T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YAPAY SİNİR AĞLARININ UYARLANABİLİR DONANIMSAL YAPILARDA GERÇEKLENMESİ

DOKTORA TEZİ

Onursal ÇETİN

Enstitü Anabilim Dalı	**:**	**ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ**
Enstitü Bilim Dalı	**:**	**ELEKTRONİK**
Tez Danışmanı	**:**	**Prof. Dr. Etem KÖKLÜKAYA**

Ekim 2014

T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YAPAY SİNİR AĞLARININ UYARLANABİLİR DONANIMSAL YAPILARDA GERÇEKLENMESİ

DOKTORA TEZİ

Onursal ÇETİN

Enstitü Anabilim Dalı : ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ

Bu tez 31/10/2014 tarihinde aşağıdaki jüri tarafından Oybirliği ile kabul edilmiştir.

Prof. Dr. **Etem KÖKLÜKAYA** **Jüri Başkanı**	**Prof. Dr.** **Nejat YUMUŞAK** **Üye**	**Yrd. Doç. Dr.** **M. Recep BOZKURT** **Üye**
Yrd. Doç. Dr. **Orhan ER** **Üye**	**Yrd. Doç. Dr.** **Halit ÖZTEKİN** **Üye**	

ÖNSÖZ

Tez çalışmam süresince bana yol gösteren, bilgi ve tecrübelerini benimle paylaşan değerli hocalarım Sayın Prof. Dr. Etem KÖKLÜKAYA ve Sayın Prof. Dr. Feyzullah TEMURTAŞ'a sonsuz teşekkürlerimi sunarım.

Ayrıca haklarını hiçbir zaman ödeyemeyeceğim sevgili anneme ve babama; desteğini, güler yüzünü benden esirgemeyen sevgili eşime ve tüm aileme teşekkür ederim.

İÇİNDEKİLER

SİMGELER VE KISALTMALAR LİSTESİ

ADC	: Analog-dijital dönüştürücü
AND	: VE kapısı
ANNs	: Yapay sinir ağları
ASIC	: Uygulamaya özel tümleşik devre
b	: Eşik değeri
BP	: Geri yayılım algoritması
CAD	: Bilgisayar destekli tasarım
CPLD	: Karmaşık programlanabilir lojik devre
CPU	: Merkezi işlemci birimi
DAC	: Dijital-analog dönüştürücü
DSP	: Dijital sinyal işleme
e	: Hata değeri
EDA	: Elektronik tasarım otomasyonu
EEPLD	: Elektriksel silinebilir programlanabilir lojik aygıt
EPLD	: Silinebilir programlanabilir lojik aygıt
EPROM	: Elektriksel olarak programlanabilir yalnız okunabilir bellek
FPAA	: Alanda Programlanabilir Analog Diziler
FPGA	: Alanda programlanabilir kapı dizileri
GAL	: Genel dizi lojik
HbA1C	: Glikohemoglobin
IEEE	: Elektrik-Elektronik Mühendisleri Enstitüsü
i	: YSA girişleri
LUT	: Look-up-Table . FPGA içinde yer alan danışma tablosu
MAX	: Çoklu dizi matrisi
MLNN	: Çok katmanlı yapay sinir ağı
MLP	: Çok katmanlı perseptron

MPGA : Maske programlanabilir lojik devre
MUX : Veri seçici
ntest : Örnek sayısı
OpAmp : İşlemsel kuvvetlendirici
OR : VEYA kapısı
PAL : Programlanabilir dizi lojik
PEEL : Programlanabilir elektriksel silinebilir lojik
PLA : Programlanabilir lojik dizi
PLD : Programlanabilir lojik aygıt
PROM : Programlanabilir yalnız okunabilir bellek
Qp : Sistem çıkışı
Qt : Gerçek çıkış
RAM : Rastgele erişimli bellek
RBF : Radyal tabanlı fonksiyon
ROM : Salt okunur bellek
SLP : Tek katmanlı perseptron
SNN : İğnecikli sinir ağı
SPLD : Basit programlanabilir lojik birim
SRM : Anahtarlamalı relüktans motor
TSA : Taylor serisi açılımı
VHDL : Yüksek hızlı tümleşik devreler için donanım tanımlama dili
VLSI : Çok büyük ölçekli tümleşik devre
w : YSA ağırlıkları
XOR : ÖZEL VEYA kapısı
y : Sistem çıkışı
yd : Gerçek çıkış
YSA : Yapay sinir ağı
δj : j. nörona ait hata faktörü
$\Delta wij(t)$: t. İterasyondaki ağırlık farkı
μ : Öğrenme katsayısı

ŞEKİLLER LİSTESİ

TABLOLAR LİSTESİ

ÖZET

Anahtar Kelimeler: Yapay Sinir Ağları (YSA'lar), Alanda Programlanabilir Kapı Dizileri (FPGA), Paralel İşlem, Çok Katmanlı Sinir Ağı (MLNN), Kayan-Noktalı Format, Şematik Tasarım, Geri Yayılım Algoritması.

Yapay Sinir Ağları (YSA'lar), biyolojik sinir sistemine dayalı elektronik modellerdir. YSA'lar girişlerden gelen verileri işleyen birbirine bağlı yapay nöronlardan oluşmaktadır. Bu mimariler, yazılım ya da donanım olarak gerçekleştirilebilirler. YSA'nın yazılım uygulamasının avantajı, tasarımcının YSA bileşenlerinin iç işleyişini bilmesine gerek olmamasıdır. Bununla birlikte, gerçek zamanlı uygulamalarda, yazılım tabanlı YSA'lar donanım tabanlı YSA'lardan daha yavaştır. YSA hesaplamaları paralel olarak gerçekleştirilmektedir ve paralel işlem için özel donanım aygıtları gereklidir. Birçok alandan araştırmacılar optimizasyon, sınıflandırma, kontrol, görüntü işleme vb. problemlerin çözümü için YSA donanım uygulamaları gerçekleştirmişlerdir. Bu uygulamalar, YSA'ların paralel doğasından yararlanmak için farklı türde cihazlar üzerinde gerçekleştirilmiştir. YSA'nın FPGA uygulamaları, yeniden yapılandırılabilir yapısı ve paralel mimarisi nedeniyle son yirmi yılda büyük ilgi uyandırmıştır. Bu tez çalışmasında, Quartus II şematik tasarım kullanılarak eğitilebilir çok katmanlı sinir ağı (MLNN) yapısının donanım uygulaması FPGA üzerinde tamamen kombinasyonel mantık olarak gerçekleştirilmiştir. Yapay sinir ağını eğitmek için eğim düşüm metodunu kullanan geri yayılım algoritması uygulanmıştır. Nümerik tanımlama için IEEE tek-hassasiyetli kayan-noktalı format kullanılmıştır. Bu çalışma aynı zamanda IEEE tek-hassasiyetli kayan-noktalı format ile tam uyumlu hızlı bir kayan noktalı toplayıcı, bir paralel çarpıcı ve bir sigmoid aktivasyon fonksiyonu bloğunu sunmaktadır. İşlemleri paralel olarak gerçekleştiren toplayıcı, paralel çarpıcı ve aktivasyon fonksiyonu bloğu tamamen kombinasyonel mantık olarak tasarlanmıştır. Bu yeni tasarımda, gecikmeyi azaltmak için kaydırma işlemlerinde kaydırmalı yazmaçlar yerine üç-durumlu tampon serileri kullanılmıştır. Üç-durumlu tampon serileri kullanıldığından kaydırma işlemi için saat darbesi gerekli değildir ve böylece sonuç tek bir çevrimde üretilir. Sadece kapı gecikmesi maliyetli önerilen tasarım, YSA'nın donanım uygulamaları için uygundur.

IMPLEMENTATION OF ARTIFICIAL NEURAL NETWORKS ON ADAPTIVE HARDWARES

SUMMARY

Key Words: Artificial Neural Networks (ANNs), Field Programmable Gate Array (FPGA), Parallel Processing, Multi Layer Neural Network (MLNN), Floating-Point Format, Schematic Design, Back Propagation Algorithm.

Artificial Neural Networks (ANNs) are electronic models based on biological nervous system. ANNs are made up of interconnected artificial neurons which can process values from inputs. These architectures can be implemented either in software or in hardware. The advantage of the software implementation of ANN is that the designer does not need to know the inner workings of ANN components. However, in real time applications, software based ANNs are slower than hardware based ANNs. ANN computations are carried out in parallel and special hardware devices are required for parallel processing. Researchers from many disciplines have been performing ANN hardware implementations to solve a variety of problems in optimization, classification, control, image processing etc. These applications have been performed on different types of devices to take advantage of the parallel nature inherent to ANNs. FPGA implementations of ANN have aroused great interest during the last two decades due to its reconfigurable structure and parallel architecture. In this thesis, hardware implementation of trainable Multi Layer Neural Network (MLNN) structure on FPGA (Field Programmable Gate Array) is realized as entirely combinational logic by using Quartus II schematic design. The back propagation algortihm, which uses gradient descent metod is implemented in order to train the neural network. IEEE single-precision floating-point format is used for numerical description. This study also presents the hardware designs of a fast floating point adder, a parallel multiplier and a sigmoid activation function block that are fully compliant with the IEEE single-precision floating-point format. The adder, parallel multiplier and the activation function block are designed as entirely combinational logic that perform operations in parallel. In this novel design, tri state buffer series are used for shifting operations instead of shift registers for reducing latency. Because the use of tri-state buffer series, clock pulse is not required for shifting and thus the result is generated in only a single clock-cycle. The proposed design is suitable for hardware implementation of ANN at the cost of gate delays only.

BÖLÜM 1. GİRİŞ

Yapay sinir ağları (YSA'lar) (ANN, Artificial Neural Network) insan beyninin fonksiyonlarından olan öğrenme ve bu yolla yeni bilgiler türetebilme özelliklerini gerçekleştirmek amacıyla geliştirilmiş modellerdir. Birden fazla yapay nöronun çeşitli şekillerde bağlandığı ve uyum içerisinde çalıştığı katmanlı bir yapıya sahiptirler [1].

YSA'nın bilgi işleme gücü, öğrenme ve genelleme yeteneklerinin yanı sıra paralel doğasından da kaynaklanmaktadır. Bilgi işlem yöntemlerinin çoğu sıralı işlem yaparken, YSA'lar eş zamanlı çalışan birçok hücre sayesinde karmaşık işlemleri çok daha hızlı yerine getirirler.

YSA'lar yazılımsal ve donanımsal olmak üzere iki şekilde gerçeklenebilirler. Sıralı işlem yapan geleneksel işlemcilerin kullanıldığı yazılımsal gerçeklemenin dezavantajları maliyet ve eğitim süresinin uzunluğudur. Eğitim süresi donanım gerçeklemesi ile kısaltılabilir. YSA donanımsal olarak uygulandığında, paralel yapının verdiği avantajı verimli bir şekilde kullanır ve yazılımsal simülasyonlara göre daha hızlıdır [2].

YSA donanımları son on yılda hızlı bir gelişim göstermiş ve YSA'nın paralel yapısını en verimli şekilde kullanabilmek amacıyla çeşitli donanımlar tasarlanmıştır. Çok geniş ölçekli tümleşik (VLSI, Very Large Scale Integration) sistemler, YSA'nın paralel yapısına uygun olmalarına rağmen farklı YSA'ların aynı devreyi kullanamayışından ve üretim aşamalarının hem maliyetli olması hem de çok zaman almasından dolayı kullanışlı değillerdir [3].

Bu aşamada alanda programlanabilir kapı dizilerinin (FPGA, Field Programmable Gate Array) kullanımı başarılı sonuçlar vermiştir. Programlanabilir mantık blokları

ile bu bloklar arasındaki ara bağlantılardan oluşan ve üretimden sonra iç konfigürasyonu tasarımcı tarafından değiştirilebilen tümleşik devrelerdir. FPGA'lar defalarca programlanabilme ve tasarım esnekliği sayesinde değişiklikleri kısa sürede başarabilme özellikleri sayesinde, zamandan ve maliyetten tasarruf sağlamışlardır. FPGA'lar paralel mimarileri sayesinde YSA donanımları için oldukça uygun platformlardır. Bu yapılar ilk yıllarda sınırlı donanım kaynakları sebebi ile verimsiz olmalarına karşın, günümüz teknolojisi sayesinde daha etkili bir şekilde kullanılmaktadır.

1.1. Literatür Taraması

Biyolojik nöronun matematiksel bir modeli olan "yapay nöron" ilk olarak 1943 yılında McCulloch ve Pitts tarafından gerçekleştirilmiştir. Bu modele göre nöron, ikili sayı sistemine sahip bir yapıda olup eşik mantığına göre çalışmaktadır [4].

Donald O. Hebb, 1949 yılında yayımladığı "Davranışın Örgütlenmesi" isimli çalışmasında "Hebb Sinapsı" olarak bilinen sinaptik modifikasyon için psikolojik öğrenme kuralının ilk açık ifadesini ortaya koymuştur. Bu çalışmaya göre nöronlar arsında bağlantılar bulunmaktadır ve bu bağlantılar öğrenme sürecinde sürekli değişmektedir [5].

Yapay nöronun ortaya atılması ile zaman içerisinde değişik paradigmalar ortaya çıkmıştır. Bunlar, "Esnek Hesaplama" (Soft Computing) ana başlığı altında toplanan bulanık sistemler, yapay sinir ağları, genetik algoritmalar ve bunların birleşmesiyle oluşan hibrit sistemlerdir. YSA, biyolojik nöronlar ve ara bağlantılarının matematiksel modelidir. Bu ağlar öğrenme, hafızaya alma ve veriler arasındaki ilişkiyi ortaya çıkarma kapasitesine sahip adaptif sistemlerdir [6].

Yapay sinir ağlarının öğrenme ve uyarlanabilirlik özellikleri robotik [7-9], görüntü işleme [10-12] ve ses tanıma [13, 14] gibi alanlarda kullanılmaktadır. Hibrit sistemler içerinde yer alan nöro-bulanık sistemler, yapay sinir ağlarının öğrenme yeteneği ile bir uzman tarafından oluşturulan dilsel kuralları birleştirir. Nöro-bulanık sistemler örüntü tanıma [15, 16], robotik [17, 18], doğrusal olmayan sistem tanımlama [19, 20]

ve adaptif işaret işleme [21, 22] alanlarında kullanılmaktadır. YSA'nın özelliklerinden birisi de doğrusal olmayan fonksiyon yakınsama yeteneğidir. Bu konuda yapılan çalışmalar arasında [23-27] numaralı referanslar dikkat çekicidir.

YSA uygulamaları, gerçekleştirildiği zaman diliminin teknolojik şartlarına bağlı olarak çeşitli donanımsal yapılarda gerçeklenmiştir. Bu amaç doğrultusunda genel amaçlı işlemciler, özel amaçlı işlemciler, VLSI sistemler ve yeniden programlanabilir donanımlar kullanılmıştır. Bu donanımlara ek olarak, YSA'nın paralel yapısı ve yüksek hızlı hesaplamalar için çoklu-işlemcili platformlar da kullanılmaktadır. Güç tüketimi, işlem hızı, boyut, taşınabilirlik ve maliyet gibi unsurlar kullanılacak donanımı belirlemek açısından önem arz etmektedir [6].

YSA donanımları son on yılda hızlı bir gelişim göstermiş ve YSA'nın paralel yapısını en verimli şekilde kullanabilmek amacıyla çeşitli donanımlar tasarlanmıştır [28]. YSA donanımlarındaki hızlı gelişim ve çeşitlilik nedeniyle bazı uygulamalar [29-34] bu alan için sınırlı kalmıştır.

FPGA'lar, çok sayıda programlanabilir eleman ve paralel yapısı sayesinde, öğrenme algoritmaları içeren ve çok sayıda nöron ve katmandan oluşan YSA'lar için uygun platformlardır[6].

Sinirsel hesaplamanın gerçekleştirildiği ilk dijital mimarilerden biri NetSim'dir [35]. Bu mimaride "çözüm birimi" ve "haberleşme birimi" olmak üzere iki ana birim kullanılmıştır. "Çözüm birimi" sinirsel hesaplamaları gerçekleştirirken, "haberleşme birimi" sinirsel işlemleri yönlendirmektedir. Bunlara ek olarak ağırlıkların saklanması için hafıza birimi kullanılmıştır. Sözü edilen bu birimler NetSim mimarisini oluşturmaktadır.

1989 yılında Holler ve arkadaşları [36], 10240 kayan kapı sinirsel hücre kullanan analog işlemci ETANN'ı (Electrically Trainable Artificial Neural Network) geliştirmiştir. Bu çalışmada ileri besleme ve geri besleme bağlantıları, paralel yapının sağladığı hız, analog giriş çıkış gerilimleri ve hızlı statik modda ya da çoğullanmış harici yolların kullanıldığı saat darbeli modda çalıştırma seçenekleri vurgulanmıştır.

Ağırlıklar “Elektrikle Silinip Programlanabilir Salt Okunur Bellek” (EEPROM, Electrically Erasable Programmable Read Only Memory) üzerinde saklanmıştır.

1992 yılında Satyanarayana ve arkadaşları, VLSI üzerinde katman sayısı yeniden yapılandırılabilir genel amaçlı YSA uygulaması gerçekleştirmişlerdir [37]. Devrede 74 ms’de yenilenen dinamik ayarlanabilir kondansatörler kullanılmıştır.

Morie ve Amemiya, çip üzerinde karşılaştırmalı geriyayılım öğrenme (Contrastive backpropagation learning) algoritması kullanan, gerçek zamanlı çalışan analog bir cihaz tasarlamışlardır. Cihaz 9 nöron ve 81 sinapstan meydana gelmektedir [38].

Sun ve arkadaşları, kesintisiz güç kaynaklarının inverter uygulamaları için analog YSA kontrolörü tasarlamışlardır. Tasarım esas olarak direçlerden ve işlemsel kuvvetlendiricilerden (OpAmp, Operational Amplifier) oluşmaktadır. Ağırlıklar “Silinip Programlanabilir Salt Okunur Bellek” (EPROM, Erasable Programmable Read Only Memory) üzerinde saklanmıştır. Tasarlanan hafıza birimi 6 bitlik olup 5 bit ağırlığa, 1 bit ise işaret bitine ayrılmıştır [39].

Yamasaki ve Shibata, analog görüntü tanıma sınıflandırıcısı gerçekleştirmişlerdir. Yaptıkları çalışmada el ile çizilmiş ve iç içe geçmiş geometrik şekilleri tanımayı mümkün kılmışlardır [40].

Khodabandehloo ve arkadaşları YSA'nın analog uygulaması amacıyla rezistif tip nöron tasarlamışlardır. Sigmoid aktivasyon fonksiyonu taylor serisi açılımı kullanılarak 1.3% hata ile elde edilmiştir [41].

Sekerli ve Butera, "Alanda Programlanabilir Analog Diziler" (FPAA, Field Programmable Analog Arrays) kullanarak Morris-Lecar nöron modeli tasarlamışlardır. Diferansiyel denklemler FPAA tasarım yazılımında mevcut toplayıcılar, çarpıcılar ve invertörler yardımıyla çözülmüştür. Nöron modeli üç adet AN221E04 FPAA yongası kullanılarak gerçekleştirilmiştir [42].

Grzechca ve arkadaşları, yaptıkları çalışmada FPAA'nın yapılandırılabilir analog bloklarındaki (CABs, Configurable Analog Blocks) parametrik hataları zaman domeni cevabına dayalı olarak teşhis etmişlerdir. Sınıflandırma işlemi için radyal temelli fonksiyon (RBF, Radial Basis Function) kullanan YSA kullanılmıştır. YSA 5 nöronlu giriş katmanı, 27 nöronlu gizli katman ve 1 nöronlu çıkış katmanından oluşmaktadır [43].

Kamala-Kannan ve arkadaşları, faz akımı ve faz gerilimini ile rotor pozisyonunu kestirerek anahtarlamalı relüktans motorun (SRM-Switched Reluctance Motor) kontrolünü gerçekleştirmişlerdir. Kullanılan YSA 3 katmanlı olup FPAA üzerinde tasarlanmıştır [44].

Dijital YSA uygulamaları, işlem yürütme hızındaki artış ve karmaşık sistem tasarımında sağladığı kolaylıklar gibi sebeplerle son 20 yılda tercih edilir hale gelmiştir. Bu uygulamalar özel VLSI mimarileri, uygulamaya özel tümdevreler (ASIC, Application Specific Integrated Circuit) ve FPGA yongaları üzerinde gerçekleştirilmiştir.

Doğrusal olmayan aktivasyon fonksiyonu yapay sinir ağlarının temel yapı taşlarından biridir. Hiperbolik tanjant ve sigmoid fonksiyonları en çok kullanılan aktivasyon fonksiyonlarıdır. Bu aktivasyon fonksiyonlarının dijital YSA’larda doğru şekilde uygulanabilmesi belli zorlukları beraberinde getirir. Zamanlooy ve Mirhassani’nin yaptıkları çalışmada hiperbolik tanjant fonksiyonu için bit düzeyinde haritalama ile doğrusal yaklaşım esasına dayanan etkin bir yöntem önerilmiştir. Bu yöntem 4x3x2 YSA donanımında kullanılmıştır. YSA donanımı Verilog yazılımı ile VLSI mimarisi üzerinde gerçekleştirilmiştir [45].

Defalarca ve sahada (üretim yerinin dışında) programlanabilmesi, tasarım sırasında büyük esneklik sağlaması ve paralel işlem yapabilme kabiliyeti gibi özellikleri FPGA kullanımını oldukça yaygınlaştırmıştır.

Hikawa, FPGA üzerinde basitleştirilmiş çok katmanlı yapay sinir ağı (MLNN, Multilayer Neural Network) mimarisini gerçekleştirmiştir. MLNN yapısı

çarpma işlemi olmadan tasarlanmıştır. Çarpıcıların olmaması devre için gereken silikon alanı küçültmüştür. Çip üzerinde öğrenme işlemini gerçekleştirebilmek amacıyla geri yayılım algoritması çarpma işlemine gerek duymayacak şekilde düzenlenmiştir. Çarpım işlemi kaydırmalı kaydediciler (shift registers) ve lojik "VE" (AND) kapıları ile gerçekleştirilmiştir [46].

Abramson ve arkadaşları, FPGA tabanlı Hopfield YSA kullanarak "N-Vezir problemini" (N-Queen Problem) ele almışlardır. N-Vezir problemi NxN boyutunda bir satranç tahtasına N adet vezirin birbirini alamayacak biçimde yerleştirilmesi problemidir. Devrede kullanılan ağırlıklar küçüktür ve küçük tamsayı değerleri ile temsil edilmiştir. Bu da kaydetme işleminde gereken donanımın küçük tamsayı değerleri için optimize edilmesini sağlamıştır. Nöronlar arasındaki bağlantı sabittir ve problemdeki kısıtlamalara göre belirlenmektedir. Optimize edilen YSA, Xilinx firmasının XC4020 yongası üzerinde test edilmiştir [47].

Omondi ve Rajapakse, Xilinx firmasının XCV812C FPGA yongası üzerinde bağımsız bileşen analizi (ICA, Independent Component Analysis) ile bağımsız bileşen sinir ağını (Independent Component Neural Network) tasarlamışlardır. Çalışma üç farklı doğrusal olmayan fonksiyonu incelerken uygulamalar sadece ikisi üzerine odaklanmıştır. LUT (Look up table) ve kombinasyonel lojik (CL, Combinational Logic) kullanan iki donanım tasarlanmıştır. LUT donanımında doğrusal olmayan fonksiyon değerleri rastgele erişimli bellek (RAM, Random Access Memory) üzerinde depolanmıştır. CL donanımında ise ağırlıklar RAM üzerinde depolanmıştır ve fonksiyon değerleri fonksiyonel bloklar tarafından üretilmiştir [48].

Kim ve arkadaşları, FPGA üzerinde ICA algoritması tasarlamışlardır. Yapılan çalışmada yankılı ve gürültülü ortamlarda konuşma tanıma işlemini gerçekleştirmişlerdir. Altera firmasının EP20K600EBC652-1 FPGA kartı üzerinde sinyal ayırma ve adaptif gürültü önleme algoritmalarını uygulamışlardır [49].

Ide ve Saito [50] REOMP (Reconfigurable Orthogonal Memory Multiprocessor) adı verilen FPGA tabanlı yeniden yapılandırılabilir paralel bilgisayar mimarisi üzerinde

YSA uygulaması gerçekleştirmişlerdir. Bu mimaride Fukushima [51] tarafından sunulan Neocognitrons uygulanmıştır. Neocognitrons YSA mimarisi Hubel ve Wiesel [52] modelini temel alan ileri beslemeli topolojidir. Paralel mimari, kontrol ünitesi olarak görev yapan bir ana işlemciden ve hafıza birimlerine bağlı yeniden yapılandırılabilir işlemcilerden meydana gelmektedir. Uygulama Altera Quartus II yazılımı ile gerçekleştirilmiştir.

Bastos ve arkadaşları, güç elektroniği uygulamaları için FPGA üzerinde YSA tabanlı kontrolör tasarlamışlardır. SACT (Synergetic Approach to Control Theory) kontrolörü tabanlı Doğru akım-Doğru akım gerilim azaltan dönüştürücü (DC-DC buck converter) YSA ile kontrol edilmiştir. YSA 4x4x1 olarak tasarlanmış ve SACT kontrolörünün yüksek performans vermesi için eğitilmiştir [53].

Ferreira ve arkadaşları, FPGA üzerinde kayan noktalı aritmetik kullanan ileri beslemeli sinir ağı tasarlamışlardır. Aktivasyon fonksiyonu parçalı-doğrusal fonksiyon (piecewise-linear function) yaklaşımı ile elde edilmiştir. YSA'nın eğitimi bilgisayar üzerinde MATLAB programı ile gerçekleştirilmiş ve elde edilen ağırlık değerleri FPGA kartına gönderilmiştir [54].

Hu ve arkadaşları, yaptıkları çalışma ile YSA'nın FPGA uygulamalarının kilit noktalarına değinmişlerdir. YSA'nın tasarımı ile alakalı veri gösterimi, iç hesaplamalar, aktivasyon fonksiyonunun uygulanması, ağırlıkların depolanması ve güncellenmesi, öğrenme algoritmasının doğası ve tasarım kısıtlamaları gibi konulara değinmişlerdir [55].

Shoushan ve arkadaşları, FPGA üzerinde geri yayılım YSA uygulaması ile karbon fiber takviyeli plastiğin hatalarını sınıflandırmışlardır. Hesaplanan veriler RAM üzerinde depolanmış ve sigmoid aktivasyon fonksiyonu LUT yöntemi ile hesaplanmıştır. Tasarım Altera firmasının iki FPGA kartı (Cyclone ve Cyclone II) üzerinde gerçekleştirilmiş ve kullanılan kaynaklar arasında karşılaştırma yapılmıştır [56].

Mekki ve arkadaşları, FPGA üzerinde gerçek zamanlı fotovoltaik (PV, Photovoltaic) modülün simülasyonunu ve uygulamasını yapmak amacıyla MLNN mimarisini sunmuşlardır. PV-modülünün performansı sadece bu hava sıcaklığı ve toplam güneş radyasyonu gibi meteorolojik verilere dayanmaktadır. Model, MATLAB/Simulink ile geliştirilip simülasyonu yapıldıktan sonra VHDL programı ile FPGA üzerinde uygulanmıştır. Tasarım, gerçek iklim koşullarında PV elektrik enerjisi üretiminin tahmini için kullanılabilir [57].

Cárdenas ve arkadaşları, birkaç enerji kaynağının birbirine bağlandığı tek fazlı güç kontrol sistemi için FPGA tabanlı mimariyi ve deney sonuçlarını ortaya koymuşlardır. Elektrik şebekesinin güç ve senkronizasyon kontrolü için adaptif lineer sinir ağı (ADALINE, Adaptive Linear Neural Network) kullanılmıştır [58]. Öğrenme işlemi Widrow-Hoff algoritması [59] ile gerçekleştirilmiştir. MATLAB üzerinde gerçekleştirilmiş hızlı Fourier dönüşümü (FFT, Fast Fourier Transform) ile ADALINE ağı kıyaslanmış ve sonuçların benzer olduğu gözlenmiştir.

Soleimani ve arkadaşları, FPGA üzerinde iğneli sinir ağını gerçekleştirmek için az sayıda çarpıcıdan oluşan parçalı-doğrusal modeli önermişlerdir [60]. YSA danışmanlı (supervised) ve danışmansız (unsupervised) öğrenme algoritmaları ile eğitilmiştir. Çalışmada, Izhikevich modelinin [61] geliştirilmiş, bir dizi parçalı-doğrusal çarpıcısız modeli sunulmuştur. Önerilen model, büyük ölçekli simülasyon projeleri için hem analog hem de dijital platformlarda uygulanabilirdir. Parçalı-doğrusal model, karakter tanımada 91.7% doğruluk sağlamıştır ve Izhikevich modeline göre önemli ölçüde hızlıdır.

Saadi ve Bettayeb, işlem sırasında bozulmuş radyolojik görüntülerin iyileştirilmesine çalışmışlardır. Otoregresif hareketli ortalamalar modeli (ARMA, autoregressive moving averages) YSA kullanılarak tanımlanmıştır. Ağ eğitimi, yeni bir sürü optimizasyon algoritması olan yapay arı kolonisi (ABC, Artificial Bee Colony) optimizasyonu kullanılarak geliştirilmiştir. ARMA-YSA modeli Modelsim programı ile simüle edilmiş ve en iyi yapı FPGA üzerinde uygulamaya konulmuştur [62].

YSA'nın literatürde yer alan Sayısal İşaret İşlemci (DSP, Digital Signal Processor) uygulamaları da oldukça dikkat çekicidir.

Card ve arkadaşları, yarışmacı öğrenmenin (competitive learning) ve kendi kendini düzenleyen haritaların (SOFMs, self organizing feature maps) sayısal mimarilerine uygulanan, yeniden yapılandırılabilir pararlel bir nöro-bilgisayarı sunmuşlardır. Mimari, birincil işlem elemanı olarak görev yapan bir DSP'den ve mesaj dekoder, ön işlemci ve post işlemci olarak görev yapan 3 FPGA kartından oluşmaktadır [63].

Boquete ve arkadaşları, bir tekerlekli sandalyenin ilerletilmesi için DSP üzerinde YSA kontrol sistemi tasarlamışlardır. Sistem kontrolörü olarak yineleyen radyal temelli sinir ağı kullanılmıştır. Bir Kalman filtresi tekerlekli sandalyeyi tespit etmekte ve kontrol hatalarını nöro-kontrolöre iletmektedir. Kullanılan tekerlekli sandalye iki giriş iki çıkışlı bir sistemdir. Sağ el ve sol el tekerleklerinin açısal hızı girişleri oluştururken doğrusal hız ve açısal hız çıkışları oluşturmaktadır. Kontrol işlemi, düşük seviye kontrol (tekerlekli sandalye motor sürücüleri) ve yüksek seviye kontrol (nöro-kontrolör) olmak üzere ikiye ayrılmıştır. Düşük seviye kontrol FPGA üzerinde gerçekleştirilirken yüksek seviye kontrol DSP üzerinde gerçekleştirilmiştir [64].

Venayagamoorthy ve arkadaşları, yaptıkları çalışmada çift sezgisel programlama (DHP, dual heuristic programming) tabanlı türbin nöro-kontrolörünün donanımsal tasarımını göstermişlerdir. Türbinli jeneratörleri kontrol etmek amacıyla DHP uyartımı ve türbin nöro-kontrolörleri DSP üzerinde gerçekleştirilmiştir. Öne sürülen DHP nöro-kontrolör, sistem çalışma koşullarındaki değişimlere rağmen hem voltaj regülasyonunda hem de güç sisteminin kararlılığında iyileştirme sağlamıştır [65].

Lee ve Sheu, 64 nöron ve 4096 programlanabilir sinapstan meydana gelen genel amaçlı YSA mimarisi gerçekleştirmişlerdir. Sinaps ağırlıkları tampon bellekte depolanmıştır ve ağırlıklar D/A dönüştürücü yardımıyla periyodik olarak yenilenir. Sinirsel işlemler ve ağırlıkların güncellenmesi eşzamanlı olarak gerçekleşmektedir [66].

Boser ve arkadaşları, 2000'in üzerinde çarpma ve toplama işlemini aynı anda yapan ANNA (Analog Neural Network Arithmetic unit) isimli analog-dijital karma mimariyi sunmuşlardır. Hesaplamalar ağırlıklarda 6 bit hassasiyetle, nöronlarda ise 3 bit hassasiyetle gerçekleştirilmiştir. Nöron başına giriş sayısı 16 ile 256 arasında değişmektedir. Ağırlıkların sayısı 4096 olup 110 μs'de bir yenilenen kapasitörler üzerinde saklanmaktadırlar. 133000 üzerinde bağlantıya sahip olan YSA ile optik karakter tanıma uygulaması yapılmıştır [67].

Shima ve arkadaşları, 0.8 mikron CMOS teknolojisi ile gerçekleştirilmiş, iki adet yüksek hızlı VLSI aygıtından oluşan mimariyi sunmuşlardır. Birinci cihaz sinaps matrisini ve ikinci cihaz nöral matrisi içermektedir. Birinci Matris ağırlıkların lokal kontrol mekanizmasını barındırmaktadır. Nöral matris ise geri yayılım ve/veya Hebbian öğrenme algoritmaları ile nöronları içermektedir. YSA, 8 bit hassasiyette 576 sinaps ve 24 nörondan oluşmaktadır [68].

Lu ve arkadaşları, sigmoid fonksiyonu ve türevlerini üreten, parametreleri programlanabilir yeni bir nöron devresi sunmuşlardır. Nöron analog yapıdadır ve analog ağırlıklar analog-dijital dönüştürücü (ADC, analog digital converter) vasıtasıyla dijital olarak RAM üzerinde kaydedilmekte ve dijital-analog dönüştürücü (DAC, digital analog converter) ile tekrar analog hale dönüştürülmektedir. Ağırlıklar dijital olarak güncellenmektedir. Nöronun davranışını belirlemek amacıyla iki deney yapılmıştır. Birinci deneyde doğrusal olmayan bölümleme problemi için 1 ms'de yakınsama elde edilmiştir. İkinci deneyde ise sin(x) fonksiyonuna iyi bir yakınsama sağlanmıştır [69].

Erkmen ve arkadaşları, 0.5 mikron CMOS teknolojisi kullanarak genel amaçlı konik kesit fonksiyonlu sinir ağı (CSFNN, Conic Section Function Neural Network) geliştirmişlerdir. Kontrol ünitesi ve depolama birimi dijital tasarlanmış iken ileri beslemeli hesaplama birimleri analog tasarlanmıştır. Dijital kısım hafızalar (EEPROM hücreleri), kod çözücüler ve kontrol ünitesinden meydana gelmektedir. YSA mimarisi en fazla 16 giriş, 16 nöronlu bir gizli katman ve 8 çıkıştan oluşmaktadır. Çalışma, tipik nesne tanıma problemleri için yazılım uygulamalarına yakın sonuçlar vermektedir [70].

Sackinger ve arkadaşları, yüksek hızlı görüntü analizi ve YSA için ANNA tabanlı mimariyi sunmuşlardır. Tasarım, iki adet tamamıyla programlanabilen analog VLSI (ANNA) mimarisinden, 4 FPGA kartından ve hafıza birimlerinden meydana gelmektedir [71].

Karmaşık hesaplamalara sebep olan düzensiz haritalama ve düzlemsel olmayan bağlantı topolojisi VLSI donanım tasarımcıları için önemli ölçüde sorun teşkil etmektedir. Ayrıca donanım kısıtlamaları (özellikle analog bileşenlerde) tasarım aşamasında hesaplama hatalarına ve doğruluğun azalmasına sebep olabilmektedir. Öğrenme sürecinde iterasyon sayısı artırılarak bu tür hatalar azaltılmaya çalışılmıştır. Aktivasyon fonksiyonlarının doğrusal olmayan yapısı donanım tasarımında ortaya çıkan bir diğer zorluktur [72]. Bu zorlukların üstesinden gelebilmek amacıyla çeşitli mimariler geliştirilmiştir. Bunlar dijital [73-75], analog [76, 77], hibrit [78, 79], FPGA tabanlı [80-82] ve optik [83-85] mimarilerdir.

Glesner ve Poechmueller [86], Kung [73] ve Mead [76] YSA'nın VLSI donanımlar üzerindeki uygulamaları arasında ilk sıralarda yer almaktadır. Heemskerk [87] hızlandırıcı birimler, genel amaçlı işlemciler ve nöro-yongalardan meydana gelen nöro-bilgisayar mimarisini sunmuştur. Ienne ve arkadaşları [88] dijital uygulamaları standart dijital bileşenli paralel sistemler ve özel işlemcili paralel sistemler olmak üzere iki temel tasarım açısından incelemişlerdir. Aybay ve arkadaşları [89] dijital nöro-bilgisayarlar ve nöro-yongaları sınıflandırmak ve karşılaştırmak için kullanılabilecek parametreleri ortaya koymuşlardır. Sundararajan ve Saratchandran [90], birkaç YSA modelinin paralel uygulama yönlerini detaylıca ele almışlardır. Yapılan çalışmada Geri Yayılım (BP, Back Propagation) YSA uygulamaları ile ilgili yorumlar, analizler ve deneysel çalışmalara değinmişlerdir. Burr [91, 92], YSA mimarileri için yonga alanı, performans ve güç tüketimi kestirim teknikleri sunmuştur.

Zhu ve Sutton yaptıkları araştırmada, FPGA tabanlı YSA uygulamalarında tasarım konularını ve uygulama tekniklerini incelemişlerdir. Bu uygulamalar rekonfigürasyonun (prototip ve simülasyon, yoğunluk artırma ve topoloji

adaptasyonu) yanı sıra veri gösterim teknikleri (tamsayı, kayan noktalı sayı, bit dizisi) ile sınıflandırılmaktadır [93].

Schrauwen ve D'Haene [80] İğnecikli Sinir Ağı'nın (SNN, Spiking Neural Network) FPGA tabanlı uygulamalarını incelemişlerdir. Maguire ve arkadaşları [94] FPGA tabanlı SNN uygulamalarının detaylı bir incelemesini yapmışlar ve önemli zorluklarını ortaya koymuşlardır. Bartolozzi ve Indiveri [95] iğneli sinaptik (spiking synaptic) modellerin çeşitli donanım uygulamaları için karşılaştırmalı analizler yapmışlardır.

Muthuramalingam ve arkadaşları, FPGA üzerinde doğrusal/doğrusal olmayan aktivasyon fonksiyonları ile çok girişli nöron uygulamasını gerçekleştirmişlerdir. İşlem hızını artırmak amacıyla doğrusal olmayan aktivasyon fonksiyonunu için LUT yöntemini kullanmışlardır [96].

Hikawa, aktivasyon fonksiyonu olarak parçalı-doğrusal fonksiyon kullanan dijital darbe-modu çok katmanlı sinir ağı tanımlamışlardır. Sigmoid fonksiyonunun parçalı-doğrusal yaklaşımı ile öğrenme ve genelleme yakınsama kapasitesi iyileştirilmiştir [97].

1.2. Çalışmanın Amacı

Literatürde analog, dijital ve karma yöntemler kullanılarak VLSI veya ASIC üzerinde oluşturulmuş YSA donanımları bulunmaktadır [2]. Güç tüketimi, işlem hızı, boyut, taşınabilirlik ve maliyet gibi unsurlar kullanılacak donanımı belirlemek açısından önem arz etmektedir. Analog YSA modelleri düşük maliyetin yanı sıra yüksek hız sunarken, sabit mimarinin dezavantajlarını da beraberinde getirir.

Literatürde önerilen dijital YSA donanımları genellikle paralel mimarinin içerisine gömülmüş ve sıralı işlem yapan toplayıcı devreler ya da çok fazla donanım kaynağı kullanan büyük ölçekli çarpıcı devreler içermektedir. Bununla birlikte, tek bir çarpıcı ve aktivasyon fonksiyonu bloğunun paylaşılarak kullanıldığı DSP uygulamaları da

dikkat çekicidir. Fakat sözü edilen yapılar paralel hesaplamanın doğasına aykırıdır [2].

Karmaşık hesaplamalara sebep olan düzensiz haritalama ve düzlemsel olmayan bağlantı topolojisi VLSI donanım tasarımcıları için önemli ölçüde sorun teşkil etmektedir. Tasarım sürelerinin uzunluğu ve üretim maliyetleri nedeniyle VLSI ve ASIC, YSA donanımları için uygun platformlar değillerdir [2].

Paralel işlem yapabilme, tekrar programlanabilme, uygulama kolaylığı, çok sayıda programlanabilir elemana sahip olma gibi özellikleri nedeniyle bu çalışmada uyarlanabilir donanım olarak FPGA seçilmiştir. FPGA'lar, çok sayıda nöron ve katmandan oluşan YSA donanım uygulamaları için uygun platformlardır.

Bu çalışmada IEEE 754 32 bit kayan noktalı nümerik tanımlama kullanılarak eğitilebilir bir MLNN donanımının FPGA üzerinde gerçeklenmesi hedeflenmiştir. Bu amaç doğrultusunda, MLNN FPGA üzerinde gerçeklenirken, hızdan ödün vermemek için paralel çarpıcı ve saat darbesine ihtiyaç duymayan bir toplayıcı tasarlanacaktır. Bu sayede, YSA'nın paralel mimarisinin sunduğu avantajların en verimli şekilde kullanılması, sıralı işlem yapan elemanlar yerine yeni tasarımlar geliştirilerek işlem zamanının azaltılması amaçlanmaktadır.

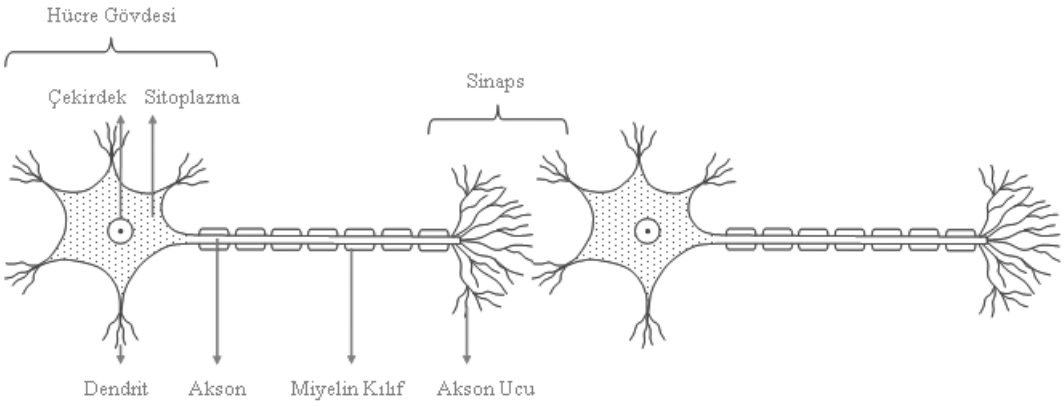
Bu tez çalışmasının ikinci bölümünde, YSA'nın yapısı ve temel işlevleri anlatılmıştır. Üçüncü bölümde programlanabilir lojik, FPGA donanımı ve donanım tanımlama dili VHDL hakkında kısaca bilgi verilmiştir. Dördüncü bölümde, Altera firması tarafından özel tasarım ihtiyaçlarına cevap verebilmek amacıyla geliştirilmiş bir CAD (Computer Aided Design) programı olan Quartus II yazılımı hakkında bilgi verilmiştir. Yeni bir projenin şematik olarak nasıl oluşturulacağı ve derleneceği örnek üzerinden anlatılmıştır. Beşinci bölümde, eğitilebilir 2x3x1 MLNN mimarisi FPGA üzerinde gerçeklenmiştir. Yapay sinir ağını eğitmek için eğim düşüm metodunu kullanan geri yayılım algoritması uygulanmıştır. Paralel yapıyı en verimli şekilde kullanabilmek için sıralı işlem yapan elemanlar yerine yeni tasarımlar geliştirilmiştir. Bu amaç doğrultusunda paralel çarpıcı ve kaydırmalı yazmaç kullanmayan dolayısıyla saat darbesine ihtiyaç duymayan bir toplayıcı tamamen

donanımsal olarak tasarlanmıştır. Kaydırma işlemleri için farklı bir donanım gerçekleştirilmiş ve kaydırmalı yazmaçlar yerine üç-durumlu tampon serileri kullanılmıştır. Tasarlanan donanım sayesinde girişlerinde kaydırılmış veriler bulunan üç-durumlu tampon serilerinden yalnız biri aktif hale getirilerek kaydırma işlemi gerçekleştirilmiştir. Sigmoid aktivasyon fonksiyonu bloğu Taylor seri açılımı kullanılarak gerçeklenmiştir. Gene bu bloğun tasarımında toplayıcı ve paralel çarpıcı devrelerden faydalanılmıştır. Altıncı bölümde, FPGA üzerinde donanımsal olarak gerçeklenen 2x3x1 MLNN modeli, biyomedikal bir problem ve doğrusal olmayan bir yapıya sahip olan XOR (Özel Veya) problemi kullanılarak test edilmiştir. Son bölümde ise uygulamanın detayları ve sonuçları üzerine yapılacak önerilere ve tartışmalara yer verilmiştir.

BÖLÜM 2. YAPAY SİNİR AĞLARI

2.1. Biyolojik Sinir Hücresi

Biyolojik bir sinir hücresi; hücre gövdesi, akson, sinir ucu (dendrit) ve akson ucundaki ince uzantılardan (sinaps) oluşmaktadır. Elektrik sinyali, hücre duvarındaki gerilim değiştirilerek üretilir ve sinapslar yardımıyla bir diğer hücreye iletilir. Bir hücre, sahip olduğu dürtü miktarınca diğer hücreleri etkiler. Bazı hücreler diğerlerinin dürtülerini pozitif yönde etkilerken, bazı hücreler de negatif yönde etkiler ya da bastırır. İnsan sinir sistemi, bu prensiple çalışan milyonlarca hücrenin bir araya gelmesinden oluşmuştur [98, 99]. Biyolojik sinir hücresi Şekil 2.1'de gösterilmiştir [99].

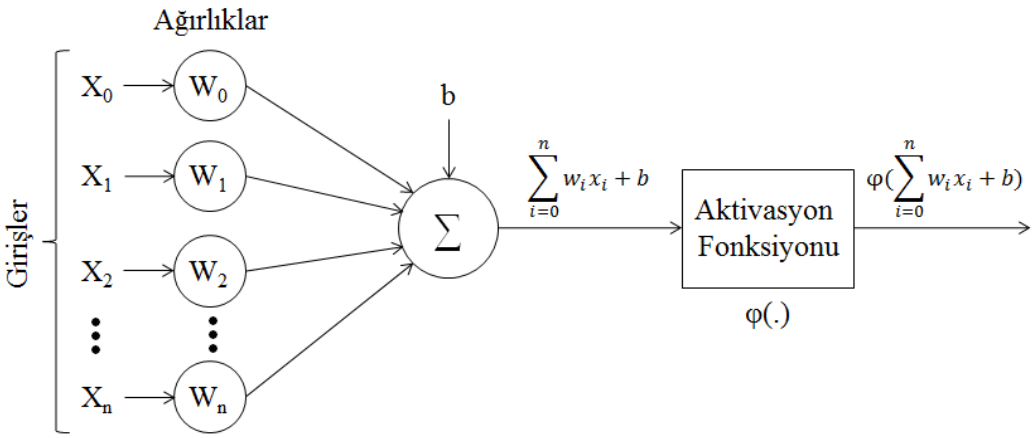


Şekil 2.1. Biyolojik sinir hücresi [99]

2.2. Yapay Sinir Hücresi

İnsan beynindeki nöronlar ve ara bağlantılarının çalışma mantığını modelleyebilmek amacıyla çeşitli çalışmalar yapılmıştır. Biyolojik nöronun matematiksel bir modeli olan "yapay nöron" ilk olarak 1943 yılında McCulloch ve Pitts tarafından gerçekleştirilmiştir. Bu modele göre nöron, ikili sayı sistemine sahip bir yapıda olup eşik mantığına (treshold logic) göre çalışmaktadır [4].

İnsan sinir hücresi temel alınarak oluşturulan yapay nörona ait genel yapı Şekil 2.2'de gösterilmiştir. Yapay nörondaki işlem basamakları üç aşamalıdır. İlk olarak, girişler (x) bağlantılara ait ağırlık değerleri (w) ile çarpılır. Girişler dış dünyadan ya da bir başka sinir hücresinden gelen bilgilerdir. Ağırlıklar ise girişlerin sinir hücresi üzerindeki etkisini belirleyen katsayılardır [100]. İkinci aşamada, ağırlıklarla çarpılan değerler hücrenin eşik değeri (b) ile toplanır. Üçüncü aşamada ise toplama fonksiyonu çıkışı aktivasyon fonksiyonundan ($\varphi(.)$) geçirilerek yapay nöron çıkışı elde edilir [101]. YSA'nın işlevine göre doğrusal ya da doğrusal olmayan aktivasyon fonksiyonları kullanılabilir.



Şekil 2.2. Yapay sinir hücresi

Yapay nöron mühendislik biliminde "işlemci eleman" olarak adlandırılmaktadır ve beş temel işlevi vardır [1, 100].

2.2.1. Girişler

Girişler, yapay nörona dış dünyadan ya da başka nöronlardan gelen bilgilerdir. YSA'nın öğrenmesi istenen örnekler tarafından belirlenmektedirler.

2.2.2. Ağırlıklar

Ağırlıklar, girişlerin yapay nöron üzerindeki önemini ve etkisini belirleyen katsayılardır. Her nöronun her bir girişi için bir ağırlık değeri tanımlanmıştır.

Ağırlıklar “sıfır” olabileceği gibi negatif, çok büyük ya da çok küçük değerler de alabilirler.

2.2.3. Toplama fonksiyonu

Bu fonksiyon, yapay nöronun her bir girişi (*x*) ile o girişlere ait ağırlıkların (*w*) çarpımlarının toplamıdır. Bu amaçla değişik fonksiyonlar kullanılmaktadır. Birçok uygulamada eşik değeri (*b*) toplama dahil edilmektedir. (Denklem 2.1)

$$Toplam = \sum_{i}^{n} x_i w_i + b \qquad \textbf{(2.1)}$$

b eşik değeri sıfırdan farklı bir sayı seçilerek, tüm giriş değerlerinin sıfır olması durumunda çıkışın sıfır olmasını engellemektedir.

2.2.4. Aktivasyon fonksiyonu

Bu fonksiyon, toplam nöron giriş değerini işleyerek nöronun bu toplam için üreteceği değeri belirler. Bir ağdaki bütün nöronların aynı aktivasyon fonksiyonunu kullanması gerekmez. Aktivasyon fonksiyonu olarak çıkış değerini hesaplamak amacıyla çeşitli fonksiyonlar kullanılabilir. En çok kullanılan aktivasyon fonksiyonları lineer, rampa, basamak ve sigmoid fonksiyonlarıdır. Şekil 2.3’te bu fonksiyonlar gösterilmiştir [102].

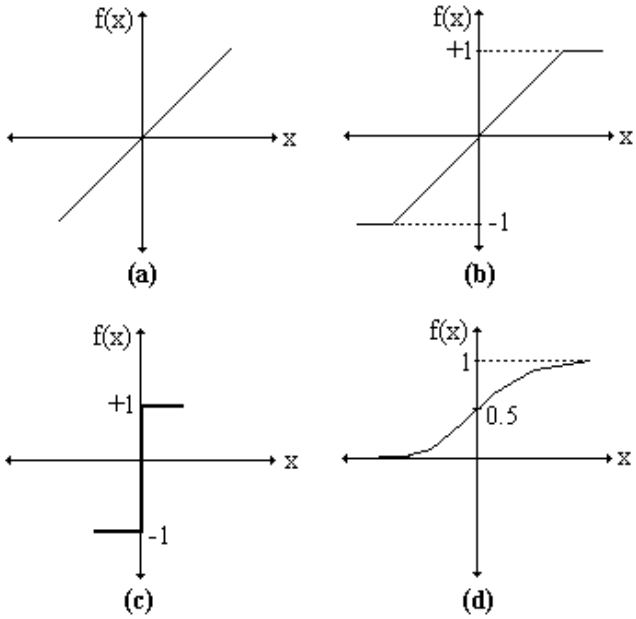
2.2.5. Nöron çıkışı

Aktivasyon fonksiyonu tarafından belirlenen çıkış değeridir. Üretilen çıkış değeri, dış dünyaya veya kendisinden sonra gelen hücre/hücrelere giriş olarak verilebilir.

2.3. Yapay Sinir Ağlarının Yapısı

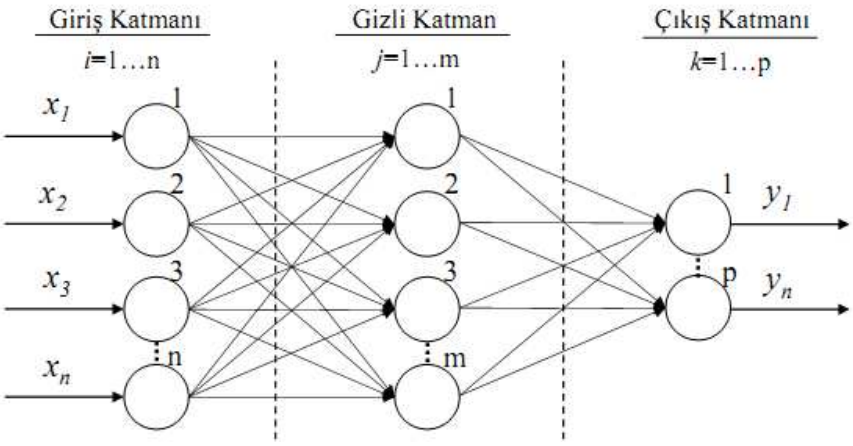
Donald O. Hebb, 1949 yılında yayımladığı “Davranışın Örgütlenmesi” isimli çalışmasında “Hebb Sinapsı” olarak bilinen sinaptik modifikasyon için psikolojik

öğrenme kuralının ilk açık ifadesini ortaya koymuştur. Bu çalışmaya göre nöronlar arasında bağlantılar bulunmaktadır ve bu bağlantılar öğrenme sürecinde sürekli değişmektedir. Sinir ağı teorisi bu temel üzerine oturtulmuş olup birçok yapay sinir ağı modeli geliştirilmiştir [5].



Şekil 2.3. YSA için kullanılan aktivasyon fonksiyonları

YSA, insan beyninin fonksiyonlarından olan öğrenme ve bu yolla yeni bilgiler türetebilme özelliklerini gerçekleştirmek amacıyla geliştirilmiş modellerdir. Birden fazla yapay nöronun çeşitli şekillerde bağlandığı katmanlı bir yapıya sahiptirler. Genellikle, kendi içinde paralel 3 katmandan oluşmaktadırlar [1]. Şekil 2.4'te 3 katmanlı bir YSA gösterilmektedir [100].



Şekil 2.4. 3 katmanlı YSA modeli

Giriş katmanı, gizli katman ve çıkış katmanından oluşan YSA'da her bir nöron çıkışı bir sonraki katmanın nöronlarına giriş olarak verilmiştir.

2.3.1. Giriş katmanı

Giriş katmanındaki nöronlar, dış dünyadan aldıkları bilgileri ağa iletirler. Bazı ağlarda giriş katmanında herhangi bir bilgi işleme gerçekleştirilmez [1].

2.3.2. Gizli katman

Giriş katmanından gelen bilgiler gizli katmanda işlenir ve çıkış katmanına gönderilir. Bir YSA'da birden fazla gizli katman olabilir [1].

2.3.3. Çıkış katmanı

Gizli katmandan gelen bilgiler işlenerek, verilen giriş değerleri için üretilmesi gereken çıkış değeri bu katmanda üretilir. Üretilen çıkış dış dünyaya gönderilir [1].

2.4. Yapay Sinir Ağlarının Özellikleri

YSA'nın hesaplama ve bilgi işleme gücü, paralel yapısından, öğrenebilme ve genelleme yeteneğinden kaynaklanmaktadır. Genelleme, eğitim ya da öğrenme sürecinde karşılaşılmayan girişler için de YSA'nın uygun tepkileri üretmesi olarak tanımlanır. Bu üstün özellikleri, YSA'nın karmaşık problemler için de çözüm üretebilme yeteneğini göstermektedir. Günümüzde birçok bilim alanında YSA, aşağıdaki özellikleri nedeniyle etkin olmuş ve uygulama yeri bulmuştur [99, 101].

2.4.1. Paralellik

Bilgi işlem yöntemlerinin çoğu sıralı işlem yaparken, YSA'lar eş zamanlı çalışan birçok hücre sayesinde karmaşık işlemleri çok daha hızlı yerine getirirler. İşlem sırasında YSA'nın bazı nöronlarının bozulması ve çalışamaz duruma düşmesi durumunda dahi sistem çalışmasına devam edebilir [100, 101].

2.4.2. Doğrusal olmama

YSA yapı olarak doğrusal bir yapıya sahip iken temel birim olan hücre doğrusal bir yapıda değildir. YSA'nın doğrusallığı transfer fonksiyonu ile belirlenir. YSA bu özelliği sayesinde karmaşık problemlerin çözümünde tercih edilmektedir [103].

2.4.3. Öğrenme

YSA'nın temel işlevi bilgisayarların öğrenmesini sağlamaktır. Böylece YSA benzer durumlar karşısında benzer cevaplar verecektir. Bu da nöronlar arasında doğru bağlantılar kurulması ve bu bağlantılara uygun ağırlık değerlerinin atanması ile mümkün olur. Genellikle ağırlıklara başlangıç değeri olarak rastgele değerler atanır. Seçilen öğrenme algoritmasına ve hata değerine bağlı olarak ağırlıklar yenilenir ve öğrenme işlemi gerçekleştirilir [1, 103].

2.4.4. Uyarlanabilirlik

YSA'lar, üzerinde çalıştığı probleme göre ağırlıklarını ayarlarlar. Bir problemi çözmek için eğitilen YSA, herhangi bir başka problemde de kullanılabilir. Bunun için yeni problemin giriş ve çıkış verilerine göre ağın tekrar eğitilmesi gerekmektedir [100, 103].

2.4.5. Genelleme

Genelleme özelliği, eğitim esnasında kullanılan nümerik bilgilerden eşleştirmeyi betimleyen kaba özelliklerin çıkarılması ve böylelikle eğitim sırasında verilmeyen giriş değerleri için de anlamlı yanıtlar üretilebilmesidir [1].

2.4.6. Hata toleransı

YSA'lar, klasik hesaplama sistemlerinin aksine hata toleransına sahip sistemlerdir. Paralel doğaları nedeniyle bilgi tek bir noktada saklanmayıp sisteme dağıtılmıştır.

YSA'nın bazı nöronlarının bozulması ve çalışamaz hale düşmesi durumunda performans düşer fakat ağ çalışmaya devam eder [1, 100].

2.4.7. Analiz ve tasarım kolaylığı

YSA'ların temel işlem elemanı olan yapay nöron modeli, hemen hemen bütün YSA yapılarında aynıdır. Bundan dolayı, standart yapay nöron modeli tasarlandıktan sonra farklı YSA modelleri için kullanılabilmektedir [100].

2.5. YSA'ların Sınıflandırılması

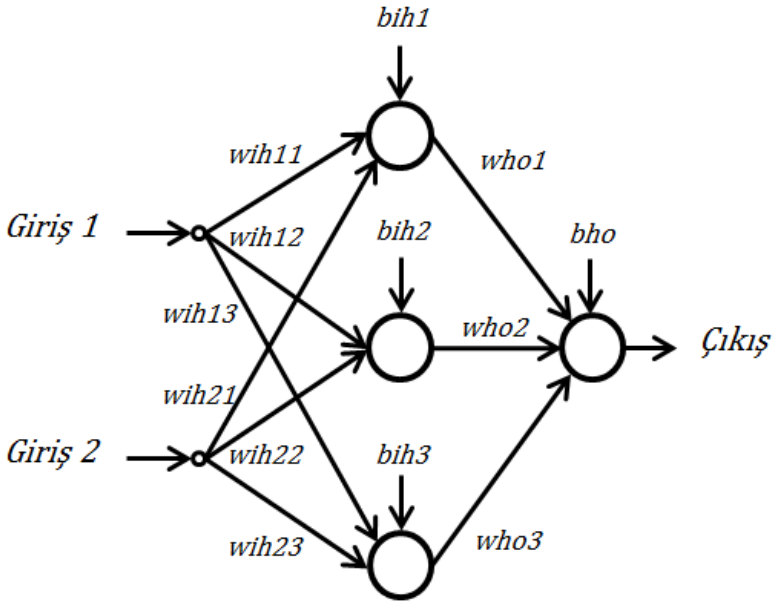
2.5.1. YSA'ların ağ yapılarına göre sınıflandırılması

Yapay sinir ağları ağın yapısına göre ileri beslemeli ve geri beslemeli olarak ikiye ayrılmaktadır [104].

2.5.1.1. İleri beslemeli yapay sinir ağları

İleri beslemeli YSA'lar en genel hali ile giriş katmanı, gizli katman ve çıkış katmanından meydana gelir. Her katman yalnızca kendinden sonra gelen katmana bağlıdır ve bir katmanın çıkışı bir sonraki katmanın girişidir. Dolayısıyla, bir katmandaki nöronların çıkışları ağırlıklar ile çarpılarak bir sonraki katmana iletilir. Ağa sunulan girişler giriş katmanı tarafından çoğullanarak gizli katmana iletilir. Gizli katman, giriş bilgilerini bağlantı ağırlıklarının etkisi ile alır. Bilgi gizli katman ve çıkış katmanında işlendikten sonra dış dünyaya verilir.

Uygulamalarda bir ya da iki gizli katman kullanmak yeterli olmaktadır [99]. Bu ağ yapılarına örnek olarak Tek Katmanlı Perseptron (SLP, Single Layer Perceptron), Çok Katmanlı Perseptron (MLP, Multi Layer Perceptron), Adaline ve LVQ (Learning Vector Quantization) yapıları gösterilebilir [2, 104-106]. İleri beslemeli YSA yapısı Şekil 2.5'te gösterilmiştir.



Şekil 2.5. İleri beslemeli YSA

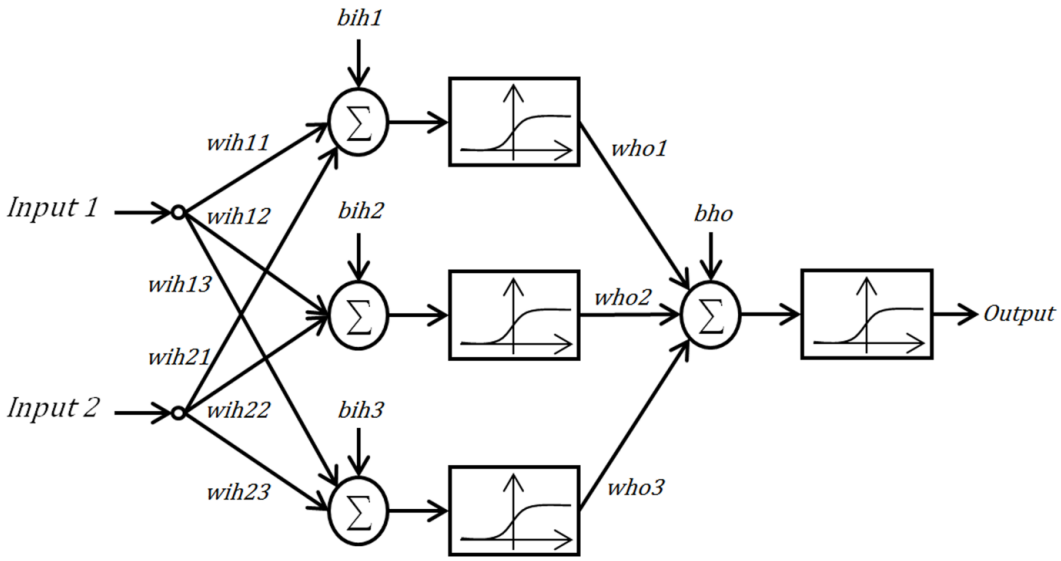
Bir YSA'nın öğrenmesi istenen olayların, giriş ve çıkışları arasındaki ilişkiler doğrusal değil ise bu olayların öğrenilmesi için gelişmiş sinir ağı modellerine ihtiyaç duyulur. Çok katmanlı yapay sinir ağı (MLNN, Multilayer Neural Network) modeli bu ağlardan birisidir [1].

MLNN eğitilmesi kısa süren, veri işlemede etkili bir ağdır. MLNN bir giriş katmanı, bir veya daha fazla gizli katman ve bir çıkış katmanından meydana gelmektedir. Giriş katmanı bir tampon gibi davranır ve veriler üzerinde herhangi bir işlem yapmadan gizli katmana iletir. Gizli katman ve çıkış katmanı, verilerin işlendiği katmanlardır [2]. Örnek bir MLNN yapısı Şekil 2.6'da gösterilmiştir.

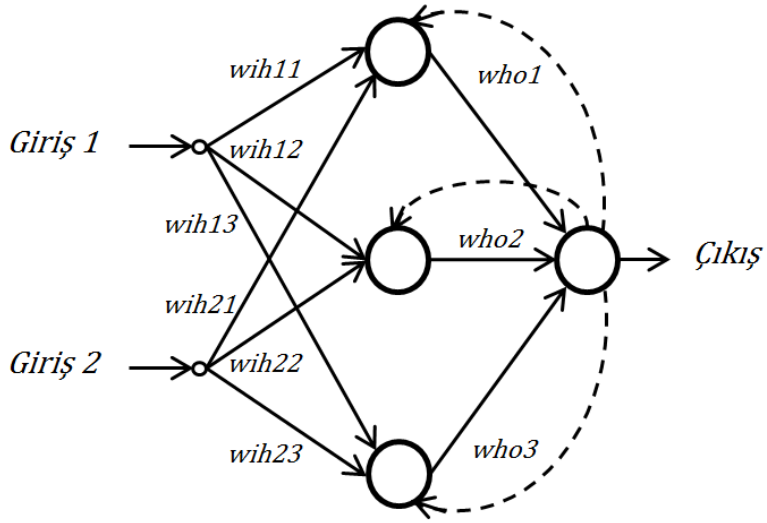
2.5.1.2. Geri beslemeli yapay sinir ağları

Geri beslemeli YSA'da, bir nöronun çıkışı sadece kendinden sonra gelen katmana giriş olarak verilmez. Kendinden önceki veya kendi katmanında bulunan en az bir nörona gecikmeli giriş olarak verilir. Bundan anlaşılacağı üzere, geri besleme katmanlar arasında olabildiği gibi aynı katmanda yer alan nöronlar arasında da olabilir. Bu yapısı ile geri beslemeli YSA doğrusal olmayan dinamik bir davranış göstermektedir. Geri besleme bağlantılarının bağlanış şekline göre farklı davranışta YSA'lar elde etmek mümkündür [100]. Hopfield, SOM (Self Organization Map),

Kohonen, ESN (Echo State Network), Elman ve Jordan ağları bu yapılara örnek verilebilir [2, 105, 107, 108]. Geri beslemeli bir YSA yapısı Şekil 2.7'de gösterilmiştir.



Şekil 2.6. Örnek MLNN yapısı



Şekil 2.7. Geri beslemeli YSA

2.5.2. YSA'ların öğrenme yöntemlerine göre sınıflandırılması

Teknik olarak bir YSA'nın en temel görevi, kendisine verilen bir giriş setine karşılık gelen bir çıkış seti üretebilmektir. Bunu gerçekleştirebilmek için YSA, olayın örnekleri ile eğitilir ve bu işleme öğrenme adı verilir [1]. YSA'da değişebilen sistem

parametreleri, girişlerin nöron üzerindeki etkisini belirleyen ağırlıklarıdır. Öğrenme, nöronlar arasındaki bağlantı ağırlıklarının değiştirilmesi ile gerçekleştirilmektedir. Sonuç olarak öğrenme, sistem parametrelerinin, verilen girişe karşılık istenen çıkışı elde edecek şekilde uyarlanmasıdır [99, 109]. Genel olarak 3 öğrenme stratejisi uygulanmaktadır.

2.5.2.1. Danışmanlı öğrenme

Danışmanlı öğrenmede ağa bir danışmanın müdahalesi vardır. Her örnek için hem giriş değerleri hem de o girişlere karşı oluşturulması gereken çıkış değerleri sisteme verilir. YSA'nın ürettiği çıkış değerleri ile gerçekte olması gereken çıkış değerleri karşılaştırılarak hata üretilir. Ağırlık değerleri başlangıçta rastgele atanır ve yeni ağırlıklar hata payına göre düzenlenir. Ağırlık değerlerindeki değişim, hatayı minimize edecek, mümkünse sıfıra indirecek şekilde ayarlanarak kabul edilebilir bir ağ performansına ulaşılır [1, 109]. Widrow&Hoff [110] ve Rumelhart&McClelland [111] tarafından geliştirilen delta kuralı veya geri yayılım algoritması danışmanlı öğrenmeye örnek olarak verilebilir [2].

2.5.2.2. Danışmansız öğrenme

Danışmansız öğrenmede, sisteme sadece giriş değerleri verilir ve ağın ürettiği çıkışların doğruluğuna dair herhangi bir geri besleme yoktur. Sistemin, girişler arasındaki ilişkileri kendi kendine öğrenmesi gerekir. Danışmansız öğrenme daha çok sınıflandırma problemleri için kullanılır ve ağ her bir örneği kendi arasında sınıflandıracak şekilde kendi kurallarını oluşturur [1]. Khonen tarafından geliştirilen SOM [107] ile Carpenter ve Grossberg tarafından geliştirilen Adaptif Rezonans Teorisi (ART) [112] yapılarında danışmansız öğrenme metodu kullanılmaktadır.

2.5.2.3. Takviyeli öğrenme

Takviyeli öğrenmede, her giriş değeri için olması gereken çıkış değerini sisteme göstermek yerine, her iterasyon sonucunda elde edilen sonucun doğru veya yanlış

olduğunu gösteren bir sinyal üretilir. Sistem bu sinyali dikkate alarak öğrenme sürecini devam ettirir [1].

2.6. YSA Öğrenme Algoritmaları

YSA yapılarının eğitiminde çok sayıda öğrenme algoritması kullanılmaktadır. Geri yayılım (BP, Back Propagation) algoritması, MLNN eğitiminde sıkça kullanılmaktadır. Bu tez çalışmasında kullanılan geri yayılım algoritması aşağıda kısaca açıklanmıştır.

2.6.1. Geri yayılım (BP) algoritması

Geri yayılım algoritması, basit yapısı sebebiyle MLNN eğitiminde en çok tercih edilen algoritmadır [2, 103]. Bu algoritma, hatayı çıkıştan girişe doğru azaltmaya çalışır. Bu nedenle geri yayılım algoritması ismini almıştır. Ağ çıkışındaki hataya göre, çıkış katmanından giriş katmanına doğru her bir katmandaki ağırlıklar yeniden hesaplanır.

Bu algoritma ile *i* ve *j* katmanındaki nöronlar arasındaki ağırlık farkının *t*'inci iterasyondaki değeri olan *Δwij(t)* hesaplanır. (Denklem 2.2)

$$\Delta w_{ij}(t) = \eta \delta_j x_i \qquad \textbf{(2.2)}$$

Burada η öğrenme katsayısıdır. Öğrenme katsayısı, ağırlıkların hangi oranda değiştirileceğini gösteren bir katsayıdır. Küçük seçilmesi durumunda ağın sonuca ulaşması yavaşlayacaktır. Büyük seçilmesi durumunda ise ağın global minimumu bulması zorlaşır. Bu nedenle, öğrenme katsayısı 0,01 ile 0,9 arasında bir değer seçilir. δ_j ara katmandaki veya çıkış katmanındaki herhangi bir *j* nöronuna ait bir faktördür. (Denklem 2.3)

$$\delta_j = \frac{\partial f}{\partial net_j}\left(O_{dj}^t - O_j\right) \qquad \textbf{(2.3)}$$

Burada O_{dj}^{t} , j işlemci elemanın hedeflenen çıkış değeridir. net_j şu şekilde hesaplanır. (Denklem 2.4)

$$net_j = \sum x_i w_{ij} \quad \textbf{(2.4)}$$

Gizli katmanlardaki nöronların δ_j faktörü, çıkış katmanından giriş katmanına doğru aşağıda verilen şekilde güncellenir. (Denklem 2.5)

$$\delta_j = \frac{\partial f}{\partial net_j} \sum w_{qj} \delta_q \quad \textbf{(2.5)}$$

BÖLÜM 3. ALANDA PROGRAMLANABİLİR KAPI DİZİLERİ

3.1. Programlanabilir Lojik

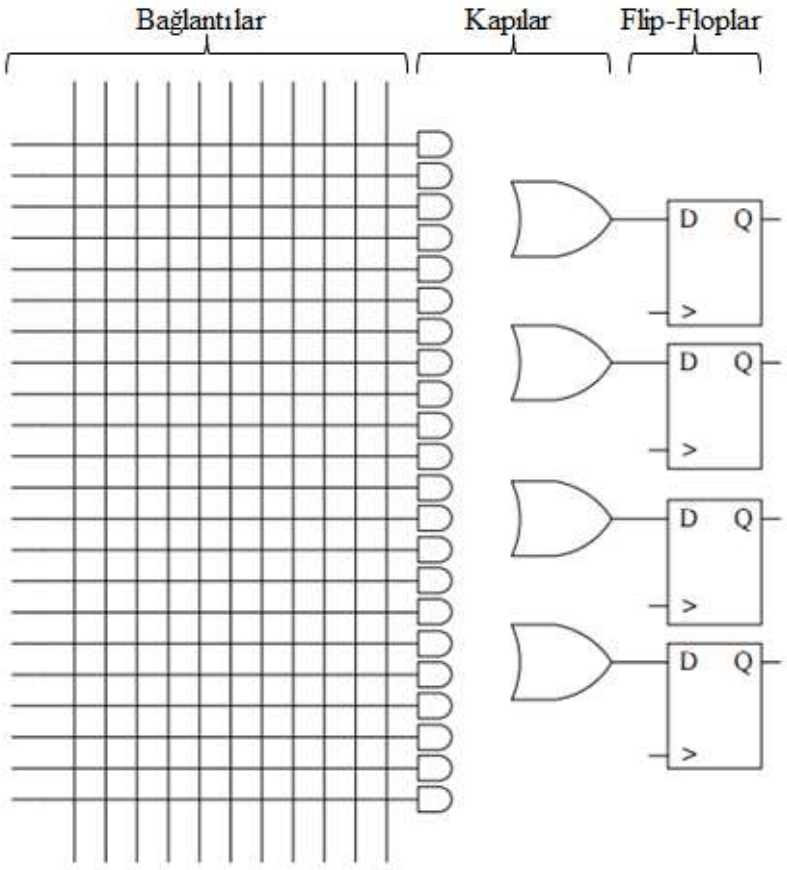
Geniş kullanım alanına sahip ilk programlanabilir lojik devreler, salt okunur bellek (ROM, Read Only Memory) devreleridir. Programlanabilen ROM elemanlarına PROM adı verilir. PROM'un özellikleri temelde ROM ile aynı olup, bir kez programlanabilir. ROM'dan farklı olarak PROM'lar üretim esnasında programlanmak zorunda değildir. Programlanabilen ve silinebilen EPROM VE EEPROM devreleri de üretilmiştir. EPROM elektrik ile programlanıp, ultraviyole ışık ile silinebilen ROM çeşididir. EEPROM'da ise yazma ve silme işlemleri elektrik ile gerçekleştirilir [113].

3.1.1. SPLD

İçerisinde AND ve OR kapılarından oluşan diziler bulunan programlanabilir lojik aygıtlar (PLD), mantıksal devreler oluşturmak için kullanılırlar. SPLD (Simple Programmable Logic Device), basit programlanabilir lojik aygıt olarak adlandırılır. Bir SPLD ünitesinde 4 ila 22 arası programlanabilir hücre vardır. Programlanabilir lojik ailesinin en ucuz üyesi olan SPLD'ler teknolojilerine göre isimlendirilirler [113, 114].

- PROM (Programlanabilir Salt Okunur Bellek)
- PAL (Programlanabilir Dizi Lojik)
- GAL (Genel Dizi Lojik)
- PLA (Programlanabilir Lojik Dizi)
- PLD (Programlanabilir Lojik Aygıt)

Basit bir SPLD yapısı Şekil 3.1'de gösterilmiştir.



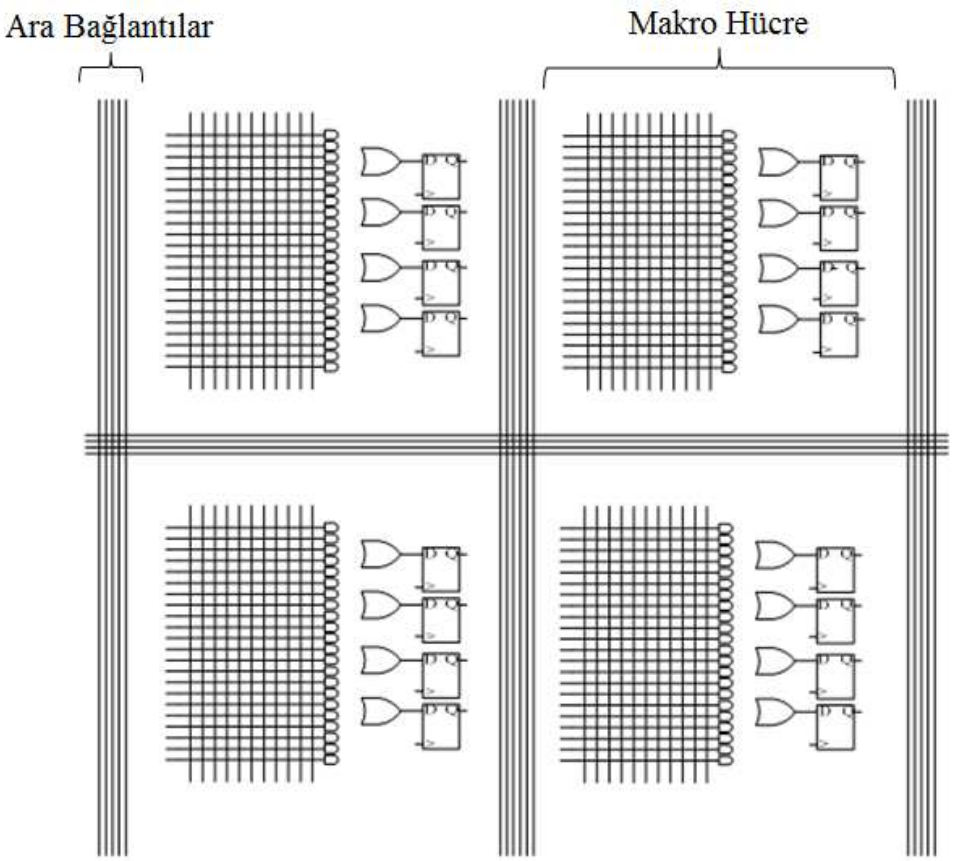
Şekil 3.1. Basit SPLD yapısı

3.1.2. CPLD

CPLD (Complex Programmable Logic Device), karmaşık programlanabilir lojik aygıt olarak adlandırılır. CPLD'ler, SPLD'lere göre daha fazla kapasiteye sahiptiler. CPLD'nin modeline göre 8 ile 16 arasında makro hücre bir araya gelerek bir fonksiyon bloğu oluştururlar. Bu fonksiyon blokları arasında iletişim sağlanmıştır. Programlanabilir anahtar matrisi ile işlem yapılacak blok belirlenir. CPLD'ler teknolojilerine göre literatürde aşağıdaki isimler ile anılırlar [114].

- EPLD (Silinebilir Programlanabilir Lojik Aygıt)
- PEEL (Programlanabilir Elektriksel-Silinebilir Lojik)
- EEPLD (Elektriksel-Silinebilir Programlanabilir Lojik Aygıt)
- MAX (Çoklu Dizi Matrisi, Altera)

Şekil 3.2'de basit CPLD yapısı gösterilmiştir.



Şekil 3.2. Basit CPLD yapısı

3.1.3 MPGA

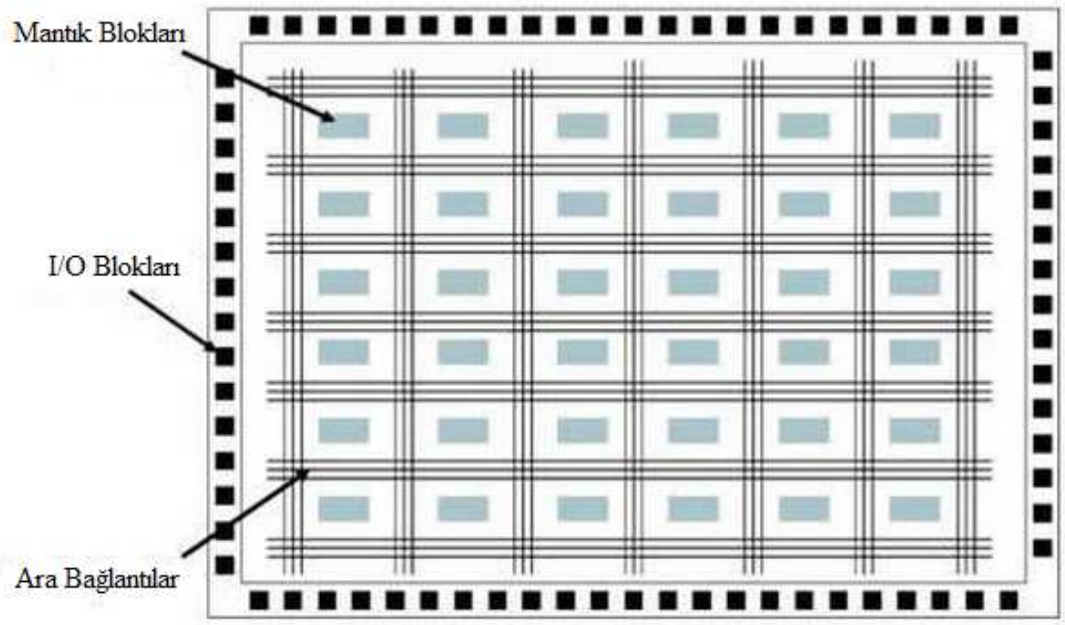
Bir MPGA (Mask Programmable Gate Array), arzu edilen lojik devreyi oluşturmak amacıyla birbirine bağlanmış transistörler ve hücrelerden meydana gelmektedir. Satır ve satırlar arası bağlantılar ile lojik kapılar birbirine bağlanır. MPGA'lar, özel üretim teknikleri ve üretimin zaman alması nedenleriyle yüksek sayıda üretim için uygundur. Üretim sayısı arttıkça maliyet azalır [113].

3.1.4. FPGA

FPGA, alanda programlanabilir kapı dizileri anlamına gelmektedir. Programlanabilir mantık blokları ile bu bloklar arasındaki ara bağlantılardan oluşan ve üretimden sonra iç konfigürasyonu tasarımcı tarafından değiştirilebilen tümleşik devrelerdir. Üretimden sonra programlanabilmesi nedeniyle alanda programlanabilir ismi verilmiştir. Standart entegrelerde, transistörler arasındaki bağlantılar sabittir ve

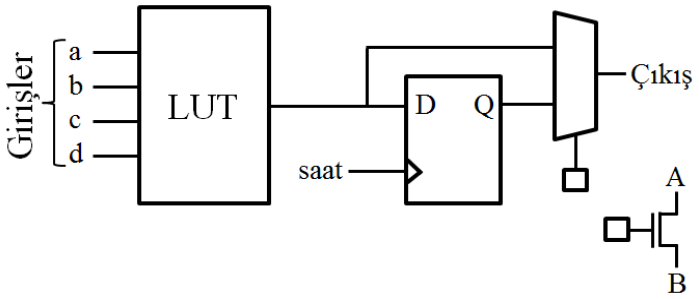
FPGA ile standart entegreler arasındaki fark, programlanabilir ara bağlantılara sahip olmasıdır. Mantık bloklarının fonksiyonu, programlanabilir ara bağlantılar yardımıyla tasarımcının ihtiyaç duyduğu şekilde düzenlenir [114, 115].

FPGA mimarisi aslında üç ana parçadan oluşmaktadır. Bunlar, mantık blokları, ara bağlantılar ve giriş/çıkış (I/O) bloklarıdır. I/O bloklar mantık bloklarının etrafını saracak biçimde şekillendirilmiştir. Hem mantık blokları arasında hem de I/O bloklar ile mantık bloklar arasında programlanabilir ara bağlantılar yer almaktadır [115]. FPGA'nın genel yapısı Şekil 3.3'te gösterilmiştir [116].



Şekil 3.3. FPGA genel yapısı

FPGA yapısındaki mantık bloklar, 1 adet LUT, 1 adet D flip-flop ve 2x1 MUX elemanlarından meydana gelmektedir. Mantık bloğunun yapısı Şekil 3.4'te gösterilmektedir. Tipik bir FPGA yongası onbinlerce mantık hücresi içerebilir. Programlanabilir ara bağlantılar ile binlerce mantık hücresi birleştirilerek karmaşık işlemler gerçekleştirilir [100].



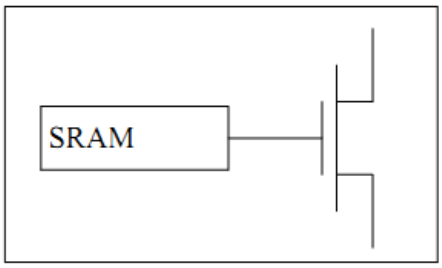
Şekil 3.4. Mantık hücresi [117]

3.1.4.1. FPGA üretim teknolojileri

Programlanabilir cihazlar, programlanabilir ve bir kez programlanabilir olmak üzere iki ana gruba ayrılmaktadır [115]. FPGA kartları farklı teknolojiler kullanılarak üretilmektedirler. Bu teknolojiler SRAM, sigorta (antifuse), EEPROM/Flash ve Flash-SRAM tabanlı mimarilerdir [114].

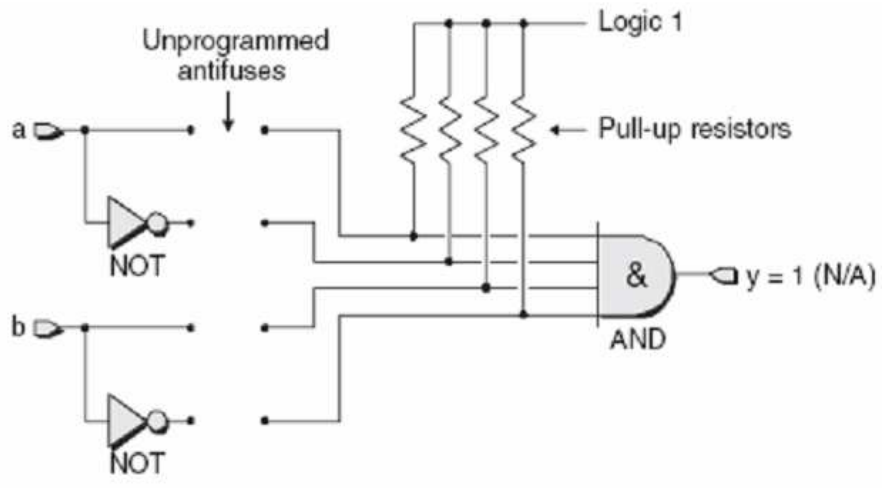
i) SRAM tabanlı mimari: Yazılabilir RAM yongaları DRAM (Dinamik RAM) ve SRAM (Statik RAM) olma üzere iki ana gruba ayrılmaktadır. DRAM'da her hücre bir transistör ve kapasitörden oluşmaktadır. Kapasitörler, yapıları gereği bir süre sonra boşalacağından belirli aralıklarla yükleri tazelenmelidir. Bu sebeple DRAM teknolojisi programlanabilir lojik cihazlar için uygun değildir. SRAM teknolojisinde ise RAM üzerindeki bilgi yükleme sonrasında sabit kalır ve güç kesilene kadar kaybolmaz [115].

FPGA'nın SRAM tabanlı yapılandırma hücreleri kullanması cihazın tekrar kullanılabilmesini sağlamaktadır. Böylece yeni tasarımlar kolay bir şekilde hazırlanıp test edilebilmektedir. SRAM teknolojisi yongada kullanılan diğer birimlerle aynı CMOS teknolojisine sahip olup ek bir geliştirme sürecine ihtiyaç duymazlar. 4 veya 6 transistörden oluşan bu mimaride güç kesildiğinde tüm veriler kaybolmaktadır. Buna rağmen hızlı ve yeniden programlanabilir olmaları büyük bir avantaj sağlamaktadır [114, 117]. SRAM tabanlı programlanabilir hücre Şekil 3.5'te gösterilmektedir.



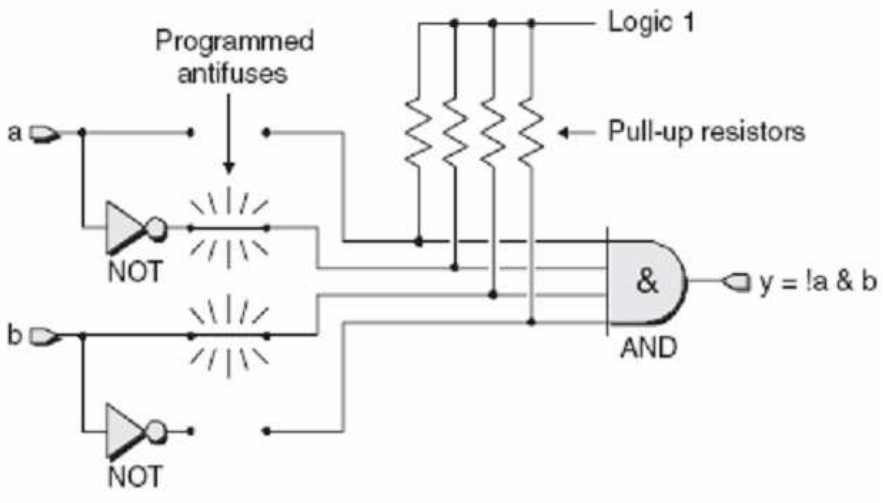
Şekil 3.5. SRAM tabanlı programlanabilir hücre [118]

ii) Sigorta (antifuse) tabanlı mimari: SRAM tabanlı FPGA'lerin aksine devre dışından özel araçlarla programlanırlar. Saklanan veriler güç kesilmesi durumunda silinmezler ve sistem açılışlarında tekrar yapılandırılma gereksinimleri yoktur. Bu sayede harici bellek ihtiyacı ortadan kalkmış olur [118]. Bu teknolojide, her yapılandırılabilir yol için programlanabilir bir bağlantı vardır. Programlanmamış sigorta yüksek empedans göstermektedir ve açık devre gibi davranmaktadır. Programlanmamış sigorta mimarisi Şekil 3.6'da gösterilmiştir.



Şekil 3.6. Programlanmamış sigorta mimarisi [118]

Yüksek gerilim uygulanması ile sigorta iletkene dönüşür ve bu bağlantı geri alınamaz. Bu sebeple sigorta tabanlı mimari bir kez programlanabilir bir yapıya sahiptir [115]. Programlanmış sigorta mimarisi Şekil 3.7'de gösterilmiştir.



Şekil 3.7. Programlanmış sigorta mimarisi [118]

iii) EEPROM/Flash tabanlı mimari: EEPROM tabanlı FPGA hücreleri, SRAM tabanlı mimaridekine benzer olarak uzun ötelemeli yazmaç şeklindeki zincirlerle bağlıdırlar. Aygıt içerisinde ve dışarısında programlamaya izin veren çeşitleri vardır. SRAM tabanlı FPGA'lara nazaran daha yavaştırlar [118].

Programlandıktan sonra içeriği kalıcı olarak saklarlar. Bu sayede sistem başlangıcında yeniden yapılandırılmaları gerekmemektedir. Güvenlik amaçlı olarak 50 bit ile birkaç yüz bit genişliğinde olabilen çoklu anahtar (multibit key) teknolojisini kullanmaktadırlar [114].

EPROM tabanlı aygıtlara göre daha büyük olmalarına karşın SRAM tabanlı aygıtlara nazaran daha küçüktürler. Bu da SRAM tabanlı aygıtlara göre daha az bağlantı gecikmesi anlamına gelmektedir. Diğer yandan standart CMOS teknolojisine ilaveten yaklaşık 5 adım gerektirdiğinden SRAM temelli aygıtların birkaç nesil gerisinde kalmaktadırlar.

iv) Karma Flash-SRAM tabanlı mimari: Her yapılandırma öğesi Flash tabanlı ve SRAM tabanlı aygıt hücrelerinin birleşimi şeklindedir. Bu mimaride önceden yapılandırılmış Flash hücrelerindeki veriler, sistem başlangıcı ile SRAM hücrelerine kopyalanır. Bu sayede hem sigorta tabanlı mimarideki kalıcılık sağlanmış olur hem de SRAM hücreleri sayesinde yeniden yapılandırma özelliği mümkün kılınır.

Farklı mimarilere sahip FPGA teknolojilerinin karşılaştırılması Tablo 3.1'de verilmiştir.

Tablo 3.1. Üretim teknolojilerinin karşılaştırılması

Özellik	**SRAM**	**Sigorta**	**Flash (EEPROM)**
Teknoloji	Gelişmiş	Birkaç nesil geride	Birkaç nesil geride
Tekrar programlanabilme	Evet	Hayır	Evet
Tekrar programlanabilme hızı	Hızlı	-----	SRAM'dan 3 kat yavaş
Uçuculuk (başta programlanmalı)	Evet	Hayır	Hayır
Harici program verisi gereksinimi	Evet	Hayır	Hayır
İlk örnek geliştirme	Evet (çok iyi)	Hayır	Evet (kabul edilebilir)
Başlangıçta çalışırlık	Hayır	Evet	Evet
IP güvenliği	Kabul edilebilir	Çok iyi	Çok iyi
Hücre genişliği	Geniş (6 transistör)	Çok küçük	Orta (2 transistör)
Güç tüketimi	Orta	Düşük	Orta
Radyoaktif dayanıklılık	Hayır	Evet	Hayır

3.2. VHDL

VHDL (Very high speed integrated circuit Hardware Description Language) yüksek hızlı tümleşik devreler için donanım tanımlama dili anlamına gelmektedir. Fiziksel olarak gerçeklenebilecek sayısal devrelerin tasarlanması ve denenmesi amacıyla yaygın olarak kullanılan özel bir dildir.

VHDL dili IEEE tarafından IEEE 1076-1987 standardı haline getirilmiştir. Daha sonra dile yeni özellikler eklenerek IEEE 1076-1993 standardı oluşturulmuştur. Yeni versiyon, ihtiyaç duyulan veri tipleri ve alt program paketlerini (std_logic_1164, std_logic_arith, numeric_bit, numeric_std, …) içermektedir. VHDL, FPGA

yongasına yüklenecek kodu sentezlemek ya da bu kodun simülasyonunu yapmak amacıyla kullanılmaktadır [114].

3.2.1. VHDL ile donanım tasarımı

VHDL dilinin, standart donanım tasarım yöntemine göre tasarım süresi, tasarım esnekliği ve uygulama kolaylığı açısından önemli üstünlükleri vardır.

3.2.1.1. Tasarım süresi

Teknolojideki hızlı gelişimin devrelerin kullanım ömrünü azaltması sebebiyle tasarım süresinin kısalığı önem arz etmektedir. Devre tasarımı, VHDL dili kullanılarak doğrudan tasarıma göre çok daha kısa sürede gerçekleştirilebilmektedir [114].

3.2.1.2. Tasarım esnekliği

Teknolojideki hızlı değişim yonga yapılarını da değiştirmekte, fiziksel boyutlar küçülürken işlem yeteneği artmaktadır. Daha önceden gerçekleştirilmiş bir tasarımın yeni nesil yongalarda gerçeklenebilmesi için ortamın buna uygun olması gerekmektedir. VHDL tasarım dili ile gerçekleştirilen bir tasarımın donanım yapısı dönüştürücü programlar vasıtasıyla oluşturulur. VHDL ile oluşturulmuş bir tasarımın yeni nesil yongalara uygulanabilmesi için dönüştürücü programların bu yongaları destekler hale getirilmesi yeterli olacaktır [114].

3.2.1.3. Uygulama kolaylığı

VHDL ile gerçekleştirilen uygulamalarda donanım bilgisine daha az ihtiyaç duyulmaktadır. Böylece devreler VHDL kullanılarak kolaylıkla tasarlanabilmektedir.

3.2.2. VHDL veri nesneleri

Bir veri nesnesi, belirli bir tipin değerini tutar ve donanımda doğrudan sentezlenir. Sinyal (signal), değişken (variable) ve sabit (constant) olmak üzere üç farklı veri nesnesi vardır. En önemli veri nesnesi sinyaller olup devredeki lojik sinyalleri tanımlamaktadırlar. Değişkenler ve sabitler sinyallere nazaran daha seyrek kullanılmaktadırlar [114].

Sinyal (Signal): Güncel değer ve olası sonraki değerleri içeren bir listeye sahiptir. Sinyal tanımlaması aşağıdaki gibi yapılır. Aşağıda sinyal tanımlama örnekleri verilmiştir.

```
SIGNAL cntrl_bit: STD_LOGIC;
SIGNAL data_1: STD_LOGIC_VECTOR (0 TO 7);
SIGNAL data_in: STD_LOGIC_VECTOR (15 DOWNTO 0);
```

Değişken (Variable): Geçici bilgi saklayan değişkenler, hesaplamaların sonuçlarını tutmak için kullanılırlar. Örnek değişken tanımlamaları aşağıda verilmiştir.

```
VARIABLE cst : INTEGER range 0 to 99 := 20 ;
VARIABLE bellek : BIT_MATRIX (0 to 7, 0 to 511) ;
```

Sabit (Constant) : Program boyunca değişmeyecek bir değer içerir. Simülasyon başlamadan saptanmalıdır. Örnek sabit tanımlamaları aşağıda verilmiştir.

```
CONSTANT dongu_zamani: TIME := 10 ns;
CONSTANT cnst: UNSIGNED (7 downto 0);
CONSTANT LogicalGND: BIT:= 0;
```

3.2.2.1 Ön tanımlı veri tipleri

VHDL tasarım dilinde bütün veri nesneleri bir veri tipi ile tanımlanmaktadır. Ön tanımlamalı veri tipleri bit, tamsayı, standart lojik, kayan noktalı, fiziksel ve liste tipli

olmak üzere 6 çeşittir [114]. Ön tanımlamalı veri tipleri Tablo 3.2'de gösterilmektedir.

Tablo 3.2. Ön tanımlamalı veri tipleri

Veri Tipi	Tanımlama
Bit	BIT, BIT_VECTOR
Tamsayı	INTEGER
Standart Lojik	STD_LOGIC, STD_LOGIC_VECTOR
Kayan noktalı	REAL
Fiziksel	TIME
Liste	CHARACTER, BOOLEAN

Bit tipine göre daha esnek bir yapıya sahip olan standart lojik veri tiplerini tasarımda kullanmak için programın başına USE IEEE.STD_LOGIC_1164.ALL; ifadesi yazılmalıdır.

3.2.2.2. Operatörler

VHDL tasarım dilinde kullanılan temel operatörler yüksek öncelikli ve düşük öncelikli olarak sınıflandırılabilir. Temel operatörler Tablo 3.3'te verilmiştir [114].

Tablo 3.3. Operatörler

Öncelik	Operatör Sınıfı	Operatör
Yüksek Öncelikli	Çeşitli	abs, not, **
	Çarpım	*, /, mod, rem
	İşaret	+, -
	Toplama	+, -, &
	Kaydırma/Döndürme	sll, srl, sla, sra, rol, ror
Düşük Öncelikli	İlişkisel	= , /= , < , <= , > , >=
	Mantıksal	and, or, nand, nor, xor, xnor

3.2.3. VHDL temel yapıları

Sayısal bir sistemin VHDL tanımlaması yapılırken 4 temel yapıdan faydalanılır. Bu yapılar varlık (entity), mimari (architecture), biçim (configuration) ve paket (package) yapılarıdır.

3.2.3.1. Varlık (entity)

Entity, VHDL dilinin temel elemanıdır ve her tasarım bir entity bölümü ile ifade edilir. Entity ile tasarım ve dış çevre arasındaki arayüz tanımlanır fakat aralarındaki ilişki verilmez. Yapısında bulundurduğu PORT sayesinde giriş ve çıkış sinyallerinin biçimi (IN, OUT, INOUT ve BUFFER) belirlenir. IN, giriş için kullanılan sadece okunabilir varlık elemanıdır. OUT, çıkış için kullanılan yazma özelliğine sahip varlık elemanıdır. INOUT hem okuma hem de yazma özelliklerine sahip varlık elemanı olup giriş ve çıkışlarda kullanılır. BUFFER ise hem okunabilir hem de yazılabilir tampon elemanıdır [114]. Bir tasarım için varlık tanımlaması aşağıdaki şekilde yapılabilir.

```
ENTITY tasarim_adi IS
PORT(
        Port tanımlamaları
);
END tasarim_adi;
```

3.2.3.2. Mimari (architecture)

Mimaride, bir tasarımın işlevi tanımlanır. Bu bölümde entity kısmında tanımlanmış portlar arasındaki ilişkiler gösterilir [114, 116]. Bir mimari aşağıdaki şekilde tanımlanır.

```
ARCHITECTURE mimari_adi OF tasarim_adi IS
        Sinyal tanımlamaları;
        Sabit tanımlamaları;
```

```
        Tip tanımlamaları;
        Parça tanımlamaları;
BEGIN
        İşlem ifadeleri;
END mimari_adi;
```

3.2.3.3. Biçim (configuration)

Biçim, alt bileşenlerin nasıl bir araya gelerek tasarım oluşturduğunu, blokların nasıl bağlandığını belirleyen yapıdır [116]. Biçim aşağıdaki şekilde tanımlanır.

```
CONFIGURATION configuration_adi OF entity_adi IS
        Biçim tanımlamaları
FOR mimari_adi
        FOR cagirma_etiketi: component_adi
        USE entity kutuphane_adi.entity_adi(architecture_adi);
        END FOR;
        Diğer ifadeler
END FOR;
END [configuration] [configuration_adi];
```

3.2.3.4. Paket (package)

Paket, deklarasyonları farklı tasarımlarda kullanmak üzere gruplamaya yarayan bir birimdir. İki ya da daha fazla birim tarafından kullanılan ortak elemanlar bir araya toplanmıştır. Bir paket, paket bildirimi ve paket gövdesi olmak üzere iki bölümden oluşur. Paket içerisinde, sabitler, veri tipleri, bileşenler, fonksiyonlar ve alt programlar tanımlanabilir ve böylece birçok tasarımda kullanılabilirler [116]. Bir paket bildirimi aşağıdaki şekilde tanımlanır.

```
PACKAGE paket_adi IS
        Paket_bildirimleri
END PACKAGE paket_adi;
```

Alt program tanımlamalarını içeren paket gövdesi aşağıdaki şekilde tanımlanır.

```
PACKAGE BODY paket_adi IS
        Paket_gövdesi_bildirimleri
        Alt programlar
        Sabit bildirimleri
END PACKAGE BODY paket_adi;
```

3.2.4 Altprogramlar

Altprogramlar, tekrar kullanılabilmeleri sayesinde VHDL dilinde de önemli bir yere sahiptirler. VHDL dili “procedure” ve “function” olmak üzere iki tip altprogram yapısını desteklemektedir. Bunlardan “procedure” tipi altprogramlar dönüş değeri almazken “function” tipi altprogramlar dönüş değeri döndürürler [114].

BÖLÜM 4. QUARTUS II YAZILIMI

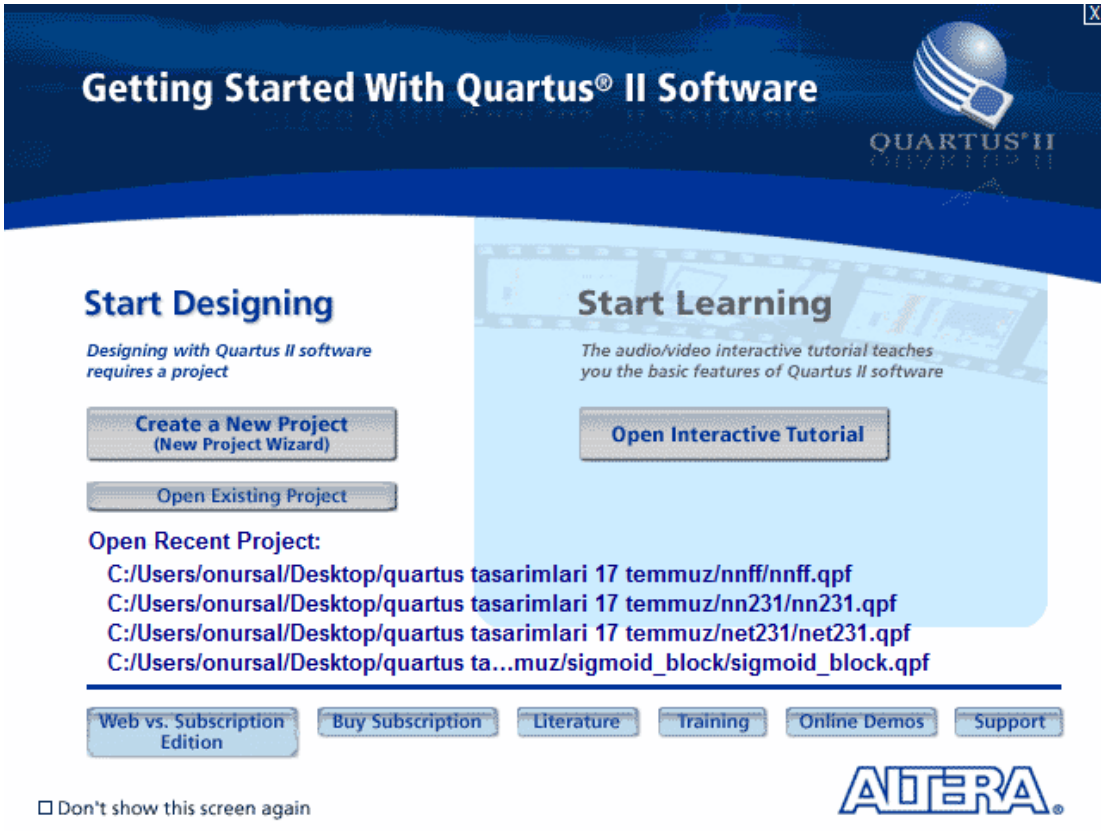
Quartus II, Altera firması tarafından özel tasarım ihtiyaçlarına cevap verebilmek amacıyla geliştirilmiş bir CAD (Computer Aided Design) programıdır [119]. FPGA ve CPLD tasarım aşamaları için çözümler içermektedir. Quartus II programında lojik tasarım yazılımsal ya da donanımsal olarak gerçekleştirilir. Yazılımsal tasarım için Vhdl veya Verilog programlama dilleri, donanımsal tasarım içinse şematik çizim kullanılır.

4.1. Yeni Bir Proje Oluşturma

Quartus II ile tasarlanan her bir lojik devreye proje adı verilir. Program ilk çalıştırıldığında Şekil 4.1'de gösterilen başlangıç penceresi açılır.

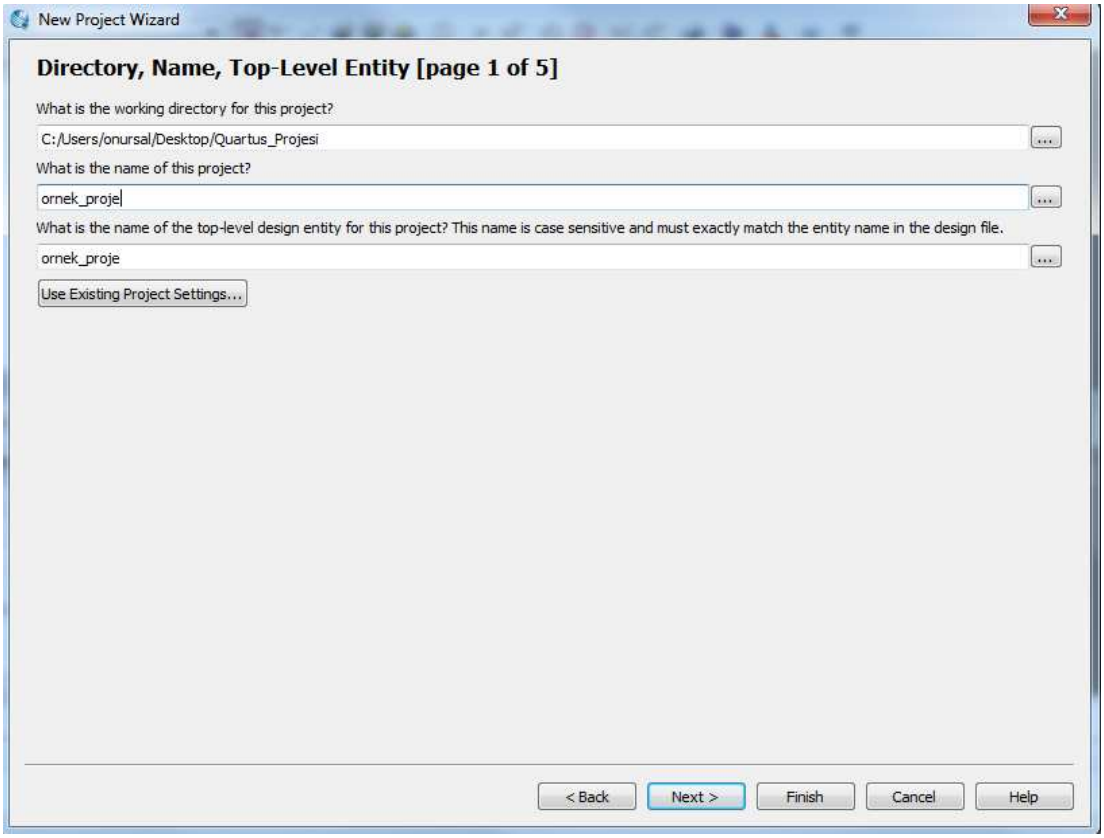
Bu pencerede "Create a New Project" sekmesine fare ile tıklanarak yeni bir proje oluşturmak için ilk adım atılmış olur. Yeni bir proje oluşturmak için kullanılan bir diğer yöntem de Şekil 4.2'de verilen Quartus II ana penceresinde "File | New Project Wizard" seçeneğinin seçilmesidir.

Daha sonra proje dizininin ve proje adının seçildiği Şekil 4.3 ile gösterilen ekran açılır. Bu ekranda, üst düzey tasarım öğesinin ismi, yazılım tarafından proje ismi olarak önerilir. Proje ismini belirlemek kullanıcıya bırakılmış olup bu uyarı dikkate alınmayabilir. Şekil 4.3'te proje ismi olarak "Quartus_Projesi" belirlenmiştir. "Next (İleri)" tuşu yardımıyla "C:/Users/Desktop/Quartus_Projesi" dizini oluşturulur. Henüz dizin oluşturulmadı ise Şekil 4.4'te gösterilen uyarı penceresi ile kullanıcıya dizin oluşturmak isteyip istemediği sorulur. Şekil 4.5'te gösterilen pencerede, tasarım esnasında ihtiyaç duyulacak dosyalar eklenebilmektedir.

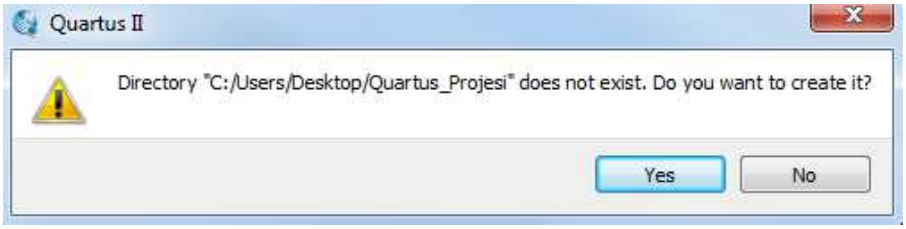


Şekil 4.1. Quartus II başlangıç penceresi

Şekil 4.2. Quartus II ana penceresi



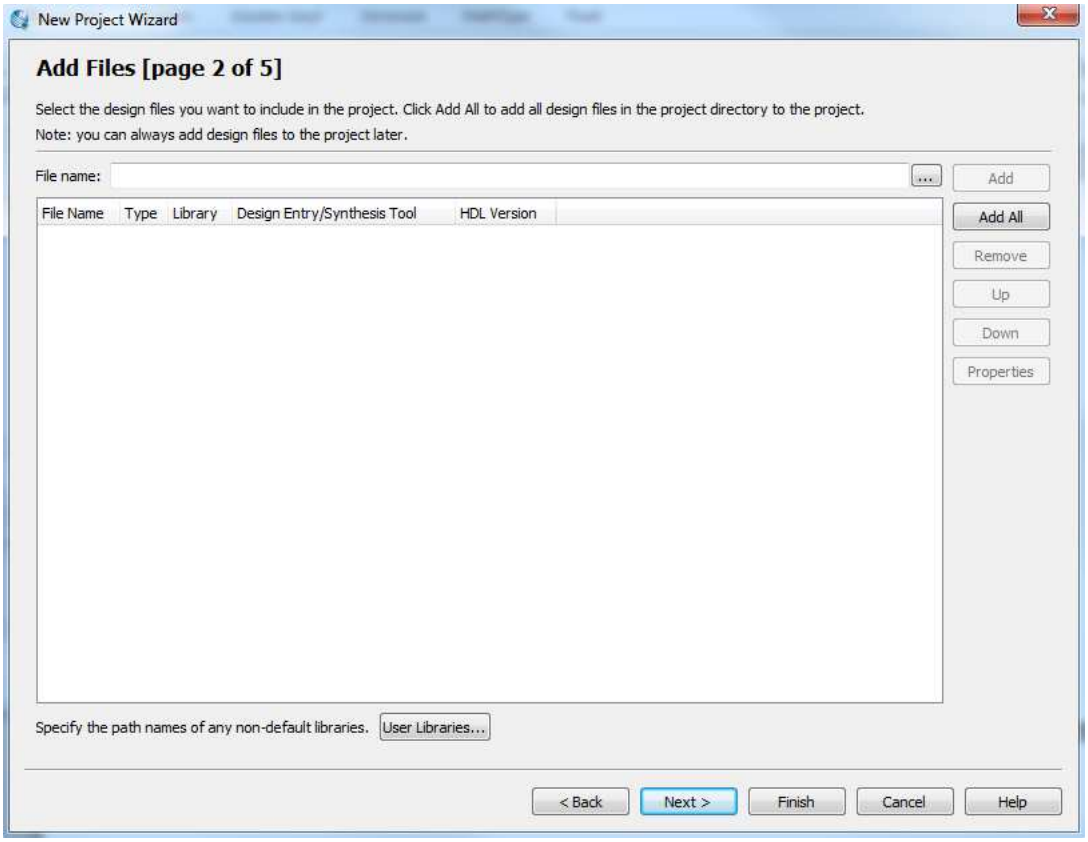
Şekil 4.3. Proje isminin ve dizininin belirlenmesi



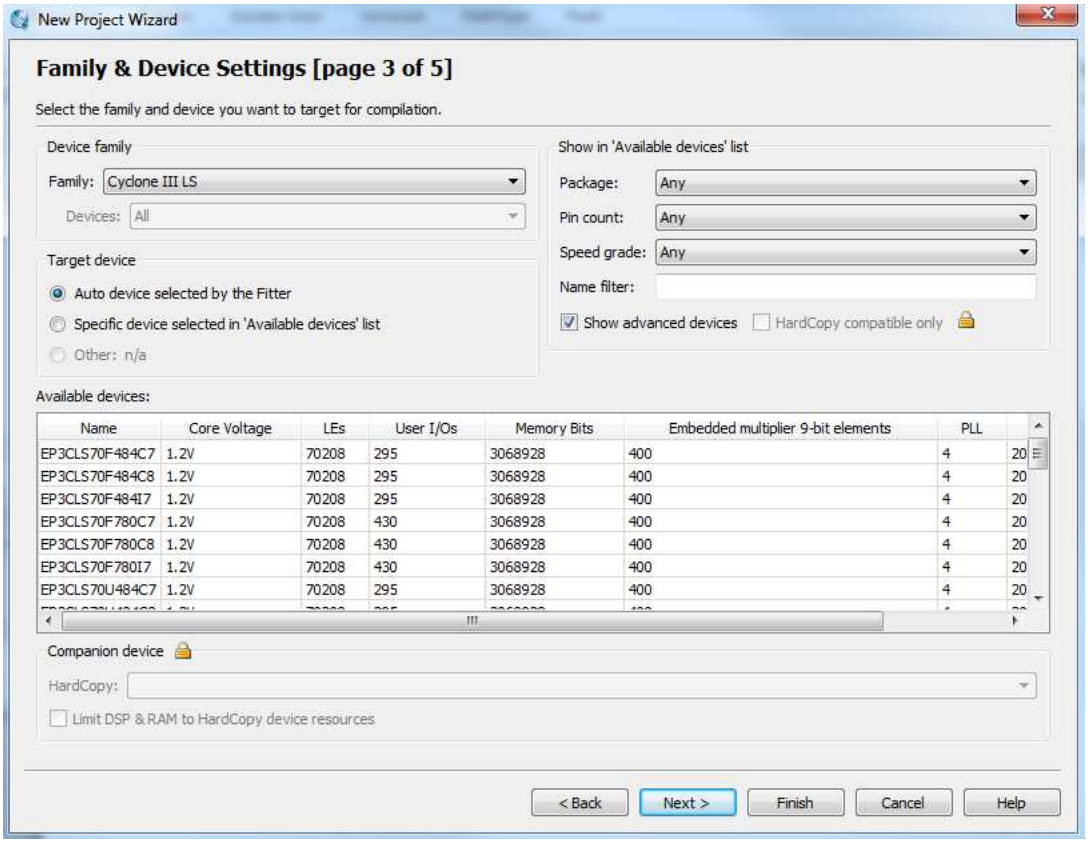
Şekil 4.4. Proje dizininin onaylanması

Şekil 4.6'da gösterilen pencerede, tasarlanacak projenin hangi aygıt üzerinde yürütüleceği belirlenir. Bu aşamada seçim yapılırken, FPGA yongası üzerindeki lojik eleman sayısına ve hafıza elemanlarına ve giriş-çıkış pin sayısına dikkat edilmelidir.

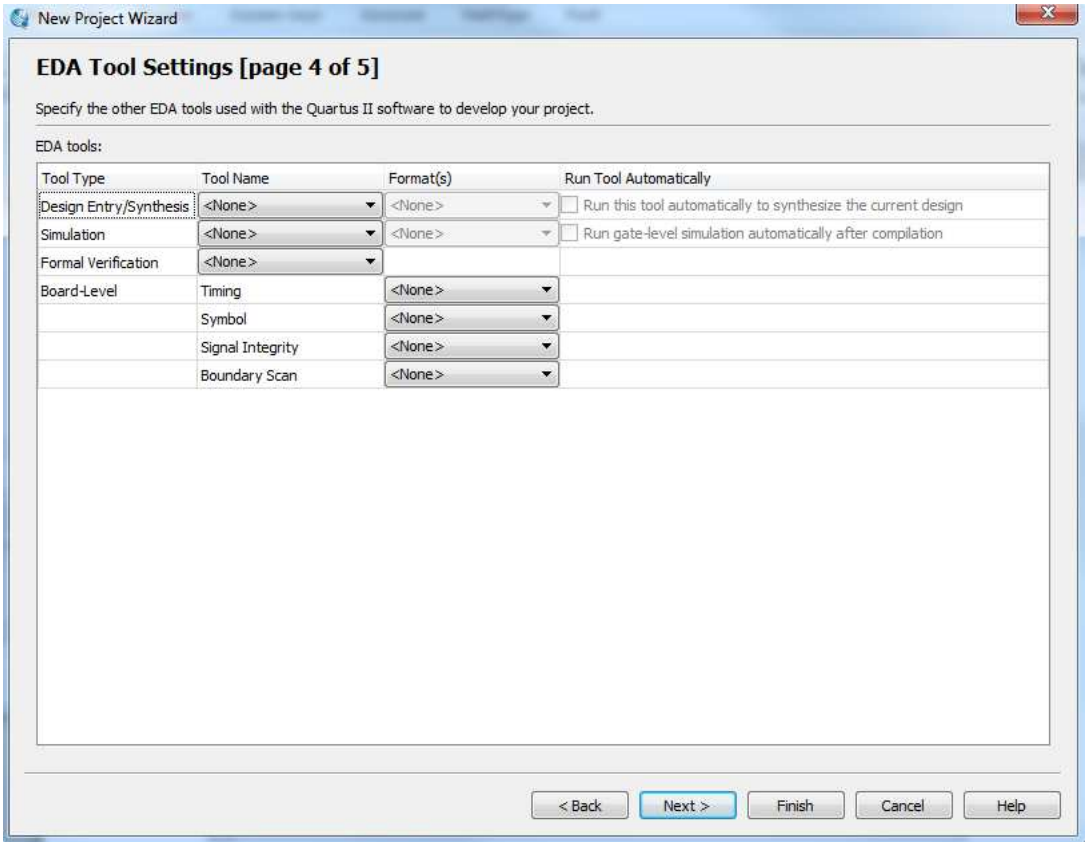
Şekil 4.7 ile gösterilen pencerede Altera dışındaki diğer şirketlerin geliştirdiği üçüncü parti CAD araçlarının kullanılıp kullanılmayacağı belirlenmektedir. Şekil 4.8'de verilen bir sonraki pencerede, tasarıma dair gerçekleştirilen seçimlerin kısa bir özeti kullanıcıya gösterilmektedir. "Finish (Bitir)" sekmesi ile yeni bir proje için gerekli özellikler tanımlanmış olur ve Şekil 4.2'de verilen ana pencereye dönülerek tasarım işlemine geçilir.



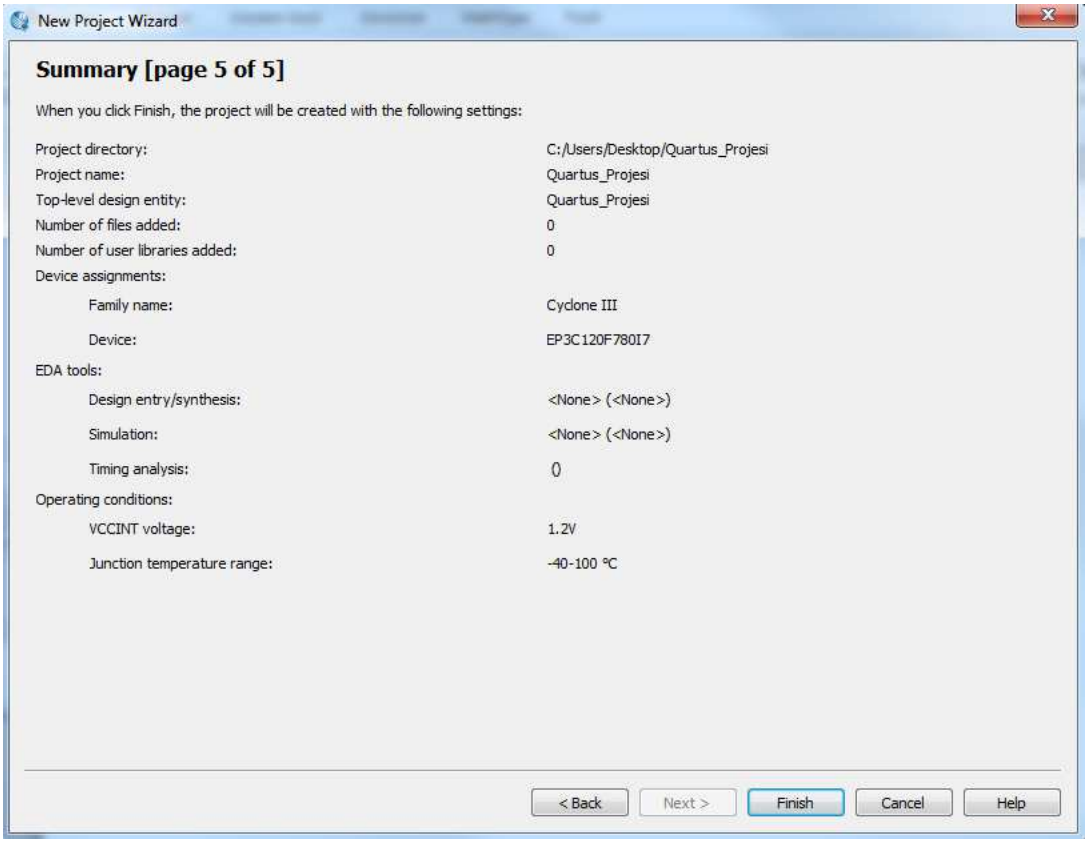
Şekil 4.5. Mevcut dosyaların tasarıma eklendiği pencere



Şekil 4.6. Aygıt ailesinin belirlendiği pencere



Şekil 4.7. EDA araçlarının seçilmesi



Şekil 4.8. Oluşturulan projeye ait özet

4.2. Şematik Çizim Kullanılarak Tasarıma Giriş

Bu kısımda, şematik çizim ile bir rpojenin nasıl oluşturulacağını analatmak amacıyla örnek bir tam-toplayıcı devresi tasarlanacaktır. Tam-toplayıcı devresi *A*, *B* ve *Cin* (elde giriş) girişleri ile *S* ve *Cout* (elde çıkış) çıkışlarından meydana gelmektedir. Devreye ait doğruluk tablosu Şekil 4.9'da verilmiştir.

Inputs			Outputs	
A	***B***	C_{in}	C_{out}	***S***
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	1	0
0	0	1	0	1
1	0	1	1	0
0	1	1	1	0
1	1	1	1	1

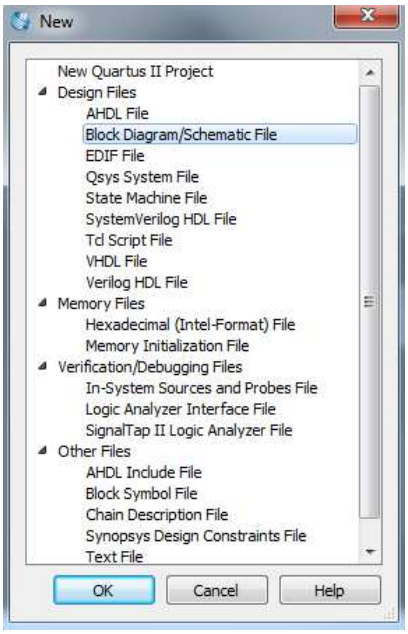
Şekil 4.9. Tam-toplayıcı doğruluk tablosu

Devreyi çizmek için, ilk olarak "File/New" seçeneği ile açılan pencereden "Block Diagram/Schematic File" seçilir. Şekil 4.10 ile gösterilen bu pencerede kullanıcının devreyi VHDL, Verilog, AHDL gibi yazılımlarla mı yoksa şematik olarak mı tasarlayacağı belirlenmektedir. "Block Diagram/Schematic File" seçeneği ile Şekil 4.11'de gösterilen block editör ekranı açılır. Devre çizimi bu ekran üzerinde gerçekleştirilir.

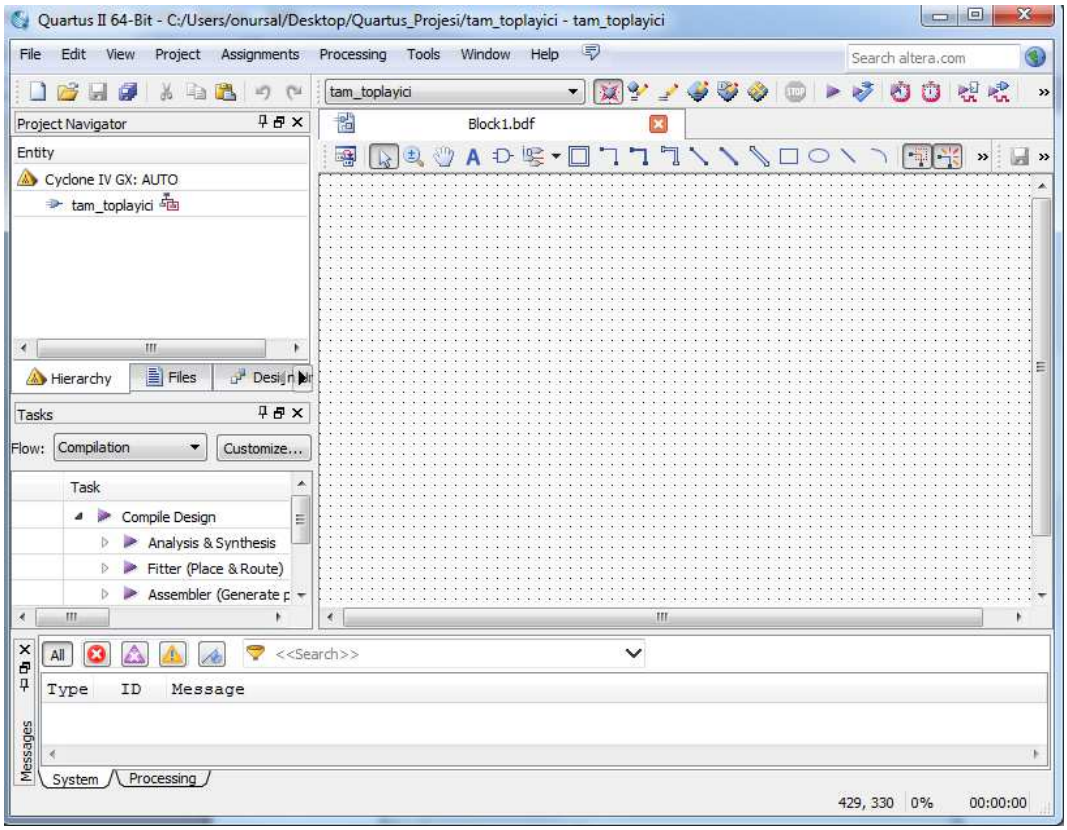
4.2.1. Kullanılan lojik elemanların eklenmesi

Block editörün, devrede kullanılacak lojik elemanları içeren kütüphaneleri bulunmaktadır. Tam-toplayıcı devresi için kullanılacak lojik kapılar, Şekil 4.12'de gösterildiği gibi "primitives" kütüphanesinden seçilir. Tam-toplayıcı devresi için 2 adet 2 girişli *AND* kapısı, 2 adet *XOR* kapısı ve 1 adet *OR* kapısı seçilir. Daha sonra bu devre elemanları, fare yardımıyla devrede kullanılacakları noktalara taşınırlar. Tam-toplayıcı devresi için eklenen lojik elemanlar Şekil 4.13'te gösterilmiştir. Aynı

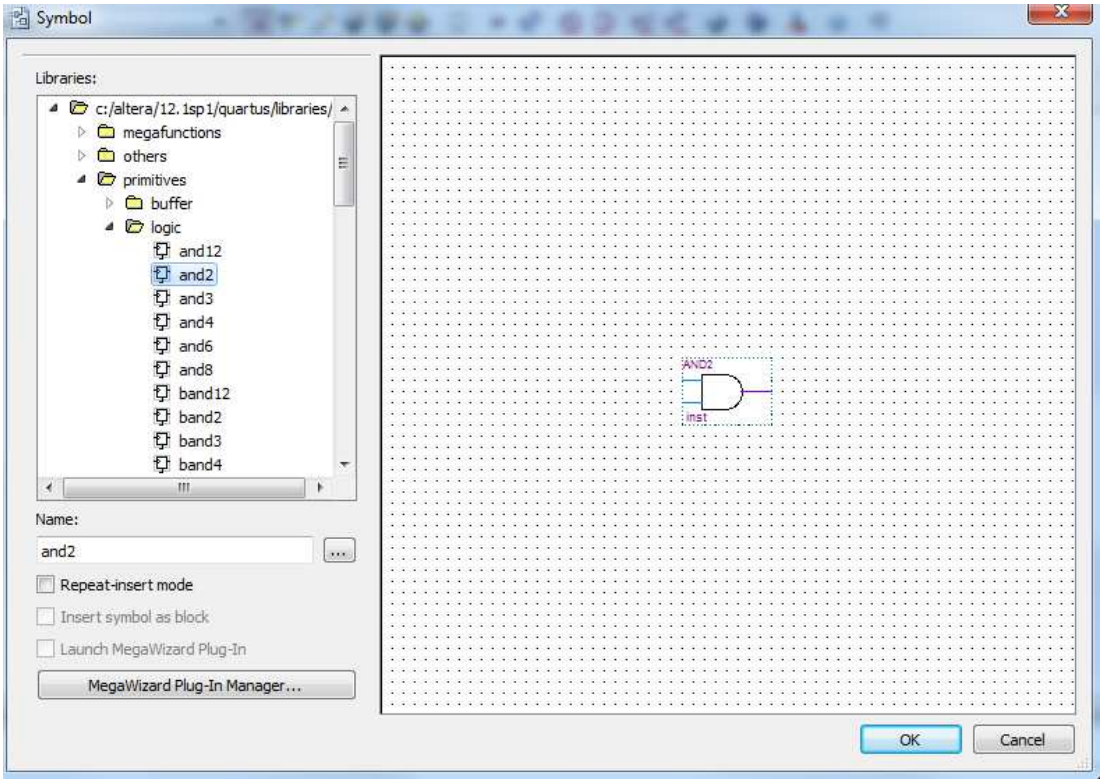
devre elemanından birden fazla kullanılacaksa, her seferinde kütüphaneyi açmak gerekmez. Devre elemanı/elemanları seçilip, klavyede CTRL tuşuna basılı iken sürüklenerek ilgili kısmın kopyası elde edilir.



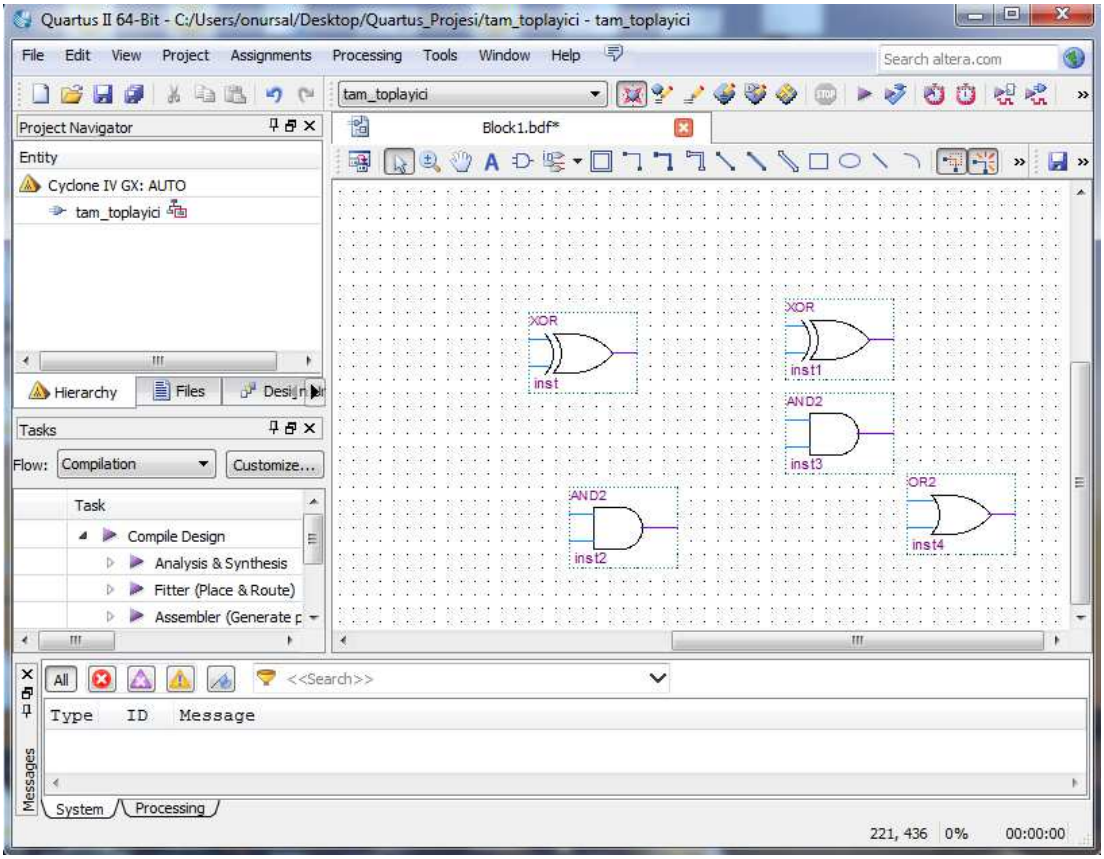
Şekil 4.10. Tasarım dosya türünün seçilmesi



Şekil 4.11. Block editör ekranı



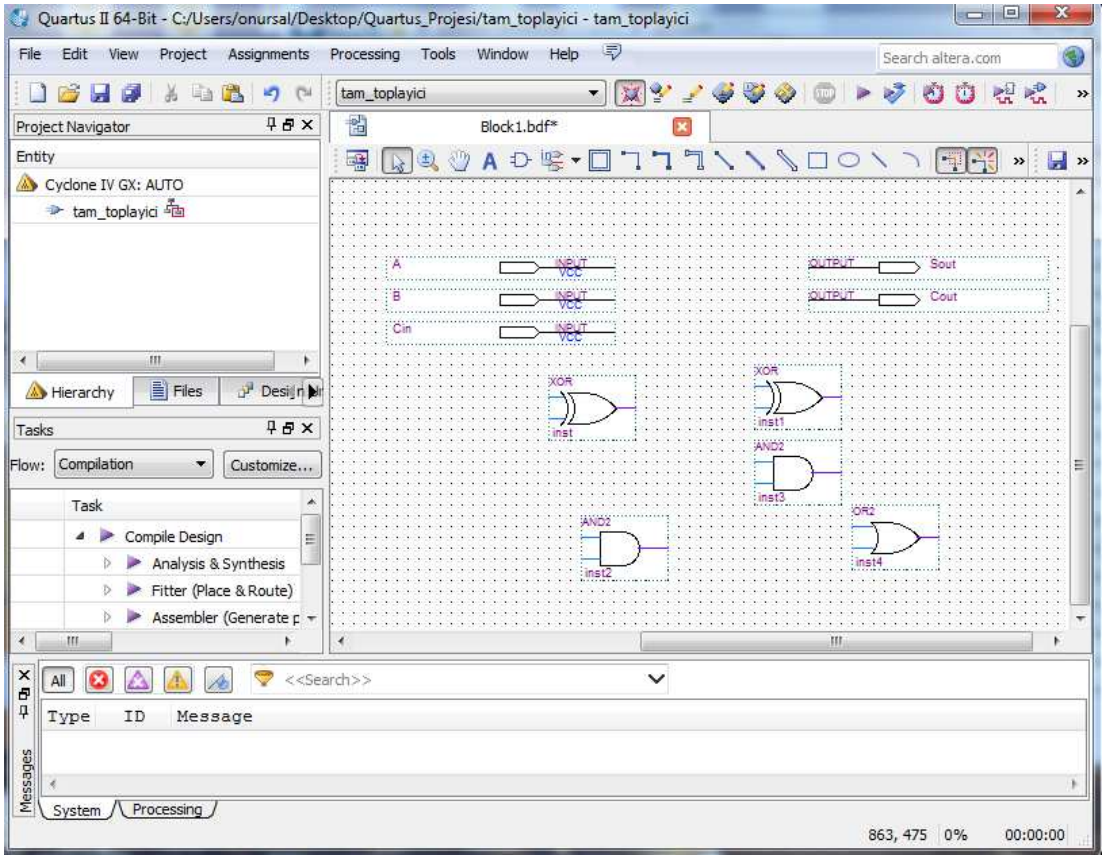
Şekil 4.12. Lojik elemanların devreye eklenmesi



Şekil 4.13. Devreye eklenen lojik elemanlar

4.2.2. Giriş-çıkış sembollerinin eklenmesi

Tasarlanan devrenin giriş-çıkış portlarına veri gönderilmesi için giriş-çıkış tanımlamaları yapılmalıdır. Bu amaçla, “primitives” kütüphanesindeki “pin” klasöründen girişler için input, çıkışlar için output sembolleri eklenir. Eklenen giriş ve çıkışlar, sembol fare ile çift tıklanarak isimlendirilir. Eklenmiş ve isimleri değiştirilmiş giriş-çıkış sembolleri Şekil 4.14’te gösterilmiştir.

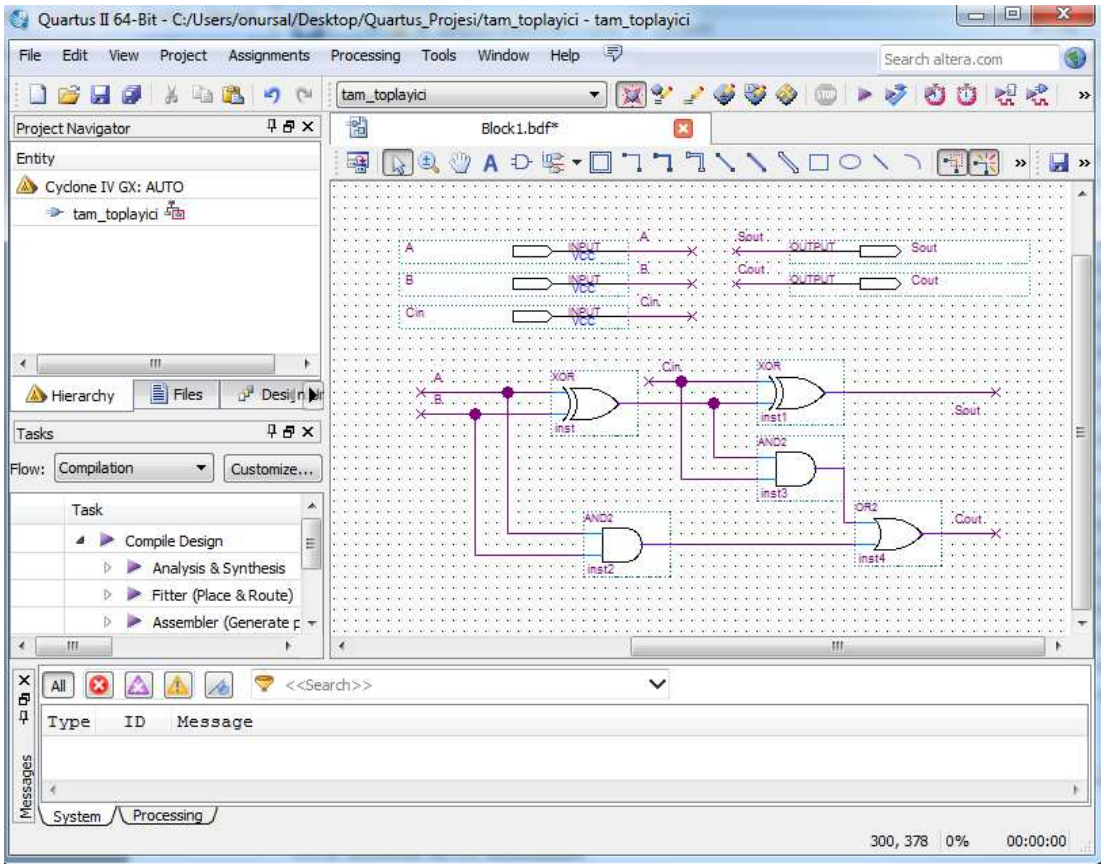


Şekil 4.14. Giriş-çıkışların düzenlenmesi

4.2.3. Hat (wires) ile bağlantıların yapılması

Devredeki sembolleri birbirine bağlamak için yatay araç çubuğunda yer alan büyük bir ok ucu şeklindeki “Selection Tool” (Seçim Aracı) seçilir. Farenin imleci bağlantı yapılacak devre elemanlarının giriş ya da çıkış uçlarına gelindiğinde “+” halini alacaktır. Bu durumda farenin sol tuşuna basılı tutarak bağlanmak istenen iki nokta arasındaki hat sağlanmış olur.

Çok karmaşık tasarımlarda, uzak iki nokta arasındaki bağlantı hat çekmek yerine düğümlerin etiketlenmesi ile yapılabilir. Şekil 4.15 ile gösterilen devrede giriş ve çıkış düğümleri etiketlenmiştir.



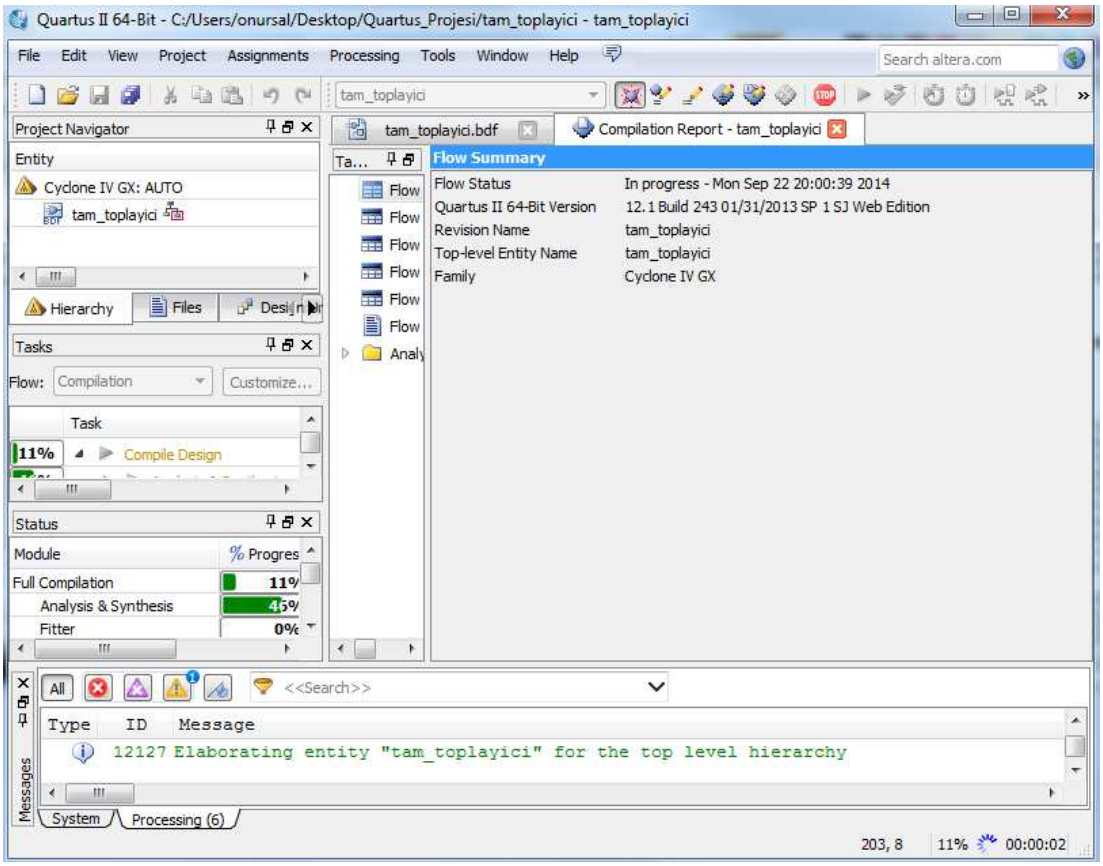
Şekil 4.15 Tamamlanmış devre

4.3. Derleyici Kullanımı

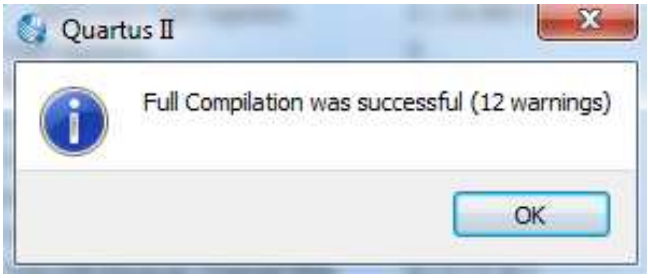
Quartus II içinde yer alan Analysis&Synthesis modülü sentezleme adımlarını yürütmektedir. Lojik elemanlardan çip içerisinde her bir elemanın doğrudan uygulandığı devre üretimini yapar. Fitter modülü, sentez sonucu üretilen elemanların çip üzerindeki konumunu belirlemektedir [114].

Quartus II modülleri Compiler (Derleyici) olarak adlandırılan uygulama programı tarafından kontrol edilir. Derleyici tek seferde bir modülü ya da sırayla birden fazla modülü çalıştırabilir. Processing | Start Compilation ya da araç çubuğunda yer alan “ ” simgesi ile derleme işlemi başlatılır.

Derleme devam ederken işlemin hangi aşamada olduğu Şekil 4.16'da gösterilen ekranın sağ alt köşesinden ve sol taraftaki Status penceresinden görülebilir. Başarılı ya da başarısız derleme sonucu Şekil 4.17'deki pencere ile tasarımcıya bildirilir.

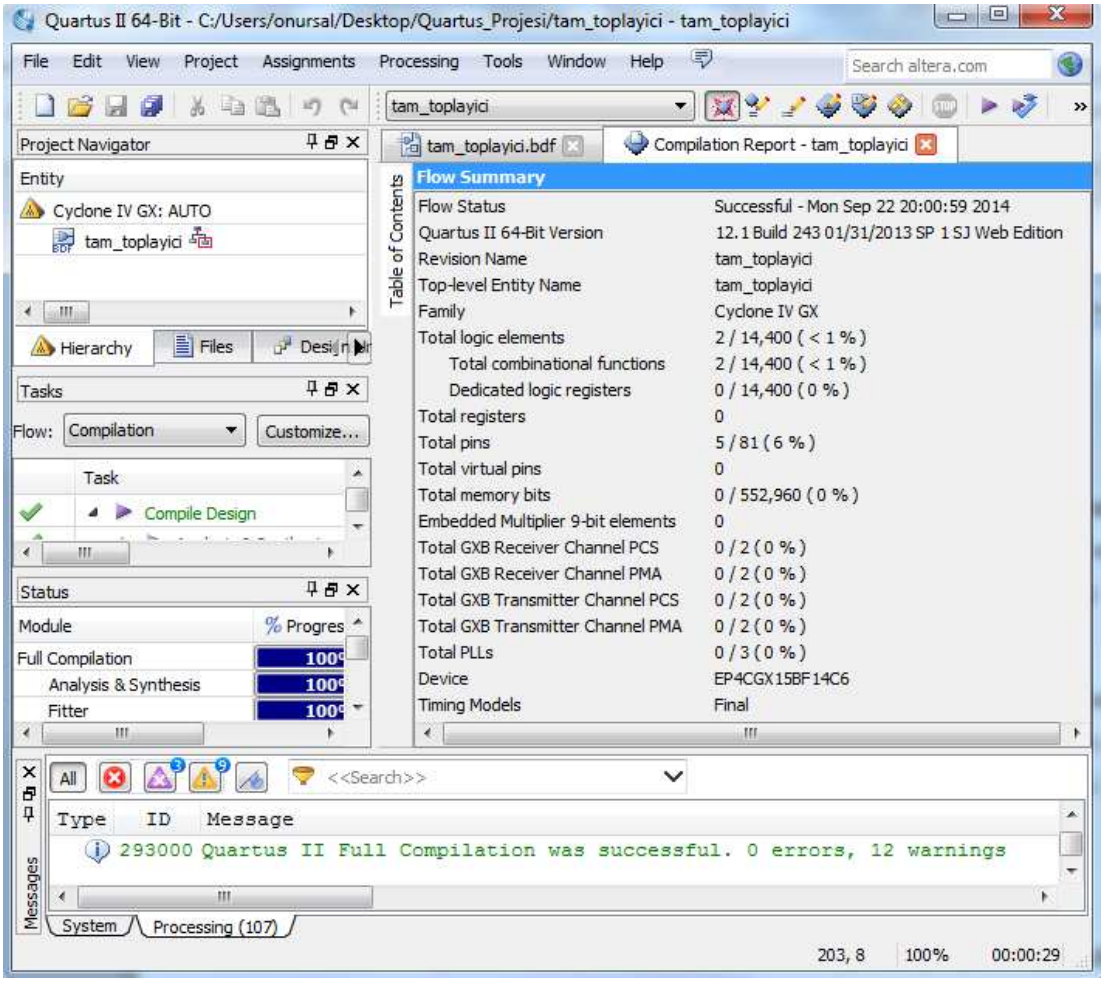


Şekil 4.16. Derleme işleminin ilerlemesi



Şekil 4.17. Derleme işleminin sonucunu gösteren pencere

Derleme işleminin sonucunu gösteren pencere OK seçeneği tıklanarak kapatılır ve deney raporunun kontrolü Şekil 4.18 ile tasarımcıya sunulur. Şekil 4.18'de verilen rapora göre tam-toplayıcı devresi 5 pin ve 2 kombinasyonel fonksiyondan meydana gelmektedir.



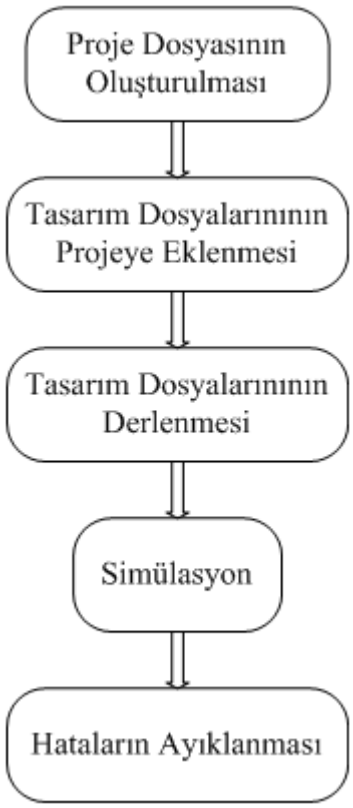
Şekil 4.18. Tam-toplayıcı derleme raporu

4.4. Hatalar

Quartus II, derleme sürecinde mesajları, Şekil 4.11'de verilen başlangıç ekranının alt kısmında görüntülenmektedir. Devre doğru tasarlanırsa, tasarımcıya "Hata bulunamadı ve derleme başarıyla sonuçlandı" mesajı verilecektir. Eğer devrenin çiziminde bir hata yapılırsa, derleme işlemi kesintiye uğrar ve mesaj penceresinde hata mesajları belirir. Büyük ve karışık bir devrede hatayı bulmak zordur. Mesaj penceresinde hatanın üzerine çift tıklanarak hatanın meydana geldiği noktaya gitmek mümkündür. Bu sayede hata düzeltilebilir.

4.5. ModelSim

ModelSim, HDL tasarımların simülasyonlarını gerçekleştirmek amacıyla Mentor Graphics tarafından geliştirilmiş bir simülasyon programıdır. ModelSim programında proje akışı Şekil 4.19'da gösterilmiştir [116].



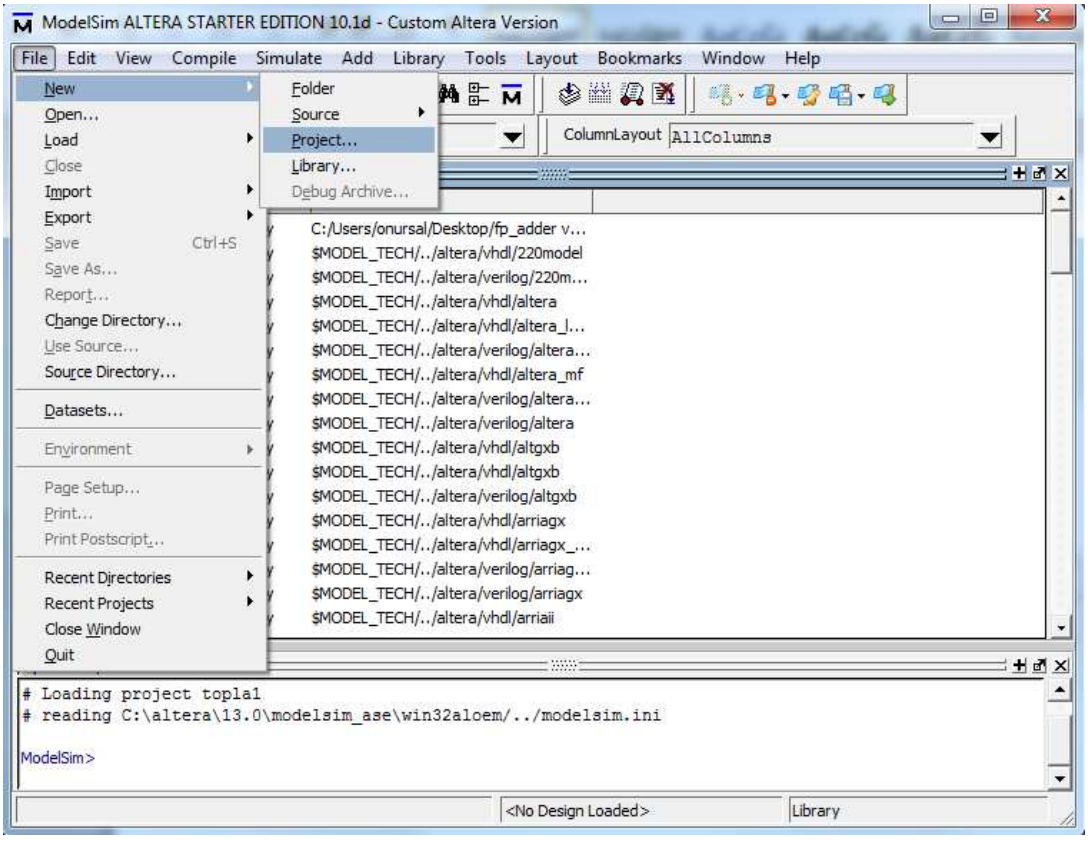
Şekil 4.19. ModelSim programında proje akışı

4.5.1. ModelSim'de proje dosyası oluşturma

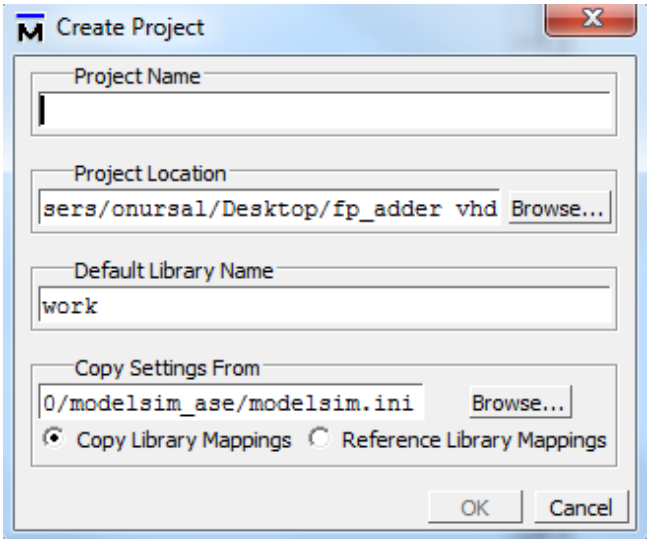
Modelsim programı çalıştırıldığında, açılan ana pencerede Şekil 4.20'de gösterildiği gibi "File > New > Project" ile yeni bir proje oluşturmak için ilk adım atılmış olur. Şekil 4.21'de gösterilen pencerede proje ismi ve proje dizini belirlenir ve "OK" butonuna basılır.

Şekil 4.22'de gösterilen "Add Items to the Project" penceresinde "Add Existing File" seçeneği seçilir ve açılan pencerede "Browse" seçeneği ile dizin açılır.

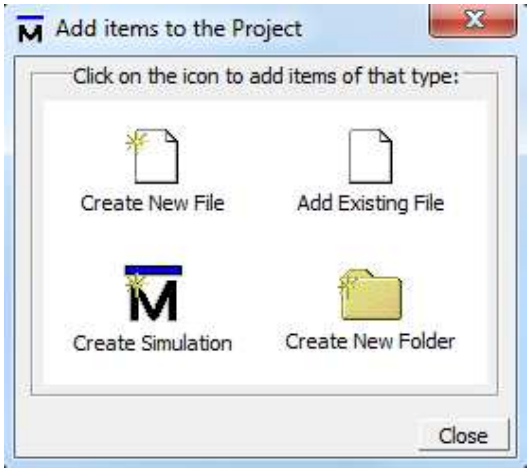
Şekil 4.23'te verilen proje dizinindeki ".vhd" uzantılı VHDL dosyaları projeye eklenir.



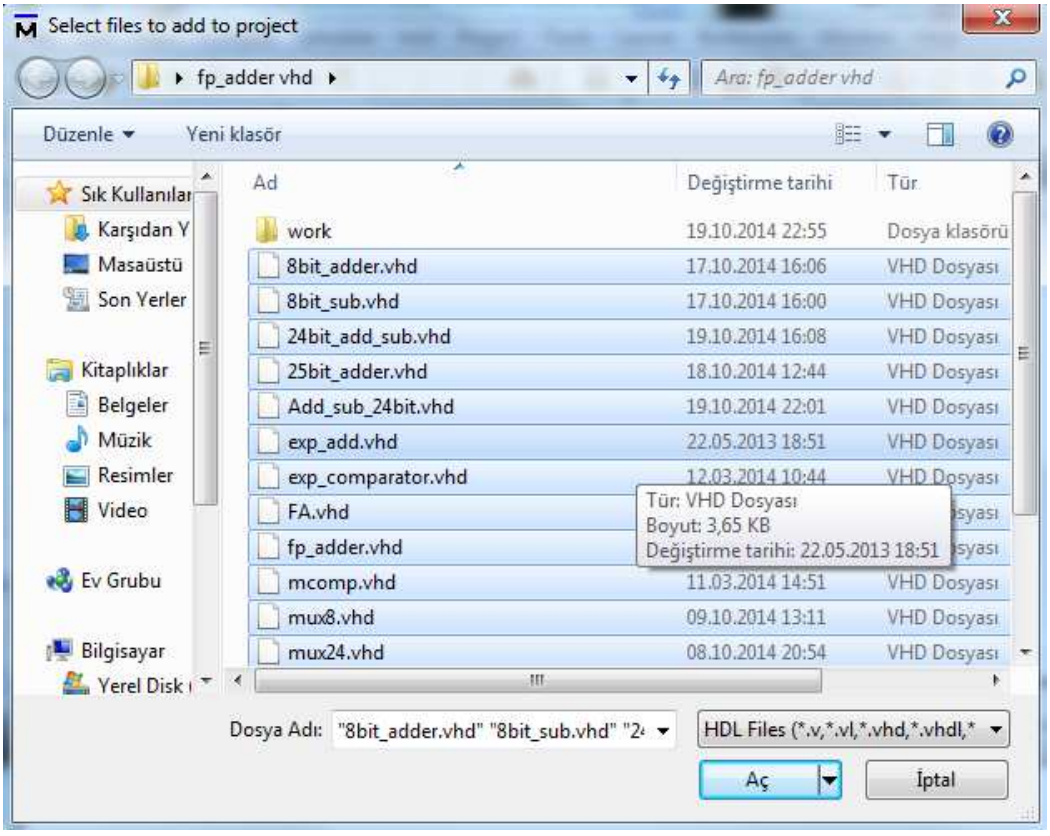
Şekil 4.20. ModelSim ana penceresi



Şekil 4.21. ModelSim'de proje adı ve dizininin belirlenmesi



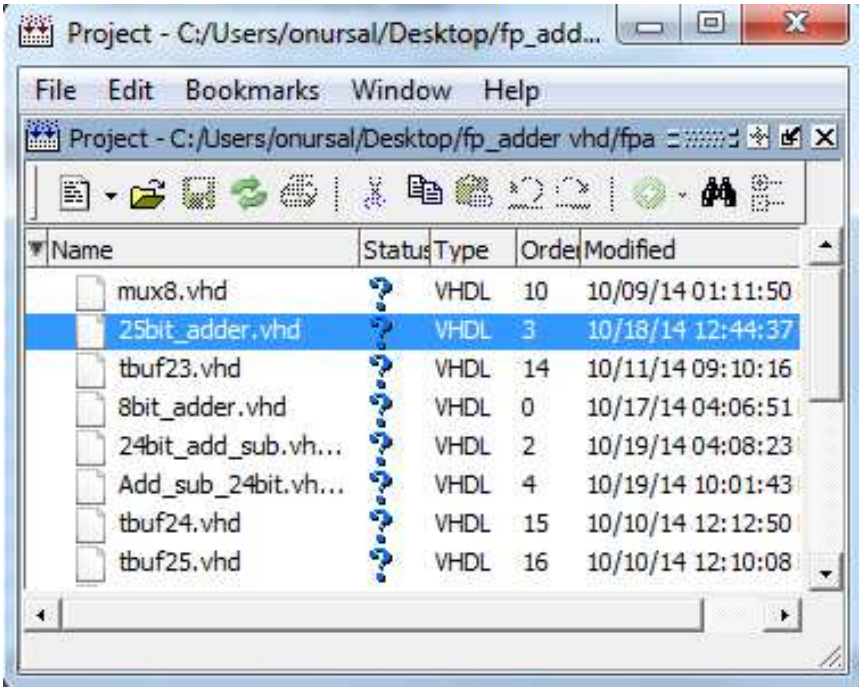
Şekil 4.22. Tasarım dosyalarının eklenmesi



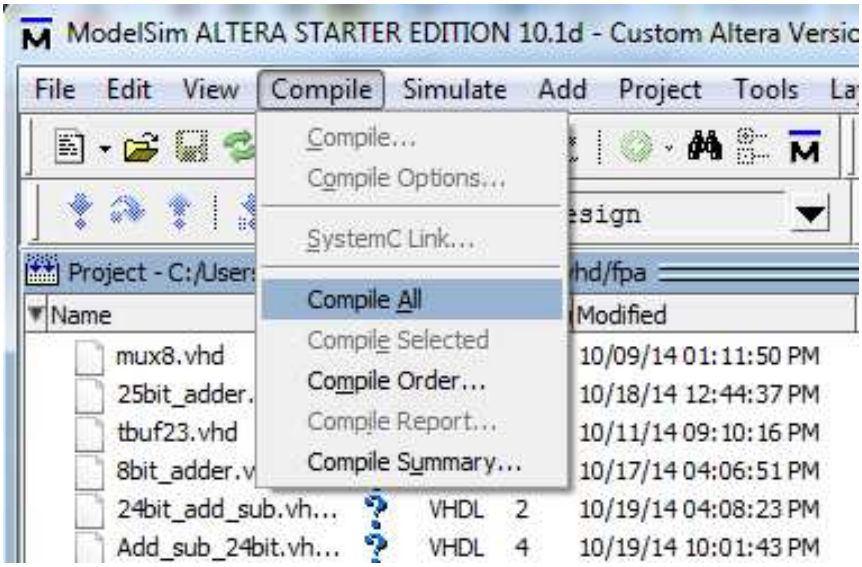
Şekil 4.23. Projeye eklenecek dosyaların seçilmesi

4.5.2. ModelSim'de derleme ve simülasyon

Şekil 4.24 ile gösterilen dosyaların "Status" kısmındaki "?" işaretleri dosyaların henüz derlenmediğini belirtmektedir. Şekil 4.25'te gösterilen menüden "Compile > Compile All" satırı seçilerek dosyaların derlenmesi sağlanır.

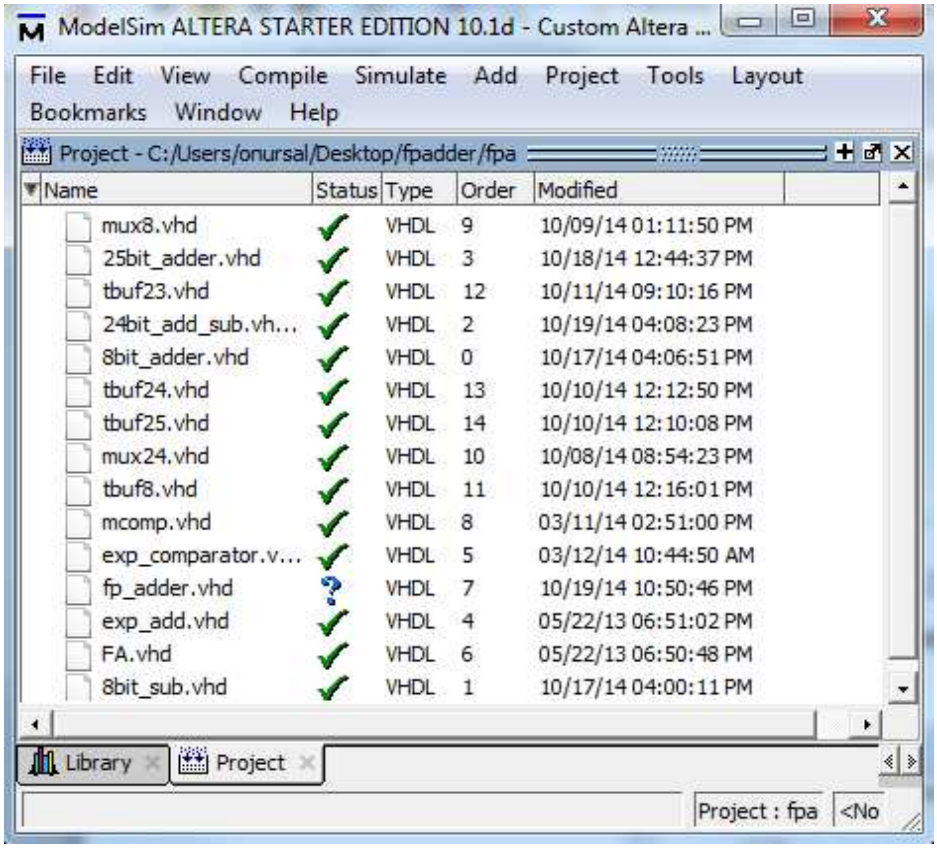


Şekil 4.24. Henüz derlenmemiş dosyalar

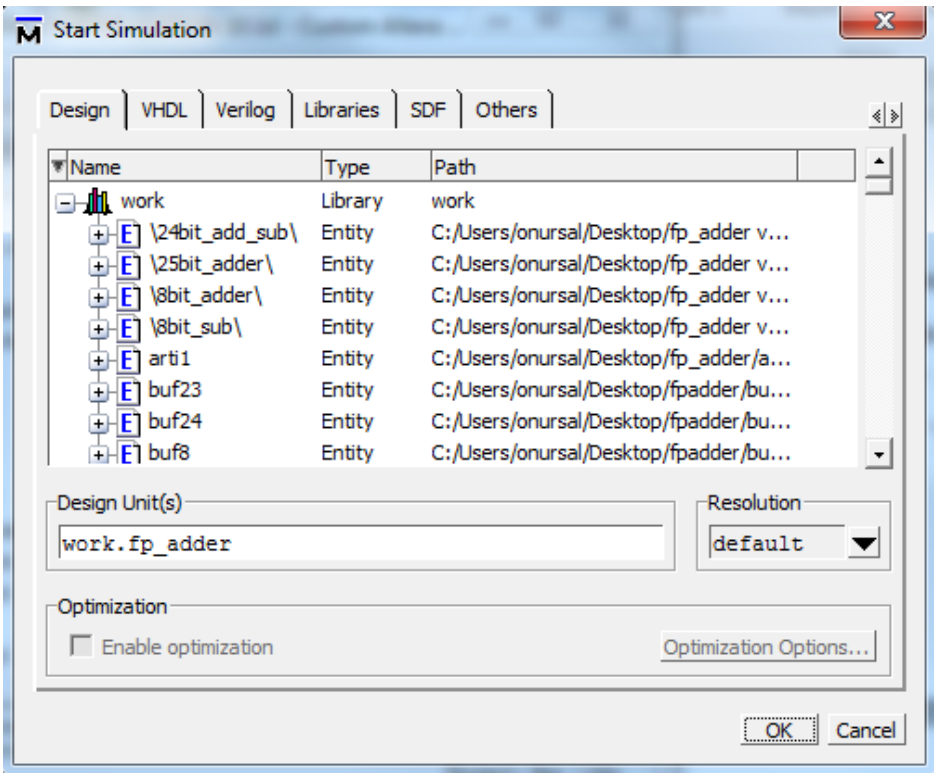


Şekil 4.25. Dosyaların derlenmesi

Dosyalar sırası ile derlenir ve derleme işleminin başarılı olması durumunda Şekil 4.26'da gösterildiği gibi "Status" alanında her bir eleman için "✓" simgesi oluşturulur. Daha sonra menüden "Simulate > Start Simulation" satırı seçilerek simülasyon işlemine geçilir. Şekil 4.27'de gösterilen pencerede "work" klasörü altında simülasyonu yapılacak dosya seçilir ve "OK" butonuna basılır.

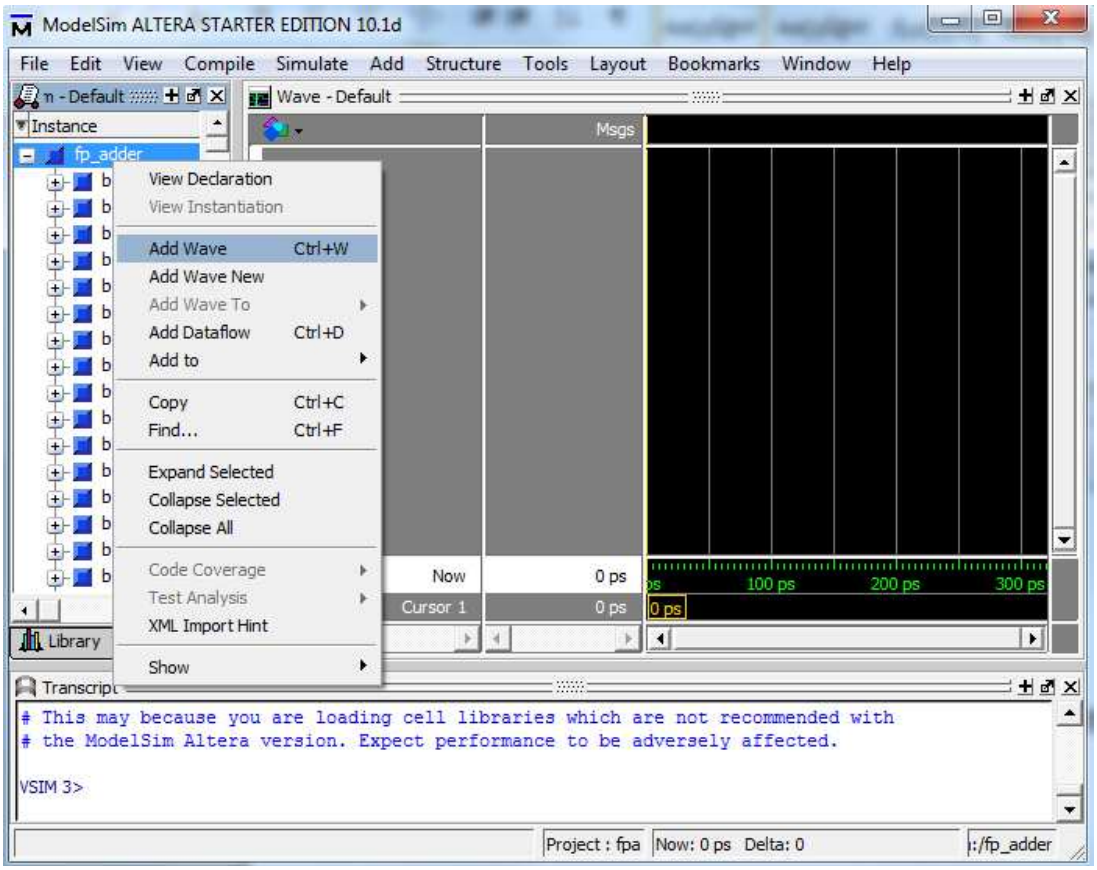


Şekil 4.26. Derlenmiş dosyalar



Şekil 4.27. Simülasyonu yapılacak dosyanın seçilmesi

Şekil 4.28'de gösterilen pencerede simülasyonu yapılacak dosyaya farenin sağ tuşu ile tıklanır ve "Add Wave" satırı seçilir. Böylece tasarımın giriş ve çıkışları simülasyona hazır hale getirilir.

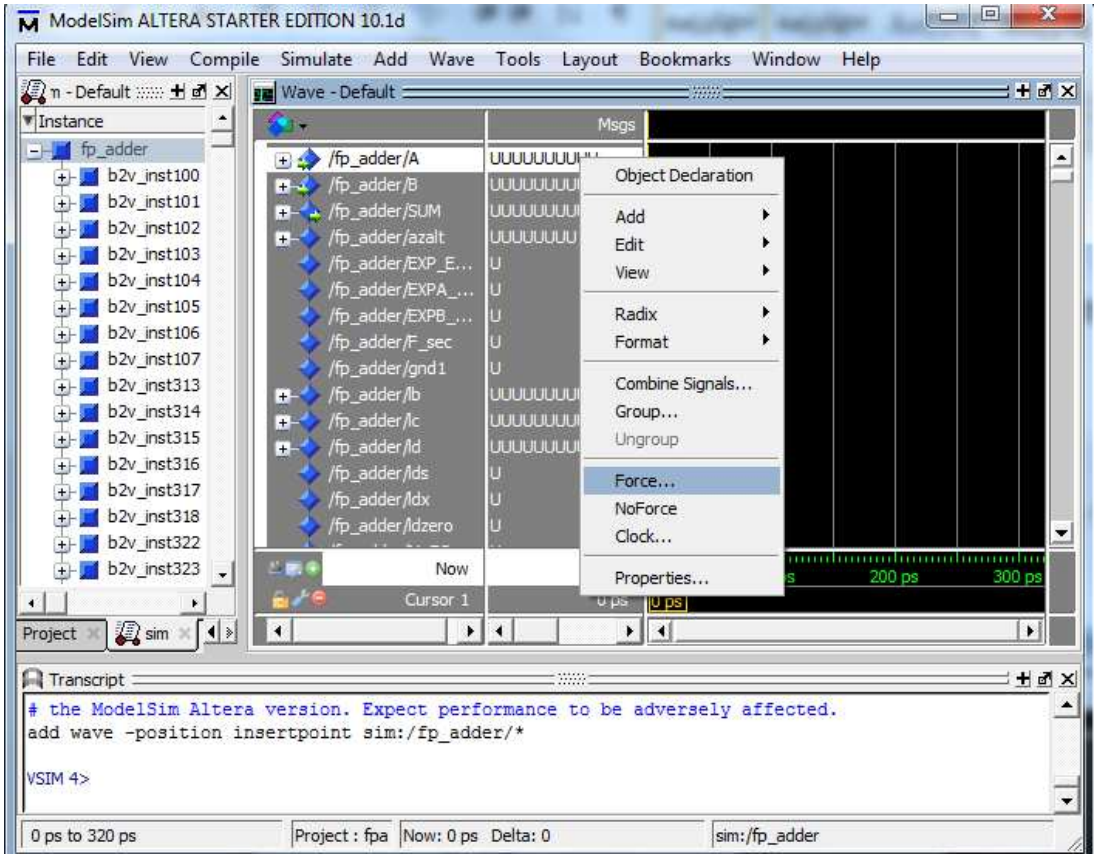


Şekil 4.28. Simülasyon penceresi

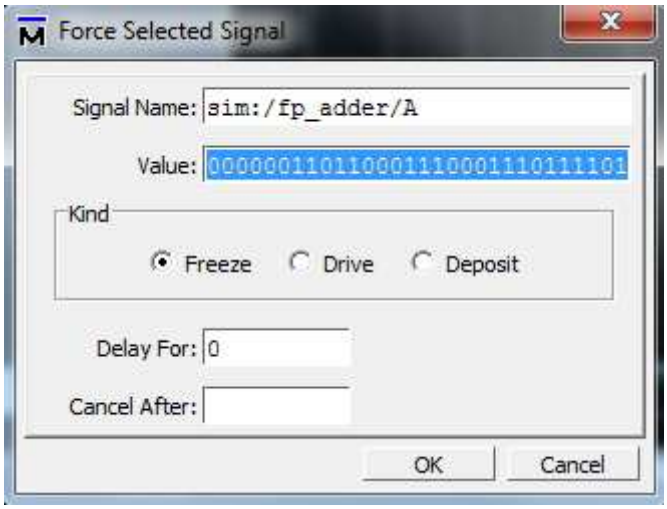
Şekil 4.29'da gösterildiği gibi tasarıma verilecek giriş değerlerine fare ile sağ tıklanarak açılan menüden "Force" satırı seçilir. Böylece giriş değerinin verileceği Şekil 4.30'da gösterilen pencere açılır.

Şekil 4.30'da gösterilen pencerede "Value" alanına giriş değeri yazılarak "OK" butonuna basılır ve giriş değeri verilmiş olur. Tasarımdaki bütün girişler için bu işlem tekrarlanır.

Giriş verme işlemi tamamlandıktan sonra tasarımın çalıştırılması ve sonucun doğruluğunun test edilmesi gerekir.

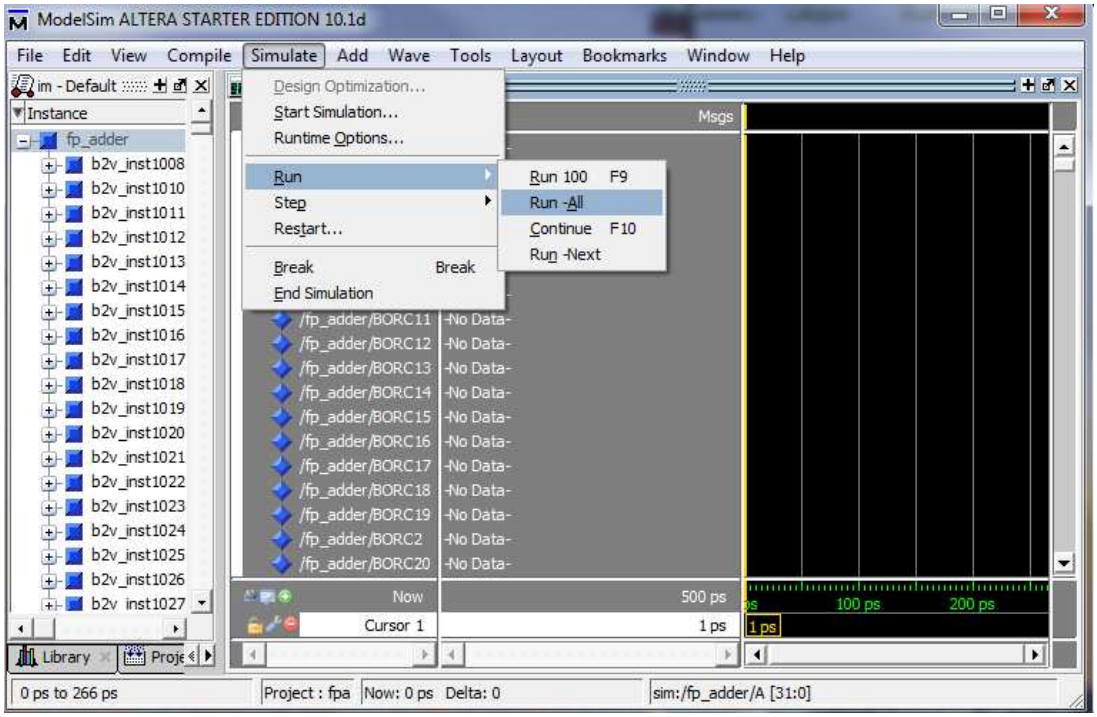


Şekil 4.29. Simülasyon giriş değerlerinin belirlenmesi

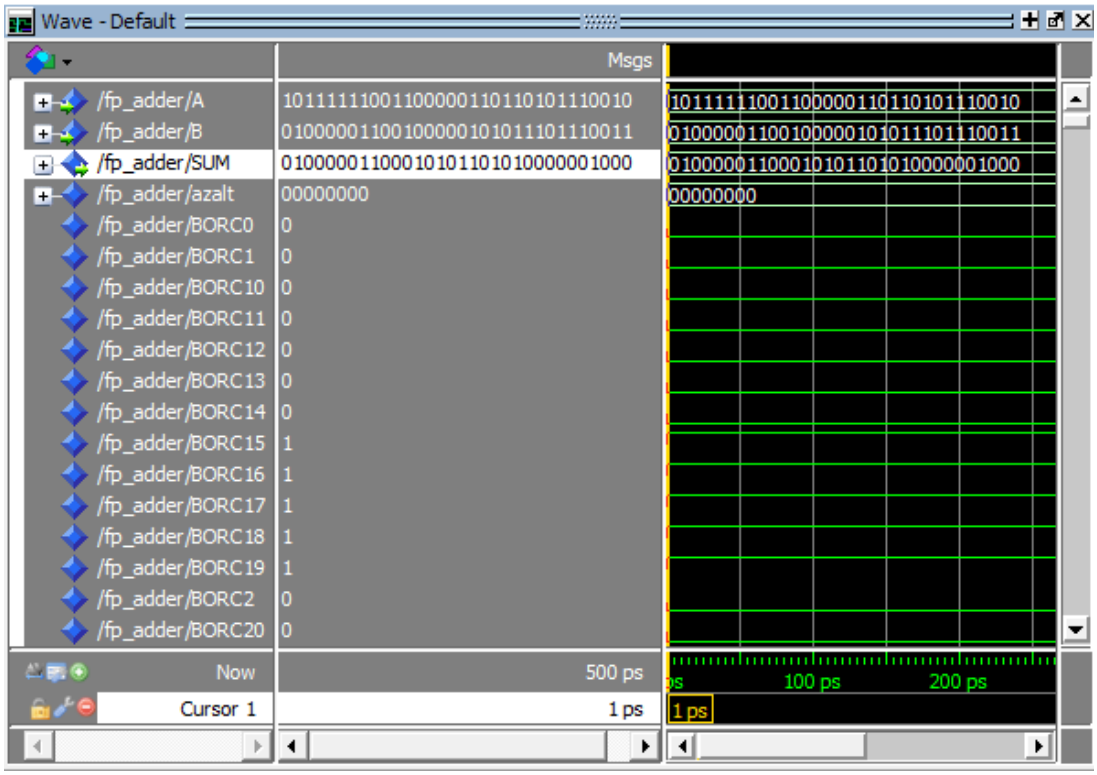


Şekil 4.30. Girişlere değer verilmesi

Şekil 4.31'de gösterildiği gibi menüden "Simulate > Run > Run All" satırı seçilerek tasarımın çıkış değerini üretmesi beklenir. Simülasyon tamamlandığında Şekil 4.32'deki pencerede tasarıma ait giriş ve çıkış değerleri gösterilir.



Şekil 4.31. Tasarımın yürütülmesi



Şekil 4.32. Tasarıma ait giriş-çıkış değerleri

BÖLÜM 5. YSA'NIN VE EĞİTİM ALGORİTMASININ FPGA ÜZERİNDE DONANIMSAL OLARAK GERÇEKLENMESİ

FPGA'ların hız, güvenlik ve paralel işlem yapabilme yeteneklerinin yanı sıra VLSI teknolojinin sahip olmadığı yeniden düzenlenebilirlik kabiliyetine sahip olması sebebiyle yapay sinir ağları ile çok uyumlu çalışmalar yapılabilmekte ve yapay sinir ağları konusuna ışık tutmaktadır. Donanım tabanlı YSA tasarımlarında FPGA kullanılması son derece mantıklı ve doğru bir karardır. Bu nedenle YSA donanımının gerçeklenmesinde FPGA kullanılmıştır. Alt birimler Altera'nın FPGA tasarım programı QUARTUS II ile tasarlanmıştır. FPGA üzerinde devre tasarlanırken, mevcut olan VHDL, Verilog, Şematik dizayn yöntemlerinden büyük ölçüde şematik dizayn kullanılmıştır. Şematik dizayn kullanılmasının sebebi tasarım aşamasında devrelerin iç yapısına mümkün olduğunca hakim olmaktır. İç yapının önemsenmediği basit ve küçük devrelerde ise VHDL donanım tanımlama dili kullanılmıştır.

Hem YSA hem de FPGA paralel bir doğaya sahiptirler ve bu sebeple işlem hızları oldukça yüksektir. Toplayıcı devrelerde kullanılan kaydırmalı yazmaçlar birden fazla saat darbesine ihtiyaç duydukları için paralel işlem yapmaya pek de uygun değillerdir. YSA, FPGA üzerinde gerçeklenirken sıralı işlem yapan birimlerin mümkün olduğunca azaltılması gerekmektedir.

Bu çalışmada, YSA donanımı FPGA üzerinde gerçeklenirken, hızdan ödün vermemek amacıyla temel elemanlar olan paralel bir çarpıcı ve saat darbesine ihtiyaç duymayan bir toplayıcı şematik olarak tasarlanmıştır. Kaydırmalı yazmaç kullanmadan tasarlanan toplayıcı devre ve paralel çarpıcı kullanılarak aktivasyon fonksiyonu elde edilmiştir. Hassas hesaplamalar için IEEE 32-bit kayan noktalı nümerik formattan faydalanılmıştır.

Bu bölümde önce nümerik tanımlamadan bahsedilecek, ardından paralel çarpıcı, toplayıcı, aktivasyon fonksiyonu bloğu ve YSA donanımının tasarımı tüm detaylarıyla anlatılacaktır.

5.1. Kayan Noktalı Nümerik Tanımlama

Yapay sinir ağı (YSA) gerçekleştirilirken gerek girişlerin ağırlıklar ile çarpılması ve toplanması gerekse aktivasyon fonksiyonunun elde edilmesi için çarpma ve toplama işlemlerinin yapılması gerekmektedir. Bu sebeple kayan noktalı çarpıcı ve toplayıcı birimlerine ihtiyaç duyulur. Nümerik hassasiyetin önem taşıdığı yazılımsal ve donanımsal uygulamalarda kayan noktalı nümerik tanımlamalar kullanılırken, maliyeti düşürmek ve hızı artırmak için sabit noktalı nümerik formattan faydalanılır [2]. Bilgisayar üreticilerinin kendilerine has kayan noktalı nümerik tanımlama standartları kullanmaları sonucunda aynı veri için farklı sonuçlar elde edilmiştir [120]. Bu da ortak bir kayan noktalı nümerik tanımlama standardı ihtiyacını doğurmuştur. Kayan noktalı sayılar 1985 yılında IEEE tarafından 754 numaralı referans ile tanımlanmış (IEEE 754 Floating Point Numerical Description) [121] ve revize edilerek IEEE 754 2008 standardı ile son halini almıştır [122]. Bu sayede kayan noktalı sayılar geniş bir kullanım alanına sahip olmuştur [123].

Kayan noktalı nümerik tanımlama IEEE 754 2008 standardı ile 16 bit hassasiyetten (half precision) 128 bit hassasiyete (quadruple precision) kadar tanımlanmış olup, 32 bit (tek hassasiyetli-single precision) ve 64 bit (çift hassasiyetli-double precision) dışındaki tanımlamalar geniş bir kullanım alanına sahip değillerdir. Bu tanımlamalarda üs değerinin negatif olmaması için üs değeri, bit sayısına göre saptırılmıştır. Üs değerini tutan bit sayısı k ise, saptırma değeri 2^{k-1}-1 olarak belirlenmiştir.

Bu tez çalışmasında IEEE 754 32 bit (tek hassasiyetli) kayan noktalı nümerik tanımlama standardı kullanılmıştır. IEEE 754 standardına göre kayan noktalı nümerik sayı formatı üç parça şeklinde tanımlanabilir. Sayıyı oluşturan 32 bitin 8 biti üs (exponent), 23 biti çarpan (matissa) olarak kullanılırken 1 bit de işaret (sign) biti kullanılmıştır. En anlamlı bit işaret biti olarak kullanılmaktadır. İşaret biti 0 ise

pozitif, 1 ise negatif değer tanımı yapmaktadır. 32 bit kayan noktalı nümerik tanımlamada üs saptırma miktarı 127 olarak belirlenmiş olup üsse 127 eklenir. 127'den büyük sayılar pozitif, 127'den küçük sayılar negatif üsleri temsil eder. 32 bit (tek hassasiyetli) kayan noktalı nümerik tanımlama için kullanılan sayı formatı Şekil 5.1'de verilmiştir.

İşaret Biti (S)	Üs (exp+127)	(1.)	Çarpan (mantissa)
1 bit	8 bit	(1.)	23 bit
31	30 23	22	0

Şekil 5.1. 32 bit kayan noktalı nümerik sayı formatı.

5.2. FPGA Üzerinde 32 Bit Kayan Noktalı Çarpıcı Tasarımı

5.2.1. Kayan noktalı sayılarda çarpma

İlk sayısal işaret işleme sistemlerinde, çarpma işlemi süresinin performans üzerinde en etkili ölçü olduğu anlaşılmıştır. CMOS teknolojisi sayesinde yüksek yoğunlukta transistör kullanımı sağlanmış, lojik kapıların anahtarlama süreleri düşürülerek daha hızlı işlemciler tasarlanmıştır [124]. 1994 yılında 32 bit kayan noktalı nümerik tanımlama için 40 ns'lik çarpma süresine sahip, 4000000 transistörden oluşan sayısal işaret işleme çipleri üretilmiştir [125].

Yapay sinir ağları FPGA üzerinde gerçeklenirken, paralel yapının sağladığı hızlı işlem yapabilme avantajını koruyabilmek için yüksek hızlı çarpma devreleri kullanmak gerekir. Bu ve benzeri amaçlar doğrultusunda hızlı çarpma devresi tasarımı üzerine birçok araştırma yapılmıştır. Bu çalışmalardan birkaçı şu şekilde sıralanabilir [124]:

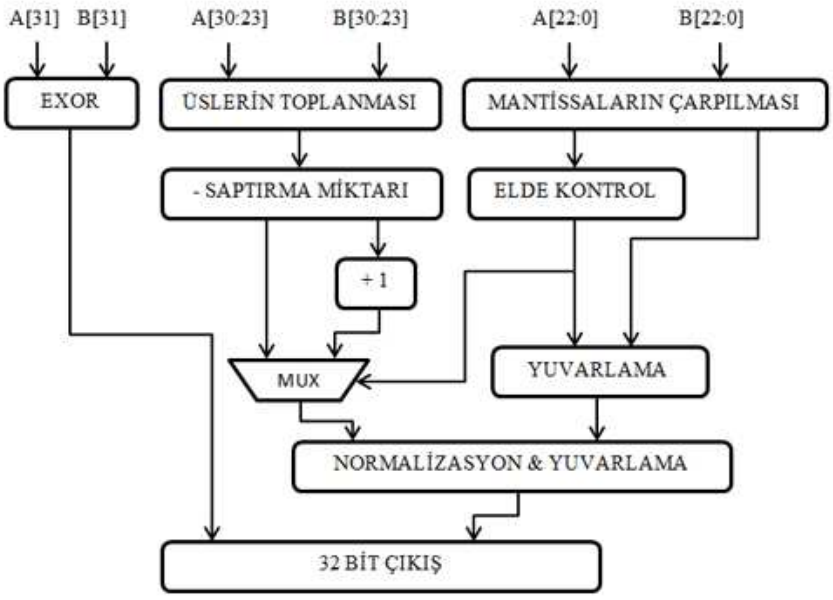
1. Fazlalıklı ikili toplama ağacı çarpma algoritması [126].
2. Geliştirilmiş Booth algoritması kullanan paralel çarpma devresi [127].
3. Booth algoritması tabanlı hızlı çarpıcı [128].
4. Paralel tam çarpıcı [129].
5. Fazlalıklı ikili gösterilim kullanan çarpıcı [130].
6. Fazlalıklı ikili gösterilim ve Booth algoritması kullanan çarpıcı [131].

Bu çalışmalara ek olarak Benrekia ve Attari VHDL kullanarak kayan noktalı çarpıcı tasarlamışlardır [132]. Jain ve arkadaşları 32 bit kayan noktalı nümerik tanımlama için hızlı bir çarpıcı geliştirmişlerdir [133]. Rajesh ve arkadaşları FPGA üzerinde kayan noktalı çarpıcı tasarlamışlardır [134].

Kayan noktalı sayılarda çarpma işlemi karmaşık ve zor bir işlemdir. Bu işlem yapılırken;

1. İki sayının üs değerleri toplanır. (Ancak bu üs değerleri saptırılmış olduğundan, sonucun üs değeri iki kez saptırılmış olarak elde edilir. Bu nedenle sonucun üs değerinden saptırma değeri çıkarılır)
2. İki sayının mantissa kısımları çarpılır.
3. İşaret bitlerine bakılır. İşaret bitleri aynı ise sonuç pozitif, değilse negatiftir.
4. Sonuç normalize edilir.

32 bit kayan noktalı çarpıcı birimi üç ana kısımdan oluşmaktadır. Çarpma işleminin akış şeması Şekil 5.2'de gösterilmiştir.



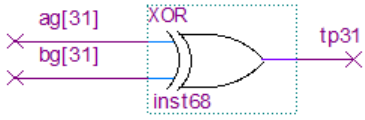
Şekil 5.2. Çarpma işlemi akış şeması

Birinci kısımda çarpılacak sayılara ait işaret bitleri XOR işlemine tabi tutularak işaret biti belirlenir. İkinci kısımda üsler toplanır ve saptırma değeri olan 127 çıkarılarak

toplam elde edilir. Daha sonra mantissa çarpımından gelen “elde biti” kontrol edilir ve toplam veya bir fazlası çoğullayıcı (mux, multiplexer) yardımıyla seçilerek sonucun üs değeri bulunur. Üçüncü kısımda ise sayıların mantissaları çarpılır. 23 bitlik iki mantissanın çarpımı sonucunda 46 bitlik bir sayı elde edilir ve bu sayı 23 bite yuvarlanır. Sözü edilen kısımlar Quartus II programı ile en küçük parçasından bütüne doğru donanımsal olarak tasarlanmıştır.

5.2.2. İşaret bitinin elde edilmesi

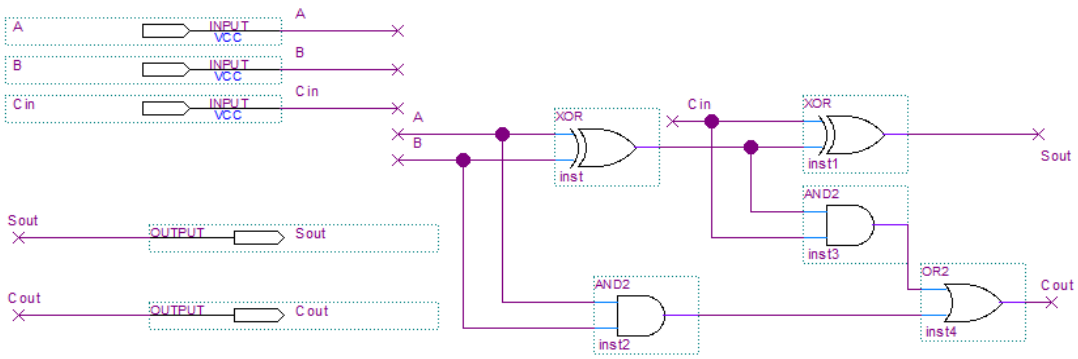
Kayan noktalı sayı gösteriminde en anlamlı bit işaret biti olarak kullanılmaktadır. İşaret biti “0” ise pozitif, “1” ise negatif değer tanımı yapmaktadır. İşaret bitinin elde edilmesi çarpıcı biriminin en kolay kısmıdır. Çarpılacak sayıların işaret bitleri Şekil 5.3’te gösterildiği gibi XOR işlemine tabi tutularak çıkışa aktarılır.



Şekil 5.3. İşaret bitinin elde edilmesi

5.2.3. Üslerin toplanması

IEEE 754 32 bit (tek hassasiyetli) kayan noktalı nümerik tanımlama standardına göre sayıyı oluşturan 32 bitin 8 biti üs (exponent) olarak kullanılır. Çarpılacak iki sayının üs değerleri toplanır. Toplama işleminin temel elemanı tam-toplayıcı (full-adder) olup Şekil 5.4’te gösterilmiştir.



Şekil 5.4. Tam-toplayıcı devresi

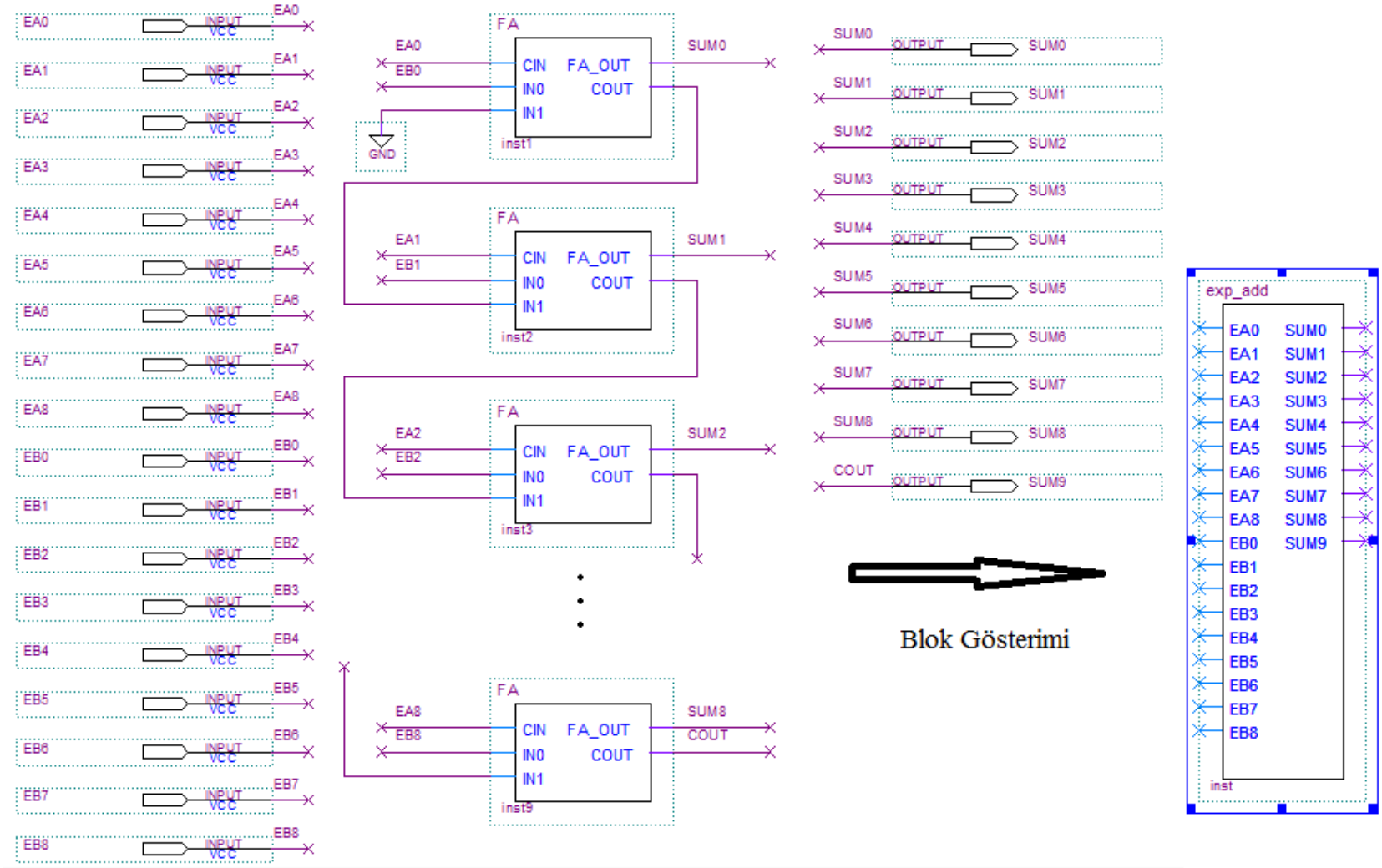
Tam-toplayıcı devrelerinden oluşan ve toplama işlemi için tasarlanan toplayıcı devre ve sembol dosyası Şekil 5.5'te gösterilmiştir.

Ancak girişlerin üs değerleri saptırılmış olduğundan, işlem sonucunda üs değeri iki kez saptırılmış olarak elde edilir. Bu nedenle sonucun üs değerinden 127 çıkarılır. Çıkarma işlemi için 2'ye tümleyen yöntemi kullanılmış, üslerin toplamına 127'nin tümleyeni eklenerek çıkarma işlemi gerçekleştirilmiştir. Sonuç olarak 127'den büyük sayılar pozitif, 127'den küçük sayılar negatif üsleri temsil eder. Mantissa çarpımından gelen "elde bitinin" değerine bağlı olarak üslerin toplamı ya da bir fazlası seçilerek sonucun üs değeri elde edilir. Bu noktada seçme işlemi üç-durumlu tampon elemanları kullanılarak yapılır. Üs toplama işlemi Şekil 5.6'da gösterilmiştir.

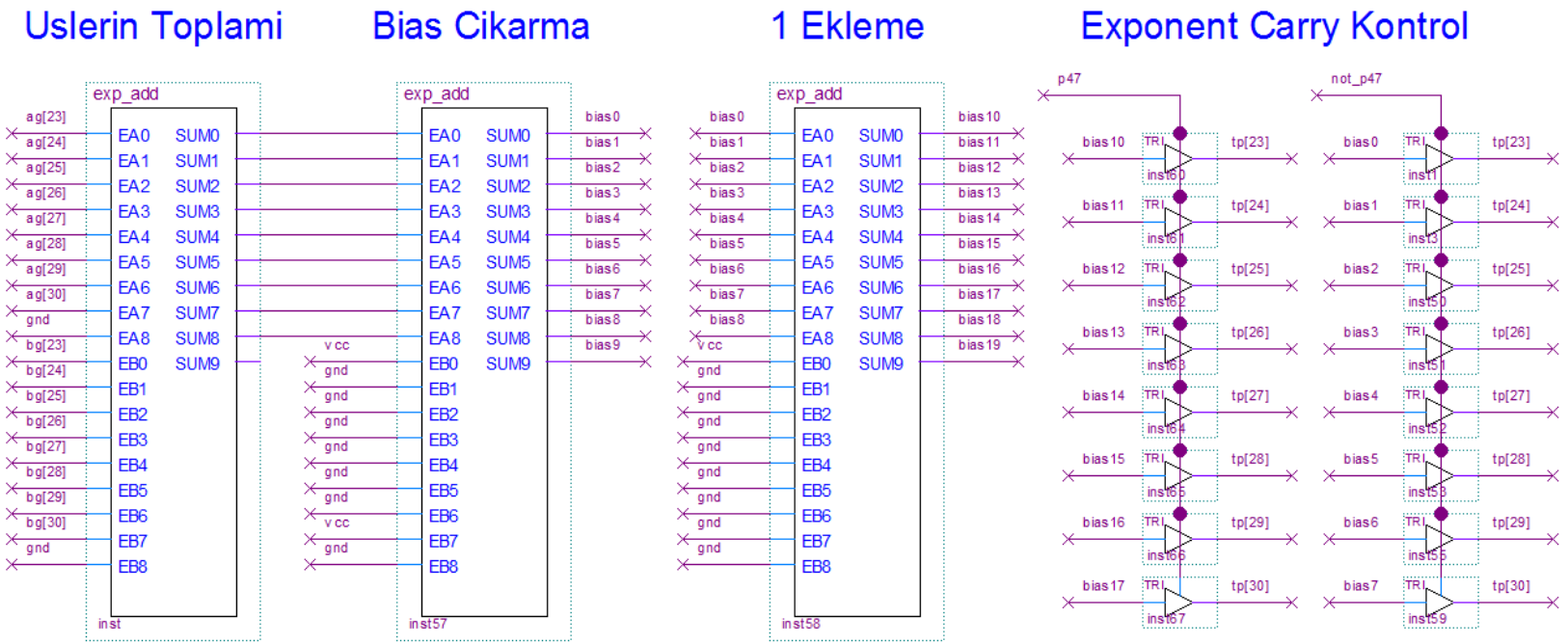
5.2.4. Mantissaların çarpımı

Mantissa bitlerinin çarpımının bulunması, çarpma bloğunun en zor işlemidir. Çarpma işleminde kısmi çarpımların elde edilişi ve toplama işlemi birleştirilerek paralel çarpıcı gerçekleştirilebilir [135]. 24 bitlik bir paralel çarpıcının blok devresi Şekil 5.7'de gösterilmiştir.

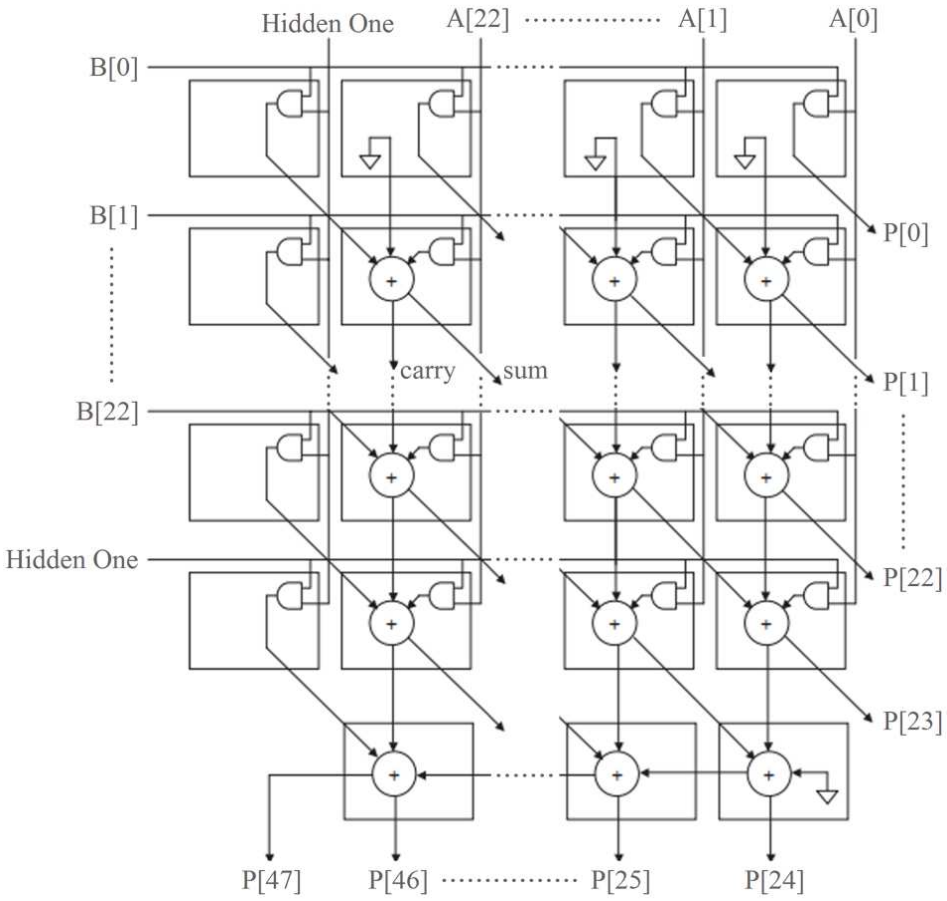
Şekil 5.7 ile gösterilen yapıdaki her bir blok kısmi bir çarpım oluşturur ve bu çarpımlar bir önceki bloktan gelen kısmi çarpım ile toplanır. Son satır ise tam-toplayıcı devrelerinden oluşmaktadır ve eldenin yatay yayılımı söz konusudur. Blokların içyapısı ve sembol dosyası Şekil 5.8'de gösterilmiştir.



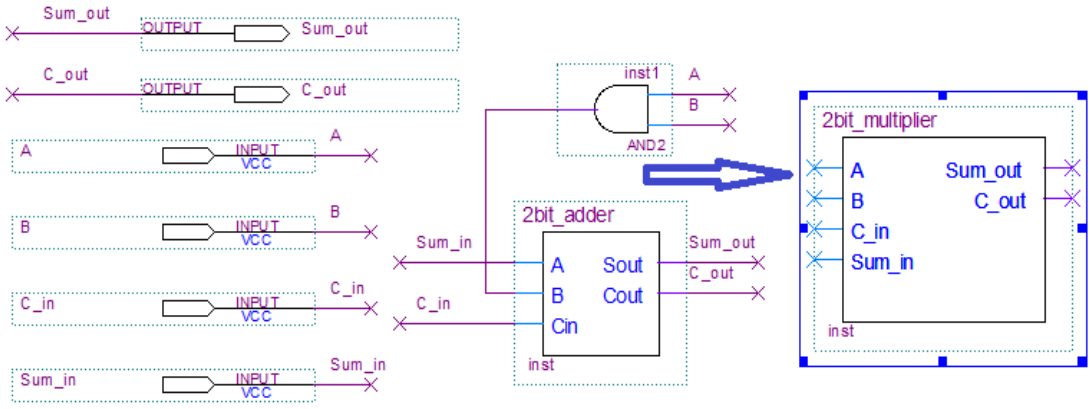
Şekil 5.5 Toplayıcı devre ve sembol dosyası



Şekil 5.6. Üs toplama işlemi

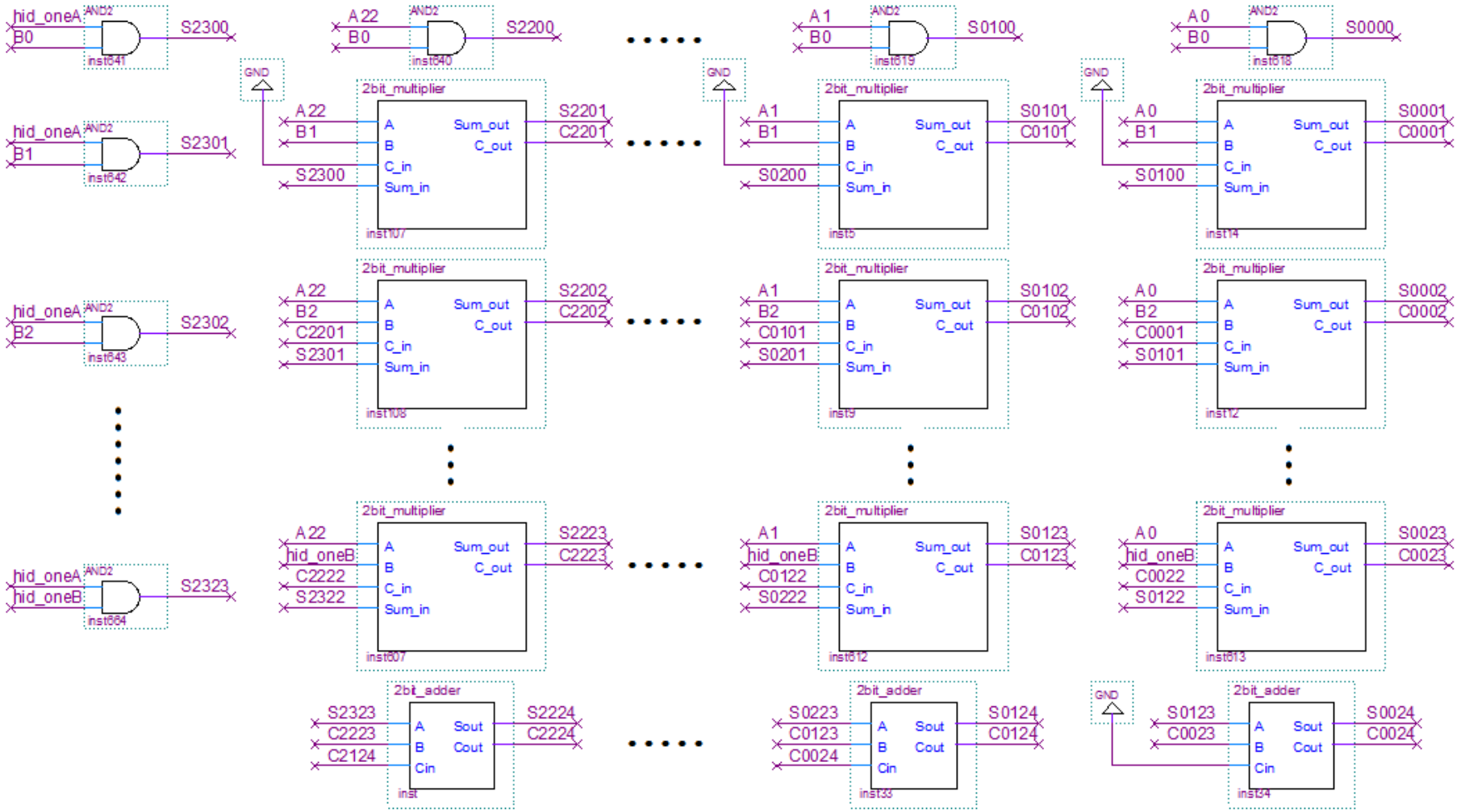


Şekil 5.7. 24 bitlik paralel çarpıcı blok gösterimi



Şekil 5.8. Çarpıcı bloğun içyapısı ve sembol dosyası

Çarpma işlemini gerçekleştirmek için, 23 bitlik mantissanın yanı sıra virgülden önceki bit de dikkate alınmalı ve 24 bitlik çarpıcı tasarlanmalıdır. Bu amaçla 24 bitlik paralel çarpıcı devresi gerçekleştirilmiştir. 25 satır, 24 sütun ve toplamda 599 bloktan ve çok sayıda ara bağlantıdan oluşan yapı Şekil 5.9'da gösterilmiştir.



Şekil 5.9. 24 bit mantissa çarpıcı

Bu blokların birleştirilmesi sonucunda, Şekil 5.2 ile verilen akış şemasına göre işlem yapan 32 bit kayan noktalı çarpıcı birimi donanımsal olarak tasarlanmıştır. Tasarlanan yapı Şekil 5.10'da gösterilmiştir.

32 bit kayan noktalı çarpıcıda, işaret bitleri XOR işlemine tabi tutulmuş ve doğrudan çıkışa aktarılmıştır. 23 bitlik mantissaların başına '1' eklenerek en anlamlı biti '1' olan 24 bitlik iki sayı elde edilmiş ve bunlar 24 bit paralel çarpıcıya uygulanmıştır. 24 bitlik iki sayının çarpımı olarak 48 bitlik bir sayı elde edilmiş ve sonuç yuvarlama işlemiyle 23 bite indirgenmiştir. Bu amaçla 48. bit olan elde biti (carry) kontrol edilir. Eğer elde biti '1' ise sonucun 47-25 arası bitleri çıkışa aktarılır. Eğer elde biti '0' ise sonucun 46-24 arası bitleri çıkışa aktarılır ve mantissa elde edilmiş olur. Çıkışa hangi bitlerin aktarılacağı üç-durumlu tampon elemanları kullanılarak yapılır. Elde biti aynı zamanda üs değerini de belirleyicidir ve elde bitinin '1' olması durumunda üslerin toplam değeri bir artırılır.

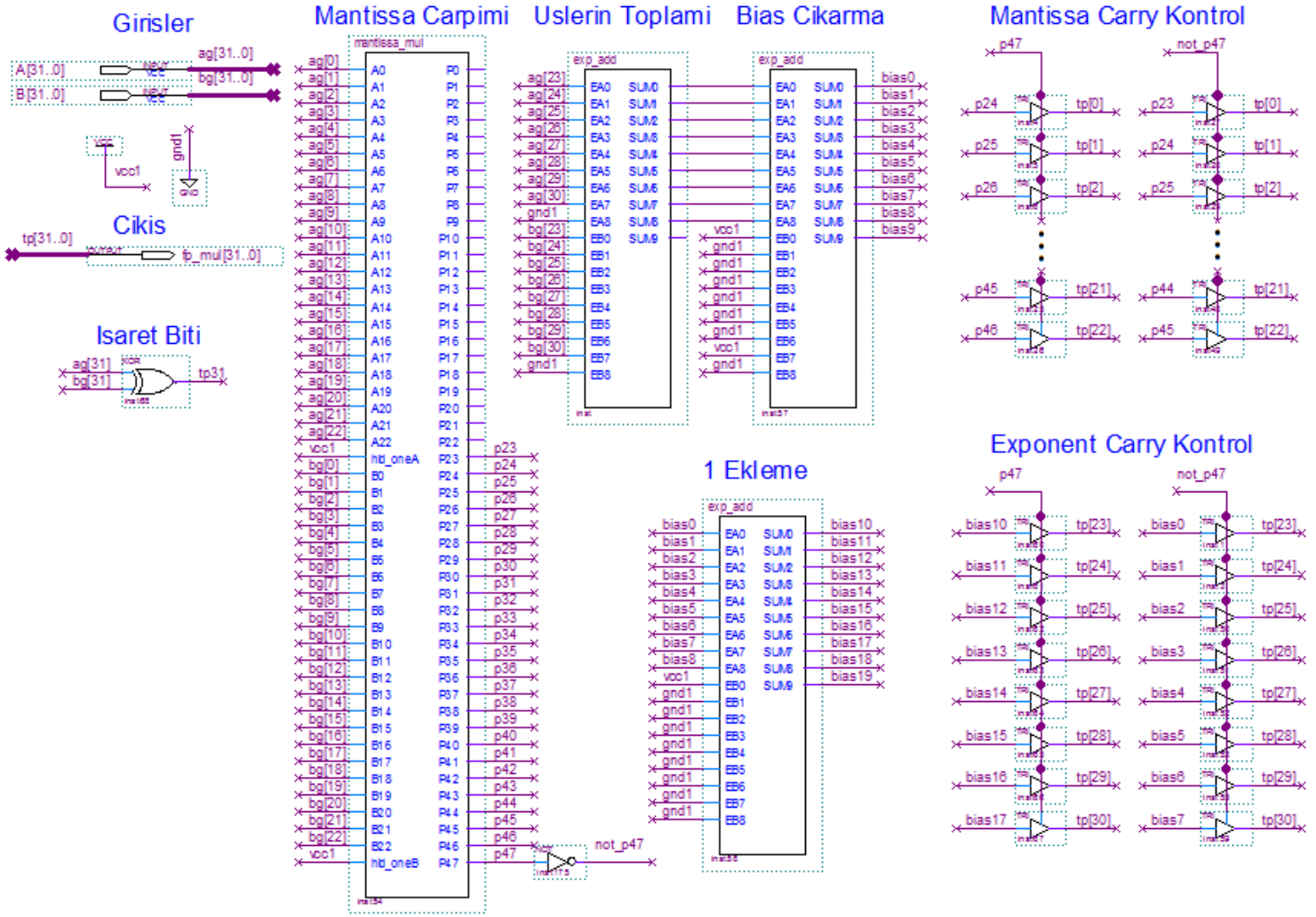
5.2.5. 32 bit kayan noktalı çarpıcının test edilmesi

Yapay sinir ağlarında kullanılmak üzere IEEE tarafından 754 numaralı referans ile tanımlanmış 32 bit kayan noktalı sayılar için çarpıcı birim tasarlanmıştır. Tasarım işlemi Quartus II programı vasıtasıyla tamamen donanımsal olarak gerçekleştirilmiştir. Çarpıcı, Tablo 5.1, Tablo 5.2 ve Tablo 5.3'te verilen rastgele sayılar için ModelSim kullanılarak test edilmiştir.

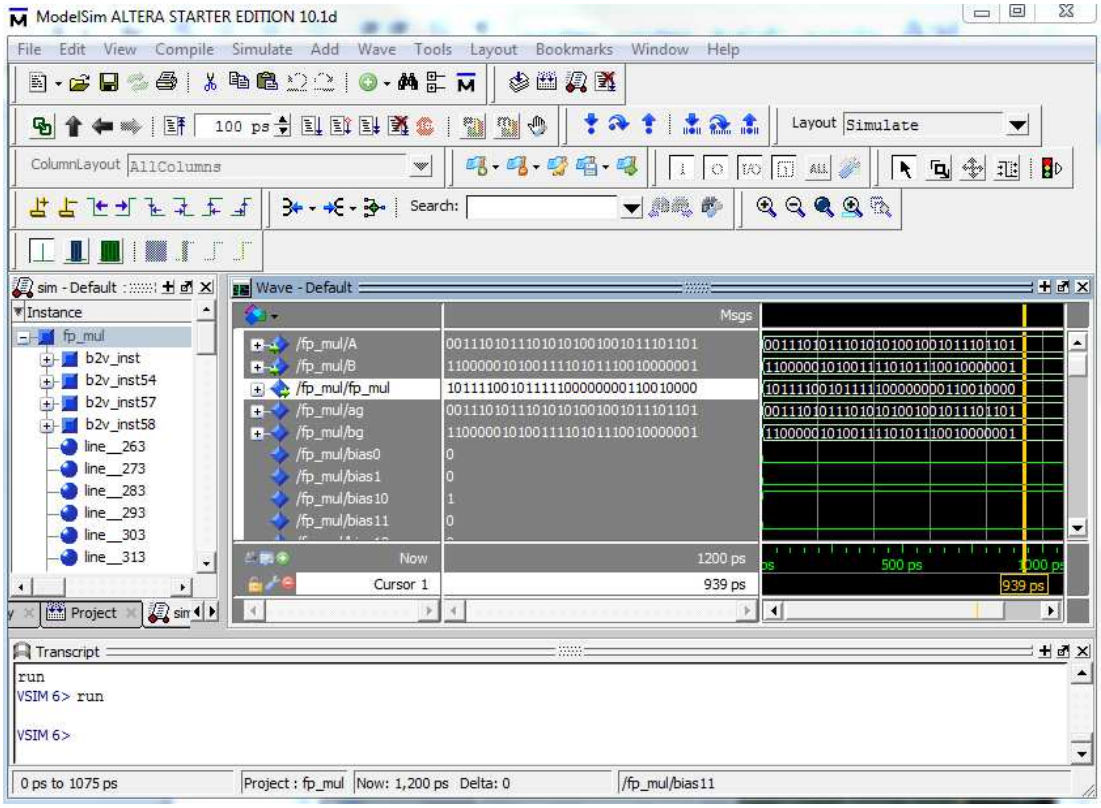
Tablo 5.1. Çarpıcının rastgele sayılar için test edilmesi 1

	10'luk Taban	**32 Bit Kayan Noktalı Gösterim**
Çarpan	0.001789657	00111010111010101001001011101101
Çarpılan	-12.96008351	11000001010011110101110010000001
Çarpıcı Çıkışı	-0.023194104	10111100101111100000000110010000
Gerçek Sonuç	-0.023194104	10111100101111100000000110010000

Tablo 5.1'de verilen sayılar için çarpıcı doğru sonuç vermiştir. ModelSim ile elde edilen simülasyon sonucu Şekil 5.11'de gösterilmektedir.



Şekil 5.10. 32 bit kayan noktalı çarpıcı birimi

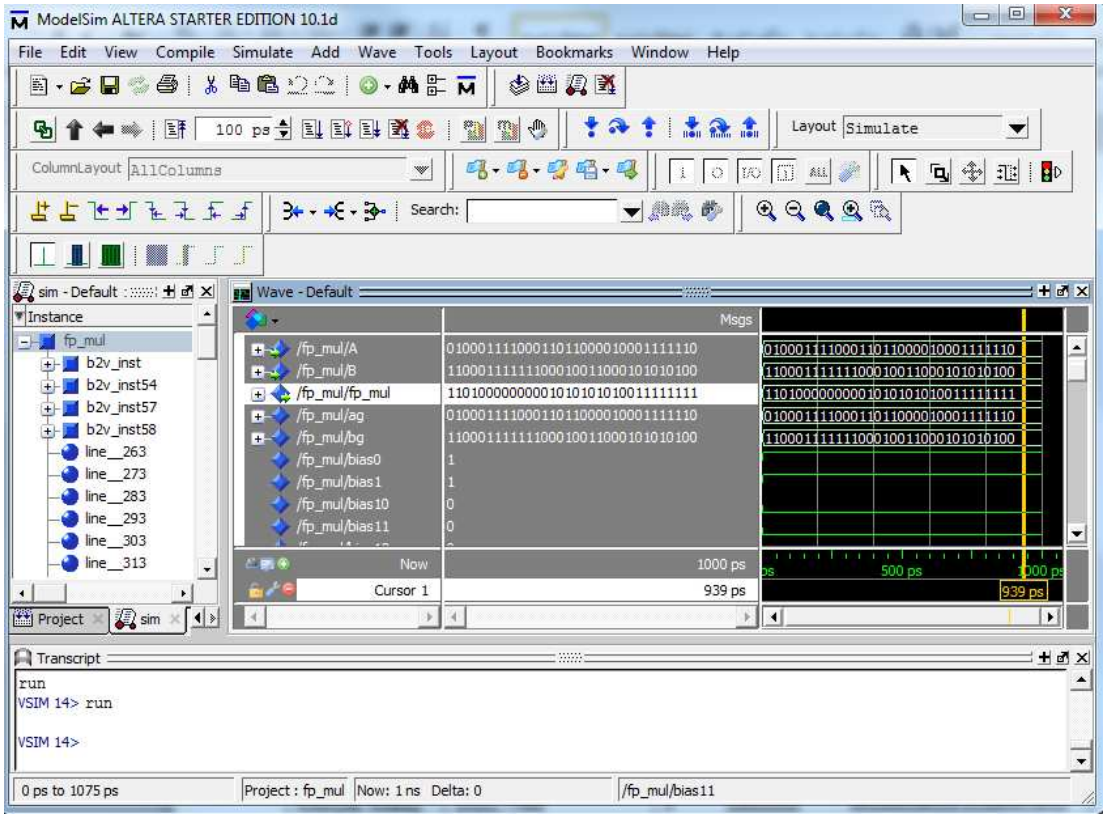


Şekil 5.11. Tablo 5.1'de verilen sayılar için simülasyon sonucu

Tablo 5.2. Çarpıcının rastgele sayılar için test edilmesi 2

	10'luk Taban	**32 Bit Kayan Noktalı Gösterim**
Çarpan	72456.98400	01000111100011011000010001111110
Çarpılan	-123490.6600	11000111111100010011000101010100
Çarpıcı Çıkışı	-8.9477601E9	11010000000001010101010011111111
Gerçek Sonuç	-8.9477607E9	11010000000001010101010100000000

Tablo 5.2'de verilen sayılar için çarpıcı doğru sonuç vermiştir. Ancak yuvarlama nedeniyle sonuçta çok küçük bir fark söz konusudur. ModelSim ile elde edilen simülasyon sonucu Şekil 5.12'de gösterilmektedir.



Şekil 5.12. Tablo 5.2'de verilen sayılar için simülasyon sonucu

Tablo 5.3. Çarpıcının rastgele sayılar için test edilmesi 3

	10'luk Taban	**32 Bit Kayan Noktalı Gösterim**
Çarpan	-0.123456790	10111101111111001101011011101010
Çarpılan	-18824.12500	11000110100100110001000001000000
Çarpıcı Çıkışı	2323.966000	01000101000100010011111101110101
Gerçek Sonuç	2323.966000	01000101000100010011111101110101

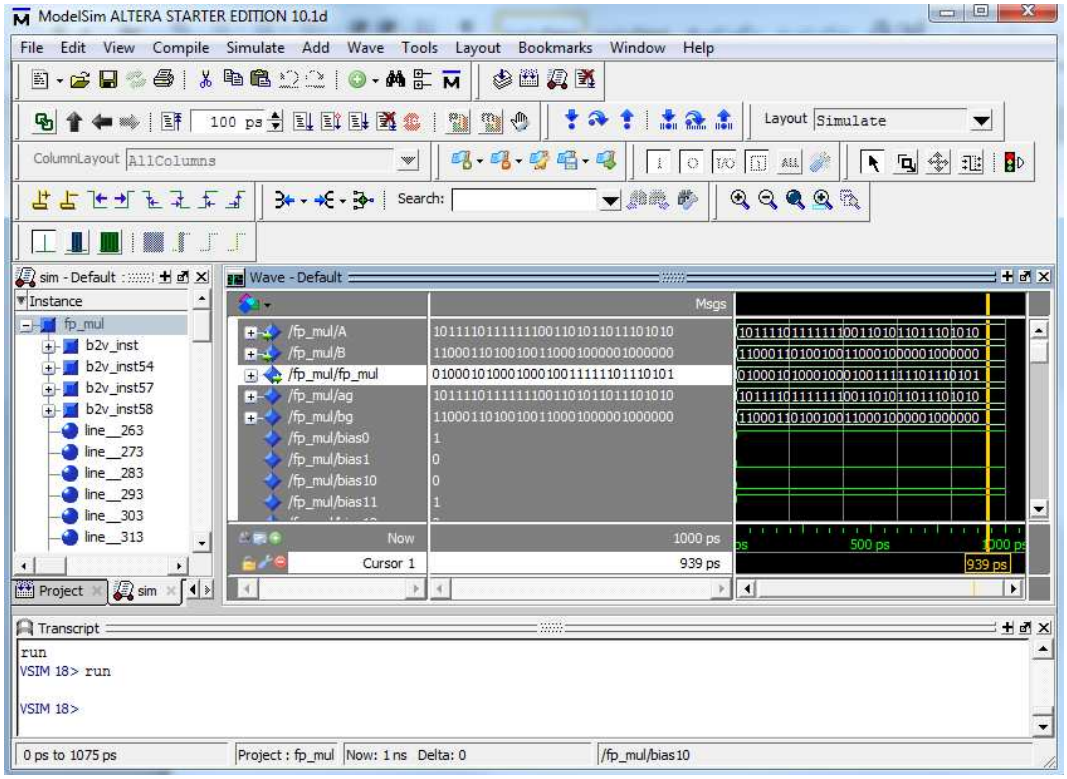
Tablo 5.3'te verilen sayılar için çarpıcı doğru sonuç vermiştir. ModelSim ile elde edilen simülasyon sonucu Şekil 5.13'te gösterilmektedir.

5.3. FPGA Üzerinde 32 Bit Kayan Noktalı Toplayıcı Tasarımı

5.3.1. Kayan noktalı sayılarda toplama

Yapay sinir ağları FPGA üzerinde gerçeklenirken, paralel yapının sağladığı hızlı işlem yapabilme avantajını koruyabilmek için yüksek hızlı toplama devreleri

kullanmak gerekmektedir. Bu amaç doğrultusunda hızlı toplama devresi tasarımı gerçekleştirilmiştir [136].



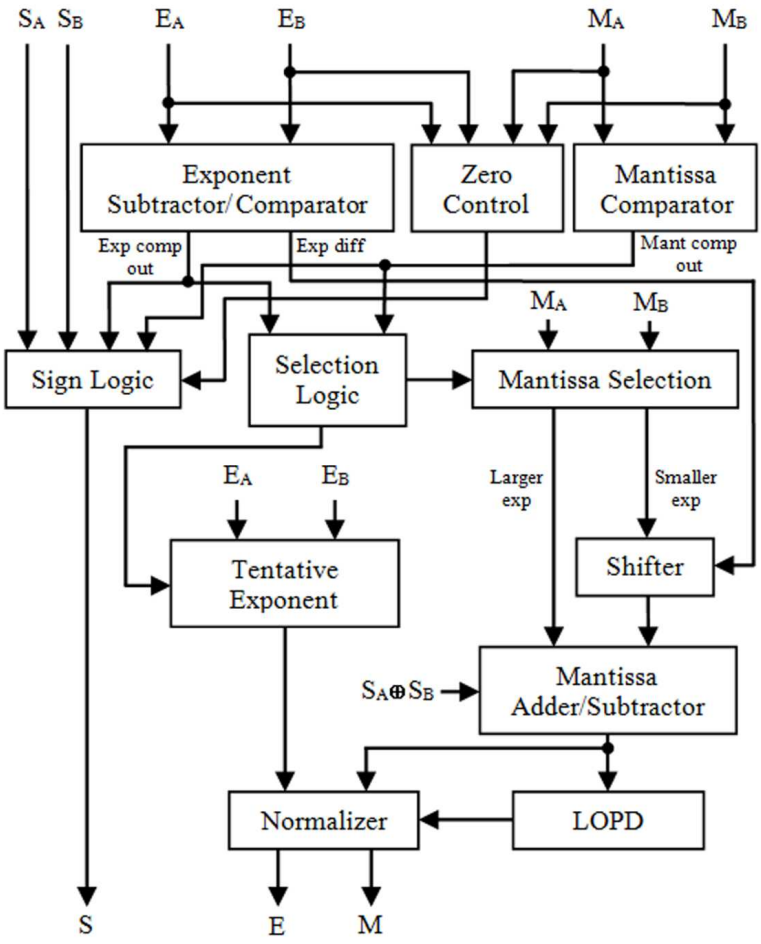
Şekil 5.13. Tablo 5.3'te verilen sayılar için simülasyon sonucu

Kayan noktalı sayılarda toplama işlemi çarpma işlemine göre çok daha karmaşık ve zor bir işlemdir [137-141]. Bu işlem yapılırken;

1. İki sayının işaret bitleri kontrol edilir. İşaret bitleri aynı ise toplama, farklı ise çıkarma işlemi yapılır.
2. Çıkarma işlemi için borç çıkışı ve karşılaştırıcı (comparator) çıkışına bağlı olarak işaret biti belirlenir.
3. İki sayının üs değerleri karşılaştırılır. Büyük üs değeri sonucun üs değeri olarak seçilir.
4. Büyük üs değerinden küçük üs değeri çıkarılarak fark bulunur.
5. Küçük sayının mantissası üsler arasındaki fark kadar sağa kaydırılır.
6. Büyük sayının mantissası ile kaydırılmış mantissa toplanarak ya da çıkarılarak sonucun mantissa kısmı elde edilir.

7. Toplam sonucunda taşma olup olmadığı, çıkarma sonucunda borç olup olmadığı kontrol edilir.

32 bit kayan noktalı toplayıcı biriminin genel yapısında kaydırmalı yazmaçlar bulunmaktadır. Kaydırmalı yazmaçlar saat darbesi ile çalışan elemanlardır ve paralel yapıya çok da uygun değillerdir. Bu nedenle toplayıcı tasarımında kaydırmalı yazmaç kullanılmamıştır. Kaydırmalı yazmacın kullanılmaması devreyi daha karmaşık bir hale getirirken hızını artırmıştır. Tasarımı gerçekleştirilen 32 bit kayan noktalı toplayıcı biriminin genel yapısı Şekil 5.14'te gösterilmiştir.



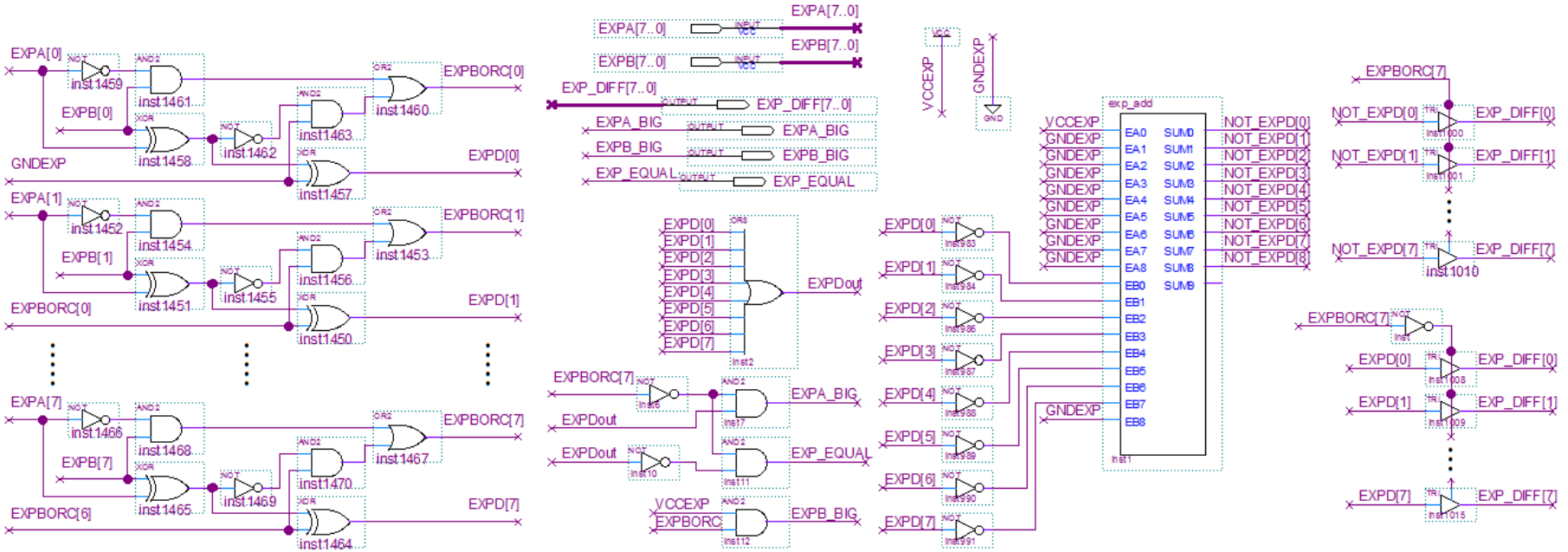
Şekil 5.14. 32 bit kayan noktalı toplayıcının genel yapısı [134]

5.3.2. Üs değerinin belirlenmesi

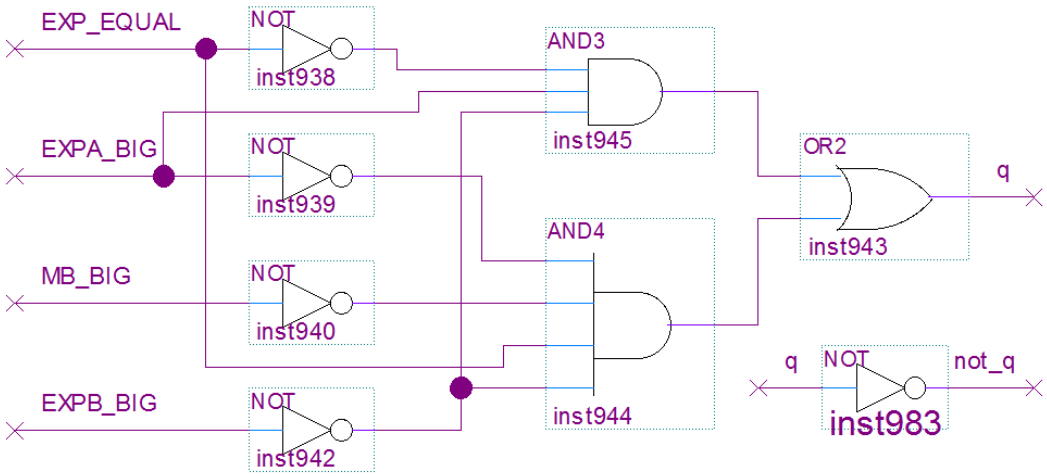
İki sayının üs değerleri Şekil 5.15'te verilen karşılaştırma (comparator) devresi kullanılarak karşılaştırılır [142]. Şekil 5.16'da verilen seçici devre yardımıyla seçici

değişken “q” belirlenir. Çıkış değeri “q=1” olursa ilk sayının üs değeri ikinci sayının üs değerinden büyük veya bu değere eşittir (A≥B). Çıkış değeri “q=0” olursa ikinci sayının üs değeri ilk sayının üs değerinden büyüktür. “q” değeri Şekil 5.17’de verilen devreye aktarılır ve üç-durumlu tamponlar kullanılarak büyük olan üs değeri ön üs değeri olarak seçilir.

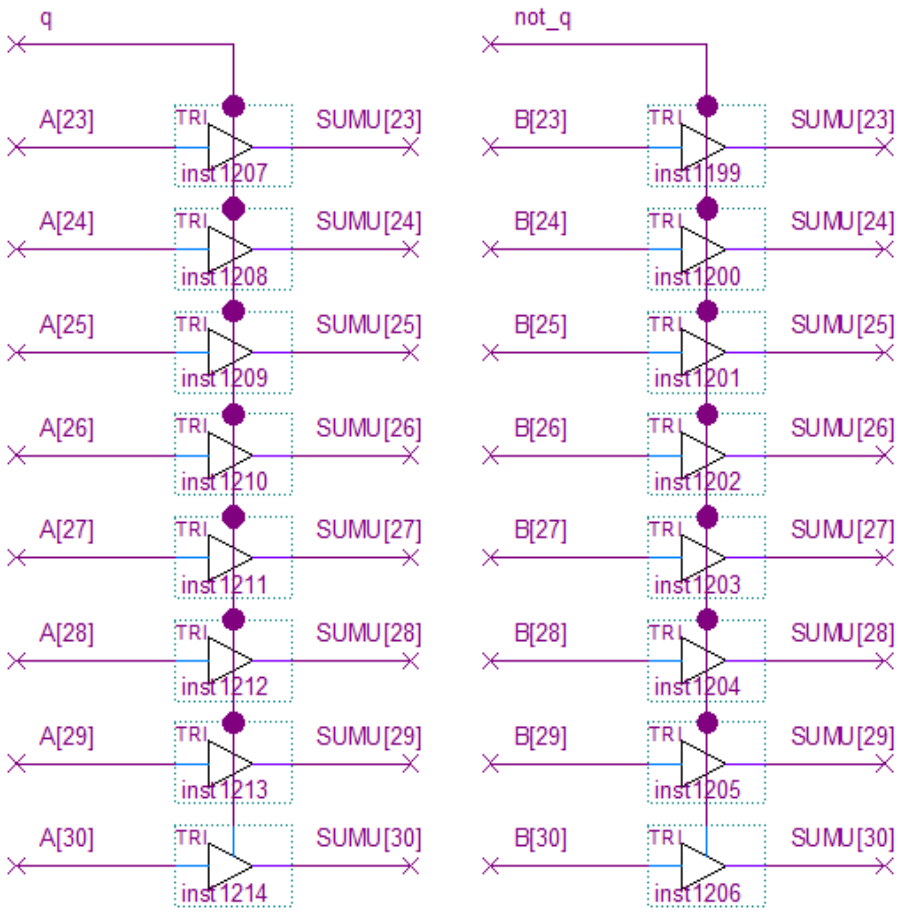
Mantissaların toplanması ile elde edilen sonuç kontrol edilir ve en ağırlıklı “1” tespit edilir. Bu tespit işlemi en ağırlıklı “1” konum tespit devresi (LOPD, Leading One Position Detector) yardımıyla gerçekleştirilir. LOPD Şekil 5.18’de gösterilmiştir.



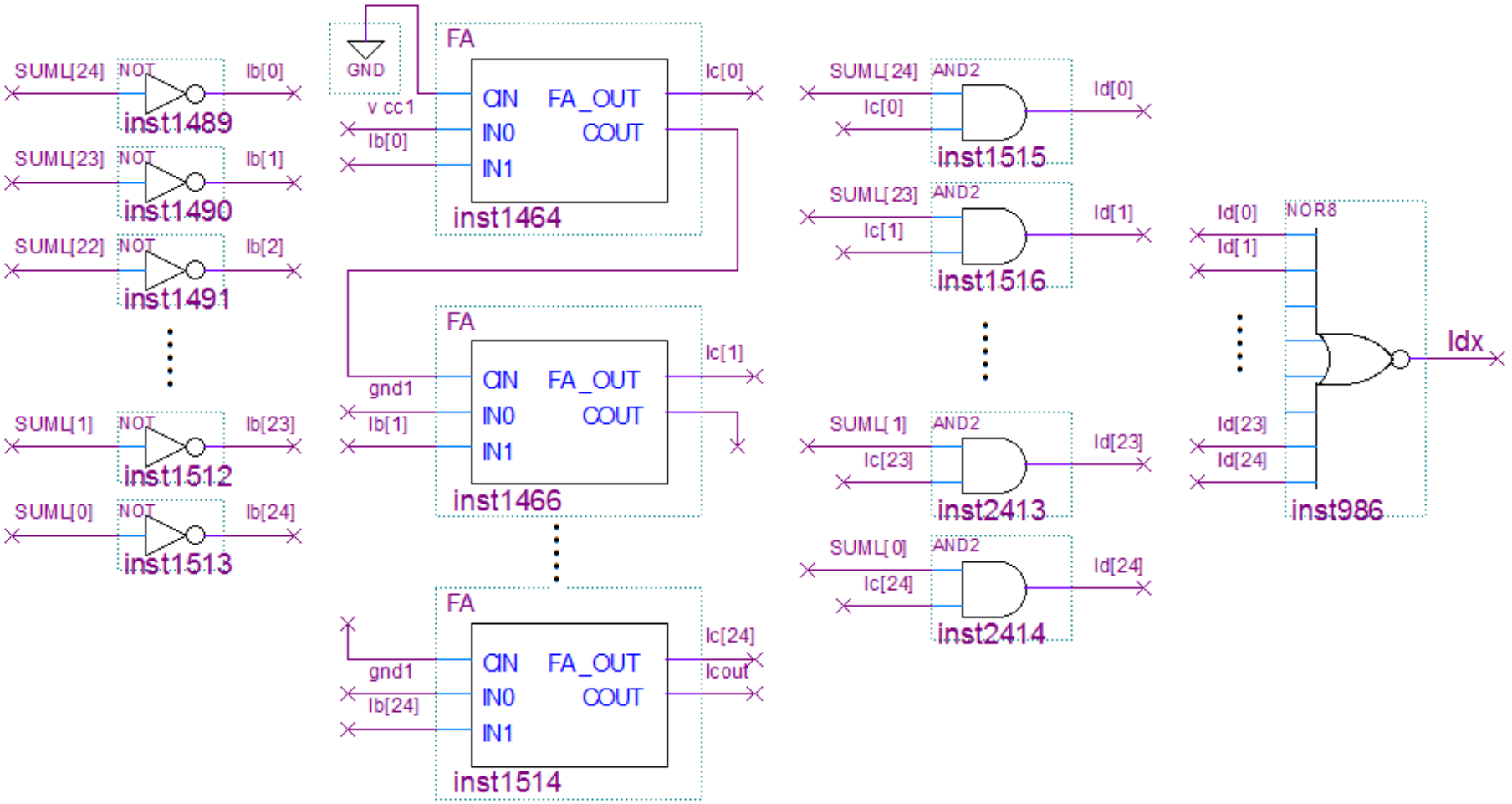
Şekil 5.15. Üs karşılaştırma devresi



Şekil 5.16. Üs için tasarlanan seçici devre



Şekil 5.17. Üs değerinin belirlenmesi



Şekil 5.18. LOPD devresi

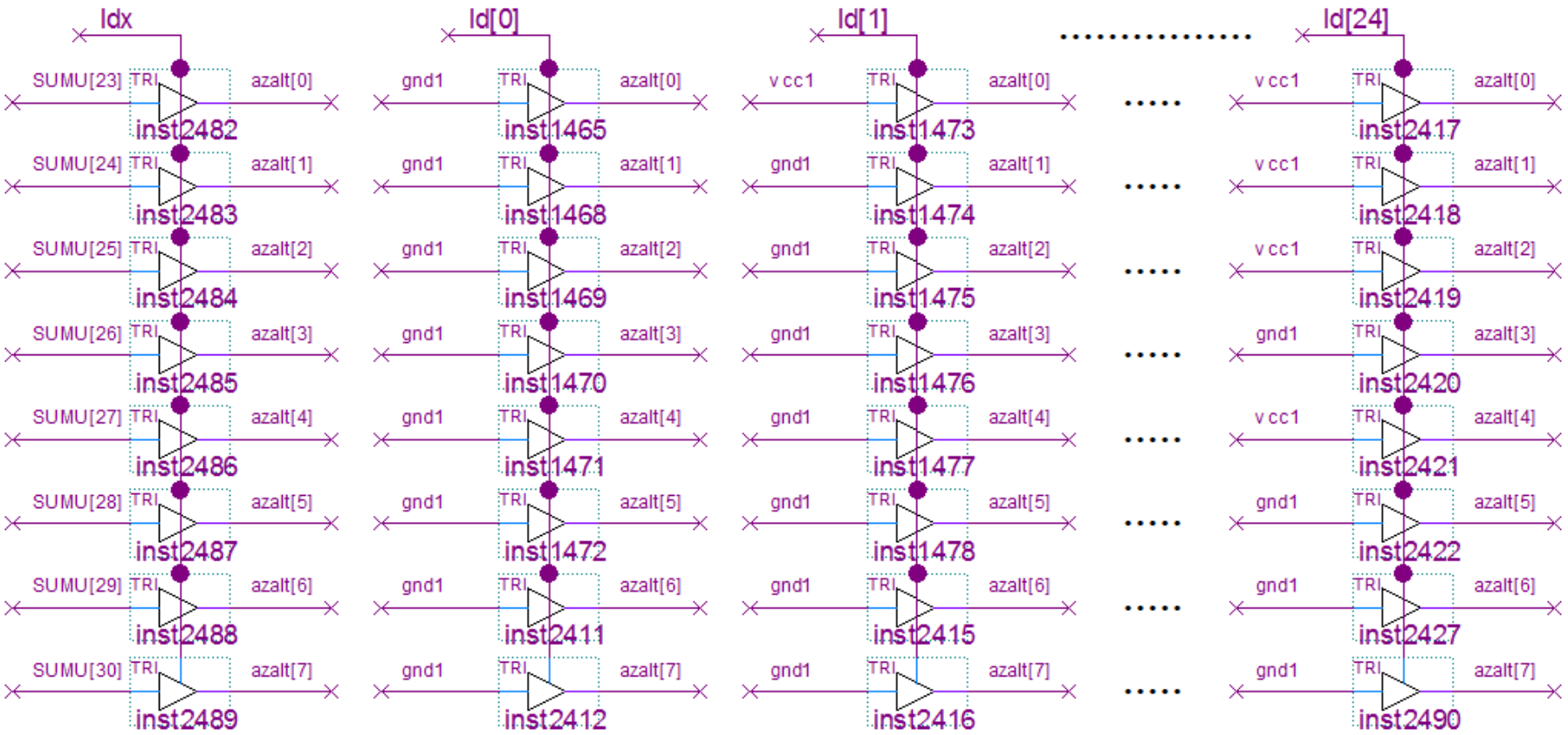
LOPD devresine mantissaların toplamı (SUML[24]-SUML[0]) tersten verilir ve “NOT” kapıları ile her bir bitin tersi alınır. Daha sonra toplayıcı devreler yardımıyla terslenmiş “SUML” toplamına (lb[0]-lb[24]) 1 eklenir ve “lc[0]-lc[24]” toplamı elde edilir.

“lc” ve “SUML” bitlerine “AND” işlemi uygulanır ve “ld[0]-ld[24]” çıkışı elde edilir. Mantissaların toplamındaki (SUML) en ağırlıklı bitin konumuna göre “ld[0]-ld[24]” çıkışlarından yalnız bir tanesi “lojik 1” çıkışı verir. Eğer mantissa toplamındaki bütün bitler “0” ise “ldx” çıkışı “lojik 1” olacaktır.

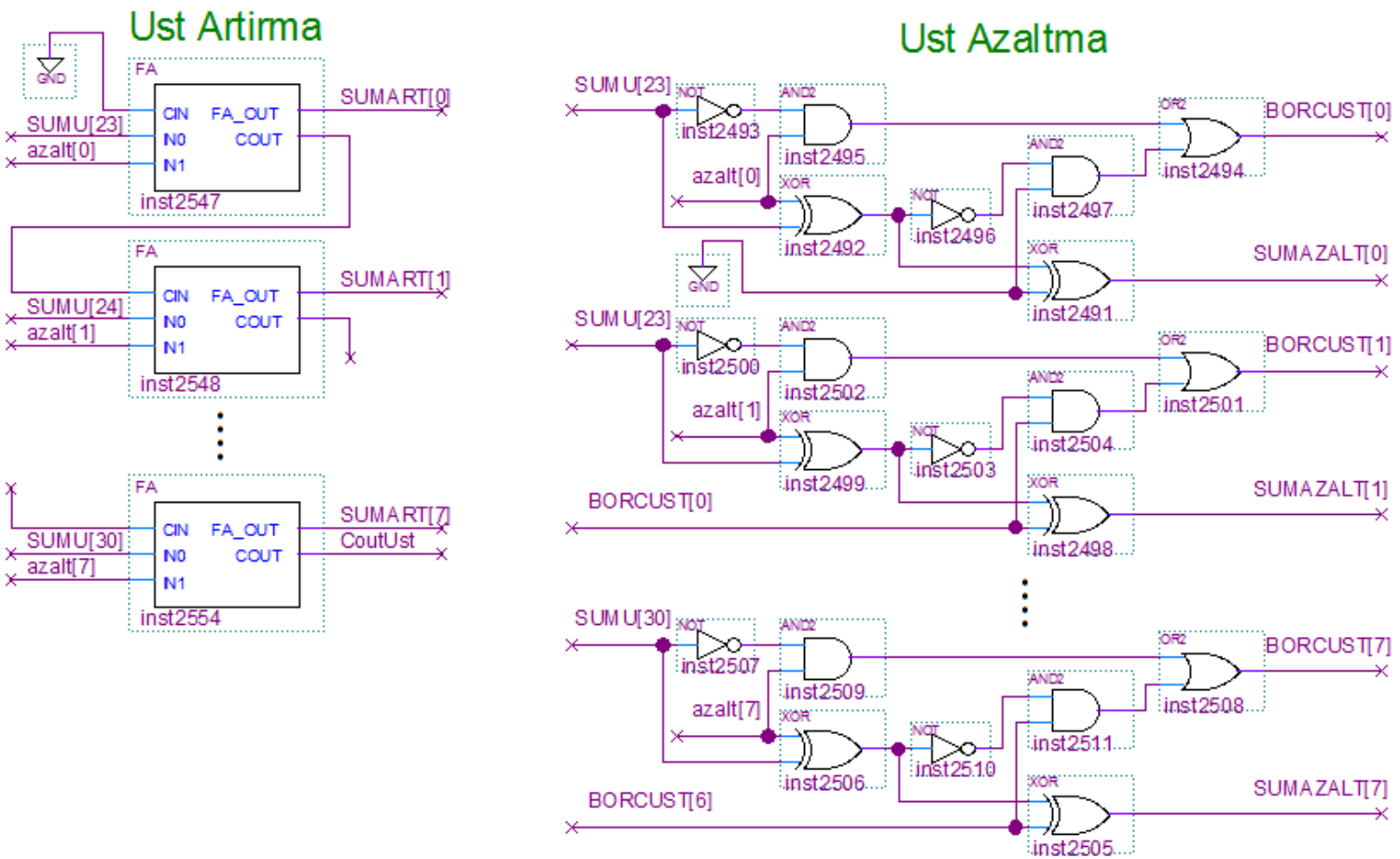
“ldx” ve “ld[0]-ld[24]” bitlerine bağlı olarak ön üs değeri artırılır ya da azaltılır. Bu amaçla artırma-azaltma değerinin (azalt[0]-azalt[7]) belirlenmesi gerekmektedir. Artırma-azaltma değerinin seçilmesi 26 adet üç-durumlu tampon dizisi yardımıyla yapılır. “ldx” ve “ld[0]-ld[24]” bitleri Şekil 5.19’da verilen üç-durumlu tampon dizilerine verilir ve bu dizilerden yalnız biri aktif hale getirilir.

Artırma-azaltma değeri belirlendikten sonra bu değer hem toplayıcı devre yardımıyla ön üs değerine eklenir hem de çıkarıcı bir devre yardımıyla üs değerinden çıkarılır. Toplama ve çıkarma işlemleri eşzamanlı olarak yapılmaktadır. Toplayıcı ve çıkarıcı devre Şekil 5.20’de gösterilmiştir.

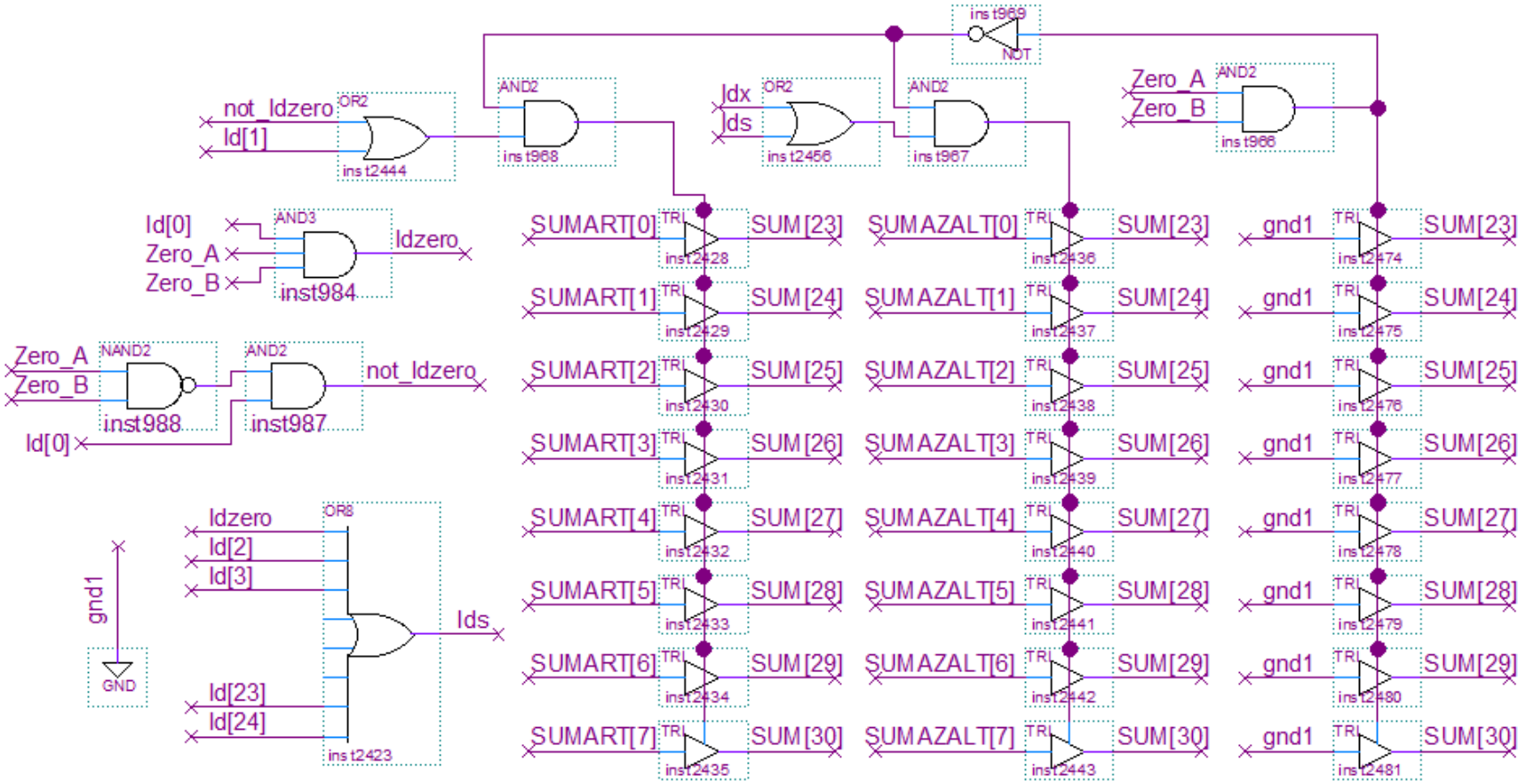
Toplayıcı çıkışı (SUMART[0]-SUMART[7]) ve çıkarıcı devrenin çıkışı (SUMAZALT[0]-SUMAZALT[7]) Şekil 5.21’de verilen üç-durumlu tampon dizilerine giriş olarak verilir. Burada sonucun sıfır olup olmadığına dikkat etmek gerekmektedir. Bu nedenle üç-durumlu tampon dizilerinden bir diziye giriş olarak sıfır (gnd1) verilmiştir ve sonuç sıfır olduğunda bu dizi aktif hale gelir. Sonucun üs değerini belirlemek amacıyla seçici bir devre tasarlanmış ve üç-durumlu tampon dizilerinden yalnız biri aktif hale üs değeri çıkışa aktarılmıştır.



Şekil 5.19. Artırma-azaltma değerini belirlemede kullanılan üç-durumlu tampon dizileri



Şekil 5.20. Ön üs değerinin artırılması ve azaltılması

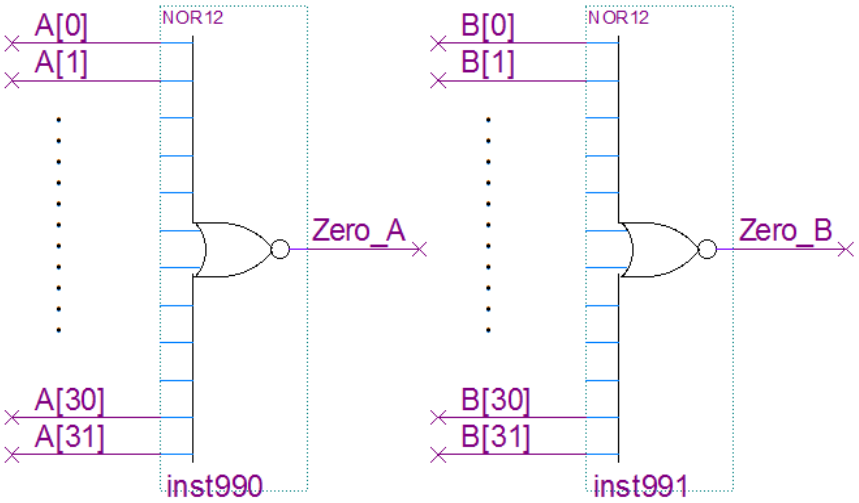


Şekil 5.21. Sonucun üs değerinin seçilmesi ve çıkışa aktarılması

5.3.3. İşaret bitinin belirlenmesi

Sonucun işaret biti, Şekil 5.15'te verilen üs karşılaştırma devresi, Şekil 5.22'de verilen sıfır kontrol devresi ve Şekil 5.23'te verilen mantissa karşılaştırıcı devrenin çıkışları kullanılarak belirlenir. Bu üç devrenin çıkışları Şekil 5.24'te verilen devreye giriş olarak verilir ve işaret biti olarak ilk sayının işareti, ikinci sayının işareti ya da sıfır seçilir.

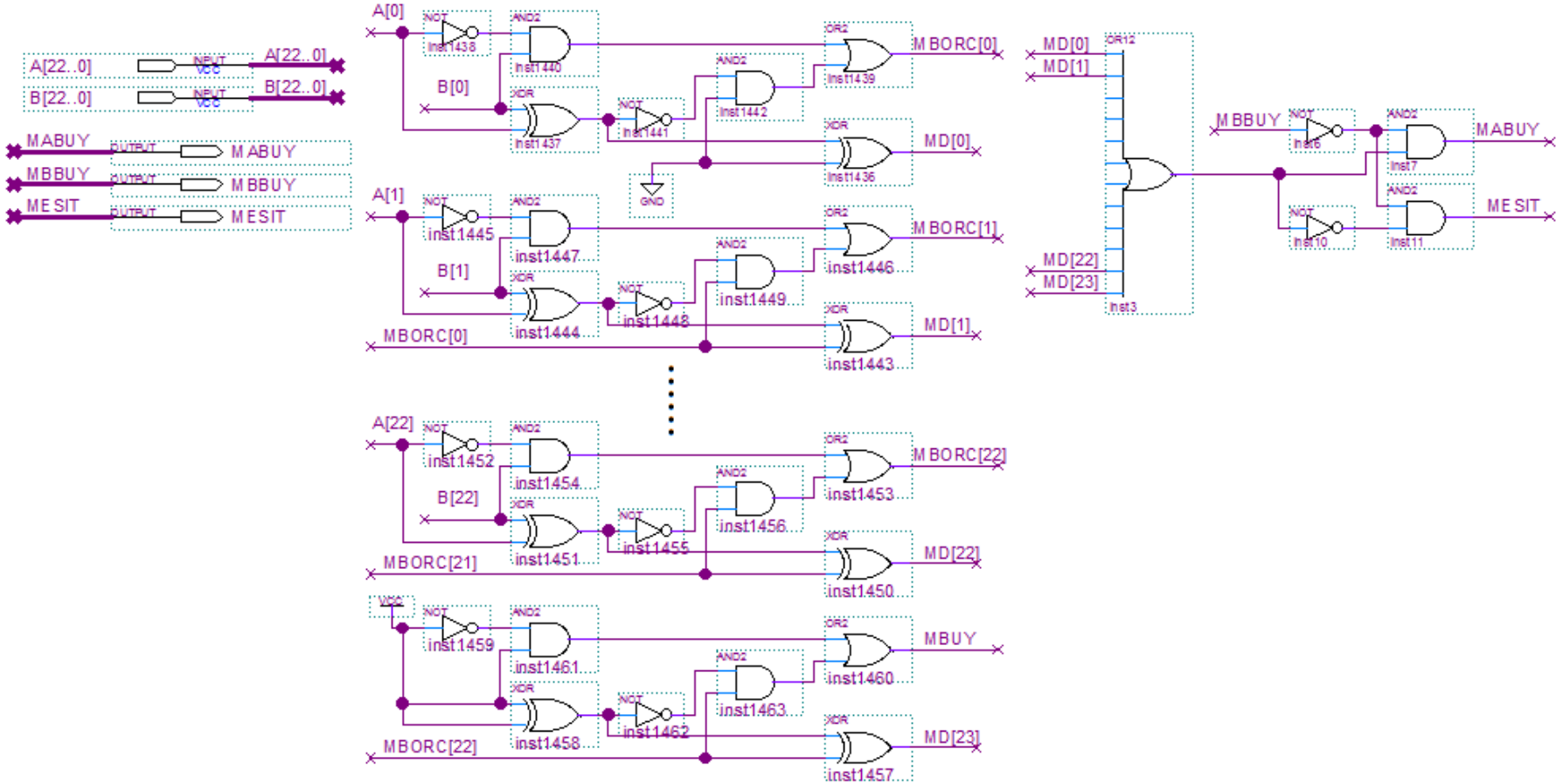
Sıfır kontrol devresi 2 adet NOR kapısından oluşmaktadır. A ve B sayılarının tüm bitleri bu iki kapıya verilir ve sayıların sıfır olup olmadığı kontrol edilir. Sayılardan herhangi birinin sıfır olması durumunda o sayının uygulandığı NOR kapısı lojik 1 çıkışı üretir.



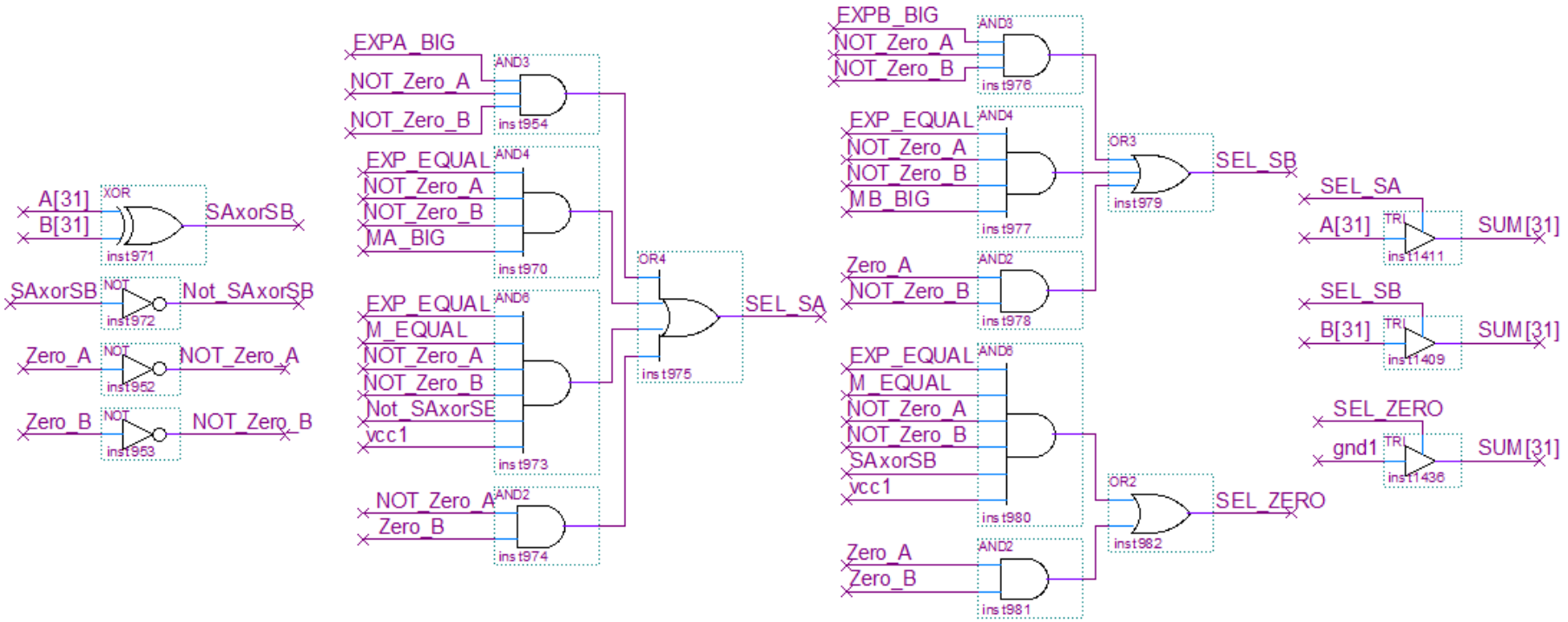
Şekil 5.22. Sıfır kontrol devresi

5.3.4. Mantissaların toplamı

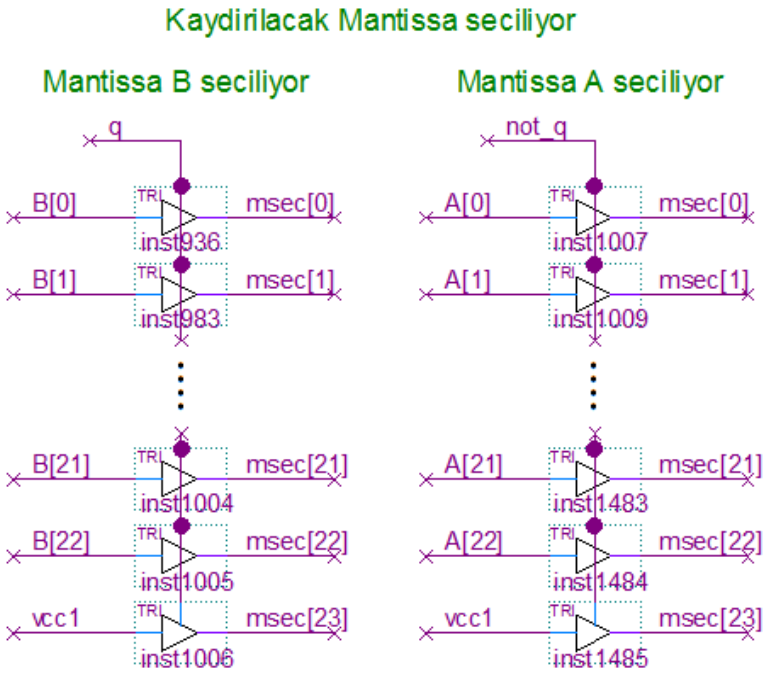
İki sayının mantissa kısımlarının toplanabilmesi için üslerinin aynı olması gerekir. Bu çalışmada toplama işlemini gerçekleştirmek için büyük üs sonucun üssü olarak seçilmiştir. Küçük üssü büyük üsse eşitlemek için küçük sayının mantissası üsler arası fark kadar sağa kaydırılmalıdır. Üsler arası fark Şekil 5.15'te verilen devre ile bulunmuştur. Kaydırılacak küçük mantissa ise Şekil 5.16'da verilen devre çıkışı "q" kullanılarak Şekil 5.25'te verilen üç-durumlu tampon dizileri yardımıyla belirlenir.



Şekil 5.23. Mantissa karşılaştırıcı devre



Şekil 5.24. İşaret bitinin elde edilmesi



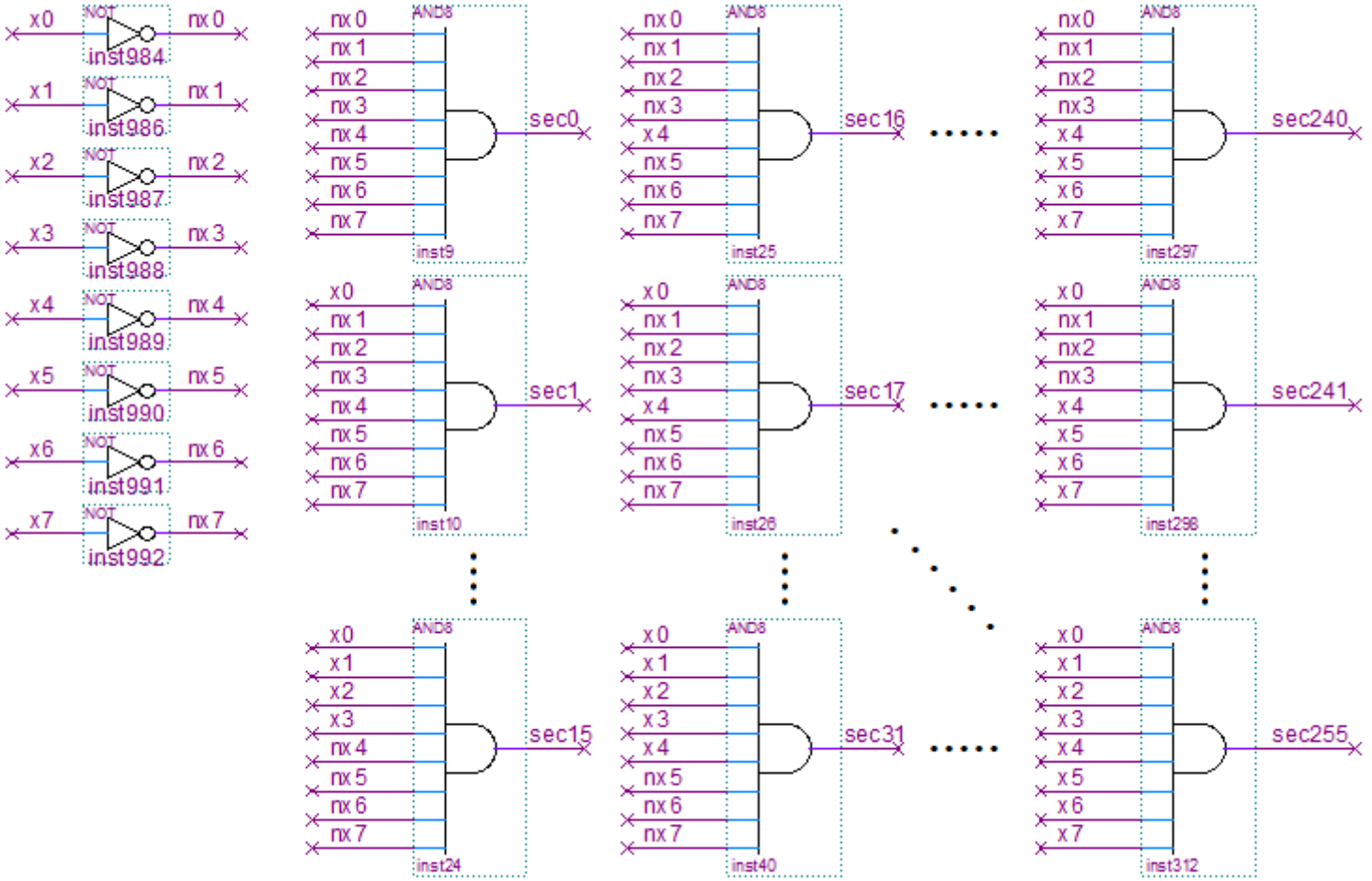
Şekil 5.25. Kaydırılacak küçük mantissanın seçilmesi

Üsler arasındaki farka göre 8x256 seçici devre kullanılarak mantissanın kaç bit kaydırılacağı belirlenir. Tasarlanan 8x256 seçici devre Şekil 5.26'da gösterilmiştir.

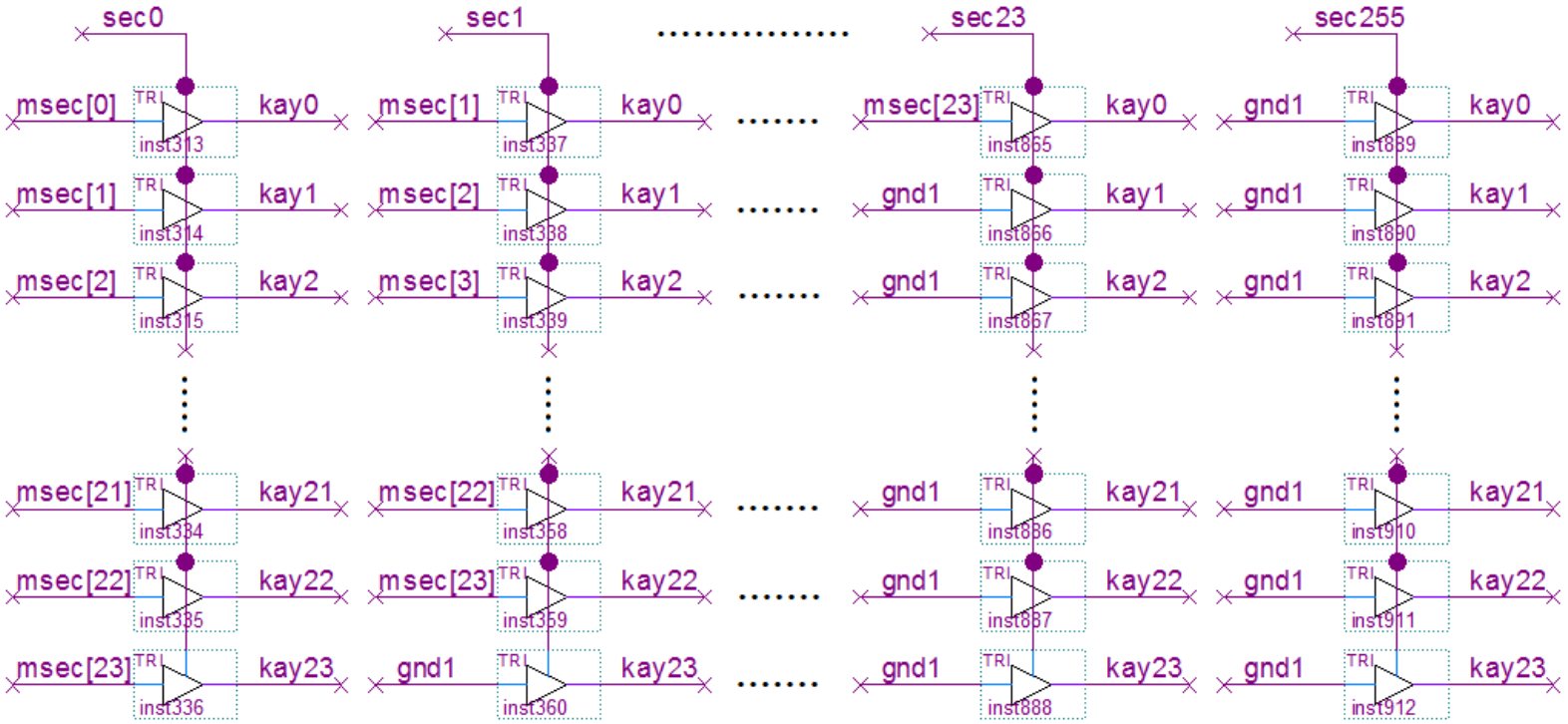
Seçici devre çıkışına bağlı olarak Şekil 5.27'de gösterilen üç-durumlu tampon dizisinden yalnız biri aktif hale getirilir ve kaydırılmış mantissa elde edilir. Kaydırılmış mantissa ile büyük sayının mantissası eş zamanlı olarak toplanır ve çıkarılır. Toplama ve çıkarma için kullanılan devreler Şekil 5.28'de gösterilmiştir.

İşaret bitlerine bağlı olarak Şekil 5.29'da gösterilen üç-durumlu tampon dizilerinden biri aktif hale getirilir ve toplama veya çıkarma sonucunun ön seçimi (SUML[0]-SUML[24]) gerçekleştirilir. Daha sonra bu sonuç Şekil 5.18'de gösterilen LOPD devresine ve Şekil 5.30'da gösterilen üç-durumlu tampon dizilerine verilir.

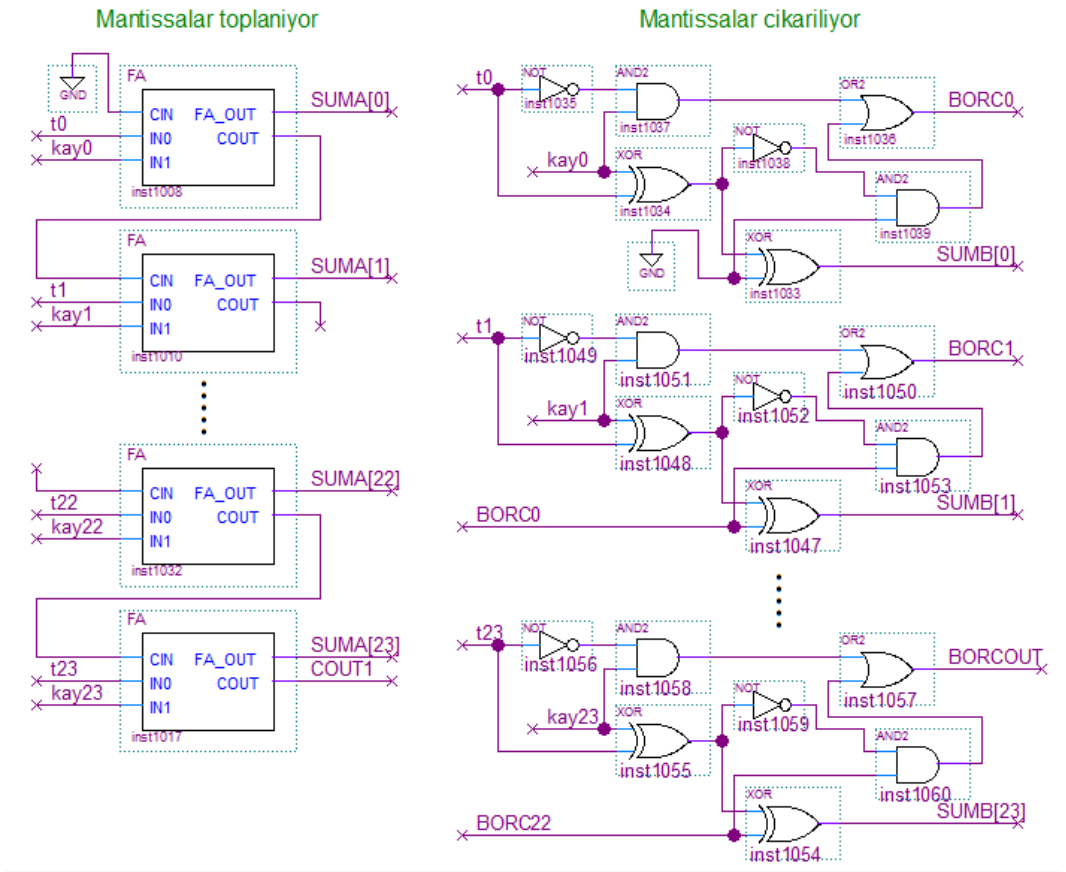
LOPD tarafından üretilen "ldx" ve "ld[0]-ld[24]" bitlerine göre Şekil 5.30'da gösterilen üç-durumlu tampon dizilerinden bir tanesi aktif hale getirilir ve çıkışın mantissa değeri belirlenmiş olur.



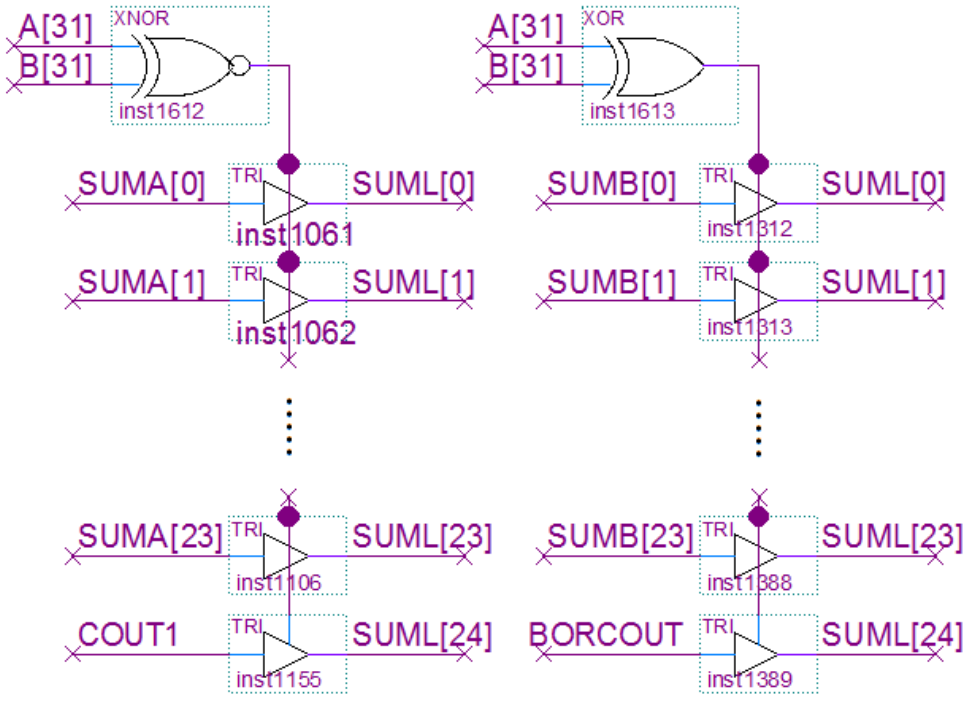
Şekil 5.26. 8x256 seçici devre



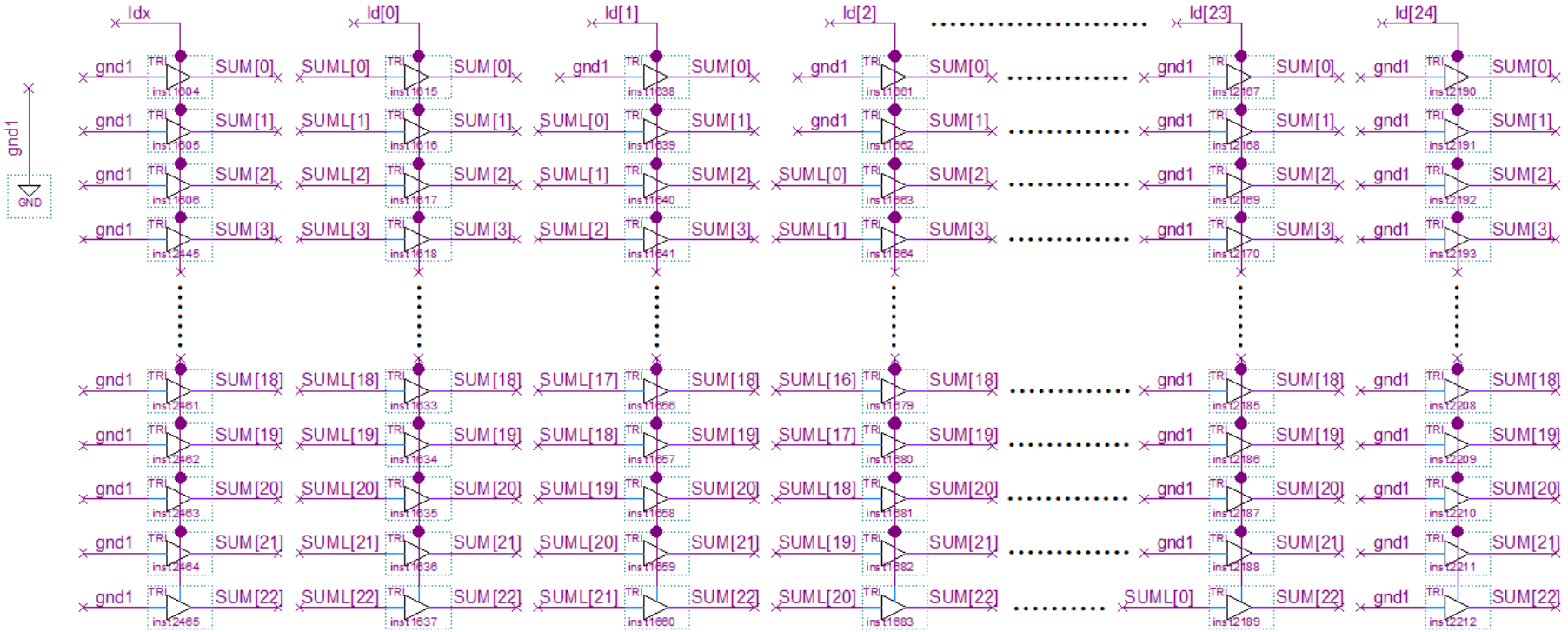
Şekil 5.27. Mantissayı kaydıran üç-durumlu tampon dizisi



Şekil 5.28. Mantissaların toplanması ve çıkarılması



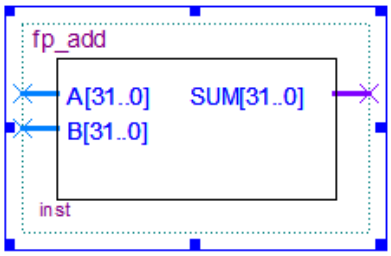
Şekil 5.29. Mantissa toplama veya çıkarma sonucunun ön seçimi



Şekil 5.30. Sonucun mantissa değerinin belirlendiği üç-durumlu tampon dizileri

5.3.5. 32 bit kayan noktalı toplayıcının test edilmesi

Yapay sinir ağlarında kullanılmak üzere IEEE tarafından 754 numaralı referans ile tanımlanmış 32 bit kayan noktalı sayılar için toplayıcı birim tasarlanmıştır. Tasarım işlemi Quartus II programı vasıtasıyla tamamen donanımsal olarak gerçekleştirilmiştir. Toplayıcı tasarımında, kaydırma işlemi için kaydırmalı yazmaç devreleri kullanılmamıştır. Böylece toplayıcı devre, saat darbesinden bağımsız olarak tasarlanmış ve çok daha hızlı çalışır hale gelmiştir. Üç-durumlu tamponlar kaydırma işlemi için kullanılmış bu da devrenin daha karmaşık hale gelmesine sebep olmuştur. Tasarlanan toplayıcının blok yapısı Şekil 5.31'de gösterilmiştir.



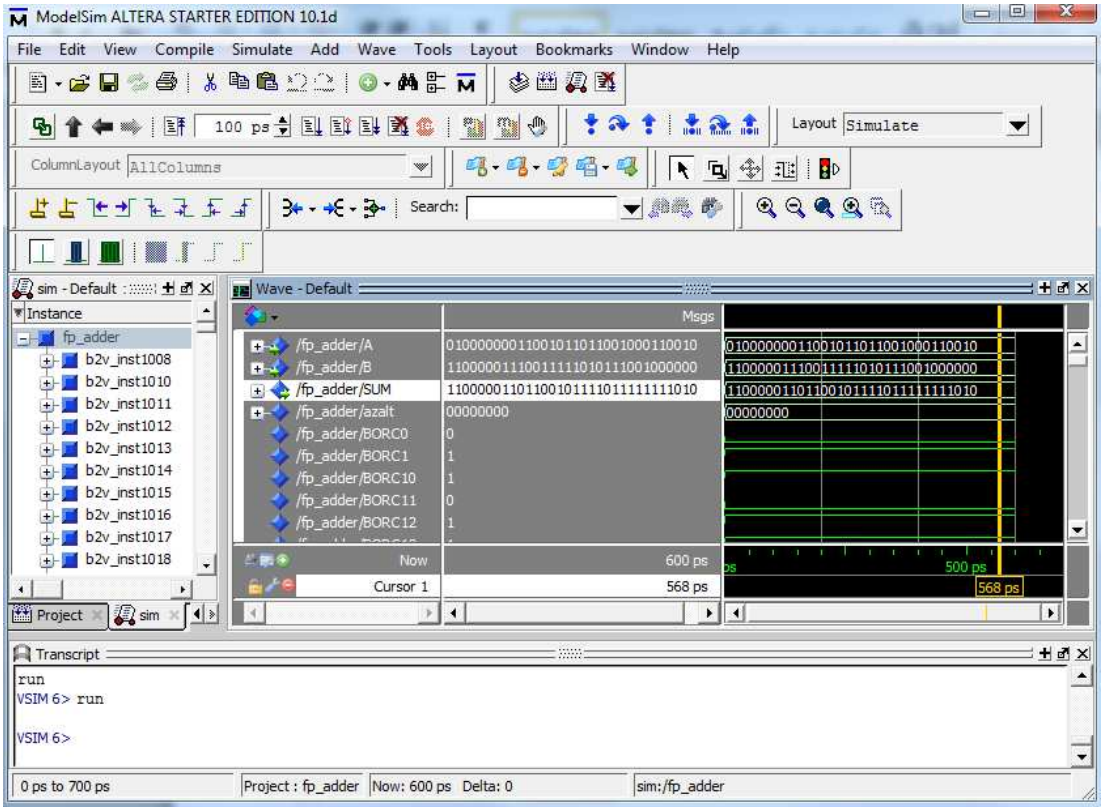
Şekil 5.31. Toplayıcı Blok Gösterimi

Toplayıcı, Tablo 5.4, Tablo 5.5 ve Tablo 5.6'da verilen rastgele sayılar için ModelSim kullanılarak test edilmiştir.

Tablo 5.4. Toplayıcının rastgele sayılar için test edilmesi 1

	10'luk Taban	**32 Bit Kayan Noktalı Gösterim**
Sayı 1	3.589001200	01000000011001011011001000110010
Sayı 2	-25.96008300	11000001110011111010111001000000
Toplayıcı Çıkışı	-22.37108200	11000001101100101111011111111010
Gerçek Çıkış	-22.37108200	11000001101100101111011111111010

Tablo 5.4'te verilen sayılar için toplayıcı doğru sonuç vermiştir. ModelSim ile elde edilen simülasyon sonucu Şekil 5.32'de gösterilmektedir.

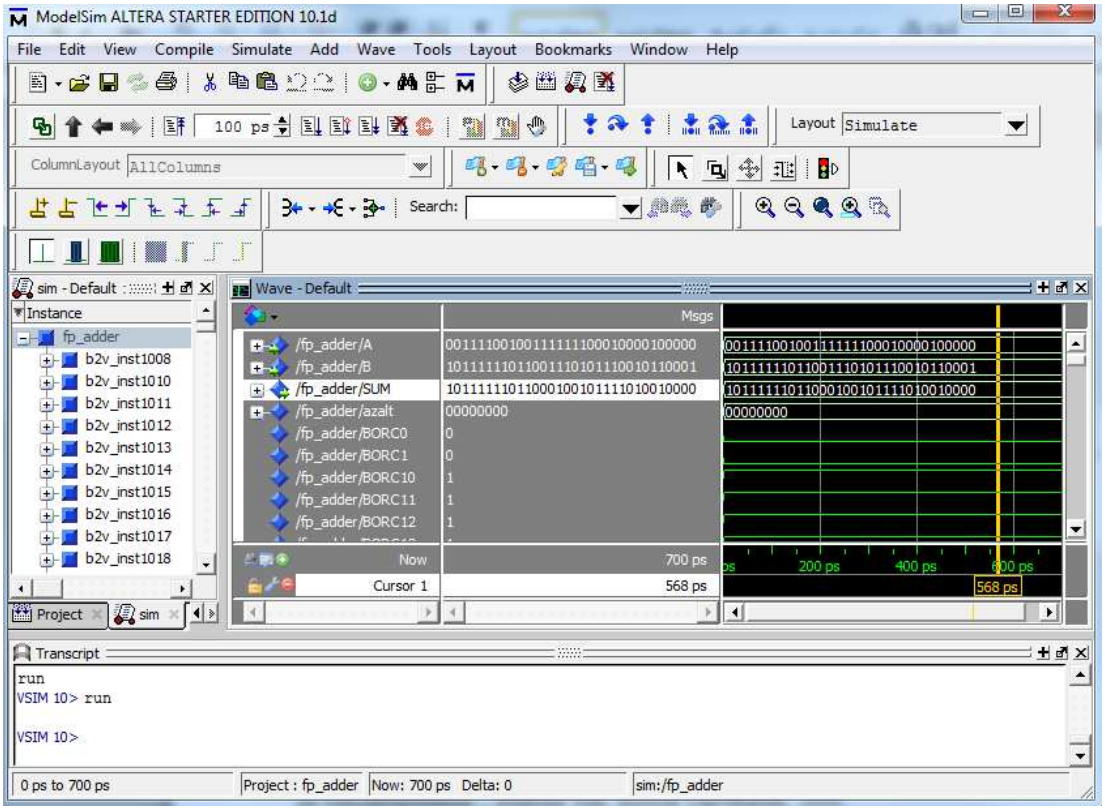


Şekil 5.32. Tablo 5.4'te verilen sayılar için simülasyon sonucu

Tablo 5.5. Toplayıcının rastgele sayılar için test edilmesi 2

	10'luk Taban	**32 Bit Kayan Noktalı Gösterim**
Sayı 1	0.019502700	00111100100111111100010000100000
Sayı 2	-0.903758100	10111111011001110101110010110001
Toplayıcı Çıkışı	-0.884255400	10111111011000100101111010010000
Gerçek Çıkış	-0.884255400	10111111011000100101111010010000

Tablo 5.5'te verilen sayılar için toplayıcı doğru sonuç vermiştir. ModelSim ile elde edilen simülasyon sonucu Şekil 5.33'te gösterilmektedir.

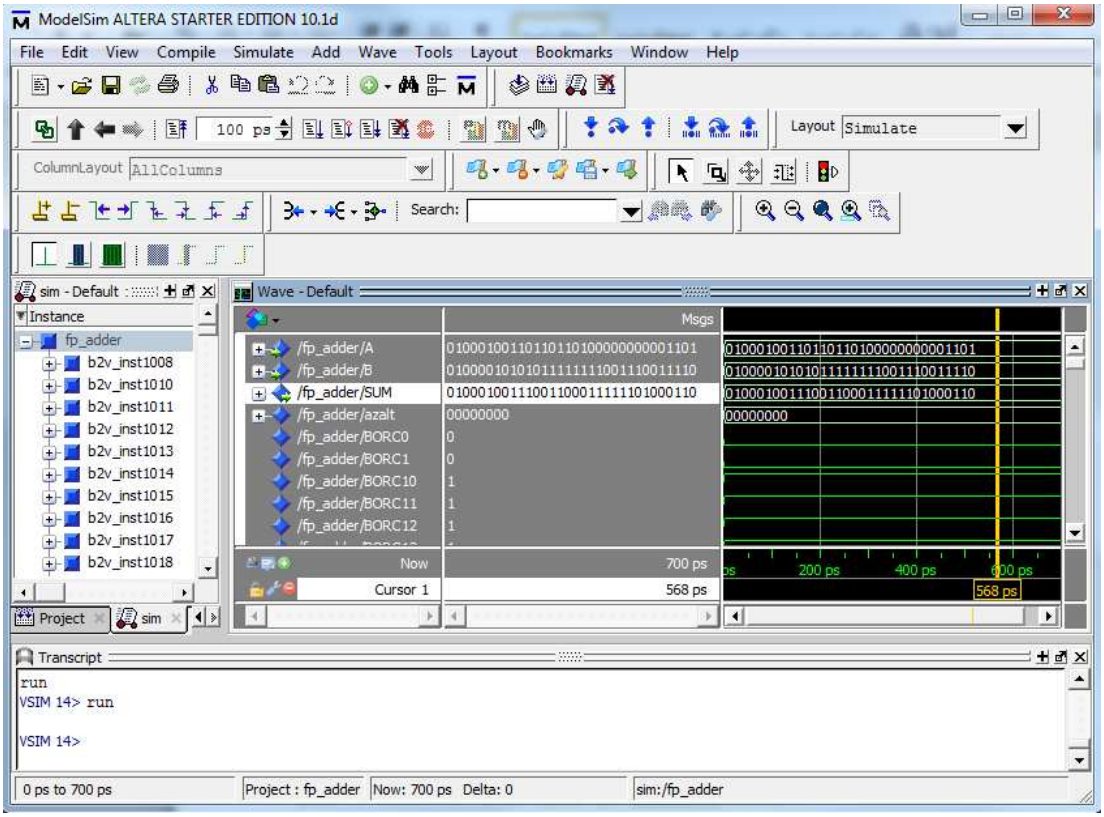


Şekil 5.33. Tablo 5.5'te verilen sayılar için simülasyon sonucu

Tablo 5.6. Toplayıcının rastgele sayılar için test edilmesi 3

	10'luk Taban	**32 Bit Kayan Noktalı Gösterim**
Sayı 1	1754.001600	01000100110110110100000000001101
Sayı 2	87.97581500	01000010101011111111001110011110
Toplayıcı Çıkışı	1841.977300	01000100111001100011111101000110
Gerçek Çıkış	1841.977400	01000100111001100011111101000111

Tablo 5.6'da verilen sayılar için toplayıcı doğru sonuç vermiştir. Ancak yuvarlama nedeniyle sonuçta çok küçük bir fark söz konusudur. ModelSim ile elde edilen simülasyon sonucu Şekil 5.34'te gösterilmektedir.



Şekil 5.34. Tablo 5.6'da verilen sayılar için simülasyon sonucu

5.4. Aktivasyon Fonksiyonu Biriminin Tasarımı

YSA içinde kullanılan aktivasyon fonksiyonlarının yazılım gerçekleştirmelerinde doğruluk ve performans oldukça önemlidir. YSA içinde genelde logaritmik sigmoid (logsig) veya tanjant sigmoid (tansig) fonksiyonları kullanılmaktadır. Bu tez çalışmasında, doğrusal olmayan yapısı ve türevi kolayca alınabilen sürekli bir fonksiyon olması gibi avantajları nedeni ile logaritmik sigmoid fonksiyonu kullanılmıştır.

Fakat logaritmik sigmoid fonksiyonunu FPGA üzerinde gerçeklemek oldukça fazla yer kaybına sebep olmaktadır. Bu sebeple logaritmik sigmoid fonksiyonu "Taylor Seri Açılımı" (TSA) yöntemi ile FPGA üzerinde gerçeklenmiştir. TSA ile elde edilen fonksiyonun, esas fonksiyon ile neredeyse tamamen örtüşmesi ve esas fonksiyona göre daha basit bir aritmetik formülizasyona sahip olması donanım kaynak kullanımlarının azalması bakımından önemlidir. Aktivasyon fonksiyonunu elde

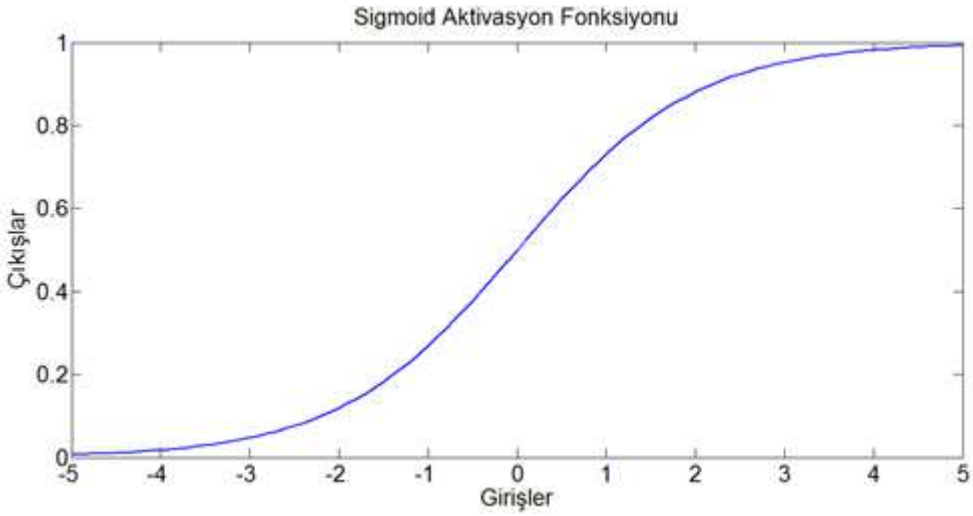
etmek amacıyla daha önce gerçekleştirilen çarpıcı ve toplayıcı devreleri kullanılmıştır.

5.4.1. Logaritmik sigmoid aktivasyon fonksiyonu

İşlemci elemanın davranışını belirleyen en önemli birim aktivasyon fonksiyonudur. Aktivasyon fonksiyonlarına öğrenme eğrisi de denilmektedir. Öğrenme eğrileri türevlenebilir sürekli fonksiyonlardır. Logaritmik sigmoid en çok tercih edilen aktivasyon fonksiyonudur. (Denklem 5.1)

$$\log sig(x) = \frac{1}{1+e^{-x}} \tag{5.1}$$

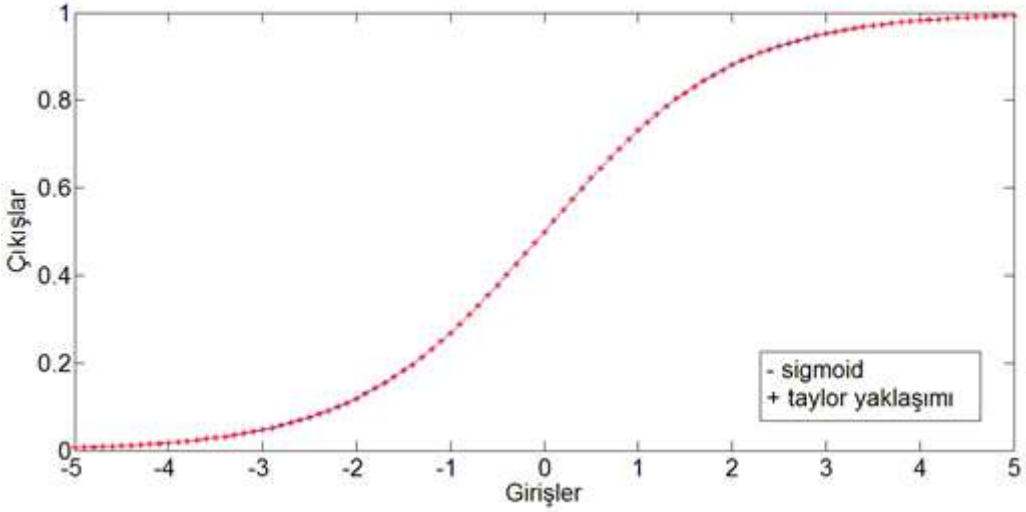
Logaritmik sigmoid fonksiyonu MATLAB komut satırından “logsig” komutu ile işletilir. Fonksiyonun davranış cevabı Şekil 5.35’te görülmektedir.



Şekil 5.35. Logaritmik sigmoid aktivasyon fonksiyonu

Logaritmik sigmoid aktivasyon fonksiyonunun FPGA üzerinde gerçeklenmesi kaynak kullanımını artıracaktır. Bu nedenle TSA yöntemine başvurulmuştur [143]. Denklem 5.2 ile verilen yaklaşımın cevabı Şekil 5.36’da gösterilmektedir.

$$y=\begin{cases}0 & -\infty < x \le -5.45 \\ 0.571859+(0.392773)x+(0.108706)x^2 & \\ +(0.014222)x^3+(0.000734)x^4 & -5.45 < x \le -1.5 \\ \frac{1}{2}+\frac{1}{4}x-\frac{1}{48}x^3+\frac{1}{480}x^5 & -1.5 < x < 1.5 \\ 0.428141+(0.392773)x-(0.108706)x^2 & \\ +(0.014222)x^3-(0.000734)x^4 & 1.5 \le x < 5.45 \\ 1 & 5.45 \le x < \infty \end{cases} \quad \textbf{(5.2)}$$

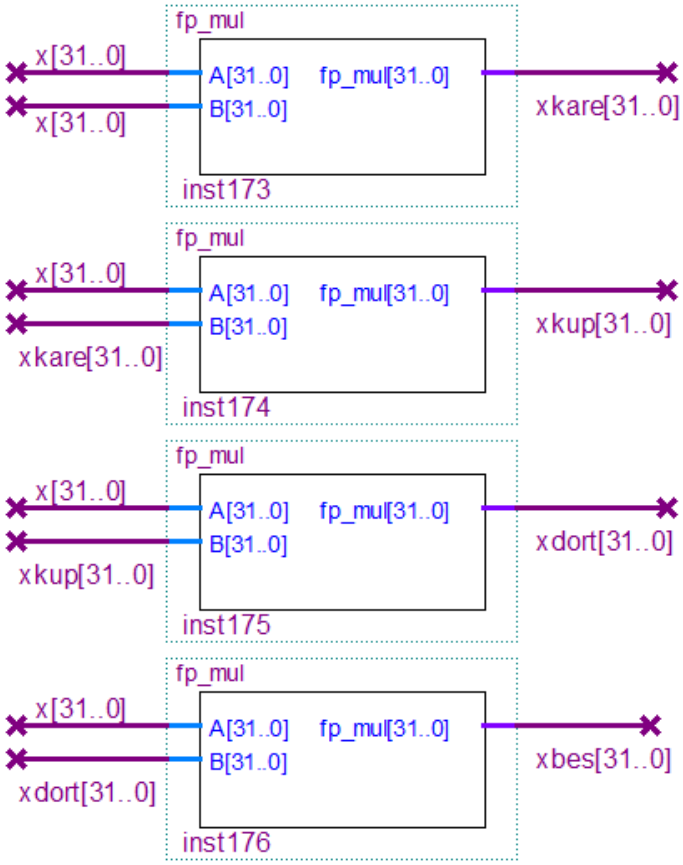


Şekil 5.36. Sigmoid fonksiyonu taylor serisi yaklaşımı

5.4.2. FPGA üzerinde aktivasyon fonksiyonu biriminin gerçeklenmesi

Aktivasyon Fonksiyonu Birimi 12 adet paralel çarpıcı, 9 adet toplayıcı, Denklem 2 ile verilen aralıkları belirleyen karşılaştırıcı dizisi, çıkışı belirleyen mantık devreleri ve çok sayıda üç-durumlu tampon dizisinden meydana gelmektedir. Sisteme verilen giriş Şekil 5.37'de verilen çarpıcı dizisi yardımıyla art arda kendisiyle çarpılır ve girişin kuvvetleri üretilir.

Giriş, Şekil 5.38'de verilen karşılaştırıcı devreler vasıtasıyla "1.5" ve 5.45 değerleri ile karşılaştırılır. Üs ve mantissa kısımları için ikişer adet karşılaştırma devresi kullanılır.

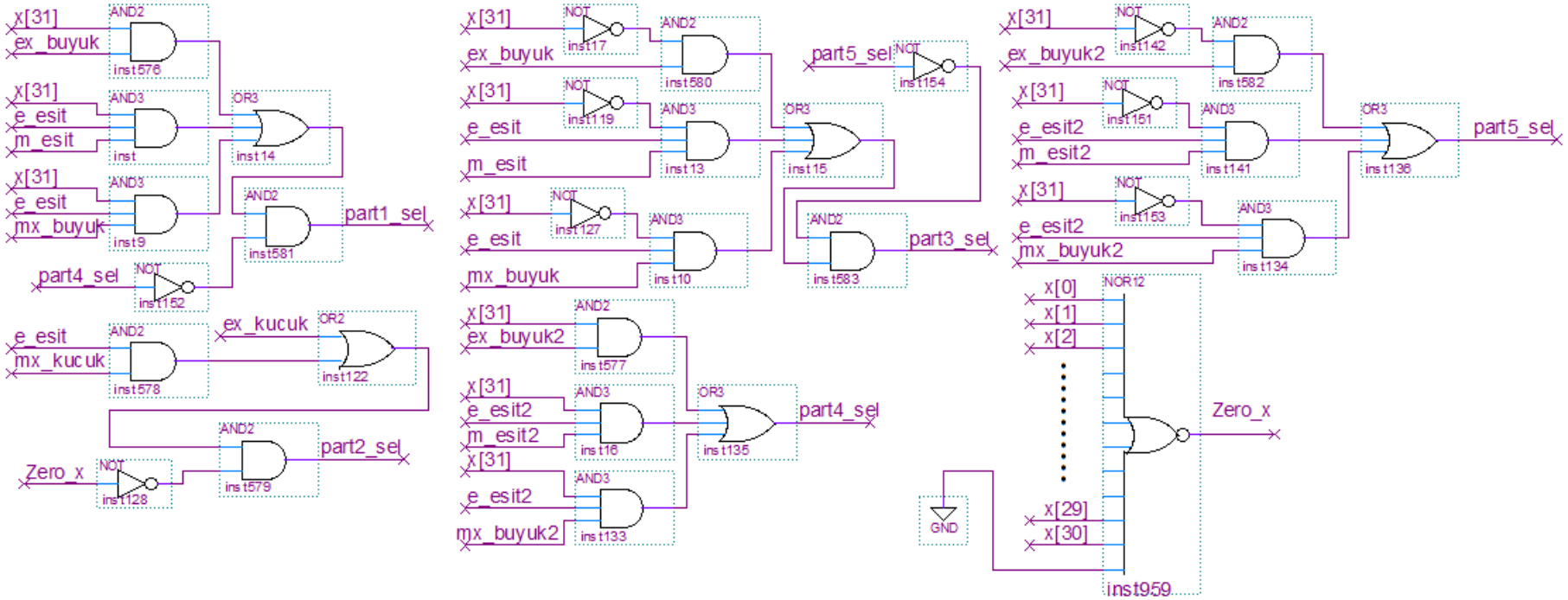


Şekil 5.37. Çarpıcı dizisi

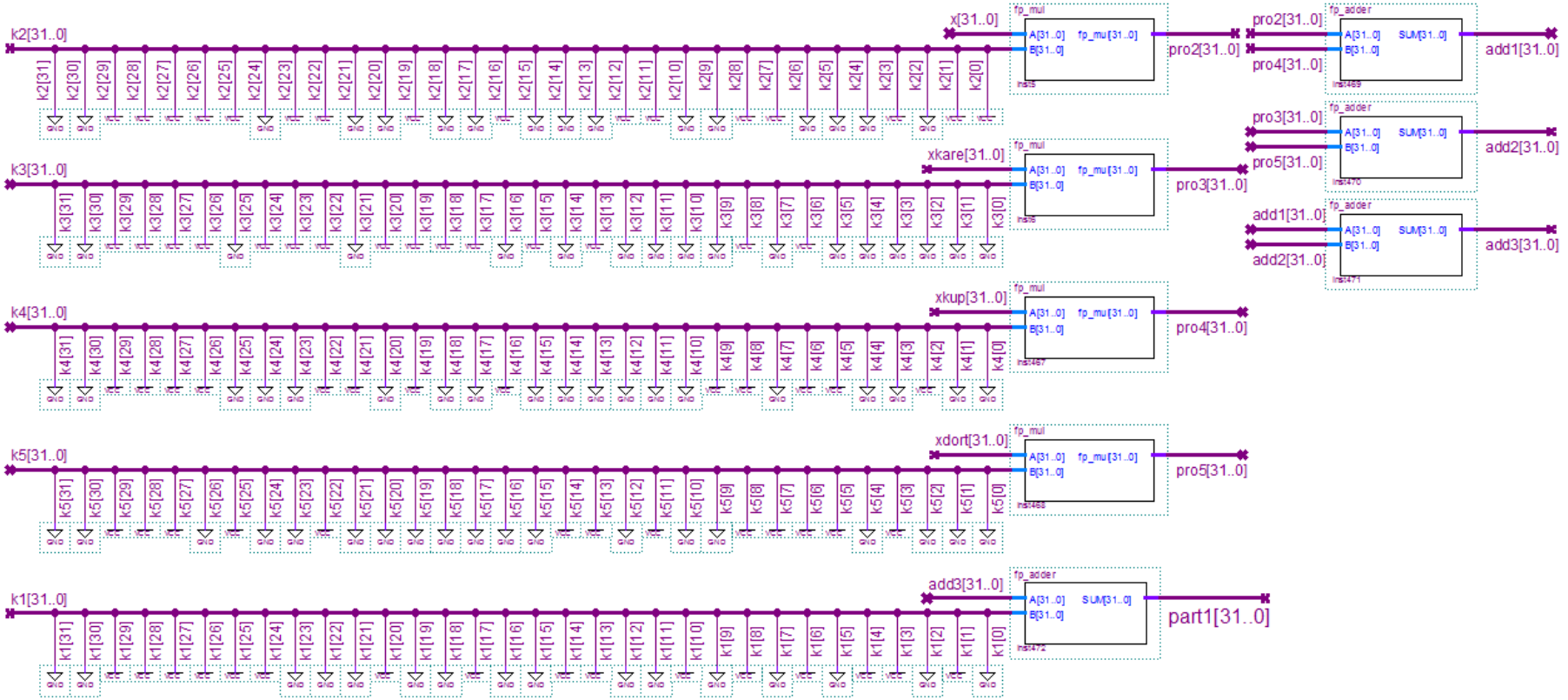
Giriş değerinin “sıfır” olup olmadığı, 5.45’ten büyük olup olmadığı ve -5.45’ten küçük olup olmadığı kontrol edilir. Şekil 5.39’da, girişin bütün bitleri bir “NOR” kapısına verilir. Girişin “sıfır” olması durumunda NOR kapısı çıkış olarak (Zero_x) “0” üretecektir ve çıkışa “0.5” değeri aktarılacaktır. Girişin 5.45’ten büyük olması durumunda çıkış “1”, -5.45’ten küçük olması durumunda çıkış “0” olacaktır.

Şekil 5.38 ile verilen karşılaştırma devrelerinin çıkışları Şekil 5.39’da gösterilen seçici mantık devrelerine giriş olarak verilir. Aktivasyon fonksiyonu bloğunun hızlı cevap verebilmesi amacı ile “0” çıkışının yanı sıra Denklem 5.2’de verilen beş parçalı fonksiyon için beş ayrı cevap aynı anda üretilir. $-5.45<x\leq-1.5$ aralığı için tasarlanan devre Şekil 5.40’ta, $-1.5<x<1.5$ aralığı için tasarlanan devre Şekil 5.41’de ve $1.5\leq x<5.45$ aralığı için tasarlanan devre Şekil 5.42’de gösterilmiştir. Üretilen cevaplar Şekil 5.43’de verilen üç-durumlu tampon dizilerine aktarılır. Seçici mantık devreleri yardımıyla üç diziden birisi aktive edilerek sonuç çıkışa aktarılır.

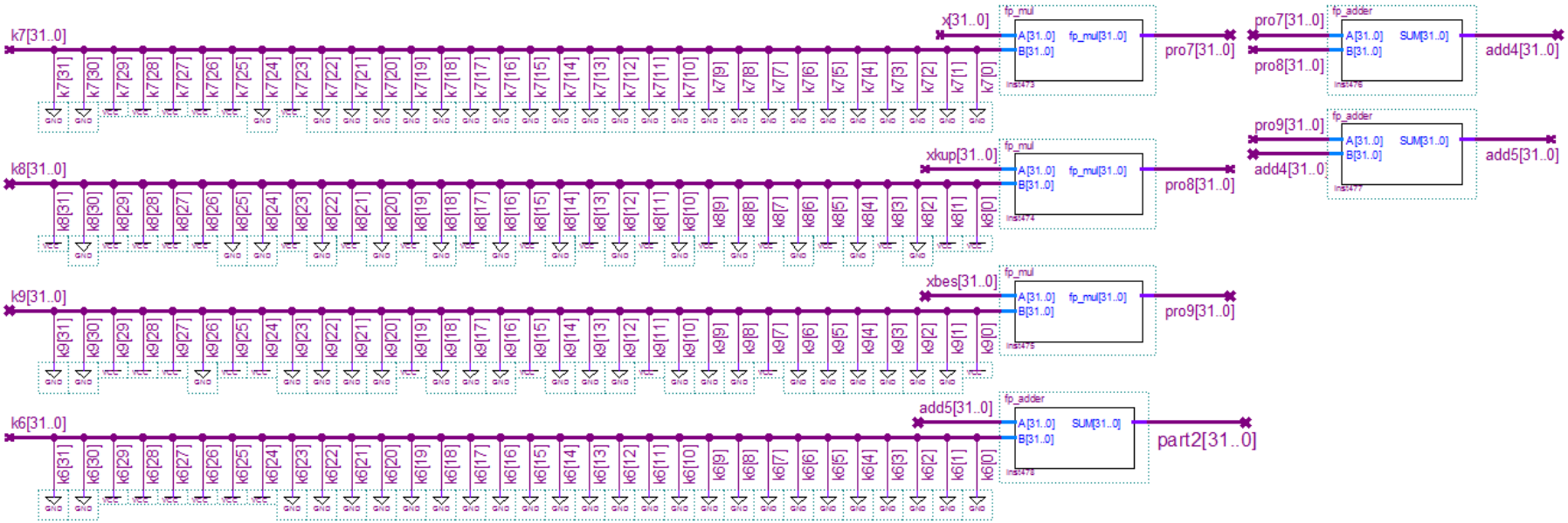
Şekil 5.38. Karşılaştırma devresi



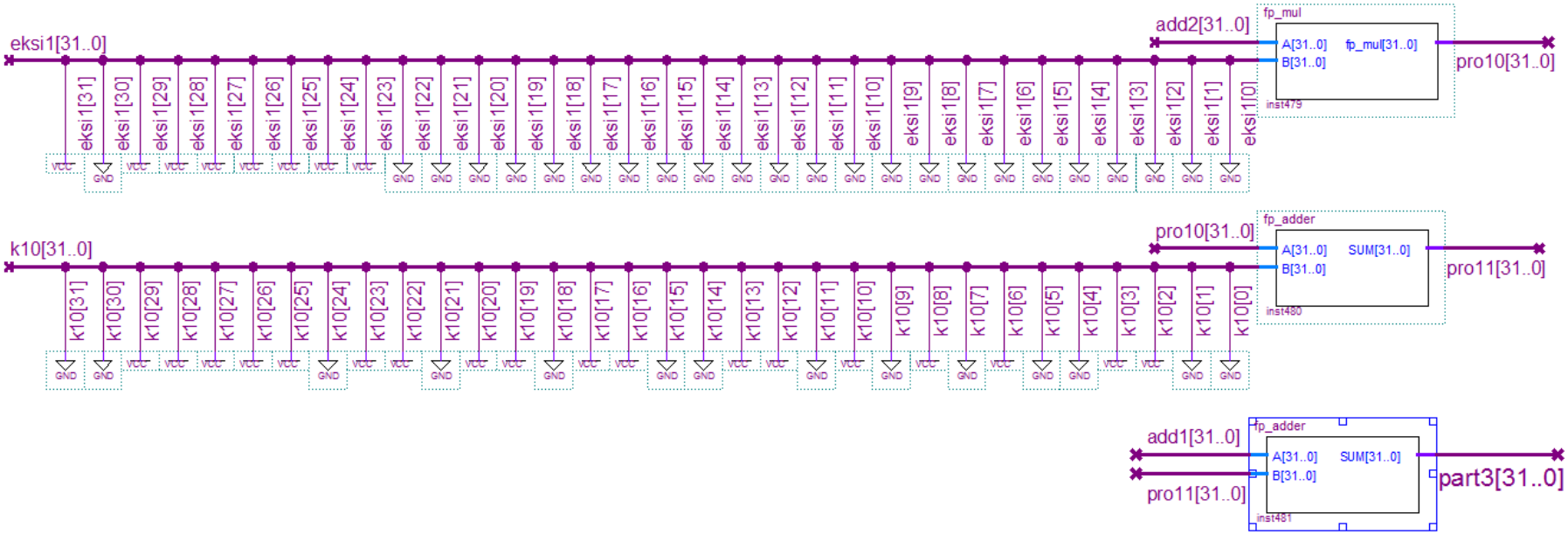
Şekil 5.39. Üç-durumlu tampon dizilerini aktive eden mantık devreleri



Şekil 5.40. -5.45<x≤-1.5 aralığını hesaplayan devre



Şekil 5.41. -1.5<x<1.5 aralığını hesaplayan devre

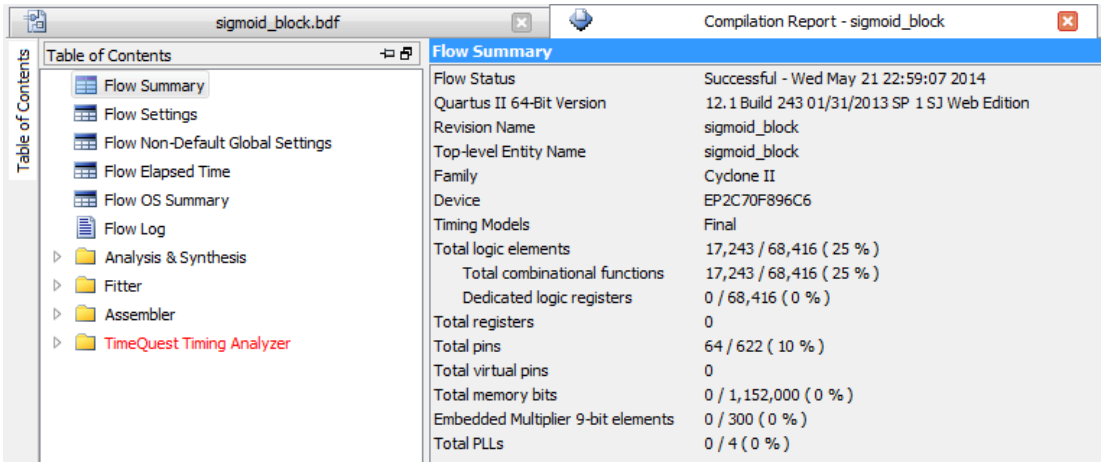


Şekil 5.42. 1.5≤x<5.45 aralığını hesaplayan devre

Şekil 5.43. Çıkışı belirleyen üç-durumlu tampon dizileri

5.4.3. Aktivasyon fonksiyonu biriminin derlenmesi

Quartus II programı ile donanımsal olarak tasarlanan devre Altera Cyclone II EP2C70F896C6 cihazı üzerinde derlenmiştir. Derleme sonuçları Şekil 5.44'te gösterilmektedir.

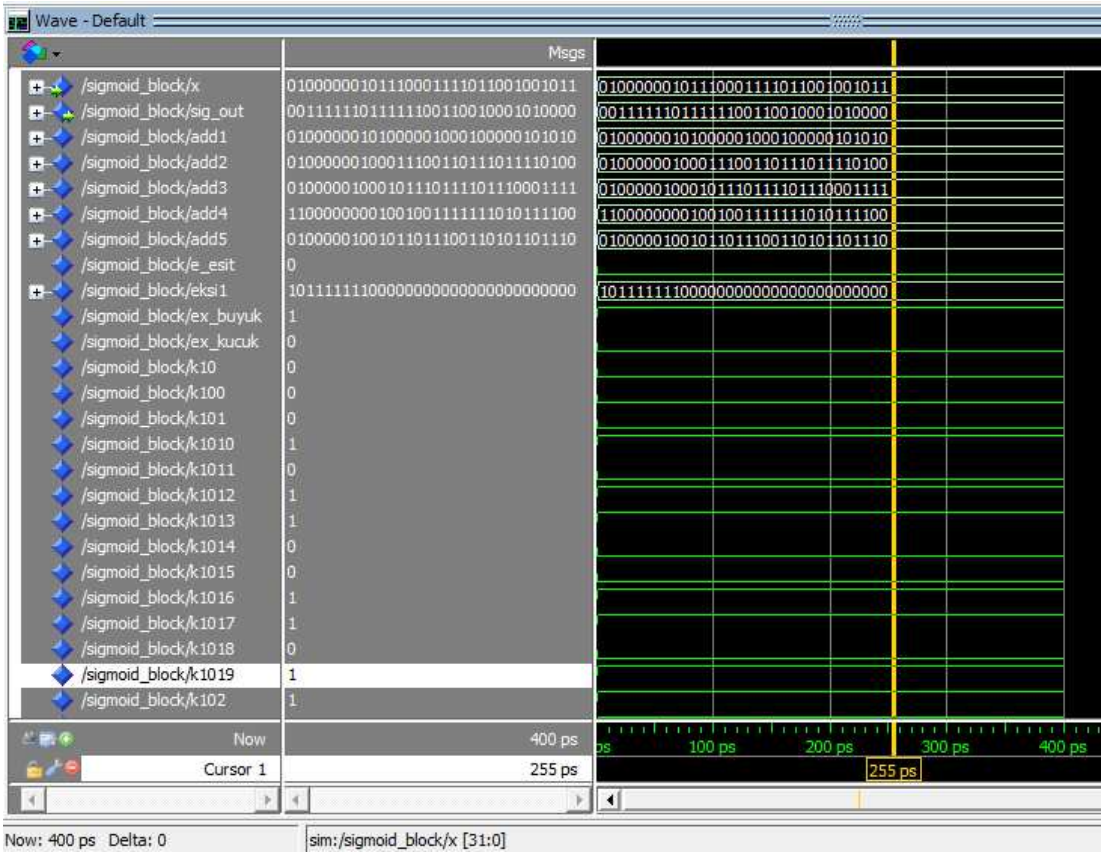


Şekil 5.44. Derleme sonuçları

Devre entegre üzerinde 25%'e karşılık gelen 17243 lojik eleman, 64 pin ve 10%'a karşılık gelen bağlantı kullanmaktadır.

5.4.4. Aktivasyon fonksiyonu biriminin test edilmesi

Quartus II programı ile aktivasyon fonksiyonu bloğunun VHDL yazılımı üretilmiş ve Şekil 5.45'te gösterildiği gibi ModelSim programı ile simülasyonu yapılmıştır. Tablo 5.7'de gösterildiği gibi simülasyon sonuçları MATLAB programında elde edilen gerçek sonuçlar ile karşılaştırılmış ve simülasyon sonuçlarının gerçek sonuçlar ile aynı olduğu görülmüştür.



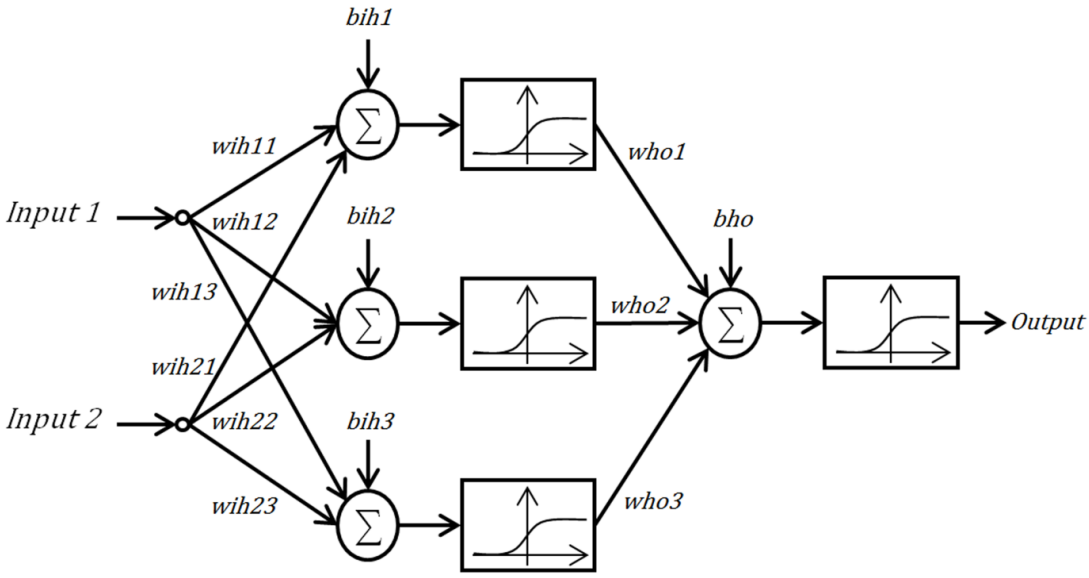
Şekil 5.45. ModelSim ile simülasyon sonuçlarının elde edilmesi

5.5. 2x3x1 Boyutlu YSA'nın FPGA Üzerinde Gerçeklenmesi

Bu tez çalışmasında 2 giriş, 3 nöronlu bir gizli katman ve 1 nöronlu çıkış katmanından oluşan 2x3x1 boyutlu bir YSA mimarisi FPGA üzerinde gerçeklenmiştir. YSA'nın ve FPGA'nın doğasında bulunan paralel yapıya uygun olması amacıyla paralel bir çarpıcı ve kaydırmalı yazmaç kullanmayan bir toplayıcı tasarlanmıştır. Bu devreler kullanılarak aktivasyon fonksiyonu bloğu elde edilmiştir. Paralel çarpıcı, toplayıcı ve aktivasyon fonksiyonu bloğu kullanılarak Şekil 5.46 ile verilen 2x3x1 boyutlu MLNN ağı gerçekleştirilmiştir.

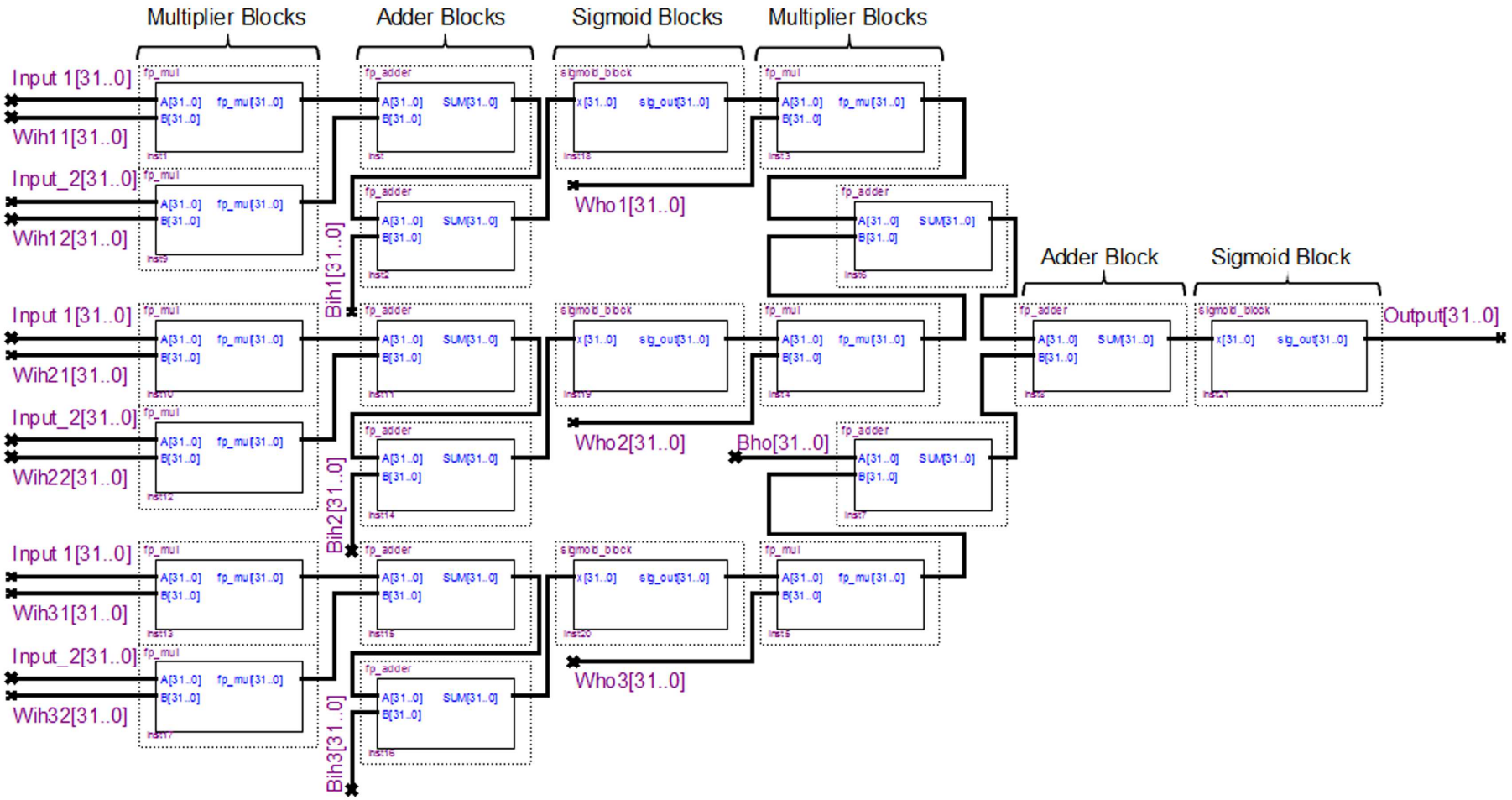
Tablo 5.7. Simülasyon sonuçları ile gerçek sonuçların karşılaştırılması

Sayı	Kayan Noktalı Gösterimi	Gerçek Sonuç	Simülasyon Sonucu
5.780065	01000000101110001111011001001011	0.9937	0.99371815
-3.196784	11000000010011001001100000011100	0.0392	0.03919542
-2.486906	11000000000111110010100101111000	0.0767	0.07671416
-1.500000	10111111100000000000000000000000	0.1830	0.18300468
-0.745902	10111111001111101111001101101111	0.3217	0.32168925
0.018963	00111100100110110101100001001011	0.5047	0.5047406
1.500000	00111111110000000000000000000000	0.8170	0.8169953
2.095836	01000000000001100010001000101101	0.8906	0.8906005



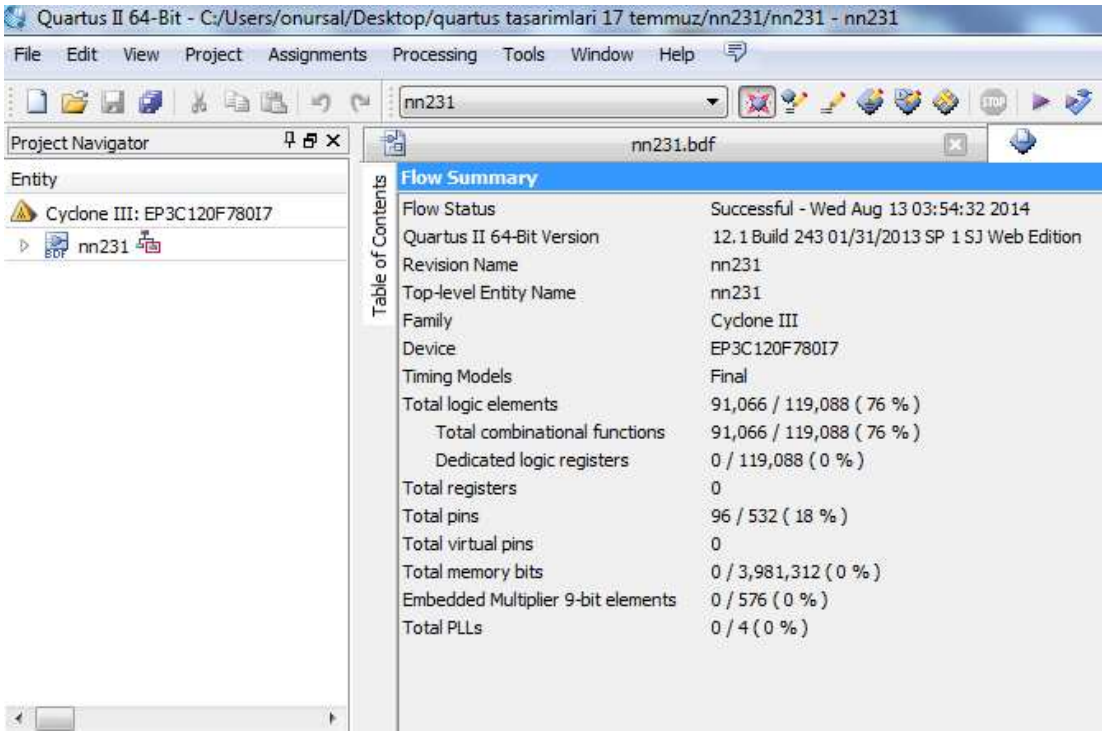
Şekil 5.46. 2x3x1 boyutlu MLNN ağı

MLNN, 32 bit kayan noktalı nümerik tanımlamaya göre işlem yapacak şekilde tamamen donanımsal olarak tasarlanmıştır. Quartus II programı ile tasarlanan 2x3x1 boyutlu MLNN ağı Şekil 5.47'de gösterilmiştir. MLNN yapısı, kayan noktalı işlem yapan 9 paralel çarpıcı, 9 toplayıcı ve 4 sigmoid aktivasyon fonksiyonu bloğundan meydana gelmektedir.



Şekil 5.47. FPGA üzerinde gerçeklenen MLNN donanımı

2x3x1 boyutlu MLNN ağı, Altera firmasına ait Cyclone III EP3C120F780I7 FPGA yongası üzerinde tasarlandığında gerçekleşen kaynak kullanımı Şekil 5.48'de gösterilmektedir.



Şekil 5.48. MLNN kaynak kullanımı

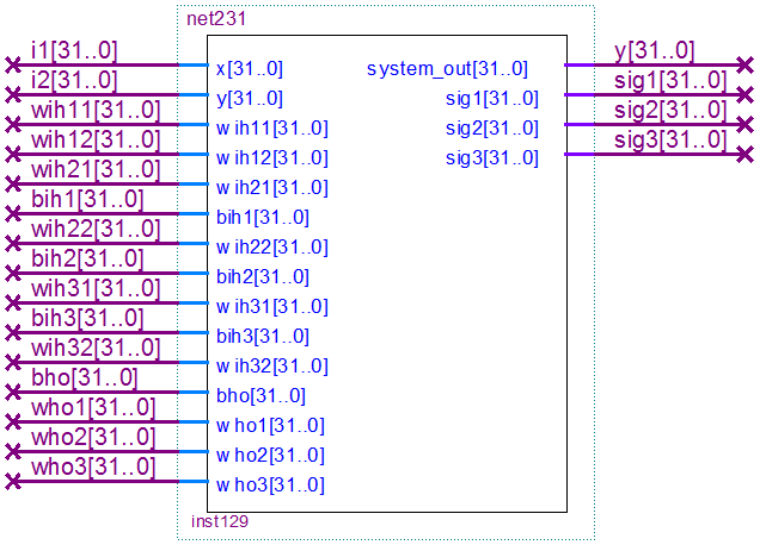
MLNN, toplam lojik elemanların 76%'sına karşılık gelen 91066 lojik eleman kullanmıştır. MLNN donanımı ve alt birimlerine ait kaynak kullanımı Tablo 5.8'de gösterilmiştir.

Tablo 5.8. MLNN donanımı ve alt birimlerin kaynak kullanımı

Birim	**Toplam Lojik Eleman**
MLNN	91066
Paralel Çarpıcı	1595
Toplayıcı	864
Sigmoid Bloğu	16698

5.6. YSA'nın Eğitim Algoritmasının FPGA Üzerinde Gerçeklenmesi

XOR probleminin eğitimi amacıyla, Şekil 5.49'da blok tasarımı gösterilen 2x3x1 boyutlu MLNN'nin eğitim algoritması, bu yapıya kontrol donanımları, geri besleme donanımları ve hafıza donanımları eklenmek suretiyle gerçekleştirilmiştir.

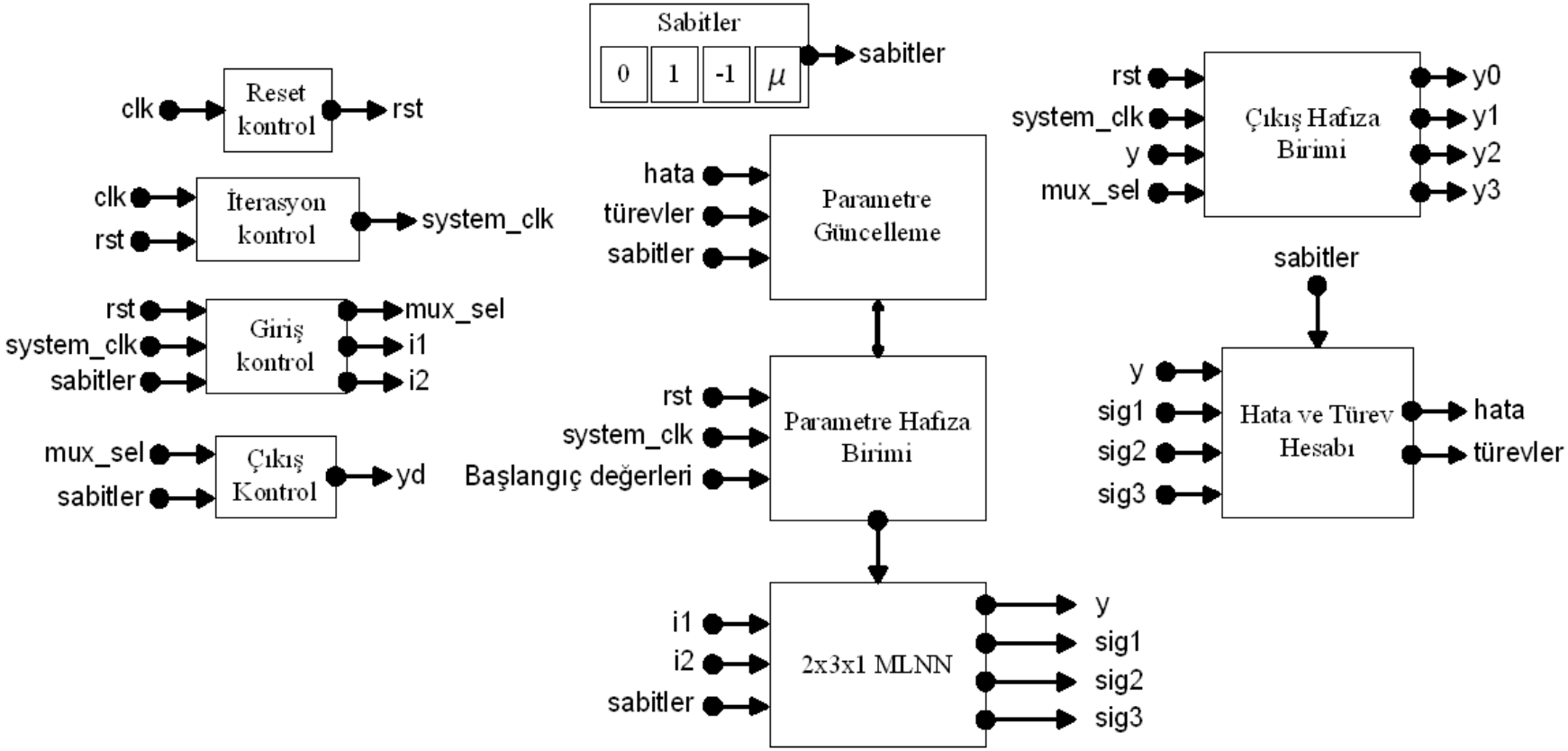


Şekil 5.49. MLNN'nin blok gösterimi

2x3x1 boyutlu MLNN ve eklenen eğitim algoritması blokları Şekil 5.50'de gösterilmektedir.

Ağın çalışmaya sağlıklı bir biçimde başlayabilmesi için başlangıçta bütün elemanlarının "reset" adı verilen bir bit yardımıyla sıfırlanması gerekmektedir. Bu amaçla kullanılan "reset kontrol birimi" aşağıdaki VHDL kod dizini ile oluşturulmuştur.

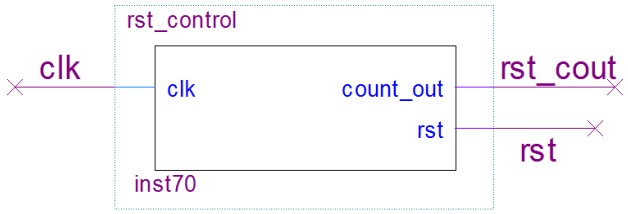
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY rst_control IS
PORT (clk : IN STD_LOGIC;
        count_out : OUT STD_LOGIC;
        rst : OUT STD_LOGIC);
END rst_control;
```



Şekil 5.50. MLNN ve eğitim algoritması blokları

```
ARCHITECTURE arc OF rst_control IS
signal tmp: std_logic;
        begin
        process (clk)
        begin
                if (clk'event and clk='1') then
                tmp <= '0';
                rst <='1';
                        if (tmp='0') then
                        rst <='0';
                        end if;
                end if;
        end process;
count_out <= tmp;
END arc;
```

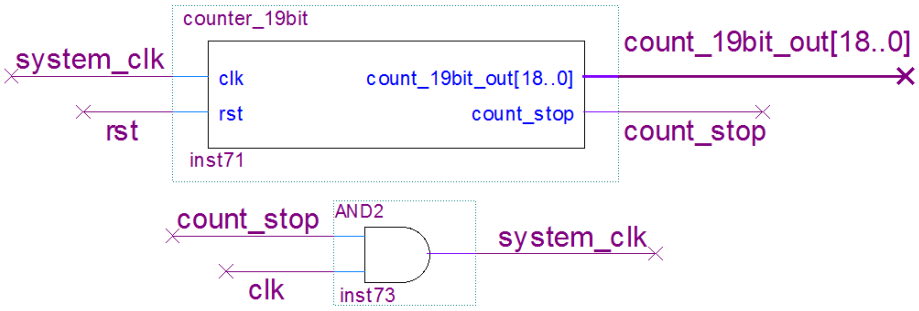
Sistemi ilk saat darbesi (clock) ile sıfırlayan reset kontrol biriminin blok gösterimi Şekil 5.51'de verilmiştir.



Şekil 5.51. Reset kontrol birimi

Reset kontrol birimi basit bir sayıcı gibi davranmaktadır. Clock sinyali ile "rst" çıkışı 'lojik 1' olur ve tüm sistemin sıfırlanması sağlanır. İkinci clock sinyali ile "rst" biti 'lojik 0' olur ve sistem çalışmaya başlayarak azami öğrenme sayısı (iterasyon sayısı) süresince çalışmasını sürdürür.

2X3X1 boyutlu ağın eğitimi iterasyon sayısına kadar sürdürülmektedir. Bu amaçla kullanılan "iterasyon kontrol birimi" Şekil 5.52'de gösterildiği gibi 19 bitlik sayıcı birimi ve bir adet lojik "AND" kapısından meydana gelmektedir.



Şekil 5.52. İterasyon kontrol birimi

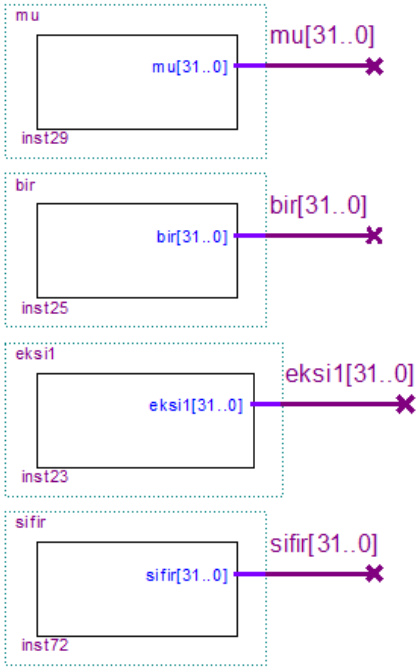
19 bitlik sayıcı, iterasyon sayısı olarak belirlenen “500000” değerine kadar saymaktadır. Bu değer “count_19bit_out[18..0]” çıkışından görülebilmektedir. 19 bitlik sayıcı aşağıdaki VHDL kod dizini ile oluşturulmuştur.

```
library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity counter_19bit is
        port(clk : in  std_logic;
             rst : in  std_logic;
             count_19bit_out : out std_logic_vector(18 downto 0);
             count_stop : out std_logic );
end counter_19bit;
architecture archi of counter_19bit is
        signal tmp: std_logic_vector(18 downto 0);
        begin
        process (clk, rst)
        begin
                if (rst='1') then
                tmp <= "0000000000000000000";
                count_stop <='1';
                elsif (clk'event and clk='1') then
                tmp <= tmp + 1;
                if (tmp = "1111010000100100000") then
                count_stop <='0';
```

```
            end if;
            end if;
        end process;
count_19bit_out <= tmp;
end archi;
```

19 bitlik sayıcının diğer çıkışı olan “count_stop” biti clock sinyali (clk) ile birlikte bir “AND” kapısına giriş olarak verilir. AND kapısı “count_stop” bitine bağlı olarak saat darbesi üretir (system_clk) ve “system_clk” çıkışı tüm sisteme clock sinyali olarak verilir. İterasyon sayısı azami değerine ulaştığında “count_stop” çıkışı ‘lojik 0’ değerini alır. Böylece “system_clk” çıkışı da ‘lojik 0’ olur ve tüm sistemin azami öğrenme sayısına ulaşıldığında durması sağlanır.

Sistemde kullanılan bir, sıfır, eksi bir ve öğrenme katsayısı (µ) Şekil 5.53’te gösterildiği gibi sabit olarak tanımlanmışlar ve sisteme gömülmüşlerdir.

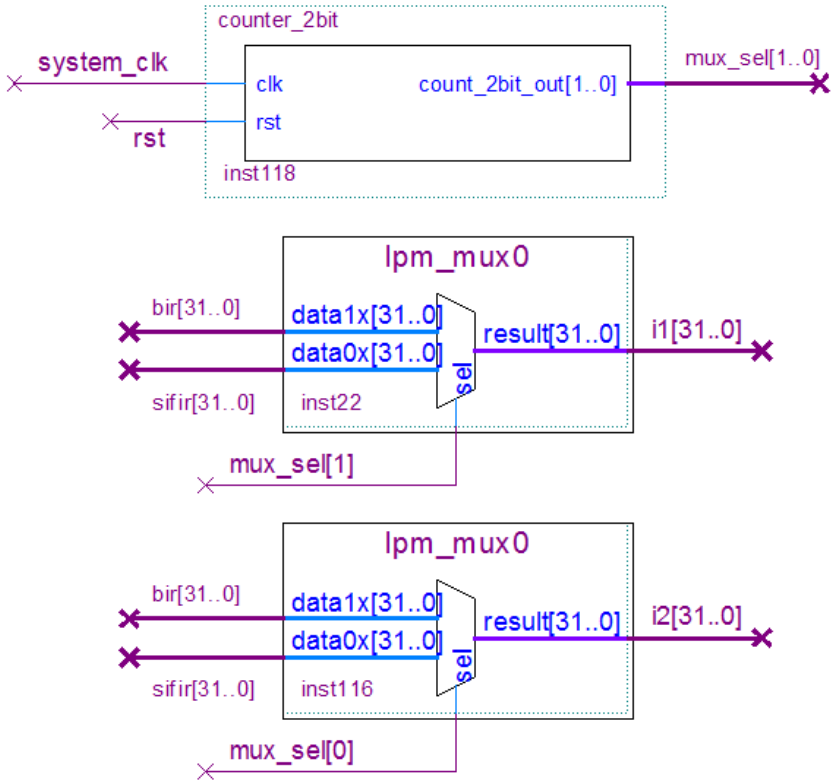


Şekil 5.53. Sistemde kullanılan sabitler

Eğitimde kullanılan öğrenme katsayısı (µ) “0.1” olarak belirlenmiş ve aşağıdaki VHDL kod dizini ile oluşturulmuştur.

```
library IEEE;
use ieee.std_logic_1164.all;
entity mu is
        port(mu:out std_logic_vector(31 downto 0));
end mu;
architecture mu of mu is
        begin
        mu<="00111101110011001100110011001101";
end mu;
```

Ağa verilecek girişleri kontrol etmek amacıyla Şekil 5.54'te gösterilen "giriş kontrol birimi" tasarlanmıştır. Bu birim "system_clk" girişine bağlı olarak çalışan bir adet 2 bitlik sayıcı ve 2 adet 2x1 Mux elemanlarından meydana gelmektedir. Böylece Tablo 5.9'da verilen XOR kapısı doğruluk tablosuna göre girişler ağa sırasıyla ve kontrollü bir şekilde verilmiştir.



Şekil 5.54. Giriş kontrol birimi

Tablo 5.9. XOR kapısının doğruluk tablosu

Girişler		**i1 XOR i2**
i1	**i2**	
0	0	0
0	1	1
1	0	1
1	1	0

2 bitlik sayıcı aşağıdaki VHDL kod dizini ile oluşturulmuştur.

```
library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity counter_2bit is
        port (clk, rst : in  std_logic;
              count_2bit_out : out std_logic_vector(1 downto 0));
end counter_2bit;
architecture archi of counter_2bit is
        signal tmp: std_logic_vector(1 downto 0);
        begin
        process (clk, rst)
        begin
                if (rst='1') then
                tmp <= "00";
                elsif (clk'event and clk='1') then
                tmp <= tmp + 1;
                end if;
        end process;
count_2bit_out <= tmp;
end archi;
```

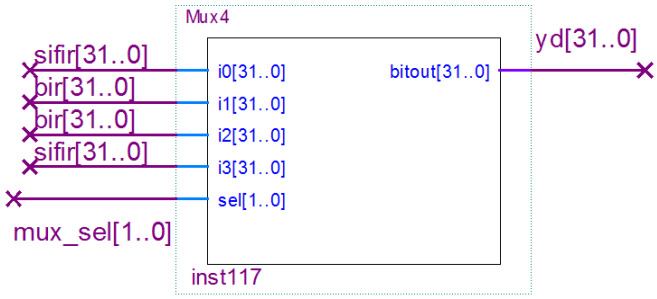
2 bitlik sayıcı her saat darbesinde 2 bitlik çıkış (mux_sel[1..0]) üretmektedir. “mux_sel” bitleri 2x1 Mux elemanlarına seçici bit olarak verilir. 2x1 Mux elemanları seçici bite bağlı olarak girişlerindeki “sıfır” ya da “bir” değerlerinden birisini seçerek

MLNN'nin girişine uygulanacak "i1" ve "i2" girişlerini oluştururlar. 2 bitlik sayıcı çıkışları ve bunlara karşılık gelen 2x1 Mux çıkışları Tablo 5.10'da verilmiştir.

Tablo 5.10. 2 bitlik sayıcı çıkışlarına karşılık üretilen 2x1 Mux çıkışları

2 Bitlik Sayıcı Çıkışı		**2x1 Mux Çıkışları**	
mux_sel[1]	**mux_sel[0]**	**i1**	**i2**
0	0	0	0
0	1	0	1
1	0	1	0
1	1	1	1

MLNN kendisine verilen girişlere karşılık bir çıkış üretecektir. Gerçek çıkış ile MLNN'nin ürettiği çıkış arasındaki fark hatayı vermektedir. Hatanın hesaplanabilmesi için verilen girişlere karşı üretilmesi gereken gerçek çıkışın belirlenmesi gerekmektedir. Bu amaçla tasarlanan "çıkış kontrol birimi" Şekil 5.55'te gösterilmektedir. Çıkış kontrol birimi, "mux_sel" seçici bitlerini kullanarak gerçek çıkışı (yd) oluşturan 4x1 mux elemanından oluşmaktadır.



Şekil 5.55. Çıkış kontrol birimi

4x1 mux elemanı aşağıdaki VHDL kod dizini ile oluşturulmuştur.

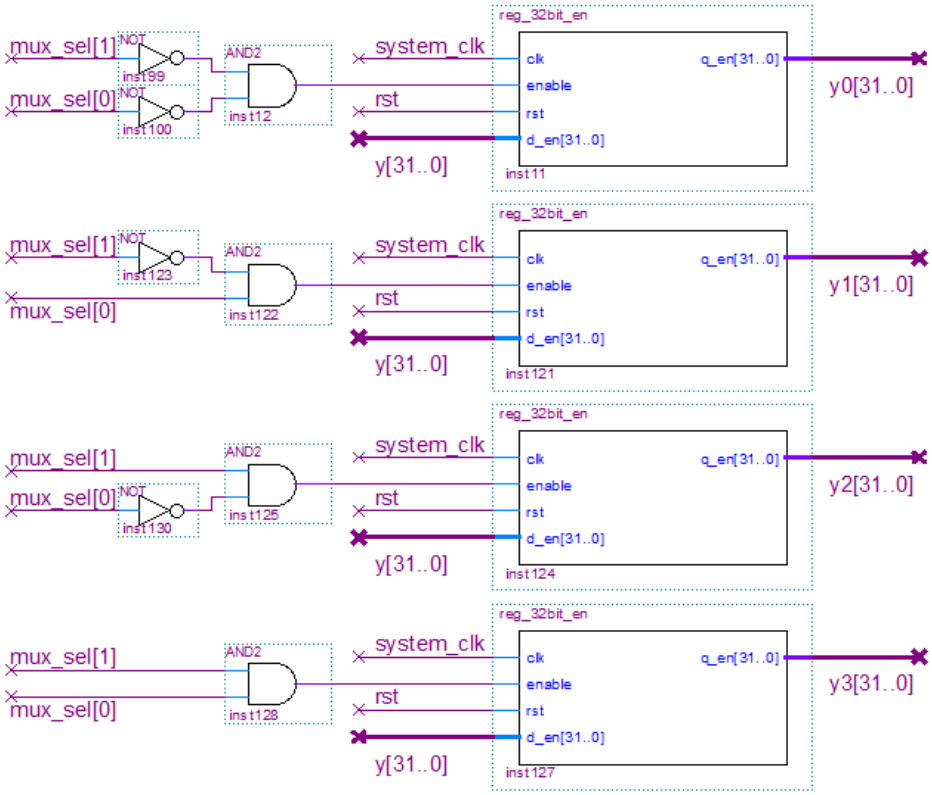
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Mux4 is
port ( i0 : in std_logic_vector(31 downto 0);
```

```
        i1 : in std_logic_vector(31 downto 0);
        i2 : in std_logic_vector(31 downto 0);
        i3 : in std_logic_vector(31 downto 0);
        sel : in std_logic_vector(1 downto 0);
        bitout : out std_logic_vector(31 downto 0));
end Mux4;
architecture Behavioral of Mux4 is
begin
process(i0,i1,i2,i3,sel)
begin
  case sel is
    when "00" => bitout <= i0;
    when "01" => bitout <= i1;
    when "10" => bitout <= i2;
    when others => bitout <= i3;
  end case;
end process;
end Behavioral;
```

MLNN'nin ürettiği çıkışlar hafıza elemanlarında (register) saklanmaktadır. XOR kapısının giriş değerlerine karşılık üretilen çıkışların saklandığı hafıza elemanları, hafıza elemanlarının çıkışlarına göre etiketlenerek Tablo 5.11'de verilmiştir. Bu amaçla Şekil 5.56'da gösterilen 4 adet 32 bitlik hafıza elemanı tasarlanmıştır.

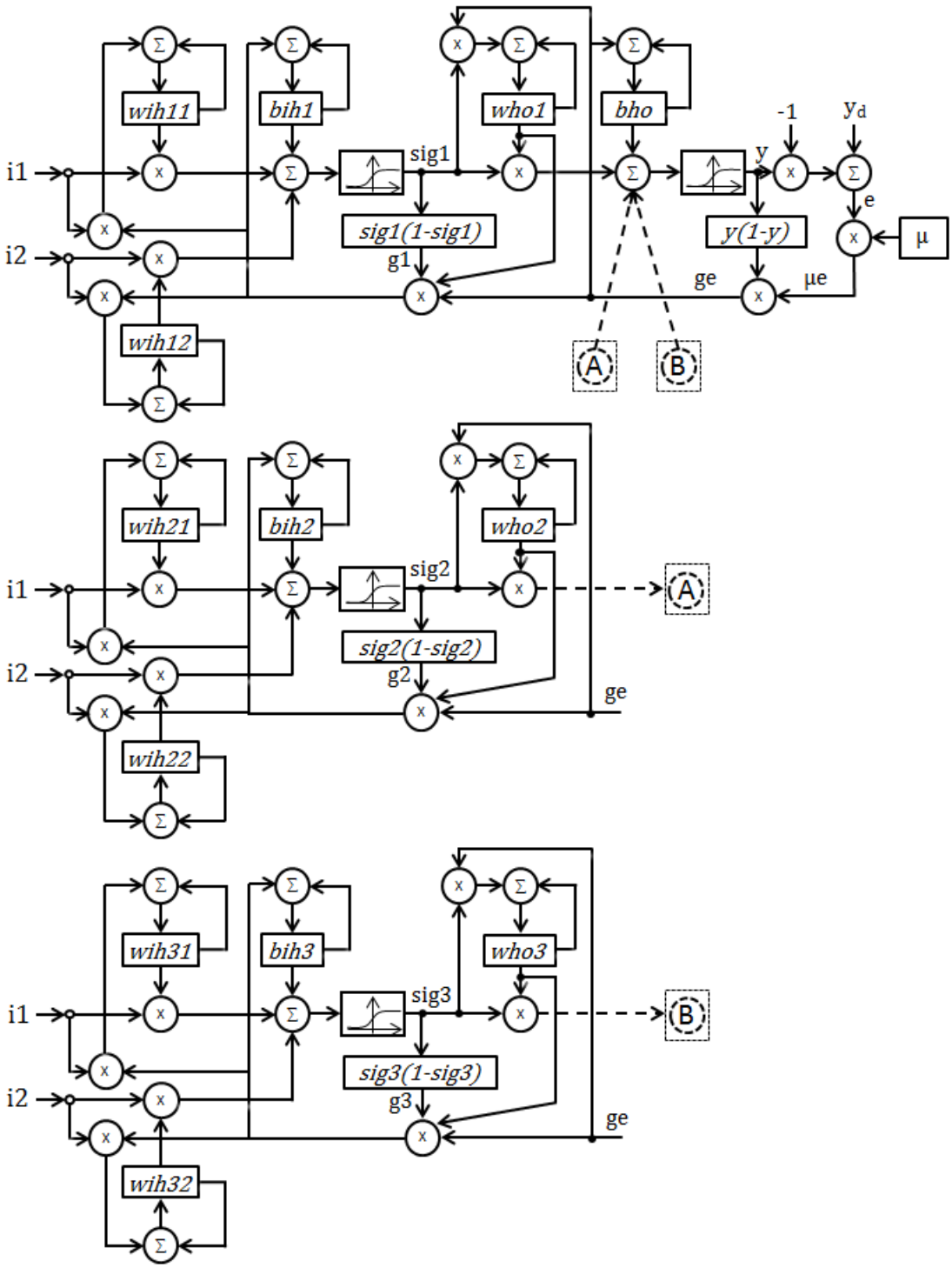
Tablo 5.11. Çıkışların saklandığı hafıza elemanları

Girişler		**i1 XOR i2**	**Çıkışın Saklandığı Hafıza Elemanı**
i1	**i2**		
0	0	0	y0[31..0]
0	1	1	y1[31..0]
1	0	1	y2[31..0]
1	1	0	y3[31..0]



Şekil 5.56. Çıkışların saklandığı hafıza elemanları

MLNN eğitiminde basit yapısı sebebiyle geri yayılım algoritması kullanılmıştır. Geri yayılım algoritmasına göre sistem parametrelerinin güncellenmesi Şekil 5.57'de gösterilmiştir.



Şekil 5.57. Geri yayılım algoritması ile sistem parametrelerinin güncellenmesi

Şekil 5.57'ye göre sistem hatası ve türevler aşağıdaki formüllerle hesaplanmaktadır. (Denklem 5.3)

$$
\begin{aligned}
&e = yd - y \\
&G = y(1-y) \\
&G1 = sig1(1-sig1) \\
&G2 = sig2(1-sig2) \\
&G3 = sig3(1-sig3) \\
&ge = e \times \mu \times G
\end{aligned}
\qquad \textbf{(5.3)}
$$

Burada *e* hatayı, *y* sistem çıkışını, *yd* gerçek çıkışı, μ öğrenme katsayısını, *G* çıkışın türevini, *sig1*, *sig2*, *sig3* gizli katman aktivasyon fonksiyonları çıkışlarını ve *G1*, *G2*, *G3* ise gizli katman aktivasyon fonksiyonlarına ait çıkışların türevlerini ifade etmektedir. Çıkış katmanına ait eşik ve ağırlık parametreleri aşağıdaki şekilde güncellenmektedir. (Denklem 5.4)

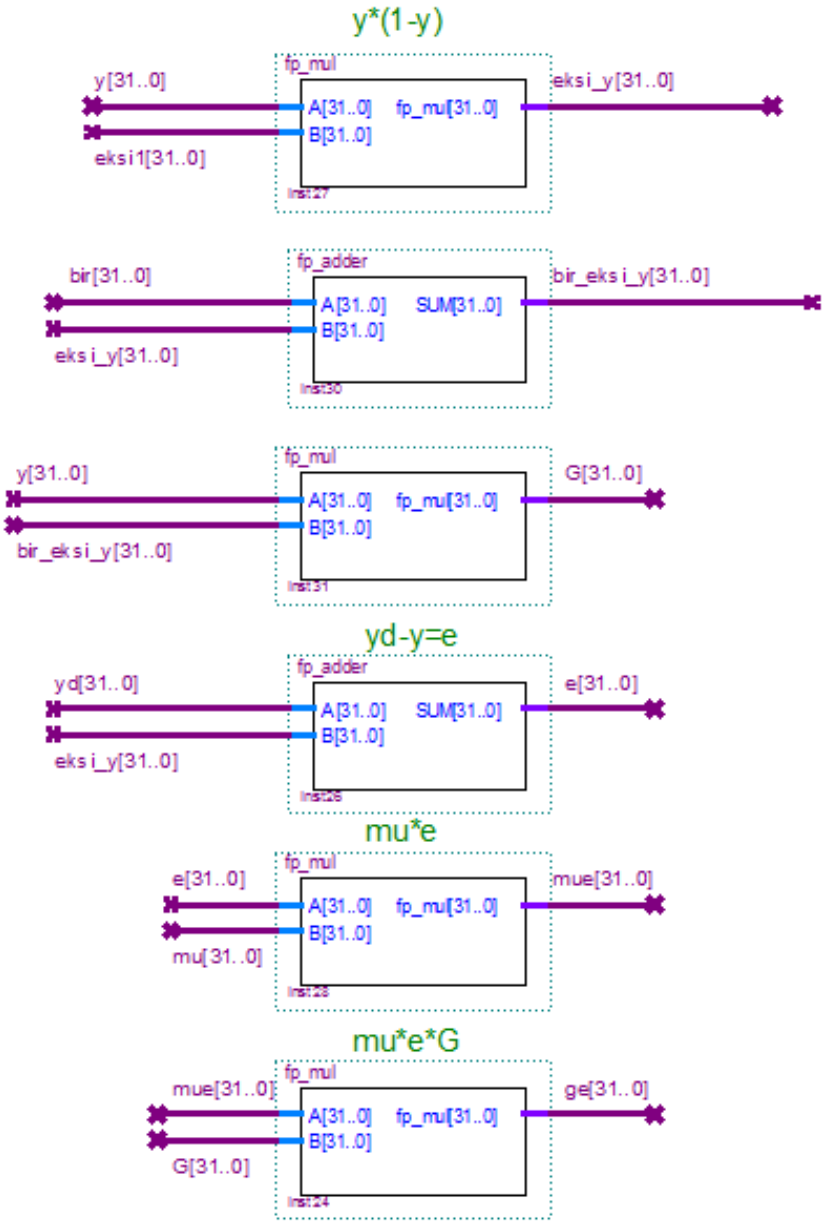
$$
\begin{aligned}
&bho^{n+1} = bho^{n} + ge^{n} \\
&who1^{n+1} = who1^{n} + ge^{n} \times sig1^{n} \\
&who2^{n+1} = who2^{n} + ge^{n} \times sig2^{n} \\
&who3^{n+1} = who3^{n} + ge^{n} \times sig3^{n}
\end{aligned}
\qquad \textbf{(5.4)}
$$

Burada *n+1* bir sonraki güncel değeri ifade etmektedir. Gizli katman parametreleri ise aşağıda verilen denklemler kullanılarak güncellenmektedir. (Denklem 5.5)

$$
\begin{aligned}
&bih1^{n+1} = bih1^{n} + ge^{n} \times who1^{n} \times G1^{n} \\
&bih2^{n+1} = bih2^{n} + ge^{n} \times who2^{n} \times G2^{n} \\
&bih3^{n+1} = bih3^{n} + ge^{n} \times who3^{n} \times G3^{n} \\
&wih11^{n+1} = wih11^{n} + ge^{n} \times who1^{n} \times G1^{n} \times i1^{n} \\
&wih12^{n+1} = wih12^{n} + ge^{n} \times who1^{n} \times G1^{n} \times i2^{n} \\
&wih21^{n+1} = wih21^{n} + ge^{n} \times who2^{n} \times G2^{n} \times i1^{n} \\
&wih22^{n+1} = wih22^{n} + ge^{n} \times who2^{n} \times G2^{n} \times i2^{n} \\
&wih31^{n+1} = wih31^{n} + ge^{n} \times who3^{n} \times G3^{n} \times i1^{n} \\
&wih32^{n+1} = wih32^{n} + ge^{n} \times who3^{n} \times G3^{n} \times i2^{n}
\end{aligned}
\qquad \textbf{(5.5)}
$$

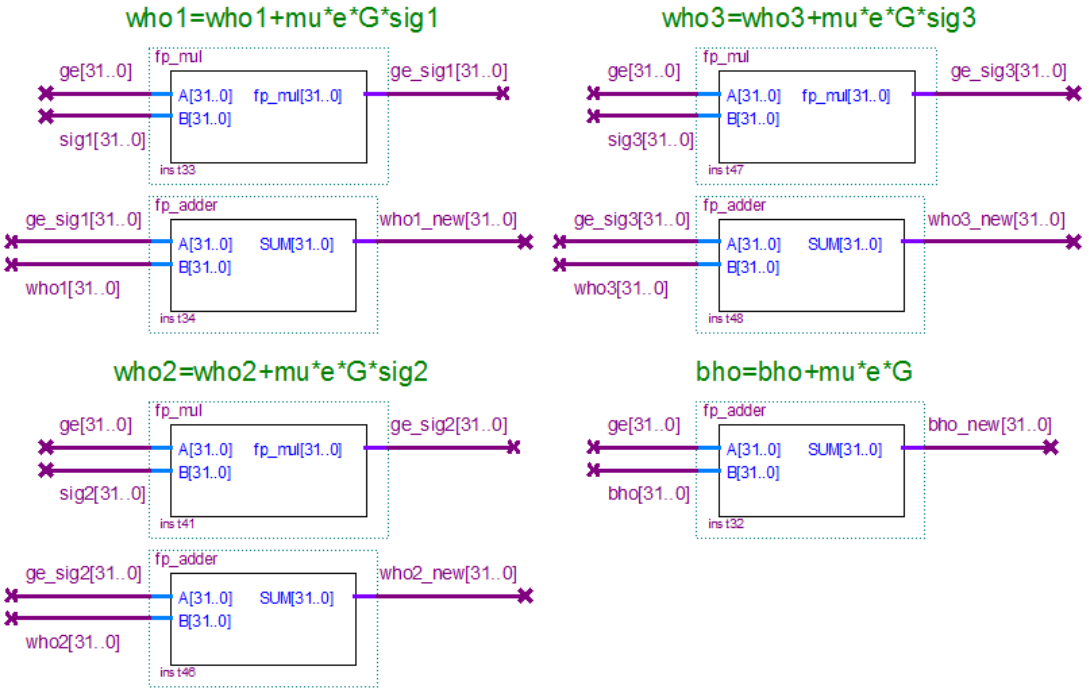
Burada *i1* ve *i2* giriş değerlerini ifade etmektedir.

Hata hesabı, çıkışın türevinin hesaplanması, türevin eğitim katsayısı ve hata ile çarpılmasını içeren işlem basamakları Şekil 5.58'de gösterilmiştir.



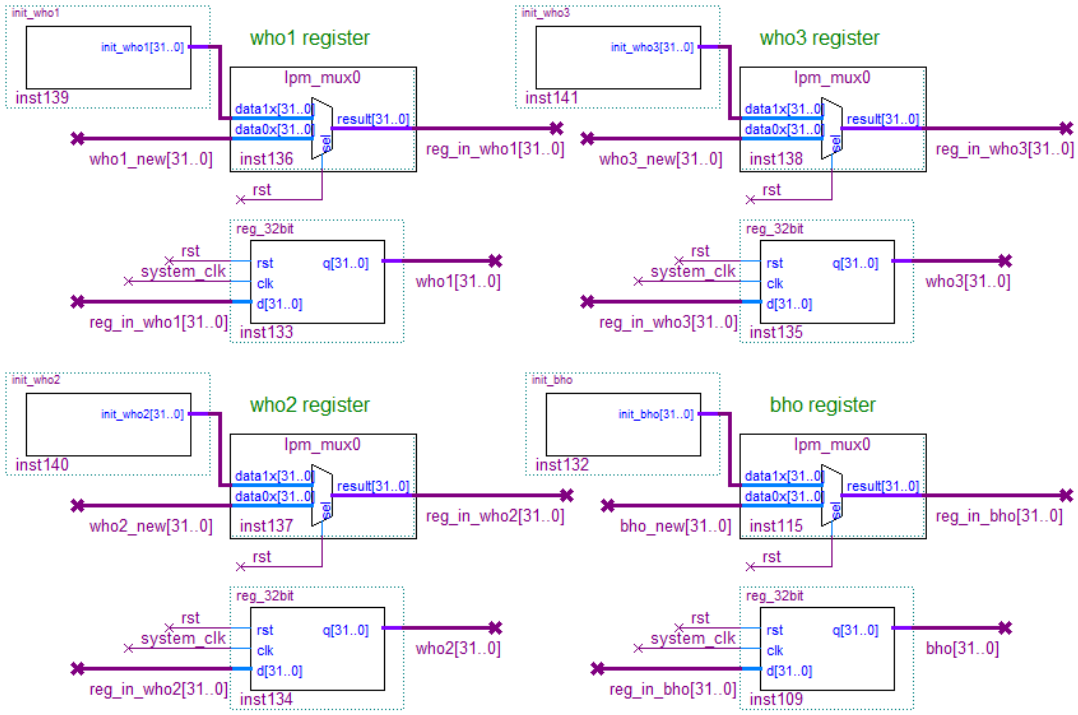
Şekil 5.58. Geri yayılım algoritmasında çıkışın türevi ve hata hesabı

Gizli katman ve çıkış katmanı arasındaki ağırlık ve eşik parametrelerinin güncellenmesi, "parametre güncelleme birimi" içerisinde yer alan Şekil 5.59'da verilen devreler ile gerçekleştirilmektedir.



Şekil 5.59. Gizli katman ve çıkış katmanı arasındaki parametrelerin güncellenmesi

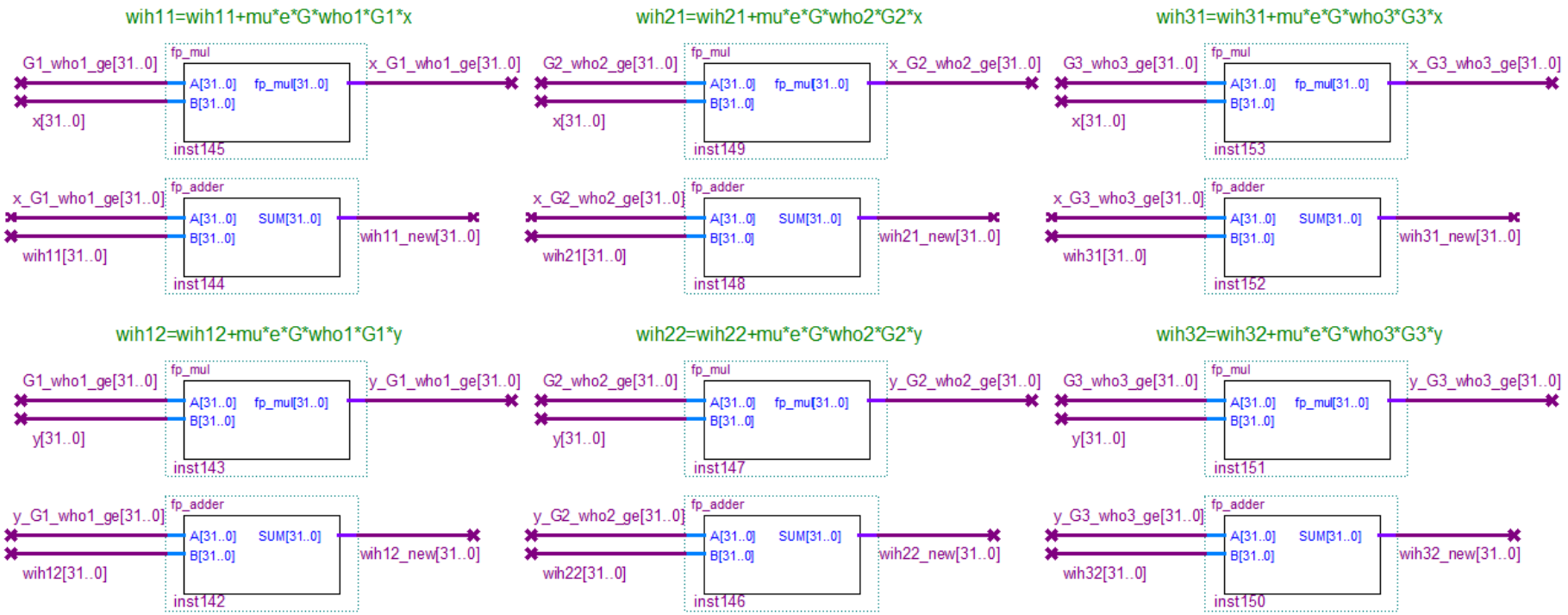
Gizli katman ve çıkış katmanı arasındaki parametreler güncellendikten sonra Şekil 5.60'ta verilen ve "parametre hafıza birimi" içerisinde yer alan hafıza elemanlarında saklanmaktadır.



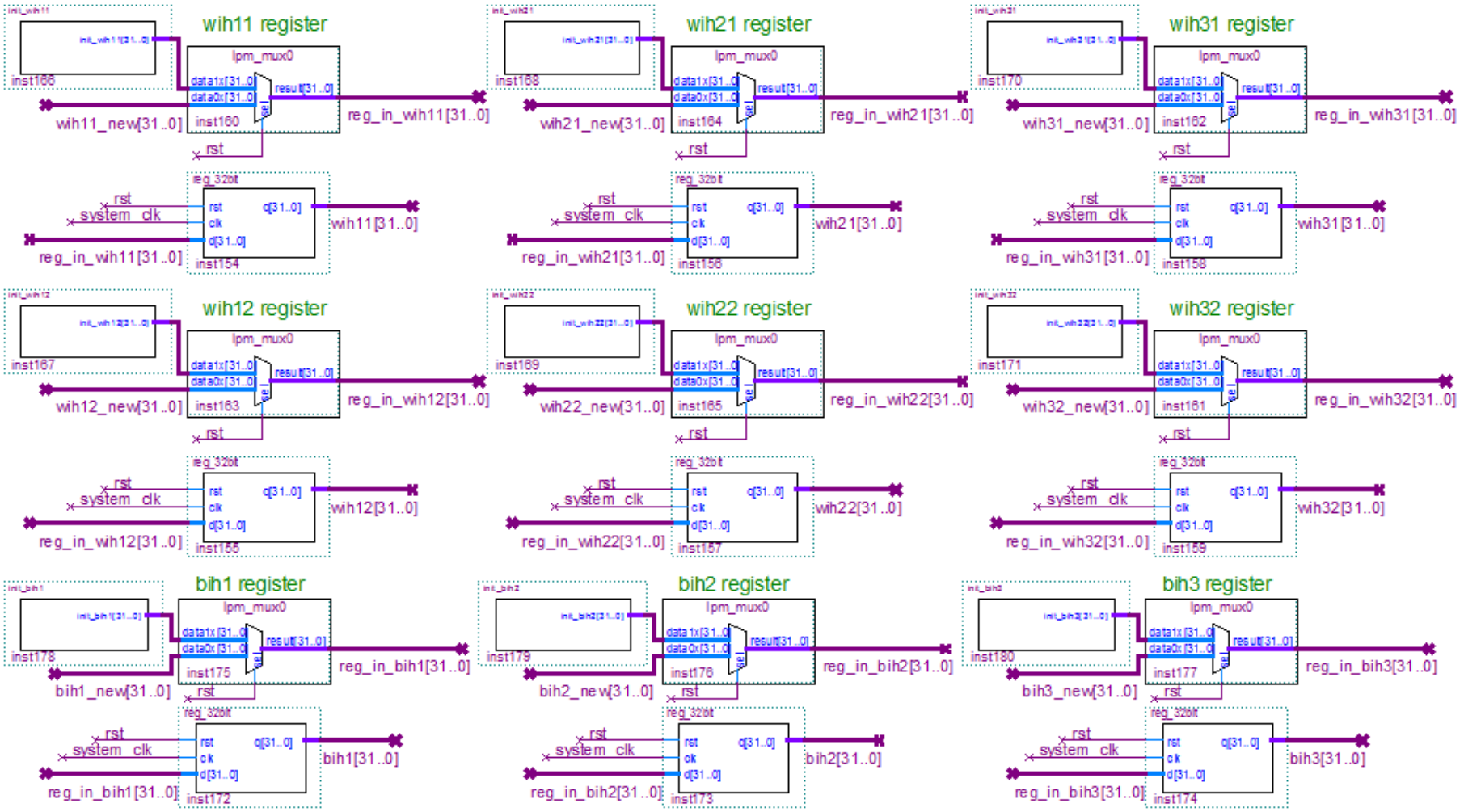
Şekil 5.60. Çıkış katmanı parametrelerinin saklandığı hafıza elemanları

Parametre hafıza elemanlarından her biri başlangıç değerlerine sahiptir ve 2x1 Mux ile 32 bitlik register elemanlarından meydana gelmektedir. Sistem ilk çalışmaya başladığında bu parametreler ağın başlangıç parametreleri olarak kullanılmaktadır. Her iterasyonda parametreler geri yayılım algoritması ile güncellenmekte ve hafız birimlerine kaydedilmektedir.

Giriş katmanı ve gizli katman arasındaki ağırlık ve eşik parametrelerinin güncellenmesi, “parametre güncelleme birimi” içerisinde yer alan ve Şekil 5.61’de gösterilen devreler ile gerçekleştirilmektedir. Parametreler güncellendikten sonra Şekil 5.62’de verilen ve “parametre hafıza birimi” içerisinde yer alan hafıza elemanlarında saklanmaktadır.



Şekil 5.61. Giriş katmanı ve gizli katman arasındaki parametrelerin güncellenmesi



Şekil 5.62. Gizli katman parametrelerinin saklandığı hafıza elemanları

BÖLÜM 6. GERÇEKLENEN YSA DONANIMININ TEST EDİLMESİ

FPGA üzerinde donanımsal olarak gerçeklenen 2x3x1 boyutlu MLNN modeli, iki ayrı problem üzerinde test edilmiştir.

İlk olarak 109E234 numaralı TÜBİTAK projesi kapsamında elde edilen tıbbi veriler kullanılmıştır [144]. Bu çalışmada MLNN'nin eğitimi MATLAB (Lisans No: 834260) programında gerçekleştirilmiştir. Elde edilen ağ parametreleri MLNN donanımına gömülmüş ve sonuçlar karşılaştırmalı olarak verilmiştir.

İkinci olarak XOR (Özel Veya) problemi ele alınmıştır. XOR kapısı doğrusal olmayan bir yapıya sahiptir. Bu çalışmada ağın başlangıç parametreleri MATLAB programı ile belirlenmiş ve sisteme gömülmüştür. Ağın eğitimi eğitilebilir MLNN donanımı üzerinde gerçekleştirilmiştir.

6.1. HbA1C ve Kan Glikoz Seviyesinin Sınıflandırması

Sistem doğruluğunu test etmek amacıyla 109E234 numaralı TÜBİTAK projesi kapsamında elde edilen tıbbi veriler kullanılmıştır [144]. Veriler, Kütahya Dumlupınar Üniversitesi Hastanesi'nde, 21 °C oda sıcaklığında 297 gönüllüden alınmıştır. Saraoğlu ve arkadaşlarının yaptığı çalışmada diyabet teşhisi için farklı sayıda öznitelik kullanılarak (2, 4 ve 10), YSA ile HbA1C (Glikohemoglobin) ve kan glikoz seviyesinin sınıflandırması yapılmıştır. Sınıflandırma için MLNN, Elman ve radyal tabanlı YSA kullanılmıştır [144].

Bu tez çalışmasında, 2 öznitelik için, 224 gönüllüye ait veriler MLNN ağının eğitiminde kullanılmıştır. Ağın eğitimi MATLAB programında gerçekleştirilmiş,

ağırlık (weight) ve eşik (bias) parametreleri belirlenmiştir. HbA1C ve kan glikoz seviyesine ait parametreler sırasıyla Tablo 6.1 ve Tablo 6.2'de verilmiştir.

Tablo 6.1. HbA1C için ağırlık ve eşik değerleri

Ağırlık/Eşik	Desimal	32-bit kayan noktalı gösterim
Wih11	-7.25030	11000000111010000000001001110101
Wih12	-15.9684	11000001011111110111111010010001
Wih21	6.77780	01000000110110001110001110111101
Wih22	-15.6395	11000001011110100011101101100100
Wih31	18.0427	01000001100100000101011101110011
Wih32	-17.4391	11000001100010111000001101000111
Who1	-0.68920	10111111001100000110110101110010
Who2	1.08440	00111111100010101100110110011111
Who3	-1.82160	10111111111010010010101000110000
Bih1	16.2951	01000001100000100101110001011101
Bih2	9.61970	01000001000110011110101001001011
Bih3	6.15400	01000000110001001110110110010001
Bho	1.52370	00111111110000110000100010011010

Tablo 6.2. Kan glikoz seviyesi için ağırlık ve eşik değerleri

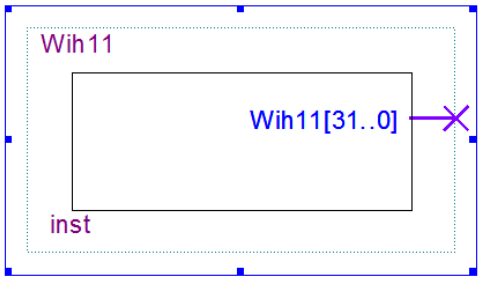
Ağırlık/Eşik	Desimal	32-bit kayan noktalı gösterim
Wih11	15.2646	01000001011101000011101111001101
Wih12	-2.75240	11000000001100000010011101010010
Wih21	-13.9629	11000001010111110110100000001010
Wih22	4.54690	01000000100100011000000000110100
Wih31	12.9314	01000001010011101110011100000100
Wih32	-6.47000	11000000110011110000101000111101
Who1	3.26080	01000000010100001011000011110010
Who2	1.59520	00111111110011000010111110000011
Who3	-0.16310	10111110001001110000001110110000
Bih1	-11.4381	11000001001101110000001001110101
Bih2	8.61620	01000001000010011101101111110101
Bih3	2.50320	01000000001000000011010001101110
Bho	-2.32190	11000000000101001001101000000010

Tablo 6.1 ve Tablo 6.2'de gösterilen parametreler ile MLNN donanımı test edilmiştir. MLNN donanımını test etmek amacıyla eğitimde elde edilen parametreler sabit olarak tanımlanıp Şekil 5.47'de gösterilen MLNN mimarisine eklenmiştir.

Örneğin HbA1C için Wih11 ağırlığını sabit olarak tanımlayan VHDL kodu şu şekildedir;

```
library IEEE;
use ieee.std_logic_1164.all;
entity Wih11 is
        port( Wih11:out std_logic_vector(31 downto 0));
end Wih11;
architecture value of Wih11 is
begin
        Wih11<="11000000111010000000001001110101";
end value;
```

VHDL kodu üzerinde "Wih11" adı ve 8. satırda yer alan 32 bitlik parametre değeri, diğer ağırlık ve eşik değerleri için değiştirilerek diğer parametreler için de sabit değerler oluşturulur. Daha sonra program derlenir ve bu değerler için sembol dosyaları oluşturularak MLNN ağına eklenir. "Wih11" ağırlığı için elde edilen sembol dosyası Şekil 6.1'de gösterilmiştir.



Şekil 6.1. Wih11 ağırlığı için oluşturulan sembol dosyası

Ağırlıklar ve eşik değerleri devreye eklendikten sonra, giriş değerleri MLNN donanımına verilir ve çıkış elde edilir. MLNN donanımı verilen giriş değerleri için saat darbesinden bağımsız olarak çıkış değerini üretir. Bunun sebebi, tasarlanan

donanım sayesinde giriş değerleri çıkışa doğru işlenerek ilerlerken toplama, çarpma gibi işlemler esnasında herhangi bir kaydırmalı yazmaç kullanılmamasıdır. Çıkış üretilirken yalnızca kapı gecikmeleri söz konusudur.

MLNN modelinin hatası aşağıdaki formül ile hesaplanmıştır [144]. (Denklem 6.1)

$$E = \frac{1}{|n_{test}|}\sum_{1}^{n_{test}}\left(\left|\frac{Q_p - Q_t}{Q_t}\right|\right) \quad \forall Q_t \neq 0 \tag{6.1}$$

Burada E ortalama mutlak hata, n_{test} örnek sayısı, Q_p sistemin çıkış değeri ve Q_t gerçek çıkış değerini ifade etmektedir. MLNN donanımı [144] numaralı referanstaki yazılımsal MLNN modelleri ile karşılaştırılmış ve hata değerleri Tablo 6.3'te gösterilmiştir.

Tablo 6.3. MLNN modellerinin karşılaştırılması

Çalışma	Model	Öznitelik Sayısı	Glikoz Seviyesi Hatası (%)	HbA1C Hatası (%)
[144]	MLNN	10	27.65	16.79
[144]	MLNN	4	27.86	16.50
[144]	MLNN	2	30.36	17.74
Bu Çalışma	**MLNN (FPGA)**	**2**	**30.83**	**17.97**

Tablo 6.3'ten yola çıkarak, donanımsal MLNN modeli diğer yazılımsal MLNN modellerine yakın ve kabul edilebilir sonuçlar vermiştir. Öznitelik sayısı artırılarak hatanın daha da azaltılması mümkündür. Fakat öznitelik sayısının artması MLNN donanımının kullanacağı işlemci elemanları ve dolayısıyla kaynak kullanımını da artıracaktır. Ağ daha da büyüyecek, karmaşık hale gelecek ve donanımsal olarak gerçeklenmesi zorlaşacaktır.

6.2. XOR Problemi

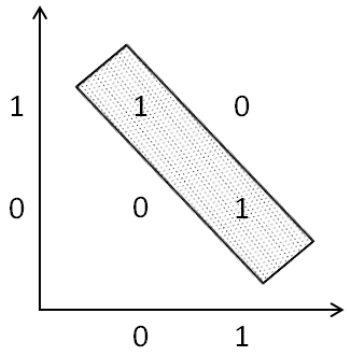
Bu çalışmada gerçeklenen eğitilebilir MLNN donanımı, doğrusal olmayan yapısı sebebiyle sınıflandırma ve modelleme uygulamalarında sıklıkla kullanılan

XOR problemi ile test edilmiştir. XOR kapısının doğruluk tablosu Tablo 6.4'te verilmiştir.

Tablo 6.4. XOR kapısının doğruluk tablosu

i1	**i2**	**i1 XOR i2**
0	0	0
0	1	1
1	0	1
1	1	0

XOR kapısı, Tablo 6.4'ten de anlaşılacağı üzere farklı girişler için lojik 1 çıkışı üretmektedir. Şekil 6.2 ile verilen XOR kapısının iki boyutlu gösteriminden anlaşılabileceği gibi lojik 0 ve lojik 1 değerlerini tek bir doğru ile birbirinden ayırmak mümkün değildir. Problemin doğrusal olmayan yapısı buradan ileri gelmektedir.



Şekil 6.2. XOR kapısının iki boyutlu gösterimi

Bu çalışmada MLNN'nin başlangıç parametreleri MATLAB programı ile belirlenmiş ve sisteme gömülmüştür. Başlangıç parametreleri ve bunların 32 bit kayan noktalı gösterimleri Tablo 6.5'te verilmiştir.

Tablo 6.5'te verilen başlangıç parametreleri donanıma gömüldükten sonra ağın eğitimi eğitilebilir MLNN donanımı üzerinde gerçekleştirilmiştir. Eğitim sonrası elde edilen parametre değerleri Tablo 6.6'da verilmiştir.

Tablo 6.5. XOR problemi için başlangıç parametreleri

Ağırlık/Eşik	Onluk	32-bit kayan noktalı gösterim
Wih11	0.087008	00111101101100100011000101000000
Wih12	0.944060	00111111011100011010110111101010
Wih21	0.751213	00111111010000000100111101111110
Wih22	0.200476	00111110010011010100100110010100
Wih31	0.724113	00111111001110010101111101111000
Wih32	0.587237	00111111000101100101010100101001
Who1	0.050874	00111101010100000110000101000001
Who2	0.935850	00111111011011111001001111011101
Who3	0.648091	00111111001001011110100101001010
Bih1	0.694296	00111111001100011011110101100001
Bih2	0.960845	00111111011101011111100111110000
Bih3	0.198889	00111110010010111010100110001110
Bho	0.322306	00111110101001010000010101001010

Tablo 6.6. XOR problemi için eğitim sonrası elde edilen parametreler

Ağırlık/Eşik	Onluk	32-bit kayan noktalı gösterim
Wih11	5.115610	01000000101000111011001100010100
Wih12	5.147220	01000000101001001011011000000111
Wih21	2.189950	01000000000011000010100000100100
Wih22	2.744040	01000000001011111001111001011010
Wih31	6.662560	01000000110101010011001110110001
Wih32	6.737890	01000000110101111001110011001011
Who1	-12.15630	11000001010000101000000000110100
Who2	0.290626	00111110100101001100110011101110
Who3	11.39990	01000001001101100110010111111110
Bih1	-7.876100	11000000111111000000100100000011
Bih2	0.317062	00111110101000100101010111110011
Bih3	-3.103250	11000000010001101001101110100110
Bho	-5.544670	11000000101100010110110111110000

XOR problemi için eğitilebilir MLNN donanımı ile elde edilen sonuçlar ve gerçek değerler Tablo 6.7'de karşılaştırmalı olarak sunulmuştur.

Tablo 6.7. XOR problemi için gerçek çıkış ile eğitilebilir MLNN çıkışının karşılaştırılması

GİRİŞLER		**GERÇEK ÇIKIŞ**	**MLNN ÇIKIŞI**
i1	**i2**	**i1 XOR i2**	**i1 XOR i2**
0	0	0	0.007315755
0	1	1	0.993824960
1	0	1	0.993783500
1	1	0	0.006700397

Tablo incelendiğinde, eğitilebilir MLNN donanımının çıkışının gerçek değerlere oldukça yaklaştığı ve kabul edilebilir olduğu görülmektedir.

BÖLÜM 7. SONUÇ VE ÖNERİLER

Yapay Sinir Ağları (YSA'lar), biyolojik sinir sistemine dayalı elektronik modellerdir. YSA'lar girişlerden gelen verileri işleyen birbirine bağlı yapay nöronlardan oluşmaktadır. Bu mimariler, yazılım ya da donanım olarak gerçekleştirilebilirler. Yazılım olarak tasarlanan YSA'ların avantajı, tasarımcının YSA bileşenlerinin iç işleyişini bilmesine gerek olmamasıdır. Ancak gerçek zamanlı uygulamalarda yazılım olarak tasarlanan YSA'lar istenen performansı gösterememektedir.

YSA hesaplamaları paralel olarak gerçekleştirilmektedir ve paralel işlem için özel donanım aygıtları gereklidir. Birçok alandan araştırmacılar alternatif uygulamalar üzerinde çalışmışlardır. Bu uygulamalar, YSA'ların paralel doğasından yararlanmak için farklı türde cihazlar üzerinde gerçekleştirilmiştir.

YSA'nın FPGA uygulamaları, yeniden yapılandırılabilir yapısı ve paralel mimarisi nedeniyle son yirmi yılda büyük ilgi uyandırmıştır. Bu tez çalışmasında, eğitilebilir YSA donanımı, Altera'nın FPGA tasarım programı QUARTUS II ile tasarlanmıştır. YSA modeli olarak literatürde en çok kullanılan modellerden biri olan MLNN seçilmiştir. FPGA üzerinde MLNN mimarisi gerçeklenirken, mevcut olan VHDL, Verilog, Şematik dizayn yöntemlerinden büyük ölçüde şematik dizayn kullanılmıştır. Şematik dizayn kullanılmasının sebebi tasarım aşamasında devrelerin iç yapısına mümkün olduğunca hakim olmaktır. İç yapının önemsenmediği basit ve küçük devrelerde ise devre tasarım ve tanımlama dili olan VHDL kullanılmıştır.

Hem YSA hem de FPGA paralel doğaya sahiptirler ve bu sebeple işlem hızları oldukça yüksektir. YSA, FPGA üzerinde gerçeklenirken kullanılan alt birimlerin paralel yapıya uygun işlem yapmaları gerekmektedir. Toplayıcı devrelerde kullanılan kaydırmalı yazmaçlar birden fazla saat darbesine ihtiyaç duydukları için paralel işlem yapmaya pek de uygun değillerdir.

Bu çalışmada, eğitilebilir MLNN donanımı FPGA üzerinde gerçeklenirken, hızdan ödün vermemek amacıyla temel elemanlar olan paralel bir çarpıcı ve saat darbesine ihtiyaç duymayan bir toplayıcı tamamen şematik olarak tasarlanmıştır. Gecikmeyi azaltmak amacıyla kaydırma işlemleri için farklı bir donanım gerçekleştirilmiş ve kaydırmalı yazmaçlar yerine üç-durumlu tampon serileri kullanılmıştır. Tasarlanan donanımda, girişlerinde kaydırılmış veriler bulunan üç-durumlu tampon serilerinden yalnız biri aktif hale getirilerek kaydırma işlemi gerçekleştirilmektedir. Üç-durumlu tampon serileri kullanıldığından kaydırma işlemi için saat darbesi gerekli değildir ve böylece sonuç tek bir çevrimde üretilir. Sonuç üretilirken sadece kapı gecikmeleri söz konusudur. Tasarlanan toplayıcı devre ve paralel çarpıcı kullanılarak aktivasyon fonksiyonu bloğu elde edilmiştir. Hassas hesaplamalar için IEEE 32-bit kayan noktalı nümerik formattan faydalanılmıştır.

Şematik olarak tasarlanan toplayıcı devre, paralel çarpıcı ve aktivasyon fonksiyonu bloğu ModelSim programı ile test edilmiş ve devrelerin doğru sonuç verdiği gözlenmiştir.

FPGA üzerinde donanımsal olarak gerçeklenen 2x3x1 boyutlu ve eğitilebilir MLNN modeli, iki ayrı problem üzerinde test edilmiştir.

İlk olarak 109E234 numaralı TÜBİTAK projesi kapsamında elde edilen tıbbi veriler kullanılmıştır [144]. Diyabet hastalığının teşhisi için HbA1C (Glikohemoglobin) ve kan glikoz seviyesinin sınıflandırması yapılmıştır. Bu problemde, 2 öznitelik için, 297 gönüllüden 224'üne ait veriler MLNN ağının eğitiminde kullanılmıştır. Ağın eğitimi MATLAB (Lisans No: 834260) programında gerçekleştirilmiş, ağırlık (weight) ve eşik (bias) parametreleri belirlenmiştir. Donanımsal MLNN modeli diğer yazılımsal MLNN modellerine yakın ve kabul edilebilir sonuçlar vermiştir. Öznitelik sayısı artırılarak hatanın daha da azaltılması mümkündür. Fakat öznitelik sayısının artması MLNN donanımının kullanacağı işlemci elemanları ve dolayısıyla kaynak kullanımını da artıracaktır. Ağ daha da büyüyecek, karmaşık hale gelecek ve donanımsal olarak gerçeklenmesi zorlaşacaktır.

İkinci olarak, doğrusal olmayan bir yapıya sahip olan XOR problemi ele alınmıştır. XOR problemi doğrusal olmayan yapısı sebebiyle sınıflandırma ve modelleme uygulamalarında sıklıkla kullanılmaktadır. XOR problemi için MLNN'nin başlangıç parametreleri MATLAB programı ile elde edilmiştir. Parametreler donanıma gömüldükten sonra ağın eğitimi eğitilebilir MLNN donanımı üzerinde gerçekleştirilmiştir. Eğitim sonrası elde edilen MLNN donanımının çıkışının gerçek değerlere oldukça yaklaştığı ve kabul edilebilir olduğu gözlenmiştir.

Bu tez çalışmasında sadece MLNN mimarisi dikkate alınmış olsa da şematik olarak tasarlanan toplayıcı devre, paralel çarpıcı ve aktivasyon fonksiyonu bloğu diğer YSA topolojileri için de kolaylıkla kullanılabilir. Bu sayede YSA mimarilerinde işlem hızı artacak fakat kaynak kullanımının da artması ile tasarımın düşük kapasiteli FPGA'lar üzerinde gerçeklenmesini zorlaşacaktır. Bununla birlikte, gelişen teknolojiye paralel olarak yeni nesil FPGA'ların kullanılmasıyla kapasite sorunu ortadan kalkacaktır.

KAYNAKLAR

[1] ÖZTEMEL, E., Yapay sinir ağları, Papatya Yayıncılık, İstanbul, 2003.

[2] ÖZDEMIR, AT., Erken ventriküler kasılmalarda YSA tabanlı bir sınıflandırıcının FPGA ile gerçekleştirilmesi, Doktora, Erciyes Üniversitesi, Elektrik Elektronik Mühendisliği, 2010.

[3] MAYR, C., ERLICH, M., HENKER, S., WENDT, K., SCHUFFNY, R., Mapping complex, large-scale spiking networks on neural VLSI, Proc. Wrld. Acad. Sci. E, Vol. 19, pp. 40-45, 2007.

[4] MCCULLOCH, W., PITTS, W., A logical calculus of the ideas immanent in nervous activity, Bulletin of Mathematical Biophysics, Vol. 5, pp. 115-133, 1943.

[5] http://deeplearning.cs.cmu.edu, Erişim Tarihi: 10.09.2014.

[6] BOSQUE, G., DEL CAMPO, I., ECHANOBE, J., Fuzzy systems, neural networks and neuro-fuzzy systems: a vision on their hardware implementation and platforms over two decades, Engineering Applications of Artificial Intelligence, Vol. 32, pp. 283-331, 2014.

[7] BEKEY, GA., GOLDBERG, KY., Neural networks in robotics, Kluwer Academic Publishers, Vol. 202, pp. 580, 1993.

[8] RAO, D., Neural networks in robotics and control: some perspectives, In: International Conference on Industrial Automation and Control IEEE/IAS, IEEE, pp. 451-456, 1995.

[9] ZOU, A-M., HOU, Z-G., FU, S-Y., TAN, M., Neural networks for mobile robot navigation: a survey, Lecture Notes in Computer Science, Vol. 3972, Springer, 2006.

[10] CARPENTER, GA., GROSSBERG, S. (Eds.), Neural networks for vision and image processing, The MIT Press, 1992.

[11] EGMONT-PETERSEN, M., DE RIDDER, D., HANDELS, H., Image processing with neural networks-a review, Pattern Recognit., Vol. 35, pp. 2279-2301, 2002.

[12] HONG, W., CHEN, W., ZHANG, R., The application of neural network in the technology of image processing In: Proceedings of the International Multi-Conference of Engineers and Computer Scientists, Vol. 1, pp. 18-20, 2009.

[13] OTHMAN, A., RIADH, M., Speech recognition using scaly neural networks, World Acad. Sci. Eng. Technol., pp. 253-258, 2008.

[14] LIPPMAN, R., Neural network classifiers for speech recognition, Linc. Lab. J., Vol. 1, pp. 107-124, 1988.

[15] RAY, K., GHOSHAL, J., Neuro fuzzy approach to pattern recognition, Neural Netw., Vol. 10 (1), pp. 161-182, 1997.

[16] PAL, SK., MITRA, S., Neuro-fuzzy pattern recognition: methods in soft computing, Wiley-Interscience.

[17] RUSU, P., PETRIU, EM., WHALEN, TE., CORNELL, A., SPOELDER, HJW., Behavior-based neuro-fuzzy controller for mobile robot navigation, IEEE Trans. Instrum. Meas., Vol. 52 (4), pp. 1335-1340, 2003.

[18] WONGSUWARN, H., LAOWATTANA, D., Neuro-fuzzy algorithm for a biped robotic system, World Acad. Sci. Eng. Technol., Vol. 15, pp. 138-144, 2006.

[19] BABUSKA, R., VERBRUGGEN, H., Neuro-fuzzy methods for nonlinear systems identification, Annu. Rev. Control, Vol. 27, pp. 73-85, 2003.

[20] PANCHARIYA, P., PALIT, A., POPOVIC, D., SHARMA, A., Nonlinear system identification using Takagi-Sugeno type neuro-fuzzy model, In: 2nd IEEE International Conference on Intelligent Systems, IEEE, pp. 76-81, 2004.

[21] LI, C., TSAI, K-B., Adaptive interference signal processing with intelligent neuro-fuzzy approach, In: ICCOMP'06 Proceedings of the 10th WSEAS International Conference on Computers, pp. 393-398, 2006.

[22] CHABAA, S., ZEROUAL, A., ANTARI, J., Application of adaptive neuro-fuzzy inference systems for analyzing non-gaussian signal, In: International Conference on Multimedia Computing and Systems, ICMCS'09, pp. 377-380, 2009.

[23] STINCHCOMBE, M., WHITE, H., Multilayer feedforward networks are universal approximator, Neural Netw, Vol. 2, pp. 359-366, 1989.

[24] COTTER, N., The Stone-Weiertrass theorem and its application to neural networks, IEEE Trans. Neural Netw., Vol. 1 (4), pp. 290-295, 1990.

[25] HORNIK, K., Approximation capabilities of multilayer feedforward networks, Neural Netw., Vol. 4, pp. 251-257, 1991.

[26] ATTALI, J-G., PAGES, G., Approximations of functions by a multilayer perceptron: a new approach, Neural Netw., Vol. 10 (6), pp. 1069-1081, 1997.

[27] CASTRO, J., MANTAS, C., BENITEZ, J., Neural networks with continuous squashing function in the output are universal approximators, Neural Netw., Vol. 13, pp. 561-563, 2000.

[28] LIAO, Y., Neural networks in hardware: A survey, http://bit.csc.lsu.edu/~jianhua/shiv2.pdf, Erişim Tarihi: 09.08.2014.

[29] SCHWARTZ, TJ., A neural chips survey, AI Expert, Vol. 5, pp. 34-39, 1990.

[30] IENNE, P., Architecture for neuro-computers: review and performance evaluation, Technical Report no. 93/21, Microcomputing Laboratory, Swiss Federal Institute of Technology, Lausanne, 1994.

[31] GLESNER, M. POCHMULLER, W., An overview of neural networks in VLSI, Chapman & Hall, London, 1994.

[32] LINDSEY, C., LINDBLAD, T., Review of hardware neural networks: a user's perspective, Proceeding of 3rd Workshop on Neural Networks: From Biology to High Energy Physics, Isola d'Elba, Italy, Sept., pp. 26-30, 1994.

[33] HEEMSKERK, JNH., Overview of neural hardware. Neurocomputers for brain-style processing. Design, implementation and application, Doktora, Leiden University, Netherlands.

[34] MISRA, M., Parallel environment for implementing neural networks, Neural Computing Survey, Vol. 1, pp. 48-60, 1997.

[35] GARTH, S., A chipset for high speed simulation of neural network systems. In: IEEE 1st International Conference on Neural Networks, pp. 443-452, 1987.

[36] HOLLER, M., TAM, S., CASTRO, H., BENSON, R., An electrically trainable artificial neural network (ETANN) with 10240 ‘floating gate’ synapses. In: International Joint Conference on Neural Networks (IJCNN), IEEE, Vol. 2, pp. 191-196, 1989.

[37] SATYANARAYANA, S., TSIVIDIS, Y., GRAF, H., A reconfigurable VLSI neural network, IEEE J. Solid-State Circuits, Vol. 27 (1), pp. 67-81, 1992.

[38] MORIE, T., AMEMIYA, Y., An all-analog expandable neural network LSI with on-chip backpropagation learning, IEEE J. Solid-State Circuits, Vol. 29, pp. 1086-1093, 1994.

[39] SUN, X., CHOW, M., LEUNG, F., XU, D., WANG, Y., LEE, Y-S., Analogue implementation of a neural network controller for UPS inverter applications, IEEE Trans. Power Electron., Vol. 17 (3), pp. 305-313, 2002.

[40] YAMASAKİ, T., SHIBATA, T., Analog soft-pattern-matching classifier using floating-gate MOS technology, IEEE Trans. Neural Netw., Vol. 14 (5), pp. 1257-1265, 2003.

[41] KHODABANDEHLOO, G., MIRHASSANI, M., AHMADI, M., Analog implementation of a novel resistive-type sigmoidal neuron, IEEE Trans. Very Large Scale Integr. (VLSI) Syst., Vol. 20 (4), pp. 750-754, 2012.

[42] SEKERLI, M., BUTERA, RJ., An implementation of a simple neuron model in field programmable analog arrays, In: Proceedings of the 26th Annual International Conference of the IEEE EMBS. IEEE, pp. 4564-4567, 2004.

[43] GRZECHCA, D., GOLONEK, T., RUTKOWSKİ, J., Diagnosis of specification parametric faults in the FPAA-the RBF neural network approach, In: 2nd European Computing Conference (ECC 08), pp. 275-280, 2008.

[44] KAMALA-KANNAN, C., KAMARAJ, V., PARANJOTHI, S., Sensorless control of SR drive using ANN and FPAA for automotive applications, Energy Procedia, Vol. 14, pp. 1831-1836, 2012.

[45] ZAMANLOOY, B., MIRHASSANI, M., Efficient VLSI implementation of neural networks with hyperbolic tangent activation function, In: IEEE Transactions on Very Large Scale Integration (VLSI), Vol. 22 (1), pp. 39-48, 2014.

[46] HIKAWA, H., Implementation of simplified multilayer neural networks with on-chip learning, In: IEEE International Conference on Neural Networks, Vol. 4, pp. 1663-1637, 1995.

[47] ABRAMSON, D., SMITH, K., DUKE, D., FPGA based implementation of a Hopfield neural network for solving constraint satisfaction problems, In: Euromicro Conference, Vol. 2, IEEE Xplore, pp. 688-693, 1998.

[48] OMONDI, A., RAJAPAKSE, J., Neural networks in FPGAs, In: Proceedings of the 9th International Conference on Neural Information Processing (ICONIP'02), Vol. 2, 2002.

[49] KIM, C-M., PARK, H-M., KIM, T., CHOI, Y-K., LEE, S-Y., FPGA implementation of ICA algorithm for blind signal separation and adaptive noise canceling, IEEE Trans. Neural Netw., Vol. 14 (5), pp. 1038-1046, 2003.

[50] IDE, A., SAITO, J., FPGA implementations of neocognitrons, In: Omondi, A ., Rajapakse, J. (Eds.), FPGA Implementations of neural networks. Springer, Netherlands, pp. 197-224, 2006.

[51] FUKUSHIMA, K., Neocognitron: a new algorithm for pattern recognition tolerant of deformations and shift in position, In : Patter Recognition, Vol. 15, pp. 455-469, 1982.

[52] HUBEL, D., WIESEL, T., Receptive fields and functional architecture of monkey striade cortex, J. Physiol, Vol. 165, pp. 215-243, 1968.

[53] BASTOS, J., FIGUEROA, H., MONTI, A., FPGA implementation of neural network-based controllers for power electronics applications, In: Applied Power Electronics Conference and Exposition (APEC 06), IEEE, pp. 1443-1448, 2006.

[54] FERREIRA, P., RIBEIRO, P., ANTUNES, A., MORGADO, F., A high bit resolution FPGA implementation of a FNN with a new algorithm for the activation function, Neurocomputing, pp. 71-77, 2007.

[55] HU, H., HUANG, J., XING, J., WANG, W., Key issues of FPGA implementation of neural networks, In: 2nd International Symposium on Intelligent Information Technology Application, IEEE Computer Society, pp. 259-263, 2008.

[56] SHOUSHAN, L., YAN, C., WENSHANG, X., TONGJUN, Z., A single layer architecture to FPGA implementation of bp artificial neural network, In: 2nd International Asia Conference on Informatics in Control, Automation and Robotics. IEEE, pp. 258-264, 2010.

[57] MEKKI, H., MELLIT, A., KALOGIROU, S., MESSAI, A., FURLAN, G., FPGA-based implementation of a real time photovoltaic module simülatör, Prog. Photovolt., Res. Appl., Vol. 18, pp. 115-127, 2010.

[58] CARDENAS, A., GUZMAN, C., AGBOSSOU, K., Development of a FPGA based real-time power analysis and control for distributed generation interface, IEEE Trans. Power Syst., Vol. 27 (3), pp. 1343-1353, 2012.

[59] http://ecee.colorado.edu/~ecen4831/Demuth/Ch10_pres.pdf, Erişim Tarihi: 05.09.2014.

[60] SOLEIMANI, H., AHMADI, A., BAVANDPOUR, M., Biologically inspired spiking neurons: piecewise linear models and digital implementation, IEEE Trans. Circuits Syst., Vol. 12, pp. 2991-3004, 2012.

[61] IZHIKEVICH, E., Simple model of spiking neurons, IEEE Trans. Neural Netw., Vol. 14 (6), pp. 1569-1572, 2003.

[62] SAADI, AGS., BETTAYEB, M., Abc optimized neural network model for image deblurring with its FPGA implementation, Microprocess. Microsyst., Vol. 37, pp. 52-64, 2013.

[63] CARD, H., ROSENDAHL, G., MCNEILL, D., MCLEOD, R., Competitive learning algorithms and neurocomputer architecture, IEEE Trans. Comput., Vol. 47 (8), pp. 847-858, 1998.

[64] BOQUETE, L., MARTIN, P., MAZO, M., GARCIA, R., BAREA, R., RODRIGUEZ, F., FERNANDEZ, I., Hardware implementation of a new neurocontrol wheelchair-guidance system, Neurocomputing, Vol. 47, pp. 145-160, 2002.

[65] VENAYAGAMOORTHY, G., HARLEY, R., WUNSCH, D., Implementation of adaptive critic-based neurocontrollers for turbogenerators in a multimachine power system, IEEE Trans. Neural Netw., Vol. 14 (5), pp. 1047-1064, 2003.

[66] LEE, B., SHEU, B., A compact and general-purpose neural chip with electrically programmable synapses, In: IEEE Custom Integrated Circuits Conference, pp. 26.6.1-26.6.4., 1990.

[67] BOSER, B., SACKINGER, E., BROMLEY, J., LE CUN, Y., JACKEL, L., An analog neural network processor with programmable topology, IEEE J. Solid-State Circuits, Vol. 26 (12), pp. 2017-2025, 1991.

[68] SHIMA, T., KIMURA, T., KAMATANI, Y., ITAKURA, T., FUJITA, Y., IIDA, T., Neuro chips with on-chip back-propagation and/or Hebbian learning, IEEE J. Solid-State Circuits, Vol. 27 (12), pp. 1868-1876, 1992.

[69] LU, C., SHI, B., CHEN, L., A programmable on-chip BP learning neural network with enhanced neuron characteristics, In: IEEE International Symposium on Circuits and Systems (ISCAS 2001), Vol. 3, 2001.

[70] ERKMEN, NKB., VURAL, RA., YILDIRIM, T., A mixed mode neural network circuitry for object recognition application, Circuits Syst. Signal Process., Vol. 32, pp. 29-46, 2013.

[71] SACKINGER, E., GRAF, H., A board system for high-speed image analysis and neural networks, IEEE Trans. Neural Netw., Vol. 7 (1), pp. 214-221, 1996.

[72] MISRA, J., SAHA, I., Artificial neural networks in hardware: A survey of two decades of progress, Neurocomputing, Vol. 74, pp. 239-255, 2010.

[73] KUNG, SY., Digital neural networks, Prentice-Hall, Upper Saddle River, NJ, USA, 1992.

[74] IENNE, P., Digital hardware architectures for neural networks, Speedup Journal, Vol. 9 (1), pp. 18-25, 1995.

[75] BERMAK, A., MARTINEZ, D., A compact 3-D VLSI classifier using bagging threshold network ensembles, IEEE Transactions on Neural Networks, Vol. 14 (5), pp. 1097-1109, 2003.

[76] MEAD, C., Analog VLSI and neural systems, Addison-Wesley, Boston, MA, USA, 1989.

[77] BROWN, B., YU, X., GARVERICK, S., Mixed-mode analog VLSI continuous-time recurrent neural network, in: Proceedings of International Conference on Circuits, Signals and Systems, pp. 104-108, 2004.

[78] SCHMID, A., LEBLEBICI, Y., MLYNEK, D., A mixed analog digital artificial neural network with on chip learning, IEE Proceedings-Circuits, Devices and Systems, Vol. 146, 1999.

[79] LEHMANN, T., BRUUN, E., DIETRICH, C., Mixed analog/digital matrix–vector multiplier for neural network synapses, Analog Integrated Circuits and Signal Processing, Vol. 9 (1), pp. 55-63, 2004.

[80] SCHRAUWEN, B., D'HAENE, M., Compact digital hardware implementations of spiking neural networks, in: J. Van Campenhout (Ed.), Sixth FirW Ph.D. Symposium, in CD, 2005.

[81] NEDJAH, N., DE MACEDO MOURELLE, L., Reconfigurable hardware for neural networks: binary versus stochastic, Neural Computing and Applications, Vol. 16 (3), pp. 249-255, 2007.

[82] ADAM RAK, GC., SOOS, BG., Stochastic bitstream-based CNN and its implementation on FPGA, International Journal of Circuit Theory and Applications, Vol. 37 (4), pp. 587-612, 2002.

[83] MOERLAND, PD., FIESLER, E., SAXENA, I., Incorporation of liquid-crystal light valve nonlinearities in optical multilayer neural networks, Applied Optics, Vol. 35, pp. 5301-5307, 1996.

[84] TOKES, S., ORZO, L., VARO, G., ROSKA, T., Bacteriorhodopsin as an analog holographic memory for joint fourier implementation of CNN computers, Technical Report DNS-3-2000, Computer and Automation Research Institute of the Hungarian Academy of Sciences, Budapest, Hungary, 2000.

[85] LAMELA, H., RUIZ-LLATA, M., Optoelectronic neural processor for smart vision applications, Imaging Science Journal, Vol. 55 (4), pp. 197-205, 2007.

[86] GLESNER, M., POECHMUELLER, W., Neurocomputers: An overview of neural networks in VLSI, Chapman and Hall, London, 1994.

[87] HEEMSKERK, J., Overview of neural hardware, in: Neurocomputers for Brain-Style Processing, Design, Implementation and Application, 1995.

[88] IENNE, P., CORNU, T., KUHN, G., Special-purpose digital hardware for neural networks: an architectural survey, Journal of VLSI Signal Processing Systems, Vol. 13 (1), pp. 5-25, 1996.

[89] AYBAY, I., CETINKAYA, S., HALICI, U., Classification of neural network hardware, Neural Network World, Vol. 6 (1), pp. 11-29, 1996.

[90] SUNDARARAJAN, N., SARATCHANDRAN, P., Parallel architectures for artificial neural networks: paradigms and implementations, IEEE Computer Society Press, Los Alamitos, CA, USA, 1998.

[91] BURR, JB., Digital neurochip design, in: PRZYTULA, KW., PRASANNA, VK., (Eds.), Parallel digital implementations of neural networks, Prentice-Hall, Upper Saddle River, NJ, USA, pp. 223-281, 1992.

[92] BURR, JB., Energy, capacity, and technology scaling in digital VLSI neural networks, NIPS'91 VLSI Workshop, 1991.

[93] ZHU, J., SUTTON, P., FPGA implementations of neural networks-a survey of a decade of progress, Field-Programmable Logic and Applications, vol. 2778, pp. 1062-1066, 2003.

[94] MAGUIRE, LP., MCGINNITY, TM., GLACKIN, B., GHANI, A., BELATRECHE, A., HARKIN, J., Challenges for large-scale implementations of spiking neural networks on FPGAs, Neurocomputing Vol. 71 (1-3), pp. 13-29, 2007.

[95] BARTOLOZZI, C., INDIVERI, G., Synaptic dynamics in analog VLSI, Neural Computation, Vol. 19 (10), pp. 2581-2603, 2007.

[96] MUTHURAMALINGAM, A., HIMAVATHI, S., SRINIVASAN, E., Neural network implementation using FPGA: issues and application, International Journal of Information Technology, Vol. 4 (2), pp. 2-12, 2007.

[97] HIKAWA, H., A digital hardware pulse-mode neuron with piecewise linear activation function, IEEE Transactions on Neural Networks, Vol. 14 (5), pp. 1028-1037, 2003.

[98] PARLAKYILDIZ, Ş., Yapay sinir ağları kullanılarak parmak izi tanıma ve sınıflandırma, Yüksek Lisans, Gazi Üniversitesi, Elektrik Elektronik Mühendisliği, 2014.

[99] ER, O., Esnek hesaplama ve biyobilişim teknikleri ile bir klinik karar verme simülatörünün oluşturulması, Doktora, Sakarya Üniversitesi, Elektrik Elektronik Mühendisliği, 2009.

[100] TEMÜR, G., Yapay sinir ağlarının otomatik olarak FPGA çipine uygulanması için denetleyici tasarım aracı, Yüksek Lisans, Düzce Üniversitesi, Elektrik Eğitim Anabilim Dalı, 2013.

[101] ÇAVUŞLU, MA., Yapay sinir ağları eğitiminin gradyen tabanlı ve global arama algoritmaları ile FPGA üzerinde donanımsal gerçeklenmesi, Yüksek Lisans, Niğde Üniversitesi, Elektrik Elektronik Mühendisliği, 2013.

[102] DOĞUÇ, U., Esnek imalat sistemlerinde makine sayılarının ve teslim tarihinin belirlenmesinde yapay sinir ağlarının kullanılması, Doktora, Sakarya Üniversitesi, Fen Bilimleri Enstitüsü, 2001.

[103] HAYKIN, S., Neural networks a comprehensive foundation, Prentice Hall Publishing, New Jersey, USA, Vol. 1, pp. 1-14, 1994.

[104] DORF, RC., The electrical engineering handbook, 3. Baskı, CRC/Taylor&Francis, Boca Raton, 2006.

[105] GRAUPE, D., Principles of artificial neural networks, World Scientific: Singapore, River Edge, NJ, 1997.

[106] ALAVALA, CR., Fuzzy logic and neural networks: basic concepts and application, New Age International, Daryaganj, Delhi, IND, 2008.

[107] KOHONEN, T., Self-organization and associative memory, Springer-Verlag, 1984.

[108] ZURADA, JM., Introduction to artificial neural systems, West, St. Paul, 1992.

[109] ÖZTEMEL, E., Integrating expert systems and NNs for intelligent on-line statistical process control, University of Wales, PhD. Thesis, School of Electrical-Electronic and System Engineering, Cardiff, pp. 14, 1992.

[110] WIDROW, B., HOFF, ME., Adaptive switching circuits, In Neurocomputing: Foundations of Research, MIT Press, 1988.

[111] RUMELHART, DE., MCCLELLAND, JL., University of California San Diego. PDP Research Group., Parallel Distributed Processing: Explorations in the Microstructure of Cognition, MIT Press, Cambridge, Mass., 1986.

[112] CARPENTER, GA., GROSSBERG, S., The art of adaptive pattern-recognition by a self-organizing neural network, Computer, Vol. 21, pp. 77-88, 1988.

[113] YILMAZ, N., Alan programlamalı kapı dizileri (FPGA) üzerinde bir YSA'nın tasarlanması ve donanım olarak gerçekleştirilmesi, Yüksek Lisans, Selçuk Üniversitesi, Bilgisayar Mühendisliği, 2008.

[114] ÖZTEKİN, H., Eğitim amaçlı yapılandırılabilir modüler donanım üzerine gömülü işletim sistemi tasarımı, Doktora, Sakarya Üniversitesi, Bilgisayar ve Bilişim Mühendisliği, 2012.

[115] ÖLMEZ, E., FPGA tabanlı mikrobilgisayar mimarisi kullanılarak DC motor sürücü tasarımı ve uygulaması, Yüksek Lisans, Bozok Üniversitesi, Mekatronik Mühendisliği, 2012.

[116] http://www.fpganedir.com, Erişim Tarihi: 05.09.2014.

[117] BAUMANN, C., Field programmable gate array (FPGA), Summer Paper For The Seminar "Embedded System Architecture", University of Innsbruck., 2010.

[118] http://web.itu.edu.tr/orencik/BilgMimYenYakl2007/Mehmet_Aktas/FPGA_Mimarisi_Rapor.pdf, Erişim Tarihi: 05.09.2014.

[119] http://ee.sharif.edu/~asic/Tutorials/Quartus/AppendixB_quartus.pdf, Erişim Tarihi: 22.09.2014.

[120] SALEH, HHM., Fused floating-point arithmetic for DSP, University of Texas, Austin, 2009.

[121] IEEE, Standard for binary floating-point arithmetic, ANSI/IEEE Standard 754-1985, 1985.

[122] IEEE, Standard for floating-point arithmetic, IEEE Standard 754-2008, 2008.

[123] ERLE, MA., Algorithms and hardware designs for decimal multiplication, Lehigh University, 2008.

[124] ÖRS, SB., VHDL ile lojik devre tasarımı ve DSP uygulamaları için çarpma bloklarının modellenmesi, Yüksek Lisans, İstanbul Teknik Üniversitesi, 1998.

[125] MARVEN, C., EWERS, G., A simple approach to digital signal processing, Texas Instruments, 1994.

[126] TAKAGI, N., YASUURA, H., YAJIMA, S., High-speed VLSI multiplication algorithm with a redundant binary addition tree, IEEE Trans. on Computers, Vol. C-34 (9), pp. 789-796, 1985.

[127] COOPER, AR., Parallel architecture modified Booth multiplier, IEE Proceedings, Vol. 135, Pt. G, no. 3, pp. 125-128, 1988.

[128] SUNDER, S., A fast multiplier based on the modified Booth algorithm, Int. J. Electronics, Vol. 75 (2), pp. 199-208, 1993.

[129] MORI, J., et al., A 10-ns 54x54-b parallel structured full array multiplier with 0.5-μm CMOS technology, IEEE J. Solid-State Circuits, Vol. 26 (4), pp. 600-606, 1991.

[130] KUNINOBU, S., et al., High speed MOS multiplier and divider using redundant binary representation and their implementation in a microprocessor, IEICE Trans. Electron., Vol. E76-C, no. 3, pp. 436-445, 1993.

[131] MAKINO, H., An 8.8 ns 54x54 bit multiplier with high speed redundant binary architecture, IEEE J. Solid-State Circuits, Vol. 31 (6), 1996.

[132] BENREKIA, F., ATTARI, M., A floating point multiplier based FPGA synthesis for neural networks enhancement, International Journal of Engineering Science and Technology (IJEST), Vol. 2 (5), 1433-1440, 2010.

[133] JAIN, A., et al., FPGA design of a fast 32-bit floating point multiplier unit, International Conference on Devices, Circuits and Systems (ICDCS), pp. 545-547, 2012.

[134] MANVI, SS., RAJESH, L., V JOSHI and PRASHANT,An FPGA based implementation of floating-point multiplier, International Conference on Computing and Control Engineering (ICCCE 2012), 2012.

[135] PEDRONI, VA., Circuit Design with VHDL, MIT Press: Cambridge, Mass., 2004.

[136] CETİN, O., OLMEZ, E., TEMURTAS, F., KOKLUKAYA, E., Hardware implementation of a fast floating-point adder for embedded systems, Electronics World, Vol. 120 (1938), pp.16-22, 2014.

[137] KARLSTRÖM, P., et al., High performance, low latency FPGA based floating point adder and multiplier units in a Virtex 4, IEEE Norchip Conference, 24th, 31-34, 2006.

[138] CHONG, YJ., PARAMESWARAN,S., Configurable multimode embedded floating-point units for FPGAs, IEEE Trans. on Very Large Scale Integration (VLSI) Systems, Vol. 19 (11), 2011.

[139] NIELSEN, AM., et al., An IEEE compliant floating-point adder that conforms with the pipelined packet-forwarding paradigm, IEEE Trans. on Computers, vol. 49 (1), 2000.

[140] KRUEGER, SD., SEIDEL, P-M., Design of an on-line IEEE floating-point addition unit for FPGAs, IEEE Symposium on Field-Programmable Custom Computing Machines, 239-246, 2004.

[141] LOUCA, L., et al., Implementation of IEEE single precision floating point addition and multiplication on FPGAs, IEEE Symposium on FPGAs for Custom Computing Machines, pp. 107-116, 1996.

[142] SHARMA, G., et al., A low power 8-bit magnitude comparator with small transistor count using hybrid PTL/CMOS logic, IJCEM International Journal of Computational Engineering & Management, Vol. 12, ISSN (Online), pp. 2230-7893, 2011.

[143] LEON, MAA., CASTRO, AR., ASCENCIO, RRL., An artificial neural network on a field programmable gate array as a virtual sensor, Design of Mixed-Mode Integrated Circuits and Applications, pp. 114-117, 1999.

[144] SARAOGLU, HM., TEMURTAS, F., ALTIKAT, S., Quantitative classification of HbA1C and blood glucose level for diabetes diagnosis using neural networks, Australas. Phys. Eng. Sci. Med., DOI 10.1007/s13246-013-0217-x, 2013.

ÖZGEÇMİŞ

Onursal ÇETİN, 06.06.1982 yılında Kayseri'de doğdu. İlkokulu Osman Kavuncu İlk Öğretim Okulu'nda, orta ve lise öğrenimini Nuh Mehmet Küçükçalık Anadolu Lisesi'nde tamamladı. 2001 yılında başladığı Erciyes Üniversitesi Elektronik Mühendisliği Bölümü'nü 2006 yılında bitirdi. Aynı yıl Erciyes Üniversitesi Elektrik-Elektronik Mühendisliği Bölümü'nde yüksek lisansa başladı. 2007 yılında Araştırma Görevlisi olarak Bozok Üniversitesi Elektrik-Elektronik Mühendisliği Bölümü'nde göreve başladı ve halen bu görevde akademisyenlik hayatını sürdürmektedir.