DOKTORI (PhD) ÉRTEKEZÉS

KINCSES ZOLTÁN

Veszprém

2012

Pannon Egyetem
Informatikai Tudományok Doktori Iskola

# CNN-alapú képfeldolgozó és adaptív optikai rendszer FPGA-s implementációi

DOKTORI (PhD) ÉRTEKEZÉS

Kincses Zoltán

Témavezető:
Dr. Szolgay Péter

Veszprém

2012

**CNN-ALAPÚ KÉPFELDOLGOZÓ ÉS ADAPTÍV OPTIKAI RENDSZER FPGA-S IMPLEMENTÁCIÓI**

Értekezés doktori (PhD) fokozat elnyerése érdekében
a Pannon Egyetem Informatikai Tudományok
Doktori Iskolájához tartozóan.

Írta:
Kincses Zoltán

Témavezető: Dr. Szolgay Péter

Elfogadásra javaslom (igen / nem)

(aláírás)

A jelölt a doktori szigorlaton .....%-ot ért el,

Az értekezést bírálóként elfogadásra javaslom:

Bíráló neve: …..... …..... igen /nem

…….
(aláírás)

Bíráló neve: …..... …..... igen /nem

…….
(aláírás)

A jelölt az értekezés nyilvános vitáján ….....%-ot ért el.

Veszprém/Keszthely, …….
a Bíráló Bizottság elnöke

A doktori (PhD) oklevél minősítése….....

……
Az EDHT elnöke

# Köszönetnyilvánítás

Mindenekelőtt meg szeretném köszönni szüleimnek, nagyszüleimnek, és testvéremnek a PhD tanulmányaim alatt nyújtott folyamatos támogatásukat, mely nélkülözhetetlen volt a feladataim sikeres teljesítéséhez.

Szeretném megköszönni témavezetőmnek, ***Professzor Szolgay Péternek*** is a folyamatos irányítását, szakmai támogatását és segítségét, mely nélkül jelen dolgozat nem készülhetett volna el.

Emellett szeretném még megköszönni ***Dr. Nagy Zoltánnak*** is az állandó segítőkészségét, hasznos tanácsait, és azt hogy bevezettet a magas szintű digitális tervezés rejtelmeibe.

Szeretném megköszönni továbbá a korábbi és jelenlegi munkatársaimnak is (***Dr. Vörösházi Zsolt, Sonkoly Péter, Kocsárdi Sándor, Bokrossy-Csiba Mária, Dr. Pletl Szilveszter, Dr. Gingl Zoltán***) a segítségüket, és megértésüket, mellyel támogattak a tanulmányaim és a dolgozatom elkészítésé ideje alatt.

Végül, de nem utolsósorban szeretném megköszönni feleségemnek, ***Beának*** is a türelmét, a folyamatos támogatását, mely nagyban hozzájárult, hogy ezen dolgozat elkészülhessen.

# Tartalomjegyzék

# Kivonat


## CNN-alapú képfeldolgozó és adaptív optikai rendszer FPGA-s implementációi

A Celluláris Neurális Hálózatok (Cellular Neural Network, CNN) hatékonyságát a kémia/fizika vagy a képfeldolgozás területén már számos tanulmány bizonyította. Ezeken a területeken vannak azonban olyan problémák, mint például a Navier-Stokes, az Euler parciális differenciál egyenletek, vagy a blokk alapú mozgásbecslésnél használt Abszolút Különbségek Összege (Sum of Absolute Differences, SAD) operátor, melyek CNN alapú megoldásához csak nemlineáris template-ek alkalmazhatóak. Ezen template-ek alkalmazására eddig csak a CNN szoftveres szimulációja esetében volt lehetőség, ami azonban igen lassú még optimalizált kód esetében is.

A SAD operátort nem csak a képfeldolgozásban, hanem nagy felbontású képek készítésére, vagy az emberi látás javítására használt Adaptív Optikai (AO) rendszerek egyik fő komponensében, a hullámfront szenzor megvalósításához is alkalmazzák.

A disszertáció első felében egy nemlineáris template-eket is kezelni képes FPGA (Field Programmable Gate Array) alapú emulált digitális CNN architektúrára ad javaslatot a szerző. Az architektúra esetében állítható az állapot, a template, a bias értékek pontossága és az alkalmazható template-ek mérete. A disszertáció részletesen elemzi az architektúra felület-teljesítmény paramétereit. Az értekezés második felében bemutatásra kerül az MTA SZTAKI-ban kifejlesztett FPGA alapú AO rendszer, mely a Hartmann-Shack (HS) hullámfront szenzort alkalmazza. A szerző javaslatot tesz két, a HS hullámfront szenzor SAD alapú megvalósítását biztosító architektúrára is. Az architektúrák minden fontos paraméterükben konfigurálhatók, mely paraméterek beállításának hatásai az értekezésben részletesen bemutatásra kerülnek.

# Abstract


## Implementations of CNN-based image processing and adaptive optic system on FPGA

Several studies have already proved the effectiveness of Cellular Neural Networks (CNN) in the field of chemistry/physics and the image processing. However, there are several problems in these fields such as the Navier-Stokes, Euler partial differential equations, or the Sum of Absolute Differences (SAD) operator which is used in the block-based motion estimation, where the CNN based solution requires nonlinear templates. The only way to use these nonlinear templates is the software simulation of the CNN, which is very slow even if it is optimized.

The SAD operator is not only used in the field of image processing, but in the implementation of wavefront sensor part of the Adaptive Optic (AO) system, which is used to acquire high resolution images or improve human vision.

In the first part of the dissertation, the author makes a proposal for an FPGA (Field Programmable Gate Array) based emulated digital CNN architecture which can handle the nonlinear templates. The precision of the state, template, and bias values and the size of the applied template are fully configurable. In the dissertation, area-performance parameters of the architecture are analyzed in detail . In the second part of the dissertation an FPGA-based AO system using the Hartman-Shack (HS) wavefront sensor is introduced, which was developed in the HAS (Hungarian Academy of Science). The author makes a proposal for two SAD based wavefront sensor architecture. All of the important parameters of the architectures are fully configurable, and the impacts of the different settings of these parameters are shown in detail.

# Выписка


**CNN на основе обработки изображений и адаптивной оптической системы и FPGA реализаций**

Эффективность Клеточной Нейронной Сети, (Cellular Neural Network, CNN) химии/физики и обработки изображения было продемонстрено во многих исследованиях. В этих районах, однако, есть такие проблемы, как Navier-Stokes, Euler ПДЕ (Partial Differential Equation, PDE), или блок на основе ощушения движения использованной Суммы Абсолютных Разностей (Sum of Absolute Differences, SAD) оператор. С CNN на основе решения могут быть использованы не линеальных темплетов. Эти темплеты до сих пор было возможно использровать только для моделирования программного обеспечения CNN, которое очень медленно даже при оптимизированого кода.

SAD оператор используется не только для обработки изображений, а также изображений с высоким разрешением для изготовления или ремонта человеческого зрения Адаптивной Оптики (Adaptive Optic, AO) по и в однум из основных компонентов реализации датчика волнового фронта.

В первой половины тезис автор дает советы нелинейных темплетов может управлять FPGA (Field Programmable Gate Array)-основано на эмулированную цифровную CNN архитектуру. В архитектуре может быть установлено состояние, темплет, бидас размера темплетов. Тезис представляет детальное тестирование архитектуры. Во второй половине диссертации представлены MTA SZTAKI, которие разработаны на основе FPGA AO-системы, которая использует Hartmann-Shack (HS) датчик волнового фронта. Автор предлагает два HS датчик волнового фронта SAD на основе реализации архитектуры. Архитектуры всех важных параметрах могут регулированими.

# Bevezetés

A Celluláris Neurális Hálózatok (Cellular Neural Network, CNN) elméletét 1988-ban publikálták [6], és azóta már számos területen bizonyították a CNN hatékonyságát. Ilyen területek voltak például a kémia/fizika vagy a képfeldolgozás. A kémia és fizika komplex tér-idő dinamikával rendelkező effektusai között fontos szerepük van azoknak a feladatoknak, ahol a kimenet- bemenet- és az állapotváltozók kapcsolata nemlineáris. Ilyen feladatok például a fizika területéről a viszkózus vagy nem viszkózus folyadékok áramlását leíró Navier-Stokes vagy Euler parciális differenciál egyenletek. A parciális differenciálegyenletek megoldásában már számos tanulmány bizonyította a CNN hálózatok használatának létjogosultságát, azonban a fentiekhez hasonló parciális differenciálegyenletek megoldásához nemlineáris template-ekre lenne szükség. Ezen problémák mellett a képfeldolgozás területén is számos olyan feladat van, melynek CNN alapú megoldásához nemlineáris template-eket fejlesztettek ki. Ilyen feladatok például a Medián szűrés, a szürkeskálás erózió és dilatáció, vagy pedig a blokk alapú mozgásbecslés esetében használt Abszolút Differenciák Összege (Sum of Absolute Differences, SAD) operátor.

Természetesen kifejlesztettek már olyan módszereket, melyek segítségével lehetőség nyílik a nemlineáris template-ek helyettesítésére, ezek azonban csak olyan esetekben alkalmazhatók, ahol a feladat megoldásához nemlineáris B template-eket kell használni [29]. Ezeket a nemlineáris B template-eket ugyanis helyettesíteni lehet lineáris template-ek sorozatával. A nemlineáris A vagy D típusú template-eket tartalmazó CNN eljárások esetében azonban nincs más út, csak a nemlineáris template-ek alkalmazása.

Napjainkban a CNN-nek négy különböző implementációja van, ezek az analóg VLSI (Very Large Scale Integration) megvalósítás [13-17], az emulált digitális megvalósítás [9][11], az optikai megvalósítás [18-22] és a szoftveres szimuláció [24-26]. Az analóg VLSI megvalósítás ugyan nagy számítási teljesítménnyel rendelkezik (8-9 TerraOP/mp), azonban számítási pontossága csupán 7-8 bit, érzékeny a feszültség és hőmérsékletváltozásra, továbbá csak lineáris template-ek használhatók. Az emulált digitális VLSI megvalósítás esetén megkülönböztetjük az ASIC (Application Specific Integrated Circuit) és az FPGA (Field Programmable Gate Array) alapú implementációkat. Mindkét megvalósítás hátránya, hogy lassabb, mint az analóg implementáció, azonban flexibilisebbek, könnyebben konfigurálhatók, rövidebb a

tervezési ciklusidejük, felületre optimalizáltak, azonban ezekkel a megoldásokkal is csak lineáris template-ek futtathatók. Az optikai megvalósítás nagy előnye, hogy rendkívül gyors és nagyfelbontású képek feldolgozását is támogatja, azonban a visszacsatolás megvalósítása nehéz, nagy szilícium felületet igényel és itt is csak lineáris template-ek használhatók. A legflexibilisebben és legegyszerűbben megvalósítható CNN implementáció a szoftveres szimuláció, amelynek nagy előnye a lebegőpontos számítás, továbbá a nemlineáris templete-ek használata. Hátránya azonban, hogy jelenleg az asztali számítógépek esetén elterjedt 2, 3, vagy 4 magos processzorokat használva, lényegesen lassabb megoldást nyújt - még optimalizált kód esetén is -, mint a másik három megvalósítás bármelyike. A szoftveres szimuláción belül egyre inkább előtérbe kerülnek a sokprocesszoros tömbszámítógépek, amelyek már lényegesen gyorsabb megoldást nyújtanak, mint az asztali számítógépek esetén használt processzorok. Ilyen processzor például az IBM, Sony, Toshiba cégek együttműködéséből született Cell Broadband Engine (CBEA, röviden Cell), melyet a Sony PlayStation 3 számára terveztek. Az IBM Blade Center QS20 rendszer két, egymással egy szélessávú interface-en keresztül kommunikáló Cell processzorból épül fel, melynek maximális számítási teljesítménye 400 GFLOPS (Floating-point Operation Per Second). Egy Blade Center házban hét darab QS20 kapcsolható össze, mellyel így elérhető a 2.8 TFLOPS számítási teljesítmény is. Az IBM Blade Center legújabb generációjával a QS22-vel pedig már a 6.4 TFLOPS egyszeres és a 3 TFLOPS dupla pontosságú számítási teljesítmény is elérhető.

Egy jó alternatívát kínál az FPGA alapú emulált digitális CNN megvalósítás, mely kellő flexibilitással rendelkezik egy olyan processzor implementálásához, mely képes nemlineáris template-ek kezelésére. Ehhez azonban pontosan ismerni kell a képfeldolgozásban és a parciális differenciál egyenletek CNN alapú megoldása során használt nemlineáris template-eket, hogy milyen elvárásoknak is kell megfelelnie az új architektúrának.

A fentiekben a képfeldolgozás területéről megemlített SAD operátor nemcsak ezen a területen, hanem a csillagászatban, a hadiiparban, és az orvostudomány területén, a szemészetben is fontos szerepet játszik. Ezeken a területeken a nagyobb felbontású képek létrehozására, vagy az emberi látás javítására gyakran alkalmazzák az AO (Adaptív Optikai) rendszereket. Az AO rendszerek dinamikusan képesek kompenzálni a turbulens közeg által torzított fényhullám hullámfrontját egy deformálható tükör (Micro-Electro-Mechanical Systems (MEMs), [53]), vagy egyéb aktuátor eszköz

segítségével [54], a hullámfront szenzor által mért hullámfront torzulási adatok alapján. Ennek köszönhetően az AO rendszerek alkalmazásával jelentősen javítható egy képfeldolgozó rendszer teljesítménye. Az adaptív optikai rendszerek egyik szerves része tehát a hullámfront szenzor, mely segítségével meghatározhatók a hullámfront torzulásai. Egy ilyen hullámfront szenzor megvalósítás a HS (Hartmann-Shack) szenzor, mely esetében a képalkotó szenzor elé egy lencsetömböt, úgynevezett lenslet tömböt helyeznek. A lenslet tömb sok miniatűr lencséből épül fel, melyek mindegyike létrehoz egy miniatűr képet a forrás objektumról, azaz szétbontja az apertúrát úgynevezett szub-apertúrákra. A szub-apertúrák képeit egy képalkotó szenzor segítségével detektáljuk. Az így detektált képek eltolódása a referencia pozíciótól (a torzulásmentes hullámfront esetén meghatározott pozíció) meghatározza a hullámfront lokális görbületeit az éppen aktuális szub-apertúra pozíciójában. Ezt az eltolódást, azaz a két kép optimális illeszkedési pozícióját meghatározhatjuk a SAD operátor segítségével. A SAD operátor két kép megfelelő képpontjainak abszolút különbségének összegét határozza meg, ezzel definiálva egy metrikát a képek illeszkedési pozíciójának meghatározásához. Első lépésben a kép SAD értékeit számoljuk ki a keresési ablakban (ahol az optimális illeszkedést keressük), ezután pedig meghatározzuk ezen értékek minimumát. Az MTA-SZTAKI-ban kifejlesztésre került egy FPGA alapú adaptív optikai kártya, mely a fentiekben bemutatott HS hullámfront szenzort alkalmazza a hullámfront torzulásainak meghatározásához.

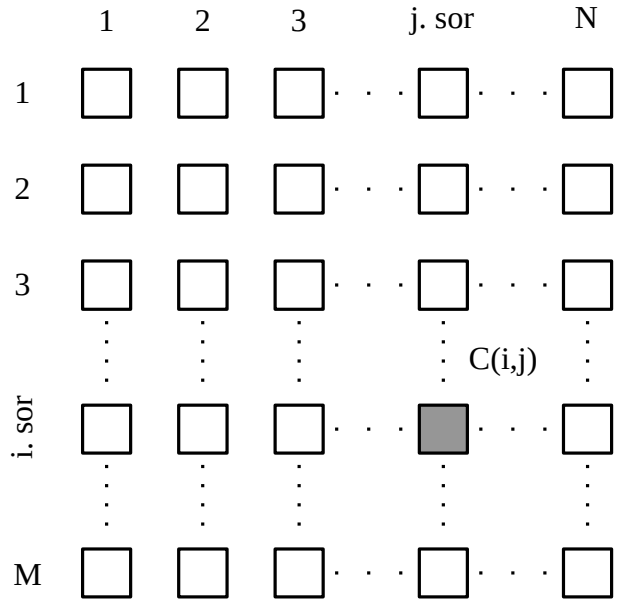
A dolgozatom 1. fejezetében bemutatom a Celluláris Neurális Hálózatokat, majd a 2. fejezetben az általam használt FPGA újraprogramozható áramköröket. A 3. fejezetben bemutatom a szóba jöhető nemlineáris template-eket, és egy olyan emulált digitális CNN architektúrára teszek javaslatot, mely képes kezelni ezeket a nemlineáris template-eket. A 4. fejezetben bemutatásra kerül egy, az MTA SZTAKI-ban kifejlesztett FPGA alapú adaptív optika rendszer, és javaslatot teszek két különböző SAD alapú hullámfront szenzor implementációra is.

# 1. A Celluláris Neurális Hálózatok és implementációi

A Celluláris Neurális Hálózatok elméletét 1988-ban publikálta L. O. Chua és L. Yang [6], melyet az óta már számos feladat megoldására felhasználtak. Ilyen feladat típusok voltak a különböző tér-idő dinamikák számítása, vagy képfeldolgozási feladatok megoldása.

## *1.1. A CNN hálózatok elmélete*

A CNN egy $M \times N$ -es téglalap alakú rács rácspontjaiban elhelyezkedő teljesen azonos, analóg módon működő $C(i, j)$ ($i = 1,2,...,M$ és $j = 1,2,...,N$) processzáló elemekből, úgynevezett cellákból épül fel, ahogy ez az 1.1. ábrán is látható [4].

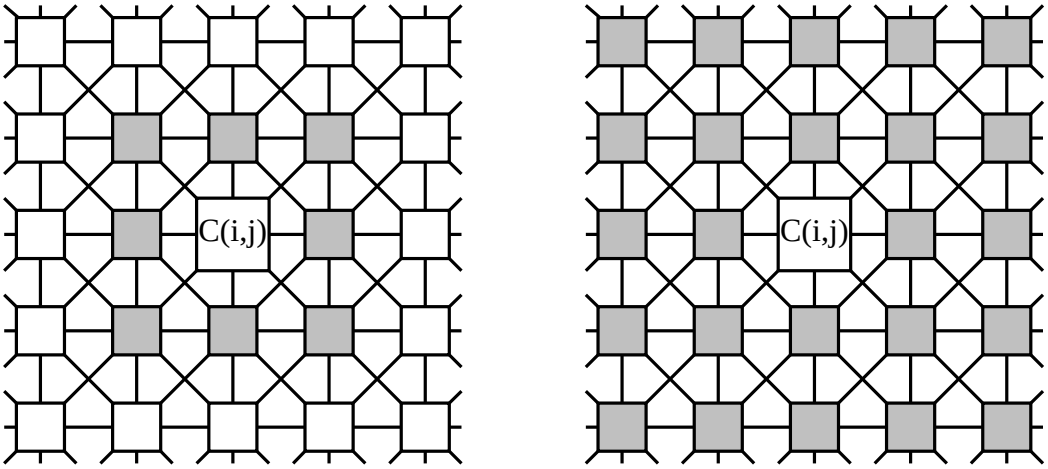



**1.1. ábra. A Celluláris Neurális Hálózat felépítése**

Egy $C(i, j)$ cella környezetét $S_r(i, j)$-vel jelöljük, és azokat a cellákat tartalmazza, amelyekre teljesül az alábbi:

$$S_r(i,j) = \left\{ C(k,l) \left| \max_{1 \le k \le M,\, 1 \le l \le N} \{ |k-i|, |l-j| \} \le r \right. \right\}, \quad r,i,j,k,M,N \in \mathbb{Z}^+ \tag{1.1}$$

A $C(i, j)$ cella egy $S_r(i, j)$ környezetét gyakran a cella $(2r+1)\times(2r+1)$-es környezetének nevezzük. Az egyes cellák csak a környezetükben lévő cellákkal tudnak kommunikálni, programozható összeköttetések, úgynevezett szinapszisok révén. A $C(i, j)$ cella $3\times3$-as ($r = 1$) és $5\times5$-ös ($r = 2$) környezete, és a cellák közötti összeköttetések az 1.2. ábrán láthatóak.

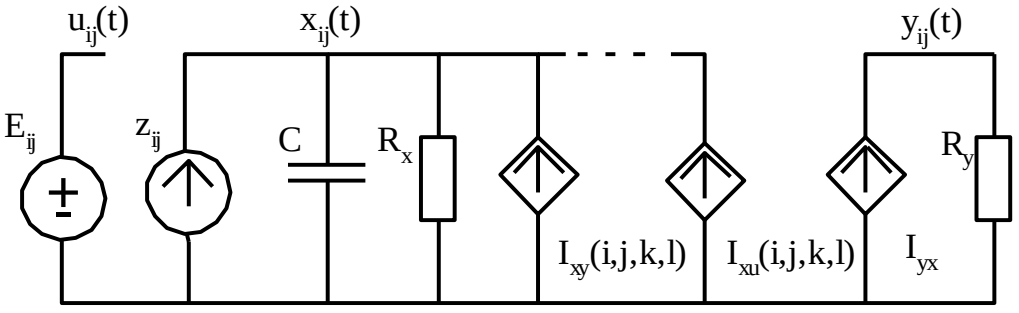



a.) $r=1$ b.) $r=2$

**1.2. ábra. Egy $C(i,j)$ cella $r=1$ és $r=2$ sugarú környezete**

A $C(i,j)$ cellát reguláris cellának nevezzük, ha a cella $S_r(i,j)$ környezetében minden $C(i,j) \in S_r(i,j)$ cella létezik. Ha ez a feltétel nem teljesül, akkor a $C(i,j)$ cellát határ cellának nevezzük [4].

Az 1.1. ábrán látható hálózat rácspontjaiban elhelyezkedő cellák egyszerű analóg áramköri elemekből épülnek fel. Minden egyes $C(i,j)$ cella rendelkezik egy állapotváltozóval, egy bemenettel és egy kimenettel. Az állapotváltozó értékében folytonos. A $C(i,j)$ cella a bemenetén keresztül figyeli a szomszédok bemeneteit és kimeneteit, melyek a saját állapotának megváltoztatásához szükségesek. A kimeneten keresztül pedig értesítheti szomszédjait a saját aktuális állapotáról. Egy ilyen analóg $C(i,j)$ cella kapcsolási rajza látható az 1.3. ábrán [6].




**1.3. ábra. Egy analóg $C(i,j)$ cella felépítése**

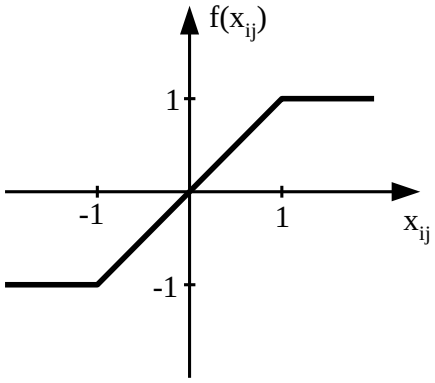
Az 1.3. ábra bal oldalán látható $E_{ij}$ a cella bemenete, innen kapja a cella a külvilág felől érkező $u_{ij}(t)$ jeleket. A feszültségforrás mellett egy $z_{ij}$ áramforrás is található, mely a

cellán belüli konstans eltolás vagy bias. A $C(i, j)$ cella $x_{ij}(t)$ állapotváltozója a $C$ kondenzátor árama, melyet az áramforrás forrásárama, az $R_x$ ellenálláson folyó áram és a feszültségvezérelt áramforrások forrásáramai ($I_{xu}(i,j,k,l)$ és $I_{xy}(i,j,k,l)$) határoznak meg. A vezérelt források áramát a szomszédos cellák bemenetén és kimenetén lévő, valamint a $C(i, j)$ cella saját be- és kimenetének megfelelően súlyozott értékei határozzák meg, az alábbi egyenletek alapján:

$$I_{xu} = \frac{1}{R_B(i,j,k,l)} u_{kl},\ I_{xu}, u_{kl} \in \mathbb{R}, R_B \in \mathbb{R} \times \mathbb{R},\ i,j,k,l \in \mathbb{Z}^+ \qquad \textbf{(1.2)}$$

$$I_{xy} = \frac{1}{R_A(i,j,k,l)} y_{kl},\ I_{xy}, y_{kl} \in \mathbb{R},\ R_A \in \mathbb{R} \times \mathbb{R},\ i,j,k,l \in \mathbb{Z}^+ \qquad \textbf{(1.3)}$$

ahol az $R_B(i, j, k, l)$ és az $R_A(i, j, k, l)$ a $C(i, j)$ és a $C(k, l)$ cella között lévő összeköttetés súlya. Az 1.3. ábra jobb oldalán látható egységnyi ellenálláson eső $y_{ij}(t)$ feszültség a $C(i, j)$ cella kimenete, melyet a tőle balra eső nemlineáris, úgynevezett szigmoid karakterisztikával (1.4. ábra) rendelkező feszültségvezérelt áramgenerátor $I_{yx}$ árama hoz létre az $R_y$ ellenálláson.




**1.4. ábra. A szigmoid karakterisztika**

Ebből adódik, hogy egy $M \times N$-es CNN hálózatot két egyenlettel a $C(i, j)$ cella ($i = 1,2,...,M$, $j = 1,2,...,N$) állapotegyenletével és egy kimeneti egyenlettel lehet leírni. Az állapotegyenlet Kirchhoff csomóponti törvénye alapján az (1.4) egyenletben megadott differenciálegyenlettel lehet leírni.

$$C\frac{dx_{ij}(t)}{dt} = -\frac{1}{R_x}x_{ij}(t) + \sum_{C(k,l)\in S_r(i,j)} A(i,j,k,l)y_{kl}(t) + \sum_{C(k,l)\in S_r(i,j)} B(i,j,k,l)u_{kl}(t) + z_{ij},$$

$$x_{ij}(t), y_{kl}(t), u_{kl}(t), z_{ij}, t, C, R_x \in \mathbb{R},\ A,B \in \mathbb{R}\times\mathbb{R},\ \text{és}\ i,j,k,l \in \mathbb{Z}^+ \qquad \textbf{(1.4)}$$

ahol az előbbieknek megfelelően az $x_{ij}(t)$ az állapot, az $y_{kl}(t)$ a kimenet, az $u_{kl}(t)$ a bemenet, és a $z_{ij}$ pedig a bias érték. Az (1.4) egyenletben két mátrix is található, amelyek a cellák között lévő összeköttetések súlyát határozzák meg, azaz:

$$A(i,j,k,l) = \frac{1}{R_A(i,j,k,l)},\ A, R_A \in \mathbb{R}\times\mathbb{R},\ i,j,k,l \in \mathbb{Z}^+ \qquad \textbf{(1.5)}$$

$$B(i,j,k,l) = \frac{1}{R_B(i,j,k,l)},\ B, R_B \in \mathbb{R}\times\mathbb{R},\ i,j,k,l \in \mathbb{Z}^+ \qquad \textbf{(1.6)}$$

ahol az $A(i,j,k,l)$ mátrix az úgynevezett visszacsatoló, míg a $B(i,j,k,l)$ mátrix az úgynevezett előrecsatoló szinaptikus operátor vagy template [6].

A kimeneti egyenlet az (1.7) egyenletben megadott nemlineáris karakterisztika, mely lényegében az 1.4. ábrán látható szigmoid karakterisztikát írja le [6].

$$y_{ij} = f(x_{ij}) = \frac{1}{2}\left|x_{ij}+1\right| - \frac{1}{2}\left|x_{ij}-1\right|,\ y_{ij}, x_{ij} \in \mathbb{R},\ i,j \in \mathbb{Z}^+ \qquad \textbf{(1.7)}$$

A legtöbb esetben az 1.3. ábrán látható $C$ kondenzátor és $R$ ellenállás egységnyi nagyságú, így az (1.4) egyenlet felírható az alábbi alakban is:

$$\dot{x}_{ij} = -x_{ij} + \sum_{C(k,l)\in S_r(i,j)} A(i,j,k,l)y_{kl} + \sum_{C(k,l)\in S_r(i,j)} B(i,j,k,l)u_{kl} + z_{ij},$$

$$x_{ij}, y_{kl}, u_{kl}, z_{ij} \in \mathbb{R},\ A,B \in \mathbb{R}\times\mathbb{R},\ \text{és}\ i,j,k,l \in \mathbb{Z}^+ \qquad \textbf{(1.8)}$$

Az $u_{kl}$ bemenet értéke rendszerint egy $M\times N$-es szürkeskálás kép adott pixelének intenzitása, a -1 és a +1 tartományra normalizálva, azaz $-1 \le u_{kl} \le +1$, ahol a -1 jelenti a fehér pixel intenzitás, míg a +1 a fekete pixel intenzitás értéket. Álló képek esetében az $u_{kl}$ értéke független az időtől, míg mozgó képek, mint például videó esetében időfüggő. A többi változó, azaz az $x_{ij}$, az $y_{kl}$ és a $z_{ij}$ is definiálhatók, mint kép pixel intenzitás értékek.

Az (1.4) egyenletből látszik, hogy minden egyes cella esetében az új állapotértéket a cella $S_r(i,j)$ környezetébe tartozó cellák be- és kimenetei alapján kell kiszámítani. A határcellák esetében azonban az $S_r(i,j)$ környezetnek van olyan része,

amely kívül esik a CNN $M \times N$-es struktúráján. Ezeken a pontokon úgynevezett virtuális cellákat kell definiálnunk, amelyek $u_{kl}$ és $y_{kl}$ értékeit az úgynevezett határfeltételek alapján lehet meghatározni [4]. A határfeltételeknek alapvetően három típusát különböztetjük meg:

- A Dirichlet vagy Fix határfeltétel, mely esetében a virtuális cellák bemenete: $u_{k,0} = E_1$, $u_{k,N+1} = E_2$, $u_{0,l} = E_1$, $u_{M+1,l} = E_2$, $l = 0,...,N+1$ és $k = 1,...,M$ és $k,l,M,N \in \mathbb{Z}^+$, $u,E_1,E_2 \in \mathbb{R}$,

  kimenete: $y_{k,0} = E_3$, $y_{k,N+1} = E_4$, $y_{0,k} = E_3$, $y_{M+1,l} = E_4$, $l = 0,...,N+1$ és $k = 1,...,M$ és $k,l,M,N \in \mathbb{Z}^+$, $y,E_3,E_4 \in \mathbb{R}$.
- A Neumann vagy Zero-flux határfeltétel, mely esetében a virtuális cellák bemenete: $u_{k,0} = u_{k,1}$, $u_{k,N+1} = u_{k,N}$, $u_{0,l} = u_{1,l}$, $u_{M+1,l} = u_{M,l}$ $l = 0,...,N+1$ és $k = 1,...,M$, és $k,l,M,N \in \mathbb{Z}^+, u \in \mathbb{R}$,

  kimenete: $y_{k,0} = y_{k,1}$, $y_{k,N+1} = y_{k,N}$, $y_{0,l} = y_{1,l}$, $y_{M+1,l} = y_{M,l}$ $l = 0,...,N+1$ és $k = 1,...,M$, és $k,l,M,N \in \mathbb{Z}^+, y \in \mathbb{R}$.
- A periódikus vagy toroid határfeltétel, mely esetében a határcellák bemenete: $u_{k,1} = u_{k,N}$, $u_{1,l} = u_{M,l}$, $l = 1,...,N$ és $k = 1,...,M$, és $k,l,M,N \in \mathbb{Z}^+, u \in \mathbb{R}$,

  kimenete: $y_{k,1} = y_{k,N}$, $y_{1,l} = y_{M,l}$, $l = 1,...,N$ és $k = 1,...,M$, és $k,l,M,N \in \mathbb{Z}^+, y \in \mathbb{R}$.

## *1.2.A CNN template-ek bemutatása*

Az (1.8) egyenletben bemutatott CNN állapotegyenletet kiegészíthetjük két újabb template-el is. Az egyik az állapottól ($C$ template), a másik pedig mindhárom változótól ($D$ template) függ. Ebben az esetben a CNN az alábbi állapotegyenlettel írható le:

$$\dot{x}_{ij} = -x_{ij} + \sum_{C(k,l)\in S_r(i,j)} A(i,j,k,l)y_{kl} + \sum_{C(k,l)\in S_r(i,j)} B(i,j,k,l)u_{kl} + \sum_{C(k,l)\in S_r(i,j)} C(i,j,k,l)x_{kl} + \\ + \sum_{C(k,l)\in S_r(i,j)} \hat{D}(i,j,k,l)(u_{kl},x_{kl},y_{kl}) + z_{ij},$$

$$x_{ij}(t),x_{kl}(t),y_{kl}(t),u_{kl}(t),z_{ij} \in \mathbb{R},\ A,B,C,\hat{D} \in \mathbb{R}\times\mathbb{R}, \text{ és } i,j,k,l \in \mathbb{Z}^+ \qquad \textbf{(1.9)}$$

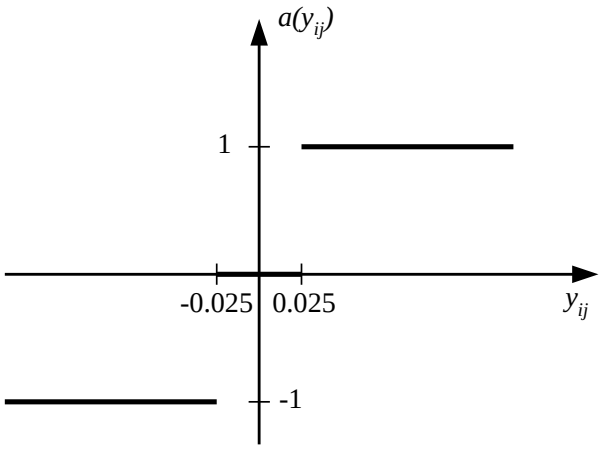
Az (1.9) egyenletben szereplő $A(i,j,k,l)$, a $B(i,j,k,l)$, a $z_{ij}$ illetve a $C(i,j,k,l)$ és $D(i,j,k,l)$ template-ek jelentik a CNN struktúra programját. A template-eket az alábbi három szempont szerint csoportosíthatjuk:

- helyfüggő és helyfüggetlen template-ek,
- idő variáns és idő invariáns template-ek,
- lineáris és nemlineáris template-ek.

Helyfüggő template-ek esetében az egyes template értékek függnek az $(i,j)$ pozíciótól, míg az idővariáns templete-ek esetében az egyes template értékek időfüggőek. A lineáris template-ek kizárólag konkrét számértékeket tartalmaznak. A nemlineáris templete-ek az aktuális cella valamely változójának (bemenet, kimenet, állapot), vagy pedig az aktuális és a szomszédos cella valamely változóinak (pl. különbségének) nemlineáris függvényei. Tehát lineáris template-ek esetén az aktuális template értékkel súlyozni kell a hozzá tartozó cella állapotát, bemenetét, vagy kimenetét az (1.9) egyenlet alapján. A nemlineáris template-ek esetén azonban a nemlinearitás alapján meghatározott nemlineáris template érték már a hozzá tartozó cella állapotának, bemenetének, vagy kimenetének template-el súlyozott értékét jelenti [4].

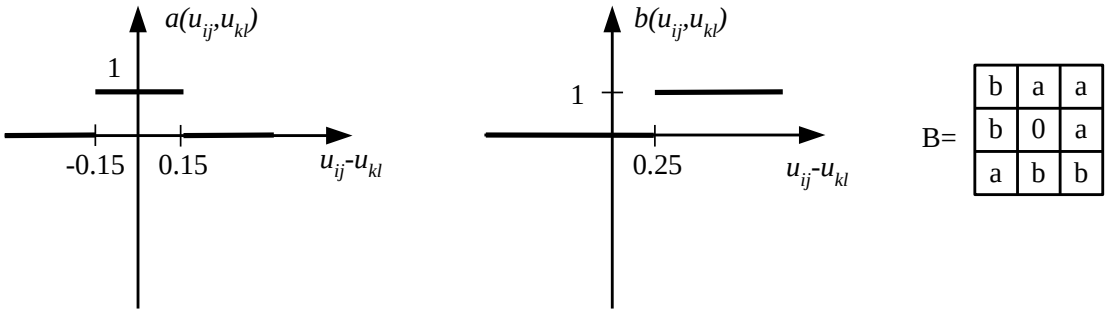
A nemlineáris template-ek alkalmazására számos példát találhatunk a CNN Template Library 3.1-es verziójában [4],[5]. Nézzünk most meg néhányat ezek közül.

1. **Példa:** A Majority Vote-Taker eljárás, mely segítségével eldönthető, hogy egy bináris bemeneti képen a kép soraiban található fekete és fehér pixelek száma között milyen reláció áll fenn. Az eljárás olyan $A(i,j,k,l)$ template-et alkalmaz, mely tartalmaz olyan értéket, amely az $y_{ij}$ kimenet 1.5. ábrán látható nemlineáris függvényeként van meghatározva ( $A(i,j,k,l) = a(y_{ij})$ ).



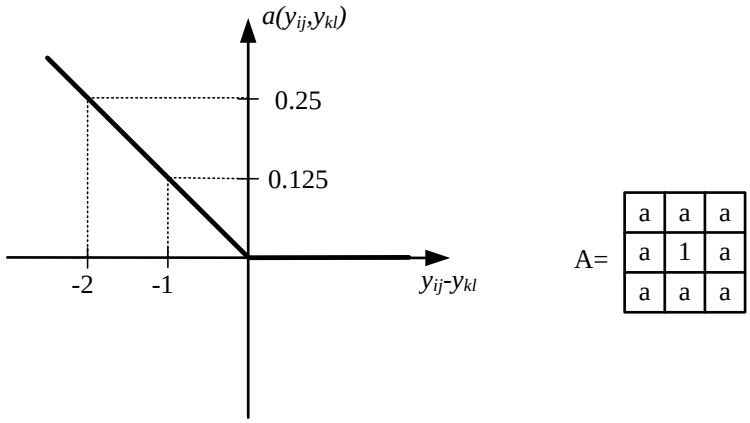

**1.5. ábra. Majority Vote-Taker eljárásban használt nemlinearitás és template**

2. **Példa:** A Grayscale Line Detector eljárás, mely egy bináris bemeneti képen meghatározza az egymással $30^{\circ}$-os szöget bezáró egyeneseket. Ennél az eljárásnál az $A(i,j,k,l)$ template megfelelő értékei az aktuális cella és a szomszédos cella kimenetének ($y_{ij}, y_{kl}$) különbségének nemlineáris függvénye által definiáltak ($A(i,j,k,l) = a(y_{ij}, y_{kl})$). Továbbá a $B(i,j,k,l)$ template értékei az aktuális cella és a szomszédos cella bemenetének ($u_{ij}, u_{kl}$) különbségének nemlineáris függvénye által meghatározottak ($B(i,j,k,l) = b(u_{ij}, u_{kl})$). Az eljáráshoz tartozó nemlinearitások és a template az 1.6. ábrán láthatóak.




**1.6. ábra. A Grayscale Line Detector eljárásban használt nemlinearitások és template**

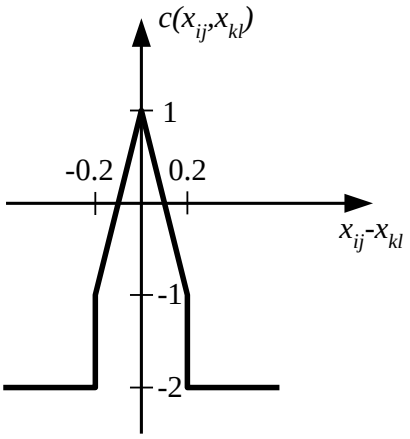
3. **Példa:** A Global Maximum Finder eljárás, mely meghatározza a bemeneti szürkeskálás képen a maximális intenzitás értéket. Ennél az eljárásnál az $A(i,j,k,l)$ template bizonyos értékei az aktuális cella és a szomszédos cella kimenetének ($y_{ij}, y_{kl}$) különbségének nemlineáris függvényei ($A(i,j,k,l) = a(y_{ij}, y_{kl})$), amely nemlinearitás és a hozzá tartozó template az 1.7. ábrán láthatóak.




**1.7. ábra. A Global Maximum Finder eljáráshoz tartozó nemlinearitás és template**

4. **Példa:** Az Extreme eljárás, mely a bemeneti szürkeskálás képen meghatározza egy adott küszöbértéknél kisebb gradiens értékek pozícióját. Ennél az eljárásnál

alkalmazott $C(i, j, k, l)$ template tartalmaz olyan értékeket, amelyek az aktuális cella, és a szomszédos cella állapotának $(x_{ij}, x_{kl})$ különbségének nemlineáris függvényei ($C(i, j, k, l) = c(x_{ij}, x_{kl})$). A nemlinearitás és a hozzá tartozó template az 1.8. ábrán láthatók.

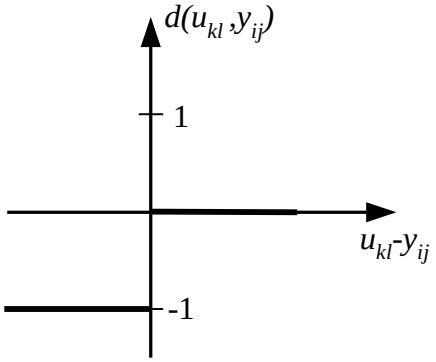



C=

| | | |
|---|---|---|
| c | c | c |
| c | 0 | c |
| c | c | c |

**1.8. ábra. Az Extreme eljáráshoz tartozó nemlinearitás és template**

5. **Példa:** A Grayscale Erosion eljárás, mely a bemeneti szürkeskálás képen eróziót hajt végre. Ennél az eljárásnál alkalmazott $D(i, j, k, l)$ template megfelelő értékei a szomszédos cella bemenetének és az aktuális cella kimenetének $(u_{kl}, y_{ij})$ különbségének 1.9. ábrán látható nemlineáris függvénye ($D(i, j, k, l) = d(u_{kl}, y_{ij})$).




D=

| | | |
|---|---|---|
| d | d | d |
| d | 0 | d |
| d | d | d |

**1.9. ábra. Grayscale Erosion eljáráshoz tartozó nemlinearitás és template**

## *1.3.A CNN különböző megvalósításai*

A CNN-nek alapvetően négy különböző implementációja van. Az első az analóg/kevert jelű VLSI megvalósítás (Ace400, Ace4K, Ace16k, Xenon, Eye-Ris), amelynek előnye a nagy számítási teljesítmény (akár 30 TerraOP/mp), azonban csak kis számítási pontossággal (7-8 bit) rendelkezik, valamint érzékeny a feszültség és hőmérsékletváltozásra, továbbá csak lineáris template-ek használhatk. A második

megvalósítás az emulált digitális VLSI, melyen belül megkülönböztetjük az ASIC (pl.: CASTLE architektúra) és az FPGA alapú (pl.: FALCON processzor) megvalósításokat. Mindkét megvalósítás hátránya, hogy lassabb, mint az analóg implementáció, és csak lineáris template-ek alkalmazhatók, azonban flexibilisebbek, könnyebben konfigurálhatóak, rövidebb a tervezési ciklusidejük, felületre optimalizáltak (a disszipált teljesítmény alacsony), és digitális környezetbe illeszthetőek. A harmadik megvalósítás az optikai megvalósítás, aminek előnye, hogy rendkívül gyors és nagyfelbontású képek feldolgozását támogatja, azonban a visszacsatolás megvalósítása nehéz, nagy szilícium felületet igényel és csak a lineáris template-ek alkalmazhatók. A negyedik CNN megvalósítás a szoftveres szimuláció, amelynek nagy előnye a lebegőpontos számítás, a nagy flexibilitás, az egyszerű implementáció, továbbá a nemlineáris template-ek használata. Hátránya, hogy jelenleg az asztali számítógépek esetén elterjedt 2, 3, vagy 4 magos processzorokat (Intel Core i5, AMD Phenom X2-X4,) használva, lényegesen lassabb megoldást nyújt, mint a másik három megvalósítás. A szoftveres szimuláción belül egyre inkább előtérbe kerülnek a sokprocesszoros tömbszámítógépek (például az IBM Cell processzor), amelyek már lényegesen gyorsabb megoldást nyújtanak, mint az asztali számítógépek esetén használt processzorok.

## *1.4.Analóg/kevert jelű CNN VLSI implementáció*

A CNN VLSI implementációi esetében megkülönböztetjük az analóg és az analóg-digitális (kevert jelű) megvalósításokat. Az analóg megvalósítások például az ACE-16k (Analogic Cellular Engine-16k) [13] és az ACLA (Asynchronous Cellular Logic Array) [14][15], míg a kevert jelű megvalósítások a SCAMP (Single Instruction Multiple Data Current-mode Analogue Matrix Processor) [16] és a Q-Eye [17]. Ezen megvalósítások esetében $128\times128$ vagy $176\times144$ méretű CNN cella tömböket alkalmaznak. Ebből következik, hogy egy ilyen VLSI CNN megvalósítás több mint 25000 ($176\times144$) speciális analogikai processzor elemet (cella) is tartalmazhat. Az analogikai elnevezés azt jelenti, hogy ezek a processzorok analóg és logikai műveletek elvégzésére egyaránt képesek. Köszönhetően az alkalmazott analogikai processzoroknak a VLSI CNN chip-ek sokkal kisebbek, mint a digitális chip-ek (például a DSP (Digital Signal Processor) chip). Nincs szükség sem időbeli sem értékbeli diszkretizálásra és iterációk végrehajtására sem, hisz a fizika törvényszerűségei alapján a numerikus integrálás folyamatosan kerül végrehajtásra. Hogy összehasonlíthassuk az analóg megvalósítás teljesítményét a digitális megvalósítással, meg kell határozni az analóg megvalósítás

ekvivalens számítási teljesítményét. Vegyünk például egy tetszőleges tér-idő dinamikát, mint például a hő- illetve hullámegyenletek, melyek megoldásához általában 10 iterációra van szükség. Ha $3\times3$-as template-eket feltételezünk, akkor ez cellánként 19 szorzás/összeadás műveletet jelent, azaz 190 szorzás/osztás műveletet kell elvégezni cellánként. Ez egy $176\times144$ CNN cellából álló rendszer esetében 5 millió ekvivalens cella műveletet jelent másodpercenként. Ezzel szemben egy analóg CNN chip, mint például az ACE-16k esetében, mely $128\times128$ cellát tartalmaz, és a fent említett feladatok 100 ns-os idő konstanssal megoldatók, ez 30 trillió ekvivalens cella műveletet jelent másodpercenként. Tehát az ilyen problémák megoldására sokkal jobb alternatívát jelentenek az analóg/kevert jelű CNN implementációk, ha csak a sebesség a lényeg. Ahogy az előző fejezetben is említettem, ezek a chip-ek azonban nagyon érzékenyek a hőmérséklet és feszültség ingadozásokra, és a pontosságuk is igen (7-8 bit) alacsony. Emellett ezen chip-eket alkalmazva nem lehet olyan tér-idő dinamikákat megoldani, mint a Navier-Stokes egyenlet, vagy az Euler egyenlet, melyek megoldásához nemlineáris template-ek alkalmazására lenne szükség [55].

## *1.5. Emulált digitális VLSI (ASIC/FPGA) implementáció*

### *1.5.1. A CASTLE architektúra*

A CNN Univerzális Gépet (CNN Universal Machine, CNN-UM) 1993-ban publikálták [8], mely lényegében az eredeti CNN paradigma egy kiterjesztése volt. Ennek egyik hatékony implementációja a CASTLE architektúra [9]. A CASLTE architektúra egy processzor magja közel $2\times2$ $mm^2$-es szilícium felületen implementálható 0.35 μm-es CMOS (Complementary Metal-Oxide-Semiconductor) technológiával. Ha 24 processzor mag működik párhuzamosan a CASTLE architektúrában, akkor 1ns/virtuális cella/CNN iterációt lehet elvégezni 12 bites pontosság mellett. Egy 25 fps sebességű digitális videó stream-et feltételezve, amely $240\times320$ pixel méretű képeket tartalmaz ez azt jelenti, hogy 500 CNN iterációt képes elvégezni a CASTLE architektúra 12 bites pontosság mellett. A CASTLE architektúra implementálása során az eredeti ((1.8) egyenlet) CNN egyenlet forward Euler formulával diszkretizált formájából indultak ki.

$$x_{ij}(n+1)=(1-h)x_{ij}(n)+h\left(\sum_{C(k,l)\in N_r(i,j)} A(i,j,k,l)y_{kl}(n)+\sum_{C(k,l)\in S_r(i,j)} B(i,j,k,l)u_{kl}+z_{ij}\right)$$

,

$$x_{ij},y_{kl},u_{kl},z_{ij},h\in\mathbb{R},\ A,B\in\mathbb{R}\times\mathbb{R}\text{, és } i,j,k,l,n\in\mathbb{Z}^{+} \qquad \textbf{(1.10)}$$

Az (1.10) egyenletben található változók közül a lehető legtöbb kiküszöbölése volt a cél. Ezért az eredeti ((1.8) egyenlet) CNN egyenlet helyett az úgynevezett Full Signal Range (FSR) modell került alkalmazásra [10].

$$x_{ij}(n+1)=\begin{cases}1 & v_{ij}(n)<1\\ v_{ij}(n) & |v_{ij}(n)|\le 1\\ -1 & v_{ij}(n)<-1\end{cases},$$

$$v_{ij}(n)=(1-h)x_{ij}(n)+h\left(\sum_{C(k,l)\in S_r(i,j)} A(i,j,k,l)x_{kl}(n)+\sum_{C(k.l)\in S_r(i,j)} B(i,j,k,l)u_{kl}+z_{ij}\right)$$

,

$$x_{ij},x_{kl},u_{kl},z_{ij},v_{ij},h\in\mathbb{R},\ A,B\in\mathbb{R}\times\mathbb{R}\text{, és } i,j,k,l,n\in\mathbb{Z}^{+} \qquad \textbf{(1.11)}$$

Ebben a modellben, ahogyan ez az (1.11) egyenletben is látható, a CNN kimenete és állapota egyenlő. Ha az állapotváltozó értéke meghaladná a +1 vagy a -1 értéket, akkor egyszerűen egy vágással bekorlátozták azt a kívánt $[+1,-1]$ tartományba. A számítások további egyszerűsítése céljából, a $h$ és az $(1-h)$ tagok bekerülnek az $A$ és a $B$ template-ekbe ( $\hat{A}$, $\hat{B}$ ), ahogyan ez az (1.12.) egyenletben is látható.

$$\hat{A}=\begin{bmatrix} ha_{-1,-1} & ha_{-1,0} & ha_{-1,+1}\\ ha_{0,-1} & (1-h)(1+a_{0,0}) & ha_{0,+1}\\ ha_{+1,-1} & ha_{+1,0} & ha_{+1,+1}\end{bmatrix} \qquad \hat{B}=\begin{bmatrix} hb_{-1,-1} & hb_{-1,0} & hb_{-1,+1}\\ hb_{0,-1} & hb_{0,0} & hb_{0,+1}\\ hb_{+1,-1} & hb_{+1,0} & hb_{+1,+1}\end{bmatrix} \qquad \textbf{(1.12)}$$

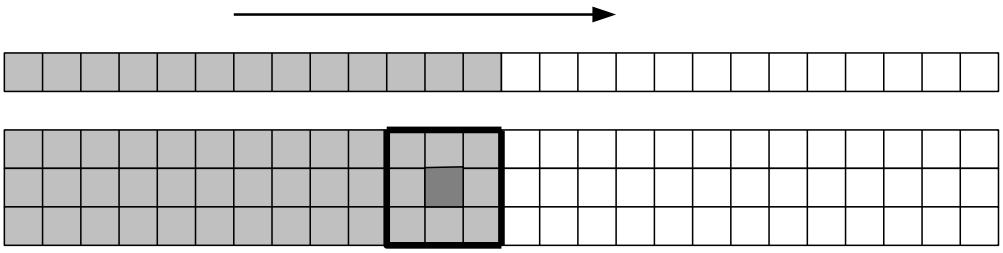
Így a diszkretizált CNN egyenlet kiszámításának iterációs lépéseiben csupán egy $3\times 3$-as konvolúciót és egy összeadást kell elvégezni, ahogy ezt az (1.13) egyenletben is láthatjuk.

$$v_{ij}(n+1)=\sum_{C(k,l)\in S_r(i,j)}\hat{A}(i,j,k,l)x_{kl}(n)+g_{ij}$$
$$g_{ij}=\sum_{C(k,l)\in S_r(i,j)}\hat{B}(i,j,k,l)u_{kl}+hz_{ij}$$

,

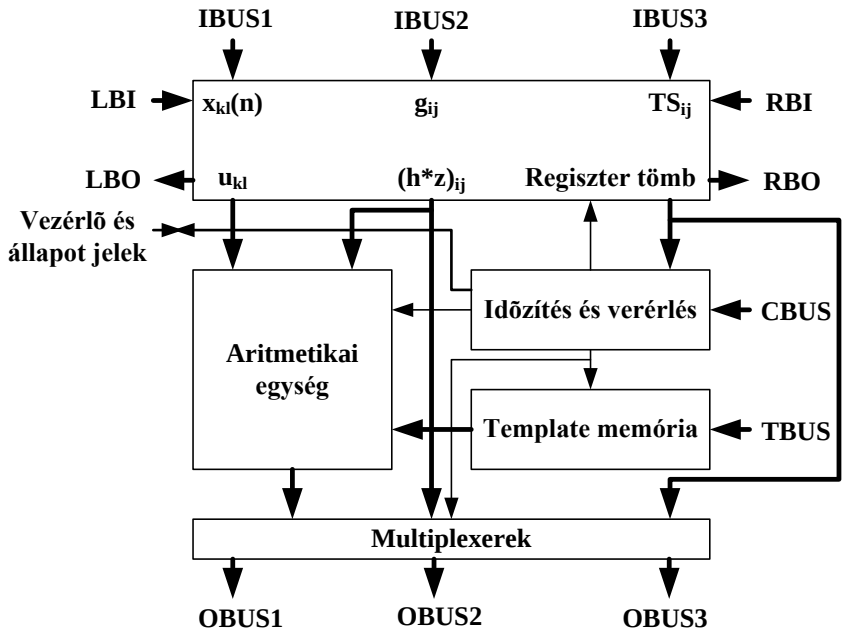
$$x_{kl},u_{kl},z_{ij},v_{ij},h\in\mathbb{R},\ \hat{A},\hat{B}\in\mathbb{R}\times\mathbb{R}\text{, és } i,j,k,l,n\in\mathbb{Z}^{+} \qquad \textbf{(1.13)}$$

Az (1.13) egyenlet kiszámítása során az első iterációs lépésben a $g_{ij}$ érték kerül meghatározásra, hisz a bemenet csak lassan vagy egyáltalán nem változik, majd a következő iterációs lépésekben a $v_{ij}$ következő értékét számítja ki a CASTLE egy processzor magja. Ha megnézzük az (1.13) egyenletet, akkor láthatjuk, hogy 9 template értékre, 9 állapot értékre és egy konstans értékre van szükség. Ezt az összesen 20 értéket nem lehet külső forrásból valós időben a processzorhoz eljuttatni, továbbá a teljes feldolgozandó kép sem tárolható el a processzor memóriájában. A fenti problémáknak a kiküszöbölésére csak a számításokhoz szükséges 9 template érték és a képből egy akkora sáv kerül eltárolásra a processzor memóriájában, ami a számítások elvégzéséhez szükséges ($3\times3$-as template esetében ez a kép 3 sorát jelenti), ahogy ez az 1.10. ábrán látható.



**1.10. ábra. A képből egy megfelelő sáv eltárolásával a szükséges I/O operációk jelentősen csökkenthetőek**

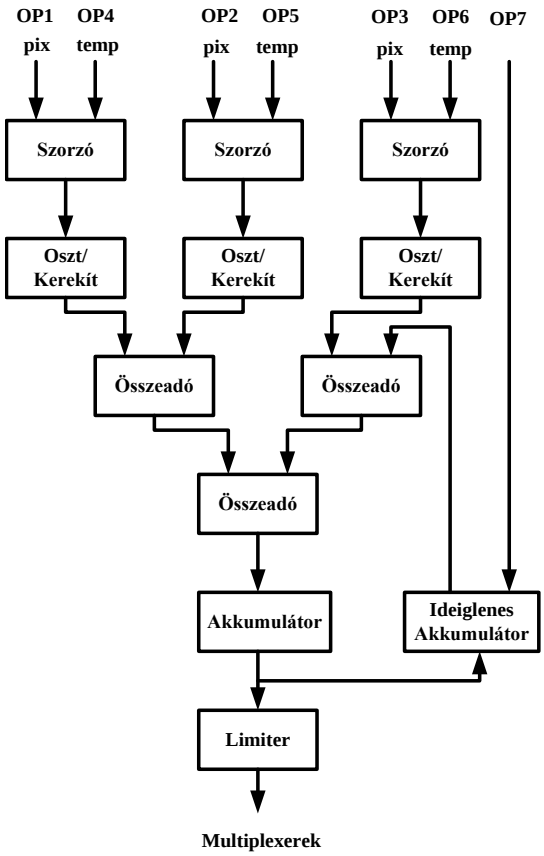
Ezzel a megoldással minden iterációs lépésben csak egy új értékre van szükség, azaz a processzornak csak 3 I/O műveletet (2 bemeneti és 1 kimeneti érték) kell elvégeznie iterációnként. A CASTLE architektúra egy processzor magjának rendszer szintű struktúrája látható az 1.11. ábrán.




**1.11. ábra. A CASTLE egy processzor magjának rendszer színű felépítése [9]**

A regiszter tömbbe az IBUS1 bemeneten keresztül érkeznek az $x_{kl}(n)$ és az $u_{kl}$ értékek, míg az IBUS2 bemeneten keresztül a $g_{ij}$ és a $(h \times z)_{ij}$ értékek. A template kiválasztó $TS_{ij}$ vektorok az IBUS3 bemeneten keresztül érkeznek a processzor maghoz. A $g_{ij}$ és az $x_{ij}(n+1)$ értékek kiszámolásához szükséges összes érték a regiszter tömbben kerül eltárolásra. A template értékek a TBUS bemeneten keresztül töltődnek a *Template memóriába*. Az LBI, LBO, RBI, RBO buszok a processzor jobb és bal oldali szomszédjaival történő kommunikáció lebonyolításáért felelősek. Az OBUS1, OBUS2, OBUS3 buszokon keresztül továbbítódnak a kiszámított értékek egy *Multiplexer* egységen keresztül a következő processzorhoz.

A CASTLE egy processzor magjában implementált *Aritmetikai egység* felépítése az 1.12. ábrán látható.

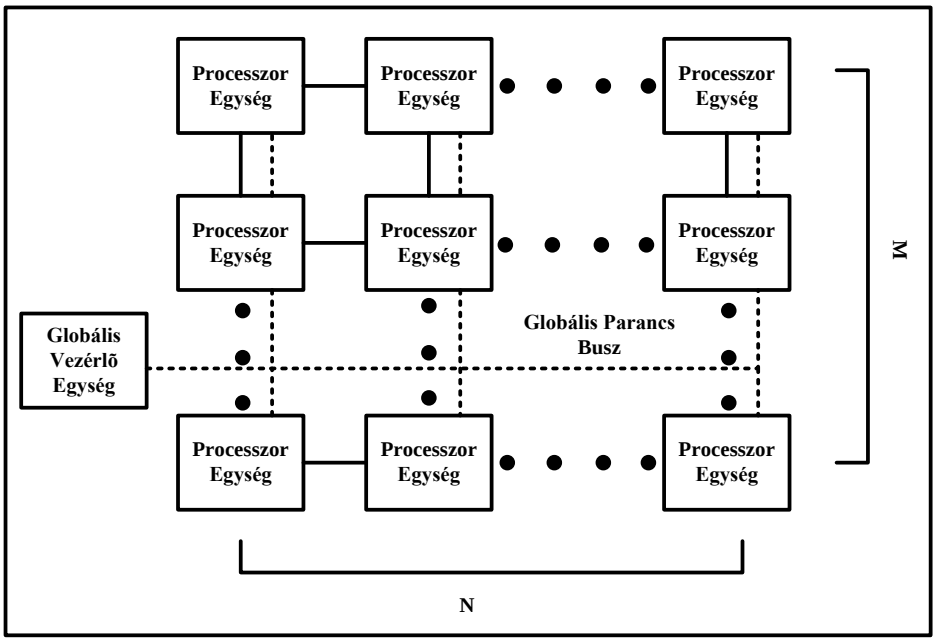



**1.12. ábra. A CASTLE processzor *Aritmetikai egységének* felépítése [9]**

Az 1.12. ábrán látható *Aritmetikai egységben* a három párhuzamos *Szorzó* egység biztosítja a pixel értékek és a hozzájuk tartozó template értékek összeszorzását. A *Szorzó* egységek kimenete egy *Oszt/Kerekít* egység bemenetére van kötve, mely elvégzi a szorzás után a szükséges kerekítéseket. A kétszintű összeadó fa a kerekített értékek összeadásáért felelős. Az összeadó fa első szintjén lévő egyik összeadó bementére az *Ideiglenes Akkumulátor* van kötve, melyben minden iterációs lépésben kezdeti értékként a $g_{ij}$ és a $(h \times z)_{ij}$ értékek töltődnek be. A közbülső aritmetikai ciklusokban az *Akkumulátor* a részösszegeket tárolja és továbbítja az összeadó fa első szintjén lévő egyik *Összeadónak*. A végső eredmény az *Akkumulátorba* kerül, mely kimenete egy *Limiter*-hez van kapcsolva, az eredmény szigmoid karakterisztika szerinti $[+1,-1]$ tartományba történő normalizálásához.

A CASTLE architektúra blokk diagramja az 1.13. ábrán látható. Ez a struktúra egy *Globális Vezérlő Egységből* (Global Analogic Programing Unit, GAPU) és egy szisztolikus tömbben elhelyezett $M \times N$ $(3 \times 2)$ *Processzor Egységből* áll. A feldolgozandó kép függőleges szeletekre van felosztva, így minden egyes *Processzor Egységnek* a kép csak egy hosszú keskeny sávját kell feldolgoznia. Egy *Processzor*

*Egység* csak a vele szomszédos processzorokkal kommunikálhat, ezzel minimalizálva a processzorok közötti kommunikációs problémát. A processzor struktúrában az egyes processzorok teljes mértékben szinkronizáltan működnek és az egyes processzorok a *Globális Parancs Buszon* keresztül kapják meg az aktuális utasításokat és paramétereket. Egy órajel ciklus alatt egy processzorsor megkapja a felette lévő sortól az előző iteráció eredményét, elvégez egy iterációt, és továbbküldi az eredményeket az alatta lévő processzor sornak.




**1.13. ábra. A CASTLE architektúra blokk diagramja [9]**

Az 1.13. ábrán látható CASTLE architektúrának alapvetően három működési módja van, melyek az alábbiak:

- az 1 bites logikai mód,
- a 6 bites pontosságú mód,
- a 12 bites pontosságú mód.

A pontosság növelésével az architektúra számítási teljesítménye jelentősen csökken, így az egyes feladatok megoldása során mindig meg kell találni az optimumot a számítás pontossága és a sebessége között.

A fentiekben bemutatott úgynevezett full-custom VLSI ASIC áramkörök megtervezése igen bonyolult feladat, még akkor is, ha a tervező rendelkezésére állnak a különböző professzionális CAD (Computer Aided Design) eszközök. Az elkészült áramkör valós hardver környezetben történő tesztelése is igen hosszadalmas folyamat lehet, az áramkör bonyolultságának függvényében. Egy jó alternatívát kínálnak a fent

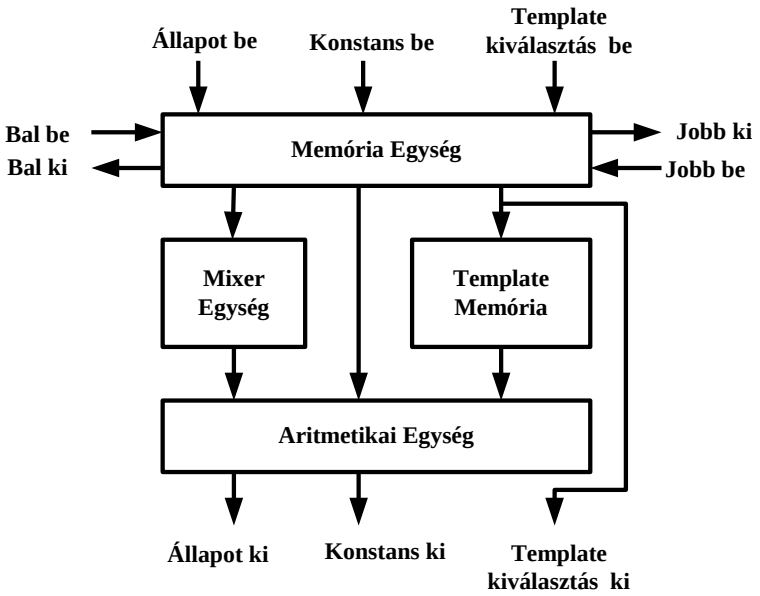
említett problémák megoldására az FPGA áramkörök, melyek lehetőséget biztosítanak a gyors prototípusfejlesztésre, mert a tervezés számos iterációja és az eközben szükséges javítások gyorsan ugyanazon eszközön megvalósíthatók. Ezen megfontolások alapján jött létre egy új FPGA alapú emulált digitális CNN-UM architektúra, az úgynevezett FALCON architektúra, melyet 2003-ban publikáltak [11],[55].

### *1.5.2. A FALCON architektúra*

A CNN-UM hatékony megvalósítása az FPGA alapú FALCON architektúra [11], mely alapvető tulajdonságait tekintve a CASTLE architektúrán alapszik, azonban kihasználva az FPGA alapú hardverfejlesztés előnyeit, egy sokkal flexibilisebben alkalmazható CNN-UM emulátor. A FALCON architektúrában az alábbi paraméterek konfigurálhatóak:

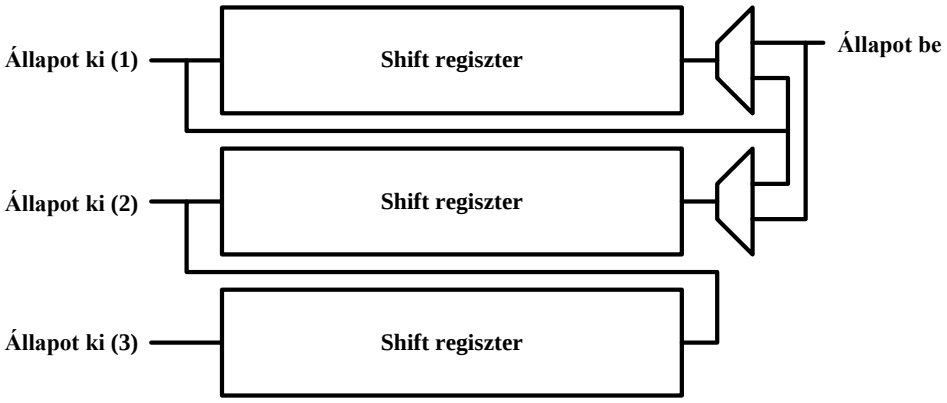
- az állapot a template értékek és a konstans érték bitszélessége és a radix pont elhelyezkedése 2 bittől egészen 64 bitig,
- a template-ek száma és szomszédossága,
- a cellatömb mérete,
- a processzor magok sorainak és oszlopainak száma.

A FALCON processzor, hasonlóan a CASTLE architektúrához, az (1.13) egyenletet számítja ki minden egyes iterációs lépésben, azonban van egy különbség, méghozzá az, hogy az $\hat{A}$ template középső elemét máshogyan, a $h\times(a_{00}-1)$ összefüggésnek megfelelőn kell kiszámítani. Ez azt jelenti, hogy egy plusz összeadást kell elvégezni, hogy összeadjuk a $h$-val megszorzott deriváltakat és az előző állapotértéket ( $x_{ij}(t)$ ). A diszkretizáláshoz használt $h$ időlépést praktikusan mindig kettő hatványának kell választani, így a $h$-val történő szorzás egy egyszerű eltolással (shift-eléssel) megvalósítható. Ennek köszönhetően csökken a FALCON architektúrában használt *Aritmetikai Egység* mérete, javítva ezzel az architektúra hardver kihasználtságát. A kimeneten korlátozzák az állapot értéket, ezzel egyszerűsítve az implementációt, mert ez lehetővé teszi, hogy a szükséges bitszélességgel lehessen számolni az *Aritmetikai Egység* egyes pontjain. A FALCON architektúra a CASTLE processzorhoz hasonlóan processzáló elemek egy $M\times N$-es szisztolikus struktúrájából épül fel. Egy processzáló egység négy fő részből, a *Memória Egységből*, a *Mixer Egységből*, az *Aritmetikai Egységből* valamint a *Template Memóriából* épül fel, ahogy ez az 1.14. ábrán is látható.



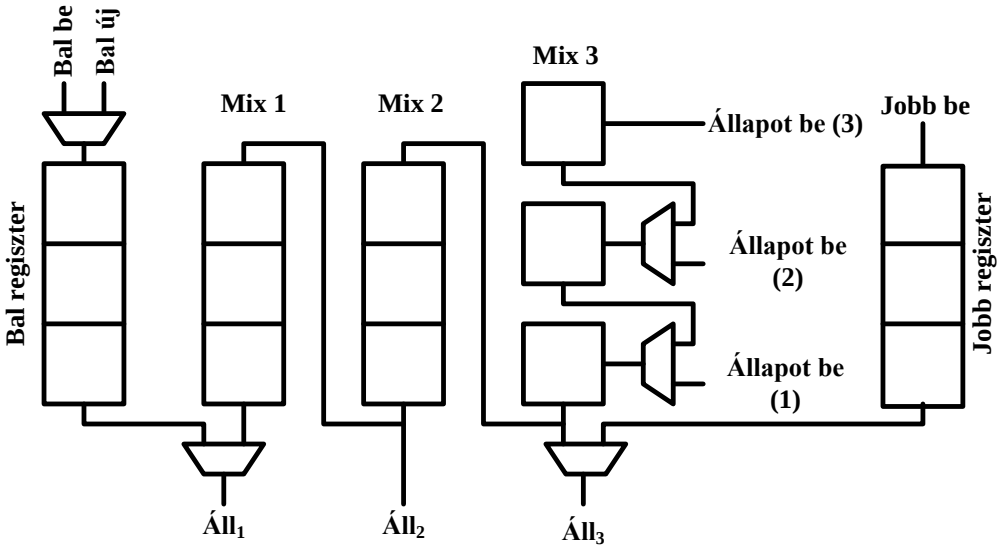

**1.14. ábra. A FALCON architektúra felépítése**

A *Memória Egység* a kép megfelelő részletét tárolja. Az egység egy 3×3 template-et feltételezve három *Shift regiszterből* épül fel, ahogyan ez az1.15. ábrán látható. A *Shift regiszterek* alkalmazásával kihasználhatjuk az FPGA-s chip-en lévő dedikált memória elemeket. Kisebb memóriák esetében a LUT-ok (Look Up Table) 16 bites – vagy az újabb FPGA-k esetében a 64 bites – regisztereit, míg nagyobb memóriák esetén a chip-en lévő BRAM-okat (Block RAM) használhatjuk, ahol az egyik port-ot mint bemenet, a másikat pedig mint kimenet definiálhatjuk. A *Shift regiszterek* hossza megegyezik a CNN cella tömb szélességével. Az *Állapot be* bemeneten keresztül történik az állapotértékek betöltése az első *Shift regiszterbe*, és az egyes *Shift regiszterek* kimenetei az alattuk lévő *Shift regiszterek* bemeneteihez vannak kötve. A *Shift regiszterek Állapot ki (1), (2), (3)* kimenetei pedig a *Mixer Egység* megfelelő bemeneteihez vannak csatlakoztatva. A zero-flux határfeltétel kialakítása érdekében, ahol a cellatömb első és utolsó sorát meg kell duplázni, a cellatömb első és utolsó sorát két *Shift regiszter* sorba töltjük be egyszerre. Nagyobb template-ek alkalmazása esetén természetesen több *Shift regiszter* sort tartalmaz a *Memória Egység*.




**1.15. ábra. A *Memória Egység* felépítése**

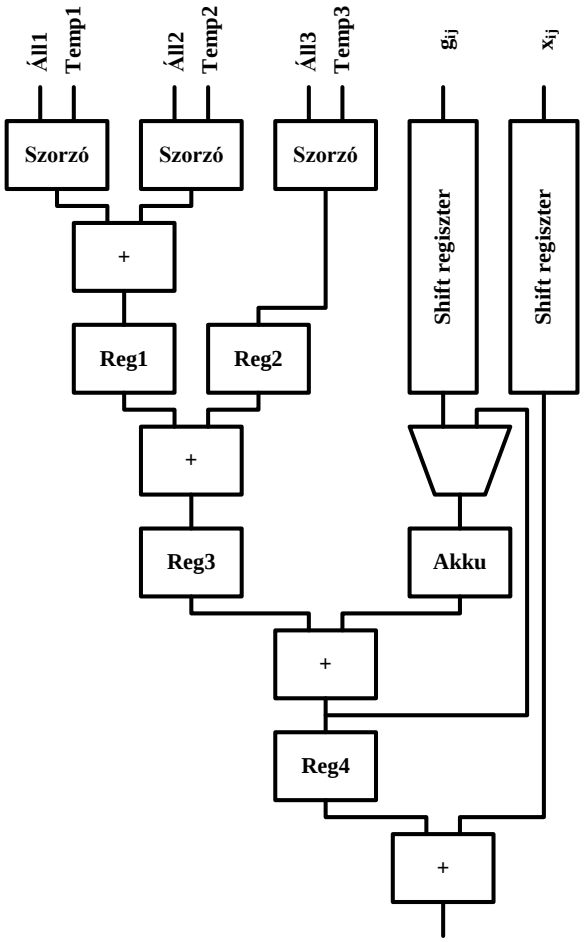
A *Mixer Egység* feladata, hogy a képpontok a megfelelő sorrendben jussanak az *Aritmetikai Egység* bemenetére. Az egység egy párhuzamosan írható és sorosan olvasható regiszterből, az aktuálisan feldolgozás alatt álló cella körül lévő ablak tárolására két shift regiszterből, és a jobb és bal oldali szomszédoktól érkező adatok tárolására még két shift regiszterből épül fel, ahogy ez az 1.16. ábrán látható. A három darab úgynevezett *Mix* regiszter egymással sorba van kapcsolva, továbbá a kimenetük az *Aritmetikai Egységgel* is össze van kötve. A szomszédos processzorok közötti kommunikáció a *Bal be*, *Bal új* és *Bal ki* bemeneteken és a *Bal Jobb ki* kimeneten keresztül történik. Az aktuális processzor *Jobb be* és *Bal be* bemenete a hozzá tartozó szomszéd *Bal ki* és *Jobb ki* kimenetéhez van kapcsolva. A *Mixer* belsejében a *Bal ki* és *Jobb ki* kimenet az utolsó *Mix3*-as regiszterhez van kapcsolva. A *Bal új* bemenet egy segédbusz, amely a bal oldali processzor *Memória Egységének* az *Állapot be* bemenetéhez kapcsolódik. Hasonlóan a *Memória Egységhez*, a *Mixer Egység* is több oszlopot tartalmaz, ha $3\times3$-as template-eknél nagyobb template-eket kell alkalmaznunk.




**1.16. ábra. A *Mixer Egység* felépítése**

Az *Aritmetikai Egység* (1.17. ábra) feladata, hogy elvégezze a szükséges szorzásokat és az összeadásokat. Az egység három szorzóból, négy összeadóból, két *Shift regiszterből*, négy általános és egy akkumulátor (*Akku*) regiszterből, valamint egy multiplexerből épül fel. A szorzók implementálásához a Xilinx Core Generator [12] könyvtárban megtalálható megfelelő IP (Intellectual Property) magot használhatjuk fel. Ennek segítségével lehetőség nyílik az FPGA-kban dedikált erősforrásként rendelkezésre álló MULT18x18 (például Virtex-II) vagy DSP48E1 (például Virtex 6) Slice-ok optimális kihasználására. Az *Aritmetikai Egységnek* három órajel-ciklusra van szüksége egy új cellaérték kiszámítására. Az első órajelre az egység bemenetén megjelenik az aktuális template első sora a hozzá tartozó állapotértékekkel, illetve az aktuálisan feldolgozott cella állapotértéke és a hozzá tartozó konstans. A következő két órajelben a template maradék két sora, valamint a megfelelő állapotértékek töltődnek be a szorzókba. A következő cella feldolgozása a következő órajelben kezdődhet. A szorzók késleltetése után két plusz órajel kell a szorzatok összeadásához. Amikor az első érték betöltődik a *Reg3*-ba, akkor az aktuális cella konstans értékét be kell tölteni az *Akku* regiszterbe. A következő két órajelben az első és második template sorhoz tartozó részeredmények tárolódnak az *Akku* regiszterben. A végeredmény a *Reg4* regiszterben tárolódik, mert ebben az órajel-ciklusban az *Akku*-ba betöltődik a következő cellához tartozó konstans érték. A folyamat következő lépéseként az állapotérték fentiekben számított derivált értéke hozzáadódik a régi értékhez, és az így kapott új állapotértéket a szigmoid karakterisztikának megfelelően limitáljuk. A konstans érték és a régi cellaérték tárolására szolgáló *Shift regiszterek* hossza változó, hisz hosszukat a szorzó egységek

késleltetése határozza meg. Az *Aritmetikai Egység* esetében is több szorzóra és nagyobb összeadó fára van szükség, ha a template mérete nagyobb lenne, mint $3\times3$.




**1.17. ábra. Az *Aritmetikai Egység* felépítése**

A *Template Memória* feladata, hogy eltárolja a CNN programját jelentő template értékeket. Minden egyes órajel ciklusban a template megfelelő sorát továbbítja az *Aritmetikai Egység* megfelelő bemeneteire. Lehetőség van akár több template eltárolására is, melyek közül a *Template kiválasztás be* jel segítségével választhatunk [55].

Az itt bemutatott FALCON processzor csak lineáris template-ek futtatására alkalmas, azonban alkalmassá tehető nemlineáris template-ek futtatására is, ha megfelelően módosítjuk a *Template Memóriáját*. Ezt a későbbiekben fogom bemutatni (lásd 3. fejezet).

## 1.6. Optikai megvalósítás

Az optikai számítógépek nagyfokú párhuzamosságot biztosítanak, és ez által igen magas számítási teljesítménnyel rendelkeznek. Ezek a számítógépek azonban kevésbé flexibilisek [18][19][20], és a szűk keresztmetszet a számítási eredmények utólagos feldolgozása volt, amit rendszerint valamilyen DSP segítségével végeztek el [21][22]. Ezekre a problémákra jelentett megoldást a programozható optoelektronikai analóg CNN számítógép (POAC, Programmable Optoelectronic Analog CNN), mely hatékonyan használható sokféle képfeldolgozási feladat megoldásához [23]. A POAC segítségével egy $3500 \times 4500$, vagy pedig egy $4500 \times 4500$ pixel felbontású folyadékkristályos kijelzőt használva bemenetként elérhető akár a 10-100 TerraOperáció/másodperces számítási teljesítmény is. Ez jócskán felülmúlja az analóg/kevert jelű CNN implementációk teljesítményét, azonban ennél a megvalósításnál a visszacsatolás megvalósítása nehézkes és csak lineáris template-ek alkalmazhatók.

## 1.7. Szoftveres szimuláció

A CNN implementációjának legegyszerűbb és legflexibilisebb módja a szoftveres szimuláció. Ilyen szoftveres szimulációs programok például a SimCNN [24] vagy a MatCNN [25], melyeket konvencionális asztali számítógépeken lehet futtatni. Ezen szimulációs programok esetében a CNN miden paramétere tetszőlegesen állítható, mint például a template mérete, az állapot pontossága, az alkalmazott rétegek száma, lineáris vagy nem lineáris template-ek használata. Továbbá ezeknek a szoftvereknek van saját beépített template és kép könyvtára is. Hátrányuk azonban, hogy igen lassúak, még ha a jelenleg az asztali számítógépekben elterjedt 2, 3, vagy 4 magos processzorokat (Intel Core i5, AMD Phenom X2-X4,) használjuk [55].

Ezen szimulációs programok általában az alábbi lépéseket hajtják végre:

- numerikus integrálást használva kiszámítják a CNN dinamikát egy megadott template-el,
- megjelenítik a bemeneti és kimeneti képeket legyen az bináris szürkeskálás vagy színes,
- szimulálják a CNN dinamikát egy adott template sorozatra.

A CNN szoftveres szimulációjának gyorsítására jó alternatíva lehet a sokprocesszoros tömbszámítógépek alkalmazása, mint például az IBM, Sony, Toshiba cégek

együttműködéséből született Cell, melyet a Sony PlayStation 3 számára terveztek [26]. Ez a tömbprocesszor egy 64-bites PowerPC és nyolc 128-bites SIMD (Single Instruction Multiple Data) utasításkészlettel rendelkező RISC (Reduced Instruction Set Computer) magot tartalmaz. Az IBM Blade Center QS20 rendszer két, egymással egy szélessávú interface-en keresztül kommunikáló Cell processzorból épül fel, melynek maximális számítási teljesítménye 400 GFLOPS. Egy Blade Center házban hét darab QS20 kapcsolható össze, mellyel így elérhető a 2.8 TFLOPS számítási teljesítmény is. Az IBM Blade Center legújabb generációival, a QS22-vel pedig már a 6.4 TFLOPS egyszeres és a 3 TFLOPS dupla pontosságú számítási teljesítmény is elérhető. Ezt a processzort felhasználva, és a CNN dinamika szimulációját a processzor speciális igényeihez igazítva egy $176 \times 144$ pixel méretű képet feltételezve elérhető a közel 230 millió cellaiteráció/másodperces számítási teljesítmény a lineáris és 130 millió cellaiteráció/másodperces számítási teljesítmény a nemlineáris template-ek alkalmazása esetén.

# 2. Field Programmable Gate Array áramkörök

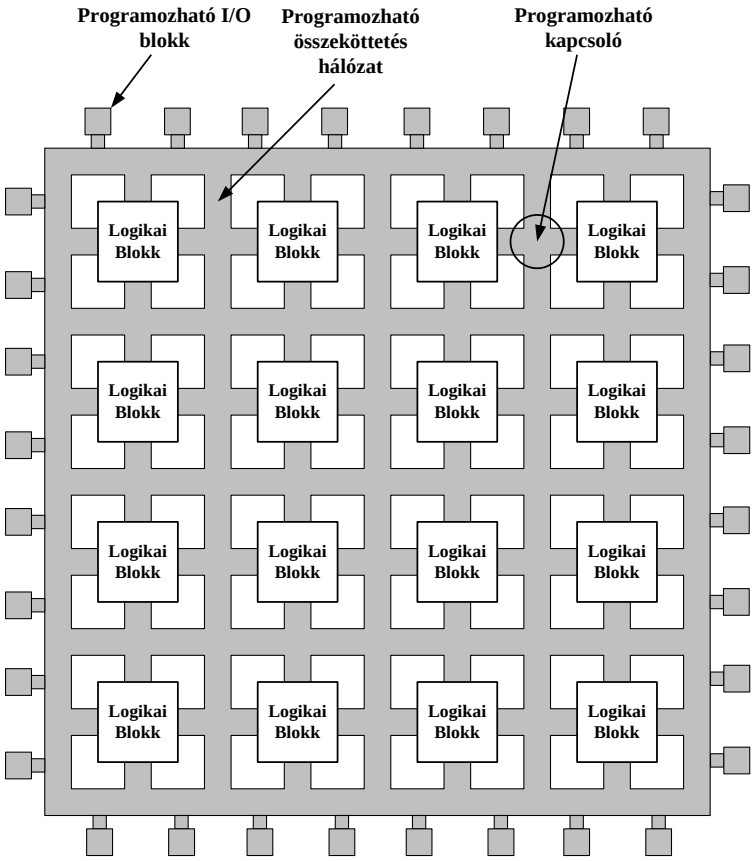
Az újrakonfigurálható számítás alapgondolatát 1960-ban Gerald Estrin fogalmazta meg [1]. Az első újrakonfigurálható eszközök az FPGA áramkörök voltak, melyeknek első kereskedelemben is kapható képviselője 1985-ben jelent meg. Ezek a chip-ek lényegében különböző programozható funkciókat megvalósító blokkokból és memóriaelemekből épülnek fel, melyeket programozható összeköttetés hálózat segítségével lehet összekapcsolni egymással. Az első eszközök csak kevés ilyen logikai egységet tartalmaztak, általában VLSI chip-ek közötti összeköttetésekhez kisebb áramköröket („glue logic") valósítottak meg bennük. A későbbi FPGA generációk viszont már elegendő logikai erőforrással rendelkeztek ahhoz, hogy komplett számítási algoritmusokat lehessen velük megvalósítani. A mai modern FPGA-k logikai sűrűsége meghaladja az 1.200.000 darab 6-bemenetű Look Up Table-t. Emellett ezek az FPGA-k már tartalmaznak beágyazott nagysebességű DSP blokkokat, SRAM (Static Random-Access Memory) memória blokkokat és mikroprocesszor magokat, valamint nagysebességű soros I/O blokkokat. Ezek segítségével komplett SoC (System on Chip) rendszerek építhetők fel.

## *2.1. Az FPGA áramkörök*

Az FPGA áramköröket az első kereskedelmi forgalomba kerülésük (1985) óta, ha szűk körben is, de alkalmazták különböző ipari projektekben [1]. Az alacsony költségük, tetszőleges újrakonfigurálhatóságuk, és a gyors prototípusfejlesztés lehetősége miatt azonban annyira elterjedtek, hogy ma már megtalálhatók az autóipar, hadiipar, az orvosi eszközök, a képfeldolgozás, csillagászat stb… területén is. Az FPGA áramkörök egyik legnagyobb előnye a párhuzamos működés, tehát hogy több különböző feladatot ténylegesen (nem multitask-os rendszerben) egy időben tudnak végrehajtani. A speciális alkalmazásokhoz gyártott, integrált ASIC vagy MPGA (Mask Programmed Gate Array) chip-ekhez képest az FPGA áramkörök kis sorozatban olcsóbbak, viszont nagyobb mennyiségben az ASIC vagy MPGA áramkörök gyártása térül meg. Egy speciális feladat FPGA-n történő megvalósítása lassabb lehet, mint egy ASIC chip esetén, viszont a tetszőleges újrakonfigurálhatóság miatt az FPGA egy vonzóbb alternatíva.

## *2.2.Az FPGA architektúra*

Az FPGA-k architektúrája három fő részből épül fel. Ezek a logikai blokkok, az összeköttetések és az I/O blokkok [1]. Az FPGA-k programozható logikai blokkjait programozható összeköttetés hálózat kapcsolja össze egymással, illetve a chip-en kívüli kommunikációt biztosító programozható I/O blokkokkal. Az FPGA-k általános felépítése az 2.1. ábrán látható.



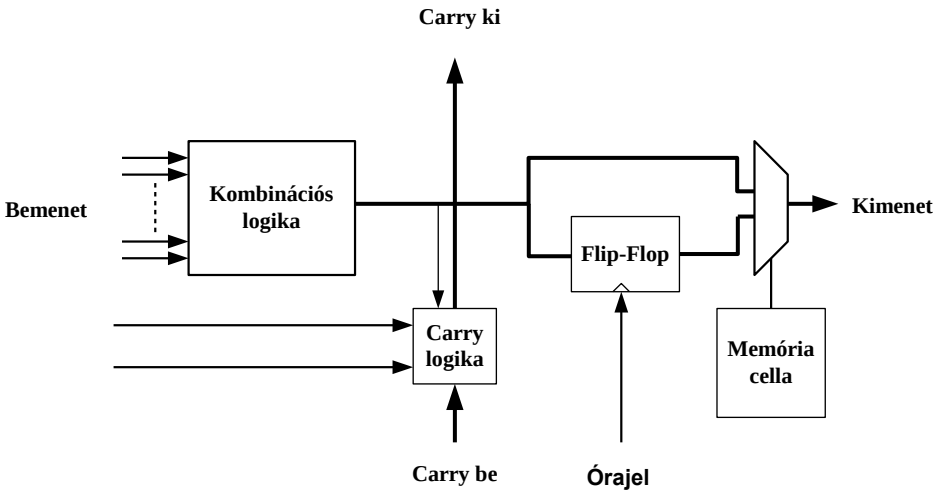

**2.1. ábra. Az FPGA-k általános felépítése**

Az FPGA-kat megkülönböztethetjük architektúrájuk szemcsézettsége (finom és durva) alapján. A durva szemcsézettségű FPGA-k programozható logikai blokkjaikban rendszerint kombinációs logikák, memóriaelemek, és multiplexerek vannak. Az ilyen komplex logikai blokkok a relatíve kis tranzisztorszám mellett is magas fokú funkcionalitást biztosítanak. Ez azonban nagyszámú bemenetet és emiatt bonyolultabb huzalozást igényel. Más részről az architektúra optimalizált szintézis a logikai blokkok magas szintű kihasználását teszi lehetővé. A finom szemcsézettségű FPGA-k rendszerint csak kevés tranzisztort vagy egy egyszerű két bemenetű kaput tartalmaznak. Az ilyen FPGA-kkal magas fokú logikai kihasználtság érhető el, hisz a tranzisztorok vagy kapuk szintjén könnyebb a komplex logikai függvényeket megvalósítani. Más

részről ez a struktúra sok összeköttetést és programozható kapcsolót igényel, ami miatt megnövekszik a felhasznált szilícium felület és a késleltetési idő. Ezáltal a finom szemcsézettségű FPGA-k lassabbak és nagyobb méretűek, mint a durva szemcsézettségű társaik.

### *2.2.1. Programozható logikai blokk*

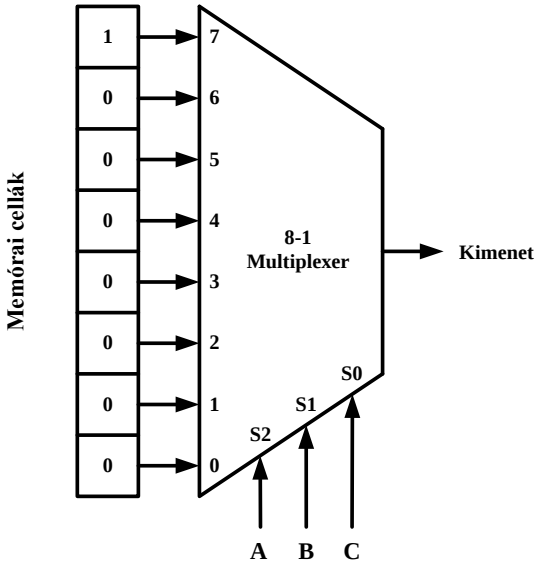
Az FPGA tervezők számos különböző programozható logikai struktúrát fejlesztettek ki az FPGA-k 1980-as feltalálása óta [1]. Ezeknek a struktúráknak a felépítése kicsivel több, mint tíz év alatt egységesedett. A programozató logikai blokk általános felépítése a 2.2. ábrán látható.




**2.2. ábra. Programozható logikai blokk**

Egy programozható logikai blokk rendszerint programozható kombinációs logikából, memóriaelemből (Flip-Flop), multiplexerből és gyors carry logikából épül fel. A 2.2. ábrán látható programozható logikai blokk kimenetére a kombinációs hálózat kimenete vagy a Flip-Flop kimenete kerülhet. Ennek kiválasztását a multiplexer végzi el, melyhez egy konfigurációs memóriacella kapcsolódik, mely segítségével beállítható, hogy mi kerüljön a multiplexer kimenetére. A kereskedelemi forgalomban kapható FPGA-k a 2.2. ábrán látható általános programozható logikával ellentétben lényegesen nagyobb flexibilitást biztosítanak. Például olyan Flip-Flop-okat tartalmaznak, amelyek beállíthatók, mint egyszerű késleltetés, vagy, hogy rendelkezzenek egy vagy több szinkron vagy aszinkron set-reset bemeneti kombinációval, illetve az is megadható, hogy az órajel fel- vagy lefutó éle vezérelje a működést. Emellett a carry logika is tartalmazhat plusz funkciókat. A Xilinx Virtex sorozatban található carry logika például segíthet a szorzás megvalósításában. A logikai blokk kombinációs logikai részét illetően

számos különböző implementációs eljárást fejlesztettek ki. A legelterjedtebb módja a kombinációs logikák megvalósításának a LUT alkalmazása. Egy ilyen 3 bemenetű LUT koncepcionális felépítése látható a 2.3. ábrán.




**2.3. ábra. 3-bemenetű LUT**

A LUT lényegében programozható memóriaelemek tömbje, melyek egy multiplexerhez csatlakoznak. Ez a multiplexer kiválasztja, hogy melyik memóriaelem tartalma kerüljön a kimenetre. A LUT $n$ darab címbemenettel és $2^n$ darab memóriahellyel rendelkezik. Ahhoz, hogy egy LUT segítségével implementálhassunk valamilyen logikai függvényt, a függvény igazságtáblázatát be kell tölteni a memóriába. A 2.3. ábrán a LUT például egy 3-bemenetű ÉS függvényt valósít meg. Az alkalmazott memóriaelemek általában SRAM típusúak, és 3, 4 vagy 6 bemenetű multiplexereket alkalmaznak.

Az összeköttetés hálózat költségeinek csökkentése érdekében sok FPGA architektúrában cluster-ekbe szervezik a 2.2. ábrán látható logikai blokkokat. Egy cluster-en belül a logikai elemeket rövid és gyors összeköttetések segítségével kapcsolják össze. Ez a csoportosítás nagyobb logikai függvények nagyobb sebességű implementálását biztosítja egy cluster-en belül.

### *2.2.2. Programozható összeköttetés hálózat*

A programozható összeköttetések megvalósításának alapvetően két különböző módja van, ezek a memória és az antifuse alapú technikák. A memória alapú technikák esetében három különböző módszert különböztetünk meg, melyek az SRAM-alapú, az

EEPROM/Flash-alapú (Electrically Erasable Programmable Read-Only Memory) és a kevert módszerek [2][3].

A legtöbb FPGA gyártó az SRAM-alapú technikát alkalmazza. Ennek a módszernek a lényege, hogy egy statikus memóriát és egy áteresztő tranzisztort használnak a keresztező vezetékek összekapcsolására [2][3]. Ezt a technológiát alkalmazza a Xilinx cég is az FPGA áramköreiben a programozható összeköttetés hálózat megvalósítására.

Az EEPROM/Flash-alapú technológia az SRAM-alapú technológiához hasonló, csak a keresztező vezeték közötti kapcsolat állapotát nem egy SRAM-ban tárolják el, hanem két tranzisztort alkalmaznak. A két tranzisztor közös lebegő kapun osztozik, mely a két vezeték közötti kapcsolat állapotát tárolja [3].

A kevert technológia esetében ötvözik az SRAM-alapú és az EEPROM/Flash-alapú technológiákat. Ennél a megvalósításnál az SRAM alapú technológiához hasonlóan, a keresztező vezetékek összekapcsolásához áteresztő tranzisztorokat alkalmaznak, melyek állapotát a hozzájuk tartozó SRAM cellákban tárolják el. Az SRAM cellák feltöltése pedig egy Flash memóriaelemben tárolt értékek alapján történik, mely tartalmát az FPGA felprogramozásakor állítjuk be. [3].

Az antifuse FPGA-k az SRAM-alapú FPGA-kkal szemben nem vesztik el programjukat a tápfeszültség kikapcsolásakor. A keresztező vezetékek egy antifuse réteg alsó és felső rétegéhez kapcsolódnak, egy nagy ellenállást képviselő dielektrikumon keresztül. Megfelelően nagy feszültséget alkalmazva a szigetelésként szolgáló dielektrikum átég, és így véglegesen létrejön a kapcsolat a két vezeték között [3].

A programozható összeköttetések esetében alapvetően két típust különböztetünk meg [1]. Az egyik az úgynevezett lokális összeköttetés hálózat, mely a cluster-ekbe szervezett logikai blokkok összekapcsolását biztosítja. A másik pedig a globális összeköttetés hálózat, mely az egyes cluster-eket egymással, illetve a külvilággal történő kapcsolattartásért felelős programozható I/O blokkokkal kapcsolja össze.

A logikai cluster-ek belsejében kialakított programozható összeköttetések több feladatot láthatnak el. Egyrészt meghatározzák, hogy hol vannak a logikai blokkok bemenetei, és hová mennek a kimenetei. Másrészt meghatározzák, hogy hogyan haladnak át a jelek a logikai blokkokon. Harmadrészt pedig meghatározzák az egyes logikai blokkok egymással történő összekapcsolását is.

A globális összeköttetés hálózatot kialakításuk módja szerint négy fő csoportba sorolhatjuk. Ezek az island, a celluláris, a long-line és a sor architektúra, melyek közül a Xilinx cég az island típusú globális összeköttetés hálózatot alkalmazza [1][12].

### *2.2.3. Programozható I/O architektúra*

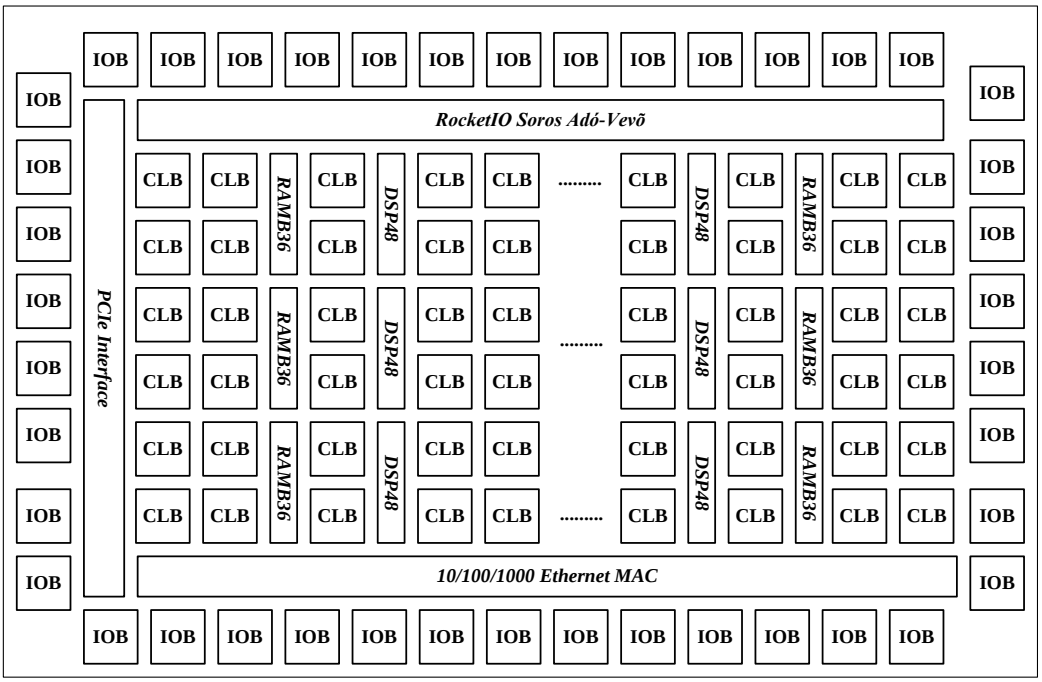
A programozható I/O architektúra három-állapotú (tri-state) puffereket tartalmaz a kimeneteken és bemeneti puffereket a bemeneteken. A három-állapotú engedélyező jelek, a kimeneti jelek, és a bemeneti jelek egymástól függetlenül konfigurálhatók az I/O blokkok felprogramozásának függvényében [1]. A mai modern FPGA-k további konfigurációs lehetőségeket is tartalmaznak.

## *2.3.A Xilinx FPGA-k*

Az FPGA áramkörök piacát jelenleg két gyártó, a Xilinx és az Altera cégek uralják. Mindkét cég FPGA áramkörei alapvetően két csoportba sorolhatók. Az egyik csoport a nagy számítási teljesítményre optimalizált (Highest Bandwidth and System Performance) FPGA-k, míg a másik csoport az alacsony költségű és fogyasztásra optimalizált (Low Cost and Power) FPGA-k. Mindkét cég gyártási palettáján szerepelnek a fent említett csoportokba tartozó FPGA-k, azonban a Xilinx az első, míg az Altera inkább a második csoportba tartozó FPGA-k gyártásában vállal vezető szerepet. A disszertációmban bemutatatásra kerülő alkalmazások számára nem kritikus az alacsony fogyasztás, viszont annál kritikusabb a magas számítási teljesítmény, ezért az alkalmazások megvalósításához a Xilinx cég által gyártott FPGA-kat választottam. Ebben az alfejezetben tehát a Xilinx cég által gyártott nagy teljesítményű Virtex-6 FPGA család felépítése kerül bemutatásra.

### *2.3.1. A Xilinx Virtex-6 FPGA architektúra*

A Virtex-6 sorozatú FPGA-kat 2009-ben mutatta be a Xilinx cég [12]. Ez az FPGA család 40nm-es gyártási technológiával készült, és három alcsaládot különböztethetünk meg, melyek az LXT, az SXT és a HXT. Az LXT alcsaládba a nagy teljesítményű logikával, az SXT alcsaládba a legnagyobb számítási teljesítménnyel, míg a HXT alcsaládba a legnagyobb sávszélességgel rendelkező Virtx-6 FPGA-k tartoznak. Megjegyzendő, hogy a legújabb Xilinx FPGA a Virtex-7, melyet 2010-ben jelentettek be, már 28 nm-es technológiával készült. A Virtex-6 FPGA vázlatos felépítése a 2.4. ábrán látható, kiemelve a legfontosabb építőelemeket.

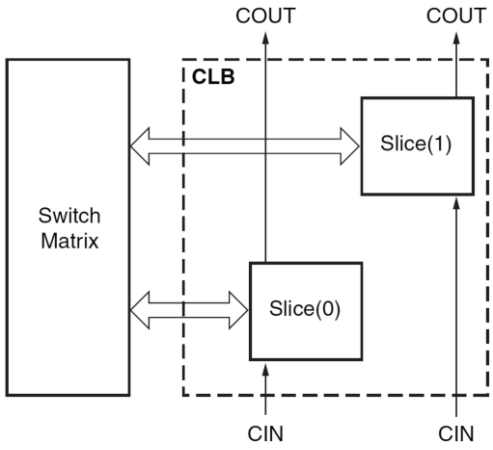



**2.4. ábra. A Virtex-6 FPGA általános felépítése**

A Xilinx sorozatú FPGA-k fő építőeleme a Konfigurálható Logikai Blokk (CLB, Configurable Logic Block), mely segítségével a különböző logikai funkciók valósíthatók meg, és az Input/Output Blokk (IOB, Input/Output Block), mely az FPGA és a külvilág közötti kapcsolattartást biztosítják. Emellett az Xilinx FPGA-k tartalmaznak még speciális blokkokat is. Ezek a blokkok a Virtex-6 FPGA estében a DSP48E1 Slice, a RAMB18E1 BlokkRAM memória, a RocketIO Soros Adó-Vevő, a 10/100/1000 Ethernet MAC (Media Access Control), és PCIe (Peripheral Component Interconnect Express) interface.

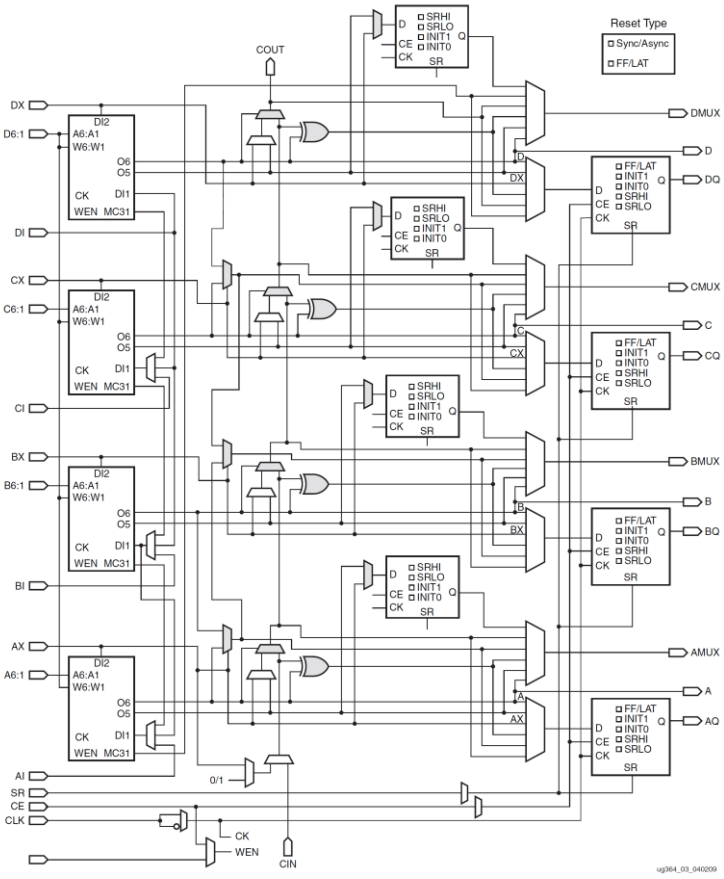
### *2.3.2. A Virtex-6 CLB felépítése*

A Virtex-6 FPGA-ban található Konfigurálható Logikai Blokk az FPGA fő logikai erőforrása, mely segítségével megvalósíthatók a különböző szekvenciális és kombinációs logikai áramkörök [12]. Minden CLB blokk az FPGA-ban található programozható összeköttetés hálózathoz kapcsolódik egy Kapcsoló Mátrixon (Switch Matrix) keresztül, ahogyan ez a 2.5. ábrán is látható.




**2.5. ábra. A Virtex-6 CLB felépítése [12]**

Minden CLB két Slice-ot tartalmaz, melyek nem kapcsolódnak közvetlenül egymáshoz, és oszlopokba vannak rendezve. Minden egyes oszlopban a Slice-ok között egy carry lánc található, mely felhasználható például nagyméretű összeadók nagysebességű implementációjához. Minden egyes Slice négy darab 6 bemenetű LUT-ot, nyolc darab tároló elemet, multiplexereket, és carry logikát tartalmaz. Ezek az elemek felhasználhatók különböző logikai, aritmetikai és ROM funkciók megvalósításához. Egy Virtex-6 Slice felépítése látható a 2.6. ábrán.



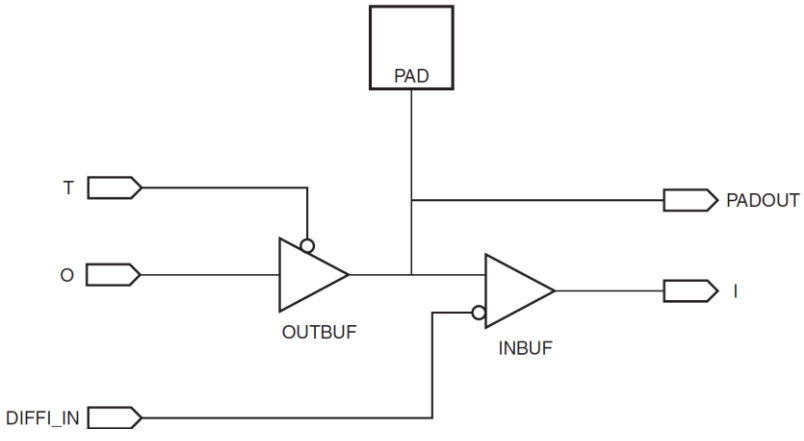

**2.6. ábra. A Virtex-6 CLB felépítése [12]**

A Slice-okban található 6-bemenetű LUT-ok segítségével egy tetszőleges 6-bemenetű logikai függvény, vagy két tetszőleges 5-bemenetű logikai függvény valósítható meg. Emellett a LUT-ok segítségével megvalósítható 32 bites shift regiszter vagy 64 bites elosztott RAM (Distributed RAM). A Slice-ban található 2:1 multiplexerek segítségével a 6-bemenetű LUT-ok összekapcsolhatók és így tetszőleges 7 vagy 8 bemenetű logikai függvények, 128 bites vagy 256 bites elosztott RAM-ok, 8:1/16:1 multiplexerek is megvalósíthatók. Megjegyzendő, hogy a Virtex-5 FPGA sorozat előtti FPGA-k Konfigurálható Logikai Blokkja 4 darab Slice-ot tartalmazott, és a Slice-okban található LUT-ok 4 bemenetűek voltak.

### *2.3.2. A Virtex-6 IOB felépítése*

Minden Virtex-6 FPGA tartalmaz nagy teljesítményű konfigurálható SelectIO driver-eket és receiver-eket, melyek a szabványos interface-ek széles skáláját támogatják [12]. A SelectIO által támogatott funkciók között megtalálhatjuk a kimeneti erősség és felfutási idő (slew rate) programozható vezérlését, vagy a digitálisan vezérelhető impedanciát (Digitali Controlled Impedance, DCI).

Minden IOB tartalmaz mind bemeneti, mind kimeneti, mind három-állapotú SelectIO driver-eket. Ezek a driver-ek a különböző I/O szabványoknak megfelelően konfigurálhatók, támogatva mind a single-ended (LVCMOS, HSTL, SSTL), mind pedig a differenciális jelkezelést (LVDS, HT, LVPECL, BLVDS, differenciális HSTL és SSTL). AVirtex-6 IOB felépítése a 2.7. ábrán látható.

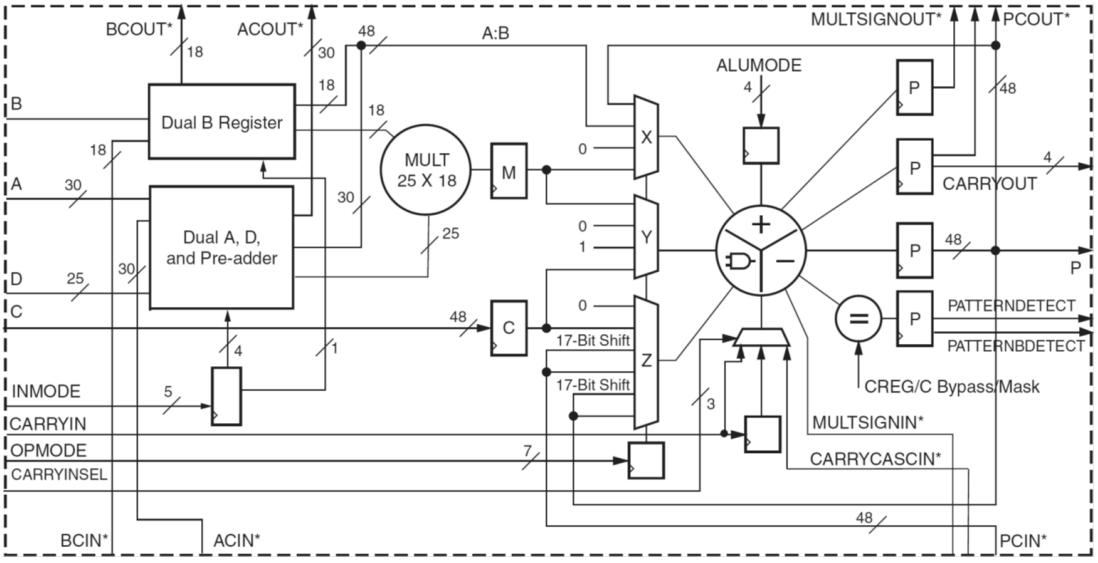



**2.7. ábra. A Virtex-6 IOB felépítése [12]**

### *2.3.3. A Virtex-6 DSP48E1 Slice*

A digitális jelfeldolgozási feladatok megoldásához számos fixpontos szorzó és akkumulátor szükséges, melyek leghatékonyabban a direkt erre a feladatra optimalizált

DSP Slice-okban implementálhatók [12]. Minden Virtex-6 FPGA számos ilyen dedikált DSP Slice-ot tartalmaz. Alapvetően minden DSP48E1 Slice tartalmaz egy dedikált $25\times18$ bites kettes komplemens szorzót és egy 48 bites akkumulátort, melyek maximális működési frekvenciája 600MHz. Nagyobb méretű szorzók megvalósításához a DSP48E1 Slice-ok összefűzhetők. A szorzó dinamikusan átkapcsolható áteresztő módba, és így két 48 bites bemeneten keresztül küldhetők adatok egy aritmetikai egységre vagy egy logikai egységre, mely 10 logikai függvényt képes végrehajtani a beérkező adatokon. Az aritmetikai egység egy SIMD egység, mely használható mint két 24 bites vagy négy 12 bites összeadó/kivonó/akkumulátor. A DSP48E1 Slice tartalmaz még egy pre-adder áramkört is, melyet tipikusan szimmetrikus szűrőkben alkalmazhatnak. A DSP48E1 Slice fent említett tulajdonságai mellett nagyfokú párhuzamosítást is támogat a pipelene regiszterei segítségével. A DSP48E1 Slice felépítése a 2.8. ábrán látható.

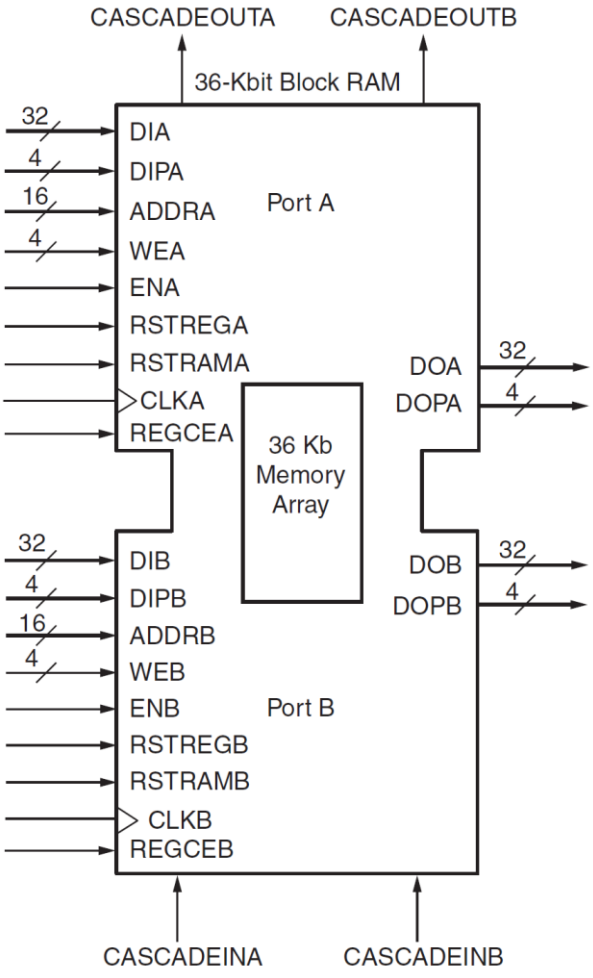



**2.8. ábra. A Virtex-6 DSP48E1 Slice felépítése [12]**

Megjegyzendő, hogy külön dedikált DSP blokkok csak a Virtex-4 FPGA sorozattól voltak elérhetőek a Xilinx FPGA-kban. A korábbi Virtex sorozatok esetében nem voltak külön dedikált DSP Slice-ok, csak $18\times18$ bites dedikált szorzók. A $25\times18$ bites DSP48 Slice pedig csak a Virtex-5 FPGA családtól kezdve volt elérhető. A Virtex-4 FPGA-kban hasonló felépítésű $18\times18$ bites szorzókat tartalmazó XtremeDSP Slice-ok voltak integrálva.

### *2.3.4. A Virtex-6 RAMB36E1 BlokkRAM*

A Virtex-6 FPGA-ban megtalálható RAMB36E1 egy 36Kbites dual-portos BlokkRAM, mely kizárólag szinkron működésre képes [12]. A BlokkRAM minden író/olvasó port-ja egymástól teljesen függetlenül konfigurálható, mint $32K\times1$, $16K\times2$, $8K\times4$, $4K\times9$, $2K\times18$, $1K\times16$, $512\times72$ bites memória port. Minden BlokkRAM felosztható két egymástól teljesen független 18 Kbit-es RAM-ra, melyek esetében a port-ok konfigurálhatók $16K\times1$ bittől egészen $512\times36$ bitig. Az utolsó beállítás csak SDP (Simple Dual Port) módban lehetséges. Ebben a módban az egyik port csak olvasásra, a másik port csak írásra használható, valamint írás közben a RAM nem olvasható. Két egymással szomszédos 36Kbites RAM, mindenféle további logikai erőforrás felhasználása nélkül összefűzhető $64K\times1$ bit méretű dual-portos RAM-ba. A beépített FIFO (First In First Out) vezérlő segítségével a RAMB36E1 BlokkRAM-ok használhatók, szinkron vagy aszinkron FIFO-ként. A RAMB36E1 memória a 2.9. ábrán látható.




**2.9. ábra. A Virtex-6 RAMB36E1 BlokkRAM memória [12]**

Megjegyzendő, hogy a 36 Kbit-es RAMB36 blokkok csak a Virtex-5 sorozattól elérhetőek el. A korábbi Virtex sorozatok esetében 18 Kbit méretű BlokkRAM memóriák álltak a rendelkezésünkre.

### *2.3.5. RocketIO soros adó-vevő*

Az integrált áramkörök közötti nagy sebességű soros átvitel jelentősége egyre növekszik. Ehhez azonban dedikált chip-be épített áramkörökre van szükség [12]. A Virtex-6 FPGA család tagjai 8-72 darab gigabites soros adó-vevőt tartalmaznak. Minden GTX adó-vevő egy adó és egy vevő részből áll, melyek maximális sávszélessége 480 Mb/s-6.6 Gb/s. A GTH adó-vevő is egy adóból és egy vevőből áll, melyek maximális sávszélessége 9.95 Gb/s-11.18 Gb/s. A GTX adó és vevő két független áramkör, melyek külön PLL-t (Pulse Locked Loop) használnak a referencia órajel felszorzásához, hogy előállítsák a soros bitfolyam órajelét. A szorzó értékek programozhatóan állíthatóak 4 és 25 között. A GTH adó-vevőket 10 Gb/s-os átviteli sávszélességekre tervezték és egy nagy teljesítményű PLL-en négy adó és négy vevő áramkör osztozik. Minden GTX és GTH adó-vevőnek számos felhasználó által beállítható paramétere és tulajdonsága van. Ezek között vannak a felprogramozáskor, illetve működés közben is állítható paraméterek.

### *2.3.6. PCIe interface*

A PCI Express szabvány egy csomag alapú pont-pont közötti soros interface szabvány [12]. Differenciális jelátvitelt és beágyazott órajelet alkalmaz, mely segítségével kiküszöbölhetők az adat és az órajel közötti eltolódások, melyek jellemző problémái a hagyományos széles párhuzamos buszoknak. A PCIe szabvány kétirányú, vonalanként 2.5 Gb/s vagy 5 Gb/s átviteli sávszélességet definiál.

Minden Virtex-6 FPGA (kivéve az XCV6LX760) tartalmaz egy integrált PCIe blokkot, mely konfigurálható a PCIe szabványban definiáltaknak megfelelően úgynevezett EndPoint és RootPort módban. Ez a beépített blokk a rendszer követelményeinek megfelelően nagymértékben konfigurálható és működtethető 1, 2, 4 és 8 vonalon, 2.5 Gb/s-os vagy 5 Gb/s-os adatátviteli sebességgel. A nagyteljesítményű alkalmazások számára a fejlett pufferelési technika biztosítja a flexibilisen állítható csomagméretet, mely maximálisan 1024 byte lehet. A Xilinx cég a PCIe blokk használatához LogiCORE wrapper-t biztosít, mely lehetővé teszi a felhasználó számára a különböző paraméterek beállítását.

### *2.3.7. 10/100/1000 Ethernet vezérlő*

Az integrált tri-mode Ethernet MAC egyszerűen hozzákapcsolható az FPGA logikához, a GTX adó-vevőkhöz, és a SelectIO erőforrásokhoz [12]. Minden Virtex-6 FPGA négy darab ilyen Ethernet MAC blokkot tartalmaz, mely az OSI protokoll stack adatkapcsolati rétegét valósítja meg. A Xilinx CORE Generator szoftver segítségével könnyen konfigurálható az interface a GTX adó-vevő, a SelectIO, az FPGA logika vagy egy mikroprocesszor között.

### *2.3.8. MicroBlaze Processzor*

A Xilinx cég által gyártott FPGA-k esetében lehetőség van beágyazott processzorok használatára [12]. Ezen processzorok segítségével nagy bonyolultságú vezérlési feladatok is megoldhatóak, mint például a TCP/IP stack (Transmission Control Protocol/Internet Protocol), az USB (Universal Serial Bus) kommunikáció, vagy a DVI (Digital Visual Interface) videó vezérlés. Ezeknek a beágyazott processzoroknak alapvetően két típusa van. Ezek a hard és a soft core processzor magok. A hard core processzor magok IBM PowerPC 405 magok, ezek használatára azonban a Virtex-6 családtól kezdve nincs lehetőség. A soft core processzor mag a megfelelő Xilinx eszközök segítségével az FPGA belső erőforrásai segítségével kerül kialakításra. Ilyen soft core processzor mag a Xilinx MicroBlaze processzor is. A MicroBlaze processzor egy 32-bites RISC (Reduced Instruction Set) processzor.

A MicroBlaze processzor főbb paraméterei:

-32 darab 32 bites általános célú regiszter,

- 3 operandusú 32 bites utasítás szó, kétféle címzési móddal,
- 32 bites cím busz,
- egyszeres pipeline támogatás.

Ezen fix funkciók mellett a MicroBlaze processzor számos paraméterében állítható az adott alkalmazásnak megfelelően.

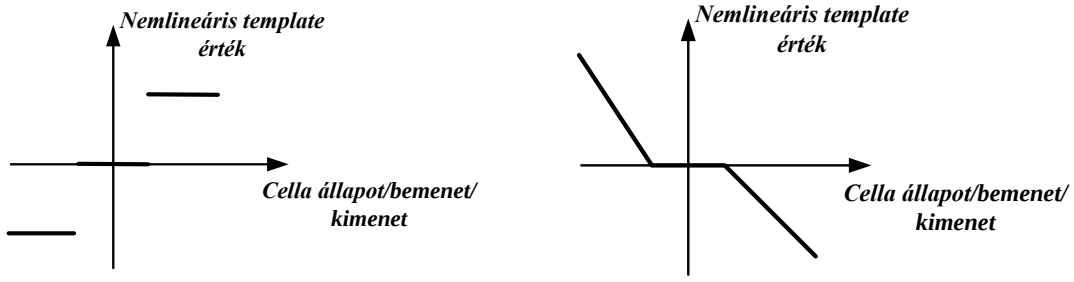
# 3. Nemlineáris template futtató emulált digitális CNN-UM megvalósítása FPGA-n

A fizika, kémia tér-idő dinamikával jellemezhető effektusai között fontos szerepet játszanak azok a problémák, ahol a kimenet-, a bemenet-, és az állapotváltozók kapcsolata nemlineáris. Ilyen problémák például a nem viszkózus folyadékok áramlását leíró Euler, vagy a viszkózus folyadékok áramlását leíró Navier-Stokes nemlineáris parciális differenciálegyenletek. Habár már számos tanulmány bizonyította a CNN hatékonyságát a különböző parciális differenciálegyenletek megoldásában [27][28], a fenti és az ezekhez hasonló problémák CNN alapú megoldásához nemlineáris template-ek alkalmazására van szükség. Emellett számos olyan CNN-el megoldható képfeldolgozási feladat is van, melyeket csak nemlineáris template-ek alkalmazásával lehet hatékonyan megoldani. Ezek között vannak olyan feladatok, mint például a gradiens intenzitásbecslés vagy a szürkeskálás kontúrdetektálás, melyek esetében az alkalmazandó nemlineáris B template-ek helyettesíthetőek megfelelő lineáris template-ek halmazával [29]. Természetesen ez a helyettesítés, még ha ugyanolyan eredménnyel is szolgál, bonyolultabb és végrehajtási ideje hosszabb, mint egyetlen nemlineáris template alkalmazása. Vannak azonban olyan feladatok is, mint például a mediánszűrés, vagy a szürkeskálás erózió/dilatáció, hisztogramgenerálás, melyek esetében nemlineáris A vagy D template-et kell használni, ami azonban nem helyettesíthető lineáris template-ek halmazával. Így ezeket a feladatokat csak nemlineáris template-ek alkalmazásával lehet megoldani, azonban a jelenlegi CNN implementációk esetén - kivéve a lassú szoftveres szimulációt - nincs lehetőség a nemlineáris template-ek alkalmazására.

## *3.1. A nemlineáris template-ek csoportosítása*

Ahogyan azt korábban is említettem, a nemlineáris template-ek abban különböznek a lineáris template-ektől, hogy esetükben a template értékek nem konstansok, hanem az aktuális cella valamely változójának (1.2 alfejezetet 1-es és 2-es példa), vagy pedig az aktuális és a szomszédos cella valamely változóinak különbségének nemlineáris függvényei (1.2 alfejezet 3-as, 4-es és 5-ös példa). Emellett a nemlineáris template-ek esetén a nemlinearitás alapján meghatározott nemlineáris template érték már a hozzá tartozó cella állapotának, bemenetének, vagy kimenetének template-el súlyozott értékét jelentik [4].

A CNN Template Library v3.1 [5] tanulmányozása során a nemlineáris template-eket - a template nemlineáris értékeit meghatározó nemlinearitás alapján - két csoportba soroltam. Ezek az úgynevezett nullad- és elsőrendű nemlineáris template-ek. A nulladrendű nemlineáris template-eknek nevezzük azokat a template-eket, amelyek olyan szakaszokból épülnek fel, amelyeken belül a függvény értéke konstans, ahogyan az a 3.1. ábra a.) részén látható. Az elsőrendű nemlineáris template-ek közé olyan nemlineáris template-ek tartoznak, amelyekben a nemlinearitás olyan szakaszokat is tartalmaz, ahol a függvény értéke nem konstans, hanem a független változó lineáris függvénye, a megfelelő meredekséggel, ahogyan az a 3.1. ábra b.) részén látható.




a.) Nulladrendű nemlinearitás b.) Elsőrendű nemlinearitás

**3.1. ábra. A nullad- és elsőrendű nemlinearitás**

A CNN Template Library v3.1 [5] alapvetően olyan nemlineáris template-eket tartalmaz, amelyek esetében vagy az A, vagy a B, vagy a D template nemlineáris, illetve tartalmaz olyan nemlineáris template-eket is melyek esetében az AB vagy az AD template-ek nemlineárisak. A nemlineáris template-ek csoportosítása az 3-1. táblázatban látható.

| | A template | B template | D template | A,B template | B,D template |
|---|---|---|---|---|---|
| **Nulladrendû nemlinearitás** | GameofLifeDTCNN1<br>HistogramGeneration<br>MajoritiyVoteTaker (Second Template) | ContourExtraction<br>GrayscaleDiagonal-LineDetector<br>GrayscaleLine-Detector<br>LocalMaxima-Detector<br>PatternMatching-Finder<br>Grayscale Skeletonization (Selection)<br>Hamming Distance Generation (Difference) | GrayscaleDilatation<br>GrayscaleErosion<br>ImageInPainting<br>MedianFilter | ParityCounting1 | J-Function Shortest Path (Minimum Selection) |
| **Elsõrendû nemlinearitás** | 1-DArraySorting<br>GlobalMaximum-Finder<br>LaplacePDESolver<br>PoissonPDESolver<br>SpikeGeneration4<br>Grayscale Skeletoinzation (Replacement)<br>Hamming Distance Generation (Minimum Distance)<br>Nonlindear Wave Metric Computation (Current Filling)<br>Shortest Path (Explore)<br>Depth Calculation (Depth) | GradientDetection<br>GradientIntensity-Estimation<br>Rotation<br>Translation(dx,dy)<br>Thresholded-Gradient<br>Shortest Path (Select) | ParityCounting2 | Complex-Gabor<br>Two-Layer Gabor | |

**3-1. táblázat. A nemlineáris template-ek csoportosítása**

Az 3-1. táblázatban látható, hogy sok olyan CNN operátor van, amelyek csak nullad- vagy elsőrendű nemlineáris B template-et tartalmaznak. Ezen operátorok esetében a B template - ahogyan azt korábban is említettem - helyettesíthető lineáris B template-ek egy megfelelő sorozatával [29]. Azonban sok olyan CNN operátor is van, melyek esetében az A vagy D template nemlineáris. Ezek alkalmazására nincs más lehetőség, csak a szoftveres szimuláció, ami egy asztali számítógép esetében igen lassú lehet. Ebből következően célszerű egy olyan megoldás megvalósítása, mely a szoftveres szimulációnál lényegesen gyorsabban képes a nemlineáris template operátorok végrehajtására. Egy jó alternatíva a FALCON emulált digitális CNN-UM architektúra átalakítása úgy, hogy képes legyen a nemlineáris template-ek kezelésére.

## *3.2.A FALCON processzor átalakítása*

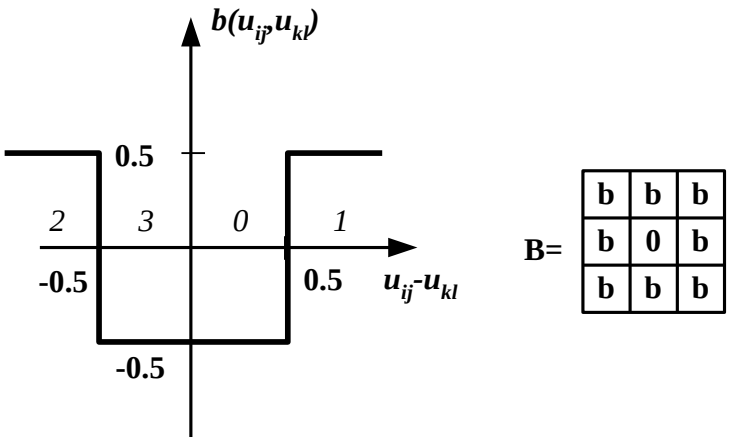
Annak érdekében, hogy az 1.5.2 alfejezetben bemutatott FALCON architektúrát alkalmassá tegyem nemlineáris template-ek futtatására, a *Memória Egységen* és a *Mixer Egységen* nem kell változtatni, azonban az *Aritmetikai Egységet* és a *Template Memóriát* módosítani kell a FALCON processzor vezérlésével együtt. A nemlineáris template-ek esetén a nemlineáris template értéket a cella valamely paraméterének vagy pedig az aktuális cella és a szomszédos cella valamely paraméterének különbségének nemlineáris függvénye alapján határozzuk meg. Ezért a *Template Memóriában* nemcsak a *Memória Egységből* érkező adatok szükségesek, hanem a *Mixer Egység* felől érkező adatok is. Mivel a nemlineáris template-ek esetében a nemlinearitás alapján meghatározott nemlineáris template érték már a hozzá tartozó cella állapotának, bemenetének, vagy kimenetének template-el súlyozott értékét jelentik, ezért szükséges a lineáris template-ekhez használt *Aritmetikai Egység* egyszerűsítése is. A következő alfejezetekben bemutatom, hogy hogyan kell átalakítani az eredeti lineáris FALCON processzor *Template Memóriáját*, *Aritmetikai Egységét* és vezérlését annak érdekében, hogy képes legyen kezelni mind a nullad-, mind pedig az elsőrendű nemlineáris template-eket. A bemutatott egységek $3\times3$-as template-ek kezelésére alkalmasak, azonban mind a *Template Memória*, a *Mixer Egység* és az *Aritmetikai Egység* is bővíthető annak érdekében, hogy nagyobb méretű például $5\times5$-ös template-ek is futtathatóak legyenek.

### *3.2.1. Nulladrendű nemlineáris template memória*

Az eredeti FALCON processzorban a template műveletek soronként hajtódtak végre. A *Template Memóriában* egy kétdimenziós RAM tartalmazza a template-ek minden egyes oszlopához a template értékeket. Ez a RAM egyszerűen implementálható, mint egy regiszter tömb. Az aktuális template értékek innen kerülnek továbbításra az *Aritmetikai Egység* bemenetére. A nulladrendű nemlineáris template futtató emulált digitális FALCON architektúrában található *Template Memóriában* is a nemlineáris template értékeket tartalmazó RAM a legfontosabb. Mivel ez a RAM nagyméretű is lehet az alábbiakban bemutatott okok miatt, érdemes ezt a RAM-ot dedikált memóriaelemekként implementálni. A RAM oszlopainak számát a template mérete adja meg. A sorok számát a template méretének és a nemlinearitás szakaszainak, számának a szorzata adja. Ebbe a memóriába a megfelelő sorrendben és helyre betöltöm a nemlineáris elemek lehetséges értékeit, illetve a konstans értékeket. A továbbiakban ebből a memóriából az

aktuális nulladrendű template értéket úgy kapom meg, hogy az éppen feldolgozás alatt álló cella értékének (bemenet vagy állapot) és a template megfelelő eleméhez tartozó érték különbségével címezem meg a memóriát. Abban az esetben, ha olyan a nulladrendű nemlineáris template, hogy nincs szükség a cella adott paramétereinek különbség képzésére, akkor a kivonó megfelelő bemenetére egyszerűen 0-át vezetek. Mivel a *Memória Egységből* az első érvényes adat a *Mixer Egységnél* három órajel ciklussal hamarabb érkezik meg, ezért ezt először egy shift regiszterbe rakom, majd három órajelet késleltetem ezeket az értékeket, mielőtt kiolvasom belőle.

A könnyebb érthetőség érdekében a fent leírt folyamatot egy példán keresztül mutatom be. Legyen az előrecsatoló B template egy nulladrendű nemlineáris template. A B template és a hozzátartozó nemlinearitás a 3.2. ábrán látható.



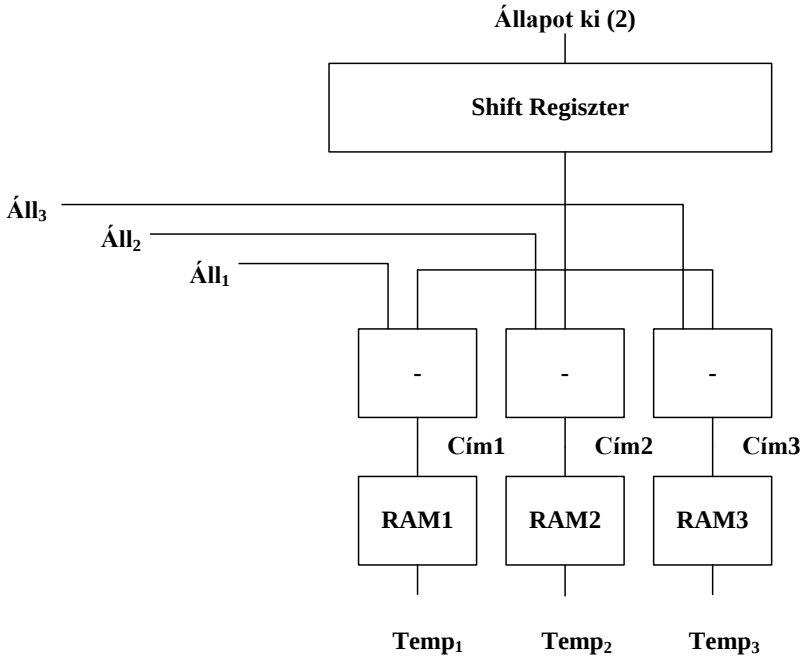

**3.2. ábra. A nulladrendű nemlineáris B template**

Ebben az esetben a nemlineáris template értékeit tároló memória tartalma a 3.3. ábrán látható. A memória feltöltése során a nemlinearitást négy részre osztom és a nemlinearitás szakaszait a 3.3. ábrán látható módon töltöm be a memóriába. Azért járok el így, mert a kiolvasás során a fentiekben tárgyalt módon kapott különbséget (vagy a cella aktuális paraméterét) eltolom, úgy hogy jelen esetben csak a két legmagasabb helyi értékű bit maradjon meg és az így kapott értékkel címzem a memóriát. Általános esetben a nemlinearitás szakaszainak felosztását kettő tetszőleges hatványával végezhetjük, ilyenkor természetesen a különbség bitjeiből vagy a cella aktuális paraméteréből is több MSB (Most Significant Bit) bitet kell megtartani.

| A template értékek elhelyezkedése a RAM-ban | | | |
|---|---|---|---|
| **Cím** | **Oszlopok** | | |
| | ***0*** | ***1*** | ***2*** |
| ***0*** | **-0.5** | **-0.5** | **-0.5** |
| ***1*** | **0.5** | **0.5** | **0.5** |
| ***2*** | **0.5** | **0.5** | **0.5** |
| ***3*** | **-0.5** | **-0.5** | **-0.5** |
| ***4*** | **-0.5** | **0** | **-0.5** |
| ***5*** | **0.5** | **0** | **0.5** |
| ***6*** | **0.5** | **0** | **0.5** |
| ***7*** | **-0.5** | **0** | **-0.5** |
| ***8*** | **-0.5** | **-0.5** | **-0.5** |
| ***9*** | **0.5** | **0.5** | **0.5** |
| ***10*** | **0.5** | **0.5** | **0.5** |
| ***11*** | **-0.5** | **-0.5** | **-0.5** |

**3.3. ábra. A nemlineáris template értékek elhelyezkedése a RAM-ban**

Ebből a memóriából tehát a fent leírtak alapján olvasom ki a szükséges template értéket. Egy hozzáférés során egyszerre három template értéket veszek ki a *Template Memóriából*. A *Mixer Egység* felől érkező adatokat egy shift regiszterben addig várakoztatom, amíg az első három érvényes template értéket meg nem kapom a nemlineáris *Template Memóriából*. Az így megkapott adatokat egyszerre az *Aritmetikai Egység* megfelelő bemeneteire vezetem. A nulladrendű nemlinearitást kezelő *Template Memória* a 3.4. ábrán látható. Az $Áll_1$, $Áll_2$, $Áll_3$ értékek a *Mixer Egység* felől érkező értékek, azaz a szomszédos cellák bemenete, vagy pedig állapota. Az *Állapot ki (2)* a *Memória Egység* középső sora, amely az éppen feldolgozás alatt álló cellához kapcsolódó megfelelő értékeket tartalmazza. A *Shift Regiszter* az adatok időzítéséért felelős, azaz hogy az $Áll_1$, $Áll_2$, $Áll_3$ és az *Állapot ki (2)* értékek egyszerre érkezzenek a kivonók bemeneteire.

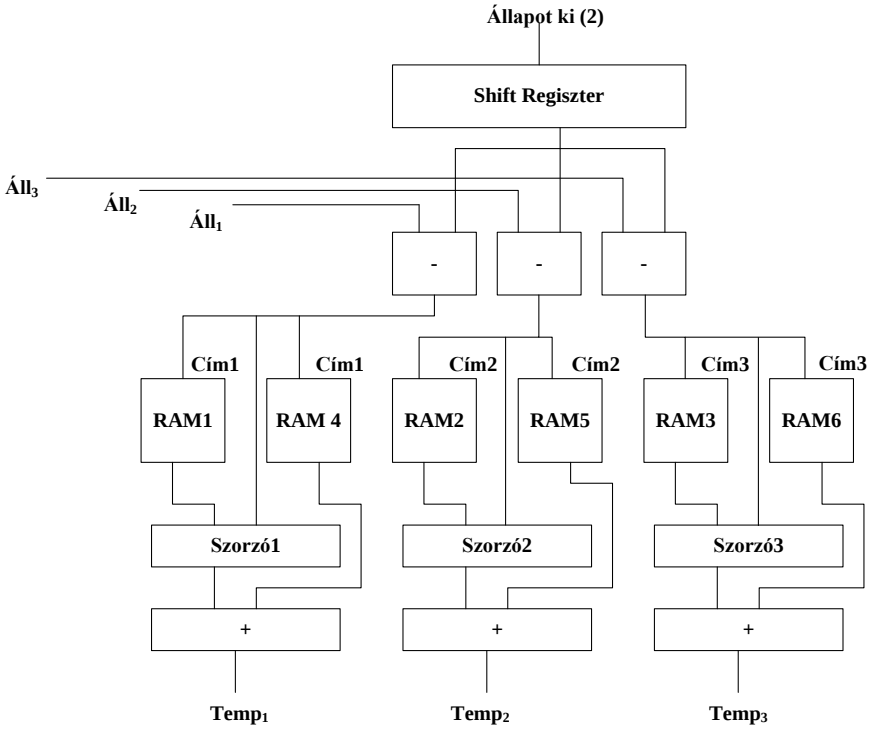



**3.4. ábra. A nulladrendű nemlinearitást kezelő *Template Memória* felépítése**

### *3.2.2. Az elsőrendű nemlineáris template memória*

Az elsőrendű nemlineáris template-eket kezelő *Template Memória* esetében is a nemlinearitást kezelő RAM játssza a főszerepet, amelyből itt kettőt használok. Természetesen ezeket a RAM-okat is célszerű memóriaelemként implementálni. A RAM-ok sorainak és oszlopainak száma megegyezik a nulladrendű *Template Memória* esetén tárgyalt RAM sorainak és oszlopainak számával. A különbség a fent említett RAM és az ebben az esetben használt RAM-ok között a bennük tárolt értékekben van. Mint tudjuk, az elsőrendű nemlinearitás tartalmaz olyan szakaszokat, ahol az értékét egy lineáris függvény írja le, de tartalmaz olyanokat is, amelyeken belül az értékét egy konstans függvény határozza meg. Ennek megfelelően az egyik RAM-ban az adott szakaszhoz tartozó függvény meredekség értékét tárolom el, míg a másikban az adott függvényhez tarozó konstans eltolás értékét. A meredekségeket tároló RAM azon helyeire, amelyek egy olyan szakaszhoz tartoznak, amelyen belül egy lineáris függvény van, oda az adott függvény meredekségét írom be, azokra a helyekre viszont, amelyek olyan szakaszokat írnak le, ahol egy konstans függvény van, oda nulla meredekség értéket írok be. Az ilyen módon feltöltött RAM-okból a már bemutatott módon az éppen feldolgozás alatt álló cella és a template megfelelő eleméhez tartozó cellaérték segítségével, vagy egyszerűen csak az aktuális cella megfelelő paramétere alapján olvasom ki a megfelelő értékeket. A kiolvasott meredekség értékekkel megszorozom az

éppen aktuális különbséget, vagy cella paramétert, és hozzáadom a másik RAM-ból kiolvasott konstanst. Az így kapott érték pedig már maga a keresett nemlineáris template érték. A nemlineáris *Template Memória* a fent leírtakból következően változik és a 3.5. ábrán látható.



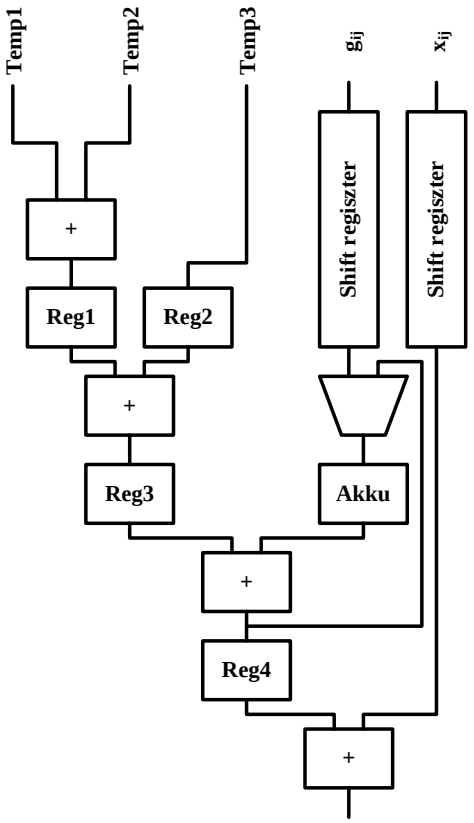

**3.5. ábra. Az elsőrendű nemlinearitást kezelő nemlineáris *Template Memória***

A 3.4. ábra tehát kiegészült a fent leírtaknak megfelelően még egy RAM-mal, egy szorzóval és egy összeadóval.

### *3.2.3. A módosított Aritmetikai Egység*

Az *Aritmetikai Egységet* is át kell alakítani annak érdekében, hogy a FALCON processzor képes legyen a nullad- és elsőrendű nemlineáris template-ek kezelésére. Ez lényegében az jelenti, hogy mivel a *Template Memória* által meghatározott értékek már a cella bemenetének vagy kimenetének nemlineáris template értékkel súlyozott értékei, így az *Aritmetikai Egységben* már nincs szükség szorzásra. Az *Aritmetikai Egység* felépítése a 3.6. ábrán látható. Hasonlóan az eredeti *Aritmetikai Egységhez*, az átalakított egység is négy összeadóból, két shift regiszterből, négy általános és egy akkumulátor (*Akku*) regiszterből, valamint egy multiplexerből épül fel. Az új *Aritmetikai Egység* működése is megegyezik az eredeti FALCON architektúrában használt egységgel, azaz ennek az *Aritmetikai Egységnek* is három órajel-ciklusra van szüksége egy új cellaérték kiszámítására. Az első órajelre az *Aritmetikai Egység*

bemenetén megjelenik a nemlineáris template első sorához tartozó nemlineáris template érték, illetve az aktuálisan feldolgozott cella megfelelő paramétere (állapot, bemenet) és a hozzá tartozó konstans. A következő két órajelben az összeadóba, illetve a *Reg2* regiszterbe betöltődnek a template maradék két sorához tartozó nemlineáris template értékek, valamint a megfelelő állapotértékek. Két órajel szükséges a nemlineáris template egy sorához tartozó nemlineáris template értékek összeadásához. Az újabb cella feldolgozása a következő órajelben kezdődhet. Amikor az első érték betöltődik a *Reg3*-ba akkor az aktuális cella konstans értékét be kell tölteni az *Akku* regiszterbe. A következő két órajelben az első és második nemlineáris template sorhoz tartozó részeredmények tárolódnak az *Akku*-ban. A végeredmény a *Reg4* regiszterben tárolódik, mert ebben az órajel-ciklusban az *Akku*-ba a következő cellához tartozó konstans érték töltődik be. A folyamat következő lépéseként az állapotérték fentiekben számított derivált értéke hozzáadódik a régi értékhez és az így kapott új állapotértéket a szigmoid függvénnyel limitálom. A konstans érték és a régi cellaérték tárolására szolgáló *Shift Regiszterek* hossza is változik, hisz méretüket egyrészt a *Template Memória*, másrészt pedig az összeadó fa késleltetése határozza meg.




**3.6. ábra. Az átalakított *Aritmetikai Egység* felépítése**

### 3.2.4. *Vezérlés módosítása*

Ahhoz tehát, hogy az eredeti FALCON processzor képes legyen nemlineáris template-ek és a hozzájuk tartozó nemlinearitások kezelésére, a fentiekben tárgyalt *Template Memóriák* valamelyikét kell tartalmaznia. Abban az esetben, ha új *Template Memóriával* látom el a processzort, akkor a helyes működés érdekében a vezérlésben is végre kell hajtani kisebb változtatásokat. Az eredeti *Template Memória* alkalmazása esetén az őt és az *Aritmetikai Egységet* elindító vezérlőjeleket elegendő volt akkor magas szintre állítani, amikor az első érvényes adat megjelent a *Mixer* kimenetén. Azonban az új *Template Memóriákat* már akkor el kell indítani, amikor az első érvényes adat megjelenik a *Memória* kimenetén, azaz három órajellel hamarabb, hisz a *Mixerből* érkező adatok mellett a *Memória Egység* középső *Shift Regiszterében* található értékre (az éppen feldolgozott cella megfelelő paraméterének értéke) is szükségem lesz a címképzésnél. A nulladrendű nemlineáris *Template Memória* esetén a megfelelő template érték meghatározásához az eredeti memóriánál két órajel ciklussal több kell, hisz itt képezni kell a címzéshez használt különbségeket, majd ezek segítségével kiolvasni a megfelelő template értékeket, így az *Aritmetikai Egység* indításáért felelős vezérlőjelet két órajellel később kell magas szintre állítani. Az elsőrendű nemlineáris *Template Memória* esetén az indítás a nulladrendű *Template Memóriával* azonos időpontban történik, viszont ebben az esetben még két órajel kell, hisz itt szorzást és összegzést is végezünk a template értékek meghatározása érdekében, így még két órajellel később indítható csak az *Aritmetikai Egység*. A feldolgozási időt nem módosítja, viszont mégis meg kell említeni, hogy az új típusú *Template Memóriák* esetében a RAM-ok feltöltéséhez is több idő kell, hisz nagyobb méretűek, így a RAM-ok feltöltéséért felelős vezérlőjelet az eredetihez képest tovább kell magas szinten tartani.

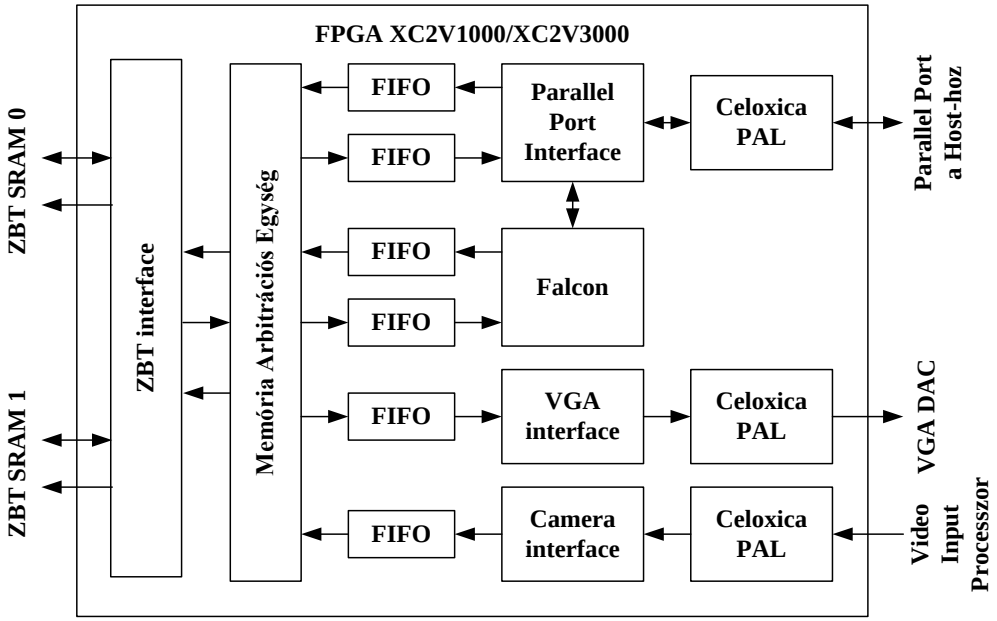
## 3.3. *Tesztelés*

A következőkben megvizsgálom az általam létrehozott új típusú *Template Memóriával* rendelkező FALCON processzorok helyigényét és a velük elérhető képfeldolgozási sebességet, különböző bitszélességű és pontosságú állapot és template értékek esetén, különböző FPGA-kat felhasználva.

### *3.3.1. A FALCON processzor megvalósítása FPGA-n*

A módosított és az eredeti FALCON processzort a Celoxica RC203-as fejlesztőkártyán valósítottam meg, a processzor tesztelése céljából. A kártyán egy Virtex-II 3000-es típusú FPGA, 4MB SRAM található és párhuzamos port-on keresztül kapcsolódik a számítógéphez. A FALCON processzor implementálásához a Handle-C magas szintű hardver leíró nyelvet és a Celoxica DK Design Suite fejlesztő környezetet használtam.

Ahhoz, hogy a FALCON processzort megvalósíthassam a fentiekben ismertetett FPGA-n, az általam készített kódból egy EDIF formátumú kapuszintű leírást szintetizálok a Celoxica DK Design Suite segítségével, amelyből aztán a megfelelő Xilinx Tool-ok segítségével generálok egy konfigurációs fájlt, amely segítségével a megfelelő módon fel lehet konfigurálni az FPGA-t és így már fizikailag is a létrejöhet a processzor. Ha azonban csak magát a processzor átfordított kódját tölteném fel, akkor a processzor egyáltalán nem működne, hisz szüksége van az FPGA-n lévő memória elérését, a párhuzamos port-hoz való hozzáférést továbbá a videó bemenet, valamint kimenet elérését biztosító interface-ekre. A kártyán lévő memória eléréséhez a számítás részeredményeinek eltárolása miatt, a párhuzamos port-hoz való hozzáféréshez a processzor vezérlése miatt, míg a videó bemenet és a videó kimenet elérésére pedig a kép beolvasása, illetve az eredménykép megjelenítése miatt van szükség. Ezen interface-ek bemutatására nem térek ki, hisz ez nem tartozik jelen disszertáció keretei közé, illetve ezeket az interface-eket nem én valósítottam meg, hisz ezek már a munkám kezdetén is rendelkezésemre álltak. A teljes, az FPGA-n is működni képes architektúra a 3.7. ábrán látható.




**3.7. ábra. A FALCON processzor és a működéséhez szükséges elemek**

A 3.7. ábrán látható *ZBT* (Zero-Bus Turnaround) *interface* a kártyán található ZBT SRAM-ok elérését szolgálja. A *Memória Arbitrációs Egység* feladata eldönteni, hogy mikor melyik egység kapja meg a memóriát. A *FIFO*-k feladata, hogy egyeztesse a memória és az egyes funkcionális egységek bitszélességét. A *Parallel Port Interface* feladata, hogy biztosítsa a párhuzamos port elérését. A *VGA interface* a monitorral, a *Camera interface* pedig a kamerával való kapcsolattartást biztosítja.

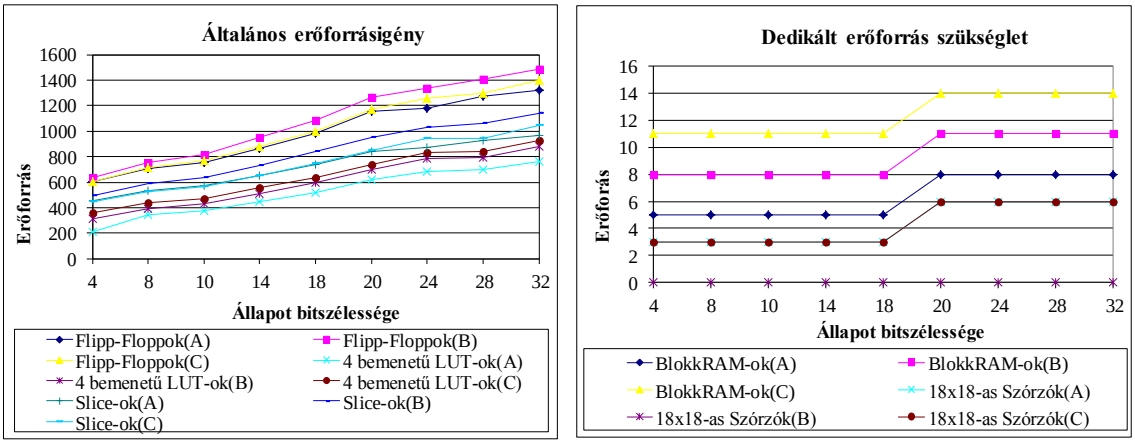
A FALCON processzort a Handle-C nyelv mellett implementáltam a VHDL nyelv segítségével is, melyhez a Xilinx ISE Design Suite 13.2-t (Integrated Software Environment) használtam fel [12]. Az így elkészített processzort több különböző aspektusból, úgymint helyigény, illetve elérhető maximális képfeldolgozási sebesség teszteltem különböző FPGA-k (Virtex-6, Virtex-7) esetében.

### *3.3.2. A helyigény tesztelése*

Az általam létrehozott FALCON processzor fixpontos értékekkel dolgozik és változtatható az állapotérték, az előrecsatoló egyenlet által szolgáltatott $g_{ij}$ és a template értékek bitszélessége, a törtbitjeik száma, valamint az alkalmazható template-ek mérte így a különböző beállítási lehetőségek igen széles skálája áll a rendelkezésünkre. A tesztelés során természetesen nem vizsgáltam meg minden esetet, hisz a célom csupán annak reprezentálása volt, hogy hogyan változik a helyigény a különböző méretű és pontosságú adatok esetén. A konfigurációs file generálása során a Xilinx Foundation ISE Tool-ok kimenete tartalmaz egy statisztikát, amelyben a chip-en rendelkezésre álló

és a processzor által lefoglalt általános (Flip-Flop-ok, 4 bemenetű LUT-ok, Slice-ok) és dedikált (BlokkRAM-ok, $18\times18$ bites szorzók) erőforrások száma található.

A tesztelés során megvizsgáltam mind az eredeti (A), a nulladrendű (B) és az elsőrendű (C) *Template Memóriával* rendelkező FALCON processzort. A tesztben a bementi kép egy $128\times128$ pixelből álló kép volt. A template értékek és a bias bitszélességét 18 bitre, a törtbitjeinek számát pedig 16 bitre állítottam. A bemenő kép pixelei 8 bitesek, így a $g_{ij}$ értéke 31 bit a törtbitjeinek a száma, pedig 29 bit volt. Az alkalmazott template mérete $3\times3$-as volt. Az állapotérték bitszélességét 4 bitről növeltem egészen 32 bitig, míg a törtbitjeinek számát mindig az aktuális állapot bitszélességénél kettővel kisebbre állítottam. Az így kapott eredmény a 3.8. ábrán látható.



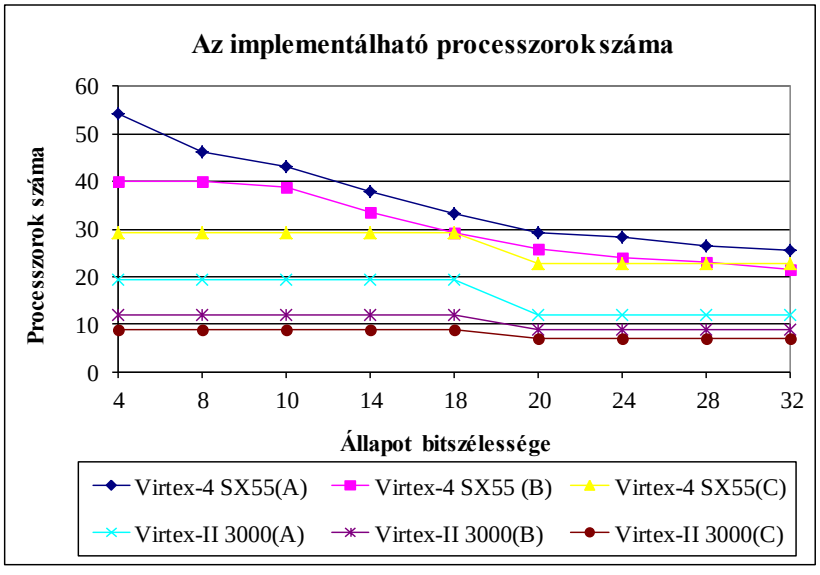

a.) Általános erőforrás szükséglet b.) Dedikált erőforrás szükséglet

**3.8. ábra. A lineáris (A), nulladrendű (B) és elsőrendű (C) FALCON processzor általános és dedikált erőforrás szükséglete**

A teszteset kiválasztásakor azért esett ezekre a beállításokra a választás, mert ezek a legtöbb alkalmazás számára megfelelőnek tűntek. A vizsgálataim megmutatták, hogy az általános erőforrás szükséglet lineárisan függ az állapot bitszélességétől (3.8. ábra a.)). A tesztelés során az is kiderült, hogy a dedikált erőforrás szükséglet szintén lineárisan függ az állapotértékek bitszélességétől (3.8. ábra b.)). Hasonló eredményekre jutottam akkor is, ha a template értékek bitszélességét változtattam. Az eredeti lineáris FALCON processzor 5, a nulladrendű FALCON processzor 8, míg az elsőrendű FALCON processzor 11 darab BlokkRAM-ot használ 18 bitig, és 3-mal több (lineáris 8, nulladrendű 11, elsőrendű 14) BlokkRAM-ot 18 bites állapotszélesség felett. A felhasznált BlokkRAM-ok száma közötti különbség abból adódik, hogy a nullad- és az elsőrendű FALCON processzor esetében a *Template Memóriában* BlokkRAM-ra van

szükség a template értékeinek eltárolásához. A lineáris FALCON processzor esetében egyszerűen csak egy regiszter tömböt alkalmaztam a *Template Memória* implementációja során. A három implementáció esetében a felhasznált szorzók számában is van különbség. A nulladrendű nemlineáris FALCON processzor esetében nincs szükség szorzásra, így itt a bitszélességtől függetlenül, nem használok fel 18×18 bites szorzókat A lineáris és elsőrendű nemlineáris *Template Memóriával* rendelkező processzor 18 bitig 3 szorzót, míg 18 bites állapotszélesség felett 6 szorzót használ. A lineáris FALCON processzor esetében a 3 szorzóra az *Aritmetikai Egységben*, az elsőrendű nemlineáris FALCON processzor esetében pedig a *Template Memóriában* van szükség. 18 bites állapotérték felett az általános és a dedikált erőforrás szükségletben tapasztalható ugrás annak köszönhető, hogy az általam használt FPGA-n csak 18×18 bites dedikált szorzók vannak, így ha két 18 bitesnél nagyobb számot akarok összeszorozni, akkor több szorzóra és több általános erőforrásra van szükségem a művelet elvégzéséhez.

A következő lépésben azt vizsgáltam meg, hogy hány darab eredeti lineáris (A), nulladrendű (B) és elsőrendű (C) nemlineáris *Template Memóriával* rendelkező FALCON processzor fér el az általam használt Virtex-II 3000-es FPGA-n, és ezt összehasonlítottam a hozzá architektúrálisan hasonló Virtex-4 SX55-ös FPGA-n elhelyezhető processzorok számával. A vizsgálat eredménye a 3.9. ábrán látható.




**3.9. ábra. Az implementálható FALCON processzorok számának összehasonlítása a Virtex-II 3000 és a Virtex-4 SX55 FPGA-k esetén**

A vizsgálat megmutatta, hogy a Virtex-II 3000-es FPGA esetében mindhárom FALCON processzor esetében az FPGA-ban rendelkezésre álló BlokkRAM erőforrás a

szűk keresztmetszet. A Virtex-4 SX55-ös FPGA esetében pedig a lineáris és a nulladrendű FALCON processzornál a Slice-ok száma, míg az elsőrendű FALCON processzornál a BlokkRAM-ok száma volt a szűk keresztmetszet, tehát az implementálható processzorok számát egyik esetben sem korlátozza az FPGA-kon rendelkezésre álló szorzók száma. Ez azért érdekes, mert nulladrendű FALCON processzor esetében nincs szükség, míg a lineáris és elsőrendű FALCON processzor esetében szükség van szorzók alkalmazására.

### *3.3.3. A sebesség tesztelése*

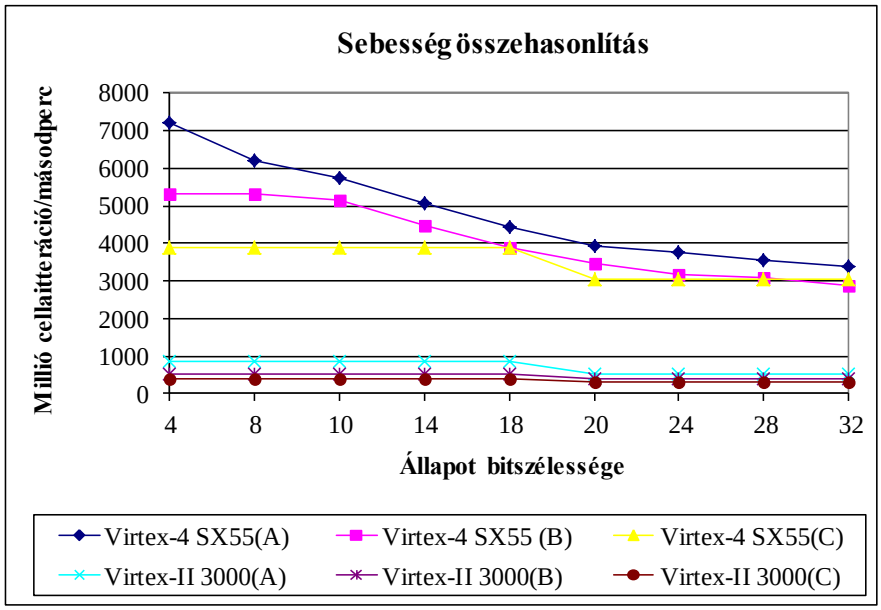
A processzorokkal elérhető számítási sebességet is megvizsgáltam mind a lineáris, mind pedig a kétfajta nemlineáris *Template Memóriával* rendelkező FALCON processzor esetében és összehasonlítottam a MatCNN szoftver szimulációval elérhető sebességgel.

A processzorok által elérhető képfeldolgozási sebesség attól függ, hogy hány darab processzort tudok egymással párhuzamosan futtatni, azaz hogy hány processzor implementálható egy adott FPGA-n. Az általam használt Virtex-II 3000-es FPGA és az összehasonlításként alapul vett Virtex-4 SX55-ös FPGA esetén, mint azt már említettem a szűk keresztmetszet a BlokkRAM-ok illetve a Slice-ok száma volt, hisz csak ez korlátozta, az FPGA-n implementálható processzorok számát . Ebből pedig az következik, hogy a Virtex-II 3000-es FPGA-t használva mind a lineáris, mind pedig a nullad-, és elsőrendű nemlineáris *Template Memóriával* rendelkező FALCON processzorok esetében az elérhető képfeldolgozási sebesség 18 bites pontosság alatt nem függ a feldolgozott adatok bitszélességétől és a törtbitjeinek számától (lásd 3.9. ábra).

A sebességteszt során $128 \times 128$-as képet, 18 bites állapotot és 18 bites template és bias értékeket használtam. A szoftveres szimuláció teljesítményét egy Intel i5 M520 2.53 GHz-es 4 magos processzoron mértem.

A szoftveres szimuláció teljesítménye a lineáris template (Heat Diffusion) esetén 25 millió cellaiteráció/másodperc, a nulladrendű nemlineáris template (Contour Extraction) esetén 13 millió cellaiteráció/másodperc volt, míg az elsőrendű nemlineáris template (Thresholded Gradient) esetén a szimuláció sebessége 9 millió cellaiteráció/másodpercre csökkent. A lineáris FALCON processzorból 19, a nulladrendű FALCON processzorból 12 darab, míg az elsőrendű nemlineáris *Template Memóriával* rendelkező processzorból 9 darab implementálható a Virtex-II 3000-es FPGA-n. Ennek az FPGA-nak a maximális működési frekvenciája 133 MHz, amit az

RC203-as kártyán rendelkezésre álló memóriák órajel frekvenciája határoz meg. A lineáris FALCON processzorral 851 millió cellaiteráció/másodperc, a nulladrendű FALCON processzorral 532 millió cellaiteráció/másodperc, míg az elsőrendű FALCON processzorral 387 millió cellaiteráció/másodperc az elérhető maximális számítási teljesítmény. Ez lineáris esetben 34-szeres sebességnövekedést, a nulladrendű template futtatása esetében 41-szeres, míg az elsőrendű template futtatása esetében 43-szoros sebességnövekedést jelent a szoftveres szimulációhoz képest. A Virtex-4 SX55-ös FPGA szorzói 400 MHz-es működési frekvenciára képesek. Ennél az FPGA-nál a nullad- és elsőrendű nemlineáris *Template Memóriával* rendelkező FALCON processzorból ugyanannyi, azaz 29 darab implementálható, így a velük elérhető maximális számítási sebesség 3879 millió cellaiteráció/másodperc. Ez a nulladrendű esetben 298-szoros, míg az elsőrendű nemlineáris esetben 431-szeres sebességnövekedést jelent. A lineáris FALCON processzorból kicsivel több, azaz 33 darab implementálható ezen az FPGA-n így az ezzel elérhető maximális számítási teljesítmény 4452 millió cellaiteráció/másodperc, ez 178-szoros növekedés a szoftveres szimulációhoz képest. A sebességteszt eredményét összefoglaló grafikon a 3.10. ábrán látható.

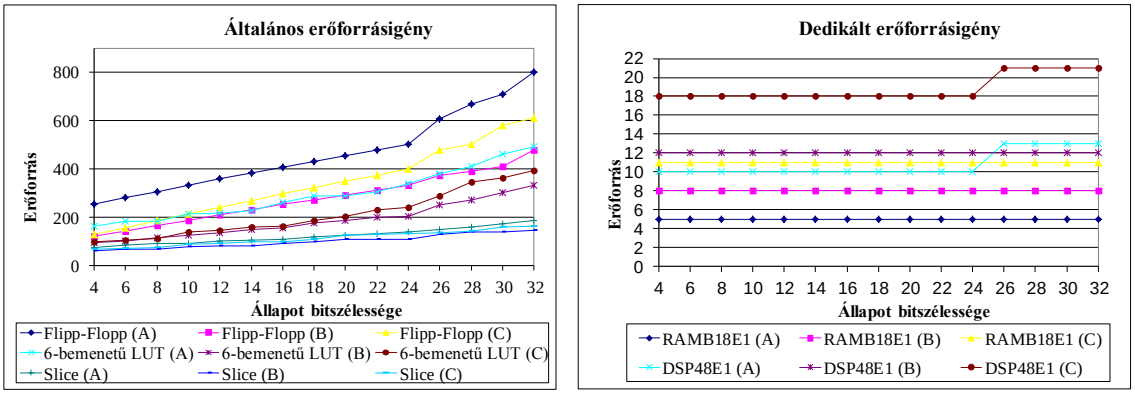



**3.10. ábra. A FALCON processzorokkal elérhető számítási sebesség összehasonlítása a Virtex-II 3000 és a Virtex-4 SX55 FPGA-k esetén**

Emellett megvizsgáltam, hogy a jelenlegi csúcskategóriát képviselő Virtex-6 és Virtex-7 FPGA-k esetében hány darab processzor implementálható (ebből következik a maximális számítási teljesítmény), és mekkora ezen FPGA-k esetében a lineáris, a nullad- és elsőrendű nemlineáris FALCON processzort implementálva az általános és

dedikált erőforrásigény. A tesztek során a Virtex-6 FPGA családból az XC6VSX475T típust, míg a Virtex-7 FPGA családból pedig a XC7VX980T FPGA-t választottam. Az elkészült lineáris, nullad- és elsőrendű FALCON processzor architektúrájának szintézise során a szintézis eszközt úgy állítottam be, hogy kihasználja a Virtex-6/Virtex-7 FPGA-k esetében rendelkezésünkre álló DSP48E1 Slice és RAMB18E1 BlokkRAM dedikált erőforrásokat a lehető legnagyobb sebesség elérése érdekében. A tesztek során ugyanazokat a beállításokat használtam, mint korábban a Virtex-II és Virtex-4 FPGA-kkal végzett tesztek esetében.

A lineáris (A), nullad- (B) és elsőrendű (C) FALCON processzorok általános és dedikált erőforrásigénye a 3.11. ábrán látható a Virtex-6, Virtex-7 FPGA architektúrák esetében. Természetesen nem ábrázoltam külön a két FPGA típus esetében az általános és dedikált erőforrás szükségletet, hisz a két architektúra esetében nincs lényegi különbség.



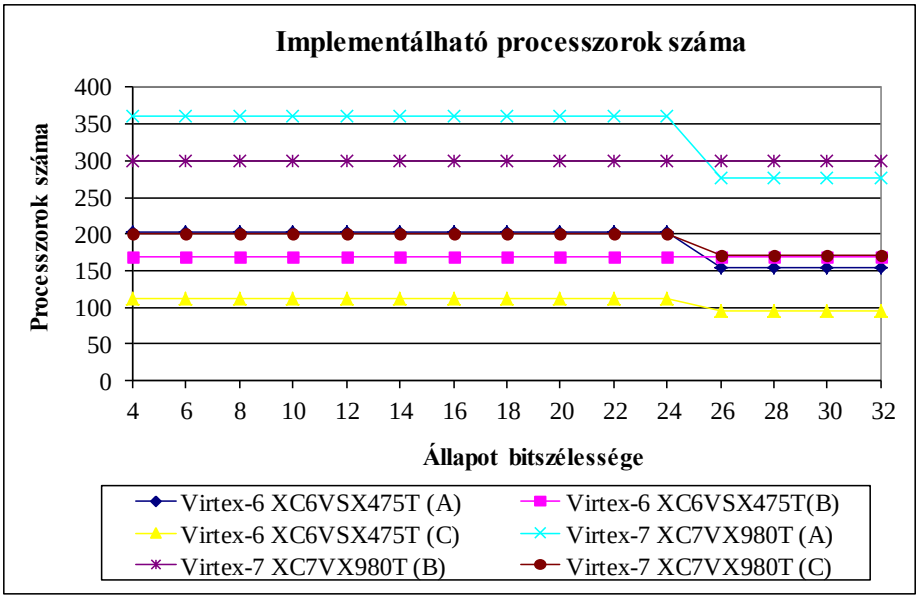

a.) Általános erőforrás igény b.) Dedikált erőforrás igény

**3.11. ábra. A lineáris (A), a nullad- (B), és elsőrendű FALCON processzor általános és dedikált erőforrásigénye a Virtex-6, Virtex-7 FPGA architektúrák esetében**

A vizsgálataim során hasonló eredményekre jutottam, mint a Virtex-II és a Virtex-4 FPGA-k esetében, azaz az általános és dedikált erőforrás szükséglet lineárisan függ az állapot és a template érték bitszélességétől. Mindhárom implementáció esetében a korábbi tesztekkel megegyezően alakul az architektúrák által felhasznált BlokkRAM-ok száma. Van azonban egy különbség: a Virtex-6 és Virtex-7 FPGA-kat használva nincs ugrás a felhasznált BlokkRAM-ok számában 18 bites állapotszélesség felett. Ennek oka a korábban bemutatott RAMB18E1 blokk, mely SDP módban konfigurálható, mint 36 bites bemenetű és 512 elem mélységű BlokkRAM. Ebből következik, hogy az eltárolható legnagyobb érték bitszélessége 36 bit, így az általam végzett tesztek során (maximum 32 bites állapotszélesség) nem volt szükség több ilyen BlokkRAM memória

összefűzésére. A processzorok által felhasznált szorzók, pontosabban a DSP48E1 Slice-ok száma is másképpen alakul, mint a korábbi tesztekben. Mivel a DSP48E1 Slice-okat nem csak a szorzás elvégzésére, hanem összeadás/kivonás/akkumulálás elvégzésére is lehet használni, így mindhárom implementáció esetében felhasználásra kerülnek, a lehető legnagyobb sebesség elérése érdekében. A lineáris FALCON processzor esetében 10, a nulladrendű FALCON processzor esetében 12, míg az elsőrendű FALCON processzor esetében 18 darab DSP48E1 Slice szükséges 25 bitig, és a lineáris valamint az elsőrendű processzorok esetében 3-mal több (13 darab a lineáris és 21 darab az elsőrendű FALCON processzor esetén) 25 bites állapotszélesség felett. A DSP48E1 Slice szükségletben fellépő ugrás a lineáris és az elsőrendű processzorok esetében annak köszönhető, hogy az ebben a Slice-ban lévő szorzók $25\times18$ bitesek, így nagyobb méretű szorzások elvégzéséhez több ilyen Slice-ot kell felhasználni. A nulladrendű FALCON processzor esetében nincsen szükség szorzásra, így a felhasznált DSP48E1 Slice-okat a számítások során szükséges egyéb aritmetikai és logikai műveletek elvégzésére használom, így itt nem is tapasztalható a lineáris és az elsőrendű FALCON processzorok esetében a Slice szükségletben megfigyelhető ugrás.

A Virtex-6 XC6VSX475T, és a Virtex-7 XC7V2000T FPGA-k esetében is megvizsgáltam az implementálható eredeti lineáris (A), nulladrendű (B) és elsőrendű (C) nemlineáris *Template Memóriával* rendelkező FALCON processzorok számát, melyből aztán meghatározható az ezekkel az FPGA-kkal elérhető maximális sebesség. A vizsgálat eredménye a 3.12. ábrán látható.

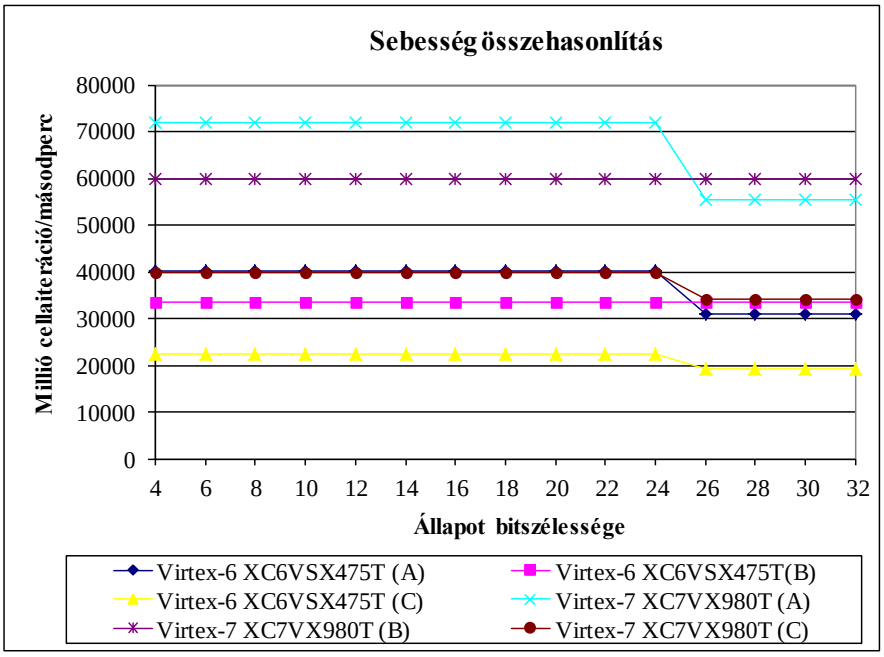



**3.12. ábra. Az implementálható lineáris (A), a nulladrendű (B), és az elsőrendű (C) FALCON processzorok száma a Virtex-6 XC6VSX475T és a Virtex-7 XC7VX980T FPGA-k esetében**

A vizsgálatok megmutatták, hogy mind a Virtex-6 XC6VSX475T, mind pedig a Virtex-7 XC7VX980T FPGA esetében a szűk keresztmetszet az FPGA-kon rendelkezésre álló DSP48E1 Slice-ok száma, és ez korlátozza mindhárom esetben az implementálható processzorok számát. Az implementálható processzorok számát tehát nem korlátozza az FPGA-kon rendelkezésre álló BlokkRAM-ok száma sem a lineáris, sem a nullad- és elsőrendű FALCON processzorok esetében, pedig a különböző megvalósítások különböző számú BlokkRAM-ot igényelnek. Ezeknél az FPGA-knál tehát fordított eredményre jutottam, mint a korábbi tesztekben, hisz ott a szűk keresztmetszet bizonyos esetekben a BlokkRAM-ok száma volt, és az implementálható processzorok számát egyik megvalósítás esetén sem korlátozta a rendelkezésre álló szorzók száma.

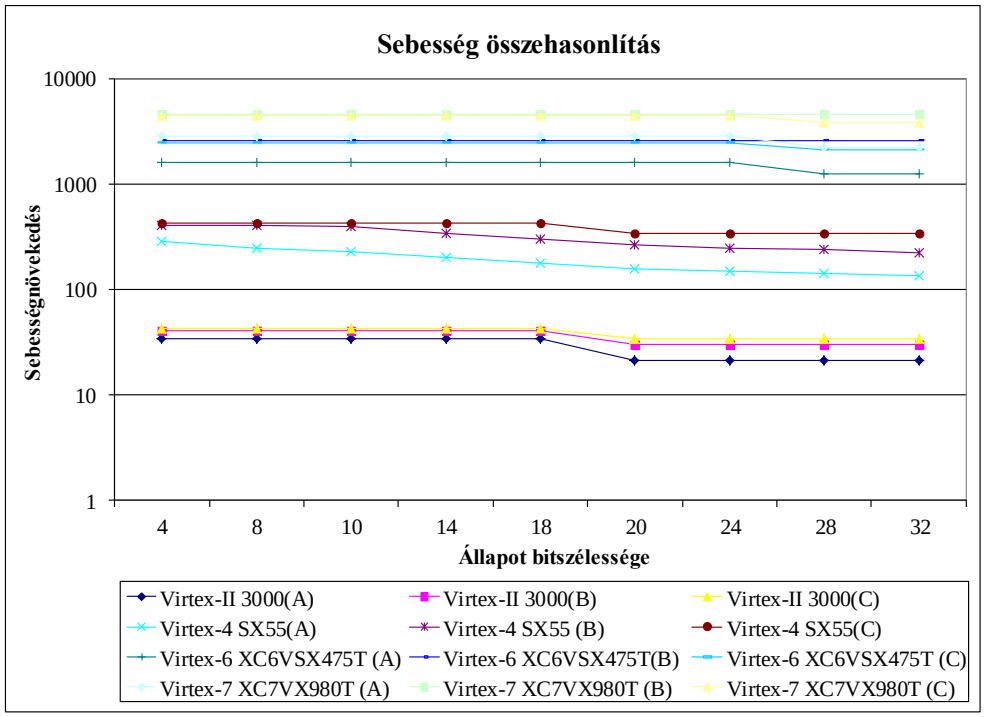
A tesztelés során megvizsgáltam az implementálható processzorok számával összefüggésben a Virtex-6 XCV6SX475T és a Virtex-7 XC7VX980T FPGA-kkal elérhető maximális számítási teljesítményt is. A teljesítmény meghatározása során ugyanazt a konfigurációt vettem alapul, mint korábban a Virtex-II és a Virtex-4 SX55 FPGA-k esetében. Mindkét FPGA típusnál a DSP48E1 Slice-ok maximális működési órajel frekvenciája 600MHz. Ezt alapul véve a Virtex-6 FPGA esetén, a lineáris FALCON processzorral 40320 millió cellaiteráció/másodperc, a nulladrendű FALCON processzor esetében 33600 millió cellaiteráció/másodperc, és az elsőrendű FALCON processzor esetében 22400 millió cellaiteráció/másodperc az elérhető maximális számítási sebesség. A Virtex-7 FPGA alkalmazásával az egyes processzorokkal elérhető

maximális számítási teljesítmény majdnem megkétszerezető (1,786-szoros sebességnövekedés). A szoftveres szimulációhoz képes tehát 3 nagyságrendnyi teljesítménynövekedés érhető el, mind a Virtex-6 mind pedig a Virtex-7 FPGA-kon implementált FALCON processzorok segítségével. Az elvégzett sebességteszt összefoglaló eredményei a 3.13. ábrán láthatóak.




**3.13. ábra. A FALCON processzorokkal elérhető maximális sebesség a Virtex-6 XC6VSX475T és a Virtex-7 XC7VX980T FPGA esetén**

A vizsgálataim utolsó lépéseként egy összefoglaló tesztet is elvégeztem, melyben meghatároztam, hogy a lineáris (A), a nulladrendű (B) és elsőrendű (C) FALCON processzorral hányszoros sebességnövekedés (SpeedUp) érhető el a szoftveres szimulációhoz képest. Az összehasonlítást elvégeztem az összes, a fenti tesztekben szereplő FPGA típuson implementált processzor esetében. A FALCON processzorokkal elérhető sebességnövekedést szintén a korábbi tesztekben is szereplő Intel i5 M520 2.34GHz-es négymagos processzorral elérhető számítási teljesítményhez viszonyítottam. A teszt eredménye a 3.14. ábrán látható.




**3.14. ábra. A lineáris (A), a nulladrendű (B) és elsőrendű (C) FALCON processzorral elérhető sebességnövekedés a szoftveres szimulációhoz képest, különböző FPGA-k esetében**

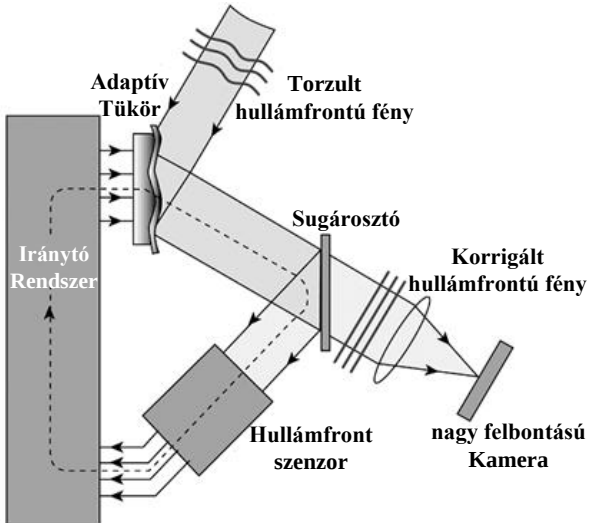
Az egyes FPGA-kon implementált FALCON processzorok esetében a szoftveres szimulációhoz képesti sebességnövekedés összehasonlítását 4-32 bites állapot szélességekre végeztem el. A 3.14. ábrán látható, hogy a Virtex-II, a Virtex-4 és a mai legmodernebb Virtex-6/Virtex-7 sorozatú FPGA-kkal elérhető sebességnövekedés közel konstans a szoftveres szimulációhoz képest az állapot pontosságától függetlenül. Természetesen itt is láthatók a Virtex-II és Virtex-4 FPGA-k esetében a 18 bitnél, és a Virtex-6/Virtex-7 FPGA-k esetében 25 bitnél a korábbiakban bemutatott ugrások.

# 4. SAD operátor alapú hullámfront szenzor megvalósítása FPGA alapú adaptív optikai rendszeren

A nagy expozíciós idejű és nagy felbontású képalkotás esetén alapvető jelentőségű a légkörben jelen lévő turbulenciák valós idejű kompenzálása [50]. Ezen feladat megoldására fejlesztették ki az adaptív optikai (AO) rendszereket, melyek képesek korrigálni az objektum és képe között lévő közeg által létrehozott optikai hatásokat. Egy ilyen rendszer alapvetőn két feladatot hajt végre: egyrészt érzékeli a hullámfrontban megjelenő zavarokat, másrészt valós időben kikompenzálja azokat. Az aktív optikával szemben, mely esetében a hullámfront torzulásait néhány 10 másodpercenként korrigálják egyszer, az adaptív optika esetében hullámhossztól függően néhány ezred- vagy néhány tíz ezredmásodpercenként történik korrekció. Az AO rendszer nagy előnye, hogy a kép élességét azonnal képes javítani. Az AO rendszer fejlesztését Babcock [51] ajánlásai alapján kezdte el az Amerikai hadsereg, melyet aztán két évtizedes fejlesztést követően a csillagászati gyakorlatban is elkezdtek alkalmazni. A katonasági és csillagászati célú rendszerekkel kapcsolatos elvárások nagymértékben különböznek. A csillagászati AO rendszerek esetében kevés mód, nagy optikai és kis időbeli sávszélesség szükséges. A katonai célú AO rendszerek esetében sok mód, szűk (leggyakrabban monokróm) optikai, és nagy időbeni sávszélesség szükséges. A csillagászati AO rendszerek jól alkalmazhatóak nagy teleszkóppal történő kis fényerejű képalkotásnál, spektrális vizsgálatoknál, és csillagászati interferométerek esetében. Az AO rendszerek csillagászati alkalmazásának egyik legjobb bizonyítéka a Neptunusz gyűrűinek felfedezése volt. Az úgynevezett optikai fázis konjugációs módszerrel szemben, mely csak az atmoszféra által keltett zavarokat képes kiküszöbölni, az AO rendszerek multidiszciplinárisak. Ennek köszönhetően nem csak a csillagászatban, hanem egyéb területeken is jól alkalmazhatóak, mint például a katonai védelmi rendszerekben, az egészségügyben, a mikroszkópiában, a méréstechnikában és az optikai kommunikációs rendszerekben. Liang és társai [52] 1997-ben például egy olyan adaptív optikával ellátott kamerát készítettek, mely segítségével az emberi szem felbontóképességét jelentősen növelték.

Egy adaptív optikai rendszer általános felépítése a 4.1. ábrán látható. Az AO rendszernek alapvetően négy fő része van, melyek az *Adaptív Tükör*, a *Sugárosztó*, a *Hullámfront Szenzor*, egy nagy felbontású *Kamera*, és az *Irányító rendszer*. Az AO rendszerhez a torzult hullámfrontú fény egy távcső, mikroszkóp vagy valamilyen egyéb

képalkotó rendszer felől érkezik az *Adaptív Tükörre*. A fény erről visszaverődve a *Sugárosztóba* kerül, mely két részre osztja a fényt. A fény egyik része a *Hullámfront Szenzorba,* a másik része pedig egy nagy felbontású *Kamerába* kerül, amely lényegében a kép digitalizálásáért és megjelenítésért felelős. A *Hullámfront Szenzor* folyamatosan méri a hullámfront torzulásait, mely mérések alapján a vezérlő rendszer folyamatosan módosítja az adaptív tükör felületét. Ennek segítségével folyamatosan kompenzálhatók a hullámfront torzulásai, ezzel javítva a nagy felbontású *Kamera* felé érkező kép minőségét.

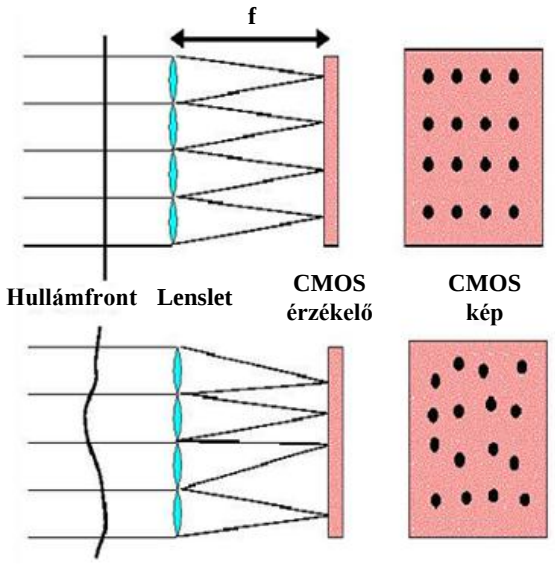



**4.1. ábra. Az Adaptív Optikai rendszer általános felépítése**

Az AO rendszer korrigáló képessége jobb, ha a késleltetés a hullámfront torzulásának mérése és korrigálása között kisebb, mint a hullámfront torzulását okozó közeg változási sebessége. Ez a késleltetés kétféleképpen csökkenthető. Egyrészt ha gyorsabb szenzort, aktuátort és számító-vezérlő komponenseket használunk, másrészt pedig ha növelni tudjuk az AO rendszer ezen komponensei közötti kommunikációs sebességet az eszközök közötti kommunikációs sávszélesség növelésével. Az MTA SZTAKI-ban kifejlesztésre került egykártyás adaptív optikai rendszer ezeket a megoldásokat ötvözi. Alapvetően három fő részből épül fel: egy nagy sebességű CMOS (Complementary Metal-Oxide Semiconductor) szenzorból, egy igen gyors LCOS (Liquid Crystal On Silicon) kijelzőből, és egy FPGA-ból, mely a vezérlési és számítási feladatokat látja el. A beérkező fény hullámfrontjának torzulását az on-boad CMOS szenzor és az FPGA segítségével határozzuk meg (*Hullámfront Szenzor*). Ezek alapján az FPGA kiszámítja a szükséges korrekciókat (*Irányító Rendszer*), végül az LCOS megjeleníti ezeket a korrekciókat (*Adaptív Tükör*), eltávolítva ezzel a beérkező fény

hullámfrontjának torzulásait. Ez a megfizethető adaptív optikai rendszer számos új alkalmazási területét nyithatja meg az adaptív optikai rendszereknek.

Bár számos hullámfront szenzor létezik, ebben a megvalósításban a HS hullámfront szenzor egy speciális verzióját alkalmazták. A HS szenzor esetén a képalkotó rendszer lencséje által létrehozott kép egy úgynevezett lenslet tömbre esik, ahogy az a 4.2. ábrán is látható. A lenslet tömböt alkotó elemi lencsék mindegyike létrehoz egy miniatűr képet a forrás objektumról, és szétbontja az apertúrát úgynevezett szub-apertúrákra. A szub-apertúrák képeit egy felület scan szenzor segítségével detektálják (CMOS szenzor). Az így detektált képek eltolódása a referencia pozíciótól (a torzulás mentes pozíciótó) meghatározza a hullámfront lokális görbületeit az éppen aktuális szub-apertúra pozíciójában. Ezután a teljes hullámfront görbület meghatározható az egyes szub-apertúrák esetén meghatározott lokális görbületek összesítése alapján.




**4.2. ábra. Hartmann-Shack hullámfront szenzor**

A pontszerű objektumok esetén (mint például egy csillag), egyszerű négy-cellás HS szenzorokkal mérhetőek ezek az eltolódások [30][31]. Azonban a kiterjedt objektumok esetén (mint például a Nap vagy a Hold) az eltolódásokat csak magasabb felbontású szenzorok segítségével lehet meghatározni a szub-apertúra kép és a referencia szub-apertúra kép közötti korreláció alapján. A korreláció alapú HS szenzorok jobb jel-zaj viszonnyal rendelkeznek, mint a négycellás [32] HS szenzorok, illetve használhatók a pontszerű objektumok esetén is, azonban jelentősen nagyobb számítási teljesítmény igényük van. Így leggyakrabban elsőrendű aberráció kompenzációs, nagy sebességű korreláció követőket alkalmaznak [33]. A

konvencionális alkalmazások esetén a hullámfrontot sok relatíve nagy felbontású szub-apertúra alkalmazásával detektálják. Mivel a hullámfront nagy sebességgel dinamikusan változik (ezredmásodpercenként), így egy meglehetősen gyors valós idejű korreláció alapú hullámfront szenzor szükséges [34].

Vannak olyan párhuzamos feldolgozással rendelkező eszközök, melyek képesek elvégezni a szükséges számításokat, ezen a sebességen. Még ha a modern CPU-k, stream processzorok (GPU, Graphics Programming Unit) [35][36][37], vagy DSP-k rendelkeznek is a szükséges számítási teljesítménnyel, az FPGA-k versenyképes alternatívának tűnnek a gyors fejlesztési ciklusuknak köszönhetően. Mivel a szenzorok és aktuátorok vezérléséhez is szükség van valamilyen programozható logikai eszközre, így az FPGA-ak alkalmazása még kézenfekvőbb. Mostanában számos FPGA alapú hullámfront szenzort és AO rendszert mutattak be [38][39] és behatóan tanulmányozták a teljesítményüket [40][41].

Figyelembe véve az FPGA-k korlátait és az alkalmazandó hullámfront szenzor speciális paraméterezését, a SAD eljárást választottam a korrelációszerű feldolgozás megvalósításához, mely segítségével két kép optimális illeszkedési pozícióját határozhatjuk meg (a referencia kép és az adott szub-apertúra). A SAD algoritmusnak számos hatékony FPGA alapú implementációja van [42], köszönhetően a mozgókép tömörítési eljárások nagy számítási teljesítmény igényének.

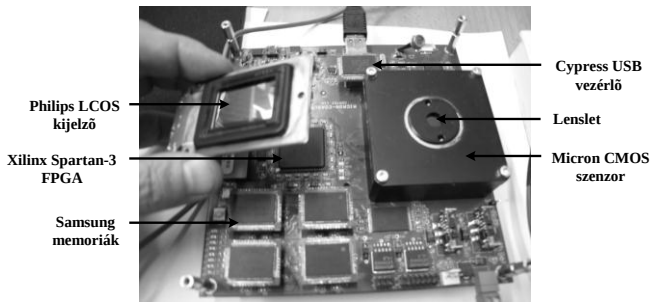
Első lépésben a kép SAD értékeit számoljuk ki a keresési ablakban (ahol az optimális illeszkedést keressük), ezután pedig meghatározzuk ezen értékek minimumát. A keresési ablak határozza meg a kiszámítandó SAD értékek számát, azaz a SAD érték tömb méretét. A módszer segítségével az összes szub-apertúra kép referenciához képesti elmozdulása meghatározható. Számos módszer van, hogy meghatározhassuk a megfelelő referencia képet: itt referenciaként a középső szub-apertúra képét választom ki. Egy másik lehetséges megoldás, hogy több szub-apertúra kép átlagát választom referencia képként. A SAD értékeket az alábbi képlet alapján számolhatjuk:

$$SAD_{k,l} = \sum_{i}^{S}\sum_{j}^{S}\left|P_{i,j} - R_{i+k,j+l}\right|,\ k,l\ <\ A \tag{4.1}$$

ahol S a szub-apertúra mérete, az A a SAD érték tömb mérete, P a szub-apertúra pixel és R a referencia pixel. A referencia kép méretének a szub-apertúra mérete és a SAD érték tömb mérete mínusz 1-el kell egyenlőnek lennie.

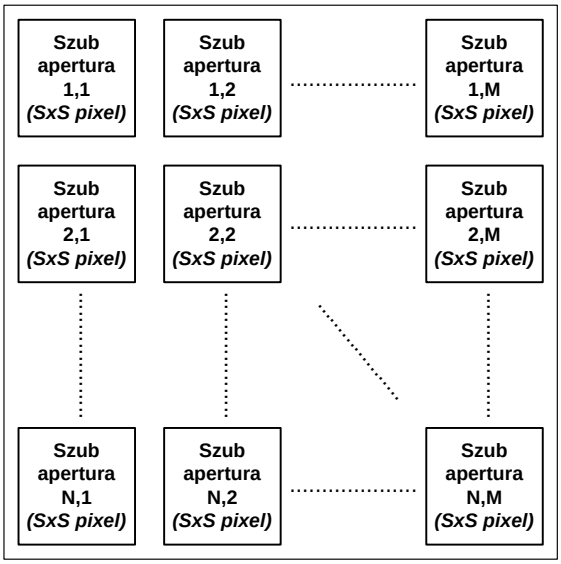
## *4.1.Az FPGA alapú adaptív optikai rendszer bemutatása*

Az MTA SZTAKI-ban elkészített FPGA alapú adaptív optikai rendszer három fő komponensből épül fel, ahogyan azt már korábban is említettem. Az első egy nagysebességű millió pixel felbontású CMOS szenzor és a fölé helyezett lencsetömb, azaz lenslet. A második egy szintén nagy sebességű és felbontású LCOS kijelző, mely feladata a hullámfront torzulásainak kompenzálása. A harmadik pedig egy on-board FPGA, mely a teljes rendszer vezérléséért és a hullámfront korrekciós értékek kiszámításáért felelős. Emellett az FPGA valós időben képes a CMOS szenzor (gyártás során keletkezett egyenetlenségek, Flat-Field, Fixed Pattern Noise (FPN)) esetében fellépő hibák kompenzálására, különböző lineáris és nemlineáris térbeli szűrő műveletek segítségével. Az adaptív optikai rendszer a 4.3. ábrán látható.




**4.3. ábra. Az FPGA alapú adaptív optikai rendszer**

Az adaptív optikai rendszerben alkalmazott szenzor egy Micron MT9M413 típusú CMOS szenzor [43], melynek felbontása 1280×1024 pixel (egy pixel 12μm×12μm méretű), és 500 teljes méretű képet képes készíteni másodpercenként. A szenzor kimeneteként 10 darab 10 bites digitális adatbusz szolgál, melyeknek maximális működési frekvenciája 60 MHz. A CMOS szenzor fölé, ahogy ez az 4.3. ábrán is látható, egy 32×32 darab elemi lencséből felépülő lencse tömb, azaz lenslet került elhelyezésre. Ez 32×32 darab szub-apertúraként szolgál, melyek mindegyikén keresztül egy 16×16 pixel méretű kép keletkezik a CMOS szenzor felületén. Egy N×M-es lenslet, mely S×S méretű szub-apertúrákat tartalmaz, a 4.4. ábrán látható.




**4.4. ábra. A lenslet felépítése**

A 4.4. ábrán látható lenslet és a hozzá kapcsolt CMOS szenzor alkotja a HS hullámfront szenzort, mely az adaptív optikai rendszer bemeneteként szolgál. Az adaptív optikai rendszer esetében alkalmazott hullámfront szenzor optikai geometriája a rendszer kívánt alkalmazásától függ. A paraméterezéssel kapcsolatban részletes információk találhatóak a [44]-ben.

A hullámfront szükséges korrekcióit egy Philips DD720 LCOS kijelző végzi el, mely felbontása 1280×768 pixel (egy pixel 20μm×20μm) és maximálisan 540 darab teljes méretű képet képes megjeleníteni másodpercenként [45]. Ennek az eszköznek a segítségével amplitúdó vagy fázis moduláció hajtható végre a megfelelő hullámlemez, vagy polarizátor segítségével. A kijelző működtetéséhez szükséges I/O sávszélesség megközelítőleg 750 megabájt/másodperc. Ebből következően valós idejű adatfeldolgozás csak valamilyen párhuzamos eszköz alkalmazásával lehetséges. A hullámfront korrekciós eljárás részletei nem kapcsolódnak szorosan a disszertációhoz, így ezeket itt nem fejtem ki részletesen.

Az alkalmazott szenzor és kijelző vezérlésére nyilvánvaló választás lehet egy programozható logikai eszköz alkalmazása. A kereskedelemben kapható FPGA-s fejlesztőkártyák rendszerint nem teszik lehetővé nagy teljesítményű képalkotó eszközök (kamera) csatlakoztatását. Továbbá a fentiekben említett 750 megabájt/másodperces I/O sávszélesség szintén egy szűk keresztmetszete ezeknek az FPGA-s fejlesztő kártyáknak. Ebből következően egy egyedi rendszer készült el, mely a következő elemeket tartalmazza: 1) a korábbiakban bemutatott CMOS szenzor, 2) az LCOS kijelző,

3) Xilinx Spartan-3 XC3S4000 FPGA [12], mely 4 millió kaput, 96 darab 18 KB BlockRAM-ot és 96 18×18 bites szorzót tartalmaz.

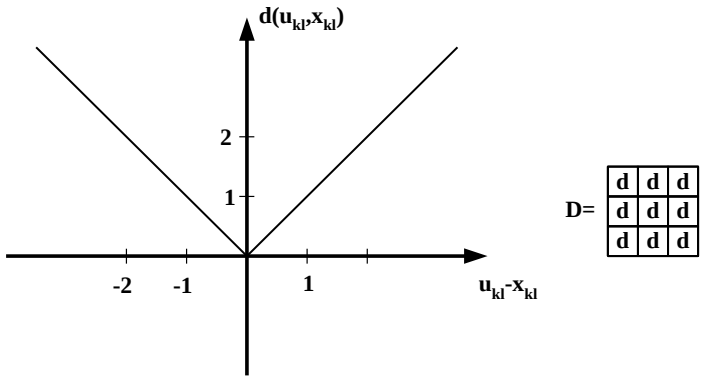
Bár az adaptív optikai kártya egy önálló működésre képes rendszer, azonban kalibrációs és konfigurációs célokra egy USB interface is elhelyezésre került a kártyán a hozzá tartozó USB mikrokontrollerrel együtt. A szenzor adatok feltöltése, bias minták megjelenítése, a kép korrekciós és program paraméterek letöltése ezen az USB interface-en keresztül valósul meg. Mivel az FPGA nem tartalmaz elegendő belső memóriát egy teljes kép eltárolására, ezért külső memória elemek is szükségesek. Ezek a memóriák statikus RAM-ok, melyek képesek 8 teljes kép eltárolására is. Ebből következően az FPN eltávolítása, a Flat-Field korrekció és az LCOS kijelző gyártásából adódó egyenetlenségek kompenzációja is elvégezhető. A szenzor és a kijelző eszközök különböző referencia feszültséget igényelnek, melyeket megfelelő D/A (Digital Analog Converter) átalakítók állítanak elő. A D/A átalakítók szabványos SPI (Serial Peripheral Interface) interface-en keresztül programozhatók. Továbbá az LCOS kijelző megfelelő konfigurálását egy mikrokontroller végzi, amellyel az FPGA egy UART (Universal Asynchronous Receiver/Transmitter) interface-en keresztül kommunikál.

## *4.2.A nemlineáris template futtató emulált digitális CNN-UM architektúra, mint SAD operátor alapú hullámfront szenzor*

Feladatom a 4.3. ábrán látható, és az előbbiekben bemutatott az MTA-SZTAKI-ban tervezett adaptív optikai kártya FPGA alapú hullámfront szenzorjának implementálása volt, melyhez egy olyan eljárást kellet megvalósítanom, amely segítségével meg tudom határozni a szub-apertúrák referencia képhez viszonyított elmozdulását. Az általam választott módszer a korábban a (4.1) egyenletben bemutatott a képfeldolgozásban is alkalmazott SAD operátor alkalmazása volt, mely segítségével lehetőség van két kép optimális illeszkedési pozíciójának meghatározására. A CNN hálózatoknál alkalmazott nemlineáris template-ek egy részénél a CNN cella valamely paramétereinek különbségét határozzuk meg és ezen, különbség alapján egy nemlineáris függvény segítségével adjuk meg a nemlineáris template értékeket. Ezt a tulajdonságot kihasználva a SAD operátor elképzelhető, mint egy elsőrendű nemlineáris template, ahol az elsőrendű nemlinearitás az abszolútérték függvény. Az elsőrendű nemlineáris template-ek végrehajtására rendelkezésemre áll a 3. fejezetben bemutatott elsőrendű nemlineáris FALCON processzor, mely néhány átalakítás után képes a SAD operátort, mint template-et végrehajtani.

### *4.2.1. A SAD operátor, mint elsőrendű nemlineáris template*

Ahogyan azt a 4.2. alfejezetben is említettem, a SAD operátor megvalósítható, mint elsőrendű nemlineáris template, méghozzá mint elsőrendű nemlineáris D típusú template. A nemlineáris D template és a hozzá tartozó nemlinearitás a 4.5. ábrán látható módon képzelhető el.

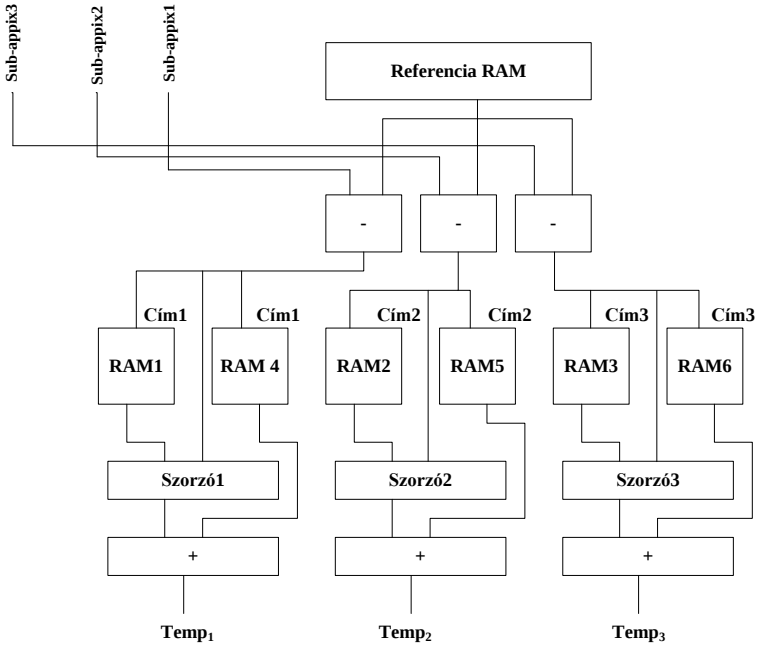



**4.5. ábra. A SAD operátort megvalósító nemlineáris D template és a hozzá tartozó nemlinearitás**

A SAD operátor esetében, ahogyan az a (4.1.) egyenletben is látható, a szub-apertúra pixelekből kell kivonni a referencia kép pixeleit, majd az így kapott értéknek meg kell határozni az abszolútértékét. Az egyszerűség kedvéért a 4.5. ábrán egy $3\times3$-as szub-apertúra esetében adtam meg az elsőrendű nemlineáris D template-et, az $x_{kl}$ értékek a referencia kép egyes pixeleit jelentik, míg az $u_{kl}$ értékek a szub-apertúra egyes pixelei, a nemlinearitás pedig maga az abszolút érték függvény. Ebben az alfejezetben, a továbbiakban az egyszerűség kedvéért ezt a $3\times3$-as szub-apertúra méretet és $5\times5$-ös referencia kép méretet fogom használni.

### *4.2.2. Az elsőrendű FALCON processzor átalakítása*

Ahogyan azt korábban említettem, az elsőrendű FALCON processzor alkalmas a SAD operátor, mint nemlineáris template futtatására. Ehhez azonban szükség van ennek a processzornak a kismértékű átalakítására. Az átalakítás lényegében a *Template Memória* átalakítását jelenti, hisz ebben a *Template Memóriában* most nem csak a nemlineáris template-et kell a megfelelő módon eltárolni, hanem a referencia kép pixeleit is, melyet majd a *Template Memóriában* található nemlineáris template értékeket tartalmazó RAM-ok címzéséhez használt különbség képzésére fogok használni. A SAD operátornál ugyanis az egyes referencia pixelekből kell kivonni a FALCON processzor bemenetére az adaptív optikai kártyán található CMOS szenzor felől érkező szub-apertúra pixeleket (4.5. ábra). A referencia kép pixelei egy 18 Kbit-es BlokkRAM-ban eltárolhatók, így ez az átalakítás nem jár a processzornál jelentős felületigény növekedéssel. Az átalakított *Template Memória* a 4.6. ábrán látható.




**4.6. ábra. Az átalakított elsőrendű nemlineáris *Template Memória***

A 4.6. ábrán látható módosított *Template Memória* lényegében ugyanúgy működik, mint azt korábban az elsőrendű FALCON processzor esetében bemutattam. Az SAD operátorhoz tartozó abszolút érték függvényt két része oszthatom. Az egyes tartományokhoz tartozó meredekség értékeket az egyik, a konstans eltolás értékeket pedig a másik RAM-ban tárolom el a 4.7. ábrán látható módon.

| **Meredekséget értékek elhelyezkedése a RAM-ban** | | | |
|---|---|---|---|
| **Cím** | **Oszlopok** | | |
| | ***0*** | ***1*** | ***2*** |
| ***0*** | **-1** | **-1** | **-1** |
| ***1*** | **1** | **1** | **1** |
| ***2*** | **-1** | **-1** | **-1** |
| ***3*** | **1** | **1** | **1** |
| ***4*** | **-1** | **-1** | **-1** |
| ***5*** | **1** | **1** | **1** |

| **Konstans eltolás értékek elhelyezkedése a RAM-ban** | | | |
|---|---|---|---|
| **Cím** | **Oszlopok** | | |
| | ***0*** | ***1*** | ***2*** |
| ***0*** | **0** | **0** | **0** |
| ***1*** | **0** | **0** | **0** |
| ***2*** | **0** | **0** | **0** |
| ***3*** | **0** | **0** | **0** |
| ***4*** | **0** | **0** | **0** |
| ***5*** | **0** | **0** | **0** |

**4.7. ábra. Az elsőrendű Template Memória RAM-jainak tartalma az abszolút érték függvény, mint nemlinearitás esetén**

A RAM-okat a referencia kép egyes pixelei és a szub-apertúra kép megfelelő pixeleinek különbségével címzem. A kiolvasott meredekség értéket megszorzom a címzéshez használt különbséggel, majd hozzáadom a konstans eltolás értéket. Így megkapom a

SAD operátorhoz tartozó nemlineáris template értékeket, melyet aztán az *Aritmetikai Egység* bemenetére továbbítok.

### *4.2.3. Felületigény és sebesség*

Elvégeztem a fentiekben bemutatott a SAD operátor végrehajtására alkalmas átalakított elsőrendű FALCON processzor vizsgálatát is, mely két részből épült fel. Először meghatároztam az átalakított FALCON processzor általános és dedikált erőforrás igényét. Második lépésként az erőforrás igény alapján meghatározható, hogy hány darab processzor működhet párhuzamosan egy adott FPGA-n, majd ebből kiszámítható a maximális működési sebesség. Mivel itt egy kész hardver rendszerről van szó, amelynek a tesztelés szempontjából fontos hardver elemei a Spartan-3 XC3S4000 FPGA és a szub-apertúra pixeleket biztosító 10 bit-es CMOS szenzor, így a tesztelést csak erre az egy speciális esetre végeztem el. A teszt során így a referencia kép pixeleinek bitszélességét, a bemenetként szolgáló szub-apertúra kép pixeleinek bitszélességét, és ebből következően a template értékek pontosságát 10 bitre választottam. Az átalakított elsőrendű FALCON processzor általános és dedikált erőforrás igénye a 4-1. táblázatban látható.

| **Általános erőforrás** | **Felhasznált** | **Rendelkezésre álló** | **%-os foglaltság** |
|---|---|---|---|
| *Flip-Flop* | 64 | 55296 | 0,12% |
| *4-bemenetű LUT* | 473 | 55296 | 0,86% |
| *Slice* | 567 | 27648 | 2,05% |
| **Dedikált erőforrás** | | | |
| *BlokkRAM* | 12 | 96 | 12,50% |
| *18x18 szorzó* | 3 | 96 | 3,13% |

**4-1. táblázat. Az általános és dedikált erőforrás szükséglete az átalakított elsőrendű FALCON processzornak**

Látható hogy a szűk keresztmetszet ennél a megvalósításnál a rendelkezésre álló BlokkRAM-ok száma. A Spartan-3 XC3S4000 FPGA-n implementált FALCON processzor 120 MHz-es órajel frekvencián képes működni, így a vele elérhető maximális számítási teljesítmény, ha mind a 8 darab implementálható FALCON processzort párhuzamosan futtatom, 320 millió cellaiteráció/másodperc.

A jelen tesztekben $3\times3$ szub-apertúrákkal és $5\times5$ referencia képekkel dolgoztam. A valós alkalmazásokban azonban $16\times16$ szub-apertúrákat és $31\times31$ méretű referencia képeket alkalmaznak (ilyen méretű szub-apertúrákat tartalmaz a CMOS szenzor előtt található optika is). Ebben az esetben egy átalakított FALCON

processzor implementálásához 16 darab szorzóra és 25 BlokkRAM-ra van szükség. Ebben az esetben már csak 3 darab átalakított FALCON processzor implementálható, mellyel már csak 120 millió cellaiteráció/másodperces számítási teljesítmény érhető el.

Mivel az adaptív optikai kártyán a CMOS szenzor soronként továbbítja a pixeleket, így ahhoz, hogy elkezdhessem az első szub-apertúra feldolgozását, meg kell várni, amíg megérkezik az adott szub-apertúra sorban található összes szub-apertúra pixele (lásd 4.4. ábra). Ezeket el kell tárolni, és nagy szub-apertúrák esetében ez nagy mennyiségű adat eltárolását jelenti. Ráadásul a következő szub-apertúra sor pixeleit nem lehet közvetlenül ugyanabba a memóriába tölteni, mert azok éppen feldolgozás alatt vannak, így tehát dupla pufferelést kell alkalmazni, ami tovább növeli az implementáció memória szükségletét, emiatt tovább csökken az implementálható FALCON processzorok száma, és ezzel összefüggésben az elérhető sebesség. További hátránya ennek az implementációnak, hogy az egyszerű kivonás és abszolút érték képzés (4.1 egyenlet) helyett szorzásokat kell végezni a *Template Memóriában*. Ezen problémák miatt az elkészített általános célú elsőrendű FALCON processzor alapú rendszer nem hozta meg a várt teljesítményt. Annak érdekében, hogy megfeleljek a SAD alapú hullámfront szenzor nagy sebességigényeinek és a rendelkezésemre álló kártya hardver követelményeinek, a fentiekben bemutatott architektúrát újragondoltam, és létrehoztam egy nagy sebességű, speciálisan erre a feladatra optimalizált architektúrát. Ennek bemutatására az alábbi alfejezetben térek ki.

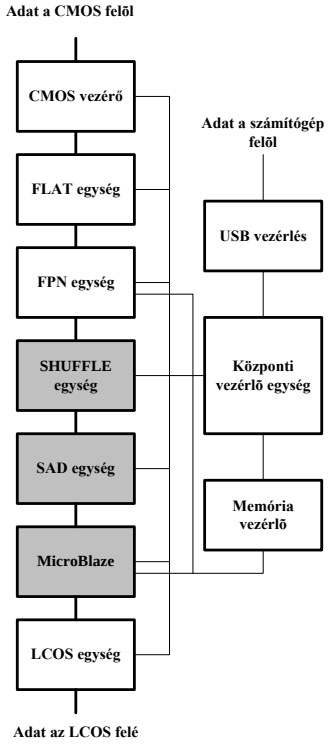
## 4.3. *SAD operátor alapú hullámfront szenzor optimális megvalósítása az FPGA alapú adaptív optikai rendszeren*

A fő célom egy nagy sebességű, magas szintű párhuzamossággal rendelkező SAD architektúra FPGA alapú megvalósítása volt, mely képes meghatározni a szub-apertúrák elmozdulásait és kiszámolni a hullámfront torzulásait 500 teljes kép/másodperces sebességgel. Bár a SAD algoritmus egyszerű, a hatékony FPGA alapú megvalósítása különleges tervezési megfontolásokat igényel.

### *4.3.1. A teljes rendszer*

Az FPGA-n megvalósított SAD architektúrának három fő komponense van, ahogy ez a 4.8. ábrán is látható. A *SHUFFLE egység* feladata, hogy a CMOS szenzor felől párhuzamosan érkező pixeleket egy soros pixelfolyammá alakítsa, a SAD értékek kiszámolásához. A *SAD egység* minden egyes képhez kiszámolja a SAD értékeket,

melyek közül végül kiválasztja a legkisebbet. Ez az érték és a 4 szomszédja (bal, jobb, alsó, felső) a Xilinx MicroBlaze processzor felé továbbítódik, mely kiszámítja a hullámfront elhajlásának értékét az adott pontban.



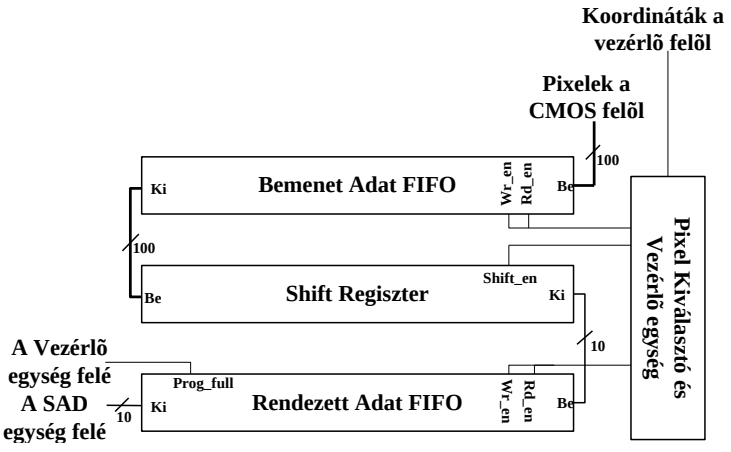

**4.8. ábra: A teljes rendszer**

Ahhoz hogy a 4.8. ábrán látható SAD architektúra működni tudjon, számos kiegészítő modul implementálására is szükség volt. Ezek a modulok a *CMOS vezérlő*, a *FLAT* és az *FPN egység*, az *LCOS egység*, az *USB vezérlő* egység, a *Memória vezérlő* egység és a *Központi vezérlő egység*. A *CMOS vezérlő* egység fogadja a Micron CMOS szenzor felől érkező képi információkat és vezérli annak működését. Az *LCOS egység* kiszámítja a korrekciós tagokat, és továbbítja azokat az LCOS kijelző felé. Az *FPN egység* eltávolítja az Fixed Pattern Noise-t, míg a *FLAT egység* pedig a Flat-Field korrekciót végzi el a CMOS szenzor által alkotott képen. Erre azért van szükség, mert a képen fényességkülönbségek és furcsa alakzatok jelenhetnek meg, melynek oka az egyes CMOS pixelek különböző érzékenysége, és esetlegesen a CMOS chip-re került porszemek. A Flat-Field korrekció egy olyan technika, mely segítségével ezek a hibák eltávolíthatók a képről. Az *USB vezérlő* feladata, hogy kommunikáljon az FPGA-s kártyához csatlakoztatott hoszt számítógéppel. A CMOS szenzor felől érkező feldolgozandó pixelek koordinátáit (nem minden pixelt, csak a szub-apertúrák pixeleit kell feldolgozni) a számítógép határozza meg, míg a referencia képet a MicroBlaze

egység frissíti alkalmanként. A *Memória vezérlő* feladata, hogy az adatáramlást vezérelje a memória felől és a memória felé. A memóriák az *FPN* és *FLAT egységek* számára tartalmazzák a szükséges képeket. A *Központi vezérlő egység* feladata pedig a teljes rendszer vezérlése. Mivel a fentiekben megemlített modulok részletes leírása nem tartozik jelen disszertáció keretei közé, így a továbbiakban a SAD architektúra fő egységeinek bemutatására koncentrálok.

### *4.3.2. A SHUFFLE egység*

Ahogy azt a 4.1. alfejezetben is említettem, a CMOS szenzor 10 darab 10 bites adatbuszon keresztül párhuzamosan továbbítja a kép pixeleit. A CMOS szenzor által továbbított képből azonban csak azokra a pixelekre van szükség a SAD értékek kiszámításához, amelyek valamelyik szub-apertúrához tartoznak (4.4. ábra). A *SUFFLE egység* feladata egyrészt, hogy a CMOS szenzor által továbbított párhuzamos 10-szer 10 bites adatfolyamból 10 bites soros adatfolyamot generáljon, másrészt, hogy csak azokat a pixeleket továbbítsa a SAD egység felé, melyek valamely szub-apertúrához tartoznak. A feldolgozandó szub-apertúrák bal felső sarkának koordinátáit a hoszt számítógép határozza meg, és küldi el az FPGA-s kártyára, ahol ezek a koordináták a *SHUFFLE egységben* található *Pixel Kiválasztó és Vezérlő egységben* tárolódnak el. Ezen koordináta-információk alapján a *SHUFFLE egység* képes kiválasztani a szub-apertúrákhoz tartozó pixeleket, és a párhuzamos pixel folyamot soros pixelfolyammá alakítani. A *SHUFFLE egység* felépítése a 4.9. ábrán látható.




**4.9. ábra. A *SHUFFLE egység***

A *SHUFFLE egység* architektúrája egy *Bemenet Adat FIFO*-ból, egy *Rendezett Adat FIFO*-ból, egy párhuzamos bemenetű és soros kimenetű *Shift Regiszterből* és a *Pixel Kiválasztó és Vezérlő egységből* épül fel. A *Bemeneti Adat FIFO* 100 bites bemenettel

és 100 bites kimenettel rendelkezik. Ennek a FIFO-nak a feladata, hogy eltárolja a 10×10-bitnyi pixelt, amit a CMOS szenzor generál minden egyes órajel impulzusban. A FIFO bemenete a CMOS szenzor adatbuszának órajel frekvenciáján, míg kimenete a hozzá kapcsolt egységek órajel frekvenciáján működik, ezzel órajel tartomány konverziót hajtva végre. Annak érdekében, hogy ne fordulhasson elő adatvesztés, a FIFO-nak elég mélynek kell lennie ahhoz, hogy el tudjon tárolni egy teljes szub-apertúra sort (1280×P). A párhuzamos bemenetű és soros kimenetű FIFO a 100-bites bemenő adatból 10-bites kimenő adatot generál, úgy hogy csak azokat a pixeleket továbbítja a rendezett adat FIFO-hoz, amelyek valamely szub-apertúrához tartoznak, a többit pedig eldobja. A *Rendezett Adat FIFO* egészen addig tárolja a pixeleket, amíg a szub-apertúra egy adott sora meg nem érkezik, és csak ezután továbbítja a pixeleket sorosan a *SAD egység* bementére. A párhuzamos bemenetű és soros kimenetű FIFO, valamint a *Rendezett Adat FIFO* a *SAD egység* működési órajel frekvenciájával megegyező frekvencián működik. A *Pixel Kiválasztó és Vezérlő egység* pedig vezérli a FIFO-k és a *Shift Regiszter* egység működését.

### *4.3.3. A SAD egység működése*

A *SAD egységet* arra terveztem, hogy képes legyen kiszámítani a (4.1) egyenletet a szub-apertúra pixeleken valós időben. A képi információ, azaz a pixelek soronként érkeznek a CMOS szenzor felől. Ebből következően S×S pixelt kellene eltárolni annak érdekében, hogy végre lehessen hajtani a SAD értékek kiszámítását. Emellett egy sorban számos szub-apertúra található, amelyek eltárolása tovább növelné a *SAD egység* memória igényét, mivel a szub-apertúrák dupla pufferelésére lenne szükség. Ez a megoldás nem túl praktikus nagy szub-apertúrák esetén. A szükséges memória igény az alábbi összefüggés alapján adható meg:

$$s_{memory} = 2 \cdot S \cdot S \cdot p \cdot N \tag{4.2}$$

ahol az S a szub-apertúra mérete, a p a pixel értékek bit-szélessége, és N a szub-apertúrák száma.

Az általam készített architektúrában ahelyett, hogy eltárolnám a szub-apertúra ablakokat, a SAD értékek kiszámítását átrendeztem annak érdekében, hogy a számítás illeszkedjen a CMOS szenzor felől érkező adatfolyamhoz. Így tehát nem várom meg, amíg megérkezik a szub-apertúrához tartozó összes pixel, hanem a rész SAD értékeket párhuzamosan számolom ki. Amikor például megérkezik az első pixel ($P_{0,0}$), a (4.1)-es egyenlet első tagját kiszámolom – $AD_{k,l,0,0}=|P_{0,0}-R_{k+i,l+j}|$ – az összes lehetséges *k,l*

értékre. Amikor a szub-apertúra első sorához tartozó rész SAD értékeket kiszámoltam, egy ideiglenes pufferben tárolom el ezeket az értékeket, majd a számítás a következő szub-apertúra első sorával folytatódik. Ezt a módszert használva A×A pixelt kell eltárolni, ahol az

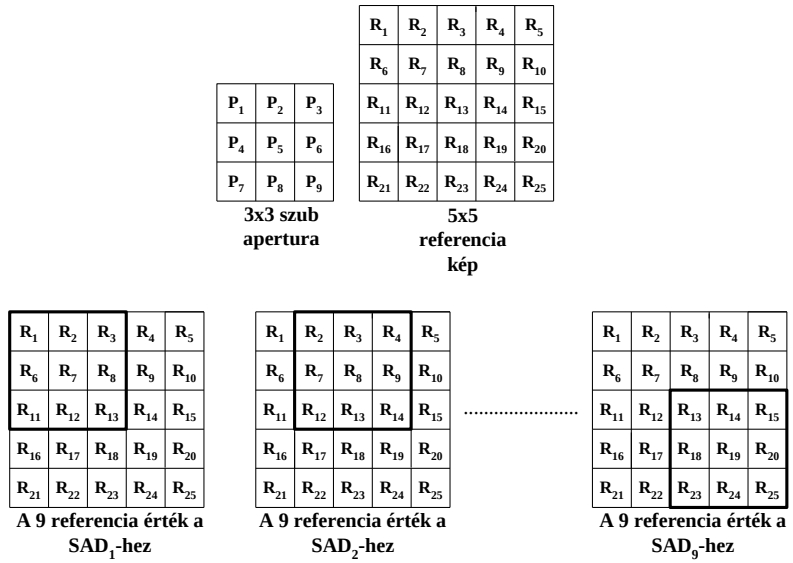
$$A = r - S + 1 \tag{4.3}$$

és az r a referencia kép mérete. A túlcsordulás elkerülése érdekében a SAD számítása során a rész SAD értékek bitszélességét kiterjesztem $\log_2\lceil S \cdot S \rceil$ bittel. Így ebben az esetben a SAD értékek kiszámításához szükséges memória igény:

$$s_{memory} = A \cdot A \cdot (p + \log_2 \lceil S \cdot S \rceil) \cdot N \tag{4.4}$$

A hullámfront szenzor alkalmazási területének megfelelően a referencia kép relatíve nagy, és a szub-apertúrákat csak kis mértékben tolom el ezen belül, így az S minden esetben nagyobb lesz, mint az A. Ennek köszönhetően a rész SAD értékek eltárolásának módszerét alkalmazva jelentősen kevesebb memória szükséges a SAD értékek kiszámításához, mintha a dupla pufferelési technikát alkalmaznák. Esetemben a tipikus értékek r=32, S=28, A=5, p=8, N=32. Ebben az esetben a dupla pufferelés 392 Kbit memóriát igényel a SAD értékek kiszámításához, míg a rész SAD értékek eltárolást végző módszer csak 14 Kbit memóriát igényel. Ez jelentős csökkenés.

A *SAD egység* működését egy egyszerű példán keresztül fogom bemutatni, ahol csak egy 3×3-as lenslet-et használok, mely 3×3-as szub-apertúrákból épül fel. A lenslet felépítése a 4.4. ábrán látható. Annak érdekében, hogy kiszámoljam a 3×3-as SAD érték tömböt a 3×3-as szub-apertúrához, egy 5×5-ös referencia képet kell használni, annak érdekében, hogy minden egyes pixelre ki tudjam számolni a SAD értékeket. A szub-apertúra elemei, a referencia kép és az egyes SAD értékek kiszámításához szükséges referencia pixelek a 4.10. ábrán láthatóak.




**4.10. ábra. A 3x3-as szub-apertúra, az 5x5-ös referencia kép és a SAD1, SAD2,... SAD9 értékek kiszámításához szükséges referencia pixelek**

Az (4.5)-(4.7) egyenletnek megfelelően látható, hogy a $P_1$-es pixel szükséges mind a 9 SAD érték első elemének kiszámításához, azonban mindegyik esetben más referencia pixel értékkel képezem az abszolút különbséget. Hasonlóan, a $P_2$-es pixelérték pedig a 9 SAD érték második elemének kiszámításához szükséges, és így tovább. Ezzel a módszerrel a rész SAD értékek egymással párhuzamosan számíthatóak. Egy szub-apertúrára az összes SAD érték kiszámításához A×A darab számítási ciklus szükséges.

$$SAD_1 = |P_1 - R_1| + |P_2 - R_2| + |P_3 - R_3| + |P_4 - R_6| + |P_5 - R_7| + |P_6 - R_8| + |P_7 - R_{11}| + |P_8 - R_{12}| + |P_9 - R_{13}| \quad \textbf{(4.5)}$$

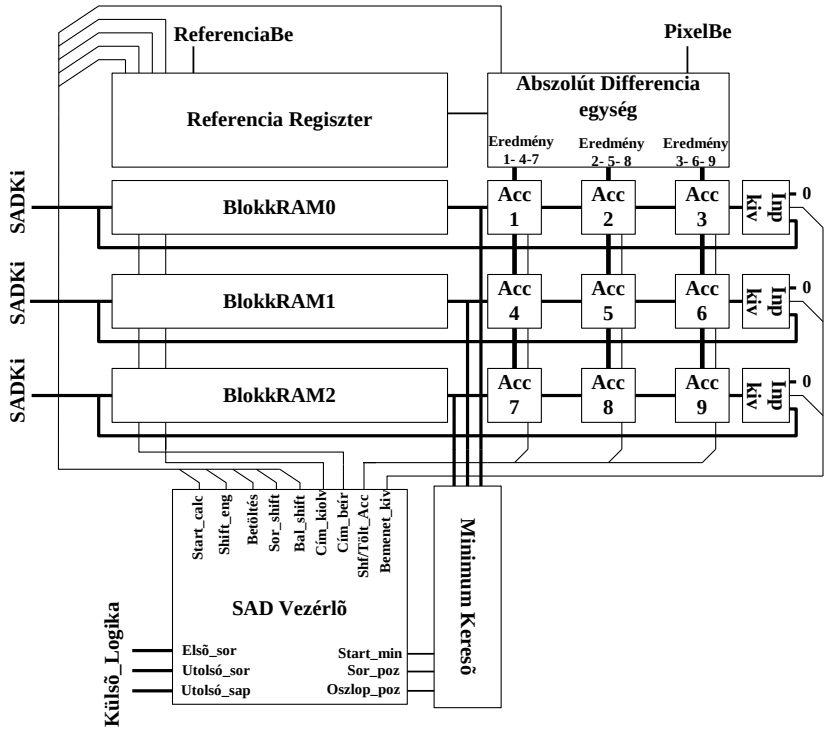
$$SAD_2 = |P_1 - R_2| + |P_2 - R_3| + |P_3 - R_4| + |P_4 - R_7| + |P_5 - R_8| + |P_6 - R_9| + |P_7 - R_{12}| + |P_8 - R_{13}| + |P_9 - R_{14}| \quad \textbf{(4.6)}$$

$$\cdots\cdots\cdots\cdots$$

$$SAD_9 = |P_1 - R_{13}| + |P_2 - R_{14}| + |P_3 - R_{15}| + |P_4 - R_{18}| + |P_5 - R_{19}| + |P_6 - R_{20}| + |P_7 - R_{23}| + |P_8 - R_{24}| + |P_9 - R_{25}| \quad \textbf{(4.7)}$$

### *4.3.4. A SAD egység*

A *SAD egység* fő építőelemei a *Referencia Regiszter* egység, az *Abszolút Differencia egység* és a *Minimum Kereső* egység, ahogy ez a 4.11. ábrán is látható, egy 3×3-as szub-apertúra esetén.




**4.11. ábra. A *SAD egység***

A *Referencia Regiszter* egység a referencia kép pixeleinek értékét tárolja, illetve előállítja a szub-apertúra adott pixeléhez tartozó referencia ablakot. Az *Abszolút Differencia egység* kiszámítja a rész SAD vagy másképpen az Abszolút Differencia (AD) értékeket a SAD kiszámításához. A *Minimum Kereső* egység meghatározza a legkisebb SAD értéket és a négy szomszédját. A *SAD Vezérlő* egység vezérli a *SAD egység* működését. Ezen elemek mellett a *SAD egység* tartalmaz még egy akkumulátor regiszter tömböt is, és néhány BlokkRAM memóriát. Az akkumulátor regiszter tömb és a BlokkRAM memóriák számát a SAD értékek tömbjének mérete határozza meg, míg a BlokkRAM memóriák méretét a lenslet-ben egy sorban található szub-apertúrák száma határozza meg. A SAD érték tömb minden egyes eleméhez az akkumulátor regiszter tömb egy eleme tartozik, melyek feladata a megfelelő SAD részeredmények eltárolása, és az AD értékek összeadása a 4.5, 4.6, 4.7 egyenleteknek megfelelően. A kiszámított AD értékek a szub-apertúra aktuális sorának feldolgozása után átkerülnek a BlokkRAM memóriákba. A *SAD egység* akkor indul el, amikor a szub-apertúra első sora megérkezik a *Rendezett Adat FIFO*-ba (4.9. ábra). A *Referencia Regiszter* egységet a *SAD Vezérlő* egység indítja el, amikor megérkezik a szub-apertúra sor első pixele, hogy előállítsa az AD értékek kiszámításához szükséges referencia értékeket. Eközben az *Abszolút Differencia egység* is elindul, hogy kiszámítsa az ehhez a pixelhez tartozó (ezen pixel segítségével kiszámítható) összes abszolút differencia értéket. Az *Abszolút*

*Differencia egységnek* két órajel ciklusra van szüksége annak érdekében, hogy kiszámítsa, az AD értékeket. A számítás eredménye az akkumulátor regiszter tömbben tárolódik el. Ha megérkezik a szub-apertúra sor következő pixele, akkor az összes hozzá tartozó AD érték kiszámításra kerül, majd hozzáadom az akkumulátor regiszterben az előbbiekben eltárolt pixel értékekhez. A szub-apertúra aktuális sorának feldolgozása után az akkumulátor tömb tartalma bemásolódik a BlokkRAM memóriákba. Az akkumulátor regiszter tömböt az adott szub-apertúra első sorával kiszámított AD értékek kiszámítása előtt nullákkal inicializálom. Minden más esetben a következő szub-apertúra előző sorával kiszámított AD értékekkel töltöm fel az akkumulátor regiszter tömböt. Nagyon fontos, hogy az akkumulátor regiszter tartalma mindig a következő szub-apertúrához tartozó részeredményeket tartalmazza. A *Referencia Regiszterben* található értékek újrainicializálása is ez idő alatt történik meg.

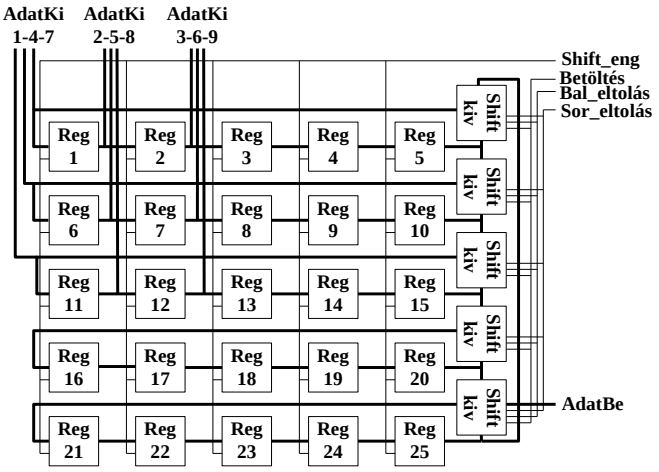
A szub-apertúrához tartozó SAD értékek kiszámítása akkor fejeződik be, amikor a szub-apertúra utolsó sora is feldolgozásra került. A kiszámított SAD értékek minimumát és az ehhez a pixelhez tartozó négy szomszédot (jobb, bal, alsó, felső) a *Minimum Kereső* egység határozza meg. Amikor az összes SAD érték kiszámításra került, akkor ezek az értékek oszloponként kiolvashatóak a BlokkRAM memóriákból. Az egy szub-apertúrához kiszámított SAD érték tömb oszlopaiban található értékek közül a legkisebbet egy komparátorokból álló fa segítségével párhuzamosan határozom meg. Az így kapott minimum értékek közül a szub-apertúra teljes területére vonatkozóan a legkisebb értéket sorosan határozom meg. Ebből következően a *Minimum Kereső* egység késleltetése a SAD érték tömb méretétől függ. Miután a lenslet egy adott sorában található összes szub-apertúrára kiszámítottam a SAD értékeket, a legkisebb érték és a négy szomszédja kiolvasható a BlokkRAM memóriákból a *Minimum Kereső* egység által meghatározott koordináták alapján. Ezeket az értékeket ezután továbbítom a Xilinx MicroBlaze processzor felé további feldolgozás céljából.

A *SAD egység* a CMOS szenzor adatbusz órajel frekvenciájának kétszeresével működik. A SAD érték tömb mérete, amely meghatározza a referencia kép méretét, a szub-apertúrák mérete és száma tetszőlegesen állítható az egység VHDL leírásában, a hullámfront detektálási feladatnak megfelelően. Ezekben a feladatokban a szub-apertúrák mérete tipikusan 8×8-tól 32×32-ig terjedhet. Emellett egyszerre akár több *SAD egység* is használható párhuzamosan, ha a feldolgozandó képet több részre osztom, több *SHUFFLE egység* alkalmazásával.

### *4.3.5. A Referencia Regiszter egység*

A *Referencia Regiszter* egység a referencia kép pixel értékeit tárolja, és előállítja a szub-apertúra aktuális pixeléhez tartozó referencia érték ablakot. Az S×S referencia ablak balról jobbra mozog a számítás során, ahogy ez a 4.10. ábrán is látható.

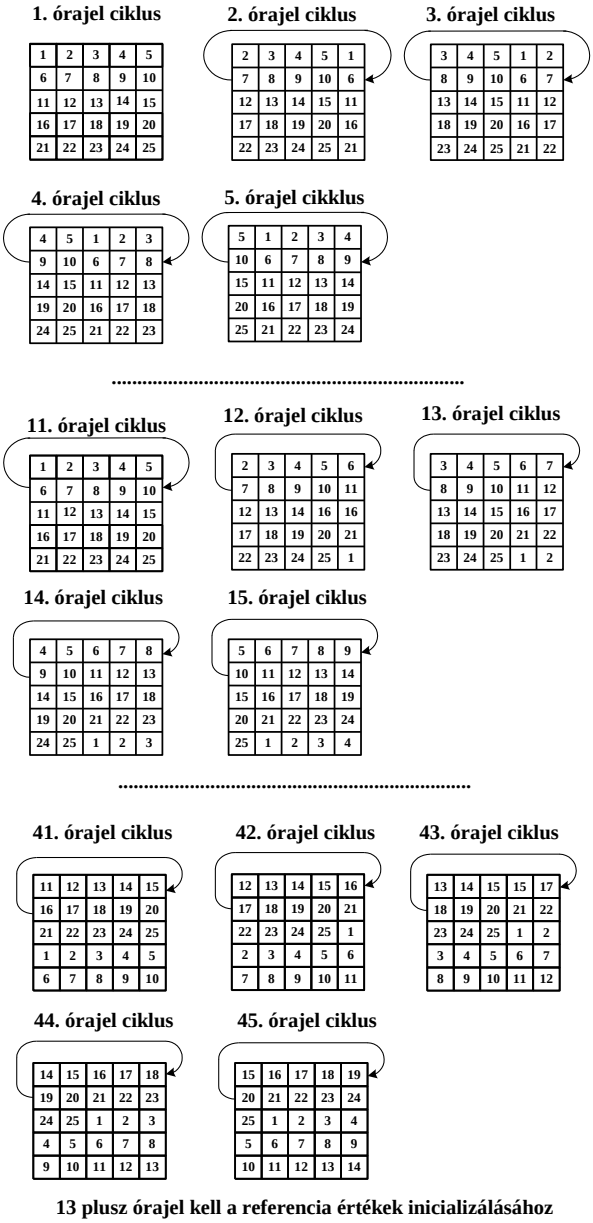
Annak érdekében, hogy a *Referencia Regiszter* implementációját egyszerűbbé tegyem és kihasználhassam az FPGA-akban lévő shift regiszter erőforrásokat, a referencia ablak fix és a referencia értékek shiftelődnek ebben az implementációban. A shift regiszter erőforrások használatával alacsonyabban tartható az egység felület és route-olási erőforrás igénye, mintha egy regiszter tömböt alkalmaznák. A referencia tömb nem használt részét pedig BlokkRAM memóriában tárolom, hogy még optimálisabban használjam ki a rendelkezésre álló erőforrásokat. A 4.10. ábrán látható példában használt kilenc referencia érték mindig a referencia tömb bal felső sarkában helyezkedik el. A *Referencia Regiszter* egység felépítése látható a 4.12. ábrán, egy 5×5-ös referencia kép esetében.




**4.12. ábra. A *Referencia Regiszter* egység 5x5 referencia kép esetében**

Ez az egység egy shift regiszter tömböt tartalmaz, mely az értékek tárolásáért és shift-eléséért felelős. A regiszter tömb méretét a referencia kép mérete határozza meg. A regiszter tömb feltöltése a referencia kép pixeleinek megfelelő értékekkel az inicializációs szakaszban történik meg. Ahhoz, hogy kiszámíthassam a SAD értékeket, a referencia értékeket cirkulárisan körbe-körbe kell shift-elni, mindig az aktuális szub-apertúra pixelnek megfelelően. Ebből következően a *Referencia Regiszter* egységnek három működési módja van: 1) betöltés, 2) balra eltolás, 3) sor eltolás. A betöltési módban a referencia értékek a jobb legalsó regiszterbe ($Reg_{25}$) töltődnek, és pixelenként shiftelődnek balra a regiszter láncban. A legbaloldaliabb regiszterek

($Reg_{21}$, $Reg_{16}$, $Reg_{11}$, $Reg_{6}$, $Reg_{1}$) tartalma az egy sorral felettük lévő legjobboldaliabb regiszterekbe ($Reg_{20}$, $Reg_{15}$, $Reg_{10}$, $Reg_{5}$) töltődik. A balra eltolás módban a regiszterek tartalma körbe-körbe shift-elődik egy regiszter soron belül. Végül a sor eltolás módban a regiszterek működése megegyezik a betöltés módbeli működéssel, a jobb legalsó regiszter ($Reg_{25}$) kivételével, amely a bal legfelső regiszter ($Reg_{1}$) tartalmával töltődik fel. A referencia értékek mozgásának részletes lépései láthatóak a 4.13. ábrán, a 4.10. ábrán említett példa esetén.



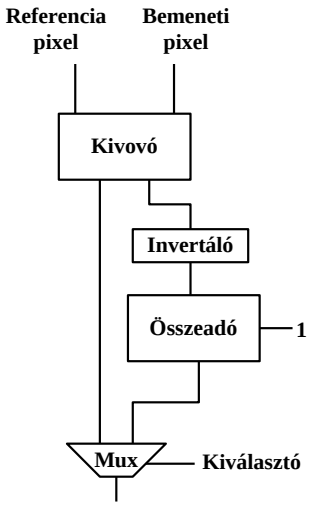

**4.13. ábra. Adatmozgás az 5x5 referencia regiszter tömbben**

Az első órajel ciklusban, amikor az első szub-apertúra legelső pixele megérkezik, a referencia értékek nem shiftelődnek, mert pontosan a megfelelő pozícióban vannak. A második órajel ciklusban a referencia értékek eggyel balra shiftelődnek, ahogy ez a

4.13. ábrán is látható. A harmadik órajel ciklusban, amikor megérkezik az első szub-apertúra harmadik pixele, szintén eggyel balra shiftelődnek a referencia értékek. Mielőtt a következő szub-apertúra első sorához tartozó AD értékek kiszámítása megkezdődne, a referencia értékeket vissza kell shift-elni az eredeti pozíciójukba. Ez a negyedik és az ötödik órajel ciklusban hajtódik végre, ahogy ez a fenti 4.13. ábrán is látható. A hatodik órajel ciklusban megkezdődik a második szub-apertúra első sorához tartozó számítások végrehajtása. E számítás alatt a referenciaértékek az előbbiekben leírtaknak megfelelően mozognak a referencia tömbben. A tizenegyedik órajel ciklusban a lenslet adott sorában lévő utolsó, azaz a harmadik szub-apertúra első sorának a feldolgozása kezdődik el. Ebben az esetben azonban a referencia értékek nem csak balra, hanem eggyel felfele is shiftelődnek. Ez azért szükséges, hogy előkészítsem a referencia értékeket a lenslet első sorában lévő első szub-apertúra második sorához tartozó AD értékek kiszámításához. A tizenharmadik órajel ciklus után egy inicializációs szakasz is szükséges, egészen addig, amíg a referencia regiszter tömb második sorának minden egyes eleme át nem shift-elődik az első sorába. Ezek a lépések a tizennegyedik és tizenötödik órajel ciklusban hajtódnak végre. A fent leírt folyamat, amely az első és tizenötödik órajel ciklus között ment végbe, egészen addig ismétlődik, amíg a lenslet első sorában található utolsó szub-apertúra utolsó sorához nem érek a negyvenegyedik órajel ciklusban. Ezen a ponton egy sor shift-elés kerül végrehajtásra a negyvenegyedik órajel ciklustól a negyvenötödik órajel ciklusig, hogy a referencia értékeket megfelelően pozícionáljam. A negyvenötödik órajel ciklusban a referencia regiszter tömb tartalma megcserélődik, azaz a regiszter tömb első fele átkerül a regiszter tömb alsó, míg második fele a regiszter tömb felső részébe, ahogy ez a 4.13. ábrán is látható. Ebből következően ezeket az értékeket vissza kell shift-elni a kiinduló helyzetükbe, hogy el lehessen kezdeni a lenslet következő sorában található szub-apertúrák feldolgozását. Ehhez a folyamathoz tizenhárom órajel ciklus szükséges.

### *4.3.6. Az Abszolút Differencia egység*

Az *Abszolút Differencia egység* felelős a referencia értékek és a bemeneti kép közötti abszolút differencia kiszámításáért. Ez az egység a processzáló elemek egy tömbjéből épül fel. A processzáló elemek számát a szub-apertúra mérete határozza meg. Ebből következően az összes AD érték párhuzamosan számítható ki. Egy processzáló elem felépítése látható a 4.14. ábrán.



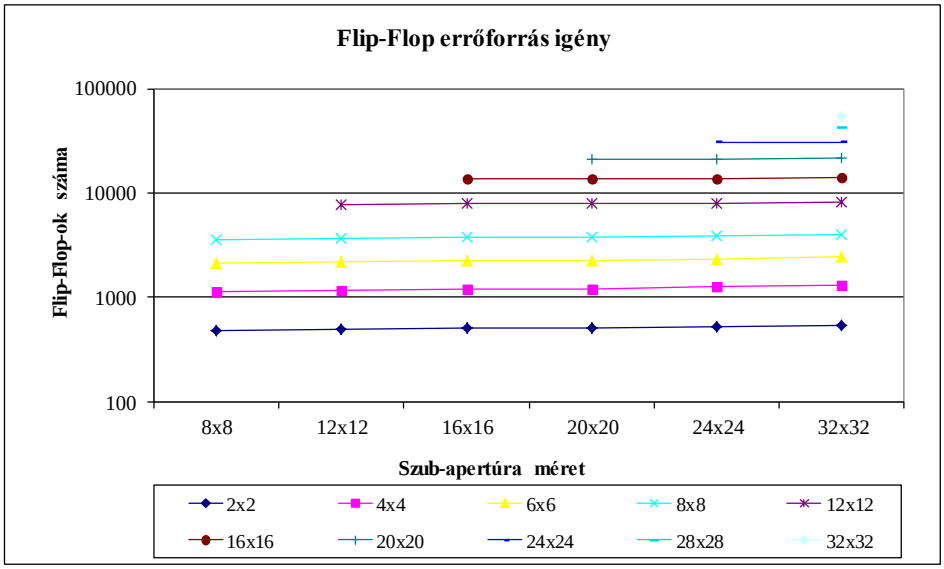

**4.14. ábra. Az *Abszolút Differencia egység***

Egy elemi processzáló elem elsőként kiszámolja a bemenet és a referencia kép közötti különbséget, invertálja az eredményt és hozzáad egyet, tehát lényegében előállítja a különbség kettes komplemensét. Végül egy *Kiválasztó* jellel az MSB bit-nek megfelelően a kettes komplemens, vagy pedig az eredeti különbség kerül a kimenetre, mint abszolút differencia. Ennek az egységnek két órajelre van szüksége, ahhoz hogy kiszámítson egy AD értéket.

### *4.3.7. Az elért eredmények*

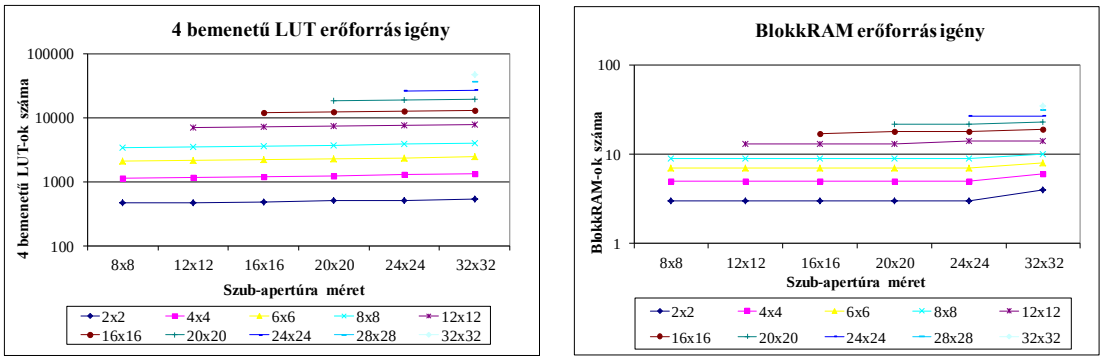
Elkészült az FPGA alapú adaptív optikai rendszer szerves részét képző hullámfront szenzor architektúra, melyet egy Spartan-3 XC3S4000 FPGA-n implementáltam, a VHDL nyelvet felhasználva. A rendszer fő komponense a *SAD egység*, amely a SAD értékek kiszámításáért felelős. A *SAD egység* a szub-apertúrák számának és méretének megfelelően tetszőlegesen konfigurálható. A *SAD egységet* a VHDL leírásának segítségével lehet konfigurálni, a szintézis folyamat megkezdése előtt, az aktuális hullámfront érzékelő algoritmusnak megfelelően, ahol 8×8-tól 32×32 pixel méretű szub-apertúrák használatosak. A referencia kép mérete, amely meghatározza a SAD értéktömb méretét, szintén konfigurálható. Az alkalmazható maximális szub-apertúra méretet az egység által elfoglalt, és az FPGA-n rendelkezésre álló erőforrások határozzák meg. Az egy sorban alkalmazható szub-apertúrák számát a CMOS szenzor sor szélessége határozza meg. A *SAD egységben* felhasznált 18Kbites BlokkRAM memóriák elegendően nagyok ahhoz, hogy eltárolják a kiszámított SAD értékek teljes sorát. A szub-apertúrák mérete kulcsfontosságú az egység FPGA-n elfoglalt erőforrásigénye szempontjából, hisz a szub-apertúrák mérete határozza meg az egység számára szükséges BlokkRAM memória és egyéb logikai erőforrások mennyiségét.

Megvizsgáltam a *SAD egység* teljesítményét különböző méretű szub-apertúrák és referencia képek esetében. A *SAD egység* szintetizálása során a place and route fázis végén elvégzett statikus időzítési analízis kimutatta, hogy a választott konfigurációtól függetlenül az architektúra képes elérni a 120 MHz-es maximális működési frekvenciát. A Spartan-3 XC3S4000 FPGA-n implementált *SAD egység* Flip-Flop erőforrásigénye a 4.15. ábrán látható különböző méretű szub-apertúrák és SAD érték tömb (keresési ablak) méret esetén.




**4.15. ábra. A *SAD egység* Flip-Flop erőforrás igénye**

A 4.15. ábrán látszik, hogy a *SAD egység* erőforrás igénye négyzetesen függ a szub-apertúrák méretétől, azonban független a SAD értéktömb méretétől. Ez a következménye a magas szintű párhuzamosításnak. Ha a szub-apertúra mérete növekszik, akkor növekszik a Flip-Flop erőforrás igény is, mivel növekszik a szükséges akkumulátor regiszterek, *Referencia Regiszterek*, és összeadók száma. Megvizsgáltam azt is, hogy hogyan alakul a *SAD egység* BlokkRAM memória és 4 bemenetű LUT erőforrás igénye. A vizsgálat eredménye a 4.16. ábrán látható különböző méretű szub-apertúrák és SAD érték tömb (keresési ablak) méret esetén.



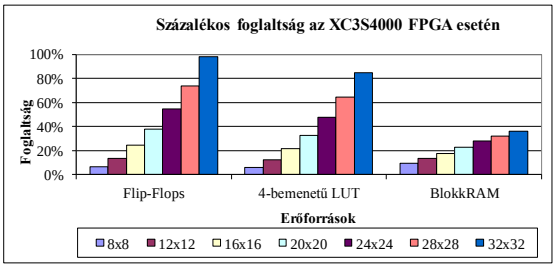

a.) 4-bemenetű LUT erőforrás igény b.) BlokkRAM erőforrás igény

**4.16. ábra. A *SAD egység* 4-bemenetú LUT és BlokkRAM erőforrás igénye**

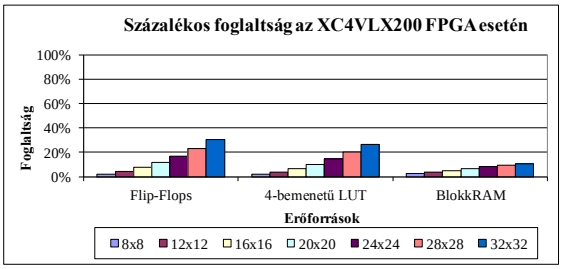
Hasonló jelenség figyelhető meg az egység BlokkRAM memória és 4 bemenetű LUT erőforrás igénye estében is, mint amit láthattunk a Flip-Flop erőforrás igény esetében is. A BlokkRAM erőforrás esetében azonban, 24×24-es szub-apertúra méret után egy kis növekedés figyelhető meg, a SAD érték tömb méretének függvényében.

A Virtex-5 FPGA családtól kezdve az FPGA-k architektúrális váltáson mentek keresztül (4 bemenetű LUT helyett 6 bemenetű LUT), így nem lenne fair a Spartan-3 és a mai modern FPGA-ak összehasonlítása. Ezért az összehasonlításokban a Spartan-3 FPGA-hoz architektúrálisan hasonló Virtex-4 FPGA család egy tagját választottam. A teljesség kedvéért azonban elvégeztem a teszteket a Virtex-6 és Virtex-7 FPGA család egy-egy tagjára is.

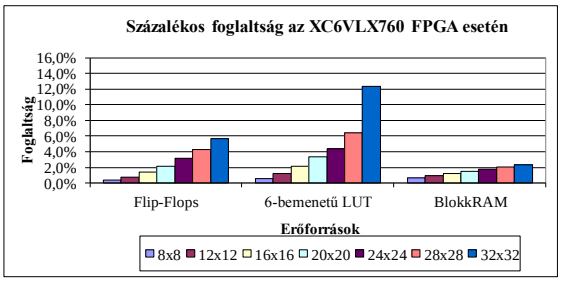
Annak érdekében, hogy még nagyobb feldolgozási sebességet érhessek el, több *SAD egység* is használható párhuzamosan. A megvalósítható *SAD egységek* száma nyilvánvalóan attól függ, hogy egy egység az FPGA erőforrásainak hány százalékát foglalja le. Ezért megvizsgáltam, hogyan alakul egy *SAD egység* erőforrás igénye, ha növelem a szub-apertúra méretet a rendelkezésre álló erőforrások tükrében. A 4.17. ábrán a *SAD egység* százalékos erőforrás igénye látható a Spartan-3 XC3S4000 a Virtex-4 XCVLX200, a Virtex-6 XC6VLX760 valamint a Virtex-7 XC7V2000T FPGA-k esetében. A referencia kép mérete $(2S-1)\times(2S-1)$ pixel volt az összehasonlítás során.



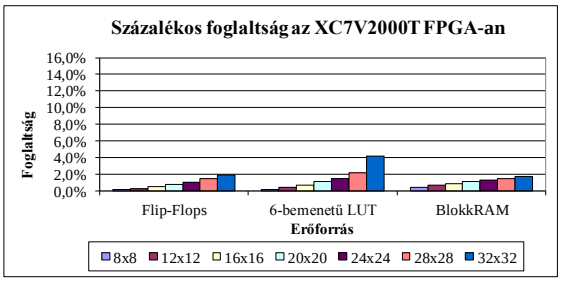

a.) Erőforrás foglaltság az XC3S4000 FPGA-n




b.) Erőforrás foglaltság az XC4VLX200 FPGA-n




c.) Erőforrás foglaltság az XC6VLX760 FPGA-n




d.) Erőforrás foglaltság az XC7V2000T FPGA-n

**4.17. ábra. A *SAD egység* százalékos erőforrásigénye különböző FPGA-ak esetében**

A 4.17. ábra a.) részén látható, hogy a Spartan-3-as FPGA esetében 12x12-es szub-apertúra méretig a szűk keresztmetszet az FPGA-n rendelkezésre álló BlokkRAM memória erőforrás, míg 12x12-es szub-apertúrákat kezelni képes *SAD egységek* esetében a Flip-Flop erőforrás lesz a szűk keresztmetszet. Hasonló helyzet figyelhető meg a Virtex-4-es FPGA esetében is a 4.17. ábra b.) részén. A 4.17. ábra c.) és d.) részén látható, hogy a Virtex-6 és Virtex-7 FPGA-ak esetében a szűk keresztmetszet minden szub-apertúra méret esetében a 6 bemenetű LUT erőforrás. Itt a maximum százalékos foglaltságot 16%-ra választottam ellentétben az 4.17. ábra a.) és b.) részével, hogy szemléletesebb legyen az erőforrás foglaltság a diagramokon. Ezek alapján meghatározható, hogy az egyes FPGA típusok esetében maximálisan hány darab *SAD egység* implementálható különböző szub-apertúra méretek esetében. A 4-2. táblázatban ezek az adatok láthatóak. A referencia kép mérete $(2S-1)\times(2S-1)$ pixel volt.

| Szub-apertúra méret pixelben (S×S) | Spartan-3 XC3S4000 | Virtex-4 XC4VLX200 | Virtex-6 XC6VLX760 | Virtex-7 XC7V2000T |
|---|---|---|---|---|
| 8×8 | *11* | *37* | *160* | *215* |
| 12×12 | *7* | *23* | *81* | *149* |
| 16×16 | *4* | *13* | *47* | *114* |
| 20×20 | *3* | *9* | *30* | *88* |
| 24×24 | *2* | *6* | *23* | *68* |
| 28x28 | *1* | *4* | *16* | *46* |
| 32×32 | *1* | *3* | *8* | *24* |

**4-2. táblázat. Implementálható *SAD egységek* száma különböző FPGA-k esetében**

A SAD értékek kiszámításához a *SAD egység* számára szükséges órajel ciklusok száma a szub-apertúrák méretétől függ, ahogy ez a 4-3. táblázatban is látható, maximális SAD érték tömb esetében. A referencia kép mérete $(2S-1)\times(2S-1)$ pixel volt.

| Szub-apertúra mérete pixelben (S×S) | Szükséges órajel ciklusok száma | Maximum szub-apertúra szám/frame | CMOS szenzor felületének százaléka |
|---|---|---|---|
| 8×8 | *120* | *2171* | *42.60%* |
| 12×12 | *156* | *941* | *41.35%* |
| 16×16 | *496* | *522* | *40.78 %* |
| 20×20 | *780* | *331* | *40.39%* |
| 24×24 | *1128* | *228* | *40.10%* |
| 28×28 | *1540* | *167* | *39.86%* |
| 32×32 | *2016* | *127* | *39.64%* |

**4-3. táblázat. A SAD értékeke kiszámításához szükséges órajel ciklusok száma 500 fps sebességű 1.3 megapixel méretű video esetében**

A 4-3. táblázatban fel van tüntetve továbbá az egy CMOS szenzor által alkotott képen az egy *SAD egységgel* maximálisan feldolgozható szub-apertúrák száma 500 fps sebességű 1.3 megapixeles képfolyamot feltételezve, valamint az is, hogy ez hány százaléka a teljes CMOS szenzor felületének. Az 4-3. táblázatban látható, hogy a valós időben egy *SAD egységgel* feldolgozható szub-apertúrák száma különböző. Egy *SAD egység* 120 Mhz-en működve 2171 darab 8×8-as méretű szub-apertúrát képes feldolgozni frame-enként. Ez csupán a 42.60%-a teljes CMOS szenzor felületnek, azonban egy ilyen *SAD egység* csupán 8.72%-át foglalja el a Spartan-3 FPGA esetében rendelkezésre álló erőforrásoknak, így három ilyen *SAD egységet* alkalmazva a teljes CMOS felület valós időben feldolgozható. Ha 32×32-es szub-apertúrákat szeretnék használni, akkor egy *SAD egységgel* 127 darab szub-apertúrát tudok valós időben kezelni. Ez 39.64%-a a CMOS szenzor teljes felületének. Ha a nagyobb teljesítményű

Virtex-4 FPGA-at használom, akkor egy ilyen *SAD egységből* 3 is használható párhuzamosan, így a teljes CMOS szenzor felület valós időben feldolgozható. Emellett ha nem Spartan-3 FPGA-t, hanem Virtex-4 FPGA-t használok, nemcsak több *SAD egységet* tudok párhuzamosan használni, hanem az egységek maximális működési órajel frekvenciája is növelhető, egészen 230 MHz-ig, ahogy azt a statikus időzítési analízis is megmutatta. A működési sebesség tovább növelhető, ha a Virtex-6 vagy Virtex-7 sorozatú FPGA-kat használok, hisz például a Virtex-7 FPGA-t használva elérhető akár a 300MHz-es működési sebesség, és 24 darab 32x32-es *SAD egység* is működtethető rajta párhuzamosan.

Az általam elért eredményeket összehasonlítottam egy korreláció alapú hullámfront szenzor megvalósításhoz, melyről részletes leírás a [46]-ban található. Ez a rendszer egy Virtex-4 SX35-10 FPGA segítségével lett implementálva. A számításokat végző mag 100 Mhz-en volt képes működni. A pontosabb összehasonlítás miatt az általam készített SAD alapú rendszer órajel frekvenciáját is lecsökkentettem 100 Mhz-re. Az összehasonlítás eredménye a 4-4. táblázatban látható.

| | **CMOS felület: 256x256 Szub-apertúra: 8x8 (pixelben)** | | **CMOS felület: 512x512 Szub-apertúra:16x16 (pixelben)** | |
|---|---|---|---|---|
| | Corr. | SAD | Corr. | SAD |
| Slice-ok | *5431* | *2769* | *9932* | *10186* |
| LUT-ok | *10161* | *3435* | *18607* | *12093* |
| Flip-flop-ok | *2096* | *3615* | *5981* | *13615* |
| Idő | *2.89ms* | *1.267ms* | *18.1ms* | *5.238ms* |
| **FI** | **15.69** | **3.51** | **179.76** | **53.35** |

**4-4. táblázat. A SAD és a korrelációs alapú rendszer összehasonlítása**

Az eredmények megmutatták, hogy a rendszerem FI (Felület-Idő) paramétere kisebb (a kisebb itt jobb), mint a korrelációs alapú rendszer FI paramétere. Egy 8×8-as szub-apertúra esetében az általam készített SAD alapú rendszer FI paramétere 22%-kal jobb, mint a korrelációs alapú rendszeré. Ez az arány pedig csak növekszik, ha nagyobb szub-apertúrákat használok. Ha 16×16-os szub-apertúrákkal dolgozom, az előbbi különbség 29%-os lesz. Tehát a SAD alapú megoldás hasonló pontosságot [47], viszont nagyobb sebességet biztosít. A rendszer teljesítménye tovább növelhető, ha több *SAD egységet* használok párhuzamosan, illetve növelem az órajel frekvenciát, akár 230Mhz-re a Virtex-4 FPGA esetében, vagy 300Mhz-re a Virtex-6 és Virtex-7 FPGA-ak esetében.

Számos FPGA alapú SAD implementációt publikáltak [42][48][49] már a különböző irodalmakban, azonban ezek mindegyike Altera FPGA-kon lett megvalósítva. Ezért az általam készített Xilinx FPGA alapú megvalósítás összehasonlítása az Altera alapú megvalósításokkal nem lenne helyénvaló a két FPGA közötti architektúrális különbözőségek miatt, azonban úgy tűnik, hogy az általam készített rendszer teljesítménye felülmúlja a korábban publikált rendszerek teljesítményét. Az általam készített speciális célú SAD architektúra jobb teljesítményt mutat (16x16 szub-apertúra: 10,186 Slice és 496 órajel ciklus szemben a publikált architektúrával, amely esetében a paraméterek 9.478 Slice és 1,600 órajel ciklus), mint a vele összehasonlítható mozgásbecslő SAD implementáció [49].

# Konklúzió

A disszertációmban két fő témát dolgoztam fel. Az egyik téma egy már létező FPGA alapú emulált digitális CNN-UM processzor, az úgynevezett FALCON processzor átalakítása annak érdekében, hogy képesek legyünk vele nemlineáris template-ek futtatására. Ennek első lépéseként a CNN Template Library v3.1-et [5] tanulmányoztam, és megállapítottam, hogy a nemlineáris template-eknek alapvetően két csoportját különböztethetjük meg. Ezek a csoportok a nullad- és az elsőrendű nemlineáris template-ek, melyek között a fő különbség a template-ek esetében alkalmazott nemlinearitás. Ezen csoportosítás alapján két FALCON processzort, egy nulladrendű nemlineáris és egy elsőrendű nemlineáris FALCON processzort fejlesztettem ki. Ezek a processzorok az eredeti FALCON processzortól a *Template Memóriájuk* és az *Aritmetikai Egységük* felépítésében különböznek. Az elkészült processzorokat alapos tesztelésnek vetettem alá. A tesztek első felében a processzorok általános és dedikált erőforrás igényét vizsgáltam meg, majd azt, hogy hogyan változik a párhuzamosan futtatható processzorok száma a Virtex-II FPGA család és a mai legmodernebb Virtex-6 és Virtex-7 FPGA család egy-egy képviselője esetén. Ezek alapján pedig meghatároztam mind a nullad- mind pedig az elsőrendű FALCON processzorral elérhető maximális számítási teljesítményt is. A kapott eredményeket összehasonlítottam a szoftveres szimulációval elérhető számítási teljesítménnyel. Ezekben a tesztekben természetesen összehasonlítottam az általam elkészített FALCON processzorokat az eredeti lineáris FALCON processzorral is.

A dolgozatom második felében pedig egy FPGA alapú adaptív optika hullámfront szenzorjának implementálásával foglalkoztam. A hullámfront szenzor FPGA alapú implementálásához a SAD eljárást alkalmaztam. A SAD eljárás segítségével meghatározható két kép optimális illeszkedési pozíciója, lényegében a képek megfelelő pixeleinek különbségének abszolút értékének meghatározásával. A SAD eljárás tehát megvalósítható, mint egy elsőrendű nemlineáris CNN operátor. Felhasználtam tehát a dolgozatom első felében elkészített elsőrendű nemlineáris FALCON processzort, hogy implementáljam a SAD operátort. Ennek érdekében kisebb átalakításokat kellett végeznem a processzor *Template Memóriájában*. Az így elkészült átalakított FALCON processzort a felületigény és az elérhető maximális sebesség szempontjából is teszteltem. Az elkészült processzorral kapcsolatban azonban két probléma is fellépett. Az egyik probléma az volt, hogy ennek a processzornak az

alkalmazásával az egyszerű kivonás és abszolútérték képzést visszavezettem szorzások elvégzésére. A másik probléma pedig, hogy az adaptív optikai kártyán található speciális hardver eszközök működése nem tette lehetővé ennek a processzornak a hatékony alkalmazását. Ezen problémák kiküszöbölése érdekében kifejlesztettem direkt a szóban forgó FPGA alapú adaptív optikai kártyára optimalizált SAD alapú hullámfront szenzor architektúrát. Az elkészült rendszer teljes mértékben konfigurálható a szub-apertúrák mérete, száma és a referencia kép tekintetében. A rendszer teljesítményét különböző méretű szub-apertúrák és referencia képek esetében is teszteltem. Az általam elkészített rendszer teljesítményét összehasonlítottam egy korrelációs alapú megvalósítással [32] is. Ez alapján megállapítottam, hogy az elkészült SAD alapú rendszer egy olyan hullámfront szenzor megvalósítás, mely jobb teljesítményt nyújt, mint az eddigi megvalósítások.

# Referenciák


[1] Maya Gokhale, (2005). *Reconfigurable Computing Accelerating Computation with Field-Programmable Gate Arrays*. Első kiadás, Springer

[2] Wayne Wolf, (2004). *FPGA-Based System Design.* Első kiadás, Prentice Hall, Modern Semiconductor Design Series

[3] Clive „Max" Maxfiled, (2009). *FPGAs World Class Designs.* Első kiadás, Elseiver

[4] Leon Chua, Tamás Roska, (2004). *Cellular Neural Networks and Visual Computing.* Első kiadás, Cambridge University Press, Cambridge

[5] K. Karacs, Gy. Cserey, Á. Zarándy, P. Szolgay, Cs. Rekeczky, L. Kék, V. Szabó, G. Pazienza, T. Roska, (2010). *Software Library for Cellular Wave Computing Engines.* 3.1. verzió, Cellular Sensory Wave Computers Labratory, Hungarian Academy of Sciences és Jedlik Laboratories of the Pazmany University, Budapest

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

[7] L. O. Chua, T. Roska, (March 1993). *The CNN paradigm*. IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, Vol. 40, Issue 3, pp. 147-156

[8] T. Roska, L.O. Chua, (March 1993). *The CNN Universal Machine: An analogic array computer*. IEEE Transactions on Circuits and Systems-II, Vol. 40, pp. 163–173

[9] P. Keresztes, Á. Zarándy, T. Roska, P. Szolgay, T. Bezák, T. Hidvégi, P. Jónás, (1999). *An Emulated Digital CNN Implementation*. Journal of VLSI Signal Processing, Vol. 23, pp. 291-303

[10] R. Dominguez-Castro, S. Espejo, A. Rodriguez-Vazquez, R. Carmona, (Dec. 1994). *A CNN Universal Chip in CMOS Technology*. Proc. of the third IEEE Int. Workshop on Cellular Neural Networks and their Application (CNNA-94), Rome, pp. 91–96

[11] Z. Nagy, P. Szolgay (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

[12] Xilinx termékek [weboldal]: http://www.xilinx.com

[13] Gustavo Linan Cembrano, Á Rodríguez-Vázquez, Servando Espejo-Meana, Rafael Domínguez-Castro (2003). *ACE-16k: A* 128×128 *Focal Plane Analog Processor with Digital I/O*, International Journal of Neural Systems, Vol. 13(6), pp. 427-434

[14] P. Dudek (2006). *An Asynchronous Cellular Logic Network for Trigger-Wave Image Processing on Fine-Grain Massivelly Paralell Arrays*, IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, Vol. 53(5), pp. 354-358

[15] A. Lopich, P. Dudek (2007). *Implementation of an Asynchronous Cellular Logic Network as a Co-Processor for a General-Purpose Massively Parallel Array*, ECCTD, Seville, Spain

[16] P. Dudek, S. J. Carey (2006). *A General-Purpose* 128×128 *SIMD Processor Array with Integrated Image Sensor*, Eletronic Letters, Vol. 42(12), pp. 678-679

[17] 176×144 Q-Eye chip [weboldal]: www.anafocus.com

[18] B. Javidi, J. L. Horner (1994). *Real-Time Optical Information Processing*, Academic Press Inc. London

[19] J. Tanida, Y. Ichioka (2000). *Digital Optical Computing*, Progress in Optics

[20] A. VanderLugt (1992). *Optical Signal Processing*, John Wiley and Sons, Inc

[21] M. S. Alam (1999) *Selected Papers on Optical Pattern Recognition Using Joint Transform Correlation*, SPIE Milestone Series Vol. MS 157, SPIE-The International Society of Optical Engineering, Bellingham, Washington USA

[22] F. T. S. Yu, S. Jutamulia (1998). *Optical Pattern Recognition*, Cambridge University Press

[23] L. Orzó, Sz. Tőkés, T. Roska (2002). *Application Issues of a Programmable Optical CNN Implementation*, Proceedings of the 7th IEEE International Workshop on Cellular Neural Networks and their Applications, pp. 156-164

[24] Aladdin 3.2 CNN Simulator [weboldal]: http://lab.analogic.sztaki.hu

[25] MatCNN – AnaLogic CNN Simulation Toolbox for MATLAB [weboldal]: http://www.analogic-computers.com/Downloads/IVToolboxes.html

[26] Cell Broadband Engine [weboldal]: http://researchweb.watson.ibm.com/cell/

[27] T. Roska, T. Kozek, D. Wolf, L. O. Chua (1992). *Solving Partial Differential Equations by CNN*, Proceedings of European Conference on Circuits Theory and Design

[28] P. Szolgay, G. Vörös, Gy. Erőss (1993). *On the Applications of the Cellular Neural Network Paradigm in Mechanical Vibrating System,* IEEE. Transactions Circuits and Systems-I, Fundamental Theory and Applications Vol. 40(3), pp. 222-227

[29] L. Kék, Á. Zarándy (1998) *Implementation of Large-Neighbourhood Non-Linear Templates on the CNN Universal Machine,* International Journal of Circuit Theory and Applications, Theory, Design and Applications of Cellular Neural Networks, Vol. 26(6), pp. 551-566

[30] Zhaoliang Cao, Lifa Hu, Dayu Li, and Li Xuan (2006). *Adaptive optics imaging system based on a high-resolution liquid crystal on silicon device,* Opt. Express Vol. 14, pp. 8013-8018

[31] D. W. de Lima Monteiro, G. Vdovin, P. M. Sarro (2004). *High-speed wavefront sensor compatible with standard CMOS technology,* Sensors and Actuators A: Physical, Vol. 109 (3), pp.220-230

[32] Poyneer, L.A., Palmer, D.W., LaFortune, K. N., Bauman, B., (2005). *Experimental results for correlation-based wavefront sensing,* Advanced Wavefront Control: Methods, Devices, and Applications III. Proc. SPIE, Vol. 5894, pp. 207-220

[33] Chang-Hui Rao, Wen-Han Jiang, Cheng Fang, Ning Ling, Wei-Chao Zhou,Ming-De Ding, Xue-Jun Zhang, Dong-Hong Chen, Mei Li, Xiu-Fa Gao and Tian Mi, (2003). *A Tilt-correction Adaptive Optical System for the Solar Telescope of Nanjing University,* Chin. J. Astron. Astrophys. Vol. 3(6), pp. 576–586.

[34] Serati, S., Xiaowei, X., Mughal, O., Linnenberger A., (2003). *High-resolution phase-only spatial light modulators with sub-millisecond response,* Proc. SPIE, Vol. 5106, pp. 138-145

[35] Rodríguz-Ramos, L. F., Marichal-Hernández, J.G., Rosa, F., (2006). *Modal Fourier wavefront reconstruction on graphics processing units,* Advances in Adaptive Optics II. Proc. SPIE, Vol. 6272,

[36] Marichal-Hernandez, G., Rodriguez-Ramos, J.M., and Fernando Rosa, J., (2007). *Modal Fourier wavefront reconstruction using graphics processing units,* Journal of Electronic Imaging Vol. 16(2)

[37] Rosa, F.L., Marichal-Hernandez, J.G., Rodriguez-Ramos, J.M., (2004). *Wavefront phase recovery using graphic processing units (GPUs),* Optics in Atmospheric Propagation and Adaptive Systems VII. Proc. SPIE, Vol. 5572, pp. 262-272

[38] Saunter C.D., Love G.D., Johns, M., Holmes, (2005). *FPGA technology for high speed, low cost adaptive optics,* Proc. SPIE 5th International Workshop on Adaptive Optics in Industry and Medicine

[39] Rodríguez-Ramos, L.F., Viera, T., Herrera, G., Gigante, J.V., Gago, F., Alonso, Á., (2006). *Testing FPGAs for real-time control of adaptive optics in giant telescopes,* Advances in Adaptive Optics II. Proc. SPIE, Vol. 6272

[40] Rodríguez-Ramos, L.F., Viera, T., Gigante, J.V., Gago, F., Herrera, G., Alonso, Á., Descharmes, N., (2005). *FPGA adaptive optics system test bench,* Astronomical Adaptive Optics Systems and Applications II. Proc. SPIE, Vol. 5903, pp. 120-128

[41] Saunter C.D. and Love, G.D., (2007) *Low cost, high speed control for adaptive optics,* Proc. SPIE 6th International Workshop on Adaptive Optics for Industry and Medicine

[42] Wong, S., Vassiliadis, S., Cotofana, S., (2002) *A sum of absolute differences implementation in FPGA hardware,* Euromicro Conference Proc. Vol. 28, pp. 183–188

[43] Microdisplay [webooldal]: http://www.microdisplay.com

[44] Jason Porter, Hope Queener, Julianna Lin, Karen Thorn, Abdul A. S. Awwal, (2006). *Adaptive Optics for Vision Science: Principles, Practices, Design and Applications,* Wiley

[45] Holoeye [weboldal]: http://holoeye.com/download_area.html

[46] Trujillo J.S., Valido, M.R., Rodríguez Ramos, L.F., Boemo, E., Rosa, F., Rodríguez Ramos, J.M., *Real time phase-slopes calculations using FPGAs,* Proceedings of SPIE, 7015.

[47] R. Sridharan, A. Raja Bayanna and P. Venkatakrishnan (2005). *Simulations of Solar AO Systems,* Springer Berlin, Science with Adaptive Optics

[48] Ambrosch K., Humenberger, M., Kubinger, W., Steininger, A., (2008). *SAD-Based Stereo Matching Using FPGAs,* Springer, Embedded Computer Vision

[49] Li, B.M. and Leong, P.H. (2008). *Serial and Parallel FPGA-based Variable Block Size Motion Estimation Processors,* Journal of Signal Processing Systems Vol. 51, pp. 77–98.

[50] Swapan K. Saha (2007). *Diffraction-Limited Imaging with Large and Moderate Telescopes,* World Scientific Publishing Co. Pte. Ltd. pp. 259-260

[51] Babcock, H. W. (1953). *The Possibility of Compensating Astronomical Seeing* PASP Vol. 65, pp. 229-360.

[52] Junzhong Liang, David R. Williams, and Donald T. Miller (1997). *Supernormal vision and high-resolution retinal imaging through adaptive optics*, Journal of Opt. Soc. Am., Vol. 14(11), pp. 2884-2892.

[53] Dayton, D., Gonglewski, J., Restaino, S., Martin, J., Philips, J., Hartman, M., Kervin, P. Snodgress, J. Browne, S., Heimann, N., Shilko, M., Pohle, R., Carrion, B., Smith, C. and Thiel, D. (2002), *Demonstration of new technology MEMS and liquid crystal adaptive optics on bright astronomical objects and satellites*, Opt. Express Vol. 10, pp. 1508-1519.

[54] Richards, K., Rimmele, T., Hill, R., Chen, J. (2004). *High speed low latency solar adaptive optics camera*, Proc. SPIE, Vol. 5171, pp. 316-325.

[55] Zsolt Vörösházi (2009). *Investigation of Emulated-Digital CNN-UM architectures: Retina model and Cellular Wave Computing architecture implementation on FPGA*, Ph.D. Thesis.

# Tézisek

1. Téziscsoport: Nemlineáris template-eket futtató emulált digitális CNN-UM megvalósítása FPGA-n

A jelenlegi CNN implementációk esetén - kivéve a lassú szoftveres szimulációt - nincs lehetőség a nemlineáris template-ek alkalmazására. Bizonyos CNN-el megoldható feladatok (pl.: gradiens intenzitásbecslés, szürkeskálás kontúrdetektálás) esetén ez a hiányosság nem jelent problémát, ugyanis a nemlineáris B template-ek helyettesíthetők megfelelő lineáris template-ek halmazával. Természetesen ez a helyettesítés, még ha ugyanolyan eredménnyel is szolgál, bonyolultabb, mint egyetlen nemlineáris template alkalmazása. Vannak azonban olyan feladatok (pl.: mediánszűrés, a korábbiakban említett SAD operátor, szürkeskálás erózió/dilatáció, hisztogramgenerálás), melyek esetében nemlineáris A vagy D template-et kell használni, ami azonban nem helyettesíthető lineáris template-ek halmazával. Így ezeket a feladatokat csak nemlineáris template-ek alkalmazásával lehet megoldani.

A CNN Template Library v3.1 tanulmányozása során a nemlineáris template-eket - a template nemlineáris értékeit meghatározó nemlinearitás alapján - két csoportba soroltam. Ezek az úgynevezett nullad- és elsőrendű nemlineáris template-ek. Nulladrendű nemlineáris template-eknek nevezzük azokat a template-eket, amelyek olyan szakaszokból épülnek fel, amelyeken belül a függvény értéke konstans. Az elsőrendű nemlineáris template-ek közé olyan nemlineáris template-ek tartoznak, amelyekben a nemlinearitás olyan szakaszokat is tartalmaz, ahol a függvény értéke nem konstans, hanem a független változó lineáris függvénye, a megfelelő meredekséggel.

Figyelembe véve a nullad- és elsőrendű nemlineáris template-ek alkalmazásának speciális igényeit, a FALCON emulált digitális CNN-UM architektúrát alakítottam át úgy, hogy képes legyen kezelni az ebbe a template osztályba tartozó template-eket is. Az elkészült architektúrát felületigény és számítási teljesítmény szempontjából teszteltem és összehasonlítottam az eredeti FALCON architektúrával és a szoftveres szimulációval (MatCNN, Intel Core i5 M540 2.53 GHz). A vizsgálataim megmutatták, hogy a nullad- és elsőrendű nemlineáris FALCON architektúrával elérhető számítási teljesítmény jelentősen túlszárnyalja a szoftveres szimulációval elérhető teljesítményt.

1.1. Nulladrendű nemlineáris template-eket futtató emulált digitális CNN-UM architektúra implementációja FPGA-n

**Megadtam a FALCON emulált digitális CNN-UM architektúra kiterjesztését nulladrendű nemlineáris CNN template-ek használatára.** Ennek érdekében az eredeti FALCON architektúra template-ek kezeléséért felelős részét alakítottam át úgy, hogy képes legyen a nulladrendű nemlinearitás szakaszaihoz tartozó konstans értékek eltárolására, és ez alapján a nemlineáris template értékek meghatározására. Elemeztem az így kiterjesztett architektúra FPGA implementációját felület és végrehajtási sebesség szempontjából különböző FPGA családok (Virtex-II, Virtex-4, Virtex-6, Virtex-7) esetében.

Megmutattam, hogy a Virtex-6 XC6VS475T és a Virtex-7 XC7VX9800T típusú FPGA-k esetében az állapot bitszélességétől függetlenül 168 darab processzort lehet elhelyezni a Virtex-6, míg 300 darabot a Virtex-7 FPGA-n. Megmutattam, hogy a Virtex-6 FPGA-n a maximális számú 18 bites állapotszélességű nulladrendű FALCON processzort implementálva 33600 millió cellaiteráció/másodperc a maximális számítási teljesítmény, míg a Virtex-7 FPGA-t alkalmazva további majdnem kétszeres (1,786-szoros) sebességnövekedés érhető el. Ezeket az eredményeket összehasonlítva a szoftveres szimulációval, mely esetén az elérhető maximális számítási teljesítmény 13 millió cellaiteráció/másodperc, a teljesítménynövekedés 3 nagyságrendnyi.

1.2. Elsőrendű nemlineáris template-eket futtató emulált digitális CNN-UM architektúra implementációja FPGA-n

**Megadtam a FALCON emulált digitális CNN-UM architektúra kiterjesztését elsőrendű nemlineáris template-ek használatára.** Ebben az esetben is az eredeti FALCON processzor template memóriáját kellett kibővíteni a megfelelő összeadó és szorzó áramkörökkel, úgy, hogy képes legyen meghatározni a nemlineáris template értékeket. Elemeztem az így kiegészített architektúra FPGA implementációját a felület és a végrehajtási sebesség szempontjából különböző FPGA családok (Virtex-II, Virtex-4, Virtex-6, Virtex-7) esetében.

Megmutattam, hogy a Virtex-6 XC6VSX475T és a Virtex-7 XC7VX9800T típusú FPGA-k esetében az állapot bitszélességétől függően 112-96 darab elsőrendű FALCON processzor implementálható a Virtex-6, míg 200-171 a Virtex-7 FPGA-t használva. Megmutattam, hogy ha a maximális 112 darab 18 bites elsőrendű FALCON processzort implementálom a Virtex-6 FPGA-n, akkor 22400 millió cellaiteráció/másodperces számítási teljesítmény, míg a Virtex-7 FPGA alkalmazásával további majdnem kétszeres sebességnövekedés érhető el. Ez mindkét FPGA esetében három

nagyságrendnyi teljesítménynövekedést jelent a szoftveres szimulációhoz képes, mellyel a maximálisan elérhető számítási teljesítmény 9 millió cellaiteráció/másodperc.

2. Téziscsoport: SAD operátor alapú hullámfront szenzor megvalósítása FPGA alapú adaptív optikai rendszeren

Az AO rendszerek igen nagy jelentőséggel bírnak a csillagászat területén, azonban egyre nagyobb teret hódítanak a hadiipar és az orvostudományon belül a szemészetben is. Minden olyan területen jól alkalmazhatók, ahol nagy expozíciós idejű, nagy felbontású képek készítése a cél. A fény hullámfrontját (a hullámfront a fényhullám egy vonalban vagy felületen elhelyezkedő azonos fázisban lévő pontjainak halmaza) a gyorsan változó turbulens közeg módosítja, eltorzítja. Ez a jelenség jelentősen rontja a képalkotó rendszer teljesítményét. Az adaptív optikai rendszer célja, hogy a hullámfront torzulásait kiküszöbölje. Az adaptív optikai rendszer két fő komponense az adaptív tükör és a hullámfront szenzor. A hullámfront szenzor folyamatosan méri a hullámfront torzulásait, mely mérések alapján a vezérlő rendszer folyamatosan módosítja az adaptív tükör felületét, ezzel kompenzálva a hullámfront torzulásait. Az adaptív optikai rendszerekben hullámfront szenzorként gyakran alkalmazzák a HS hullámfront szenzort, mely alapvetően két részből épül fel, egy elemi lencse tömbből, melyet lenslet-nek nevezünk, és egy CCD (Charge-Coupled Device) vagy CMOS érzékelőből. A képalkotó rendszer lencséje által létrehozott kép a lenslet tömbre esik. A lenslet tömböt alkotó elemi lencsék mindegyike létrehoz egy miniatűr képet a forrás objektumról, és szétbontja az apertúrát úgynevezett szub-apertúrákra. A szub-apertúrák képei egy CCD vagy CMOS szenzor segítségével detektálhatók. A torzulásmentes hullámfrontú fény esetén a szub-apertúrák a szenzor egy-egy pontjára esnek, melyről a CCD vagy CMOS szenzorral alkotott kép a referencia kép. A torzult hullámfrontú fény esetében a szub-apertúrák nem ugyanoda esnek, mint a referencia kép esetében. Ezt a képet összehasonlítva a referencia képpel meghatározhatók a hullámfront torzulásai. Ezen információ birtokában a hullámfront torzulásai kiküszöbölhetők az adaptív tükör segítségével.

Kisméretű objektumok esetén négy cellás HS szenzor is elegendő, nagyobb objektumok esetében azonban az eltolódásokat csak magasabb felbontású szenzorok, és nagyobb szub-apertúrák alkalmazásával lehet meghatározni. Ehhez nagyobb számítási teljesítmény kell, ráadásul az adaptív optikai rendszer korrigáló képessége annál jobb, minél kisebb a késleltetés az érzékelés és a korrigálás között. Ezen problémák kiküszöbölése érdekében az MTA SZTAKI-ban elkészítettek egy FPGA alapú kompakt

adaptív optikai rendszert (1. ábra), melynek FPGA alapú hullámfront szenzor részének implementálása volt a feladatom. Ennek megvalósításához az FPGA tulajdonságait és korlátait figyelembe véve a SAD eljárást választottam. Első lépésben a kép SAD értékeit kell kiszámítani a keresési ablakban (ahol az optimális illeszkedést keressük), ezután pedig meghatározni ezen értékek minimumát. A módszer segítségével az összes szub-apertúra kép referenciához képesti elmozdulása meghatározható.

2.1. A nemlineáris template futtató emulált digitális CNN-UM architektúra, mint SAD operátor alapú hullámfront szenzor.

**Megadtam a SAD operátor alapú hullámfront szenzor implementációját a nemlineáris template-ekkel kiegészített FALCON architektúrára és megvizsgáltam a megoldás hatékonyságát.**

Megmutattam, hogy a SAD operátort futtatni képes FALCON processzor általános és dedikált erőforrásigénye lényegében megegyezik az eredeti elsőrendű FALCON processzor erőforrás igényével, csupán egy plusz BlokkRAM-ot igényel. Megmutattam, hogy az adaptív optikai kártyán rendelkezésre álló Sptartan-3 XC3S4000 FPGA-n maximálisan 8 darab átalakított FALCON processzor implementálható és ezzel 320 millió cellaiteráció/másodperces számítási sebesség érhető el 3x3 pixel méretű szub-apertúra és 5x5 pixel méretű referencia kép esetben. Megmutattam, hogy ha a szub-apertúra méretét a valós alkalmazásokban használt méretre (16x16) választom, akkor már csak 3 darab FALCON processzor implementálható, és csak 120 millió cellaiteráció/másodperces számítási sebesség érhető el. A számítási teljesítmény növelése érdekében, valamint hogy megfeleljek a rendelkezésemre álló kártya hardver követelményeinek, az architektúrát újragondoltam, és létrehoztam egy nagy sebességű, speciálisan erre a feladatra optimalizált architektúrát.

2.2. SAD operátor alapú hullámfront szenzor optimális megvalósítása az FPGA alapú adaptív optikai rendszeren.

**A SAD operátor alapú hullámfront szenzor optimalizált FPGA alapú implementációját adtam meg.** Elemeztem az elkészített architektúrát felület és végrehajtási sebesség szempontjából különböző FPGA családok (Virtex-II, Virtex-4, Virtex-6, Virtex-7) esetében.

Megmutattam, hogy az elkészített hullámfront szenzor központi SAD egységének általános és dedikált erőforrásigénye négyzetesen nő a szub-apertúrák méretének növelésével. Megmutattam, hogy a Spartan-3 XC3S4000 FPGA esetén egy darab 120MHz-en működő SAD egységgel 2171 8x8-as és 127 32x32-es szub-apertúrát lehet

valós időben feldolgozni 500 fps sebességű 1.3 megapixel felbontású video esetében. Megmutattam, hogy ha a lényegesen nagyobb Virtex-7 XC7V2000T FPGA-at használom melyen egy SAD egység akár 300MHz-es órajellel is képes működni, akkor a teljes CMOS felület feldolgozható valós időben. Megmutattam, hogy 8x8-as, illetve 16x16-os szub-apertúra esetén az általam készített rendszer Felület-Idő (FI) paramétere 29%, illetve 22%-kkal jobb, mint egy hagyományos korreláció alapú rendszer FI paramétere. Az elkészült rendszert összehasonlítva más SAD alapú rendszerekkel az általam készített architektúra jelentős teljesítménynövekedést mutat (16x16-os szub-apertúra esetén: 10186 slice és 496 órajel ciklus szemben a 9478 slice és 1600 órajel ciklus).

# A Szerző publikációi

**Referált idegen nyelvű folyóiratcikkek:**

Z. Nagy, L. Kék, **Z. Kincses**, A. Kiss, P. Szolgay: „Toward Exploitation of Cell Multi-processor Array in Time-Consuming Applications by Using CNN Model" (International Journal of Circuit Theory and Applications, Wiley, Vol. 36., Special Issue: Cellular Wave Computing Architecture, Július-Szeptember. 2008. pp. 605-622 IF: 2.389 - 2008, ISSN: 0098-9886)

**Z. Kincses**, L. Orzó, Z. Nagy, Gy, Mező, P. Szolgay: „High-speed, SAD based wavefront sensor architecture implementation on FPGA" (Journal of Signal Processing Systems, Springer, DOI: 10.1007/s11265-010-0487-4, 2011, IF: 0.623-2010, Vol. 63, pp. 279-290, ISSN: 1939-8018)

**Nemzetközi referált konferencia kiadványban megjelent idegen nyelvű közlemények:**

**Z. Kincses**, Z. Nagy, P. Szolgay: "Implementation of nonlinear template runner emulated digital CNN-UM on FPGA" (CNNA 2006, Törökország, Isztambul, Augusztus 28-30)

Z. Nagy, L. Kék, **Z. Kincses**, P. Szolgay: "CNN model on Cell multiprocessor array" (ECCTD 2007, Spanyolország, Sevilla, Augusztus 26-30.)

Z. Nagy, L. Kék, **Z. Kincses**, A. Kiss, P. Szolgay: "Toward Exploitation of Cell Multi-processor Array in Time-Consuming Applications by Using CNN Model" (CNNA 2008, Spanyolország, Santiago de Compostela, Július 14-16.).

**Z. Kincses**, Z. Nagy, L. Orzó, P. Szolgay, Gy. Mező: "Implementation of a parallel SAD based wavefront sensor architecture on FPGA" (ECCTD 2009, Törökország, Antalya, Augusztus 23-27.)

**Z. Kincses**, Zs. Vörösházi, Z. Nagy, P. Szolgay, P. Szolgay, T. Laviniu, A. Gacsádi: „Investigation of area and speed trade-offs in FPGA implementation of an image correlation algorithm” (CNNA 2012, Torino, Olaszország, Augusztus 29-31.)

**Magyar konferencia kiadványban megjelent közlemények:**

**Z. Kincses** (2009): "High-speed, SAD based wavefront sensor architecture implementation on FPGA", Proceedings of the 7th PhD Mini-Symposium, Pannon Egyetem, Veszprém

**Kincses Z.** (2008), "Toward Exploitation of Cell Multi-processor Array in Time-Consuming Applications by Using CNN Model", Proceedings of the 6th PhD Mini-Symposium, Pannon Egyetem, Veszprém

**Kincses Z.** (2007), "CNN model on Cell multiprocessor array", Proceedings of the 5th PhD Mini-Symposium, Pannon Egyetem, Veszprém

# Theses in English

1. Thesis group: Implementation of nonlinear template runner emulated digital CNN-UM on FPGA

Using present CNN implementations – except for the slow software simulation – there are no possibilities to use nonlinear templates. In case of some tasks which can be solved by a CNN (for example: gradient intensity estimation, grayscale contour detection), this inadequacy can be eliminated, because nonlinear B templates can be decomposed into a series of linear templates. Naturally, this substitution is more complicated than applying one nonlinear template, even if the result is the same. However, template decomposition cannot be used in case of several tasks (median filter, the previously mentioned SAD operator, grayscale erosion/dilation, histogram generation), where nonlinear A or D type templates should be applied. Therefore, the only way to solve these tasks is the application of nonlinear templates.

In the course of studying the CNN Template Library v3.1, I classified the nonlinear templates – based on the nonlinearity defined by the nonlinear template values – into two groups. These are the zero- and first-order nonlinear templates. A template is called a zero-order nonlinear template if the nonlinear function of the template only contains constant sections. A template is called a first-order nonlinear template if the nonlinear function of the template contains first-order linear functions of the variables.

I extended the FALCON emulated digital CNN-UM architecture to handle zero- and first-order nonlinear templates taking into account their special requirements. I measured the area requirement and computational speed of these architectures and compared them to the results of the original linear FALCON architecture and the software simulation (MatCNN, Intel Core i5 M540 2.53 GHz). My investigations show that the computational performance of the zero- and first-order FALCON processor is significantly higher than the performance of the software simulation.

1.1. Implementation of the zero-order nonlinear template runner emulated digital CNN-UM architecture on FPGA

**I gave the extension of the FALCON emulated digital CNN-UM for the application of zero-order nonlinear templates.** I redesigned the template-handling part of the FALCON processor, to make it able to store the constant values of the sections of the zero-order nonlinearity, and based on these values, to determine the nonlinear template values. I analyzed the area requirement and computating

performance of the extended architecture in case of different FPGA families (Virtex-II, Virtex-4, Virtex-6, Virtex-7).

I showed that in case of Virtex-6 XC6VS475T and Virtex-7 XC7VX9800T type FPGAs, 168 processors can be implemented on Virtex-6 and 300 processors can be implemented on Virtex-7 independently from the bit width of the state value. I showed that the maximal computational performance is 33600 million celliterations/second when implementing a 168 zero-order FALCON processor on Virtex-6 FPGA using 18 bit state width, and the Virtex-7 FPGA is almost 2-times (1.786) faster. Compared these results to the software simulation, where the maximal computational performance is 13 million celliterations/second, the performance is increased by 3 orders of magnitude.

1.2. Implementation of first-order nonlinear template runner emulated digital CNN-UM architecture on FPGA

**I gave the extension of the FALCON emulated digital CNN-UM for the application of first-order nonlinear templates.** In this case, the template memory of the FALCON processor is also extended with the appropriate adder and multiplier circuits to be able to determine the nonlinear template values. I analyzed the area requirement and computing speed of the extended architecture in case of different FPGA families (Virtex-II, Virtex-4, Virtex-6, Virtex-7).

I showed that in case of Virtex-6 XC6VSX475T and Virtex-7 XC7VX9800T type FPGAs, 112-96 first-order FALCON processors can be implemented on Virtex-6 and 200-171 first-order FALCON processors can be implemented on Virtex-7, depending on the bit width of the state value. I showed that the maximal computational performance is 22400 million celliterations/second when implementing 112 first-order FALCON processors using 18 bit state width on Virtex-6 FPGA, while the Virtex-7 FPGA is almost 2-times (1.786) faster. In this case, both FPGAs offer 3 orders of magnitude higher performance compared to the software simulation where the maximal computing speed is 9 million celliterations/second.

2. Thesis group: Implementing a SAD operator-based wavefront sensor architecture on an FPGA-based adaptive optic system.

AO systems have a great importance in the field of astronomy. However, it is also becoming increasingly important in the field of military and medicine (ophthalmology). Therefore, these systems can be applied in fields where the goal is creating pictures with high exposition time and high resolution. The rapidly changing turbulent media can modify or distort the wavefront (the wavefront is a line or surface of the points which

are in the same phase) of the incoming light. The performance of the imaging system is significantly deteriorated by this phenomenon. The goal of the adaptive optic system is to eliminate the distortions of the wavefront. The adaptive optic system is built up of two main components, the adaptive mirror and the wavefront sensor. Distortions of the wavefront are continuously measured by the wavefront sensor. Based on these measurements, the control system can modify the surface of the adaptive mirror continuously, to compensate the distortions of the wavefront. The HS wavefront sensor, which is frequently applied in adaptive optic systems, is basically built up of two parts, the lenslet array and the CCD (Charge-Coupled Device) or CMOS sensor. In a HS sensor, the image of the incoming pupil is projected on a lenslet array. Each lens of the array forms a miniature image of the source object and divides the incoming aperture into sub-apertures. The images of these sub-apertures are detected by the CCD or CMOS sensor. The shifts of these images from the reference positions (positions without aberrations) specify the local wavefront slopes at the locations of the corresponding sub-apertures (the wavefront is regarded locally flat but tilted). The overall shape of the wavefront extended for the whole pupil can be reconstructed by assembling all these local tilted surfaces with the adaptive optic system.

For a point source object (such as a star), simple quad-cell based HS sensors can measure these shifts. However, in the case of extended objects (the Sun or the Moon), these shifts can be determined only with a higher resolution sensor from the correlations between the sub-aperture images and the reference sub-aperture images. In this case, a higher computational speed is required, moreover, correcting ability of the adaptive optic system is better if the delay between sensing the wavefront and correction is smaller than the evolution time of the medium being corrected for. To eliminate these problems, a novel FPGA-based compact adaptive optic system was developed in MTA-SZTAKI (Figure 1). My task was to implement the wavefront sensor part of this system. Considering the limitation of the FPGA devices and the special parameterization of the mandatory wavefront sensors, the Sum of Absolute Differences (SAD) method was chosen to implement the required correlation-like processing. First, the SAD values of the picture are calculated, and the minimum of them is determined. With this method, the displacement of the images of all sub-apertures is determined with respect to a reference image.

### 2.1. Nonlinear template runner emulated digital CNN-UM as SAD-based wavefront sensor architecture.

**I gave the implementation of the SAD operator-based wavefront sensor on the FALCON architecture extended with nonlinear templates, and I investigated the effectiveness of this architecture.**

I showed that the general and dedicated resource requirement of the SAD operator runner FALCON processor is essentially the same as the original first-order FALCON processor. It only requires one additional BlockRAM. I showed that using the Spartan-3 XC3S4000 FPGA on the adaptive optic card, the number of implementable extended FALCON processors is 8. Using all these processors, a 320 million celliterations/second computing performance can be reached in case of 3x3 pixels sub-apertures and 5x5 pixels reference image. I showed that when the sub-aperture size is increased to 16x16 (commonly used in real applications), only 3 extended FALCON processors can be implemented, and the maximal computing performance is 120 million celliterations/second. In order to improve the computational performance of the architecture and to suit the requirement of the adaptive optical board, I redesigned the architecture to create a high speed special purpose architecture.

2.2. Optimal implementation of SAD operator based wavefront sensor on FPGA-based adaptive optical card.

**I gave the optimized FPGA-based implementation of the SAD operator-based wavefront sensor architecture.** I analyzed the area requirement and computational speed of the architecture in case of different kinds of FPGA families (Virtex-II, Virtex-4, Virtex-6, Virtex-7).

I showed that the general and dedicated resource requirement of the SAD unit, which is the central part of the wavefront sensor, is increasing quadratically according to the size of the sub-apertures. I showed that in case of Spartan-3 XC3S4000 FPGA using one 120MHz SAD unit 2171 8x8 pixels and 127 32x32 pixels sized sub-apertures can be processed in real time. I showed that using the significantly larger Virtex-7 XC7V2000T FPGA, where the clock frequency of the SAD unit can reach the 300Mhz, the whole CMOS surface can be processed in real time. I showed that in case of 8x8 pixels and 16x16 pixel sized sub-apertures, the AT (Area Time) parameter of my system is 29%, and 22% better respectively than the traditional correlation-based system. Compared to other SAD based architectures, my architecture shows superior performance (in case of 16x16 pixels sub-aperture: 10186 slice and 496 clock cycles facing 9478 slice and 1600 clock cycles).