Kisfogyasztású integrált áramkörök tervezési kérdései

VARGA LÁSZLÓ, HOSSZÚ GÁBOR

Budapesti Műszaki és Gazdaságtudományi Egyetem, Elektronikus Eszközök Tanszék
hosszu@nimrud.eet.bme.hu

Lektorált

Kulcsszavak: kisfogyasztású CMOS áramkör, adiabatikus töltésátvitel, töltésvisszanyerő kapcsolás

Napjaink elektronika alkalmazásainál a kis fogyasztás alapvető követelmény, különösen telepes készülékeknél. Ilyenek például az emberi implantátumok, az alkalmi (ad-hoc) érzékelő hálózatok, valamint egyéb, hordozható berendezések. Ezeknek a követelményeknek a kielégítésére lehetőséget nyújtanak az adiabatikus töltésvisszanyerő kapcsolások, amelyek fő jellemzője, hogy az áramköri kapacitások feltöltése és kisütése Joule-veszteség nélkül történhet. A cikk első része bemutatja az ismert adiabatikus áramkörök egy javított változatát, majd annak alkalmazását kisfogyasztású dinamikus tárolók megvalósítására. A második rész a kisfogyasztású áramkörök tervezésének különleges feladatát, a működés optimális ütemezését tárgyalja.

1. Bevezetés

A nagy bonyolultságú integrált áramkörök teljesítményfelvételének csökkentésére több ismert módszer létezik (telepfeszültség csökkentés, jelváltások számának minimalizálása), azonban ezek egyikével sem érhető el nagyságrendi csökkenés. Ha az áramkör sebessége nem túl kritikus (mint ahogy ez több alkalmazásban előfordul), akkor az adiabatikus kapcsolások nyújtják a legkedvezőbb megoldást [1]. A hőtanból átvett „adiabatikus" elnevezés arra utal, hogy a generátor által szolgáltatott energia veszteség nélkül alakítható át az általa feltöltött kapacitás energiájává. Továbbá az így feltöltött kapacitás töltését visszanyerve és valamilyen formában ideiglenesen tárolva, az a következő ciklusban újra felhasználható.

2. Az adiabatikus töltésvisszanyerő logikai kapcsolások működése

Ha egy kapacitást a szokásos ugrásfüggvénnyel töltünk fel, akkor a kapcsoló soros ellenállásán eső feszültség miatt jelentős (Joule) teljesítmény-veszteség lép fel. Ha a feltöltést egy olyan feszültségforrás végzi, amelynek feszültsége csak infinitezimálisan nagyobb a kapacitás aktuális feszültségénél, akkor a töltőáram, és ezzel a soros ellenálláson fellépő teljesítmény is a zérushoz tart. Ez tehát egy igen jó hatásfokú feltöltés, hátránya viszont, hogy a művelet ideje eközben a végtelenhez tart. Látható tehát, hogy így nem valósíthatók meg nagysebességű áramkörök. Egy közbülső megoldás az *1. ábrán* látható rámpafeszültség alkalmazása, amely jelentős fogyasztáscsökkenés mellett még elfogadható működési sebességet enged meg.

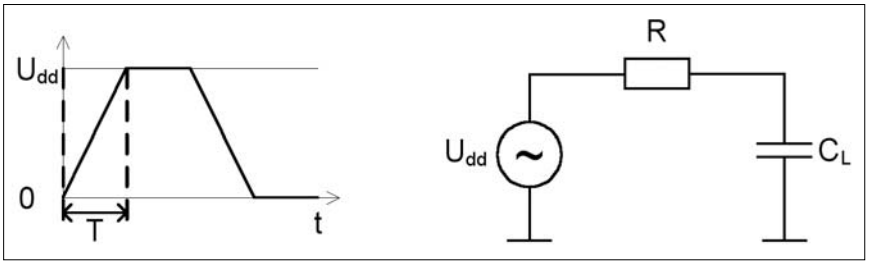
Ha az U_{dd} bemeneti feszültséget lassan emelve a C_L kapacitást T ideig töltjük, úgy, hogy T sokkal nagyobb, mint a kapcsolás RC időállandója, akkor a kapacitás feszültsége szorosan követi a bemeneti feszültséget és a kapacitásba befolyó áramot közel állandónak tekinthetjük: $I \approx (C_L \cdot U_{dd}) / T$. A töltési folyamat energiaveszteségét a következő képlet fejezi ki (1):

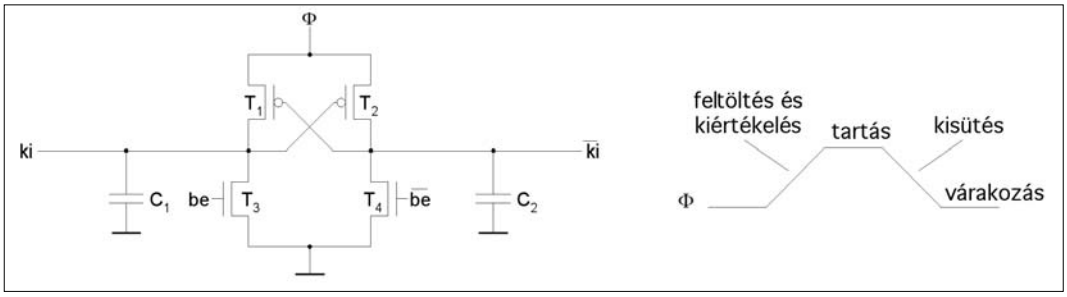
$$E_{\text{veszteség}} = P \cdot T = I^2 R \cdot T = \left(C_L \frac{U_{dd}}{T} \right)^2 R \cdot T = \frac{R \cdot C_L}{T} C_L \cdot U_{dd}^2$$

Az *(1) képlet* az úgynevezett adiabatikus veszteség értékét adja. A gyakorlatban a lassan emelkedő jellel történő táplálást rezonáns ütemező (fázis-) jelekkel valósítják meg, amelyeknél a kapacitásból visszanyert töltést átmenetileg egy induktivitás tárolja. A logikai magas szintet alkotó energia így mintegy „hintázik" az áramkör és az induktivitás között, az áramköri kapacitásnak és így a rezgés frekvenciájának állandóságához pedig mind a ponált, mind a negált logika megvalósítása szükséges.

Az irodalomban található adiabatikus töltésvisszanyerő megoldások közül egyszerűségében és működési sebességében kitűnik a *2N-2P kapcsolás* [2], amelynek invertere a következő oldalon, a *2. ábrán* látható. Az ábrán látható a működéséhez szükséges fázisjel elvi alakja is, amely a feltöltés és egyben kiértékelés, a tartás, a kisütés és a várakozás periódusokból áll.

1. ábra Adiabatikus töltés szemléltetése





2. ábra

A 2N-2P inverter és négy szakaszból álló fázisjele

A be=1 esetén T_3 kinyit és C_1 a földre kapcsolódik, míg T_4 lezár és így C_2 lebeg. Ha a felfutó Φ fázisjel értéke eléri a T_2 *pMOS* tranzisztor küszöbfeszültségének abszolút értékét ($|U_{t,p}|$-t), akkor T_2 kinyit, és innentől C_2 adiabatikusan töltődik. A negált kimenet feszültsége követi a fázisjel értékét. T_2 kinyitásának pillanatában egy rövid időre T_1 is kinyit, azonban C_2 feszültségének emelkedésével azonnal le is zár, így C_1 feltöltetlen marad, mivel T_3 még mindig nyitott.

Amint Φ fázisjel eléri a legnagyobb értékét, a kimeneteken érvényes logikai szint van (tartási periódus). A kisütési periódusban C_2 a feltöltéshez is használt útvonalon keresztül adiabatikusan kisül, egészen addig, míg a lefutó Φ fázisjel értéke $|U_{t,p}|$ alá nem csökken, és T_2 le nem zár. Ekkor a C_2 kapacitásban $|U_{t,p}|$ feszültség marad, ami nem gond, ha a következő fázisjel-ciklusban ugyancsak a C_2 lesz feltöltve. Ha azonban a fázisjel következő felfutásakor a kapu logikai állapota megváltozik, akkor a C_2 töltése a T_4-en keresztül nem-adiabatikusan sül ki, amelynek energiaveszteségе $E=0.5 \bullet C \bullet (U_{t,p})^2$. Ugyanekkora nem-adiabatikus veszteséget okoz az is, hogy feltöltéskor a pMOS tranzisztor nem nyit ki azonnal.

3. Javított tulajdonságú adiabatikus kapcsolás

A javított tulajdonságú adiabatikus kapcsolás alapinvertere, valamint a vezérlését és egyben a tápellátását is biztosító fázisjele a *3. ábrán* látható [3]. Működése a *2N-2P* kapcsoláséhoz hasonló, azzal kiegészítve, hogy feltöltéshez itt két áramút áll rendelkezésre. Az egyik a meglévő *pMOS* tranzisztoron halad keresztül, a másik pedig a hozzáadott T_3, illetve T_4 tranzisztoron, amely által a feltöltendő kimenet kezdettől fogva követi a fázisjel értékét, így feltöltéskor nem jön létre nem-adiabatikus veszteség.

A javított tulajdonságú adiabatikus kapcsolás és a *2N-2P* kapcsolás órajel-ciklusonkénti energiaveszteségét a frekvencia függvényében, különböző terhelőkapacitás értékek mellett a *4. ábrán* láthatjuk. A szimuláció ideális fázisjel-generátort feltételezve trapéz alakú fázisjelekkel, egyenlő feltöltés, tartás, kisütés és várakozás időkkel az AMS 0,8 µm-es technológiáján 100 darab láncba kapcsolt inverteren történt. Az inverterek láncba kapcsolásához négy, egymástól 90°-kal eltolt fázisjelre van szükség.

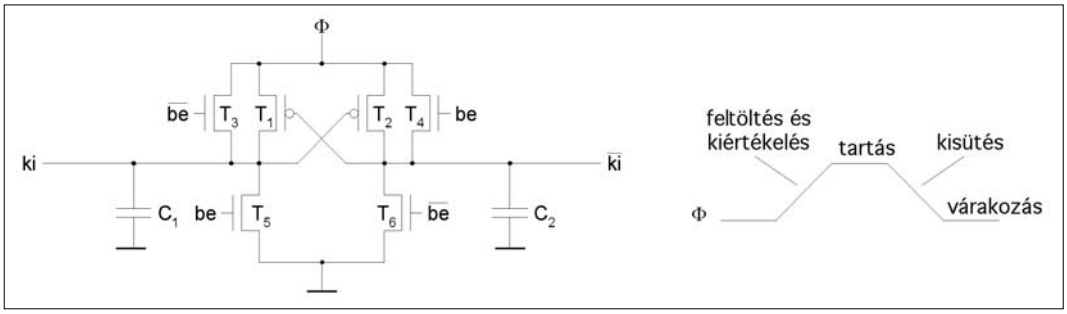
A *2N-2P* kapcsolás energiaveszteségе kis frekvenciákon igen jelentős, ami a lassú kapcsolási idő miatt a hosszabb ideig keresztbefolyó áramokból adódik. A javított tulajdonságú kapcsolás energiaveszteségе a *2N-2P* kapcsolásénál kedvezőbb, mindamellett, hogy a működési határfrekvenciája is magasabb. A működés első határfrekvenciájához közeledve az energiaveszteségnek a működési frekvenciától való függése az addigi lineáris helyett négyzetes lesz, mivel az adiabatikus töltési feltétel már csak kismértékben teljesül.

Az energiaveszteség tovább javítható teljesen adiabatikus kisütéssel, amihez egy külön kisütő áramútra van szükség. Az ezt megvalósító *nMOS* tranzisztor fázishelyes vezérlését a következő logikai fokozatban elhelyezett inverter adja, csak a fázisjel kisütés szakaszában engedélyezve azt.

4. Adiabatikus elven működő tárolóelem kialakítása

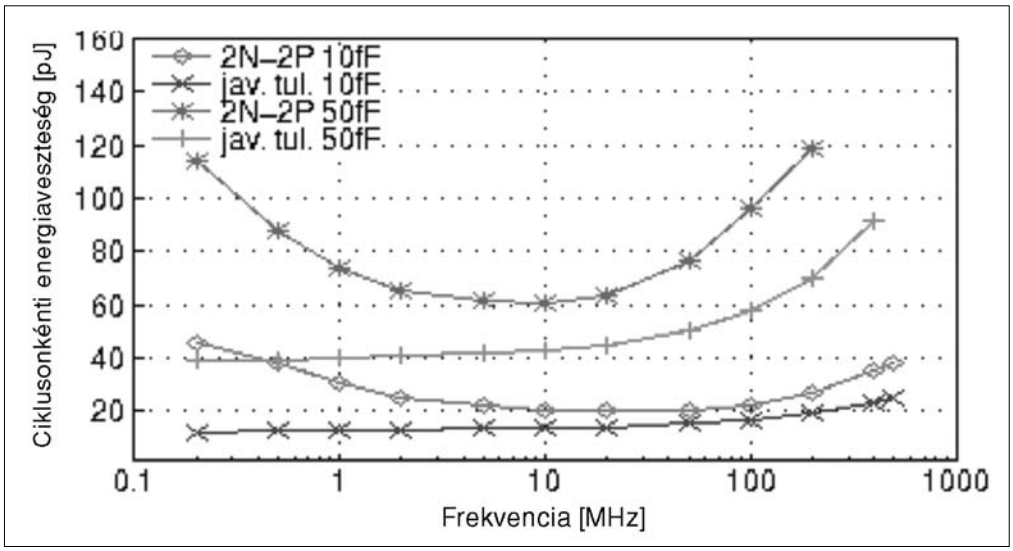
Adiabatikus elven működő kapcsolással statikus tárolóelem kialakítása nem lehetséges, mivel a vezérlő fázisjel egyben a kapcsolás tápvezetékét is jelenti, így ennek alacsony szintje alatt az áramkör nem táplált.

A bemutatott adiabatikus kapcsolások négyfázisú működésűek, ahol egy logikai kapu kimenete a bemeneténél képest negyed periódussal késik, ezáltal a ne-



3. ábra

A javított tulajdonságú adiabatikus kapcsolás invertere és négy szakaszból álló fázisjele

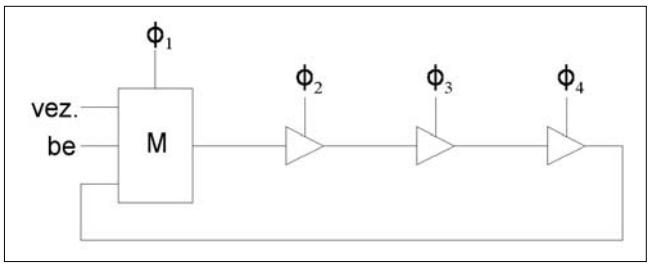


4. ábra

100 láncba kapcsolt inverter energiavesztesége 10fF és 50 fF terhelőkapacitások mellett

gyedik logikai fokozat kimenete fázishelyesen visszavezethető az első fokozat bemenetére. Invertereket ily módon gyűrűbe fűzve az adattárolás a tárolt adat folyamatos másolásával megoldható.

Az ezen az elven működő tároló felépítése az *5. ábra* szerinti, ahol a Φ_1 fázisjelhez kapcsolódó logikai kapu egy összetett kapu, míg a többi három kapu csak inverter (puffer).



5. ábra Adiabatikus elven működő tárolóelem felépítése

Az *„M”*-el jelölt összetett logikai kapu egy kétbemenetű multiplexer, amely az új adat beírására szolgál, ami az ábrán látható elrendezésben a felfutó Φ_1 fázisjelre történhet meg.

5. Adiabatikus működésű áramkörök ütemezése

Az önmagában lassú működésű, fázisjel-vezérelt adiabatikus logika felépítéséből adódóan pipeline működésű, így átbocsátása a logikai mélységtől független. Ez előnyösen használható a digitális jelfeldolgozásban, ahol adiabatikus logikával kétszintű pipeline feldolgozó egység építhető. Mindemellett négyfázisú kapcsolással a lappangási idő is alacsonyan tartható, mivel egy órajel-ciklus alatt négy logikai fokozatot értékelünk ki.

Ismert, hogy a magas szintű logikai szintézis kulcslépése az ütemezés, amely megadja, hogy mely műveletek végrehajtása melyik vezérlési lépésben kezdődik. Az eddig használt ütemező algoritmusok azonban adiabatikus áramkörökre nem alkalmazhatók, mivel nem veszik figyelembe az adatútba az ütemezés után beépítendő multiplexerek műveletvégzési idejét.

Az adiabatikus logika fázisjel-vezérelt működéséből következően minden egyes fázisjelre csak egyetlen logikai szint kiértékelése történik. Azáltal, hogy ütemezéskor valamennyi művelet valamennyi logikai szintjét valamely vezérlési lépéshez rögzítjük, multiplexerek nem szúrhatók be az adatútba az ütemezés után, mivel ki nem használt vezérlési lépések jellemzően nem állnak rendelkezésre.

Az adiabatikus ütemező eljárás alapja az azonos típusú műveletvégző egységek között a hozzájuk csatlakozó multiplexer bemenet darabszámának egyenletes elosztása [4].

Ehhez a következő adatokra van szükség:

– viselkedési leírás adott tervezési megkötésekkel történő megvalósításához szükséges műveletvégző egységek darabszáma minden egyes műveletvégző-egység típusból,
– az adatforrások forrástípusonkénti darabszáma, ahonnan az egyes típusú műveletvégző egységekhez bemenő adatok érkeznek.

Az ütemező eljárás először kiszámítja az előírt újraindítási idő eléréséhez szükséges műveletvégző egységek lehető legkisebb darabszámát, anélkül, hogy a tulajdonképpeni ütemezést végrehajtaná. A következő lépésben, felhasználva a műveletvégző egységek darabszámát és a viselkedési leírásban megadott műveletek közötti függőségeket, az azonos típusú műveletvégző egységekhez tartozó multiplexer-bemenetek darabszámának kiszámítása következik. A harmadik lépésben az azonos típusú műveletvégző egységek, és a rájuk jutó multiplexer-bemenetek darabszámából az algoritmus műveletvégző-egység típusonként meghatározza az egy műveletvégző egységre jutó multiplexer-bemenetek számát, és így az egyes multiplexer-műveletek végrehajtásához szükséges vezérlési lépések darabszámát.

A negyedik lépésben kerül sor a tényleges ütemezésre, ahol az algoritmus egy módosított lista ütemezéssel figyelembe veszi mind a multiplexer-műveletek vég-

rehajtásához szükséges vezérlési lépések darabszámát, mind a kétszintű pipeline működésből adódó művelet átlapolásokat.

A továbbiakban bevezetjük a következő jelöléseket:

N_k – *k* típusú műveletek darabszáma a viselkedési leírásban
M_k – *k* típusú műveletet végrehajtó műveletvégző egységek darabszáma
DII – újraindítási idő vezérlési lépésben mérve
L – felső korlát az ütemezés hosszúságára vezérlési lépésben mérve
S_k – *k* típusú műveletvégző egységekhez csatlakozó multiplexerek bemeneteinek száma
I_k – *k* típusú műveletvégző egység egy bemenetére jutó multiplexer-bemenetek darabszáma
d_k – *k* típusú műveletvégző egység előtt elhelyezkedő multiplexerek műveletvégzési ideje vezérlési lépésben mérve
t_k – *k* típusú műveletvégző egység műveletvégzési ideje vezérlési lépésekben mérve
t'_k – *k* típusú műveletvégző egység ütemezéshez használt műveletvégzési ideje vezérlési lépésekben mérve

A fenti jelölésekkel a viselkedési megadás megvalósításához szükséges műveletvégző egységek lehető legkisebb darabszáma műveletvégző-egység típusonként a (2) összefüggés szerint számítható ki, ahol a „⌈ ⌉” jel a következő egész számra való felfelé kerekítést jelenti.

$$M_k = \left\lceil \frac{4 \cdot N_k}{DII} \right\rceil \qquad (2)$$

A következőkben ismertetésre kerülő eljárással a különböző típusú műveletvégző egységek darabszámának függvényében műveletvégző-egység típusonként meghatározhatjuk a hozzájuk csatlakozó multiplexerek bemeneteinek számát. A keresett S_k értékek a viselkedési leírásban megadott műveletek közötti függőségek vizsgálatával kaphatók meg, a következő eljárást minden egyes *k*-ra elvégezve.

Kezdetben S_k egyenlő nullával. A kiinduló leírás alapján egy listát készítünk, amelyben a *k* típusú műveletekre összegyűjtjük azon különböző jellegű forrásokat, ahonnan a *k* típusú műveletekhez bemenő adat érkezik, valamint a különböző jellegű források mellett feltüntetjük a hozzá tartozó bemenetek előfordulási darabszámát is, amelyet E_a-val jelölünk, ahol *a* különbözteti meg az egyes forrásokat.

Az elkészített listákban az egyes források jellege háromféle lehet, amelytől függően S_k a következőképpen változik: ha a forrás

– bemeneti kapu, akkor $S_k = S_k + E_{bemeneti\ kapu}$,
– *i* típusú művelet, akkor $S_k = S_k + \min(E_{i\ típusú\ művelet}, M_i)$,
– állandó érték, akkor $S_k = S_k + \min(E_{állandó\ érték}, 2 \cdot M_k)$

ha $S_k > M_k$, egyébként pedig S_k változatlan.

Ennek magyarázata a következő. A bemeneti kapukhoz szükséges multiplexer-bemenetek darabszáma a bemeneti kapuktól eredő bemenő csatlakozások számával egyenlő.

Ha egy *k* típusú művelethez bemenő adat egy *i* típusú művelet eredményeként érkezik, akkor az ehhez szükséges multiplexer-bemenetek száma attól függ, hogy az *i* típusú műveletektől a *k* típusú műveletekhez vezető csatlakozások száma és az *i* típusú műveletvégző egységek darabszáma hogyan viszonyul egymáshoz. Ha az előző a kisebb, a helyzet ugyanaz, mint a bemeneti kapuk esetén. Ha azonban az utóbbi a kisebb, akkor csak az *i* típusú műveletvégző egységek darabszámával növekszik a szükséges multiplexer-bemenetek száma, mivel ekkor az *i* típusú műveletvégző egységek megosztásra kerülnek az *i* típusú műveletek között.

Állandó érték kiválasztása érdekében egy műveletvégző egységhez elég csak egyetlen multiplexer-bemenetet figyelembe venni, mivel az már előre kiválasztható úgy, hogy a szükséges vezérlési lépésben rendelkezésre álljon. Ezt fejezi ki a $2 \cdot M_k$ tag, ahol a kettes szorzó a későbbi kettővel való osztás miatt szükséges. Ha nem jut minden műveletvégző egységre állandó érték, akkor a szükséges multiplexer-bemenetek darabszáma csak az állandó értékek darabszámával növekszik. Ha a bemeneti kapuktól és a többi típusú műveletvégző egységektől érkező bemenetek száma nem nagyobb mint a rendelkezésre álló műveletvégző egységek száma, akkor a multiplexer-bemeneteket egyáltalán nem kell figyelembe venni, mivel a nem állandó értékű bemenetek csak a műveletvégző egységek egyik bemenetéhez csatlakoznak.

A műveletvégző egységekhez csatlakozó multiplexerek bemeneteinek darabszámát műveletvégző-egység típusonként meghatározó eljárást a *6. ábra* foglalja össze, amely eljárással az S_k értékek megkaphatóak minden *k*-ra.

Mivel minden műveletvégző egység két bemenettel rendelkezik, és a *k* típusú műveletvégző egységekből M_k darab van, ezért az egyes műveletvégző egységek egyes bemeneteire jutó multiplexer-bemenetek egyenletesen elosztott darabszámát a (3) összefüggés adja.

$$I_k = \left\lceil \frac{S_k}{2 \cdot M_k} \right\rceil \qquad (3)$$

Egy *k* típusú műveletvégző egységekhez csatlakozó multiplexerek műveletvégzési ideje vezérlési lépésben mérve kiszámítható a (4) összefüggés ismeretében, ahol egy logikai szinten legfeljebb négy bemenetű multiplexereket engedélyezünk.

$$d_k = \left\lceil \frac{\log_2(I_k)}{2} \right\rceil \qquad (4)$$

Ütemezéskor egy műveletvégző egységet és a hozzá csatlakozó multiplexert együtt úgy tekinthetjük, mintha a műveletvégző egység műveletvégzési ideje a hozzá csatlakozó multiplexer műveletvégzési idejével megnőtt volna.

Ezt fejezi ki az (5) összefüggés, ahol a t'_k jelenti a tényleges ütemezéshez használandó, vezérlési lépésben mért műveletvégzési időt.

$$t'_k = t_k + d_k \qquad (5)$$

A tényleges ütemezés egy módosított lista ütemezés, amely képes a pipeline működésű műveletvégző egységek használatával kétszintű pipeline adatutat ütemezni. Először meghatározzuk minden egyes művelet mozgékonyságát az ASAP (olyan hamar, amint lehetséges) és az ALAP (olyan későn, amint lehetséges) indítási értékek kiszámolásával, műveletvégzési időnek a t'_k értékeket használva.

A módosított lista-ütemező a műveletvégző egységek minden típusára egy lefoglalási táblát tart fenn, mely táblának M_k darab sora van, oszlopainak száma pedig megegyezik a vezérlési lépésben megadott megengedhető legnagyobb ütemezési hosszúsággal.

Egy művelet kezdete csak akkor ütemezhető a *t* vezérlési lépésbe, ha az adott művelethez tartozó lefoglalási tábla *t* sorszámú oszlopában még van üres hely. Ha egy művelet kezdetét a *t* vezérlési lépésbe ütemezünk, akkor az ütemező egy jelet tesz az adott művelethez tartozó lefoglalási tábla mindazon oszlopaiba, amelynek sorszámára igaz a (6) összefüggés, ahol C_k jelöli a *k* típusú művelethez tartozó lefoglalási tábla oszlopának sorszámát, *n* pedig egész szám és $0 < n < L/DII$.

$$C_k = t + n \cdot DII \quad (6)$$

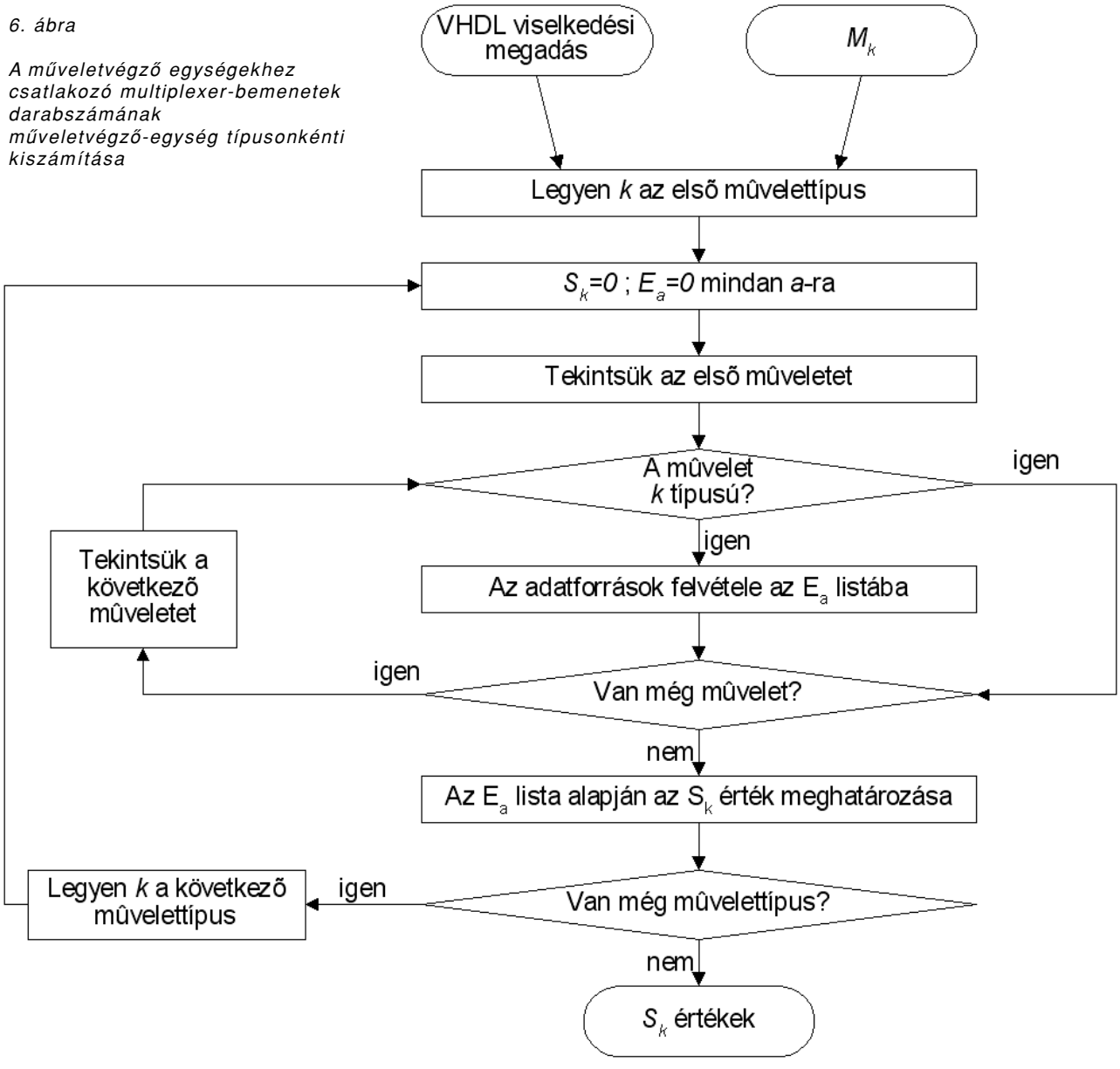
Ha egy *k* típusú művelet első alkalommal kerül ütemezésre, akkor az ütemező a *k* típusú művelethez tartozó lefoglalási tábla minden olyan oszlopának minden sorába egy-egy jelet tesz, amelyre a (7) összefüggés igaz.

$$C_k \neq t + 4 \cdot n \quad (7)$$

Az első eset a pipeline működés biztosításához szükséges, hiszen a lista ütemezés eredetileg erre nem alkalmas, míg az utóbbi a műveletvégző egységeknek a műveletek közötti megosztását teszi lehetővé, és ezzel biztosítja, hogy az ütemezés a (2) összefüggéssel meghatározott darabszámú műveletvégző egységekkel megvalósítható legyen.

6. ábra

A műveletvégző egységekhez csatlakozó multiplexer-bemenetek darabszámának műveletvégző-egység típusonkénti kiszámítása



Egy hagyományos ütemező algoritmusban az adott típusú műveletvégző egységek az ugyanolyan típusú műveletek között megoszthatók, ha ezen műveletek időben nem lapolják át egymást. Adiabatikus műveletvégző egységek használatakor a második szintű pipeline lehetővé teszi az átlapolást, azonban az előbbi feltételen túl az erőforrás-megosztáshoz a négy fázisjeles vezérlés miatt még az is szükséges, hogy a megosztandó műveletvégző egységeken végrehajtandó műveletek azonos fázisjel alatt induljanak. A bemutatott ütemező algoritmus ezt úgy biztosítja, hogy az azonos típusú műveletvégző egységek ugyanazon logikai szintjeit ugyanazon fázisjelhez rendeli.

6. Egészértékű lineáris programozáson alapuló algoritmus négyfázisú adiabatikus áramkörök ütemezéséhez

Az előzőekben közölt ütemező eljárás ugyan optimális eredményt ad a szükséges műveletvégző egységek darabszámára nézve, azonban az ütemezés vezérlési lépésekben mért hosszúsága általánosságban nem a legkedvezőbb, így a szükséges járulékos elemek, mint például a pufferek száma több lehet a kelleténél. Ezért bemutatunk egy egészértékű lineáris programozáson alapuló algoritmust is, amely tetszőleges szempontból optimális megoldást nyújt.

Az egészértékű lineáris programozáson alapuló ütemezés lineáris egyenlőségek és egyenlőtlenségek felállítását jelenti, amelyek leírják mind a viselkedési megadás műveleteinek egymástól való függését, mind a tervezési korlátokat. A tényleges ütemezés ezen egyenlőségek és egyenlőtlenségek megoldása azon feltétel mellett, hogy valamely célfüggvény értéke a legnagyobb vagy a legkisebb legyen.

A következőkben néhány újabb jelölést adunk meg, amelyeket – az előző alpontban bevezetett jelöléseken felül – a továbbiakban használunk, valamint néhány korábban bevezetett jelölés most új értelmet kap:

o_i – az i-dik művelet a kiinduló leírásban
s_i – az o_i művelet legkorábbi (olyan hamar, amint lehetséges) indítási ideje
l_i – az o_i művelet legkésőbbi (olyan későn, amint lehetséges) indítási ideje
$x_{i,t}$ – értéke 1, ha az o_i művelet a t. vezérlési lépésben indul, egyébként nulla
t_i – az o_i művelet végrehajtásához szükséges vezérlési lépések száma
d_i – az o_i műveletet végrehajtó műveletvégző egység előtt elhelyezkedő multiplexer műveletvégzési ideje vezérlési lépésben mérve

A. Ütemezési korlátok

Valamennyi művelet végrehajtását a rá vonatkozó legkorábbi és a legkésőbbi indítási időpont között kell megkezdeni. Mivel a multiplexerek műveletvégzési idejét nem tudjuk előre megmondani, ezért az egyes műveletek indítási ablakát csak a hagyományos esetre tudjuk felírni, azonban ezt is felhasználhatjuk arra, hogy elhagyhassunk nyilvánvalóan nulla értékű változókat. Így a (8) szerinti egyenletek írhatók:

$$\sum_{t=s_i}^{l_i} x_{i,t} = 1 \qquad \text{minden egyes } i\text{-re.} \tag{8}$$

B. Függőségi korlátok

Egy műveletvégző egység a bemenő adatait multiplexereken keresztül kapja, amelyek műveletvégzési idejét figyelembe kell venni. Azért, hogy a műveletek közötti függőségeket a kiinduló leírásnak megfelelően megtartsuk, egy művelet végrehajtása csak akkor kezdődhet, ha az azt megelőző műveletek már befejeződtek. A fentieknek megfelelő egyenletek a (9) szerintiek:

$$\sum_{t=s_i}^{l_i} t * x_{i,t} + t_i + d_i - \sum_{t=s_j}^{l_j} t * x_{i,j} \le 0$$

minden egyes i-re és j-re, ahol o_i közvetlenül megelőzi o_j-t,

$$\sum_{t=s_i}^{l_i} t * x_{i,t} + t_i + d_i - L \le 1$$

minden egyes i-re, ahol o_i-t nem követi további művelet.

C. Pipeline átlapolási korlátok

Egy műveletvégző egység, amely egy számítást vezérlési lépésben kifejezve a t időpontban kezd meg, pipeline működésének köszönhetően a $t+4$ időpontban újabb számításba kezdhet anélkül, hogy az előző műveletet befejezte volna. Egy művelet, amely vezérlési lépésben kifejezve a T időpontban indul, lefoglal egy műveletvégző egységet $T \le t < T+4$ ideig. Ez a (10) szerinti függvényekkel fejezhető ki, amelyet minden egyes műveletre (minden i-re) fel kell írni:

$$\begin{aligned} x'_{i,t} &= 1 \quad \text{ha } T + d_i \le t < T + d_i + 4 \\ &= 0 \quad \text{egyébként.} \end{aligned} \tag{10}$$

A párhuzamosan használt k típusú műveletvégző egységek darabszáma a vezérlési lépésben kifejezett t időpontban egyenlő azon műveletek darabszámával, amelyek ugyanezen t időpontban k típusú műveletvégző egységet lefoglalnak. A k típusú műveletvégző egységek szükséges darabszáma egyenlő a bármely időpontban párhuzamosan használt k típusú műveletvégző egységek darabszámának legnagyobbikával. Ezt fejezik ki a (11) függvények:

$$M_k = \max \sum_{t=1}^{L/DII} x'_{i,j+t*DII} \tag{11}$$

minden i,j és k-ra, ahol $0 < j \le DII$ és o_i k típusú

D. Multiplexer-korlátok

A multiplexer-korlátok felírásához meg kell határozni az egyes típusú műveletvégző egységek darabszámának függvényében a különböző típusú műveletvégző egységekhez csatlakozó multiplexerek bemeneteinek számát, amely a 6. ábrán ismertetett algoritmus szerinti

S_k értékek kiszámítását jelenti. Egy műveletvégző egység két bemenettel rendelkezik, így a szükséges multiplexer bemenetek száma megoszlik a két bemenet között, ezen kívül a *k* típusú műveletvégző egységből M_k darab van. Így a (12) egyenlőtlenségeknek kell teljesülniük:

$$\frac{S_k}{2} \le \sum_{o_i} d_i \le \sum_{k=1}^{M_k} 4^{d_k} \qquad \text{ahol } o_i \; k \text{ típusú} \qquad (12)$$

7. Ütemezési algoritmusok eredményei

Az *1. táblázat* a bemutatott lista-ütemező algoritmussal kapott ütemezési eredményeket mutatja a véges válaszú szűrő tervezési példára. Az adatút szélességét 16 bitre választva az adiabatikus összeadó műveletvégzési ideje 6, a szorzóé pedig 9 vezérlési lépés. A táblázat oszlopai rendre az újraindítási idő vezérlési lépésben mérve, az összeadók darabszáma, a szorzók darabszáma, a multiplexer bemenetek száma, a pufferek száma és az ütemezési hosszúság vezérlési lépésben kifejezve. Az egészértékű lineáris programozáson alapuló ütemezési eredmények a *2. táblázatban* láthatók

Mindkét ütemező algoritmus azonos és egyben a legkisebb szükséges számú műveletvégző egységet alkalmazza, azonban a lista ütemezés használatával ütemezési hosszúság nagyobb és a szükséges pufferek száma is több. Ennek az a magyarázata, hogy ez az ütemezés az azonos típusú műveletek kezdetét azonos fázisjelhez köti.

8. Értékelés, következtetések

A cikkben bemutattunk egy továbbfejlesztett adiabatikus töltésvisszanyerő kapcsolást, amely egy korábbi kapcsoláson alapul, azonban a terhelő kapacitás feltöltése alatt teljesen kiküszöböli a nem-adiabatikus veszteséget. A kapcsolás bonyolultságának kézbentartása érdekében a terhelő kapacitásnak csak a részleges kisütése megengedett, azonban lehetőség nyílik teljesen adiabatikus kisütésre is. A logikai kapcsolások mellett szekvenciális hálózatok építhetők a bemutatott tároló elrendezés alkalmazásával.

Adiabatikus rendszerek automatizált magas szintű tervezésekor olyan problémát kell megoldani, amelyre a jelenleg használatos ütemező algoritmusok nem alkalmasak. A bemutatott ütemező algoritmus úgy teszi lehetővé nagy bonyolultságú adiabatikus rendszerek tervezését, hogy a tervezőnek az adiabatikus működéstől elvonatkoztatva, csak feladatköri szempontokra kell ügyelnie.

Köszönetnyilvánítás

A szerzők köszönetüket fejezik ki Dr. Kovács Ferenc egyetemi tanárnak, akinek szakmai segítsége meghatározó volt az egész kutatás kivitelezése során. Szintén köszönet illeti az Országos Tudományos és Kutatási Alapot a T023963 számú „Kisfogyasztású CMOS áramkörök tervezése” és a T029331 számú „Kisfogyasztású CMOS analóg áramkörök tervezési metodikájának kidolgozása” című OTKA kutatási támogatásokért.

Irodalom

[1] J. Fischer, P. Teichmann, A.G. Stoffi, E. Amirante, D.S. Landseidel, „Scaling Trends in Adiabatic Circuits”, 1st Int. Workshop on Reversible Computing, May 2005.

[2] E. Amirante, A.B. Stoffi, J. Fischer, G. Iannaccone, D.S. Landsiedel, „Variations of the Power Dissipation in Adiabatic Logic Gates”, 11th Int. Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), September 2001., pp.9.1.1–10.

[3] L. Varga, F. Kovács, G. Hosszú, „An Efficient Adiabatic Charge-Recovery Logic”, IEEE SoutheastCon 2001, Clemson, South Carolina, pp.17–20.

[4] L. Varga, F. Kovács, G. Hosszú, „Approaches for Scheduling of Adiabatic Logic”, IEEE Int. Workshop on Logic & Synthesis, Granlibakken, CA, June 2001, pp.18–22.

1. táblázat A bemutatott lista szerinti ütemezést alkalmazó eljárás eredményei a véges válaszú szűrő tervezési példára, különböző újraindítási idők mellett.

DII	M_+	M_*	Mplex. bem.	Puffer	L
4	15	8	0	20	38
8	8	4	36	70	51
12	5	3	42	142	63

2. táblázat Az ILP-t alkalmazó ütemező eljárás eredményei különböző újraindítási idők mellett.

DII	M_+	M_*	Mplex. bem.	Puffer	L
4	15	8	0	20	38
8	8	4	36	34	43
12	5	3	42	81	57