

Magyar Tudományos Akadémia
Számítástechnikai és Automatizálási Kutatóintézet

Analogikai és Neurális Számítások Laboratórium

Layout hibadetekció a nyomtatott áramköri lapokon
(PCB) és egy emulált digitális CNN-UM architektúra
(CASTLE) optimalizálása

PhD értekezés

írta: ***Hidvégi Timót***

Témavezető:

Dr. Szolgay Péter

MTA doktora

Budapest

2002

Az értekezés bírálatai és az értekezés nyilvános védéséről készült jegyzőkönyv a Budapesti Műszaki Egyetem Villamosmérnöki és Informatikai Kar Dékáni Hivatalában — Budapest, XI., Egeri József u. 18. V.1. épület, földszint — elérhető.

Principium sapientiae timor Domini et scientia sanctorum prudentia.

(Proverbia 9,10)

A bölcsesség kezdete az Úr félelme, és a Szentet ismerni - az az okosság.

(Példabeszédek 9,10)

Tartalomjegyzék

<i>KÖSZÖNETNYÍLVÁNÍTÁS</i>	1
1. BEVEZETÉS	3
2. A CNN HÁLÓZAT ALAPJAI	6
2.1 Neurális hálózatok	6
2.2 Celluláris Neurális Hálózatok elmélete	6
2.2.1 A CNN processzor modellje.....	8
2.3 A CNN-UM definíciója.....	11
2.4 CNN-UM implementációk	13
2.4.1 Szoftveres implementáció	13
2.4.2 Emulált digitális megvalósítás.....	14
2.4.3 Analóg CNN-UM.....	14
2.4.4 Optikai rendszerek.....	14
2.5 A CNN-UM implementációk összehasonlítása.....	14
3. A NYOMTATOTT ÁRAMKÖRÖK GYÁRTÁSA, ELLENORZÉSE	17
3.1 A nyomtatott áramkörök felépítése, osztályozása.....	17
3.2 A nyomtatott áramkörön található hibák.....	18
3.3 Különböző optikai hibakereső eljárások bemutatása	20
3.3.1 Referencia bázisú módszerek	21
3.3.2 Nem referencia bázisú módszerek.....	22
3.3.3 Hibrid módszerek	23
3.4 A hibadetektáló algoritmusok használata.....	26
4. ANALOGIKAI ALGORITMUSOK A NYOMTATOTT ÁRAMKÖRÖK GYÁRTÁSI, TERVEZÉSI HIBÁINAK DETEKTÁLÁSÁRA	28
4.1 A zárlatdetektáló algoritmus.....	28
4.1.1 A “recall” template működése.....	29

4.1.2 A zárlatdetektáló analogikai algoritmus.....	30
4.1.3 Egy tesztáramkör analízise	34
4.2 Illesztési hibát detektáló analogikai algoritmus	37
4.2.1 Logikai függvények a bináris képeken.....	37
4.2.2 Az illesztési hibát detektáló algoritmus.....	40
4.2.3 Két tesztáramkör analízise.....	42
4.3 Összefoglalás.....	44
5. AZ EMULÁLT DIGITÁLIS CNN-UM (CASTLE) BEMUTATÁSA	46
5.1 CASTLE architektúra.....	46
5.2 A processzormag kialakítása	47
5.3 CASTLE processzor felépítése.....	50
5.4 CASTLE processzor megvalósítása szilíciumon.....	52
5.5 CASTLE processzor tervezési metodikája.....	56
6. OPTIMALIZÁLT, KIBOVÍTETT CASTLE ARCHITEKTÚRÁK.....	58
6.1 Különböző architektúrák osztályozása, csoportosítása	59
6.2 Újrakonfigurálható aritmetikai egységek a CASTLE architektúrában.....	59
6.2.1 Hely szerint optimalizált újrakonfigurálható aritmetikai egység.....	60
6.2.2 Sebesség szerint optimalizált újrakonfigurálható aritmetikai egység.....	62
6.2.3 A különböző aritmetikai egységek összehasonlítása.....	63
6.2.4 Az újrakonfigurálható architektúrák memóriaszervezése, vezérlése	65
6.2.5 Az újrakonfigurálható architektúrák használata.....	67
6.3 Pipeline egy digitálisan emulált CNN-UM (CASTLE) aritmetikában.....	67
6.3.1 Pipeline az aritmetikai modulok között.....	67
6.3.2 Pipeline a blokkokon belül	70
6.3.3 Az új CASTLE architektúra felhasználása.....	72
6.4 CASTLE aritmetika optimalizálása szilícium felület szerint	74
6.4.1 Oszlopszimmetrikus template-ek használata.....	76
6.4.2 Nem oszlopszimmetrikus template-ek használata.....	78
6.4.3 Az aritmetikai egységek összehasonlítása.....	79

6.5 Az optimalizált CASTLE aritmetikai egységek összefoglalása.....	80
7. AZ OPTIMÁLIS ARITMETIKAI EGYSÉGEK “KEVERÉSE”.....	82
7.1 Egy és két szorzós aritmetikai egység.....	82
7.2 Újrakonfigurálható aritmetikai egység pipeline-nal.....	83
7.2 Összefoglalás.....	85
8. ÖSSZEFOGLALÁS.....	86
IRODALOMJEGYZÉK.....	89
A szerzo publikációi.....	89
CNN, PCB, tervezés tárgyú publikációk.....	90
1. FÜGGELÉK (TEMPLATE GYUJTEMÉNY)	95
1.1 Bináris matematikai morfológia.....	95
1.2 HLF5: 5x5-ös blokkokban történő alszürke árnyalatok létrehozása (5x5 halftoning).....	97
1.3 LOGAND: Logikai ÉS kapcsolat.....	99
1.4 LOGNOT: Szürke tónusú – bináris leképzés küszöböléssel.....	100
1.5 LOGOR: Logikai VAGY kapcsolat.....	101
1.6 SMKILLER: A kisebb fekete objektumok eltüntetése.....	102
1.7 TRESHOLD: Szürke tónusú – bináris leképzés küszöböléssel.....	103
2. FÜGGELÉK (AZ AMC FILE-OK, FUTTATÁS).....	104
2.1 Zárlatkereső algoritmus AMC kódja.....	104
2.2 Illesztési hibákat detektáló algoritmus AMC kódja	105
3. FÜGGELÉK (LOGIKAI PROCESSZOR TESZTKÖRNYEZETE).....	107
4. FÜGGELÉK (A CASTLE).....	111
4.1 Néhány egység layout ábrája.....	111
4.2 Néhány egység VIRTEX schematic rajza	114

Köszönetnyilvánítás

Ezúton is szeretném megköszönni Édesanyámnak, Édesapámnak és Bátyámnak azt az önzetlen támogatást, segítséget, türelmet, figyelmességet, amelyek nélkül számomra a doktori fokozat megszerzése lehetetlen lett volna.

Köszönöm Roska Tamásnak azt a lehetőséget, hogy a doktori iskolájában tanulhattam, képezhettem magam.

Nagyon köszönöm Szolgay Péternek, hogy vállalta azt a feladatot, azt a kihívást, hogy konzulensemként elvezet a doktori fokozatig. Köszönettel tartozom Neki azért is, hogy az általa vezetett chiptervezésekben részt vehettem, illetve a Veszprémi Egyetemen oktathattam.

Külön köszönet illeti Keresztes Pétert, akitől nagyon sokat tanultam. Ő volt az, aki megmutatta nekem a mikroelektronika szépségeit, a digitális áramkörök tervezésének különböző trükkjeit. Keresztes Péter indított el a mikroelektronika szép, de rögös útján.

Nagyon köszönöm Nagy Nórának a segítségét, hogy a különböző angol nyelvű cikkeimet lektorálta, és így azok megfeleltek az angol nyelvtan szabályainak. Hálás vagyok Neki azért is, hogy nagyon sokat segített nekem az angol nyelv megismerésében is. A lelkiismeretes, türelmes tanítására, a baráti beszélgetésekre, az önzetlen segítségére mindig emlékezni fogok.

Köszönettel tartozom Hosszú Gábornak és Szatmári Istvánnak, akik a házi védésem során bírálókként hasznos megjegyzéseket tettek, emelve ezzel a disszertációm színvonalát.

Itt is szeretném megköszönni Bezák Tamásnak, *Tomasznak* is azt az önzetlen, baráti segítségét, amelyre mindig "építhettem". Nagy öröm és megtiszteltetés számomra, hogy a barátomnak mondhatom. Az életstílusa, életvitele engem mindig lenyugozott.

Köszönöm Jónás Petinek, *Pedronak*, az önzetlen segítségét, amelyre mindig számíthattam, nem csak az elmúlt években, hanem a disszertációírás "hajrájában" is. Köszönöm azokat a beszélgetéseket is, amelyeket nem felejték el.

Külön köszönet illeti Kék Lászlót, aki magára vállalta a nyelvi hibák, a téves hivatkozások kiszűrését a disszertációmban. Nagyon sok munkája fekszik abban, hogy ez a dolgozat érthetően ismerteti a CNN elméletét és a téziseimet.

Ezúton is szeretném megköszönni Tóth Péter és Vörösházi Zsolt veszprémi informatikushallgatóknak a segítségüket, figyelmességüket. Örömmel dolgoztam velük.

Köszönöm Fodroczi Zoltánnak és Kiss Attilának a közvetlen beszélgetéseket. Üde, kellemes színfoltot vittek a labor hétköznapi életébe.

Köszönöm a labor összes munkatársának a támogatását is, amivel a különböző munkáimat segítették. (Külön szeretném itt kiemelni Radványi Andrást, Süto Istvánt és Török Leventét.)

Szeretném megköszönni Kerekes Tibor segítségét is, amely nélkül a mikroelektronikai tervezőrendszer használatának útja rögösebb lett volna.

Hálás vagyok az egyetemi oktatóimnak, tanárainak, akiknek nagyon sokat köszönhetek. Külön szeretném itt kiemelni a konzulensemét, Eged Bertalant, akitől nagyon sokat tanultam. Szeretném még megemlíteni Hosszú Gábort, Ipsits Imrét, Iváncsy Szabolcsot és Mojzes Imrét is, akik szintén nagyon sokat segítettek nekem.

Hálás vagyok a szakközépiskolai tanárainak is, akik megszerettették velem az

elektronikát. Külön szeretném itt megemlíteni Szarvas Lászlót, aki nagyon sokat segített nekem az egyetemi felvételre való felkészülés során.

1. Bevezetés

A Celluláris Nemlineáris (vagy Neurális) Hálózat [16], [17] két, esetleg több dimenzióban szabályosan elhelyezkedő, egymással lokálisan összekötött, nemlineáris dinamikájú analóg cellákból, processzorokból áll. Ha az elemi cellákat, processzorokat kiegészítjük különböző típusú lokális memóriákkal, a processzortömböt egy központi vezérloegységgel és a program utasításokat tároló, globális regiszterekkel, akkor a CNN-UM-et (Celluláris Univerzális Számítógép) kapjuk [18]. Ez az első tárolt programú analogikai számítógéparchitektúra.

A CNN-UM-ek nem csak az elméletben, hanem a gyakorlatban is léteznek. A legegyszerűbb, legpontosabb, de egyben a leglassúbb megoldása egy PC-n futó szimulátorprogram [19], [20]. Jóval gyorsabb a működése a digitálisan emulált CNN-UM-nek [1], [5], amelyek megvalósíthatók VIRTEX FPGA-n [37], [38], vagy akár az ASIC (Application Specific Integrated Circuits) tervezés segítségével is (például az AMS technológiával [47]), [53], [54]. A leggyorsabb működési sebességet pedig akkor érjük el, ha a processzortömbben analóg cellákat helyezünk el. Ezt a megvalósítást analóg CNN-UM-nek is nevezzük [21], [22], [23], [24]. Ezek a megoldások a leggyorsabbak, de sajnos pontatlanok, mert az analóg VLSI gyártási technológia kevésbé kézben tartható. Ezért születtek meg az első digitálisan emulált chip-ek, amelyek segítségével "könnyen" megoldhatók akár a különböző parciális differenciálegyenletek is. A különböző CNN-UM megoldásokon futó programokat analogikai algoritmusoknak, programoknak nevezzük. Ezek a programok elemi utasításai a template-ek, amelyeknek a mérete $n \times n$. Legegyszerűbb és a leggyakoribb esetben $n=3$. A template-ek segítségével állíthatjuk be a lokálisan összekötött processzorok közötti összeköttetés "nagyságát".

A jelenlegi technológiával készült analóg CNN-UM chip-ek számítási teljesítménye néhányszor 10 Teraops (10^{12} op/s). Az analogikai processzortömbök új algoritmikus szemléletet, módszert jelentenek a tér-idobeli számítások világában.

A CNN-UM Turing értelemben univerzális, ami azt jelenti, hogy tetszőleges algoritmus megoldható ezzel a neurális processzortömbbel. Mivel a felépítése lokálisan összekötött és általában kétdimenziós, ezért a CNN-UM kétdimenziós jelfeldolgozásban alkalmazható. Az elmúlt években számos ilyen algoritmus született. Ha egy analogikai processzort megfeleltetünk egy pixelnek, akkor könnyen belátható, hogy a CNN-UM-ek elenyősen használhatók a különböző képfeldolgozási feladatoknál. Természetesen a különböző képfeldolgozási feladatokon kívül a CNN-UM-ek felhasználhatók olyan nagy számításigényű feladatoknál is, mint például a parciális differenciálegyenletek megoldása. Ezek szintén leírhatók analogikai algoritmusok formájában is.

A CNN hálózatok felépítése egyszerű, viszonylag könnyen realizálhatók a szilíciumfelületen is. Az elmélet publikálása után "néhány évvel" később már kikerültek a foundry-ból az első chip-ek. Az első CNN-UM-ek megjelenése után különböző akadályokba ütköztek az algoritmusok fejlesztői. Az analogikai algoritmusok sebessége függ a processzortömb méretétől, hiszen egy letöltendő nagyobb képet először fel kell darabolni, majd ezeket a képrészleteket külön-külön kell letölteni a CNN-re, és ezután kell futtatni az algoritmust. Ezért lényeges az, hogy a tervezők képesek legyenek minél nagyobb processzortömbök elkészítésére. Ez függ egyrészt a technológiától, hiszen a vonalvastagság csökkenésével kisebbek lesznek az elkészített array-k fizikai méretei, kevesebbek lesznek ezáltal a gyártási költségek és

nagyobb lesz a gyártási kihozatal is. A másik nem annyira köztudott probléma, amivel a tervezőknek szembe kell nézniük, az a tok maximális lábszáma. Ha a processzortömb első sorában lévő összes processzort egyszerre szeretnénk ellátni bemeneti adattal és az utolsó sorból ugyanabban az ütemben olvasnánk ki az értékeket, akkor jelentősen megnő a tömb I/O lábszáma. Jelenleg elérhető legnagyobb lábszámú tok a BGA 356, és a különböző meghajtó áramkörök buszainak felbontása rögzített (16, 32 bites). Természetesen nem szükséges egyszerre aktivizálni egy-egy sor összes processzorát, ez megtörténhet különböző időpillanatokban. Ekkor viszont az elkészítendő CNN-UM sebessége lényegesen kisebb lesz.

Az algoritmusok tervezőinek más problémákkal is szembe kell nézniük. Az egyik az, hogy az elkészült chip-ek csak egyes szomszédosságú template-eket képesek kezelni (3*3), a másik az pedig a pontatlanság. Az analogikai chip-eknek ez a legfőbb hátránya, ezért kell használni ún. hibáturo template-eket.

Feladatomban volt ezeknek a kérdéseknek a megválaszolása is. Nagyon sok feladat megoldása (pl.: textúrafelismerés) igényli a 3*3-asnál nagyobb template-ek (pl.: 5*5) használatát. Különböző méretű template-eket használhatunk template dekompozíció nélkül is [1], [2], [13], az átkonfigurálható architektúra [9] segítségével, ezáltal lecsökken az algoritmusok futási ideje. A pontosság kézből tartható a digitálisan emulált CNN-UM-ek alkalmazásával, ami csökkenti viszont az analogikai algoritmus sebességét. Viszont ez a sebesség jelentősen növelhető a pipeline módszerrel [1], [2], [6], [13]. Az ASIC tervezés egyik döntő szempontja volt az, hogy a felhasznált szilícium mérete minimális legyen. Feladatomban volt olyan eljárás megadása, aminek a segítségével a digitálisan emulált CNN-UM chip-ek szilíciumfelülete jelentősen csökkenthető [1], [2], [13]. Így nem csak a gyártási költségek lesznek kisebbek, hanem a disszipáció is, hiszen a szilíciumon realizált kisebb áramkör kevesebb FET-et tartalmaz.

Az előbb felsorolt problémák meghatározzák a CNN architektúrák fejlesztőinek, tervezőinek a kutatási irányt. Természetesen ez az én kutatási utamat is kijelölte. Ezért foglalkozom a dolgozatomban egy emulált digitális CNN-UM architektúra különböző szempontok szerinti optimalizálásával.

A nyomtatott áramkörök (PCB) gyártása viszonylag egyszerű folyamat. Azonban ahogyan csökken a vonalvastagság és a szigetelótávolság, úgy válik egyre precízebbé, bonyolultabbá a panelek elkészítése, és így megnő a hibaszázalék is. A szakirodalomban találkozhatunk olyan megoldásokkal [32], [33], amelyek segítségével a gyártási hibák kiküszöbölhetők.

A nyomtatott áramkörök gyártásakor különböző hibák léphetnek fel, amelyeknek a detektálása nem egyszerű. A kereskedelemben kapható ellenőrző rendszerek drágák, használatuk nem triviális. Feladatomban volt olyan hibakereső analogikai algoritmusok kidolgozása, amelyek segítségével valós időben belül detektálhatók a különböző gyártási és/vagy tervezési hibák.

A dolgozatomban két, egymástól teljesen független CNN alkalmazási területet fogok bemutatni.

A dolgozatomban az alábbi tagolást követi.

A második fejezetben részletesen bemutatom a CNN struktúrát és a CNN Univerzális Gépet. Ebben a fejezetben fogom áttekinteni a legújabb CNN kutatások eredményeit is.

A következő fejezetben bemutatom a nyomtatott áramkörök elkészítésének

fázisait, különböző lépéseit. Itt fogom a szakirodalomban is megtalálható különböző hibakereső algoritmusokat is ismertetni, rávilágítva néhány hibáikra is.

Az ezután következő részben ismertetem az általam megadott CNN hibakereső analogikai algoritmusaimat.

Egy emulált digitális CNN-UM architektúrát (CASTLE) fogok ismertetni az ötödik fejezetben [5]. Itt fogok kitérni a fizikai realizálásra is, hiszen ez az architektúra elkészült 0.35 μ m-es CMOS technológián. Ebben a fejezetben fogom ismertetni az optimalizálási problémákat is.

A CASTLE architektúra optimalizálási eredményeimet a hatodik fejezetben fogom bemutatni.

A hetedik fejezetben olyan emulált digitális CNN-UM aritmetikai egységeket mutatok be, amelyek az általam megadott módszerekkel készültek el.

A nyolcadik fejezetben az eredményeim összefoglalása olvasható. Az összefoglalás után található az irodalomjegyzék, amely nem csak a saját publikációmat tartalmazza, hanem áttekintést ad az általam érintett területekről a nemzetközi szakirodalom alapján.

A függelékben a különböző, általam megadott aritmetikai egységek schematic rajzai és a nyomtatott áramkörök detektálására megadott algoritmusaim "AMC" kódjai találhatóak. A függelékben mutatom be néhány szó erejéig a CASTLE logikai processzort és a működtetéséhez szükséges platformot. Ezen a rendszeren szintén futtattam a nyomtatott áramkörök gyártásközi ellenőrzésére szolgáló algoritmusaimat. Ezek az eredmények szintén megtalálhatóak a függelékben.

2. A CNN hálózat alapjai

2.1 Neurális hálózatok

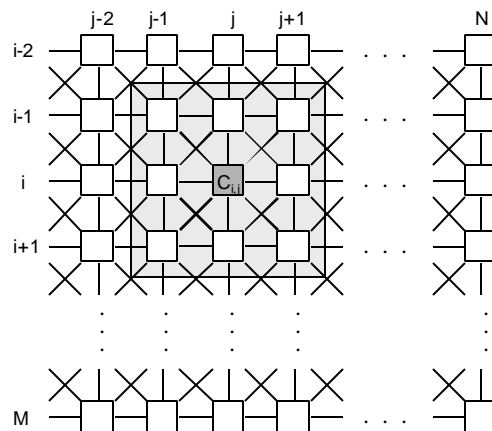
A neurális hálózatok olyan számítási problémák megoldására létrehozott áramkörök, amelyek eredete a biológiai rendszerekre vezethetők vissza. Ezek az adaptív eszközök párhuzamos feldolgozást végeznek. Az idegrendszer kutatásában elért eredmények indították arra a kutatókat, hogy megkíséreljenek a neuronok mintájára különböző áramköröket, hálózatokat létrehozni. A neurális hálózatoknak több csoportja, fajtája létezik. Ilyen például a Hopfield és a Celluláris Neurális Hálózat (Cellular Neural Network, CNN) [16], [17], [63].

A neurális áramkörök számos feladat megoldásánál nem csak alkalmasabbak, hanem jobbnak bizonyulnak, mint a hagyományos számítógép architektúrák. Ilyen feladatok például a különböző alakzatok felismerése egy képen, de a differenciál-egyenletek megoldásánál is előnyösen használhatók.

Ebben a fejezetben definiálom a CNN, CNN-UM fogalmakat és megadom a CNN formális definícióját is.

2.2 Celluláris Neurális Hálózatok elmélete

Több mint egy évtized telt el azóta, hogy a Celluláris Neurális Hálózatok elméletét publikálták [16], [17], [18]. Azóta egyre több kutatócsoport kapcsolódik be világszerte a CNN kutatásba. Ez a fajta hálózat jelentősen különbözik a többi neurális hálózattól mind a felépítésben, mind pedig a súlyok meghatározásának módjában. A CNN struktúráis felépítése a 2.1. ábrán látható. A CNN hálózat két vagy több dimenziós analóg processzortömb, ahol a processzáló elemek rácsban helyezkednek el. A rácsban elhelyezkedő processzorok azonosak, a vezérlésük azonban különbözhet.



2.1. ábra A CNN hálózat felépítése

Ha a CNN tömb processzorai két dimenzióban helyezkednek el, akkor ezt a struktúrát előnyösen alkalmazhatjuk képfeldolgozásra. Természetesen a képi

feldolgozáson kívül használhatjuk nagy számításigényű matematikai, fizikai feladatok megoldására is, ha a különböző feladatok bemeneti paramétereit 2D jelként értelmezhetjük.

A CNN definíciója:

A CNN [16] egy $N \times M$ -es, k dimenziós ($k \geq 2$) processzortömb, amelyre az alábbi feltételek teljesülnek:

- Majdnem minden pozícióban azonos processzáló elemek vannak
- Az összeköttetések lokálisak, minden processzor csak a szomszédjával van összekötve.
- Egy - egy analóg processzorhoz tartozó állapotváltozó értékben folytonos.
- A CNN processzortömb súlyokkal, úgynevezett template-ekkel programozható.

A cellák szabályos geometriai rácson helyezkednek el. Ez a rács lehet 1-, 2- vagy k -dimenziós. Egy CNN cella, processzor csak a vele szomszédos cellákkal áll összeköttetésben.

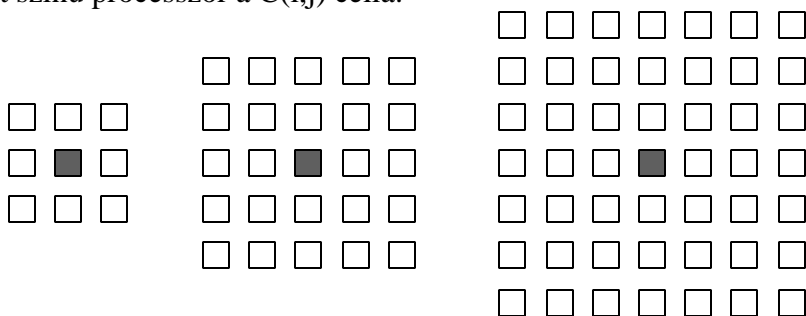
A legegyszerűbb esetben a hálózat egy $M \times N$ -es négyzetrácscsal reprezentálható. Minden cella a saját, közvetlen ($r = 1$ sugarú) környezetével van összekötve. A $C(i,j)$ cella r -sugarú környezetén az

$$N_r(i,j) = \left\{ C(k,l) \mid \max \{ |k - i|, |l - j| \} \leq r \right\} \quad (2.1)$$

cellahalmazt értjük.

Természetesen az " r " értéke nagyobb is lehet mint 1, bár még nem készült nagyobb szomszédosságú analóg CNN hálózat szilíciumfelületen. (Egyes digitálisan emulált CNN változatoknál lehetséges nagyobb szomszédosság alkalmazása is, erre később, a 6. fejezetben térünk ki. Ilyenek például a "CASTLE" digitálisan emulált CNN architektúra egyes változatai, amelyek VIRTEX FPGA-n realizáltak [42].)

Az 2.2. ábrán egyes, kettes és hármas szomszédosságú CNN látható. A középen látható sötét színű processzor a $C(i,j)$ cella.



2.2. ábra $r = 1, r = 2, r = 3$ sugarú CNN hálózatok

A mai analóg és digitálisan emulált CNN hálózatok 3*3-as súlymátrixokat, úgynevezett template-eket használnak [35]. (2.3. ábra)

$$A = \begin{bmatrix} a_{i-1,j-1} & a_{i-1,j} & a_{i-1,j+1} \\ a_{i,j-1} & a_{i,j} & a_{i,j+1} \\ a_{i+1,j-1} & a_{i+1,j} & a_{i+1,j+1} \end{bmatrix} \quad B = \begin{bmatrix} b_{i-1,j-1} & b_{i-1,j} & b_{i-1,j+1} \\ b_{i,j-1} & b_{i,j} & b_{i,j+1} \\ b_{i+1,j-1} & b_{i+1,j} & b_{i+1,j+1} \end{bmatrix} \quad z = z_{i,j}$$

2.3. ábra A 3*3-as (egyenes sugarú) template általános alakja

A CNN processzortömböt ilyen súlymátrixokkal, template-ekkel tudjuk programozni. Különböző template-ekből és bizonyos esetekben néhány logikai műveletből épül fel egy analógiai algoritmus. Az ilyen analógiai algoritmus fut a CNN Univerzális Gépen (CNN-UM) [18], amely egy későbbi alfejezetben kerül ismertetésre.

A súlymátrixoknak, template-eknek két változata létezik. Az egyik a kimeneteket visszacsatolja a súlytényezők arányában (A mátrix), a másik az előreszató template, amely a bemeneteket súlyozza a "B" mátrix szerint. A CNN processzor működése, dinamikája még egy tagtól függ, az ún. eltolási áramtól, amit a szakirodalom "z"-vel jelöl.

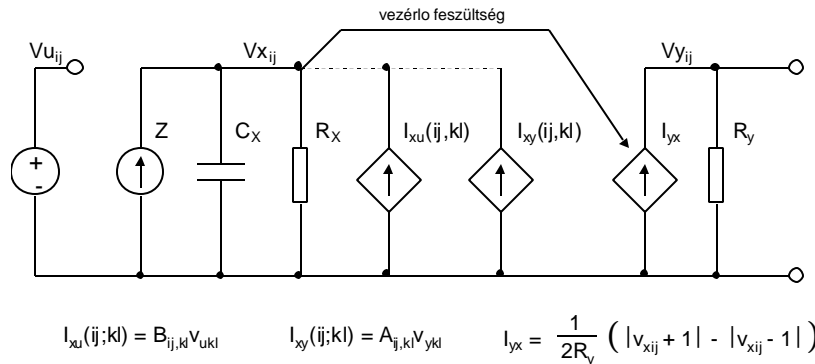
Egyes sugarú összeköttetés esetén a CNN dinamikája 19 értéktől függ (9 + 9 + 1).

A template-eket különbözőképpen osztályozhatjuk. Ha $A_{ij,kl}$, $B_{ij,kl}$ értékei nem függenek az "i", "j" értékeiktől, akkor a template térinvariáns, tehát pozíciófüggetlen. A legtöbb esetben az eltolási áramérték nem változik az "i" és a "j" függvényében ($z = z_{ij}$). A dolgozatomban olyan digitális emulált architektúrát [5] is fogok ismertetni, ahol a nem térinvariáns template-ek is használhatók.

A template értékek nem csak konstansok, hanem függvények is lehetnek. Ezeket nemlineáris template-eknek nevezzük [35].

2.2.1 A CNN processzor modellje

A CNN processzor bemeneti, kimeneti jelei és a belső állapotai folytonos feszültség és áram időfüggvények. Az 2.4. ábra mutatja egy CNN cella áramköri modelljét [18]. Ezzel az áramkörrel modellezhetjük egy cella működését.



2.4. ábra A CNN processzor áramköri modellje

A cella állapotegyenlete a következő:

$$C_x \frac{dv_{xij}(t)}{dt} = - \frac{1}{R_x} V_{xij}(t) + \sum_{C(k,l) \in N_r(i,j)} A(i,j,k,l) V_{ykl}(t) + \sum_{C(k,l) \in N_r(i,j)} B(i,j,k,l) V_{uk}(t) + Z_{ij}, \quad (2.2)$$

ahol

- V_{xij} az (i,j)-edik cella belső állapotát tükröző feszültség;
- V_{yij} a (k,l)-edik elem kimeneti feszültsége;
- V_{uij} a (k,l)-edik cella bemeneti feszültsége;
- $A(i,j,k,l)$ és a $B(i,j,k,l)$ az (i,j)-edik és a (k,l)-edik processzorok közötti súlyozási együtthatók.

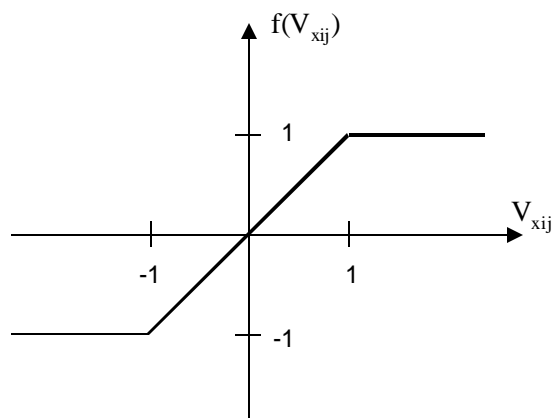
A processzorokban csak egyetlen nemlineáris elem található, amelynek a kimeneti egyenlete a következő:

$$V_{yij}(t) = f(V_{xij}(t)) = \frac{1}{2} (|V_{xij}(t) + 1| - |V_{xij}(t) - 1|) \quad (2.3)$$

Peremfeltételek:

$$\begin{aligned} |V_{xij}| &\leq 1 \\ |V_{uij}| &\leq 1 \\ A(i,j,k,l) &= A(k,l,i,j), \\ C &> 0, \\ R_x &> 0. \end{aligned} \quad (2.4)$$

A cellákban csak egy nemlineáris elem található. A kimeneti karakterisztikát, amelyet a 2.3. egyenlet ír le, a 2.5. ábra mutatja.



2.5. ábra A CNN cella kimeneti karakterisztikája

A fenti peremfeltételeknek megfelelő hálózat egy kezdeti állapotból kiindulva

stabil állapotba jut, tehát a hálózat stabil választ ad [16].

Célszerű a cellák állapotait különböző színekkel ábrázolni. A processzorok értékeit a szürke szín különböző árnyalataival fejezzük ki. A fehér megfelel a "-1"-nek, a fekete pedig a "1"-nek. Léteznek már logikai CNN processzorok is, itt csak két érték lehet. A fehér színt itt a "0"-nak feleltetjük meg (2.6 ábra).



2.6. ábra Színskála

A negyedik fejezetben fogom bemutatni a nyomtatott áramkörök gyártásközi ellenőrzésére szolgáló algoritmusaimat. A képeken a fekete szín ("1") megfelel a vezetonek, a fehér ("0") pedig a szigetelonek.

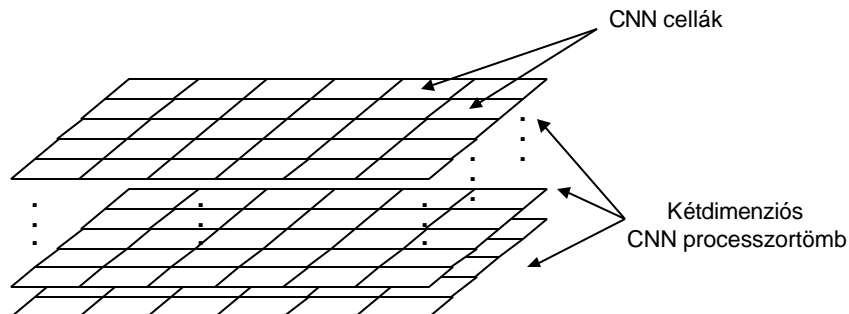
A dolgozatomban olyan analogikai algoritmusokat fogok ismertetni, amelyek térinvariáns template-eket használnak. Ilyen template-ek használatakor az elobb bemutatott CNN egyenlet (2.1.) átírható, illetve a jelölések egyszerűsödnek. Bevezethetők az $A_{j;k;l} = A_{k;l}$, $B_{ij;k;l} = B_{k;l}$, $v_u = u$, $v_x = x$, $v_y = y$ jelölések. Az egyenlet tovább egyszerűsödik, ha feltételezzük, hogy $C_x = R_x = 1$. Ezek figyelembe vételével az 2.2. CNN egyenlet a következő alakba írható.

$$\dot{x}_{ij}(t) = -x(t) + \sum_{C(k,l) \in N_r(i,j)} A_{k,l} y_{kl}(t) + \sum_{C(k,l) \in N_r(i,j)} B_{k,l} u_{kl}(t) + z_{ij} \quad (2.5)$$

ahol

- "u_{kl}" a bemeneti változó
- "x_{ij}" az állapotváltozó
- "y_{kl}" a kimeneti változó
- "A" a visszacsatoló template
- "B" az előre csatoló template
- "z_{ij}" az eltolási áram

Ha kétdimenziós CNN hálózatokat egymás felé helyezünk és összekapcsolunk, akkor többrétegu hálózatot kapunk (2.7. ábra).



2.7. ábra Többrétegu CNN hálózat

Ennek a hálózatnak az "m." rétegének (i,j)-ik cellájának a dinamikáját az alábbi

egyenlet (2.6.) írja le:

$$C_{xm} \frac{dv_{xmij}(t)}{dt} = - \frac{1}{R_{xm}} v_{xmij}(t) + \sum_{n=1}^P \left(\sum_{C(k,l) \in N(i,j)} A(m,n,i,j,k,l) v_{ykl}(t) + \sum_{C(k,l) \in N(i,j)} B(m,n,i,j,k,l) v_{ukl}(t) \right) + Z \quad (2.6)$$

ahol:

- "p" a rétegek száma
- "m"-mel jelölik a szakirodalomban az aktuális réteget
- "A_{mn}", "B_{mn}" az m. réteg állapotának n. réteg kimeneti és bemeneti értékeitől való függését fejezi ki.

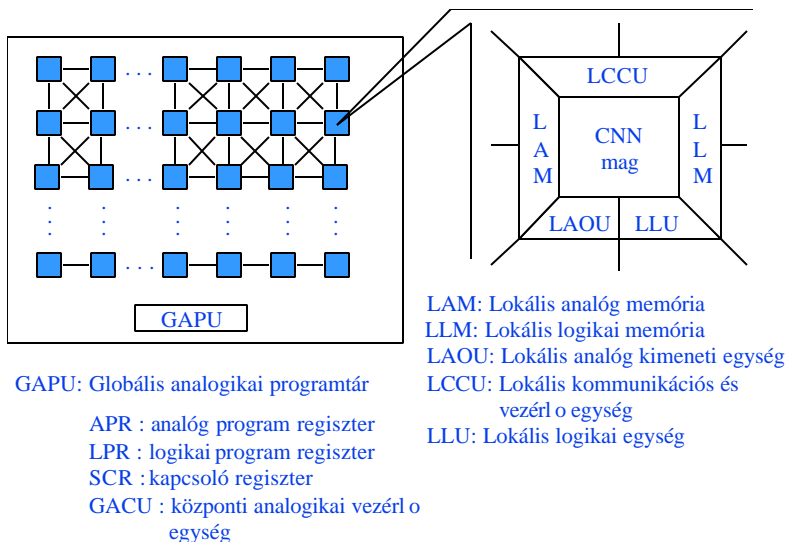
Ilyen többrétegu hálózat lehet például a CNN alapú retinamodell [58].

2.3 A CNN-UM definíciója

Noha a CNN megvalósítása VLSI technikával nagy számítástechnikai teljesítményt tesz lehetővé, a CNN csak akkor használható széles körben, ha megoldott az algoritmikus programozhatóság. Ez a programozhatóság rugalmasságot ad a celluláris hálózatoknak. A CNN-UM (CNN univerzális gép [18]) egy analóg tárolt alapú számítógép, amely a CNN processzortömbre épül, amit a 2.2 fejezetben mutattam be. A CNN-UM egy analóg számítógép, amely egy független operációs rendszerrel, programozási nyelvvel és lokális analóg és logikai memóriával rendelkezik. Több fizikai implementációja létezik már, pl.: [19], [20].

A CNN-UM a duális számítás elméletén alapul. Ez az elmélet az analóg és logikai műveleteket párosítja a lokális analóg memóriákkal és a programozhatósággal. Ezt a szakirodalom analogikai számításnak nevezi. Tehát az analóg értékeket nem kell átalakítani digitálissá, mert minden jel analóg vagy digitális.

A CNN-UM felépítését mutatja a 2.8. ábra. Az univerzális gép két, egymástól jól elkülöníthető részből áll. Az univerzális gép tartalmaz egy "GAPU" központi vezérlőt és egy CNN processzortömböt.

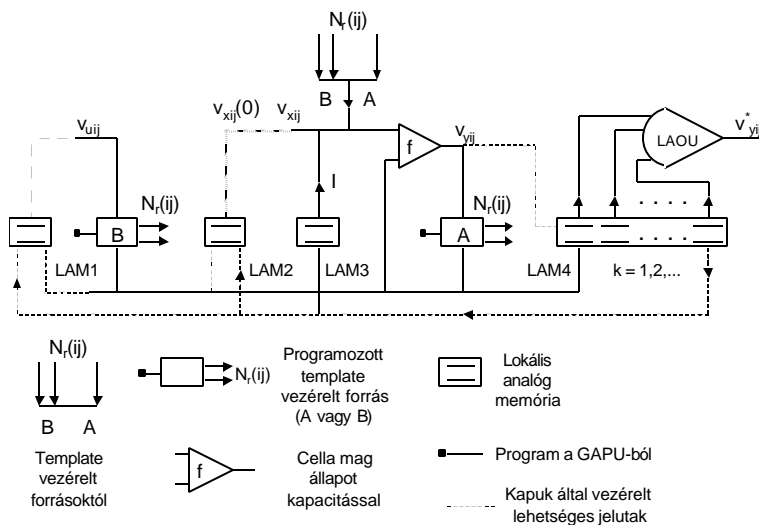


2.8. ábra A CNN-UM felépítése

A CNN-UM celláit egy központi egység, a GAPU (Global Analogic Programming Unit) vezérli. Ez az egység analóg és logikai egységekből áll. Ez a központi vezérlőegység tartalmazza az utasításregisztereket. Itt kapott helyet az "APR" analóg program regiszter, ahol tároljuk a template-eket. Megtalálható itt még az "LPR" (logikai programregiszter) is, ahol a logikai utasítások tárolódnak. Az "SCR" kapcsolóregiszter a cellák konfigurációs állapotát meghatározó adatokat tárolja. A "GACU" az analogikai algoritmus egymást követő utasításait kódolja.

A CNN processzorokhoz, cellákhoz tartozik egy lokális analóg memóriaegység (LAM), lokális logikai memória (LLM) és a lokális kommunikációért felelős LCCU egység. A lokális memóriaegységek biztosítják azt az eredményt, amit az adott template-et futtatva kapunk. Az itt tárolt eredményből indíthatjuk az újabb iterációt.

Az 2.4. ábrán már bemutattam a CNN cella áramköri modelljét. Az 2.9. ábra mutatja egy analogikai cella blokkvázlatát [23].



2.9. ábra CNN cella analóg részének modellje

Az ábrán különböző kapcsolók láthatók, amelyek segítségével eldönthetjük, hogy a cella milyen funkciókat végezzen el. A konfigurációs regiszterben ezeknek a kapcsolóknak az állapotait tárolják.

Ha a CNN univerzális számítógép architektúrát implementáljuk, akkor kapjuk az analogikai mikroprocesszort. Mostanában és az elmúlt években több ilyen mikroprocesszort készítettek. Ilyen például a 32*32-es [34], 64*64-es [22], a 128*128-as [24] a 176*144 [25] és a most elkészült emulált digitális CNN-UM chip, amelyet a készítő CASTLE-nek [5] hívnak.

Turing-Church tézis

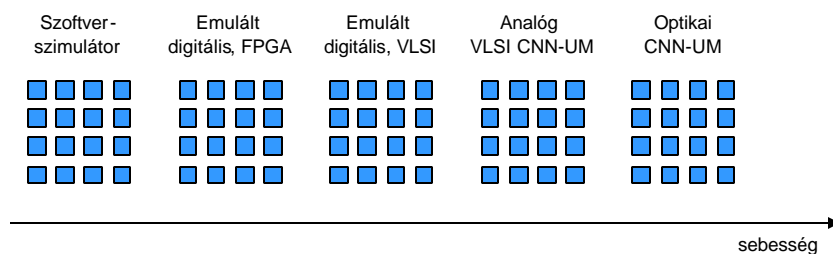
Minden μ -rekurzív függvény Turing kiszámítható. A Turing-gép, a nyelvtan és a μ -rekurzív függvény egymásba ekvivalensen átalakíthatók.

Nem találtak még az egész számokon olyan algoritmust, amely a μ -rekurzív függvényekkel ne lenne eloállítható [59].

A CNN-UM univerzális gép abban az értelemben, hogy rajta minden μ -rekurzív függvény megvalósítható [64].

2.4 CNN-UM implementációk

A CNN architektúra alkalmas különböző, nagyteljesítményt igénylő számítások elvégzésére, ezért több CNN-UM megvalósítási módszer is megtalálható a szakirodalomban. Létezik analóg VLSI, emulált digitális (FPGA vagy VLSI), szoftver szimulátor, vagy optikai megvalósítás. A működési sebességük különböző, ezt mutatja a 2.10. ábra [9].



2.10. ábra A különböző CNN megvalósítások összehasonlítása sebesség alapján

2.4.1 Szoftveres implementáció

Ez a megvalósítás a legpontosabb, lelassabb, de a legrugalmasabb. Ezt az implementációt kiválóan alkalmazhatjuk algoritmusok fejlesztésére, illetve a template-ek tervezésére, optimalizálására.

2.4.2 Emulált digitális megvalósítás

A CNN kutatásnak ez az egyik új területe. Ez a megoldás gyorsabb, mint a szoftveres megvalósítás, de az analógnál lassabb, viszont a pontossága lényegesen jobb. Sokoldalúbbak, rugalmasabbak, könnyebben elkészíthetők. A különböző számításokat egy speciális hardver, a céláramkörök végzik el [1], [5], [42]. Az elkészített digitális CNN-UM-ek könnyebben integrálhatók egy adott áramkörbe.

Több emulált digitális CNN-UM megoldását publikálták, az egyik a CASTLE architektúra [5]. Ennek az első képviselője a logikai CASTLE processzor, amely csak bináris képeket tud kezelni. A CASTLE architektúráról később még részletesen fogok szólni.

2.4.3 Analóg CNN-UM

A CNN-UM megvalósítások közül ez a megoldás a leggyorsabb [21-25], de sajnos a legpontatlanabb is. A hatalmas mikroelektronikai fejlődésnek köszönhetően ma már a gyártók képesek $\sim 1\text{cm}^2$ felületre $128*128$ darab analóg CNN cellát is elhelyezni [24].

Az analóg implementációknak a hátránya a pontatlanság, a hőmérsékletfüggés és a zavarérzékenység. Ezek csak a gray-scale feldolgozására képes CNN-UM-ekre mondható el, a bináris megoldásokra [21], [25] nem.

2.4.4 Optikai rendszerek

Az optikai megoldások [65], [66] nagy előnye a sebességük és az, hogy nagy felbontású képek feldolgozhatók nagy méretű template-ekkel.

Az optikai CNN-UM sebességét csökkenti az a tény, hogy a visszacsatoló template-ek hatását csak elektromosan erősített fényvel lehet megoldani. Ezeknek a megoldásoknak további hátrányuk a mechanikai kivitelezésük, hiszen nagyobbak és sérülékenyebbek, mint a chip-es vagy szimulátoros megvalósítások.

2.5 A CNN-UM implementációk összehasonlítása

A szakirodalomban különböző CNN-UM összehasonlítások találhatók [5]. Az egyik legelterjedtebb változat az, amikor egy képen (mérete: $128*128$ pixel) különböző műveleteket végzünk, és ezeknek az idejét adjuk meg μs -ban (2.1. táblázat)

	Pentium IV [39]	IBM 10 TeraOps [41]	TMS 320C6X [40]	VIRTEX XCV300 [38], [42]	Original CASTLE [5]	64*64 CNN-UM [22]	128*128 CNN-UM [24]
Implementáció	szoftver	emulált	emulált	emulált	emulált	analóg	analóg
Frekvencia	2GHz	700MHz	1.2GHz	200MHz	100MHz	1/10MHz	32MHz
vonalszélesség	0.13 μm	0.18 μm	0.12 μm	0.22 μm	0.35 μm	0.5 μm	0.35 μm
terület [cm ²]	1.27	6.9468 m ²	1.1	1.2	0.07152	1	1.45
A fizikai processzorok száma	1	65536	1	< 12 (2bits)	3*4	4096	16384
Kaskádosisítható?	nem	nem	nem	igen	igen	igen	igen
Disszipáció	50 W	491.520 kW	1 W	1.8 W	< 0.3 W	1.3 W	< 4 W
3*3 konvolúció	140	3.18	16.384	41 (6 bit)	5.34 (12 bit) 2.67 (6 bit)	10.6	1.749
Erózió/Dilatáció	270	3.18	32.768	82 (6bit)	10.69 (12 bit) 5.34 (6 bit)	10.6	1.749
Laplace (15 iteráció)	2000	3.45	245.7	615	79.2 (12 bit) 39.6 (6 bit)	11.5	1.8975
Algoritmus A: (10 konv. + 10 eros. + 1 Lapl.)	3970	7.05	737.22	1845	239.5 (12 bit) 119.4 (6 bit)	23.5	3.8775
Algoritmus B: (10 konv. + 10 eros. + 10 Lapl. + 10 logic.)	21980	11.1	2948.52	7380	958.9 (12 bit) 476.1 (6 bit)	37	6.105

2.1. táblázat A "mai" CNN-UM implementációk összehasonlítása

A ma leggyakrabban használt CNN-UM chip a 64*64-es [22] és az elmúlt évben jelent meg a 128*128-as CNN-UM [24]. Ezért fontosnak tartom külön is bemutatni ezeket az analóg chip-eket (2.2. táblázat).

	64*64	128*128
Technológia	ST-0.5 μm 3M-1P	ST-0.35 μm 5M-1P
Tervezési stílus	Full Custom	Full Custom
Tokozás	Kerámia QFP144	Kerámia QFP144
Cellák száma	4096 (64*64)	16384 (128*128)
Tranzisztorok száma	1.000.000	3.748.170
Tranzisztorok száma cellánként	172	198
Chip mérete	9.145*9.534 mm ²	11.885*12.23 mm ²
Cellák eloszlása	~ 82 cella/mm ²	~ 180 cella/mm ²
Idokonstans	~ 1.2 μs	~ 0.8 μs
LAM	4	8
LLM	4	0
I/O Digitális rate	1 MHz(analóg), 10 MHz(bináris)	32 MHz
Tápfeszültség	3.3 V	3.3 V
Disszipáció cellánként	~ 180 μW	~ 180 μW
Disszipáció	< 1.5 W	< 4W

2.2. táblázat A 128*128-as és a 64*64-es CNN-UM chip adatai

A két táblázatból könnyen észrevehető, hogy a cellaszám növelésével megnövekszik a számítási teljesítmény. Ez természetesen maga után vonja a disszipáció növekedését

is, de ez elhanyagolható lehet, ha a Celluláris Neurális Hálózatot szilíciumfelületen készítjük el.

Látható tehát az, hogy a elektronikának ez a területe is hatalmasat fejlődött az elmúlt tíz év alatt. Ennek a hálózati struktúrának létjogosultsága van olyan feladatok megoldásánál, amelyek eredete a biológiai rendszerekre vezethetők vissza. A bemenet(ek) és a kimenet(ek) megfeleltethetők képeknek (2D bemenet/kimenet), hiszen a képnek egy pixele egy processzort jelent a CNN technikában.

De elonyösen használhatók a neurális áramkörök a képeken a különböző alakzatok felismerésénél vagy a differenciális egyenletek megoldásánál is.

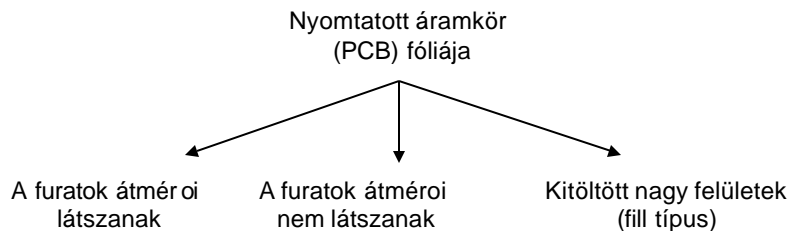
3. A nyomtatott áramkörök gyártása, ellenorzése

A nyomtatott áramkörök használata elengedhetetlen a modern elektronikában. Ezek gyártása során a gyártásközi ellenorzés nagyon fontos. Az ellenorzés során a hibás paneleket kiválasztjuk és a hibától függoen javíthatjuk. A technika fejlődésével a geometriai méretek csökkentek, a gyártás precízebb lett, bonyolultabb nyomtatott áramkörök jelentek meg. Az ilyen nyomtatott áramkörök az ún. AOI (Automatikus Optikai Ellenorzés) rendszerekkel hatékonyan ellenorizhetok [32], [33]. Ilyen rendszerek alkalmazhatók az iparban, hiszen nagy sebessége miatt a gyártás során az ellenorzés folyamatos és kizárt az emberi tévedés is.

Ebben a fejezetben bemutatom a nyomtatott áramköröket, (PCB, Printed Circuit Board), az eloforduló hibákat és ezen hibák detektálási módjait.

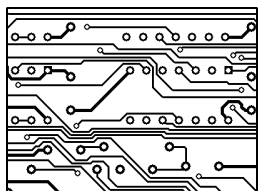
3.1 A nyomtatott áramkörök felépítése, osztályozása

A nyomtatott áramköröket többféleképpen osztályozhatjuk. Az egyik megoldást a 3.1. ábrán láthatjuk. A gyártófilmen (aminek a rajzolatát rávilágítják fotokémiai úton a gyártás során az előre kifúrt, rézfóliával ellátott szigetelőlemezre) a forrponok furatai vagy látszanak, vagy nem. A rádiófrekvenciás és a nagysebességu áramköröknél a gyártandó panel szigetelő részeit kitöltik vezetovel, rézzel. Ezt "fill" típusúnak is nevezi a szakirodalom.

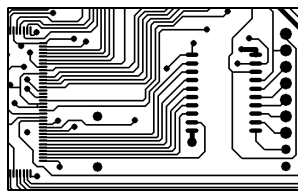


3.1. ábra A nyomtatott áramkörök csoportosítása gyártófilm típusai szerint

Ezekre láthatunk egy-egy példát a 3.2., 3.3. és a 3.4. ábrákon.



3.2. ábra
A furatok
átmérői látszanak

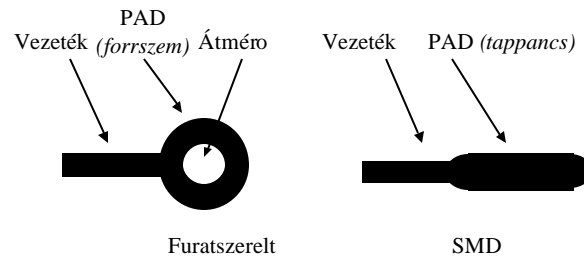


3.3. ábra
A furatok átmé-
roi nem látszanak



3.4. ábra
"Fill" típusú panel

A szereletlen nyomtatott áramköri lapon a következő elemeket használják (3.5. ábra). Két megoldást láthatunk. Megfigyelhető a gyakorlatban az SMD technika elterjedése az elonyös tulajdonságai miatt (pl.: minimálisak a kivezetések, kisebbek a parazitahatások). Ugyanakkor a furatszerelt alkatrészeknek is van jövőjük, jelenlétükkel később is számolnunk kell (pl.: csatlakozó sorok, tápkivezetések).



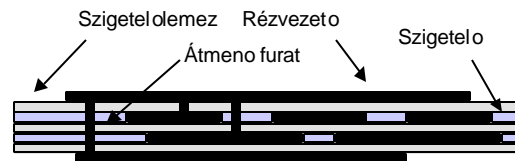
3.5. ábra Különbözo forrszemek (furatszerelt, SMD)

Természetesen csak SMD alkatrészeket tartalmazó áramkörök is analizálhatók a két hibakereso eljárás segítségével, amelyeket a következő fejezetben mutatok be, hiszen egy többoldalas nyomtatott áramkörnél furatgalvanizált átmenetek (via) is vannak.

Egy kétoldalas nyomtatott áramkört mutat a 3.6. ábra, a 3.7. ábrán pedig egy több rétegu látható.



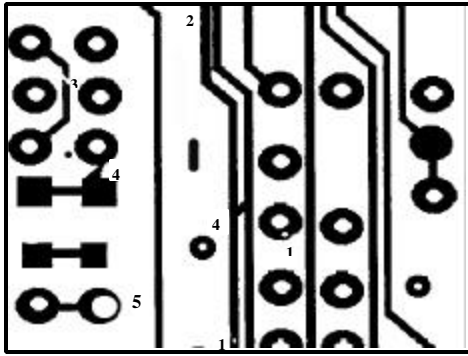
3.6. ábra
Egy kétoldalas panel
felülnézeti képe



3.7. ábra
Egy négyoldalas nyomtatott áramkör
metszete

3.2 A nyomtatott áramkörön található hibák

A 3.8. ábrán és a 3.1. táblázatban néhány, gyakran előforduló layout hibafajtát láthatunk. A disszertációom következő fejezeteiben két hibatípus (a 3.1. táblázatban 4. és 5.-ként jelölt hibatípusok) detektálását fogom bemutatni. A zárlatok előfordulása veszélyes a nyomtatott áramköri lemezen lévo alkatrészekre illetve az áramkör környezetére, ezért ezeknek a kiszurése kulcskérdés.

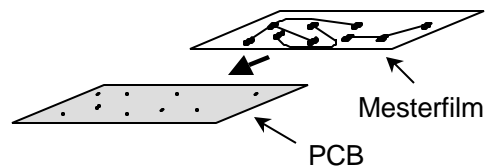


3.8. ábra
Egy hibás panel rajzolata [33]

1	Tuhelyek, elvékonyodott vezetok, vonalak
2	A vezetok között túl vékony a szigetelo
3	Szakadt vezetó
4	Zárlat van a vezetok között
5	Illesztési hiba

3.1. táblázat
A hibák leírása

Az illesztési hibák a rossz gyártófilmből adódnak. A 3.8. ábrán az ötös számmal jelöltem ezt a fajtát. Ennek a hibának a keletkezése a 3.9. ábrán látható, amikor is a mesterfilm rosszul illeszkedik az előre kifűrt nyomtatott áramkör paneljára. Ugyanakkor ez a hiba nem csak a furatpontok elcsúszásáért "felelos", hanem ez okozhatja akár a kettes hibafajtát (túl vékony szigetelo) is. Az illesztési hiba úgy keletkezik, hogy az előregedett, megnyúlt, rossz gyártófilmet illesztik az előre kifűrt szigetelolemezre. Errol a következő fejezetben részletesebben is lesz szó.



3.9. ábra A mesterfilm illesztése a kifűrt panelre

A Hite-Lap cég által rendelkezésre bocsájított hibakeresési adatokat [57] mutatja a 3.2. táblázat.

Ez egy példa, amely előfordult egy nyomtatott áramkörök gyártásával foglalkozó cégnél.

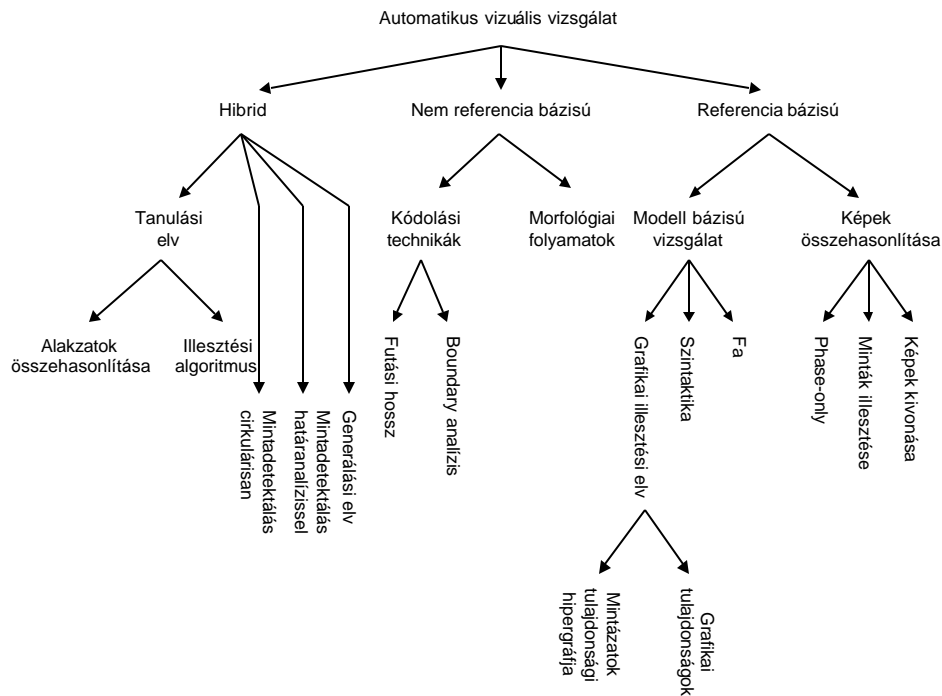
Kód	Méret (mm)	Réteg	Vonalszélesség (mm)	Panel	Tipikus hiba
E/1-1	75x124	1	≤ 0.3	w: 2mm FR4	részdarabok a panelon
E/1-2	75x124	1	≤ 0.3	w: 2mm FR4	szakadás
K/1-1	270x208	2	≥ 0.5 ≤ 0.3	w: 2mm FR4	hibás fémbevonat, szakadás
K/2-1	347x195	2	≥ 0.5 ≤ 0.3	w: 2mm FR4	hibás fémbevonat a furatnál
K/3-1	266x354	2	≤ 0.3	w: 2mm FR4	Min. vonalszélesség megsértése
K/3-2	266x354	2	≤ 0.3	w: 2mm FR4	szakadás
T/1-1	380x242	4	≥ 0.5 ≤ 0.3	w: 2mm FR4	szakadás
T/1-2	380x242	4	≥ 0.5 ≤ 0.3	w: 2mm FR4	szakadás

3.2. táblázat Különböző vizsgált paneleknél előforduló gyártási hibák [57]

3.3 Különböző optikai hibakereső eljárások bemutatása

Több kutatócsoport is foglalkozik a nyomtatott áramkörök gyártásakor felmerülő hibák detektálásával, kiküszöbölésével.

A különböző gyártásközi hibakereső eljárásokat az alábbi osztályokba sorolhatjuk (3.10. ábra) [33].



3.10. ábra A különböző hibadetektáló eljárások osztályozása

- Referencia bázisú összehasonlítás (A vizsgálandó nyomtatott áramköri lapot egy hibátlan panellel hasonlítjuk össze.)
- Nem referencia bázisú összehasonlítás (Ennél az eljárásnál a különböző technológiai szabályok megtartását a vizsgált panelen ellenőrzik.)
- Hibrid (Az előző két eljárás kombinációja)

A legegyszerűbb hibakeresési eljárásoknak az ún. "referencia bázisú" eljárások számítanak.

A mesterfilmeken és a gyártás során a következő hibák fordulhatnak elő:

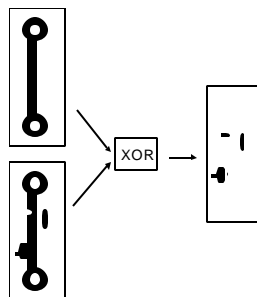
- szakadás a vezetón
- zárlatok keletkezése
- kisebb a vezető szélessége egy adott minimumnál
- a két vezető között a szigetelő kisebb egy adott minimumnál
- szakadás a forrponton
- tühelyek a vezetón
- forrpont és a hozzátartozó furat nem illeszkedik pontosan

3.3.1 Referencia bázisú módszerek

Ezt az eljárást két részre osztja a szakirodalom [32], [33]. A "Modell bázisú vizsgálat"-nál a vizsgálandó nyomtatott áramköri lapot különböző modellekkel hasonlítják össze. A "Képek összehasonlítása" esetén a vizsgálandó és a referencia képen található ponthalmazokat hasonlítjuk össze lépésről lépésre.

Képek összehasonlítása

A "Képek kivonása" a legegyszerűbb módszer. A referencia képből a vizsgálandó képet kivonjuk. Ezt legegyszerűbben a logikai XOR művelettel valósíthatjuk meg (3.11. ábra) [33].



3.11. ábra XOR művelet [33]

Sajnos ez a módszer érzékeny a megvilágításra és a fényvisszaverődésre, viszont nagy előnye, hogy könnyen implementálható.

A következő módszer a "*Minták összeillesztése*". Itt a vizsgálandó nyomtatott áramköri lap jellemzőit, tulajdonságait meghatározzák. Ezeket utána a referencia panel tulajdonságaival összehasonlítják. Nagy a számításigénye ennek az eljárásnak, de nem érzékeny a bemeneti jellemzőkre (megvilágításra, fényvisszaverődésre), tehát a bemeneti jellemzőkre nem érzékeny. Ez az eljárás nagy adattömörítést tesz lehetővé, viszont a nagy adatmennyiség tárolása nehézkes.

A referencia bázisú képek összehasonlításának harmadik technikája a "*Csak fázis*" módszer. A vizsgálandó panel és a referencia panel képét hasonlítják itt is össze, de csak fázis transzformációt hajtanak végre. Az eredményt utána normalizálják. Nagy a számítási teljesítményigény, de ez a módszer sem érzékeny a megvilágításra. Nagy előnye még ennek az eljárásnak az, hogy a képek pontos elhelyezkedésére nem érzékeny.

Modell bázisú vizsgálat

A "*Szintaktikai vizsgálatnál*" a nyomtatott lapon lévő rajzolatot betűknek és különböző szimbólumoknak, karaktereknek feleltetik meg. Itt megkeresik az alakzatok határait is, amelyekből egy listát készítenek. A hibák detektálását a reguláris kifejezésekben lévő irregularitások megkeresésére vezetik vissza. Nagyon fontos, hogy a kritikus, egyszerű layout-okat megfelelően válasszák ki és írják le [33].

A "*Grafikai illesztési elv*" [32] használatánál a vezető és a szigetelő területekből gráfokat állítanak elő. Ezekhez elemi layout modelleket párosítanak és ezekhez rendelhetők a gráf csomópontjai. Csak azokat a csomópontokat köti össze él, amelyek azonos alakzathoz tartoznak. Ezután a vizsgált layoutból nyert gráfot összehasonlítják a referencia kép gráfiájával. Ennek összehasonlítása időigényes. Ez csökkenthető a hipergráf módszer alkalmazásával.

3.3.2 Nem referencia bázisú módszerek

Ezeknek a módszereknek a nagy előnye, hogy nincs szükség referenciaképre [32], [33]. Ezáltal a beviteli és a tárolási problémák is megszűnnek. Akkor tekintenek hibásnak egy vizsgált nyomtatott áramköri lapot, ha az nem felel meg a tervezési, gyártási szabályoknak. Ezért ezekkel a módszerekkel csak minimális vezetősélességet, vezetők közötti szigetelési távolságot, rossz rézfoltokat, hibás forrponokat lehet detektálni. Ezeknek a tervezési szabályoknak az alkalmazása viszont nagyon időigényes. Ezért az ellenőrzendő képet transzformálják, ezáltal csökkentik a módszer futási idejét.

- minimális és maximális vonalszélesség az összes vezetőre
- minimális és maximális kör alakú forrszemátmérok
- minimális és maximális furatátmérok
- minimális szigetelési távolság

- minimális vonalvégződés

További hátránya ezeknek a módszereknek, hogy nem képesek az összes hibát megtalálni. Egységesíteni kell továbbá a vezetok szélességét és fajtaikat is. Nagy elonyük viszont a sebességük és a minimális tárolási kapacitás.

Morfológia

Leginkább ezt a vizsgálatot használják. Kiküszöbölik az illesztés problémáját. Hátránya viszont, hogy különböző hibák felderítésére különböző előfeldolgozásra van szükség. Ez megnöveli a rendszer futási idejét.

A morfológiai alapú hibakereso eljárásokat párhuzamos számítási architektúrákon érdemes futtatni, mert az egy processzoros rendszereken lassúak.

Kódolási technikák

A layout ábrázolások speciális kódolásokon alapulnak. Ezt két módszer is felhasználja.

A "*Boundary analízis*" használatánál fontos kérdés a határok könnyu kezelhetosége. Az alakzatok határainak a leírására lánckódolást használnak. Eloszor meghatározzuk az alakzat határán lévo két pont közötti euklideszi és a határon lévo távolságokat. A hibakeresés arra épül, hogy ha hibátlan a határ, akkor ez a különbség kicsi, hibásnál pedig ez az érték nagy lesz.

Ezután kiveszik a sarokpontokat, és ezeknek a jellemzoibol következtetnek egy esetleges hibára.

A "*Futási hossz*" kódolásnál a különböző vezetok függoleges és vízszintes rajzolatait analizálja. Meghatározzák az összes sorban és oszlopban lévo egybefüggó pixelek számát és létrehoznak egy hisztogramot. A különböző szélességek láthatók lesznek a hisztogramban. Ez felhasználható a hibák detektálásához. Nem szükséges itt a pontos illesztés, viszont előfordulhat, hogy nem valódi hibát találunk.

3.3.3 Hibrid módszerek

Ezeknek a technikáknak az a nagy elonye, hogy az elobb ismertetett módszereknek a pozitív tulajdonságait ötvözik. Ebbol következik, hogy nagyon sok hibafajta detektálható ezekkel a módszerekkel.

Generálási elv

Akkor kapjuk ezt a módszert, ha kombináljuk a referencia és a nem referencia összehasonlítást. A keresett mintákat hasonlítjuk össze az előre kiválasztott minta típusokkal. Eloszor a vizsgálandó nyomtatott áramköri lapból előállítunk egy olyan layout-ot, ahol minden geometriai alakzat szélessége egy pixel. Ekkor a hibák könnyebben detektálhatók. Ezután összehasonlítjuk az előre meghatározott mintákkal a kapott rajzolatot. Ezzel a módszerrel a tulykakakat, forrpontokat, minimális vonal és szigetelési távolságot is ellenőrizhetünk.

Mintadetektálás határanalízissel

A geometriai alakzatokra határanalízist végzünk. Ekkor viszonylag könnyen meghatározhatjuk a lehetséges hibás helyeket. Ezután ezeken a lehetséges hibahelyeken mintadetektálást végzünk, megmérjük a vezeték szélességét. Ez a hibrid módszer tehát a határanalízisen és a minták detektálásán alapul.

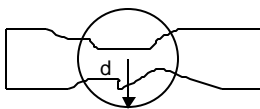
A módszer sebessége jelentősen megnövekedik, hiszen nem az egész nyomtatott áramkört ellenőrizzük, hanem csak a lehetséges hibahelyeket.

Mintadetektálás cirkulárisan

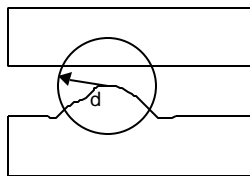
Ennek a módszernek az a lényege, hogy a vizsgálandó panelt és a mintát egy komparátor segítségével összehasonlítjuk a radiális kódolás után. Ez hardveresen történik. Egy 32*32-es ablakot húzunk végig a vizsgálandó nyomtatott áramköri lapon. A hibakereséshez szükséges minták függenek a felhasznált technológiától.

A 3.12. ábrán különböző példákat láthatunk. Ha a kör két terület között van (nem metszi egyik vezeték sem), akkor azon a részen a nyomtatott áramkör jó. Egyébként, ha a vezeték szélessége kisebb, mint a "d", akkor a vezeték (nyomtatott áramkörön) hiba van (3.12 a. ábra). Más hibafajtákat is kiszűrhetünk ezzel az eljárással (3.12 b-e).

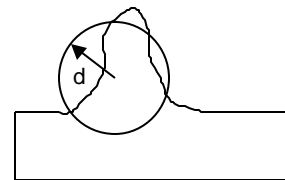
Ez az eljárás nem érzékeny a jelvezetékek irányára, viszont a 3.12 f. ábrán látható hibát nem képes kimutatni.



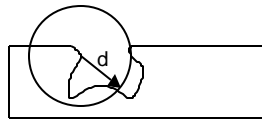
3.12 a. ábra



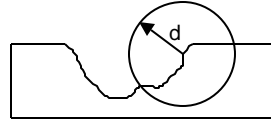
3.12 b. ábra



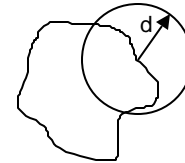
3.12 c. ábra



3.12 d. ábra



3.12 e. ábra



3.12 f. ábra

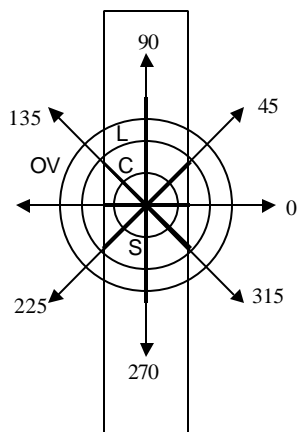
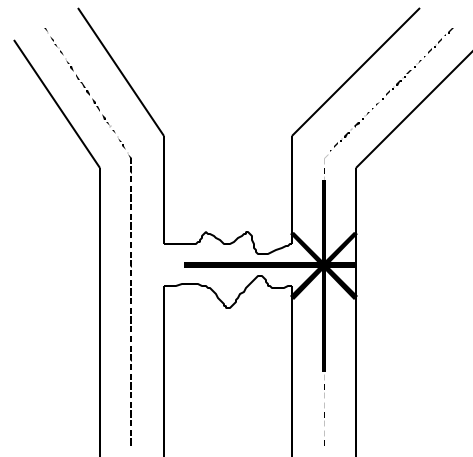
Különböző hibák keresése (a-f)

A tanulási elv

Ennél a két módszernél (3.10. ábra) tanítást alkalmazunk. Mintákat veszünk egy jónak tartott nyomtatott áramköri lapból. Ezután például "körkódokká" alakíthatjuk a mintákat. A kódokból kiolvashatjuk ezután a hibákat.

Ezzel a módszerrel megtalálhatjuk a szakadásokat, zárlatokat és a különböző elvékonyodásokat.

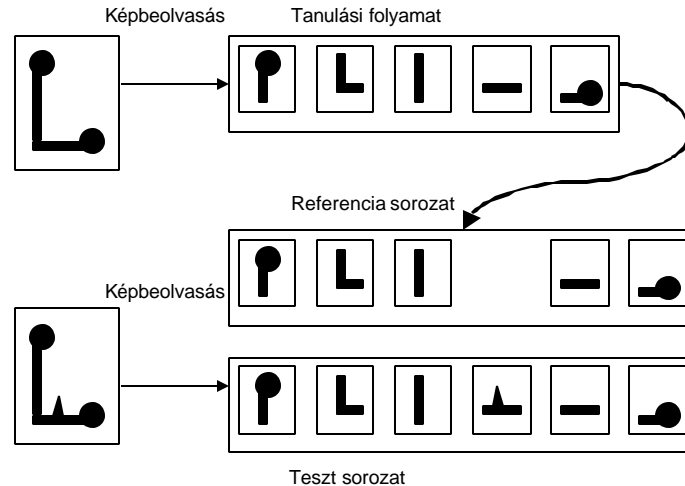
Ilyen rendszer például az FVIS-110 rendszer, amelyet a Fujitsu készített [33]. Itt szenzorpárokkal vizsgáljuk a gyártás során a nyomtatott áramköröket. Ez a megoldás a következő kódokat használhatja az ellenőrzés során: S (shorter), C (correct), L (longer), OV (over) (3.13 a. ábra). Ha a vizsgált jelvezeték jó, akkor ennek a szélessége belül van a "C" területen. A 3.13 a. ábrán látható példán mutatom be a kódmegállapítást. A 0° -on a mért távolság "C", a 45° -on a távolság "L", 90° -nál "OV" és 135° -nál ismét "L". Tehát a körkód a következő: C, L, OV, L.

3.13 a. ábra *Az elv*3.13 b. ábra *Rövidzár detektálás*

A következő példán (3.13 b. ábra) láthatjuk a zárlatdetektálás folyamatát ezzel a módszerrel. A helyes kód C, L, OV, L. De a zárlat miatt a kód megváltozik, mert 0° -nál "OV"-t kapunk. Tehát az új kód: OV, L, OV, L.

A 3.14. ábra mutat egy példát a tanulási elv használatára. Ezt a megoldást (Ai-1029) a Nikon fejlesztette ki [33]. Az első lépésben megtanítjuk a rendszernek az etalon nyomtatott áramkör szegmenseit. Ez a tanulási folyamat. Ezután eltávolítjuk a

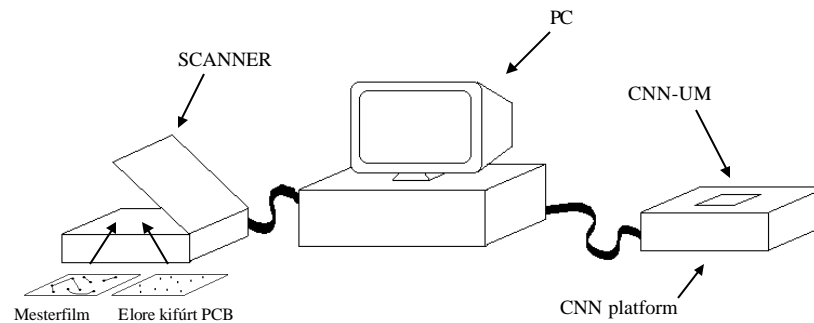
szegmenseket a referencia file-ban (referencia sorozat). Ezeket a lépéseket ismétljük a vizsgálendő nyomtatott áramköri lapokon is, és a referencia sorozatot összehasonlítjuk a teszt sorozattal. Az ábrán látható, hogy a teszt sorozatban van egy kép, amely nem szerepel a referencia sorozatban. Ez a hiba, amelyet az Ai-1029 rendszer detektált.



3.14. ábra Ai-1029 rendszer (Nikon) működési elve

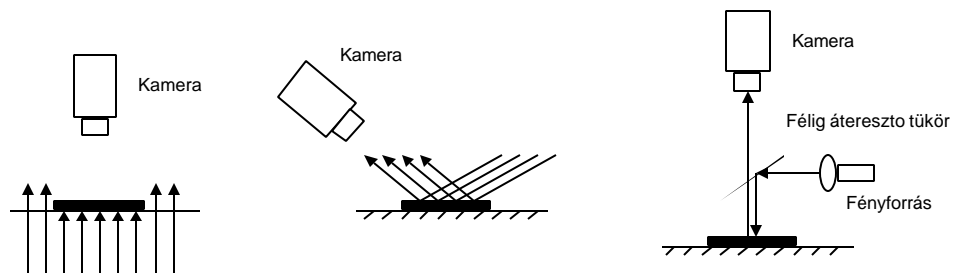
3.4 A hibadetektáló algoritmusok használata

A különböző teszteredményeket a 3.15. ábrán feltüntetett elrendezéssel állítottam elő. Ez azonban az iparban nem használható, és a CNN-UM sebessége sem aknázható ki teljes mértékben.



3.15. ábra A kísérleti környezet

Az iparban a következő képbeviteli eljárások egyikét használják (3.16. ábra), mert ezek könnyen alkalmazhatók a gyártósoroknál.



3.16. ábra *Az iparban használatos képbeviteli eljárások*

A következő fejezetben két hibakereso, detektáló analogikai algoritmust fogok bemutatni.

4. Analogikai algoritmusok a nyomtatott áramkörök gyártási, tervezési hibáinak detektálására

Ebben a fejezetben mutatom be az eredményeim közül azokat [3], amelyek a különböző geometriai alakzatok és ezek közötti térbeli viszonyok ellenőrzésére szolgálnak. Ezek az eredmények nyomtatott áramkörök gyártása során felhasználhatók.

A nyomtatott áramköri lapok (Printed Circuit Board, PCB) előállítása igen komplex feladat, folyamatosan csökken a vezető és a szigetelőtávolság (min.: ~ 6mil, ~ 0.1524mm), kisebbek lesznek a furatok, forrponatok méretei is. Továbbá a layout alakzatok méretcsökkenésének következtében a layout elemek száma egy adott területen folyamatosan nő. Ezek miatt a gyártás ma már bonyolult feladat.

A gyártók nagy hangsúlyt fektetnek arra, hogy a hibás nyomtatott áramköri lapokat már a gyártás elején, közepén kiszurják. A piacon különböző módszerek jelentek meg (AOI, Automatikus Optikai Rendszer) [32].

Ezért olyan analogikai algoritmusokat dolgoztam ki, adtam meg [4], [8], amelyek segítségével kiszűrhetők a gyártás során keletkező illesztési hibák, zárlatok illetve az ezekből következő hibák is. Ezek nem csak a gyártás során használhatók, hanem az utólagos ellenőrzéseknél is. Ugyanis ez a két hibatípus fordul elő leggyakrabban illetve ezekre hibákra vezethető vissza az esetek többségében a nyomtatott áramkörök gyártásakor keletkező hibák.

4.1 A zárlatdetektáló algoritmus

A zárlatok komoly problémákat okoznak a különböző áramkörök készítésénél, használatánál. Ezek a zárlatok keletkezhetnek a gyártás során, de akár a huzamosabb működés után is. Ebben a fejezetben azt az eljárásomat ismertetem, amelynek a segítségével a gyártás során keletkező zárlatok jelenlétét detektálhatjuk. Ezek a hibák tetszőleges vezeték, jelek között fennállhatnak. A bemutatandó analogikai algoritmus egyoldalas, de akár két- vagy többretegű nyomtatott áramköröket is képes analizálni.

A zárlatkereső algoritmus hullámgenerálás segítségével mutatja ki a zárlat meglétét. Egy előre kijelölt pontból (forrpon, PAD) bináris hullámot indítunk, és ennek segítségével felderítjük az összes olyan forrpon, amelyek egy közös ekvipotenciális jelvezetékhez tartoznak. Ha zárlat található a kijelölt és egy másik jelvezeték között, akkor ez a bináris hullám terjedni fog a másik jelvezetéken is, tehát más PAD-ek is megjelennek. Az összes forrpon, amelyeket a hullámgenerálással találtunk meg, egy külön képen (marker) "gyűjtjük össze", és a feldolgozás végén egy másik képpel, az ún. referenciaképpel hasonlítjuk össze. Ezek a képeket (marker, referencia) egy későbbi alfejezetben részletesen definiálni fogom.

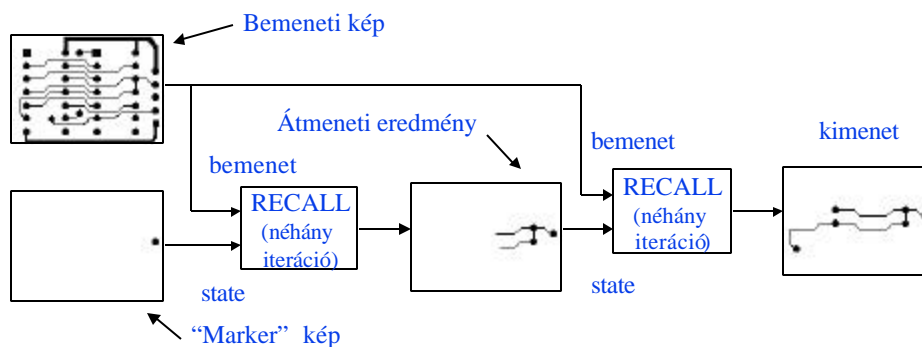
4.1.1 A "recall" template működése

Az analogikai algoritmus az úgynevezett "recall" template-re épül (4.1.ábra), ezért fontosnak tartom egy külön fejezetben ismertetni ezt a template-et [35]. Ez a template irányított hullámokat állít elő. Ezzel az irányított bináris hullámmal keressük meg a nyomtatott áramkört egy kijelölt jelvezetékhez tartozó összes forrponot.

$$A = \begin{bmatrix} 0.5 & 0.5 & 0.5 \\ 0.5 & 4 & 0.5 \\ 0.5 & 0.5 & 0.5 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad Z = 3$$

4.1. ábra Recall template

A "recall" template "blokkvázlatát" láthatjuk a 4.2. ábrán. A template-nek két bemenete van. Az egyik bemenetre (a tényleges input) betöltjük a teljes képet. A másik bemenetre (kezdeti állapot, state) azt a képet töltjük le, amelyen a hullámot szeretnénk generálni.



4.2. ábra A "recall" template működése

A template futtatása során a generált hullám a kezdeti állapotra kerülő képen lévő fekete objektumból indul ki, és csak abba az irányba megy, csak azt az alakzatot veszi fel, ami a template tényleges bemenetén lévő képen van.

Természetesen a marker képen közvetlenül is előállítható a kijelölt objektumból a hozzátartozó rajzolat, nem kell az átmeneti eredményt ismételtén letölteni a "recall" template state bemenetére. Ekkor még hosszabb ideig, több iteráción keresztül kell futtatni a template-et a CNN struktúrában.

Rövidzárkereso analogikai algoritmust már publikáltak a CNN szakirodalomban [26], de az az algoritmus nem képes megkeresni az olyan zárlatokat, amelyek az átvezetések (via) miatt keletkeznek. További hátránya, hogy csak a gyártási hibás rövidzárakat találja meg. A most bemutatásra kerülő algoritmus [3], [8] ezeket a hátrányokat küszöböli ki. Tekintettel arra, hogy ez az analogikai algoritmus propagáló template-et is tartalmaz, ezért az algoritmus futási ideje függ a vizsgálandó nyomtatott áramkör méretétől.

4.1.2 A zárlatdetektáló analogikai algoritmus

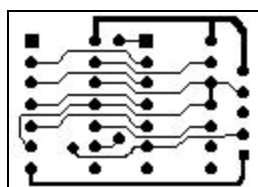
Az 4.7. ábrán látható analogikai algoritmus egy kétoldalas nyomtatott áramkör analízisét képes elvégezni [3], [4]. Az algoritmus azon az ötleten alapszik, hogy egy vagy több, előre kijelölt pontból (célszerűen egy PAD-ból, egy forrpontból) hullámokat indítunk. Ezeket a pontokat tartalmazó képet "marker" képnek nevezem. A markerképen alapesetben csak egy forrpont van, az ehhez tartozó jelvezetéknel keressük a zárlatot.

Tehát marker képnek nevezem azt a képet, amelyen hullámokat generálunk egy (vagy több) forrszemből (pad). Ilyen kép látható a 4.6. ábrán. Ezen a képen építjük fel bináris hullámok segítségével a teljes vizsgálandó jelvezetékét.

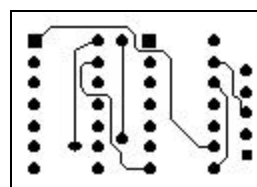
Ezek a hullámok csak azokon a helyeken terjedhetnek, ahol a vizsgálandó képen (4.3., 4.4. ábrák) fekete vonal, tehát vezető van. A hullámok segítségével felderítjük az adott jelhez tartozó összes forrpontot. Ha zárlat van, akkor a más jelvezetékhez tartozó forrpontok is megjelennek a kimeneti képen. Végül ezt a képet egy "referencia" képnek nevezett ábrával összevetve következtethetünk arra, hogy van-e zárlat a kijelölt vezeték között.

A referenciaképen két vagy több forrpont van. Mindegyik forrponthoz egy-egy, egymástól független jelvezeték tartozik, és ezek között a jelvezeték között keressük a zárlatot. A vizsgálandó jelvezetéknek csak egy-egy forrpontja található a képen. Ilyen referenciakép látható a 4.5. ábrán.

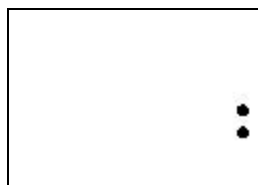
Az algoritmusnak tehát több bemenete van (ez függ a nyomtatott áramkör rétegszámától). Ezek a bemenetek a következők: minden oldalnak (forrasztási, alkatrész) van egy-egy képe, van egy markerkép és egy referenciakép. Az alkatrészfeloldi oldal (TOP) a 4.3. a forrasztásfeloldi oldal (BOTTOM) a 4.4., a referenciakép a 4.5. és a markerkép a 4.6. ábrán látható.



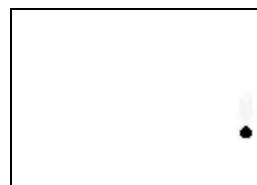
4.3. ábra
A TOP oldal képe



4.4. ábra
A BOTTOM oldal képe



4.5. ábra
A referenciakép

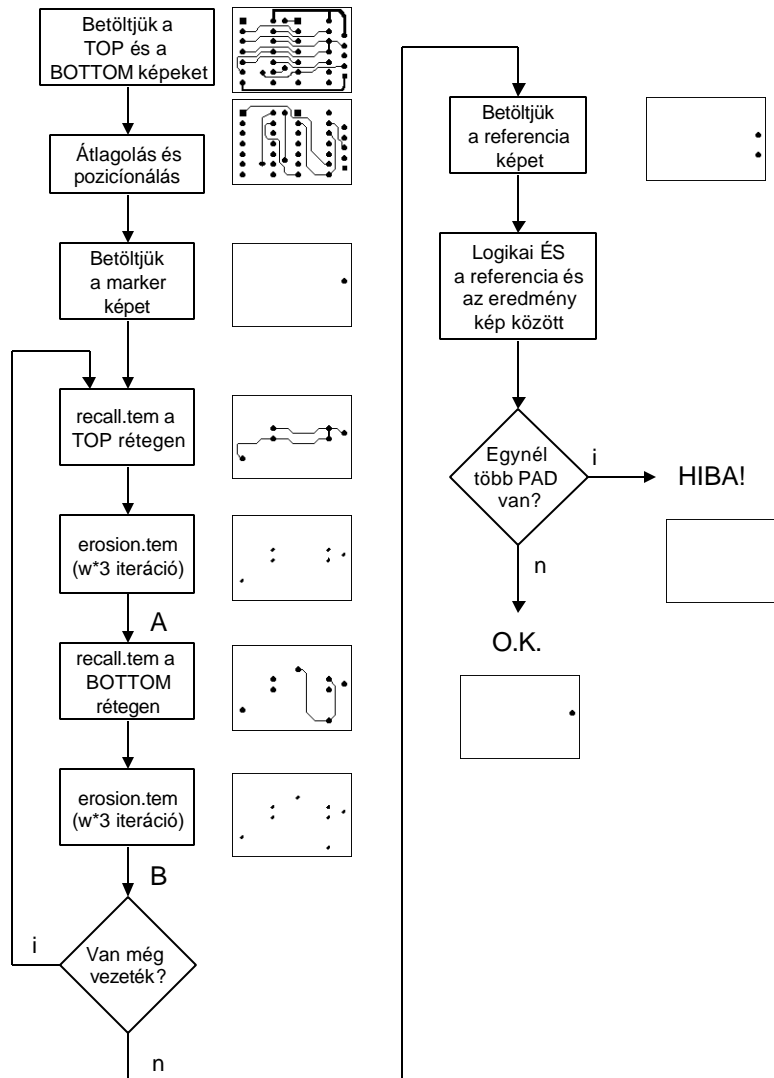


4.6. ábra
A markerkép

Tekintettel arra, hogy a szakirodalomban és a különböző áramkörtervező szoftvereknél a "TOP" és a "BOTTOM" kifejezés terjedt el, ezért a későbbiekben ezeket a szakkifejezéseket használom.

Az egyszerűség kedvéért csak két jelvezeték között fogjuk keresni a zárlatot egy kétoldalú nyomtatott áramkört. Természetesen az algoritmus képes több, egymástól független ekvipotenciális felület között lévő zárlatok detektálására egy többretegű panelen.

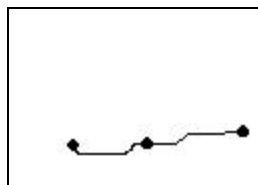
Eloszor a két réteg beszkenelt képét (TOP, BOTTOM layer) töltjük be a CNN-UM platform memóriájába. Ezután a képeket átalakítjuk fekete-fehér képekké az átlagoló (average.tem) vagy a küszöböző (threshold.tem) template segítségével.



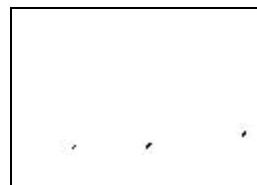
4.7. ábra A rövidzárdetektáló algoritmus folyamatábrája

Az így kapott fekete-fehér képeket pozícionáljuk egymáshoz a két képet, hogy a ketto egymással szinkronban legyen, majd betöltjük a CNN-UM egyik memóriájába (LLM) a "marker" képet (4.6. ábra). Ezt az egyik beszkenelt képből is előállíthatjuk úgy, hogy a vezeteket és a felesleges forrszemeket letöröljük. Ezután a muvelet után a "recall" template-et futtatjuk a marker képen, hogy kialakuljon rajta a forrpontoz

(forrponthoz) tartozó jelvezeték (4.8. ábra). Amikor egy meghatározott oldalon (itt a TOP) az összefüggő jelvezeték egy adott részéhez tartozó összes forrponot meghatároztuk, eróziót alkalmazunk az erosion.tem [35] template-tel három iteráción át (4.9. ábra). Az így kapott képet (amelyen csak a forrponok vannak) fogjuk marker képként használni, ezen fogjuk felépíteni a kijelölt jelvezeték további részét. Ezután a "recall" template bemenetére betöltjük a másik réteg (BOTTOM) képét és a template-et meghatározott ideig futtatjuk.



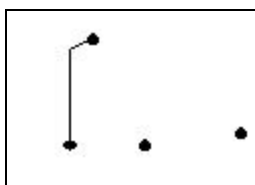
4.8. ábra
*A marker kép néhány
iteráció után, a TOP oldalon*



4.9. ábra
*A marker kép az
erózió után*

Az új marker képen megjelenik a BOTTOM oldalon található jelvezeték, és a hozzá kapcsolódó forrponok (4.10. ábra). Ezek után ismét néhány iteráció erejéig eróziót hajtunk végre, hogy csak a forrponok legyenek a képen (4.11. ábra).

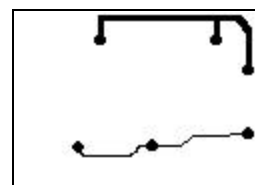
(Természetesen az iterációk pontos számát a vizsgálandó nyomtatott áramkör méretei határozzák meg.) Ez a kép lesz az új marker kép, amelyet a recall template "state" bemenetére töltünk, a template bemenetére pedig ismét a TOP oldali réteg képe kerül. Néhány iteráció után kapjuk a 4.12. ábrán látható rajzolatot.



4.10. ábra
*A megrajzolt jelvezeték
az erózió után*



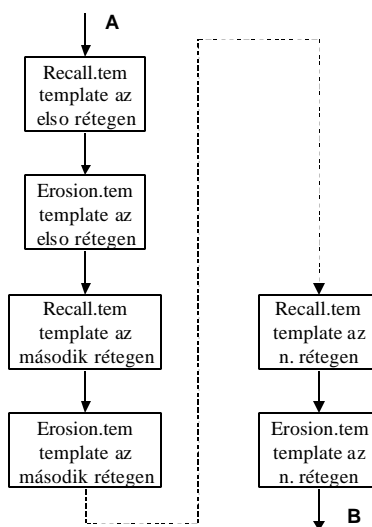
4.11. ábra
*A marker kép
néhány iteráció után,
a BOTTOM oldalon*



4.12. ábra
*A megtalált forrponokból
felépítettük a TOP oldalon
a jelvezeteket*

Ezeket a lépéseket, eljárásokat addig kell ismételnünk, amíg az előre kijelölt jelvezetékhez tartozó összes forrponot meg nem határoztuk. Ha az előre kijelölt ekvipotenciális felülethez tartozó összes forrponot, furatot megtaláltuk az eredményképen, akkor logikai ÉS kapcsolatot készítünk az eredménykép és a referenciakép között. Ha ennek a műveletnek a kimeneti képe megegyezik a referenciaképpel, akkor az adott két (vagy több) jelvezeték között zárlat található. Azért van ekkor zárlat, mert a referenciaképen legalább két, egymástól független ekvipotenciális felülethez tartozó PAD található, és ezeket kaptuk vissza a logikai ÉS után.

Az elobb bemutatott algoritmust kétoldalas nyomtatott áramkör analízisére mutattam be. Természetesen kiegészíthető többrétegu nyomtatott áramkörökre is, ezt mutatja a 4.13. ábra.



4.13. ábra *Kiegészítés a folyamatábrához*

A 4.7. ábrán látható algoritmus "A" és "B" pontjai közé kell beilleszteni ezt a kiegészítést, és tetszőleges rétegszámú áramkör vizsgálható.

Az általam megadott analógiai algoritmus képes az előre kijelölt jelvezetékek között a zárlatok megtalálására.

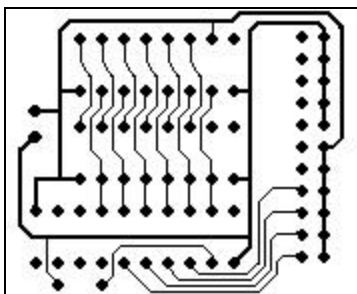
Az algoritmus fobb tulajdonságai:

- a futási idő függ a nyomtatott áramkör lineáris méretétől;
- a futási idő független a hibák számától, elhelyezkedésétől.
- az algoritmus csak előre kijelölt, egymástól független jelvezetékek között tud zárlatot keresni

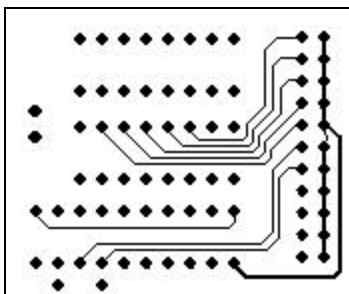
4.1.3 Egy tesztáramkör analízise

Szeretném egy egyszeru példán bemutatni az analogikai algoritmusom működését. A tesztáramkör egy kétoldalas nyomtatott áramkör, amelynek a mérete 176*144 pixel. Két futtatási eredményt ismertetek, az első esetben a nyomtatott áramkör forrasztási oldalán (BOTTOM) egy zárlat található, a második példában egy teljesen jó áramkört fogok analizálni.

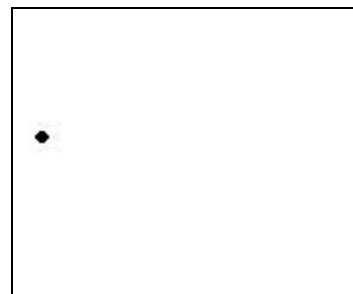
A 4.14. és a 4.15. ábra mutatja a tesztáramkör alkatrész és forrasztási oldalát, míg a marker kép a következő ábrán látható.



4.14. ábra
A tesztáramkör TOP oldala

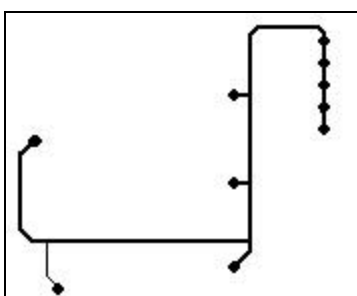


4.15. ábra
A tesztáramkör BOTTOM oldala

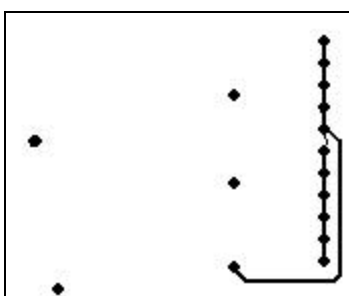


4.16. ábra
A marker kép

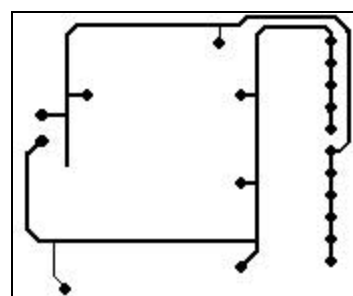
A recall template két bemenetére betöltjük a "TOP" és a marker képet. Néhány iteráció után a marker képen megjelenik a kijelölt jelvezeték egy része (4.17. ábra), amely az alkatrész oldalon látható. Ezután eróziót hajtunk végre, hogy csak a forrponatok maradjanak, majd ezt és a forrasztásfeloldali oldalt töltjük a recall template-re. Néhány iteráció után a következő képet kapjuk (4.18. ábra).



4.17. ábra
A marker kép néhány iteráció után (TOP)



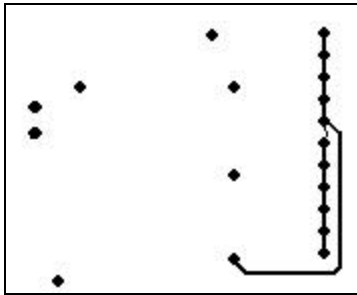
4.18. ábra
A marker kép néhány iteráció után (BOTTOM)



4.19. ábra
A TOP oldal a marker képen a BOTTOM oldal eróziója és néhány iteráció után

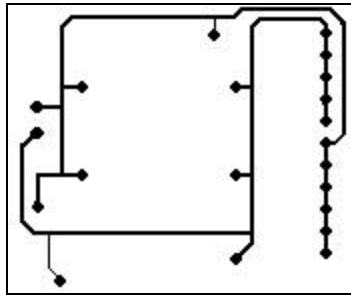
Ezután ismét a "TOP" oldalon generálunk hullámokat a marker képen (4.19. ábra). A hullámgenerálás és az erózió után ismét hullámokat generálunk a recall template segítségével a BOTTOM oldalon. Így még több forrponatot kapunk meg, amelyek a kijelölt jelvezetékhez tartoznak (4.20. ábra). A következő recall template futtatásakor már megkapjuk az összes olyan vezeték (track) és furatpontot, amelyek a kijelölt

jelvezetékhez (vagy zárlat esetén más vezetohöz) tartoznak (4.21. ábra). Ezekből a furatpontokból indítjuk újra a hullámokat, hogy a "BOTTOM" oldalon lévő maradék jelvezetéseket, és ezekhez tartozó esetleges zárlatokat, forrponthoz is megtaláljuk (4.22. ábra).



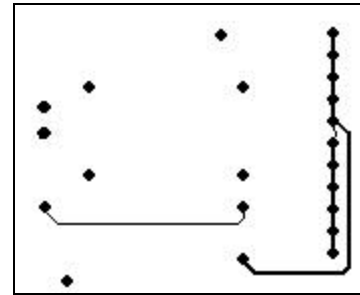
4.20. ábra

A BOTTOM oldal a marker képen az erózió és a recall iteráció után (TOP)



4.21. ábra

A marker képen a TOP oldal a kijelölt jelvezetékkel

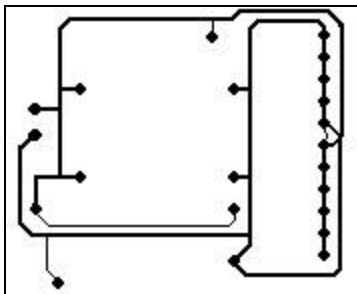


4.22. ábra

A BOTTOM oldalon látható, egy potenciálra lévő jelvezetékkel

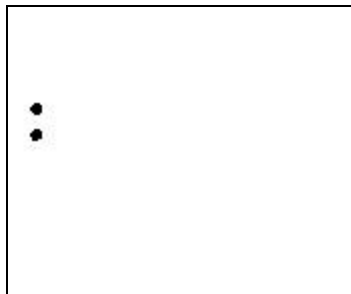
Miután megkaptuk az alkatrész és a forrasztási oldalon lévő, a kijelölt jelvezetékhez tartozó összes forrponthoz, logikai ÉS kapcsolatot hajtunk végre az eredménykép és a referencia kép között. Az eredményképet úgy kaptam meg, hogy a TOP és a BOTTOM oldal képeit logikai úton összeadtam (logikai VAGY, 4.23. ábra). Ekkor az összes megtalált vezeték egy képen ábrázoltam, de mivel a két képen a forrponthoz száma, helyzete megegyezik (hiszen ugyanazok), ezért a logikai VAGY függvény használata nem követelmény az algoritmus futtatásánál.

Az eredményképet, vagy az egyik oldal (TOP, BOTTOM) képét logikai ÉS kapcsolatba hozzuk a referenciaképpel, amely az 4.24. ábrán látható.



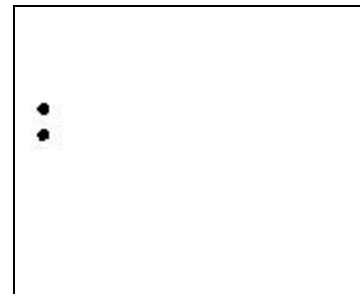
4.23. ábra

A kijelölt forrponthoz tartozó összes jelvezeték a TOP és a BOTTOM oldalon ("eredménykép")



4.24. ábra

A referenciakép



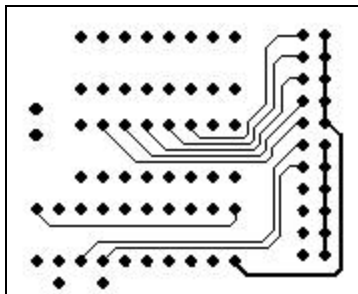
4.25. ábra

Az algoritmus kimeneti képe a logikai ÉS után

Az 4.24. ábrán mutatott referenciakép megegyezik az 4.25. ábrán lévő eredményképpel (ez az algoritmus kimeneti képe). Ez azt jelenti, hogy a nyomtatott áramkörön zárlat található a referenciaképen megjelölt két jelvezeték között. Látható

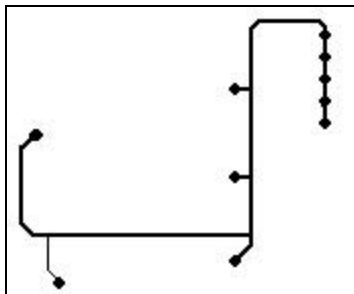
tehát az, hogy az analogikai algoritmus megmutatja a zárlat meglétét, de annak pontos helyét nem.

Ha a forrasztási (BOTTOM) oldal képét kicseréljük, akkor a következő eredményeket kapjuk.



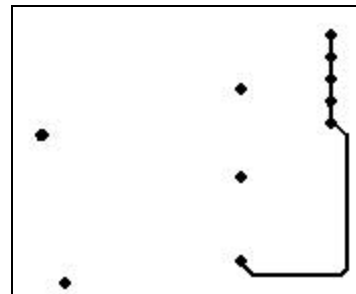
4.26. ábra

A jó BOTTOM oldal



4.27. ábra

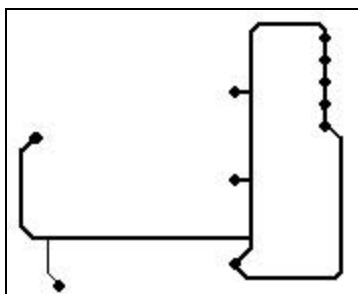
*A marker képen megjelenő,
a TOP oldalon látható
jelvezetékek*



4.28. ábra

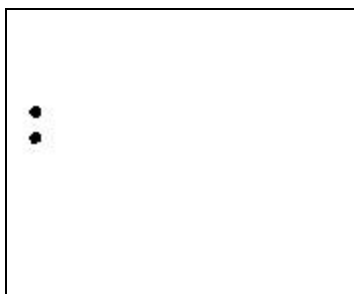
*Erózió és a recall template
futtatása után a BOTTOM
oldali jelvezetékek látható*

Ebben az esetben az algoritmus kimeneti képe különbözik a referenciaképtől, tehát nincs zárlat a kijelölt jelvezetékek között. Ha több jelvezeték között vizsgálunk a zárlat meglétét, akkor nem lenne zárlatos a panel, ha a referenciaképen lévő forrponok közül csak azt az egy PAD-et látnánk az eredményképen, ahonnan a hullámot indítottuk.



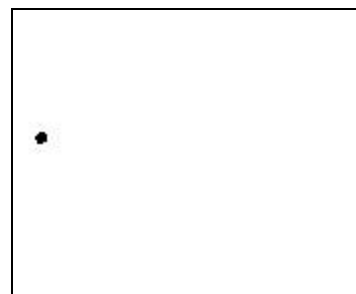
4.29. ábra

*A kijelölt forrponthoz tartozó
összes jelvezetékek a TOP és a
BOTTOM oldalon*



4.30. ábra

A referenciakép



4.31. ábra

*Az algoritmus kimeneti
képe a logikai ÉS után*

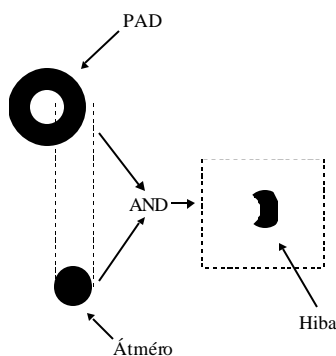
4.2 Illesztési hibát detektáló analogikai algoritmus

Egy nyomtatott áramkör készülhet furatszerelt vagy SMD technológiával. Amióta a kereskedelemben is beszerezhetőek az SMD alkatrészek, azóta az áramkörök döntő többségét főleg ezekkel az alkatrészekkel készítik el. (A vezető- és a szigetelési távolság a technika fejlődésével egyre kisebb lesz. Bonyolultabb áramkörök készíthetők el kisebb felületen.) Ugyanakkor a méretek csökkenésével megnövekszik a mesterfilm és a gyártásra elokészített panel között az illesztési hiba valószínűsége. Az 3. fejezetben már láthattuk, hogy az illesztési hiba az, amikor a forrpont furata nem illeszkedik szabályosan (vagy egyáltalán nem) a forrpontra. Ez a forrszem gallérjának elszakadásához vezet, sot, bizonyos esetekben zárlatokat eredményezhet.

Ebben a fejezetben egy olyan analogikai algoritmusomat [3], [4] mutatom be, amelynek a segítségével az illesztési hibák tetszőleges nyomtatott áramkör esetében kiküszöbölhetők. Az algoritmus kimeneti képén láthatók a hibák helyei és méretei. Elemezni fogom a vizsgálandó nyomtatott áramkörök méretei, az algoritmus lépésszámai és a futási idő közötti összefüggéseket.

4.2.1 Logikai függvények a bináris képeken

Ha a mesterfilmen a forrpontok furatai látszanak (tehát a forrpont nem egybefüggően fekete színű), akkor a mesterfilm és a gyártásra elokészített kifűrt panel képét logikai ÉS kapcsolatba hozzuk (4.32. ábra).

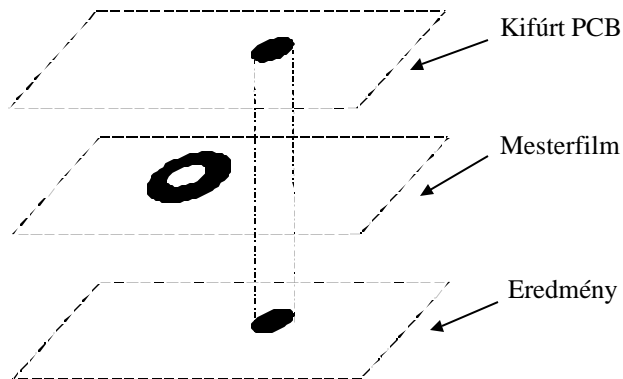


4.32. ábra Logikai "ÉS" a két kép között

Ha a mesterfilm jó, akkor a forrszem (PAD) furatát teljesen lefedi az előre kifűrt panelen lévo furat, akkor az ÉS kapcsolat eredménye egy fehér kép, hiszen a két képen lévo fekete objektumok között nincsen átfedés. Ha a mesterfilm hibás, akkor a forrszem nem az eredeti helyén van, tehát a két képen a fekete objektumok között átfedés lesz, és emiatt a logikai ÉS kapcsolat eredményképén a hiba lesz (4.32. ábra). A kimeneti képen az eltérés nagyságától függ a hiba mérete.

Ha a gyártásra elokészített panelen lévo furat és a mesterfilmen a hozzátartozó forrpont között túl nagy a távolság, akkor nem lesz átfedés eközött a két objektum között. Tehát a logikai ÉS kapcsolat nem fog hibát jelezni. Ezért célszeru a kifűrt

nyomtatott áramkör képéből kivonni a mesterfilm képét (4.33. ábra).

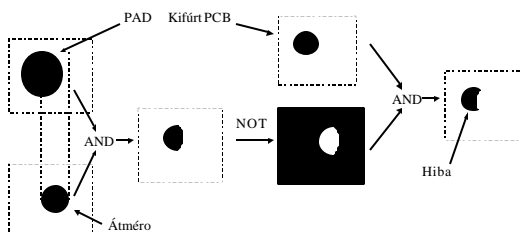


4.33. ábra Logikai kivonás a gyártásra elokészített nyomtatott áramköri lap és a mesterfilm között

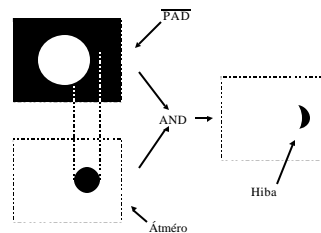
Ha ezt a kivonást használjuk, akkor elötte futtatni kell a "hole" template-et, hogy a forrpon belül, fehér színű részét kitöltsük feketével. Ekkor a jó helyen lévő illesztéseknél nem lesz látható a kimeneti képen a furat. Tekintettel arra, hogy ha egy mesterfilm megsérül, akkor ez nem csak egy forrponnál látható, lesznek kisebb eltérések is, amelyeket az ÉS függvényel kiszűrhetünk. (A kivonásos eljárást nem ajánlom, mert a "hole" template nehezen kezelhető.)

Ha a mesterfilmen a forrponok furatai nem látszanak (tehát a képen a furat nem látszik, a forrponnak és a furatnak is fekete a színe), akkor az elöbb leírt eljárás nem alkalmazható, hiszen egy teljesen jó mesterfilmmel az algoritmus kimeneti képen az előre kifűrt panel képét kapnánk vissza. Ezért ebben az esetben egy másik eljárást kell alkalmazni. Ezt mutatja a 4.34. a. ábra.

Eloször logikai ÉS függvényt kell alkalmazni a mesterfilm és a gyártásra elokészített (tehát előre kifűrt) panel képe között. Ha a két fekete objektum teljesen lefedi egymást, akkor a kimeneti képen egy fekete kört kapunk. Ha a lefedés nem teljes, akkor csak egy körcikkely látható a képen.



4.34 a. ábra Logikai "ÉS" és invertálás a két kép között



4.34 b. ábra Egyszerűsített megoldás

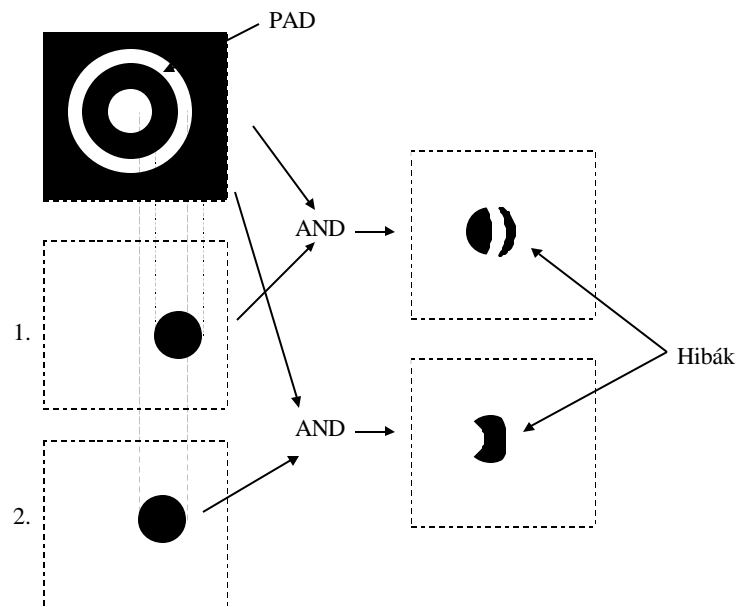
Ezután az eredményképet negáljuk, ekkor az ÉS függvény eredménye fekete felületen fehér körcikk lesz. Ezt a képet ismét logikai ÉS kapcsolatba hozzuk az előre kifűrt panel képével. A most megjelenő kimeneti képen már csak az illesztési hiba lesz látható, amelynek a mérete a mesterfilm illesztési hibájának nagyságától függ.

Ha a mesterfilm jó, a furat teljesen lefedi a forrponthoz tartozó furatot. Ekkor az első ÉS kapcsolatnál egy teljes fekete kört kapunk, viszont az újabb ÉS függvény alkalmazásánál a negált kép miatt a kimeneti kép teljesen fehér lesz.

Használhatunk egy egyszerűbb megoldást is, ezt mutatja a 4.34 b. ábra. Ha ezt a megoldást használjuk, akkor elhagyható egy logikai ÉS függvény. Ez megoldás a De Morgan azonosságából (4.1.) következik. Az "A" a mesterfilm (PAD), a "B" pedig az előre kifűrt panel képe.

$$(\overline{A \& B}) \& B = (\overline{A} + \overline{B}) \& B = \overline{A} \& B \quad (4.1.)$$

A nagysebességu digitális technikában illetve a rádiótechnikában gyakran alkalmazzák az úgynevezett "fill" típusú nyomtatott áramkört. Ennek az a lényege, hogy a fel nem használt szigetelofelületet rézzel töltik ki és ezt a szigetet állandó potenciálra, általában testre kötik. Ilyen típusú nyomtatott lap vizsgálatának néhány eleme látható a 4.35. ábrán.



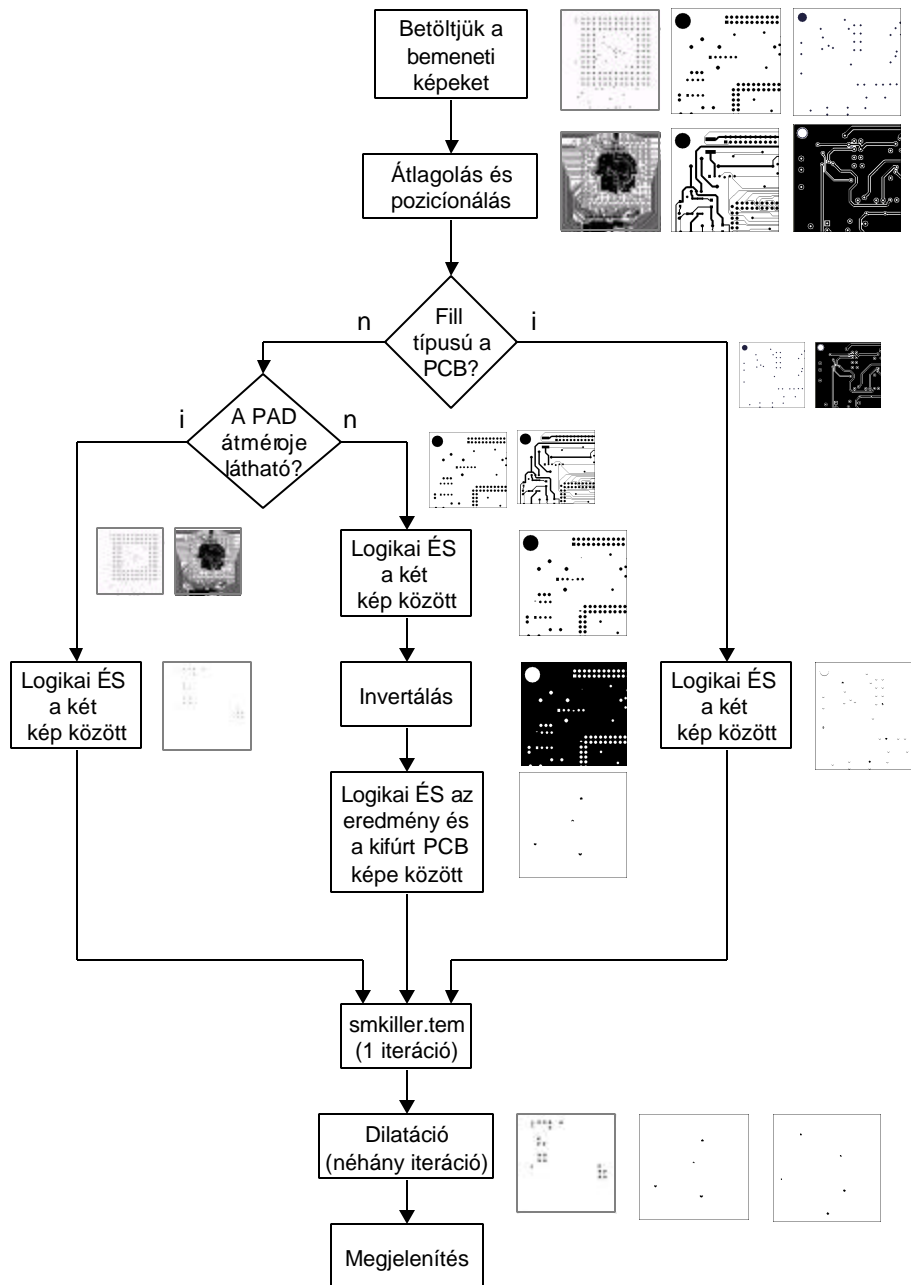
4.35. ábra Logikai "ÉS"-ek a két kép között

Ezen az ábrán két esetet látunk. Az elsonél nagyobb az eltérés a furat és a forrponthoz között. Az ÉS kapcsolat után a hibának a jelzése két részből tevődik össze. A "szakadás" a forrponthoz és a fix potenciálra kötött réz között lévő szigetelogyűrű miatt van. A második esetben az eltérés kisebb, a logikai ÉS függvény után a hiba egy deformált körcikkből áll.

4.2.2 Az illesztési hibát detektáló algoritmus

Az illesztési hibákat detektáló algoritmust a 4.36. ábra mutatja. Az algoritmus első két illetve az utolsó három lépése bármilyen mesterfilm típusnál megegyezik. Az algoritmus középső része függ a használni kívánt mesterfilmtől. Ezeket a lépéseket az előző fejezetben mutattam be.

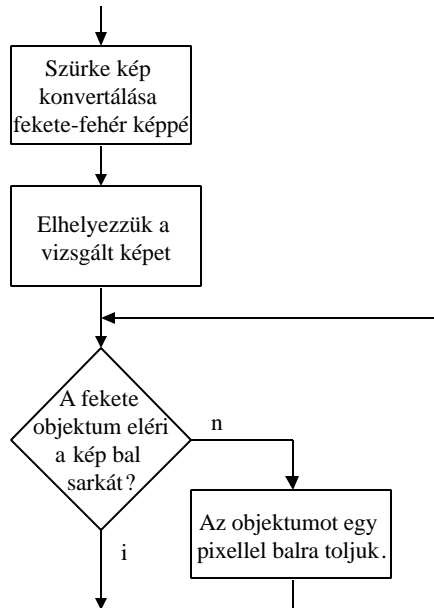
Az algoritmusnak két bemeneti és egy kimeneti képe van. A bemeneti képek a következők: a mesterfilm képe és a gyártásra előkészített, előre kifűrt lap képe.



4.36. ábra Az illesztési hibát detektáló algoritmus folyamatábrája

Eloszor betöltjük a bemeneti képeket. Tekintettel arra, hogy ezek színes képek, ezért a betöltés után átlagoljuk a képeket. Ez történhet a "treshold.tem" vagy az "average.tem" template-tel [35]. Ezután már fekete-fehér képekkel dolgozhatunk.

A két bemeneti kép (mesterfilm, gyártásra elokészített panel) egymással nincs teljesen fedésben, ezért illeszteni kell a két képet egymáshoz. Erre szolgál a 4.37-es ábrán lévő algoritmus részlet. Az egyik képet rögzítjük, a másik képet pedig addig mozgatjuk vízszintesen és függőlegesen, amíg el nem éri a kép bal felső sarkát. Tekintettel arra, hogy a két kép (mesterfilm, drill) mérete megegyezik, ezért ha a két kép bal felső sarka fedi egymást, akkor a képek teljesen fedik egymást, és ekkor különböző logikai műveletek hajthatók végre.



4.37. ábra Az algoritmus részlet a két kép fedésére

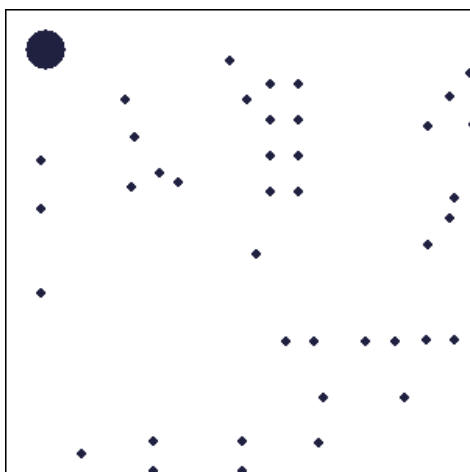
Az általam megadott analogikai algoritmus képes egy nyomtatott áramkör készítéséhez szükséges mesterfilmen megkeresni az összes illesztési hibát és ezeket a hibákat mérethelyesen kijelzi.

- Az algoritmus futási ideje független a vizsgálandó nyomtatott áramkör méretétől, a hibák számától és ezeknek a méretétől (chip-es implementálásnál a futási idő csak akkor független, ha a kép mérete kisebb vagy megegyezik a CNN array méretével).

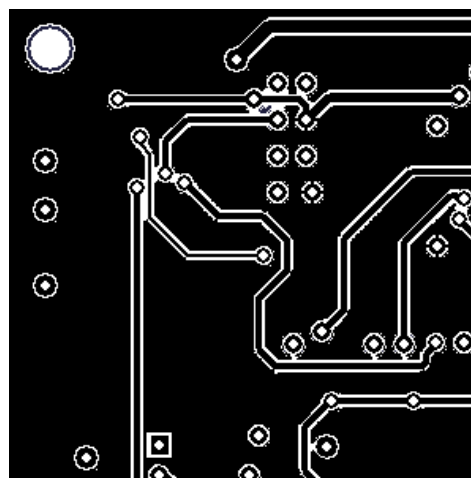
4.2.3 Két teszttáramkör analízise

Ebben a fejezetben két teszttáramkör analízisét mutatom be. Az egyik áramkör csak furatszerelt alkatrészeket tartalmaz és "fill" típusú. A második teszttáramkör tartalmaz SMD alkatrészeket is, de nincs rézzel kitöltve a szigetelofelülete.

Az első teszttáramkör mérete 301*314 pixel. Az algoritmus bemeneti képeit a 4.38. (kifúrt panel) és a 4.39. ábra (mesterfilm) mutatja.

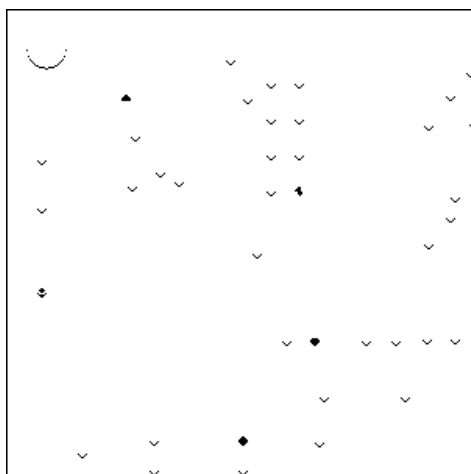


4.38. ábra
A kifúrt panel képe

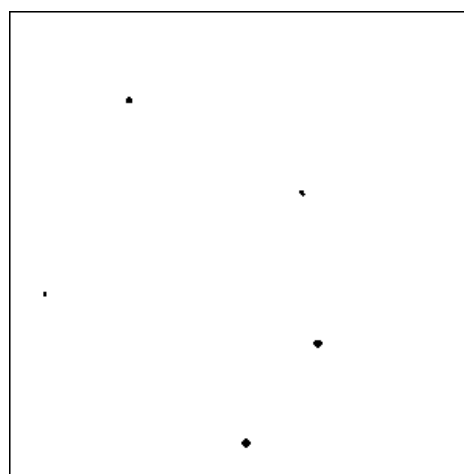


4.39. ábra
A mesterfilm képe

Az első két kép között logikai "ÉS" kapcsolatot hozunk létre, amelynek eredménye a 4.40. ábrán látható kép. A kisebb illesztési hibákat az "smkiller" template-tel eltüntethetjük. Az smkiller.tem template-et egy vagy két iteráción keresztül futtatjuk. Ekkor kapjuk a következő, 4.41. ábrán látható képet.

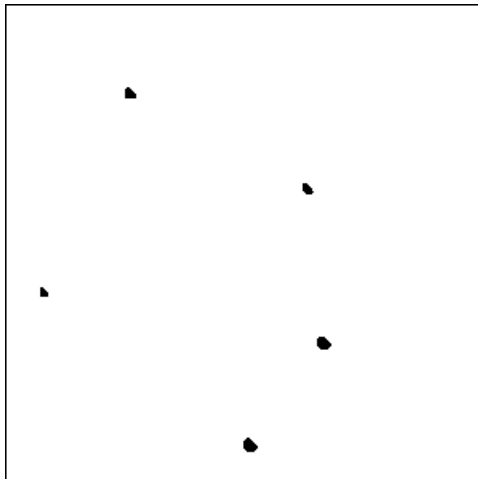


4.40. ábra
Az ÉS kapcsolat eredmény képe



4.41. ábra
*Az ÉS kapcsolat képe az "smkiller"
template futtatása után*

Nagyobb méretű mesterfilmeknél a néhány pixel átméretű hibák szemmel nehezen észrevehetők. Ezért érdemes használni a "dilatation" nevű template-et, amely néhány iteráció után a fekete objektumokat megnöveli a képen. Ezt a template-et futtatva az 4.41. ábrán látható képre, a következő eredményt kapjuk (4.42. ábra). Ez az algoritmus kimeneti képe. Itt pontosan láthatóak az illesztési hibák.

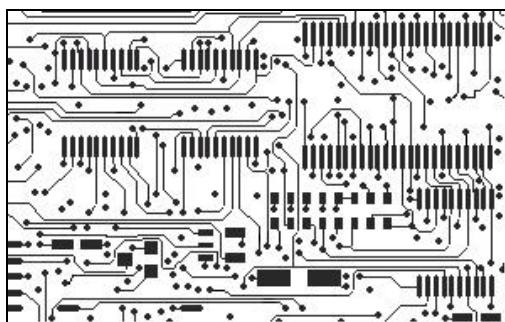


4.42. ábra

Az algoritmus kimeneti képe (Itt jól láthatók az illesztési hibák elhelyezkedései és méretei)

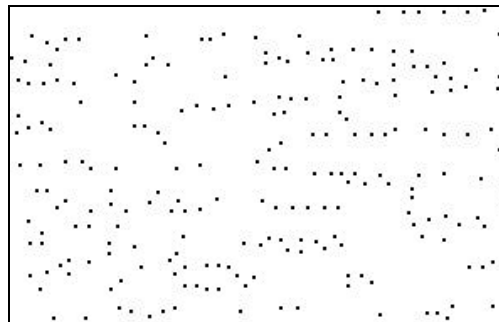
A következő tesztáramkör SMD és furatszerelt alkatrészeket is tartalmaz. A forrpontokon lévő furatok nem látszanak a mesterfilmen. Az ilyen filmek a leggyakoribbak a nyomtatott áramkörök gyártásában, ezért szeretnék most egy ilyen példát is bemutatni.

Ilyen filmet láthatunk a 4.43., az előre kifűrt, gyártásra elokészített nyomtatott áramköri lapot pedig a 4.44. ábrán. A tesztáramkör mérete 296*187 pixel.



4.43. ábra

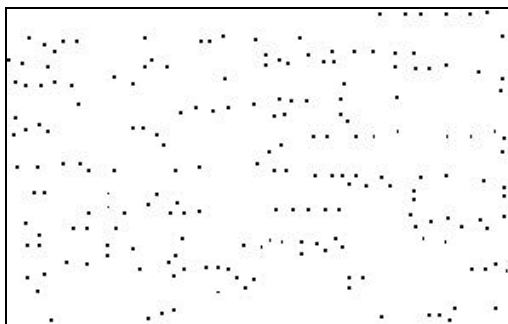
A mesterfilm képe



4.44. ábra

A kifűrt panel képe

A két film között egy logikai "ÉS" kapcsolatot valósítunk meg, akár a "logand" template-tel, akár a CNN chip aritmetikai funkciójának a segítségével (4.45. ábra). Ezután a kapott képet invertáljuk (4.46. ábra).

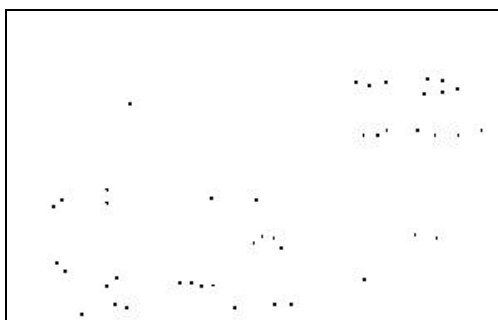


4.45. ábra
A logikai ÉS eredményének a képe

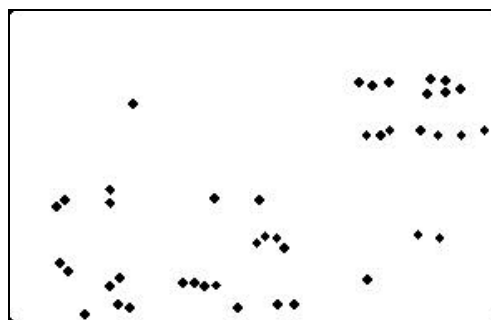


4.46. ábra
Az invertálás után kapott kép

Az invertált kép és a kifűrt panel képe között ismét logikai "ÉS" kapcsolatot hajtunk végre, és ezzel a lépéssel megkaptuk az illesztési hibákat (4.47. ábra).



4.47. ábra
Az illesztési hibák



4.48. ábra
A felnagyított illesztési hibák

A hibák nehezen látszanak, ezért a fekete objektumokat, pixeleket célszerű felnagyítani. Ekkor kapjuk meg az algoritmusom kimeneti képét, amely a 4.48. ábrán látható.

4.3 Összefoglalás

Az ebben a fejezetben bemutatott nyomtatott áramkörök gyártásközi ellenőrzésére szolgáló algoritmusok könnyen megvalósíthatók a CNN segítségével, hiszen a kép minden egyes pixeléhez egy CNN cella rendelhető.

Ezek az algoritmusok a gyártás során előforduló leggyakoribb hibák kiszűréséhez készültek, tehát az iparban elonyösen használhatók.

A nyomtatott áramkörök gyártásában használható hibakereső algoritmusaimat nem csak szoftver szimulátorral teszteltem, hanem a 20*22-es [21] és a 64*64-es [22] CNN-UM chip-ekkel is. Az elobb ismertetett hibakereső algoritmusaim futási idejeit megmértem az előző fejezetekben bemutatott tesztképeken, összehasonlítottam és a 4.1. táblázatban feltüntettem [8].

Algoritmusok	ALADDIN Szoftver szimulátor, Pentium, 466MHz	ALADDIN 20*22-es CNN-UM	ALADDIN 64*64-es CNN-UM
Illesztési hibákat detektáló	9.28 sec	52 msec	22 msec
Zárlatkereso	105 sec	N/A	90 msec *

4.1. táblázat A két algoritmus futási ideje

(* Az algoritmus magjának a futási ideje 30 msec a 64*64-es CNN-UM chip-en.)

Jól látható, hogy a különböző CNN-UM chip-eken nagyságrendekkel jobb eredményeket kapunk. Ezeknek az analógiai algoritmusaimnak nagy előnye az, hogy a futási idő független a vizsgálandó kép méretétől. Természetesen, ha a kép mérete nagyobb, mint a processzortömb, akkor a képet fel kell darabolni, és ezeket külön-külön letölteni a chip-re. Ekkor függ csak a futási idő a kép méretétől. Ezek a járulékos műveletek illetve a képrészletek feldolgozása megnöveli a futási időt. Ebben a fejezetben ilyen példákat mutattam be.

A tesztáramkörök kétrétegu nyomtatott áramköri lapok voltak, de természetesen az általam megadott analógiai algoritmusok képesek "n" számú réteg feldolgozására is. Ekkor természetesen a futási idő megnövekszik.

A két analógiai algoritmusom csak szimmetrikus template-eket használ, így a stabilitás automatikusan biztosított.

Fontos kérdés, hogy a megoldásaim hogyan viszonyulnak a ma kapható legjobb AOI (Automatic Optical Inspection, Automatikus Optikai Rendszer) rendszerekhez [32]. Az általam megadott algoritmusok számolási teljesítménye kb. $0.3 \cdot 10^6 - 4 \cdot 10^6$ pixels/s. A ma elterjedt, leginkább használt AOI rendszerek sebessége egy nagyságrendbe esik az általam megadott eljárásokkal (vagy eggyel jobb), de ezeknek az árai jóval nagyobbak a CNN fejlesztorendszerek árainál. Ezeket a sebességértékeket a 64*64-es CNN-UM chip használata esetén értem el. Ez a sebesség tovább növelhető, ha a nem rég megjelent 128*128-as chip-et használjuk egy gyorsabb, jobb kiépítettségű PC-vel. Tovább növelhető ez a sebesség, ha a legújabb, Aladdin Pro fejlesztorendszert használjuk.

További előnye az analógiai ellenőrző algoritmusoknak, hogy könnyen kiegészíthetők, könnyen módosíthatóak. A mai AOI rendszerek erre képtelenek.

A harmadik függelékben található a logikai processzor és a hozzátartozó platform rövid bemutatása. Ezen a platformon lefuttattuk a két analógiai algoritmust, ezek az ábrák is láthatóak ebben a függelékben.

5. Az emulált digitális CNN-UM (CASTLE) bemutatása

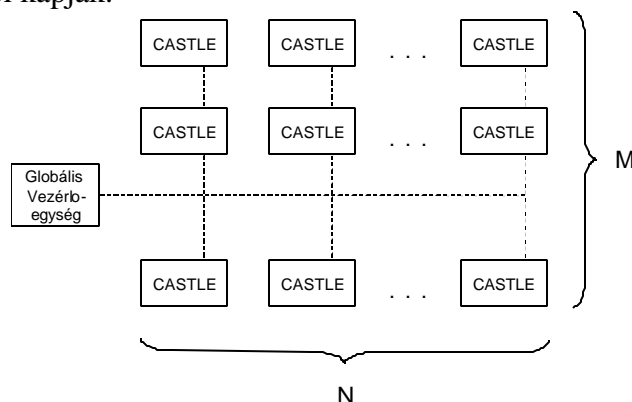
Az első fejezetben már láthattuk azt, hogy a digitálisan emulált CNN-UM megoldásoknak létjogosultságuk van. A digitális technika és a mikroelektronika fejlődésével egyre gyorsabb, kisebb digitális chip-ek készíthetők. Ezeknek a megoldásoknak nagy előnye az, hogy pontosak, sajnos azonban a felbontás növelésével csökken a működési sebesség.

1998-2001 között az MTA-SZTAKI-ban elkészült egy digitálisan emulált (CASTLE) architektúra [5]. Ezt a megoldást megvalósítottuk szilíciumon is, $0,35\ \mu\text{m}$, négy réteges és egy poliszilíciumos technológiával. A készítő az architektúra tervezésének során nem csak a valós idejű videojelek feldolgozására gondoltak, hanem a különböző parciális differenciálegyenletek megoldására is [10]. Ennek a megoldásnak a pontossága lényegesen jobb, mint az analóg társainak, és a sebessége is összemérhető az analóg CNN-UM-ekkel. A pontosság változtatható a feladat megoldásának függvényében és ezáltal a sebesség is. Minél kisebb a felbontás, annál gyorsabb a számítás.

A különböző számítások szerint [5] a CASTLE architektúra képes komplex képfeldolgozási problémák megoldására. Feltételezve, hogy egy digitális videó képfolyam 240×320 pixelből áll, másodpercenként 25 képváltással több mint 500 CNN iteráció (3×3 -as konvolúció) végezhető el 12 bites felbontás mellett [5].

5.1 CASTLE architektúra

A processzortömb elrendezése az 5.1. ábrán látható [5]. Az array tartalmaz egy központi vezérlő egységet és egy $N \times M$ darab fizikai processzort. Ez a központi egység funkciója megfelel a 2. fejezetben bemutatott 2.8. ábra GAPIU egység feladatával. A processzor egységek külön memóriaegységekkel rendelkeznek, ezeknek a bemutatására később kerül sor. A processzorandó képet függőleges csíkokra osztjuk, és ezek a képrészek függőlegesen haladnak az array-n felülről lefelé. Az egy sorban lévő processzorok egymással is kommunikálnak. A processzorok a központi vezérlőegységen keresztül is kapnak utasításokat, de minden chip-nek külön vezérlőegysége van. A processzortömbben a processzorok szinkronizálva vannak egymáshoz, a vezérlőjelek teljesen digitálisak, ezeket a jeleket a vezérlőegységtől kapják.



5.1. ábra *Processzortömb CASTLE processzorokkal*

Egy órajel ciklusban egy processzorsor kapja meg az elozo iteráció eredményét a felette lévo processzorsortól, számol egy iteráció alatt, és ezt az eredményt a következő iterációban elküldi az alatta lévo processzorsornak. Később látni fogjuk, hogy egy iteráció nem felel meg egy órajel ciklusnak!

A processzorok nem csak a saját képmemóriájukat olvashatják, hanem a vízszintes szomszédjaik képmemóriáinak a széleit is. Ezáltal valósul meg a szomszédok közötti kommunikáció.

5.2 A processzormag kialakítása

Az architektúra tervezése során a fejlesztok a Chua-Yang modelltól indultak ki (5.1).

$$\dot{x}_{ij}(t) = -x(t) + \sum_{C(k,l) \in N_r(i,j)} A_{k,l} y_{kl}(t) + \sum_{C(k,l) \in N_r(i,j)} B_{k,l} u_{kl}(t) + z_{ij} \quad (5.1)$$

Ez az állapotegyenletet időben diszkrétizálható az előrelépo Euler formula segítségével (5.2).

$$x_{ij}(k+1) = (1-h)x_{ij}(k) + h \left[\sum_{C(k,l) \in N_r(i,j)} A_{ij,kl} y_{kl}(k) + \sum_{C(k,l) \in N_r(i,j)} B_{ij,kl} u_{kl}(k) + z_{ij} \right]$$

$$h = \frac{\Delta t}{t} \quad t = R_x C_x \quad (5.2)$$

Az összes processzornak minden képpontban ennek az egyenletnek az alapján kell számolnia. A kiszámítás egyszerűsítésének érdekében megpróbálták a lehető legtöbb változót kihagyni, egyszerűbbé tenni a működési egyenletet, ezáltal a szilíciumon realizálandó processzor működése, vezérlése és felépítése egyszerűbb lesz. Így csökkenthetjük a processzor szilíciumméretét is. Ezen egyenlet legegyszerűbb formájának keresésénél használhatjuk az FSR (Full Signal Range) modellt. Az FSR modell lényege, hogy a CNN processzor állapotváltozójának az értéke megegyezik a kimenetével, tehát az $[-1,+1]$ intervallumban változhat az állapotváltozó értéke.

Tovább egyszerűsíthetjük az egyenletet (és ezáltal a céláramkört is), ha a "h" és az "(1-h)" tagokkal módosítjuk az "A", "B" template-eket. Bevezetjük tehát a \hat{A} , \hat{B} template-eket (5.2. ábra).

$$\hat{A} = \begin{bmatrix} ha_{-1-1} & ha_{-10} & ha_{-11} \\ ha_{0-1} & 1-h+ha_{00} & ha_{01} \\ ha_{1-1} & ha_{10} & ha_{11} \end{bmatrix}; \quad \hat{B} = \begin{bmatrix} hb_{-1-1} & hb_{-10} & hb_{-11} \\ hb_{0-1} & hb_{00} & hb_{01} \\ hb_{1-1} & hb_{10} & hb_{11} \end{bmatrix};$$

5.2. ábra Módosított "A" és "B" template

A template-ek módosítása és a levezetés után a következő egyenleteket kapjuk: [5], [6]

$$x_{ij}(n+1) = \sum_{C(kl) \in N_r(i,j)} \hat{A}_{ij,kl} * x_{kl}(n) + g_{ij} \quad (5.3)$$

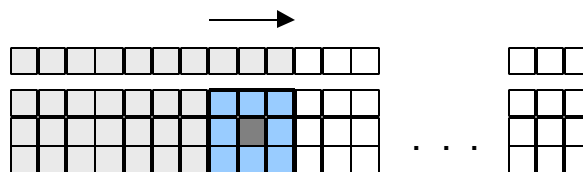
$$g_{ij} = \sum_{C(kl) \in N_r(i,j)} \hat{B}_{ij,kl} * u_{kl}(n) + h * z_{ij} \quad (5.4)$$

ahol

- "h" az idolepés
- "u_{kl}" a bemenet
- "x_{ij}(n+1)" a CNN cella következő állapota
- "x_{kl}(n)" a szomszédos cellák (pixelek) állapota
- "g_{ij}" és "z_{ij}" konstansok

Az első lépésben kiszámoljuk a "g_{ij}" konstans értékét. Látható az 5.3 és az 5.4 egyenletekből, hogy a számoláshoz kilenc template-érték, kilenc állapotérték (pixel) és egy áramérték ("z_{ij}") szükséges. Ezeket az értékeket (template, pixel és áram) a chip különböző memóriaegységeiben tároljuk.

Az egyenletek alapján látható, hogy a valós idejű működésnél 20 darab 12 bites (6 bites) érték (paraméter) mozgatását kell megoldanunk. Ez ugyanolyan nehéz, mintha az egész képet a chipen tárolnánk. A paraméterek nagy része a konvolúció elvégzéséhez szükséges. Ha nem az egész képet, hanem csak annak egy sávját tároljuk a chipen (5.3. ábra), akkor mindig csak egy paramétert kell betöltenünk.

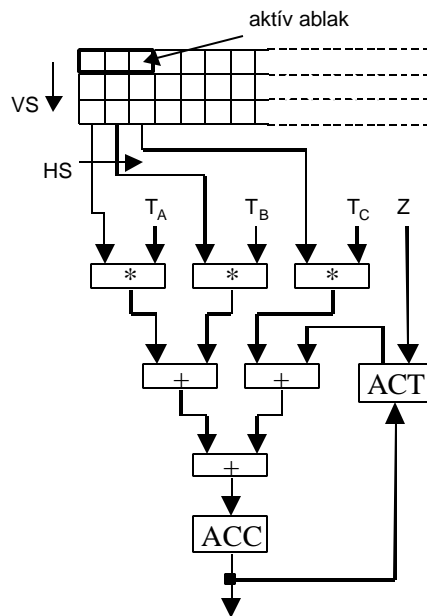


5.3. ábra Képfeldolgozás a CASTLE architektúrában

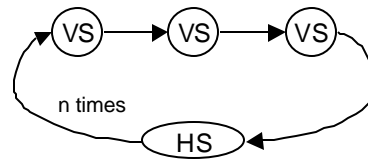
Így minden iterációban a processzornak csak 3 I/O műveletet kell végrehajtania, és a szilíciumon lévő memóriák mérete is minimális, hiszen nem tárolunk el olyan pixeleket, amelyeket csak "később" (~40 iteráció) fogunk felhasználni.

Az 5.4. ábrán látható a pixelmemória és az aritmetikai egység egyszerűsített vázlatja. Itt nincs feltüntetve a különböző határoló, kerekítő egység.

Az első sorban lévő három szorzó párhuzamosan, egyszerre működik. A szorzók egyik bemenetére a pixelek, a másik bemenetére pedig a template értékek (T_A, T_B, T_C, T_A: baloldali, T_B: középső, T_C: jobboldali template érték) kerülnek. Ezeknek a szorzatai jelennek meg a szorzók kimenetein, amelyeket a következő szintnél összeadjuk. Az első iterációnál a negyedik operandus a "z" áramérték (ezt töltjük be az ACT regiszterbe), a következőkben pedig az előző iterációkban kiszámolt értékek lesznek. Először kiszámoljuk a 3*3-as ablakban lévő felső téglalapban a pixelek (és template-ek) szorzatainak az értékeit. Ezután a téglalapot eggyel lejjebb mozgatjuk (VS, vertical shifting). Ez látható az 5.5. ábrán az állapotgráfon [7].



5.4. ábra
A CASTLE
aritmetikai egysége

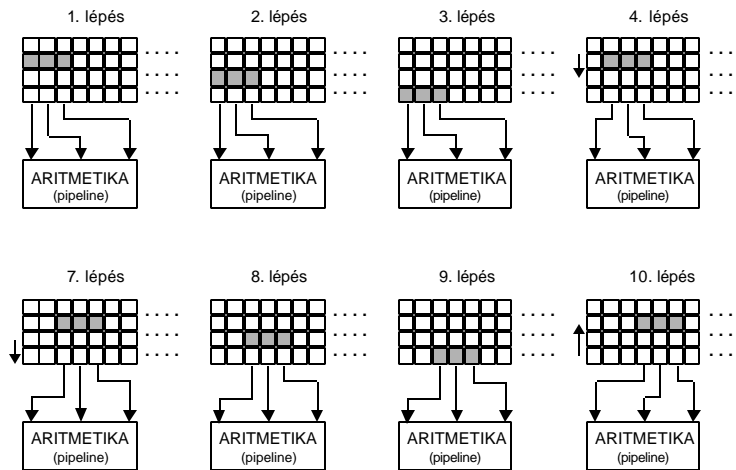


5.5 ábra
A CASTLE
aritmetikai egységének
vezérlési gráfja

Az aritmetikai egység tartalmaz még egy regisztersort is, aminek a master kimenete az ACC és a slave-é pedig az ACT. A CASTLE architektúra kétfázisú órajellel (ph1, ph2) működik.

Az 5.6. ábrán nyomon követhetjük a CASTLE architektúra képfeldolgozását [6], [7]. A legfelső sorban (0. sor) lévő pixelek nem vesznek részt a számításban, a szerepük csak az, hogy eloszór ebbe a sorba töltjük le a feldolgozandó pixeleket. Ebből a sorból töltjük le később az első sorba, onnan a másodikba, majd végül a harmadikba.

Az első lépésben az első sorban lévő pixelekkel számolunk (ezt mutatja a 5.4. ábrán lévő téglalap). A második lépésben a "téglalapot" egy sorral lejjebb visszük, és a következő sor pixeleit használjuk. A harmadik lépésben az utolsó sor pixelei alapján számoljuk a konvolúciót. A következő lépésben a téglalapot felvisszük az első sorba és egy pixellel jobbra mozdítjuk (5.6. ábra). Az imént leírt műveleteket itt is megismételjük.



5.6. ábra A feldolgozandó pixelek

Látható, hogy az "i" és az "i+1" helyu pixelek ismét felhasználásra kerülnek, igaz, hozzájuk más template értékek rendelődnek, hiszen az "aktív ablakot" (5.4. ábra) egy pixellel jobbra mozgattuk ($P_{i-1} * T_A$, $P_i * T_B$, $P_{i+1} * T_C$ helyett $P_i * T_A$, $P_{i+1} * T_B$, $P_{i+2} * T_C$ lesznek).

5.3 CASTLE processzor felépítése

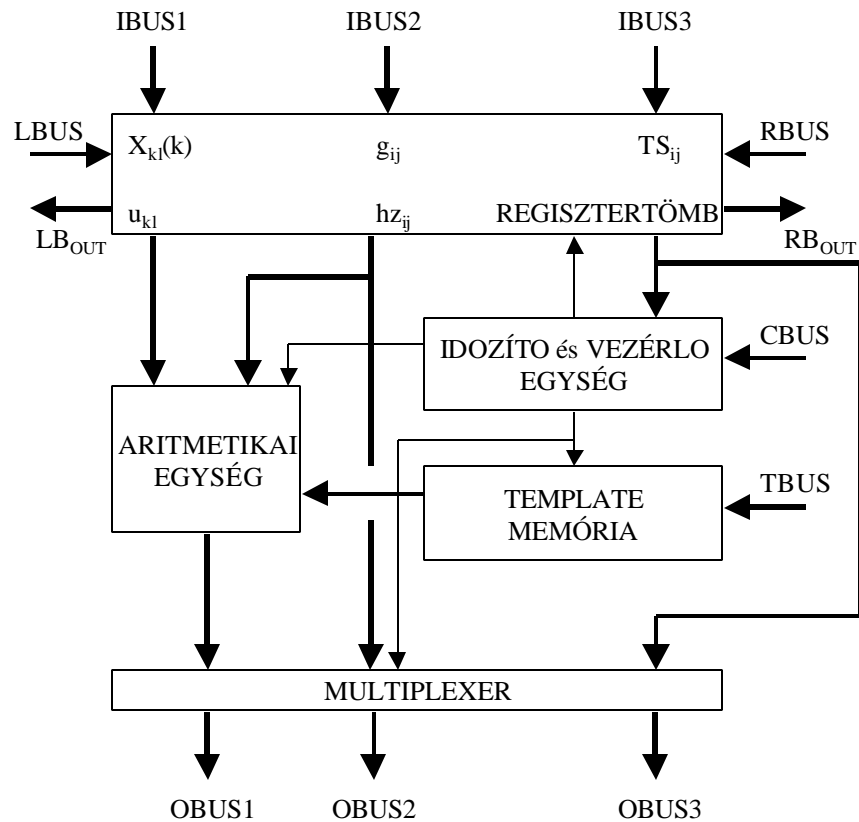
Az 5.7. ábrán látható a CASTLE architektúra [5], amely tartalmazza az aritmetikai magot, a regisztertömböt illetve a vezérlo, késlelteto egységet. A regisztertömbön belül három regisztersor található, ezeknek a bemenetei az IBUS1, IBUS2 és az IBUS3. Ezek a regiszterek FIFO típusú memóriákat valósítanak meg. Az IBUS1-en keresztül érkeznek a feldolgozandó kép pixeljei, az IBUS2-n az adott pixelkhez tartozó áramértékek. Az IBUS3 segítségével választhatjuk ki az adott pixelhez tartozó template-et. A CASTLE architektúrában maximum 16 template tárolható egyszerre a chipen. Az RBUS, LBUS, L_{OUT}, R_{OUT} jelek segítségével valósítható meg az oldalirányú kommunikáció, hiszen az 5.7. ábrán csak egy processzor látható. Ezekből a processzorokból építhető fel egy tömb.

A különböző vezérlo adatok a CBUS bemeneti buszon érkeznek, ezek határozzák meg a processzor működési üzemmódját.

Következo üzemmódok lehetnek:

- logikai processzor mód (a felbontás egy bit, *csak bináris képek használatához*)
- aritmetikai processzor mód
 - 6 bites felbontás
 - elojel nélküli
 - 2-es komplement
 - 12 bites felbontás
 - elojel nélküli
 - 2-es komplement

A TBUS bemeneti buszon tudjuk a chip-re betölteni a template-eket. Tekintettel arra, hogy minden egyes pixelhez külön template rendelhető (ez az architektúra egyik sajátossága), ezért a gyors, folyamatos működés feltétele, hogy több template is rendelkezésre álljon. A template memóriába 16 template tölthető.



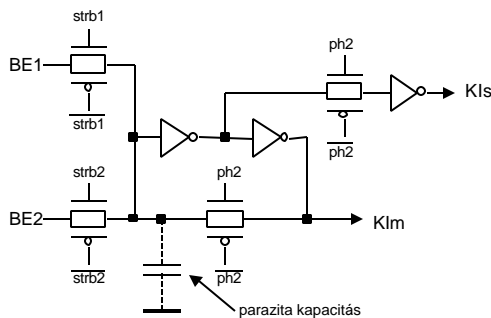
5.7. ábra A CASTLE architektúra

A regisztertömb három, független memóriaegységből áll. A legnagyobb memóriában tároljuk el a feldolgozandó képeket, a következő kettőben a számításokhoz szükséges áram- és template-ek kiválasztásához szükséges értékeket. Ezek a memóriák FIFO típusúak.

A CASTLE architektúra egyik fontos tulajdonsága a változtatható pontosság. A most tervezés alatt álló változat 12, 6 vagy 1 bites pontossággal rendelkezik. A 12 biteshez képest a 6 bites kétszer gyorsabb, míg az egybites felbontással használt processzor sebessége 12-szeres. A felbontási lehetőségeket a központi vezérlőegység választja ki.

5.4 CASTLE processzor megvalósítása szilíciumon

A CASTLE chip full-custom tervezéssel készült el [47]. Lényeges szempont volt a tervezés megkezdésekor, hogy a processzor minimális területet foglaljon el a szilíciumon. Az alapcellákat (pl.: dinamikus latch, különböző AND, OR, stb. kapuk) is az MTA-SZTAKI tervezőcsapata tervezte meg, így ezek a szilíciumon minimális méreteket foglalnak el, hiszen nem standard cellákat használtunk fel. A különböző regiszterek, latch-ek kétfázisú órajeleket használnak (ph1, ph2). Az architektúrát VHDL nyelven írták le [46]. Az aritmetikai egység különböző logikai kapukat tartalmaz, a regisztertömbök pedig szintvezérelt dinamikus master-slave típusú tárolókból épülnek fel. Ilyen latch-et láthatunk az 5.8. ábrán és annak VHDL kódját az 5.9. ábrán. Az "strb_{1,2,...,n}" vezérjelekkel engedélyezzük azokat az IN_{1,2,...,n} bemeneteket, amelyekkel feltöltjük a képen látható parazita kapacitást. Az "strb_{1,2,...,n}" vezérjeleket a "ph1" órajellel szinkronizáltuk. A "ph2" órajellel frissítjük a kapacitás tartalmát, illetve írjuk ki a slave (KIs) kimenetre a master (KIm) állapotát. Természetesen az "strb_{1,2,...,n}" jelek közül mindig csak egy lehet aktív, hiszen különben több bemenet egymást hajtaná meg a kétirányú transzferkapun át.



5.8 ábra

Szintvezérelt, két bemenetű MS dinamikus latch

```
KIm <= BE1 when (strb1 = '1')
      else BE2 when (strb2 = '1')
      else KIm when ph2 = '1'
      else anyvalue after 100 ns;
```

```
KIs <= KIm when (ph2='1')
      else anyvalue after 100 ns;
```

5.9 ábra

A szintvezérelt MS dinamikus latch VHDL leírása

Az 5.4. ábrán láthattuk a CASTLE aritmetikai egységének a felépítését, amelyet szorzók és összeadók alkotnak (a különböző határolóelemeket nem tüntettem fel a könnyebb érthetőség miatt). A továbbiakban ezeknek a moduloknak a működését, illetve a megvalósításukat fogom ismertetni. A felhasznált alapáramkörök, cellák késleltetése és a mérete az 5.1. táblázatban látható [6].

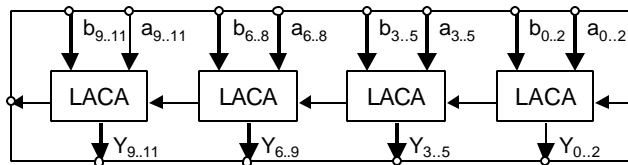
	késleltetés * [ns]	méret * [μm^2]
Regiszter	~ 0.5	256
Full Adder	~ 0.5	457
Párhuzamos összeadó (LACA)	~ 2	3476.16
Hat bites szorzó	~ 6	110120
Hat bites soros összeadó	~ 2	28188
Multiplexer (4->1)	~ 0.5	705

5.1. táblázat *Az alapáramkörök késleltetése, mérete (* 5 fan-out)*

A táblázatban szereplő értékek 5 fan-out-ra lettek meghatározva. Ez azt jelenti, hogy egy kimenet adott idő alatt 5 egységnyi bemenetet tud meghajtani. Természetesen több bemenet is meghajtható, de ez a késleltetési időt megnöveli.

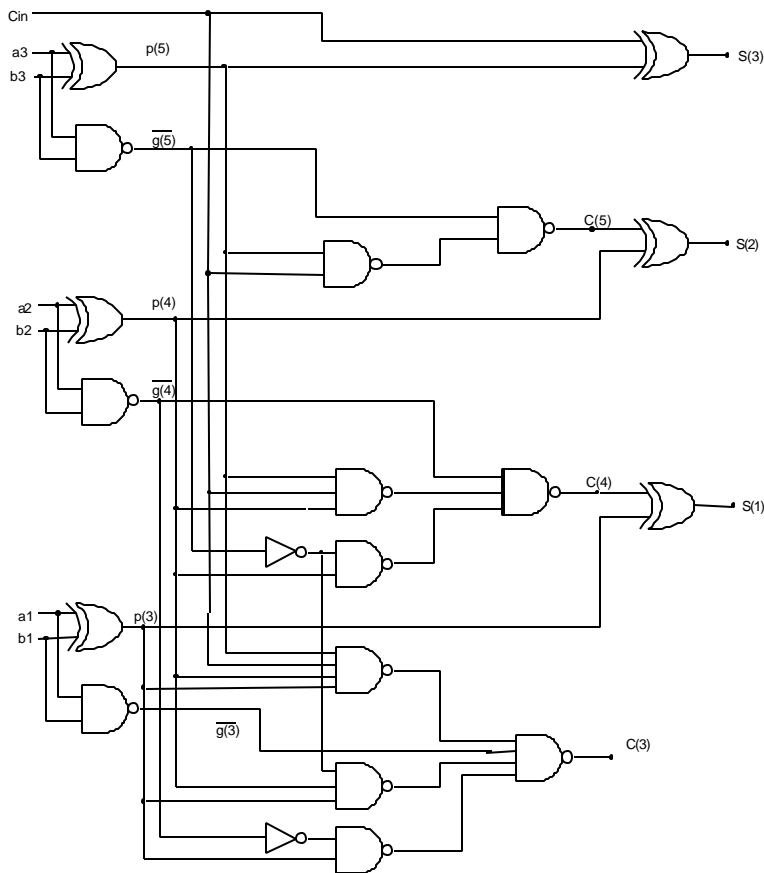
Az elkészített CASTLE processzornál a következő összeadó és szorozógységeket használtuk (5.10., 5.14. ábra).

Az összeadó az 5.10. ábrán látható. Négy darab, 3 bites párhuzamos (LACA) összeadóból áll. Azért nem Full-Adderbol építettük fel, mert ott csak soros carry átvitel van és ez csökkenti az összeadó sebességét.



5.10. ábra 12 bites összeadó

A 3 bites párhuzamos összeadó kapcsolási rajzát mutatja a 5.11. ábra [43]. Látható, hogy a "carry" párhuzamos képzésu.



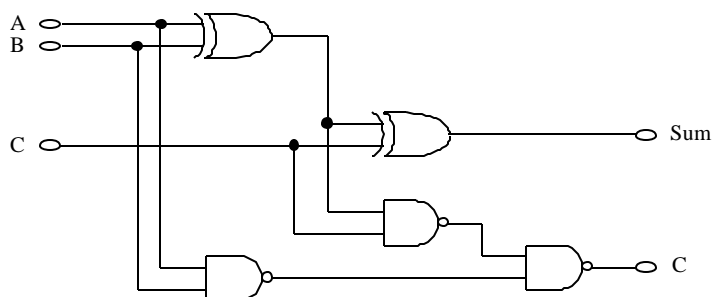
5.11. ábra A 3 bites "LACA" kapcsolási rajza

Az ábrán látható "A" és a "B" operandusokat ábrázolhatjuk a 5.5. egyenlet szerint. Ezeknek a szorzatát mutatja a 5.6. egyenlet [43].

$$A_V = -a_{m-1}2^{m-1} + \sum_{i=0}^{m-2} a_i 2^i \quad B_V = -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i \quad (5.5.)$$

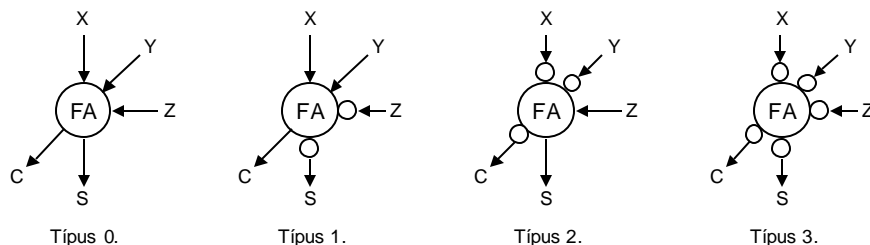
$$P_V = A_V B_V = -p_{m+n-1} 2^{m+n-1} + \sum_{i=0}^{m+n-2} p_i 2^i \quad (5.6.)$$

A szorzók full-adder-ekből könnyen elkészíthetők, szilíciumon kis helyen megvalósíthatók (5.12. ábra).



5.12. ábra A Full Adder kapcsolási rajza

A full-adder-eknek négy csoportja van, ezek láthatók a 5.13. ábrán. A belső felépítésük természetesen megegyezik, de vagy a bemeneten, vagy a kimeneten negálni kell a jeleket. Ez a művelet helytöbblettel jár, hiszen az inverter mérete egy nagyságrendbe esik az egész full-adder méretével.



5.13. ábra A Full Adder-ek fajtái, típusai

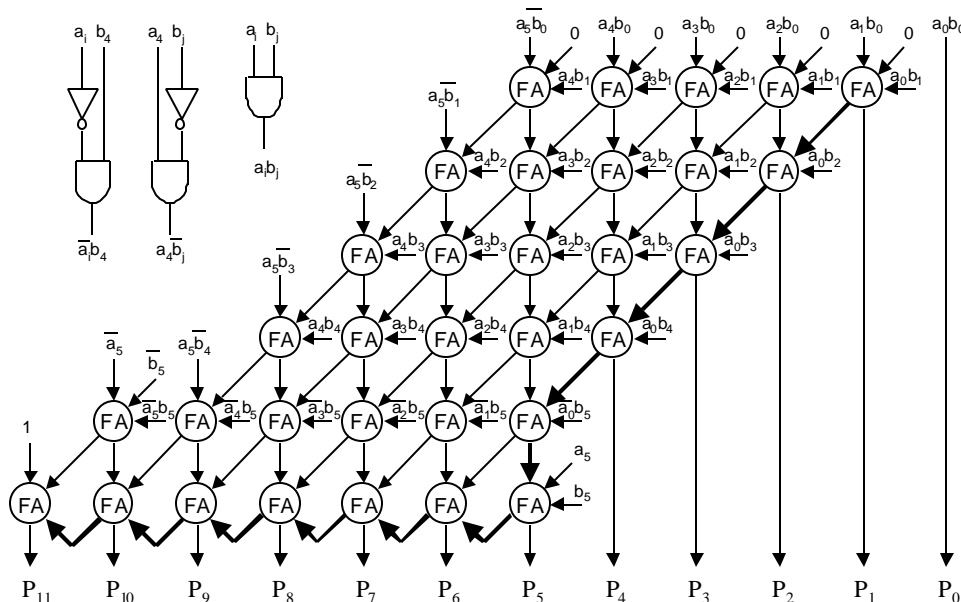
A legkönnyebb megvalósítást akkor kapjuk, ha tömb-szorzót használunk. További előnye ennek a típusnak a gyors működés. Több array szorzó létezik, a legismertebb négy változatot hasonlítottam össze egymással. Egy "n" és egy "m" méretű bináris szám összeszorzásához a következő futási idok tartoznak (5.2. táblázat) [43].

		Pezaris	Tri-Section	Bi-Section	Baugh-Wooley
Használt Full-Adder	Típus 0.	$(m-2)(n-2)$	$(m-1)(n-2)/2$	$(m-2)(n-1)$	$m(n-1)+3$
	Típus 1.	$m-2$	$(m-1)(n-2)/2$	0	0
	Típus 2.	$m+n-3$	$2(m-1)$	$2(m-1)$	0
	Típus 3.	1	0	0	0
Futási idő		$2(m+n)\Delta-2\Delta$	$2(m+n)\Delta-2\Delta$	$2(m+n)\Delta$	$2(m+n)\Delta$

5.2. táblázat A különböző array szorzók összehasonlítása a felhasznált full-adderek száma szerint

Ez a táblázat mutatja meg nekünk azt is, hogy egy meghatározott szorzó milyen típusú full-adder-ekből épül fel. Látható, hogy a "Baugh-Wooley" szorzónak nem a legkisebb a futási ideje, de ez az egység foglalja el a legkisebb területet a szilíciumon (hiszen a full-adderben nem használunk külön invertereket). Ezért esett a választás a "Baugh-Wooley" szorzóra.

A felhasznált "Baugh-Wooley" szorzóegység hat bites változata az 5.14. ábrán látható [43]. Ez a szorzótípus kettes komplementes és "array alapú". A tervezés során azért esett erre a választás, mert a tömb-szorzók a leggyorsabb működésűek és könnyen implementálhatók a szilíciumfelületre. Vastag vonallal jelöltem azt a leghosszabb utat, amit meg kell tenni egy szorzat kiszámításához. Ez az "út" hat bites felbontásnál 12 Full-Adder, 12 bites felbontásnál pedig 24. A 5.1. táblázat ismertetésénél már láthattuk, hogy egy Full-Addernek a késleltetési ideje legrosszabb esetben 0.5ns 0.35µm-es technológia használatánál. Ezért a hat bites szorzó késleltetési ideje legrosszabb esetben 6ns. Ez nagy szám, főleg, ha figyelembe vesszük, hogy az összeadók késleltetése sem elhanyagolható. Később látni fogjuk, hogy ez a késleltetési idő jelentősen csökkenhető (1ns alatt is elvégezhető a 6 bites szorzás).



5.14. ábra A 12 bites Baugh-Wooley szorzó

A szorzók full-adder-ekből könnyen elkészíthetők, szilíciumon kis helyen megvalósíthatók (5.14. ábra).

A későbbiekben bemutatok néhány optimalizált emulált digitális CNN-UM (CASTLE) aritmetikai egységet [2]. Ezeket fogom összehasonlítani az eredeti aritmetikai maggal (5.4. ábra) a felület, $A*T$ (felület*ido) és $A*T^2$ (felület*ido²) szorzatok szerint. Ezért most megadom azokat az egyenleteket, amelyeket a következő fejezetekben használni fogok.

A felületet (A_{EC} : eredeti CASTLE felülete) a következő egyenlettel (5.7) számolhatjuk ki. Tekintettel arra, hogy a három összeadó felülete elhanyagolható a szorzóéhoz képest, ezért az aritmetikai egység mérete jól közelíthető a szorzók felületével.

$$A_{EC} = (n^2 - n + 3) * 3 * 465, \quad (5.7)$$

ahol

"n" az aritmetikai egység felbontása (pl.: 6, 12 bit)
 "465" egy full-adder-nek a felülete μm^2 -ben

Az $A*T$ és az $A*T^2$ szorzatok úgy számolhatók ki, hogy az elobb bemutatott felületet megadó egyenletet megszorozzuk a "T" illetve a " T^2 " tagokkal. Az ebben a fejezetben bemutatott CASTLE aritmetikai egység működési sebességét egy egységnek (1) választottam, tehát a következő fejezetben ismertető optimalizált aritmetikai magokat ehhez viszonyítom.

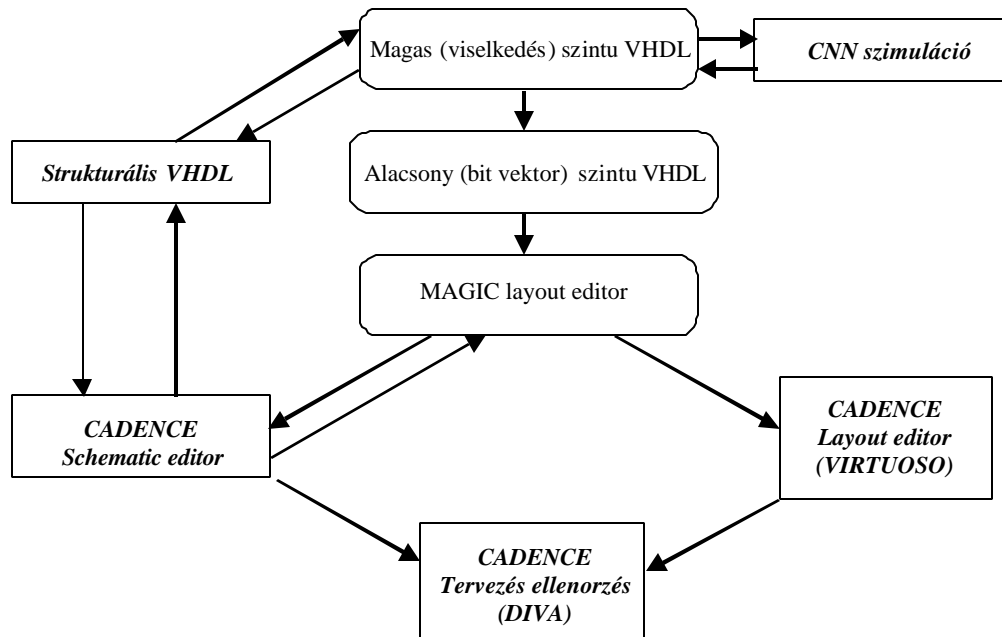
$$A_{EC} * T_{EC} = A_{EC} * T_{EC}^2 = \underbrace{(n^2 - n + 3) * 3 * 465}_{T_{EC} = 1} * T_{EREDETI} \quad (5.8)$$

5.5 CASTLE processzor tervezési metodikája

A CASTLE emulált digitális CNN-UM elkészítésekor a full-custom módszert használtuk. Ez azt jelenti, hogy az összes áramkörü modult (az alapcellákat is) mi terveztük meg.

A tervezés a magas szintű VHDL nyelv megadásával, elkészítésével kezdődött (5.15. ábra). A full-custom tervezés miatt az ellenőrzéseknek kulcsfontosságú szerepük van.

A layout tervezés során az AMS 3 fémrétegu, két polyszilíciumos, 0.35 μ technológiáját használtuk. A viselkedési leírásból készült VHDL specifikációból készült el a MAGIC layout tervezőszoftver [53] segítségével a processzor layout terve. Ezután használtuk a CADENCE tervezőszoftvert [54].



5.15. ábra A tervezés folyamatábrája

6. Optimalizált, kibovított CASTLE architektúrák

Ebben a fejezetben mutatom be azokat az eredményeimet, amelyeket az emulált digitális architektúra, a CASTLE architektúra optimalizálása során elértem, továbbá azt a megoldásomat is, amelynek a segítségével 3*3-as template-ek mellett az 5*5-ös template-ek is kezelhetők [1], [6]. Itt több aritmetikai magot is bemutatok [1], amelyeket működési sebesség és felület szerint optimalizáltam. Az eredeti CASTLE architektúra memória és vezérloegységei nem használhatók fel ezeknél az aritmetikai egységeknél, ezért itt mutatom majd be azokat a kép- és template memória módosításokat is, amelyek ahhoz szükségesek, hogy az újrakonfigurálható aritmetikai egységek használhatók legyenek.

Jelentősen megnöveltem a CASTLE architektúra működési sebességét [1], [7], ezáltal összemérhetővé vált az emulált digitális CNN-UM az analóg megoldással úgy, hogy a digitális CNN-UM pontossága változatlan maradt (tehát jobb, mint az analógé). Két sebességnövelési eljárásomat fogom ismertetni. Az első esetben a sebességnövekedés kb. kétszeres. A második esetben ez a növekedés tízszeres. Tekintettel arra, hogy ezt a működési sebességet nem tudjuk kiaknázni az eredeti CASTLE architektúrával, ezért egy teljesen új architektúrát adtam meg, amivel a működési frekvencia (~1GHz) kihasználható. A részletes VLSI megoldást természetesen nem ismertetem, hiszen csak az elv bemutatása volt a célom, nem a különböző fizikai megvalósítási kérdések megválaszolása.

A layouttervezőknek mindig nagy problémát jelent a szilíciumfelületre történő optimalizálás, minimális szilíciumfelületen biztosítani az eredeti működési feltételeket. Bemutatom azt a megoldásomat, amelynek segítségével az eredeti CASTLE architektúra szilíciummérete kb. harmadával csökkent [2], [6]. A méretcsökkenést nem a különböző aritmetikai modulok (szorzók, összeadók) szilíciumfelületen történő optimális elrendezésével valósítottam meg, hanem a CASTLE aritmetikai mag módosításával. Csökkentettem a szorzók számát, és ezáltal nem csak a felület csökkent, hanem a fogyasztás is. A szorzók számának a csökkenése bizonyos esetekben nem vonja maga után a működési sebesség csökkenését.

Ebben a fejezetben bemutatásra kerülő aritmetikai egységeket szilíciumfelületre optimalizáltam, módosítottam. Természetesen ezeket az aritmetikai magokat teszteltem VIRTEX FPGA-n [37] [38] is.

Az eredeti CASTLE architektúra működési egyenletei, amelyeket az 5. fejezetben ismertettem, a most bemutatásra kerülő, általam megadott aritmetikai egységekre is teljesül.

$$x_{ij}(n+1) = \sum_{C(kl) \in N_r(i,j)} \hat{A}_{ij,kl} * x_{kl}(n) + g_{ij} \quad (6.1.)$$

$$g_{ij} = \sum_{C(kl) \in N_r(i,j)} \hat{B}_{ij,kl} * u_{kl}(n) + h * z_{ij} \quad (6.2.)$$

Természetesen az ebben a fejezetben bemutatott megoldásaim "keverhetők", egymással kombinálhatók. Ismertetni fogok néhány ilyen architektúrát is 7. fejezetben.

6.1 Különböző architektúrák osztályozása, csoportosítása

Az áramköröket, architektúrákat különböző szempontok alapján optimalizálhatjuk. Ilyen szempontok a felület (A), disszipáció, működési sebesség (késleltetési idő, T), $A \cdot T$ és az $A \cdot T^2$ szorzatok. Természetesen ezek egymásnak ellentmondanak, hiszen ha például a működési sebességet növelni szeretnénk (pl.: pipeline megoldással), akkor növelem a felületet, a többlet áramkörök használata miatt pedig megno a disszipáció is.

Ma az elektronikában ezek a szempontok az irányadók, ezek szerint osztályozzák a különböző áramköri megoldásokat. Ezért én is ezeket a szempontokat tartottam szem előtt, amikor analizáltam az általam megadott aritmetikai egységeket.

6.2 Újrakonfigurálható aritmetikai egységek a CASTLE architektúrában

Az eredeti CASTLE emulált digitális CNN-UM architektúra és a különböző analóg CNN chip-ek csak egyes szomszédosságú template-eket (3×3 méretű) képesek kezelni. Ilyen template-eket láthatunk például a 6.29 és a 6.30. ábrán.

Vannak azonban olyan analogikai algoritmusok, feladatok, amelyeknek a megoldásához kettes szomszédosságú (5×5) vagy még nagyobb template kell [35], [60], [61], [62]. A mai analóg és digitálisan emulált CNN-UM chip-ek nem képesek kezelni 3×3 -asnál nagyobb template-eket. Ezért ebben a fejezetben ismertetett aritmetikai egységeket nem lehet összehasonlítani más által készített aritmetikai magokkal, tehát a most bemutatásra kerülő aritmetikai egységeknek mindenképp létjogosultsága van.

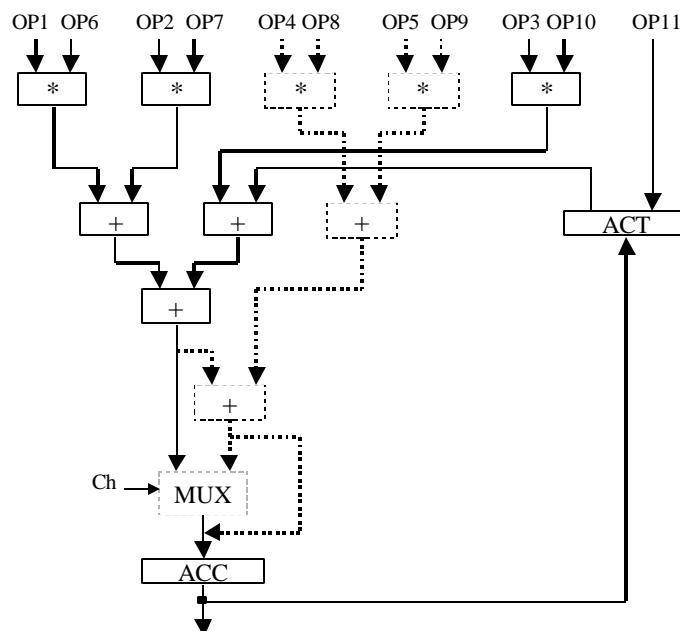
A megvalósítás során két megoldás közül választhatunk:

- template dekompozíció (nagyobb template-et felbontunk 3×3 -asokra [36]);
- módosítjuk az eredeti CASTLE architektúrát.

A template dekompozíció elve [36] ismert, használata nehézkes és a futási időt is jelentősen megnöveli. Ezért kidolgoztam egy teljesen új digitálisan emulált aritmetikai egységet [1], [6], [9] amelynek a segítségével egyes (3×3 -as) és kettes (5×5 -ös) szomszédosságú template-ek futtathatók. Ez az új megoldás optimalizálható a szilíciumfelület illetve a futási idő szerint is.

Nem csak az aritmetikát kellett megváltoztatnom, hiszen a chip-en a pixelmemóriában már 5×5 -ös "képet" dolgozunk fel egy 5×5 -ös template-tel. A memóriák kiegészítését és az új architektúra teljes blokkvázlatát a 6.2.4 alfejezetben ismertettem.

Az általános alapszintű megoldást láthatjuk a 6.1. ábrán az újrakonfigurálható emulált digitális CNN-UM rendszerekre [9], amely kiindulópont az optimális megoldásokénak.



6.1. ábra

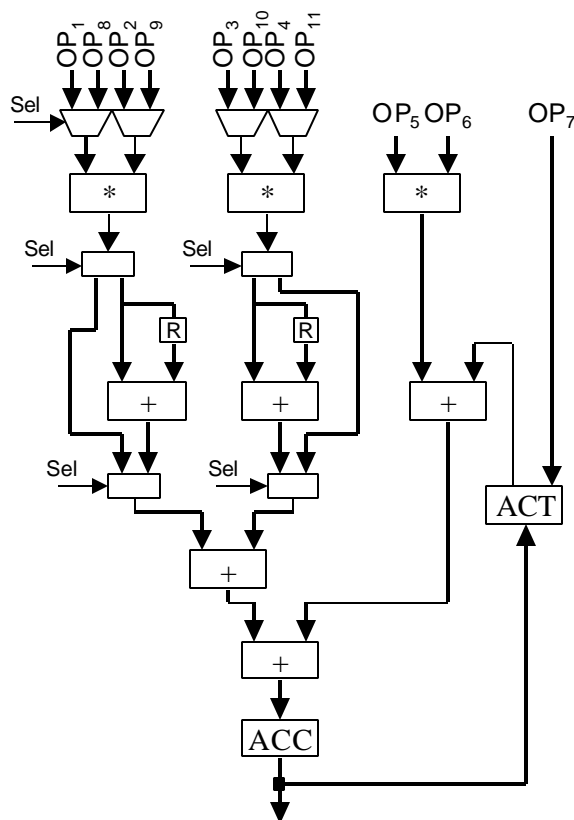
Általános megoldás az újraprogramozható CASTLE aritmetikai egységre

Kettes szomszédosságú (5*5-ös) template használatánál nem három, hanem öt szorzatot kell kiszámolnunk egy időegység alatt. Tekintettel arra, hogy a szorzók nagy helyet foglalnak el a szilíciumfelületen (5.12. ábra, 5. fejezet), ezért a 6.1. ábrán látható elrendezés nem javasolt.

Ezt a CASTLE aritmetikai egységet többféleképpen is átalakíthatjuk. Optimalizálhatjuk felület és működési sebesség szerint. A következő fejezetekben ezeket a megoldásokat fogom bemutatni.

6.2.1 Hely szerint optimalizált újraprogramozható aritmetikai egység

A 6.2. ábrán látható a minimális szilíciumfelület szempontjából optimális aritmetikai egység [1]. Az aritmetika legnagyobb egysége a szorzómodul. Ezért ez az újraprogramozható aritmetikai mag csak három szorzót tartalmaz. Mivel öt szorzatot kell előállítani három szorzóval, ezért ez a feladat csak két időegység (két órajel ciklus) alatt hajtható végre.



6.2. ábra

Szilíciumfelületre optimalizált újrakonfigurálható CASTLE aritmetikai egység

Amikor egyes szomszédosságú template-et használunk, akkor az aritmetikai mag működése megegyezik az eredeti CASTLE működésével. A "Sel" jel segítségével választhatjuk ki, hogy 3*3-as vagy 5*5-ös template-tel kívánunk-e dolgozni.

Ha kettes szomszédosságú template-tel dolgozunk, akkor először kiszámoljuk az $OP_1 * OP_2$, $OP_3 * OP_4$, $OP_5 * OP_6$ szorzatokat, és az első kettőnek az értékét eltároljuk a két "R" regiszterben. A következő lépésben kiszámoljuk az $OP_8 * OP_9$, $OP_{10} * OP_{11}$ értékeket is. Még ebben az iterációban összeadjuk az összeadókkal az öt szorzat értékét.

A "Sel" jellel, egy multiplexerrel választjuk ki az aritmetikai mag üzemmódját. A különböző OP_{1-11} operandusok megfeleltetése a 6.1. táblázatban található.

	OP ₁	OP ₂	OP ₃	OP ₄	OP ₅	OP ₆	OP ₇	OP ₈	OP ₉	OP ₁₀	OP ₁₁
3*3	P _{i-1}	T _A	P _i	T _B	P _{i+1}	T _C	Z	NC	NC	NC	NC
5*5	P _{i-2}	T _A	P _{i-1}	T _B	P _i	T _C	Z	P _{i+1}	T _D	P _{i+2}	T _E

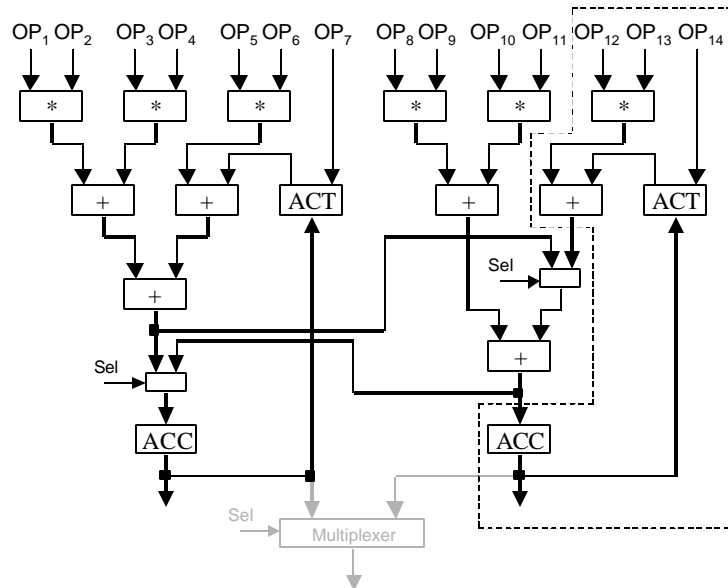
6.1. táblázat *Az OP_{1-11} megfeleltetése az aritmetikai egység bemeneteivel*

Egyes szomszédosságú template esetén az $OP_{8,9,10,11}$ bemeneteket nem használjuk, és ekkor a működési sebesség megegyezik az eredeti CASTLE aritmetikai egység (5.4. ábra, 5. fejezet) sebességével, tehát $V_{SPEED3*3} = V_{OCSPEED}$. Kettes szomszédos-

ságú template-ek esetén az öt szorzatot két időegység alatt számoljuk ki a három szorzóval, tehát a működési sebesség az eredeti aritmetikai egység sebességének a fele lesz ($V_{SPEED5*5} = 0.5 * V_{OCSPEED}$).

6.2.2 Sebesség szerint optimalizált újrakonfigurálható aritmetikai egység

Ebben a fejezetben azt a megoldást ismertetem, ahol a működési sebesség szerint optimalizáltam az újrakonfigurálható CASTLE aritmetikai egységet (6.3. ábra) [1].



6.3. ábra

Működési sebesség szerint optimalizált újrakonfigurálható CASTLE aritmetikai egység

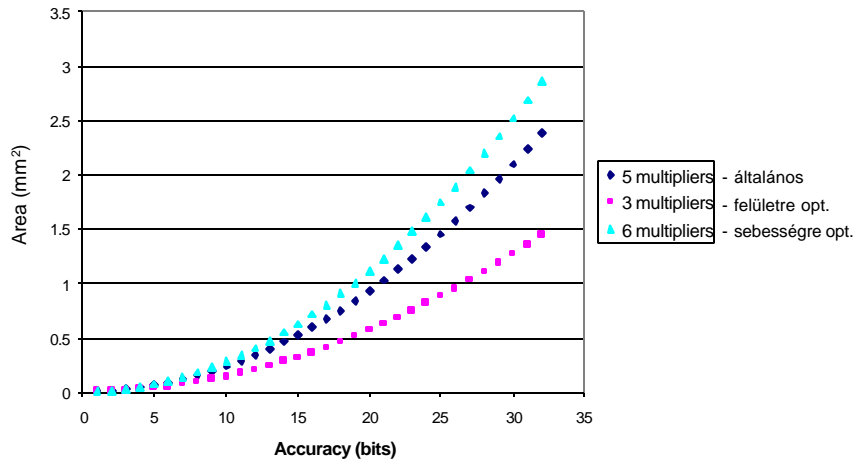
Könnyen észrevehető, hogy ez az elrendezés két CASTLE aritmetikai egységből áll. Ezért ha egy egyes szomszédosságú template-tel egy iteráció alatt két pixelt képes feldolgozni, akkor tehát a működési sebesség megkétszereződik ($V_{SPEED3*3} = 2 * V_{OCSPEED}$). Ha kettes szomszédosságú template-tel kell számolnunk, akkor a szaggatott vonallal bekeretezett áramköri elemeket nem használjuk, és az öt szorzatot öt szorzóval számoljuk ki. Tehát a működési sebesség ebben az esetben $V_{SPEED5*5} = V_{OCSPEED}$.

	OP ₁	OP ₂	OP ₃	OP ₄	OP ₅	OP ₆	OP ₇	OP ₈	OP ₉	OP ₁₀	OP ₁₁	OP ₁₂	OP ₁₃	OP ₁₄
3*3	P _{i-1}	T _A	P _i	T _B	P _{i+1}	T _C	Z	P _{i-3-1}	T _{2A}	P _{i+3}	T _{2B}	P _{i+3+1}	T _{2C}	Z
5*5	P _{i-2}	T _A	P _{i-1}	T _B	P _i	T _C	Z	P _{i+1}	T _D	P _{i+2}	T _E	NC	NC	NC

6.2. táblázat Az OP₁₋₁₄ megfeleltetése az aritmetikai egység bemeneteivel

6.2.3 A különböző aritmetikai egységek összehasonlítása

Az elobb bemutatott aritmetikai egységeket különböző tulajdonságok alapján hasonlítottam össze egymással [2]. A 6.4. ábrán az a grafikon látható, amely a felületfelbontás összefüggést adja meg. Legkisebb a felülete annak az egységnek, amely csak három szorzót tartalmaz.

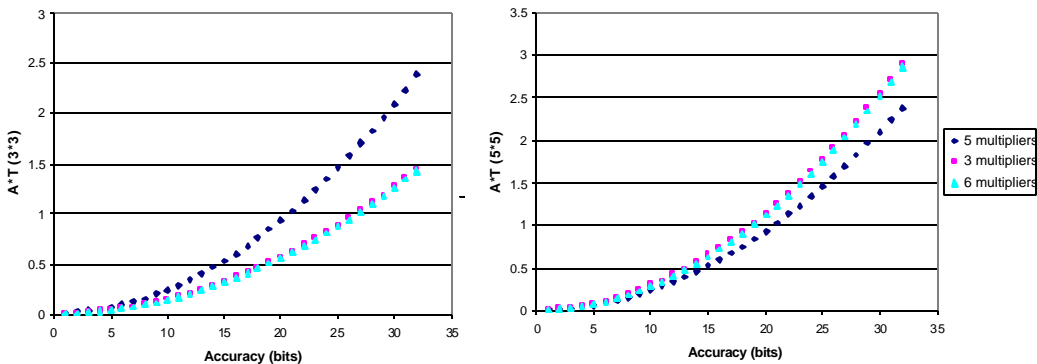


6.4. ábra
A három különböző újrakonfigurálható aritmetikai mag szilíciumfelületének összehasonlítása

A következő ábrákon a jelölések megegyeznek a 6.4. ábrán látható megjegyzésekkel, ezért a későbbiekben ezeket már nem tüntetem fel. Általános megoldásnak nevezem azt az aritmetikai egységet, amely öt szorzóegységből épül fel. A felületre optimalizált megoldás három, a sebességre optimalizált aritmetikai mag pedig hat szorzóból áll.

A következő két ábra (6.5., 6.6.) mutatja az A*T összefüggést 3*3-as és 5*5-ös template-ek használata esetén. Látható, hogy az idő szerint optimalizált (6 szorzós) CASTLE aritmetikai egység nagyon kedvező 3*3-as template használata esetén, viszont 5*5-ös template-nél már azt az aritmetikai egységet célszerű használni, amely öt szorzómodulból áll.

A 6.5 és a 6.6 ábrákon látható két grafikon (három és a hat szorzós megvalósítások) egybeesik.



6.5. ábra

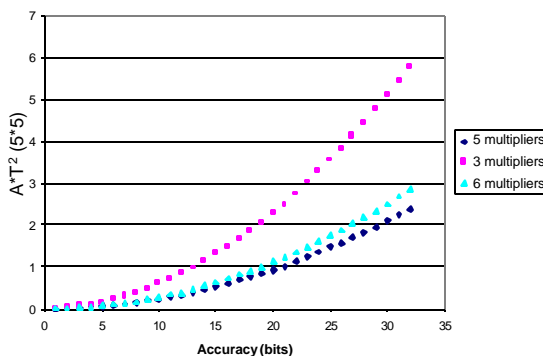
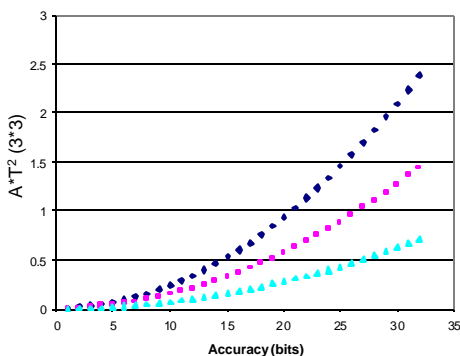
A három különböző aritmetikai egység összehasonlítása $A*T$ alapján, $3*3$ -as template esetén

6.6. ábra

A három különböző aritmetikai egység összehasonlítása $A*T$ alapján, $5*5$ -as template esetén

A 6.7. és a 6.8. ábra az $A*T^2$ összefüggést adja meg. Ha $3*3$ -as template-et használunk, akkor a legjobb megoldás a sebesség szerint optimalizált aritmetikai egység. Ez hat szorzóból áll és egy időegység alatt két $3*3$ -as pixelterületet számol ki.

Kettes szomszédosságú template-eknél az eredeti újrakonfigurálható aritmetikai magot, az öt szorzóból álló egységet célszerű alkalmazni.



6.7. ábra

A három különböző aritmetikai egység összehasonlítása $A*T^2$ alapján, $3*3$ -as template esetén

6.8. ábra

A három különböző aritmetikai egység összehasonlítása $A*T^2$ alapján, $5*5$ -ös template esetén

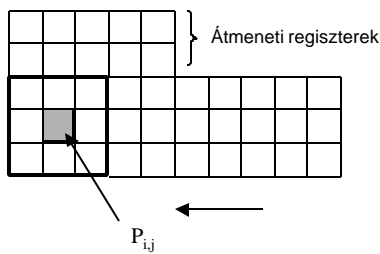
Az analogikai algoritmusok optimális futásához különböző aritmetikai egységek kellenek. Egy konkrét ASIC-es megvalósításhoz ajánlható a hat szorzós aritmetikai egység, ennek a magnak a legnagyobb a működési sebessége. Igaz, ez igényli a legnagyobb felületet a szilíciumon, de az aritmetikai egysége két, egymástól függetleníthető aritmetikai egységből áll. Tekintettel arra, hogy a CNN az egy processzortömb, ezért ez az aritmetikai megoldás felfogható $2*1$ -es tömbnek is, ha $3*3$ -as template-et használunk. Látható tehát az, ha ezt a megoldást választjuk, akkor a CNN tömbmérete template függo lesz.

6.2.4 Az újrakonfigurálható architektúrák memóriaszervezése, vezérlése

Az elobb bemutatott újrakonfigurálható aritmetikai egységek önmagukban nem használhatók, hiszen a pixel- és a template memóriát $3*3$ -as template-ekre tervezték [5]. Ezért ezeket a tárolóegységeket ki kellett egészítenem. Fontos szempont volt, hogy olyan megoldást találjak, amelynek az alkalmazása nem okoz lényeges változtatást a memóriák felépítésében.

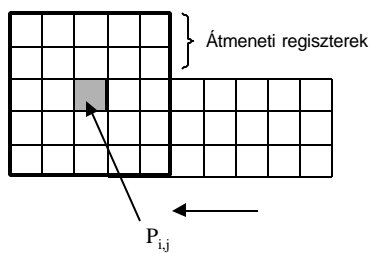
Az 5. fejezetben bemutatott pixelmemóriát egy sorral (öt regiszterrel) ki kellett egészítenem (legfelső sor), és az addig csak a pixelek betöltésére használt, ún. "ARL0" sor utolsó öt regiszterének a szerepe is megváltozott [1]. Az "átmeneti

regiszterek"-ben tároljuk el az 5*5-ös template használatához szükséges pixeleket (6.9. és 6.10. ábra).



6.9. ábra

*Pixelmemória 3*3-as template-nél*



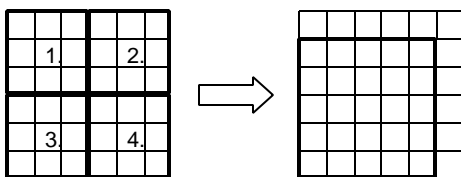
6.10. ábra

*Pixelmemória 5*5-as template-nél*

Az ábrákból kitunik, hogy csak öt regiszterre volt szükségünk ahhoz, hogy a pixelmemória alkalmas legyen 5*5-ös template-ek tárolására is. Ennek azonban az az ára, hogy amikor kettes szomszédosságú template-eket akarunk használni, akkor töltjük be az IBUS1 buszon keresztül az 5*5-ös pixelterületet.

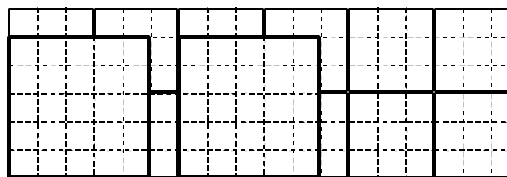
A megvalósított eredeti CASTLE architektúrában 16 template tárolható el egyszerre. Ha négy darab 3*3-as memóriaszeletet egy darabként kezelünk, akkor ebben a memóriaegységben négy egyes vagy egy kettes szomszédosságú template tárolható (6.11. ábra).

Ennek az elrendezésnek nagy elonye, hogy ezek az egységek kaszkádosíthatók, több 5*5-ös template memória is kialakítható (6.12. ábra).



6.11. ábra

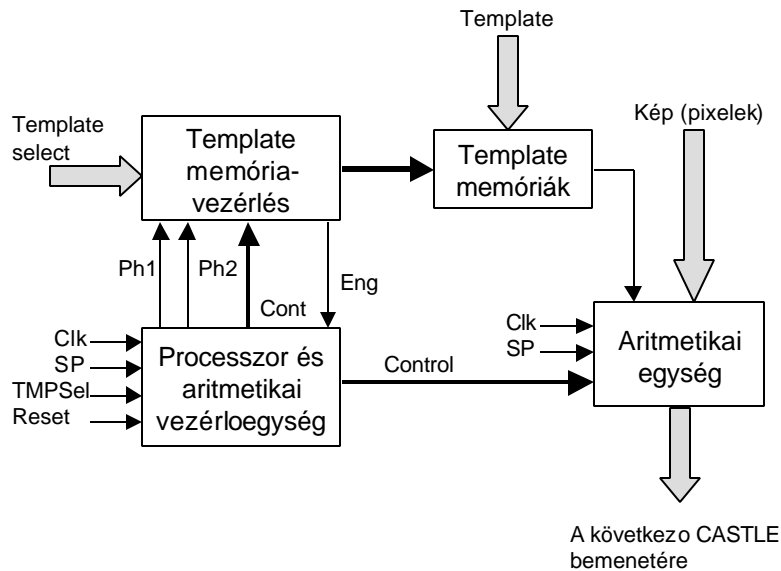
*Templatememória kialakítása
3*3-as vagy 5*5-ös template-eknek*



6.12. ábra

Templatememóriák kaszkádosítása

Az újrakonfigurálható aritmetikai egységek és a memóriák vezérlését mutatja a 6.13. ábra. A "TMPSel" és az "SP" jelek a vezérloegységre csatlakoznak. Ezeknek a jeleknek az állapota szerint vezéreljük a template- és a pixelmemóriát.



6.13. ábra A módosított CASTLE architektúra

Az eredeti CASTLE digitálisan emulált CNN-UM architektúrát [5] összehasonlítottam a fizikai jellemzők alapján az általam megadott újrakonfigurálható aritmetikai egységekkel [2]. Ezt mutatja a 6.3. táblázat.

	Eredeti CASTLE	Újrakonf. 3 szorzós, felületre optimalizált	Újrakonf. 5 szorzós, általános megoldás	Újrakonf. 6 szorzós, sebesség szerint optimalizált
Késleltetési formula	$t_{szorzó} + 3 \cdot t_{\text{ö.adó}}$	$t_{szorzó} + 3 \cdot t_{\text{ö.adó}} + 2 \cdot t_{\text{mux/dem}}$	$t_{szorzó} + 2 \cdot t_{\text{ö.adó}} / t_{szorzó} + 3 \cdot t_{\text{ö.adó}} + t_{\text{mux}}$	$t_{szorzó} + 2 \cdot t_{\text{ö.adó}} + t_{\text{demux}} + t_{szorzó} + 3 \cdot t_{\text{ö.adó}} + 2 \cdot t_{\text{demux}}$
Késleltetés [ns]	~ 10	~ 13	~ 10/12	~ 10/13
Szilíciumméret [mm ²] (Számábrázolási pontosság: 12 bit)	0.596	0.6	0.816	1.2
Max. órajel (VIRTEX300)	46 MHz	34.406 MHz	56.189 MHz	33.95 MHz
Max. órajel (szilíciumon, 0.35µm)	~ 100 MHz	~ 80 MHz (r=1,2)	~ 100 MHz (r=1) ~ 80 MHz (r=2)	~ 100 MHz (r=1) ~ 80 MHz (r=2)
Disszipáció (W)	< 0.3	< 0.3	< 0.3	< 0.5

6.3. táblázat Az újrakonfigurálható aritmetikai egységek összehasonlítása fizikai jellemzők alapján

A táblázatból kitűnik, hogy a késleltetés jelentősen függhet a felhasznált template méretétől (általános és a sebességre optimalizált változatoknál). Ezeket az aritmetikai egységeket teszteltem a XILINX cég által elkészített VIRTEX 300 FPGA-n. Ezek az értékek is szerepelnek a táblázatban.

6.2.5 Az újrakonfigurálható architektúrák használata

Nagyon sok olyan feladat van, amely csak kettes vagy nagyobb szomszédosságú template-ekkel oldható meg. Olyan analóg CNN-UM chip-ek még nem készültek, amelyek ezt a feladatot képesek lennének megoldani. Ezidáig ilyen feladatot főleg számítógépen futó szimulátorral lehetett megoldani (Természetesen template dekompozícióval fel lehet bontani nagyobb méretű template-et, de ez időigényes és nem mindig ad robusztus template-eket).

6.3 Pipeline egy digitálisan emulált CNN-UM (CASTLE) aritmetikában

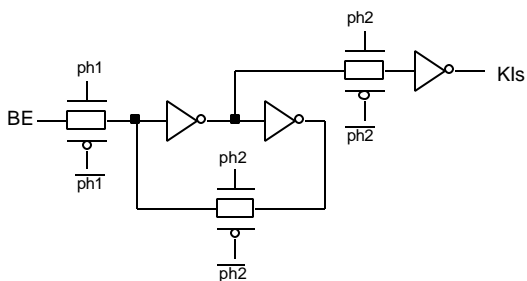
Az eredeti CASTLE architektúra működési sebességét lehatárolja az aritmetika működési sebessége, késleltetése. 12 bites felbontás esetén a szorzó késleltetése 12ns, az összeadóké pedig 6-6ns. Így az összkésleltetés 24ns [5], ha a kapukésleltetés 5 fan-out-ra nézve 0.5ns. Hat bites felbontás esetén a késleltetés értéke lecsökken a felére. A következő fejezetekben két pipeline megoldásomat fogom bemutatni, amelyeknek a segítségével a működési sebesség jelentősen növelhető. Látni fogjuk majd, hogy nem elég az aritmetikai modulok (szorzó, összeadó) közé átmeneti regisztereket tenni, hanem módosítani kell a pixel-, áram- és a template-kiválasztó memóriák vezérlését is. A megnövelt sebességet nem tudjuk kihasználni az eredeti architektúrával, ezért a fejezet végén ismertetek egy lehetséges elrendezést is, amivel az új sebességérték (1GHz) kihasználható [1], [6], [7].

Eloszor ismertetem azt a megoldásomat, amikor csak a fobb modulok (szorzó, összeadó) közé tettem átmeneti regisztereket. Azután bemutatom azt a sebességnövelő eljárásomat, aminek a segítségével az eredeti digitálisan emulált CNN-UM sebessége jelentősen megnőtt, az eredeti érték tízszerese lett.

6.3.1 Pipeline az aritmetikai modulok között

A késleltetési érték jelentősen csökkenthető, ha az eredeti CASTLE aritmetikában (5.4. ábra) a szorzó és az összeadók közé átmeneti tárolókat teszünk. Ilyen átmeneti, dinamikus szintvezérelt master-slave tárolót láthatunk a 6.14. ábrán, valamint ennek a VHDL kódját a 6.15 ábrán.

Ennek a latch-nak a használatával jelentős sebességnövekedést érünk el.



6.14. ábra

Szintvezérelt, egy bemenetű MS dinamikus latch

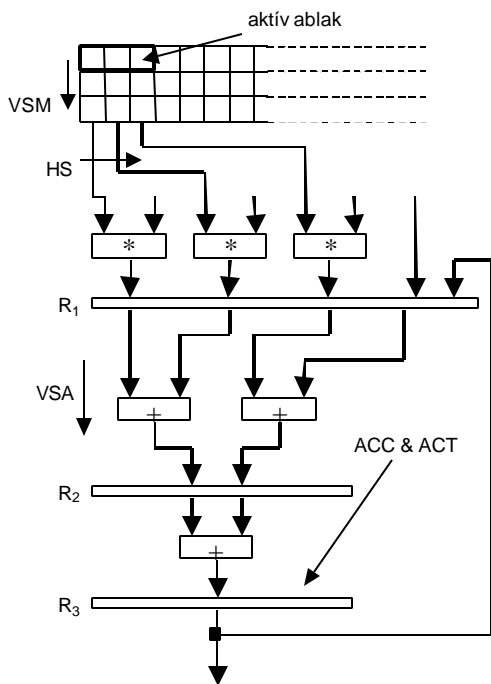
$K_{lm} \leq BE$ when (ph1 = '1')
 else K_{lm} when ph2 = '1'
 else anyvalue after 100 ns;

$K_{ls} \leq K_{lm}$ when (ph2='1')
 else anyvalue after 100 ns;

6.15. ábra

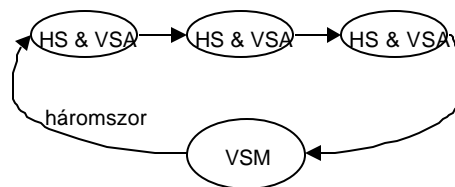
A szintvezérelt MS dinamikus latch VHDL leírása

Az R_1 , R_2 és R_3 regiszterek szintén master-slave típusúak és ilyen latch-ekből épülnek fel. Az eredeti CASTLE aritmetikai egységben található "ACC" és "ACT" regisztereket egybevontuk az R_3 -as regisztersorral (6.16 ábra). Esetünkben a maximális késleltetés (tehát az összkésleltetés, az aritmetika késleltetése) lecsökken, megfelel egy szorzó késleltetésének (legrosszabb esetben 12ns, 12 bites felbontásnál), hiszen a pipeline használatánál a maximális működési sebességet a legnagyobb késleltetéssel rendelkező modul (a szorzó) határozza meg.



6.16. ábra

Pipeline-tartalmazó aritmetikai egység



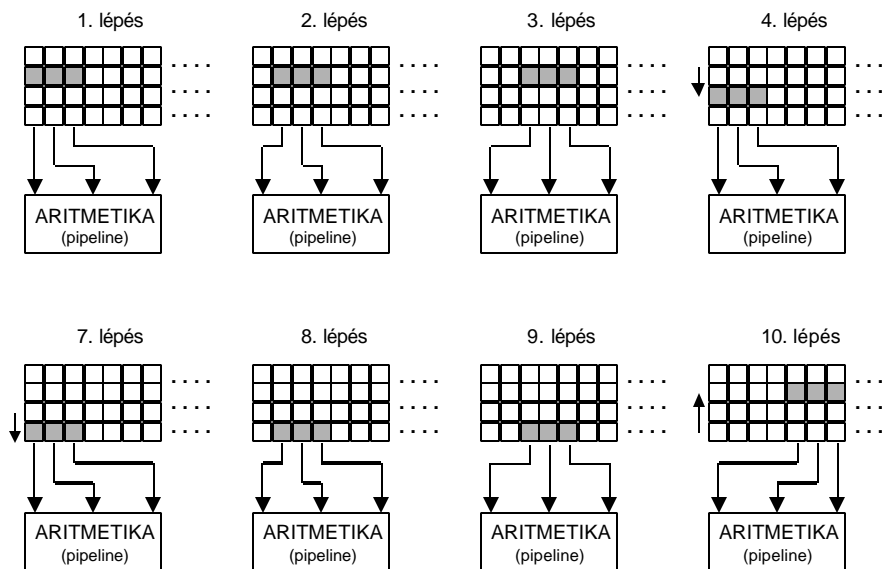
6.17. ábra

Pipeline-t tartalmazó aritmetikai egység vezérlő gráfja

Az 5. fejezetben mutattam be az eredeti CASTLE architektúra működését, ezért erre most nem térek ki, csak a pipeline-nal rendelkező aritmetikai egységet ismertetem.

A 3*1-es "ablak"-ot másképpen kell vezérelni, mint az eredeti CASTLE megoldásnál. Az eredeti változatnál három függőleges shiftelés (lépés) után léptettük jobbra egy pixelnyit a téglalapot (5.6 ábra). Ha így vezérelnénk a pipeline-t tartalmazó aritmetikát, akkor a pipeline-nal járó sebességnövekedést nem tudnánk kihasználni, hiszen a középső pixelhez tartozó értéket három lépésben számoljuk ki. Ezért ebben a fejezetben egy három szintű pipeline használatát mutatom be, azt követően pedig az általános megoldást fogom ismertetni.

A három szintű pipeline-t tartalmazó aritmetikát másképpen kell tehát vezérelni, mint azt az eredeti emulált digitális megoldásnál láthattuk (5. fejezet, 5.5. ábra). Először (1. iteráció) kiszámoljuk a template és a képmemóriában a felső sor három értékének a szorzatát ($P_{i-1,j-1} * T_A$, $P_{i,j-1} * T_B$, $P_{i+1,j-1} * T_C$), és ezeket a szorzatokat elmentjük az R_1 -es regisztersorba. Ezután a téglalapot egy pixelrel jobbra toljuk a képmemóriában. Itt is kiszámoljuk a három pixel értékét, és ezeket a szorzatösszegeket is elmentjük az R_1 regisztersor slave részébe, a master részt pedig az R_2 -be. A harmadik ütemben (a harmadik "ph1" órajel felfutásakor) ismét jobbra toljuk egy pixelrel az aktív téglalapot. Amikor itt is kiszámoljuk a különböző szorzatokat a kép- és a template memória megfelelő értékeiből, akkor az R_2 regiszterek állapotát elmentjük az R_3 -ba, az R_1 -ét pedig az R_2 -be. A kiszámolt szorzatokat az R_1 -ben tároljuk el. Mivel az R_{1-3} tárolóregiszterek master-slave típusúak, ezért ezek a lépések és a szorzatok előállításuk egy-egy iteráció alatt zajlanak le. A negyedik iterációban a téglalapot visszahelyezzük a kiindulási helyére és eggyel lejjebb tesszük (j. sor, 6.18. ábra). Ekkor ismét megismételjük a fentebb leírt folyamatot, kiszámoljuk a középső sorban található pixelek és template-ek értékeit ($P_{i,j} * T_A$, $P_{i,j} * T_B$, $P_{i+1,j} * T_C$), jobbra toljuk az ablakot és a számolást megismételjük. Ezek a lépések láthatók a 6.18. ábrán.



6.18. ábra Az új pixelmemória szervezése

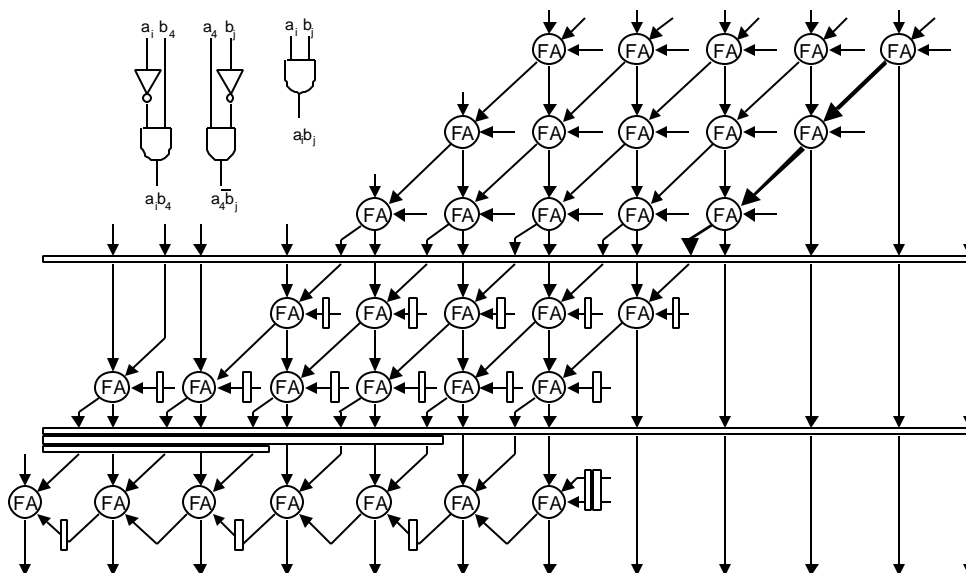
Az első eredményt a 7. iteráció végén kapjuk meg, az utána következő két iterációban (8.,9.) pedig a másodikat és a harmadikat is. A tizedik lépésben újra kezdődik az előbb leírt folyamat.

Ha ezt a pipeline megoldást használjuk, akkor a legnagyobb késleltetési érték

kiszámolt adatok bekerülnek az "R₂", "R₃" regiszterekbe, illetve ugyanígy kiszámoljuk az "a" és a "b" operandusok felső bitjei is. Mivel a felső tároló sorok (R₁₋₃) száma megegyezik az alsó tároló sorok (R₁₋₃) számával, ezért a négyszer három bites kimeneti érték (Y_{0..2}, Y_{3..5}, Y_{6..8}, Y_{9..11}) egyszerre jelenik meg az összeadó kimenetén.

Az eredeti Baugh-Wooley szorzó késleltetését azok a Full-Adder-ek összkésleltetése határozza meg, amelyeknél a legnagyobb a "carry" átvitel (6 bites felbontásnál 12 Full-Adder, 12 bites felbontásnál 24 Full-Adder). Az aritmetikai egységben ez a szorzóegység befolyásolja legjobban az összkésleltetést, mert ezen egységen belül leghosszabb a jelterjedési idő (emlékeztetőül: 6 bites felbontás esetén az összeadónak 2 ns a jelterjedési késleltetése). Ezért ennek a késleltetési értéknek a lecsökkentése kulcskérdés. A szorzóegységben a pipeline szintek számát, elhelyezkedését a 6.20. ábrán látható módon határoztam meg [1]. Tekintettel arra, hogy a tárolóelemek késleltetése egy nagyságrendbe esik a full-adder-ek késleltetésével, ezért csak minden második full-adder után célszerű betenni az átmeneti regisztereket.

Látható, hogy a legnagyobb carry átvitel 12-ről 3-ra lecsökkent (vastag vonal).

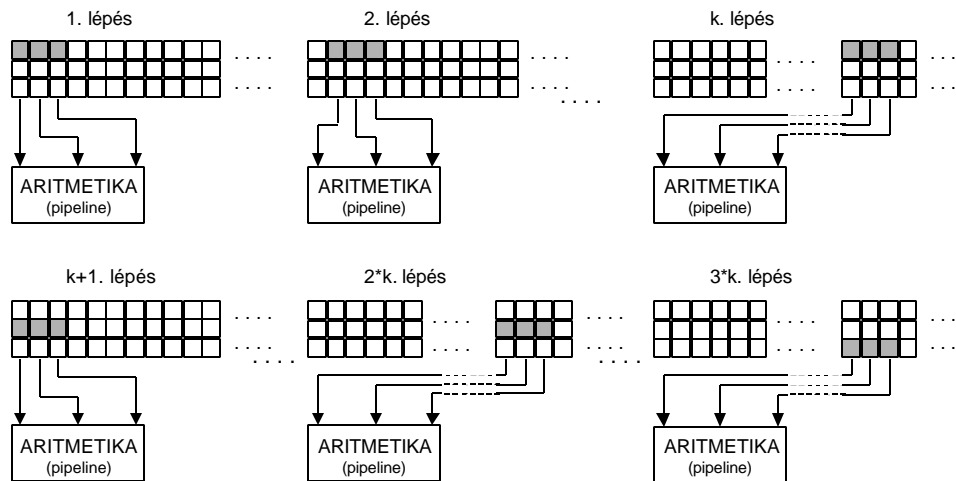


6.20. ábra *Módosított Baugh-Wooley szorzó*

Az eredeti CASTLE aritmetikában maximum 13 szintű pipeline használható. A szorzóban négy tároló sor helyezhető el, az összeadóknak pedig három (4+3+3+R₁+R₂+R₃ = 13). Ugyanakkor a 13 tároló szint kialakítása felesleges, mert a modulok közötti R₁₋₃ regiszterek olyan egységeket kötnek össze, aminek a kimenetén vagy van tároló sor (összeadó, 5.10. ábra) vagy csak egy full-adder-nyi késleltetés van (szorzó, 5.14. ábra). Az R₁₋₃ regiszterek elhagyása miatt be kell tenni ACC és ACT regisztereket, amelyek az eredeti CASTLE aritmetikában láthatók (5.8. ábra). Természetesen ezek a regiszterek megtalálhatók voltak az előbb ismertetett megoldásnál is, de akkor egybevettem az ábrázolásnál az ACC, ACT regisztereket a tároló regiszterekkel.

Észrevehetjük tehát, hogy a CASTLE aritmetikában csak maximum 10 szintű pipeline alkalmazható (R₁₋₃ - et elhagyjuk). A 6.21. ábrán látható memóriavezérlést

általános esetben adtam meg, a "k" (k: pipeline szintek száma) értéke maximum 13 lehet, ha a template mérete 3*3. A 6.1. fejezetben olyan aritmetikai egységeket mutattam be, amelyek működés közben átkapcsolhatók 3*3-as vagy 5*5-ös template-üzemmódra. Látni fogjuk majd azt is, hogy 5*5-ös template-eknél a "k" értéke nagyobb lesz.



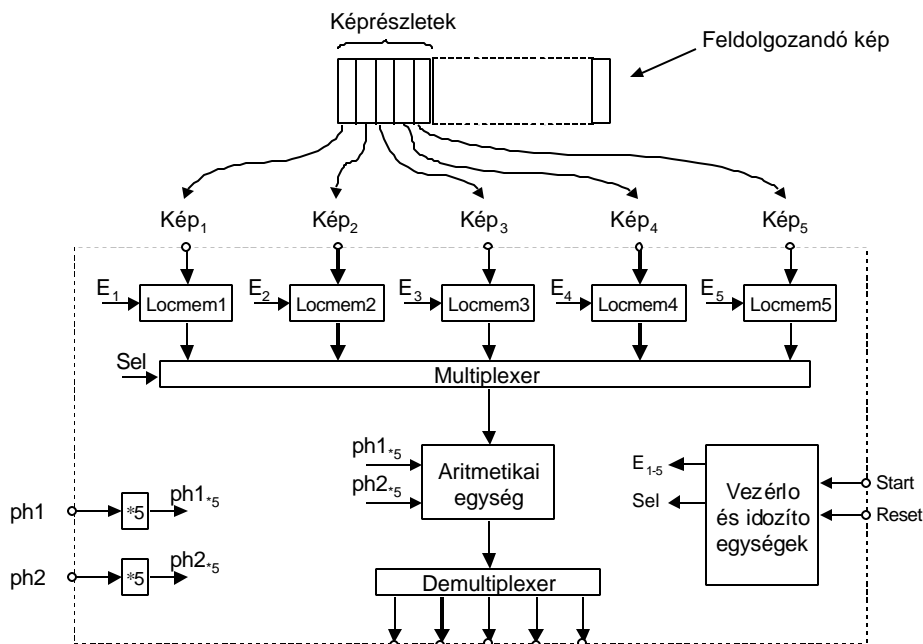
6.21. ábra "k" szintu pipeline memóriaszervezése

Észrevehető tehát az, hogy ha az összeadóknak és a szorzóknak belül is vannak regiszterek, akkor a működési sebesség jelentősen megnövekszik. A legnagyobb késleltetés három Full-Adder késleltetési idejének felel meg. Ez 0.35 μm -es technológián max. 1.5ns (3*500ps, worst case, 5 fan-out). A három Full-Adder késleltetési idő a szorzó bemenete miatt van (6.20 ábra).

6.3.3 Az új CASTLE architektúra felhasználása

A pipeline-nal kibővített CASTLE architektúrának a használhatósága nagyon behatárolt, hiszen a működési frekvencia maximális értéke kb. 1 GHz (6 bites felbontás mellett, ez a legnagyobb késleltetési értékből adódik), ilyen sebesség mellett a képletöltés nem lehetséges. Két megoldás közül választhatunk. Vagy lecsökkentjük a működési sebességet (például nem használunk pipeline-t a modulokon belül) vagy a CASTLE architektúrát kiegészítjük különböző segédáramkörökkel.

Megadtam egy új processzortömb elrendezést [1], [2], ami mellett a maximális működési frekvencia kihasználható (6.22. ábra). Egy aritmetikai egységhez nem egy lokális memória tartozik, hanem öt. Az itt felhasznált aritmetikai egység nem az eredeti elrendezés (5.4. ábra), hanem a tárolóregiszterekkel kiegészített megoldás. Az aritmetikai mag mindig csak egy memóriából (LocmemX) kapja az adatot, a másik négyet ez idő alatt adatokkal töltjük fel. A letöltendő képet feldaraboljuk öt részre. Ezt az öt részt különböző időpontokban töltjük le a chipre, a szilíciumfelületen található lokális memóriákba. Tehát egyszerre mindig négy lokális memóriába töltjük le kívülről a képrészleteket, és az ötödik memóriában található adatokat továbbítjuk az aritmetikai egység felé. Hasonlóan olvassuk ki az eredményeket is a processzorból.

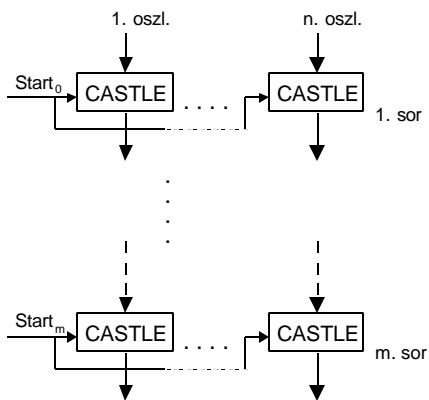


6.22. ábra A módosított CASTLE architektúra

Látható tehát az, hogy az 1GHz-es sebesség könnyen kihasználható, mert az I/O muveletek "csak" 200MHz-el történnek.

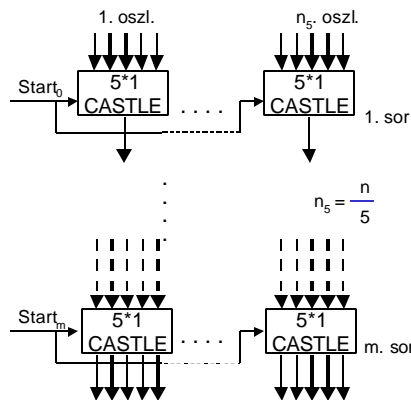
Könnyen észrevehető, hogy a 6.22. ábrán öt CASTLE processzor van egymás mellett, csak az aritmetikai magjuk és a vezérlőegységük közös. Tehát ez a módosított CASTLE architektúra megfelel egy 5*1-es processzortömbnek is. A módosított CASTLE architektúra (6.22. ábra) konkrét fizikai megvalósításával nem foglalkoztam, így az órajelelosztás problémáját sem részletezem.

Az eredeti processzortömb (n*m) egyszerűsített rajza a 6.23. ábrán látható (A teljes tömbrol a 3. fejezetben írtam). A pipeline-osított processzorokból felépített tömb a 6.24. ábrán látható. Az "n" oszlop helyett csak n/5 van. A sorok száma ("m") nem változik. Látható tehát az, hogy a pipeline használatával nem csak a sebesség nőtt meg jelentősen (tízszerezésre), hanem csökkent a felhasznált szilícium felülete is.



6.23. ábra

Az eredeti CASTLE processzortömb



6.24. ábra

Az új CASTLE (5*1) processzortömb

Az igaz, hogy az ötödére csökkent az oszlopszám, de a processzornak megnőtt a felülete. Ez a felületnövekedés viszont nem ötszörös, hiszen nem öt független aritmetikai egységet használunk, hanem egyet, amely tárolóregiszterekkel van kiegészítve. Ez a "aritmetikabovítás" viszont csak 4%-os felületnövekedést eredményez.

Láttuk, hogy a maximális működési frekvencia 0.35 μm -es technológiánál $\sim 1\text{GHz}$. Ez jelentősen növelhető, hogy ha a technológiát megváltoztatjuk. Ha a vonalvastagságot lecsökkentjük 0.18 μm -re, akkor a legnagyobb működési frekvenciára kb. $\sim 4\text{GHz}$ -et (*természetesen ez csak egy elvi érték*) kapunk! A különböző eredményeket a 6.4. táblázatban foglaltam össze.

	Eredeti CASTLE	Pipeline az aritmetikai modulok között	Pipeline a különböző modulokon belül
Késleltetési formula	$t_{\text{szorzó}} + 2 * t_{\text{összeadó}}$	$t_{\text{szorzó}}$	$3 * t_{\text{full-adder}}$
Késleltetés [10ns]	~ 10	~ 6	~ 1.5
Szilíciumfelület [mm ²]	0.596	0.6	0.62
Max. órajel a VIRTEX FPGA-n	46 MHz	196.92 MHz	313.97 MHz
Max órajel a szilíciumon (0.35 μm CMOS)	~ 100 MHz	~ 170 MHz	~ 1 GHz
Disszipáció [W]	< 0.3	< 0.5	< 3

6.4. táblázat

A különböző aritmetikai egységek összehasonlítása

A 6.4. táblázatban megtalálhatók azok a maximális frekvenciaértékek is, amelyek a VIRTEX FPGA használatával elérhetők.

6.4 CASTLE aritmetika optimalizálása szilícium felület szerint

A szilíciumfelületen az aritmetikai mag szorzóegysége meghatározó helyet foglal el. Ha az aritmetikai egység szilíciumfelületét csökkenteni szeretnénk, akkor a következők közül választhatunk.

- a felbontást csökkentjük;
- elhagyunk egy vagy két szorzóegységet.

Ha a felbontást csökkentjük, akkor csökken a "Baugh-Wooley" szorzó és az összeadó mérete (5. fejezet), csökken a soros carry képzésének az ideje is, tehát gyorsabb lesz a szorzó sebessége. A szakirodalom megadja azt a minimális felbontást (6 bit) is, amelynél kisebb nem lehet egy emulált digitális CNN-UM felbontása, mert akkor nem lehet vele különböző számítási feladatot megoldani. Ezért ez az út

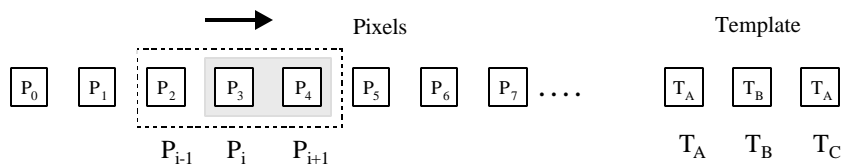
járhatatlan, viszont egy vagy két szorzóegység elhagyható. Ez a fejezet az általam megadott eljárást mutatja be, amelynek használatával kb. 30%-kal lecsökken az aritmetikai egység mérete.

Az elkészült chip-en (0.35 μm) a szorzó mérete 110120 μm^2 (5.1. táblázat). Látható tehát az, hogy ha elhagyunk egy szorzót, akkor a processzor mérete kb. 30%-kal szilíciumon lecsökken. Ebben a fejezetben egy olyan CASTLE megoldást mutatok be, amelyik két szorzóval készült el. Továbbá megvalósítható olyan aritmetikai mag is, amely egy szorzóból épül fel, viszont ennek a sebessége annyira lecsökken, hogy ennek alkalmazása ésszerűtlennek tunik, ezért ezt a megoldást nem ismertetem.

Különbséget kell tennünk a template-ek között, mert olyan aritmetikai egységet adtam meg, hogy ha oszlopszimmetrikus template-et használunk, akkor a működési idő megfelel az eredeti CASTLE aritmetikai mag (T_{OC} , három szorzós) feldolgozási idejének. Ha a template nem oszlopszimmetrikus, akkor $T_{2szorzó} = 2 * T_{OC}$.

Az analogikai algoritmusok jelentős része csak oszlopszimmetrikus template-eket használ. Kevés olyan algoritmus létezik a szakirodalomban, amely nem használ oszlopszimmetrikus template-et. Az 4. fejezetben bemutatott hibakereső analogikai algoritmusaim is oszlopszimmetrikus template-ekre épülnek.

A 6.25. ábrán látható az eredeti CASTLE aritmetikai mag feldolgozása [1]. Szaggatott vonallal van jelölve az a három pixel, amely egy ütem alatt feldolgozásra kerül. Ez a három pixel érték szorozódik össze a template memória $T_A - C$ értékeivel ($P_2 * T_A, P_3 * T_B, P_4 * T_C$, általános esetben: $P_{i-1} * T_A, P_i * T_B, P_{i+1} * T_C$). Ezt a szaggatott vonallal jelölt ablakot "húzzuk" végig a pixelmemórián és szorozzuk össze a pixeleket a megfelelő template értékekkel.



6.25. ábra A pixelek feldolgozásának sorrendje

Ezeket a szorzatokat a 6.5. táblázatban tüntettem fel.

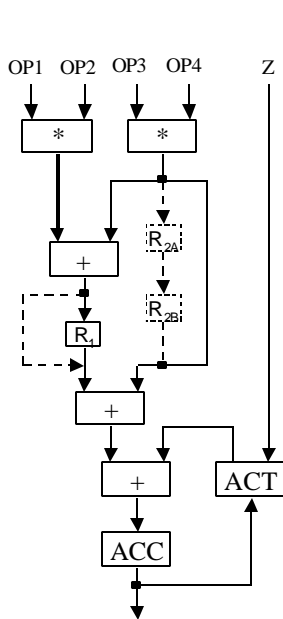
	1. szorzó	2. szorzó	3. szorzó
1. ütem	$P_1 * T_A$	$P_2 * T_B$	$P_3 * T_C$
2. ütem	$P_2 * T_A$	$P_3 * T_B$	$P_4 * T_C$
3. ütem	$P_3 * T_A$	$P_4 * T_B$	$P_5 * T_C$
4. ütem	$P_4 * T_A$	$P_5 * T_B$	$P_6 * T_C$
5. ütem	$P_5 * T_A$	$P_6 * T_B$	$P_7 * T_C$
6. ütem	$P_6 * T_A$	$P_7 * T_B$	$P_8 * T_C$
7. ütem	$P_7 * T_A$	$P_8 * T_B$	$P_9 * T_C$

6.5. táblázat A feldolgozott pixelek és template-ek szorzatai

Azt az esetet, amikor két szorzót használunk az aritmetikai egységben, azt szürke színnel jelöltem a 6.25 ábrán.

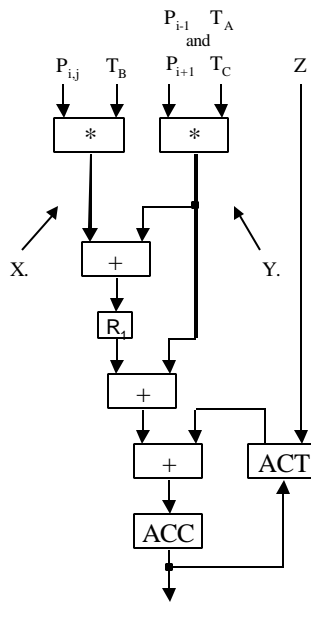
A két szorzós aritmetikai egység bal oldalt, a 6.26. ábrán látható. Két különböző

működése lehet ennek az aritmetikai magnak (6.27. és 6.28. ábra). A következő alfejezetekben ezeket fogom ismertetni.



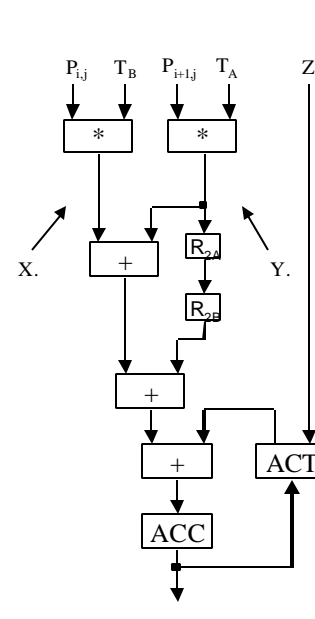
6.26. ábra

Módosított aritmetikai egység



6.27. ábra

Nem oszlopszimmetrikus template



6.28. ábra

Oszlopszimmetrikus template

Ennek az aritmetikának a felhasználási módja attól függ, hogy oszlopszimmetrikus template-et használunk-e vagy nem (6.29., 6.30. ábrák).

$$T_{\text{nem szimmetrikus}} = \begin{bmatrix} T_{A11} & T_{B12} & T_{C13} \\ T_{A21} & T_{B22} & T_{C23} \\ T_{A31} & T_{B32} & T_{C33} \end{bmatrix}$$

6.29. ábra

Nem oszlopszimmetrikus template

$$T_{\text{szimmetrikus}} = \begin{bmatrix} T_{A11} & T_{B12} & T_{A11} \\ T_{A21} & T_{B22} & T_{A21} \\ T_{A31} & T_{B32} & T_{A31} \end{bmatrix}$$

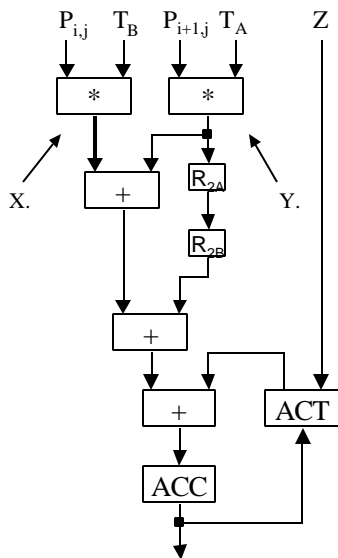
6.30. ábra

Oszlopszimmetrikus template

6.4.1 Oszlopszimmetrikus template-ek használata

A 6.31. ábra mutatja be azt az esetet, amikor az elobb ismertetett aritmetikai egységen oszlopszimmetrikus template-ekkel számolunk. Ebben az esetben az R_{2A} és az R_{2B} -es regisztert használjuk az Y . jelu szorzómodul eredményeinek tárolására. A 6.6. táblázat mutatja a kiszámolt szorzatokat a különböző ütemekben. Szürke színnel jelöltem azt a szorzatot, amit ténylegesen nem számolunk ki az első szorzóegységgel, hiszen az aritmetikai magban az a szorzó nem szerepel. Ezeket az értékeket a

"harmadik" (Y) szorzóval számoljuk ki két időegységgel korábban. Ezért tehát az első ütem előtt már ki kell számolnunk a $P_0 * T_A$, $P_1 * T_A$ szorzatokat az Y szorzóval.



6.31. ábra

Oszlopszimmetrikus template-ek használata

	1. szorzat	2. szorzat (X)	3. szorzat (Y)
1. ütem	$P_0 * T_A$	$P_1 * T_B$	$P_2 * T_A$
2. ütem	$P_1 * T_A$	$P_2 * T_B$	$P_3 * T_A$
3. ütem	$P_2 * T_A$	$P_3 * T_B$	$P_4 * T_A$
4. ütem	$P_3 * T_A$	$P_4 * T_B$	$P_5 * T_A$
5. ütem	$P_4 * T_A$	$P_5 * T_B$	$P_6 * T_A$
6. ütem	$P_5 * T_A$	$P_6 * T_B$	$P_7 * T_A$
7. ütem	$P_6 * T_A$	$P_7 * T_B$	$P_8 * T_A$

6.6. táblázat

A szorzatok kiszámolása az idő függvényében

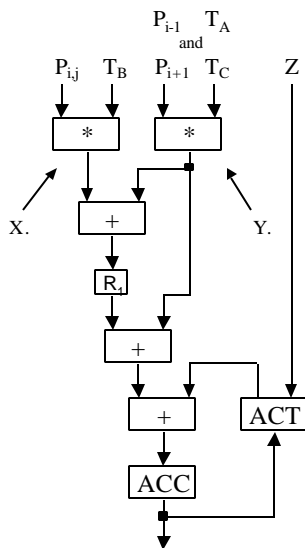
Az aritmetikai egység inicializálásakor kiszámoljuk először a $P_0 * T_A$ szorzatot és ezt az értéket eltároljuk az R_{2A} regiszterben. Ezután kiszámoljuk a $P_1 * T_A$ értékét is. Ezt szintén beírjuk az R_{2A} regiszterbe, de a beírást megelőzően az előző állapotot elmentjük az R_{2B} regiszterbe. Ezzel a néhány lépéssel sikerült a két tárolóregisztert feltöltenünk, és így a képfeldolgozás elkezdődhet. Az aritmetikai egység előkészítésekor az X. jelu szorzómodul nem használjuk, mert ezzel az egységgel csak azokat a szorzatokat számoljuk ki, amelyekben a "T_B" template érték szerepel.

Az első ütemben kiszámoljuk a két szorzóval a $P_1 * T_B$, $P_2 * T_A$ szorzatok értékeit. Ezután a $P_2 * T_A$ szorzatot eltároljuk az "R_{2A}" regiszterben. Az "R_{2A}" értéke ($P_1 * T_A$) beíródik az "R_{2B}" regiszterbe, így ez megjelenik a középső számláló jobb oldali bemenetén. Az összeadások ebben az időegység alatt történnek meg. A második ütemben ez az érték átkerül az "R_{2B}" regiszterbe, és az "R_{2A}" regiszterbe beíródik a $P_3 * T_A$ értéke, amelyet az Y. szorzóval számolunk ki. Ebben az ütemben számoljuk ki a $P_1 * T_B$ értékét is. Ezt viszont nem tároljuk el, hiszen erre az értékre csak ebben az ütemben van szükségünk.

Látható tehát az, hogy a különböző időegységekben, ütemekben minden modul (szorzó, összeadó, regiszter) dolgozik. A következő eset vizsgálatakor (amikor nem csak oszlopszimmetrikus template-ek is használhatók) látni fogjuk, hogy az X. jelu szorzómodul minden második ütemben fog működni. Emiatt abban az üzemmódban az aritmetikai egység működési sebessége a felére csökken az eredeti CASTLE aritmetikai mag (és a oszlopszimmetrikus template-et kezelő aritmetikai egység) működési sebességéhez képest.

6.4.2 Nem oszlopszimmetrikus template-ek használata

A 6.32. ábra mutatja azt az esetet, amikor a használni kívánt template nem oszlopszimmetrikus. A gyakorlatban ez az eset ritkábban fordul elő, de az aritmetikai egységet úgy terveztem meg, hogy ezeket a template típusokat is kezelni tudja. A 6.7. táblázatban láthatók azok a szorzatok, amelyek a különböző időintervallumokban, ütemekben keletkeznek.



6.32. ábra

Nem oszlopszimmetrikus template-ek használata

	1. szorzat	2. szorzat	3. szorzat
1. ütem	$P_0 * T_A$	$P_1 * T_B$	$P_2 * T_C$
2. ütem	$P_1 * T_A$	$P_2 * T_B$	$P_3 * T_C$
3. ütem	$P_2 * T_A$	$P_3 * T_B$	$P_4 * T_C$
4. ütem	$P_3 * T_A$	$P_4 * T_B$	$P_5 * T_C$
5. ütem	$P_4 * T_A$	$P_5 * T_B$	$P_6 * T_C$
6. ütem	$P_5 * T_A$	$P_6 * T_B$	$P_7 * T_C$
7. ütem	$P_6 * T_A$	$P_7 * T_B$	$P_8 * T_C$

6.7. táblázat

A szorzatok kiszámolása az idő függvényében

Tekintettel arra, hogy három szorzatot kell előállítanunk két szorzóval, ezért ezt csak két időegység alatt tudjuk megvalósítani. Ezért egy ütem két részből (két időszakból) áll.

Az aritmetikai mag az "n." ütem első felében kiszámolja a $R_{i,j} * T_B$, $P_{i-1,j} * T_A$ szorzatokat, és ezeknek az összegét, amelyet az első összeadó számol ki, eltároljuk az R_1 -es regiszterben. Az "n." ütem második részében számoljuk ki a $P_{i+1,j} * T_C$ szorzatot, amelyet összeadunk az R_1 -ben tárolt értékkel. Az "n+1." ütemben megismételjük a fent leírt műveleteket a $P_{i+1,j} * T_B$, $P_{i,j} * T_A$, $P_{i+2} * T_C$ szorzatokkal is.

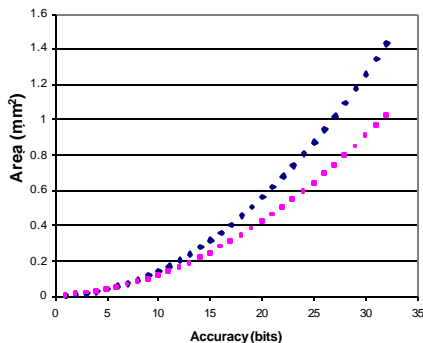
Ezt az aritmetikai egységet nem kell inicializálni, az első ütemmel (6.7. táblázat) kezdődnek az aritmetikai műveletek.

Az első ütem "első" részénél kiszámoljuk a $P_0 * T_A$, $P_1 * T_B$ szorzatokat a két szorzóegység segítségével. Az első ütem "végén" pedig kiszámoljuk a $P_2 * T_C$ értéket is. Látható tehát az, hogy az ütem két, jól elkülöníthető részből áll. Ez megfelel két órajelciklusnak is.

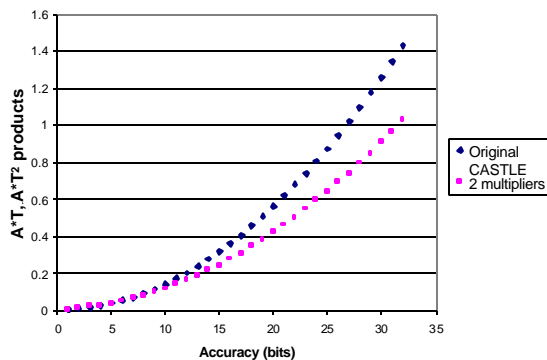
6.4.3 Az aritmetikai egységek összehasonlítása

A módosított, két szorzóból álló és az eredeti CASTLE aritmetikai egységeket összehasonlítottam felület, $A*T$ és $A*T^2$ szorzatok alapján. A felület nem változik, ez a felhasznált template típusától független. A 6.33. ábra mutatja felület-felbontás függvényt a kétszorzós és az eredeti CASTLE architektúra között.

A 6.34. ábrán látható az $A*T$ és az $A*T^2$ szorzat változása a felbontás függvényében, ha oszlopszimmetrikus template-et használunk. Látható az is, hogy az $A*T$ és az $A*T^2$ szorzatok értékei megegyeznek.

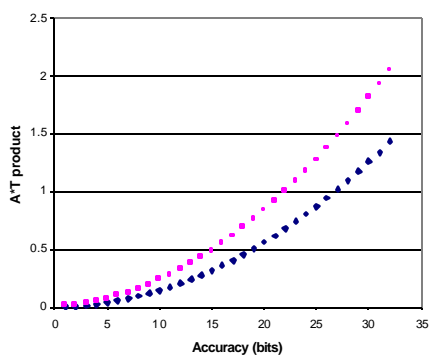


6.33. ábra
Szilíciumfelületek összehasonlítása

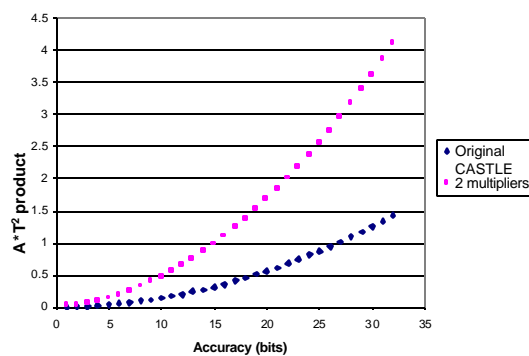


6.34. ábra
Az $A*T$, $A*T^2$ szorzatok ábrázolása

Ha a felhasznált template-ek nem oszlopszimmetrikusak, akkor az $A*T$ és az $A*T^2$ szorzatok értékei nem lesznek egyenlők. Ezek a szorzatok láthatók a 6.35. és a 6.36. ábrán a felbontás függvényében.



6.35. ábra
Nem oszlopszimmetrikus template-ek A szorzatok kiszámolása az idő használata



6.36. ábra
 $A*T^2$ szorzatok kiszámolása az idő függvényében

Látható, hogy ha növeljük a felbontást, akkor no a két aritmetikai mag felületi különbsége. Nagyobb felbontás esetén tehát célszerű a kétszörös aritmetikai egység használata.

A különböző aritmetikai egységek fizikai összehasonlítása látható a 6.8. táblázatban.

	Eredeti CASTLE aritmetikai egység	Nem oszlop-szimmetrikus template-ek	Oszlopszimmetrikus template-ek
Késleltetés [ns]	~ 10	~ 10 + 2 clock	~ 10
Késleltetési formula	$t_{mult} + 3 * t_{ö.adó}$	$t_{mult} + 3 * t_{ö.adó}$	$t_{mult} + 3 * t_{ö.adó}$
Felület [mm ²]	0.596	0.4006	0.4006
Max. órajel a VIRTEX-en [MHz]	46	51.169	51.169
Max. órajel a szilíciumon [MHz]	~ 100	~ 100	~ 100
Disszipáció [W]	< 0.3	< 0.3	< 0.3

6.8. ábra A különböző aritmetikai egységek fizikai jellemzőinek az összehasonlítása

6.5 Az optimalizált CASTLE aritmetikai egységek összefoglalása

A tesztkép mérete 128*128 pixel. Ilyen méretű kép segítségével végeztünk el különböző szimulációkat, amelyeknek a futási ideje a 6.9. táblázatban látható.

	Eredeti CASTLE	CASTLE egy szorzós	CASTLE két szorzós (oszlopszimmetrikus template-ek)	CASTLE két szorzós (általános template-ek)	CASTLE pipeline-nal
Órajel frekvenciája	100 MHz	66 MHz	100 MHz	100 MHz	1GHz
Technológia	0.35 μm	0.35 μm	0.35 μm	0.35 μm	0.35 μm
chip terület [mm ²]	7.152	1.44	4.8072	4.8072	7.44
processzorok száma	3*4	3*4	3*4	3*4	3*4
Kaszkádosítható?	igen	Igen	igen	igen	igen
Disszipáció [W]	< 0.3	< 0.2	< 0.3	< 0.3	< 3
3*3 konvolúció	21.18 (12 bit) 10.59 (6 bit)	63.54 (12 bit) 31.77 (6 bit)	21.18 (12 bit) 10.59 (6 bit)	42.36 (12 bit) 21.18 (6 bit)	2.11 (12 bit) 1.05 (6 bit)
Erozió/ Dilatáció	21.18 (12 bit) 10.59 (6 bit)	63.54 (12 bit) 31.77 (6 bit)	21.18 (12 bit) 10.59 (6 bit)	42.36 (12 bit) 21.18 (6 bit)	2.11 (12 bit) 1.05 (6 bit)
Laplace (15 iteráció)	317.8 (12 bit) 158.94 (6 bit)	953.4 (12 bit) 476.7 (6 bit)	317.8 (12 bit) 158.94 (6 bit)	635.6 (12 bit) 317.8 (6 bit)	31.7 (12 bit) 15.89 (6 bit)
Algoritmus A: (10 conv. + 10 eros. + 1 Lapl.)	741.7 (12 bit) 370.86 (6 bit)	2225.1 (12 bit) 1112.55 (6 bit)	741.7 (12 bit) 370.86 (6 bit)	1483.4 (12 bit) 741.7 (6 bit)	741.7 (12 bit) 37.08 (6 bit)
Algoritmus B: (10 conv. + 10 eros. + 10 Lapl. + 10 logic.)	3602.6 (12 bit) 476.1 (6 bit)	10807.8 (12 bit) 5403.9 (6 bit)	3602.6 (12 bit) 476.1 (6 bit)	7205.2 (12 bit) 3602.6 (6 bit)	360.26 (12 bit) 47.61 (6 bit)

6.9. táblázat Különböző CASTLE megoldások futási ideje egy 128*128-as tesztképen

Az egy szorzóval felépített CASTLE aritmetikai egység használatát nem ajánlom, hiszen egy szorzóval végzünk el kilenc szorzást 3*3-as template használata esetén. Ez jelentősen lecsökkenti a működési sebességet, még akkor is, ha a pipeline módszert használjuk.

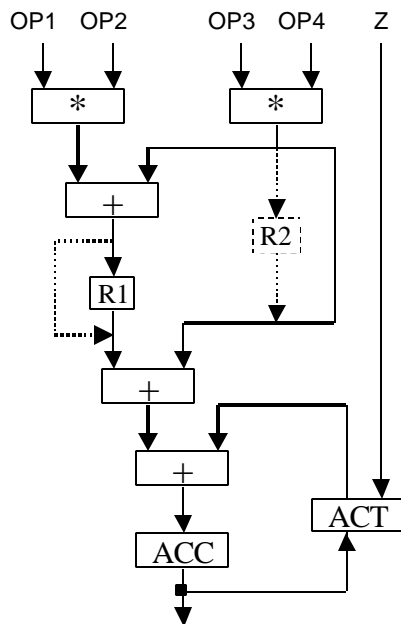
Minimalizáljuk a szilíciumfelületet, ha két szorzóval készítjük el a CASTLE aritmetikai magot. Oszlopszimmetrikus template-ek használata esetén a sebesség nem változik, megegyezik az eredeti CASTLE architektúra sebességével. Ha a felhasznált template nem ilyen, akkor a működési sebesség lecsökken a felére.

7. Az optimális aritmetikai egységek "keverése"

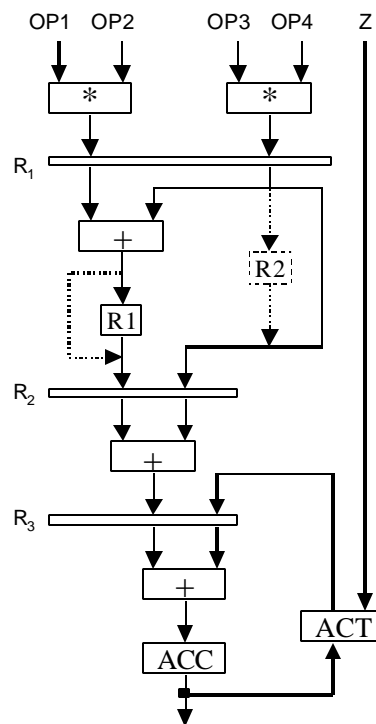
Az előző fejezetben bemutatott optimalizált emulált digitális aritmetikai egységek egymással "keverhetők". Ebben a fejezetben ilyen aritmetikai magokat mutatok be. Nem fogom részletezni a működésüket, mert a hangsúlyt nem a konkrét áramkörök, aritmetikai magok bemutatására, hanem az előzőekben ismertetett optimalizálási módszerekre helyezem. Ebben a fejezetben inkább csak ötleteket adok, amelyeket könnyen felhasználhatunk akár ennél a speciális emulált digitális architektúránál (CASTLE), akár más, tervezés alatt álló digitális CNN-UM-nél is.

7.1 Egy és két szorzós aritmetikai egység

A 7.1. ábra mutatja azt az aritmetikai egységet, amelyiket a 6. fejezetben részleteztem. Ez a kétszorzós egység szintén kiegészíthető átmeneti tárolókkal (7.2. ábra), ahogyan ezt a 6. fejezetben láttuk. A maximális működési sebesség szintén ~1GHz, és természetesen az előző fejezetben ismertetett memóriavezérlés is alkalmazható ebben az esetben.



7.1. ábra
Az eredeti kétszorzós aritmetikai egység



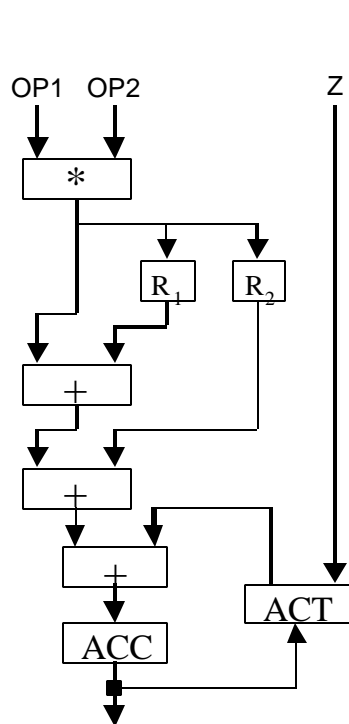
7.2. ábra
Két szorzóval és átmeneti tárolókkal megvalósított egység

Látható tehát az, hogy csökkenthető az aritmetikai egység területe sebességromlás nélkül. Természetesen ez csak szimmetrikus template-ek esetére igaz.

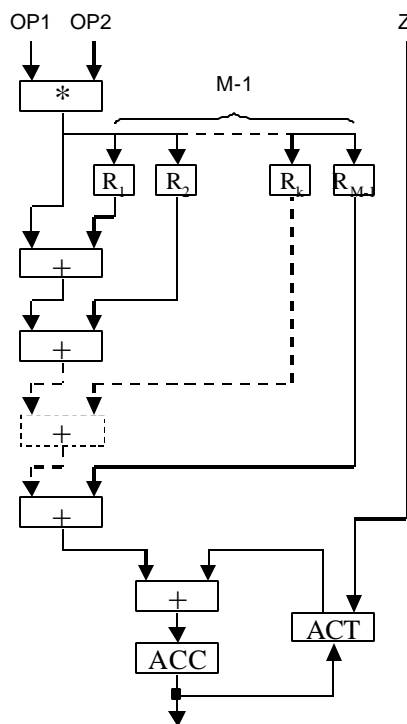
Ha használjuk a pipeline megoldást, akkor a szorzó számát tovább csökkenthetjük. Ekkor az összes szorzást egy szorzó fogja elvégezni, tehát a működési sebesség a

harmadára csökken ($V_{\text{Iszorzó}} = 0.33 \cdot V_{\text{OC}}$). Ezt az aritmetikai megoldást mutatja a 7.3. ábra.

A 7.4. ábra mutatja azt az esetet, amikor a használni kívánt template mérete "M" ($M > 3$). Ekkor a működési sebesség "M"-szer csökken ($V_{\text{Iszorzó}} = 1/M \cdot V_{\text{OC}}$).



7.3. ábra
Egy szorzóval felépített
aritmetikai egység 3*3-as
template-ekhez

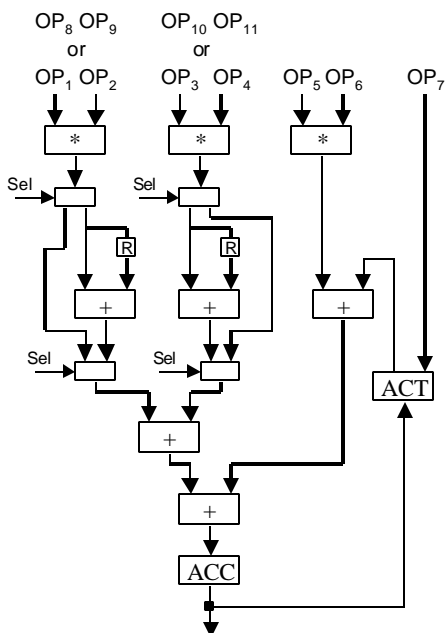


7.4. ábra
Egy szorzóval megvalósított
aritmetikai egység M*M-es
template-ekhez

7.2 Újrakonfigurálható aritmetikai egység pipeline-nal

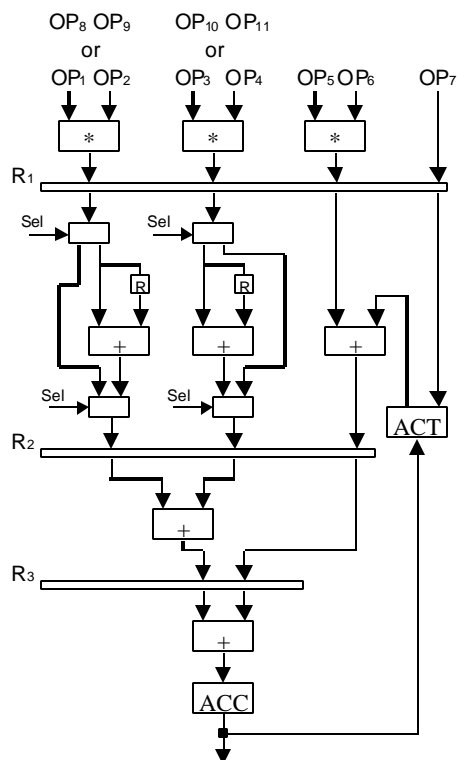
Az újrakonfigurálható aritmetikai egységeket szintén kiegészíthetjük regiszterekkel jelentős felületnövekedés nélkül. Példaként azt a megoldásomat fogom átalakítani és bemutatni, amelyet minimális felületre optimalizáltam.

Az eredeti aritmetikai egységet a 7.5. ábra mutatja, míg a kibovított változatát a 7.6. ábra. Az új változatnak a sebessége megegyezik a 6. fejezetben ismertetett pipeline változat sebességével, de ennek az egységnek az a nagy előnye, hogy képes 5*5-ös template-ek kezelésére is.



7.5. ábra

Az eredeti újrakonfigurálható aritmetikai egység



7.6. ábra

Ugyanez az egység tároló-regiszterekkel

A 7.1. táblázatban láthatóak a szimulációs és a számolt eredmények.

	Eredeti CASTLE	Három szorzós újrakonfigurálható aritmetika	Pipeline a három szorzós újrakonfigurálható aritmetika között	Oszlop-szimmetrikus template-ek	Pipeline az oszl.szim. templ. között
Késleltetés formula	$t_{\text{szorzó}} + 2 * t_{\text{összeadó}}$	$t_{\text{szorzó}} + 3 * t_{\text{ö.adó}} + 2 * t_{\text{mux/dem}}$	$t_{\text{szorzó}}$	~ 10	$t_{\text{szorzó}}$
Késleltetés [10ns]	~ 10	~ 13	~ 6	$t_{\text{mult}} + 3 * t_{\text{ö.adó}}$	~ 6
Szilíciumfelület [mm ²]	0.596	0.6	0.62	0.4006	0.62
Max. órajel a VIRTEX FPGA-n	46 MHz	34.406 MHz	196.92 MHz	51.169 MHz	196.92 MHz
Max órajel a szilíciumon (0.35 μm CMOS)	~ 100 MHz	~ 80 MHz (r=1,2)	~ 170 MHz (r=1,2)	~ 100	~ 170 MHz
Disszipáció [W]	< 0.3	< 0.3	< 0.5	< 0.3	< 0.5

7.1. táblázat

A különböző aritmetikai egységek összehasonlítása

A táblázatból kitunik, hogy érdemes a hatodik fejezetben bemutatott módszereket "keverni", hiszen ekkor az adott feladatokra optimalizálható az eredeti CASTLE architektúra [5].

Jelentosen megnövelhető a szilíciumfelületre optimalizált újrakonfigurálható aritmetikai egység működési sebessége a pipeline megoldással.

7.2 Összefoglalás

Ebben a fejezetben néhány olyan aritmetikai egységet ismertettem, amely az előző fejezetben ismertetett optimalizálási eljárások segítségével készült el. Nem ezeknek az aritmetikai modulok analizálása volt a célom, hanem az optimalizálási módszerek bemutatása.

8. Összefoglalás

Ebben a fejezetben sorolom fel a téziseimet.

1. téziscsoport [3], [4], [8], [11], [14], [15]

A nyomtatott áramkörök(Printed Circuit Board, PCB) különböző, a gyártás során keletkező hibák detekciójával foglalkozik ez a téziscsoport. Két olyan analogikai algoritmusokat adtam meg, amelyek segítségével a nyomtatott lapok gyártása során a hibák (illesztési hibák, zárlat) kiszűrhetők. Ezeknek az algoritmusoknak a sebességei egy nagyságrendbe esnek professzionális rendszerek sebességével, de a futtatásához szükséges CNN-UM platformok ára olcsóbb.

A nyomtatott áramkörök elkészítése, utólagos tesztelése bonyolult, összetett feladat. A gyártás során több fajta hiba keletkezhet, például zárlat, rövidzárlat a különböző jelvezetékek között, vezetékszakadás, illesztési hiba a gyártófilm és a lap között. Az ebben a csoportban leírt analogikai algoritmusok futási ideje elvileg független a vizsgálandó nyomtatott áramköri lapok nagyságától, pontosságuk egy pixel. A disszertációmban egy konkrét mérési összeállítást is ismertetek. Az algoritmus futási ideje csak a technológia függvénye és kevéssé függ a lap méretétől, ha a teljes felület a CNN-UM chipre letölthető. Mind a két algoritmust teszteltem 64*64-es CNN-UM chipen, bináris CNN-UM implementáción és szoftver szimulátoron.

1.1 tézis Megadtam egy layout ellenőrző analogikai algoritmust, amelynek segítségével megállapítható, hogy az egy-, vagy többretegű nyomtatott áramkör zárlatos-e [3], [4], [8]. Az algoritmus futási ideje a vizsgált kép lineáris méretével arányos (analóg VLSI implementációt feltételezve). Az eljárás ketto, vagy több jel, (vezeték) között keres zárlatot. Az algoritmus további előnye, hogy az olyan rövidzárakat is detektálja, amelyek egy-egy oldalon nem okoznak zárlatot.

A hibák eredhetnek a nyomtatott lap gyártásából, de akár tervezési hibából is. Az algoritmusnak egyoldalas nyomtatott áramkörnél három bemenete van. A fóliázat képe, a "marker-" és a "referenciakép". A markerképen egy fekete forrpont, ún. pad található. Ezen a képen alakítjuk ki hullámgenerálással azt a jelvezetékét, amelyhez ez a forrpont tartozik. A referenciaképen legalább két pad található. Az egyik az a forrpont, ami a markerképen is megtalálható, a másik forrpont pedig ahhoz a jelvezetékhez tartozik, ahol a zárlatot keressük. Ha két pad van ezen a referenciaképen, akkor azon két jelvezeték között keresi az algoritmus a zárlatot, amelyekhez a két pad tartozik. Többretegű áramköröknél a bemenetek száma a rétegek számával megegyező. Az általam megadott zárlatkereső eljárásnak egy kimeneti képe van.

1.2 tézis Kidolgoztam egy olyan analogikai algoritmust, amellyel az illesztési hibák kiszűrhetők a nyomtatott lapok gyártása során [3], [8], [11] és amelynek futási ideje analóg VLSI implementáció esetén független a vizsgált kép méretétől (ha a PCB mérete teljes egészben processzálható, egyébként darabolni kell a képet és ez lineárisan megnöveli a futási időt). Az előregedett, megnyúlt, hullámos filmek hibái detektálhatók ezzel, tetszőleges nyomtatott áramköröknél.

Az általam megadott illesztési hibákat detektáló eljárásnak két bemeneti és egy kimeneti képe van. Az egyik bemeneti kép a gyártófilmet, a másik pedig az előre kifűrt, fotózásra, szitázásra elokészített lemezt mutatja. Az algoritmus nem csak a hiba tényét jelzi, hanem megadja a hibás pixelek helyét is az algoritmus kimeneti képén.

Ennek az eljárásnak a további előnye az, hogy a futási idő független a hibák számától.

Ezt az algoritmust nem csak a téziscsoport bevezetőjében említett CNN-UM megoldásokon (64*64) futtattam, hanem a 20*22-es chipen és az MTA-SZTAKI-ban elkészített logikai processzoron is.

2. téziscsoport [1], [2], [5], [6], [7], [9], [12], [13]

Az analóg CNN-UM chip-eknek számos implementációja létezik. Ezek a chip-ek gyorsabbak és kevesebbet fogyasztanak a digitálisan emulált társaiknál, viszont pontatlanabbak és 3 dimenziós problémák nem oldhatók meg a segítségükkel. Elkészült egy digitálisan emulált architektúra (CASTLE) [5], amelynek segítségével egyszerre 16 darab 1-es szomszédosságú template használható: a feldolgozás során minden pixelhez külön template rendelhető. A template-ek letöltése a chip memóriájába független a chip működésétől. Ezeknek a template-eknek a mérete szintén 3*3 (egyes szomszédosság). A CASTLE maximális működési frekvenciája 12 bites működés esetén kb. 80 MHz [1]. Ezt a digitálisan emulált architektúrát úgy módosítottam, hogy rugalmasabb és gyorsabb megvalósításokhoz jussunk. Ez a téziscsoport ezeket a megoldásokat, a különböző szempontok alapján optimalizált architektúrákat mutatja be.

2.1 Olyan emulált digitális CNN-UM architektúrát [1], [9], [13] adtam meg, ami egyes (3*3-as) illetve kettes (5*5-ös) szomszédosságú template-eket képes kezelni egységnyi idő alatt. Ennek az architektúrának a működési sebessége (tehát a feldolgozási idő) megegyezik az eredeti digitálisan emulált, 3*3-as CNN-UM (CASTLE) [5], [12] működési sebességével. Bemutattam továbbá két optimalizálási eljárást is, amelyek segítségével ez az általános, úgynevezett "újrakonfigurálható" architektúra optimalizálható szilícium-felületre és működési sebesség szerint is. Ezeknél a megoldásoknál működés közben tudjuk megváltoztatni (tudjuk újrakonfigurálni) a felhasznált template-ek méretét.

A szilíciumfelületre optimalizált aritmetikai egység működési sebessége változatlan 3*3-as template használata esetén, és fele az eredeti CASTLE sebességének, ha a template mérete 5*5. Az eredeti elrendezésben található

három szorzónak ugyanis öt szorzatot kell megvalósítani a szilíciumon. A

felületi növekedés viszont még a 5 %-ot sem éri el.

A működési sebesség szerint optimalizált újrakonfigurálható aritmetikai egység sebessége kétszerese az eredeti CASTLE működési sebességének, ha 3*3-as template-eket használunk. Ha a használt template-ek mérete 5*5-ös, akkor az aritmetika működési sebessége megegyezik az eredeti CASTLE sebességével.

2.2 Megoldást adtam az MTA-SZTAKI-ban kifejlesztett digitálisan emulált CNN-UM (CASTLE) chip működési sebességének növelésére.

Két eljárást adtam a CASTLE architektúra sebességnövelésére az úgynevezett "pipeline" technikával [1], [2], [7], [13]. Az első változatnál csak a főbb műveletet végző egységek (szorzó, összeadó) [43], [45] között vannak átmeneti tárolók. Így kétszeres sebességnövekedést tudtam elérni.

A második megoldásnál a szorzóban és az összeadóban is elhelyeztem regisztereket. A maximális működési frekvencia értéke így tízszerese lett az eredetinek, a maximális működési sebesség ekkor ~ 1GHz [7]. Ahhoz, hogy a szakirodalomból ismert, a gyakorlatban is használt "pipeline" elvét [44] a digitálisan emulált CNN-UM-ek "világában" is használni tudjuk, át kellett alakítanom az eredeti architektúra adatfeldolgozását és vezérlését. Ezért egy új adatfeldolgozási eljárást javasoltam. Ennek lényege, hogy egy aritmetikai egységhez öt, egymástól független lokális memória tartozik.

Egy új digitálisan emulált CNN-UM architektúrát adtam a modulokon belül is alkalmazott pipeline-osított aritmetika felhasználására is, hiszen közvetlenül nem aknázható ki sikeresen ez a jelentős sebességnövekedés. Megmutattam, hogy ezzel a megoldással biztosan kezelhető a belső, 1GHz-es órajel.

2.3 Megadtam egy olyan digitálisan emulált CNN-UM aritmetikai egységet, amelynek a szilíciumfelülete 30 %-kal csökkent [1], [2], [13]. Ez az aritmetikai egység csak 3*3-as template-ekkel dolgozik.

Ezt az elrendezést oszlopszimmetrikus template-ekre [35] optimalizáltam. Az aritmetikai egységnél a futási idő változatlan, ha oszlopszimmetrikus template-eket használunk. Ha a használt template nem oszlopszimmetrikus, akkor a futási idő megkétszereződik.

Tekintettel arra, hogy egy analogikai algoritmus tartalmazhat nem szimmetrikus template-eket, ezért egy olyan aritmetikai egység felépítését adtam meg, amellyel nem csak szimmetrikus template-ek használhatók [35]. Megadtam továbbá a futási idők függését is a felhasznált template-ektől függően.

Irodalomjegyzék

A szerzo publikációi

- [1] **T. Hidvégi**, P. Keresztes and P. Szolgay, "Enhanced Modified Analyzed Emulated Digital CNN-UM (CASTLE) Arithmetic Cores", *Journal of Circuits, Systems, and Computers*, Special Issue on "CNN Technology and Visual Microprocessors (*accepted, in print*)
- [2] **T. Hidvégi**, "Optimized emulated digital CNN-UM (CASTLE) Architectures" *Acta Cybernetica*, 2002 (*submitted*)
- [3] **T. Hidvégi** and P. Szolgay, "Some New Analogic CNN Algorithms for PCB Quality Control" *International Journal of Circuit Theory and Applications*, 30, pp. 231-245, 2002
- [4] **T. Hidvégi** and P. Szolgay, "Short Circuit Detection on Printed Boards During the Manufacturing Process by Using an Analogic CNN Algorithms" *IEA/AIE-2001 June 4-7*, 2001 Budapest, Hungary
- [5] P. Keresztes, Á. Zarándy, T. Roska, P. Szolgay, T. Bezák, **T. Hidvégi**, P. Jónás and A. Katona, "An Emulated Digital CNN Implementation", *Journal of VLSI Signal Processing*, Special Issue: Spatiotemporal Signal Processing with Analogic CNN Visual Microprocessors, Vol.23., No.2/3., pp. 291-304, Kluwer, 1999
- [6] **T. Hidvégi**, "Optimized emulated digital CNN-UM (CASTLE) Architectures", *Third Conference of PhD Students in Computer Science*, Szeged, Hungary, (abstract) pp. 45, 2002
- [7] **T. Hidvégi**, P. Keresztes, and P. Szolgay, "An Accelerated CNN-UM (CASTLE) Architecture by using the Pipe-Line Technique", *Proc. of IEEE CNNA'02*, Frankfurt, pp. 355-362, 2002
- [8] **T. Hidvégi** and P. Szolgay, "Some New Analogic CNN Algorithms for PCB Quality Control", *Proc. of ECCTD'01 IEEE*, European Conference on Circuit Theory and Design, Espoo, Finland, pp. I-365-368, 2001
- [9] **T. Hidvégi**, T. Bezák, P. Keresztes, and P. Szolgay, "A re-configurable arithmetic of an emulated digital CNN-UM", *Proc. of DDECS'01 IEEE*, Győr, Hungary, pp. 195-200, 2001
- [10] P. Szolgay, **T. Hidvégi**, Zs. Szolgay, and P. Kozma, "A comparison of the different CNN implementations in solving the problem of spatiotemporal dynamics in mechanical

systems”, 6th. Proc. of IEEE International Workshop on Cellular Neural Networks and Their Applications, CNNA2000, Vol. pp. 9-12.

[11] **T. Hidvégi**, P. Szolgay, and Á. Zarándy, „A New Type of Analogic CNN Algorithm for Printed Circuit Board Layout error Detection”, Proc. of INES'99 IEEE, Stará Lesná, pp. 501-506, 1999.

[12] P. Keresztes, T. Bezák, Á. Zarándy, T. Roska, P. Szolgay, **T. Hidvégi**, P. Jónás, and A. Katona, "Design methodology of an emulated digital CNN-UM chip", *Proc. of Engineering of Modern Electric Systems '99*, Oradea, 1999

[13] **T. Hidvégi**, P. Keresztes and P. Szolgay, “Enhanced Modified Analyzed Emulated Digital CNN-UM (CASTLE) Arithmetic Cores” DNS-8-2002, Budapest, 2002

[14] **T. Hidvégi** and P. Szolgay, “Short Circuit Detection on Printed Circuit Boards during the manufacturing process by Using an Analogic CNN Algorithm” DNS-12-2000, Budapest, 2000

[15] **T. Hidvégi** and P. Szolgay, “Misalignment Error Detection in Printed Circuit Board Fabrication Using an Analogic CNN algorithm” DNS-11-2000, Budapest, 2000

CNN, PCB, tervezés tárgyú publikációk

[16] L.O.Chua and L.Yang, “Cellular neural networks: Theory”, IEEE Trans. On Circuits and Systems, Vol.35, pp. 1257-1272, 1988.

[17] L.O.Chua and L.Yang, “Cellular neural networks: Applications”, IEEE Trans. On Circuits and Systems, Vol.35, pp. 1273-1290, 1988

[18] T.Roska and L.O.Chua, “The CNN Universal Machine: an analogic array computer”, IEEE Transactions on Circuits and Systems-II Vol.40, pp. 163-173, March, 1993

[19] T. Roska, P. Szolgay, Á. Zarándy, P. Venetianer, A. Radványi, T. Szirányi, “On a CNN chip-prototyping systems” Proc. of CNNA'94, Rome, pp. 375-380, 1994.

[20] Ákos Zarándy, “ACE Box: High-performance Visual Computer based on the ACE4k Analogic Array Processor Chip” Proc. of ECCTD'01, pp. I-361-364, 2001, Espoo, Finland

[21] R.Dominguez-Castro, S.Espejo, A.Rodriguez-Vazques, R.Carmona, P.Földesy, Á.Zarándy, P.Szolgay, T.Szirányi and T.Roska “A 0.8 μm CMOS 2-D programmable mixed-signal focal-plane arrayprocessor with on-chip binary imaging and instructions

storage” Vision Chip with Local Logic and Image Memory, IEEE J. of Solid State Circuits 1997

[22] G. Linan, S. Espejo, R. Domínguez-Castro, A. Rodríguez-Vázquez "The CNNUC3: An Analog I/O 64*64 CNN Universal Machine Chip Prototype with 7-bit Analog Accuracy" Proc. of CNNA '2000, Catania, pp. 201-206, 2000

[23] G. Linan, S. Espejo, R. Domínguez-Castro, S. Espejo, A. Rodríguez-Vázquez "Design of a Large-Complexity Analog I/O CNNUC" Proc. of ECCTD '99, Stresa, pp. 42-57, 1999

[24] G. Linan, R. Dominguez-Castro, S. Espejo, A. Rodriguez Vazquez “ ACE16k: A Programmable Focal Plane Vision Processor with 128*128 Resolution” Proc. of ECCTD'01, pp. I-345-348, 2001, Espoo, Finland

[25] A. Paasio, A. Kananen, V. Porra "A 176*144 processor binary I/O CNN-UM chip design" Proc. of ECCTD '99, Stresa, pp. 82-86, 1999

[26] P. Szolgay, K. Tömördi, “Analogic algorithms for optical detection of breaks and short circuits on the layouts of printed circuit boards using CNN” International Journal of Circuit Theory and Applications 27, pp. 103-116, 1999

[27] R. T. Chin, C. A. Harlow, “Automated Visual Inspection: A Survey”, IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. PAMI-4, No. 6, pp. 557-573, (nov. 1982.)

[28] A. M. Darwish, A. K. Jain, “ A Rule Based Approach for Visual Pattern Inspection”, IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. PAMI-10, No. 1, pp. 56-68, (Jan. 1988.)

[29] Y. Hara, H. Doi, K. Karasaki, T. Iida, “A System for PCB Automated Inspection Using Flurorescent Light”, IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. PAMI-10, No. 1, pp.69-78, (Jan. 1998.)

[30] E. B. D. Lees and P. D. Hensshaw, “Printed circuit board inspection - a novel approach”, SPIE - Automatic Inspection Measurements, 1986.

[31] J. R. Mandeville, “Novel method for analysis of printed circuit images”, IBM J. Res and Dev. Vol. 29, pp. 73-87, (1985.)

- [32] M. Moganti, F. Ercal, C. H. Dagli and S. Tsunekawa, "Automatic PCB Inspection Algorithms: a Survey", Computer vision and Image Understanding, Vol. 63, No. 2. pp. 287-313 (March 1995.)
- [33] M. Moganti, F. Ercal "A subpattern level inspection system for printed circuit boards", Computer vision and Image Understanding, Vol. 70, No. 1. pp. 51-62 (April 1998.)
- [34] A. Paasio , J. Paakkulainen, J. Isoaho "A Compact Digital CNN Array for Video Segmentation System" Proc. of CNNA '2000, Catania, pp. 229-233, 2000
- [35] "CNN Software Library" in CADETWIn, T. Roska, L. Kék, L. Nemes, A. Zarándy, M. Brendel, and P. Szolgay, Eds. Budapest, Hungary: Hungarian Academy of Sciences, 1998.
- [36] L. Kék and Á. Zarándy, "Implementation of Large-Neighborhood Nonlinear Templates on the CNN Universal Machine", International Journal of Circuit Theory and Applications, Vol. 26, No. 6, pp. 551-566, 1998.
- [37] <http://www.xilinx.com>
- [38] <http://www.xess.com>
- [39] <http://www.intel.com>
- [40] <http://www.ti.com>
- [41] G. Almasi et al., "Cellular Supercomputing with System-On-A-Chip" IEEE Proc. of Solid-State Circuits Conference 2002, San Francisco
- [42] Z. Nagy, P. Szolgay "An emulated digital CNN-UM implementation on FPGA with programmable accuracy" DDECS'01 IEEE pp. 203-208, Gyor, Hungary 18-20. April 2001.
- [43] Kai Hwang, "Computer Arithmetic Principles, Architecture, and Design" John Wiley & Sons, New York, 1979
- [44] P. Arató, T. Visegrády, I. Jankovits, Sz. Szigeti, "High-Level Synthesis of Pipelined Datapaths" Edited by P. Arató, Panem Budapest, 2000
- [45] B. Parhami, "Computer Arithmetic: Algorithms and Hardware Designs" New York, Oxford, Oxford University Press, 2000
- [46] Raul Camposano, Wayne Wolf, "High-Level VLSI Synthesis" Boston, Kluwer Academic Publishers, 1991

- [47] <http://www.austriamicrosystems.com/>
- [48] K.A.Wen, J.Y.Su and C.Y.Lu, "VLSI design of digital Cellular Neural Networks for image processing", *J. of Visual Communication and Image Representation*, Vol.5, No.2, pp. 1117-126, 1994.
- [49] T. Ikenaga, T. Ogura, "Discrete-time Cellular Neural Networks using highly-parallel 2D Cellular Automata CAM²" *Proc. of Int. Symp. on Nonlinear Theory and its Applications*, pp. 221-224, 1996.
- [50] M.D.Doan, M.Glesner, R. Chakrabaty, M. Heidenreich, S. Cheung, "Realisation of digital Cellular Neural Network for image processing", *Proc. of the IEEE CNNA'94, Rome*, pp. 85-90, 1994.
- [51] CNAPS/PCI Parallel Co-processor, Adaptive Solutions Inc.
- [52] A. Zarandy, P. Keresztes, T. Roska, P. Szolgay "An emulated digital architecture implementing the CNN Universal Machine" *proc. of the fifth IEEE Int. Workshop on Cellular Neural Networks and their Applications*, London pp: 249-252, April, 1998.
- [53] Livermore MAGIC Release User's manual, PaloAlto, 1990.
- [54] CADENCE User's manual, 1999.
- [55] <http://www.cadence.com>
- [56] <http://www.siliconinterfaces.com/ServicesVLSI.htm>
- [57] P. Szolgay, I. Kispál, T. Kozek, "An experimental system for optical detection of layout errors using analog software on a dual CNN" *DNS-14-1992*
- [58] T. Roska, J. Hátori, E. Lábos, K. Lotz, L. Orzó, J. Takács, P. Venetianer, Z. Vidnyánszky, and Á. Zarándy, "The Use of CNN Models in the Subcortical Visual Pathway", *IEEE Transactions on Circuits and Systems-I: Fundamental Theory and Applications*, Vol. 40, pp. 182-195, March 1993.
- [59] Roska Tamás, "Neurális hálózatok és dinamikus processzor tömbök elmélete" *előadás jegyzet, Budapest-Veszprém, 1988-1996*
- [60] T. Szirányi and M. Csapodi, "Texture classification and Segmentation by Cellular Neural Network using Genetic Learning", *Computer Vision and Image Understanding*, Vol. 71, No. 3, pp. 255-270, September 1998.

- [61] K. R. Crouse, T. Roska and L. O. Chua, "Image halftoning with Cellular Neural Networks", IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing, Vol. 40, No. 4, pp. 267-283, 1993.
- [62] L. O. Chua, T. Roska, P. L. Venetianer and Á. Zarándy, "Some Novel Capabilities of CNN: Game of Life and Examples of Multipath Algorithms", Proceedings of the International Workshop on Cellular Neural Networks and their Applications (CNNA-92), pp. 276-281, Munich, 1992.
- [63] Dunay Rezso, Horváth Gábor, Pataki Béla, Strausz György, Szabó Tamás, Várkonyiné Kóczy Annamária, "Neurális hálózatok és muszaki alkalmazásaik", Muegyetemi Kiadó, Budapest, 1998.
- [64] L. O. Chua, T. Roska and P. L. Venetianer, "The CNN is universal as the Turing machine," IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications, 40(4), 289-291, April 1993.
- [65] Sz. Tokés, L. Orzó and T. Roska, "Design Aspects of an Optical Correlator Based CNN implementation" Proceeding of ECCTD'01 "Circuit Paradigm in the 21st Century", held in Espoo, Finland, 28-31 August, 2001.
- [66] S. Tokés, L.R. Orzó, G. Váró, A. Dér, P. Ormos and T. Roska, "Programmable Analogic Cellular Optical Computer using Bacteriorhodopsine as Analog Rewritable Image Memory", NATO Book: "Bioelectronic Applications of Photochromic Pigments" (Eds.: L. Keszthelyi, J. Stuart and A. Der) IOS Press, Amsterdam, Netherlands, pp 54-73. 2000
- [67] <http://www.modelsim.com>
- [68] Á. Zarándy, T. Roska, P. Szolgay, S. Zöld, P. Földesy and I. Petrás, "CNN chip Prototyping and Development Systems", European Conference on Circuit Theory and Design - ECCTD'99, Stresa, Italy, 1999.
- [69] I. Szatmári, Á. Zarándy, P. Földesy and L. Kék, "An Analogic CNN Engine Board with the 64*64 Analog I/O CNN-UM Chip", 2000 IEEE International Symposium on Circuits and Systems (ISCAS-2000), May 28 - May 31, 2000, pp. 395-400, Geneva, Switzerland.

1. Függelék (Template gyűjtemény)

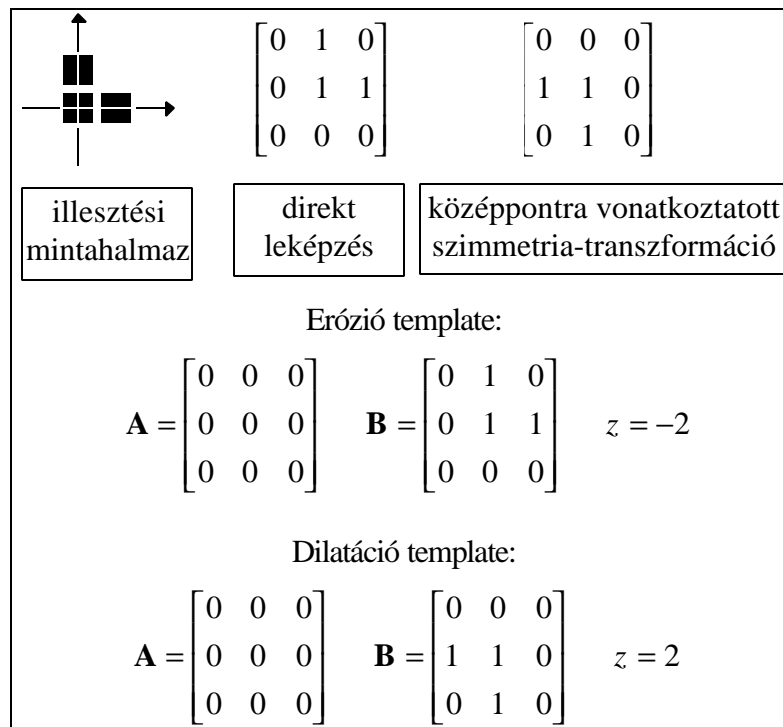
Ebben a függelékben mutatom be azokat a template-eket, amelyeket az analogikai algoritmusaimnál használtam. Található egy 5*5-ös template is, amelyet csak azért mutatok be, mert az általam megadott újrakonfigurálható aritmetikai egységek ezeket is képesek kezelni. Ezek a template-ek megtalálhatók a template könyvtárban [35] is.

Munkám során az Aladdin rendszert használtam, de ezt nem ismertetem a dolgozatomban.

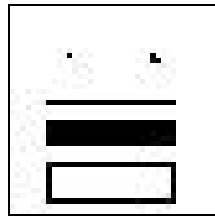
1.1 Bináris matematikai morfológia

Az erózió és a dilatació a bináris matematikai morfológia alapmuveletei. Ezeket két bináris képen definiáljuk, amelyek közül az egyik az operandus, a másik pedig az illesztési minta. CNN implementációban az előbbi jelenti a bemenetet, míg az utóbbi határozza meg a funkciót (template-eket). Ha az illesztési mintahalmaz nem haladja meg a CNN template méretét, akkor az erózió és a dilatació egy CNN template-tel implementálható. Az implementáció a következőképpen történik: az A template minden elemét nullával egyenlőnek fogadjuk el; az illesztési mintahalmazt leképezzük a B template-re (lásd az ábrát). Erózió esetén $z = 1-n$, ahol n az eggyel egyenlő B template-elemek száma. Dilatació esetén $z = n-1$, és B-re középpontra vonatkoztatott szimmetria-transzformációt hajtunk végre. A template-ek futtatásakor a kezdeti állapotot mindig nullába állítjuk.

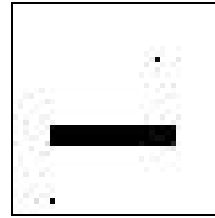
A következő ábrán a template-szintézis egy példája látható.



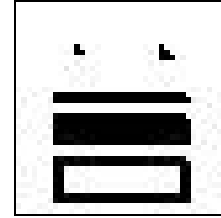
Példa: Erózió és dilatació adott illesztési mintahalmazzal. Képnév: binmorph.bmp; képméret: 40x40.



bemenet



erózió



dilatació

1.2 HLF5: 5x5-ös blokkokban történő alszürke árnyalatok létrehozása (5x5 halftoning)

$$A = \begin{bmatrix} -0.02 & -0.07 & -0.10 & -0.07 & -0.02 \\ -0.07 & -0.32 & -0.46 & -0.32 & -0.07 \\ -0.10 & -0.46 & -1.05 & -0.46 & -0.10 \\ -0.07 & -0.32 & -0.46 & -0.32 & -0.07 \\ -0.02 & -0.07 & -0.10 & -0.07 & -0.02 \end{bmatrix} \quad B = \begin{bmatrix} 0.02 & 0.07 & 0.10 & 0.07 & 0.02 \\ 0.07 & 0.32 & 0.46 & 0.32 & 0.07 \\ 0.10 & 0.46 & 0.81 & 0.46 & 0.10 \\ 0.07 & 0.32 & 0.46 & 0.32 & 0.07 \\ 0.02 & 0.07 & 0.10 & 0.07 & 0.02 \end{bmatrix} \quad Z = 0$$

I. Globális feladat

Adott: P - statikus szürke tónusú kép

Bemenet: $U(t) = P$

Kezdeti állapot: $X(0) = P$

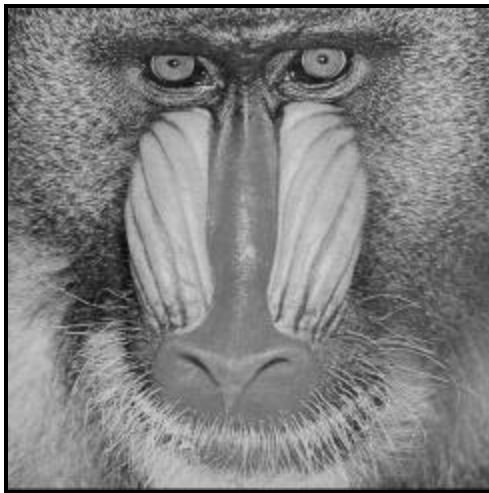
Határfeltételek: Rögzített típusú, $u_{ij} = 0$, $y_{ij} = 0$ minden virtuális cella esetén
($[U] = [Y] = 0$)

Kimenet: $Y(t) \hat{=} Y(\infty) =$ Bináris kép, amely megőrzi P főbb jellemzőit.

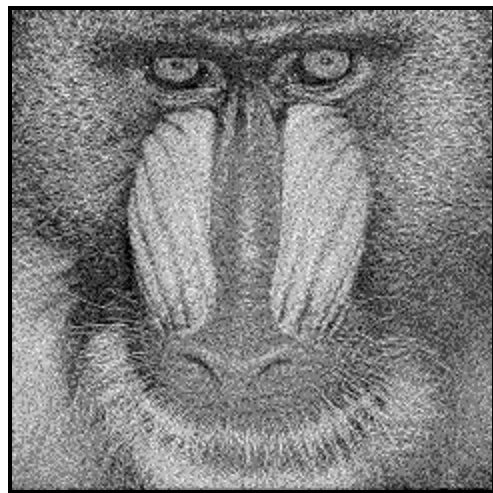
Megjegyzés: **INVHLF5** a template inverze. Az eredmény négyzetes hiba mértéket tekintve optimális.

II. Példák

1. példa: képnév: baboon.bmp; képméret: 512x512.



bemenet

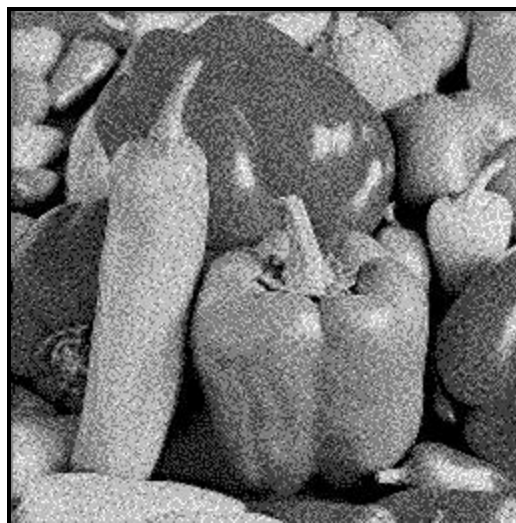


kimenet

2. példa: képnév: peppers.bmp; képméret: 512x512.



bemenet



kimenet

1.3 LOGAND: Logikai ÉS kapcsolat

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad Z = -1$$

I. Globális feladat

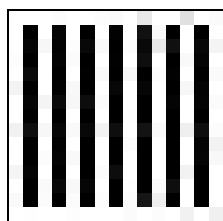
Adott: P_1 és P_2 - statikus bineáris képek

Bemenet: $U(t) = P_1$

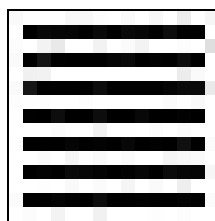
Kezdeti állapot: $X(0) = P_2$

Kimenet: $Y(t) \text{ P } Y(\text{Y}) = \text{Bináris}$

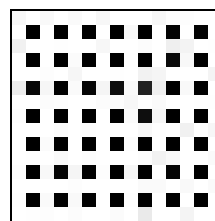
II. Példa: képnév: logic01.bmp; logic02.bmp, képméret: 44x44.



bemenet



kezdeti állapot



kimenet

1.4 LOGNOT: Szürke tónusú – bináris leképzés küszöböléssel

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad Z = 0$$

I. Globális feladat

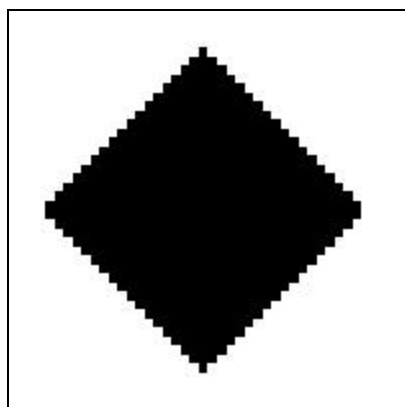
Adott: P - statikus bineáris kép

Bemenet: $U(t) = P$

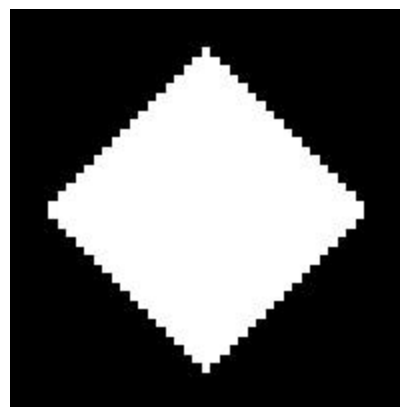
Kezdeti állapot: $X(0) = Tetszôleges$ (alapértelmezésként $X(0) = 0$)

Kimenet: $Y(t) \text{ P } Y(\forall) = Bináris$

II. Példa: képnév: logic05.bmp; képméret: 44x44.



bemenet



kimenet

1.5 LOGOR: Logikai VAGY kapcsolat

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad Z = 1$$

I. Globális feladat

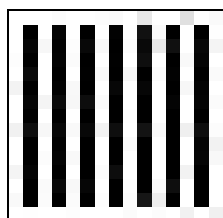
Adott: P_1 és P_2 - statikus bineáris képek

Bemenet: $U(t) = P_1$

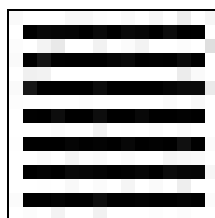
Kezdeti állapot: $X(0) = P_2$

Kimenet: $Y(t) \text{ P } Y(\infty) = \text{Bináris}$

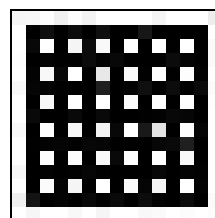
II. Példa: képnév: logic01.bmp; logic02.bmp, képméret: 44x44.



bemenet



kezdeti állapot



kimenet

1.6 SMKILLER: A kisebb fekete objektumok eltüntetése

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad Z = 0$$

I. Globális feladat

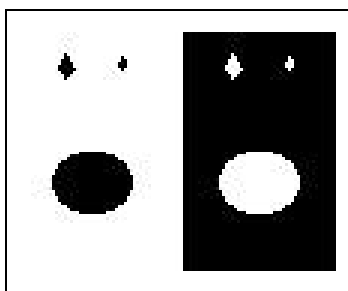
Adott: P - statikus bineáris kép

Bemenet: $U(t) = P$

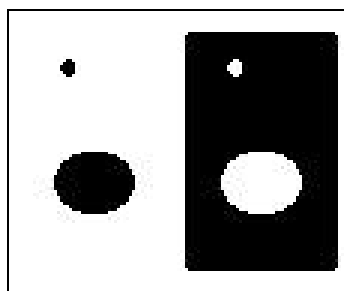
Kezdeti állapot: $X(0) = P$

Kimenet: $Y(t) \hat{=} Y(\infty) = \text{Bináris kép kis fekete objektumok nélkül}$

II. Példa: képnév: smkiller.bmp; képméret: 115x95.



bemenet



kimenet

1.7 TRESHOLD: Szürke tónusú – bináris leképezés küszöböléssel

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad Z = -Z^*, -1 < Z^* < 1$$

I. Globális feladat

Adott: P - statikus szürke tónusú kép, z^* - küszöbérték

Bemenet: $U(t) = Tetszôleges$ (alapértelmezésként $U(t) = 0$)

Kezdeti állapot: $X(0) = P$

Kimenet: $Y(t) \hat{=} Y(\forall) = Bináris$ kép, amelyen a fekete pixelek a P azon pozícióit jelölik, ahol a szürke tónusú intenzitás az adott küszöbértéknél nagyobb ($p_{ij} > z^*$).

II. Példa: képnév: madonna.bmp; képméret: 59x59; $z^* = 0.4$.



bemenet



kimenet

2. Függelék (Az AMC file-ok, futtatás)

2.1 Zárlatkereső algoritmus AMC kódja

Ezen az oldalon látható a zárlatkereső algoritmus egyik AMC kódja.

```
;          PCB ellenorzes
;          Zarlatdetekcio

host.load.tem recall.tem TEM1 ; beolvassuk a
; template-eket
host.load.tem erosion.tem TEM6 ; és eltároljuk a
; memóriában

host.load.pic top.bmp LLM1 ; beolvassuk a
; képeket
host.load.pic botjo.bmp LLM2 ; és eltároljuk a
; memóriában
host.load.pic marker.bmp LLM3
host.load.pic marker2.bmp LLM30

host.display LLM1 1 ; Megjelenítjük a képe-
; ket
host.display LLM2 2
host.display LLM3 3

;***** Algoritmus *****
ar.tem TEM1 LLM1 LLM3 LLM4 260 0 0 nil nil ;A "recall" template-et
; futtadjuk.

host.display LLM4 4 ; Az eredményt
; megjelenítjük.

ar.tem TEM6 LLM4 LLM4 LLM5 3 0 0 nil nil ;Az "erosion" template-
; et futtatjuk.
ar.tem TEM6 LLM5 LLM5 LLM6 3 0 0 nil nil
ar.tem TEM6 LLM6 LLM6 LLM5 3 0 0 nil nil

ar.tem TEM1 LLM2 LLM5 LLM6 200 0 0 nil nil ; Újabb "recall" futtatás

host.display LLM6 5 ; Az eredményt
```

```

; megjelenítjük.

ar.and.llm    LLM30 LLM6 LLM34    ; Logikai összeadás a
; két részeredmény
; között.

host.display LLM34 6              ; Az algoritmus kimeneti
; képe, itt látható a vég-
; eredmény

```

2.2 Illesztési hibákat detektáló algoritmus AMC kódja

Itt látható az illesztési hibákat detektáló algoritmusnak az az AMC kódja, amelyikkel a nem FILL típusú panelek vizsgálhatók.

```

;          PCB ellenorzes
;          Nem FILL tipusu panel

host.load.tem    lognot.tem    TEM1    ; beolvassuk a
; host.load.tem
; paranccsal lognot.tem
; template-et
host.load.tem    logand.tem    TEM2    ; template beolvasás
host.load.tem    dilation.tem  TEM3    ; template beolvasás

host.load.pic    artwrossz1.bmp  LLM1 BIN ; beolvassuk a
; host.load.pic
; paranccsal a
; mesterfilm képét
host.load.pic    fur1.bmp      LLM8 BIN ; képbeolvasás

host.display    LLM1 1          ; Megjelenítjük a
; mesterfilmet a
; képernyőn
host.display    LLM8 2          ; A képernyőn
; megjelenik a kifűrt
; panel képe is

;***** Algorithmus *****

```

```

ar.tem TEM2 LLM1 LLM8 LLM2 3 0 0 nil nil ; a logand.tem futtatása

```

```

host.display LLM2 3              ; az eredmény

```

; megjelenítése

ar.tem TEM1 LLM2 LLM2 LLM3 3 0 0 nil nil ; ezt az eredményt
; invertáljuk

host.display LLM3 4 ; megjelenítjük a
; részeredményt

ar.tem TEM2 LLM3 LLM8 LLM4 3 0 0 nil nil ; logikai ÉS kapcsolat a
; részeredmény és a
; kifűrt panel képe
; között

host.display LLM4 5 ; ezt a részeredményt is
; megjelenítjük

ar.tem TEM3 LLM4 LLM4 LLM5 3 0 0 nil nil ; a dilation.tem
; segítségével
; megnöveljük a
; megtalált illesztési
; hibákat

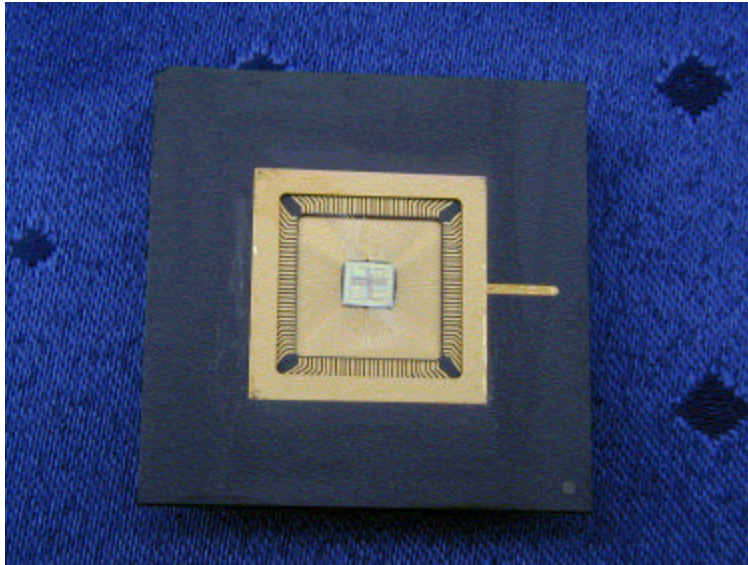
ar.tem TEM3 LLM5 LLM5 LLM4 3 0 0 nil nil ; ismételten futtatjuk a
; dilation.tem-et

host.display LLM4 6 ; megjelenik a
; végeredmény, ahol az
; illesztési hibák
; láthatóak

3. Függelék (Logikai processzor tesztkörnyezete)

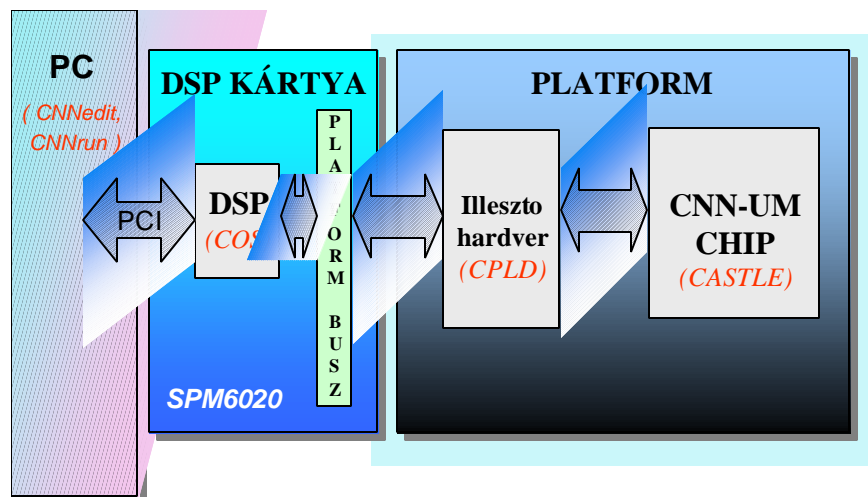
Az MTA-SZTAKI kutatóintézetben elkészült egy logikai processzor, amely a CASTLE architektúra egybites felbontású változata. Egy processzor 40 pixel széles képet képes feldolgozni. A tokozott chip látható a 3.1. ábrán. A tok nem egy CNN tömböt, hanem csak annak egy celláját tartalmazza.

A logikai processzor fizikai megvalósítását Bezák Tamás és e disszertáció szerzője készítette el, a platformot és a most bemutatásra kerülő PC-n futó demót pedig Sütő István.



3.1. ábra A logikai processzor tokozott állapotban

A CNN processzort a platform és a DSP kártya segítségével tudjuk a PC-hez csatlakoztatni (3.2. ábra).



3.2. ábra A logikai processzor tesztkörnyezete

A platformot mutatja a 3.3. ábra. Ez a kártya úgy készült el, hogy a PC104-es szabványú DSP kártyához (3.4. ábra) és az adapterkártya segítségével a PC PCI buszához csatlakoztatható legyen.

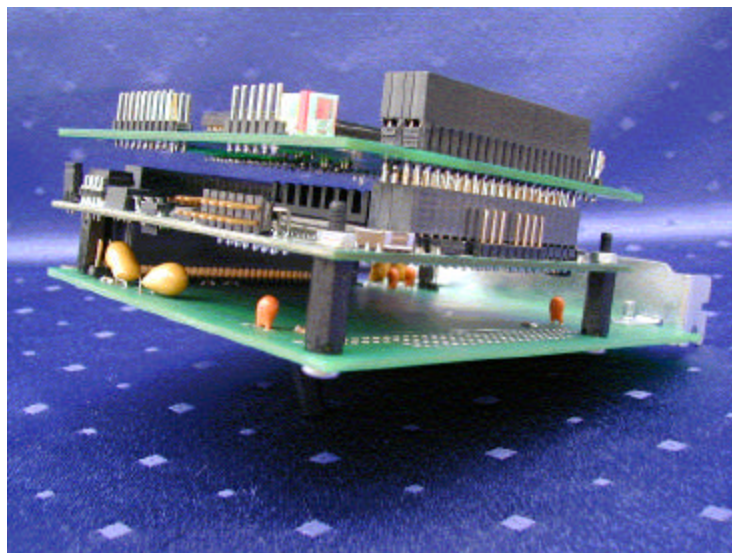


3.3. ábra A platform



3.4. ábra A DSP kártya és a PCI illeszto adapter

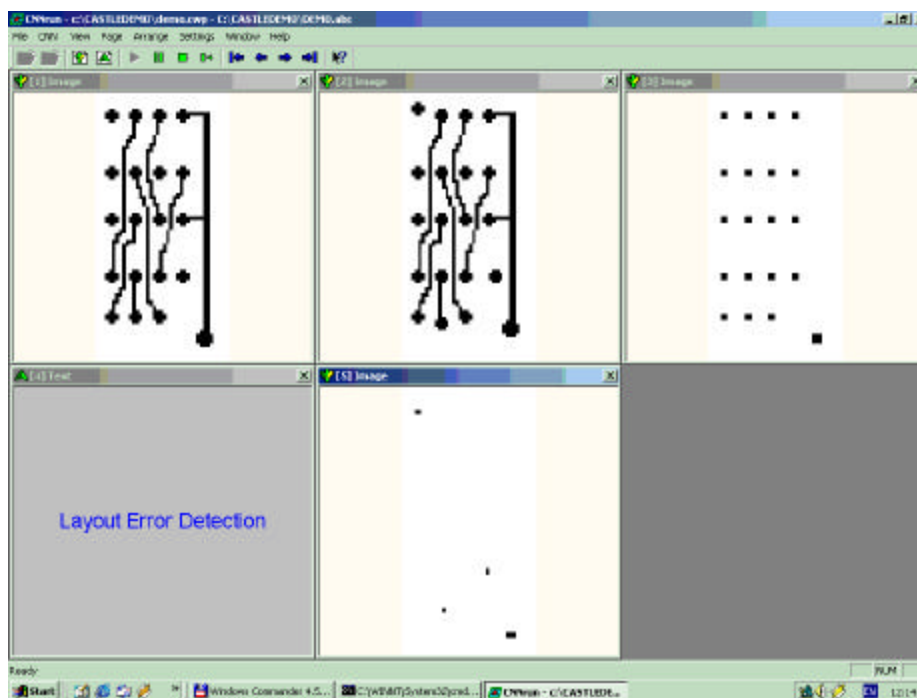
Az összeszerelt adapter- és DSP kártya valamint a platform látható a következő ábrán.



3.5. ábra A DSP kártya és a PCI illesztő adapter összeszerelt állapotban

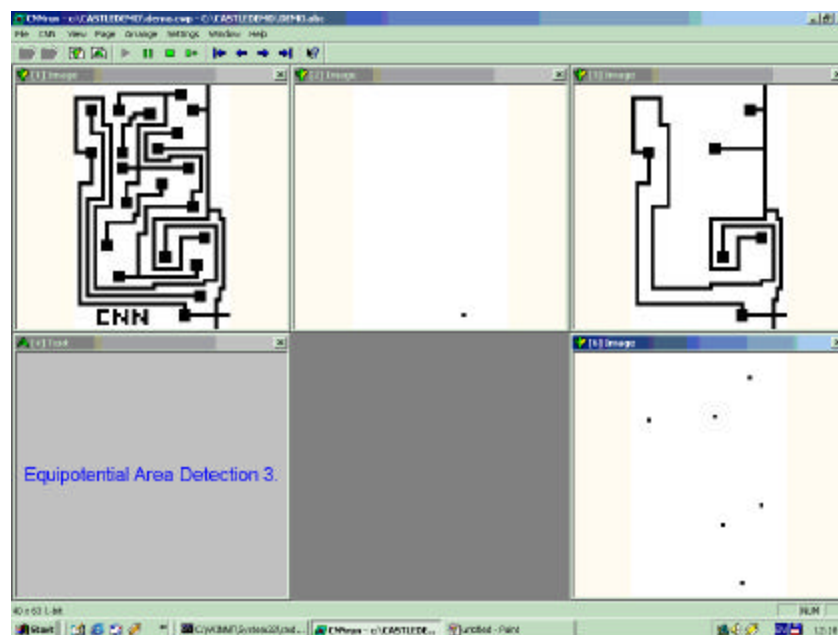
Az előbb bemutatott logikai processzoron is lefuttattam az illesztési hibákat detektáló és a zárlatkereső algoritmusaimat. A következő ábrákon az algoritmusok bemeneti és kimeneti képei láthatók. A képek mérete 40*80 pixel. A kép szélességét nem lehet megváltoztatni, egy processzor (tehát nem CNN tömb!) csak 40 pixel méretű képet képes feldolgozni. A kép hossza viszont korlátlan lehet.

A 3.5. ábrán látható az illesztési hibákat detektáló algoritmus futása. A felső sor közepén látható a rossz mesterfilm, jobboldalon pedig az előre kifűrt panel képe. Az alsó sorban az algoritmus kimeneti képe látható.



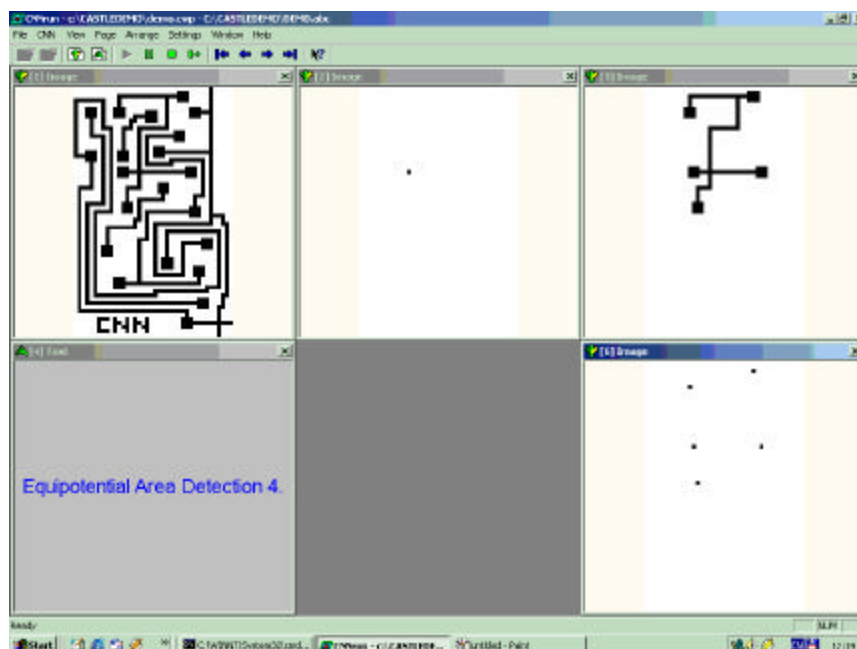
3.5. ábra Az illesztési hibákat kereső algoritmus futtatása

A következő ábra mutatja a zárlatkereso algoritmus egy részletének a futtatását. A felső sor baloldalán a teljes nyomtatott áramkör van, középen a marker kép, ahonnan indítjuk a bináris hullámot. A jobboldalon találjuk azt az ekvipotenciális jelvezetékét, amelyet felépítettünk a markerképen. Az alsó kép mutatja a PAD-ek helyét.



3.5. ábra A zárlatkereso algoritmus egy részletének futtatása

Ugyanez az algoritmusrészlet futott le a következő ábrán, csak a markerképen más jelvezetékét jelöltük ki, és így más PAD-eket találtunk meg.

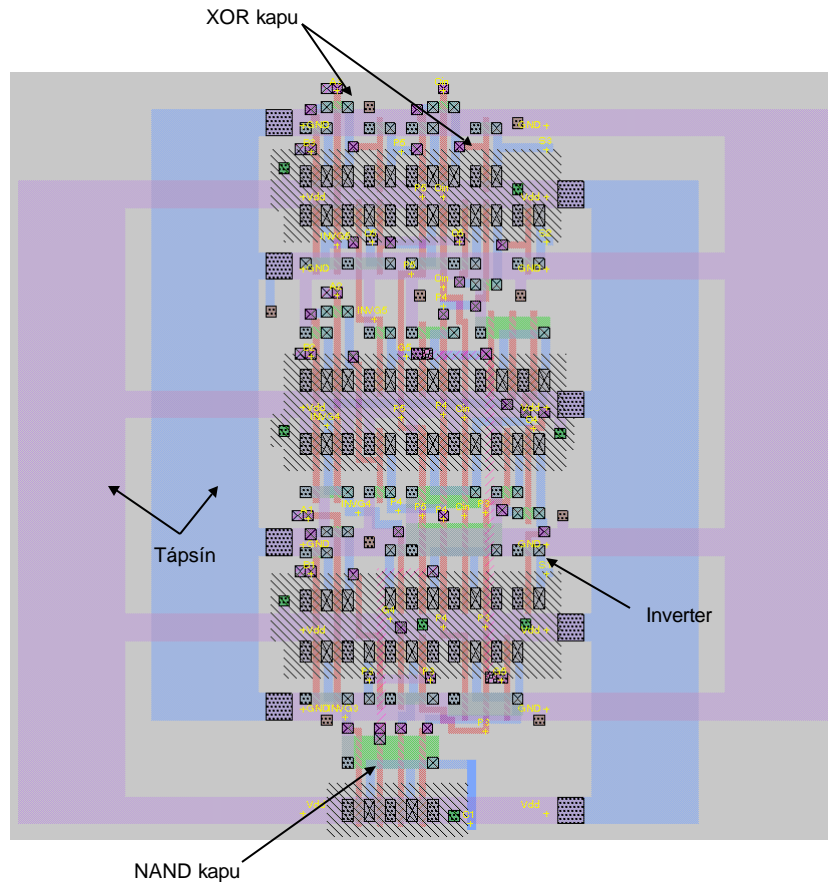


3.6. ábra A zárlatkereso algoritmus egy részletének futtatása

4. Függelék (A CASTLE)

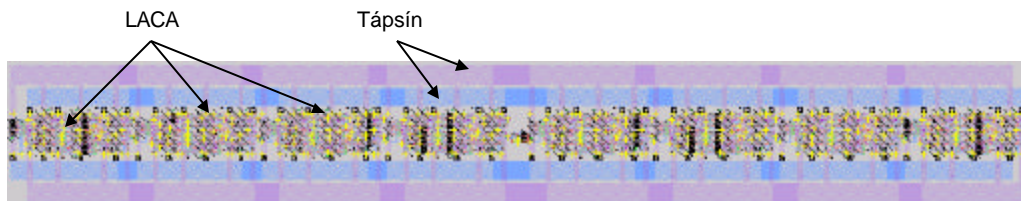
4.1 Néhány egység layout ábrája

Egy párhuzamos képzésű, három bites összeadó (LACA) layout ábrája látható az 4.1 ábrán. A kék szín az első, a lila a második fémréteg.



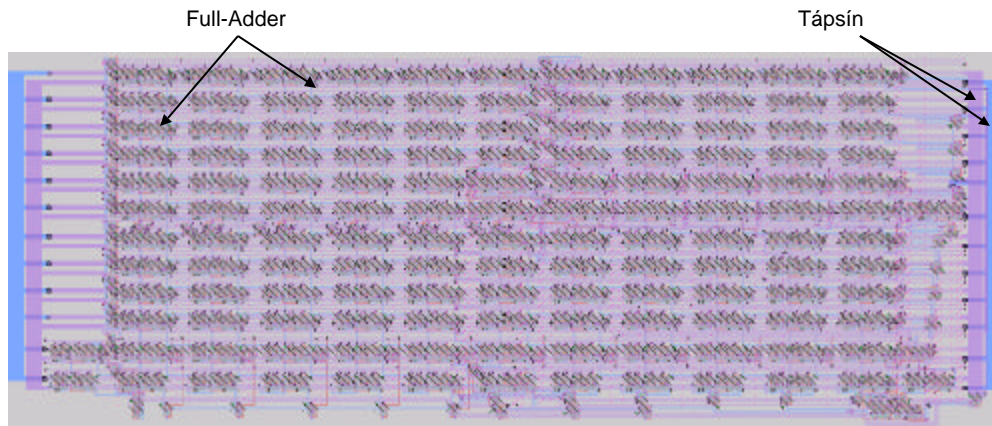
4.1. ábra LACA

Ezekből az ún. "LACA" egységekből épül fel a szorzó (4.2. ábra). Ezen a képen 2*4 darab LACA látható.



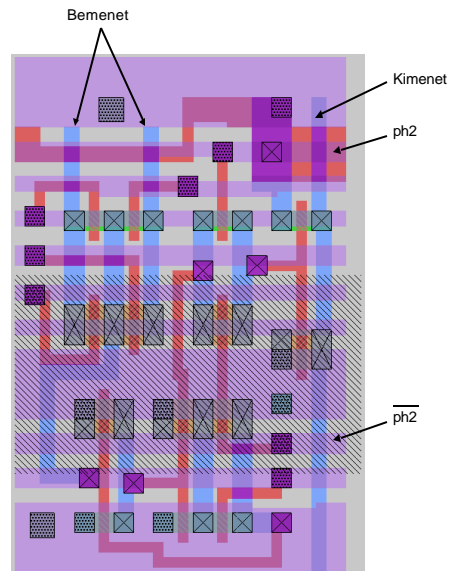
4.2. ábra Az összeadó

A szorzóegység layoutképét mutatja az 4.3. ábra.



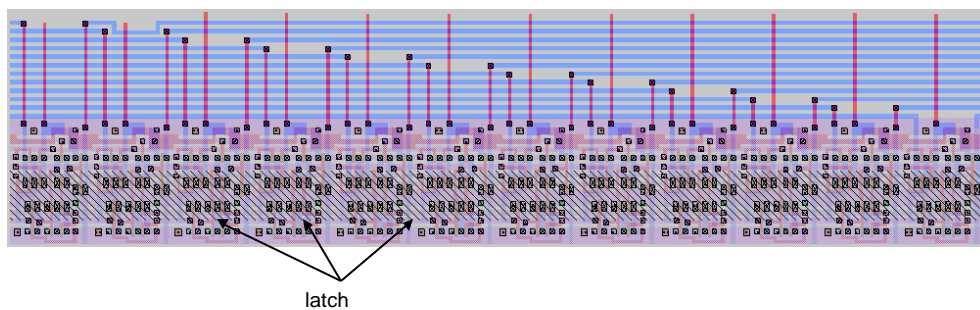
4.3. ábra A szorzóegység

A latch az egyik legkisebb építőelem, amit használtunk az ASIC tervezés során. Ennek több változata létezik, az 4.4. ábra egy kétbemenetű változatot mutat.



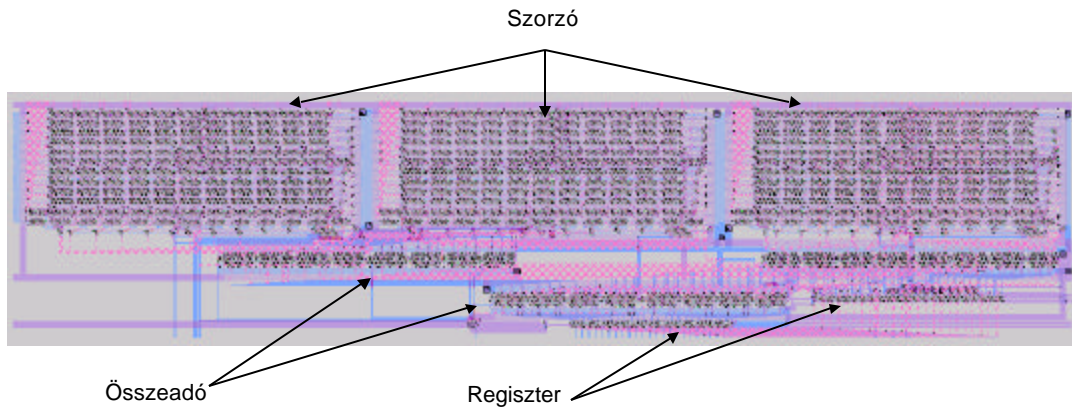
4.4. ábra Egy kétfemenetű latch

Ilyen latch-ekből épül fel egy regiszter (4.5. ábra).



4.5. ábra Regiszter latch-ekből felépítve

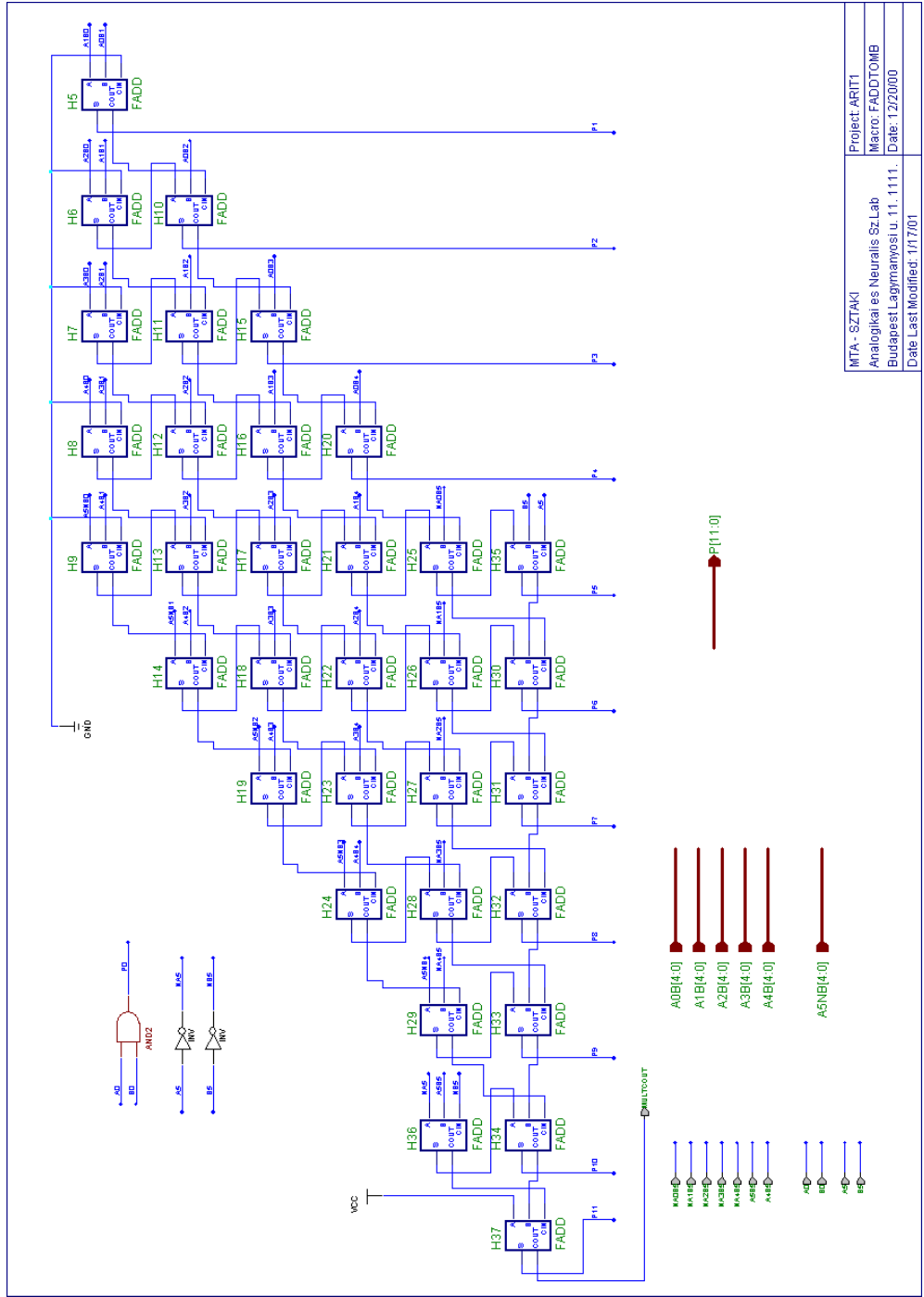
Az elobb bemutatott nagyobb egységekből áll az aritmetikai mag (4.6. ábra).



4.6. ábra A teljes aritmetikai egység

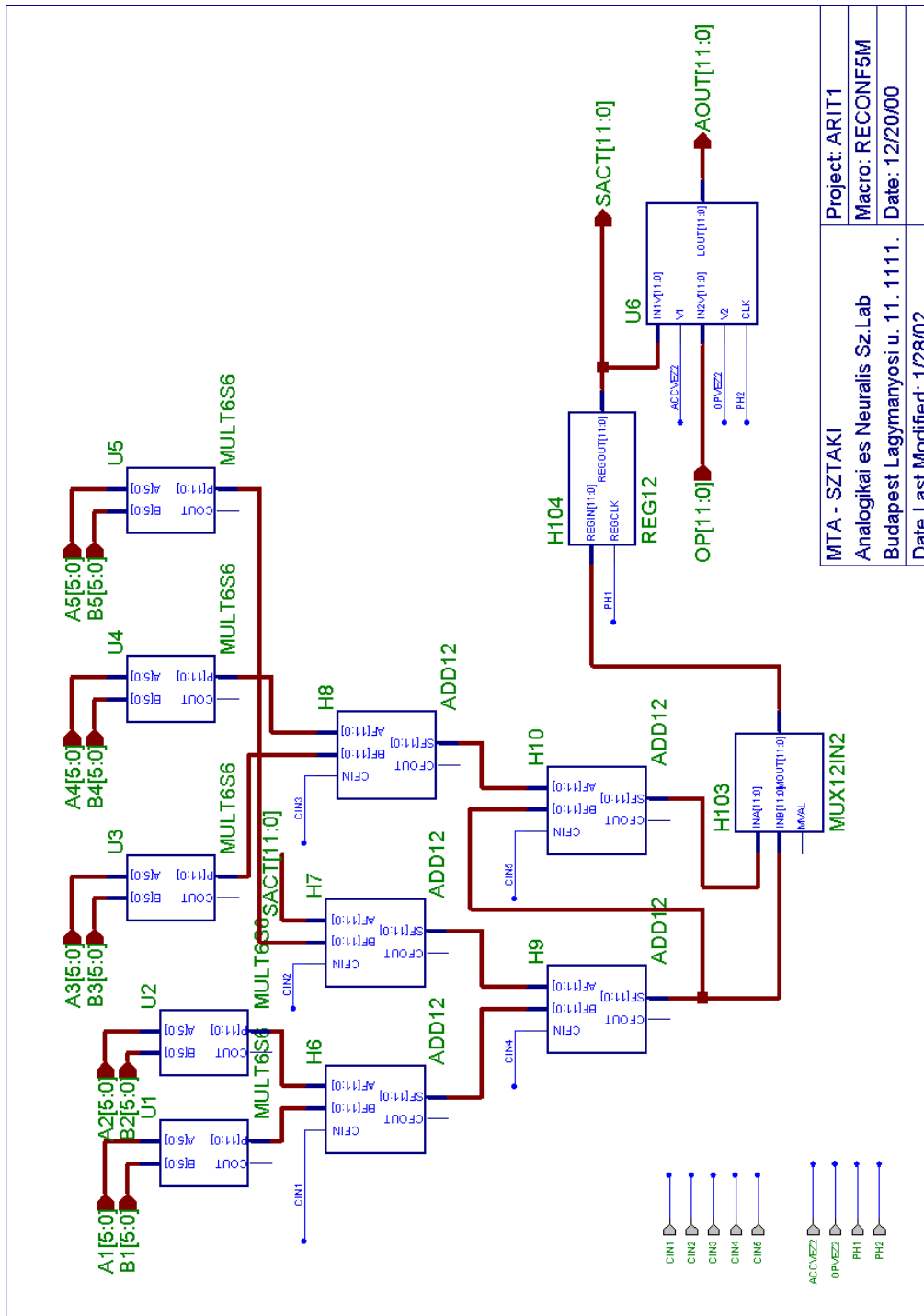
4.2 Néhány egység VIRTEX schematic rajza

Az 4.7. ábra egy 6*6-os Baugh-Wooley szorzómodult ábrázol, amely csak egy-fajta Full-Adder-ekből épül fel.



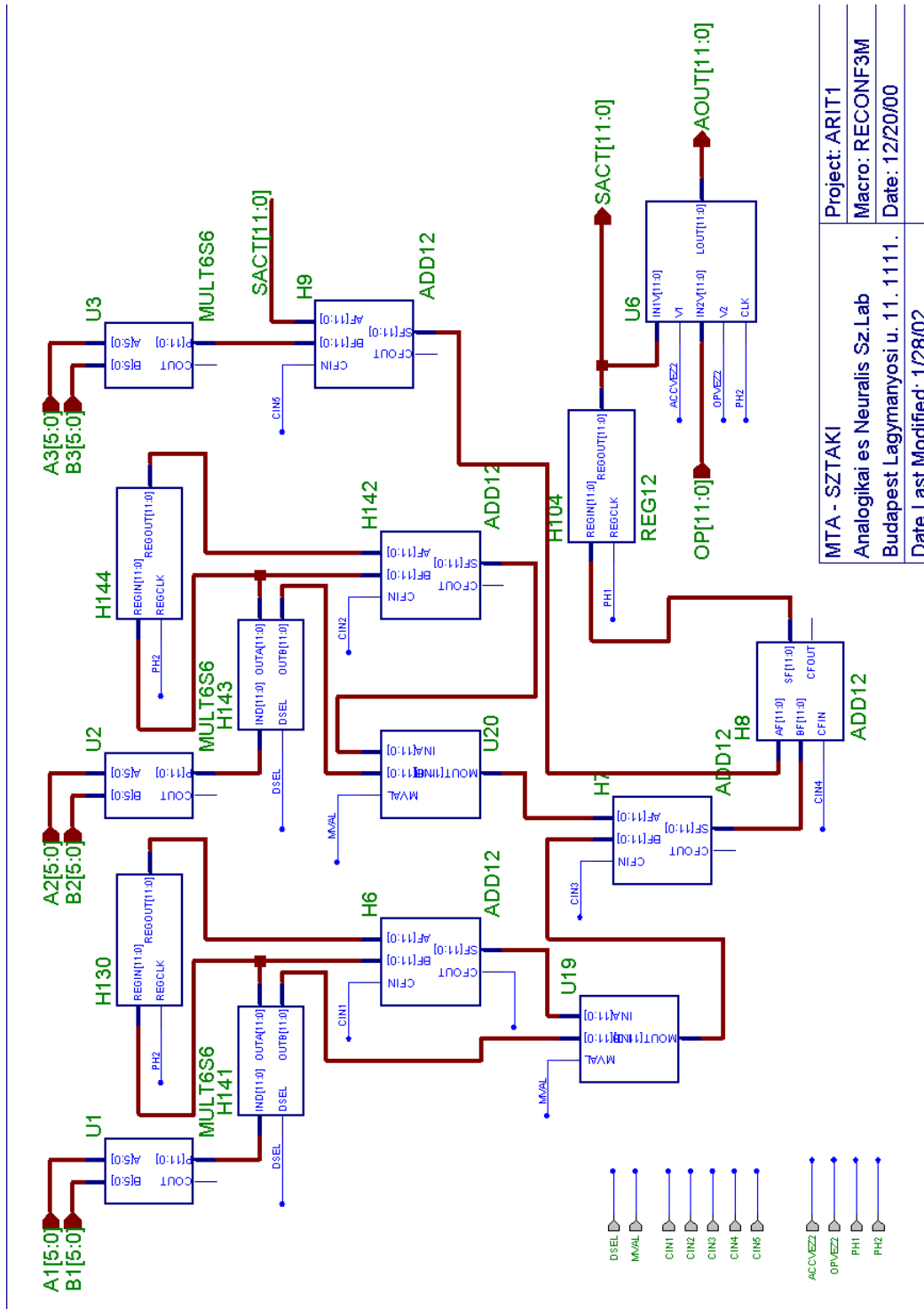
4.7. ábra Egy 6*6-os Baugh-Wooley szorzómodul

A hatodik fejezetben mutattam be az általam megadott újrakonfigurálható aritmetikai egységet. Ezeket az aritmetikai egységeket is elkészíttem VIRTEX FPGA-n, a 4.8. ábra mutatja az öt szorzóból álló aritmetikai magot.



4.8. ábra Az öt szorzóból álló aritmetikai egység

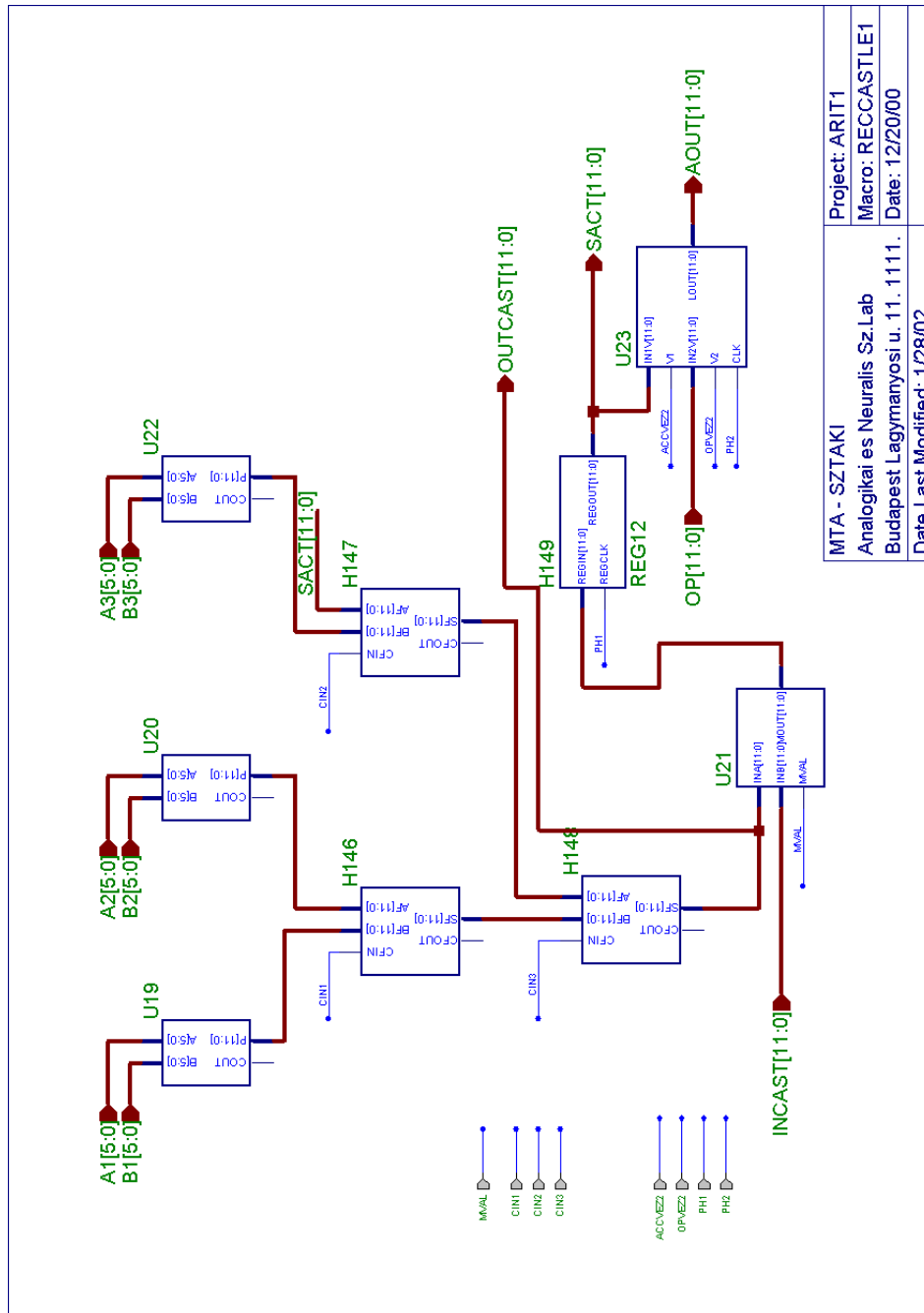
Ezen az oldalon (4.9. ábra) látható a három szorzóból felépülő, hely szerint optimalizált újrakonfigurálható aritmetikai mag.



Project: ARIT1
Macro: RECONF3M
Date: 12/20/00
MTA - SZTAKI
Analogikai és Neuralis Sz.Lab
Budapest Lagymányosi u. 11. 1111.
Date Last Modified: 1/28/02

4.9. ábra A teljes aritmetikai egység

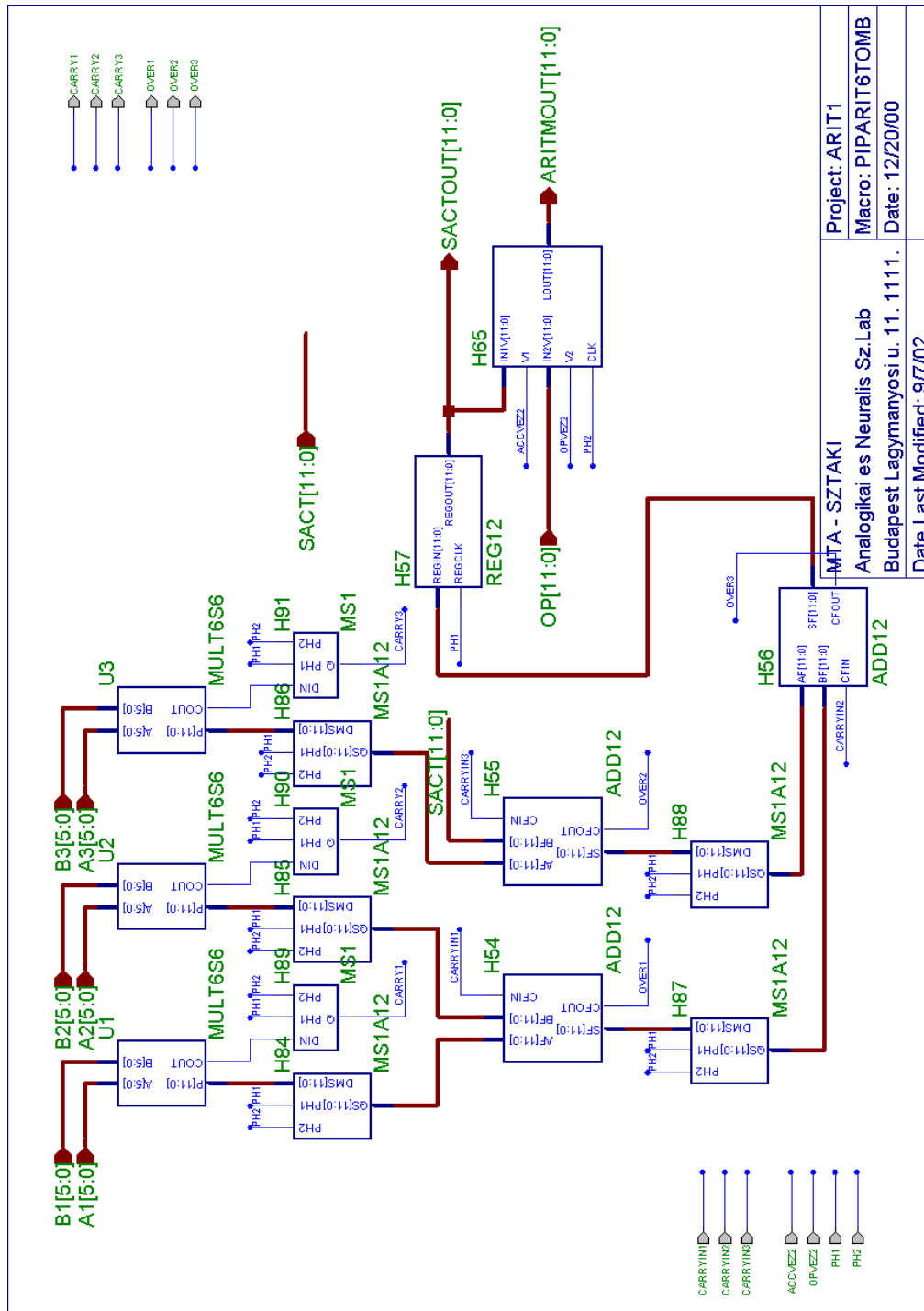
Elkészült VIRTEX FPGA alá az az újrakonfigurálható aritmetikai egység változata is, amely működési sebesség szerint optimalizált. Ez két egyforma CASTLE aritmetikai magból épül fel (kiegészítve egy multiplexerrel). Az egyik egység látható az 4.10. ábrán.



MTA - SZTAKI	Project: ARIT1
Analogikai es Neuralis Sz.Lab	Macro: RECCASTLE1
Budapest Lagymanyosi u. 11. 1111.	Date: 12/20/00
Date Last Modified: 1/28/02	

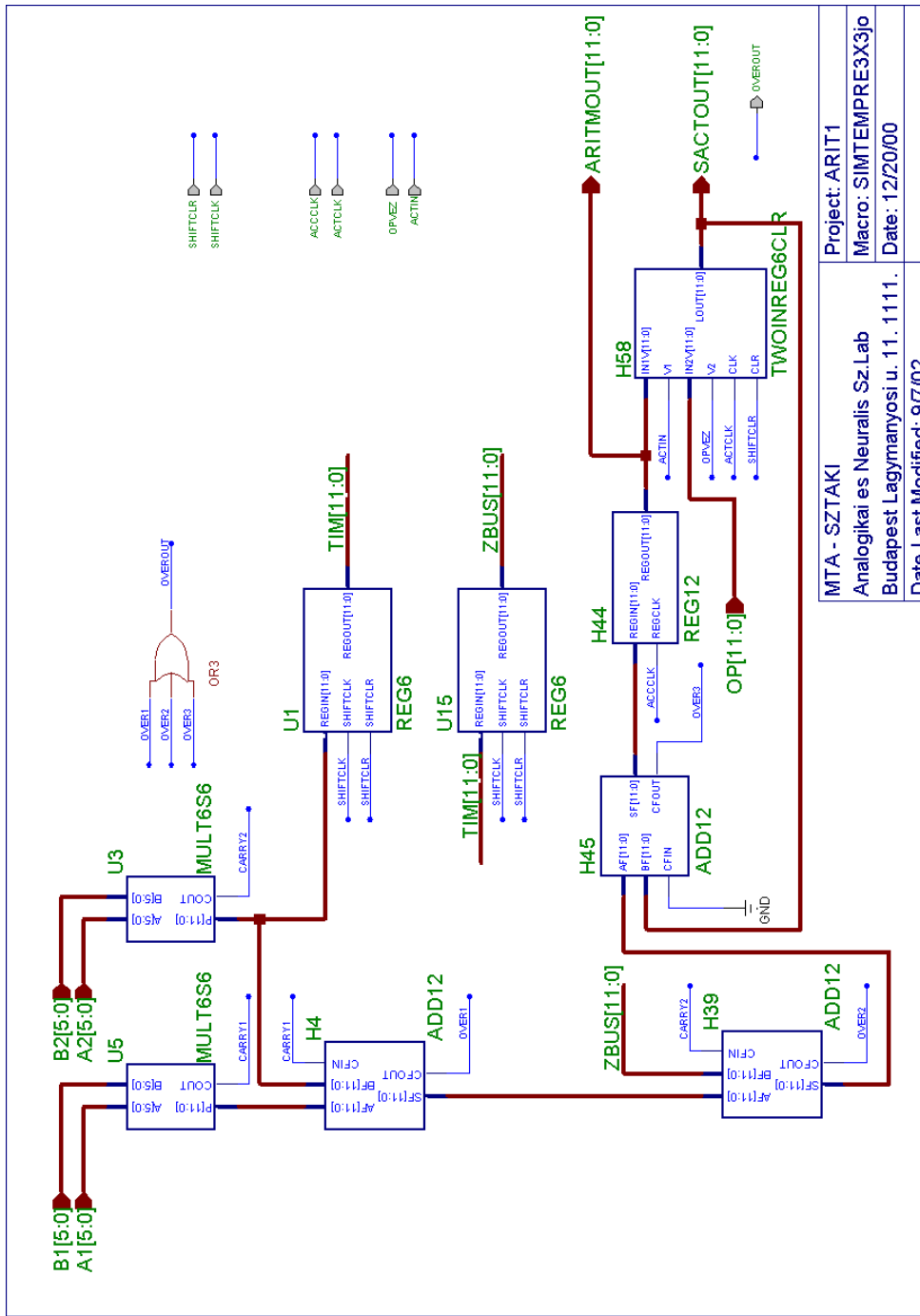
4.10. ábra A sebesség szerint optimalizált aritmetikai egység

A pipeline-nal kiegészített aritmetikai egységet is teszteltem a VIRTEX FPGA alatt (4.11. ábra).



4.11. ábra A pipeline-nal kiegészített aritmetikai egység

Az 4.12. ábra mutatja a két szorzóból álló aritmetikai egységet, amely csak szimmetrikus template-ekkel képes számolni.



Project: ARIT1
Macro: SIMTEMPRE3X3jo
Date: 12/20/00
MTA - SZTAKI
Analogikai es Neuralis Sz.Lab
Budapest Lagymanyi u. 1. 1111.
Date Last Modified: 9/7/02

4.12. ábra A két szorzóból álló aritmetikai egység