İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

VHDL İLE HÜCRESEL YAPAY SİNİR AĞI GERÇEKLEMESİ

YÜKSEK LİSANS TEZİ

Mehmet TÜKEL

Anabilim Dalı : Elektronik ve Haberleşme Mühendisliği

Programı : Elektronik Mühendisliği

OCAK 2009

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

VHDL İLE HÜCRESEL YAPAY SİNİR AĞI GERÇEKLEMESİ

YÜKSEK LİSANS TEZİ
Mehmet TÜKEL
(504051214)

Tezin Enstitüye Verildiği Tarih : 29 Aralık 2008

Tezin Savunulduğu Tarih : 21 Ocak 2009

Tez Danışmanı : Doç. Dr. Müştak Erhan YALÇIN (İTÜ)
Diğer Jüri Üyeleri : Prof. Dr. Ece Olcay GÜNEŞ (İTÜ)
Prof. Dr. Vedat TAVŞANOĞLU (YTÜ)

ŞUBAT 2009

ÖNSÖZ

Tez danışmanım Doç. Dr. Müştak Erhan Yalçın'a, Hücresel Yapay Sinir Ağı konusunda yüksek lisans tezinden faydalandığım Recep Uslu'ya, araştırma görevlisi arkadaşım Özlem Özbudak'a ve benden hiçbir zaman desteklerini esirgemeyen çok sevdiğim aileme teşekkür ederim.

Aralık 2008 Mehmet TÜKEL

İÇİNDEKİLER

Sayfa

ÖZET ..... **xiii**
SUMMARY ..... **xv**
1. GİRİŞ **1**
2. HÜCRESEL SİNİR AĞLARI **3**
2.1 Tarihsel Gelişim 3
2.2 Hücresel Sinir Ağlarının Mimarisi 4
2.3 Hücre Yapısı 7
2.4 Zamandan ve Konumdan Bağımsız HSA 8
2.4.1 Klonlama şablonu gösterilimi 9
2.4.1.1 Geri besleme operatöründen (A(i-k , j-l)) gelen katkı 9
2.4.1.2 Giriş (kontrol) operatöründen (B(i-k, j-l)) gelen katkı 10
2.5 HSA Benzetim Programı 11
3. VHDL İLE DİJİTAL TASARIM **13**
3.1 Dijital Tasarımın Safhaları 13
3.2 Tanımlama Seviyeleri 14
3.3 Çok Hızlı Tümleşik Devre Donanım Tanımlama Dili 15
3.4 VHDL ile Benzetim 16
3.5 Blok RAM ile İlgili Ön Bilgi 17
4. DİJİTAL HÜCRESEL SİNİR AĞI GERÇEKLEMELERİ **19**
4.1 Matematiksel Modelin Yeniden İfadesi 19
4.2 Devrenin Dışarıdan En Genel Halde Gösterilişi 20
4.3 Görüntünün Saklanması ve İşlenmesi 22
4.4 Kaydedici Transfer Düzeyinde Bir Hücresel İşlemcinin Tasarımı 23
4.4.1 Kaydedici dizisi tasarımının ayrıntıları 24
4.4.2 Aritmetik birimin tasarımının ayrıntıları 26
4.5 Tasarımın Diğer Tasarımlarla Karşılaştırılması 27
4.6 Tasarımın Pipe-Line Yöntemiyle Hızlandırılması 29
5. VHDL İLE YENİ BİR DİJİTAL HÜCRESEL SİNİR AĞI TASARIMI **35**
5.1 Önerilen Gerçeklemenin Mimarisi 36
5.2 Bir Hücresel İşlemcinin İç Yapısı 37
5.3 Görüntünün Saklanması 42
5.3.1 Görüntünün dilimlere ayrılması 43
5.3.2 Görüntünün RAM'lere yüklenmesi 44
5.4 Pipe-Line Yöntemiyle Aritmetik Birimin Hızlandırılması 46
6. SONUÇLAR **49**
KAYNAKLAR **53**

KISALTMALAR

ASIC	: Application Specific Integrated Circuit
CNN	: Cellular Neural Networks
CNN-UM	: CNN Universal Machine
FPGA	: Field Programmable Gate Array
HSA	: Hücresel Yapay Sinir Ağları
RAM	: Random Access Memory
ROM	: Read Only Memory
VHDL	: VHSIC Hardware Description Language
VLSI	: Very Large Scale Integration

ÇİZELGE LİSTESİ

Sayfa

Çizelge 3.1: Farklı tanımlama seviyelerinin elemanları [12] 15
Çizelge 4.1: Anlatılan tasarımın diğer bazı yongalarla karşılaştırılması [3]. 28
Çizelge 4.2: Analog CNN yongalarıyla CASTLE tasarımının karşılaştırılması [3]. 29
Çizelge 6.1: İşlemci birimlerinin karşılaştırılması. 50

ŞEKİL LİSTESİ

Sayfa

Şekil 2.1: Standart HSA mimarisi. 4
Şekil 2.2: (a)r=1 (3x3 komşuluk) , **(b)**r=2 (5x5 komşuluk), **(c)**r=3 (7x7 komşuluk).. 5
Şekil 2.3: Bir HSA yapısındaki hücreler arası etkileşim. 5
Şekil 2.4: Bir hücrenin blok diyagram gösterilimi. 6
Şekil 2.5: Eşik Aktivasyon Fonksiyonu..... 7
Şekil 2.6: Bir Hücrenin devre şeması. [11]..... 7
Şekil 2.7: "Cameraman" resmi ve bazı şablonlar uygulandıktan sonraki çıkışı..... 12
Şekil 4.1: Dijital hücresel sinir ağında işlemci üniteleri ve global kontrol bloğu. 21
Şekil 4.2: Bir işlemci ünitenin giriş ve çıkış işaretleri..... 21
Şekil 4.3: Görüntünün işlemci içindeki kaydedici dizilerinde saklanışı..... 22
Şekil 4.4: Resim diliminin kaydedici dizilerine yüklenişi ve konvolüsyon yönü 23
Şekil 4.5: Bir işlemci biriminin sistem seviyesinde blok gösterimi. 24
Şekil 4.6: Kaydedici transfer seviyesinde kaydedici dizisi tasarımı..... 25
Şekil 4.7: Aritmetik işlem yapan birimin iç yapısı. 27
Şekil 4.8: Pipe-line tekniği ile yapılan aritmetik blok ve çalışma algoritması. 31
Şekil 4.9: Tasarımda kullanılan (CASTLE) 12-bitlik pipe-line yapılmış toplayıcı... 32
Şekil 4.10: Tasarımda kullanılan pipe-line yapılmış 6-bit Baugh-Wooley çarpıcısı. 32
Şekil 4.11: "k" seviye pipe-line yapılmış sisteme uygun veri okuma sırası. 33
Şekil 5.1: Dijital Hücresel Sinir Ağının dışarıya açılan uçları 36
Şekil 5.2: Tasarlanan devrenin iç genel mimarisi..... 37
Şekil 5.3: Hücresel işlemcinin iç yapısı..... 38
Şekil 5.4: Aritmetik işlem yapan birime ilişkin iç yapı. 41
Şekil 5.5: Görüntünün dilimlere ayrılması. 43
Şekil 5.6: Görüntülerin RAM'e yerleştirilmesi. 44
Şekil 5.7: Pipe-line ile hızlandırılmış devre 46
Şekil 5.8: Pipe-line yapılmış devrede veri okuma yöntemi..... 47

VHDL İLE HÜCRESEL YAPAY SİNİR AĞI GERÇEKLEMESİ

ÖZET

Görüntü işleme, biyomedikal uygulamalardan uydu görüntüsü tanımaya kadar birçok uygulamada ya ana unsur olarak etkin bir şekilde veya işin belirli bir sürecinin tamamlanmasında yan unsur olarak kullanılır. Ardışıl işlem yapan klasik işlemciler üzerinde işlenen görüntü veya video uygulamaları için gereken toplam işlem zamanı görüntü boyutuna direk bağlıdır. Bu, zamanın kritik öneme sahip olduğu uygulamalarda alternatif görüntü işleyen yapılara yönelmeyi gerektirmektedir. Hücresel Sinir Ağı (HSA), yapısı gereği görüntü işlemeye müsaittir. Günümüzde birçok analog HSA gerçeklemesi (ACE4K, ACE16K, vb) vardır. Analog kırmık üzerinde gerçeklenen HSA'da analog tasarımın getirdiği sorunlar ciddiyet arz etmektedir. HSA'nın görüntü işlemedeki performansından yararlanıp analog tasarımın güçlüklerinden uzak durmak için bu yapının dijital olarak tasarlanması gündeme gelmiştir. Dijital tasarım, analog HSA'dan yavaş olmasına rağmen gerçek zamanlı görüntüleri işleyebilecek kadar hızlıdır. Donanım tasarlama dillerinin sağladıkları kolaylıklarla birlikte dijital HSA tasarımı oldukça değiştirilebilir olup değişik ihtiyaçlara cevap verebilecek şekilde yeniden düzenlenebilir. Sağladığı bu faydalarla, dijital HSA tasarımı görüntü işleme problemlerine oldukça etkili çözümler sunmaktadır.

Bu tezin ilk bölümünde yapılan çalışma genel hatlarıyla anlatılmıştır. İkinci bölümde ise HSA yapısından genel olarak bahsedilmiştir. Üçüncü bölümde bir donanım tanımlama dili olan VHDL ile tasarım yöntemleri hakkında ön bilgi verilmiştir. Dördüncü bölümde literatürde gerçeklenen dijital HSA tasarımları tanıtılmıştır. Bu bölümün ilerleyen kısımlarında ise literatürde var olan hızlandırma metotlarından bahsedilmiştir. Beşinci bölümde VHDL ile yapılan yeni bir dijital HSA gerçeklemesi sunulmuştur. Gerçeklenen HSA literatürdeki gerçeklemelerle karşılaştırılmıştır. Son bölümde sonuçlar yorumlanmış ve geliştirmeler için bazı önerilerde bulunulmuştur.

IMPLEMENTATION OF CELLULAR NEURAL NETWORK WITH VHDL

SUMMARY

Image processing is used as a main process or part of a process in a wide range from bioinformatics to recognition of satellite messages. The process time of an image or vision processing task is highly dependent on the size of image when a classical sequential command executing processor is used. This leads us to alternative approaches in time critical applications. Cellular Neural Network (CNN), due to its nodal structure, is suitable to image processing. Analog implementation of CNN, e.g. ACE4K, ACE16K is available for research purposes. Problems of analog design have serious restrictions on the design of analog CNN chip. In order to benefit from image processing performance of CNN while avoiding problems of analog design, digital implementation of CNN has become a current issue. Digital emulation of CNN is slower than the analog one, but still fast enough for real time image processing. With the advantages of hardware description languages (HDL), reconfigurable designs of CNNs can be easily implemented for different purposes. By these eases, digitally emulated CNN presents effective solutions to image processing tasks.

In chapter one, the overview of this thesis is presented. In chapter two, detailed information about CNN is given. In chapter three, digital design methods using hardware description languages (VHDL) are mentioned. In chapter four, previous designs of digital CNN are covered. In following passages of chapter four, mentioned are acceleration methods. In chapter five, the proposed method implemented by VHDL is presented. Finally, some concluding remarks are stated on the performance of new design by comparing previous designs and some suggestions are given for better performance.

1. GİRİŞ

Görüntü işleme, birçok mühendislik uygulamasında kullanılmaktadır. Görüntü boyutu ve görüntü çözünürlüğü arttıkça görüntüyü bilinen yöntemlerle işlemek hayli zaman almaktadır. Görüntüyü analog olarak işleyen ve sadece zaman sabiti kadar gecikmenin ardından sonuç veren bir mimariye sahip olan Hücresel Sinir Ağı (HSA) (Cellular Neural Network), 1988'de ortaya atılmıştır [1]. HSA'yı merkezi işlem birimi olarak alan ve bu birimi yerel ve evrensel (global) bellek birimleri, program belleği gibi çevre birimleriyle birlikte bir bilgisayara dönüştüren CNN-UM [2] 1993 yılında Roska ve Chua tarafından önerilmiştir. Bu yapıya analog ve logic kelimelerinin birleşmesinden oluşan "analogic computer" adı verilmiştir. Bu yapıda işlem zamanı görüntü boyutundan bağımsızdır. Ancak bu yapı bahsedildiği gibi analog işlemcilerin çalıştığı bir yapıdır. CNN-UM'yi - bilhassa çok hücre olduğu durumlarda - tek yonga üzerinde gerçeklemek hayli zor olduğu için HSA'yı gerçekleyen alternatif yollar üzerinde durulmuştur [3-6].

Analog tasarımın zorluğuna karşın dijital tasarımın gürültüye olan bağışıklılığı ve eleman parametrelerine toleransı gibi avantajları göz önüne alınırsa, HSA'yı gerçekleyen dijital yapılar hem görüntü işleme hızı açısından hem de tasarım kolaylığı açısından dikkate değerdir. Tasarlanan bir hücresel işlemci kolaylıkla paralel çalıştırılabilir. Bu hücresel işlemcilerin kontrolü birbirinin aynıdır. Ayrıca hücresel işlemcilerdeki blokların tasarımı olabildiğince değiştirilebilirdir. İşlemci bloklarının içindeki toplayıcı, çarpıcı ve karşılaştırıcı gibi yapılar sistemin veri çözünürlüğüne göre analog tasarımla kıyas edilemeyecek bir hızda yeniden tasarlanabilir. Tasarım kolaylığı ve tasarım esnekliği değerlendirilince HSA'yı dijital olarak gerçeklemek önemli bir konu olarak karşımıza çıkmaktadır.

Dijital tasarım yapılırken tasarımın ne ile yapılacağı sorusu da hayli önem arz etmektedir. Günümüzde HDL (Hardware Description Language) ile dijital tasarım yapmak hayli kolaylaşmıştır. Bu dillerin sağladığı yüksek seviyeli tasarım imkânlarıyla tasarıma olabildiğince tepeden bakıp, detayları bu dillere bırakarak ve

daha az zaman harcayarak hem hız hem de işgal edilen yonga alanı açısından verimli devreler üretmek mümkündür. Bu dillerle yapılan tasarımın bazı avantajları şunlardır: yeniden kullanılabilirliktir, taşınabilirlik, tasarım esnekliği. Tasarlanmış bir toplayıcı bloğu çok özel bir uygulamada kullanılabileceği gibi toplayıcı gerektiren her hangi bir uygulama için de kullanılabilir (yeniden kullanılabilirlik). Bu dillerin standartları olduğu için, kurallara göre yazılan bir kod her sistemde geçerlidir (taşınabilirlik). Tasarım esnekliğinde ise, dil olabildiğince geniş imkânlar sağlamaktadır. Örneğin, sentez öncesinde kaydedicilerin büyüklüklerini sembolik sabitlerle değiştirmek oldukça kolaydır. Bütün bunlara rağmen, tasarım yapılan dili iyi bilmek gerekmektedir. Aynı tasarımın farklı kodlayıcılar tarafından kodlanması farklı büyüklükteki devrelerin oluşmasına sebep olabilir. Dili iyi bilmek ve yazılan kodun devre karşılığını en azından sezinlemek devreyi daha verimli bir şekilde tasarlamayı kolaylaştıracaktır.

Bu tezde üstte anlatılan konuların ışığında HSA'nın dijital gerçeklemesi anlatılmaktadır. Dijital gerçeklemede literatürdeki birçok devre incelenmiş ve hepsinden bazı yönler seçilmiştir. Sonuç olarak; devrenin hızını azaltmadan, daha az kaynak kullanan hücresel işlemcilerle, tam paralel çalışan bir devre tasarımı yapılmıştır. Tasarımda bir donanım tasarlama dili olan VHDL kullanılmıştır.

2. HÜCRESEL SİNİR AĞLARI

2.1 Tarihsel Gelişim

Sinir sistemimizdeki nöronların birbirine olan bağlantılarına yoğunlaşan iki bilim adamı Leon O. Chua ve Lin Yang 1988 yılında yeni bir yapay sinir ağı mimarisi ortaya koydukları makalelerini [1,7] yayınladılar ve bu makaleler ile Hücresel Sinir Ağı (HSA) doğmuş oldu.

Sinir sistemimizdeki nöronların diğer nöronlarla bölgesel olarak bağlı olması HSA yapısının dayandığı en önemli özelliktir. Hopfield sinir ağlarındaki [8] bütün hücrelerin birbirleriyle doğrudan bağlılığına karşın HSA'da her bir hücre en yakın komşuluğundaki hücre ile doğrudan, diğer hücrelerle dolaylı olarak bağlıdır.

Bölgesel bağlantı sonucu; hem devrenin kapladığı alan azalmakta hem de verilerin devre üzerindeki taşınım yolları kısa olduğundan toplam güç harcaması düşük olmaktadır. HSA'nın önemli avantajlarından bir diğeri de kararlı bir sonuca yakınsama süresinin devrenin boyutundan bağımsız olması ve bir kaç nöron gecikmesi ile belirlenmesidir.

Prof. Roska ve Prof. Chua tarafından HSA'yı merkezi işlem birimi alan yeni bir bilgisayar mimarisi ortaya atıldı [2]. Bu mimari aslında analog ve sayısal işlem yapma yeteneğine sahip bir merkezi işlem birimine sahiptir. Bu nedenle HSA-UM (CNN-UM) olarak adlandırılan işlemci, analog ve lojik işlem yeteneği nedeniyle bu iki terimin birleşimi olan analogic işlemci olarak nitelendirilmektedir. Mimarisi analog ve sayısal olmak üzere yerel ve evrensel (global) belleğe ayrıca HSA'yı programlamaya yarayan şablonların tutulduğu bellek alanlarına sahiptir.

HSA'nın bu özellikleri gelecekte pek çok uygulama alanı bulacağının göstergeleridir. HSA günümüzde retina modellemesi, biyonik göz gibi birçok uygulamaya sahiptir.

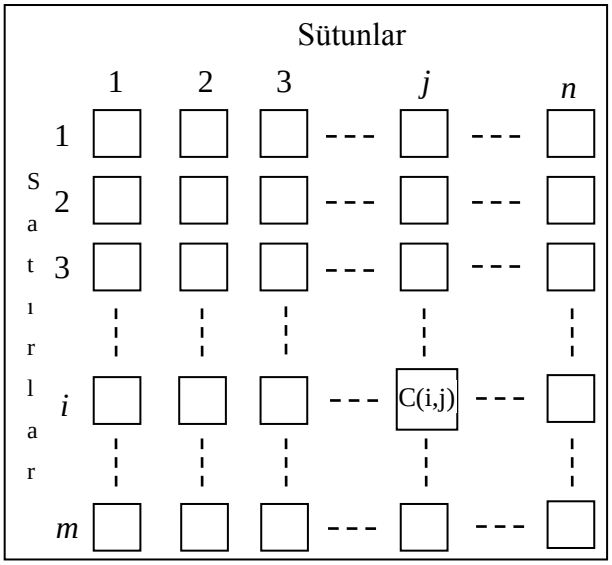
HSA'nın görüntü işlemedeki başlıca uygulamaları ise karakter tanıma, görüntü bölütleme, görüntü onarımı olarak sayılabilir [9-10].

2.2 Hücresel Sinir Ağlarının Mimarisi

Bu kısımda hücresel sinir ağı mimarisi, ayrıntılı olarak incelenmiştir.

Tanım 2.1. Standart HSA Mimarisi

Standart HSA yapısı *m* x *n* boyutundaki dikdörtgen hücreler dizisinden oluşur. Burada *m*: satır sayısını *n*: sütun sayısını gösterir. HSA'nın en küçük birimine hücre adı verilir. Hücreler devrenin boyutuna göre iki boyutlu uzayda kartezyen koordinat sistemi düzeninde yerleştirilmişlerdir. *C(i,j), i.* satır, *j.* sütun'daki hücre olarak ifade edilir. (*i=1,2,3...m, j=1,2,3...n*) (Şekil 2.1)



Şekil 2.1: Standart HSA mimarisi.

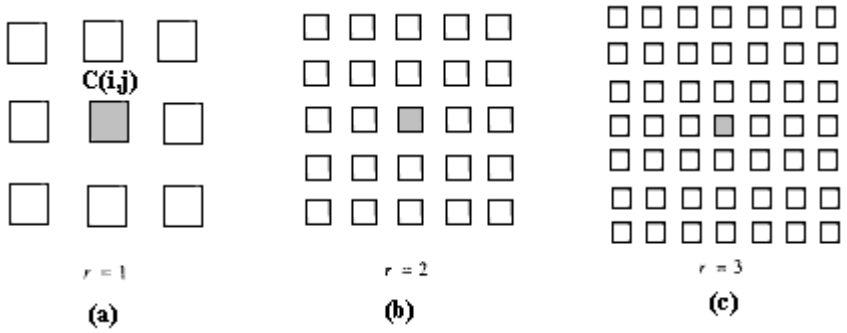
Uygulamalarda genellikle $m = n$ alınsa da $m \neq n$ de olabilir. Örneğin *5 x 512*'lik bir HSA yapısı tarayıcı, faks ya da fotokopi makinesi için uygundur.

Tanım 2.2. C*(i,j)* Hücresinin Etki Küresi [10]

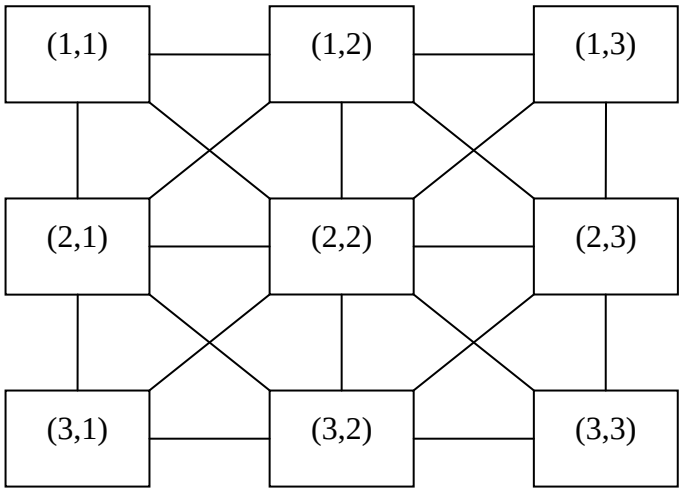
C(i,j) hücresinin r yarıçaplı etki küresi $S_r(i, j)$ olmak üzere;

$$S_r(i, j) = \left\{ C(k,l) \,\middle|\, \max_{1 \le k \le m, 1 \le l \le n} \{|k - i|, |l - j|\} \le r \right\} \qquad \textbf{(2.1)}$$

koşulunu sağlayan tüm komşu hücreler topluluğuna denir. Buradaki r pozitif bir tam sayıdır.



Şekil 2.2: (a)r=1 (3x3 komşuluk) , **(b)**r=2 (5x5 komşuluk), **(c)**r=3 (7x7 komşuluk)



Şekil 2.3: Bir HSA yapısındaki hücreler arası etkileşim.

Komşuluk *r* yarıçapı ile belirtilebildiği gibi *(2r+1)*x*(2r+1)* şeklinde de gösterilebilir. Etki küresi içerisinde $(2r+1)^2$ kadar hücre bulunmaktadır.

Komşuluk sistemi simetri özelliğine sahiptir.

$$C(k, l) \in S_r(i, j) \Leftrightarrow C(i, j) \in S_r(k, l) \qquad \textbf{(2.2)}$$

Hücresel Sinir Ağı tanımlanırken, bir r değeri seçilir (Çoğu uygulamada r =1 alınır). Bu seçimden sonra her hücre sadece kendi komşuluk grubundaki hücrelerle doğrudan bağlantılı bulunur. Yani sadece komşuları ile sinaptik bağlantıları mevcuttur.

Tanım 2.3. Bir Hücrenin Matematiksel Tanımı

Chua-Yang olarak adlandırılan bir HSA'da *C(i,j)* hücresine ilişkin matematiksel model şu şekildedir;

Tanım 2.4. Durum Denklemi [10]

$$\frac{dx_{ij}}{dt} = -x_{ij} + \sum_{C(k,l)\in S_r(i,j)} A(i,j;k,l)y_{kl} + \sum_{C(k,l)\in S_r(i,j)} B(i,j;k,l)u_{kl} + z_{ij} \qquad \textbf{(2.3)}$$

Burada;

$x_{ij} \in R$; *C(i,j)* Hücresinin durum değişkeni,

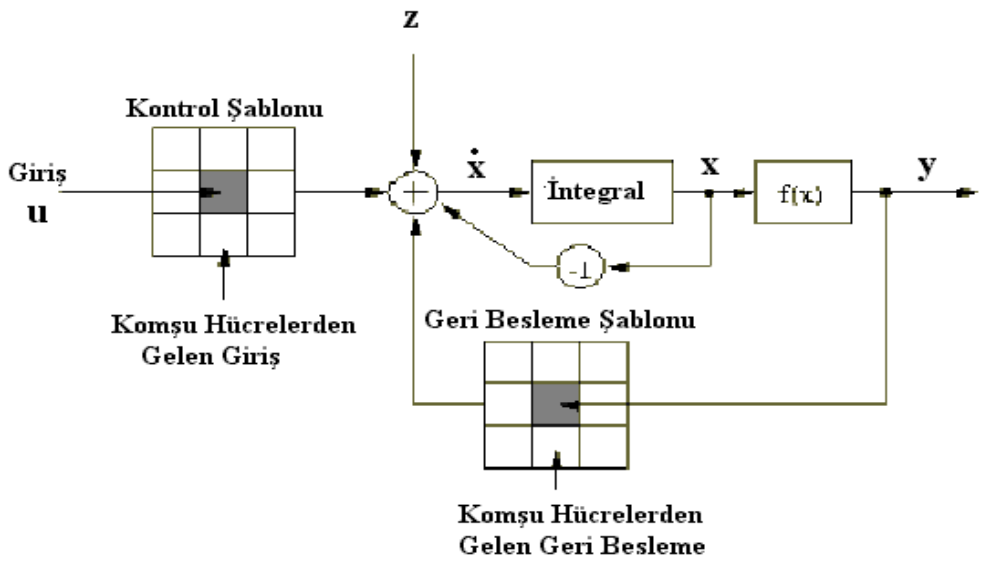
$y_{kl} \in R$; Etki küresi içerisindeki hücrelerin çıkışları,

$u_{kl} \in R$; Etki küresi içerisindeki hücrelerin girişleri,

$z_{ij} \in R$; Eşik değeri

A(i, j ; k, l) ; Geri Besleme operatörü

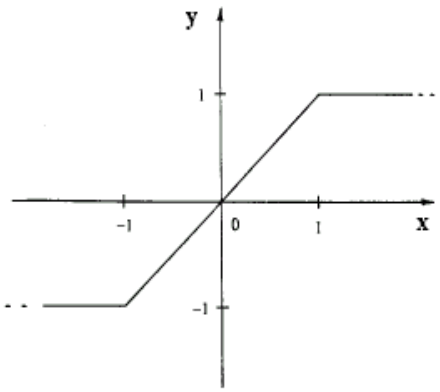
B(i, j ; k, l) ; Giriş (Kontrol) operatörüdür ve bu modele ilişkin blok diyagram Şekil 2.4'te verilmiştir. HSA ya ilişkin doğrusal olmayan fonksiyon *f(x,y)* Şekil 2.5'te gösterilmiştir.



Şekil 2.4: Bir hücrenin blok diyagram gösterilimi.

Tanım 2.5. Çıkış Denklemi

$$y_{ij} = f(x_{ij}) = \frac{1}{2}|x_{ij}+1| - \frac{1}{2}|x_{ij}-1| \qquad \textbf{(2.4)}$$



Şekil 2.5: Eşik Aktivasyon Fonksiyonu.

HSA'ya lineer olmama özelliğini veren bu fonksiyondur.

Tanım 2.6. Sınır Koşulları

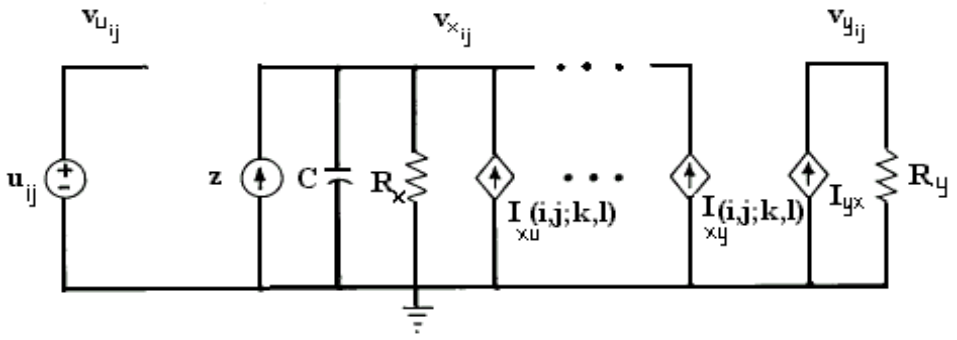
Sınır hücrelerinin etki küresi içinde bulunup da *m x n* boyutundaki hücre dizisinin dışında kalan hücrelerin durum ve çıkış değerleri koşuludur.

Tanım 2.7. Başlangıç Koşulları

Dizideki tüm hücrelerin ilk koşuludur. $x_{ij}(0)$, i=1,2...m , j=1,2...n

2.3 Hücre Yapısı

HSA'nın bir hücresinin devre modeli Şekil 2.6 da görülmektedir.



Şekil 2.6: Bir Hücrenin devre şeması. [11]

Hücrede sırasıyla giriş, durum ve çıkışa karşı düşen ***u, x*** ve ***y*** düğümleri bulunur. Hücrede bulunan elemanlar u_{ij}; sabit gerilim kaynağı, ***Z***; sabit akım kaynağı, ***C***; lineer kapasite, R_x ve R_y ; lineer dirençler, m komşu hücre sayısına eşit olmak üzere en fazla 2m adet $I_{xu}(i,j;k,l)$ ve $I_{xy}(i,j;k,l)$ bağımlı akım kaynağı, son olarak da I_{yx} lineer olmayan gerilim kontrollü akım kaynağıdır. Komşu hücrelerle bağlantı, kontrol girişi $v_{u_{kl}}$ ve geri besleme gerilimi $v_{y_{kl}}$ 'nin ağırlıklı toplamları üzerine kurulmuştur.

$$I_{xy}(i,j;k,l) = A(i,j;k,l)v_{y_{kl}} \quad , \quad \forall\, C(k,l) \in S_r(i,j) \tag{2.5}$$

$$I_{xu}(i,j;k,l) = B(i,j;k,l)v_{u_{kl}} \quad , \quad \forall\, C(k,l) \in S_r(i,j) \tag{2.6}$$

Hücredeki tek lineer olmayan eleman ise I_{yx} parça-lineer gerilim kontrollü akım kaynağıdır.

$$I_{yx} = \frac{1}{R_y} f(v_{x_{ij}}) \tag{2.7}$$

Buradaki *f(.)* fonksiyonu Tanım 2.6'da verilen çıkış fonksiyonudur.

HSA'nın VLSI gerçeklemelerinde *f(.)* fonksiyonunun keskin karakteristiğini elde etmek mümkün olmadığından daha yumuşak ve sürekli bir fonksiyon olan Sigmoid fonksiyonu kullanılır.

2.4 Zamandan ve Konumdan Bağımsız HSA

Eğer geri besleme A(i, j ; k, l), giriş B(i, j ; k, l) operatörleri ve z_{ij} eşik değeri, hücrenin ağ içerisindeki yerleşiminden ve zamandan bağımsız olarak belirleniyorsa bu yapıya 'Zamandan ve Konumdan Bağımsız HSA' adı verilir. Uygulamaların neredeyse tamamı bu yapıyı kullanır. Bu durumda aşağıdaki ifadeleri yazabiliriz;

$$\sum_{C(k,l)\in S_r(i,j)} A(i,j;k,l)y_{kl} = \sum_{|k-i|\le r} \sum_{|l-j|\le r} A(i-k, j-l)y_{kl} \tag{2.8}$$

$$\sum_{C(k,l)\in S_r(i,j)} B(i,j;k,l)u_{kl} = \sum_{|k-i|\le r} \sum_{|l-j|\le r} B(i-k, j-l)u_{kl} \tag{2.9}$$

$$z_{ij} = z \tag{2.10}$$

2.4.1 Klonlama şablonu gösterilimi

HSA'nın uygulamalarında en fazla *3x3* komşuluğun (etki küresi yarıçapı r =1) bulunduğu, konumdan bağımsız yapı kullanılır. Bu uygulamalarda analizini kolaylaştıracak bazı tanımlamalar yapılmıştır.

Geri besleme (*A(i-k , j-l)*), giriş (*B(i-k , j-l)*) operatörleri ve z_{ij} eşik değeri, hücrenin ağ içerisindeki yerleşiminden bağımsız olduklarından, bir şablon şeklinde gösterilebilirler.

2.4.1.1 Geri besleme operatöründen (A(i-k , j-l)) gelen katkı

$$\sum_{C(k,l)\in S_r(i,j)} A(i,j;k,l)y_{kl} = \sum_{|k-i|\le r} \sum_{|l-j|\le r} A(i-k,j-l)y_{kl}$$

$$\begin{aligned} &= a_{-1,-1}y_{i-1,j-1} + a_{-1,0}y_{i-1,j} + a_{-1,1}y_{i-1,j+1} \\ &\quad + a_{0,-1}y_{i,j-1} + a_{0,0}y_{i,j} + a_{0,1}y_{i-1,j+1} \\ &\quad + a_{1,-1}y_{i+1,j-1} + a_{1,0}y_{i+1,j} + a_{1,1}y_{i+1,j+1} \\ &= \sum_{k=-1}^{1}\sum_{l=-1}^{1} a_{k,l}y_{i+k,j+l} \end{aligned} \qquad \textbf{(2.11)}$$

$$= \begin{array}{|c|c|c|}\hline a_{-1,-1} & a_{-1,0} & a_{-1,1} \\ \hline a_{0,-1} & a_{0,0} & a_{0,1} \\ \hline a_{1,-1} & a_{1,0} & a_{1,1} \\ \hline \end{array} \otimes \begin{array}{|c|c|c|}\hline y_{i-1,j-1} & y_{i-1,j} & y_{i-1,j+1} \\ \hline y_{i,j-1} & y_{i,j} & y_{i-1,j+1} \\ \hline y_{i+1,j-1} & y_{i+1,j} & y_{i+1,j+1} \\ \hline \end{array} = A \otimes Y_{ij}$$

$$A = \begin{array}{|c|c|c|}\hline a_{-1,-1} & a_{-1,0} & a_{-1,1} \\ \hline a_{0,-1} & a_{0,0} & a_{0,1} \\ \hline a_{1,-1} & a_{1,0} & a_{1,1} \\ \hline \end{array} \qquad \textbf{(2.12)}$$

$$Y_{ij} = \begin{array}{|c|c|c|}\hline y_{i-1,j-1} & y_{i-1,j} & y_{i-1,j+1} \\ \hline y_{i,j-1} & y_{i,j} & y_{i-1,j+1} \\ \hline y_{i+1,j-1} & y_{i+1,j} & y_{i+1,j+1} \\ \hline \end{array} \qquad \textbf{(2.13)}$$

Burada *3 x 3* boyutundaki ***A*** matrisi **'Geri Besleme Klonlama Şablonu'** olarak isimlendirilir.

$\boldsymbol{Y_{ij}}$ matrisi *m x n* boyutundaki çıkış görüntüsünün üzerinde *3*x*3* bir pencere gezdirilerek elde edilebileceğinden **'*C(i,j)* Hücresindeki Çıkış Görüntüsü'** olarak adlandırılır.

$\otimes$ sembolü nokta çarpımların toplamını ifade eder ve **'Şablon Nokta Çarpım'** olarak isimlendirilir. Nokta çarpım matrisel bir çarpım olmayıp; skaler çarpımla, matrislerin biribirleriyle aynı indisteki elamanlarının çarpılmasıyla yapılır. Ayrık matematikte bu işlem uzaysal konvolüsyona karşı düşer.

Bazı uygulamalarda **A** şablonu şu şekilde ikiye ayrılabilmektedir:

a_{00}*; *A şablonunun merkez elemanını göstermek üzere (a_{kl} ; $k=l=0$);

$$A = A^0 + \bar{A} \quad , \quad A^0 = \begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & a_{00} & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array} \quad , \quad \bar{A} = \begin{array}{|c|c|c|}\hline a_{-1,-1} & a_{-1,0} & a_{-1,1} \\ \hline a_{0,-1} & 0 & a_{0,1} \\ \hline a_{1,-1} & a_{1,0} & a_{1,1} \\ \hline \end{array} \qquad \textbf{(2.14)}$$

A^0 ; Şablonun Merkez Bileşeni,

$\bar{A}$; Şablonun Çevre Bileşeni olarak adlandırılır.

2.4.1.2 Giriş (kontrol) operatöründen (B(i-k, j-l)) gelen katkı

Bir önceki bölümde yapılan işlemlere benzer işlemlerle;

$$\sum_{C(k,l)\in S_r(i,j)} B(i,\, j;k,l)u_{kl} = \sum_{|k-i|\le r} \sum_{|l-j|\le r} B(i-k,\, j-l)u_{kl}$$

$$\begin{aligned} &= b_{-1,-1}u_{i-1,j-1} + b_{-1,0}u_{i-1,j} + b_{-1,1}u_{i-1,j+1} \\ &\quad + b_{0,-1}u_{i,j-1} + b_{0,0}u_{i,j} + b_{0,1}u_{i-1,j+1} \\ &\quad + b_{1,-1}u_{i+1,j-1} + b_{1,0}u_{i+1,j} + b_{1,1}u_{i+1,j+1} \\ &= \sum_{k=-1}^{1} \sum_{l=-1}^{1} b_{k,l}u_{i+k,j+l} \end{aligned} \qquad \textbf{(2.15)}$$

$$= \begin{array}{|c|c|c|}\hline b_{-1,-1} & b_{-1,0} & b_{-1,1} \\ \hline b_{0,-1} & b_{0,0} & b_{0,1} \\ \hline b_{1,-1} & b_{1,0} & b_{1,1} \\ \hline \end{array} \otimes \begin{array}{|c|c|c|}\hline u_{i-1,j-1} & u_{i-1,j} & u_{i-1,j+1} \\ \hline u_{i,j-1} & u_{i,j} & u_{i-1,j+1} \\ \hline u_{i+1,j-1} & u_{i+1,j} & u_{i+1,j+1} \\ \hline \end{array} = B \otimes U_{ij}$$

$$B = \begin{bmatrix} b_{-1,-1} & b_{-1,0} & b_{-1,1} \\ b_{0,-1} & b_{0,0} & b_{0,1} \\ b_{1,-1} & b_{1,0} & b_{1,1} \end{bmatrix} \quad \textbf{(2.16)}$$

$$U_{ij} = \begin{bmatrix} u_{i-1,j-1} & u_{i-1,j} & u_{i-1,j+1} \\ u_{i,j-1} & u_{i,j} & u_{i-1,j+1} \\ u_{i+1,j-1} & u_{i+1,j} & u_{i+1,j+1} \end{bmatrix} \quad \textbf{(2.17)}$$

*3*x*3* boyutundaki ***B*** matrisi **'Giriş (Kontrol) Klonlama Şablonu'** olarak isimlendirilir.

$\boldsymbol{U_{ij}}$ matrisi *m* x *n* boyutundaki giriş görüntüsünün üzerinde *3x3* bir pencere gezdirilerek elde edilebileceğinden '***C(i,j)* Hücresindeki Giriş Görüntüsü'** olarak adlandırılır.

Bazı uygulamalarda **B** şablonu şu şekilde ikiye ayrılabilmektedir.

$\boldsymbol{b_{00}}$**;** ***B*** şablonunun merkez elemanını göstermek üzere. (b_{kl} *; k=l=0*)

$$B = B^0 + \bar{B} \quad , \quad B^0 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & b_{00} & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad , \quad \bar{B} = \begin{bmatrix} b_{-1,-1} & b_{-1,0} & b_{-1,1} \\ b_{0,-1} & 0 & b_{0,1} \\ b_{1,-1} & b_{1,0} & b_{1,1} \end{bmatrix} \quad \textbf{(2.18)}$$

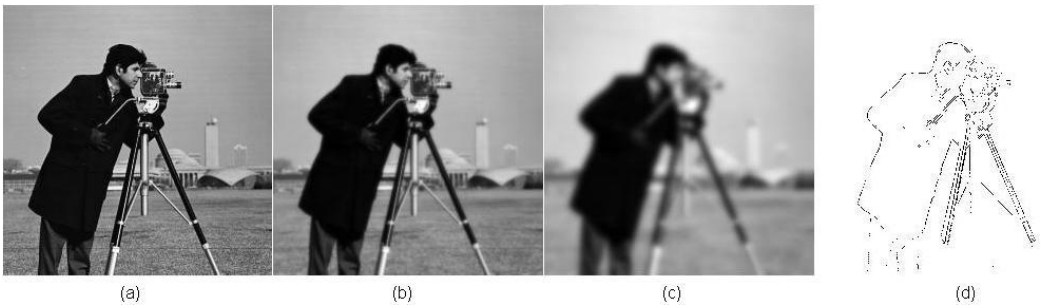
B^0**;** Şablonun Merkez Bileşeni

$\bar{B}$**;** Şablonun Çevre Bileşeni olarak adlandırılır.

2.5 HSA Benzetim Programı

Yapılan uygulamanın doğruluğunu görmek amacıyla HSA yapısının çalışmasını modelleyen bir benzetim programı C++ dilinde yazılmıştır. Benzetim programında C++ dilinin yüksek seviyeli araçları kullanılarak görüntü satır ve sütun uzunluğu değiştirilebilir olarak alınmıştır. Ayrıca şablon ve piksel değerlerinin ifade edildiği tür C++ dilinin doğal türlerinden biri olabilecek şekilde seçilebilir yapılmıştır.

Bu program, HSA yapısıyla kıyas edilemeyecek kadar yavaş çalışmaktadır. Çünkü programın çalıştığı işlemci ardışıl komut işleyen bir yapı olduğu için, her bir işlemi ayrı çevrimlerde yapmaktadır. Fakat programlamadaki kolaylık ve esneklik amacıyla elde edilen sonuçları teyit edebilmek için bu program yazılmıştır. Bu program yazılırken kullanılan matematiksel model 4. bölümde detaylı olarak anlatılacaktır. Programa verilen "cameraman" görüntüsünün üç farklı şablonla filtre edilip çıkışları yorumlanmıştır. İlgili şablonlarla yapılan filtrelemeler, beklenilen çıkışları vermiştir. Benzetim programında kullanılan yöntem çok az farkla VHDL ile gerçeklenen devrede kullanılmıştır. VHDL ile gerçeklenen devrenin veri hassasiyeti C++ benzetim programına göre daha azdır. Benzetim programının çıktısı Şekil 2.7'de gösterilmiştir.



Şekil 2.7: "Cameraman" resmi ve bazı şablonlar uygulandıktan sonraki çıkışı.

Şekil 2.7'de ilk görüntü "cameraman" resminin kendisidir, diğerleri filtrelenmiş çıkış görüntüleridir. Bütün resimlere aynı z değeri uygulanmıştır. Uygulanan *A, B* şablonları ve *z* değeri aşağıdaki gibidir:

$$A=\begin{bmatrix} 0 & \alpha & 0 \\ \alpha & -4\alpha^2 & \alpha \\ 0 & \alpha & 0 \end{bmatrix} \quad B=\begin{bmatrix} 0 & 0 & 0 \\ 0 & (1-2\alpha)^2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad z=0$$

Şekil 2.7'deki (b), (c), (d) çıkışlarını veren şablonlar sırayla $\alpha = 0.25$, $\alpha = 0.45$, $\alpha = -0.25$ değerlerini almışlardır.

3. VHDL İLE DİJİTAL TASARIM

Elektronik sistemlerin karmaşıklığının artması tasarım yöntemlerinin de gelişmesini gerektirmiştir. Bu sebeple, geleneksel kâğıt ve kalem kullanarak yapılan tasarımların yerini “tanımla ve sentezle” yöntemleri almıştır. Bu bölümde, sayısal tasarım, genel olarak ele alınacaktır. Daha sonra da, tasarım adımları anlatılacaktır. Donanım Tanımlama Dilleri'nin (Hardware Description Languages: HDLs) "tanımla ve sentezle" yönteminde önemli bir rolü vardır. HDLs bir elektronik sistemin tanımlanmasında, test edilmesinde ve sentezlenmesinde kullanılırlar. Pek çok donanım tanımlama dillerinin arasında VHDL en yaygın kullanılanıdır.

3.1 Dijital Tasarımın Safhaları

Bir sayısal sistemin tasarımı, genel olarak aşağıda ifade edildiği gibi dört aşamada yapılır:

- Sistemden istenenlerin belirlenmesi ve tanımlama aşaması
- Tasarım aşaması
- Gerçekleme ve test etme aşaması
- Üretim aşaması

Her tasarımın ilk adımı, sistemden istenenlerin belirlenmesi ve sistemin tanımlanması oluşturur. Bundan sonraki aşama, tasarım aşamasıdır. Tasarıma yukarıdan aşağıya yaklaşımda, sayısal sistem daha az karmaşık alt sistemlere ayrılır ve bu alt sistemler gerçekleştirilmeye çalışılır. Yukarıdan aşağıya doğru, tasarım şu adımları gerektirir: Sistem tasarımı, yapısal tasarım, kaydedici transfer seviyesinde (Register Transfer Level: RTL) tasarım ve lojik seviyede tasarım. Sistem tasarımı seviyesinde, tasarımcı sistemi alt sistemlere ayırır ve alt sistemler arasındaki haberleşmeyi tanımlar. Yapısal tasarımda, yapı özellikleri ve her alt sistemin performansı tanımlanır. Saklayıcı iletişim seviyesindeki tasarımda, yapı birbirine bağlı modüllere ayrılır. Son olarak da, modüller, lojik kapılar kullanılarak tanımlanır.

Gerçekleme aşamasında alt sistemler gerçeklenir ve test edilir; bunun için parçalama ve yerleştirme işlemleri yapılır. Alt sistemler de bütün sistemi oluşturacak şekilde birleştirilir ve tüm sistem test edilir.

Sayısal tasarımın son aşamasında sistemin prototipi oluşturulup üretilir ve ortaya çıkan tümleşik devre test edilir [12].

3.2 Tanımlama Seviyeleri

Genel olarak problem belirlendikten sonra sistem, birbirine bağlantıları tamamıyla belirli alt parçalara ayrılmalıdır. Bu yapılırken, parçaların, herbiri bir birim olarak düşünülmeli ve birimlerin arasındaki bağlantılar tamamen belirlenmelidir. Parçalama işlemine, basit lojik kapılara ulaşıncaya kadar devam edilir.

Tanımlama genel olarak şu seviyelere ayrılır: Sistem seviyesinde, RTL'de, lojik seviyede ve devre seviyesinde. Bu seviyeler tanımlanırken üç farklı tanımlama yöntemi kullanılır: Davranışsal, yapısal ve fiziksel. Davranışsal tanımlamada, sistemin nasıl gerçekleneceği göz önüne alınmaksızın, sistem fonksiyonu tanımlanır. Yapısal tanımlama kullanılarak sistem daha önce tanımlanmış elemanların birleşimi olarak tanımlanır. Fiziksel tanımlamada tümleşik devrenin silikona geçirilişi tanımlanır.

Tasarım şekli ve devre karmaşıklığına göre tasarımcı bu tanımlama seviyelerinden herhangi birini veya hepsini kullanabilir. Tablo 3.1'de her tanımlama seviyesi için temel elemanlar gösterilmiştir [12].

Çizelge 3.1: Farklı tanımlama seviyelerinin elemanları [12]

Tanımlama Seviyeleri	Davranışsal Tanımlama	Yapısal tanımlama	Fiziksel Tanımlama
Sistem Seviyesi	Algoritmalar Akış grafları	İşlemciler Bellekler	Baskı devre kartları Çipler
RT Seviyesi	Saklayıcı iletişimleri	Saklayıcılar Fonksiyon birimleri Veri toplayıcılar	Çipler Modüller
Lojik Seviye	Boole fonksiyonları	Kapılar Flip-floplar	Çipler Hücreler
Devre Seviyesi	Transfer fonksiyonları	Tranzistörler Bağlantılar	Hücreler Yol parçaları

3.3 Çok Hızlı Tümleşik Devre Donanım Tanımlama Dili

VHDL en çok kullanılan yapısal tanımlama dilidir. HDL'ler sayısal sistemin yapısını veya davranışını tanımlamada kullanılır. HDL'de yapılan devre tanımı kullanılarak, bir benzetim programı ile yazılan tanımlamanın istenilen fonksiyonu sağlayıp sağlamadığı kontrol edilir. Tanımlamanın doğruluğu görüldükten sonra, bir sentezleme programı ile devrenin fiziksel yapısı elde edilir.

HDL tabanlı tasarım yöntemlerinin geleneksel kapı seviyesinde tasarım yöntemlerine göre bir takım üstünlükleri vardır. İlk olarak, tasarımcının daha az zamanda ve devre tasarımı hakkında çok fazla bilgi sahibi olmadan tasarım yapmasını sağlamaktadırlar. İkinci olarak, aynı tanımlama ile devrenin teknolojilerdeki yapısı elde edilebilmektedir. Devrenin hangi teknoloji kullanılarak gerçekleneceği göz önünde bulundurulmaksızın HDL tanımlaması yapılır. Bir sentezleme programından yararlanarak, HDL ile yapılan tanımlama her defasında farklı teknolojilerdeki elemanlardan oluşan devre şemasına çevrilebilir.

VHDL, IEEE (IEEE standard 1076) ve Birleşik Devletler Silahlı Kuvvetleri (United States Military Department) (MIL-STD-454L) tarafından standart bir HDL olarak kabul edilmiştir.

VHDL kullanılarak, sayısal tasarımın algoritmik, RTL ve lojik kapı seviyeleri gibi farklı seviyelerde tanımlanması mümkündür. Yukarıdan aşağıya tasarım yönteminde, sistem yüksek seviyede tanımlanır ve sonra daha alt seviyedeki bloklara ayrılır.

Davranışsal mimari ile bir sistemden beklenen fonksiyon geleneksel programlara benzer şekilde, tanımlanır. Fakat sistemin nasıl gerçekleneceği konusunda ayrıntılı bilgi verilmez.

Veri akışı şeklindeki mimari (RTL) ile kontrol ve veri işaretlerinin devre içindeki akışı tanımlanır. Bu işlem yapılırken, toplayıcılar, karşılaştırıcılar, kod çözücüler gibi daha önceden tanımlanmış kombinezonsal devre fonksiyonları ve basit lojik kapılar kullanılır.

Yapısal mimari ile tasarımın alt sistemlerini oluşturan daha önce tanımlanmış modüllerin birbirleriyle olan bağlantıları tanımlanır [12].

3.4 VHDL ile Benzetim

VHDL'de yazılan bir tasarım tanımı, bir VHDL benzetim programından faydalanılarak test edilir. Tanımlamanın benzetimini yapabilmek için benzetim programına bir takım test girişleri verilir, program daha önceden belirlenen aralıklarla bu test girişlerini tasarımın modeline uygular ve çıkışları üretir. Bu sonuçlar tasarımcı tarafından gözlenerek modelin istenildiği gibi çalışıp çalışmadığına karar verilir.

Benzetim tasarımın her aşamasında kullanılabilir. Tasarımın yüksek seviyelerinde yapılan benzetimde sadece tasarımın fonksiyonel davranışı hakkında bilgi elde edilir. Bu aşamada benzetim çok hızlıdır, fakat benzetim sonuçlarından tasarımın gerçek devre elemanları ile çalışması ve zamanlaması konusunda çok fazla bilgi elde edilemez. Daha düşük tasarım aşamalarına gidildikçe benzetim daha fazla zaman alacaktır, fakat benzetim sonuçlarından tasarımın çalışması ve zamanlaması konusunda daha fazla bilgi elde edilebilir [12].

3.5 Blok RAM ile İlgili Ön Bilgi

Blok-RAM bilinen klasik RAM'den farklı bir işleyişe sahip değildir. FPGA yongalarının üstünde bulunan bu RAM dual-port olmak, veri yolu genişletebilme gibi özelliklere sahiptir. RAM'in dual-port olması aynı hafızaya erişen iki farklı port olduğu manasına gelmektedir. [13]

4. DİJİTAL HÜCRESEL SİNİR AĞI GERÇEKLEMELERİ

CNN Universal Machine (CNNUM) 1993'te bulunmuştur [2]. Bu yapı CNN paradigmasını gerçekleyen analog elektronik bir devredir. Görüntü işlemede çok yüksek hızlara çıktığı bilinen bu devre, ne var ki analog olarak gerçeklemesi zor olan bir yapıdır. Hücresel Sinir Ağının dijital gerçeklenmesi, analog gerçeklemedeki zorluklar ve çözünürlüğün az (6 bit) olmasından dolayı düşünülmüştür. Bu amaçla yapılan devrelerde farklı yöntemler kullanılmıştır [3-6]. Yapılan tasarımlar daha sonra içlerindeki bazı blokların hızlandırılmasıyla [14-15] ya da FPGA ile tasarımın imkânlarından faydalanarak yapılan tasarımda veri uzunluğu, katman sayısının seçilebilmesi gibi özelliklerin eklenmesiyle geliştirilmiştir [16].

Burada üzerinde duracağımız çalışma, P. Keresztes ve diğerleri [3] tarafından 1999 yılında yaptıkları çalışmadır.

4.1 Matematiksel Modelin Yeniden İfadesi

Chua-Yang modeli olarak adlandırılan ve (2.3) eşitliğinde verilen bir HSA hücresinin matematiksel modelini göz önüne alalım. Bu diferansiyel denklemi yaklaşıklık yapıp ileri Euler yöntemini kullanarak ayrıklaştıralım.

$$x_{ij}(n+1) = (1-h)x_{ij}(n) + h\left(\sum_{C(k,l)\in S_r(i,j)} A_{ij,kl} y_{kl}(n) + \sum_{C(k,l)\in S_r(i,j)} B_{ij,kl} u_{kl} + z_{ij} \right) \qquad \textbf{(4.1)}$$

Denklemdeki n değişkeni iterasyon adımını göstermektedir. Bu denklem, gerçeklemede kolaylık sağlanması amacıyla, bazı parametreler elenerek basitleştirilmeye çalışılmıştır. Burada ilk değişiklik olarak Chua-Yang modeli yerine Full Signal Range (FSR) modeli kullanılmıştır [3]. Bahsedilen ikinci modelde HSA'nın çıkışları, durumlara eşit alınmıştır. Durumların değeri 1 veya -1 değerini aştığı zaman kesme uygulayarak değerler 1 ya da -1 alınmıştır. Bu ikinci modele göre ayrıklaştırılmış denklem

$$v_{ij}(n) = (1-h)x_{ij}(n) + h\left(\sum_{C(k,l)\in S_r(i,j)} A_{ij,kl}x_{kl}(n) + \sum_{C(k,l)\in S_r(i,j)} B_{ij,kl}u_{kl} + z_{ij} \right) \quad \textbf{(4.2)}$$

olmak üzere, aşağıdaki gibi ifade edilebilir:

$$x_{ij}(n+1) = \begin{cases} 1 & v_{ij}(n) > 1 \\ v_{ij}(n) & |v_{ij}(n)| \le 1 \\ -1 & v_{ij}(n) \le 1 \end{cases} \quad \textbf{(4.3)}$$

Bu adımda *x* ve *y* parametreleri aynı alındığı için denklemi daha basite indirgeyen bir yöntem uygulanmıştır. *h* ve *(1-h)* terimleri *A* ve *B* şablonlarının içine sokularak denklem yeniden düzenlenmiştir. $\hat{A}$ ve $\hat{B}$ yeni oluşan şablonlar olmak üzere, elde edilen yeni denklem aşağıdaki gibidir:

$$g_{ij} = \sum_{C(k,l)\in S_r(i,j)} \hat{B}_{ij,kl}u_{kl} + hz_{ij} \quad \textbf{(4.4)}$$

$$v_{ij}(n) = \sum_{C(k,l)\in S_r(i,j)} \hat{A}_{ij,kl}x_{kl}(n) + g_{ij} \quad \textbf{(4.5)}$$

Burada dikkat edilmesi gereken şöyle bir husus vardır: *B* şablonu $\hat{B}$ olurken sadece bir *h* terimiyle çarpılmaktadır; buna karşın *A* şablonu $\hat{A}$ olurken orta elemanı (a_{00}) dışındakiler yine aynı şekilde *h* ile çarpılmakta, fakat *A* şablonunun orta elamanı diğerlerinden farklı olarak *(1+(h(a_{00}-1))* ile çarpılmaktadır.

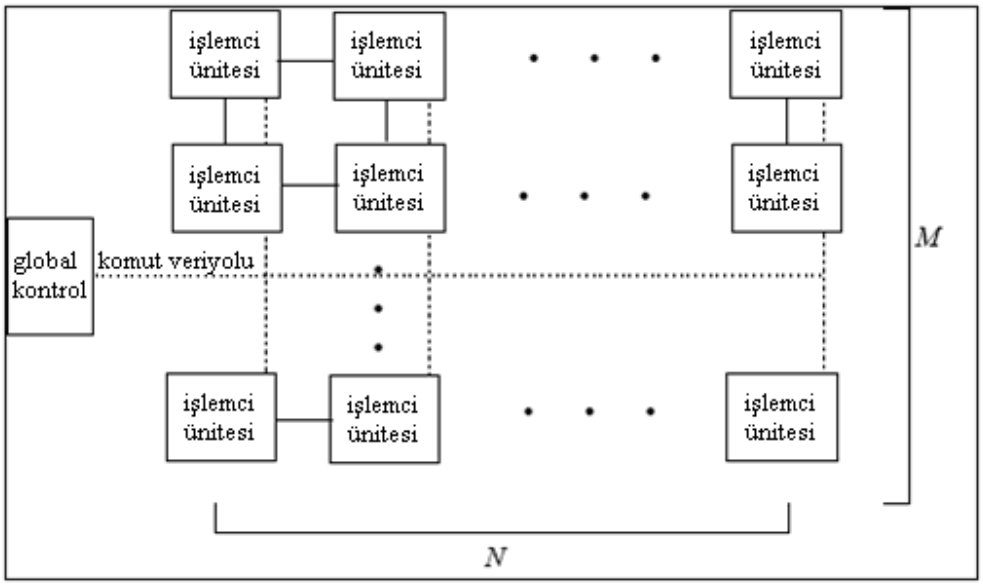
Denklemde yapılan bu iyileştirmeyle birlikte toplam hesap, bir konvolüsyon (nokta çarpım, *dot product*) ve buna eklenen bir sabitten oluşmaktadır.

Bu işlem dijital devrede hesaplanırken önce görüntü ve hz_{ij} alınıp, g_{ij} değerleri oluşturulmaktadır, buna *birinci evre* adı verilmektedir. Daha sonra bu oluşan g_{ij} değerleriyle birlikte iterasyon yapılarak $v_{ij}(n)$ değerleri, yani durumlar hesaplanmaktadır; bu bahsedilen ise *ikinci evre* olarak adlandırılmaktadır.

4.2 Devrenin Dışarıdan En Genel Halde Gösterilişi

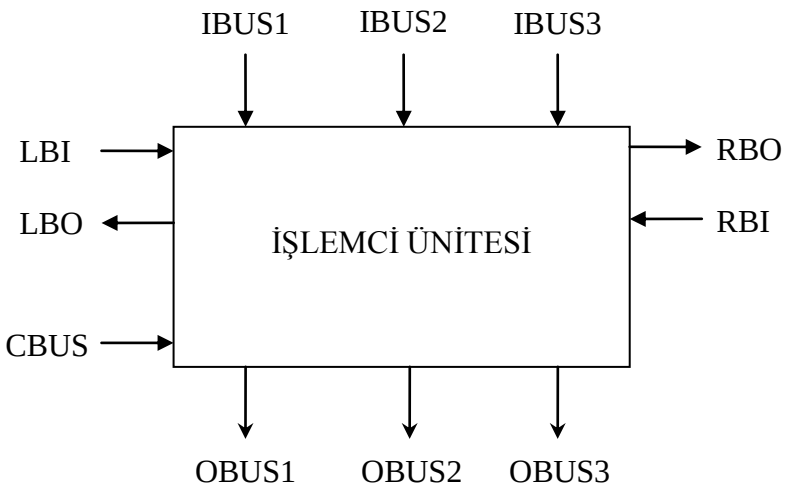
Şekil 4.1'de görüldüğü gibi her bir hücre satırda ve sütunda komşularıyla bağlantılıdır. Bütün hücreler ortak bir kontrol ünitesi tarafından kontrol edilmektedir.

Yataydaki hücreler arasındaki bağlantılar komşuluk bilgilerinin hücreler arasında iletişimini sağlamak amacıyla koyulmuştur. Fakat hücreleri birbirlerine düşeyde bağlayan bağlantılar aynı amaçla yapılmamıştır. Buradaki bağlantılar iterasyon sonucu iletimi için yapılmıştır. Bir üstteki hücre bir alttaki hücreye hesapladığı iterasyon sonucunu verir; kendisi ise bir üstteki hücrenin hesapladığı iterasyon sonucunu alır.



Şekil 4.1: Dijital hücresel sinir ağında işlemci üniteleri ve global kontrol bloğu.

Üstte gösterilen bir işlemci ünitesinin bağlantıları detaylı olarak Şekil 4.2'de gösterilmiştir. Bu giriş ve çıkışlar daha detaylı olarak Bölüm 4.5'te anlatılacaktır.

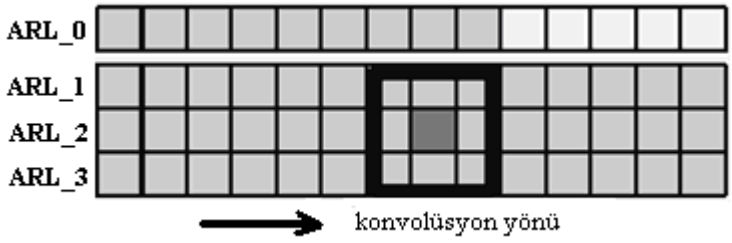


Şekil 4.2: Bir işlemci ünitenin giriş ve çıkış işaretleri.

IBUS1 veri yolu üstteki işlemciden iterasyon sonucunu alır. OBUS1 bir alt hücreye iterasyon sonucunu gönderir. RBI/O ve LBI/O uçları ise sağ ve sol hücrelerle yapılan komşu değerleri almak-vermek için koyulan veri yollarıdır. CBUS veri yolundan sürekli olarak global kontrol ünitesinden gönderilen işaretler alınır. Bu işaret hücrenin kontrol işlemlerinde kullanılır.

4.3 Görüntünün Saklanması ve İşlenmesi

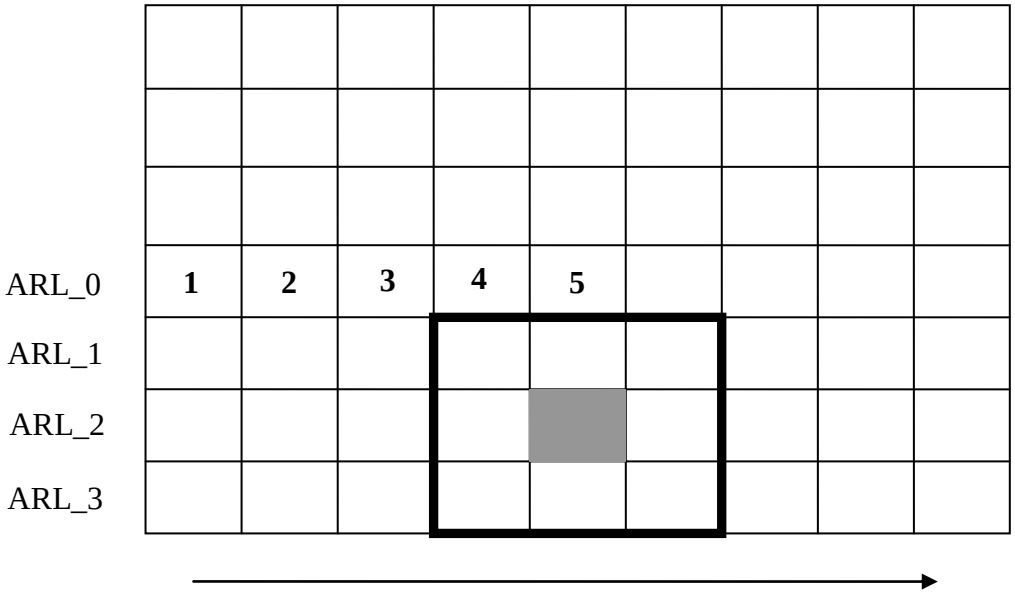
Dışarıdan yapılan okuma yazma (görüntünün saklandığı RAM ile yapılan) işlemi sayısını azaltmak için işlemci birimin içerisindeki ARL_1, ARL_2, ARL_3 kaydedicilerine resmin bir bölümü üç satır halinde yüklenmektedir. Daha sonra, Şekil 4.3'te okla belirlendiği gibi, konvolüsyon işlemi soldan sağa doğru yapılmaktadır. Kaydedici transfer seviyesinde gösterimin anlatıldığı Bölüm 4.4.1 incelenince veri yükleme işlemi daha iyi anlaşılacaktır.



Şekil 4.3: Görüntünün işlemci içindeki kaydedici dizilerinde saklanışı.

Şekil 4.3'deki gri olan kısım, önceden doldurulmuş kısım; koyu siyah çizgiyle çerçevelenen kısım, konvolüsyonun o an gerçeklendiği kısımdır. Konvolüsyon yapıldıkça ARL_0 kaydedici dizisine pikseller yüklenmektedir.

Şekil 4.4 incelendiğinde bir resmin ilgili diliminin topolojik olarak kaydedici dizilerine nasıl yüklendiği gösterilmiştir. 3x3 şablonla gri olan bölge hesaplanıp sonucunu bir aşağıdaki işlemciye gönderirken ARL_2'nin gri renkle belirlenmiş pikselinin hizasına düşen ARL_0'daki 5 nolu kaydedici de üstteki işlemciden gelen veriyle doldurulur. Bu işlem konvolüsyon yönünde ilerlenip resmin ilgili diliminin sonu gelene kadar devam eder ve ARL_0 üstten gelen değerlerle doldurulmuş olur.



Şekil 4.4: Resim diliminin kaydedici dizilerine yüklenişi ve konvolüsyon yönü

4.4 Kaydedici Transfer Düzeyinde Bir Hücresel İşlemcinin Tasarımı

Şekil 4.5'te görüldüğü gibi, IBUS1 veri yolundan ilk evrede u_{kl} değerleri, ikinci evrede ise $x_{kl}(k)$ girmektedir. IBUS2 veri yolundan ilk evrede $h*z_{ij}$, ikinci evrede ise g_{ij} değerleri girmektedir. IBUS3 veri yolundan hangi piksel için hangi şablon uygulanacağı bilgisi gelmektedir; kısaca şablon adresi gelmektedir denilebilir. RBI, LBI sağ-sol girişler; RBO, LBO sağ-sol çıkışlardır. Bu giriş ve çıkışlar sağ ve sol komşu işlemcilerle komşuluk ilişkilerini sağlamaktadır. ZAMANLAMA ve KONTROL birimi işlemcinin içsel işlemlerinde (hesaplama, yükleme, vb,) gerekli zaman ayarını yapan, diğer birimlere kontrol işaretleri gönderen birimdir.

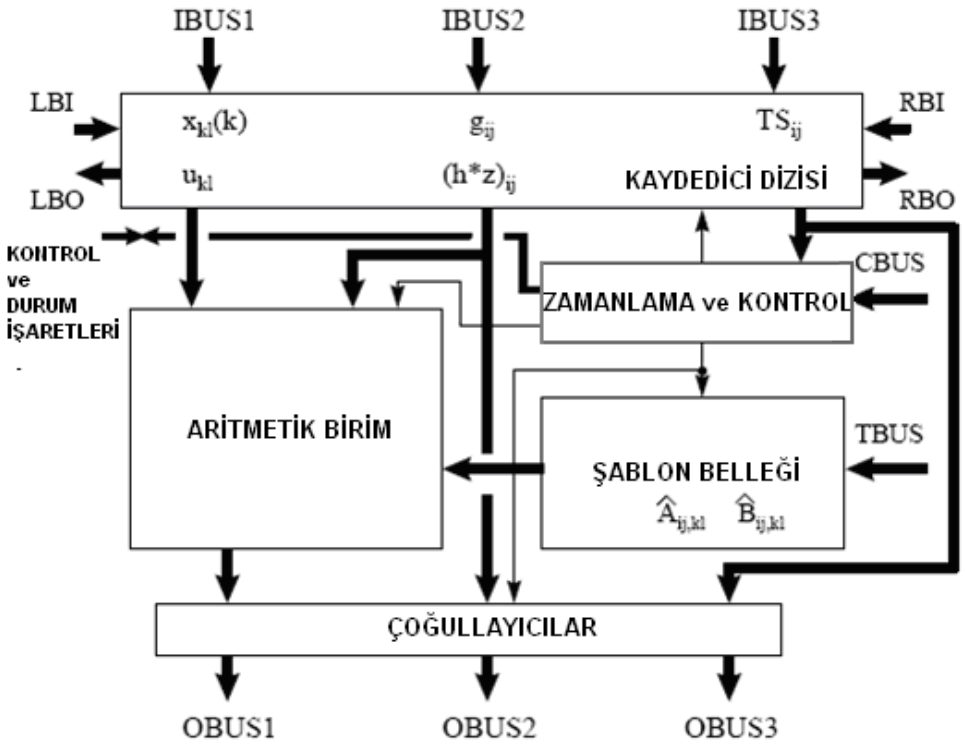
CBUS veri yoluyla genel kontrol bloğundan zamanlama kontol birimine işaret gönderilir. Böylece tüm satırdaki ve sütundaki işlemciler genel bir kontrol bloğu ile denetlenmiş olmaktadır.

Şablon belleği bloğuna kullanılacak şablonlar yazılmıştır. Bu birim bir nevi ROM görevi görmektedir. Devre çalışırken içindeki bilgiler değişmemektedir. Bu hafızaya yüklenen şablonların esas *A* ve *B* şablonları olmayıp, değiştirilmiş şablonlar olan $\hat{A}$ ve $\hat{B}$ olduğu unutulmamalıdır. Buradaki şablonlar, TBUS veri yolundan gelen

bilgiyle (şablon adresi) seçilerek aritmetik işlem yapan ARİTMETİK BİRİM isimli bloğa gönderilir.

ARİTMETİK BİRİM, içerisinde toplayıcı ve çarpıcılar olan, gerekli konvolüsyon işlemini yapan ve frekansı belirleyen gecikmelerin oluştuğu bloktur.

OBUS1, OBUS2 ve OBUS3 veri yolları sırayla, oluşan sonucu, g_{ij} değerini ve şablon adresini bir aşağıdaki hücreye gönderir.

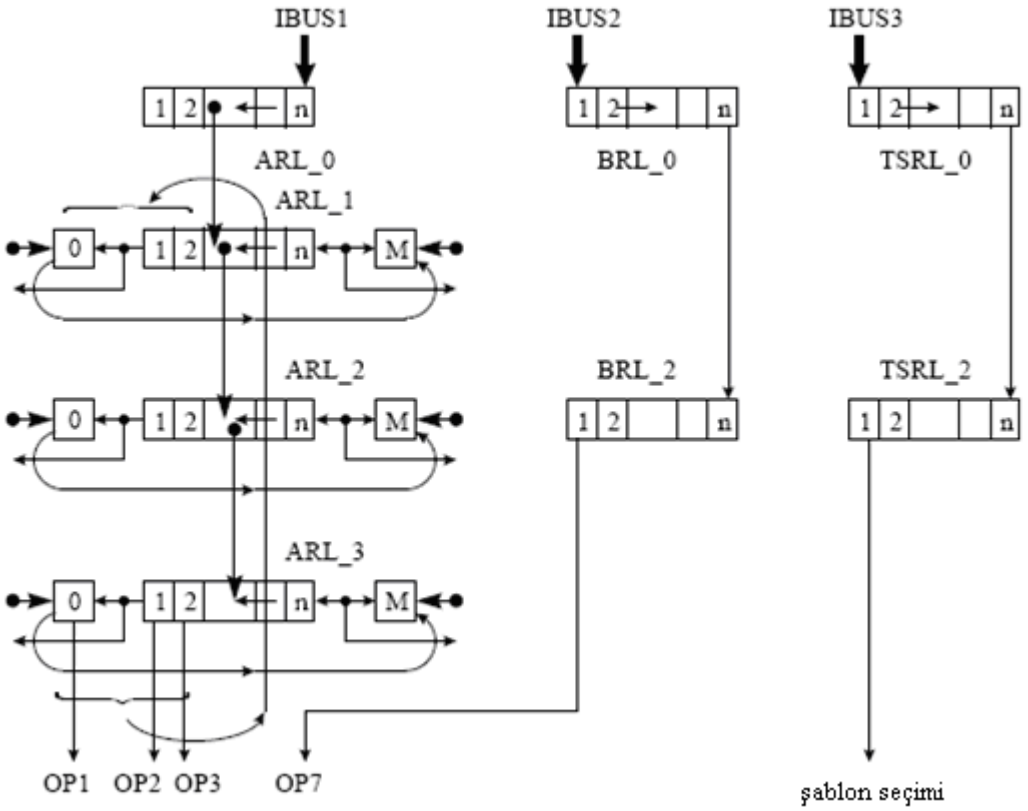


Şekil 4.5: Bir işlemci biriminin sistem seviyesinde blok gösterimi.

4.4.1 Kaydedici dizisi tasarımının ayrıntıları

En karmaşık olan kısımlardan birisi kaydedici dizisidir. Burada yapılan tasarım görüntünün dışarıdan okuma ve dışarıya (RAM, port, vb.) yazma işlemlerini azaltmak, ayrıca komşu hücrelerle iletişim problemlerini azaltmak için yapılmıştır. Şekil 4.6'da birinci sütunda gösterilen kaydediciler durumları, ikinci sütundaki kaydediciler sabit değerleri ve üçüncü sütundaki kaydediciler şablon adreslerini tutmaktadır. Her küçük kutu, sol sütundakiler için bir pikseli, ortadakiler için bir sabit değeri, sondakiler için ise bir şablon adresini tutan kaydediciyi temsil etmektedir. Üçüncü sütunda her bir piksel için bir şablon adresi olduğu düşünülünce

her bir piksel için ayrı bir konvolüsyon işlemi (ayrı bir filtreleme) yapılabilme imkânı görülecektir.



Şekil 4.6: Kaydedici transfer seviyesinde kaydedici dizisi tasarımı.

Sol sütundaki kaydedicilerden 0 ve M ile kodlanmış olanlar diğer işlemcilerden gelen komşu durum değerlerini içermektedir; bu, üç satır için de geçerlidir.

Yükleme işlemi şöyle olmaktadır: İlk önce en üstteki ARL_0 kaydedici dizisi teker teker seri olarak yüklenmektedir. Dizi tamamıyla dolduğunda, bu dizi olduğu gibi paralel olarak ARL_1'e yüklenmektedir aynı zamanda ARL_1'de ARL_2'ye yüklenmektedir. BRL_2 ve TSRL_2 yüklenme işlemi, ardışıldır (tek tek seri olarak yüklenmektedir) ve BRL_0 ile TSLR_0 kaydedici dizisinin ötelenmesiyle aynı anda olmaktadır. Üstten BRL_0 ve TSRL_0'ın her bir kaydırmada veri aldığı düşünülürse çalışma prensibi daha iyi anlaşılacaktır.

Görüldüğü gibi IBUS1 veri yolundan gelen durum değerleri şablon elemanlarıyla çarpılarak konvolüsyon işlemi yapılacaktır. İlk önce en alttaki satırdaki soldan üç tane durum (paralel öteleme göz önüne alınırsa bunların kaydedici dizisine ilk giren

değerler olduğu anlaşılacaktır) ile ilgili şablon elemanları çarpılacaktır. İkinci safhada ikinci satırdaki ilk üç durumla şablonun ilgili elemanları çarpılacaktır. Bu, işlemin üçüncü safhasında da aynı olacaktır. Görüldüğü gibi bir konvolüsyon işlemi (3x3) üç safhadan oluşmaktadır. Bu üç safha sonunda durumları saklayan üç satır kaydedici dizisinin hepsi, bir sola döndürülür (rotate); böylece artık bir sağdaki piksel için işlem yapılabilecek duruma gelinmiştir. Döndürme işlemiyle aynı anda BRL_2 ve TSRL_2'nin bir sola kaydırıldığı, BRL_2 ve TSRL_2'ye BRL_0 ve TSRL_0'dan ayrıca üstteki ARL_0, BRL_0 ve TSRL_0'a veri yollarından birer tane veri girmektedir. Bu işlemlerin hepsi aynı anda olmaktadır.

Bir satıra ilişkin konvolüsyon işlemi bittiğinde, konvolüsyon işlemi biten satırın iki üst veya iki alt (alttan okunduğunda iki üst, üstten okunduğunda ise iki alt olmak üzere) satırı tamamen ARL_0 kaydedici dizisine yazılmış olmaktadır.

4.4.2 Aritmetik birimin tasarımının ayrıntıları

Bu bölümde aritmetik ünitenin kaydedici transfer seviyesinde yapısı incelenecektir. Şekil 4.7 aritmetik işlem yapan ünitenin elemanlarını alt bloklar olarak açıklamaktadır.

ÇARPICI birimleri Baugh-Wooley [18] algoritmasına uygun şekilde tasarlanmış n-bit çarpıcılardır.

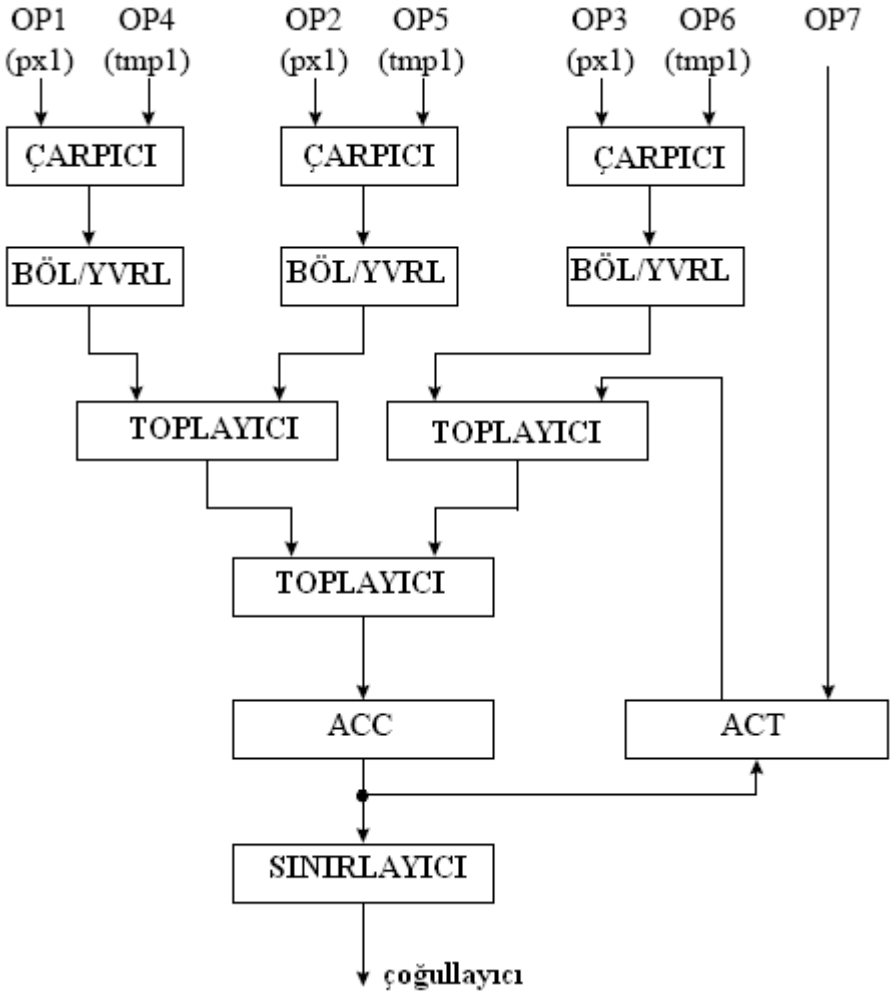
BÖL/YVRL birimleri çarpma bloklarının çıkışına eklenmiş olup bölme yapan devrelerdir. Bölmede kaybolan bitlerden dolayı sayı hassasiyetinde bir azalma görülür. Bu blok içinde hassasiyeti artırmak üzere yuvarlama işlemi de yapılmaktadır.

TOPLAYICI birimleri ileri bakmalı eldeli toplayıcılardır.

ACC birimi en son toplamı saklayan bir yükleyicidir (accumulator).

ACT birimi iterasyonun ilk safhasında sabit değerin toplama katılmasını, ikinci ve üçüncü safhada da oluşan kısmi toplamların toplama katılmasını (işaret seçimi) yapan bloktur.

SINIRLAYICI birimi oluşan en son toplamda gerekli durumlarda kesme işlemi yapan bloktur. Bu matematiksel ifadede *y* fonksiyonuna tekabül etmektedir.



Şekil 4.7: Aritmetik işlem yapan birimin iç yapısı.

4.5 Tasarımın Diğer Tasarımlarla Karşılaştırılması

P. Keresztes ve diğerlerinin yaptığı çalışmada [3] bazı dijital görüntü işleme yongalarıyla karşılaştırmalar yapılmıştır. Çizelge 4.1'deki bir çok uygulama gözetilerek yapılan karşılaştırmalarda görülmüştür ki, dijital HSA gerçeklemesi (CASTLE) hız açısından diğer yongalardan oldukça avantajlıdır.

Çizelge 4.1: Anlatılan tasarımın diğer bazı yongalarla karşılaştırılması [3].

	Taiwan	CAM^{2}Toshiba	Darmstadt	CNAPS	CASTLE*
Saat Frekansı	33 Mhz	40 Mhz	25 Mhz	25 Mhz	66 Mhz
Kullanılan teknoloji	1.2 µm	0.5 µm	0.7 µm	0.5 µm	0.35 µm
Yonga alanı (cm^2)	1 cm^2	-	1 cm^2	2 cm^2	1 cm^2
Fiziksel işlemci sayısı	9 (3x3)	Dağılmış	9 (9x1)	64	24
güncelleme/piksel/işlem (lojik işlem)	30 ns	0.86 ns	13 ns	9.4 ns	0.1 ns
güncelleme/piksel/işlem (3x3 konvolüsyon)	30 ns	85 ns	13 ns	9.4 ns	1 ns
Kaskatlanabilirlik	Yok	Var	Var	Yok	Var
Hesap karmaşıklığı (AT^2)	900	-	196	176	1

(AT^2: A, yonga alanı; T, hesaplama zamanı)

Burada anlatılan tasarım (CASTLE), 24 hücrelidir.

Çizelge 4.2'de analog HSA gerçeklemeleri ve bazı ardışıl işlemcilerle dijital HSA gerçeklemesi olan CASTLE karşılaştırılmıştır. Bu karşılaştırmalarda analog ve dijital HSA gerçeklemesinin bariz bir şekilde ardışıl işlemcilerden daha performanslı olduğu görülmüştür. Analog HSA gerçeklemesi dijital HSA gerçeklemesinden hızlı olmakla birlikte aradaki fark kabul edilebilirdir. Dijital HSA tasarımının kolaylığı düşünülünce aradaki bu küçük hız farkının pek bir önemi yoktur.

Çizelge 4.2: Analog CNN yongalarıyla CASTLE tasarımının karşılaştırılması [3].

	Pentium II MMX 0.25µm, 400MHz (Windows NT)	TMS3206x 0.25µm, 200MHz 8 işlemci	CASTLE dijital 0.35µm, 24 işlemci	CNN yongası 0.8µm, τCNN: 250ns	CNN yongası 0.5µm, τCNN: 100ns
3x3 konvolüsyon B şablonu 6τCNN veya 1 iterasyon	500	1.638	16 (12-bit) 8 (6-bit)	14.5	10.6
Erozyon/Yayma A_{00}+B şablonu $6\tau_{CNN}$ veya 1 iterasyon	237.5	1.200	16 (12-bit) 8 (6-bit) 1.6 (1-bit)	14.5	10.6
Laplace A+B şablonu 15 τ_{CNN} veya 15 iterasyon	7.500	24.570	240 (12-bit) 120 (6-bit)	16.8	11.5
A Algoritması: 10 konvolüsyon + 10 Erozyon + 1 Laplace	14.875	52.950	560 (12-bit) 280 (6-bit)	46.8	23.5
B Algoritması: 10 konvolüsyon + 10 Erozyon + 10 Laplace + 10 Lojik işlem	82.375	274.080	2.720 (12-bit) 1.360 (6-bit)	80.5	37

Üstteki tablodaki veriler elde edilirken; 128x128 görüntü işlenmiştir, hesaplama zamanı µs cinsinden olup üstte görülen zamanlara veri yükleme zamanı dâhildir, τ_{CNN} analog CNN yongasının zaman sabitini temsil etmektedir.

4.6 Tasarımın Pipe-Line Yöntemiyle Hızlandırılması

Devrenin çalışma frekansı devrede oluşan en büyük gecikme yardımıyla hesap edilir. Aritmetik işlem biriminde görüldüğü gibi birbiri ardınca bağlanmış çarpıcı ve toplayıcı blokları vardır. Burada oluşan gecikmenin çok büyük olacağı görülmektedir. Toplam gecikme; bir çarpıcı, iki toplayıcı bloğunun gecikmeleri toplamına eşittir. Devrenin daha yüksek frekanslarda çalışabilmesini sağlamak

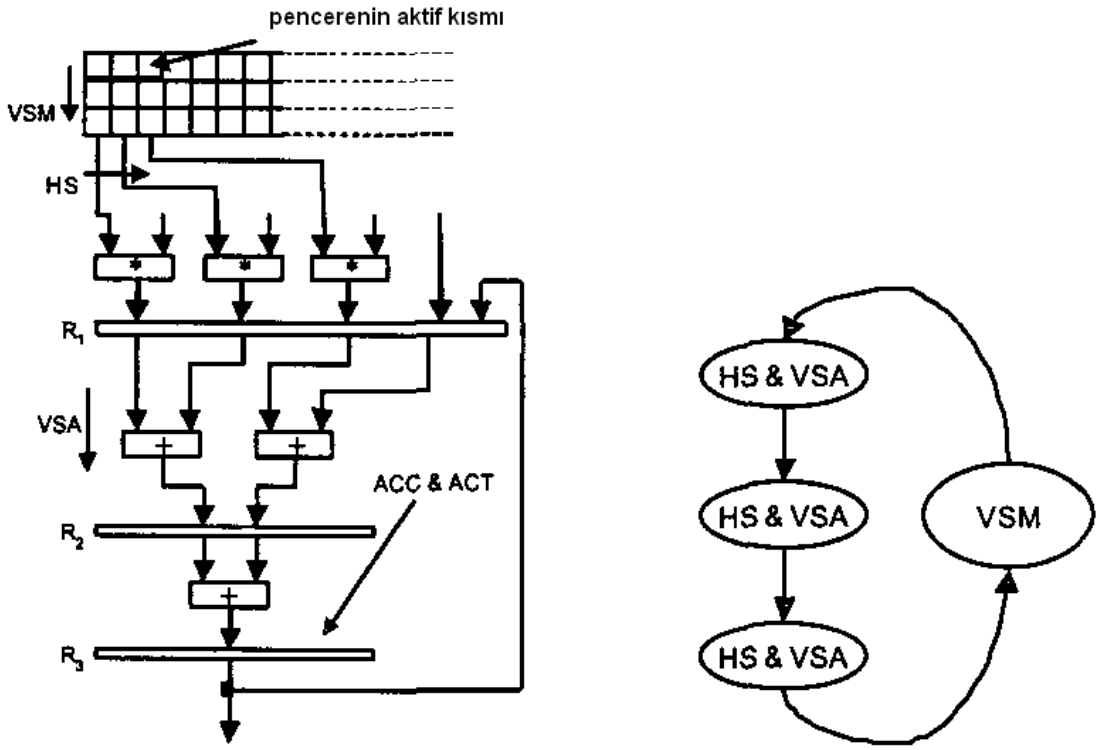
amacıyla pipe-line tekniğiyle toplama ve çarpma bloklarının arasına kaydediciler koyarak en uzun gecikme, en yavaş çalışan bloğun gecikmesi olacak şekilde azaltılmıştır [14]. Bu iyileştirmenin ayrıntıları Şekil 4.8'de görülmektedir. Yine de istenilen ölçüde bir hızlanma olmadığı açıktır. Çünkü en uzun gecikmeye sahip olan çarpıcı bloğunun gecikmesi devrenin gecikmesinin nerdeyse tamamını oluşturmaktadır. Bu açıdan bakıldığında toplayıcı ve çarpıcı bloklarının içine de bazı kaydediciler koyarak bu blokların da hızlandırılması düşünülmüştür. Toplayıcı bloğunun pipe-line yapılması Şekil 4.9'da, çarpıcının pipe-line yapılması Şekil 4.10'da ve en son durumda yapılması gereken okuma sırası Şekil 4.11'de açıklanmıştır. Yapılan bloklar arası ve blok içi pipe-line ile yaklaşık olarak "6" kat bir hız artımı olmuştur [14].

Burada anlatılan kaydedici yerleştirmek şeklinde yapılan pipe-line ile giren bir girişin sonucu ancak araya konulan toplam kaydedici sayısınca saat çevriminden sonra çıkışta görülür. Çıkışta oluşan bu değişim unutulmamalıdır. Ayrıca giriş değerlerini de bir yönteme göre okumak gerekmektedir. Bu yöntem [14] nolu referansta detaylı olarak anlatılmıştır.

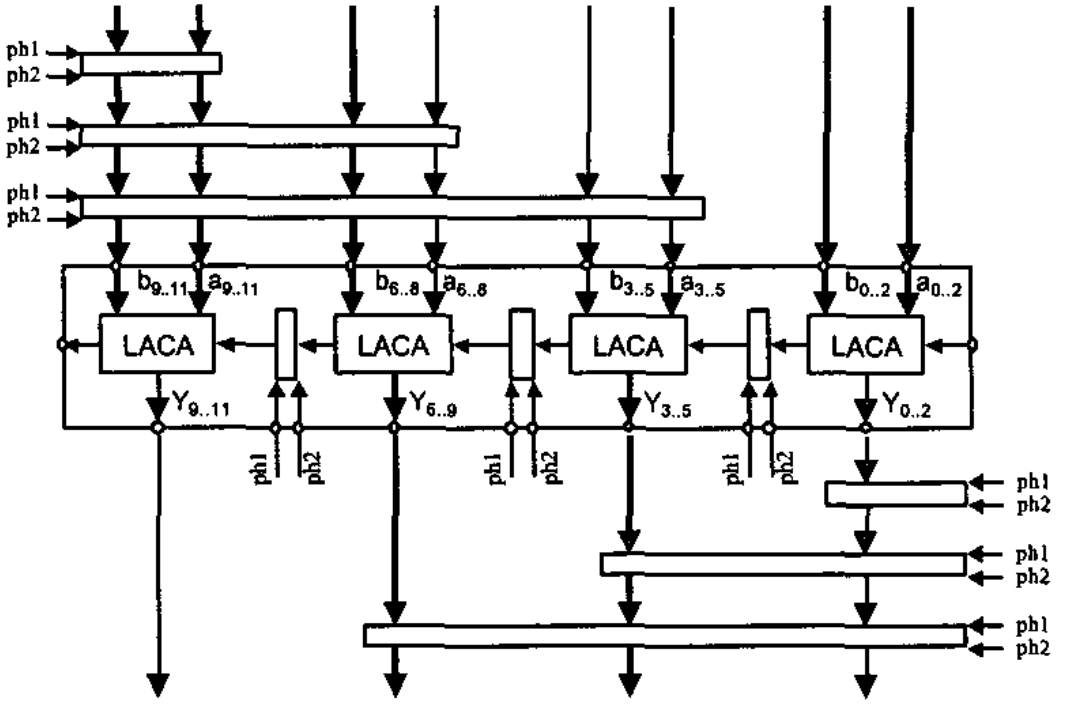
Burada ilgili makalede detaylı incelenilmeyen fakat pipe-line ile ilgili önemli olan bir husus şudur: yeni veri okuma şekliyle her bir işlemci bloğu içindeki kaydedici dizisi boyunca yatayda ilerlenmektedir ve yataydaki kaydedici uzunluğu pipe-line seviyesinden (aralara yerleştirilen kaydedici sayısından) az olursa ilk sonuç henüz çıkmadan dizi tamamen dolaşılmış olacaktır. Bu yüzden ilgili tasarımdaki işlemci blokları içindeki kaydedici yatay uzunluğu belirli bir sayının altına inemez. Yani yapılan tasarımdaki yatayda sıralanan işlemci blok sayısı resmin yataydaki piksel sayısının belirli bir oranından daha fazla olamaz. Böylece, sistemin kendi tasarımından dolayı oluşan sebeplerle eğer pipe-line yapılmak isteniyorsa yatayda sıralanabilecek işlemci sayısı kısıtlanmış olmaktadır. Yatayda hücre sıralamanın düşeyde hücre sıralamaya avantajı ilerde anlatılacaktır.

Şekil 4.8 incelenecek olunursa aritmetik ünite içindeki her bir alt bloğun çıkışına, çıkışın bit genişliğinde bir kaydedici yerleştirilmiştir. Devre tam senkron olarak saatle çalıştığı için her bir saat işareti ardından çıkış ilgili kaydediciye aktarılmış olacaktır. Kaydedici dizisinden okumalar uygun şekilde yapıldığı zaman, üç piksel için yapılmış bir konvolüsyonun sonucu toplama katılma sırası geldiğinde girişlere

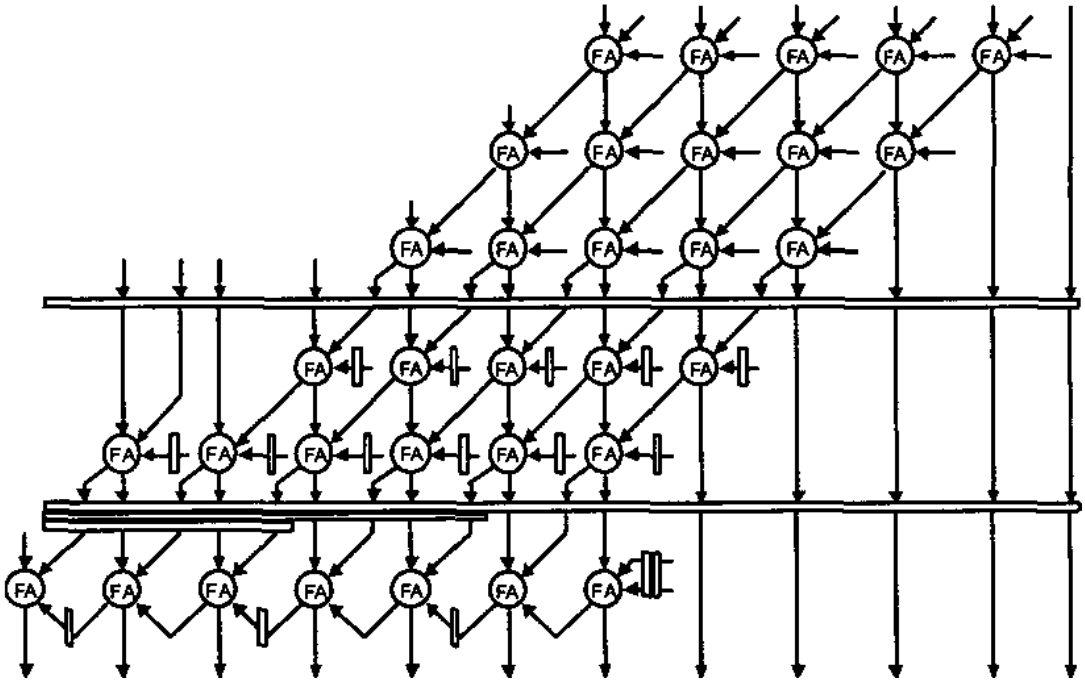
aynı üç pikselin hemen bir üst yada alt (okuma yönüne bağlı) sırasındaki pikseller gönderilirse ilgili kısmi toplamlar ilgili çarpımlarla toplanmış olacaktır. Bu sayede bekleme süresi sadece bir bloğun süresine düşerken işlem toplamda aynı olacağından ciddi bir zaman kazancı olacaktır. Şekil 4.8'de HS yatay ötelemeyi, VSA kaydedicilere gelen saat işaretiyle, kaydedicilerdeki verilerin ötelenmesini ve VSM ise pencerenin aktif kısmındaki ötelemeyi göstermektedir.



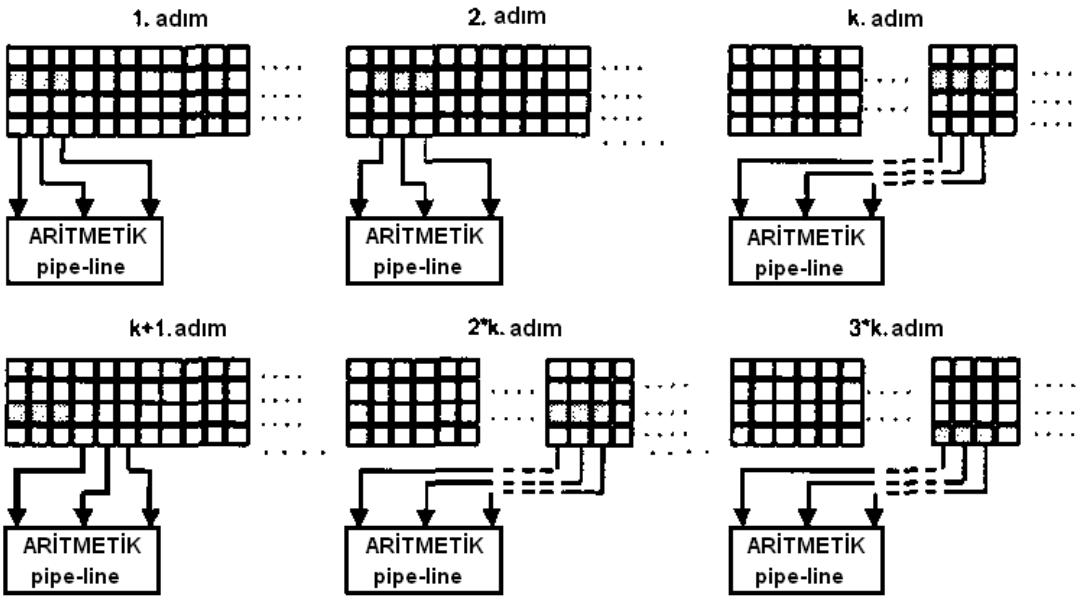
Şekil 4.8: Pipe-line tekniği ile yapılan aritmetik blok ve çalışma algoritması.



Şekil 4.9: Tasarımda kullanılan (CASTLE) 12-bitlik pipe-line yapılmış toplayıcı.



Şekil 4.10: Tasarımda kullanılan pipe-line yapılmış 6-bit Baugh-Wooley çarpıcısı.



Şekil 4.11: "k" seviye pipe-line yapılmış sisteme uygun veri okuma sırası.

Görüldüğü gibi, "k" seviye pipe-line için k+1 kadar yatayda kaydedici olması gerekmektedir. Bu da belirli bir resim için yatayda koyulabilecek en fazla işlemci sayısını belirlemektedir. Bu şu şekilde ifade edilebilir:

yataydaki işlemci sayısı ≤ yataydaki piksel sayısı / (pipe-line seviyesi + 1)

5. VHDL İLE YENİ BİR DİJİTAL HÜCRESEL SİNİR AĞI TASARIMI

Yapılan literatür araştırması sonucunda, var olan dijital Hücresel Sinir Ağı gerçeklemelerinde bazı temel problemlerin olduğu görülmüştür. Bunlar, önem sırası tasarımda aranan kritik özelliklere bağlı olarak değişebilmekle birlikte, aşağıdaki gibi sıralanabilir:

1. Görüntünün nerede ve nasıl saklanacağı problemi, bilhassa büyük görüntüler işlenirken bir hayli önemlidir.

2. Görüntü işlemek için tasarlanan hücre işlemcilerin birbirleriyle olan komşulukları ve kenar koşullarının davranışlarının dijital devrede gerçeklenmesi ek bir sorun olarak karşımıza çıkmaktadır.

3. Görüntü boyutu arttıkça ve özellikle gerçek zamanlı çalışan sistemler söz konusu olunca, tasarlanan devrede hız kritik bir mesele olarak karşımıza çıkmaktadır. Yapılan tasarım, gerçek zamanlı sistemleri destekleyebilmelidir.

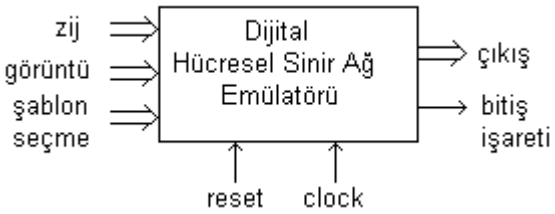
4. Yeni bir tasarım, birçok tasarımda olduğu gibi kabul edilebilir bir tümdevre alanında gerçeklenebilir olmalıdır. Barındırdığı birimler, eğer kapı bazında düşünülürse, mümkün olduğunca az sayıda kapı içermelidir. Görüntünün çözünürlüğü - piksellerin kaç bitle ifade edileceği – konusu hem devre büyüklüğü hem de kaliteli sonuç bağlamında düşünülebilir.

5. Genel bir özellik olmakla birlikte, tasarlanan devrenin veya devrede kullanılan işlemci birimlerin çoklu - paralel ve kaskat kullanıma müsait olup olmaması, devrenin istenildiğinde büyütülebilmesi için önemlidir.

6. Bütün dijital devrelerde olduğu gibi tasarımda karmaşıklık önemlidir; fakat hayati önem arz eden sorunlara uygun çözüm bulunduğu durumlarda bu noktada bazı tavizler verilebilir. Örneğin, bu tezde önerilen gerçeklemede hücre işlemcilerin kontrollerini basitleştirebilmek için daha karmaşık bir ana kontrol birimi uygun görülmüştür.

5.1 Önerilen Gerçeklemenin Mimarisi

Tezde önerilen devrenin giriş ve çıkışları Şekil 5.1'de gösterilmiştir. Z_{ij}, eşik değerlerinin iletildiği uçtur. Görüntü adı verilen uçtan görüntü pikselleri tek tek gönderilmektedir. Şablon seçme diye belirtilen uçtan kullanılacak A ve B şablonlarının hangileri olacağı bilgisi gönderilecektir. Çıkış ucundan işlenen görüntüler alınabilecektir. Bitiş işareti işlemin bittiğini belirtmektedir.



Şekil 5.1: Dijital Hücresel Sinir Ağının dışarıya açılan uçları

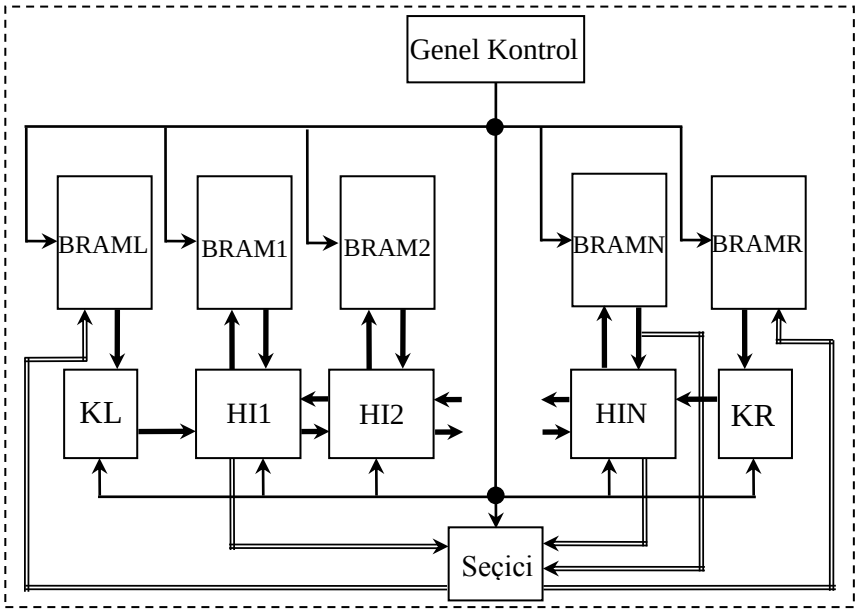
Şekil 5.1'de tasarlanan devrenin dışarıyla olan bağlantıları yani ara yüzü gösterilmiştir. Bu devre, herhangi bir donanımdan (bilgisayar veya kamera) pikselleri teker teker okur ve Şekil 5.2'de gösterilen RAM elemanlarına kaydeder. Görüldüğü gibi her bir hücresel işlemciyle giriş çıkış işlemleri yapan bir RAM vardır. Bu RAM ile ilk evre için piksel ve $h*z_{ij}$ değerleri, ikinci evre içinse durum (ilk durum olarak görüntünün kendisi kullanılır, bunlar da RAM içinde hazır bulunmaktadır) ve g_{ij} (4.4 nolu denklemde gösterilmiştir) değerleri saklanacaktır. Birinci ve ikinci evre için yapılması gereken matematiksel işlemler bölüm 4.1'de anlatılmıştır. İkinci evre tanımlanan bir iterasyon sayısı kadar tekrarlanır. Yapılan iterasyonlarda RAM içerikleri güncellenmektedir.

Genel Kontrol, RAM birimlerine adres, yazma aktif ve yonga aktif işaretlerini gönderir. Bu işlerin aynısı komşu RAM blokları için de gereklidir. Ayrıca diğer bütün birimlere kontrol işaretleri göndermektedir. Hücresel işlemciler içindeki kontrol işlemini azaltmak için, bunlara sadece basit kontrol işaretleri gönderilmiştir. Dolayısıyla karmaşık hesapları yapan blok Genel Kontrol olmuştur. Bunun getirdiği faydalar vardır: Hücresel işlemcilerin sayısı birden fazla olduğu için, bunların içindeki bütün bloklar hücresel işlemci sayısınca gerçeklenecektir; eğer işlemci içindeki kontrol birimi karmaşık veya büyük olursa bundan işlemci sayısınca gerçeklenince otomatik olarak devre boyutu büyümüş olacaktır. İşlemci içindeki

kontrol kısmı olabildiğince küçük tutulursa, çok sayıda hücre gerçeklenince işlemci içindeki kontrol birimi devre olarak daha az yer kaplar. Bunun yanı sıra, Genel Kontrol birimi tüm devrede bir tane olduğu için, hücresel işlemcilerdeki kontrol karmaşıklığının bu bloğa kaydırılması daha mantıklı olacaktır.

KL ve KR birimleri, kenardaki hücrelere komşuluk sağlamak amacıyla koyulmuştur. Bunların hücresel işlemcilerden farkı herhangi bir aritmetik işlem yapmayıp sadece komşuluk ilişkisi sağlamalarıdır.

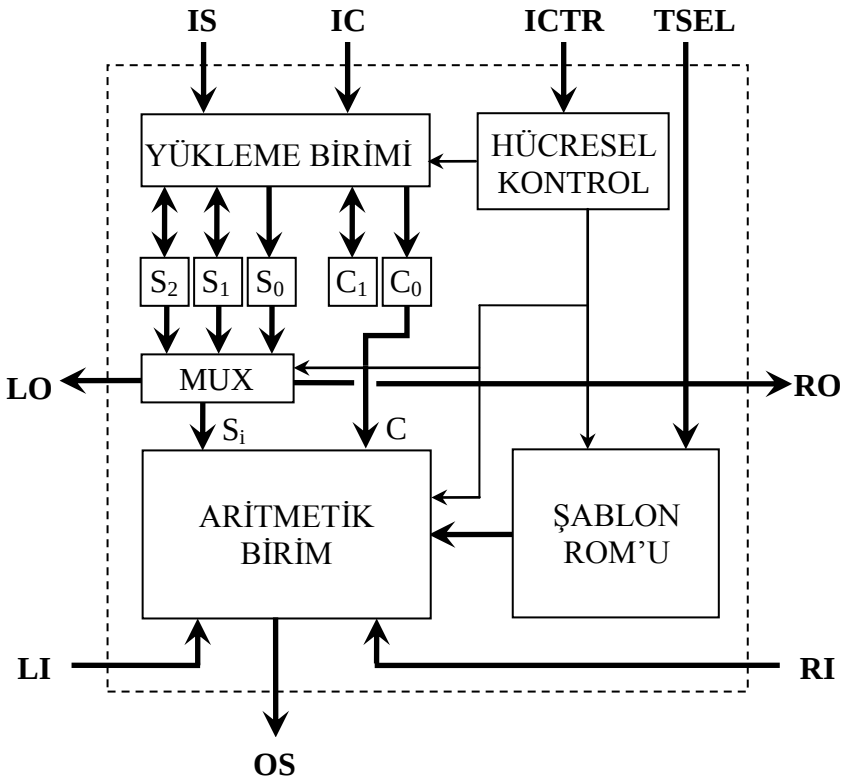
Seçici birimi Genel Kontrol biriminden gerekli kontrol işaretlerini alıp kendisine gelen hücresel işlemcilerin ya da RAM'in çıkışını komşu RAM birimlerine gönderen kombinezonsal bir seçim devresidir.



Şekil 5.2: Tasarlanan devrenin iç genel mimarisi.

5.2 Bir Hücresel İşlemcinin İç Yapısı

Şekil 5.3'te bir hücresel işlemcinin sistem seviyesinde gösterilimi sunulmuştur. Bu şekilde gösterildiği üzere: RI, LI sağ ve sol girişler; RO, LO sağ ve sol çıkışlardır. Bunlar komşu hücresel işlemcilerle bağlantıları sağlamak üzere konulmuştur. Her biri, bir pikselin ifade edildiği bit genişliğindedir. Literatürde var olan tasarımlarda [3,16] sağ ve sol komşuluklar için ayrılan bağlantılar üç pikselin ifade edildiği bit genişliğinde olmasına karşın, burada bir piksel bit genişliği kullanılıp hücreler arası bağlantılarda daha az bant genişliği kullanılması sağlanmış olmaktadır.



Şekil 5.3: Hücresel işlemcinin iç yapısı.

IS, çalışmanın birinci evresinde piksellerin (görüntünün) değerlerinin girdiği, ikinci evrede ise durumların işlemciye girdiği veri yoludur. IS, bir pikselin ifade edildiği bit genişliğindedir. IC, birinci evrede $h*z_{ij}$'nin, ikinci evrede de g_{ij}'nin girdiği veri yoludur. IC, bir pikselin ifade edildiği bit genişliğindedir. İncelenilen bazı tasarımlarda IC'nin bir pikselden farklı bit genişliğinde yapılabileceği görülmüştür. Fakat bu uygulamada piksel genişliğinde alınmıştır. ICTR, Genel Kontrol alt devresinden gelen kontrol işaretlerinin hücresel işlemciye girdiği veri yoludur. TSEL, şablon seçme işaretinin geldiği veri yoludur. Bir hücresel işlemcide saklanan şablon sayısına göre bu veri yolunun bit genişliği değişebilir. Literatürdeki uygulamalarda, TSEL'e eşdeğer olan veri yolundan gelen veriler, aynı g_{ij} değerleri gibi kaydedilip her bir piksele ayrı bir şablon uygulama imkânı sağlanmıştır. Bu uygulamada bu özellik gereksiz görülüp eklenmemiştir. Ancak eklenmesinde bir zorluk yoktur. Bu uygulamada tüm görüntüye aynı A ve B şablonu uygulanmaktadır. Bu şablonların adresi de Genel Kontrol alt devresinden gelmektedir.

OS, birinci evrede g_{ij}'nin, ikinci evrede de iterasyon sonuçlarının çıktığı veri yoludur. Bu çıkış her iki evrede de ilgili RAM bloğuna girmektedir.

$S_{0,1,2}$, piksel değerlerini konvolüsyon hesabı yapılırken saklayan üç kaydedicidir. Hesap için gereken diğer piksel değerleri ise komşu bağlantılardan gelmektedir. C değeri, sabit değerdir (g_{ij} veya $h*z_{ij}$). Bir iterasyonda sadece bir tane sabit değere ihtiyaç olmasına rağmen iki tane kaydedici koyulmasının sebebi, aynı anda durum ve sabit değerlerin okunmasına başlanmasından dolayıdır. İlk değerin bir defa okunup iki kaydediciye birden yazılmasını da göz önünde bulundurursak iterasyona başlayabilmek için durum RAM'inden iki okuma yapılması gerekmektedir. İlk satırda, sınır koşulları olduğu için ilk satırı iki defa kaydedicilere yazmak gerekmektedir. Bunun nedeni "zero-flux" yöntemiyle hesap edilen sınır koşullarında sınır durumu kendi durumuna eşit alınmaktadır. Görüldüğü gibi sabit değer için tek kaydedici kullanılması halinde iterasyona katılması gereken değer kaybolabilir. Bunun önüne geçilmek için bu yöntem uygulanmıştır. En son satırda durum ve sabit değerler için RAM'den okuma olmayacaktır (RAM'den son okunan değer tekrar yazılacaktır). C_1'de olan değer C_0'a kaydırılmış olur ve böylece bir iterasyon tamamlanmış olmaktadır.

Yükleme Birimi alt devresi giriş veri yollarından okunan değerleri sınır koşulu olup olmamasını gözeterek kaydedicilere uygun şekilde kaydetmek için tasarlanmıştır. Sınır koşulları olmadığı zamanlarda bu devre, durum kaydedicileri arasında sadece ötelemeli kaydetme (shift-register) yapmaktadır. Bu devre, sabit değerin saklandığı kaydediciler için her zaman ötelemeli kaydetmek görevi yapmaktadır. Çünkü sabit değer için komşuluk olmaması sebebiyle kenar koşulu da yoktur. Dikkat edilmesi gereken nokta, ilk yüklemeler hariç her bir iterasyon için sadece bir durum ve bir sabit değer yüklendiğidir.

Şablon ROM'u, şablonların saklandığı hafıza birimidir. Bu birime ROM denmesinin sebebi çalışma sırasında buradaki bilgilerin üstüne herhangi bir yazılma yapılmamasından dolayıdır. Şablon ROM'u kaydedicilerin boyutuna bağlı olarak birçok şablon saklayabilir.

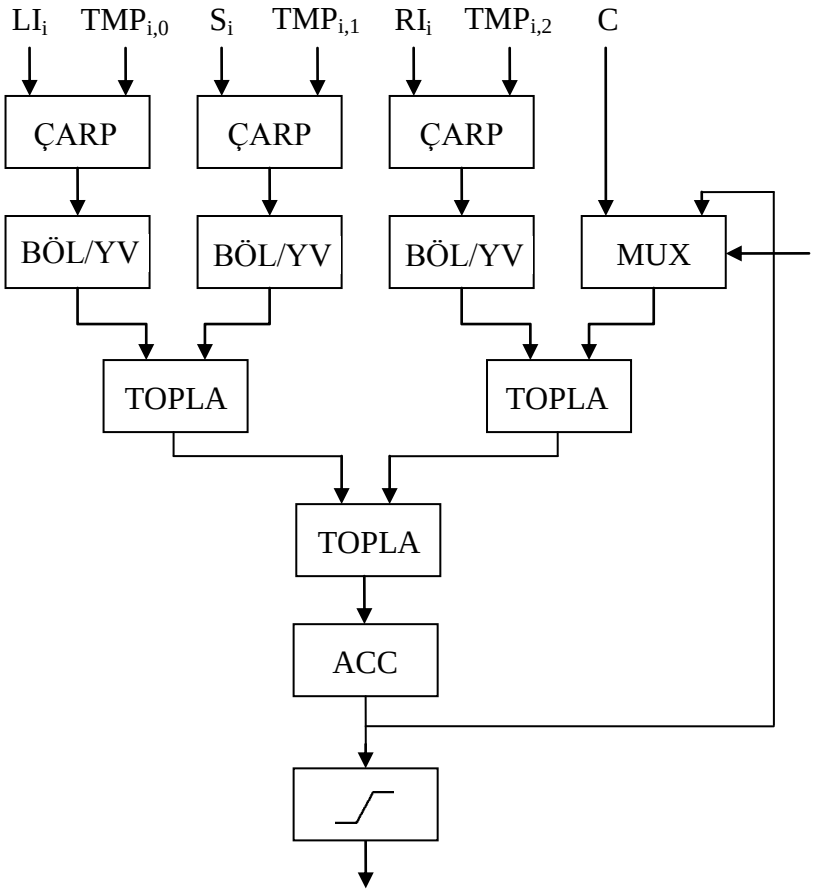
Konvolüsyon hesabı üç saat çevriminden oluşmaktadır. Bu üç çevrimde Aritmetik Birim alt devresine girmesi gereken ilgili girişleri (S_i) seçen devre MUX olarak isimlendirilmiştir.

Hücresel Kontrol, hücresel işlemci içindeki işlemleri kontrol eden birimdir. Bu birim Genel Kontrol tarafından gönderilen işaretleri alıp hücresel işlemci içindeki diğer birimlerin çalışmasını yönetir. Bu birim olabildiğince basit tutulmaya çalışılmıştır. Bu birim MUX devresinin seçimini sağlar. Aritmetik Birim alt devresine gönderilen bilgi ile toplama neyin katılacağı belirlenir. Yükleme Birimi devresinde bir defa yükleme yapıldıktan sonra, iterasyon bitene kadar tekrar yükleme yapılmamasını sağlar.

Bir konvolüsyon işleminin üç saat çevriminde olduğunu daha önce belirtmiştik. Şekil 5.4 incelendiğinde aritmetik işlemin nasıl yapıldığı daha iyi anlaşılacaktır. Bu üç çevrimden ilkinde, dokuz tane hesaplanması gereken şablon-durum nokta çarpımının ilk üçü yapılmaktadır. İlk çevrimde ek toplam olarak MUX ile C değeri seçilir. Çarpıcılara gelen değerler ise RI, S0, LI'dır. RI ve LI değerlerinin diğer hücresel işlemcilerin MUX birimi tarafından gönderilen ilgili değerler olduğu unutulmamalıdır. Bunlar resimde topolojik olarak S0'ın sağ ve sol komşularıdır. İkinci ve üçüncü çevrimde ise ACC geçici yükleyicisi ile saklanan kısmi toplam değerleri MUX ile seçilir ve toplama dâhil edilir. Böylece şablon ve durumun nokta çarpım işlemi gerçeklenmiş olur. Bu üç çevrim bittiğinde, RAM'den S ve C için yeni veri okunur, okunan veriler sırasıyla S2 ve C1'e kaydedilir. RAM'den ilk ve son okuma (son okumada RAM'den okuma yapılmıyor, sadece en son okunmuş veri tekrar en sondaki kaydedicilere yükleniyor) dışındaki her veri okumada ötelemeli kaydetme işlemi yapılır. Dolayısıyla veriler kaydediciler içinde birer kaydırılmış olur. İterasyondaki üçüncü çevrimden sonraki okuma işlemi S0 ve C0'a yeni veri yüklemesiyle bir sonraki konvolüsyonun ilk çevrimini oluşturmaktadır. Böylece konvolüsyon işlemi resmin altından üstüne doğru yapılmış olmaktadır.

Aritmetik Birim içindeki ÇARP birimleri şablon ve durum çarpımını gerçekleştirmektedir. TOPLA birimleri ise toplayıcılardır. Aradaki BÖL/YV birimi iterasyon adımı olan "h" ile bölmeyi sağlamak için tasarlanmış olan bölücü devredir. Basit bir öteleme işlemiyle bölme yapılmaktadır. Bölme yapıldıktan sonra kaybedilen bitlerden oluşan hassasiyet kaybını gidermek amacıyla yuvarlama

yapılmıştır. Görüldüğü gibi bölme her üç çevrimde de yapılmaktadır; bunun yerine en son kesme işleminden önce (y fonksiyonu) tek bir defa olmak üzere, oluşan bütün toplama bölme işlemi yapılabilir. Böylece bir defa bit kaybı olacağından hassasiyet daha iyi olacaktır. Fakat burada çarpma sonucu oluşan sayının sola kaydırılmaması halinde daha büyük bir sayı olacağı unutulmamalıdır. En sona bir tane bölücü konulan tasarımlarda toplayıcılar piksel bit genişliğinden daha büyük bit genişliğine sahip olmak zorundadır. Çarpma işleminde çarpılan sayıların aynı bit genişliğinden olması zorunluluğu yoktur. Burada anlatılan tasarımda örnek olarak 10 bit şablon ve 8 bit piksel genişliği kullanılmıştır. Şablon değerleri, Â şablonunun elemanlarının alabileceği en büyük değer 10 bitle ifade edilen sayının en büyük değeri olacak şekilde ölçeklenmiştir.



Şekil 5.4: Aritmetik işlem yapan birime ilişkin iç yapı.

5.3 Görüntünün Saklanması

Görüntü, özellikle büyük ölçülere ulaşınca, devrede en çok hafıza kullanan kısım olmaktadır. Görüntüyü RAM gibi bellek elamanlarında saklamak uygundur. Fakat kaç tane RAM kullanılacağı ve eğer çok sayıda RAM kullanılırsa görüntünün bu RAM'lere nasıl yerleştirileceği gibi konular tasarımdaki önemli sorunlardandır. Çok sayıda RAM kullanmak bütün hücrelere aynı anda veri girişini sağlayabilmek için, yani gereken yüksek bant genişliğini sağlayabilmek için önemlidir.

Kaynak taraması sürecinde incelenilen tasarımlarda, işlenecek görüntünün, işlemci birimlerinin içinde kaydedicilerde (register) saklandığını belirtmiştik. Tasarımcılar, görüntü parçaları arasındaki komşuluk ilişkilerini kolaylıkla sağlamak ve RAM biriminden daha az okuma - yazma işlemi yapmak için, fazla miktarda kaydediciyi işgal etme pahasına bu yolu seçmişlerdir. Kaydedicilerde saklamanın getirdiği farklı bir boyut da pipe-line ile ilgilidir. Bu konu 4.6 bölümünde işlenmiştir. Önemli olduğu düşünüldüğü için daha sonra tekrar ele alınacaktır.

Görüntünün tamamı kapasite yetmemesi sebebiyle yonga üzerindeki kaydedicilerde saklanamayacağı için, tasarımcılar görüntüyü doğal olarak parça parça işleme yoluna gitmişlerdir. İncelenilen tasarımlarda görüntü basit bir mantıkla düşey parçalara ayrılmış ve işlemci birimlere yüklenmiştir. Yapılan incelemelerden ilk yapılan uygulamalarda tek RAM birimi kullanıldığı çıkarılmıştır. Daha sonraki uygulamalarda [17] bant genişliği önemine binaen çok sayıda RAM kullanıldığı görülmüştür.

İncelenilen tasarımlardaki eksik yönlere bakıldığında görüntü saklama ve RAM'e okuma-yazma işlemleri açısından şunlar göze çarpmaktadır: İşlemci ünitelerine piksellerin yüklenmesi üstte de bahsedildiği gibi çok fazla hafıza elemanı işgal etmektedir. Ayrıca zaten RAM'de saklanılan görüntüyü tekrar kaydedicilerde saklamak fuzuli bir hafıza kullanımı diye düşünülüp, tezde önerilen tasarımda, öncelikli çözüm bulunması gereken sorunlar arasında kabul edilmiştir. Bir tane RAM kullanılması durumunda ise, satırdaki her bir işlemciye tek tek pikseller yüklenmektedir. Aynı sorunun RAM'e yazma için de geçerli olduğu, ayrıca her bir iterasyon adımında bütün bu bahsedilen işlemlerin tekrarlanacağı düşünüldüğünde sorunun ciddiyeti anlaşılacaktır. Bu sorunun ciddiyetini artıran bir diğer faktör ise

işlemci birimi sayısı arttıkça toplam hesaplama işlemi süresi azalmasına karşın RAM'le yapılan işlemlerin sürelerinde işlemci sayısıyla doğru orantılı bir artış olmasıdır.

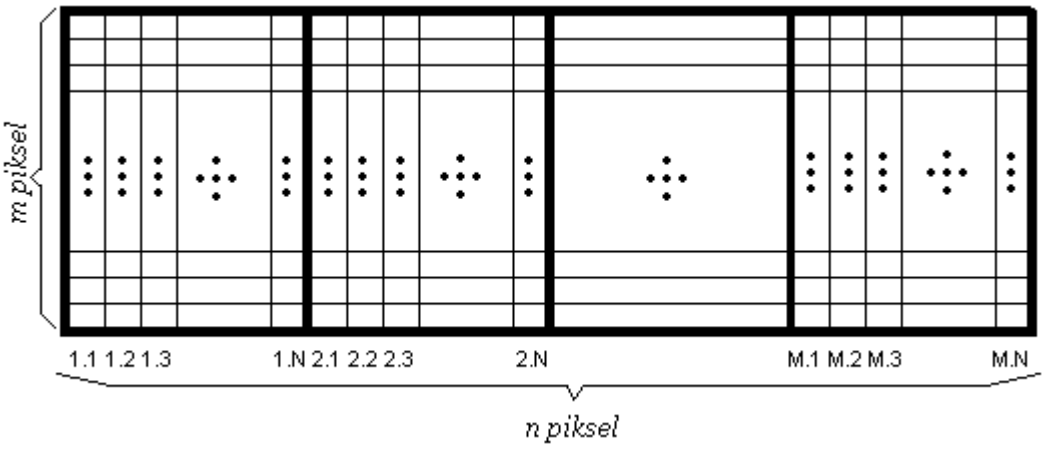
Bu tezde önerilen görüntüyü dilimlere ayırma mantığı, sadece konvolüsyon işlemini yapmak için gereken sayıda kaydedici kullanmayı hedeflemektedir. İşlemci içinde her bir satır için bir kaydedici yetmektedir. Bir komşuluklu şablonlar kullanıldığında konvolüsyon işleminde kullanılmak üzere bir işlemci biriminde pikselleri ve g_{ij} değerlerini saklayan toplam kaydedici sayısı beştir.

Tek RAM kullanmak yerine çok sayıda RAM kullanılması, hücre işlemci sayısı ne olursa olsun aynı sürede (1 saat çevrimi) okuma - yazma yapma imkânı sağlamaktadır. Tam manasıyla paralel çalışan (paralel hesap, okuma - yazma yapan) hücre işlemcilerle görüntü işlenebilmesi bu şekilde mümkün olmaktadır.

Tezde önerilen tasarımda, kaydedici sayısında ciddi azalmaya ve okuma - yazmadaki hıza yönelik yapılan iyileştirmeler kontrol biriminde karmaşıklık meydana getirmektedir.

5.3.1 Görüntünün dilimlere ayrılması

Şekil 5.5'te görüntünün dilimlere ayrılması anlatılmıştır. Buradaki her bir küçük küp bir pikseli temsil etmektedir.



Şekil 5.5: Görüntünün dilimlere ayrılması.

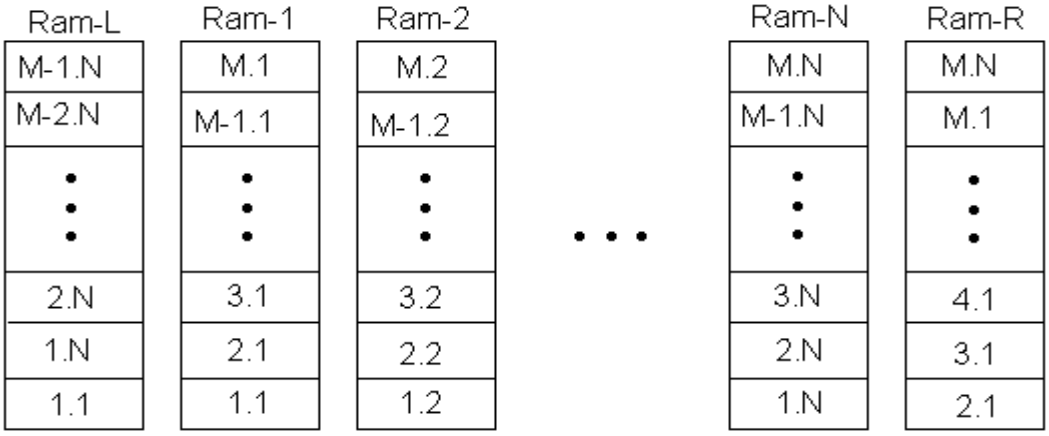
Üstte gösterilen şekilde, her ne kadar *m,* resmin piksel olarak satır sayısı; *n* de bir satırdaki piksel sayısı olsa da, resim dilimlere ayrılırken satırın mı yoksa sütunun mu dilimlere ayrıldığının herhangi bir önemi yoktur. Tek şart, dilimlere ayrılacak boyuttaki piksel sayısının *N* sayısının tam katı olmasıdır. *N,* tasarımdaki hücre işlemci sayısıdır. *M* ise dilim sayısını temsil etmektedir. *M* adet dilimin, tek tek *N* adet hücre işlemci tarafından işleneceği bir tasarım yapılmıştır.

5.3.2 Görüntünün RAM'lere yüklenmesi

Görüntü farklı uygulamalarda farklı donanımlardan alınabilir. Piksellerin alındığı donanım, bir bilgisayar portu veya bir kamera çıkışı olabilir. Bu ve buna benzer çoğu durumda, görüntü işlemenin en başında resim piksel piksel alınabilir. Bu tek tek piksel alma işleminde RAM'ler bir adres çözücü devre vasıtasıyla seçilir ve ilgili pikseller devrenin Şekil 5.1 de görülen, dışarı açılan girişinden alınır.

Şekil 5.6'da dilimlere ayrılmış bir görüntünün RAM'e yerleştirilmesi gösterilmiştir. Buradaki her bir RAM bölümü (örneğin RAM-1'deki 1.1 bölümü) görüntüye ilişkin 1 piksel genişliğindeki bütün bir sütunu saklamaktadır. Bu nedenle her bir RAM bölümü m x (bir pikselin ifade edildiği bit genişliği) boyutundadır.

Yaptığımız tasarımda görüntü belirli bir algoritmaya göre bölünmekte ve belirli bir şekilde işlemcilere yüklenip işlenmektedir. *M*, resmin dilim sayısı olmak üzere; resimler 1. dilimden M. dilime kadar işlenmektedir. İlk dilim en alttaki pikselden en üstteki piksele kadar işlendikten sonra ikinci dilime geçilmektedir ve bütün resim işlenene kadar aynı işlemler tekrarlanmaktadır.



Şekil 5.6: Görüntülerin RAM'e yerleştirilmesi.

Şekil 5.5 ve Şekil 5.6 beraber incelenirse komşu RAM blokları (Ram-R ve Ram-L) hariç diğer RAM bloklarına görüntünün dilimler halinde teker teker yüklendiği görülür. Ram-R ve Ram-L blokları, sırasıyla Ram-N ve Ram-1'in içeriklerinin görüntü üzerindeki komşulukları gözetilerek yüklenmiştir. Şekil 5.6'da yüklenen sütun numaraları verilmiştir.

Resimleri ilk defa yüklemede bir problem yoktur; zor olan kısım ise her bir iterasyon sonucunda RAM içeriklerinin güncellenmesidir. Çünkü iterasyon yapılırken çıkan sonuçları hem eski değerin üstüne hem de komşu RAM bloklarındaki ilgili yerlere yazmak gerekmektedir. İşlemcilere direk bağlı olan RAM'lerdeki değerleri güncellemekte bir zorluk yoktur. İterasyon sonucu çıktığında RAM'lere yazılmaktadır. Öyle ki, bütün ana RAM'leri (komşu RAM olmayanlar) aynı adres ve yazma aktif işaretleriyle kontrol etmek mümkündür. Komşu RAM'lerde durum biraz farklıdır. Farklılıkları şöyle sıralayabiliriz:

Bu RAM'ler için başlangıç, son ve orta durumlar olmak üzere üç durum vardır;

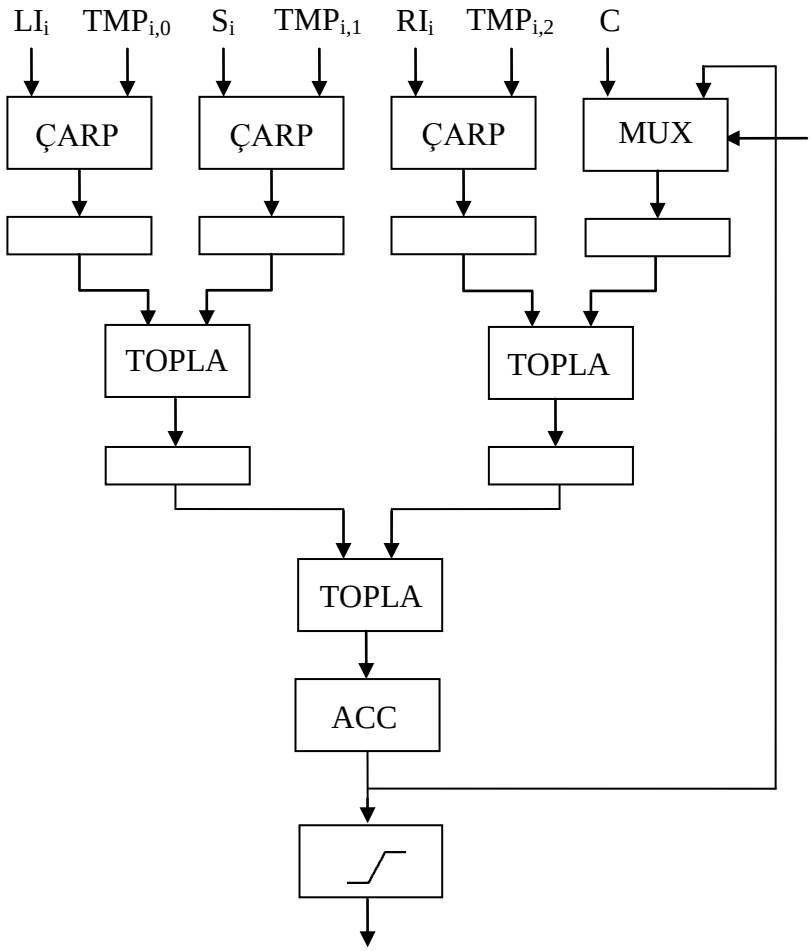
Başlangıç durumunda Ram-L, Ram-1 ile aynı değerdedir. Burada Ram-1 ve Ram-L'nin içeriği 1.1 nolu resim sütunudur. Dolayısıyla Ram1'in girişine gelen işaret, bu kısımda iterasyon yapılırken, Ram-L'nin girişine de bağlanır. Ram-R bu durumda henüz yazma yapmaya başlamaz. Çünkü onun değerleri olan 2.1 nolu resim sütunu henüz iterasyona tabi tutulmamıştır.

En son iterasyon hariç diğer bölümlerin güncellenmesi benzer şekildedir. Burada tek dikkat edilmesi gereken nokta şudur: Ram-R'ye o an iterasyondan çıkan değerler yazılırken, Ram-L'ye yazılan değerler o an iterasyondan çıkan değerler değildir. Dolayısıyla Ram-L'nin girişine Ram-N'in çıkışı bağlanmalıdır ve yazma sırasında ilgili değer Ram-N'in çıkışında olacak şekilde adres bilgisi gönderilmesi gerekecektir.

En son bölümde Ram-R'ye M.1 bölümündeki durumlar yazılmaktadır. İlk bölümde bu RAM'e yazılmadığı hatırlanırsa en son bölüm güncellenirken bir bölümün eksik kaldığı görülecektir. Bu sorun iki şekilde çözülebilir: birincisi, resim komple bir defa iterasyon yapılınca bu bölüm için ayrıca zaman ayrılıp güncellemektir. İkincisi, Ram-R'nin ikinci portundan da veri yazıp zaman kaybetmemektir. Önerilen uygulamada ikincisi yapılmıştır.

5.4 Pipe-Line Yöntemiyle Aritmetik Birimin Hızlandırılması

Bölüm 4.6'da anlatılan devrede pipe-line yönteminin ne kadar faydalı olduğundan bahsedilmişti. Yaklaşık altı kat bir hız artışı olmaktadır [14]. Bu durumda, bu yöntemi önerilen tasarım için de uygulamak mantıklı olacaktır. Yeni önerilen devre diğer devrelerin okuma yöntemlerinden farklı bir yöntemle okuma yaptığı için, aritmetik ünitedeki bloklar arasına koyulan kaydedicilerin etkileri de farklı olacaktır. Şüphesiz, devredeki çarpıcı ve toplayıcı bloklarının da pipe-line ile hızlandırılması devrenin çalışma frekansını artıracaktır.



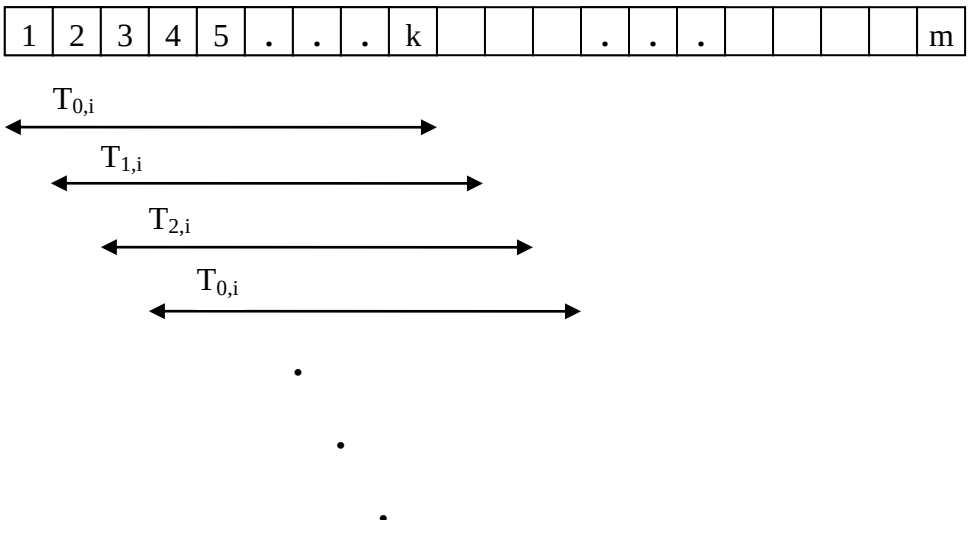
Şekil 5.7: Pipe-line ile hızlandırılmış devre

Şekil 5.7'de içi boş kutular kaydedicileri temsil etmektedir.

Şekil 5.3'te de görülebileceği gibi aritmetik ünitenin frekansını belirleyen blok, içindeki en çok gecikme yapan blok olacaktır. Bu blokların da pipe-line yapılmasıyla devre daha hızlı çalışabilecektir. Önerilen devrenin çarpma ve toplama devrelerinin

pipe-line yapılması bölüm 4.6'da anlatılan pipe-line yapılmış toplayıcı ve çarpıcıyla aynıdır.

$T = \begin{bmatrix} T_{2,2} & T_{2,1} & T_{2,0} \\ T_{1,2} & T_{1,1} & T_{1,0} \\ T_{0,2} & T_{0,1} & T_{0,0} \end{bmatrix}$ herhangi bir şablon, *m* resmin bir sütunundaki piksel sayısı ve k pipe-line seviyesi (ACC, çarpıcı ve toplayıcı blokların içindekiler dahil tüm kaydedici sayısı) olmak üzere, pipe-line yapılırken okuma ve ilgili şablon çarpımı aşağıdaki gibi olacaktır:



Şekil 5.8: Pipe-line yapılmış devrede veri okuma yöntemi

Yukarıda görülen, okla belirtilmiş her adım k tane saat çevriminden oluşmaktadır. İlk k çevrimde piksel değerleri $T_{0,i}$ ile çarpılmaktadır. Aslında *i* yerine 1 yazılabilir; RI ve LI değerleri için de aynı durumun geçerli olduğunu anlatmak ve genelleme yapabilmek için *i* ile indekslemek daha doğrudur. İşlemci içinde kaydedilen pikseller resmin topolojisi göz önüne alındığında şablonun orta kısmına denk gelmektedir. LI sol, dolayısıyla i = 0 indeksli şablon elemanlarıyla çarpılacak; RI sağ, i = 2 indeksli şablon elemanlarıyla çarpılacak. 1'den k'ya kadar olan bütün piksellerin aynı sütun üstünde olduğuna dikkat edilmelidir. Şablonun birinci satırıyla çarpma bittikten sonra, şablonun ikinci satırıyla çarpmak için işlemcilerin içindeki k tane kaydedici bir ötelenir ve yeni değer boşalan yere kaydedilir. Komşu işlemcilerdeki pikseller de ötelendiği için RI ve LI için de aynı durum sağlanmış olur. İlk sonucun çıkması bir sonraki $T_{0,i}$ ile çarpmanın ilk yapıldığı saat çevrimidir. k tane değerin hepsi art arda

çıkacaktır. Çıkan bu sonuçlar RAM'de ilgili yerlere yazılır. Görüldüğü gibi sütundaki piksel sayısının k değerinden büyük olması gerekiyor. Fakat k değeri normal görüntü boyutlarının yanında çok küçük olduğu için herhangi bir sınırlandırma yapmaz; görüntü her halükarda daha büyük olacaktır.

6. SONUÇLAR

Bu tezde bir dijital HSA tasarımının iyi olup olmadığını belirleyen bazı kıstasların olduğu görülmüştür. Bu kıstasların hepsini birden en iyi yapmak mümkün değildir. Bunları ihtiyaçları gözeterek düzenlemek gerekmektedir. Örneğin, hızlı çalışan bir devre gerekmediği, fakat büyük boyutlarda resim işleme gerektiği durumda tasarımda izlenecek yolla, orta boyutta resimleri hızlı işlemek için izlenen yol aynı olmayacaktır. Fakat en genelde bazı kıstaslar vardır ki, bunlar ihtiyaçların birçoğunda geçerlidir. Devrenin az elemanla gerçeklenmesi, paralel çalışabilmesi, çözünürlükteki iyiliği gibi kıstaslar birçok tasarımda dikkat edilmesi gereken hususlardandır.

Tezde anlatılan tasarımın üstteki bilgiler ışığında diğer tasarımlarla karşılaştırılması aşağıdaki gibi yapılabilir:

Bu tezde gerçeklenen hücresel işlemciler, kombinezonsal kısım olarak diğer dijital HSA gerçeklemelerinden çok daha küçük değildir. Çünkü HSA'daki konvolüsyon (nokta çarpım) işlemini gerçekleyen aritmetik birim çok benzer şekilde tasarlanmıştır. Bütün HSA tasarımlarında nokta çarpımın gerçeklenmesi kaçınılmaz olarak gerektiği için aritmetik ünitede fazla bir değişiklik yapmak pek mümkün değildir. Tezde önerilen devre kaydedici kullanımı açısından literatürdeki devrelerden çok daha iyidir. Çizelge 6.1'de son zamanlarda yapılan bir tasarımla [17] tezdeki tasarımın, Xilix XC2V3000 FPGA'sına göre yapılan sentezinin karşılaştırılması gösterilmiştir. Çizelgede gösterilen tasarımda kaydedici dizisi blok-RAM vasıtasıyla yapılmıştır; bundan dolayı, işlemci birimlerinde kullanılan blok-RAM sayılarına dikkat edilmelidir. Bu tezde önerilen devrede blok-RAM birimleri tüm görüntüyü ve durumları kaydetmek için kullanılmıştır; hücresel işlemcilerde blok-RAM kullanımı yoktur. Tüm görütüyü ve durumları saklama bahsedilen tasarımda [17] da vardır (GAPU adı verilen birim içerisinde 19 adet blok-RAM kullanılmaktadır). Çizelge 6.1 incelendiğinde, tezde önerilen hücresel işlemcilerin

referans [17]'deki işlemci birimden kombinezonsal kaynak kullanımı olarak kısmen, kaydedici kullanımı olarak da oldukça avantajlı olduğu görülecektir.

Çizelge 6.1: İşlemci birimlerinin karşılaştırılması.

	Hücresel İşlemciler (XC2V3000)	**Falcon PE [17] (XC2V3000)**
İşgal edilen birim (slice)	295	402
Kullanılan Blok-RAM	-	5
Kullanılan 18 bit çarpıcı	3	3

Pipe-line tekniği kaçınılmaz olarak gerekmektedir. Çünkü tasarlanan aritmetik ünitenin gecikmesi çok fazladır. Temel bloklarla birlikte toplayıcı ve çarpıcılar da pipe-line yapılırsa 6 kat gibi bir hızlanma söz konusudur. Literatürdeki incelenen devrelerde piksellerin hücre içinde dört satır halinde saklandığı durumda pipe-line yapılırsa, yatayda gerçeklenecek hücre sayısı sınırlandırılmış olur. Bu sınırlandırma ikinci satırda hücre yerleştirmeyi getirecektir. Birden çok satırda çalışan hücrelerin hem kontrolü zordur hem de bu hücreler sürekli zamanda paralel çalışmaktadır. Bu hücreler kendilerine veri gelene kadar çalışmamaktadırlar. Bu tezde önerilen devrede yatayda bir sınırlama yoktur. Tasarım yapılan sistem izin verdiği müddetçe yataydaki işlemci sayısı piksel sayısına kadar çıkabilir.

Literatürde incelenen devrede birden çok satırda hücre gerçekleme imkânı vardır. Tezde önerilen devrede böyle bir imkân yoktur. Fakat bir önceki maddede bahsedildiği gibi literatürdeki devrenin yatayda çıkabileceği maksimum hücresel işlemci sayısı sınırı vardır. Tezde önerilen devrede böyle bir sınır olmadığı için eğer gerçekleyecek kaynak (lojik kapı) varsa bu işlemciler yatayda gerçeklenebilirler. Gelişmiş bir FPGA'da bile orta çözünürlükte gerçeklenebilecek hücresel işlemci sayısı 100'ün biraz üzerindedir. 100 civarındaki görüntü boyutu resim göz önüne alındığında çok küçüktür. Dolayısıyla, gerçeklenebilecek hücresel işlemci sayısı genelde resmin bir boyutundaki piksel sayısından az olacağından bizim önerdiğimiz devrede ikinci bir satırda hücresel işlemci tasarlama ihtiyacı olmayacaktır. Ayrıca çok büyük boyutlu tasarımlar göz önüne alınırsa, tezde önerilen devre; bir satırda, resmin bir boyutundaki piksel sayısınca hücresel işlemci gerçeklendiği durumda

birden çok satırda hücresel işlemci gerçeklemeye imkân tanımaktadır. Hatta bu en iyi durumdur, çünkü bu halde kenar koşulları kenara direk olarak eşit alınabilir ve hücreler düşeyde kaskat bağlanabilir.

Çok satırlı gerçeklemede literatürde incelenen devrenin ana kontrol birimi daha karmaşık olacaktır. Çünkü her bir satırdaki işlemcilerin başlama zamanı farklı olacaktır. Üst satırdaki hücrelerden sonuçların çıkıp alttaki hücrelerin üç satırını da doldurması için geçen süre fuzuli bir bekleme oluşturacaktır. İkiden çok satır gerçeklenirse satırlar aşağıya indikçe bekleme süresi iyice artacaktır. Bu maddede açıklandığı gibi, birden çok satırda işlemci gerçeklemede literatürdeki devrede zaman kaybı olduğu görülmüştür.

Hem tezde önerilen devrede hem de literatürde incelenen devrelerde hücrelere aynı anda veri gönderme gerektiği için bant genişliği problemi vardır. Bant genişliği problemi tezde önerilen devrede blok-RAM kullanılarak aşılmıştır. Her bir hücresel işlemciye bir RAM bağlanmış ve tam paralel çalışma sağlanmıştır.

Yapılan çalışmada elde edilen tecrübeler sonucunda şu öneriler verilebilir:

Tezde önerilen tasarım, parçalı hafıza imkânı olduğu durumlarda diğer devrelerden daha az kaynak (lojik kapı) kullanıp aynı hızı (çoklu satırlılara karşı daha yüksek hız) sağlamaktadır.

ASIC tasarımlar düşünüldüğünde kaydedicilerin ve kombinezonsal kısımların hepsinin birden devre alanı kaplayacağı göz önüne alınırsa, bu tezde önerilen devrenin bir işlemci için gereken toplam devre alanı (kaydedici + kombinezonsal) az olduğu için bu tür tasarımlarda tercih edilebilir.

KAYNAKLAR

[1]**Chua, L. O., and Yang, L.,** 1988: Cellular Neural Networks: Theory, *IEEE Transaction on Circuits and Systems*, Vol. **35**, pp. 1257-1272.

[2]**Roska, T., and Chua, L.O.**, 1993: The CNN Universal Machine: An Analogic Array Computer, *IEEE Trans. Circuits and Systems-II*., Vol. **40**, pp. 163-173.

[3]**Keresztes, P., Zarándy, Á., Roska, T., Szolgay, P., Hídvégi, T., Jonás, P. ve Katona, A.,** 1999: An emulated digital CNN implementation, *Int. J. VLSI Signal Processing*, Vol. **23**, pp. 291–303.

[4]**Wen, K.A., Su, J.Y., and Lu ,C.Y.,** 1994: VLSI design of digital Cellular Neural Networks for image processing, *J. of Visual Communication and Image Representation,* Vol. **5**, No.2, pp. 117-126.

[5]**Ikenaga, T. ve Ogura, T.,** 1996: Discrete-time Cellular Neural Networks using highly-parallel 2D Cellular Automata CAM^2 *Proc. of Int. Symp. on Nonlinear Theory and its Applications*, pp. 221-224.

[6]**Doan, M.D., Glesner, M., Chakrabaty, R., Heidenreich, M. ve Cheung, S.,** 1994: Realisation of digital Cellular Neural Network for image processing, *3rd. IEEE International Workshop on Cellular Neural Networks and Their Applications Proceedings (CNNA'94)*, pp. 85-90.

[7]**Chua L.O., and Yang L.**, 1988: Cellular Neural Networks: Applications, *IEEE Transaction on Circuits and Systems*, Vol **35**, No.10, pp. 1273-1290.

[8]**Hopfield, J. J.**, 1982: Neural networks and physical systems with emergent collective computational abilities, *Proceedings of the National Academy of Sciences USA*, Vol. **79**, pp.2554–2558.

[9]**Çelebi E., ve Güzeliş C.,** 1995: Hücresel yapay sinir ağları ile görüntü onarımı, *Sinyal İşleme ve Uygulamaları Kurultayı*, **Cilt-A**, s.37–41.

[10] **Arena, P., Basile, A., Bucolo, M., and Fortuna, L.,** 2003: An Object Oriented Segmentation on Analog CNN Chip, *IEEE Transactions On Circuits And Systems—I: Fundamental Theory And Applications,* Vol. **50**, pp. 837-846.

[11] **Chua L. O., and Roska T**., 1993: The CNN paradigm, *IEEE Trans. Circuits Systems- I,* Vol. **40**, pp. 147–156.

[12] **Örs, S.B**., 1998. Sahada Programlanabilir Kapı Dizileri ile Lojik Devre Tasarımı ve VHDL Kullanılarak Bazı Devrelerin Gerçekleştirilmesi, Yüksek Lisans Tezi, İ.T.Ü. Fen Bilimleri Enstitüsü, İstanbul.

[13] **<url>**www.xilinx.com, alındığı tarih 10.05.2007.

[14] **Hidvégi, T., Keresztes, P., Szolgay, P**., 2002: An Accelerated CNN-UM (CASTLE Architecture by Using the Pipe-line Technique), *7th. IEEE International Workshop on Cellular Neural Networks and Their Applications Proceedings (CNNA'02),* pp. 355-362.

[15] **Hidvégi, T., Keresztes, P., Szolgay, P**., 2003: Enhanced Emulated Digital CNN-UM (CASTLE) Arithmetic Cores, *Journal of Circuits, Systems, and Computers*, Vol. **12**, pp. 711-738.

[16] **Nagy, Z., Szolgay, P**., 2003: Configurable multi-layer CNN-UM emulator on FPGA, *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, Vol. **50**, pp. 774–778.

[17] **Voroshazi, Z., Nagy, Z., Kiss, A., Szolgay, P**., 2006: An embedded CNN-UM Global Analogic Programming Unit implementation on FPGA, *10th. IEEE International Workshop on Cellular Neural Networks and Their Applications Proceedings (CNNA'06),* pp. 1-5.

[18] **Baugh C.R. ve Wooley B.A**., 1973. "A Twos Complement Parallel Array Multiplication Algorithm", *IEEE Trans. Comp*, Vol **C-22**, No.12, pp. 1045-1047.



ÖZGEÇMİŞ

Adı Soyadı: Mehmet TÜKEL

Doğum Yeri ve Tarihi: Nevşehir, 10 Kasım 1983

Adres: İncirköy Mh. Ay Sk. No:18/2 Beykoz/İSTANBUL

Lisans Üniversite: Yıldız Teknik Üniversitesi