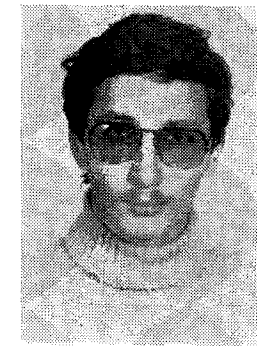
16 bites szorzó áramkör logikai tervezése a részegységek analóg szimulációjával

SZŐKE SÁNDOR
Mikroelektronikai Vállalat — BME/HEI*



ÖSSZEFOGLALÁS

A cikk rövid betekintést ad a digitális szorzók világába, majd részletesebben ismertet egy 16×16+31 bites szorzó—összeadó áramkört. A teljes áramkör logikai tervezése és szimulációja, valamint az egyes részegységek analóg szimulációja a MEV-ben levő tervező rendszer segítségével történt.

SZŐKE SÁNDOR

A Budapesti Műszaki Egyetem Villamosmérnöki Karának Híradástechnika Szakán szerzett oklevelet 1983-ban. Jelenleg a Mikroelektronikai Vállalat fejlesztő mérnökeként a BME Híradástechnikai Elektronika Intézetében folytat szakmérnöki tanulmányokat, a digitális jelfeldolgozásban felhasználható integrált áramkörök tervezését tanulmányozza. Szűkebb témája a digitális szorzók.

1. Bevezetés

Napjainkban a digitális jelfeldolgozás egyre több analóg területre tör be. A kommersz elektronikában történő tömeges elterjedésének alapfeltétele, hogy nagy mennyiségben álljanak rendelkezésre olcsó, e feladatra specializált integrált áramkörök. Az igényt felismerve a vezető nyugati félvezetőgyártók (INTEL, AMI, NEC, BELL, TI, IBM, ITT, INTERMETALL) kifejlesztették egy chipes digitális jelprocesszoraikat (DSP) [1]. Széles körűen felhasználhatók azonban olyan kisebb komplexitású IC-k is, amelyek nagy sebességgel képesek digitális számok szorzására (TRW, MMI). A MEV-ben jelenleg megbízhatóan működő 5—6 μm-es NMOS technológia lehetővé teszi egy 16×16+31 bites szorzó-összeadó áramkör gyártását. Mivel a szocialista piacon ilyen áramkör jelenleg nem szerezhető be, komoly keresletre számíthatunk.

2. Szorzó algoritmusok

Egy szorzás eredményét a szorzandók értéke egyértelműen meghatározza, így elvileg nincs akadálya, hogy egy N×M bites szorzót egy N+M be-, illetve kimenettel rendelkező kombinációs hálózattal realizáljunk. Ez az átviteli függvények gyors bonyolódása miatt gyakorlatilag csak néhány bitre valósítható meg, nagyobb szorzók minden esetben részszorzatok — igen gyakran bitszorzatok — összegzésével állítják elő az eredményt. A szorzó működése — s így tervezése is — két fázisra bontható:

— a részszorzatok előállítása az összeszorzandó számokból,
— a részszorzatok összegzésével a szorzat kiszámítása.

A szorzókat a részszorzatok előállításának és összeadásának alapján két csoportba sorolhatjuk.

A szekvenciális (SEQUENTIAL) szorzók jellemzője, hogy minden ciklusban ugyanaz az áramköri egység szolgáltatja az aktuális részszorzatot, s hozzáadja az előző ciklusból származó részeredményhez, így a szorzatot egyetlen egység többszöri felhasználásával nyerjük. Hardware igényük viszonylag kicsi, de bonyolult vezérlő jeleket kívánnak. Legfőbb hátrányuk, hogy a számolás időbeli széttolása és a ciklusok között szükséges adatmozgatások miatt a műveleti idő megnő.

A párhuzamos (PARALLEL) szorzókban egy egység csak egy pozíción vesz részt a számolásban, minden funkciót külön megvalósított hálózat lát el. A részeredmények közvetlenül a következő fokozat bemenetére jutnak, így a sebességet csak a tényleges számolásra használt idő szabja meg. A műveletek nagyfokú párhuzamosítása, a holt idők csökkentése által lehetőség nyílik olyan struktúrák kialakítására, amelyek a sebesség jelentős növekedéséhez vezetnek. A műveleti idő csökkentése általában a komplexitás és a fogyasztás növekedése mellett érhető el.

Tekintsük először a legegyszerűbb esetet, amikor egy N és egy M bites pozitív egészként értelmezett számot akarunk összeszorozni:

$$P=\left(\sum_{i=0}^{N-1} A_i 2^i\right)\times\left(\sum_{j=0}^{M-1} B_j 2^j\right) \qquad (1)$$

Ha (1) kifejezést átalakítjuk a (2)-nek megfelelő formába, azonnal adódik a léptet és összead (SHIFT AND ADD) szorzási módszer.

$$P=\sum_{i=0}^{N-1} 2^i(A_i B)$$
$$\text{Itt } A=\sum_{i=0}^{N-1} A_i 2^i \text{ és } B=\sum_{j=0}^{M-1} B_j 2^j. \qquad (2)$$

Az összeadandó bitek képe a helyiértéküknek megfelelően az *1. ábrán* látható, ahol minden bitet egy × karakter jelképez. Az összegzés megvalósítható például egy N bit hosszúságú összeadó sorral akár szekvenciálisan ugyanazon blokk ismételt alkalma-

Beérkezett: 1985. II. 5. (∧)

* MEV Szőke Sándor munkahelyét nappali szakmérnöki tanulmányai idejére a BME/HEI-be helyezte ki.

zásával, akár párhuzamosan több blokk megépítésével. A bitek előállítása bitszorzatokkal vagyis AND kapcsolatokkal történhet. Az algoritmus jól használható előjeles abszolút értékes formában ábrázolt számok szorzására is. Az előjelüktől megfosztott számokat pozitív egészként összeszorozzuk, az eredmény előjele egy EXOR kapuval meghatározható.

A digitális jelfeldolgozásban legelterjedtebb a 2's komplemens kód használata, ahol az előjegy (MSB bit) negatív súlyozása felborítja a műveleti homogenitást. Egy 2's komplemens kódban ábrázolt N bites szám értéke:

$$\mathrm{val}\{A\} = -a_0 2^0 + \sum_{i=1}^{N-1} a_i 2^{-i}. \qquad (3)$$

Ilyen formában kódolt számok szorzására is több módszer terjedt el. Szekvenciális szorzókban szívesen alkalmazzák a Booth-algoritmust [2], mert ez mind az M lépésben azonos műveleti szabályok szerint számolja az eredményt. A szorzó aktuális két bitjét vizsgálva az előző részeredmény feléhez vagy hozzáadja, vagy abból kivonja a szorzandót, vagy pedig nem végez műveletet. A kivonás megvalósítható a szorzandó komplemensének hozzáadásával, ami a bitek negálását és az LSB helyen +1 hozzáadását igényli. A műveletvégző egység így az összeadó-kivonó áramkör helyett egy egyszerűbb, csak összeadó hálózatot tartalmaz, az összeadandó tagok előállításához azonban az AND kapuk helyett itt egy bonyolultabb kapcsolásra van szükség, amely minden ciklusban a vezérlő jelektől függően vagy a_i, vagy $\overline{a_i}$ biteket állít elő.

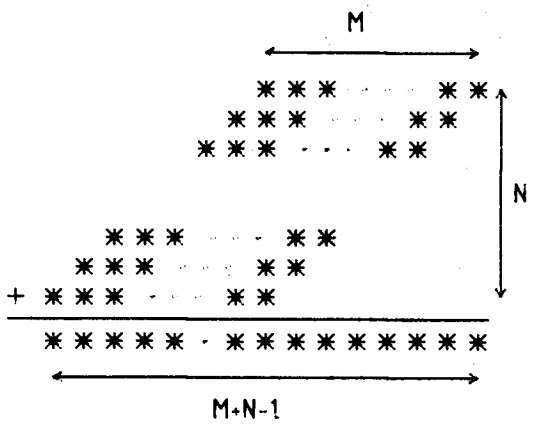
Párhuzamos szorzó áramkörök tervezésénél gyakran alkalmazott megoldás, hogy különféle korrekciók bevezetésével visszatérnek az *1. ábrának* megfelelő struktúrához és algoritmushoz. Alakítsuk át (3)-t és nézzük meg, hogyan változik a szorzat kifejezése:

$$\mathrm{val}\{A\} = -a_0 2^1 + \sum_{i=0}^{N-1} a_i 2^{-i} \qquad (4)$$

$$\mathrm{val}\{P\} = \left(-a_0 2^1 + \sum_{i=0}^{N-1} a_i 2^{-i}\right) \times \left(-b_0 2^1 + \sum_{j=0}^{M-1} b_j 2^{-j}\right) =$$

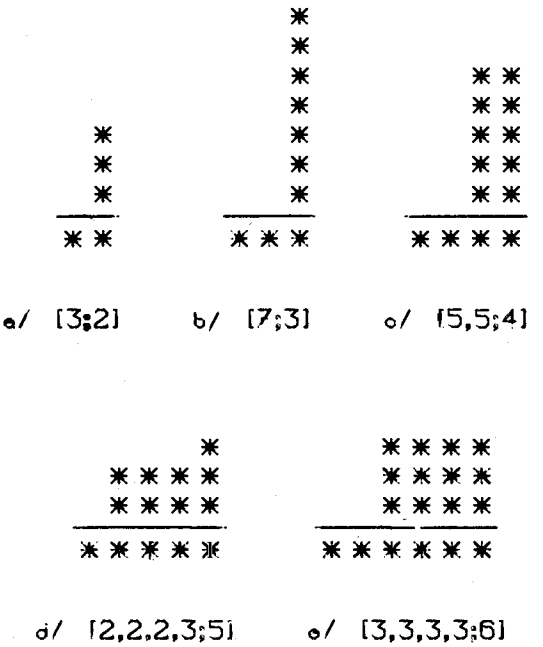
$$= \sum_i a_i 2^{-i} \times \sum_j b_j 2^{-j} + 4a_0 b_0 - 2a_0 \sum_j b_j 2^{-j} - 2b_0 \sum_i a_i 2^{-i}. \qquad (5)$$

Látható, hogy egy lehetséges megoldás az utólagos (EXPLICIT) korrekció. Az (5) első tagjának értéke az első módszerrel számolható, a második az ábrázolható tartományon kívülre esik, így figyelmen kívül hagyható. A két kivonás a 2's komplemens hozzáadásával realizálható, ami két újabb összeadó sor megvalósítását igényli. Ezt az eljárást régebben alkalmazták, ha a már meglevő, drága (nagy méretű kártyán kb. 100 IC-t tartalmazó, pl. [4]) szorzó egységet akarták felhasználni az újabb igényeknek megfelelő 2's komplemens kódban végzett számolásra. Az ezután építendő áramkört eleve ilyen kódra tervezzük, s a szükséges korrekciókat már a számolás közben figyelembe vehetjük (IMPLICIT algoritmusok). A műveleti homogenitás ezzel természetesen megbomlik, de ez párhuzamos szorzóknál nem alap-



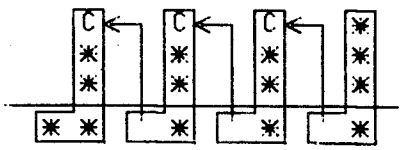
H35-1

1. ábra. Az összeadandó bitek képe egy N és egy M bites pozitív szám szorzásakor



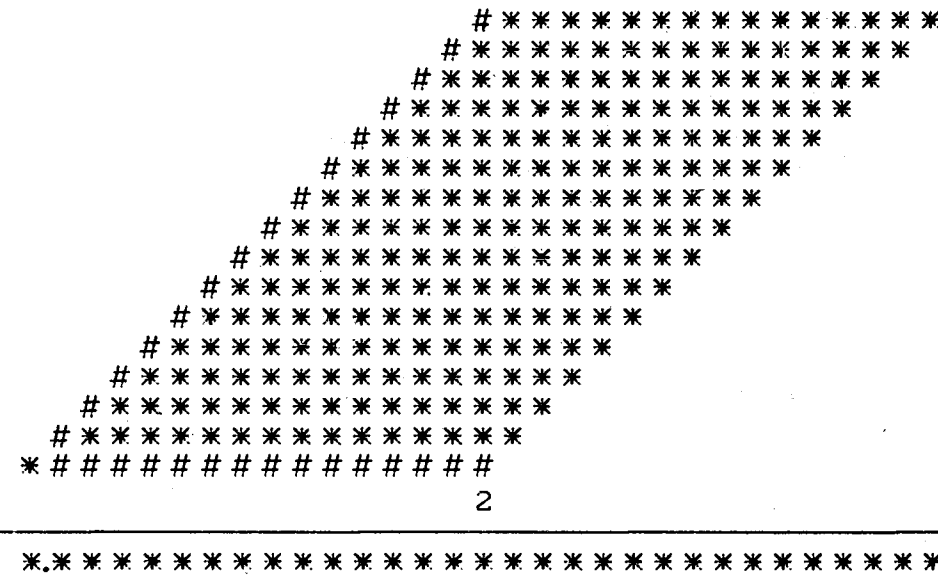
H35-2

2. ábra. Néhány gyakoribb összeadó egység



H35-3

3. ábra. 2/d előállítása 4 db 2/a blokkból



4. ábra. Az összeadandó bitek képe a 16×16 bites 2' s komplemensű szorzónál

vető követelmény, csak a tervezést könnyítő tulajdonság. Egy ilyen implicit algoritmus ismertetése található a következő pontban.

Vizsgáljuk meg ezek után a részszorzatok összeadásának problémáját! A bithalmaz redukálása különféle összeadó blokkok alkalmazásával történhet. Egy általános összeadó egység K bemenő bitből L kimenő bitet állít elő. Minél nagyobb a (K—L) érték, annál kevesebb összeadót kell használnunk. Az alkalmazandó cella kiválasztásánál természetesen figyelembe kell venni a műveleti sebességet is, hiszen több, de lényegesen gyorsabb fokozat eredményezhet eredőben gyorsabb áramkört. Néhány tipikusnak mondható cella látható a *2. ábrán* [5] alapján. A legegyszerűbb közülük az 1 bites teljes összeadóként ismert egység (FULL ADDER), amely 3 bemenő bit (a_i, b_i és egy kisebb helyiértékről származó átvitel) felhasználásával 2 kimenő bitet (szumma és átvitel) számol.

Megfigyelhetjük, hogy a $2/d$ egységgel megegyező funkciójú áramkör összeállítható 4 db FA felhasználásával a *3. ábra* szerint. Ugyanígy egymás mellé helyezve N egységet összeadhatunk két N bites számot, és még marad is egy szabad bemenetünk az LSB helyiértéken, ami pl. a komplemens hozzáadásával megvalósított kivonás esetén felhasználható a +LSB érték figyelembe vételére a számolási idő növelése nélkül. Leggyakoribb megoldás, hogy ilyen összeadó sorok segítségével az *1. ábrának* megfelelő bitképet soronként összegzik, mivel ez egy szabályos, jól áttekinthető hardware struktúrát eredményez.

A párhuzamos megvalósítás adta lehetőségeket lényegesen jobban kihasználó algoritmust ismertet Vuillemin, ahol *ld*(N) lépésben a szomszédos részeredmények páronkénti összegzésével jut el a végeredményhez [6].

Legáltalánosabb esetben a gyorsaság fokozása érdekében tetszőleges, a feladatra orientált összeadó hálózat alkalmazható, de a szabálytalan struktúra nehezen áttekinthető, számos hibalehetőséget rejt magában. Az áramkör tervezése és ellenőrzése nagyobb munkát igényel, s ezzel a tervezési időt és költséget, végső soron az integrált áramkör árát növeli.

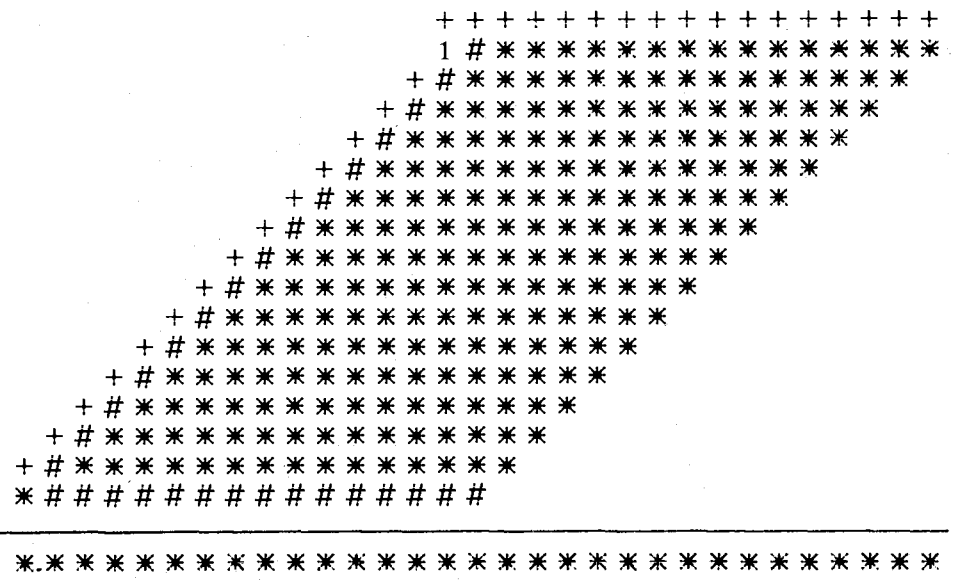
3. A tervezéshez használt algoritmus

A szorzó áramkört a szabványok, a piaci igények és a technológiai lehetőségek figyelembevételével 16×16 bitesre terveztük, így célszerű a felhasznált algoritmust — az általánosság megszorítása nélkül — erre a konkrét esetre levezetni. A feladat tehát a 16 bites, 2's komplemens kódban ábrázolt A és B számok összeszorzása. A szorzat kiszámításához használjuk a (3) képlet szerinti értelmezést:

$$\text{val}\{P\} = \left(-a_0 2^0 + \sum_{i=1}^{15} a_i 2^{-i}\right) \times \left(-b_0 2^0 + \sum_{j=1}^{15} b_j 2^{-j}\right) =$$

$$= a_0 b_0 + \sum_{i=1}^{15}\sum_{j=1}^{15} a_i b_j 2^{-i-j} - a_0 \sum_{j=1}^{15} b_j 2^{-j} - b_0 \sum_{i=1}^{15} a_i 2^{-i} \quad (6)$$

Az első két tag a számolás során nem okoz problémát, de mivel hálózatunkban csak összeadó cellákat szeretnénk használni, a két kivonást vissza kell vezetnünk összeadásra. Az előjegyüktől megfosztott



H35 - 5

5. ábra. Az összeadandó bitek képe a $16\times16+31$ bites szorzó-összeadónál

A és B számok pozitív értéket képviselnek, így kiegészíthetők egy 0-s előjeggyel. Az így nyert számokat jelöljük vesszővel (7) definíciók szerint:

$$a'_0=b'_0=0; \quad a'_i=a_i \quad \text{és} \quad b'_j=b_j, \quad i, j\rangle 0. \tag{7}$$

Jelöljük S-sel a (6) első két tagjának összegét. A szorzat értéke (6) és (7) alapján:

$$\text{val}\{P\}=S-\sum_{j=0}^{15} a_0 b'_j 2^{-j}-\sum_{i=0}^{15} b_0 a'_i 2^{-i} \tag{8}$$

A kivonásokat a komplemensek hozzáadásával helyettesítve:

$$\text{val}\{P\}=S+\sum_{i=0}^{15}\overline{a_0 b'_j}\,2^{-j}+2^{-15}+\sum_{i=0}^{15}\overline{b_0 a'_i}\,2^{-i}+2^{-15}. \tag{9}$$

Mivel $a'_0=b'_0=0$, $i=0$ és $j=0$ esetén keletkezik egy-egy 2^0 tag, amelyek összege 2. Ezen konstans érték az ábrázolható tartományon kívülre kerül, így ez a két tag elhagyható a szummából. A megmaradó vesszős tagok megegyeznek az eredeti értékekkel, így azokat visszaírva a szorzat értékére a következő végeredményt kapjuk:

$$\text{val}\{P\}=a_0 b_0+\sum_{i=1}^{15}\sum_{j=1}^{15} a_i b_j 2^{-i-j}+\sum_{j=1}^{15}\overline{a_0 b_j}\,2^{-j}+2^{-15}+$$

$$+\sum_{i=1}^{15}\overline{b_0 a_i}\,2^{-i}+2^{-15}. \tag{10}$$

Az egyenlet jobb oldalán levő szummákat felbontva, majd a tagokat 2 hatványai alapján csoportosítva meghatározhatjuk, hogy melyik bitszorzatot milyen helyiértéken kell figyelembe venni. Az összeadandó bitek képe a *4. ábrán* látható, ahol # karakter jelöli azokat a biteket, amelyeket negáltan kell előállítani. A bitszorzatok generálásában ez nem okoz különösebb nehézséget, minden bitet egy egyszerű logikai alapkapu szolgáltat.

A $16\times16+2$ bit összeadása legegyszerűbben 15 db 16 bites összeadó sorral történhet. Ha megkeressük a jelterjedés szempontjából leghosszabb utat, azt találjuk, hogy a számolási idő:

$$T\,\text{max}=29\,T_c+15\,T_s, \tag{11}$$

ahol T_c a carry, T_s pedig a szumma képzésének ideje egy FA cellában. A képletből jól látható, hogy a soros átvitelterjedés igen sok időt emészt fel, megszüntetése gyorsítaná az áramkört. Ez átvitel megőrző (CARRY SAVE = CS) összeadó sorok használatával érhető el. A CS összeadás lényege, hogy a vízszintes átvitelterjedés helyett a keletkező C értékeket egy következő fokozat bemenetére vezetjük. Az összeadás eredménye így egy N bites C és egy N bites S jellegű szám lesz, amelyek együttesen képviselik az összeg értékét. A felszabaduló C bemenetek következtében az összeadó sor 3 számot képes fogadni, tehát 3 db N bites szám vagy egy normál és egy CS szám összeadására képes, az eredmény CS formában áll elő. Az N bites összeadás mindössze 1 T_s időt igényel, mivel a FA cellák egymástól függetlenül, azonos időben működnek, s a számolási idő független az összeadó sor hosszától.

14 db 16 bites összeadó sor alkalmazásával az eredmény 14 T_s idő alatt előáll 16 bites CS formában, amiből még képezni kell a 2's komplemens kódú végeredményt. Ha ezt a legegyszerűbb módon, egy soros átvitelterjedéses 16 bites összeadóval végez-

nénk el, ehhez (15 $T_c+1\ T_s$) időre lenne szükség, ami még így is 14 T_c-nyi gyorsulást jelent az eredeti megoldáshoz képest. A végső összeadás azonban gyorsabban is elvégezhető átvitelgyorsító blokkokat tartalmazó összeadóval [2]. Ennek műveleti ideje a megtervezett kapcsolásban kb. (6 $T_c+1\ T_s$)-nek felel meg, így végeredményben a *CS* összeadó sorok és az átvitelgyorsítós összeadó bevezetésével 23 T_c-vel csökkent a számolási idő. Az átvitelgyorsító blokkok az áramkör méretét növelik, ami azonban az összkomplexitáshoz mérten nem jelentős.

A csak szorzó áramkörhöz képest lényegesen nagyobb keresletre számíthat egy szorzó-összeadó. A szorzás párhuzamos megvalósítása lehetővé teszi a kapcsolás viszonylag egyszerű, gazdaságos továbbfejlesztését. A két 16 bites számot összeszorzó, s a szorzathoz egy 31 bites számot adó áramkör bitelrendezése látható az *5. ábrán*, ahol + jelöli a hozzáadandó szám bitjeit. Ebben az elrendezésben az előző kapcsoláshoz képest eggyel több *FA* sorra van szükség, s minden sor 17 bites, az eredmény 16 bites *CS* szám. Az új funkció bevezetése miatt a műveleti idő 1 T_s-sel megnőtt, a *FA* cellák számának változása $(15\times17-14\times16)=31$. A kritikus paraméter, a sebesség tehát nem változott jelentősen, de a komplexitás kb. 10%-kal nőtt.

A főbb paramétereket összefoglalva az átvitelterjedéses csak szorzó algoritmus műveleti ideje:

$$T(MUL)=29\ T_c+15\ T_s. \tag{12}$$

A szorzó-összeadó számolási ideje:

$$T(MAD)=16\ T_s+5\ T_c. \tag{13}$$

Az előzetes becslések szerint 1 $T_s=2\ T_c$, így:

$$T(MAD)/T(MUL)=0{,}63, \tag{14}$$

tehát az átvitelmegőrzős és az átvitelgyorsítós összeadások bevezetésével sikerült a műveleti időt a 63%-ára csökkenteni az áramköri funkció bővülése mellett. A komplexitás növekedése kb. 25%-os.

4. Logikai szimuláció

A logikai szimulációhoz az egész hálózatot meg kell tervezni logikai kapu szinten, amit nagyon megkönnyíthet a szabályosan ismétlődő részegységekből felépülő struktúra. A szorzó meghatározó alapeleme az egy bites *FA* cella, így annak optimalizálása a tervezés legfontosabb feladata. Az összeadó cellát definiáló alapegyenletek:

$$S=A\oplus B\oplus C \tag{15}$$

$$CO=AB+AC+BC \tag{16}$$

Könnyen belátható a függvények szimmetria tulajdonsága:

$$\overline{S}=\overline{A}\oplus\overline{B}\oplus\overline{C} \tag{17}$$

$$\overline{CO}=\overline{A}\,\overline{B}+\overline{A}\,\overline{C}+\overline{B}\,\overline{C}, \tag{18}$$

ami gyakorlatilag azt jelenti, hogy ugyanaz az áramkör akár pozitív, akár negatív logikában korrekciók nélkül használható. Számos megoldást összehasonlítva sebességük és komplexitásuk alapján a következő függvényeket megvalósító cellára esett a választás:

$$\overline{S}=\overline{CO}(A+C+B)+ABC \tag{19}$$

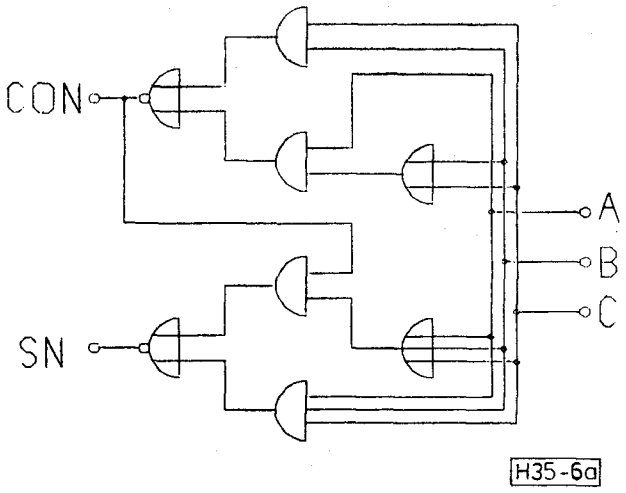
$$\overline{CO}=\overline{C}(A+B)+AB \tag{20}$$

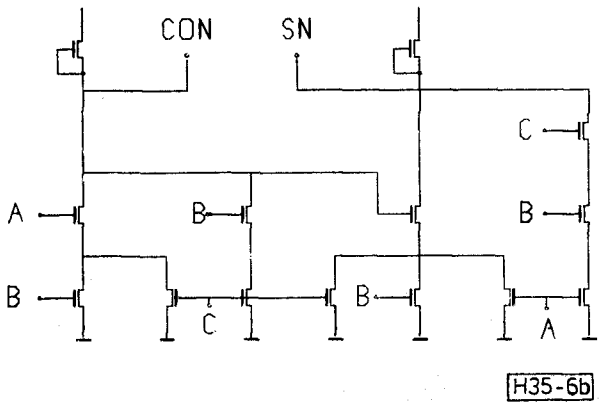
A (19), (20) egyenleteket vizsgálva azt tapasztaljuk, hogy a $\overline{C}$ és $\overline{S}$ kimenetek két komplex kapuval realizálhatók, s ezekből egy-egy inverter állítja elő *C*-t és *S*-t. Csökkentené a komplexitást, ha elhagyhatnánk az invertereket, s a negált kimeneteket tudnánk felhasználni a további számolás során. A függvény szimmetriatulajdonságát kihasználva a negált kimeneteket tekinthetjük ellentétes logikában értelmezett ponált értékeknek is, amit a következő, ellentétes logikában működő fokozat képes feldolgozni. A váltott logika használatánál arra kell ügyelni, hogy az egyes cellák bemenetei azonosan legyenek értelmezve. Ez egyszerűen megvalósítható a *CS* összeadók jóvoltából, az egymást követő sorokat felváltva pozitív, illetve negatív logikában működtetve. Ez azt eredményezi, hogy az *5. ábra* minden második sorában az algoritmus szerinti bitek negáltját kell előállítani, de így is minden bit kifejezhető $\overline{A_iB_j}$ vagy $\overline{\overline{A_i}+\overline{B_j}}$ alakban.

Egy adott sorban minden bit generálásához azonos logikában értelmezett b_j-re van szükség, míg az egymást követő sorokban az a_i-k felváltva ponált, majd negált értékkel szerepelnek. A b_j értékek a tápfeszültséggel párhuzamosan fémezéssel vezethetők végig a mátrix sorain, az erre merőlegesen futó a_i-kat poliSimal vagyunk kénytelenek továbbítani, ami a nagy távolság (3—4 mm) miatt nagy késleltetést jelentene. Mivel úgyis minden sorban az előző sorban levő értékek negáltjaira van szükség, azt az összes pozíción egy-egy inverterrel állítjuk elő, ami egyúttal a jel soronkénti frissítését is eredményezi. Igaz, hogy ez több mint 200 plusz inverter megvalósítását igényli, de ezek a sebesség szempontjából nem kritikus úton helyezkednek el, így kis területen kis fogyasztással megvalósíthatók.

Az *5. ábra* szerinti elrendezésben a bitek a helyiértéküknek megfelelően helyezkednek el, ami paralelogramma alakú chipet eredményezne. A megvalósítás során a sorokat egymás alá toljuk, így a *C* függőlegesen lefelé, a *S* jobbra lefelé lép a következő fokozat bemenetére. A téglalap alakú struktúra jobb szélén keletkezik a 15 db kisebb helyiértékű eredménybit, az alján pedig a 16 bites *CS* szám. A felhasznált *FA* cella logikai és tranzisztorszintű kapcsolása látható a *6. ábrán*.

A *CS* szorzatot a 16 bites, kétszintű átvitelgyorsítót tartalmazó összeadó fogadja, amelynek konkrét kapcsolása a definiáló egyenletek technológiához illeszkedő átalakításával nyerhető [2]. A logikai szimulációs programban a kapukésleltetések megfelelő megválasztásával elérhető, hogy a futtatások során $T_c=6$ nsec adódjon, ami reális érték egy komplex kapu késleltetésének figyelembevételére. Az eredmény a kipróbált bemenő adatokra minden esetben a becsült 37 T_c-nek megfelelő 222 nsec-on belül adódott. A 6 nsec-os becslés helyességét az analóg futtatások eredményei megerősítették.





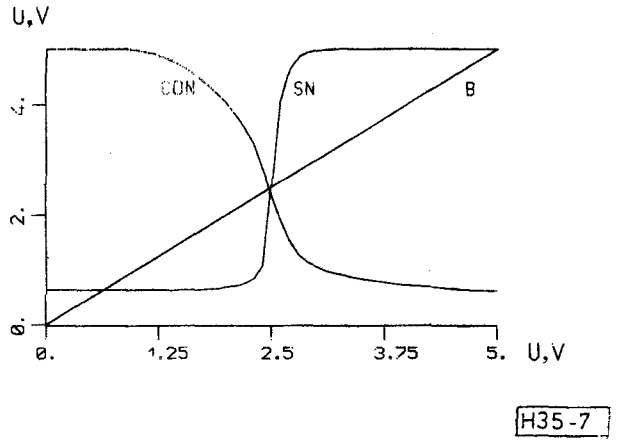
6. ábra. Az összeadó cella logikai kapu és tranzisztorszintű kapcsolási rajza

5. Analóg szimuláció

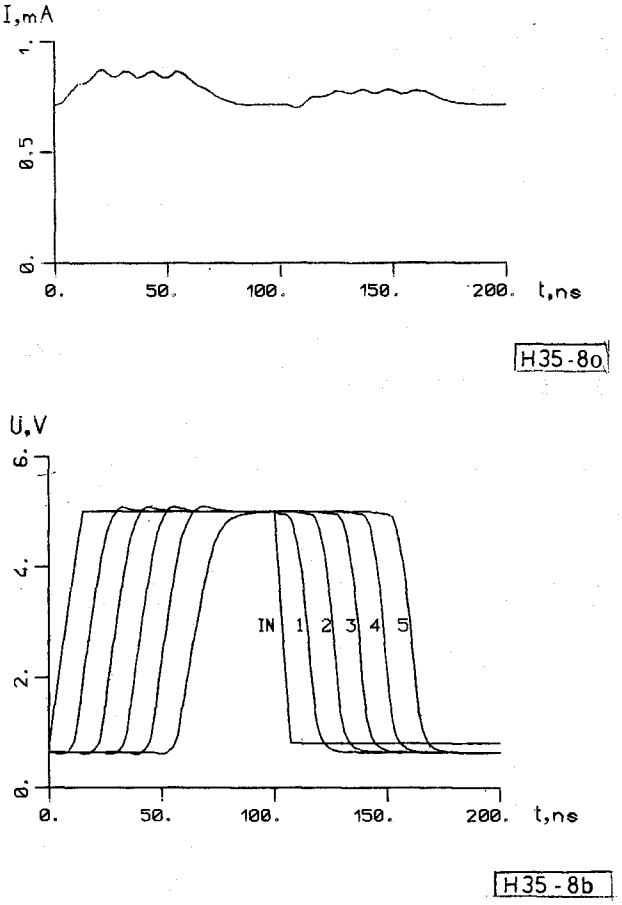
Az analóg szimulációval kapcsolatban elöljáróban meg kell említeni, hogy a szimulációs eredmények a technológiai és szimulációs bizonytalanságok miatt csak tájékoztató jellegű értékek, amelyek hasznos információkat szolgáltatnak a tervezés során, nem képesek azonban arra, hogy egy áramkör működését előre pontosan megadják.

Az analóg szimulációs futtatások a TKI-ban kifejlesztett ANAL20 nevű MOS analízis program felhasználásával készültek, amely MOS tranzisztorokból, ellenállásokból, kapacitásokból és jelforrásokból felépülő hálózatok *DC* és tranziens viselkedésének modellezésére képes. A program nagy előnye, hogy az áramkörleírás bemenő paraméterei a layoutról vett méretek, így az analóg szimuláció jól definiált kapcsolatban van a tényleges áramkörrel.

A legfontosabb feladat itt is a *FA* cella tervezése volt. A meghajtó tranzisztorok *L* értékét (csatorna hossz) célszerű a technológia által megengedett minimális értékre választani, s a szükséges csatorna szélességeket *(W)* a szimuláció során meghatározni. A számolt tranzisztor méretek felhasználásával több layout terv is készült, az ezek alapján pontosított áramkörleírások szimulációi gyakorlatilag azonos eredményeket szolgáltattak. Ez azzal magyarázható, hogy a késleltetéseket okozó *R* és *C* értékeket döntően a tranzisztorok aktív területének mérete határozza meg, az összeköttetések csak nagy távolságok esetén okoznak jelentős lassulást (poliSi > 1 mm). Az összeadó cella *DC* transzfer karakterisztikája látható a *7. ábrán*. A vezérelt bemenet a *B* jelű, a vizsgált kimenetek *CN* és *SN*. Az ábra alapján a logikai küszöbfeszültség kb. 2·3 V. Mindkét kapu erősítése a küszöbfeszültség környezetében > 1 (a kimenő jel meredeksége nagyobb a bemenőénél), ami a zavarelnyomás miatt fontos követelmény.



7. ábra. A FA cella DC transzfer karakterisztikája



8. ábra. A FA cella tranziens jellemzői: *a)* áramfelvétel, *b)* tranziens feszültségátvitel

A *FA* cella késleltetését sorba kötött blokkok tranziens vizsgálatával lehet meghatározni. A program 5 összeadót tartalmazó sort volt képes feldolgozni, ennek futtatási eredményei láthatók a *8. ábrán*. Az 5 fokozat késleltetése 60 nsec-ra adódott, amiből T_c-re a logikai futtatások során használt 6 nsec becsülhető. Az 5 cella együttes áramfelvétele még logikai szint váltáskor sem haladja meg az 1 mA-t, így az összeadó hálózat fogyasztása kisebb lesz mint 50 mA. Az inverterek együttes áramfelvétele max. 10 mA-re becsülhető, így a teljes szorzó egységé kisebb 60 mA-nál. A fogyasztás 5 V-os tápfeszültségnél kisebb 300 mW-nál, amit a 40 lábú tok könnyedén eldisszipálhat. A fogyasztás és sebesség paraméterek az igényeknek megfelelően változtathatók a tranzisztorok W/L arányának módosításával. Növekvő értékgyorsulást és nagyobb fogyasztást eredményez, míg a csökkentés hatására kisebb fogyasztású, de lassúbb áramkört nyerünk.

6. Összefoglalás

A digitális szorzók rövid áttekintése után egy konkrét párhuzamos szorzó áramkör tervezését láthattuk. Az áramkör összeszoroz 2 2's komplemens kódban adott 16 bites számot, s eközben a szorzathoz hozzáad egy 31 bites ugyancsak 2's komplemensű számot. A kivonásokat egy implicit algoritmussal visszavezettük csak összeadást igénylő struktúrára, így *CS* összeadó mátrixot használhattunk. A *CS* eredményt egy kétszintű átvitelgyorsítót tartalmazó összeadó segítségével alakítjuk át 2's komplemens kódra. A használt layout méretekkel a szorzó-összeadó 300 mW-os fogyasztás mellett max. 222 nsec alatt szolgáltatja az eredményt, ami nyugati publikációkkal összevetve is jó eredménynek tekinthető. Nem árt még egyszer megemlíteni, hogy ezek az adatok csak fenntartásokkal fogadhatók el, s az input-output egységek nélkül jellemzik magát az aktív műveletvégző egységet.

7. Köszönetnyilvánítás

Végezetül szeretném köszönetemet nyilvánítani dr. Gordos Gézának és Tuzson Tibornak munkámat segítő hasznos ötleteikért, az áramkör tervezéséhez és a cikk megírásához nyújtott segítségükért.

IRODALOM

[1] *Tuzson Tibor, Asztalos András:* VLSI Digital Signal Processing Structures and their Feasibility, Proceedings of the Third Symposium on Microcomputer and Microprocessor Application, Bp., 18—21 October, 1983 vol. I. pp. 148—164.

[2] *Rupprich Péter:* Digitális aritmetika, BME Mérnöktovábbképző jegyzet.

[3] *Gibson, J. A., Gibbard, R. W.:* Synthesis and Comparison of Two's Complement Parallel Multipliers, IEEE Transactions on Computers, October 1975 vol. C—24, No. 10 pp., 1020—1027.

[4] *Pezaris, S. D.:* A 40-ns 17-bit by 17-bit Array Multiplier, IEEE Transactions on Computers, Apr. 1971 vol. C—20, pp. 442—447.

[5] *Stenzel, W. J., Kubitz, W. J., Garcia, G. H.:* A Compact High-Speed Parallel Multiplication Scheme, IEEE Transactions on Computers, Oct. 1977. vol. C—26, No. 10 pp., 948—957.

[6] *Jean Vuillemin:* A Very Fast Multiplication Algorithm for VLSI Implementation, North-Holland Publishing Company Integration, The VLSI Journal 1983 pp. 39—52.