Görüntü İşleme Uygulamalarına Yönelik Gerçek Zamanlı İki Katmanlı Bir Hücresel Sinir Ağı Emülatörünün Gerçekleştirilmesi

A Real-Time Two-Layer Cellular Neural Network Emulator Realization for Image Processing Applications

Murathan Alpay[1], *Nerhun Yıldız*[1], *Vedat Tavşanoğlu*[1]

[1]Elektronik ve Haberleşme Mühendisliği Bölümü
Yıldız Teknik Üniversitesi
malpay@yildiz.edu.tr, nerhuny@yildiz.edu.tr, tavsanav@yildiz.edu.tr

Özet

Bu bildiride Full HD 1080p@60 video sinyallerini (1920×1080 çözünürlük ve 60 Hz çerçeve hızı) gerçek zamanlı olarak işleyebilen ve Alanda Programlanabilen Kapı Dizileri (APKD) üzerinde gerçekleştirilmiş olan iki katmanlı bir Ayrık Zamanlı Hücresel Sinir Ağı (AZHSA) emülatörü tasarımı sunulmuş ve örnek bir görüntü işleme uygulamasının benzetim sonuçları verilmiştir. Bu amaçla öncelikle tek katmanlı bir AZHSA gerçeklemesine değinilmiş, sonrasında ise bu yapının genişletilmesiyle iki katmanlı AZHSA yapısının tasarım adımları gösterilmiştir. Son olarak tasarlanan yapının Modelsim benzetim sonuçları ile Matlab benzetimi sonuçları karşılaştırılarak birebir aynı sonuçların elde edildiği gösterilmiştir. Bu yapı Altera Stratix-IV bir APKD kiti üzerinde gerçekleştirilmiştir.

Abstract

In this paper, a two-layer Discrete-Time Cellular Neural Network (DT-CNN) emulator design is revised, which is capable of processing Full HD 1080p@60 video signals (1920×1080 resolution and 60 Hz frame rate), and some simulation results are given. To this end, first, the mathematical overview and structure of a single-layer DT-CNN emulator are given, then the design steps are generalized for a two-layer realization, and finally Modelsim and Matlab simulation results are given and cross-checked. This design is implemented on a high-end Altera Stratix-IV FPGA device.

1. Giriş

Bu bildiri iki katmanlı Gerçek Zamanlı Hücresel Sinir Ağı İşlemcisi (GZHSAİv2) bloğunun tasarımının verildiği bir önceki çalışmanın devamı niteliğindedir [1]. İki katmanlı yapının tasarlanması aşamasında tek katmanlı GZHSAİv2 tasarımı temel alınmıştır [2].

Literatürde çok katmanlı Ayrık Zamanlı Hücresel Sinir Ağı (AZHSA) mimarilerine ait birçok yayın bulunmaktadır [3-13], bunların bir kısmı AZHSA mimarilerinin Alanda Programlanabilen Kapı Dizisi (APKD) birimleri üzerinde gerçekleştirmeleridir [3-7]. Bu APKD gerçekleştirmeleri arasında göze çarpan ve [2]'deki tasarıma zemin oluşturan yapı ise FALCON mimarisi ve türevleridir [3-5]. GZHSAİv2 mimarisi ise bu yapıdan farklı olup daha yüksek çözünürlüklü ve gerçek zamanlı uygulamalara yönelik olarak geliştirilmiştir.

İkinci bölümde GZHSAİv2 yapısının matematiksel modeli ele alınmış ve tasarımı özetlenmiştir. Üçüncü bölümde iki katmanlı GZHSAİv2 yapısının bir önceki çalışmada da yer verilmiş olan matematiksel modeli verilmiş ve iki katmanlı işlemci bloğunun (APU2K) iç blok diyagramı verilerek tasarım adımları anlatılmıştır. Dördüncü bölümde bu yapıya ilişkin örnek bir görüntü işleme uygulamasının hem Matlab programı ile yapılan yazılımsal benzetim sonuçlarına, hem de Modelsim programıyla yapılan VHDL donanım tanımlama dili benzetimi sonuçlarına yer verilmiştir. Son bölüm ise sonuç bölümüdür.

2. Tek Katmanlı AZHSA Yapısının Matematiksel Modeli ve Tasarımı

GZHSAİv2 yapısında kullanılan lineer, konumdan bağımsız ve tek katmanlı AZHSA yapısının matematiksel modeli

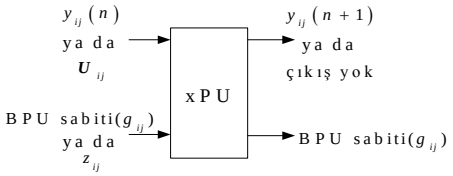
$$x_{ij}^{n+1} = \underbrace{\overline{\boldsymbol{A}} * \boldsymbol{Y}_{ij}^{\,n} + \underbrace{\overline{\boldsymbol{B}} * \boldsymbol{U}_{ij} + \bar{z}_{ij}}_{\text{BPU}}}_{\text{APU}} \tag{1a}$$

$$y_{ij} = f\left(x_{ij}\right) = \frac{1}{2}\left|x_{ij}+1\right| - \frac{1}{2}\left|x_{ij}-1\right| \tag{1b}$$

denklem takımı şeklindedir [2]. Burada x durum değişkeni, i ve j piksel indisleri, n iterasyon indisi, $\boldsymbol{A}$ k-komşuluklu durum şablonu, $\boldsymbol{X}$ şablon ile aynı boydaki durum komşuluk matrisi, $\boldsymbol{Y}$ şablon ile aynı boydaki çıkış komşuluk matrisi, * karşılıklı matris elemanlarının çarpılıp birbirleriyle toplanacağını gösteren şablon nokta çarpımı operatörü, $\boldsymbol{B}$ k-komşuluklu giriş şablonu, $\mathbf{U}$ şablon ile aynı boydaki giriş komşuluk matrisi ve z eşik değeridir.

(1)'i gerçeklemek üzere APU (A işlemci birimi) ve BPU (B işlemci birimi) işlemcileri tasarlanmıştır. Her işlemcinin bir şablon nokta çarpımı ve bir toplama yapma yeteneği vardır. Bir komşuluklu yapıda bu işlem için 3 × 3 = 9 çarpma ve 8 + 1 = 9 toplama işlemi gerekir. BPU'nun hesapladığı değer iterasyon indisine bağlı olmadığından dolayı iterasyonlardan önce her bir piksel için birer kez hesaplanarak iterasyonun sonuna kadar sabit değer olarak kullanılır. APU'lar ise tek bir iterasyon adımını iş hattı mantığı ile işler. Yani kullanılan p.

APU her zaman *p*. iterasyonu gerçekleştirir. Bu durumda *N* iterasyon için *N* tane APU ardı ardına dizilmelidir.
Tasarlanmış olan GZHSAİv2 işlemcisi (xPU), APU veya BPU olarak programlanabilir. xPU'nun kontrol uçları hariç uç blok diyagramı Şekil 2.1'de verilmiştir. Bir xPU'nun temel olarak üç görevi vardır: Verileri depolamak, verileri işlemek ve bir sonraki bloğun kontrol sinyallerini üretmek. Bu işlemci satırsal olarak paketlenmiş olan verileri ve sabitleri giriş olarak alır, işlenmiş verileri de aynı şekilde paketlenmiş olarak dışarı verir.



Şekil 2.1: Tek katmanlı AZHSA için tasarlanmış olan xPU bloğunun basitleştirilmiş uç blok diyagramı.

3. İki Katmanlı AZHSA Yapısının Matematiksel Modeli ve Tasarımı

İki katmanlı, tek katmandan giriş alan, lineer ve konumdan bağımsız bir hücresel sinir ağı

$$x_{ij,1}^{n+1} = \underbrace{\underbrace{\bar{\boldsymbol{A}}_{21} * \boldsymbol{Y}_{ij,2}^{n} + \underbrace{\bar{\boldsymbol{A}}_{11} * \boldsymbol{Y}_{ij,1}^{n} + \underbrace{\bar{\boldsymbol{B}} * \boldsymbol{U}_{ij,1}^{n}}_{\text{BPU}} + \bar{z}_{ij,1}}_{\text{APU}_{1,1}}}_{\text{APU}_{2,1}}}$$

$$x_{ij,2}^{n+1} = \underbrace{\underbrace{\bar{\boldsymbol{A}}_{22} * \boldsymbol{Y}_{ij,2}^{n} + \underbrace{\bar{\boldsymbol{A}}_{12} * \boldsymbol{Y}_{ij,1}^{n} + \bar{z}_{ij,2}}_{\text{APU}_{1,2}}}_{\text{APU}_{2,2}}} \qquad (2)$$

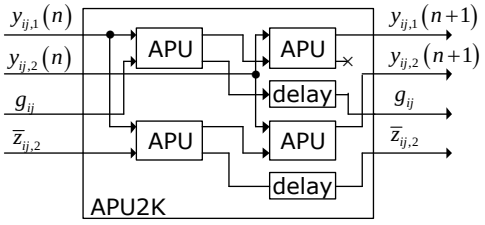
$$y_{ij,l} = f\left(x_{ij,l}\right) = \frac{1}{2}\left|x_{ij,l}+1\right| - \frac{1}{2}\left|x_{ij,l}-1\right| \qquad l = 1,2$$

denklem takımı ile ifade edilebilir. Burada '*ij,1*' ve '*ij,2*' sırasıyla birinci ve ikinci katmanların piksel indisleri, '*l*' ise katman indisidir. (2)'deki denklem takımının hesaplama akışı şu şekilde yazılabilir:

$$\begin{aligned}
&\text{BPU} &&: g_{ij} &&= && \bar{\boldsymbol{B}} * \boldsymbol{U}_{ij} + \bar{z}_{ij,1} \\
&\text{APU}_{11}(1) &&: h_{ij,1}(1) &&= && \bar{\boldsymbol{A}}_{11} * \boldsymbol{Y}_{ij,1}(0) + g_{ij} \\
&\text{APU}_{21}(1) &&: y_{ij,1}(1) &&= f\left(x_{ij,1}(1)\right) = && f\left(\bar{\boldsymbol{A}}_{21} * \boldsymbol{Y}_{ij,2}(0) + h_{ij,1}\right) \\
&\text{APU}_{12}(1) &&: h_{ij,2}(1) &&= && \bar{\boldsymbol{A}}_{12} * \boldsymbol{Y}_{ij,1}(0) + \bar{z}_{ij,2} \\
&\text{APU}_{22}(1) &&: y_{ij,2}(1) &&= f\left(x_{ij,2}(1)\right) = && f\left(\bar{\boldsymbol{A}}_{22} * \boldsymbol{Y}_{ij,2}(0) + h_{ij,2}\right) \\
&\quad\vdots &&\quad : \quad\vdots \\
&\text{APU}_{11}(N) &&: h_{ij,1}(N) &&= && \bar{\boldsymbol{A}}_{11} * \boldsymbol{Y}_{ij,1}(N-1) + g_{ij,1} \\
&\text{APU}_{21}(N) &&: y_{ij,1}(N) &&= f\left(x_{ij,1}(N)\right) = && f\left(\bar{\boldsymbol{A}}_{21} * \boldsymbol{Y}_{ij,2}(N-1) + h_{ij,1}\right) \\
&\text{APU}_{12}(N) &&: h_{ij,2}(N) &&= && \bar{\boldsymbol{A}}_{12} * \boldsymbol{Y}_{ij,1}(N-1) + \bar{z}_{ij,2} \\
&\text{APU}_{22}(N) &&: y_{ij,2}(N) &&= f\left(x_{ij,2}(N)\right) = && f\left(\bar{\boldsymbol{A}}_{22} * \boldsymbol{Y}_{ij,2}(N-1) + h_{ij,2}\right)
\end{aligned} \qquad (3)$$

Burada g_{ij} değeri yine BPU tarafından bir kez hesaplanır ve iterasyon boyunca sabit kalır. Katman sayısı ikiye çıkınca $2 \times 2 = 4$ tane APU kullanmak gerekir. $h_{ij,1}$ ile $h_{ij,2}$ sırasıyla birinci ve ikinci katmanın yeni durumunun ara değerleridir. Ara değerler olması dolayısıyla bu sonuçların sınırlayıcılardan geçirilmemesi gerekmektedir. Dolayısıyla ilk seviyedeki APU'ların sınırlayıcıları devre dışı bırakılmıştır.

(3)'teki her bir iterasyon için yazılmış olan dört APU işlemi tek bir APU2K işlemcisi ile gerçeklenebilir. İlgili bloğun iç blok diyagramı Şekil 3.1'de verilmiştir.



Şekil 3.1: İlk tasarlanan APU2K bloğunun basitleştirilmiş iç blok diyagramı

Şekil 3.1'de tasarlanan sistemde ilk kattaki APU'ların sabit çıkışından gelen veriler kırpılmadıkları için bit genişlikleri fazladır. Bu durumda ikinci seviyedeki APU'ların sabit RAM ihtiyacı artmaktadır. Daha iyi bir yaklaşım (2)'deki gibi bir işlemci ayrıştırması yerine

$$y_{ij,1}(n+1) = \underbrace{f\left(\underbrace{\bar{\boldsymbol{A}}_{21} * \boldsymbol{Y}_{ij,2}(n)}_{\text{APU (2,1)}} + \underbrace{\bar{\boldsymbol{A}}_{11} * \boldsymbol{Y}_{ij,1}(n) + \overbrace{\bar{\boldsymbol{B}} * \boldsymbol{U}_{ij} + \bar{z}_{ij,1}}^{\text{BPU}}}_{\text{APU (1,1)}}\right)}_{\text{Toplama ve Kırpma}}$$

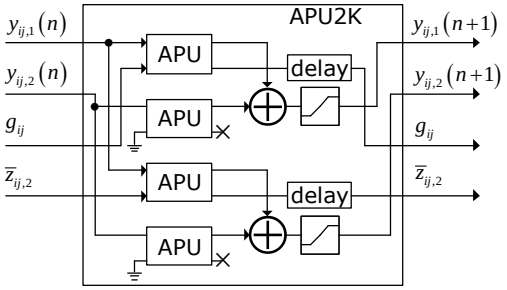
$$y_{ij,2}(n+1) = \underbrace{f\left(\underbrace{\bar{\boldsymbol{A}}_{22} * \boldsymbol{Y}_{ij,2}(n)}_{\text{APU (2,2)}} + \underbrace{\bar{\boldsymbol{A}}_{12} * \boldsymbol{Y}_{ij,1}(n) + \bar{z}_{ij,2}}_{\text{APU (1,2)}}\right)}_{\text{Toplama ve Kırpma}} \qquad (4)$$

şeklinde bir ayrıştırma yapmaktır. Böylece hem tüm APU'lar aynı anda işlem yaptığından toplam gecikme azalır, hem de RAM kullanımı optimize edilmiş olur. (4)'ten türetilen hesaplama akışı şu şekildedir:

$$\begin{aligned}
&\text{BPU} &&: g_{ij} &&= && \bar{\boldsymbol{B}} * \boldsymbol{U}_{ij} + \bar{z}_{ij,1} \\
&\text{APU}_{11}(1) &&: h_{ij,1}(1) &&= && \bar{\boldsymbol{A}}_{11} * \boldsymbol{Y}_{ij,1}(0) + g_{ij} \\
&\text{APU}_{21}(1) &&: l_{ij,1}(1) &&= && \bar{\boldsymbol{A}}_{21} * \boldsymbol{Y}_{ij,2}(0) \\
& && && y_{ij,1}(1) &= f\left(x_{ij,1}(1)\right) = && f\left(h_{ij,1}(1) + l_{ij,1}(1)\right) \\
&\text{APU}_{12}(1) &&: h_{ij,2}(1) &&= && \bar{\boldsymbol{A}}_{12} * \boldsymbol{Y}_{ij,1}(0) + \bar{z}_{ij,2} \\
&\text{APU}_{22}(1) &&: l_{ij,2}(1) &&= && \bar{\boldsymbol{A}}_{22} * \boldsymbol{Y}_{ij,2}(0) \\
& && && y_{ij,2}(1) &= f\left(x_{ij,2}(1)\right) = && f\left(h_{ij,2}(1) + l_{ij,2}(1)\right) \\
&\quad\vdots &&\quad\vdots &&\quad\vdots &&\quad\vdots \\
&\text{APU}_{11}(N) &&: h_{ij,1}(N) &&= && \bar{\boldsymbol{A}}_{11} * \boldsymbol{Y}_{ij,1}(N-1) + g_{ij} \\
&\text{APU}_{21}(N) &&: l_{ij,1}(N) &&= && \bar{\boldsymbol{A}}_{21} * \boldsymbol{Y}_{ij,2}(N-1) \\
& && && y_{ij,1}(N) &= f\left(x_{ij,1}(N)\right) = && f\left(h_{ij,1}(N) + l_{ij,1}(N)\right) \\
&\text{APU}_{12}(N) &&: h_{ij,2}(N) &&= && \bar{\boldsymbol{A}}_{12} * \boldsymbol{Y}_{ij,1}(N-1) + \bar{z}_{ij,2} \\
&\text{APU}_{22}(N) &&: l_{ij,2}(N) &&= && \bar{\boldsymbol{A}}_{22} * \boldsymbol{Y}_{ij,2}(N-1) \\
& && && y_{ij,2}(N) &= f\left(x_{ij,2}(N)\right) = && f\left(h_{ij,2}(N) + l_{ij,2}(N)\right)
\end{aligned} \qquad (5)$$

Yeni APU2K'nın iç yapısı APU'lar cinsinden Şekil 3.2'deki şekilde tasarlanmıştır. Buradaki gecikmeler satır gecikmeleri değil, yalnızca toplayıcı ve sınırlayıcı gecikmeleridir. Böylece

APU2K işlemcisinin hem RAM kullanımı, hem de giriş ile çıkış arasındaki gecikmesi azaltılmıştır.



Şekil 3.2: İkinci tasarlanan APU2K bloğunun basitleştirilmiş iç blok diyagramı (Kontrol uçları hariç)

APU'ların iş hatları birbirine paraleldir, dolayısıyla aynı kontrol sinyalleriyle kontrol edilebilirler. APU'ların orjinal kontrol sinyalleri; toplayıcı, sınırlayıcı ve geciktirici birimlerinin neden olduğu ek gecikmeler de göz önüne alınarak yeniden düzenlenmiştir.

Şekil 3.2'de verilmiş olan işlemci tek bir iterasyonu gerçekleştirdiğinden dolayı bu bloklardan istenen iterasyon sayısı kadarı birbiri ile ardı ardına kaskad bağlanarak AZHSA emülasyonu gerçekleştirilir.

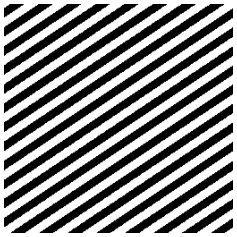
Bu yapıda ilk bloktan son bloğa kadar iletilecek olan iki tür sabit değer bulunmaktadır. Bunlardan ilki BPU tarafından $\boldsymbol{U_{ij}}$ girişi ile $\overline{z}_{ij,1}$ eşiğinden üretilen g_{ij}, ikincisi ise $\overline{z}_{ij,2}$ eşiğinin kendisidir. Bu sabitler tek katmanlı HSA'da olduğu gibi her çerçeve için bir kez hesaplanır ve iterasyonlar (APU2K blokları) boyunca değişikliğe uğramadan iletilir.

4. Benzetim Çalışmaları

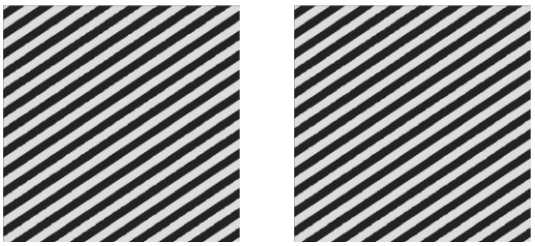
Tasarlanan yapının test edilmesi için iki katmanlı AZHSA ile gerçeklenebilen Gabor süzgeçleri seçilmiştir. Test için öncelikle Matlab programında özgün bir m-dosyası hazırlanarak Gabor-benzeri süzgeç benzetimi yapılmıştır. Matlab ile yapılan tüm benzetimlerde yapının VHDL kodlarında kullanılan sabit noktalı aritmetik kullanılmıştır. Ancak [1]'deki çalışmada kullanılan sabit-noktalı aritmetik araç kutusu işlem hızını çok yavaşlattığından dolayı yerine aynı işi 'double' türündeki sayılarla hızlı bir şekilde yapan uygulamaya yönelik Matlab kodları yazılmıştır.

Benzetimler 64-bitlik işletim sistemine sahip, 2.63GHz hızında çalışan AMD Phenom II X3 710 işlemcisi ve 4 GB RAM'i olan bir bilgisayar üzerinde yapılmıştır. Burada benzetim hızını arttırmak için giriş olarak 256x256'lık çözünürlüğe sahip bir görüntü seçilmiştir (Şekil 4.1). Yapılan Matlab ve Modelsim benzetimlerinin sonuçları üç farklı iterasyon sayısı için Şekil 4.2, Şekil 4.3 ve Şekil 4.4'te verilmiştir. Bu örnek uygulamada verilen giriş görüntüsünde yer alan iki boyutlu kare dalganın üçüncü harmoniğinin Gabor benzeri süzgeç ile süzüldüğü görülmektedir. Bu benzetimler sonucunda elde edilen Matlab ve Modelsim sonuçlarının birebir örtüştüğü gözlenmiştir.

Benzetim çalışmaları sadece sonuçları verilen Gabor benzeri süzgeç için değil, şablon katsayılarına ve eşiğe rastgele değerler verilerek elde edilmiş deneysel AZHSA örnekleriyle de test edilmiştir. Yapılan bu ek testler sonucunda da elde edilen Matlab ve Modelsim simülasyonu sonuçlarının bire bir örtüştüğü gözlenmiştir.

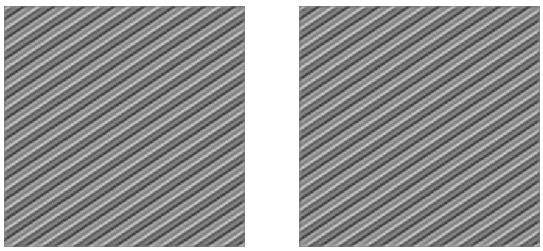


Şekil 4.1: Giriş görüntüsü



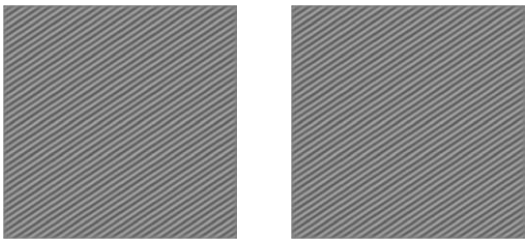
(a) (b)

Şekil 4.2: İlk katmanın 1. iterasyon sonucu: (a) Matlab çıktısı (b) Modelsim çıktısı



(a) (b)

Şekil 4.3: İlk katmanın 10. iterasyon sonucu: (a) Matlab çıktısı (b) Modelsim çıktısı



(a) (b)

Şekil 4.4: İlk katmanın 30. iterasyon sonucu: (a) Matlab çıktısı (b) Modelsim çıktısı

5. Sonuçlar ve Hedeflenen Çalışmalar

Bu bildiride yüksek çözünürlükteki hareketli ve sabit görüntüleri işleyebilen iki katmanlı bir AZHSA emülatörünün matematiksel modeli, tasarımı, gerçeklemesi ve benzetim sonuçları verilmiştir. Buna ek olarak ilgili sistem Altera Stratix-IV modeli bir APKD kiti üzerinde çalıştırılmıştır. Bu düzenekte kullanılan giriş görüntüsü Full HD 1080p@60 bir video işaretidir (1920×1080 çözünürlük, 60 Hz çerçeve hızı, 124.4 MPix/s aktif piksel hızı) ve sistem gerçek zamanlı olarak çalıştırılmıştır.

Bu kapsamda yapılan diğer bir çalışma da bu bildiride sunulan çalışmanın genişletilmesi ile ikiden çok katmanlı HSA'ların da yüksek hızlı ve gerçek zamanlı emülasyonunun yapılmasıdır. İlgili çalışmanın ayrıntılarının ileriki tarihlerde yayınlanması hedeflenmektedir.

6. Teşekkür

Bu çalışma Türkiye Bilimsel ve Teknolojik Araştırmalar Kurumu (TÜBİTAK) tarafından 108E023 numaralı proje kapsamında desteklenmiştir.

7. Kaynaklar

[1] Alpay M., Yıldız N., Tavsanoglu V., Alanda Programlanabilen Kapı Dizileri ile İki Katmanlı Hücresel Sinir Ağı Gerçeklemesi, *Akıllı Sistemlerde Yenilikler ve Uygulamaları Sempozyumu(ASYU)*, pp. 196-199, (2010)

[2] Yıldız N., Cesur E. ve Tavsanoglu V., "A New Control Structure For The Pipelined CNN Processor Arrays" *Proc. 2010 IEEE International Workshop on Cellular Neural Networks and their Applications (CNNA'10)*, San Francisco, ABD, 2-5 Şubat 2010.

[3] Nagy Z., Implementation of Emulated Digital CNN-UM Architecture on Programmable Logic Devices and its Applications, Ph. D. Thesis, Information Science Ph. D. School, University of Pannonia, Department of Image Processing and Neurocomputing, 2007.

[4] Nagy Z. ve Szolgay P., "Configurable Multilayer CNN-UM Emulator on FPGA", *IEEE Transactions On Circuits and Systems—I: Fundamental Theory and Applications*, Haziran 2003, Vol. 50, No. 6.

[5] Sonkoly P., Kozma P., Nagy Z. ve Szolgay P., "Acoustic wave propagation modeling on 3D CNN-UM architecture", *10th International Workshop on Cellular Neural Networks and Their Applications*, İstanbul, Türkiye, 28-30 Ağustos 2006.

[6] Javier Martinez-Alvarez J., Javier Toledo-Moreo F. ve Manuel Ferrandez-Vicente J., "Discrete-Time Cellular Neural Networks in FPGA", *International Symposium on Field-Programmable Custom Computing Machines*, 2007.

[7] Pardo F., Lopez P., ve Cabello D., "DT-CNN emulator: 3D heat equation solver with applications on the non-destructive soil inspection", *Proc. 2008 IEEE International Workshop on Cellular Neural Networks and their Applications (CNNA'08)*,Santiago de Compostela, İspanya, 14-16 Temmuz 2008.

[8] Hidvégi T., Keresztes P., Szolgay P., "An Accelerated CNN-UM (CASTLE) Architecture by using the Pipe-Line Technique" *7th. IEEE International Workshop on Cellular Neural Networks and Their Applications Proceedings,* 2002

[9] Hídvégi, T., Keresztes P., Szolgay P., "Enhanced Emulated Digital CNN-UM (CASTLE) Arithmetic Cores", *Journal of Circuits Systems and Computers*, 2003, Vol. 12, s.711-738

[10] Karahaliloğlu K. ve Balkır S., "Bio-Inspired Compact Cell Circuit for Reaction–Diffusion Systems", *IEEE Transactions on Circuits and Systems—II: Express Briefs*, Eylül 2005, Vol. 52, No. 9.

[11] Shi B. E., "Cortically-inspired Visual Processing with a Four Layer Cellular Neural Network", *Proceedings of the International Joint Conference on Neural Networks*, 20-24 Temmuz 2003, s.1506 - 1511 vol.2.

[12] Shi B. E., "An eight layer cellular neural network for spatio-temporal image filtering", *International Journal of Circuit Theory And Applications*, 2006, s.141-164.

[13] Boroushaki M, Ghofrani M. B. ve Lucas C., "Simulation of Nuclear Reactor Core Kinetics Using Multilayer 3-D Cellular Neural Networks", *IEEE Transactions on Nuclear Science*, Haziran 2005, vol. 52, no.3.