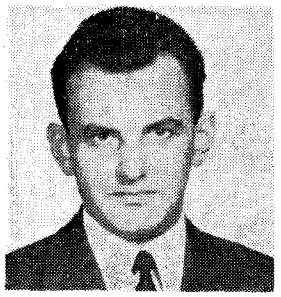
VLSI áramkörök szimulációs problémái

BOHUS MIKLÓS
BME Híradástechnikai Elektronika Intézet



ÖSSZEFOGLALÁS

A cikk összefoglalja a VLSI áramkörök szimulációjával szemben támasztott legfontosabb követelményeket. Definiálja a hardware leíró nyelv főbb jellemzőit. Meghatározza az összetartozó bemeneti és kimeneti változások halmazaként specifikálható leírási mód alaptulajdonságait és ezen jelsorozatokon alapuló szimulációs vizsgálat alkalmazhatósági feltételeit. Végezetül összefoglalja a publikált szimulációs nyelvek és eljárások főbb jellemzőit.

1. Bevezetés

Digitális hálózatok leírására, szimulálására — az elvégzendő feladattól függően — különböző *szintű* leíró nyelvet alkalmazhatunk. Ennek megfelelően különböző szintű szimulációt végezhetünk.

1. *Strukturális szimuláció* során alapvetően architekturális szinten történik a vizsgálat. A leíró nyelv az architektúra strukturális elemeit és a közöttük lévő kapcsolatot írja le. A vizsgálat célja az architektúra elemei közötti főbb adat és vezérlési áramlás vizsgálata.

2. *Funkcionális szimuláció* során a hardware elemek (összetevők) közötti azon adat és vezérlési folyamatokat írjuk le, amelyek valamely *funkció* megvalósulásához ténylegesen szükségesek. Ez legtöbb esetben a funkciót realizáló *algoritmusok* leírását jelenti. Pl. input/output folyamat leírása, műveletvégzés egyes lépéseiből álló szekvenciák leírása, megszakításkor lejátszódó események leírása stb. A leírás szorítkozhat pusztán az algoritmikus viselkedésre, ami az esetek többségében regiszterek közötti adatcserére vezethető vissza. Ez a leírási mód a regiszter-transzfer szint (rövidítve: RT). Az RT szintű nyelveket kiegészíthetjük az *eseményekhez* rendelt idő kezelésével is. (Működési idő, késleltetési idő vizsgálata.) A funkcionális szimuláció megkívánja a rendszer hardware felépítésének — funkcionális szinten történő — ismeretét.

3. A szimuláció elvégezhető a rendszerhez tartozó *bemeneti és kimeneti jelváltozások halmazának* ismeretében is. Az összetartozó bemeneti és kimeneti változások halmazával is egyértelműen vizsgálható a digitális hálózat. Ebben az esetben is két lehetőségünk van: vagy csak az adott bemeneti kombinációhoz tartozó kimeneti kombinációt vizsgáljuk, vagy mindkét kombinációhoz működési (késleltetési) időt rendelünk hozzá. A be/ki halmaz alapján történő vizsgálathoz nem szükséges még funkcionális szinten sem ismerni a vizsgálandó egység hardware felépítését, sőt a benne lejátszódó folyamatok algoritmusát sem.

BOHUS MIKLÓS

A Budapesti Műszaki Egyetem híradástechnikai szakán szerzett villamosmérnöki oklevelet. Kezdetben a Vezetéknélküli Híradástechnika Tanszéken dolgozott, jelenleg a Híradástechnikai Elektronikai Intézetben oktat docensi minőségben. Szabályozáselmélettel, mintavételes rendszerek optimalizálásával foglalkozott előző munkahelyén. Jelenlegi szakterülete digitális berendezések számítógéppel segített tervezése. Az Oktatásügy és a Gépipar Kiváló Dolgozója, a Kruspér István emlékérem, a Kolozs Richárd díj, BME Nívódíj, BME emlékplakett tulajdonosa. Kiváló dolgozó kitüntetést kapott az OMFB-től. Az IFAC és az IFIP Magyar Nemzeti Bizottság tagja. A Mérés és Automatika c. folyóirat operatív szerkesztője. A MATE Elnökségének tagja.

4. *Áramköri szimuláció* során a kapcsolási rajzot realizáló elemek áramköri jellemzőinek ismeretében meghatározzuk a bemeneti paraméterekhez tartozó kimeneti áramköri paramétereket.

5. *Fizikai szintű szimuláció*

A fenti csoportosítás összefoglalva:

Strukturális szimuláció:
Ismert: az architektúrát felépítő struktúra és az architektúra elemeinek strukturális tulajdonsága.
Funkcionális szimuláció:
Ismert: a hardware elemek funkcionális működése és a közöttük megvalósuló algoritmus.
Be/ki halmaz alapján történő szimuláció:
Ismert: a bemeneti kombinációkhoz tartozó kimeneti kombináció.

Áramköri szimuláció
Ismert: az áramkört felépítő elemek áramköri jellemzője (kapu szint, tranzisztor szint).

A szimuláció természetesen még igen sokféle képpen csoportosítható. A digitális rendszer *leírási módja* alapján:

1. *formális nyelven* történő leírás. A nyelv lehet kifejezetten hardware leírásra kifejlesztett nyelv (hardware description languages) szokásos rövidítéssel: *HDL*. Igen sok HDL nyelv készült, aminen részben az az oka, hogy minden gyár egy-egy konkrét rendszer leírására minél tökéletesebb megvalósítást tűz ki célul, másrészt az, hogy nyelvek fordító programja (transzlátor) nem áll közhasználhatóan rendelkezésre.

2. *magasszintű nyelven* történő leírás. Több magasszintű nyelvet időkezelő utasításokkal kiegészítve a nyelv valamennyi előnyét megőrizve

alkalmassá tették hardware leírásra. Ilyen szimulációs rendszer készült pl. Pascal programnyelv kiegészítésével.

3. *folyamatábrán alapuló* leírások. A folyamatábra általában három alapelemet tartalmaz: állapotelemet, döntési elemet, feltételes kimeneti elemet.

A VLSI áramkörök leírása során le kell írni a belső felépítést és a működést tehát mind a strukturális, mind a funkcionális leírásra szükség van. A strukturális leírásnak tartalmaznia kell a rendszert felépítő alacsonyabb szintű egységeket és azok összekapcsolásának módját. A funkcionális leírást deklarálja a rendszer bemeneti és kimeneti pontjait és leírja a rendszer bemeneti és kimeneti pontjai között mindazon összefüggéseket, amelyek ezek között a működés során fennállnak (algoritmus, be/ki halmaz). A strukturális leírás telhát tartalmazza a topológiát, a funkcionális leírás pl. a cellák működését.

2. Általános követelmények a leírással szemben

1. A VLSI elemek mérete szinte beláthatatlanul nagy lehet, a komponensek közötti összefüggés pedig rendkívül bonyolult. Az áttekinthetőség érdekében a rendszer egymásra épülő szintekre bontása alapvető követelmény. A különböző szintek leírása többféle módon történhet:
   1. a szimuláció minden szintjén a leírás azonos nyelvi eszközökkel történjék Pl. CARS [1].
   2. szintenként különböző leírási módszerek alkalmazása. Valamennyi szinten az ott legalkalmasabbnak bizonyuló módszer alkalmazása a felhasználó kényelmét szolgálja [2].

2. A leírás — a tervezést elősegítően — felülről lefelé történjék. A magasabb szinten definiált egység egyértelmű működés specifikálása valamilyen leírási módon történik. Ezt követően a szimuláció során fel kell építeni a definiált egységet alacsonyabb szinten, a szintre jellemző építőelemek összekapcsolásával. Az így felépített struktúrát ugyan úgy működtetve a felsőbb szinttel egyező kimeneti eredményt kell kapnunk (helyesen megválasztott alacsonyabb szintű realizálás során). Ezt az eljárást ismételve a szimuláció során katalógus-elemekig jutunk el.

3. A szimulációs rendszer a legfelső szinttől a legalsó szintig a lehető legtöbb segítséget nyújtson a felhasználónak. A szimuláció is, a tervezés is *részekre, szintekre* osztottan történik. Ennek megfelelően a szimulációt akkor is el kell tudni végezni, ha az egyes részek a szimuláció során *eltérő szinten* vannak leírva. Ebből a szempontból előnyös a különböző szintek azonos nyelvi eszközökkel történő leírása. Ilyen szimulációs módszernél természetesen meg kell engednünk a különböző szintű leírások eltérő részletezettségű („finomságú") leírását és a különböző részletezettségű leírások összekapcsolhatóságát.

Ha a szimuláció a tervezést segíti, elengedhetetlen az alsóbb szintről a felsőbb szintre történő visszalépés is. Könnyen belátható, hogy tervezés során a felsőbb szinten még nem áll rendelkezésre az az információ, amit egy alsóbb szinten hoztunk létre a tervezés folyamán, és így a felsőbb szint specifikációját módosítanunk kell.

A szintek közötti *átmenetek biztosítása* és a specifikációk szintek közötti ekvivalenciájának biztosítása általában kézi úton történik.

4. A leírás eszközei és elemei olvasható, és öndokumentáló szintaktikai szerkezetek létrehozását tegyék lehetővé.

5. A hardware leírás ne csak a szimuláció bemenő nyelveként legyen használható, hanem verifikációs célra is alkalmazható legyen. *Verifikáció* során a specifikáció és az azt megvalósító egység tényleges működése formális eszközökkel kerül összehasonlításra. A formális nyelven alapuló valamint, a be/ki halmazon alapuló leírások tegyék lehetővé a hardware szintézist.

6. Nem alapvető követelmény, de célszerű igény, hogy a specifikációs leírás tegye lehetővé a digitális rendszerek különböző hardware hibáinak figyelembevételét (pl. rövidzár, leragadás) és ilymódon hibaszimulátorként és alkalmazható legyen.

3. A leíró nyelvek összehasonlító vizsgálata

A felhasznált HDL nyelvek kiválasztását az általános követelmények teljesítésén kívül számos egyéb szempont is elősegítheti. A teljesség igénye nélkül megadunk néhány további követelményt:

1. Többszintű leírás esetén a leírási szint meghatározása
2. Strukturális és funkcionális leírás
3. A leírásban szereplő utasítások végrehajtási módja
4. Idő, időzítés kezelés módszere
5. Jelek ábrázolása
6. Adattípusok, adatabsztrakciók
7. Adat és vezérlőjelek kezelése

A továbbiakban a fenti sorrendben megvizsgáljuk a HDL nyelvek főbb tulajdonságait.

3.1. Többszintű leírásban a leírási szint kiválasztása

A jelenleg általánosan használt szimulációs eljárások a felülről lefelé (top-down) történő leíráson alapulnak. A HDL nyelvek alapvetően a regiszter-transzfer szintű leírást támogatják, többségük a logikai működés részletes vizsgálatára is alkalmas. A leírás (és a leíráson alapuló tervezés) különböző fázisaiban vagy:

— azonos nyelvi eszközöket használ
— vagy eltérő nyelvi eszközökkel az illető szinten legalkalmasabb leírási móddal rendelkezik.

Mindkét esetben a magasabb szinten leírt egységet le kell bontani ugyanolyan működést (viselkedést) biztosító megfelelően összekapcsolt alapcsonyabb szintű elemek rendszerére, és ennek az eljárásnak a rekurzív ismétlését végezve a tervező eljut az ismert elemek (pl. cellák, vagy elemi IC-k) szintjére.

Összetettebb elemek szimulációja nem képzelhető el a többszintű vizsgálat nélkül, mivel a szimuláció idő és memória igény ugrásszerűen nő a kezelt elemek számának növelésével. A legalacsonyabb szintű (elemi összetevőkön alapuló) szimu

láció sok esetben több tíz- vagy több százezer ekvivalens kapu egyidejű kezelését kívánhatja meg chippenként.

Kialakítottak olyan szimulációs rendszereket, melyek a *vegyes szintű* leírást támogatják. Ennek felhasználásával egy nagyobb áramkör teljes szimulációja megoldható anélkül, hogy az egész rendszert egyidejűleg a legalacsonyabb szintig le kelljen bontani [3], [4].

A vegyes szintű, többszintű szimuláció előnyösen alkalmazható azoknál a bonyolult felépítésű integrált áramköröknél, amelyekben az áramkör belső felépítésének bizonyos részei *ismeretlenek* a felhasználó előtt, ezek funkcionális működését tudja csak leírni. Ilyen esetekben a legalacsonyabb szintű szimuláció az áramkört felépítő valamennyi elemre el sem végezhető.

3.2. Strukturális és funkcionális leírás

A kiválasztott HDL nyelv célszerűen mind a strukturális, mind a funkcionális leírást támogatja. A strukturális és a funkcionális leírás többszintű szimulációs rendszerben szintén többszintű lehet.

A *strukturális* leírás történhet:

— architektúra szinten pl. processzor—tár—buszrendszer)
— regiszter transzfer szinten
— logikai szinten

A *funkcionális* leírás szintjeit a szimulálandó integrált áramkör típusa határozza meg. Általános esetben a következő szintek valósíthatók meg:

— operációs rendszer szintje
— utasítás készlet szintje
— utasítások végrehajtási szintje
— felhasznált funkcionális elemek működtetési szintje.

A kétféle leírás mód között természetesen bizonyos átfedés figyelhető meg. Pl. az utasítások végrehajtási szintje alapvetően regiszter transzfer szintű leírással történik. Külön nem hangsúlyoztuk, hogy mindkét leírási mód a legalacsonyabb szinten az áramköri leírásban végződhet.

A leíráshoz készült HDL nyelvek alapvetően az utasításkészletükben különböznek egymástól. Általánosságban az alábbi utasítás típusok fordulnak elő:

— *deklarációk* (pl. jelek szélessége, számrendszere, értékkészlete, jelek típusa, jelterjedés iránya, kimenő jel esetén a kimenet típusa: ellenütemű vezérlés, nyitott kollektoros kimenet, három állapotú kimenet stb.). A deklaráció a jeleken kívül kiterjedhet a leírható *elemek* típusára (pl. regiszter, memória, kombinációs hálózat stb.) valamint a fenti két jellemző között megengedett *operátorok* készletére (pl. egyesítés, kompresszió, expanzió, logikai operátorok, transzfer, stb.). Természetesen egyéb tulajdonságok is deklarálhatók: (pl. időosztásos működtetés, többkapus hozzáférés stb.).
— *adattranszfer*. Az adatmozgatás történhet a funkcionális egységek között, és a vezérlő áramkörökben (pl. állapot átmenet, megszakítási rendszer, input/output rendszer, stb.)
— *feltételrendszer*. A leírás valamennyi működés, működtetés feltételhez kötését kell, hogy biztosítsa, a feltétel időzítésének megadásával.
— *vezérlési módok*. A vezérlés az alkalmazott vezérlő egység (fázisregiszteres, mikroprogramozott) specifikációját rögzíti. Bonyolultabb vezérlések esetén különválasztható a vezérlések vizsgálata pl. ilyenek a különböző *mikroprogram szimulátorok*. A mikroprogramok lehetnek horizontális és vertikális felépítésűek.
— *időzítések*. Valamennyi utasítás típushoz (adattranszfer, feltétel, vezérlés) az időbeni működés helyes leírása hozzárendelendő. Az időkezelés a be/ki halmazon alapuló szimuláció során elkerülhetetlen. A bemeneti jelszekvenciák és a hozzárendelt kimeneti jelszekvenciák alapján a vizsgálandó egység pontosan *specifikálható, tesztelhető, verifikálható*. Külön vizsgálatot igényel az időzítések kezelése többszintű szimuláció során, mivel alacsonyabb szinten (a részletesebb áramköri megvalósítás következtében) a jelek száma nő, és a jelek „finomszerkezete" sűrűsödik.

A HDL nyelvek sajátossága, hogy bizonyos viselkedéseket nem utasításokként, hanem funkcióként írja el. Ez a nyelvekben bevezetett *procedúraként* deklarálható. A deklarált eljárásokra (procedúrákra) a szimuláció során egyszerűen lehet hivatkozni. (Ez a tulajdonság hasonlít a programozási nyelvek makroutasítás hívásához.) Láthatóan pl. ez a nyelvi elem elősegíti a *blokkstruktúrált* leírást, és a moduláris szerkezetű szimulációt. A moduláris szerkezetű szimulációs program lehetővé teszi katalogizált elemek felhasználását a szimuláció során.

Megjegyezzük, hogy az időzítések pontos leírása, ezek kezelése jelenti a szimulációs nyelvek és a szokványos programozási nyelvek közötti éles különbséget.

3.3. Végrehajtási mechanizmusok

Az utasítások végrehajtási sorredje alapján két alapvető osztályba sorolhatók a szimulációs nyelvek: procedurális és nemproceduláris nyelvek különböztethetők meg [3], [4].

Proceduláris nyelvekben az utasítások a leírás sorrendjében kerülnek végrehajtásra, kivételt képeznek a vezérlésátadó utasítások.

Nemproceduláris nyelvekben *egyidőben* bármely „aktív" utasítás végrehajtódik. Az „aktív" jelleget meghatározhatja a deklarációban szereplő *feltétel*, vagy egy (több) jelzőbit (flag bit).

A proceduíáris leírás (leírónyelv) közel áll az általános software nyelvekhez, és lehetővé teszi a működési algoritmusok leírását. Bonyolultabb algoritmusokban összetett hardware rendszerekben igen sok *párhuzamos* működés fordul elő. A párhuzamos folyamatok kezelését a proceduíáris nyelvek nem teszik lehetővé, viszont a nemproceduláris nyelvek éppen ezeknek a problémáknak a kezelésére készültek.

A gyakorlatban használt szimulációs nyelvek általában nem tisztán (szigorúan vett) proceduíáris vagy nemproceduláris működésűek, hanem tartalmaznak proceduíáris és nemproceduláris vég-

rehajtási mechanizmust. Az ilyen nyelveket *blokkorientált nemproceduláris* nyelveknek nevezik, mert a nemproceduláris blokkokra procedulárisan adható a vezérlés [3]. Ezekben a nyelvekben a *blokkok sorrendje* kötött, de a blokkon belül megengedett a párhuzamos működés. Pl. VLSI szintű áramkör leírásakor a proceduráris rész végzi az állapotanalízist, a nemproceduláris rész pedig a tényleges működést hajtja végre [5].

A végrehajtási mechanizmus fontos kiegészítő fogalma az *eseményvezéreltség*. Ez azt jelenti hogy a program utasításainak végrehajtására csak akkor kerül sor, amikor a leírt elem bemenetén valamennyi jelváltozás végbement. Az ilymódon létrejövő bemeneti „esemény" a vizsgált elem előírt kimenetének változását létrehozza. Ez az elv nagyon sok hasonlóságot mutat a dataflow elven működő számítástechnikai rendszerekkel.

3.4. Időzítési módszerek

A különböző HDL nyelvek eltérő módon veszik figyelembe az események, elemek időbeli viselkedését. Ezek főbb vonásai:

- — *szinkron időzítés:* ebben az esetben csak előre definiált fix frekvenciájú órajelhez való szinkronizálás engedhető meg. Az órajel lehet egyfázisú, vagy többfázisú
- — *aszinkron időzítés:* ezek a szimulációs rendszerek bármely jel felmenő vagy lemenő éléhez képesek a szinkronizációt elvégezni. Ilymódon lehet újabb „időzített" eseményt létrehozni. Minden átmenethez (jelváltozáshoz) külön-külön hold és setup idő rendelhető. Az időzítési viszonyok megsértése a kimenetet általában „ismeretlen" állapotba viszi [5].

A *késleltetések megadása* többféleképpen történhet:

- — Minden elemhez nominális késleltetés rendelhető
- — Külön kezelhetők a 0/1 és az 1/0 átmenethez tartozó késleltetések
- — Worst-case analízis esetén a késleltetések lehetséges minimális és maximális értékeit kell megadni.

Léteznek olyan szimulációs rendszerek, amelyekben a program előre rögzített időközönként ellenőrzi az időfeltételek teljesülését [6].

3.5. Jelek ábrázolása

Szimuláció során a specifikálandó áramkör bemenetei, kimeneteit és buszjeleit egyértelműen definiálni kell.

A *bemeneti jeleket* a következő attributomok határozzák meg:

- — jelek szélessége bitekben
- — számrendszerbeli értelmezése
- — jelek értékkészlete.

A *kimeneti jeleket* a fenti attributomok mellett jellemezni kell a kimenet típusával. A számrendszer megadásakor definiálni kell, hogy a jel bitjeit hány darab oktális, decimális vagy hexadecimális, stb. számjegyként értelmezzük.

A *sineket* meghatározó attributomok a fentieken kívül a sín típusa, jelterjedés iránya stb.

A szimuláció mélységétől függően a szimulációs programok a szokásos „0" és „1" mellett 4, sőt 7 és 8 értékű algebrát használnak. Buszok szimulációjánál elkerülhetetlen a nagy impedanciás („*Z*") állapot bevezetése. Az ismeretlen („*X*") állapotnak több jelentése is lehet:

- — teljes ismeretlen állapot (pl. bekapcsoláskor vagy szinkron beírás időviszonyainak megsértése utáni állapot)
- — átmeneti állapot (0—1 ill. 1—0 közben)
- — dinamikus „tüske" megjelenése

Egyes rendszerek ez utóbbi állapotokat elkülönítve kezelik. Így bevezetik a felmenő él (*R*), lemenő él (*F*) tüske 0 ill. 1 állapotban (*E*, *D*) jeleket. Előfordul, hogy külön állapotot használnak a buszkonfliktusok kezelésére (*V*) [5].

A tapasztalatok arra utalnak, hogy a szimulálandó áramköri készlet bonyolultságának növelésekor szűkebb jelkészlet is elegendő (0, 1, *Z*, *X*) [7]. A bő jelkészletet tipikusan a worst-case típusú szimulációs rendszer használja.

3.6. Adattípusok, adatabsztrakciók

HDL nyelvekben gyakran megtalálhatók mindazon adattípusok, melyek a magasszintű programozási nyelvekben előfordulnak [2]. Több szimulációs nyelvben a belső állapotok ábrázolása külön változóval történik. (Ez pl. megszakításrendszer leírásánál előnyösen használható).

Absztrakt adattípusokkal történő jeldeklarálás esetén a megfelelő hardware elemek későbbi szimulációs fázisokban kaphatnak értelmezést [3], [6]. Ez egyszerű lehetőséget ad nagy bonyolultságú, áramkörök hierarchikus specifikációjára és szimulációjára [4]. Ugyancsak előnyösen használhatók az absztrakt adattípusok vegyes leírású, többszintű szimulációs rendszerekben.

Az absztrakt adattípusok transzformációja ad lehetőséget a *software elemek* leírására (software szimuláció)

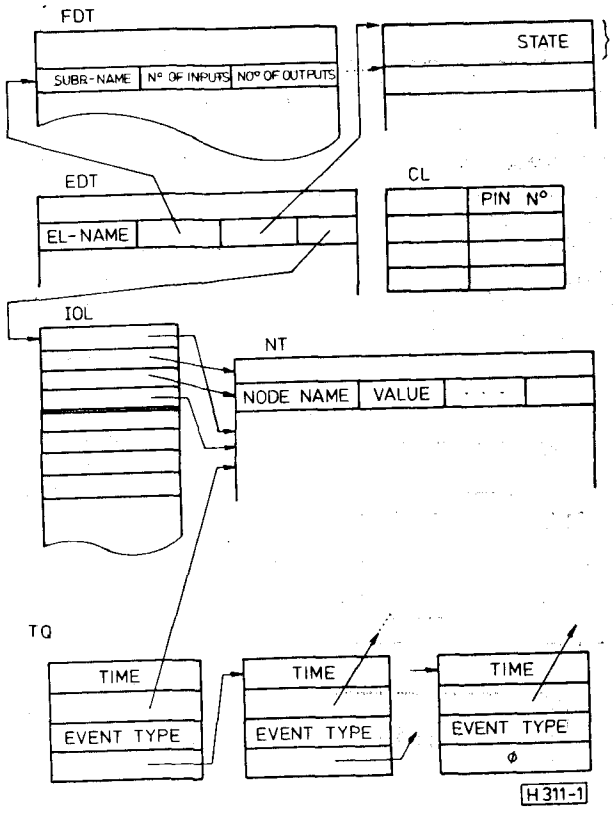
3.7. Adat és vezérlőjelek kezelése

Vezérelt állapotváltozáson alapuló szimulációs rendszer a jeleket adat típusú jelekre és vezérlő jelekre osztja szét. A vezérlőjelek száma a rendszer állapotainak számától függ, és így ezek száma sokkal kisebb, mint az adat típusú jelekké. Ez a szétválasztás lehetőséget ad egy-egy állapot jelenségeinek szimulálására. Ezt a jelábrázolási módot alkalmazzák az *automatákra* bontott struktúrát kezelő szimulátorok. Mikroprocesszorok és VLSI áramkörök leírására különösen jól alkalmazható a vezérelt állapotátmenetek módszere. Matematikai megfogalmazásban ez a *vezérlési gráfok* megadását jelenti. A vezérlési gráfhoz egyszerűen rendelhetők automaták.

A *feltételes állapotváltozások kezelése* egy másik sajátos módja a jelváltozások kiértékelésének. A jelek változása és az állapotváltozás feltételektől függ. A jel új értékét a szimulátor csak a feltétel teljesülésekor számítja ki, a feltétel teljesüléséig a szimulátor megőrzi annak régi értékét. Ez a jelkezelési rendszer az eseményvezérelt szimulátorokban kerül alkalmazásra.

Adatkonverziós jelkezeléssel működnek azok a szimulátorok, amelyek a bemenő jeleket folyamatosan és párhuzamosan kimenő jelekké alakítják át. Ez esetben a szimuláció alapvetően lista feldolgozást végez.

Az eseményvezérelt szimulációs rendszer által kezelt adastruktúra az *1. ábrán* látható.



1. ábra. Adatstruktúra

1. Az elemleíró táblázat (EDT) minden egyes bejegyzése egy áramköri elemnek felel meg. A bejegyzések a következő részekből állnak:
   - egy szimbólikus név, mely megkülönbözteti az elemet a hálózatban levő többi (esetleg hasonló) elemtől;
   - az áramköri elem típusa, ami tulajdonképpen hivatkozás a funkcionális leírások táblázatának egy bejegyzésére;
   - hivatkozás egy címre, melyen az állapotinformáció található.
     Ez csak szekvencionális típusú áramköri elemek esetén szükséges, és szerkezete, hossza változó lehet;
   - hivatkozás a ki- és bemenetek listájára.

2. Ki- és bemenetek listája (IOL). Minden listaelem egy hivatkozást tartalmaz a csomópontleíró táblázat egy bejegyzésére. A lista nem szükségszerűen láncolt, hiszen a ki- és bemenetek száma a funkcionális leírások táblázatából ismert.

3. Csomópontok táblázata (NT). Minden bejegyzése a hálózat egy csomópontjának állapotát rögzíti. Egy bejegyzés szerkezete a következő:
   - egy szimbólikus név (az adott csomópont azonosítója);
   - a jel aktuális értéke. Emellett a pontos időkezelést biztosító szimuláció számára még más információ tárolása is szükséges. Ilyen például az utolsó jelváltozás ideje, jelzés arról, hogy a jelérték bizonytalan, vagy arról, hogy változás alatt áll;
   - hivatkozás egy kötéslistára.

4. Kötéslisták (CL). Minden kötéslista megmutatja, hogy egy csomópont mely elemek mely bemeneteit hajlítja meg. A listák lezárására (egymástól való elválasztására) egy speciális jel szolgál.

5. Funkcionális leírások táblázata (FDT). Minden bejegyzés egy elemtípust határoz meg. Egy bejegyzés szerkezete:
   - hivatkozás egy szubrutinra, mely elvégzi az adott típusú elem funkcionális kiértékelését. (Ez lehet mindössze szimbólikus név, ami a hívandó szubrutin nevét jelenti);
   - a bemenetek száma;
   - a kimenetek száma.

6. Az idősor (TQ). Ez általában egy láncolt lista, melynek egy eleme a következő információkat tartalmazza:
   - az esemény ideje;
   - a változtatandó csomópontra való hivatkozás;
   - az esemény típusa (ez magában foglalja azt is, hogy mi legyen a csomóponthoz rendelendő új érték).

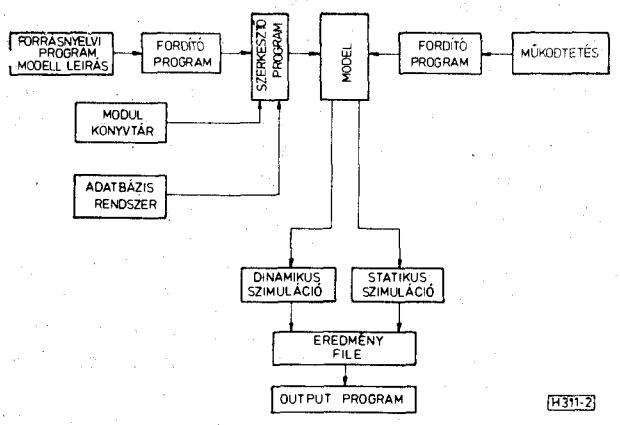
Az 1. ábrán lévő adatstruktúra főbb vonásaiban minden szimulátorban megtalálható, ezért az általános megoldásnak tekinthető valamennyi szimulátorban.

3.8. VLSI áramkörök leíró nyelve

A fentiek alapján VLSI áramkörök szimulálására alkalmas hardware leíró nyelv *főbb jellemzői:*
- proceduláris és nemproceduláris végrehajtási módok kombinált kezelése
- többszintű (esetleg a vegyes leírási módú) szimulációs rendszer támogatása
- precíz időkezelési mechanizmus, szinkron és aszinkron időzítések megadási lehetősége
- az adat és jelábrázoláshoz viszonylag szűkebb jelkészlet elegendő
- absztrakt adattípusok alkalmazása
- az adat és vezérlőjelek kezelése lehetőleg vezérelt állapotváltozáson alapuljanak.

LSI, VLSI áramkörök leírására alkalmas szimulátorok általános felépítése a *2. ábrán* látható.
A fordítóprogram tevékenységei:
- szintaktikai ellenőrzés;
- vizsgálat ellentmondásmentességre a típuson önmagán belül, valamint a külső funkcionális specifikáció és a struktúra leírása között;
- konzisztencia-vizsgálat a funkcionális specifikáció leírására;
- belső ábrázolási forma létrehozása;
- működtető jelsorozatok szintaktikai ellenőrzése.

A szimulálandó rendszer összeállítását a szerkesztőprogram végzi.
A szerkesztőprogram tevékenységei:
- összeszerkeszti a modell-leírás elemeit;



2. ábra. A szimulációs rendszer elvi felépítése

— az adatbázisban szereplő elemek adatait beépíti a modellbe;
— futtatható állapotú modellt hoz létre a szimulátorok számára.

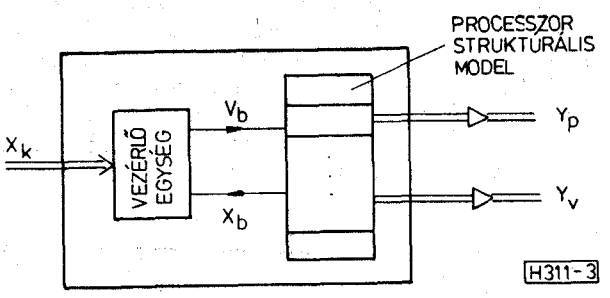
A szimulációs rendszer szolgáltatásai

— dinamikus szimuláció (a működtetendő rendszer elemeinek időhelyes működtetését végzi);
— az alacsonyabb szintű típusokból álló struktúra működtetésének a magasabb szintű funkcionális leírással történő összevetése (ehhez az elemi események szekvenciáiban fel kell ismernie a magasabb szintű összetett eseményeket, működéseket);
— bonyolultabb adatszerkezetek összeállítása az alacsonyabb szint egyszerűbb szerkezeteiből;
— statikus szimuláció (a modellbeli elemek működési idejét egységnyinek tekinti, így csak a funkcionális működés ellenőrzésére szolgál);
— output program (a szimulátorok által létrehozott eredmény file adatait írja ki a felhasználó által kívánt formátumban).

Egyes hardware leíró nyelvek sok vonatkozásban hasonlítanak a magasszintű programozási nyelvekhez. Pl. az *Ada* nyelv a VHSIC (Supergyors integrált áramkörök leírására kidolgozott nyelv) között feltűnően sok a hasonlóság. A szintaktikai felépítés jelentős részben azonos, a két nyelv szemantikai felépítése teljes egészében megegyezik [8].

Szimulációs nyelvek általános felépítése

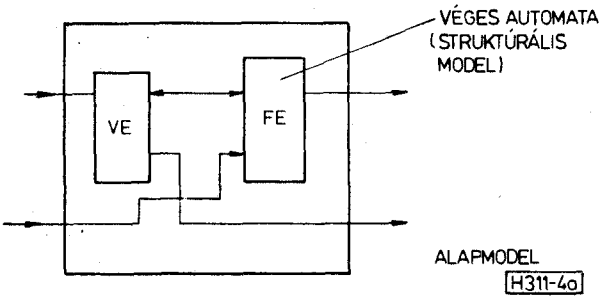
A vizsgálandó VLSI áramkört — annak részletes belső felépítésétől eltekintve — bontsuni fel két funkcionális részre *(3. ábra)*: az előírást működtető megvalósító processzorra és ennek vezérlését végző vezérlő egysége. A processzor több funkcionális egységből áll, pl. tároló, művelet végző stb. Ezeknek a funkcionális egységeknek az összehangolt működését a vezérlő egység biztosítja. Ezt a hatást fejezi ki a V_b belső vezérlő jelek halmaza. A vezérlő egység információkat kap a processzorban lévő funkcionális egységektől (pl. utasítás lehívás) és a külső környezettől (pl. perifériák input jele). Ezt a két hatást jelöli: X_b, X_k. Mind a processzor, mind a vezérlő egység további külső egységeket működtethet (Y_p, Y_v). Ilymódon mind



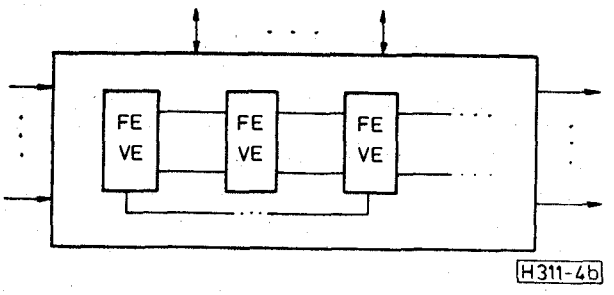
3. ábra. Alapmodell

a processzor, mind a vezérlő egység két bemenettel és két kimenettel rendelkezik (rendelkezhet). (Kommunikációs változatok a processzor között.)

Ennek a szemléleti módnak az általánosítása számos absztrakt modell kidolgozásához vezet. Az egy processzor — egy vezérlő egység különböző szimulációs nyelvekben más, más elnevezést kap. Pl. a DDL nyelvben [9] funkcionális egység —



4/a. ábra. Alapmodell

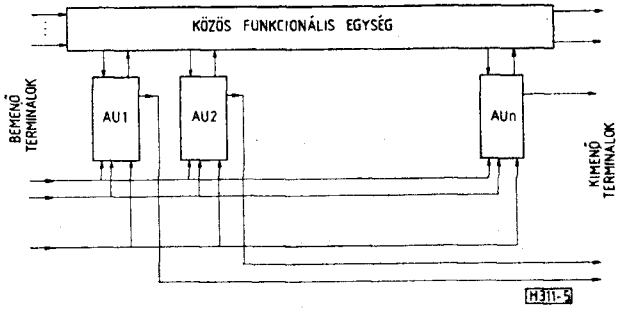


4/b. ábra. Rendszer dekompozíció

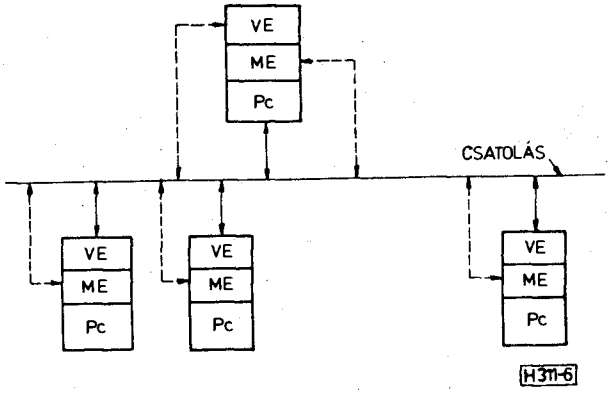
vezérlő egység (FE—VE). Az így létrejövő PE—VE párt automatának (AU) nevezik *(4. ábra)*. Több automatából rendszert építhetünk fel. Ugyancsak a DDL nyelv felfogásában ez az a *5.* ábra szerint értelmezhető. Az 5. ábrából közvetlenül származtatható az osztott feldolgozó rendszerek egyik típusa, a vektor feldolgozása *(6. ábra)*. A szaggatott adat és vezérlő utak a vektor processzorokban szokásos kapcsolatot fejezik ki. Az ábra alapján egyszerűen alakíthatunk ki lazán és szorosan csatolt rendszereket *(7., 8. ábra)*. A 7. ábrán a közös funkcionális egység a tár (ME), a 8. ábrán pedig a közös funkcionális egység a processzorok (PC) közötti csatolás.

Az egyes blokkok közti kapcsolatok állhatnak bináris jelsorozatokból, összeköttetési és transzfer

operációkból. Az összeköttetési operátor kapocspárok között jelent összeköttetést, a transzfer operátor memória (regiszter) elemek között jelent átvitelt.

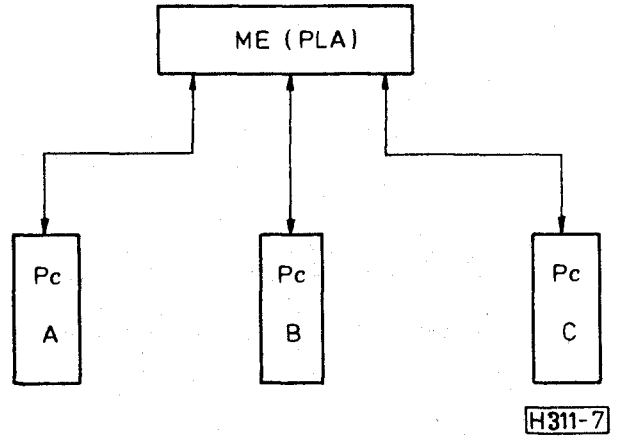


5. ábra. DDL (SDL) Nyelv hardware alapmodellje. (A vizsgálandó áramkör részekrebontásának elve)



6. ábra. Vektor (Mátrix) processzálás elve

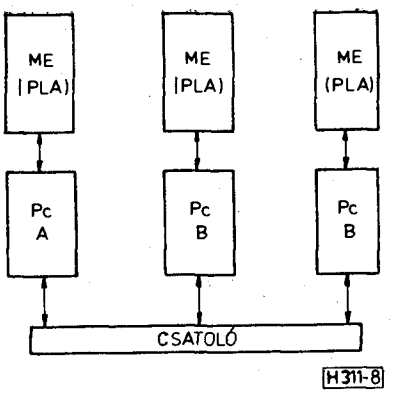
Jól látható, hogy ez a szemléleti mód a *vezérelt állapotváltozáson* alapuló adat és vezérlőjel kezelés hardware modellezéséhez vezet.



7. ábra. Szorosan csatolt processzorok

A VLSI technológia felhasználható nagyteljesítményű általános és speciális célú, fejlett architektúrájú gépek, mint építőelemek kialakítására. Ezekből az elemekből kialakítható a:

— probléma-megoldó gépek
— következtető gépek



8. ábra. Lazán csatolt processzorok

— tudásbázis kezelő gépek
— intelligens I/O vezérlő rendszerek.

Az előzőekben bemutatott blokkok leírására használjunk bináris jelsorozatot. Nevezzük a bemeneti és kimeneti kombinációból álló sorozatokat *specifikációs bemeneti és kimeneti* sorozatoknak. Ezek száma egy-egy blokk esetén igen nagy lehet. Így kiindulhatunk abból, hogy az egyes egységek specifikációját az összetartó bemeneti és kimeneti kombináció sorozatokkal adjuk meg.

5. Jelsorozaton alapuló specifikációk formális leírása

A specifikációs kombináció sorozatokból válasszuk ki az egymást közvetlenül követő kombinációpárokat. Ezek legyenek *a* specifikációs bemeneti ill. kimeneti *változások*. Ezek halmazával specifikáljuk a vizsgálandó egységet. Legyen X a bemeneti kombináció, Y a kimeneti kombináció és így egy lehetséges bemeneti ill. kimeneti kombináció sorozat pl. az alábbi:

$$\begin{matrix} X_1 & X_2 & X_5 & X_3 & X_4 & X_6 \\ Y_1 & Y_3 & Y_5 & Y_2 & X_4 & Y_6 \end{matrix}$$

A továbbiakban jelöljül az $X_i \rightarrow X_j$ változást ΔX_{ij}-nék, vagy a bemeneti változást jobban kifejezve B_{ij}-nek. Hasonlóan átértelmezzük a $\Delta Y_{kr} = K_{kr}$ jelölését is a kimeneti változásra.

A specifikációs *összetartozó* előírt bemeneti ill. kimeneti változások halmaza jellemzi tehát a vizsgálandó egységet és ezt tekintsük:

$$\{B : K\}$$

halmaznak.

6. B : K halmaz jellemzői

6.1. Δ halmaz érzéketlensége

Az egység érzéketlen lehet a bemenő sorozat egy részére, ha X_i után fellépő X_{i+1}, X_{i+2}, ..., X_{i+n} hatására az X_i-hez tartozó Y_j kimeneti jel nem változik. Pl.

$$\begin{matrix} X_1 & X_3 & X_4 & X_5 & X_6 & X_7 & X_8 & X_9, \\ Y_1 & Y_5 & Y_2 & Y_2 & Y_2 & Y_2 & Y_8 \end{matrix}$$

ugyan ezt a jelenséget a $B:K$ halmazzal jellemezve:

$$\begin{matrix} B_{13}\ B_{34}\ B_{45}\ B_{56}\ B_{67}\ B_{78}\ B_{89}, \\ K_{15}\ K_{52}\ K_2\ K_2\ K_2\ K_2\ K_{28} \end{matrix}$$

láthatóan az X_4 után fellépő $X_5\ X_6\ X_7\ X_8$ ill. a $B_{34}=\Delta X_{34}$ után Y értéke nem változik (marad Y_2). Az érzéketlenséget az X_9 bemeneti változás szünteti meg. A vizsgált egység működését tehát a B_{34} nem specifikálja $X_5\ X_6\ X_7\ X_8=\delta X_{58}$-ra. Ezt a δX_{58}-at a bemeneti *érzéketlenségi sávnak* tekinthetjük. Szokásos egyéb jelöléssel:

$$\delta X_{58}=\delta B_{58}$$

Mégegyszer fehrva:

$$\begin{matrix} \Delta X_{13}\ \Delta X_{34}\ \Delta X_{45}\ \Delta X_{56}\ \Delta X_{67}\ \Delta X_{78}\ \Delta X_{89} \\ \Delta Y_{15}\ \Delta Y_{52} \qquad\qquad\qquad\qquad\qquad\qquad \Delta Y_{28} \end{matrix}$$

vagy az érzéketlenségi sávot jelképező jelöléssel:

$$\begin{matrix} \Delta X_{13}\ \Delta X_{34}\ \delta X_{58}\ \delta X_{49} \quad \Delta X_{13}\ \Delta X_{34}\ \delta X_{49} \\ \Delta Y_{15}\ \Delta Y_{52}— \quad\ \Delta Y_{28} \quad\ \Delta Y_{15}\ \Delta Y_{52}\ \Delta Y_{28} \end{matrix}$$

6.2. Bemeneti változás nélkül fellépő kimeneti változás

Pl. Legyenek a be/ki jelsorozatok:

$$\begin{matrix} X_1\ X_3\ X_4\ X_4\ X_5\ X_6 \\ Y_1\ Y_5\ Y_2\ Y_{15}\ Y_1\ Y_3 \end{matrix}$$

Látható, hogy az X_4 változása ellenére a kimenet Y_2-ről Y_{15}-re változik. Ez a jelenség vezet el az automatának nevezett egység teljesebb leírására, mivel ez a jellemzés nem veszi tekintetbe a belső állapotokat. (Gondoljunk egy számláncra, ahol a bemeneti jel ismétlődése a kimenet változását okozza.)

Ezért az automata leírását az összetartozó előírt bemeneti ill. kimeneti változások halmazán kívül ki kellene egészítenünk az állapotok (szekunder változók) változásának halmazával. Az automata ilymódon a

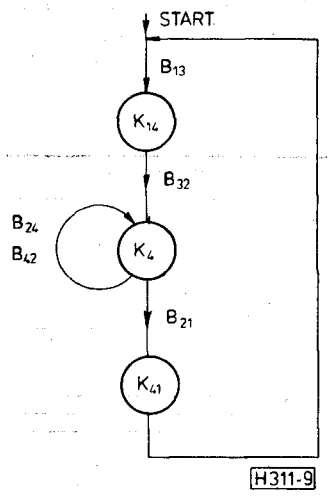
$$\{B:K:A\}$$

halmazzal specifikálhatóvá válna, ahol A az állapotok halmaza.

Ennek ellenére maradjunk a kiinduláskor megadott $\{B:K\}$ halmaz mellett, és vizsgáljuk meg, hogy ez milyen feltét mellett jelent egyértelmű és ellentmondás mentes leírást. Kimutatható ugyanis, hogy a $\{B:K\}$ halmaz ellentmondás mentessége esetén magába foglalja az általa jellemzett állapotokat vagy más kifejezéssel az általa leírt áramköri elem szekunder változóit. A szekunder változók a $\{B:K\}$ halmazból formális eszközökkel előállíthatók [10].

6.3. $B:K$ halmaz ellentmondás mentességének a feltétele

A feltétel vizsgálatát kezdjük meg egy példán. Vizsgáljuk meg az alábbi előírt bemeneti és kimeneti változás sorozattal specifikált egységet:



9. ábra. $\{B:K\}$ halmaz irányított gráfja

$$\begin{matrix} B_{13}\ B_{32}\ B_{24}\ B_{42}\ B_{24}\ B_{42}\ B_{21} \\ K_{14}\ K_4\ K_4\ K_4\ K_4\ K_4\ K_u \end{matrix} \qquad (1)$$

legyen a kiindulási állapot B_{13}. A sorozat láthatóan a kiindulási állapotba tér vissza. Ábrázoljuk a $\{B:K\}$ halmazt irányított gráffal *(9. ábra)*. A gráf csomópontjai a kimeneti változásoknak, élei pedig a bemeneti változásoknak felelnek meg. A 9. ábrát felrajzolhatjuk a csomópontok elhagyásával, is, mind a bemeneti, mind a kimeneti változásokat az élek alatt feltüntetve *(10. ábra)*. Az ábrából már látható az ellentmondás, hiszen a $B_{24}\ B_{42}$ sorozat tetszőleges számában ismételhető, és a kimenet változatlanul K_4 marad. Tehát a nem csak az (1)-el jelölt sorozatra érvényes a 9. és 10. ábra, hanem minden olyan sorozatra, melyben B_{32} után $B_{24}\ B_{42}$ tetszőleges számban ismétlődik.

Az érzéketlenségi sáv bevezetésével az (1)-es specifikáció a következőképpen írható le:

$$\begin{matrix} B_{13}\ B_{32}\ \delta B_{21} \\ K_{14}\ K_4\ K_{41} \end{matrix}$$

Ezt a gráfot a 11. ábrán tüntettük fel.

Az ellentmondás mentesség matematikailag is megfogalmazható, ha a $\{B:K\}$ halmazt kiegészítjük a következő négy $B/K_{ij}/$; $K/K_{ij}/$; $/K_{ij}/B$: $/K_{ij}/K$ halmaz bevezetésével. A jelölések értelmezése a következő:

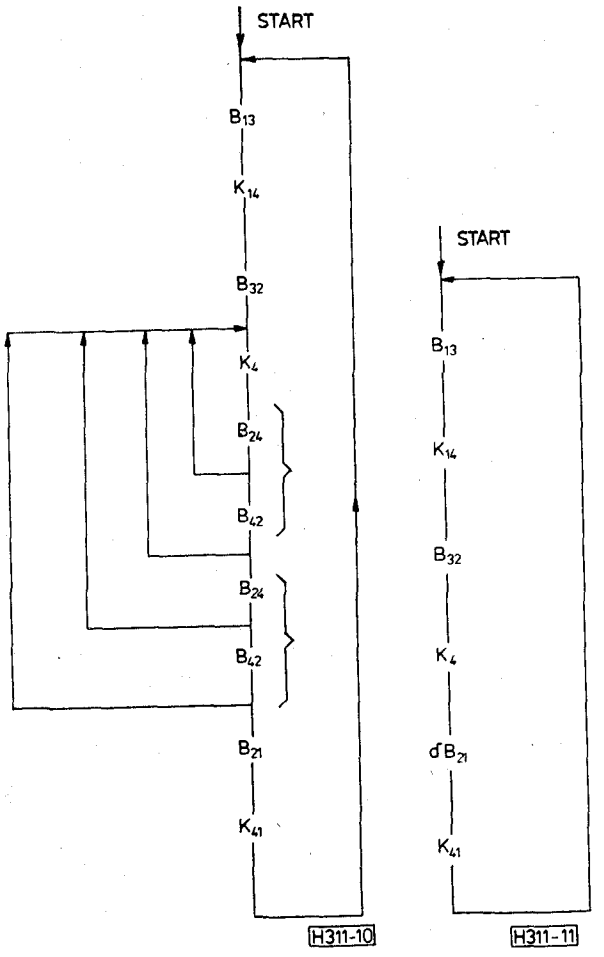
— $B/K_{ij}/$ előállítja az összes olyan bemeneti változás halmazát, amelyhez kimeneti változás van előírva és megelőzi a K_{ij} kimeneti változást. Így (1) alapján:

$$B/K_{14}/=B_{13};\ B/K_4/=\{B_{32},\ B_{24},\ B_{42}\}$$
$$B/K_{41}/=B_{24}$$

— $K/K_{ij}/$ előállítja az összes olyan kimeneti változás halmazát, amely szomszédosan megelőzi Ki_j-t. Így (1) alapján:

$$K/K_{14}/=K_{14};\ K/K_4/=\{K_{14},\ K_4\}$$
$$K/K_{41}/=K_4$$

Az utolsó kifejezést úgy írtuk fel, hogy figyelembe vettük, hogy K_{41}-et csak K_4 előzi meg *szomszédosan*



10. ábra. Ellentmondást tartalmazó $\{B : K\}$ halmaz

11. ábra. Érzéketlenségi sávot tartalmazó $\{B : K\}$ halmaz

— $/K_{ij}/B$ előállítja (kiválasztja) az összes olyan bemeneti változás halmazát, amely szomszédosan követi K_{ij}-t. Így (1) alapján:

$$/K_{14}/B = B_{32};\ /K_4/B = \{B_{24}, B_{42}, B_{21}\};\ /K_{41}/B = B_{13}$$

— $/K_{ij}/K$ előállítja az összes olyan kimeneti változás halmazát, amely szomszédosan követi K_{ij}-t. Így (1) alapján:

$$/K_{14}/K = K_4;\ /K_4/K = \{K_4; K_{41}\};\ /K_{41}/K = K_{14}$$

Igazolható, hogy ha van olyan bemeneti változás, melynek hatására két különböző, de a bemeneti vezérlésből el nem dönthető kimeneti állapot fordul elő, akkor a $\{B : K\}$ halmaz ellentmondásos, vagyis a $\{B : K\}$ halmaz nem eredményezi a vizsgált automata egyértelmű leírását.

A fenti állítás-formális-halmazelméleti megfogalmazása a következő: ha nem létezik olyan K_{im}, K_{mp} és K_{mr} előírt kimeneti változás, amelyekre:

$$K_{mp} \in /K_{lm}/K \text{ és } K_{mr} \in /K_{lm}/K \text{ és } /K_{lm}/B \cap B/K_{mp}/ \cap B/K_{mr}/ \neq 0$$

fennáll, akkor a $\{B : K\}$ halmaz az előírt bemeneti és kimeneti változások sorozataival specifikált folyamat ellentmondásmentes leírásának tekinthető.

(A jelölések a szokásos halmazalgebrai értelmezésűek: $\in$ és $\cap$ szimbólumok a tartalmazásra ill. közös rész képzésre szolgálnak, 0 pedig az üres halmazt jelenti.

7. Jelsorozaton alapuló szimulációs vizsgálat

Amennyiben a vizsgálandó áramkör belső felépítése ismeretlen, tehát nem tudunk pl. véges automatákból álló hardware modellt hozzárendelni, és ezen alapuló szimulációs leírónyelvet választani, akkor a $\{B : K\}$ halmaz lehet a szimuláció alapja.

A vizsgálat lépései:

— $\{B : K\}$ halmaz vizsgálata: érzéketlenségre, ellentmondás mentességre stb.
— Kimeneti „tüskék" fellépésének vizsgálata
— $A\ \{B : K\}$ halmaz alapján a belső állapotok meghatározása (szekunder változók számának megállapítása)
— Fentieken alapuló *„hipotétikus" modell* (automata modell) hozzárendelése a szimulálandó áramkörhöz.

Amennyiben a $\{B : K\}$ halmazzal jellemzett áramkör particionálható, a vizsgálat közelít a magas szintű leíró nyelveken alapuló eljárásokhoz, de mindenképpen csak *hipotétikus* modell létrehozását eredményezi.

8. Publikált szimulációs eljárások

A publikált szimulációs nyelvek csekély része származik áramkör gyártó cégektől — mivel egyrészt a gyártó cégek nem érdekeltek tervezési eljárásaik nyilvánosságra hozatalában, másrészt az áramköri elemek pontos belső megvalósítását — és így modellezését — nem közlik. A nyelvekre az irodalomban található rövidítéssel hivatkozunk, ezek pontos feloldására az irodalomban találunk segítséget.

8.1. IDL nyelv

Az IDL [11] egyesíti a hardware szimulációt, a leíráson alapuló szintézist és a tervezett áramkör dokumentálást. A szimuláció és a tervezés a felülről-lefelé történő módszeren alapul. Kb. 40 000—50 000 ekvivalens kapuból álló hálózat leírására alkalmas. Nonproceduláris nyelv, komplex algoritmusok kezelésére alkalmas. Bemenete lehet *grafikus*, ebben az esetben a *folyamatábra* alapján történő leírást veszi alapul, de készült hozzá magas szintű leíró nyelv is. A szimulációs leírás öndokumentáló. Az IDL implementálása részben az Iverson által kidolgozott APL nyelven készült, részen IBM S/370 assembler-ben. A magas szintű nyelv szintakxisa hasonlít a standard angol nyelvéhez.

A nyelv mindig két tervezői szintet kezel egyidejűleg és a magasabb szintű leírásból eggyel alacsonyabb szintű leírást állít elő. (Hierarchikus szintézis.) Tartalmaz statikus szimulátort, amely zérus elem késleltetéssel pusztán a logikai műkö-

dést ellenőrzi, és dinamikus szimulátort. A dinamikus szimulátor 0, 1, *Z*, *X* jeleket kezeli. A leírás alapvetően az *automatákra bontás* elvén alapul, tervezés során *optimalizálja* a kombinációs hálózatot, és a belső állapotok számát (szekunder változók számát). Mind a strukturális, mind a funkcionális leírást tartalmazza. A párhuzamos folyamatok kezelését a nyelv *nemprocedurális* jellege teszi lehetővé. A nyelvhez tartozó ekvivalencia program elvégzi az egyidejűleg kezelt két szintű leírást ekvivalencia vizsgálatát. Az utasításokhoz rendelhető feltételek komplex logikai függvényekből és komplex relációkból állhatnak. A nyelv hierarchikus jellege a cellakönyvtárig történő szimulációt és tervezést teszi lehetővé. Az elemek funkcionális működésének leírása APL nyelven történik. Az IDL rendszerrel számos áramkört terveztek. A rendszer jelenleg az IBM és a M. I. T-ben használják.

8.2. SDL nyelv

Az SDL nyelv [12] a DDL nyelven alapuló [9] rendszer tervező nyelve, alapvetően VLSI áramkörök szimulálására és tervezésére készült. Technológiától független, regiszter transzfer szintű hardware tervező és dokumentáló nyelv. Mindegy 40 000 ekvivalens kapuból álló áramkör modellezésére alkalmas, 128 KByte helyfoglalással. A nyelv az 5. ábrán bemutatott hardware koncepción alapul. Az implementálás PL/S (Programming Language for Systems) és IBM S /370 assembler nyelven történt. A nyelvek fejlődését figyelembe véve „klasszikus" elemekből álló automatákat és közös funkcionális egységet tartalmaz az 1968-ban publikált DDL nyelv szintaxisát és szemantikáját alapul véve. Tervezésre is alkalmas változata 1984-ben készült el. A szintézis két szintű, az SDL nyelvű (magasszintű leírást) egy logikai transzformáló rendszer (LTS) közbülső kódra fordítja és ebből állítja elő a pontos időzítéseket is figyelembe vevő áramköri realizációt és ennek dokumentálását. Az SDL rendszer párhuzamos, konkurens folyamatokat kezel, a vezérlés az általánosan használatos állapot koncepción alapul.

8.3. VHDL nyelv

A VHDL rendszert a USA Department of Defence megrendelésére az IBM és a Texas Instruments fejlesztette ki, hardware tervezésre és leírásra. Általános leírás a rendszerről pl. a [8] irodalomban található. Alapvetően VLSI specifikációra és az ezen alapuló tervezésre készült és 1985. decembere óta használják. A tervezhető áramkör kb. 100 K (tehát kb. 100 000) ekvivalens kapuból állhat. Az áramkör specifikálása hardware leíró nyelven történik, a tervezés kevert (vegyes) leírású, többszintes eljárással történik. (Hierarchikus szintézis.) A tervező rendszer a specifikációból több alternatív megoldást állít elő. A leíró nyelv regiszter transzfer szintű és mind a strukturális, mind a funkcionális leírásra alkalmas. Az időkezelés igen fejlett, szupergyors VLSI áramkörök leírását, tervezését is lehetővé teszi. A nyelv implementálásához az Ada nyelvet maximális mértékben felhasználták, mivel a VHDL szintaktikai szerkezete nagyrészt azonos, szemantikai felépítése megegyezik az Ada nyelvvel. A tervezéskor a megrendelő egyik fő követelménye volt az „Ada használata, mindenhol, ahol lehetséges". A nyelv nemproceduláris. A tervező rendszer hierarchikus.

8.4. MODLAN nyelv

A gliwicei egyetemen (Lengyelország) dolgoztak ki és egy strukturális és funkcionális leírásra alkalmas leírónyelvet, mely számos egyéb nyelv sajátságát magába foglalja (SARA, DIGITEST, CAP, SCL stb.). A nyelv hierarchikus szerkezetű, többszintes leírásra alkalmas [14], [15]. Alapvetően regiszter transzfer szintű nyelv, de segítségével mikroprocesszoros hálózatok, rendszerek, LSI áramkörök is leírhatók. Hétértékű jelábrázolást tartalmaz. Támogatja a worst-case analízist. A funkcionális modul proceduláris és nemproceduláris lehet. Ezek összehuzalozásából áll elő a strukturális modul. A szimulálandó áramkör vezérlő és végrehajtó (processzor) modulból áll automaták sorozatából áll (4. és 5. ábra). A vezérlés leírása szinkronizált Petri hálóval történik. Időkezelése precíz.

8.5. ADLIB nyelv

Az ADLIB nyelv [6] a SABLE szimulációs rendszer bemenő nyelve. A Pascal programnyelv kibővített változata, így igen magas szintű absztrakciós lehetőséggel rendelkezik. A Pascal procedurális vezérlési szerkezete mellett háromféle nemproceduláris vezérlést is megenged. A jelkésleltetséek lehetnek szinkron és aszinkron időzítésűek. Vegyesszinten írhatók le a funkcionális modulok. Az időkezelést a Pascal nyelv időkezelő és eseményre várakozó utasításokkal való kibővítésével valósították meg.

6.8. Conlan-projekt

Az előző rövid ismertetésből is látható, hogy igen sok harware leíró (tervező) nyelv került kidolgozásra, a speciális célok maximális figyelembevételével. A hardware leírónyelvek egyesítésére történtek kísérletek az IFIP keretén belül. Ez az ún. CONLAN-projekt [13]. Ennek keretén belül egy olyan általános szintaxist definiáltak, melyből levezethetők lennének azok a konkrét nyelvek, amiket a felhasználó alkalmazni kíván. Ennek segítségével egyetlen fordító program lenne elegendő (amely természetesen igen bonyolult lenne) valamennyi implementáció számára. Minden konkrét esetben a megfelelő deriválási szabályok bevitelére lenne szükség. A Conlan projekt idáig nem váltotta be a hozzá fűzött reményeket, nem került realizálásra, s újabb és újabb leíró nyelvek készülnek, más és más implementáló nyelvek alkalmazásával.

9. Magasszintű programozási nyelvek alkalmazása hardware leírás céljára

Jelenleg törekvések vannak modern programnyelvek kiegészítésére, abból a célból, hogy segítségük-

kel hardware rendszerek leírhatók és tervezhetők legyenek. Ez á törekvés tükröződött az igen korszerű VHDL és az ADLIB nyelv létrehozásakor. Eddig az alábbi nyelvek kerültek kipróbálásra: PL/L Pascal, ALGOL—68, C, Ada [16], [17].

A kiegészített magasszintű nyelvnek lehetővé kell tenni a:

— többszintű, hierarchikus leírást
— az áramör strukturáláis leírását
— a hálózati elemek funkcionális leírását
— a pontos időkezelést
— szimulációs parancsok megadását
— leíráson alapuló tervezést
— a tervezett áramkör dokumentációját.

Mindezen információ a felhasznált programnyelv segítségével adandó meg.

IRODALOM

[1] *Bohus M., Csopaki Gy., Filp A., Hinsenkamp A., Máté L.:* Computer Aid for Reeonrsive Synthesis. Working. Paper. Computer and Automation Institute, Hungarian Academy of Sciences, Apr. 1982.

[2] *Barbacci M. R.:* Syntax and Semantics of CHDLs. 5th Internation Symposium on CHDLs and their Applications, 1981.

[3] *Singh A. K., Tracy J. H.:* Development of Comparasion Features for Computer Hardware Description Languages. 5th International Symposium on CHDLs and their Applications. 1981.

[4] *Hill, D. D., Van Cleemput:* SABLE, a Tool for Cenerating Structured, Multi-Level Simulations. 16th Design Automation Conference. 1979.

[5] *Pawlak, A., Jezenski, J.:* MODLAN — a Language for Multilevel Description and Modeling of Digital Systems. 5th International Symposium on CHDLs and their Applications. 1981.

[6] *Hill, D. D.:* ADLIB, a Modular, Strongly-Type Computer Design Language. 4th International Symposium on CHDLs. 1979.

[7] *Wilcox P. S., Shew, E., DesMarais, P.:* A Functional Level Modelling Language for Digital Simulation. 19th Design Automation Conference. 1982.

[8] *Shahdad, M.:* VHSIC Hardware Description Language. Computer. 1985. Febr.

[9] *Duley, J. R., Dietmeyer, D. L.:* A Digital System Language (DDL). IEEE Trans. Computers. c. 17, (1968) p. 850—861.

[10] *Arató P.:* Kogikai rendszerek tervezése. Tankönyvkiadó. Budapest, 1985.

[11] *Maissel, L. J., Ostapenko, D. L.:* Interactictive Design Language: A Unified Approach to Hardware Simulation, Syntheis and Documentation. Proceedings of the 19th Design Automation Conference. 1982.

[12] *Saunders, L. F.:* An Approach to VLSI Design Electronic Design Automation. 1984.

[13] *Piloty, R.:* The CONLAN Projekt — Status and Future Plans. 19th Design Automation Conference. 1982.

[14] *Pawlak, A., Jezewski, J.:* MODLAN — a Language for Multilevel Description and Modeling of Digital Systems. 5th International Symposium on CHDLs and their Applications, 1981.

[15] *Pawlak, A.:* Digital Logic Modeling System Based on MODLAN. 19th Design Automation Conference. 1982.

[16] *Robinson, P., Dion, J.:* Programming Languages for Hardware Description. 21. st Design Automation Conference. 1983.

[17] *Doshi, M. H.:* THEMIS Logic Simulator- a Mix Mode, Multi-Level, Hiearchical, Interactive Digital Curcuits Simulator, 22nd Designs Automation Conference. 1984.