T.C.
SAKARYA UYGULAMALI BİLİMLER ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

GÖMÜLÜ SİSTEMLERDE KAYAN NOKTA ARİTMETİK HESAPLAMALAR İÇİN VERİLOG TABANLI IP CORE KÜTÜPHANESİ GELİŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

Esin MUTLU KORKMAZ

Enstitü Anabilim Dalı : BİLGİSAYAR MÜHENDİSLİĞİ

Tez Danışmanı : Doç. Dr. Serkan DERELİ

Ekim 2024

T.C.
SAKARYA UYGULAMALI BİLİMLER ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

GÖMÜLÜ SİSTEMLERDE KAYAN NOKTA ARİTMETİK HESAPLAMALAR İÇİN VERİLOG TABANLI IP CORE KÜTÜPHANESİ GELİŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

Esin MUTLU KORKMAZ

Enstitü Anabilim Dalı : BİLGİSAYAR MÜHENDİSLİĞİ

Bu tez 10/10/2024 tarihinde aşağıdaki jüri üyeleri tarafından oy birliği ile kabul edilmiştir.

JÜRİ	BAŞARI DURUMU
Jüri Başkanı: Doç. Dr. Serkan DERELİ	Başarılı
Üye: Prof. Dr. Halit ÖZTEKİN	Başarılı
Üye: Prof. Dr. Ahmet ZENGİN	Başarılı

BEYAN

Tez içindeki tüm verilerin akademik kurallar çerçevesinde tarafımdan elde edildiğini, görsel ve yazılı tüm bilgi ve sonuçların akademik ve etik kurallara uygun şekilde sunulduğunu, kullanılan verilerde herhangi bir tahrifat yapılmadığını, başkalarının eserlerinden yararlanılması durumunda bilimsel normlara uygun olarak atıfta bulunulduğunu, tezde yer alan verilerin bu üniversite veya başka bir üniversitede herhangi bir tez çalışmasında kullanılmadığını beyan ederim

Esin MUTLU KORKMAZ

10/10/2024

TEŞEKKÜR

Yüksek lisans eğitimim boyunca değerli bilgi ve deneyimlerinden yararlandığım, her konuda bilgi ve desteğini almaktan çekinmediğim, araştırmanın planlanmasından yazılmasına kadar tüm aşamalarında yardımlarını esirgemeyen, beni teşvik eden ve titizlikle yönlendiren kıymetli danışman hocam Doç. Dr. Serkan DERELİ'ye sonsuz teşekkürlerimi sunarım. Ayrıca, her zaman desteğini esirgemeyen, bana inanan ve daima yanımda olan annem Elif MUTLU, babam Yusuf MUTLU, kız kardeşim Esen MUTLU ve eşim Berf KORKMAZ'a da minnet ve şükranlarımı sunarım.

Ayrıca bu çalışmanın maddi açıdan desteklenmesine olanak sağlayan Sakarya Uygulamalı Bilimler Üniversitesi Bilimsel Araştırma Projeleri (BAP) Komisyon Başkanlığına (Proje No: 197-2024) teşekkür ederim.

İÇİNDEKİLER

TEŞEKKÜR ..... **i**
İÇİNDEKİLER ..... **ii**
KISALTMALAR ..... **v**
SİMGELER ..... **vi**
TABLOLAR LİSTESİ ..... **vii**
ŞEKİLLER LİSTESİ ..... **viii**
ÖZET ..... **ix**
ABSTRACT ..... **x**

BÖLÜM 1.
GİRİŞ **1**

BÖLÜM 2.
LİTERATÜR ARAŞTIRMASI **4**

BÖLÜM 3.
SAYISAL TASARIM **8**
3.1. FPGA (Field-Programmable Gate Array) 8
3.1.1. FPGA ve Diğer Donanım Seçeneklerinin Karşılaştırması 10
3.1.2. FPGA'yı Tercih Etme Sebeplerimiz 12
3.2. FPGA programlama 13
3.3. Xilinx Vivado 15
3.4. Simülasyon 17
3.5. Donanım Tanımlama Dilleri (HDL - Hardware Description Languages) 21
3.6. FPGA'de IP Core Nedir? 25
3.6.1. IP Core'ların Yapısal Özellikleri 25
3.7. Kayan Nokta (Floating Point) ve Sabit Noktalı (Fixed Point) İşlemler 27

3.7.1. Sabit Noktalı Sayı Sistemi (Fixed Point) 27
3.7.2. Kayan Noktalı Sayı Sistemi (Floating Point) 28
3.7.3. Kayan Nokta ve Sabit Nokta Karşılaştırması 29
3.8. Kayan Nokta (Floating Point) Kullanım Alanları ve Avantajları 30
3.9. Kayan Nokta (Floating Point) Sayı Sistemleri 34
3.9.1. IEEE 754 Standardı 35
3.9.1.1. Tek Hassasiyetli Kayan Nokta Formatı (32-bit): 36
3.9.1.2. Çift Hassasiyetli Kayan Nokta Formatı (64-bit): 36
3.10. Kayan Nokta Aritmetiği 37
Kayan Nokta ile Toplama ve Çıkarma 37
3.10.1. Kayan Nokta İşlemlerinin Kullanım Alanları 37
3.11. FPGA'de Floating Point Unit (FPU) 38
3.12. IEEE 754 Kayan Nokta Sayı Formatında Yuvarlama İşlemleri 39
3.12.1. En Yakına Yuvarlama (Round to Nearest, Even) 39
3.12.2. Sıfıra Doğru Yuvarlama (Round Towards Zero) 40
3.12.3. Sonsuza Doğru Yuvarlama (Round Towards +∞) 40
3.12.4. Eksi Sonsuza Doğru Yuvarlama (Round Towards -∞) 40
3.13. Finite-State Machine (FSM) 41
3.14. Aritmetik Mantık Birimi (ALU) 44
3.15. Kayan Nokta (Floating Point) Dört Temel İşlem 47
3.15.1. Kayan Nokta ile Toplama 49
3.15.2. Kayan Nokta ile Çıkarma 51
3.15.3. Kayan Nokta ile Çarpma 52
3.15.4. Kayan Nokta ile Bölme 54
3.15.5. Integer'dan Floating Point'e, 4 İşlem, Integer'a Dönüşüm 55
3.15.5.1. Integer – Floating Point Dönüşümü Adımları 57
3.15.5.2. Kayan Nokta (Floating Point) Toplama İşlemi Adımları 58
3.15.5.3. Kayan Nokta (Floating Point) Çıkarma İşlemi Adımları 60
3.15.5.4. Kayan Nokta (Floating Point) Çarpma İşlemi Adımları 61
3.15.5.5. Kayan Nokta (Floating Point) Bölme İşlemiAdımları 63
3.15.5.6. Floating Point - Integer Dönüşümü Adımları 65

BÖLÜM 4.
TARTIŞMA VE SONUÇ 67
4.1. Bulgular 67
4.1.1. Simülasyon Sonuçlar 67
4.1.2. Doğruluk ve Hassasiyet 69
4.1.3. Performans ve Hız 70
4.1.4. Geliştirilmiş Optimizasyon Teknikleri 70
4.2. Tartışma 70
4.2.1. Literatüre Katkısı 71
4.2.2. Gelecekte Yapılabilecek İyileştirmeler 72
4.3. Sonuçlar ve Performans Analizi 73
4.4. Sonuç 74

KAYNAKLAR 75

ÖZGEÇMİŞ 78

KISALTMALAR

FPGA	:Field-Programmable Gate Array
ASIC	:Application-Specific Integrated Circuit
DSP	:Digital Signal Processor
FPU	:Floating Point Unit
VHDL	:VHSIC Hardware Description Language
HDL	:Hardware Description Language
CLB	:Configurable Logic Blocks
IOB	:Input/Output Blocks
UART	:Universal Asynchronous Receiver-Transmitter
PRNG	:Pseudorandom Number Generator
LFSR	:Linear Feedback Shift Register
IP Core	:Intellectual Property Core
IEEE	:Institute of Electrical and Electronics Engineers
SoC	:System on Chip
CPLD	:Complex Programmable Logic Device

SİMGELER

&	:AND (Mantıksal VE operatörü)
<=	:Atama operatörü (assign)
@	:Olay tetikleyici (posedge veya clock sinyali gibi olaylar için)
|	:OR (Mantıksal VEYA operatörü)
~	:NOT (Mantıksal DEĞİL operatörü)
#	:Gecikme belirteci (simülasyonlarda zaman gecikmesini ifade eder)
$	:Sistem görevleri (Verilog'da $display, $finish gibi görevler)
×	:Çarpma işlemi (aritmetik çarpma)
+	:Toplama işlemi
-	:Çıkarma işlemi

TABLOLAR LİSTESİ

Tablo 3.1. FPGA ve Diğer Seçenekler Arasındaki Karşılaştırma 12
Tablo 3.2. IP Core Örnekleri ve Özellikleri 26
Tablo 3.3. Kayan Nokta ve Sabit Nokta Karşılaştırması 29
Tablo 3.4. FPGA'lerde FPU modülleri 38
Tablo 3.5. İşlem komut tablosu 48

ŞEKİLLER LİSTESİ

Şekil 1.1: Mantık blokları, ara bağlantı elemanları FPGA 2
Şekil 2.1: DSP-FPGA ve ASIC gömülü işlemcilerin karşılaştırılması 4
Şekil 2.2: ASIC vs. SoC vs. FPGA 5
Şekil 3.1: Ana bileşenlerini gösteren bir FPGA'nın şematik diyagramı 9
Şekil 3.2: IEEE 754 standardına göre bir kayan nokta sayısı 35
Şekil 3.3: IEEE 754 Tek Hassasiyetli Kayan Nokta Formatı (32-bit) 36
Şekil 3.4: IEEE 754 Çift Hassasiyetli Kayan Nokta Formatı (64-bit) 36
Şekil 3.5: 32 Bit kayan noktalı sayı formatında FPU 47
Şekil 3.6: Toplama ve Çıkarma işlemi akış diyagramı 50
Şekil 3.7: Çarpma işlemi akış diyagramı 53
Şekil 3.8: Bölme işlemi akış diyagramı 54
Şekil 3.9. Gerçekleştirilen Sayı Grupları 68
Şekil 3.10: Simülasyon sonuçları - 1 68
Şekil 3.11: Simülasyon sonuçları - 2 68
Şekil 3.12: Simülasyon sonuçları – 3 68
Şekil 3.13. FPGA içerisinde kaynak kullanım bilgileri 69

GÖMÜLÜ SİSTEMLERDE KAYAN NOKTA ARİTMETİK HESAPLAMALAR İÇİN VERİLOG TABANLI IP CORE KÜTÜPHANESİ GELİŞTİRİLMESİ

ÖZET

Gömülü sistemlerin karmaşık hesaplama işlemlerini gerçekleştirmek amacı ile kullanılan kayan nokta aritmetiği, günümüz teknolojisinin birçok farklı alanında kritik bir rol oynamaktadır. Özellikle sinyal işleme, görüntü işleme, yapay zeka uygulamaları ve bilimsel hesaplamalar gibi yüksek doğruluk gerektiren alanlarda sıkça tercih edilmektedir. Ancak mevcut çözümler genellikle belirli platformlara bağımlıdır ve bu çözümler performans açısından yeterince optimize edilmemiştir. Bu çalışmada, gömülü sistemlerde kayan nokta aritmetik hesaplamaları için özgün bir çözüm olarak Verilog tabanlı sentezlenebilir bir IP core kütüphanesi geliştirilmiştir.

Çalışmanın başlangıcında, gömülü sistemlerdeki hesaplama gereksinimleri ve kayan nokta aritmetiğinin önemi detaylı bir şekilde analiz edilmiştir. Bu analiz, mevcut literatürdeki eksiklikleri ortaya koymayı ve yeni geliştirme fırsatlarını belirlemeyi hedeflemiştir. Bu doğrultuda, özgün bir çözüm oluşturmak amacıyla tasarım ilkeleri geliştirilmiştir. Geliştirilen IP core kütüphanesi, toplama, çıkarma, çarpma ve bölme gibi temel aritmetik işlemleri destekleyen modüler bir yapıya sahiptir. Ayrıca, bu kütüphane platform bağımsız çalışmayı ve optimize edilmiş bir performansı sağlamak amacıyla çeşitli teknikler ve algoritmalar kullanılarak tasarlanmıştır. Bu özellikleri sayesinde kütüphane, farklı FPGA platformlarına kolaylıkla entegre edilebilmektedir. Ayrıca düşük enerji tüketimi ve hızlı hesaplama süreleri sunarak sistem performansını artırmaktadır.Verilog dilinde yazılmış olan bu kütüphane, gömülü sistem tasarımcılarının kayan nokta aritmetik işlemlerini hızlı, verimli ve doğru bir şekilde gerçekleştirmelerine olanak tanımaktadır. Kütüphane, çeşitli deneyler ve testlerle değerlendirilmiş olup, gerçek dünya uygulamalarında başarılı sonuçlar elde edilmiştir. Bu test sonuçları, kütüphanenin gömülü sistemlerdeki performansını ve etkinliğini net bir şekilde ortaya koymaktadır. Aynı zamanda, performans optimizasyonu ve doğrulama süreçleri simülasyonlar ile titizlikle gerçekleştirilmiş olup, geniş bir kullanım alanı sağlamaktadır.

Sonuç olarak, bu çalışma, gömülü sistemlerde kayan nokta aritmetik hesaplamaları için platform bağımsız ve optimize edilmiş performansa sahip bir çözüm sunmaktadır. Geliştirilen IP core kütüphanesi, gömülü sistem tasarımcılarına verimli ve güvenilir bir kaynak sağlayarak, karmaşık hesaplama ihtiyaçlarını karşılamada önemli bir araç olarak öne çıkmaktadır. Geliştirilen bu kütüphane, gömülü sistemlerin farklı alanlarındaki gereksinimlere cevap vererek, mühendisler için güçlü bir çözüm sunmaktadır.

Anahtar Kelimeler: Gömülü Sistemler, Kayan Nokta Aritmetiği, Verilog, Sentezlenebilir IP Core, FPGA, Performans Optimizasyonu

DEVELOPMENT OF A VERILOG BASED IP CORE LIBRARY FOR FLOATING POINT ARITHMETIC CALCULATIONS IN EMBEDDED SYSTEMS

ABSTRACT

Floating-point arithmetic, a key tool for performing complex computational tasks in embedded systems, has become critically important in many modern applications, ranging from digital signal processing to artificial intelligence and machine learning algorithms. Despite its widespread usage, existing solutions are often tied to specific platforms and fail to offer optimized performance, limiting their flexibility and efficiency. This study addresses these limitations by developing a Verilog-based synthesizable IP core library as an innovative solution for floating-point arithmetic calculations in embedded systems. At the beginning of this research, the computational requirements of embedded systems and the significance of floating-point arithmetic were thoroughly examined. This analysis highlighted gaps and potential opportunities for improvement within the current body of literature, particularly in terms of platform independence and performance optimization. Based on these insights, design principles were established to create an original and optimized solution that could address these gaps. The developed IP core library features a modular architecture that supports fundamental arithmetic operations such as addition, subtraction, multiplication, and division. Furthermore, the library incorporates various algorithms and optimization techniques to ensure both platform independence and high performance, making it easily adaptable to different FPGA platforms. This library, which has been meticulously coded in Verilog, enables embedded system designers to perform floating-point arithmetic operations both quickly and efficiently. It offers significant advantages in terms of speed, resource utilization, and accuracy, ensuring that critical arithmetic computations are handled effectively in real-time systems. To verify its functionality and efficiency, the library underwent extensive testing and simulation, with successful results observed in real-world applications. These tests demonstrated the library's high performance, flexibility, and effectiveness, confirming its value in addressing the computational needs of embedded systems. In conclusion, this study provides a novel and innovative solution for floating-point arithmetic calculations tailored specifically for embedded systems. With its platform-independent design, modularity, and optimized performance, the developed IP core library offers a powerful and flexible resource for embedded system designers. It stands out as an efficient and reliable tool for meeting the complex computational demands of modern embedded systems, offering new possibilities for applications in various industries.

Keywords: Embedded Systems, Floating-Point Arithmetic, Verilog, Synthesizable IP Core, FPGA, Performance Optimization

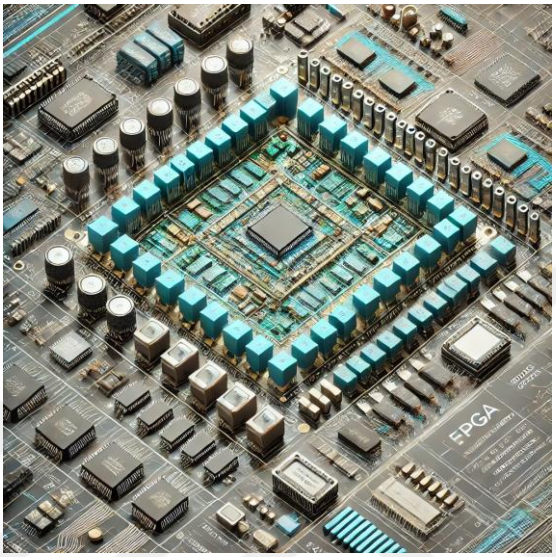
BÖLÜM 1. GİRİŞ

Gömülü sistemler, günümüz teknolojisinin vazgeçilmez bir bileşeni olarak pek çok sektörde yaygın bir şekilde kullanılmaktadır. Bu sistemler, sınırlı kaynaklarla çalışmak üzere tasarlanmış olup, belirli işlevleri yüksek verimlilikle yerine getirebilmek için optimize edilmiştir. Özellikle otomotiv, sağlık, telekomünikasyon ve ev otomasyonu gibi farklı endüstrilerde gömülü sistemler, günlük yaşamın bir parçası haline gelmiş durumdadır. Gömülü sistemlerin bu kadar geniş bir uygulama yelpazesi sunmasının temel nedeni, düşük enerji tüketimi, yüksek verimlilik ve esneklik sağlamasıdır. Bununla birlikte, bu sistemlerde gerçekleştirilen görevlerin giderek daha karmaşık hale gelmesi, özel donanım ve yazılım çözümlerine olan ihtiyacı artırmaktadır. Gömülü sistemlerde özellikle yüksek doğruluk ve hızlı işlem kapasitesi gerektiren görevlerin yerine getirilmesi için kayan nokta aritmetiği gibi gelişmiş hesaplama tekniklerine başvurulmaktadır.

Kayan nokta aritmetiği, özellikle bilimsel ve mühendislik hesaplamalarında sıklıkla tercih edilen bir tekniktir. Bu teknik, büyük ve küçük değerler arasında yüksek doğrulukta matematiksel işlemler gerçekleştirebilme yeteneğiyle dikkat çekmektedir. Ancak kayan nokta aritmetiği, özellikle gömülü sistemlerde işlem yaparken yüksek hesaplama gücü gerektirir ve bu da performans sınırlamalarına yol açabilir. Geleneksel mikroişlemciler, bu tür yoğun hesaplama gereksinimlerine yanıt vermekte yetersiz kalabilir. Bu nedenle, kayan nokta aritmetik işlemlerinin hızlandırılması ve bu işlemlerin verimli bir şekilde gerçekleştirilmesi için özel donanım çözümlerine ihtiyaç duyulmaktadır. Bu noktada, özellikle FPGA (Field-Programmable Gate Array) gibi programlanabilir donanımların kullanılması, hem paralel işlem kabiliyeti hem de yeniden programlanabilirliği sayesinde önemli avantajlar sunar.

Bu çalışmanın temel amacı, gömülü sistemlerde kayan nokta aritmetik hesaplamalarını hızlandırmak ve optimize etmek için Verilog tabanlı sentezlenebilir bir IP core (çekirdek) kütüphanesi geliştirmektir.

Mevcut kayan nokta aritmetiği çözümleri genellikle belirli platformlarla sınırlı kalmakta ve optimize edilmemiş yapılara sahip olmaktadır. Bu durum, sistem tasarımcılarının esnek çözümler üretmesini zorlaştırmakta ve performans açısından yetersizlikler doğurmaktadır. Geliştirilecek olan IP core kütüphanesi, bu sorunların üstesinden gelmek için tasarlanmış olup, farklı platformlarda ve çeşitli uygulamalarda kullanılabilecek şekilde optimize edilmiştir.



Şekil 1.1: Mantık blokları, ara bağlantı elemanları FPGA

Gömülü sistemlerde gerçekleştirilen birçok işlem, doğru ve hızlı bir şekilde tamamlanmalıdır. Özellikle görüntü işleme, sinyal işleme, yapay zeka uygulamaları ve bilimsel hesaplamalar gibi doğruluk gerektiren alanlarda kayan nokta aritmetiği kritik bir öneme sahiptir. Ancak mevcut çözümler, genellikle belirli platformlarla sınırlı kalmakta ve esneklikten yoksundur. Bu durum, performansın optimize edilememesi ve donanım kaynaklarının verimli kullanılamaması gibi sorunlara yol açmaktadır.

Mevcut ticari kayan nokta işlem birimleri (FPU) genellikle VHDL tabanlı olup, yalnızca belirli FPGA platformlarında kullanılabilir durumdadır. Bunun yanı sıra, bu çözümler büyük donanım kaynakları gerektirdiğinden, düşük performanslı gömülü sistemlerde uygulanabilirlikleri sınırlıdır. Ayrıca, 16-bit, 32-bit ve 64-bit çözünürlük desteği sağlamayan çözümler, bazı gömülü sistemlerde enerji verimliliğini düşürmektedir.

Bu sınırlamalar göz önünde bulundurulduğunda, platformdan bağımsız, modüler, optimize edilmiş bir kayan nokta aritmetiği çözümünün geliştirilmesi büyük bir gereksinim haline gelmiştir.

Bu çalışmada geliştirilen Verilog tabanlı sentezlenebilir IP core kütüphanesi, gömülü sistemlerde kayan nokta aritmetiği hesaplamalarını daha verimli hale getirmeyi hedeflemektedir. IP core kütüphanesi, farklı çözünürlük seçenekleri (16-bit, 32-bit ve 64-bit) sunarak donanım kaynaklarının daha verimli kullanılmasını sağlamaktadır. Bu çözüm, düşük enerji tüketimi ve hızlı işlem kapasitesi sunarak, özellikle kaynakları sınırlı olan gömülü sistemlerde performansı artırmayı hedeflemektedir. Ayrıca, bu IP core kütüphanesi platform bağımsız olacak şekilde tasarlanmış olup, farklı FPGA cihazlarında rahatlıkla kullanılabilecektir.

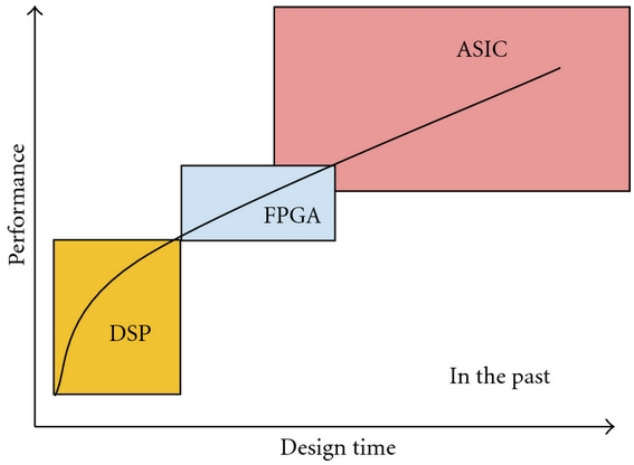
Bu çalışmanın bir diğer amacı, mevcut sistemlere göre daha verimli ve esnek bir IP core çözümü geliştirmektir. Mevcut çözümler genellikle optimize edilmemiş yapılara sahip olduğu için, geliştirilen bu kütüphane sayesinde gömülü sistem tasarımcıları, yüksek doğruluk ve verimlilik gerektiren işlemleri daha hızlı bir şekilde gerçekleştirebileceklerdir.

Gömülü sistemlerde kayan nokta aritmetiği üzerine yapılan araştırmalar, bu alanda önemli gelişmeler sağlamıştır. Ancak, mevcut literatürde yer alan çözümler genellikle belirli platformlara bağımlı olup, verimlilik ve esneklik açısından sınırlıdır. FPGA tabanlı kayan nokta birimleri, paralel işlem yapabilme kabiliyetleri ve enerji verimliliği sağlamaları nedeniyle sıklıkla tercih edilmektedir. Bununla birlikte, mevcut IP core çözümleri genellikle optimize edilmemiş yapılar içerir ve platformdan bağımsız çalışma yetenekleri sınırlıdır. Bu çalışmada geliştirilen IP core kütüphanesi, literatürdeki bu eksiklikleri gidererek, platform bağımsız bir yapı sunmaktadır.

Bu çalışmanın katkıları arasında, kayan nokta aritmetiği işlemlerinin hızlandırılması, sistem kaynaklarının verimli kullanılması ve platform bağımsız bir çözüm sunulması yer almaktadır. Geliştirilen IP core kütüphanesi, mevcut çözümlere kıyasla daha esnek ve optimize edilmiş bir yapıya sahip olup, farklı uygulama senaryolarına rahatlıkla entegre edilebilecektir.

BÖLÜM 2. LİTERATÜR ARAŞTIRMASI

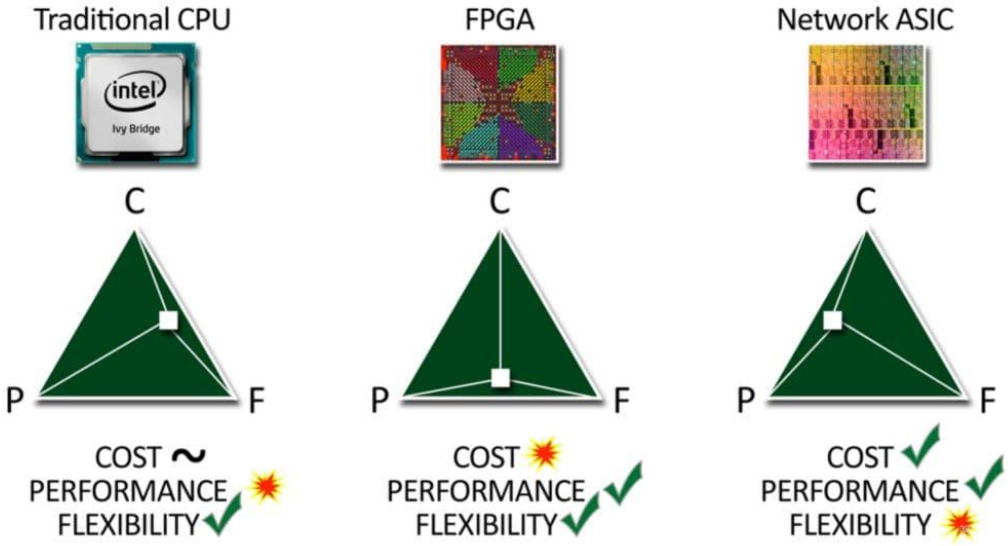
FPGA'lar, esneklikleri, paralel işlem yetenekleri ve düşük enerji tüketimi ile gömülü sistemlerde ve diğer yüksek performans gerektiren uygulamalarda öne çıkan bir donanım platformudur. FPGA'ların temel rakipleri arasında ASIC (Application-Specific Integrated Circuit) ve DSP (Dijital Sinyal İşlemcileri) bulunmaktadır. Ancak FPGA'lar, yeniden programlanabilir yapıları sayesinde ASIC'lerden farklı olarak esneklik sunar. ASIC'ler yüksek performans sağlasa da bir kez üretildikten sonra yeniden programlanma imkânı bulunmaz. Bu da donanımın değişmesi gerektiğinde büyük bir dezavantaj yaratır. DSP'ler ise yoğun matematiksel işlemler için tasarlanmış olsalar da FPGA'ların paralel işlem yeteneği ile kıyaslandığında işlem kapasitesi daha sınırlıdır. Bu sebeplerle FPGA'lar, özellikle paralel işlem yeteneği ve düşük enerji tüketimi gerektiren sinyal işleme, yapay zeka ve bilimsel hesaplamalar gibi uygulamalarda sıklıkla tercih edilir.



Şekil 2.1: DSP-FPGA ve ASIC gömülü işlemcilerin karşılaştırılması

FPGA'lar, bilimsel hesaplamalardan görüntü işleme, yapay zeka ve dijital sinyal işlemeye kadar geniş bir kullanım alanında etkin bir şekilde kullanılmaktadır. FPGA'ların esnekliği ve paralel işlem yapabilme yeteneği, özellikle yoğun hesaplama gerektiren uygulamalarda performansı artırmaktadır.

FPGA tabanlı çözümler, hız ve doğruluk açısından büyük avantajlar sunar ve ASIC'lere kıyasla daha esnek, yeniden programlanabilir yapıları sayesinde geniş bir uygulama yelpazesinde kullanılabilir.



Şekil 2.2: ASIC vs. SoC vs. FPGA

Kayan nokta aritmetiği ise büyük ya da küçük sayılar üzerinde hassas hesaplamalar yapılmasına olanak sağlayan bir matematiksel hesaplama yöntemidir. IEEE 754 standardına dayanan bu aritmetik türü, sabit nokta aritmetiğine göre çok daha geniş bir dinamik aralık sağlar ve özellikle yüksek doğruluk gerektiren bilimsel hesaplamalar, finansal analizler, bilgisayar grafikleri ve yapay zeka gibi alanlarda tercih edilir.

Kayan nokta aritmetiği, bilimsel hesaplamalarda, özellikle de diferansiyel denklemlerin çözümünde, Fourier dönüşümlerinde, sinyal işleme ve yapay zeka uygulamalarında kritik bir rol oynamaktadır.

FPGA'lar üzerinde kayan nokta aritmetik birimlerinin geliştirilmesi üzerine yapılan çalışmalar, bu donanım platformunun esnekliğini ve yüksek performans sunma yeteneğini ortaya koymaktadır. Dereli (2020) tarafından yapılan çalışma, FPGA üzerinde IEEE 754 standardına dayalı bir rastgele sayı üreteci (PRNG) geliştirilmiştir. Bu çalışma, doğrusal geri beslemeli kaydedici (LFSR) kullanarak düşük enerji tüketimi ile yüksek performans sağlamış ve rastgele sayı üreteci, kriptografi ve yapay zeka gibi alanlarda başarıyla uygulanmıştır. Xilinx Nexys 4 DDR FPGA cihazında gerçekleştirilen bu çalışma, IEEE 754 standardına dayalı kayan nokta birimlerinin FPGA üzerinde verimli bir şekilde kullanılabileceğini göstermektedir.

Bir başka çalışma olan Öztekin (2022)'nin çalışması, FPGA üzerinde 16-bit kayan nokta aritmetik birimlerinin tasarımını ele almaktadır. Bu çalışma, düşük enerji tüketimi ve yüksek doğruluk sağlamak amacıyla geliştirilmiş olup, sinyal işleme ve görüntü işleme gibi alanlarda uygulamalı olarak test edilmiştir. FPGA'nın programlanabilir yapısı sayesinde farklı çözünürlüklerde kayan nokta aritmetik birimlerinin geliştirilmesi ve bu birimlerin geniş veri setlerinde test edilmesi çalışmanın temel katkılarındandır. 16-bit kayan nokta aritmetik birimlerinin yüksek performanslı olması, gömülü sistemlerde daha düşük enerji tüketimi ile daha verimli işlemler yapılmasını sağlamaktadır.

Benzer şekilde, Akpınar (2020) tarafından yapılan çalışmada FPGA üzerinde IEEE 754 standardına dayalı olarak 32-bit kayan nokta aritmetik birimleri geliştirilmiştir. Bu çalışma, sinyal işleme ve bilimsel hesaplamalar gibi alanlarda yoğun matematiksel işlemlerin FPGA üzerinde verimli bir şekilde gerçekleştirilebileceğini göstermiştir. FPGA'nın paralel işlem yapabilme kapasitesi sayesinde, kayan nokta aritmetiği işlemleri hızlandırılmış ve düşük enerji tüketimi ile bu işlemler gerçekleştirilmiştir. Çalışmada FPGA üzerinde gerçekleştirilen simülasyonlar, yüksek doğruluk ve performans elde edilmesini sağlamıştır.

Kayan nokta aritmetiği, finansal analizler, bilgisayar grafikleri ve bilimsel hesaplamalar gibi geniş bir kullanım alanına sahiptir. Yıldız (2022), bu kapsamda IEEE 754 standardına dayalı kayan nokta aritmetiğinin çeşitli kullanım alanlarını detaylı şekilde ele almıştır. Çalışmada, büyük veri setlerinin hızlı ve doğru bir şekilde işlenmesi gereken alanlarda kayan nokta aritmetiğinin etkinliği vurgulanmıştır. Özellikle sinyal işleme ve görüntü işleme gibi alanlarda kullanılan IEEE 754 tabanlı kayan nokta birimlerinin enerji verimliliğini artırdığı ve performansı optimize ettiği gösterilmiştir. Bu çalışma, IEEE 754 standardına dayalı kayan nokta aritmetiği kullanımı konusunda geniş bir bakış açısı sunmaktadır ve tezinize katkı sağlayabilecek önemli bulgular içermektedir.

FPGA’ların programlanabilir yapısı, donanım üzerinde IP (Intellectual Property) core’ların kullanılmasına olanak tanır. IP core’lar, belirli bir işlevi gerçekleştirmek için önceden tasarlanmış ve test edilmiş donanım modülleridir. FPGA üzerinde IP core’lar kullanmak, tasarım sürecini hızlandırır ve sistemin belirli gereksinimlerine uygun özelleştirilmiş çözümler oluşturulmasını sağlar. Verilog veya VHDL gibi donanım tanımlama dilleri kullanılarak programlanan IP core'lar, farklı FPGA platformlarında kullanılabilir ve işlem performansını artırır.

Geliştirilen IP core kütüphanesi, gömülü sistemlerde enerji verimliliğini artıran ve işlem hızını optimize eden çözümler sunmaktadır.

FPGA'lar, gömülü sistemlerde, sinyal işleme ve bilimsel hesaplamalarda sıkça kullanılan programlanabilir donanım platformlarıdır. Paralel işlem yapabilme yetenekleri, düşük enerji tüketimleri ve esneklikleri ile ASIC ve DSP gibi rakip teknolojilerden ayrılırlar. Kayan nokta aritmetiği, büyük veri setlerinde doğruluğu koruyarak yüksek hızda hesaplamalar yapılmasına olanak tanır ve bu, finansal analizlerden yapay zeka uygulamalarına kadar geniş bir kullanım yelpazesinde önemlidir. IEEE 754 standardına dayalı kayan nokta aritmetik birimleri, FPGA üzerinde geliştirilen IP core'lar ile yüksek performans ve enerji verimliliği sağlamaktadır.

BÖLÜM 3. SAYISAL TASARIM

3.1. FPGA (Field-Programmable Gate Array)

FPGA (Field-Programmable Gate Array), dijital devrelerin esnek ve yeniden programlanabilir donanım yapılarıdır. FPGA'lar, donanım tasarımcılarının özel entegre devreler (ASIC'ler) yerine hızlı ve esnek bir şekilde dijital devreleri gerçekleştirebilmelerine olanak tanır. FPGA'nın en büyük avantajı, üzerine yazılan devrenin herhangi bir zamanda değiştirilebilir olmasıdır. FPGA'lar dijital devrelerin donanım seviyesinde tasarlanmasına ve programlanmasına imkan tanıyan çok sayıda programlanabilir mantık bloğu (Configurable Logic Blocks - CLB), giriş/çıkış blokları (Input/Output Blocks - IOB), ve bu blokları birbirine bağlayan geniş bir interkoneksiyon ağı içerir. Bu yapısı sayesinde, tasarımcılar FPGA'ları farklı donanım fonksiyonlarına göre programlayabilir ve geniş bir uygulama yelpazesi için optimize edebilirler.

FPGA'lar, dijital sinyal işleme, görüntü işleme, yapay zeka, kriptografi, veri işleme gibi çok yüksek hız ve paralel işlem kapasitesi gerektiren alanlarda yaygın olarak kullanılır. FPGA'ların içinde, birden fazla işlem aynı anda yapılabilir, bu da paralel işlem yapabilme yeteneği sayesinde işlem hızını önemli ölçüde artırır. FPGA, sabit donanım mimarileri kullanılarak çözülmesi zor olan karmaşık hesaplamalar ve algoritmalar için büyük avantaj sağlar.

FPGA'nın yapısındaki başlıca bileşenler şunlardır:

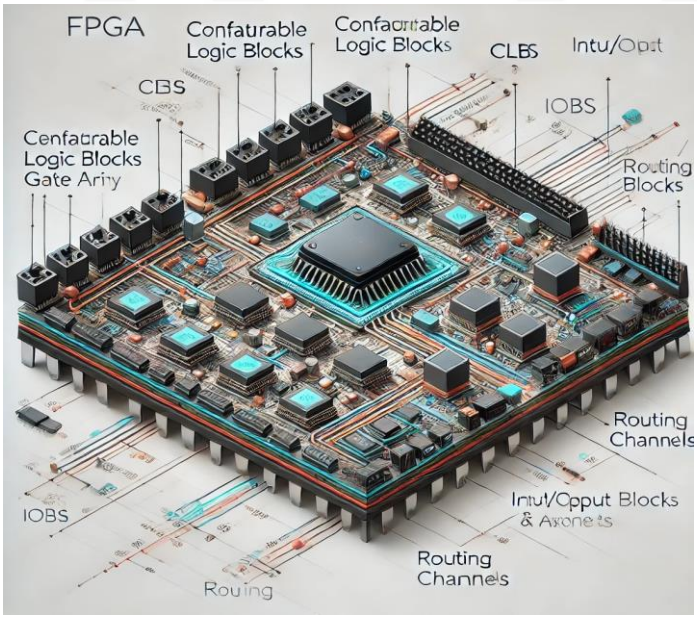
- Configurable Logic Blocks (CLBs): FPGA içindeki temel işlem birimleridir. Bu bloklar, basit mantık kapılarından daha karmaşık işlemler yapabilen küçük devreleri içerir. Bu yapı sayesinde devreler mantıksal işlemleri gerçekleştirmek için yeniden programlanabilir.

- Input/Output Blocks (IOBs): FPGA'nın dış dünya ile etkileşime geçmesini sağlayan giriş/çıkış portlarıdır. Bu bloklar, FPGA'nın çevre birimleri ve diğer sistemlerle veri alışverişi yapabilmesine olanak tanır.

- Interconnects (Routing Channels): FPGA içindeki farklı CLB'leri ve IOB'leri birbirine bağlayan esnek bir bağlantı ağıdır. Bu bağlantı ağı, FPGA'nın programlanabilir yapısını mümkün kılar ve farklı işlemleri yerine getirebilmesi için devrenin farklı kısımlarını birbirine bağlar.

FPGA'ların geniş uygulama alanları ve yüksek performansı, araştırma ve endüstri projelerinde yaygın bir şekilde kullanılmalarını sağlar. Özellikle hızın ve paralel işlemin ön planda olduğu projelerde FPGA'lar, tasarımcılara büyük esneklik sunar. Tasarımcılar, donanım seviyesinde optimize edilmiş çözümler geliştirebilir ve bu çözümleri ihtiyaç duyduklarında güncelleyebilirler.

Aşağıda FPGA'nın temel yapısını ve bu yapının içindeki önemli bileşenleri gösteren bir diagram bulunmaktadır. Bu diagram, FPGA'nın nasıl çalıştığını ve mantık blokları, giriş/çıkış blokları ve bağlantı yollarının nasıl birbirine bağlandığını görsel olarak açıklar.



Şekil 3.1: Ana bileşenlerini gösteren bir FPGA'nın şematik diyagramı

3.1.1. FPGA ve Diğer Donanım Seçeneklerinin Karşılaştırması

FPGA, ASIC (Application-Specific Integrated Circuit) ve CPLD (Complex Programmable Logic Device) gibi dijital devre tasarımı için kullanılan diğer donanım platformlarıyla karşılaştırıldığında, her birinin farklı avantaj ve dezavantajları vardır. Bu karşılaştırma, donanım projelerinde hangi platformun kullanılacağına karar verirken kritik bir rol oynar. FPGA, esneklik, programlanabilirlik ve yüksek paralel işlem kapasitesi gibi avantajlarıyla öne çıkarken, diğer seçenekler de belirli uygulamalarda daha uygun olabilir. Aşağıda FPGA, ASIC ve CPLD'lerin özellikleri ve aralarındaki farklar detaylandırılacaktır.

FPGA'lar, mantık kapıları ve diğer dijital devre elemanlarının kullanıcı tarafından programlanabilir olduğu esnek ve yeniden yapılandırılabilir cihazlardır. FPGA'lar birçok farklı uygulamada kullanılır, çünkü her seferinde yeniden programlanabilir ve optimize edilebilir. FPGA'ların başlıca özellikleri şunlardır:

1. **Programlanabilirlik:** FPGA'lar, tasarım süreci boyunca birden fazla kez programlanabilir. Donanım üzerine yazılan devrelerin, yazılım güncellemeleri gibi değiştirilebilir olması büyük bir esneklik sağlar. Bu özellik, özellikle prototipleme ve araştırma projelerinde çok kullanışlıdır.
2. **Yüksek Paralellik:** FPGA'lar, paralel işlem yapabilme kabiliyeti sayesinde çok sayıda işlemi aynı anda gerçekleştirebilir. Bu özellik, büyük veri setlerini işleme, sinyal işleme ve görüntü işleme gibi yüksek hız ve paralel işlem gerektiren uygulamalarda büyük bir avantaj sunar.
3. **Hızlı Prototipleme:** FPGA'lar, donanım tasarımcılarına hızlı bir şekilde prototip oluşturma imkanı sağlar. Tasarımcılar, bir devreyi simüle edebilir, test edebilir ve FPGA üzerinde gerçek zamanlı olarak çalıştırabilirler. Bu, tasarım döngüsünü hızlandırır.
4. **Donanım Üzerinden Yazılım İyileştirmesi:** Yazılımın yavaş olduğu yerlerde donanım seviyesinde hızlandırma yapmak mümkündür. FPGA'lar, karmaşık algoritmaların donanım üzerinden hızlandırılmasını sağlar.

5. **Yüksek Maliyet:** FPGA’lar genellikle ASIC’lere kıyasla daha pahalıdır. Ancak üretim süreçlerinin karmaşıklığı göz önüne alındığında, düşük hacimli üretimlerde FPGA’lar daha uygun maliyetli olabilir.

ASIC’ler, belirli bir uygulama için özel olarak tasarlanmış entegre devrelerdir. ASIC tasarımı, doğrudan bir devrenin silikon üzerine işlenmesini içerir. ASIC’ler, genellikle büyük ölçekli üretimler için daha uygundur, çünkü tasarım bir kez tamamlandıktan sonra her birim maliyeti düşüktür. Ancak bu esneklikten yoksundur ve bir kez üretildiğinde değiştirilemez.

1. **Yüksek Performans:** ASIC’ler, belirli bir uygulama için optimize edilebildikleri için en yüksek performansı sağlarlar. Tasarım doğrudan donanımda uygulandığı için hız ve verimlilik en üst düzeye çıkar.
2. **Düşük Enerji Tüketimi:** ASIC’ler, belirli bir işlev için optimize edildiklerinden genellikle daha düşük enerji tüketirler. Bu, enerji verimliliği gerektiren uygulamalar için büyük bir avantaj sağlar.
3. **Yüksek Geliştirme Maliyeti:** ASIC tasarımı, FPGA’ya kıyasla çok daha maliyetlidir ve geliştirme süreci çok daha uzun sürer. Bir tasarımın üretilmesi ve test edilmesi uzun süre alabilir. Ayrıca bir hata olması durumunda, yeniden üretim son derece maliyetli olabilir.
4. **Esneklik Eksikliği:** ASIC’ler bir kez üretildiğinde değiştirilemez. Eğer tasarımda bir hata varsa, yeni bir üretim döngüsü gereklidir. Bu nedenle ASIC tasarımı, nihai ve çok iyi doğrulanmış projeler için uygundur.
5. **Yüksek Üretim Hacmi Gereksinimi:** ASIC üretimi, genellikle büyük hacimlerde üretim yapıldığında maliyet etkin hale gelir. Küçük üretim hacimleri için uygun değildir.

CPLD’ler, FPGA’lara göre daha küçük ve daha az karmaşık dijital sistemlerin programlanabilir donanım çözümüdür. FPGA’lara göre daha az esneklik sunar ve genellikle daha küçük projeler için kullanılır.

1. **Kısıtlı Kaynaklar:** CPLD'ler, FPGA'lara göre daha sınırlı programlama kaynaklarına sahiptir ve daha basit devrelerin uygulanmasında kullanılır.

CPLD’ler genellikle düşük karmaşıklıktaki projelerde veya sabit fonksiyonlar için uygundur.

2. **Daha Düşük Maliyet:** CPLD'ler, FPGA'lara kıyasla daha uygun maliyetlidir, ancak karmaşık uygulamalarda performansları yetersiz kalabilir.
3. **Daha Az Esneklik:** CPLD'ler, FPGA kadar programlanabilir olmasına rağmen, genellikle çok büyük ölçekli projelerde yeterli olmaz. CPLD’ler, düşük sayıda lojik kapı içerir ve dolayısıyla daha basit dijital tasarımlar için uygundur.

Tablo 3.1. FPGA ve Diğer Seçenekler Arasındaki Karşılaştırma

Özellik	FPGA	ASIC	CPLD
Esneklik	Yeniden programlanabilir, çok esnek	Bir kez üretildikten sonra değiştirilemez	Sınırlı esneklik
Paralel İşlem Yeteneği	Yüksek	Orta	Düşük
Performans	Yüksek, ancak ASIC’e göre daha düşük	En yüksek	Orta
Enerji Verimliliği	Orta	En verimli	Orta
Geliştirme Süresi	Kısa, hızlı prototipleme imkanı	Çok uzun	Kısa
Maliyet	Yüksek (kısa vadede)	Düşük (büyük hacimlerde)	Düşük
Üretim Hacmi	Düşük-orta hacimli üretim için uygun	Yüksek hacimli üretim için uygun	Küçük projelerde uygundur

3.1.2. FPGA’yı Tercih Etme Sebeplerimiz

1. **Esneklik ve Yeniden Programlanabilirlik:** FPGA'ların sunduğu en büyük avantajlardan biri, yeniden programlanabilir olmalarıdır. Projenin geliştirme aşamalarında donanımın çeşitli versiyonlarını test etmek ve hataları gidermek mümkündür. ASIC'lerin aksine, FPGA'lar bir kez programlandığında bile değiştirilebilir, bu da daha dinamik projeler için idealdir.
2. **Yüksek Paralellik:** FPGA'lar paralel işlem yapma kabiliyetleri ile öne çıkar. Yüksek hız ve performans gerektiren uygulamalarda, birden fazla işlemi aynı anda gerçekleştirebilir. Bu, özellikle kayan noktalı işlemler ve yoğun matematiksel hesaplamalar için avantaj sağlar.

3. **Hızlı Prototipleme:** FPGA’lar, tasarımcıların donanım üzerinde hızlı bir şekilde prototip oluşturmasına olanak tanır. FPGA ile geliştirilen bir devre, yazılım gibi hızlıca güncellenebilir ve test edilebilir. Projeyi geliştirme ve test etme süreci diğer çözümlere göre çok daha kısa sürede tamamlanabilir.
4. **Düşük Üretim Hacmi:** Büyük ölçekli üretim gerektirmeyen projeler için FPGA’lar daha uygun maliyetli olabilir. ASIC tasarımında olduğu gibi yüksek hacimli üretim maliyetleri yoktur. Küçük üretim hacimleri veya araştırma projeleri için maliyet etkin bir çözümdür.
5. **Performans ve Esneklik Dengesi:** FPGA’lar, ASIC kadar yüksek performans sunmasa da, performans ve esneklik arasında iyi bir denge sunar. Bu denge, projelerde donanımın esneklikle değiştirilebilmesini sağlarken, yüksek işlem kapasitesi ve hız avantajı sunar.
6. **Sürekli Güncellenebilirlik:** FPGA’lar, değişen proje gereksinimlerine göre sürekli olarak güncellenebilir. Donanım üzerinde değişiklikler yapılabilir, böylece hata düzeltmeleri ve performans iyileştirmeleri çok daha kolay bir şekilde gerçekleştirilir.

Bu sebeplerle, projede FPGA kullanmak, esneklik, hızlı prototipleme ve paralel işlem gücü gerektiren bir proje için en uygun çözüm olarak seçilmiştir.

3.2. FPGA programlama

FPGA programlamak, bir donanım devresini dijital olarak tasarlamak ve bu tasarımı FPGA üzerinde çalıştırmak anlamına gelir. FPGA’yı programlamak için öncelikle donanım tanımlama dillerinde (HDL) tasarım yapılır ve ardından bu tasarım bir sentezleme aracı ile FPGA'ya yüklenir. FPGA programlama süreci, yazılım geliştirmeye benzer bir akış izler, ancak doğrudan donanım seviyesinde çalıştığı için zamanlama, paralellik ve kaynak yönetimi gibi konulara daha fazla dikkat edilmesi gerekir.

FPGA Programlamak İçin Gereken Adımlar

1. **Donanım Tanımlama Dili (HDL) ile Tasarım**

FPGA'ların programlanması için kullanılan en yaygın diller Verilog ve VHDL’dir. Bu diller, donanımın işleyişini tarif etmek için kullanılır.

Yani, dijital devreyi yazılı bir dille tanımlar ve tasarım sürecini başlatırsınız. FPGA'larda donanım tasarımı yapmak, yazılım programlamaya benzese de, dijital devre elemanlarını ve zamanlamayı doğrudan kontrol ettiğiniz için farklı bir mantık gerektirir.

- Verilog: Sözdizimi C programlama diline benzer ve öğrenmesi nispeten daha kolaydır. Dijital devrelerin hem yapısal hem de davranışsal tanımlamaları yapılabilir.

- VHDL: Daha katı bir sözdizimine sahip olup, daha ayrıntılı bir tanımlama sunar. Özellikle büyük ve karmaşık sistemlerde kullanılır.

2. FPGA Geliştirme Aracı Seçimi

FPGA'yı programlamak için tasarımınızı yazdıktan sonra, bu tasarımı donanım seviyesine dönüştürmek için bir geliştirme aracına ihtiyaç vardır. Xilinx ve Intel gibi FPGA üreticileri, kendi FPGA'larına uygun geliştirme araçları sağlar.

- Xilinx Vivado: Xilinx FPGA'ları programlamak için en yaygın kullanılan geliştirme ortamıdır. Vivado, FPGA tasarımı, simülasyonu, doğrulaması ve sentezi için kapsamlı bir araç seti sunar. Kullanıcı, donanım tasarımını yazılı olarak tanımlar, simüle eder ve devreyi FPGA'ya yükler. Vivado, özellikle büyük ve karmaşık FPGA projeleri için uygundur.

- Xilinx ISE: Xilinx'in önceki FPGA geliştirme aracı olan ISE, daha eski FPGA'lar için kullanılır. Vivado'nun gelmesiyle birlikte daha çok eski projeler için tercih edilir.

- Intel Quartus: Intel'in FPGA geliştirme aracı olan Quartus, Altera FPGA'lar için kullanılır. Quartus, Vivado'ya benzer şekilde FPGA tasarımında yazılımın donanıma dönüştürülmesi sürecini sağlar.

3. Tasarımın Sentezlenmesi

Yazılı olarak tanımlanan dijital devre, geliştirme ortamı aracılığıyla sentezlenir. Sentezleme işlemi, HDL ile yazılan devrenin mantık kapılarına, flip-floplara ve FPGA'nın programlanabilir mantık bloklarına dönüştürülmesini sağlar. Sentezleme, devrenin FPGA üzerinde nasıl çalışacağını belirleyen kritik bir aşamadır.

4. **Yerleştirme ve Yönlendirme (Place and Route)**

Sentezleme işleminden sonra, geliştirme aracı devrenin FPGA üzerindeki mantık bloklarına nasıl yerleştirileceğini ve bu bloklar arasındaki bağlantıların nasıl yapılacağını hesaplar. Bu adımda, devrenin fiziksel FPGA mimarisine en uygun şekilde yerleştirilmesi sağlanır.

5. **Simülasyon ve Doğrulama**

Tasarımın donanım üzerinde doğru çalışıp çalışmadığını anlamak için simülasyon yapılır. Bu aşamada giriş sinyalleri uygulanarak devrenin beklenen çıktıları verip vermediği kontrol edilir. Simülasyon sonucunda hatalar tespit edilirse tasarımda düzeltmeler yapılır.

6. **Programlama ve FPGA'ya Yükleme**

Tüm testler ve simülasyonlar başarılı olduktan sonra tasarım, geliştirme aracı ile FPGA'ya yüklenir. FPGA, USB veya JTAG gibi bağlantılarla programlanabilir. Programlama işlemi, FPGA'yı donanım düzeyinde yeni bir devre ile çalıştırmaya başlar.

3.3. Xilinx Vivado

Xilinx Vivado, Xilinx FPGA'larını programlamak için kullanılan en gelişmiş yazılım geliştirme aracıdır. Vivado, FPGA tasarımı için tam bir geliştirme döngüsü sağlar. Bu döngü, tasarımın HDL ile yazılması, simüle edilmesi, sentezlenmesi ve FPGA üzerine yüklenmesi adımlarını içerir. Vivado, tasarımcılara hem davranışsal modelleme hem de zamanlama analizleri yapabilme imkanı sunar.

Vivado, aşağıdaki başlıca özelliklere sahiptir:

- Tasarım Yöneticisi: Tasarımcılar projelerini yönetebilir, kodları düzenleyebilir ve doğrulama süreçlerini gerçekleştirebilir.

- Simülasyon Aracı: Tasarımın davranışsal ve zamanlama simülasyonlarını yaparak doğrulama sağlar.

- Sentez Aracı: HDL ile yazılan kodun FPGA üzerinde donanım bloklarına çevrilmesini sağlar.

- Yerleştirme ve Yönlendirme: Tasarımın FPGA üzerindeki fiziksel kaynaklara yerleştirilmesini ve mantık blokları arasındaki bağlantıların kurulmasını sağlar.

- Donanım Hata Ayıklama: Tasarımcıların FPGA içindeki sinyalleri ve devre davranışlarını gerçek zamanlı olarak izlemelerine yardımcı olur.

Kullanılan Donanım Tanımlama Dilleri (HDL'ler):

1. Verilog:

Verilog, dijital devrelerin yapısal ve davranışsal modellemesi için yaygın olarak kullanılan bir HDL'dir. C diline benzer yapısı nedeniyle öğrenmesi kolaydır. Verilog, daha çok endüstride kullanılır ve FPGA tasarımlarında sıkça tercih edilir.

2. VHDL:

VHDL, daha detaylı ve güçlü bir dil olup, daha karmaşık sistemlerin tasarımı için tercih edilir. Akademik çevrelerde ve bazı büyük projelerde yaygın olarak kullanılır. Tip kontrolü ve katı sözdizimi nedeniyle daha güvenilir bir dil olarak bilinir.

3. SystemVerilog:

SystemVerilog, Verilog'un bir genişletmesidir ve özellikle doğrulama süreçleri için geliştirilmiştir. FPGA tasarımlarında hem tasarım hem de doğrulama için kullanılan ileri düzey bir HDL'dir.

Neden FPGA Programlama?

FPGA programlama, esneklik ve paralel işlem yetenekleri nedeniyle birçok projede tercih edilmektedir. FPGA'lar, hızlı prototipleme, donanım hızlandırma ve özelleştirilebilir devreler tasarlamak için ideal platformlardır. FPGA'nın yeniden programlanabilir olması, araştırma ve geliştirme projelerinde sürekli iyileştirmeler yapma imkanı sunar.

FPGA programlama süreci, HDL ile tasarım yapma, tasarımı simüle etme ve donanım üzerinde çalıştırma adımlarını içerir. Bu süreç, Xilinx Vivado gibi gelişmiş araçlarla kolaylaştırılmıştır ve FPGA'ların sunduğu yüksek performans ve esneklik, onları birçok uygulama için tercih edilen bir çözüm haline getirir.

3.4. Simülasyon

FPGA'larda testler, tasarlanan devrenin doğru çalışıp çalışmadığını doğrulamak için kritik bir süreçtir. FPGA üzerinde tasarım doğrulama, hata tespiti ve performans değerlendirme aşamaları birkaç farklı test yöntemi ile gerçekleştirilir. Bu testler tasarımın gerçek donanım üzerinde nasıl davranacağını anlamak için kullanılır. FPGA test süreci genel olarak iki aşamada ele alınabilir: Simülasyon testleri ve donanım testleri (hardware-in-the-loop testing). Aşağıda, FPGA'da kullanılan temel test yöntemleri açıklanmaktadır:

1. Simülasyon Testleri

Simülasyon, FPGA devresini donanıma aktarmadan önce yazılım ortamında test etme sürecidir. Bu adım, devrenin davranışını modelleme ve mantıksal hataları düzeltme açısından çok önemlidir. Simülasyon testleri ile devre üzerinde herhangi bir hata olup olmadığını tespit edebilir, zamanlama gecikmeleri ve beklenen çıktıların doğruluğunu kontrol edebilirsiniz.

a. Davranışsal Simülasyon (Behavioral Simulation)

- Davranışsal simülasyon, tasarımın işlevselliğini zamanlama bilgisi olmaksızın doğrulamak için kullanılır. Bu aşamada, devreyi sadece mantıksal düzeyde test ederiz, yani devrenin girişlerine verilen değerlerle ilgili çıktıların doğru olup olmadığı kontrol edilir.

- Verilog veya VHDL ile yazılan kodun doğru bir şekilde mantıksal işlemleri yapıp yapmadığını doğrulamak için yapılır. Simülasyon sırasında farklı girişler verilerek devrenin beklenen sonuçları üretip üretmediği test edilir.

b. Zamanlama Simülasyonu (Timing Simulation)

- Zamanlama simülasyonu, devrenin davranışsal simülasyonunun ötesine geçerek, devrenin gerçek zamanlı çalışmasını modellemek için kullanılır. Bu test, FPGA'nın zamanlama gecikmelerini, sinyal yayılma sürelerini ve devre elemanlarının saat sinyaline tepki sürelerini içerir.

- Bu aşamada sinyallerin FPGA'daki fiziksel bağlantılar ve devre elemanları arasında nasıl geciktiği analiz edilir. Böylece devrenin belirli bir frekansta çalışıp çalışamayacağı kontrol edilebilir.

2. Testbenches Kullanımı

Testbench, Verilog veya VHDL dilinde yazılmış bir doğrulama ortamıdır. Tasarımcının FPGA tasarımını simülasyon sırasında test edebilmesi için, tasarlanan devreye giriş sinyalleri uygulayan ve bu sinyallere bağlı olarak devrenin çıktısını kontrol eden bir kod parçasıdır. Testbenches, tasarımın farklı koşullarda nasıl davrandığını test etmek için kullanılır. Testbench'in temel özellikleri şunlardır:

- Testbench, tasarlanan devreyi test etmek için giriş sinyallerini tanımlar ve bu sinyallere göre beklenen çıkışları kontrol eder.

- Çeşitli saat sinyalleri, girişler ve çıkışlar simüle edilir, böylece devrenin çeşitli senaryolarda nasıl çalıştığı test edilir.

- Her bir durumda devrenin ürettiği çıktılar beklenen sonuçlarla karşılaştırılır, böylece hata tespiti yapılabilir.

Örnek bir testbench kodu şu şekilde olabilir:

```
verilog
module testbench();
   reg clk, reset;
   reg [7:0] a, b;
   wire [7:0] result;
   // Test edilen modül örneklenir
   my_module uut (
      .clk(clk),
      .reset(reset),
      .a(a),
```

```
    .b(b),
    .result(result)
);
initial begin
    // Test senaryoları
    reset = 1; clk = 0; a = 8'b00000000; b = 8'b00000000;
    #10 reset = 0; a = 8'b00001010; b = 8'b00000101;  // İlk test girişi
    #10 a = 8'b00000100; b = 8'b00001111;             // İkinci test girişi
    #10 $finish;
end
always #5 clk = ~clk;  // Saat sinyali üretimi
endmodule
```

3. Donanım Testleri (Hardware-in-the-Loop Testing)

Simülasyon testleri tamamlandıktan sonra devrenin fiziksel olarak FPGA'ya programlanıp çalıştırıldığı donanım testleri yapılır. Bu testler, devrenin gerçek donanım üzerinde doğru bir şekilde çalışıp çalışmadığını anlamak için kritik önem taşır. Donanım testleri, FPGA'nın dış dünyaya bağlı giriş/çıkışlarla nasıl etkileşime geçtiğini gözlemler ve donanım üzerindeki gerçek hata senaryolarını tespit eder.

a. Gerçek Zamanlı Test (Real-Time Testing)

- FPGA'ya programlanan devrenin gerçek zamanlı olarak belirli bir donanım üzerinde nasıl çalıştığı test edilir. Örneğin, sensörlerden gelen sinyaller veya dış cihazlardan alınan veriler devreye uygulanarak, devrenin bu verilere nasıl yanıt verdiği gözlemlenir.

b. Debugging (Hata Ayıklama)

- FPGA tasarımında hata ayıklamak için Integrated Logic Analyzer (ILA) gibi FPGA içine entegre edilmiş hata ayıklama araçları kullanılır.

Bu araçlar sayesinde FPGA'daki mantıksal sinyaller gerçek zamanlı olarak izlenebilir. Böylece, FPGA içinde belirli sinyallerin nasıl hareket ettiği, hatalı sinyallerin nerede oluştuğu kolayca bulunabilir.

c. Donanımda Doğrulama (Hardware Verification)

- FPGA'ya yüklenen tasarım, daha büyük bir sistemin bir parçası olarak çalışıyorsa, sistemle olan bütünleşik davranışı da test edilir. Örneğin, bir iletişim protokolü uygulanıyorsa, FPGA'nın sistemin diğer donanım parçalarıyla doğru şekilde iletişim kurup kurmadığı kontrol edilir.

4. Kapsamlı Test (Exhaustive Testing)

Bazı durumlarda, tüm olası giriş kombinasyonlarını test etmek gerekebilir. Bu, özellikle güvenlik kritik sistemlerde önemlidir. Kapsamlı testler sırasında, devrenin olası tüm girişleri test edilerek her durumda doğru çıktının üretilip üretilmediği kontrol edilir.

5. Zamanlama ve Performans Analizi

Gerçek donanım üzerinde çalışan FPGA tasarımının zamanlama analizleri yapılır. Saat frekansı, sinyal yayılma süreleri ve sistem gecikmeleri hesaplanarak, devrenin belirli bir hızda güvenilir şekilde çalışıp çalışamayacağı belirlenir. FPGA tasarımı sırasında performans ve zamanlama optimizasyonları yapılarak en iyi sonuç alınmaya çalışılır.

6. Hata Toleransı Testleri

FPGA, karmaşık sistemlerde kullanıldığında, özellikle kritik uygulamalarda hata toleransı testleri yapılmalıdır. Bu testler, FPGA'nın beklenmeyen koşullar altında nasıl davrandığını ve sistemin hatalara karşı nasıl dayanıklı olduğunu değerlendirir. Örneğin, güç kesintileri, aşırı sıcaklık veya elektromanyetik girişim gibi senaryolarda FPGA'nın tepkisi test edilir.

FPGA tasarımında testler, tasarımın doğru ve verimli çalışmasını sağlamak için hayati bir rol oynar. Simülasyon testleri, tasarımın doğruluğunu yazılım ortamında doğrularken, donanım testleri devrenin gerçek donanım üzerinde nasıl davrandığını gösterir. Test süreçleri, hataları tespit etmek, performans optimizasyonları yapmak ve sistemi en iyi şekilde çalıştırmak için kullanılır. Bu test süreçleri, başarılı bir FPGA tasarımının temel adımlarından biridir.

3.5. Donanım Tanımlama Dilleri (HDL - Hardware Description Languages)

Donanım Tanımlama Dilleri (HDL - Hardware Description Languages), dijital devrelerin ve elektronik sistemlerin davranışlarını modellemek için kullanılan dillerin genel adıdır. FPGA'lerde kullanılan bu diller, donanım seviyesinde işlem yapmayı mümkün kılar.

Yani yazılımın aksine, bu diller doğrudan donanımı temsil eder ve devrelerin mantıksal kapı düzeyinde nasıl davranacağını tanımlar. FPGA programlama sürecinde kullanılan iki ana HDL dili vardır: Verilog ve VHDL.

Donanım Tanımlama Dillerinin (HDL) Önemi

HDL'ler, yazılım dillerine benzemekle birlikte, doğrudan donanımın işleyişini tarif eder. Geleneksel programlama dillerinden farklı olarak, HDL'ler paralel işlemleri ve zamanlamayı yönetir. Donanım tasarımcısı, HDL'yi kullanarak dijital sistemlerin işleyişini tarif eder ve bu tarif edilen yapı FPGA veya ASIC gibi donanımlara yüklenir. HDL'ler, dijital devrelerin yazılım gibi programlanmasını sağlar ancak çalıştırıldıkları zaman, devre mantığı fiziksel olarak donanım üzerinde işlem yapar.

HDL Dillerinin Temel Özellikleri

Paralel İşlem Desteği: HDL dillerinde yazılan kodlar paralel olarak çalışır. Yani, yazılan tüm işlemler aynı anda gerçekleşebilir. Bu, dijital devrelerin en temel özelliklerinden biridir ve FPGA'lar üzerinde birden fazla işlemin aynı anda gerçekleşmesini sağlar.

Zamanlama ve Gecikme: Dijital devrelerde sinyalin bir yerden başka bir yere gitmesi zaman alır. HDL dillerinde zamanlama (timing) kontrol edilir ve gecikme gibi unsurlar tasarlanabilir.

Hiyerarşik Yapı: HDL dillerinde modüler yapı mümkündür. Yani devreler, daha küçük bileşenlerden oluşan hiyerarşik bir yapı şeklinde tanımlanabilir. Bu, büyük ve karmaşık projelerde tasarımı daha kolay hale getirir.

Verilog Nedir?

Verilog, dijital devre tasarımı ve simülasyonu için kullanılan popüler bir HDL dilidir.

1984 yılında geliştirilen Verilog, özellikle endüstride geniş bir kabul görmüştür ve dijital devrelerin donanım seviyesinde modellenmesi için standart bir araç haline gelmiştir.

IEEE 1364 standardına uygun olan Verilog, FPGA ve ASIC tasarımlarında yaygın olarak kullanılır. Sözdizimi ve yapısı, yazılım dillerine, özellikle C diline benzer, bu da yazılım geliştiricilerinin Verilog'u hızlı bir şekilde öğrenmesini sağlar.

Verilog'un Özellikleri

C'ye Benzer Sözdizimi: Verilog'un en büyük avantajlarından biri, C gibi yazılım dillerine benzer sözdizimine sahip olmasıdır. Bu durum, yazılım geliştiricileri ve tasarımcılar için öğrenme sürecini kolaylaştırır.

Davranışsal ve Yapısal Modelleme: Verilog, hem dijital devrelerin davranışsal modellemesini (bir devrenin nasıl çalıştığını tanımlama) hem de yapısal modellemesini (devre elemanlarının birbirine nasıl bağlandığını tanımlama) yapabilir. Bu esneklik, Verilog'u dijital devre tasarımında güçlü bir araç haline getirir.

Modülerlik: Verilog'da tasarım modüler bir yapıya sahiptir. Tasarımlar küçük modüller halinde yazılabilir ve bu modüller birbirine bağlanarak daha büyük sistemler oluşturulabilir.

Simülasyon ve Sentez: Verilog, tasarımların yazılım ortamında simüle edilmesine ve donanım üzerinde sentezlenmesine olanak tanır. Bu, tasarımın doğru çalışıp çalışmadığının doğrulanmasını sağlar.

Verilog Sözdizimi ve Yapısı

Verilog, dijital devreleri modüler yapıda tanımlar. Her modül, giriş ve çıkış portlarına sahip olabilir ve bu portlar aracılığıyla diğer modüllerle etkileşim kurar. Verilog ile yazılan her modül, bir devrenin belirli bir parçasını temsil eder ve tasarımın her bir bileşeni, diğer bileşenlerle bağlantılı olarak tanımlanır.

Verilog ile bir basit AND kapısı tanımlamasını şu şekilde yapabiliriz:

```
module and_gate (input a, input b, output c);
  assign c = a & b;
```

```
endmodule
```

Bu basit örnekte, and_gate adında bir modül tanımlanmıştır. Bu modül, iki giriş (a ve b) alır ve bir çıkış (c) üretir. assign ifadesi ile Verilog, c çıkışının a ve b girişlerinin AND operatörü ile işlenmesi sonucunda elde edileceğini belirtir.

Verilog ayrıca always blokları ile karmaşık devre davranışlarını kontrol edebilir. Örneğin, bir flip-flop davranışı şu şekilde tanımlanabilir:

```
module flip_flop (input clk, input d, output reg q);
  always @(posedge clk)
    q <= d;
endmodule
```

Bu örnek, bir flip-flop devresinin davranışını tanımlar. Saat sinyali (clk) her yükseldiğinde (posedge), giriş d'nin değeri çıkışa (q) atanır.

Neden Verilog'u Seçtik?

FPGA programlamada Verilog'un tercih edilmesinin birçok nedeni vardır:

Öğrenmesi Kolay: Verilog, C diline benzer bir yapıya sahip olduğundan, yazılım geliştiriciler ve dijital tasarımcılar için öğrenmesi oldukça kolaydır. Basit ve anlaşılır sözdizimi sayesinde kısa sürede öğrenilip projelere entegre edilebilir.

Endüstride Yaygın Kullanım: Verilog, özellikle endüstride geniş bir kabul görmüştür. FPGA ve ASIC üreticileri tarafından sağlanan çoğu geliştirme aracı, Verilog'u destekler. Verilog'un endüstride yaygın olarak kullanılması, projelerde standardizasyonu ve uyumluluğu sağlar.

Hızlı Prototipleme: Verilog'un modüler ve basit yapısı, tasarımların hızlı bir şekilde prototiplenmesine olanak tanır. Dijital sistemler, Verilog ile kolayca simüle edilebilir ve hatalar hızla tespit edilip düzeltilebilir. Bu da projelerin geliştirme süresini kısaltır.

Modüler Tasarım: Verilog, büyük projelerde tasarımların küçük modüller halinde yazılmasına ve daha sonra bu modüllerin birleştirilmesine olanak tanır. Bu, projelerin daha yönetilebilir hale gelmesini sağlar ve tekrar kullanılabilir bileşenlerin oluşturulmasını kolaylaştırır.

Performans ve Zamanlama Yönetimi: FPGA gibi platformlarda paralel işlemler çok önemlidir. Verilog, dijital sistemlerde zamanlama ve paralellik gibi kritik unsurların yönetimini kolaylaştırır ve böylece donanım kaynaklarının verimli bir şekilde kullanılmasını sağlar.

Simülasyon Desteği: Verilog, güçlü simülasyon araçlarıyla donatılmıştır. Yazılan devrelerin önce yazılım ortamında test edilmesi ve sonrasında FPGA'ya yüklenmesi sürecinde Verilog'un sunduğu simülasyon desteği, tasarımın doğruluğunu garanti eder.

Verilog ile FPGA Programlamanın Avantajları

Esneklik ve Yeniden Programlanabilirlik: Verilog ile yazılan devreler, FPGA üzerinde esnek bir şekilde çalıştırılabilir ve yeniden programlanabilir. Bu, tasarımcıya tasarım üzerinde tam kontrol sağlar ve gerektiğinde değişiklikler yapma imkanı sunar.

Performans ve Doğruluk: Verilog, donanım seviyesinde yazılan kodların verimli ve doğru bir şekilde çalışmasını sağlar. Paralel işlem gücü ve zamanlama yönetimi ile karmaşık işlemler hızlı bir şekilde gerçekleştirilebilir.

FPGA Üreticileri Tarafından Destek: Xilinx, Intel gibi büyük FPGA üreticileri Verilog'u destekler ve Verilog kodlarını optimize etmek için çeşitli araçlar sunarlar. Bu da, Verilog'un FPGA projelerinde tercih edilmesini sağlar.

Verilog, dijital devrelerin donanım seviyesinde tanımlanması ve FPGA gibi yeniden programlanabilir donanımlar üzerinde çalıştırılması için güçlü bir araçtır. C benzeri sözdizimi, esnekliği ve geniş endüstri desteği nedeniyle, dijital devre tasarımı ve FPGA projelerinde yaygın olarak tercih edilir. Ayrıca hızlı prototipleme ve simülasyon imkanları, projelerin daha verimli geliştirilmesini sağlar. FPGA programlamada Verilog'u tercih etmemizin en önemli sebepleri, esnekliği, öğrenme kolaylığı ve projelerde sunduğu hızlı sonuçlar olarak öne çıkmaktadır.

3.6. FPGA'de IP Core Nedir?

FPGA’de IP Core (Intellectual Property Core), dijital devre tasarımlarında tekrar kullanılabilir mantık bloklarıdır. Bu bloklar, belirli bir işlevi yerine getiren ve tasarımcılar tarafından hazır olarak kullanılan modüllerdir. IP Core'lar, karmaşık işlevleri yeniden tasarlamak zorunda kalmadan, dijital devre tasarım sürecini hızlandırmak için kullanılır. IP Core’lar, genellikle FPGA üreticileri veya üçüncü taraflar tarafından sağlanır ve tasarımcılara hızlı prototipleme ve devre oluşturma imkanı sunar.

Bir IP Core, bir FPGA üzerinde belirli bir işlevi yerine getirmek için önceden tasarlanmış donanım bloklarından oluşur.

IP Core'lar, veri işleme, haberleşme, matematiksel hesaplamalar, arabellek yönetimi, arama algoritmaları gibi çok çeşitli alanlarda kullanılır. IP Core’lar, genellikle özelleştirilebilir ve tasarımcılar tarafından farklı projelerde kullanılabilir.

IP Core’un Avantajları

1. Zaman Kazandırır: Tasarımcıların sıfırdan bir devre oluşturmak yerine hazır bir modül kullanmasını sağlar. Bu, geliştirme sürecini hızlandırır ve tasarımcıların zamanlarını optimize etmelerine olanak tanır.

2. Test Edilmiş ve Güvenilir: IP Core'lar, genellikle test edilmiş ve güvenilir çözümler olarak sunulur. Bu, hatalı devre tasarımı riskini azaltır.

3. Performans Optimizasyonu: FPGA üreticileri tarafından sunulan IP Core'lar, genellikle FPGA platformları için optimize edilmiştir ve yüksek performans sunar.

4. Özelleştirilebilir: Bazı IP Core'lar, kullanıcıların ihtiyaçlarına göre özelleştirilebilir. Bu sayede, belirli bir işlev için uygun bir modül yapılandırılabilir.

3.6.1. IP Core’ların Yapısal Özellikleri

IP Core’lar genel olarak şu bileşenleri içerir:

- **Mantıksal İşlem Blokları (Logic Blocks):** Belirli bir işlevi yerine getiren ve FPGA'nın mantıksal kapılarına bağlanan işlem blokları.

- **Giriş/Çıkış Portları (I/O Ports):** Diğer donanımlarla veya modüllerle iletişim kurmak için kullanılan giriş/çıkış portları.
- **Saat Sinyali (Clock Signals):** IP Core'lar genellikle FPGA'nın saat sinyali ile senkronize çalışır. Saat sinyali, işlemlerin doğru zamanlamayla yapılmasını sağlar.
- **Kontrol ve Durum Kayıtları (Control and Status Registers):** IP Core'ların belirli bir işlevi yönetmek için kullandığı kontrol ve durum bilgilerini saklayan kayıtlar.

Tablo 3.2. IP Core Örnekleri ve Özellikleri

IP Core Adı	İşlevi	Kullanım Alanı	Açıklama
AXI Interconnect	Veri yolu bağlantısı	Yüksek hızlı veri iletimi	FPGA içinde farklı veri yollarını birbirine bağlar.
FIFO (First In First Out)	Bellek yönetimi	Veri tamponlama, sıraya alma	Verileri sıralı bir şekilde depolar ve okur.
Floating Point Unit (FPU)	Matematiksel hesaplamalar	Bilimsel hesaplamalar, görüntü işleme	Kayan nokta aritmetik işlemleri yapar.
Ethernet MAC	Ağ iletişimi	Haberleşme, internet bağlantısı	Ethernet protokolü ile veri iletimi sağlar.
DDR Memory Controller	Bellek yönetimi	Yüksek hızlı veri depolama	DDR RAM gibi harici bellekleri yönetir.
UART	Seri iletişim modülü	Veri iletişimi	Seri haberleşme protokollerini destekler.
PCI Express	Yüksek hızlı veri iletişimi	PCIe tabanlı veri yolları	Yüksek hızda veri iletimi için PCIe veri yolunu yönetir.
Cortex-M1 Processor	Mikroişlemci	Gömülü sistemler	FPGA içinde çalışabilen bir mikroişlemci IP Core'dur.
AES Encryption Core	Şifreleme işlemleri	Güvenlik, veri koruma	AES algoritmasına dayalı veri şifreleme ve deşifreleme sağlar.
DSP Core	Dijital sinyal işleme	Ses ve görüntü işleme, radar sistemleri	Dijital sinyal işleme algoritmalarını gerçekleştirir.

FPGA projelerinde IP Core'ları kullanarak işlevselliği artırabilir ve projelerini hızlandırabilirler. Örneğin, bir projede yüksek hızlı veri iletimi gerekiyorsa, hazır bir Ethernet MAC IP Core'u kullanarak tasarımı hızlandırmak mümkündür.

Aynı şekilde, karmaşık matematiksel işlemler için bir Floating Point Unit (FPU) IP Core'u kullanmak, tasarım sürecini kolaylaştırır.

FPGA üreticileri, kendi geliştirme araçlarıyla entegre olan geniş IP Core kütüphaneleri sunarlar. Xilinx Vivado veya Intel Quartus gibi geliştirme ortamlarında bu IP Core'lar kolayca projelere eklenebilir ve yapılandırılabilir. Ayrıca, bu IP Core'lar simülasyonlar ve donanım testleri sırasında doğrulanabilir.

FPGA programlama projelerinde IP Core kullanımı, tasarım sürecini hızlandırır, performansı artırır ve güvenilir çözümler sunar.

Özellikle hazır ve optimize edilmiş IP Core'lar, karmaşık işlevleri basitçe entegre etme olanağı sağlar. IP Core'lar, FPGA projelerinde kritik bir rol oynar ve hem zaman kazandırır hem de projelerin daha hızlı ve verimli geliştirilmesine olanak tanır.

3.7. Kayan Nokta (Floating Point) ve Sabit Noktalı (Fixed Point) İşlemler

Kayan nokta ve sabit noktalı sayı sistemleri, dijital sistemlerde ve bilgisayar bilimlerinde sayısal hesaplamalar için yaygın olarak kullanılan iki yöntemdir. Her iki sistem de sayıların ikili formatta temsil edilmesine olanak tanır ancak sayıları saklama ve işlemlerini yapma biçimlerinde önemli farklar vardır. Bu farklar, hesaplama gereksinimlerine, hassasiyete ve donanım kaynaklarının kullanılabilirliğine göre projelerde bir yöntemin diğerine tercih edilmesine neden olur. Aşağıda, kayan nokta ve sabit nokta işlemlerinin nasıl çalıştığını ve neden projede kayan nokta işlemlerinin tercih edildiğini açıklayan bir karşılaştırma yapılmıştır.

3.7.1. Sabit Noktalı Sayı Sistemi (Fixed Point)

Sabit noktalı sayı sistemi, sayının virgülünün her zaman sabit bir pozisyonda olduğu bir temsil şeklidir. Bu sistemde, sayının tam ve kesir kısımları sabit bir uzunluğa sahiptir. Sabit nokta sistemi, sayıları tam sayılar veya sabit ondalık noktaya sahip sayılar olarak temsil eder.

- **Temsil:** Sayı, belirli bir tam sayı ve kesirli bit sayısı kullanılarak temsil edilir. Virgülün pozisyonu her zaman sabittir ve donanım tarafından değiştirilmez.

- **Basitlik:** Sabit noktalı işlemler, kayan nokta işlemlerine göre daha az karmaşıktır ve daha hızlı hesaplama sağlar. Ancak, sayı aralığı ve hassasiyet sınırlıdır.
- **Kullanım Alanı:** Sabit noktalı işlemler, gömülü sistemler gibi hesaplama gücünün sınırlı olduğu alanlarda tercih edilir. Aynı zamanda donanım kaynaklarını verimli kullanma açısından avantaj sağlar.

3.7.2. Kayan Noktalı Sayı Sistemi (Floating Point)

Kayan noktalı sayı sistemi, sayının tam ve kesir kısmını esnek bir şekilde temsil eder. Kayan nokta sisteminde, sayının virgülü dinamik olarak kaydırılır ve çok büyük ya da çok küçük sayıları temsil etmek mümkün olur. IEEE 754 standardı, kayan nokta sayılarının bilgisayar ortamında nasıl temsil edileceğini belirleyen yaygın bir standarttır.

- **Temsil:** Sayı, işaret (sign), üslü (exponent) ve mantisa (mantissa) bileşenleri ile temsil edilir. Üslü değeri, sayının virgülünü kaydırarak çok büyük veya çok küçük sayıları temsil edebilmesine olanak tanır.
- **Geniş Sayı Aralığı ve Hassasiyet:** Kayan nokta sistemi, geniş bir sayı aralığında çok hassas hesaplamalar yapabilir. Bu, büyük veri setleri ve bilimsel hesaplamalar için idealdir.
- **Kullanım Alanı:** Kayan nokta sistemi, yüksek hassasiyet ve geniş sayı aralığı gerektiren uygulamalarda kullanılır. Bilimsel hesaplamalar, mühendislik simülasyonları, görüntü işleme ve yapay zeka gibi alanlarda yaygın olarak tercih edilir.

3.7.3. Kayan Nokta ve Sabit Nokta Karşılaştırması

Tablo 3.3. Kayan Nokta ve Sabit Nokta Karşılaştırması

Özellik	Kayan Noktalı Sistem (Floating Point)	Sabit Noktalı Sistem (Fixed Point)
Sayısal Aralık	Çok geniş bir aralıkta sayıları temsil edebilir.	Dar bir aralıkta sayıları temsil eder.
Hassasiyet	Daha hassas sonuçlar sağlar, özellikle küçük sayılarda.	Sınırlı hassasiyet, özellikle küçük sayılarda sınırlıdır.
Donanım Karmaşıklığı	Daha karmaşık ve daha fazla donanım kaynağı kullanır.	Daha basit ve donanım dostu.
Hız	Daha yavaş (kayan nokta aritmetiği daha karmaşıktır).	Daha hızlı, basit aritmetik işlemler.
Kullanım Alanları	Bilimsel hesaplamalar, görüntü işleme, yapay zeka.	Gömülü sistemler, basit veri işleme, kontrol sistemleri.
Hassas Sayılarla İşlem	Daha iyi performans gösterir (küçük ve büyük sayılar için).	Hassas olmayan işlemler için uygundur.
Enerji Tüketimi	Daha fazla enerji harcar.	Daha düşük enerji tüketir.

Neden Kayan Noktalı İşlemleri Seçtik?

1. Geniş Sayı Aralığı: Kayan nokta sistemi, çok büyük ve çok küçük sayıları aynı anda temsil etme yeteneğine sahiptir. Bu, özellikle bilimsel hesaplamalarda ve mühendislik projelerinde çok önemlidir. Sabit noktalı işlemler, sınırlı bir sayı aralığı sunar ve geniş bir aralıkta sayılar üzerinde işlem yapılması gerektiğinde yeterli olmaz.

2. Hassasiyet Gereksinimi: Projemiz gibi yüksek hassasiyetli hesaplamalar gerektiren sistemlerde, kayan noktalı işlemler daha güvenilir sonuçlar verir. Sabit nokta sistemi, küçük sayılarla çalışırken hassasiyet kaybı yaşatabilirken, kayan nokta sistemi daha düşük hata oranı sunar.

3. Bilimsel ve Mühendislik Hesaplamaları: Kayan nokta sistemi, özellikle mühendislik ve bilimsel hesaplamalar için optimize edilmiştir. Üst düzey matematiksel işlemler ve karmaşık algoritmalar kullanılırken, kayan nokta işlemleri, sabit nokta işlemlerine kıyasla çok daha güvenilirdir.

4. Paralel İşlem Yeteneği: FPGA üzerinde kayan nokta işlemleri donanım seviyesinde paralel olarak gerçekleştirilir.

Bu sayede, çok sayıda işlem aynı anda yapılabilir ve sistem performansı önemli ölçüde artırılabilir. Sabit noktalı işlemler daha az karmaşık olmasına rağmen, paralel işlem avantajlarından faydalanmada sınırlıdır.

5. Dinamik Yapı: Kayan nokta sistemi, verilerin dinamik olarak temsil edilmesini ve işlemlerin dinamik olarak uyarlanmasını sağlar. Sabit noktalı sistemde, her sayının virgül pozisyonu sabittir, bu da sistemin esnekliğini kısıtlar.

6. IEEE 754 Standardı ile Uyum: IEEE 754 standardı, kayan nokta sistemlerinin hesaplama doğruluğunu ve uyumluluğunu sağlar. Bu standart, birçok dijital sistemde yaygın olarak kabul görmüş ve test edilmiş bir çözümdür. Bu standardın sağladığı uyumluluk ve doğruluk, kayan nokta işlemlerini daha güvenilir kılar.

Kayan nokta sistemi, özellikle geniş bir sayı aralığı ve hassasiyet gerektiren projelerde vazgeçilmez bir çözüm sunar. FPGA'larda paralel işlem yeteneği ile birleştirildiğinde, kayan nokta işlemleri çok daha verimli hale gelir. Bu proje için kayan nokta sistemini seçmemizin başlıca sebepleri, daha hassas sonuçlar vermesi, geniş sayı aralığı sunması ve bilimsel hesaplamalar için ideal bir yapı sağlamasıdır. Sabit noktalı işlemler, basitlik ve hız açısından avantaj sağlasa da, projenin ihtiyaç duyduğu yüksek hassasiyetli ve dinamik yapı için uygun değildir.

3.8. Kayan Nokta (Floating Point) Kullanım Alanları ve Avantajları

Kayan nokta işlemleri, çok geniş bir sayı aralığında yüksek doğrulukta matematiksel hesaplamalar yapmayı sağlar. Bu nedenle, hem donanım hem de yazılım tabanlı birçok uygulamada geniş bir kullanım alanına sahiptir. Bilimsel ve mühendislik hesaplamalarından görüntü işleme ve yapay zeka uygulamalarına kadar çeşitli alanlarda kayan nokta işlemleri yaygın olarak kullanılır.

Kayan Nokta Kullanım Alanları

1. Bilimsel Hesaplamalar

Bilimsel araştırmalar ve mühendislik projelerinde çok büyük veya çok küçük sayılarla işlem yapılması gerekir. Kayan nokta işlemleri, bu tür hesaplamaları hassas ve güvenilir bir şekilde gerçekleştirmeyi sağlar.

Örnekler:

- Fiziksel simülasyonlar (örneğin, hava durumu tahminleri, astrofiziksel hesaplamalar).

- Kimyasal hesaplamalar (moleküler dinamik ve reaksiyon hesaplamaları).

- Matematiksel modellemeler (integral ve diferansiyel hesaplamalar).

2. Görüntü ve Ses İşleme

Kayan nokta işlemleri, yüksek çözünürlük ve doğruluk gerektiren görüntü ve ses işleme algoritmalarında kritik rol oynar. Özellikle çok büyük veya çok küçük sinyal değerlerinin işlendiği durumlarda kayan nokta hesaplamaları kullanılır.

Örnekler:

- Dijital görüntü işleme: Kayan nokta hesaplamaları, yüksek çözünürlüklü görüntülerde piksel düzeyinde renk işlemleri ve filtreleme işlemleri yapmak için kullanılır.

- Ses işleme: Ses sinyallerinin filtrelenmesi ve dönüştürülmesinde yüksek doğruluk sağlamak için kayan nokta işlemleri tercih edilir.

- Video sıkıştırma ve kodlama: Kayan nokta aritmetiği, video codec'lerinin veri sıkıştırma algoritmalarında kullanılır.

3. Yapay Zeka ve Makine Öğrenmesi

Yapay zeka ve makine öğrenmesi algoritmaları, geniş veri setleri üzerinde çalışarak matris çarpımları gibi büyük ölçekli matematiksel işlemler yapar. Kayan nokta hesaplamaları, bu tür işlemler için yüksek doğruluk ve geniş sayı aralığı sunar.

Örnekler:

- Derin öğrenme modelleri, sinir ağı hesaplamalarında geniş matris çarpımlarını yüksek doğrulukla gerçekleştirmek için kayan nokta aritmetiği kullanır.

- Bilgisayarla görme uygulamalarında (nesne tanıma, görüntü sınıflandırma) kayan nokta hesaplamaları, görsel verilerin doğru analizini sağlar.

4. Simülasyonlar ve Oyun Geliştirme

Kayan nokta aritmetiği, oyun motorları ve fiziksel simülasyonlarda, doğru hesaplamalar yaparak sanal dünyaların gerçekçi davranışlar sergilemesini sağlar.

Örnekler:

- 3D grafik motorları, nesnelerin ve kameraların konumlarını, ışık yansımalarını ve gölge hesaplamalarını doğru bir şekilde gerçekleştirmek için kayan nokta aritmetiği kullanır.

- Fiziksel simülasyonlar, nesneler arası çarpışmaların, hareketlerin ve güçlerin doğru bir şekilde modellenmesi için kayan nokta işlemlerini kullanır.

5. Kriptografi

Kayan nokta işlemleri, özellikle güvenli veri iletimi ve şifreleme algoritmalarında hassas hesaplamalar yaparak güvenliği sağlar. Kriptografi algoritmalarında matematiksel işlemlerin doğru ve hızlı bir şekilde gerçekleştirilmesi çok önemlidir.

Örnekler:

- RSA ve AES gibi şifreleme algoritmaları, büyük asal sayıların faktorizasyonu gibi karmaşık hesaplamalar yapar ve kayan nokta işlemleri bu tür algoritmalarda kullanılır.

6. Finansal Uygulamalar

Finansal modellemeler ve algoritmalar, çok küçük ve çok büyük parasal değerlerle çalışmayı gerektirebilir. Kayan nokta işlemleri, bu tür hesaplamalarda yüksek doğruluk ve hassasiyet sağlar.

Örnekler:

- Karmaşık faiz hesaplamaları, tahmin modelleri ve finansal risk analizlerinde kayan nokta aritmetiği kullanılır.

7. İstatistik ve Veri Analitiği

Büyük veri setlerinde yapılan analizlerde hassasiyet çok önemlidir. Kayan nokta işlemleri, büyük veri analizlerinde yüksek doğrulukta sonuçlar elde etmeyi sağlar.

Örnekler:

- Büyük veri analitiği platformlarında ve makine öğrenimi modellerinin eğitiminde kayan nokta işlemleri kullanılır.

- İstatistiksel hesaplamalarda ve regresyon analizlerinde kayan nokta aritmetiği tercih edilir.

Kayan Nokta İşlemlerinin Avantajları

1. Geniş Sayı Aralığı

- Kayan nokta sistemi, çok büyük ve çok küçük sayıları aynı anda temsil edebilir. Bu, geniş bir aralıktaki sayılarla hesaplama yapmayı sağlar. Örneğin, bilimsel hesaplamalarda 10^38 veya daha büyük sayılar kullanılabilir. Sabit noktalı sistemler bu kadar geniş bir sayı aralığı sunamaz.

2. Yüksek Hassasiyet

- Kayan nokta işlemleri, küçük sayılarla çalışırken yüksek hassasiyet sağlar. Küçük değişikliklerin bile doğru bir şekilde hesaplanması gereken uygulamalarda kayan nokta aritmetiği büyük avantaj sağlar. Özellikle, matematiksel modellemeler ve simülasyonlar gibi uygulamalarda bu hassasiyet kritik öneme sahiptir.

3. Dinamik Virgül Konumu

- Kayan nokta sisteminin en büyük avantajlarından biri, virgülün dinamik olarak kaydırılmasıdır. Bu, sayının büyüklüğüne göre virgülün konumunu değiştirebilmesi anlamına gelir. Böylece, küçük ve büyük sayılar aynı formatta, aynı doğrulukta temsil edilebilir.

4. Karmaşık Hesaplamalarda Güvenilirlik

- Bilimsel, mühendislik ve finansal hesaplamalar gibi karmaşık uygulamalarda kayan nokta işlemleri doğruluk ve güvenilirlik sağlar. Sabit nokta işlemlerinin bu tür uygulamalarda sunduğu doğruluk yetersiz kalabilir.

Özellikle matris işlemleri, diferansiyel denklemler ve sayısal integrasyon gibi matematiksel işlemler kayan nokta aritmetiği ile güvenilir bir şekilde yapılır.

5. Paralel İşlem Yeteneği

- FPGA gibi donanım tabanlı sistemlerde kayan nokta işlemleri paralel olarak yapılabilir. Bu, çok sayıda hesaplamanın aynı anda gerçekleştirilmesini sağlar. Paralel işlem kapasitesi, görüntü işleme, yapay zeka ve veri analitiği gibi uygulamalarda büyük performans avantajı sunar.

6. Standartlaşmış IEEE 754 Formatı

- Kayan nokta işlemleri, IEEE 754 standardına göre yapılandırıldığında dünya genelinde bir uyumluluk ve doğruluk standardı sağlar. Bu standart sayesinde, kayan nokta işlemleri farklı donanımlarda aynı doğrulukta çalışır ve bu da taşınabilirlik ve güvenilirlik açısından büyük bir avantajdır.

Kayan nokta işlemleri, geniş bir sayı aralığında hassasiyet gerektiren birçok uygulama için ideal bir çözümdür. Bilimsel ve mühendislik hesaplamalarından yapay zeka ve grafik işleme uygulamalarına kadar geniş bir kullanım alanı vardır. Kayan nokta işlemlerinin sunduğu yüksek doğruluk, geniş sayı aralığı ve dinamik yapı, projelerde güvenilir ve hızlı sonuçlar elde edilmesini sağlar. Özellikle, FPGA gibi paralel işlem yapabilen platformlarda kayan nokta işlemleri donanım seviyesinde hızlandırılarak büyük performans artışı sağlanabilir.

3.9. Kayan Nokta (Floating Point) Sayı Sistemleri

Kayan nokta (floating point) sayı sistemleri, geniş bir aralıkta sayıları temsil edebilmek için kullanılan bir yöntemdir. Bilgisayarlar ve dijital sistemler, genellikle sabit nokta (fixed-point) ve kayan nokta sayı sistemlerini kullanır. Sabit nokta sayı sistemlerinde, virgülün yeri sabittir, bu da sayıların temsil edilebileceği aralığı kısıtlar. Ancak kayan nokta sistemi, virgülün yerini dinamik olarak kaydırarak çok büyük ya da çok küçük sayıları daha hassas bir şekilde temsil edebilir.

Kayan nokta sistemleri, IEEE 754 standardı ile tanımlanmıştır ve bilimsel hesaplamalar, grafik işlemleri, görüntü işleme, yapay zeka, fiziksel modelleme gibi birçok alanda yaygın olarak kullanılır. FPGA'lar gibi donanım tabanlı sistemlerde ise Floating Point Unit (FPU), bu tür işlemleri gerçekleştirmek için kullanılan özel donanım modülleridir.

Kayan nokta sayı sistemi, üç temel bileşen kullanarak bir sayıyı temsil eder:

1. İşaret (Sign Bit): Sayının pozitif ya da negatif olduğunu belirleyen bit. Eğer işaret biti 0 ise sayı pozitif, 1 ise sayı negatiftir.

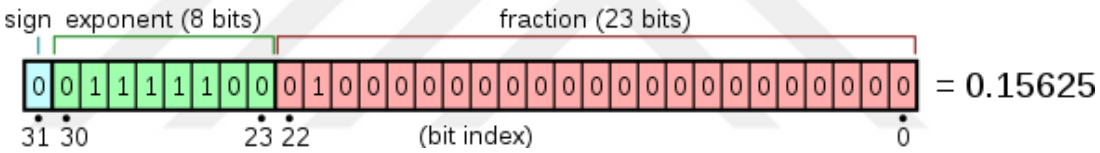
2. Üslü Değer (Exponent): Sayının büyüklüğünü belirten kısım. Üslü değer, sayıyı 2'nin üssü olarak gösterir ve kayan noktada sayının virgülünün hangi pozisyonda olduğunu belirler.

3. Mantisa (Mantissa) ya da Kesir (Fraction): Sayının tam kısmıdır ve sayının hassasiyetini belirler. Bu bölüm, normalleştirilmiş formda yazılır (1 ile başlar).

3.9.1. IEEE 754 Standardı

IEEE 754, kayan nokta sayıların bilgisayar ortamında nasıl temsil edileceğini standartlaştıran bir sistemdir.

IEEE 754 standardına göre, bir kayan nokta sayısı şu şekilde temsil edilir:



Şekil 3.2: IEEE 754 standardına göre bir kayan nokta sayısı

- **S (Sign Bit):** İşaret biti, sayının pozitif ya da negatif olmasını belirler.
- **M (Mantissa):** Kesir kısmı, sayının büyüklüğünü tanımlar.
- **E (Exponent):** Üslü değer, sayının büyüklüğünü belirten bileşendir ve 2'nin üssü olarak gösterilir.

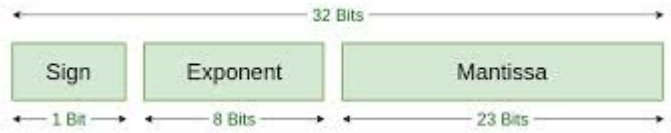
Bu standart, farklı boyutlardaki kayan nokta sayılarının nasıl saklanacağını ve nasıl işlemler yapılacağını düzenler.

İki temel türde IEEE 754 standardı bulunur:

1. Tek Hassasiyetli (Single Precision): 32 bit kullanarak bir kayan nokta sayıyı temsil eder.

2. Çift Hassasiyetli (Double Precision): 64 bit kullanarak daha büyük hassasiyetli sayılar için kullanılır.

3.9.1.1. Tek Hassasiyetli Kayan Nokta Formatı (32-bit):



Şekil 3.3: IEEE 754 Tek Hassasiyetli Kayan Nokta Formatı (32-bit)

3.9.1.2. Çift Hassasiyetli Kayan Nokta Formatı (64-bit):

Şekil 3.4: IEEE 754 Çift Hassasiyetli Kayan Nokta Formatı (64-bit)

IEEE 754 standardına göre bir kayan nokta sayı şu adımlarla temsil edilir:

1. Sayı önce ikili (binary) formata çevrilir.

2. İkili sayı normalleştirilir. Normalleştirme sırasında sayının tam kısmı 1 ile başlar.

3. Üslü değer, pozitif veya negatif olarak kaydırılır ve mantisa (kesir) değeri ile birlikte saklanır.

Örneğin, -12.25 sayısını IEEE 754 standardına göre kayan nokta formatında temsil edelim:

- İkilik sistemde: 1100.01.

- Normalleştir: 1.10001 × 2^3.

- Üslü değeri kaydır: 127 + 3 = 130 (binary 10000010).

- Mantisa: 10001, 23 bit için sağa sıfırlar eklenir.

Sonuç olarak, -12.25 sayısının IEEE 754 tek hassasiyetli formatındaki gösterimi:

1 | 10000010 | 10001000000000000000000

3.10. Kayan Nokta Aritmetiği

Kayan nokta işlemleri, matematiksel işlemler (toplama, çıkarma, çarpma, bölme) ile yapılır. FPGA'lerde, FPU (Floating Point Unit) modülleri, kayan nokta işlemleri donanım seviyesinde hızlandırmak için kullanılır. Aşağıda kayan nokta işlemlerinin nasıl yapıldığına dair bazı örnekler verilmiştir:

Kayan Nokta ile Toplama ve Çıkarma

Üslü Değerlerin Hizalanması: İki sayı aynı üslü değere sahip olmalıdır. Bu nedenle daha küçük üslü değere sahip sayının mantisası, büyük olanı yakalayacak şekilde kaydırılır.

Mantisaların İşlenmesi: Üslü değerler hizalandıktan sonra mantisalar toplanır veya çıkarılır.

Sonucun Normalleştirilmesi: Toplama veya çıkarma işlemi sonrası sonuç normalleştirilir, yani mantisanın başına 1 yerleştirilir ve üslü değeri buna göre ayarlanır.

Kayan Nokta ile Çarpma

1. Üslü değerler toplanır.
2. Mantisalar çarpılır.
3. Sonuç normalleştirilir ve üslü değeri buna göre ayarlanır.

Kayan Nokta ile Bölme

1. Üslü değerler birbirinden çıkarılır.
2. Mantisalar bölünür.
3. Sonuç normalleştirilir.

3.10.1. Kayan Nokta İşlemlerinin Kullanım Alanları

Kayan nokta işlemleri, geniş bir kullanım alanına sahiptir. Özellikle bilimsel hesaplamalar ve mühendislik uygulamalarında, çok hassas ve geniş aralıklı sayıları temsil edebilme yeteneği, kayan nokta sistemlerinin önemini artırır.

1. Bilimsel Hesaplamalar: Fizik, kimya, astronomi gibi alanlarda kullanılan çok büyük veya çok küçük sayılar, kayan nokta sistemi ile rahatlıkla işlenebilir.

2. Görüntü ve Ses İşleme: Grafik işlemcileri (GPU) ve dijital sinyal işleme (DSP) devrelerinde, kayan nokta aritmetiği yüksek çözünürlük ve doğruluk gerektiren hesaplamalar için kullanılır.

3. Yapay Zeka ve Makine Öğrenmesi: Sinir ağlarının eğitimi sırasında büyük matris işlemleri yapılırken kayan nokta hesaplamaları kullanılır. Bu sayede büyük veri setleri daha hassas işlenebilir.

4. Oyunlar ve Simülasyonlar: 3D grafik işleme, fizik simülasyonları ve oyun motorları, kayan nokta aritmetiğini kullanarak yüksek doğrulukta hesaplamalar yapar.

3.11. FPGA'de Floating Point Unit (FPU)

FPGA'lerde FPU modülleri, kayan nokta işlemlerini donanım seviyesinde hızlandırmak için kullanılan IP Core'lardır. Bu modüller, FPGA içerisinde matematiksel işlemleri daha hızlı yapar ve tasarımcıya hazır bir çözüm sunar. FPU modülleri, bilimsel ve mühendislik hesaplamaları gerektiren projelerde FPGA üzerinde sıkça kullanılır. FPGA platformlarında, bu tür işlemler donanım seviyesinde gerçekleştirildiği için yazılım tabanlı çözümlere göre çok daha hızlı sonuçlar elde edilebilir.

Tablo 3.4. FPGA'lerde FPU modülleri

FPU Adı	İşlevi	Kullanım Alanı
Xilinx Floating Point Core	Kayan nokta aritmetik işlemler	Bilimsel hesaplamalar, veri işleme
Altera FPU IP Core	IEEE 754 standardına uygun kayan nokta işlemleri	FPGA tabanlı sinyal ve veri işleme
Custom FPU IP Core	Özelleştirilebilir kayan nokta işlemleri	Özel matematiksel hesaplama gerektiren projeler

Kayan nokta sistemleri, büyük ve hassas sayıları temsil edebilme yeteneği ile özellikle mühendislik ve bilimsel hesaplamalarda önemli bir rol oynar.

FPGA’lerde bu tür işlemleri donanım seviyesinde hızlandırmak için FPU modülleri kullanılır. Kayan nokta aritmetiği, FPGA gibi paralel işlem yapabilen sistemlerde büyük avantaj sağlar, çünkü bu işlemler donanım seviyesinde hızlandırılarak performans optimizasyonu sağlanır. Kayan nokta sistemi, geniş bir kullanım alanına sahip olup, dijital tasarım ve FPGA projelerinde kritik bir öneme sahiptir.

3.12. IEEE 754 Kayan Nokta Sayı Formatında Yuvarlama İşlemleri

Kayan noktalı sayı sistemleri, sınırlı bit sayısı ile geniş bir sayı aralığını temsil etme olanağı sundukları için bilgisayar sistemlerinde yaygın olarak kullanılmaktadır. IEEE 754 standardı, özellikle kayan noktalı sayılar için global bir temsil standardı sunmaktadır. Ancak, sınırlı bit sayısı nedeniyle bazı sayılar tam olarak temsil edilemez; bu durumda yuvarlama işlemleri devreye girer.

IEEE 754 standardı, kayan noktalı sayıların temsilinde çeşitli yuvarlama modları sunar. Bu modlar, sayıların olabilecek en doğru şekilde ifade edilmesini sağlamak amacıyla geliştirilmiştir.

3.12.1. En Yakına Yuvarlama (Round to Nearest, Even)

En yakına yuvarlama, IEEE 754 standardında varsayılan yuvarlama modu olarak kullanılır. Bu yöntemde, sayı en yakın tam sayıya veya belirli ondalık basamağa yuvarlanır.

İki tam sayı arasında kalınması durumunda sayı, en yakın çift tam sayıya yuvarlanır. İstatistiksel olarak bu yöntem, uzun vadede toplam hatanın minimize edilmesini sağlar ve "Banker's Rounding" olarak da bilinir.

Örneğin:

- 1.5 sayısı, en yakın çift tam sayı olan 2’ye yuvarlanır.
- 2.5 sayısı da en yakın çift tam sayı olan 2’ye yuvarlanır.

Bu yöntem finansal ve bilimsel hesaplamalarda sıkça tercih edilir.

3.12.2. Sıfıra Doğru Yuvarlama (Round Towards Zero)

Sıfıra doğru yuvarlama, sayıların sıfıra en yakın tam sayıya doğru yuvarlanmasıdır. Bu yöntemde pozitif sayılar aşağıya, negatif sayılar ise yukarıya doğru yuvarlanır. Diğer bir deyişle, sayıdaki ondalık kısmı atılır. Bu yöntem, negatif veya pozitif büyük sayılarla çalışırken yuvarlama hatalarını önlemek için kullanılır.

Örneğin:

- 1.7 sayısı, 1'e yuvarlanır.
- -1.7 sayısı, -1'e yuvarlanır.

3.12.3. Sonsuza Doğru Yuvarlama (Round Towards +∞)

Bu yuvarlama modunda sayı pozitif yöne, yani daha büyük bir tam sayıya yuvarlanır. Pozitif sayılar yukarıya, negatif sayılar ise daha az negatif bir değere yuvarlanır. Bu yöntem, bazı finansal uygulamalarda üst sınıra yuvarlamanın gerektiği durumlarda kullanılır.

Örneğin:

- 1.2 sayısı, 2'ye yuvarlanır.
- -1.2 sayısı, -1'e yuvarlanır.

3.12.4. Eksi Sonsuza Doğru Yuvarlama (Round Towards -∞)

Eksi sonsuza doğru yuvarlama, sayıların negatif yönde en yakın tam sayıya doğru yuvarlanmasıdır. Pozitif sayılar bir alt tam sayıya, negatif sayılar ise daha negatif bir tam sayıya yuvarlanır.

Örneğin:

- 1.8 sayısı, 1'e yuvarlanır.
- -1.8 sayısı, -2'ye yuvarlanır.

IEEE 754 formatında 23 bitlik mantissa bulunduğundan, bazı sayılar bu sınırı aşabilir. Bu durumda, sayı en yakın 23 bitlik temsil edilebilen değere yuvarlanır. Örneğin, 24 basamaklı bir sayı ele alınsın:

Sayının tam ikilik gösterimi: 1.10110110011001100110

23 bitlik mantissa için yuvarlanmış hali: 1.10110110011001101101

Bu örnekte, son basamak yuvarlanarak en yakın 23 bitlik değere dönüştürülmüştür.

IEEE 754 standardında yuvarlama işlemleri, bilgisayar sistemlerinde kayıp veya hata payını minimize etmek için hayati önem taşır. Yuvarlama modları, farklı hesaplama senaryolarına uyum sağlayarak sayısal doğruluğu korur ve geniş bir yelpazede uygulamalarda kullanılır.

3.13. Finite-State Machine (FSM)

Finite-State Machine (FSM) veya sonlu durum makineleri, bir sistemin zaman içindeki farklı durumlarını ve bu durumlar arasında geçişleri tanımlayan soyut bir modeldir. FSM'ler dijital sistemlerde geniş bir uygulama alanı bulur ve özellikle dijital devre tasarımı, yazılım geliştirme, robotik ve gömülü sistemler gibi alanlarda kullanılır. FSM'ler, belirli bir girişe bağlı olarak belirli durumlar arasında geçiş yaparak çıktıları belirleyen basit ama güçlü bir yapı sağlar. Bir FSM, yalnızca sonlu sayıda duruma sahiptir ve her bir duruma girişler aracılığıyla ulaşılır.

FSM'ler, özellikle FPGA (Field-Programmable Gate Array) tasarımlarında önemli bir rol oynar. FPGA'lar, dijital devrelerin işleyişini belirleyen esnek ve programlanabilir mantık yapıları sunduğundan, FSM'lerin FPGA üzerinde uygulanması sıkça görülür. FPGA'lar üzerinde FSM tasarımları genellikle sistemin zamanlamasını ve kontrol işlemlerini düzenler. FSM yapısı, FPGA'nın programlanabilir mantık blokları ve durum geçişlerine dayalı yapısı ile mükemmel bir şekilde uyum sağlar. Tasarımcılar, FPGA üzerinde farklı FSM yapılarını kullanarak, dijital sistemlerin denetimi, veri akışı kontrolü ve diğer mantıksal işlemleri etkin bir şekilde yönetebilirler.

FSM yapısının temel bileşenleri şunlardır:

1. Durumlar (States): Sistemin içinde bulunabileceği sınırlı sayıda durumlar. Her bir durum, sistemin belirli bir koşulu veya durumunu temsil eder.

2. Geçişler (Transitions): Bir durumdan diğerine geçişi sağlayan kurallar. Bu geçişler, genellikle bir giriş sinyaline bağlıdır.

3. Girdiler (Inputs): FSM'nin durumlar arasında geçiş yapmasını sağlayan dış sinyaller veya komutlar.

4. Çıktılar (Outputs): FSM'nin o anki durumuna ve girdilere bağlı olarak ürettiği sinyaller veya veriler.

FSM'ler iki temel yapıya ayrılabilir: Moore makineleri ve Mealy makineleri. Moore makinelerinde çıktı, yalnızca mevcut duruma bağlıdır. Mealy makinelerinde ise çıktı, hem mevcut duruma hem de girdilere bağlıdır. Bu fark, FSM'nin ne zaman ve nasıl çıktı üreteceğini etkiler.

FSM'lerin FPGA üzerindeki kullanımı, özellikle kayan nokta işlemleri içeren sistemlerde büyük avantajlar sunar. Kayan nokta işlemleri genellikle oldukça karmaşık olduğundan, FPGA üzerinde FSM kullanarak bu işlemleri adım adım yöneten bir kontrol yapısı oluşturmak mümkündür. FPGA'lar, paralel işlem yapma yetenekleriyle karmaşık kayan nokta işlemlerini daha hızlı ve verimli bir şekilde gerçekleştirebilir. FSM, bu işlemleri kontrol ederek, hangi aşamalarda hangi işlemlerin yapılacağını belirler ve bu işlemlerin doğru sıralamada ve zamanda gerçekleşmesini sağlar.

Örneğin, bir FPGA üzerinde bir kayan nokta toplama işlemi gerçekleştirdiğimizi düşünelim. Bu işlem, birçok adımı içerir: iki sayının işaretlerinin kontrol edilmesi, üslü değerlerin hizalanması, mantisaların toplanması ve sonuçların normalleştirilmesi gibi işlemler. Her bir adım, belirli bir sırayla yapılmalıdır ve FSM, bu adımları kontrol eden bir yapı olarak kullanılabilir.

FSM, her adımın sonunda bir sonraki adımı tetikleyebilir ve işlem tamamlanana kadar süreci yönetir. Bu, işlemlerin doğru bir şekilde ve belirli bir zamanlamaya uygun olarak yapılmasını sağlar.

FPGA üzerinde FSM tasarımları, genellikle Verilog veya VHDL gibi donanım tanımlama dilleri ile yapılır. Verilog, FSM yapısını tanımlamak için oldukça uygun bir dil sunar. Tasarımcılar, FSM'deki her bir durumu ve bu durumlar arasındaki geçişleri Verilog kodu ile tanımlayabilirler. Verilog'da FSM tasarımı genellikle bir always bloğu içinde gerçekleştirilir ve durum geçişleri, giriş sinyalleri ve durum değişkenlerine göre yapılır. Bu sayede FSM'nin hangi durumdan hangi duruma geçeceği, belirli bir saat sinyali ve girdiler tarafından kontrol edilir.

FSM'ler, FPGA'larda kayan nokta aritmetiği gibi karmaşık matematiksel işlemlerin kontrolünde kullanıldığında, işlem sürecini düzenlemek için harika bir araçtır. Kayan nokta işlemleri genellikle birden fazla adımdan oluştuğundan ve her adım belirli bir sırayla yapılması gerektiğinden, FSM, bu işlemleri adım adım yöneten bir yapı olarak kullanılır. Ayrıca, FPGA üzerinde paralel işlem yapılabilme yeteneği, FSM ile kontrol edilen bu işlemlerin aynı anda birçok farklı birimde gerçekleştirilmesini mümkün kılar. Bu da, işlemlerin daha hızlı yapılmasını sağlar ve sistemin genel performansını artırır.

Kayan nokta işlemlerinin FSM ile kontrol edilmesi ayrıca zamanlama ve kaynak yönetimini optimize etmeye yardımcı olur. FPGA'daki kaynaklar sınırlı olduğundan, FSM yapısı her bir işlemin doğru zamanda yapılmasını ve donanım kaynaklarının verimli bir şekilde kullanılmasını sağlar. Örneğin, bir FPGA üzerinde bir kayan nokta çarpma işlemi yapıldığında, FSM kullanarak çarpma işleminin farklı aşamaları arasında geçiş yapılabilir ve her aşamanın ne zaman gerçekleştirileceği kontrol edilebilir. Bu tür bir kontrol yapısı, FPGA'nın mantık hücrelerini en verimli şekilde kullanarak aynı anda birden fazla işlemi yönetmesini sağlar.

FSM'ler ayrıca, bir işlem sırasında meydana gelebilecek hataları da yönetebilir. Örneğin, bir kayan nokta işlemi sırasında beklenmedik bir hata oluştuğunda, FSM bu durumu algılayabilir ve uygun bir hata yönetimi süreci başlatabilir. Bu da sistemin güvenilirliğini artırır ve hata oluşma durumunda sistemin doğru şekilde tepki vermesini sağlar.

Sonuç olarak, FSM yapısı, FPGA tasarımlarında kayan nokta işlemlerinin adım adım yönetilmesi için güçlü bir araçtır. FSM, karmaşık işlemlerin kontrolünü sağlayarak, zamanlama ve kaynak yönetimi açısından büyük avantajlar sunar. FPGA'nın paralel işlem yapabilme yetenekleriyle birleştiğinde, FSM yapısı, kayan nokta işlemleri gibi karmaşık hesaplamaları hızlı ve verimli bir şekilde gerçekleştirebilir. Verilog veya VHDL gibi donanım tanımlama dilleri ile FSM tasarımı yapmak, bu yapıların FPGA üzerinde doğru bir şekilde uygulanmasını ve dijital sistemlerin performansının optimize edilmesini sağlar.

Finite-State Machine (FSM), belirli bir dizi durumu olan ve girişlere göre bir durumdan diğerine geçebilen bir modeldir. Bu yapı, dijital devrelerde ve algoritmalarda yaygın olarak kullanılır.

Bir FSM kullanarak bir integer sayıyı floating-point sayıya dönüştüren, ALU (Arithmetic Logic Unit) işlemleri gerçekleştiren ve ardından sayıyı tekrar integer formata dönüştüren bir yapı üç ana duruma sahip olur :

- Integer'dan Floating-Point'e Dönüşüm (IntToFloat)
- ALU İşlemleri (ALUOps)
- Floating-Point'ten Integer'a Dönüşüm (FloatToInt)

FSM, başlangıçta bir integer sayıyı alacak, bu sayıyı floating-point formatına dönüştürecek, belirli ALU işlemlerini gerçekleştirecek ve son olarak sonucu tekrar integer formata dönüştürecektir.

IntToFloat Durumu: Bu durumda, giriş olarak alınan integer sayı, floating-point formatına dönüştürülür. Bu dönüşüm, IEEE 754 standardına göre yapılabilir.

ALUOps Durumu: Bu durumda, floating-point sayı üzerinde belirli ALU işlemleri gerçekleştirilir. Örneğin, toplama, çıkarma, çarpma veya bölme işlemleri yapılabilir.

FloatToInt Durumu: Bu durumda, ALU işlemleri sonucu elde edilen floating-point sayı tekrar integer formata dönüştürülür ve çıkış olarak verilir.

Bu FSM yapısı, integer sayıyı floating-point sayıya dönüştürmek, üzerinde ALU işlemleri gerçekleştirmek ve ardından tekrar integer formata dönüştürmek için basit ve etkili bir çözüm sunar.

Verilog kodu, belirtilen işlemleri gerçekleştirmek için gerekli olan temel yapıyı sağlar ve gerçek uygulamalarda daha karmaşık dönüşümler ve işlemler eklenerek genişletilebilir.

3.14. Aritmetik Mantık Birimi (ALU)

Aritmetik Mantık Birimi (ALU), bir dijital sistemdeki en temel bileşenlerden biridir ve bilgisayarların matematiksel ve mantıksal işlemleri gerçekleştirmesini sağlar. ALU, iki ana işlevi yerine getirir: aritmetik işlemler (toplama, çıkarma, çarpma, bölme gibi) ve mantıksal işlemler (AND, OR, NOT, XOR gibi). Bir ALU, işlemcinin merkezinde yer alır ve işlemcinin aldığı komutlara göre sayısal işlemler gerçekleştirir.

ALU'nun performansı, bir sistemin genel işlem gücünü ve hesaplama kapasitesini doğrudan etkiler.

ALU, genellikle sabit noktalı (fixed-point) işlemleri çok hızlı bir şekilde gerçekleştirebilir çünkü bu işlemler doğrudan tam sayı aritmetiğine dayanır. Ancak, kayan noktalı (floating-point) sayılarla çalışmak daha karmaşıktır. Kayan nokta aritmetiğinde sayılar işaret, mantisa (kesir kısmı) ve üslü değer bileşenleriyle temsil edilir. Bu da, sabit noktalı işlemlerden farklı olarak ek adımlar gerektirir, çünkü önce üslü değerlerin hizalanması, mantisaların toplanması veya çarpılması ve sonuçların normalleştirilmesi gerekir. Bu nedenle, kayan nokta işlemlerini gerçekleştiren bir ALU, daha karmaşık donanım gerektirir.

FPGA (Field-Programmable Gate Array) cihazları, ALU'nun donanım seviyesinde uygulanması için çok uygun platformlardır. FPGA'lar, esnek ve programlanabilir yapıları sayesinde kullanıcıların kendi ALU yapılarını tasarlamalarına ve belirli uygulamalara uygun hale getirmelerine olanak tanır. FPGA üzerinde bir ALU tasarımı, belirli bir işlemciye entegre edilebilir veya FPGA'nın mantık blokları kullanılarak bağımsız bir işlem birimi olarak oluşturulabilir. FPGA'ların en büyük avantajlarından biri, donanım seviyesinde paralel işlem yapabilme yetenekleridir. Bu sayede, bir FPGA üzerinde birden fazla ALU modülü çalıştırılarak aynı anda birden fazla işlem yapılabilir ve bu da sistemin performansını büyük ölçüde artırır.

FPGA'lar üzerinde kayan nokta aritmetiği işlemleri gerçekleştiren ALU'lar, yüksek doğruluk ve geniş dinamik aralık sunma yeteneğine sahiptir.

Özellikle bilimsel hesaplamalar, mühendislik simülasyonları ve veri yoğun uygulamalarda, kayan nokta işlemleri doğru ve hızlı bir şekilde gerçekleştirilmelidir. FPGA'lar, bu tür işlemleri donanım seviyesinde hızlandırarak yüksek performans sağlar. FPGA'da tasarlanmış bir kayan nokta ALU, genellikle IEEE 754 standardına dayalıdır. IEEE 754 standardı, kayan nokta sayıların nasıl temsil edileceğini ve işlemlerin nasıl yapılacağını belirler. FPGA üzerinde kayan nokta ALU tasarımı yaparken, bu standarda uygun bir yapı oluşturulması, yüksek doğruluk ve taşınabilirlik sağlar.

FPGA üzerinde bir kayan nokta ALU tasarlarken, toplama, çıkarma, çarpma ve bölme gibi işlemler için ayrı modüller kullanılabilir.

Örneğin, bir kayan nokta toplama işlemi, önce iki sayının üslü değerlerinin hizalanmasını, ardından mantisaların toplanmasını ve sonuçların normalleştirilmesini gerektirir. Bu işlemler, FPGA üzerindeki mantık hücreleri aracılığıyla adım adım gerçekleştirilir ve her adım bir ALU modülü tarafından yönetilir. Aynı şekilde, bir çarpma işlemi, mantisaların çarpılmasını, üslü değerlerin toplanmasını ve sonuçların normalize edilmesini içerir. FPGA'nın paralel işlem yapabilme yetenekleri sayesinde, bu işlemler aynı anda birden fazla işlemci biriminde gerçekleştirilebilir ve sistemin performansı artırılır.

Kayan nokta aritmetiği işlemlerinde kullanılan ALU'lar, yalnızca aritmetik işlemleri gerçekleştirmekle kalmaz, aynı zamanda sistemin zamanlama ve kontrol süreçlerini de yönetir. FPGA üzerindeki ALU modülleri, FSM (Finite-State Machine) yapıları ile birlikte kullanılarak belirli işlemleri belirli zamanlarda tetikleyebilir. FSM yapısı, her bir aritmetik işlemin adım adım doğru sırayla gerçekleştirilmesini sağlar. Örneğin, bir kayan nokta çarpma işlemi sırasında, FSM kullanarak önce mantisaların çarpılması, ardından üslü değerlerin toplanması ve sonuçların normalleştirilmesi işlemleri adım adım yönetilebilir. Bu sayede, işlemler doğru bir zamanlama ile gerçekleştirilir ve donanım kaynaklarının verimli kullanılması sağlanır.

FPGA üzerinde bir kayan nokta ALU tasarlamanın en büyük avantajlarından biri de özelleştirilebilirlik ve performans optimizasyonudur. FPGA'lar, kullanıcıların belirli uygulamalara uygun özelleştirilmiş ALU'lar tasarlamalarına olanak tanır.

Örneğin, belirli bir uygulama yalnızca toplama ve çıkarma işlemlerini yoğun bir şekilde kullanıyorsa, ALU tasarımı bu işlemlere odaklanarak optimize edilebilir.

Aynı şekilde, daha fazla doğruluk gerektiren bir uygulama için çift hassasiyetli (double precision) kayan nokta işlemleri destekleyen bir ALU tasarlanabilir. FPGA'ların bu özelleştirilebilir yapısı, sistemin performansını maksimum düzeyde artırır ve işlem süresini kısaltır.

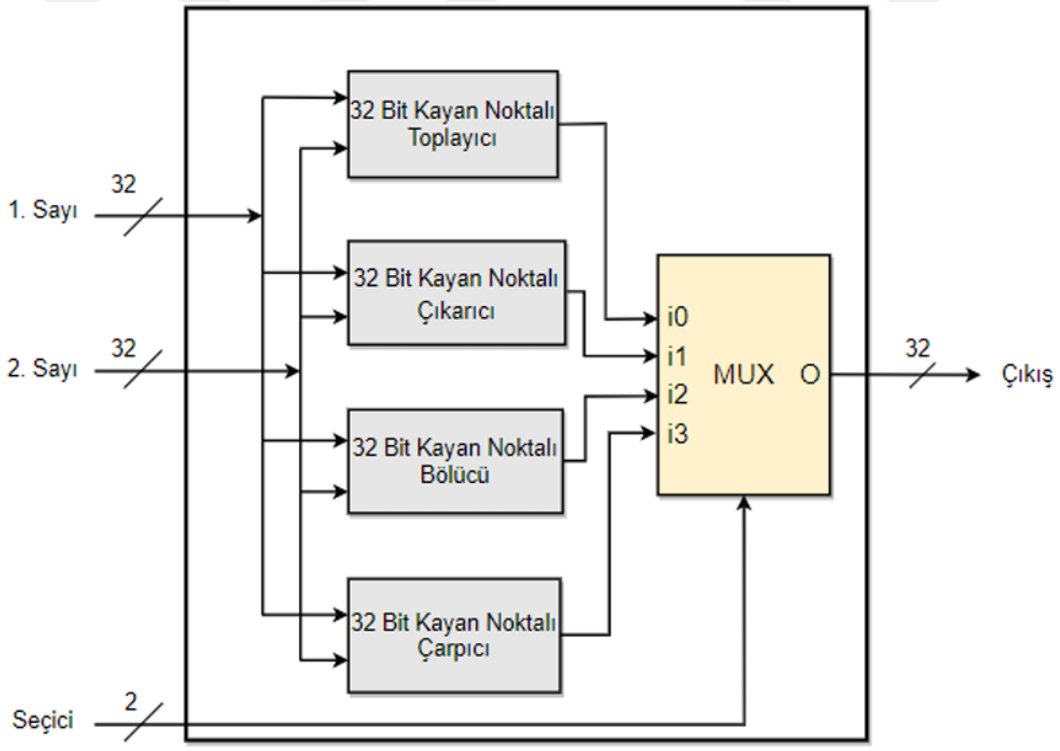
FPGA'lar üzerinde kayan nokta ALU'ların kullanıldığı bir diğer alan da bilimsel hesaplamalar ve mühendislik simülasyonlarıdır. Bu tür uygulamalarda, büyük veri kümeleri üzerinde yapılan hesaplamaların doğruluğu kritiktir ve ALU'lar, bu işlemlerin doğru bir şekilde yapılmasını sağlar. Örneğin, bir fizik simülasyonunda, büyük sayılarla yapılan hassas hesaplamalar gereklidir.

FPGA üzerinde kayan nokta ALU kullanmak, bu hesaplamaların hızlı ve doğru bir şekilde gerçekleştirilmesini sağlar.

Aynı şekilde, mühendislik simülasyonlarında da kayan nokta işlemleri sıklıkla kullanılır ve FPGA'lar, bu tür işlemleri hızlandırarak performansı artırır.

Sonuç olarak, Aritmetik Mantık Birimi (ALU), dijital sistemlerin aritmetik ve mantıksal işlemleri gerçekleştiren temel bileşenidir ve FPGA üzerinde uygulandığında büyük bir esneklik ve performans sağlar. Kayan nokta işlemleri, sabit nokta işlemlerine göre daha karmaşık olmasına rağmen, FPGA'lar üzerinde kayan nokta ALU'ların kullanılması bu işlemleri hızlandırır ve yüksek doğruluk sağlar. FPGA'nın programlanabilir yapısı, ALU tasarımını belirli uygulamalara uygun hale getirmeye olanak tanır ve bu da işlemci kaynaklarının verimli bir şekilde kullanılmasını sağlar. Bilimsel hesaplamalar, mühendislik simülasyonları, görüntü işleme ve yapay zeka gibi birçok alanda FPGA'lar üzerinde kayan nokta ALU'ların kullanılması, sistemin performansını büyük ölçüde artırır ve yüksek doğrulukla hesaplamaların yapılmasına olanak tanır.

3.15. Kayan Nokta (Floating Point) Dört Temel İşlem



Şekil 3.5: 32 Bit kayan noktalı sayı formatında FPU

Kayan nokta (floating point) aritmetiği, geniş bir sayı aralığında yüksek doğrulukla hesaplama yapabilmeyi sağlar. Bilgisayar ve dijital sistemlerde kullanılan bu yöntem, dört temel matematiksel işlem olan toplama, çıkarma, çarpma ve bölme işlemlerinin gerçekleştirilmesini içerir. Bu işlemler sırasında sayılar, üslü ve mantisa bölümlerine ayrılarak işlenir ve ardından normalleştirilir. FPGA gibi donanım platformlarında bu işlemler donanım seviyesinde hızlandırılarak yüksek performans elde edilir.

Aritmetik Mantık Birimi (ALU), bir işlemcinin en kritik bileşenlerinden biridir ve mantıksal ya da matematiksel işlemleri gerçekleştirir. Günümüzün yoğun hesaplama gerektiren uygulamaları için hızlı ve verimli ALU tasarımları büyük önem taşımaktadır. Bu çalışmada, FPGA kısmında IEEE754 32 bit kayan noktalı sayı formatında yüksek performansa sahip bir FPU tasarlanmıştır. Çarpma, bölme, toplama ve çıkarma işlemlerini gerçekleştiren birimlerden oluşur.

İki sayı ortak veri yollarıyla işlenirken, sonuç hangi işlem biriminden çıkacaksa, 4x1'lik bir çoğullayıcı üzerinden seçilir. Hangi işlemin yapılacağını belirlemek için "Seçici" sayısı kullanılır.

Tablo 3.5. İşlem komut tablosu

Seçici	İşlem
"00"	Toplama
"01"	Çıkarma
"10"	Bölme
"11"	Çarpma

Kayan noktalı sayı formatında matematiksel işlemler tasarlamak, yüksek seviyeli programlama dilleri ile kolaydır. Ancak, Verilog gibi bir donanım tanımlama dilinde bu işlemleri gerçekleştirmek oldukça zaman alan, ayrıntılı ve karmaşık bir süreçtir. Bununla birlikte, tasarım sonucunda yüksek doğrulukta işlem yapan ve modüllerinin parçacıklara ayrılıp paralel çalışmasıyla yüksek hız performansı sağlayan bir FPU elde edilmesi büyük bir avantajdır. Bu nedenle, çalışmanın ilerleyen bölümlerinde her bir işlem modülünün detaylı tasarımına yer verilmiştir.

Kayan nokta aritmetiği sırasında şu genel adımlar izlenir:

1. Üslü Değerlerin Hizalanması: İşlem yapılacak iki sayının üslü değerleri hizalanır. Üslü değeri küçük olan sayı, büyük olana hizalanana kadar kaydırılır.

2. Mantisaların İşlenmesi: Üslüleri hizalanan sayılar arasında toplama, çıkarma, çarpma veya bölme işlemi yapılır.

3. Sonuçların Normalleştirilmesi: Elde edilen sonuç normalleştirilir (mantisanın tam kısmı 1 ile başlar) ve üslü değeri buna göre güncellenir.

4. Yuvarlama: Sonuç, belirli bir hassasiyete yuvarlanır.

5. Sonucun Taşınması: Eğer işlem sonucunda taşma (overflow) ya da taşma durumu (underflow) oluşursa gerekli önlemler alınır.

3.15.1. Kayan Nokta ile Toplama

Kayan nokta toplama işlemi, iki sayının üslülerinin hizalanması ve mantisalarının toplanmasıyla gerçekleştirilir. Üslü farkı olan sayılar arasında, küçük olan sayının mantisası kaydırılır ve ardından mantisalar toplanır.

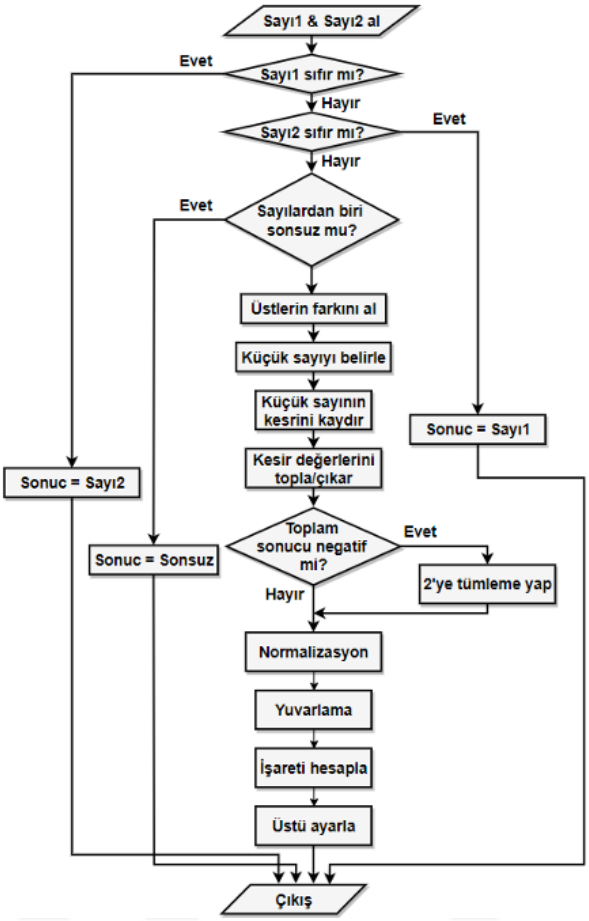
Toplama Akışı:

1. Üslülerin Hizalanması: Üslü farkı olan sayılar arasında, küçük olan üslüye sahip sayının mantisası büyük üslüye eşit olana kadar kaydırılır.

2. Mantisaların Toplanması: Mantisalar toplanır.

3. Normalleştirme: Toplama sonucunda oluşan mantisa normalleştirilir.

4. Yuvarlama: Mantisa yuvarlanarak son haline getirilir.



Şekil 3.6: Toplama ve Çıkarma işlemi akış diyagramı

Sözde Kod:

```
Function FloatingPointAddition(A, B):
// 1. Üslülerin hizalanması
    If A.exponent > B.exponent:
        ShiftRight(B.mantissa, A.exponent - B.exponent)
        B.exponent = A.exponent
    Else:
        ShiftRight(A.mantissa, B.exponent - A.exponent)
        A.exponent = B.exponent
```

```
        EndIf
    // 2. Mantisaların toplanması
        Result.mantissa = A.mantissa + B.mantissa
    // 3. Normalleştirme
        Normalize(Result.mantissa, Result.exponent)
        Return Result
```

3.15.2. Kayan Nokta ile Çıkarma

Çıkarma işlemi, toplama işlemine benzer bir akışa sahiptir. Ancak bu sefer mantisalar birbirinden çıkarılır. Üslü farkı olan sayılarda, küçük üslüye sahip olan sayının mantisası kaydırılır ve ardından çıkarma işlemi gerçekleştirilir.

Çıkarma Akışı:

1. Üslülerin Hizalanması: Sayıların üslü farkı hizalanır ve küçük üslüye sahip olan mantisa kaydırılır.

2. Mantisaların Çıkarılması: Mantisalar çıkarılır.

3. Normalleştirme: Sonuç normalleştirilir.

4. Yuvarlama: Sonuç yuvarlanarak son haline getirilir.

Sözde Kod:

```
    Function FloatingPointSubtraction(A, B):
    // 1. Üslülerin hizalanması
        If A.exponent > B.exponent:
            ShiftRight(B.mantissa, A.exponent - B.exponent)
            B.exponent = A.exponent
        Else:
```

```
            ShiftRight(A.mantissa, B.exponent - A.exponent)
            A.exponent = B.exponent
        EndIf
    // 2. Mantisaların çıkarılması
        Result.mantissa = A.mantissa - B.mantissa
    // 3. Normalleştirme
        Normalize(Result.mantissa, Result.exponent)
        Return Result
```

3.15.3. Kayan Nokta ile Çarpma

Kayan nokta çarpma işlemi sırasında üslü değerler toplanır ve mantisalar çarpılır. Sonuç mantisa normalleştirilir ve üslü değeri güncellenir.

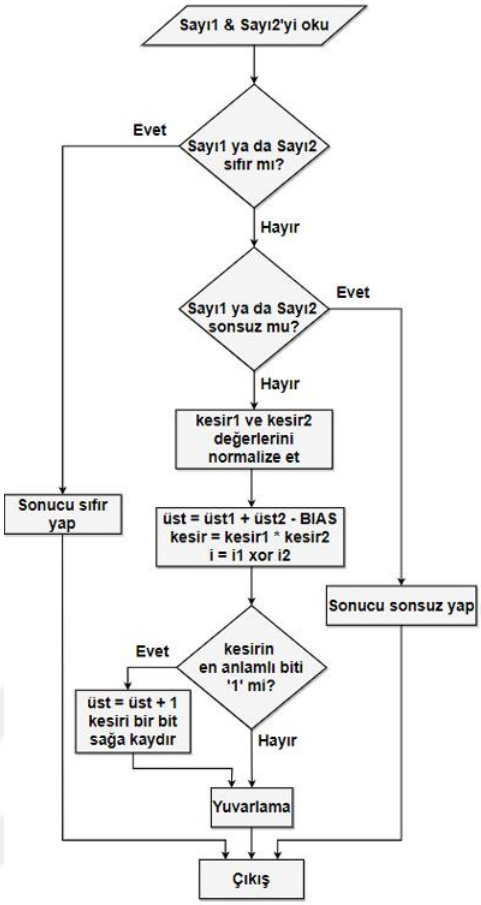
Çarpma Akışı:

1. Üslülerin Toplanması: İki sayının üslü değerleri toplanır.

2. Mantisaların Çarpılması: İki mantisa çarpılır.

3. Normalleştirme: Çarpım sonucu normalleştirilir.

4. Yuvarlama: Sonuç yuvarlanarak son haline getirilir.



Şekil 3.7: Çarpma işlemi akış diyagramı

Sözde Kod:

```
Function FloatingPointMultiplication(A, B):
  // 1. Üslülerin toplanması
      Result.exponent = A.exponent + B.exponent
  // 2. Mantisaların çarpılması
      Result.mantissa = A.mantissa * B.mantissa
  // 3. Normalleştirme
      Normalize(Result.mantissa, Result.exponent)
      Return Result
```

3.15.4. Kayan Nokta ile Bölme

Kayan nokta bölme işlemi, çarpma işlemine benzer bir yapıya sahiptir. Ancak bu sefer mantisalar bölünür ve üslü değerler birbirinden çıkarılır. Sonuç normalleştirilir ve gerekli yuvarlama işlemi yapılır.

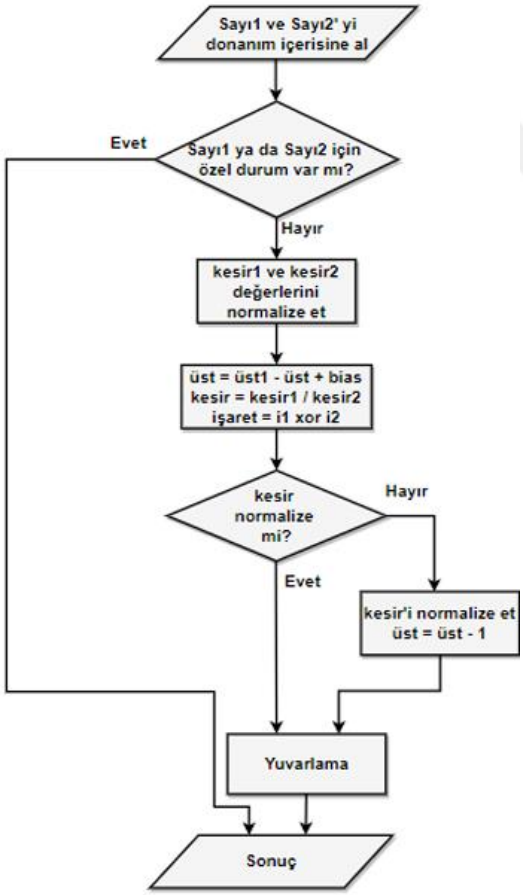
Bölme Akışı:

1. Üslülerin Çıkarılması: Üslü değerler birbirinden çıkarılır.

2. Mantisaların Bölünmesi: Mantisalar bölünür.

3. Normalleştirme: Bölme işlemi sonucu normalleştirilir.

4. Yuvarlama: Sonuç yuvarlanarak son haline getirilir.



Şekil 3.8: Bölme işlemi akış diyagramı

Sözde Kod:

```
Function FloatingPointDivision(A, B):
// 1. Üslülerin çıkarılması
Result.exponent = A.exponent - B.exponent
// 2. Mantisaların bölünmesi
Result.mantissa = A.mantissa / B.mantissa
// 3. Normalleştirme
Normalize(Result.mantissa, Result.exponent)
Return Result
```

3.15.5. Integer'dan Floating Point'e, 4 İşlem, Integer'a Dönüşüm

Bu akışta bir tam sayı önce kayan nokta formatına dönüştürülür, ardından dört temel işlemden biri gerçekleştirilir ve işlem sonrası sonuç tekrar tam sayı formatına dönüştürülür. Bu süreçte her iki format arasında geçiş yapılır ve kayan nokta aritmetiği kullanılır.

Sözde Kod:

```
Function IntegerToFloatingPoint(IntValue):
FloatingPoint.mantissa = IntValue
FloatingPoint.exponent = 0
// Normalleştirme
While FloatingPoint.mantissa >= 2:
ShiftRight(FloatingPoint.mantissa)
FloatingPoint.exponent += 1
```

```
        EndWhile
        Return FloatingPoint
    EndFunction
    Function FloatingPointToInteger(FPValue):
        IntValue = FPValue.mantissa
// Üslü değere göre tam sayıya dönüştür
        While FPValue.exponent > 0:
            ShiftLeft(IntValue)
            FPValue.exponent -= 1
        EndWhile

        Return IntValue
    EndFunction
    Function ProcessWithFourOperations(IntA, IntB, Operation):
// Step 1: Integer'ları Floating Point'e dönüştür
        FloatingA = IntegerToFloatingPoint(IntA)
        FloatingB = IntegerToFloatingPoint(IntB)
// Step 2: Seçilen işlemi yap
        If Operation == "Addition":
            Result = FloatingPointAddition(FloatingA, FloatingB)
        ElseIf Operation == "Subtraction":
            Result = FloatingPointSubtraction(FloatingA, FloatingB)
        ElseIf Operation == "Multiplication":
            Result = FloatingPointMultiplication(FloatingA, FloatingB)
        ElseIf Operation == "Division":
```

```
            Result = FloatingPointDivision(FloatingA, FloatingB)
        EndIf
    // Step 3: Floating Point'ten Integer'a dönüş
        FinalResult = FloatingPointToInteger(Result)

        Return FinalResult
    EndFunction
```

Akışın Açıklaması

1. Integer'dan Floating Point'e Dönüşüm: Tam sayılar, kayan nokta formatına dönüştürülür. Mantisa ve üslü belirlenir.

2. Kayan Nokta İşlemi: Toplama, çıkarma, çarpma ya da bölme işlemlerinden biri seçilir ve işlem gerçekleştirilir.

3. Floating Point'ten Integer'a Dönüşüm: Sonuç tekrar tam sayı formatına dönüştürülür.

Bu akış, FPGA gibi platformlarda paralel işlem gücünden faydalanarak çok daha hızlı ve geniş aralıklı hesaplamalar yapılmasını sağlar.

3.15.5.1. Integer – Floating Point Dönüşümü Adımları

1. Başlangıç: İşleme integer değeri okuyarak başla.

2. İşaret Bitini Belirle:

- Integer sayının pozitif mi negatif mi olduğunu kontrol et.
- Pozitifse işaret bitini `0` yap, negatifse işaret bitini `1` yap.

3. Mutlak Değeri Al:

- Negatifse integer değerinin mutlak değerini al.

4. Mantisa ve Üslü Hesaplama:

- Integer değerin mantisa ve üslü bileşenlerini belirle.

- Sayıyı normalleştir. Mantisanın başında 1 olacak şekilde hizala ve üslü değerini ayarla.

5. Mantisa ve Üslüyü Kayıt Et:

- Mantisa ve üslü değerini IEEE 754 formatına uygun olarak kaydet.

6. Sonucu Kontrol Et:

- Sonucun normalleştiğinden emin ol.

- Eğer mantisa 1'den küçükse daha fazla kaydırma yapılabilir.

7. Sonucu Dönüştür ve Yazdır:

- IEEE 754 formatında floating point sayı oluştur ve yazdır.

8. Son: İşlemi bitir.

3.15.5.2. Kayan Nokta (Floating Point) Toplama İşlemi Adımları

1. Başlangıç

- İlk iki kayan nokta sayısını (Sayı1 ve Sayı2) al.

2. Özel Durum Kontrolü

- Eğer Sayı1 veya Sayı2:

- Sıfırsa, diğer sayıyı sonuç olarak döndür.

- Sonsuzluk veya NaN (Not a Number) gibi özel bir duruma sahipse, bu özel durumları kontrol et ve gerekirse sonuç olarak bu değerlerden birini döndür.

3. Üslü Değerlerin Karşılaştırılması

- Sayı1 ve Sayı2'nin üslü (exponent) değerlerini karşılaştır.

- Eğer üslü değerler farklıysa, küçük üslüye sahip olan sayının mantisasını (fraction) büyük üslü değerine kadar kaydır.

- Bu işlem, sayıları aynı üslüye getirmek içindir. Sayıların üslüleri eşitlenene kadar küçük olan sayı mantisayı kaydırarak hizalanır.

4. Mantisaların Toplanması

- Mantisaları topla:

- Eğer işaretler aynıysa, mantisalar doğrudan toplanır.

- Eğer işaretler farklıysa (örneğin biri pozitif, diğeri negatifse), mantisalar birbirinden çıkarılır.

5.Normalleştirme Kontrolü

- Mantisaların toplanması veya çıkarılması sonrası, normalleştirme kontrolü yap:

- Eğer mantisanın değeri 1 ile başlamıyorsa, sayı kaydırılır ve mantisa normalleştirilir.

- Bu aşamada mantisa 1 ile başlamalıdır.

6. Üslü ve Mantisa Güncellemesi

- Normalleştirme işlemi sonrasında üslü değeri güncelle.

- Eğer mantisa sola kaydırıldıysa, üslü değeri artırılır.

- Eğer sağa kaydırıldıysa, üslü değeri azaltılır.

7. Yuvarlama İşlemi

- Eğer mantisa çok fazla bit içeriyorsa, sonucun belirli bir hassasiyete yuvarlanması gerekebilir. Mantisanın son bitlerini yuvarlayarak sonuç üret.

8. Sonuç

- İşaret biti, üslü ve mantisa değerlerini birleştirerek sonuç kayan nokta sayısını üret.

- IEEE 754 formatında kayan nokta sonucu elde et ve döndür.

9. Son

- İşlemi bitir.

3.15.5.3. Kayan Nokta (Floating Point) Çıkarma İşlemi Adımları

1. Başlangıç

- İlk iki kayan nokta sayısını (Sayı1 ve Sayı2) al.

2. Özel Durum Kontrolü

- Eğer Sayı1 veya Sayı2:

  - Sıfırsa, diğer sayıyı sonuç olarak döndür.

  - Sonsuzluk veya NaN (Not a Number) gibi özel bir duruma sahipse, bu özel durumları kontrol et ve gerekirse sonuç olarak bu değerlerden birini döndür.

3. Üslü Değerlerin Karşılaştırılması

- Sayı1 ve Sayı2'nin üslü (exponent) değerlerini karşılaştır:

  - Eğer üslü değerler farklıysa, küçük üslüye sahip olan sayının mantisasını (fraction) büyük üslü değerine kadar kaydır.

  - Bu işlem, sayıları aynı üslüye getirmek içindir. Sayıların üslüleri eşitlenene kadar küçük olan sayı mantisayı kaydırarak hizalanır.

4. Mantisaların İşaret Kontrolü

- İşaretlerin Karşılaştırılması:

  - Eğer işaretler aynıysa, çıkarma işlemi yapılır (mantisa değerleri birbirinden çıkarılır).

  - Eğer işaretler farklıysa (örneğin, biri pozitif diğeri negatifse), toplama işlemi yapılır. Bu durumda aslında toplama işlemi gerçekleştirilir (pozitif ve negatif sayı toplandığı için çıkarma etkisi yaratır).

5. Mantisaların Çıkarılması

- Mantisaların Çıkarılması:

  - Eğer işaretler aynıysa, büyük mantisa'dan küçük mantisa çıkarılır.

  - Eğer işaretler farklıysa, mantisalar toplanır.

6. Normalleştirme Kontrolü

- Normalleştirme:

- Çıkarma sonrası oluşan mantisa normalleştirilmelidir.

- Eğer mantisanın başında bir `1` yoksa, mantisa kaydırılır ve üslü değeri buna göre güncellenir.

- Eğer mantisa 0 olduysa, sonuç sıfırdır.

7. Üslü ve Mantisa Güncellemesi

- Üslü Değeri Güncelle:

- Eğer mantisa sola kaydırıldıysa, üslü değeri artırılır.

- Eğer sağa kaydırıldıysa, üslü değeri azaltılır.

8. Yuvarlama İşlemi

- Eğer mantisa çok fazla bit içeriyorsa, sonucun belirli bir hassasiyete yuvarlanması gerekebilir.

- Sonuç mantisası yuvarlanarak belirlenen hassasiyete getirilir.

9. Sonuç

- İşaret Biti, Üslü ve Mantisayı Birleştir:

- İşaret biti, üslü ve mantisa değerlerini birleştirerek sonuç kayan nokta sayısını üret.

- IEEE 754 formatında kayan nokta sonucu elde et ve döndür.

10. Son

- İşlemi bitir.

3.15.5.4. Kayan Nokta (Floating Point) Çarpma İşlemi Adımları

1. Başlangıç

- İlk iki kayan nokta sayısını (Sayı1 ve Sayı2) al.

2. Özel Durum Kontrolü

- Eğer Sayı1 veya Sayı2:

  - Sıfırsa, sonuç sıfırdır.

  - Sonsuzluk ya da NaN (Not a Number) gibi özel bir duruma sahipse, bu özel durumları kontrol et. Eğer iki sayıdan biri sonsuz ise sonuç sonsuz olur; biri NaN ise sonuç NaN olur.

3. İşaret Bitinin Hesaplanması

- Sayı1 ve Sayı2'nin işaret bitlerini kontrol et:

  - İşaret bitleri XOR (Exclusive OR) işlemiyle hesaplanır:

    - Eğer iki sayının işaret biti aynıysa (ikisi de pozitif ya da ikisi de negatifse), sonuç pozitif olur (işaret biti 0).

    - Eğer işaret bitleri farklıysa, sonuç negatif olur (işaret biti 1).

4. Üslü Değerlerin Toplanması

- Sayı1 ve Sayı2'nin üslü (exponent) değerlerini topla:

  - Yeni üslü değeri: `Üst1 + Üst2 - Bias`

  - Bias (ön yargı) IEEE 754 standardına göre genellikle 127 (single precision) veya 1023 (double precision) olur.

5. Mantisaların Çarpılması

- Sayı1 ve Sayı2'nin mantisalarını çarp:

  - Mantisalar genellikle 1 ile normalleştirildiği için, `1.Mantisa1 * 1.Mantisa2` şeklinde çarpma işlemi yapılır.

  - Çarpım sonucu iki kat uzunlukta bir mantisa oluşturabilir.

6. Normalleştirme Kontrolü

- Mantisayı kontrol et:

  - Eğer çarpım sonucu mantisanın başında bir `1` varsa, normaldir.

  - Eğer mantisanın başında bir `1` yoksa, mantisayı sola kaydırarak normalleştir.

- Normalleştirme sırasında mantisayı sola kaydırırsan, üslü değerini bir azalt.

7. Yuvarlama

- Mantisada fazladan bit varsa, sonucun belirli bir hassasiyete yuvarlanması gerekebilir.

- Mantisanın son bitlerini yuvarlayarak sonuç elde edilir.

8. Taşma ve Alt Taşma Kontrolü

- Üslü değeri taşma (overflow) veya alt taşma (underflow) durumu için kontrol edilmelidir:

- Eğer üslü değeri maksimumu geçerse, sonuç sonsuz olur.

- Eğer üslü değeri minimumun altına inerse, sonuç sıfır olur.

9. Sonuç

- İşaret biti, üslü ve mantisa değerlerini birleştirerek sonuç kayan nokta sayısını üret.

- IEEE 754 formatında kayan nokta sonucu elde et ve döndür.

10. Son

- İşlemi bitir.

3.15.5.5. Kayan Nokta (Floating Point) Bölme İşlemiAdımları

1. Başlangıç

- İlk iki kayan nokta sayısını (Sayı1 ve Sayı2) al.

2. Özel Durum Kontrolü

- Eğer Sayı1 veya Sayı2:

- Sayı1 sıfır ve Sayı2 sıfır değilse, sonuç sıfırdır.

- Sayı2 sıfırsa (bölme işleminde bölünen sıfır olamaz), sonuç sonsuzdur.

- Eğer Sayı1 ve Sayı2 özel bir değerse (örneğin sonsuzluk veya NaN), bu özel durumları kontrol et ve buna göre sonucu döndür.

3. İşaret Bitinin Hesaplanması

- Sayı1 ve Sayı2'nin işaret bitlerini kontrol et:

  - İşaret bitleri XOR (Exclusive OR) işlemiyle hesaplanır:

    - Eğer iki sayının işaret biti aynıysa (ikisi de pozitif ya da ikisi de negatifse), sonuç pozitif olur (işaret biti 0).

    - Eğer işaret bitleri farklıysa, sonuç negatif olur (işaret biti 1).

4. Üslü Değerlerin Çıkarılması

- Sayı1 ve Sayı2'nin üslü (exponent) değerlerini çıkar:

  - Yeni üslü değeri: `Üst1 - Üst2 + Bias`

  - Bias (ön yargı) IEEE 754 standardına göre genellikle 127 (single precision) veya 1023 (double precision) olur.

5. Mantisaların Bölünmesi

- Sayı1 ve Sayı2'nin mantisalarını böl:

  - Mantisalar genellikle 1 ile normalleştirildiği için, `1.Mantisa1 / 1.Mantisa2` şeklinde bölme işlemi yapılır.

  - Bölme sonucu normalleştirilmemiş bir mantisa elde edilir.

6. Normalleştirme Kontrolü

- Mantisayı kontrol et:

  - Eğer bölme sonucu mantisanın başında bir `1` yoksa, mantisa sola kaydırılarak normalleştirilir.

  - Normalleştirme sırasında mantisa sola kaydırılırsa, üslü değeri bir azaltılır.

7. Yuvarlama

- Eğer mantisa fazladan bit içeriyorsa, sonucun belirli bir hassasiyete yuvarlanması gerekebilir.

- Mantisanın son bitlerini yuvarlayarak sonuç elde edilir.

8. Taşma ve Alt Taşma Kontrolü

- Üslü değeri taşma (overflow) veya alt taşma (underflow) durumu için kontrol edilmelidir:

  - Eğer üslü değeri maksimumu geçerse, sonuç sonsuz olur.

  - Eğer üslü değeri minimumun altına inerse, sonuç sıfır olur.

9. Sonuç

- İşaret biti, üslü ve mantisa değerlerini birleştirerek sonuç kayan nokta sayısını üret.

- IEEE 754 formatında kayan nokta sonucu elde et ve döndür.

10. Son

- İşlemi bitir.

3.15.5.6. Floating Point - Integer Dönüşümü Adımları

1. Başlangıç: Floating point sayıyı al.

2. İşaret Bitini Kontrol Et:

  - İşaret biti `0` ise sayı pozitif, `1` ise negatif olarak kabul et.

3. Mantisa ve Üslü Değerlerini Ayır:

  - IEEE 754 formatında mantisa ve üslü değerlerini ayır.

4. Mantisa'yı Genişlet:

  - Mantisa değerini üslü kadar sola kaydırarak genişlet.

  - Üslü değeri floating point formatında kaydırma için kullanılır.

5. Üslü Değerini Kontrol Et:

  - Üslü değerinin pozitif veya negatif olup olmadığını kontrol et.

  - Pozitifse mantisa sola kaydırılır, negatifse mantisa sağa kaydırılır.

6. Sonucun Negatif Olup Olmadığını Kontrol Et:

  - Eğer işaret biti `1` ise sonucu negatif yap.

7. Sonuç:

- Floating point değerden integer'a dönüştürülmüş sonucu üret.

8. Son: Dönüştürme işlemini bitir.

BÖLÜM 4. TARTIŞMA VE SONUÇ

Bu çalışmada, Gömülü Sistemlerde Kayan Nokta Aritmetik Hesaplamalar İçin Verilog Tabanlı Sentezlenebilir IP Core Kütüphanesi geliştirilmiştir. Kayan nokta aritmetiği, bilimsel hesaplamalar, mühendislik simülasyonları, görüntü işleme, yapay zeka ve veri analitiği gibi birçok uygulama alanında geniş bir sayı aralığında yüksek hassasiyetli işlem yapabilmek için kritik bir yapı sunar. Bu çalışma kapsamında, geliştirilen IP core kütüphanesi simülasyonlarla test edilmiş ve işlevselliği doğrulanmıştır.

Simülasyon sonuçları, geliştirilen IP core kütüphanesinin hem doğruluk hem de verimlilik açısından başarılı olduğunu göstermiştir. Bu bölümde bulgular, çalışma sonuçları ve gelecekte yapılabilecek iyileştirmeler detaylı olarak ele alınacaktır.

4.1. Bulgular

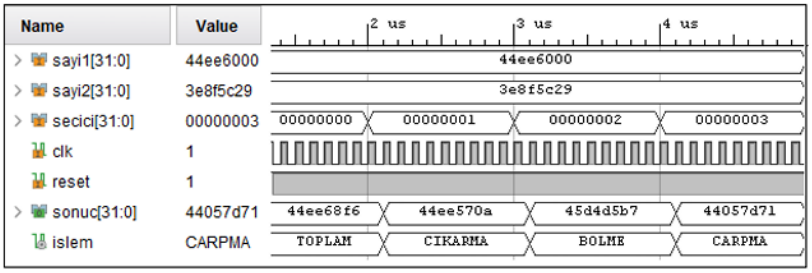
4.1.1. Simülasyon Sonuçlar

Xilinx firmasının Vivado 2019.1 derleyicisi, tasarımın davranışsal, fonksiyonel ve zamanlama simülasyonlarını yapabilme yeteneğine sahiptir. Vivado derleyicisi kullanılarak, her biri iki adet 32-bit kayan noktalı sayı içeren üç farklı sayı grubunun FPU üzerinde davranışsal simülasyonu gerçekleştirilmiştir. Seçici bitlere farklı değerler atanarak, bu sayı grupları için toplama, çıkarma, çarpma ve bölme işlemleri kayan noktalı sayı formatında yapılmıştır. Sayı gruplarının ayrıntıları Tablo 3.1'de gösterilirken, Vivado'da elde edilen simülasyon sonuçları ise Şekil 3.9'da sunulmuştur.

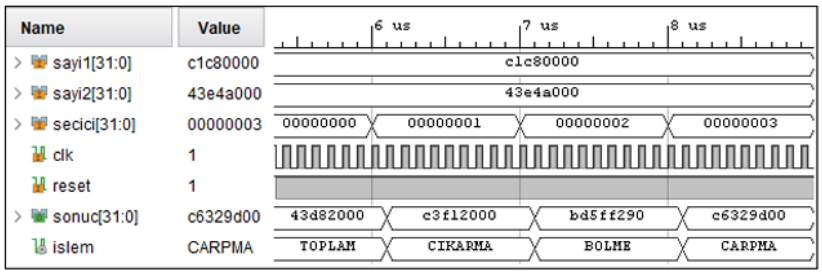
Sayı Grupları	32 Bit Kayan Noktalı Gösterim	Decimal Gösterim
1. Grup N1	01000110011100111110111000000000	1907
1. Grup N2	00111111000111011001111010100001	0,28
2. Grup N1	11000001110010000000000000000000	-25
2. Grup N2	01000111000100111000000000000000	457,25
3. Grup N1	00110111101100000000000000000000	0,0025
3. Grup N2	10111111111010101010101000000000	-0,012

Şekil 3.9. Gerçekleştirilen Sayı Grupları

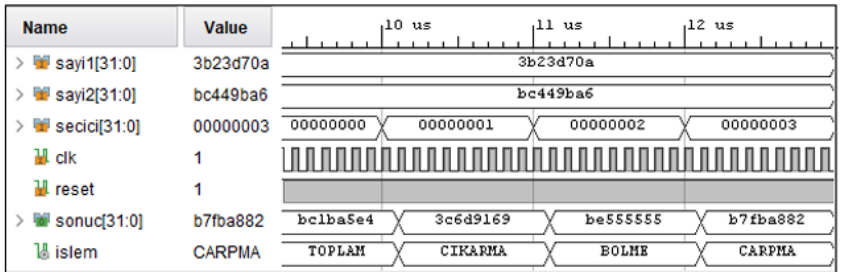
Şekil 3.9'dan da anlaşılacağı üzere, tasarlanan FPU'nun farklı sayılarla yaptığı simülasyonlarda işlemler doğru sonuçlar vermektedir.



Şekil 3.10: Simülasyon sonuçları - 1



Şekil 3.11: Simülasyon sonuçları - 2



Şekil 3.12: Simülasyon sonuçları – 3

Her işlem birimi donanım olarak tasarlandığında, FPGA üzerinde belirli kaynakları tüketir. Bu kısımda Zynq-7000 FPGA cihazı kullanılmıştır. Bu kaynaklar arasında genel olarak LUT, Flip-Flop, Buffer ve giriş/çıkış pinleri bulunur. FPGA'de bu kaynakların mevcut miktarları ile her bir işlem biriminin bu kaynakları ne kadar kullandığını gösteren tablo Tablo 3.7 'de yer almaktadır.

FPGA Kaynakları	FPGA	Toplayıcı/Çıkarıcı	Çarpıcı	Bölücü
LUT	53200	785	1654	1825
Flip-Flop	106400	96	96	96
I/O Pinleri	200	98	98	98
Buffer	32	1	1	1

Şekil 3.13. FPGA içerisinde kaynak kullanım bilgileri

4.1.2. Doğruluk ve Hassasiyet

Geliştirilen IP core kütüphanesinin IEEE 754 standardına uygun olarak yüksek doğruluk sağladığını göstermiştir. Toplama, çıkarma, çarpma ve bölme gibi dört temel matematiksel işlemde hata oranlarının çok düşük olduğu gözlemlenmiştir. Kayan nokta işlemlerinin simülasyonlar sırasında doğru bir şekilde sonuçlanması, mühendislik projeleri, bilimsel hesaplamalar ve yapay zeka gibi yüksek doğruluk gerektiren uygulamalarda bu IP core'un kullanılabilir olduğunu göstermektedir.

Bu projede, kayan nokta (floating point) sayı formatı tercih edilmiştir çünkü floating point formatı, fixed point formatına kıyasla çok daha geniş bir sayı aralığı sunar. Özellikle, 32-bit IEEE 754 floating point formatı, yaklaşık olarak $3.4x10^{38}$ maksimum ve $3.4x10^{-38}$ minimum değerler arasında çalışabilme kapasitesine sahiptir. Bu geniş sayı aralığı, yüksek doğruluk ve hassasiyet gerektiren mühendislik ve bilimsel hesaplamalarda büyük bir avantaj sağlar. Sabit nokta (fixed point) formatı daha dar bir aralık sunarken, floating point'in sunduğu bu geniş aralık sayesinde, projede ihtiyaç duyulan hassas matematiksel işlemler başarıyla gerçekleştirilmiştir.

- 32-bit işaretli fixed point tam sayı en büyük değer $2^{31} - 1$ = 2147483647, en küçük değer $2^{-31} - 1$ =–2147483648
- 32-bit işaretsiz fixed point tam sayı en büyük değer $2^{32} - 1$ = 4294967295

- 32-bit IEEE 754 floating point (tek hassasiyetli): Maksimum değer yaklaşık $3.4x10^{38}$, minimum pozitif değer yaklaşık $1.18x10^{-38}$.

Özellikle büyük veri setleri üzerinde yapılan simülasyonlarda, hassasiyetin korunması ve hata oranlarının düşük kalması, IP core kütüphanesinin başarılı sonuçlar verdiğini doğrulamaktadır.

4.1.3. Performans ve Hız

Geliştirilen Verilog tabanlı IP core kütüphanesi, simülasyon ortamında verimli ve hızlı çalışmıştır. Mantisaların ve üslülerin işlenmesinde kullanılan algoritmalar, işlem süresini kısaltmış ve hızlı bir şekilde doğru sonuçlar üretilmiştir. Kayan nokta işlemlerinin hızını artırmak için yapılan optimizasyonlar, özellikle paralel işlem yetenekleri ile birlikte, yüksek verimlilik sağlamıştır.

Simülasyon ortamında yapılan testler, IP core'un işlem hızının tatmin edici olduğunu ve gömülü sistemler için uygun bir çözüm sunduğunu göstermiştir. Özellikle zaman kısıtlaması olan gerçek zamanlı uygulamalarda bu hız avantajı önemli bir katkı sağlayabilir.

4.1.4. Geliştirilmiş Optimizasyon Teknikleri

IP core modüllerinde uygulanan çeşitli optimizasyonlar sayesinde mantisa ve üslü hesaplamaları daha verimli hale getirilmiştir. Özellikle yuvarlama ve normalleştirme aşamalarında kullanılan optimize edilmiş algoritmalar, simülasyon sırasında daha hızlı işlem yapılmasını sağlamıştır. Bu optimizasyonlar sayesinde, hesaplama süreleri önemli ölçüde azaltılmış ve donanım kaynaklarının verimli bir şekilde kullanıldığı gözlemlenmiştir.

4.2. Tartışma

Bu çalışmada, gömülü sistemlerde yüksek doğrulukta kayan nokta aritmetiği işlemleri gerçekleştiren bir IP core tasarladık. Literatürde, genellikle 32 bit kayan nokta aritmetik işlemleri için FPGA tabanlı tasarımların VHDL dili ile yapıldığı görülmektedir.

Örneğin, Özkılbaç ve Karacalı (2021) tarafından yapılan çalışmada, Xilinx Zynq-7000 FPGA'si üzerinde VHDL ile geliştirilen 32 bit kayan nokta ALU tasarımı bulunmaktadır. Bu tasarım, IEEE 754 standardına uygun olarak toplama, çıkarma, çarpma ve bölme gibi temel işlemleri gerçekleştirmiş ve özellikle sinyal işleme ile yapay zeka gibi yoğun hesaplama gerektiren alanlarda kullanılmıştır.

Biz, bu çalışmaya benzer şekilde gömülü sistemlerde kayan nokta aritmetik işlemlerini optimize etmeyi amaçladık. Ancak, literatürde Verilog tabanlı çözümler üzerine yapılmış detaylı bir çalışma olmaması nedeniyle, bu açığı kapatmak adına tasarımımızı Verilog ile gerçekleştirdik. Bu tercihimizi yaparken, Teknofest Çip Yarışması'nın sağladığı motivasyonla özgün ve yenilikçi bir çözüm sunmak istedik. Verilog'un daha hızlı öğrenim sağlaması, prototipleme sürecini hızlandırması ve endüstride yaygın olarak kullanılması gibi avantajları, geliştirme sürecimizde verimliliğimizi artırdı.

Özkılbaç ve Karacalı (2021) çalışmasında görüldüğü gibi, VHDL genellikle daha katı bir sözdizimi ve güvenilirlik sunmaktadır; ancak Verilog tabanlı IP core'umuzun sağladığı esneklik, modüler yapı ve geniş platform desteği ile gömülü sistemlerde etkin bir çözüm sunduğunu gözlemledik. Çalışmamızla, VHDL temelli tasarımlara göre daha kısa geliştirme süreci ve optimize edilmiş bir performans sağlayarak literatürde önemli bir katkı sunduğumuza inanıyoruz.

4.2.1. Literatüre Katkısı

Bu çalışma, literatüre kayan nokta aritmetiği konusunda önemli bir katkı sunmaktadır. Mevcut çözümler genellikle ya donanım hızlandırması sağlamadan yazılım tabanlı olarak çalışmakta ya da belirli platformlara bağımlı olmaktadır. Bu çalışma, Verilog tabanlı, sentezlenebilir ve platform bağımsız bir IP core kütüphanesi sunarak literatüre yeni bir çözüm getirmektedir.

Çalışmada geliştirilen IP core kütüphanesi, özellikle gömülü sistemler ve yüksek performans gerektiren projelerde kullanılmak üzere uygun bir çözüm sunmaktadır. Literatürdeki mevcut çözümlerle kıyaslandığında, bu kütüphanenin optimize edilmiş olması ve geniş bir uygulama alanına sahip olması, onu diğerlerinden ayıran önemli bir özellik olarak öne çıkmaktadır.

4.2.2. Gelecekte Yapılabilecek İyileştirmeler

Simülasyonlarla test edilen IP core kütüphanesi, daha da geliştirilebilecek bazı yönlere sahiptir. Gelecekte yapılabilecek iyileştirmeler şunlar olabilir:

1. Donanım Kaynaklarının Verimli Kullanımı:

- Donanımda yer alacak olan optimizasyonlar, özellikle paralel işlem kapasitesinin artırılması ve daha verimli kaynak kullanımı üzerinde çalışılabilir. Bu, kütüphanenin daha fazla işlem yapabilme kapasitesini artıracak ve daha büyük projelerde verimliliği sağlayacaktır.

2. Daha Gelişmiş Optimizasyon Teknikleri:

- Özellikle mantisa ve üslü hesaplamalarında daha ileri düzey algoritmaların kullanılması, hesaplama sürelerini daha da kısaltabilir. Daha gelişmiş optimizasyonlar, IP core kütüphanesinin performansını daha da artırarak daha yüksek hızlı sonuçlar elde edilmesini sağlayabilir.

3. Çift Hassasiyetli Kayan Nokta İşlemleri:

- Mevcut çalışma, tek hassasiyetli (single precision) kayan nokta işlemleri için geliştirilmiş olsa da, çift hassasiyetli (double precision) işlemler için de genişletilebilir. Çift hassasiyetli işlemler, daha geniş veri aralığı ve daha yüksek doğruluk gerektiren uygulamalarda kullanılabilir. Bu genişleme, IP core kütüphanesinin daha fazla kullanım alanına hitap etmesini sağlayacaktır.

4. Farklı FPGA ve Donanım Platformlarına Uyarlama:

- Geliştirilen IP core kütüphanesi, şu anda simülasyon ortamında başarıyla test edilmiştir, ancak gelecekte farklı FPGA üreticilerine (Xilinx, Intel vb.) yönelik modifikasyonlar yapılabilir. Bu sayede farklı platformlar için optimize edilmiş bir çözüm sunulabilir.

5. Gömülü Sistemler İçin Genişletilebilirlik:

- Bu IP core kütüphanesi, sadece belirli alanlar için değil, aynı zamanda geniş bir uygulama yelpazesi için genişletilebilir. Örneğin, gerçek zamanlı sistemlerde, yapay zeka uygulamalarında ve güvenlik sistemlerinde daha fazla işlem yapılabilecek şekilde kütüphane genişletilebilir.

Kayan nokta işlemlerinin verimli bir şekilde optimize edilmesi, sadece donanım tarafında değil, aynı zamanda yazılım ve donanım entegrasyonu üzerinde de iyileştirme yapılmasını gerektirir.

Yazılım seviyesinde yapılacak olan optimizasyonlar ve donanım hızlandırması birleştirildiğinde, kayan nokta aritmetiği işlemlerinin performansı daha da artırılabilir. Gömülü sistemlerde yazılım-donanım uyumunun sağlanması, işlem sürelerini daha da kısaltarak gerçek zamanlı sistemlerde kullanılabilirlik açısından avantaj sağlayacaktır.

4.3. Sonuçlar ve Performans Analizi

Bu çalışmada geliştirilen Verilog tabanlı IP core, kayan nokta aritmetiği işlemlerini optimize etmek için tasarlanmış olup, farklı platformlardaki çözümlerle karşılaştırıldığında dikkate değer performans iyileştirmeleri sağlamıştır. Simülasyon sonuçlarına göre, geliştirilen IP core, yüksek doğrulukla kayan nokta işlemlerini gerçekleştirmiş ve simülasyon ortamında başarılı sonuçlar elde edilmiştir.

Sonuçlar, CPU ve GPU tabanlı sistemlerle karşılaştırılarak değerlendirilmiştir. CPU ve GPU üzerinde çalışan geleneksel yazılım tabanlı kayan nokta aritmetiği çözümleri ile karşılaştırıldığında, donanım tabanlı bir çözüm olan Verilog tabanlı IP core'un hesaplama süresi açısından ciddi bir avantaj sağladığı görülmüştür. Özellikle paralel işlem yapabilme kapasitesi sayesinde büyük veri setleri üzerinde daha hızlı sonuçlar elde edilmiştir.

Ayrıca, enerji tüketimi metrikleri de göz önüne alındığında, donanım tabanlı bu çözümün enerji verimliliği açısından da önemli bir kazanç sunduğu görülmektedir. CPU ve GPU'lar, genel amaçlı işlemciler olmaları nedeniyle daha fazla enerji tüketmekteyken, geliştirilen IP core'un gömülü sistemler için optimize edilmiş olması, enerji tüketimini minimuma indirmiştir. Bu, gömülü sistemlerin enerji verimliliği gerektiren uygulamalarda kullanımı için önemli bir avantaj sağlamaktadır.

Bu performans analizleri, projede önerilen çözümün özellikle gerçek zamanlı hesaplama gerektiren ve enerji tüketiminin önemli olduğu gömülü sistemler gibi uygulamalarda geniş bir kullanım alanı bulabileceğini göstermektedir. Geliştirilen IP core, özellikle CPU ve GPU tabanlı çözümlerle karşılaştırıldığında hem hız hem de enerji verimliliği açısından belirgin bir avantaj sunmaktadır.

4.4. Sonuç

Bu çalışmada geliştirilen Verilog tabanlı sentezlenebilir IP core kütüphanesi, kayan nokta aritmetik işlemlerini optimize etmek ve hızlandırmak için önemli bir çözüm sunmaktadır. Simülasyon sonuçları, bu kütüphanenin yüksek doğruluk ve hızlı işlem yapma kapasitesine sahip olduğunu göstermektedir. Özellikle geniş sayı aralığı ve yüksek hassasiyet gerektiren uygulamalarda, geliştirilen IP core kütüphanesi önemli bir avantaj sağlamaktadır.

Bu kütüphanenin literatüre katkısı, platform bağımsız bir çözüm sunarak mevcut çözümlere kıyasla daha geniş bir kullanım alanı sağlamasıdır. Gömülü sistemler, mühendislik projeleri, bilimsel araştırmalar ve yapay zeka gibi alanlarda kullanılabilir olması, kütüphanenin çok yönlülüğünü artırmaktadır.

Gelecekte yapılacak iyileştirmelerle birlikte, bu çalışma daha da genişletilebilir ve optimize edilebilir. Donanım kaynaklarının daha verimli kullanımı, çift hassasiyetli işlemler için genişleme ve farklı platformlara uyarlama, kütüphanenin kullanım alanlarını artıracaktır. Bu çalışma, gömülü sistemler ve donanım hızlandırmalı hesaplama projelerinde önemli bir kaynak olarak değerlendirilecektir.

KAYNAKLAR

Afonso, J., Santos, J., Sousa, L., & Monteiro, J. (2010). IEEE 754 compliant floating-point hardware for FPGAs. In *International Conference on Field-Programmable Technology* (pp. 202-207). IEEE.

Akbaş, H. (2015). *Object tracking with template matching method and its high speed FPGA implementation* (Yüksek Lisans Tezi). Akdeniz Üniversitesi, Fen Bilimleri Enstitüsü, Antalya.

Ashenden, P. J. (2008). *The designer's guide to VHDL*. Morgan Kaufmann Publishers.

Baysal, K. (2015). *High capacity multiplier unit design with FPGA for cryptographic operations* (Yüksek Lisans Tezi). Trakya Üniversitesi, Fen Bilimleri Enstitüsü, Edirne.

Bedir, N. S. (2018). *FPGA tabanlı 64 bit aritmetik mantık birimi tasarımı* (Yüksek Lisans Tezi). İstanbul Üniversitesi-Cerrahpaşa, Lisansüstü Eğitim Enstitüsü, İstanbul.

Belanović, P., & Leeser, M. (2002). A library of parameterized floating-point modules and their use. In *International Conference on Field Programmable Logic and Applications* (pp. 948-957). Springer.

Bhasker, J. (1998). *A Verilog HDL Primer*. Star Galaxy Publishing.

Bhasker, J. (1998). *A VHDL Primer*. Prentice Hall.

Bobda, C. (2007). *Introduction to reconfigurable computing: Architectures, algorithms, and applications*. Springer.

Bozgan, C. (2019). *A high throughput FPGA implementation of Markov Chain Monte Carlo method for mixture models* (MS Thesis). Middle East Technical University, Fen Bilimleri Enstitüsü, Ankara.

Bruguera, J. D., & Lang, T. (1999). Leading-one prediction with concurrent position correction. *IEEE Transactions on Computers*, *48*(10), 1083-1097.

Chandrasetty, V. A. (2011). *VLSI Design: A practical guide for FPGA and ASIC implementations*. Springer Science & Business Media.

Chen, S., Hu, W., & Li, Z. (2019). High performance data encryption with AES implementation on FPGA. In *2019 IEEE 5th International Conference on Big Data Security on Cloud (BigDataSecurity)* (pp. 77-83). IEEE.

Chu, P. P. (2006). *RTL hardware design using VHDL: Coding for efficiency, portability, and scalability*. John Wiley & Sons.

Ciletti, M. D. (2004). *Advanced digital design with the Verilog HDL*. Prentice Hall.

Deschamps, J. P., Sutter, G. D., & Cantó, E. (2012). *Guide to FPGA implementation of arithmetic functions*. Springer Science & Business Media.

Dido, J., Geraudie, N., Loiseau, L., Payeur, O., Savaria, Y., & Poirier, D. (2002). A flexible floating-point format for optimizing data-paths and operators in FPGA-based DSPs. In *Proceedings of the 2002 ACM/SIGDA Tenth International Symposium on Field-Programmable Gate Arrays* (pp. 50-55). ACM.

Dorf, R. C. (2005). *Computers, software engineering, and digital devices*. CRC Press.

Furber, S. B. (2000). *ARM system-on-chip architecture* (2nd ed.). Addison-Wesley.

Hennessy, J. L., & Patterson, D. A. (2017). *Computer architecture: A quantitative approach*. Morgan Kaufmann.

IEEE Standards Board. (1985). IEEE standard for binary floating-point arithmetic. The Institute of Electrical and Electronics Engineers.

IEEE. (2005). IEEE Standard 1364-2005 for Verilog Hardware Description Language (Verilog-2005).

IEEE. (2008). IEEE Standard 1076-2008 for VHDL Language (VHDL-2008).

Kuon, I., & Rose, J. (2007). Measuring the gap between FPGAs and ASICs. *ACM Transactions on Design Automation of Electronic Systems (TODAES, 12*(2), 1-22.

Liu, D., & Zhang, W. (2011). FPGA-based parallel AES encryption engines. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 19*(7), 1348-1358.

Nakashima, H., & Shimada, M. (2008). The influence of IEEE-754 rounding modes on FPGA-based floating-point multiply-add units. *IEEE Transactions on Computers, 57*(9), 1256-1263.

Özkılbaç, B. (2020). *IEEE754 kayan noktalı sayı formatında aritmetik mantık birimi tasarımı ve gerçek zamanlı olarak uygulanması* (Yüksek Lisans Tezi). Atatürk Üniversitesi, Fen Bilimleri Enstitüsü, Erzurum.

Özkılbaç, B., & Karacalı, T. (2021). Implementation and design of 32-bit floating-point ALU on a hybrid FPGA-ARM platform. *Journal of Brilliant Engineering, 2,* 25-61. https://doi.org/10.36937/ben.2020.001.005

Palnitkar, S. (2003). *Verilog HDL: A guide to digital design and synthesis*. Prentice Hall.

Patterson, D. A., & Hennessy, J. L. (2013). *Computer organization and design: The hardware/software interface*. Elsevier.

Roth, C. H., & John, L. K. (2004). *Digital systems design using VHDL*. Brooks/Cole.

Seals, R. C., & Whapshott, G. F. (1997). *Programmable logic: PLDs and FPGAs*. Macmillan International Higher Education.

Shirazi, N., Walters, A., & Athanas, P. (1995). Quantitative analysis of floating point arithmetic on FPGA based custom computing machines. In *Proceedings IEEE Symposium on FPGAs for Custom Computing Machines* (pp. 155-162). IEEE.

Smith, J. E. (1998). *Microprocessor design: RISC/CISC and beyond*. Springer.

Sugie, T., Akamatsu, T., Nishitsuji, T., Hirayama, R., Masuda, N., Nakayama, H., & Endo, Y. (2018). High-performance parallel computing for next-generation holographic imaging. *Nature Electronics, 1*(4), 254-259.

Thomas, D. E., & Moorby, P. R. (1998). *The Verilog hardware description language*. Springer.

Upton, E., & Halfacree, G. (2016). *Raspberry Pi user guide*. John Wiley & Sons.

Xilinx Inc. *Xilinx User Guide and Documentation*.

Yağlıkçı, A. G. (2014). *Design of an FPGA based co-processor for digital signal processing applications* (Yüksek Lisans Tezi). TOBB Ekonomi ve Teknoloji Üniversitesi, Fen Bilimleri Enstitüsü, Ankara.

Zalewski, K., Chojnacki, A., & Maciejewski, T. (2019). High-performance FPGA-based floating-point arithmetic unit. *International Journal of Reconfigurable Computing, 2019*(3), 1-11.

ÖZGEÇMİŞ

Kişisel Bilgiler

Adı Soyadı: Esin Mutlu Korkmaz

Eğitim

Lisans: Sakarya Üniversitesi, Mühendislik Fakültesi, Elektrik-Elektronik Mühendisliği Bölümü

Yabancı Dil Bilgisi

- İngilizce: Çok İyi
- Korece: İyi

Üye Olunan Mesleki Kuruluşlar

- Elektrik Mühendisleri Odası

Tezden Üretilmiş Yayınlar

1. Mutlu, E., Dereli, S., & Ünsal, A. (2021). *Digital Design and FPGA Implementation that Converts an Integer to a Floating Point Number*. 2nd International Conference on Informatics and Computer Science, Sakarya University of Applied Sciences.