A RENDSZERSZINTÉZIS ÚJ ELVEI

Arató Péter

ÖSSZEFOGLALÓ

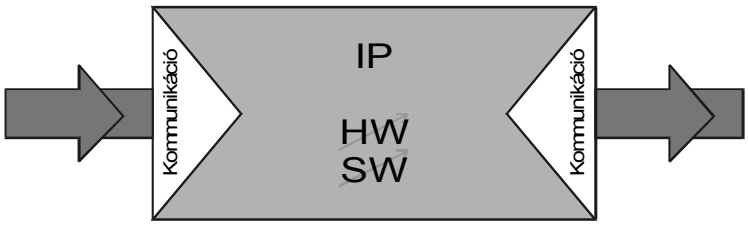
A mikroelektronika gyors fejlődése egyre bonyolultabb, „intelligens”, (Intellectual Property Units, IP-s) készen kapható funkcionális egységeket kínál építőelemekként a tervezőknek. Az ezáltal kínálkozó jelentős előnyök kihasználásához magas absztrakciós szintről, ún. viselkedési specifikációból kell indítani a tervezési folyamatot, amelyet rendszerszintű szintézisnek neveznek. A viselkedési szintről kiinduló szisztematikus tervezési módszerek az 80-as évek elejétől alkalmazott ún. magas szintű szintézis algoritmusainak kiterjesztése révén alakultak ki. A cikk áttekintést nyújt erről a folyamatról, és bemutatja a rendszerszintézishez alkalmazott dekompozíciós, valamint hardver/szoftver együttes szintézis eljárások lényegét.

SUMMARY

The rapid development in microelectronics offers more and more complex ready-made „intelligent” functional units (Intellectual Property Units, IP-s) as building blocks for the designers. In order to exploit the significant advantages of these units, the design procedure is to be started from a high abstraction level called behavioral specification. Such a design method is called system level synthesis which is based on the extensions of the algorithms of the so called high level synthesis applied from the early 1980s. This paper gives a survey on this procedure and presents the essential features of the decomposing and hardware/software codesigning methods applied in system synthesis.

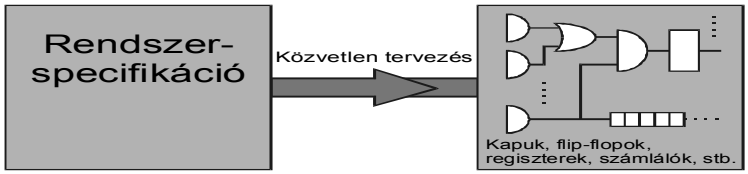
BEVEZETÉS

A mikroelektronikai ipar állandó gyors fejlődése egyre bonyolultabb, „intelligens”, (Intellectual Property, IP) készen kapható funkcionális egységeket kínál építőelemekként a tervezőknek (**1. ábra**).



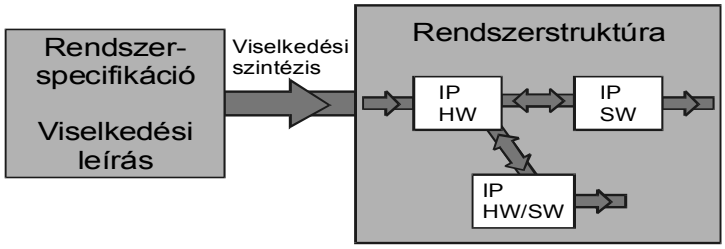
1. ábra
„Intelligens” funkcionális elem (IP) szemléltetése

Az ilyen komponensek (IP-k) adaptálhatósága, programozhatósága és rekonfigurálhatósága kedvező lehetőséget nyújt azonos típusok ismételt felhasználására (reusing) [11]. Ezáltal a piacra kerülés ideje jelentősen csökkenhet, mert a tesztelés és a gyors prototipus-készítés kisebb ráfordítást igényel. Ezzel szemben gyökeres változtatásra szorul a többé-kevésbé egzakt logikai szintézis módszertana, amely viszonylag egyszerű építőelemek (kapuk, flip-flopok, regiszterek, számlálók stb.) alkalmazásán alapul (**2. ábra**). A megoldandó feladatból kiinduló rendszerspecifikáció ugyanis már elképzelhetetlen a komplex építőelemeknek (IP-knek), konkrét fizikai erőforrásokként való kezelésével.



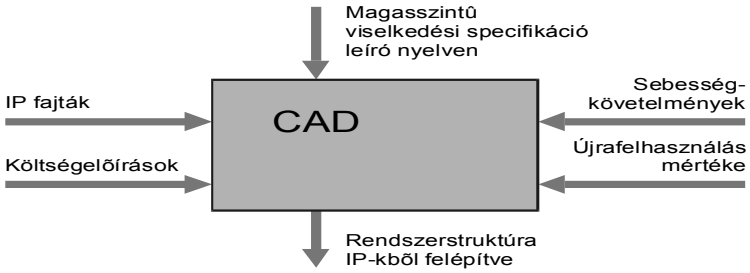
2. ábra
A közvetlen logikai szintézis folyamat szemléltetése

Jóval magasabb absztrakciós szintről kell indítani a tervezési folyamatot, amelyet emiatt rendszerszintű szintézisnek (system-level synthesis, SLS) neveznek. A kiindulási magas absztrakciós szinten (**3. ábra**) az ún. viselkedési specifikációból (behavioral specification) kell kiindulni, amely fiktív viselkedési összetevők együttműködésének leírása [1],[11]. A rendszerszintű szintézis további lépéseiben olyan dekompozíciós algoritmusra van szükség, amely a teljes rendszert felbontja előre definiált IP-kre. Ennek során sokszor egymásnak ellentmondó peremfeltételek mellett (az IP-k közötti kommunikáció sebessége, a vezérlési struktúra egyszerűsége, az ismételt felhasználás mértéke, a különböző költségtényezők egymáshoz képesti aránya, pipeline működés biztosítása stb.) kell a lehető legkedvezőbb megoldásokat megtalálni, lehetőleg szisztematikus lépésekben [11]. A gyártók egyre növekvő választékát kínálják az IP-knek, katalógusaikban többnyire viselkedési szintű specifikációval. Az IP-k között teljesen programozható cél- és általános processzorok is megjelennek (pl. DSP). Ezáltal a rendszerszintű szintézis egyre inkább olyan felépítést eredményez, amely hardver és szoftver összetevőket egyaránt tartalmaz. Így az ún. hardver/szoftver együttes tervezés (hardware-software codesign) része a rendszerszintű szintézisnek [1],[6],[11].



3. ábra
A viselkedési szintézis szemléltetése

A tervezés automatizálása (design automation) döntő jelentőségű a rendszerszintű szintézisben, mert a legtöbb lépés NP-nehéz jellegű, így nem mindig kerülhetők el a közelítő optimumkereső eljárások [5],[6],[13]. A technológiai fejlődés következtében egyre komplexebb rendszerek hozhatók létre egyetlen integrált áramköri tokban (System on Chip, SoC) a fentiekben vázolt felülről lefelé haladó (top-down) metodika szerint a magas absztrakciós szintű viselkedési specifikációból kiindulva [1]. A magas szintű specifikáció általában valamilyen fejlett programnyelven történik, miáltal a számítógéppel segített automatikus tervező eljárások (CAD tools) bemeneti adatai közvetlenül képezhetők (**4. ábra**).



4. ábra
A viselkedési leírásból kiinduló számítógéppel segített tervező rendszerek (CAD tools) bemeneti adatai és szolgáltatásai

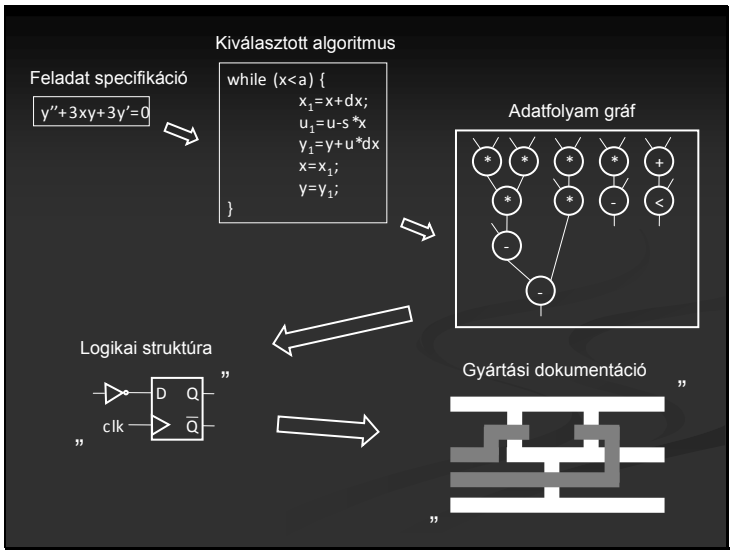
A viselkedési szintről kiinduló szisztematikus tervezési lépéseket elsőként az ún. magas szintű szintézishez (high-level synthesis, HLS) kezdték alkalmazni, a 80-as évek közepétől. A HLS algoritmusok főként a nagysebességű célhardverként megvalósított jelfeldolgozó eszközök struktúrájának optimalizálását támogatták.

A valóságos hardver erőforrások, mint összetevők ekkor már elég komplexek voltak ahhoz (de korántsem annyira komplexek, mint az IP-k ma és a jövőben), hogy leírásukat már viselkedési szinten legyen célszerű kezelni, fiktív elemi műveletekből összeállítva. A HLS algoritmusok többsége kiterjeszthető és alkalmazható a rendszerszintű szintézis céljaira. Ezen belül a HLS algoritmusok előnyösen alkalmazhatók pl.

az IP-k tervezéséhez is. A rendszerszintézis új elveinek és módszereinek bemutatásához ezért először a magas szintű szintézis főbb lépéseinek lényegét foglaljuk össze.

A MAGAS SZINTŰ SZINTÉZIS (HLS) FŐBB LÉPÉSEI

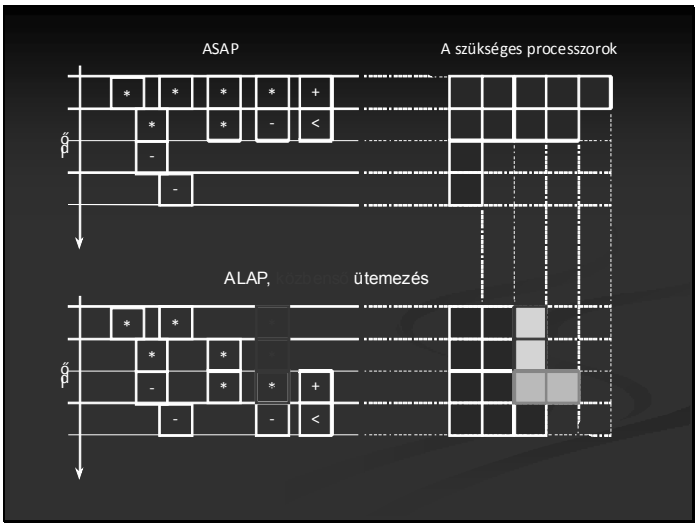
A viselkedési leírásból kiinduló tervezés alapgondolata követhető az **5. ábrán**. A megoldandó feladat specifikációja a kiindulásként megadott differenciálegyenlet. Ennek megoldására kell választanunk egy algoritmust, amelyet az ábrán egy programrészlet szemléltet. Az algoritmus-választás során természetesen elveszíthetünk szabadsági fokokat, amely a későbbi optimalizálási lépések hatékonyságát sokszor előre nem megítélhető mértékben befolyásolhatja. A folyamat NP-nehéz jellege már ilyenkor is nyomon követhető. A kiválasztott algoritmus alapján készül a konkrét tervezéshez szükséges viselkedési leírás, amely általában adatfolyam (data-flow) gráf. Fontos megjegyezni, hogy a gráf csomópontjai nem konkrét műveletvégző egységeket (esetünkben nem aritmetikai egységeket), hanem magukat a műveleteket, mint a megoldáshoz előírt viselkedéseket jelentenek. A későbbi tervezési lépések feladata annak meghatározása, hogy ezeket az előírt viselkedéseket miként lehet legalább közelítőleg optimálisan hozzárendelni, pl. a legkevesebb, vagy/és legolcsóbb műveletvégző egységekhez. Ezek a tervezési lépések, mint a magas szintű szintézis alapvető lépései az **5. ábrán** nincsenek szemléltetve, csupán a végeredményt (a logikai struktúrát és a gyártási dokumentációt) szimbolizálják az idézőjelbe tett ábrarészek. A magas szintű szintézis tervezési lépéseinek érzékeltetésére szolgál a **6. ábra**, ahol az elvi adatfolyam-gráf gyakorlati kezelhetetlensége (pl. hazárdjelenségek) miatt szinkronizáló órajelet tételeztünk fel, és az egyszerűség érdekében azt is feltételeztük, hogy minden művelet végrehajtódik egyetlen órajel-periódus alatt. Ez utóbbi egyszerűsítés a gyakorlatban általában nem indokolt, mert a műveletek bonyolultsága és így végrehajtásuk időigénye jelentősen különbözhet; a lényeg szemléltetése érdekében azonban megengedhető. Így az ábrán vízszintes vonalak választják el az órajel-periódusokat, az **5. ábra** alapján képzett szinkronizált adatfolyam-gráf csomópontjait pedig négyszögek jelölik.



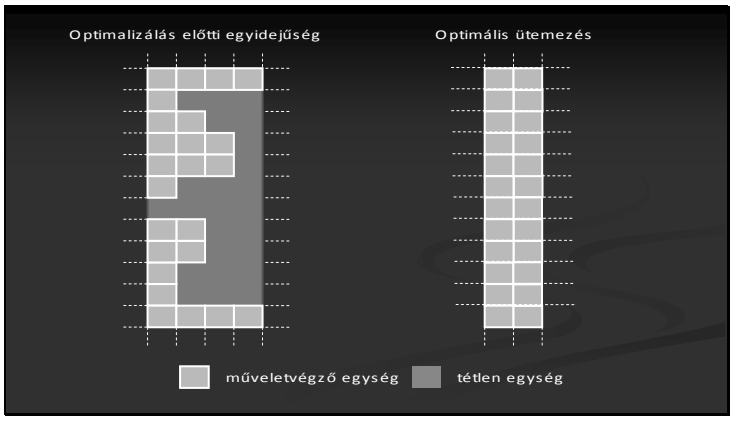
5. ábra
A viselkedési leírásból kiinduló tervezési folyamat szemléltetése

A **6. ábra** felső részén a műveletek ütemezése (scheduling) olyan, hogy minden művelet végrehajtása a lehető legkorábban (as soon as possible, ASAP) indul. A hisztogram-szerű ábrázolásból látszik, hogy ilyenkor az első órajel-periódus terheltsége miatt egyidejűleg 5 műveletet kell végrehajtani, vagyis legalább 5 konkrét műveletvégző egységre (processzorra) volna szükség az előírt viselkedések allokációjához (hozzárendeléséhez). Az ábra alsó részén további ütemezési lehetőségek hatása figyelhető meg az allokáció szempontjából. A lehető legkésőbbi (as late as possible, ALAP) ütemezés, anélkül, hogy megnövelné a teljes végrehajtási időt (lappangás, latency), eleve kedvezőbb: csak 4 processzort igényelne. Könnyen belátható azonban, hogy a közbenső ütemezések vizsgálatával eljuthatunk a csupán 3 processzort igénylő megoldáshoz. Az alapkérdés tehát az, hogy miként lehet megtalálni azt az ütemezést,

amely az allokáció szempontjából a legkedvezőbb. Az **5.** és **6. ábrán** követett terheltségi szemléltetést követve a **7. ábrán** egy elképzelt optimális ütemezés ábrázolása látható. Nyilvánvaló, hogy az ütemezés minősége csak az allokáció ismeretében értékelhető, ezért az NP-teljesség miatti közelítő optimumkereső eljárások többnyire ún. mohó (greedy) algoritmusokon alapulnak [1],[2].



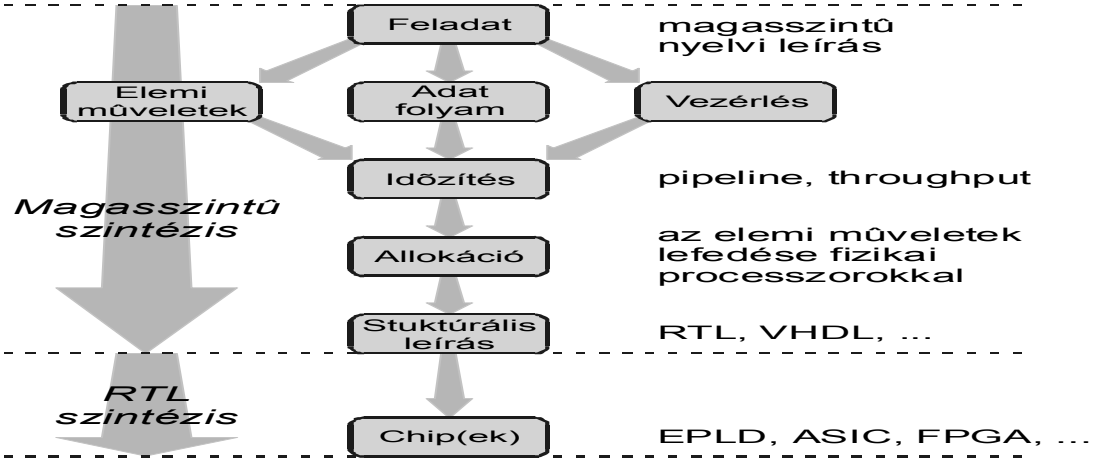
6. ábra
Az ütemezés és az allokáció szemléltetése



7. ábra
Az optimális ütemezés szemléltetése

A legtöbb alkalmazási területen a sebességi követelmények igénylik az ún. pipeline feldolgozás megvalósítását. Ezért a HLS algoritmusoknak támogatniuk kell a pipeline üzemmódban is működtethető struktúrák optimalizálását. Könnyen belátható, hogy minél „gyorsabb” a pipeline működés (rövid újraindítási, azaz restart idő, illetve nagy átbocsátási képesség, azaz throughput), annál kedvezőtlenebbek az allokációs lehetőségek adott ütemezés mellett [1],[5].

Fentiek alapján a **8. ábra** foglalja össze a magas szintű szintézis főbb lépéseit. Az eredményként értelmezett strukturális leírásnak általában valamilyen regiszter transzfer szintű (register transfer level, RTL) nyelven kell adódnia, amelyből kiindulva a kereskedelmi forgalomban lévő ún. silicon compiler jellegű rendszerek képesek szolgáltatni akár a gyártási dokumentációt is. Könnyen belátható, hogy a magas szintű (viselkedési) szintézis tartományában végzett optimumkeresés döntő mértékben befolyásolja a végeredményt.



8. ábra
A magas szintű szintézis főbb lépései

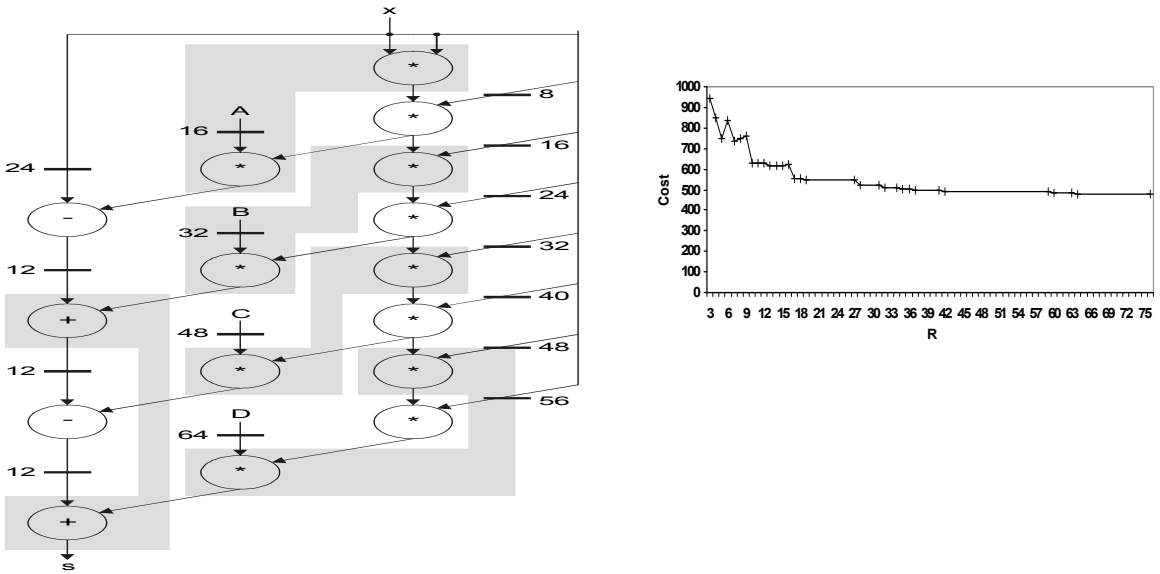
9. ábra
A PIPE tervező rendszer szolgáltatásai

A BME Irányítástechnika és Informatika Tanszéken, sok éves kutatómunka eredményeképpen kidolgoztunk egy számítógépes tervező rendszert [1],[10], a fentiekben vázolt magas szintű szintézis feladatra. A PIPE nevű tervező rendszer algoritmusai és szolgáltatásai (**9. ábra**) az alábbi szempontokból tekinthetők kedvezőbbeknek a szakirodalomból ismert, releváns megközelítésekhez képest: [1],[11].

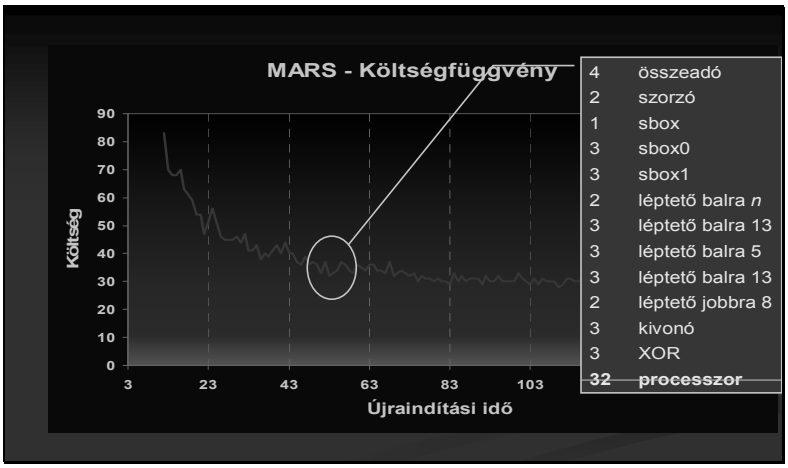
- Az algoritmusaink mindig pipeline működést tételeznek fel (a nem-pipeline esetet a pipeline speciális eseteként kezeljük, amikor az újraindítási idő egyenlő a lappangási idővel), továbbá a kívánt újraindítási idő előre megadható.
- A viselkedési modellünk olyan speciális szinkron adatfolyam-szerű (dataflow-like) elemi műveleti gráf, amelynek alapján egyszerűen elkerülhetők az időzítési- és hazárdproblémák a magas szintű szintézist követő tervezési lépések során.
- Az adatszinkronizáló algoritmusunk közvetlenül szolgáltatja az ASAP (minden művelet a lehető legkorábban indul) és az ALAP (minden művelet a lehető legkésőbben indul) ütemezést. E két szélső eset közötti bármely ütemezés egyszerűen képezhető a szinkronizáló késleltető hatásoknak, mint fiktív, egymás után kapcsolt puffer regisztereknek az adatutak mentén történő mozgatása révén.
- Az előre megadható kívánt pipeline újraindítási idő a minimális számú pótlólagos puffer regiszter beépítésével és/vagy a minimális számú műveletnek a lehető legkisebb példányszámú többszörözésével (struktúrális pipeline) valósítható meg.

- Az allokációs algoritmusunk a páronként nem konkurens műveletek közötti kompatibilitási reláció által definiált maximális kompatibilitási osztályokból képezhető fedőrendszer meghatározásán alapul.
- Az algoritmusaink kiterjeszthetők többfelhasználós rekurzív hurkokat tartalmazó alkalmazások esetére is.

A **10. ábra** szemlélteti a PIPE rendszer által létrehozott strukturális terv jellegét és a feladatfüggő költséget (pl. a szükséges processzorok számát) az órajel-periódusok darabszámával mért pipeline újraindítási idő (R) függvényében. A kiindulási gráfon szürke sávok jelölik a közös processzorba allokált műveleteket, vízszintes vonalak a beiktatott puffereket, a mellé írt számok pedig a szükséges darabszámaikat jelölik egy adott újraindítási idő mellett. Az eredményül kapott feladatfüggő költségfüggvény alapján mindig meghatározható az a legrövidebb újraindítási idő, amely még elfogadható költséggel megvalósítható. A választás elvét szemlélteti a **11. ábrán** látható költségfüggvény, amelyet a PIPE rendszer egy több mint 400 csomópontból álló kiindulási gráfhoz (MARS nevű rejtjelező algoritmus) szerkesztett. Látható, hogy a kb. 43 órajel-periódusonkénti újraindítás még kis költségű, gyors működést eredményez. Az ennél nagyobb újraindítási idejű, vagyis lassúbb működtetéssel, lényeges költség-megtakarítást ez a kiindulási gráf nem tesz lehetővé.

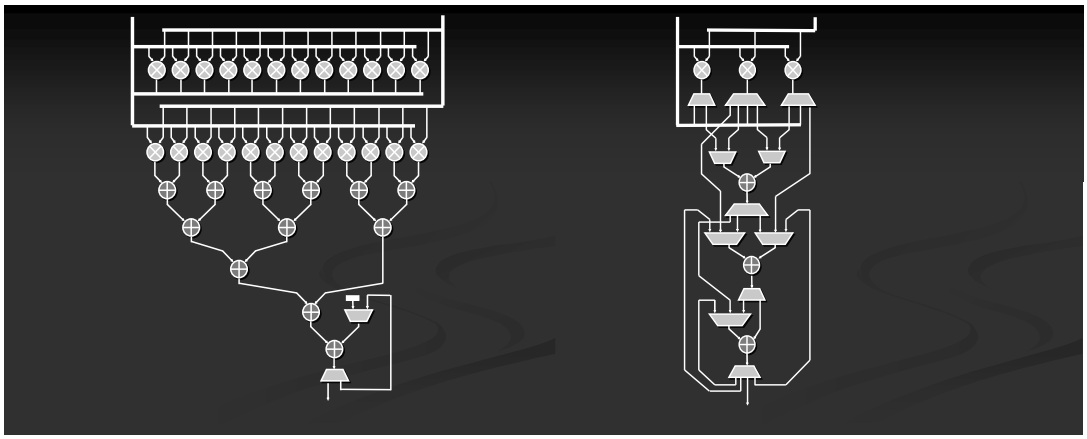


10. ábra
Strukturális terv és költségfüggvény

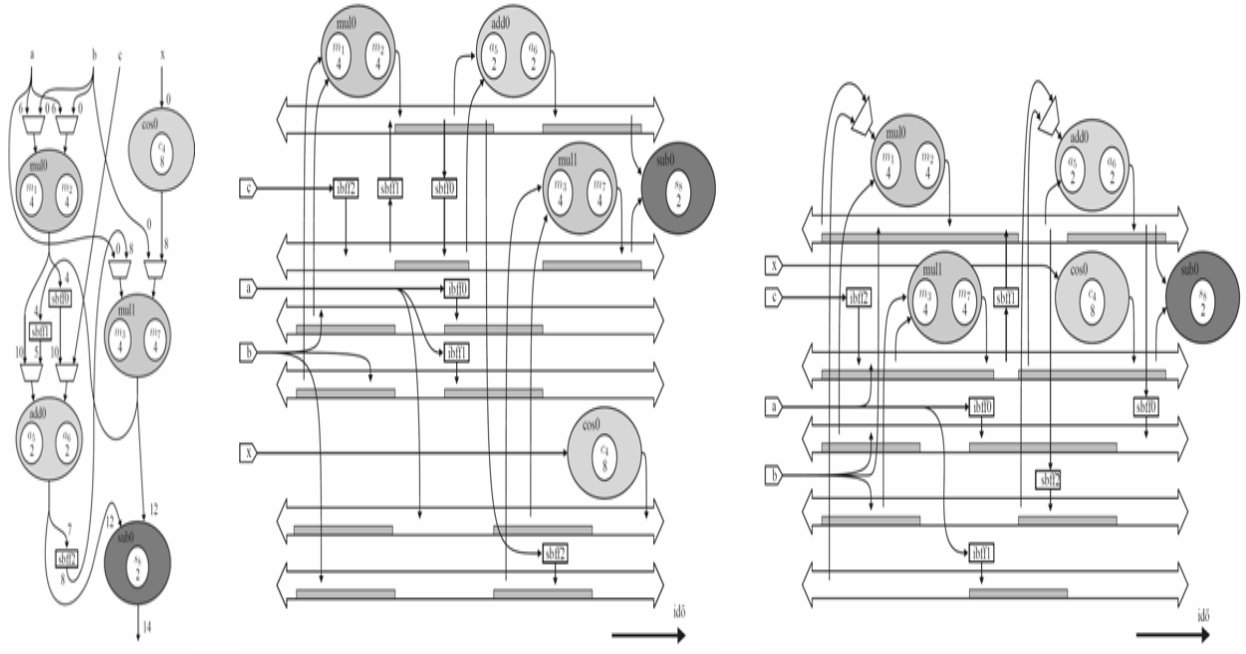


11. ábra
A pipeline újraindítási idő megválasztásának szemléltetése

A magas szintű szintézis eredményeként létrejövő struktúra a funkcionális egységek között az allokáció révén kiadódó bonyolult összeköttetések miatt sok multiplexert és demultiplexert tartalmaz, amely jelentősen ronthatja a hatékonyságot. Ezt a hátrányt szemlélteti a **12. ábra**, ahol a trapézok jelölik a multiplexer, illetve demultiplexer elemeket. Látható, hogy az allokáció során kiadódó processzorok a jobb oldalon szereplő eredmény-gráfon több kiindulási műveletet képesek megvalósítani, de a baloldalon ábrázolt kiindulási gráf által előírt adat-precedenciát csak a pótlólagos multiplexer, illetve demultiplexer elemek tudják biztosítani, ha az adat-összeköttetések közvetlenek.



12. ábra
Az allokáció által igényelt pótlólagos multiplexerek, illetve demultiplexerek



13. ábra
A MUXRED és a BUSRED algoritmusok hatásának szemléltetése

A PIPE rendszer segítségével az összeköttetések sínrendszerként optimalizálhatók, valamint a multiplexerek és demultiplexerek száma minimalizálható [10]. Ezt a feladatot a MUXRED és a BUSRED nevű algoritmusok látják el, eredményüket a **13. ábra** szemlélteti. A közvetlen adatkapcsolatokkal megvalósított

strukturális terv 6 multiplexert igényel. A MUXRED algoritmus 6 db. arbitráció-mentes, egyszerű adatsínt (kettős nyilakkal jelölve) eredményez, és az összes multiplexert megtakarítja. A BUSRED algoritmus csak 5 db. sínt alakít ki azon az áron, hogy két multiplexert meghagy.

DEKOMPOZÍCIÓ IP FUNKCIONÁLIS EGYSÉGEKRE

A rendszerszintű szintézis fontos követelménye, hogy a magas szintű szintézis által szolgáltatott strukturális tervet adott, vagy kiadódó IP készlet felhasználásával is meg tudjuk valósítani, vagyis optimumkereső eljárásokat találjunk az IP készletre történő dekompozícióra. A PIPE tervező rendszer az allokáció eredményeként a kiindulási elemi műveleteknek olyan fedőrendszerét hozza létre, amelynek minden blokkja olyan műveleteket tartalmaz, amelyek páronként nem-konkurensek, vagyis végrehajthatók közös processzorral. Ha közös processzorokként készen kapható, adaptálható és rekonfigurálható IP egységek alkalmazása a cél, akkor szükség van olyan (lehetőleg szisztematikus) eljárásra, amely az IP-k rendelkezésre álló készletéből való kiválasztást és az ismételt felhasználást (reuse) kísérli meg optimalizálni. Mivel a feladat NP-nehéz [2], a továbbiakban ismertetendő DECIP (DEComposition into IP-s) algoritmus csak egyszerű és gyors közelítő optimumkeresést tűzhet ki célul.

Az IP-gyártók egyre növekvő választékot kínálnak. Katalógusaikban legtöbbször viselkedési szintű specifikációt is megadnak. Jóllehet ezáltal az IP viselkedése a felhasználás szempontjából egyértelműen definiált [11], a dekompozíciós algoritmus megfogalmazásához kedvezőbb az olyan értelmezés, amely közvetlenül kapcsolódik a magas szintű szintézis megelőző lépéseihez. Mivel a megoldandó feladat viselkedési szintű leírása általában elemi műveleti gráf formájában adott [1],[11], ezért az elemi műveletek tekintendők a specifikáció tovább-nem-bontott egységeinek. A magas szintű szintézis ütemező és allokáló lépései hozzák létre az elemi műveleteknek azt a fedőrendszerét, amelynek blokkjait kell a rendelkezésre álló IP-kkel, mint valós erőforrásokkal megvalósítani [10]. Ezért előnyös, ha az IP-ket is az elemi műveletekkel jellemezzük. Minden egyes IP-re megadjuk azokat az elemi műveleteket, amelyek végrehajtása az adott IP-vel lehetséges és a tervező által előnyösnek ítélt (preferált). Ezt az ún. végrehajthatóságot a tervező dönti el az IP-k alkalmassága és adaptálhatósága alapján [10]. Bizonyos peremfeltételek (sebesség, kommunikációs költség, a vezérlés bonyolultsága, ismételt felhasználás stb.) kizárhatnak egyes IP-ket adott elemi műveletek végrehajtói közül annak ellenére, hogy egyébként adaptálhatók lennének a végrehajtásra [10]. A DECIP algoritmust az alábbiakban foglaljuk össze.

Jelölje M az elemi műveletek E halmazán képzett fedőrendszert. Ilyen fedőrendszernek tekinthető például a nem-konkurens elemi műveletek maximális kompatibilitási osztályainak halmaza, amelyet az ütemezés és az allokáció szolgáltat a magas szintű szintézis során [1],[10]. A fedőrendszerből kiindulva minden elemi műveletet hozzá kell rendelni végrehajtásra a felhasználható IP-knek az I halmazából választott valamelyik, de csak egyetlen elemhez. Meg kell tehát határozni az elemi műveletek E halmazán egy P teljes partíciót az M fedőrendszerből kiindulva. Ennek a P partíciónak minden egyes blokkját egy végrehajtó IP definiálja, hiszen a blokkban szereplő műveleteket éppen ennek az IP-nek kell végrehajtania. A végrehajtó IP-k, kiválasztásakor, vagyis P blokkjainak meghatározásakor a következő feltételezéseket és peremfeltételeket kell figyelembe venni:

- Minden egyes IP-t azok az elemi műveletek definiálják, amelyek végrehajtása leheséges és preferált az adott IP-vel.
- Az adott IP készletből a lehető legkevesebb IP-t kell felhasználni.
- A lehető legkevesebb IP típus kell felhasználni (törekedni kell az ismételt felhasználásra).
- A végrehajtó IP-k különböző kiválasztási szempontjai megfelelő súlytényezők alkalmazásával egyszerűen lehessenek súlyozhatók és kombinálhatók.

Az algoritmus leírásához az alábbi jelöléseket alkalmaztuk:

$E:(e_1,\ldots e_i,\ldots . e_N)$	Az elemi műveletek halmaza
$M:(M_1,\ldots M_r,\ldots M_k)$	Az E halmaz teljes fedőrendszere (pl. az allokáció során keletkező maximális kompatibilitási osztályok halmaza)
$c(M_r)$	M_r relatív költsége
$I:(I_1,\ldots I_s,\ldots I_j)$	Az alkalmazható összes IP halmaza

$c(I_s)$	I_s relatív költsége
$R:(R_1,\dots R_s,\dots R_j)$	Az E halmazbeli elemi műveletek azon – nem szükségszerűen diszjunkt – részhalmazai, amelyek végrehajtása lehetséges és preferált rendre az $I_1,\dots I_s,\dots I_j$, IP-kkel
$S:(\dots I_h,\dots I_v,\dots I_q,\dots)$	A kiválasztott végrehajtó IP-k halmaza
$n(I_s)$	Az I_s jelű IP-nek a kiválasztás során kiadódó példányszáma
$R_{s,z}$	R_s diszjunkt részhalmazai, amelyek azokat az elemi műveleteket tartalmazzák, amelyek végrehajtására az I_s jelű IP z-edik példánya lett kiválasztva ($(1 \le s \le j)$, $(1 \le z \le n(I_s))$
P	Az E halmaz végrehajtó partíciója (blokkjait az összes $R_{s,z}$ részhalmaz képezi)
W_{IPCost}	$c(I_s)$ relatív súlytényezője
W_γ	γ_s relatív súlytényezője, $\gamma_s = \sum_r \lvert M_r \cap R_s \rvert : (M_r, R_s) \in M \times R$
W_{IPSort}	N_s relatív súlytényezője, $N_s = \begin{cases} 1, \text{ if } n(I_s) > 0 \\ 0, \text{ if } n(I_s) = 0 \end{cases}$
w_s	Az I_s jelű IP-re vonatkozó súlyfüggvény érték

A bevezetett jelölésekkel megadható a DECIP algoritmus tömör leírása:

START

$\forall n(I_s) := 0$

$S := \varnothing$

while $M \neq \varnothing$,

do {

$\alpha = \max\{\lvert M_r \cap R_s \rvert : (M_r, R_s) \in M \times R\}$

for $\forall R_s$: $\gamma_s = \sum_r \lvert M_r \cap R_s \rvert : (M_r, R_s) \in M \times R$

determine γ_{max} (the maximal γ_s value)

for $\forall R_s$: $w_s = -W_{IPCost} \frac{c(I_s)}{\max\{c(I_k), I_k \in I\}} + W_\gamma \frac{\gamma_s}{\gamma_{\max}} + W_{IPSort} N_s$

for $\forall M_r$: $\delta_r = \lvert M_r \rvert$

determine δ_{min} (the minimal δ_r value)

select one R_s, for which: $\exists M_r : \left| M_r \cap R_s \right| = \alpha$ and

$$w_s = w_{\max} \text{ and}$$

$$\delta_r = \delta_{\min}$$

$S := S \cup I_s$

$n(I_s) := n(I_s) + 1$

$R_{s,n(I_s)} := M_r \cap R_s$

neglect $\forall e_i$ from M if $e_i \in R_{s,n(I_s)}$

}

STOP

A DECIP algoritmus konvergenciája nyilvánvaló, hiszen M mérete minden ciklusban csökken, így a befejeződéskor üres halmaz adódik. A konvergencia sebességét alapvetően a kritériumok (α, w_{max}, δ_{min}) szerinti heurisztikus Rs választás határozza meg. A ws értékek változtathatók a súlytényezők (W_{IPCost}, W_γ W_{IPSort}) hangolása révén. Ezáltal R_s kiválasztási stratégiája befolyásolható és hatékonysága ellenőrizhető, kikísérletezhető. A variációk száma erősen függ a kiindulási M halmaz blokkjainak számától (|M|). Nagy |M| érték várhatóan nagyfokú átfedéssel jár együtt a kiindulási fedőrendszer blokkjai között, amely meredeken növelheti a variációk számát R_s kiválasztásakor. Ez a nehézség csökkenthető a kiindulási M fedőrendszer redukálását végző ún. REDIN (REDucing the Initial Cover) algoritmus segítségével [10], amelynek leírásához az alábbi pótlólagos jelöléseket vezettük be:

z_i	Az e_i elemi művelet előfordulásánsk száma az aktuális M halmazban
m_r	Az z_i értékek súlyozott összege M_r-ben (a súlyokat az e_i műveletek relatív költségei képezik)
g	$\|M\|$ kívánt redukciójának mértéke
$s(M)$	A legkisebb m_r értékekel rendelkező M_r-k halmaza
$M(red)$	A redukált M halmaz

A REDIN algoritmus leírása:

START

If $\left| M \right| < g$ then STOP

$M(red) := \varnothing$

j: = 1

Calculate z_i-s for each e_i-s and m_r for each M_r

do {

Determine *s(M)*

Select an M_r from *s(M)* and remove it from actual *M*.

$M(red) := M(red) \cup M_r$

If $e_i \in M_r$, then $z_i := 0$.

Recalculate m_r-s and *s(M)* for the actual *M*

j = j + 1

} while $M(red)$ is not a complete cover.

While j ≤ g {

Select an M_r from *s(M)* and remove it from *M*.

$M(red) := M(red) \cup M_r$

Recalculate z_i-s, m_r-s and *s(M)* for the actual *M*

j = j + 1

}

$g := j - 1$

STOP

A REDIN algoritmus végrehajtása után is marad természetesen minden ciklusban kiválasztási lépés, de s(M) elemeinek száma általában elég kicsi ahhoz, hogy az M_r választáskor ne adódjon túl sok variáció.

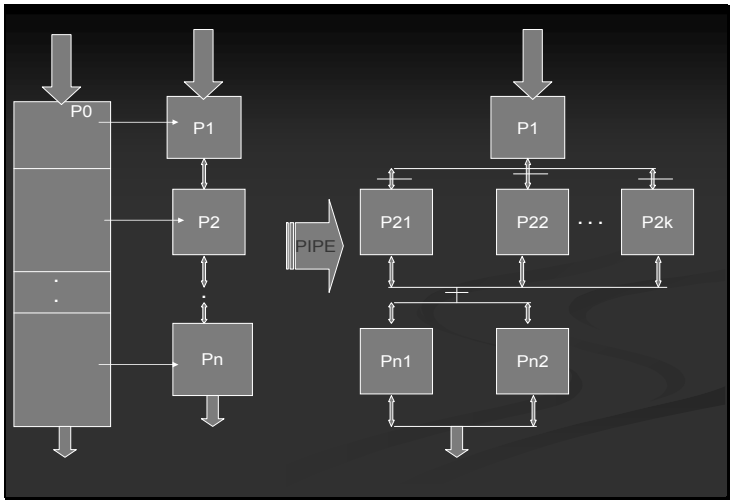
Ha a felhasználandó IP funkcionális egységek előre adottak, akkor a PIPE rendszer hierarchikus kiterjesztése révén [2],[13] formálisan lehetővé válik akár önmagukban is pipeline működésű, bonyolult IP-knek elemi műveletként való kezelése. Ezáltal lehetővé válik, hogy a korábban kidolgozott PIPE magas szintű tervező rendszerrel létrehozott struktúrák formálisan elemi műveletként legyenek értelmezhetők a PIPE rendszer ismételt, magasabb hierarchikus szinten történő alkalmazásakor. A PIPE rendszernek ez a kiterjesztése lehetővé teszi előre megadott, tetszőleges IP-k felhasználását a magas szintű szintézis során.

HARDVER/SZOFTVER EGYÜTTES TERVEZÉSI LEHETŐSÉGEK

A PIPE rendszer további kiterjesztése nem párhuzamosítható magas szintű program alapján történő többprocesszoros pipeline működésű célrendszer tervezéséhez.[1] Cél: A programot részekre bontva az egyes részprogramokat külön processzorokhoz rendeljük. Ezeket a processzorokat komplex műveleti egységeknek tekintve alkalmazzuk a hierarchikus magas szintű PIPE tervező rendszert (**14. ábra**). Ezáltal feladat- és paraméterfüggő, többprocesszoros rendszer-struktúra hozható létre, amelynek működtetése nem igényel bonyolult, többprocesszoros operációs rendszert. Így a kiindulási program végrehajtása a lehetővé váló pipeline működés révén jelentősen gyorsítható [3]. Megoldandó feladatok:

- Algoritmus az optimális részekre bontásra, minimálisra csökkentve pl. a közös memóriaterületekhez való hozzáférési igényt.
- Arbitrációmentes sínrendszer és egyszerű kommunikációs protokoll kialakítása a részek között.

[1] Jelenleg folyó kutatási témánk: OTKA 72611.



14. ábra
Feladatfüggő struktúrájú többprocesszoros rendszer létrehozásának szemléltetése

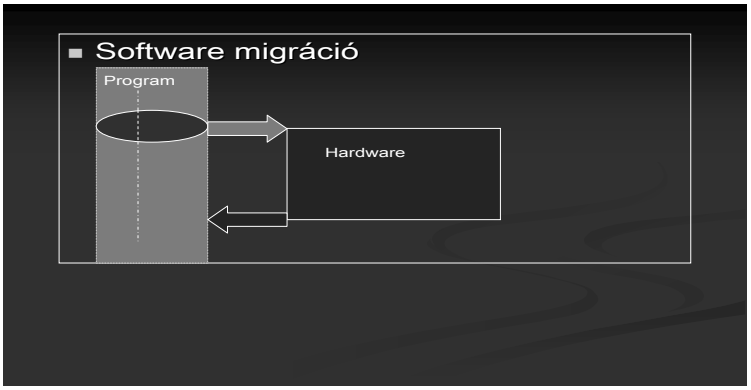
A rendszerszintézis során fontos lépés annak mérlegelése, hogy a dekompozíció eredményeként kiadódó funkcionális egységek megvalósítása hardver vagy szoftver úton előnyösebb-e. Sokszor már a fentiekben vázolt dekompozíciós eljárás során is tekintettel kell lenni erre a szempontra. Így a hardver és szoftver egységekre történő felbontás során (hardware/software codesign) során olyan dekompozíciós algoritmusokra van szükség, amelyek az alábbi peremfeltételeket egyidejűleg képesek figyelembe venni: [4],[6]

Szoftver költség (végrehajtási idő, memóriaigény, beszerzési ár, tesztelhetőség stb.).

Kommunikációs költség (adatkapcsolatok száma, kommunikációs idő, protokoll megvalósítás költsége, szoftver támogatási igény).

Hardver költség (szükséges félvezető terület, gyártási költség stb.).

A hardver/szoftver együttes tervezés egyik gyakran alkalmazott egyszerű elve az ún. szoftver migráció, amelynek lényegét a **15. ábra** szemlélteti. Egy program sebességkritikus része előnyösebben valósítható meg külső hardver egységgel, ha az így szükségessé váló kommunikációs költségek nem rontják le lényegesen a sebességnövekedés hatását. Ennek esetenkénti és általános mérlegelése is fontos, időszerű módszertani kutatási területe a rendszerszintézisnek [9],[12].



15. ábra
A szoftver migráció szemléltetése

A hardver/szoftver együttes tervezés területén ezért kutatják azokat a módszereket is, amelyek segítségével egy magas szintű nyelven megírt programból vagy programrészletből közvetlenül kísérlik meg a hardver struktúra generálását [7],[8]. Az ilyen eljárások célja, hogy például a C nyelven megírt programból közvetlenül lehessen hardver struktúrát generálni szisztematikus eljárással, lehetőleg kevés megszorítást előírva a programtervező számára. Az általános C nyelvű programból lehetőleg szisztematikusan lehessen generálni a megszorításoknak eleget tevő változatot, és a hardver struktúra lehetőleg kevés elemtípusból épüljön fel. A BME Irányítástechnika és Informatika Tanszéken kifejlesztett módszer állapotgépeken alapul, amelyek az egyes C utasításoknak felelnek meg, mint *while, do, for, if* [7]. Ezen állapotgépek (utasítás-egységek) vezérlik a megfelelő tevékenységeket, mint pl. feltételek kiértékelése, ciklusok indítása a kiértékelt feltétel- kiértékelése eredménye alapján, és végül a következő utasítás egység indítása. Az eljárások (alprogramok) definiciója változókkal együtt procedúra egységek létrehozásával történik, melyek egy állapotgépből és a helyi változók, illetve hívási paraméterek számára tárolóhelyekből állnak. Más műveletek, mint például összehasonlítás, összeadás stb. műveleti egységekre képeződnek le, amelyek szintén állapotgép alapúak és az utasítás egységekével azonos módon aktivizálhatók. Az utasítás, műveleti és procedúra egységek az eredeti forráskód alapján láncot alkotnak a trigger és ready kivezetéseik segítségével. Ez az egyszálú láncolat a fordítási időben zajló kódvizsgálat segítségével úgy módosul, hogy a függetlenül aktivizálható műveletsoroknak megfelelő láncszegmensek párhuzamosan kerülnek bekötésre. Az eljárás végén egy VHDL kód generálódik automatikusan, amelynek alapján a szoftverrel meghatározott célhardver megvalósítható.

IRODALOMJEGYZÉK

[1] P. Arató, I. Jankovits, T Visegrády: High Level Synthesis of Pipelined Datapaths, Wiley, 2001, ISBN 0-471-49582-4

[2] Zs. Palotai, T. Kandár, Z. Mohr, T. Visegrády, G. Ziegler, P. Arató, A. Lőrincz: Value Prediction in HLS Using Intellectual Properties, Applied Artificial Intelligence, Taylor and Francis, Vol. 16, Num. 2, Febr., 2002, pp. 117-157.

[3] P. Arató, Z. Á. Mann, A. Orbán: Extending Component-Based Design with Hardware Components, Journal of Science of Computer Programming, Elsevier, 2005, Vol. 56, pp. 23-39.

[4] P. Arató, Z. Á. Mann, A. Orbán: Algorithmic Aspects of Hardware/Software Partitioning, ACM Transaction on Design Automation on Electronic Systems, 2005, volume 10, issue 1, pp. 136-156

[5] P. Arató, Z. Á. Mann, A. Orbán: Time-constrained scheduling of large pipelined datapath, Journal of Systems Architecture, Elsevier, 2005, volume 51, issue 12, pp. 665-687

[6] Z. Á. Mann, A. Orbán, P. Arató: Finding Optimal Hardware/Software Partitions, Formal Methods in System Design, Springer Science, Vol. 31, 5 Oct. 2007, pp. 241-263

[7] Péter Arató, Bence Csák: Hardware Definition Based on Standard C-language Source Code, Proceedings of Forum on Specification and Design Languages, FDL'03, Frankfurt, Germany, September 23-26, 2003, pp. 727-735, ISBN 1636-9874

[8] R. Plyler: The Streamlined Design Flow from Catapult C to Precision RTL Synthesis, Mentor Graphics, 2007

[9] Péter Arató, Zoltán Ádám Mann, András Orbán: Hardware-software co-design for Kohonen's self-organizing map, Proceedings of the IEEE 7th International Conference on Intelligent Engineering Systems, Luxor (Egypt), 2003

[10] P. Arató, T. Kandár, Z. Mohr, T. Visegrády: High-level Synthesis Using Predefined IP-s, Periodica Polytechnica, El. Vol. 46. 2002. No. 3-4. pp. 123-136.

[11] P. Coussy, A. Morawiec: High-Level Synthesis (from Algorithm to digital Circuit), Springer, 2008

[12] P. Arató, G. Kocza, I. Lovanyi, L. Vajta: Hardware/Software Codesign of Feature Tracking Algorithms, Proceedings of 12th IEEE International Conference on Intelligent Engineering Systems (INES 2008) ISBN: 978-1-4244-2083-4, Miami, Florida, February 25-29, 2008, pp.41-45

[13] Péter Arató, Tibor Kandár: Systematic VHDL Code Generation Using Pipeline Operations Produced by High Level Synthesis,.Proceedings of IEEE International Symposium on Intelligent Signal Processing, Budapest, Hungary, September 4-6, 2003. pp. 185-190, ISBN: 0-7803-7864-4