uvm_driver #(FP_ALU_ADDER_seq_item);

 /*****************************************************************************
  IVB-NOTE : REQUIRED : DRIVER functionality : DRIVER
  ---------------------------------------------------------------------------
  Modify the following methods to match your protocol:
    o drive_seq_item() - Handshake and seq_item driving process
    o reset_signals() - signal reset values
  Note that if you change/add signals to the physical interface, you must
  also change these methods.
  *****************************************************************************/

  // The virtual interface used to drive and view HDL signals.
  protected virtual interface FP_ALU_ADDER_if vif;

  FP_ALU_ADDER_seq_item seq_item;

  // Count seq_items sent
  int num_sent;

  // Provide implementationsmentations of virtual methods such as get_type_name and create
  `uvm_component_utils(FP_ALU_ADDER_driver)

  // Constructor - required syntax for UVM automation and utilities
  function new (string name, uvm_component parent);
    super.new(name, parent);
  endfunction : new

  // Additional class methods
  extern virtual task run_phase(uvm_phase phase);
  extern virtual protected task get_and_drive();
  extern virtual protected task reset_signals();
  extern virtual protected task drive_seq_item();
  extern virtual function void report_phase(uvm_phase phase);

  // UVM build() phase
  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);

    if(!uvm_config_db#(virtual FP_ALU_ADDER_if)::get(this,"","vif",vif))
    `uvm_error(get_type_name(),"virtual if not configured");

  endfunction
endclass : FP_ALU_ADDER_driver
```

```
// UVM run() phase
task FP_ALU_ADDER_driver::run_phase(uvm_phase phase);
      super.run_phase(phase);

  fork
    get_and_drive();
    reset_signals();
  join

endtask

// ------------------------------------------------------------------------------------------
// TASK -> get_and_drive() ->Gets seq_items from the sequencer and passes them to the driver.
// ------------------------------------------------------------------------------------------
task FP_ALU_ADDER_driver::get_and_drive();


  `uvm_info(get_type_name(), "Reset not rised", UVM_MEDIUM)
  @(negedge vif.sig_reset);
  `uvm_info(get_type_name(), "Reset riseed", UVM_MEDIUM)
  forever begin
    @(posedge vif.sig_clock);
    `uvm_info(get_type_name(), "FP_ALU_ADDER_driver inside get_and_drive task 1", UVM_MEDIUM)
    // Get new item from the sequencer
    seq_item_port.get_next_item(seq_item);
    `uvm_info(get_type_name(), "FP_ALU_ADDER_driver inside get_and_drive task 2", UVM_MEDIUM)
    // Drive the item
    drive_seq_item();
      `uvm_info(get_type_name(), "FP_ALU_ADDER_driver inside get_and_drive task 3", UVM_MEDIUM)
    // Communicate item done to the sequencer
    seq_item_port.item_done();
    `uvm_info(get_type_name(), "FP_ALU_ADDER_driver inside get_and_drive task 4", UVM_MEDIUM)
  end
endtask : get_and_drive

// ------------------------------------------------------------------------------------------
// TASK -> reset_signals() ->Reset all signals.
// ------------------------------------------------------------------------------------------
task FP_ALU_ADDER_driver::reset_signals();
  forever begin
    @(posedge vif.sig_reset);
    `uvm_info(get_type_name(), "Reset observed", UVM_MEDIUM)
  vif.a       <= 'hz;
  vif.b       <= 'hz;
  vif.add_sub <= 'hz;
  end
endtask : reset_signals
```

```
// ---------------------------------------------------------------------------
// TASK -> drive_seq_item() ->Gets a seq_item and drive it into the DUT
// ---------------------------------------------------------------------------
task FP_ALU_ADDER_driver::drive_seq_item();
    `uvm_info(get_type_name(), "Inside drive_seq_item1", UVM_MEDIUM)
  //vif.enable <= 1'b1;
  vif.a <= seq_item.a;
  vif.b <= seq_item.b;
  vif.add_sub <= seq_item.add_sub;
  @(posedge vif.sig_clock);
      `uvm_info(get_type_name(), "Inside drive_seq_item2", UVM_MEDIUM)

      #160;
      vif.a<= 'hz;
      vif.b<= 'hz;
      vif.add_sub <= 'hz;
      @(posedge vif.sig_clock);

   num_sent++;
   `uvm_info(get_type_name(), $sformatf("Item %0d Sent ...", num_sent),
    UVM_HIGH)
endtask : drive_seq_item

// UVM report() phase
function void FP_ALU_ADDER_driver::report_phase(uvm_phase phase);
  `uvm_info(get_type_name(),
    $sformatf("\nReport: FP_ALU_ADDER driver sent %0d seq_items",
    num_sent), UVM_LOW)
    stop_matlab();
endfunction
```

EK-R

```
// Derive monitor from uvm_monitor
class FP_ALU_ADDER_monitor extends uvm_monitor;

  // Virtual Interface for monitoring DUT signals
  protected virtual interface FP_ALU_ADDER_if vif;

  int num_col; // To hold collected seq_item count

  // The following two bits are used to control whether checks and coverage are
  // done in the monitor
  bit checks_enable = 1;
  bit coverage_enable = 1;

  // This TLM port is used to connect the monitor to the scoreboard
  uvm_analysis_port #(FP_ALU_ADDER_seq_item) item_collected_port;

  // Currently monitored seq_item
  protected FP_ALU_ADDER_seq_item seq_item;

  // Covergroup for seq_item
  covergroup seq_item_cg;
    option.per_instance = 1;
  endgroup : seq_item_cg

  // Provide UVM automation and utility methods
  `uvm_component_utils_begin(FP_ALU_ADDER_monitor)
    `uvm_field_int(checks_enable, UVM_DEFAULT)
    `uvm_field_int(coverage_enable, UVM_DEFAULT)
  `uvm_component_utils_end

  // Constructor - required syntax for UVM automation and utilities
  function new (string name, uvm_component parent);
    super.new(name, parent);
    // Create the covergroup only if coverage is enabled
    void'(get_config_int("coverage_enable", coverage_enable));
    if (coverage_enable) begin
      seq_item_cg = new();
      seq_item_cg.set_inst_name("seq_item_cg");
    end
    // Create the TLM port
    item_collected_port = new("item_collected_port", this);
  endfunction : new
```

```
  // Additional class methods
  extern virtual task run();
  extern virtual protected task collect_seq_item();
  extern virtual protected function void perform_checks();
  extern virtual protected function void perform_coverage();
  extern virtual function void report_phase(uvm_phase phase);

  // UVM build() phase
  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);

    if(!uvm_config_db#(virtual FP_ALU_ADDER_if)::get(this,"","vif",vif))
    `uvm_error(get_type_name(),"virtual if not configured");

  endfunction
endclass : FP_ALU_ADDER_monitor

  // UVM run() phase
  task FP_ALU_ADDER_monitor::run();
    fork
      collect_seq_item();
    join_none
  endtask : run

  /*****************************************************************************
   IVB-NOTE : REQUIRED : Monitor : Monitors
   ---------------------------------------------------------------------------
   Modify the collect_seq_items() method to match your protocol.
   Note that if you change/add signals to the physical interface, you must
   also change this method.
   *****************************************************************************/

  // ---------------------------------------------------------------------------------
  // TASK -> collect_seq_item() -> This monitor re-uses its data items for ALL seq_items
  // ---------------------------------------------------------------------------------
  task FP_ALU_ADDER_monitor::collect_seq_item();
    seq_item = FP_ALU_ADDER_seq_item::type_id::create("seq_item", this);
    forever begin
    @(posedge vif.sig_clock iff vif.valid === 1);
      // Begin transaction recording
      void'(begin_tr(seq_item, "FP_ALU_ADDER Monitor"));

    seq_item.a = vif.a;
    seq_item.b = vif.b;
    seq_item.add_sub = vif.add_sub;
    seq_item.out = vif.out;
    seq_item.status = vif.status;
```

```
    @(posedge vif.sig_clock iff vif.valid === 0);

    // End transaction recording
    end_tr(seq_item);
    `uvm_info(get_type_name(),
      $sformatf("seq_item collected :\n%s",
      seq_item.sprint()), UVM_HIGH)
    if (checks_enable)
      perform_checks();
    if (coverage_enable)
      perform_coverage();
    // Send seq_item to scoreboard via TLM write()
    item_collected_port.write(seq_item);
    num_col++;
  end
endtask : collect_seq_item

// perform_seq_adder_seq_item_checks
function void FP_ALU_ADDER_monitor::perform_checks();
  // Add checks here
endfunction : perform_checks

/*******************************************************************************
IVB-NOTE : OPTIONAL : Monitor Coverage : Coverage
--------------------------------------------------------------------------------
Modify the seq_item_cg coverage group to match your protocol.
Add new coverage groups, and edit the perform_coverage() method to sample
them.
*******************************************************************************/

// Triggers coverage events
function void FP_ALU_ADDER_monitor::perform_coverage();
  seq_item_cg.sample();
endfunction : perform_coverage

// UVM report() phase
function void FP_ALU_ADDER_monitor::report_phase(uvm_phase phase);
  `uvm_info(get_type_name(),
    $sformatf("\nReport: FP_ALU_ADDER monitor collected %0d seq_items", num_col),
    UVM_LOW)
endfunction
```

EK-S.1

```
typedef class FP_ALU_ADDER_scoreboard;

class FP_ALU_ADDER_sb_subscriber extends uvm_subscriber #(FP_ALU_ADDER_seq_item);
`uvm_component_utils(FP_ALU_ADDER_sb_subscriber)

function new( string name , uvm_component parent);
super.new( name , parent );
endfunction:new

function void write(FP_ALU_ADDER_seq_item t);
    FP_ALU_ADDER_scoreboard FP_ALU_ADDER_sb;
    /*********************************/
    FP_ALU_ADDER_seq_item m_out_item;
    string msg, cmd, cmd_rsp1,cmd_rsp2;

    m_out_item = FP_ALU_ADDER_seq_item::type_id::create("m_out_item");

    //t is the input sequence item (transaction). Process t and then
    // write the new processed output to the m_output_ap.
    m_out_item.do_copy(t);

    $sformat(msg, "INPUT: A = %s, B = %s, add_sub = %s",t.a.convert2string(), t.b.convert2string(), t.add_sub.convert2string());
    `uvm_info(get_name(),msg, UVM_HIGH);

    $sformat(cmd, "result = float16add(%0d,%0d,%0d);", t.a, t.b, t.add_sub);
    `uvm_info(get_name(), cmd, UVM_HIGH);

    // Call our MATLAB function with our transaction inputs
    void'(send_matlab_cmd(cmd));

    // Readback the MATLAB buffer with our output
    cmd_rsp1 = get_matlab_buffer();

    `uvm_info(get_name(), $sformatf("MATLAB Buffer is %s", cmd_rsp1), UVM_HIGH);

    if (!$sscanf(cmd_rsp1, ">> %d", m_out_item.division_result)) begin
        `uvm_warning(get_name(), "Error parsing MATLAB response");
    end

    $sformat(cmd, "flg = float16add(%0d,%0d,%0d);", t.a, t.b, t.add_sub);
    `uvm_info(get_name(), cmd, UVM_HIGH);

    // Call our MATLAB function with our transaction inputs
    void'(send_matlab_cmd(cmd));

    // Readback the MATLAB buffer with our output
    cmd_rsp2 = get_matlab_buffer();
```

```
    `uvm_info(get_name(), $sformatf("MATLAB Buffer is %s", cmd_rsp2), UVM_HIGH);

    if (!$sscanf(cmd_rsp2, ">> %d", m_out_item.status)) begin
       `uvm_warning(get_name(), "Error parsing MATLAB response");
    end

    m_output_ap.write(m_out_item);
    /**********************************/

      $cast( FP_ALU_ADDER_sb, m_parent );
      FP_ALU_ADDER_sb.check_FP_ALU_ADDER_checker(t);
   endfunction: write

endclass:FP_ALU_ADDER_sb_subscriber

//------------------------------------------------------------------------------
//
// CLASS: FP_ALU_ADDER_scoreboard
//
//------------------------------------------------------------------------------
class FP_ALU_ADDER_scoreboard extends uvm_scoreboard;
   `uvm_component_utils(FP_ALU_ADDER_scoreboard)

   uvm_analysis_export#(FP_ALU_ADDER_seq_item) FP_ALU_ADDER_analysis_export;
   local FP_ALU_ADDER_sb_subscriber FP_ALU_ADDER_sb_sub;

   function new(string name, uvm_component parent);
      super.new(name, parent);
   endfunction: new

   function void build_phase(uvm_phase phase);
      super.build_phase(phase);

      if (!start_matlab("matlab -nosplash")) begin
        `uvm_fatal(get_name(), "Unable to start MATLAB");
      end

      void'(send_matlab_cmd("addpath ./MATLAB;"));
      m_output_ap = new("m_output_ap", this);

      FP_ALU_ADDER_analysis_export = new( .name("FP_ALU_ADDER_analysis_export"), .parent(this));
      FP_ALU_ADDER_sb_sub = FP_ALU_ADDER_sb_subscriber::type_id::create(.name("FP_ALU_ADDER_sb_sub"), .parent(this));
   endfunction: build_phase

   function void connect_phase(uvm_phase phase);
      super.connect_phase(phase);
      FP_ALU_ADDER_analysis_export.connect(FP_ALU_ADDER_sb_sub.analysis_export);
   endfunction: connect_phase
```

```
//-------------------------------------------------------------------------------------------
//TASK => check_FP_ALU_ADDER_checker()
//-------------------------------------------------------------------------------------------

   virtual function void check_FP_ALU_ADDER_checker(FP_ALU_ADDER_seq_item FP_ALU_ADDER_tx);
     uvm_table_printer p = new;

    shortint out_expected;    //expected value of output
    shortint out_actual;       //actual value of output
    int status_expected;    //expected value of output status
    int status_actual;       //actual value of output status

    out_expected = m_out_item.out;
    out_actual = FP_ALU_ADDER_tx.out;

    status_expected = m_out_item.status;
    status_actual = FP_ALU_ADDER_tx.status;


    //simple comparator
    if ((out_expected == out_actual) && (status_expected == status_actual)) begin
        `uvm_info("out_scoreboard",
                  { "s=a+b.\n", FP_ALU_ADDER_tx.sprint(p) }, UVM_LOW);
        end
    else begin
          `uvm_error("out_scoreboard",
                    { "s/=a+b!\n", FP_ALU_ADDER_tx.sprint(p) });
        end

   endfunction: check_FP_ALU_ADDER_checker
endclass: FP_ALU_ADDER_scoreboard
```

EK-S.2

```
/*-------------------------------------------------------
Description   : functional coverage subscriber
-------------------------------------------------------*/

// Derive from uvm_subscriber class
class FP_ALU_ADDER_fc_subscriber extends uvm_subscriber #(FP_ALU_ADDER_seq_item);
// Include utility macros
`uvm_component_utils(FP_ALU_ADDER_fc_subscriber)

FP_ALU_ADDER_seq_item pkt;

int pkt_cnt;

covergroup cov1;
a_cov: coverpoint pkt.a {bins a[8]  = {[0:15]};} //coverage register
b_cov: coverpoint pkt.b {bins b[8] = {[0:15]};}  //coverage register
cross a_cov, b_cov;
endgroup : cov1

function new( string name , uvm_component parent);
super.new( name , parent );
cov1 = new();
endfunction

function void write(FP_ALU_ADDER_seq_item t);
real current_FP_ALU_ADDER_fc_subscriber;
pkt = t;
pkt_cnt++;
cov1.sample();
// cause sampling of covergroup
current_FP_ALU_ADDER_fc_subscriber = $get_coverage();

    uvm_report_info("FP_ALU_ADDER_FC_SUBSCRIBER",$psprintf("%0d FP_ALU_ADDER_seq_items sampled, FP_ALU_ADDER_fc_subscriber = %f%% ", pkt_cnt,current_FP_ALU_ADDER_fc_subscriber));
endfunction
endclass:FP_ALU_ADDER_fc_subscriber
```

ÖZGEÇMİŞ

Adı Soyadı: Sait Furkan Teke

Doğum Yeri ve Tarihi: İstanbul, 1994

Lise: Mustafa Kemal Anadolu Öğretmen Lisesi, 2012

Lisans: İstanbul Teknik Üniversitesi, Elektronik ve Haberleşme Mühendisliği, 2016