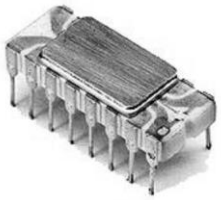
Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai kar
Elektronikus Eszközök Tanszéke

Tudományos diákköri dolgozat

Tervezési keretrendszer alkalmazásorientált mikroprocesszorokhoz



Készítette: Horváth Péter

Konzulens: Dr. Hosszú Gábor e. docens

2011.

Tartalom

1. BEVEZETÉS

A digitális rendszerek viselkedésének modellezésére kifejlesztett hardver leíró nyelvek többféle elvonatkoztatási szinten teszik lehetővé eszközeink modellezését, tervezését. Az összetettség jelentős növekedése szükségessé teszi a magas szintű szintézis (*High Level Synthesis* - **HLS**) alapú áramkörtervezés alkalmazását. Ennek lényege, hogy egy algoritmikus leírásra alkalmas nyelvből (C, C++, stb.) és az áramkör szerkezetét meghatározó specifikációból kiindulva a kapuszintű leírást egy szintézis szoftver (SystemCrafter, CatapultC, stb.) segítségével gépi úton állítjuk elő. Bár az algoritmussal megfogalmazható feladatok elvégzésére alkalmas eszközök fejlesztése során ma már elterjedt a HLS módszer használata, azonban a tárolt programú gépek (mikroprocesszorok) tervezésének elsődleges eszköze még ma is a *regiszter-átviteli szint*. Ennek két oka van: egyrészt a mikroprocesszorok nagymértékben optimalizált aritmetikai és logikai egységéhez (ALU) képest a szintézis program által létrehozott áramkör általában lassabb és nagyobb helyfoglalású, másrészt a tárolt programú gépek viselkedése bonyolult architektúrák esetén eleve nehezen fogalmazható meg algoritmikus formában.

Munkám során kifejlesztettem egy automatizált módszert a tárolt programú gépek magas szintű szintézissel (HLS úton) történő tervezésére. Ezt úgy értem el, hogy létrehoztam egy új leíró nyelvet (*Algoritmikus RTL* - **ARTL**), amely egyszerre alkalmas az algoritmikus leírásra és a mikroprocesszorokra jellemző áramköri szerkezet specifikálására. A kifejlesztett ARTL nyelv regiszter-átviteli szintű erőforrások közötti adatáramlások megadásával algoritmikus formában definiálja a mikroprocesszor viselkedését. A mikroprocesszor struktúrájának leírásához létrehoztam egy olyan általános célú mikroprocesszor típust, amelynek viselkedése algoritmus formában is megfogalmazható, szintén az ARTL nyelv felhasználásával. Ezzel a nagy elvonatkoztatási szintű leírások automatikus szintézisét (ami a HLS lényege) alkalmaztam a mikroprocesszorok tervezésére. A módszer lehetőséget nyújt arra is, hogy a megtervezett mikroprocesszor ne csak általános célú, hanem egy-egy adott alkalmazásra optimalizált legyen.

A dolgozat első része ismerteti az általam kifejlesztett kevert elvonatkoztatási szintű digitális modellezési nyelv (ARTL) szintaktikáját és erőforrástípusait, továbbá bemutatja az ARTL modellben az algoritmus és az ahhoz igazodó szerkezet viszonyát.

A második rész egy új, FPGA technológiára optimalizált mikroprocesszor architektúrát, a *folyamatvezérelt mikroprocesszor modellt* (*Process-controlled Machine*,

PCM) mutatja be, valamint kitér az erőforrásigény szempontjából kulcsfontosságú funkcionális egységek lehetséges implementációjára is.

A dolgozat harmadik része azt az általam kifejlesztett szoftverrendszert ismerteti, mely lehetővé teszi az *ARTL nyelven modellezett folyamatvezérelt mikroprocesszorok* tervezését. Összehasonlító vizsgálatokkal igazoltam, hogy az így kialakított eljárással a szakirodalomban szereplőknél lényegesen gyorsabban tervezhetők meg alkalmazásközpontú utasításkészleteket megvalósító hardver eszközök.

A dolgozat befejező részében ismertetésre kerülnek a bemutatott tervezési módszer és szoftverrendszer segítségével kifejlesztett tárolt programú mikroprocesszorok, amelyek tesztelését szimulációs környezetben és FPGA áramkörbe implementálva egyaránt elvégeztem, ezáltal igazolva a dolgozatban ismertetett tervezési módszer alkalmazhatóságát.

2. IRODALMI ÁTTEKINTÉS

2.1. Hardver modellezés és szintézis

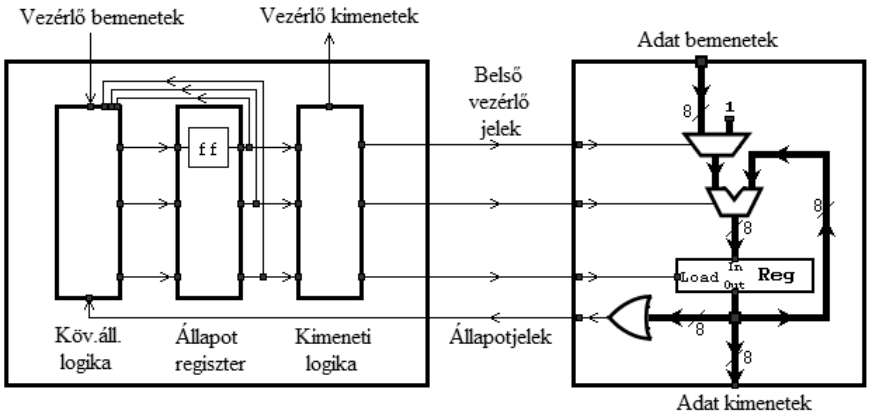
A szintézis szoftverek célja egy magas, az emberi gondolkodáshoz közelebb álló elvonatkoztatási szintű leírás átalakítása egy alacsonyabb szintű leírássá. Az RTL szintű hardver modellekből a digitális áramkörök kapuszintű leírása az ún. logikai szintézis szoftverek segítségével állítható elő [1]. A modellezésre használt hardver leíró nyelvek többféle elvonatkoztatási szinten teszik lehetővé a tervezendő áramkör működésének leírását, a szintézist végző szoftverek hatékonysága azonban erősen függ a kiindulási- és a végállapot közötti „szintkülönbségtől". Az emberi gondolkodáshoz legközelebb álló algoritmikus jellegű leírásokat a logikai szintézis szoftverek egyáltalán nem, vagy csak jelentős korlátozásokkal képesek feldolgozni. E problémát napjainkban az ún. magas szintű szintézis eljárásokkal és szoftverekkel oldják meg, melyek általában egy már létező programozási nyelvből (C, C++), illetve annak egy továbbfejlesztett, kimondottan ilyen célra optimalizált változatából (SystemC, CatapultC) állítják elő a kapuszintű leírást [2]. E megoldások hátránya azonban, hogy a szintézis során kész, a szintézis szoftverbe beépített funkcionális egységeket alkalmaznak, amelyek a tervező számára közvetlenül nem hozzáférhetők, ez pedig a tervezendő digitális rendszer optimalizációját (pl. erőforrásigény-minimalizálás) gátolja.

Egy másik fejlesztési irányt képviselnek az alkalmazásközpontú utasításkészletek és az azokat implementáló hardver megtervezését segítő eszközök. Az alkalmazásközpontú utasításkészlettel rendelkező mikroprocesszorok fejlesztésének automatizálása nehéz feladat abból adódóan, hogy szerteágazó tudományterületekben való jártasságot igényel (szoftverfejlesztés, hardvertervezés, verifikációs eljárások stb.). A meghatározó elvonatkoztatási szint ezen eszközök tervezése során még napjainkban is a regiszter-átviteli szint, de számos tervezési módszer és leíró nyelv áll rendelkezésre, melyek a tervezési folyamat hatékonyságát hivatottak hatékonyabbá tenni [10][12][14][16].

2.2. Tárolt programú mikrogépek

A tárolt programú gépek a bonyolult, nagy számításigényű feladatokat egyszerű elemi műveletek sokaságának egymás utáni végrehajtásával oldják meg. Az összetett probléma megoldására szolgáló, általános értelemben vett lépések megfelelő sorrendű végrehajtását algoritmusnak, az algoritmus konkrét *mikrogép* (mikroprocesszor alapú számítógép) által végrehajtható gépi utasítások sorozataként való leírását programnak nevezzük.

A tárolt programú gépek szerkezetileg két részre oszthatók, az adatok közvetlen manipulálására képes funkcionális elemeket tartalmazó műveletvégző egységre (*datapath*), valamint az ebben található erőforrások megfelelő időzítésű vezérlő jeleinek előállításáért felelős vezérlő egységre (*control unit*) (*2-1. ábra*).



2-1. ábra - Mikroprocesszorok általános felépítése

A műveletvégző egység tartalmazza mindazokat az erőforrásokat, melyek a rendszer által végrehajtandó feladat elvégzéséhez feltétlenül szükségesek. Ilyen erőforrások pl. az aritmetikai áramkörök (összeadók, szorzók) és az adatok ideiglenes tárolására szolgáló regiszterek és memóriák. Míg egy bizonyos feladat végrehajtására tervezett célhardver esetén a szükséges erőforrásokat maga a feladat határozza meg, addig egy általános célú mikroprocesszornál a műveletvégző egységben található funkcionális elemek mennyisége és minősége az utasításkészlettől függ.

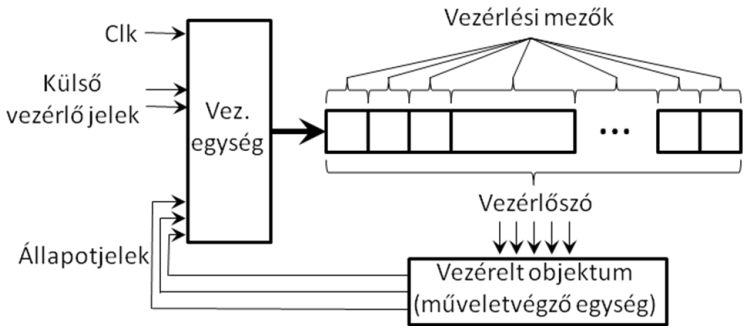
Bár a műveletvégző egység képes az adatok kezelésére, mégsem nevezhető önmagában mikroprocesszornak, hiszen már egyetlen assembler szintű utasítás végrehajtása is algoritmikus feladat, mely egy sor meghatározott sorrendű és időzítésű vezérlő jelet igényel, amit csak egy mikroprocesszor tud előállítani. Ilyen vezérlő jelek pl. a több különböző feladat elvégzésére alkalmas funkcionális egységek (pl. ALU) aktuális funkcióját kiválasztó jelek, vagy az adatok és erőforrások kiválasztását végző multiplexerek vezérlő jelei. Az ezen jeleket előállító vezérlő egységek megvalósítási lehetőségeivel foglalkozik a *2.2.1. pont.*

2.2.1. Mikroprocesszorok vezérlő egységének megvalósítási lehetőségei

A vezérlő egység feladata, hogy a mikrogép külső vezérlő bemenetei, valamint a műveletvégző egység által szolgáltatott állapotjelek alapján előállítsa a műveletvégző egység egyes funkcionális elemeit vezérlő jeleket. Ezek a vezérlő jelek egyaránt lehetnek egy- és

több bitesek. Az egy bites vezérlő jelek általában regiszterek, regisztertömbök *"load"* jelei, vagy bizonyos erőforrások háromállapotú kimenetének engedélyező jelei, míg a több bitesek főként multiplexerek kiválasztó jelei, vagy a többfunkciós erőforrások aktuális feladatát kijelölő bemenetek. Az egy- és több bites vezérlő jeleket összefoglaló néven *vezérlési mezőnek*, a mikrogép összes vezérlési mezőjét magába foglaló rendezett jelhalmazt pedig *vezérlőszónak* nevezzük. A műveletvégző egység vezérelt pontjait *rendszerpontnak* nevezzük.

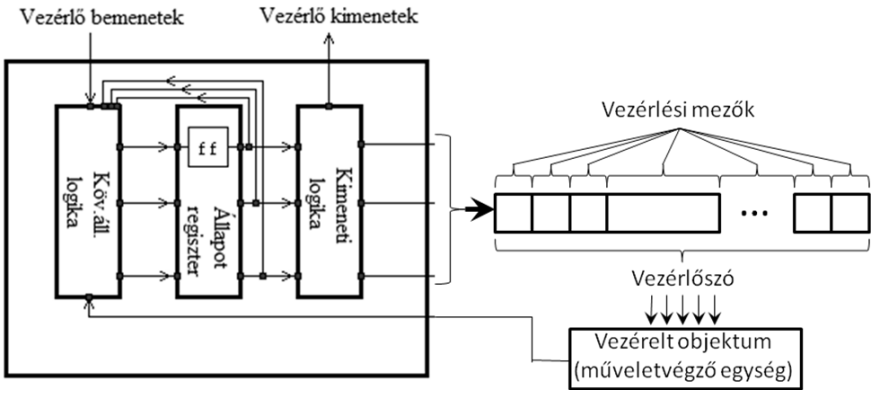
A vezérlő egység tehát olyan áramkör, mely minden órajelciklusban a külső és belső feltételeknek megfelelően újabb és újabb vezérlőszavakat állít elő (*2-2. ábra*).



2-2. ábra - Mikroprocesszorok két fő szerkezeti eleme és jeltípusai

Huzalozott vezérlő egységek

A vezérlő egység feladatára a legkézenfekvőbb megoldást a véges állapotgépek (FSM –*Finite State Machine*) jelentik. Az állapotgéppel megvalósított vezérlő egységet huzalozott vezérlőnek nevezik, utalva arra, hogy a funkció az eszköz legyártása után - a következő pontban tárgyalt mikroprogramozott vezérlőkkel ellentétben - nem változtatható meg. A véges állapotgépek szerkezetileg három részből állnak (*2-3. ábra*):

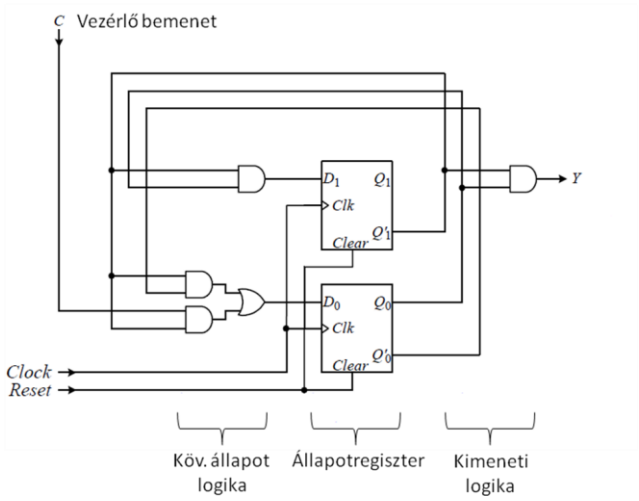


2-3. ábra - Huzalozott vezérlő egység szerkezete

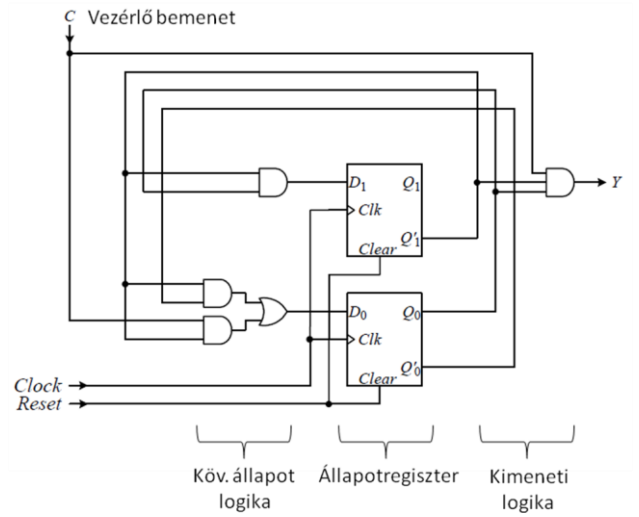
Állapotregiszter: Az állapotgép aktuális állapotát tároló regiszter. Megvalósítását tekintve egy egyszerű D-latch.

Következő állapot logika: Kombinációs hálózat, mely az állapotgép következő állapotát, az állapotregiszter következő értékét határozza meg az aktuális állapot, a külső vezérlő jelek és a műveletvégző által szolgáltatott állapotjelek alapján.

Kimeneti logika: Kombinációs hálózat, mely az aktuális állapot és esetlegesen egyéb külső vezérlő jelek alapján a műveletvégző egység vezérlését végző jeleket állítja elő. A kimeneti logika alapján az állapotgépeket két csoportba sorolhatjuk. Amennyiben a kimeneti logikai hálózat bemenetét kizárólag az állapotregiszter alkotja, úgy az állapotgép *Moore*, ha a bemenetek között egyéb külső vezérlő jelek is szerepelnek, akkor *Mealy* típusú (*2-4. ábra*) [3].



a.)



b.)

2-4. ábra - Moore (*a.*) és Mealy (*b.*) típusú állapotgép logikai szintű modellje

A *2-4. ábrán* látható kapuszintű modelleket általában - különösen bonyolult mikroprocesszorok tervezésekor - magasabb elvonatkoztatási szinten írjuk le, melyből a logikai szintézis szoftver automatikusan generálja a kapuszintű modellt. Egy tárolt programú gép vezérlő egységének szerepét betöltő állapotgép VHDL megvalósításának részlete látható a *2-5. ábrán*.

```
...
...
process(Clk,Reset)
  begin
    if (Reset = '1') then
      State <= WaitForRun;
      Exception <= '0'; IntrAck <= '0'; Read <= '0'; Write <= '0';
      ...
      ...
    elsif (Clk'event and Clk = '1') then
      case (State) is
        when WaitForRun => if (Run = '1') then State <= Load1;
                           else State <= WaitForRun; end if;
        when Load1      => State <= Load2;
        when Load2      => LD_IR <= '1';
                           State <= Decode;
        when Decode     => LD_IR <= '0';
                           if     (DivNULL = '1' or
                                   CondReg1 = '1' or
                                   CondReg2 = '1') then State <= Error; -- exception
                           elsif  (CondReg0 = '1' and
                                   IntrRqst = '1') then IntrAck <= '1';
                                                        IntrFromCont <= '1';
                                                        State <= s_intr1;
                           else
                             case (IR18_14) is
                               when i_mov  => LD_PC <= '1'; MX_GPAddr_A <= "00";
                                              MX_GPDataIn <= "0000";
                                              State <= s_mov1;
                               when i_movi => LD_PC <= '1'; MX_GPAddr_A <= "00";
                                              MX_GPDataIn <= "0001";
                                              State <= s_movi1;
```

2-5. ábra - Tárolt programú gép huzalozott vezérlő egységét leíró VHDL modell részlete

A *2-5. ábrán* a mikroprocesszor utasításdekódolást végző állapota (*"Decode"*) figyelhető meg, mely során a vezérlő egység a műveletvégző egység által szolgáltatott állapotjelek (pl. *"IR18_14"*, *"DivNULL"*, *"CondReg1"* stb.), valamint a külső vezérlő jelek (*"IntrRqst"*) alapján meghatározza a műveletvégzés folytatásának irányát, vagyis az állapotgép következő állapotát. Az egyes állapotokban a vezérlő jelek (pl. *"LD_PC"*, *MX_GPDataIn"* stb.) más-más értéket vesznek fel, így az állapotok megfelelő sorrendje egyben a vezérlő jelek meghatározott sorrendjét is jelenti. A vezérlő jelek ily módon előállított sorozata határozza meg a műveletvégző egység elemei közötti adatáramlást, megvalósítva a gépi utasítás algoritmusát.

Mikroprogramozott vezérlő egységek

Sok funkcionális egységet tartalmazó, bonyolult műveletvégző egység esetén (pl. CISC processzoroknál) a vezérlő- és állapotjelek száma meglehetősen nagy lehet. Ekkor a huzalozott vezérlő egység megtervezése nehézkessé válhat, továbbá problémát jelent az is, hogy az egyes gépi utasításokat megvalósító vezérlőszó-szekvenciák a gyártást követően nem változtathatók meg. Mindkét problémára megoldást kínálnak az úgynevezett mikroprogramozott architektúrák.

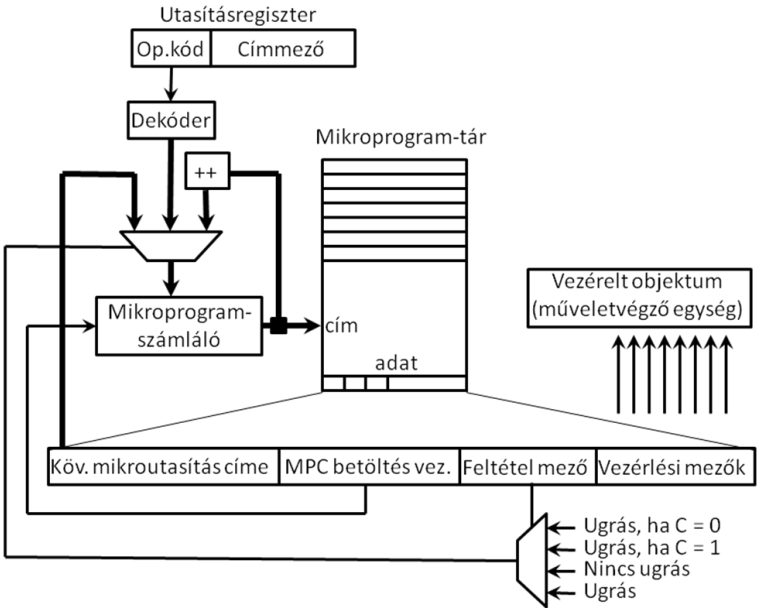
Egy mikrogép által megvalósított gépi utasítás tovább bontható még elemibb lépések sorozatára. Ezeket az elemi lépéseket *mikroutasításnak,* a mikroutasítás végrehajtásához szükséges vezérlési pontok beállítását *mikrooperációnak* nevezzük. Két regiszter összeadását végző gépi utasítás mikroutasítások sorozatával való leírása a következőképpen lehetséges:

1. "A" regiszter rákapcsolása az ALU "A" portjára
2. "B" regiszter rákapcsolása az ALU "B" portjára
3. ALU funkciójának kijelölése: összeadás
4. ALU kimenetének rákapcsolása a "C" regiszterre
5. Eredmény betöltése a "C" regiszterbe
6. Állapotregiszter értékének frissítése

Mikroprogramozott vezérlő egységek esetén a mikroutasításokat egy gyors hozzáférésű memóriában, a *mikroprogram-tárban* tároljuk. Egy gépi utasítás végrehajtása a következőképpen megy végbe:

1. Utasítás kiolvasása a programmemóriából: Utasításregiszter értékének frissítése.
2. Utasítás dekódolása: Az utasítást megvalósító mikroutasítás-sorozat első elemének a címét a mikroprogram-számlálóba töltjük, mely a mikroprogram-tár címbemenetére kapcsolódik. Ennek hatására a mikroprogram-tár megcímzett rekeszében lévő vezérlőszó a műveletvégző egység bemenetére kerül, ott kifejti hatását.
3. Mikroutasítás-szekvencia végrehajtása: A vezérlési mezők mellett a mikroprogram-tár minden szava tartalmaz egy cím- és egy feltételmezőt is. A feltételmező kiértékelése alapján a mikroprogram-számláló új értéket kap. Az új érték - a feltételvizsgálat kimenetelétől függően - egyaránt lehet a címmező értéke vagy külső címforrás. A mikroprogram-tár adatkimenetén megjelenik az új címen lévő mikroutasítás.

A mikroprogramozott vezérlő egység *Wilkes-féle* modellje a *2-6. ábra* szerint épül fel [4].



2-6. ábra - Mikroprogramozott vezérlő egység Wilkes-féle modellje

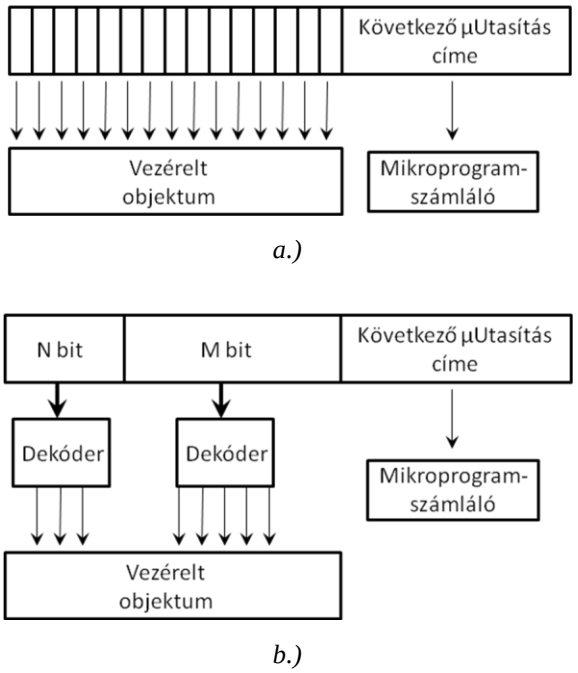
A megoldás előnye, hogy a vezérlő egység szerkezete kötött, a mikrogépre jellemző funkciót csak a mikroprogram-tár tartalma képviseli. A tervezés gyorsabb és egyszerűbb, az egyes gépi utasítások végrehajtásához szükséges mikroutasítás-sorozatok a gyártás után felülírhatók, elkészítésükhöz magas szintű szoftver eszközök állnak rendelkezésre (pl. mikro-assembler). További előny, hogy a vezérlő egység erőforrásigénye csak kevéssé függ az utasításkészlet bonyolultságától, hiszen a gépi utasítások és a vezérlési mezők száma egyaránt csak a mikroprogram-tár méretét befolyásolja.

A mikroutasítás felépítése alapján a mikroprogramozott vezérlőket két csoportba sorolhatjuk [17]. *Vertikális* mikroutasítás felépítés esetén a mikroutasításban csak egyetlen mikrooperáció van, a hozzá tartozó operanduskijelölésekkel és a következő mikroutasítás címét kiválasztó mezővel. *Horizontális* felépítés esetén a mikroutasítás számos, párhuzamos végrehajtást vezérlő mezőt tartalmaz. Ennek megfelelően a vertikális mikroutasítás szóhossza kisebb, a horizontálisé nagyobb. A horizontális mikroutasítások felépítésük miatt szorosabban kapcsolódnak a végrehajtó hardverelemekhez, ezért a segítségükkel megvalósított rendszert "kemény" (*hard*) vagy strukturális (*structural*) mikroprogramozásnak is nevezik. Ezzel szemben vertikális mikroutasítások esetén, ahol a hardver jobban eltávolodik a konkrét mikroutasításoktól, „puha” (*soft*) vagy funkcionális mikroprogramozásnak nevezik.

A mikroutasítás formátuma lehet állandó vagy változó. Az első esetben az adott bitek, bitcsoportok mindig azonos módon értelmezendők, míg változó formátum esetén az egyes

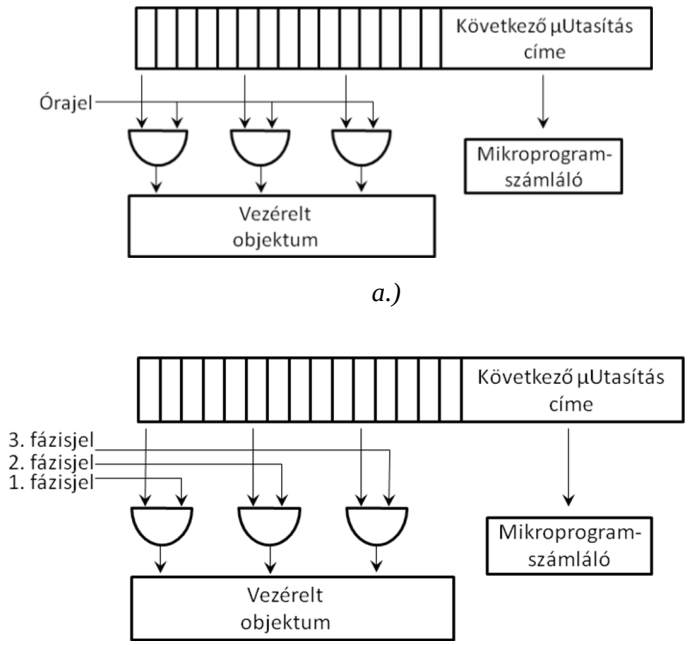
bitek (mezők) jelentése változik. Utóbbi esetben rövidebb mikroutasításokat kapunk, de kiegészítő logikai áramkörök és idő szükséges a formátumvezérlés külön utasítások segítségével való megvalósításához.

Ha a mikroutasítás minden egyes bitjét adott vezetékek vagy rendszerpontok vezérlésére külön használjuk, a mikroutasítás nem kódolt formájú. Ez ugyancsak nagyobb szóhosszat eredményez, de ugyanakkor nagyobb működési sebességet, nagyobb mértékű párhuzamosságot valósít meg. A másik megoldás az egymást kölcsönösen kizáró vezérlővezetékek, illetve állapotok (pl. ALU művelet, egy-egy regisztertömbből való regiszterkijelölés) egy kódolt mikroutasítás mezővel való vezérlése. Ily módon *n* bites mezővel 2^n vezérlési állapot állítható elő, amelyet azután dekódolni kell. Az ilyen mikroutasítást kódolt formájúnak nevezik (*2-7. ábra*).



2-7. ábra - Nem kódolt (*a.*) és kódolt (*b.*) formájú mikroutasítás

A mikroutasítás időzítése egy- vagy többfázisú lehet. A többfázisú időzítés lehet szinkron, amikor a fázisok száma állandó, vagy aszinkron, amikor ez a szám változik (*2-8. ábra*).



2-8. ábra - Egyfázisú (*a.*) és többfázisú (*b.*) vezérlés

2.3.Alkalmazásközpontú mikroprocesszorok tervezése

Az irodalomban számos keretrendszer fellelhető alkalmazásorientált mikroprocesszorok (ASIP - *Application-specific Instruction-set Processor*) fejlesztéséhez. Ezek a keretrendszerek hardver- és szoftvereszközökből állnak, és általában két részre bonthatók: A tervezési fázisban magas elvonatkoztatási szintű modellekkel dolgozunk, és olyan szoftver eszközökkel, melyek a tervezett hardver és a rajta futtatandó szoftver összhangjáról szolgáltatnak információt (assembler fordítók, compilerek, utasításkészlet-szimulátorok stb.). Ebben a fázisban történik az utasításkészlet optimalizációja, melynek eredménye maga az utasításkészlet és az egyes utasítások kihasználtságára, sebességére vonatkozó információk. Ha az utasításkészlet eleget tesz a követelményeknek, akkor kezdetét veszi az ún. implementációs, vagy megvalósítási fázis, mely során egy az utasításkészletet implementáló tárolt programú mikrogép RTL szintű hardver leíró nyelvű modelljét állítjuk elő [9][12][13][14][15].

Alkalmazásorientált utasításkészletek definiálására két fő megközelítés létezik. Az egyik az utasításkészlet egyes utasításait egy nagy, sok általános célú utasítást tartalmazó utasításhalmazból és egy konkrét szoftver alkalmazásból származtatja. A megvalósított utasításkészlet az általános célú utasítások halmazából csak azokat valósítja meg, melyeket a konkrét szoftver alkalmazás használ. Ilyen értelemben tehát nem maguk az egyes utasítások, hanem az utasításkészlet egésze lesz alkalmazásorientált [7][8]. A másik megközelítés

teljesen egyedi gépi utasításokat vezet be, melyek egy speciális problémakörben (DSP, függvénytan, számelmélet stb.) gyakran előforduló feladatok hatékony megoldását teszik lehetővé [6]. Munkám során ez utóbbi megoldást választottam. Az általam kifejlesztett alkalmazásközpontú gépekben az egyes speciális utasításokat egymástól független, „belső társprocesszorok" hajtják végre, melyek pontos architektúrája az azokat magában foglaló tárolt programú géppel azonos környezetben, egyazon leíró nyelv, az ARTL segítségével tervezhető meg. Az ARTL egy architektúra leíró nyelv (*Architecture Description Language* - **ADL**), akárcsak az nML [10], az ISP [11] vagy a LISA [12] azzal a különbséggel, hogy általános jellegénél fogva nemcsak tárolt programú gépek, hanem speciális célú funkcionális egységek viselkedésének leírására is alkalmas. A nyelv e tulajdonságát azon speciális funkcionális egységek tervezésekor használhatjuk ki, melyek tárolt programú gépünket alkalmazásorientálttá teszik.

3. FELHASZNÁLT ESZKÖZÖK

Munkám során elsősorban VHDL-alapú rendszermodellezéssel és C++ szoftverfejlesztéssel foglalkoztam. Ennek megfelelően a két legfontosabb felhasznált szoftver eszköz a *Mentor Graphics ModelSim PE Student Edition 6.0b* és a *Microsoft Visual C++ 2010 Express* volt. A *4.4.1. pontban* bemutatott SystemC modelltípus kifejlesztése során a *SystemC osztálykönyvtár 2.2.0. verzióját* használtam [19].

Az *5. pontban* bemutatott, a tervezési módszer kifejlesztése során meghatározó szerepet játszó mikroprocesszoros rendszereket a *Xilinx ISE Webpack (ISE Design Suite 12.2)* környezetben az *XST* logikai szintézis szoftverrel *Xilinx Spartan3E* típusú FPGA-ra szintetizáltam, működésüket *Digilent Nexys2* fejlesztőkártyán teszteltem [18].

A *4.4.1. pontban* bemutatott statisztikai analízis eredményeit *MATLAB (vR2008b)* környezetben vizsgáltam.

Az *5.1. pontban* tárgyalt aritmetikai társprocesszor kifejlesztése, a Taylor-polinomos függvényközelítés implementációja során a tartományhatárok pontos számításához és a hibafüggvények vizsgálatához a *Maplesoft MAPLE 13.0* szoftverét használtam fel.

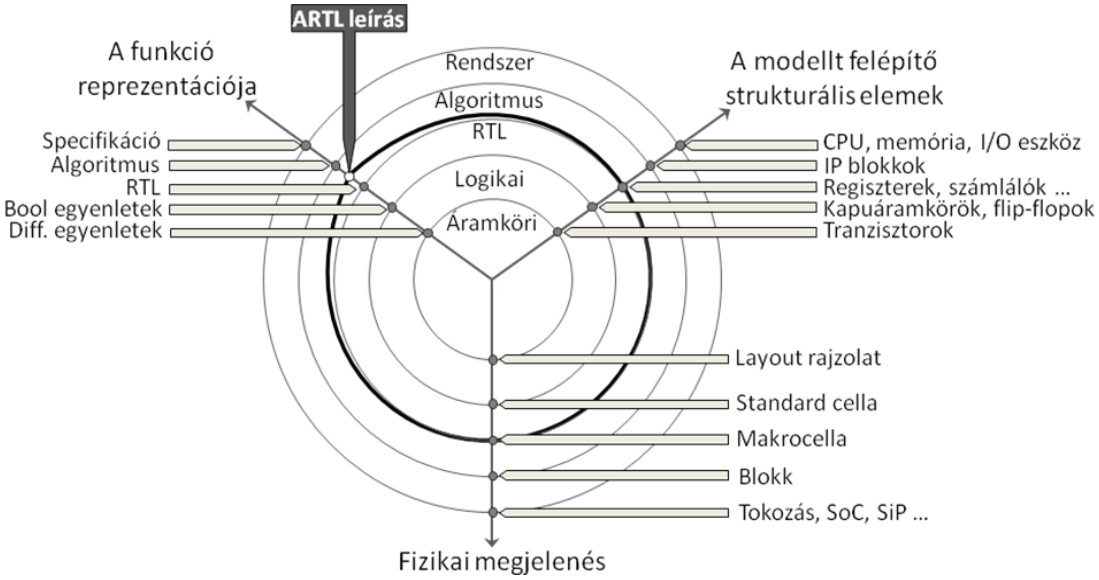
4. ÚJ EREDMÉNYEK

Az alábbi fejezetben bemutatásra kerül az általam kidolgozott kevert elvonatkoztatási szintű modellező nyelv (*Algorithmic Register Transfer Language* - **ARTL**), amellyel algoritmikus szinten lehet leírni a megtervezendő digitális rendszer elvárt funkcióit. Az ARTL nyelven így létrehozott rendszermodellnek az a különlegessége, hogy a benne szereplő nyelvi szerkezetek implicit módon tartalmazzák az algoritmikus szintű funkciókat megvalósító regiszter-átviteli szintű modellt is. Ezt úgy értem el, hogy az ARTL nyelvi elemek – szemben a viselkedési modellezés hagyományos nyelvi eszközeivel – regiszter-átviteli szintű információkat is hordoznak. Az ARTL nyelv ezen jellegzetességei részletesen ismertetésre kerülnek a *4.1. alfejezetben*. Ezt követően egy új mikroprocesszor típust (*folyamatvezérelt mikroprocesszor*) mutatok be, amelynek viselkedése az ARTL nyelv segítségével könnyen kezelhető algoritmikus formában leírható.

4.1.Az ARTL leíró nyelv

Az ARTL leíró nyelv az adatfeldolgozást végző digitális rendszerek viselkedésének magas szintű leírását teszi lehetővé regiszter-átviteli szintű összetevők közötti adatáramlás algoritmikus jellegű leírásával. Az algoritmikus jellegből adódóan a funkcionalitás áttekinthető formában, gyorsan és hatékonyan implementálható (HLS), ugyanakkor a leírásban hivatkozott erőforrások - a műveletvégző egység összetevői - függetlenül tervezhetők. Ez főként olyan nagy bonyolultságú rendszerek tervezése esetén előnyös, melyekben a műveletvégző egység egyes összetevői önmagukban is összetettek (pl. *Application-Specific Instruction-set Processor* - **ASIP**, alkalmazásorientált utasításkészletű processzor). A már kifejlesztett részáramkörök újra felhasználhatók, bonyolultabb összetevők esetén a beágyazó környezettel kapcsolatot tartó interfész akár szabványosítható is, így a vezérlő egység ide vonatkozó részletei is újra felhasználhatóvá válnak. Mivel az erőforrás menedzsment teljes egészében a tervező kezében van, a leírás rugalmasabb optimalizációs lehetőségeket biztosít, az eredményül kapott áramkör paraméterei pedig kevésbé függnek a logikai szintézist végző szoftvertől.

E tulajdonságokat figyelembe véve megállapítható, hogy az ARTL modell a funkciót reprezentáló leírás jellege alapján az algoritmus és az RTL szintű modell között helyezkedik el, ugyanakkor a modellt felépítő strukturális elemeket, illetve azok fizikai megjelenését tekintve a leírás tisztán RTL jellegű (*4-1. ábra*) [5].



4-1. ábra - A különböző elvonatkoztatási szintű reprezentációkat ábrázoló Gajski-Kuhn diagram

4.1.1. Erőforrástípusok, jeltípusok

Az ARTL leírás 13 típust használ (*4-1. táblázat*), melyekből 8 a műveletvégző egység egyes erőforrásait jelöli, egy a karakterláncok (álnevek) számkonstansokhoz való hozzárendelését segíti elő (utasítás mnemonik - operációs kód hozzárendelés), további négy pedig a vezérlő és állapotjelek megkülönböztetésére szolgál.

Erőforrástípus	Leírás
Port	Kapcsolódási pont a külvilág felé. Adatot nem tárol, így a kimeneti portot mindig egy regiszter kimenetéhez rendeljük.
Regiszter	Egyszerű D-latch. Szinkron írás, aszinkron kiolvasás
Regisztertömb	Single- és dual-port regisztertömb Szinkron írás és olvasás Külön adat be- és kimenet
Aszinkron operátor	Egyszerű aritmetikai, komparálási stb. feladatok elvégzésére alkalmas funkcionális elem. Órajelet nem igényel, az eredményt szimulációs környezetben[1] azonnal szolgáltatja. Pl.: 32 bites aszinkron fixpontos osztó
Szinkron operátor	Tipikusan többfunkciós aritmetikai egység. Az eredményt egy vagy több, operandusoktól független számú periódus késleltetéssel szolgáltatja. Pl: 32 bites fixpontos ALU
Hand-shake operátor	Több órajelciklust igénylő, bonyolult számítási feladatok elvégzésére alkalmas funkcionális egység.

[1] Az állítás csak az ún. „zero delay” szimulációra igaz, amikor az egyes funkcionális egységek késleltetését nullának tekintjük.

	Késleltetése függhet a bemenő operandusoktól. Pl: Négyzetgyök számítása Newton-iterációval.
Konstans	Álnév hozzárendelése számkonstanshoz
cnto	*Control output*: vezérlő kimenet *(2-1. ábra)*
cnti	*Control input*: vezérlő bemenet *(2-1. ábra)*
cs	*Control signal*: belső vezérlő jel *(2-1. ábra)*
ss	*Status signal*: állapotjel *(2-1. ábra)*

4-1. táblázat - ARTL erőforrás típusok

4.1.2. Erőforrások bejelentése

Az ARTL leírás a műveletvégző egységben megtalálható részáramkörök bejelentésével kezdődik (*4-2. táblázat*). A bejelentés azokat az információkat tartalmazza, melyek az adott erőforrás rendszerbe illesztésének egyértelműségét biztosítják (buszok szélessége, operátorok be- és kimenetei).

Típus	Bejelentés
Bemeneti port	**resource** <*név*>: **iport** [<*méret*>]
Kimeneti port	**resource** <*név*>: **oport** [<*méret*>]
Regiszter	**resource** <*név*>: **reg** [<*méret*>]
Single-port regisztertömb	**resource** <*név*>: **sprf** [<*címméret*>][<*adatméret*>]
Dual-port regisztertömb	**resource** <*név*>: **dprf** [<*címméret*>][<*adatméret*>]
Aszinkron operátor	**resource** <*név*>: **ao** (<*1.bemenetnév*>[<*méret*>],...,<*n.bemenetnév*>[<*méret*>]) (<*1.kimenetnév*>[<*méret*>],...,<*k.kimenetnév*>[<*méret*>])
Szinkron operátor	**resource** <*név*>: **so** (<*1.bemenetnév*>[<*méret*>],...,<*n.bemenetnév*>[<*méret*>]) (<*1.kimenetnév*>[<*méret*>],...,<*k.kimenetnév*>[<*méret*>])
Hand-shake operátor	**resource** <*név*>: **hso** (<*1.bemenetnév*>[<*méret*>],...,<*n.bemenetnév*>[<*méret*>]) (<*1.kimenetnév*>[<*méret*>],...,<*k.kimenetnév*>[<*méret*>])
Konstans	**alias** <*név*>: <*érték*>

4-2. táblázat - Erőforrás-bejelentés az ARTL nyelvben

4.1.3. Értékadások

Hasonlóan a programozási és hardver leíró nyelvekhez, az értékadások az ARTL leírásban is két részből; egy balértékből és egy jobbértékből állnak. Az értékadás értelmezésekor a jobbérték kiértékelése során kapott eredmény (visszatérési érték) lesz a bal oldal új értéke. Az értékadások során a balérték szerepét a *4-3. táblázatban* összefoglalt kifejezések tölthetik be:

Hivatkozás	Szintaktika
Regiszter	<*név*>[<*bitindex*>], ahol <*bitindex*>: [<*felső*>:<*alsó*>] Pl.: IR[7:0]
Regisztertömb eleme	<*név*>[<*regindex*>][<*bitindex*>], ahol <*regindex*>: tetszőleges jobbérték-kifejezés Pl.: GP[IR[13:11]][7:0]
Kimeneti port	<*név*>[<*bitindex*>] Pl.: Addr2DataMem[7:0]
Vezérlő kimenet	cnto <*név*> Pl.: cnto Read
Belső vezérlő jel	cs <*név*> Pl.: cs SQRT_Rqst

4-3. táblázat - Értékadások lehetséges balértékei

A lehetséges jobbérték kifejezéseket a *4-4. táblázat* foglalja össze.

Hivatkozás	Szintaktika
Regiszter	<*név*>[<*bitindex*>], ahol <*bitindex*>: [<*felső*>:<*alsó*>] Pl.: IR[7:0]
Regisztertömb eleme	<*név*>[<*regindex*>][<*bitindex*>], ahol <*regindex*>: tetszőleges jobbérték-kifejezés Pl.: GP[IR[13:11]][7:0]
Bemeneti port	<*név*>[<*bitindex*>] Pl.: IntrAddress[7:0]
Operátor kimenete	<*operátornév*>.<*kimenetnév*>[<*bitindex*>] (<*operátor bemenetei*>)[<*bitindex*>], ahol <*operátor bemenetei*>: tetszőleges jobbérték-kifejezés Pl.: ALU.Result[3:0](GP[IR[10:8]],GP[IR[7:5]])

4-4. táblázat - Értékadások lehetséges jobbértékei

Az egyszerű regiszter ("**reg**"), a single-port regisztertömb ("**sprf**") és a dual-port regisztertömb ("**dprf**") VHDL modellje adott, ezért az ARTL leírásban az ezekhez tartozó vezérlő jelek kezelése nem szükséges.

4.1.4. Vezérlési szerkezetek

Az ARTL nyelv algoritmikus jellegét a strukturált programozás vezérlési szerkezetei, az elágazás és a ciklus biztosítja. Az elágazásoknak kétféle típusa van, melyek vezérlése állapotjelekkel és külső vezérlő jelekkel történhet (*4-2. ábra*).

```
if (<condition 1.>) => <instructions>;
elsif (<condition 2.>) => <instructions>;
...
elsif (<condition n.>) => <instructions>;
else => <instructions>;
end if;

condition i.: ss <status signal ID>[<bitindex>] = <value>
              cnti <control input ID>[<bitindex>] = <value>

$ ------------------------------------------------------

case (ss <status signal ID> / cnti <control input ID>)
  value 1. => <instructions>;
  value 2. => <instructions>;
  ...
  value n. => <instructions>;
end case;
```

4-2. ábra - Elágazás az ARTL nyelvben

Az elágazások kiértékelésének kimenete a műveletvégző és a vezérlő egység közötti állapotjelek, valamint a vezérlő egység és a külvilág közötti külső vezérlő jelek listája. Az ARTL nyelv csak egyféle ciklust tartalmaz, melyből a kilépés lehetőségét az elágazás és a "**break**" utasítás biztosítja (*4-3. ábra*).

```
loop
  <body>;
  if (<condition>) => break;
end loop;
```

4-3. ábra - Ciklus az ARTL nyelvben

Az ARTL leírás felépítését mutatja a *4-4. ábra*, egy egyszerű szorzó áramkör teljes ARTL leírását pedig a *4-5. ábra*.

4-4. ábra - ARTL leírás szerkezete

```
$ ARTL description of a simple 4-bit multiplier ----------------------
$ Resource declarations ----------------------------------------------
resource p_Multiplicand:  iport [4];
resource p_Multiplier:    iport [4];
resource Multiplicand:    reg   [4];
resource Multiplier:      reg   [4];
resource Tmp:             reg   [8];
resource Result:          oport [8];
resource Concatenate:     ao    (Input1[4],Input2[4])(Output[8]);
resource Add:             ao    (Input1[8],Input2[8])(Output[8]);
resource Sub:             ao    (Input1[4],Input2[4])(Output[4]);
$ Alias definitions --------------------------------------------------
alias Zero_8:             "00000000";
alias Zero_4:             "0000";
alias One_4:              "0001";
$ Behavioral description ---------------------------------------------
  Multiplicand <= p_Multiplicand;
  Multiplier <= p_Multiplier;
  Tmp <= @Zero_8;
  if (cnti Rqst = '1') =>
    cnto Rdy <= '0';
    loop
      Tmp <= Add.Output(Tmp,Concatenate.Output(@Zero_4,Multiplicand));
      Multiplier <= Sub.Output(Multiplier,@One_4);
      if (ss Multiplier[3:0] = @Zero_4) => break;
    end loop;
  end if;
  cnto Rdy <= '1';
$ Output port assignments --------------------------------------------
Result <= Tmp;
$ --------------------------------------------------------------------
```

4-5. ábra - Egyszerű szorzó áramkör ARTL modellje

A *4-5. ábrán* jól megfigyelhető, miként jelennek meg az ARTL modellben szerkezeti és algoritmus szintű leírásokra jellemző elemek egyaránt. A viselkedés leírása egy algoritmus, melynek lépései ebben az esetben olyan értékadások, melyek bal- és jobbértéke egyaránt valamilyen RTL szintű erőforrásra, adattároló elemre vagy operátorra hivatkozik.

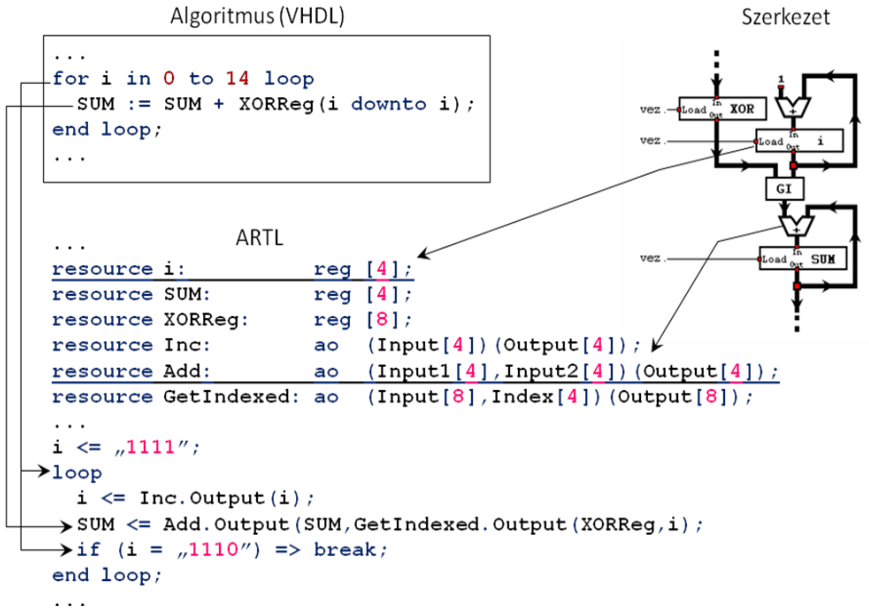
A *4-5. táblázat* azokat a nyelvi eszközöket tartalmazza, melyek az algoritmus és a szerkezeti leírás tipikus elemei. A táblázat jobb oldali oszlopában látható, hogy az adott nyelvi eszköz milyen formában található meg az ARTL leírásban.

Leírás jellege	Nyelvi eszköz	Nyelvi eszköz megjelenése az ARTL-ben
Algoritmus	Elágazás	If - else szerkezet.
	Ciklus	Egyszerű "loop" ciklus. A kilépés if-else szerkezet alkalmazásával oldható meg.
	Változókon végzett műveletek	Adattároló elemeken végzett operációk. Az operációkat az "ao" (aszinkron operátor) és az "so" (szinkron operátor) erőforrások valósítják meg.
	Szubrutin	A "hso" (hand-shake operátor) típus valósítja meg.

Szerkezeti leírás	Egyedbejelentés (deklaráció)	Erőforrások bejelentése
	Egyedbeültetés (példányosítás)	Erőforrás előfordulása egy kifejezés jobb- vagy balértékeként.
	Összetevők közötti kapcsolatok, összeköttetések	Operátorok paraméterezésén, regisztertömbök címzésén és értékadásokon keresztül valósul meg.

4-5. táblázat - Algoritmus és szerkezeti leírás jellemzői az ARTL modellben

Az ARTL leírásra jellemző algoritmus-struktúra kettősséget a *4-6. ábra* szemlélteti. A VHDL nyelvű kódrészlet által definiált funkciót a jobb oldali szerkezeti modell valósítja meg. Látható, hogy az ARTL modellben az algoritmus jellegzetes elemei (ciklus) és a strukturális modell komponensei (regiszterek, operátorok) egyaránt jelen vannak.



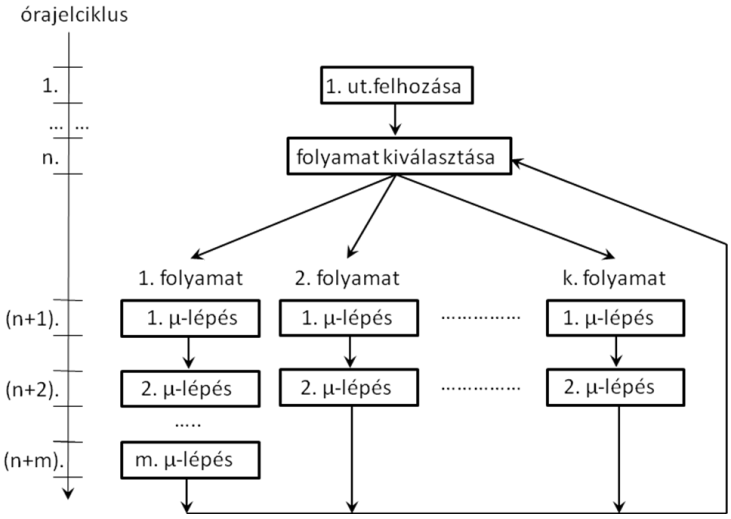
4-6. ábra - ARTL leírás algoritmus-struktúra kettőssége

4.2.A folyamatvezérelt mikroprocesszor modell

Az alkalmazásorientált mikroprocesszorok sajátossága, hogy utasításkészletük az általános célú adatfeldolgozáshoz szükséges instrukciók mellett olyan utasításokat is tartalmaz, melyek egy bizonyos feladat elvégzésére különösen alkalmassá teszik őket. Ezek a processzorok általában valamilyen beágyazott rendszer részét képezik, ahol a futó szoftver csak ritkán és kis mértékben változik (verziófrissítés). Az általam kifejlesztett mikroprocesszor architektúra a benne található hardver elemek és a vezérlő egység felépítését tekintve ugyan viszonylag kötött, azonban szinte bármilyen utasításkészlet implementációjára alkalmas, beleértve a több bájtos és az elágazásokat is tartalmazó gépi utasításokat (*8.1. pont*).

A műveletvégzés magyarázatához szükséges egy "*folyamat*" fogalom bevezetése: A folyamat a vezérlő egység által előállított *vezérlőszavak* (lásd *2.2.1. pont*) sorozata, amely a műveletvégző egységben elemi műveletek sorozatát idézi elő. Az egyes folyamatok végrehajtása egy feltételhez köthető. Az itt tárgyalt folyamatfogalom hasonló a VHDL nyelv *process* fogalmához, amelyben szintén egy feltételhez (érzékenységi listában felsorolt jelek valamelyikének megváltozása) kötött utasításszekvenciát írunk le.

A műveletvégzést a *4-7. ábra* szemlélteti.



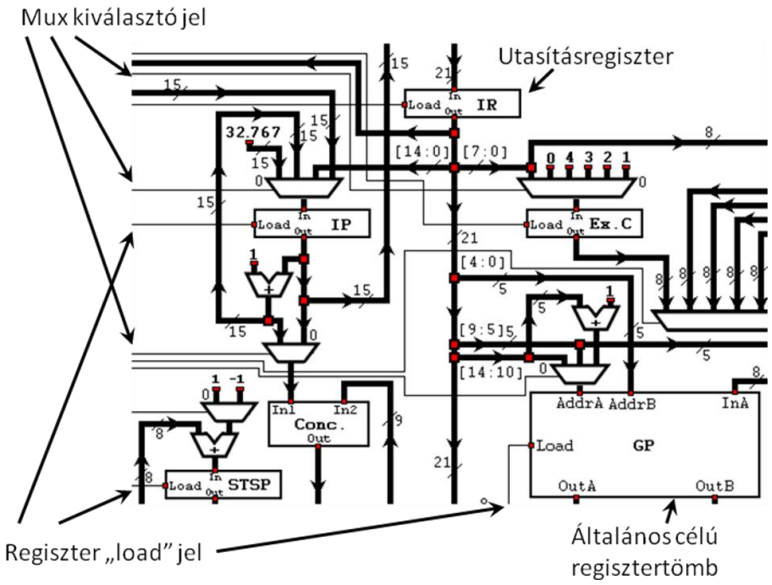
4-7. ábra - A műveletvégzés lépései a folyamatvezérelt processzorokban

A folyamat fogalom egyaránt magában foglalja a gépi utasításokat, amelyek végrehajtását a tárolt program írja elő, és a kivételeket, melyek valamely külső esemény, hiba hatására hajtódnak végre. A *4-7. ábrán* megfigyelhető, hogy egy-egy folyamat végén a végrehajtás nem a következő utasítás felhozásával, hanem annak dekódolásával folytatódik. Az egyes folyamatok tehát saját funkciójukon túlmenően a soron következő utasítás beolvasását is elvégzik. Ez kétszintű utasítás átlapolást jelent, melyben a végrehajtás (*execute*) és az utasításfelhozás (*fetch*) fázisok egymással párhuzamosan mennek végbe.

4.2.1. Mikroarchitektúra

Műveletvégző egység

Az folyamatvezérelt processzorok műveletvégző egysége viszonylag egyszerű, a kétszintű utasításátlapolás nem teszi szükségessé az erőforrások többszörözését. Az előforduló erőforrások az ARTL nyelv típusainak felelnek meg. Egy ilyen műveletvégző egység részlete látható a *4-8. ábrán.*



4-8. ábra - Egy folyamatvezérelt processzor műveletvégző egységének részlete

Látható, hogy a műveletvégző egységet vezérlő jelek többsége regiszterekhez és regisztertömbökhöz tartozó *"load"* jel, vagy adatutakat kijelölő multiplexerek kiválasztó jele. A *2.2.1. pontban* leírtaknak megfelelően a vezérlő jelek között általános esetben szerepelhetnek háromállapotú buszokra kapcsolódó erőforrások kimenetét engedélyező jelek. Az folyamatvezérelt processzorok esetén ezek a jeltípusok nem jelennek meg. Ez a megvalósítás alapjául szolgáló technológia jellegzetessége; FPGA-ban megvalósított architektúrák nem alkalmaznak háromállapotú buszokat.

A céltechnológia jellegzetességei az ARTL nyelvben rendelkezésre álló erőforrások VHDL modelljében is fellelhetők. Ezek a jellegzetességek főként a memória jellegű elemeket, a single-port és a dual-port regisztertömböket érintik, melyek előre definiált VHDL modellje a következő sajátosságokkal rendelkezik:

- A szintézer szoftver konfigurációjától függően implementálhatók blokk RAM-ként, LUT-okból felépülő ún. elosztott (*distributed*) RAM-ként és kis módosítással akár flip-flopokból álló regiszterek halmazaként, melyek közül a megcímzett szót egy multiplexer választja ki. Ez utóbbi megoldás előnye, hogy kimenete aszinkron működésűvé alakítható, mely a szinkron megoldáshoz képest lényeges sebességnövekedést (ciklusszám-csökkenést) eredményezhet, ugyanakkor erőforrásigénye a memóriamodul kapacitásának növekedésével erőteljesen nő, így alkalmazása csak nagyon kis méretű regisztertömbök esetén lehet indokolt.
- A VHDL modelljükön végzett minimális változtatással beiktatható a szimulációs környezetben esetleg szükséges szinkron vagy aszinkron *reset* jel, mely

segítségével a tárolókhoz kezdeti értékeket rendelhetünk. *Reset* jel használata esetén a blokk- és LUT RAM-ként való szintézis már nem lehetséges. A memória áramköröknek általában nincs *reset* bemenetük.

- Kimenetük szinkron működésű, melyre akkor van szükség, ha szintetizálásuk során blokk RAM modulokat szeretnénk használni.
- Független bemeneti és kimeneti adatbusszal rendelkeznek, ami szintén a blokk RAM modulokra jellemző.

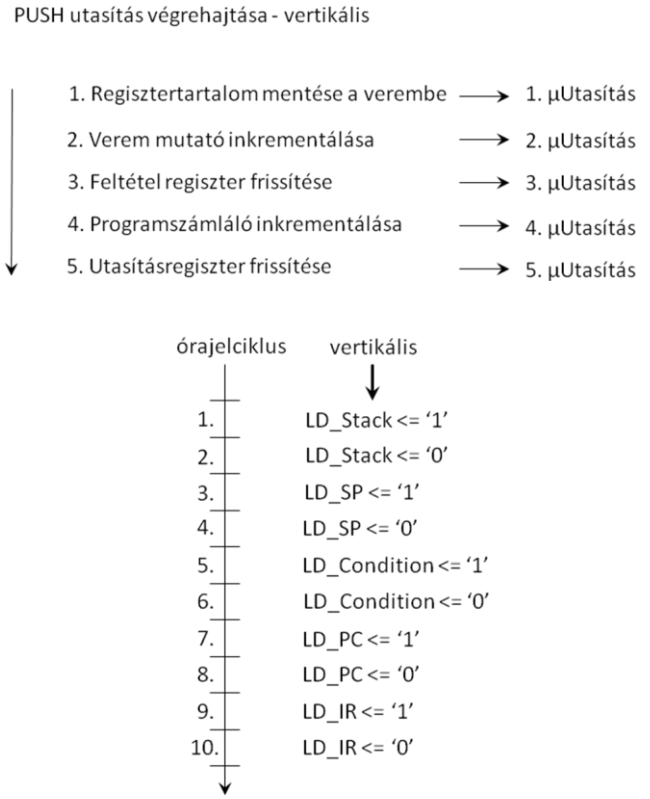
Mikroprogramozott vezérlő egység

A vezérlő jelek előállítása a folyamatvezérelt processzoroknál véges állapotgéppel vagy mikroprogramozással történhet. Az itt alkalmazott, a *2.2.1. pontban* leírtaktól lényegesen eltérő vezérlési rendszer bizonyos sajátosságai a huzalozott és a mikroprogramozott vezérlőkre is jellemzők. A két megközelítés közötti lényeges eltérés csupán az, hogy míg huzalozott esetben a vezérlőszavakat a vezérlő állapotgép kimeneti logikai hálózata generálja, addig mikroprogramozott esetben ezeket egy gyors hozzáférésű memóriában tároljuk. A vezérlési rendszert a mikroprogramozott vezérlőn keresztül ismertetem, majd bemutatom, hogy az ott megismert fogalmak miként jelennek meg a huzalozott vezérlő egységben.

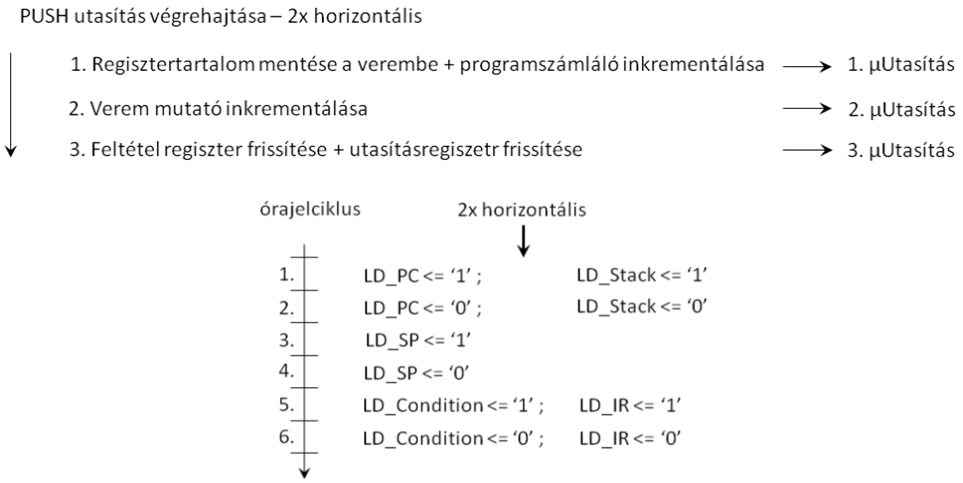
Általános esetben egy mikroprogramozott architektúra megtervezése a gépi utasítások mikroutasításokra való felbontásával kezdődik. Első lépésként tehát definiálnunk kell azokat a mikrooperációkat, melyeket a műveletvégző egység képes végrehajtani. Vertikális mikroutasítás esetén a feladat egyszerű, a mikroutasítás-készlet az egyes erőforrásokhoz kapcsolódó elemi lépések listája. Ez nagymértékben megkönnyíti és meggyorsítja a tervezést, ugyanakkor a műveletvégzés sebessége nagyon alacsony lesz. Horizontális mikroutasítás esetén bizonyos erőforrásokat egymással párhuzamosan vezérlünk, így a műveletvégzési sebesség és a vezérlőszavak hossza egyaránt megnő.

A folyamatvezérelt processzorok megtervezése során nem definiálunk konkrét erőforráshoz vagy erőforráscsoporthoz tartozó mikroutasításokat (pl. PC++ mikroutasítás), mert az egyes erőforrások vezérlése különböző folyamatok esetén más és más egyéb erőforrásokkal együtt, vagy legalábbis azokkal átlapolódva történik. Mikroutasítások definiálása helyett egy-egy folyamat vezérlőszavait közvetlenül konfiguráljuk. Minden egyes vezérlőszó egyedi, folyamatra jellemző lesz. Így megvalósítható a műveletvégző egység által lehetővé tett legnagyobb fokú párhuzamosítás, ugyanakkor a tervezési idő is lerövidül, hiszen a mikroutasítások definiálásának lépését kihagyjuk. A folyamatokhoz tartozó egyedi

vezérlőszavak megtervezéséhez kifejlesztett szoftver eszközt a *4.3.2. pontban* ismertetem. Ez a módszer tulajdonképpen a horizontális mikroprogramozás szélsőséges esete olyan értelemben, hogy itt akár minden erőforrást vezérelhetünk egyszerre minden órajelciklusban. Egy gépi utasítás (folyamat) előre definiált mikroutasítások sorozataként való leírását (tehát a klasszikus módszert) mutatja a *4-9. ábra* vertikális, a *4-10. ábra* pedig kétszeresen párhuzamosított - két erőforrás egy időben való vezérlésével megvalósított - horizontális mikroutasítás felépítés esetén. Az ábrák az egyes mikroutasításokat alkotó mikrooperációk közül csak azokat tartalmazzák, melyek az adattároló elemek megváltozására vonatkoznak. Az adatutak kijelölését végző multiplexerek kiválasztó jelei aszinkron működésűek, nem igényelnek külön órajelciklust.

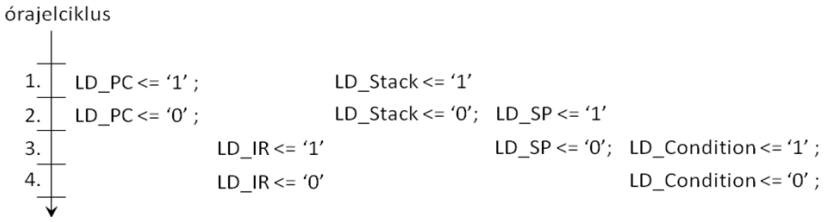


4-9. ábra - PUSH utasítás végrehajtása vertikális mikroutasítás felépítéssel



4-10. ábra - PUSH utasítás végrehajtása horizontális mikroutasítás felépítéssel

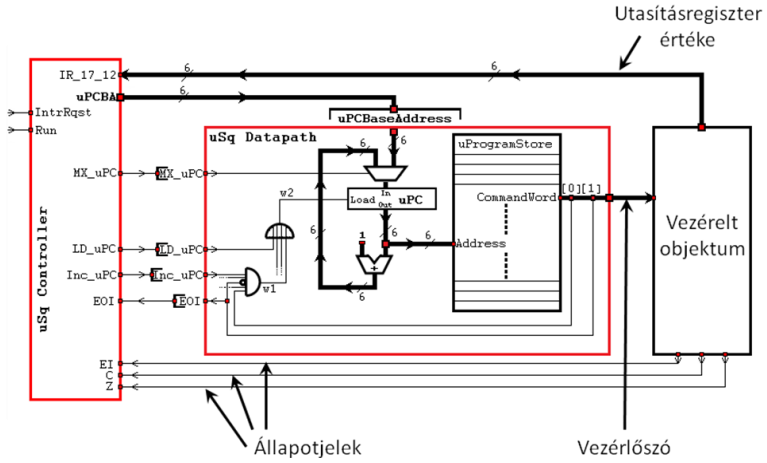
Nyilvánvaló, hogy a művelet elvégzéséhez szükséges idő az erőforrások párhuzamos vezérlésével drasztikusan lecsökkenhet. A *4-11. ábrán* megfigyelhető, hogy az erőforrások vezérlésének lehető legnagyobb fokú párhuzamosítása és átlapolása esetén miként alakul az utasítás végrehajtásához szükséges ciklusok száma.



4-11. ábra - PUSH utasítás végrehajtása a mikrooperációk legnagyobb fokú párhuzamosításával

Mivel a folyamatvezérelt processzorok esetén az egyes vezérlőszó-szekvenciák folyamatonként egyedileg vannak definiálva, ezért a mikroprogram-tárban minden folyamat külön-külön szerepel. Ebből következik, hogy a mikroprogram-tár redundáns információt tartalmazhat, ugyanakkor a tárolandó szó hossza viszonylag rövid, mivel kizárólag vezérlési mezőt tartalmaz, a Wilkes-féle modell mikroutasításaival ellentétben nem tartalmaz ugrási címet és feltételmezőt sem. A vezérlőszó címét szolgáltató mikroprogram-számláló (*µPC*) regiszter értékét két módon írhatjuk felül: Folyamat végrehajtása közben a µPC inkrementálódik, új folyamat kiválasztásakor pedig - a gépi utasítás dekódolásakor - a következő vezérlőszó-sorozat kezdőcímét egy állapotgép (az ún. *"microsequencer"*) kapuzza a µPC regiszter bemenetére. Látható, hogy a mikroprogramozott vezérlő esetén sem kerülhetjük el egy állapotgép beépítését, ennek összetettsége (és erőforrásigénye) azonban

lényegesen kisebb egy komplett huzalozott vezérlőénél. A folyamatvezérelt processzorok mikroprogramozott vezérlő egységének RTL szintű kapcsolási rajza a *4-12. ábrán* látható.



4-12. ábra - Folyamatvezérelt processzorok mikroprogramozott vezérlő egységének RTL szintű modellje

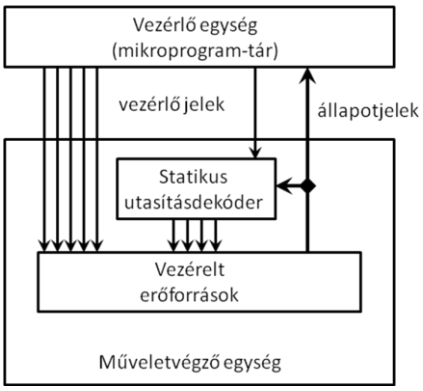
Bizonyos esetekben szükség lehet a mikroprogram futásának felfüggesztésére. A futás várakoztatása külső memóriához való hozzáférés, vagy "hso"-t (hand-shake operátor) használó művelet eredményére való várakozás során elengedhetetlen, továbbá új gépi utasítás dekódolásakor a mikroprogram-tárbeli kezdőcím megállapítása is időt vesz igénybe, mely alatt a µPC inkrementálását szüneteltetni kell. E funkciókat a vezérlőszó két speciális bitje látja el:

EuPCs jel: Ha a vezérlőszó EuPCs (*Enable µPC step*) jele logikai 0, akkor a µPC értéke nem változik. A léptetést a műveletvégző egység felől érkező állapotjelek, vagy a külvilág felől jövő külső vezérlő jelek indíthatják újra (lásd *4.3.1. pont*: *"fs_cnti"* és *"fs_ss"* jeltípusok).

EOI jel: A folyamat végét a vezérlőszó EOI (*End Of Instruction*) bitje jelzi. Ha értéke logikai 1, akkor a dekódolást végző állapotgép (*microsequencer*) - még az előző folyamat során felhozott utasítás értéke, valamint az egyéb állapotjelek és külső vezérlő jelek alapján - a µPC bemenetére kapuzza a következő utasítás első vezérlőszavának kezdőcímét.

Egy utasításon belül elágazást közvetlenül nem valósíthatunk meg, csak a folyamat felfüggesztésére és újraindítására van lehetőségünk. Ebből következik, hogy a folyamatvezérelt mikroprocesszor modell elsősorban egyszerű, RISC jellegű utasítások megvalósítására alkalmas. Bonyolult, CISC jellegű utasítás esetén szükségünk lehet a folyamaton belüli elágazásra. A *8.1. pontban* bemutatott módszerrel elérhető a folyamaton belüli elágazással egyenértékű működés a folyamatvezérelt mikroprocesszor modell által biztosított kereteken belül.

Bonyolult műveletvégző egység esetén a vezérlőszó hossza meglehetősen nagy lehet. A problémát az ún. "*statikus utasításdekóder*" áramkörrel oldottam meg. A statikus utasításdekóder a műveletvégző egység speciális eleme, mely a vezérlésben vesz részt. A rendszer állapotjeleinek és vezérlő jeleinek megfelelően választott részhalmaza (lásd *4.3.1. pont*: *"cssid_c"* és *"cssid_d"* jeltípusok) alkalmas arra, hogy belőle egy egyszerű dekóder segítségével előállítsuk azokat a vezérlő jeleket, melyek egy-egy folyamat végrehajtása során nem változnak meg. Ilyen statikus vezérlő jelek elsősorban az adatutakat kijelölő multiplexerek kiválasztó jelei. Ezeket a jeleket nem szerepeltetjük a mikroprogram-tárban, helyette a műveletvégző egységbe épített dekóderrel állítjuk elő. A vezérlési mezők összessége így két részre osztható; a vezérlő egység által előállított és a statikus utasításdekóder által előállított vezérlési mezőkre (*4-13. ábra)*.



4-13. ábra - A folyamatvezérelt processzorok vezérlési rendszerének blokkdiagramja

A statikus utasításdekóder két szinten is gondoskodik a mikroprogram-tár méretének mérsékléséről. Egyrészt előállítja a vezérlési mezők egy jelentős részét, így csökkentve a vezérlő egység által előállítandó vezérlőszó hosszát, ugyanakkor lehetőséget teremt a vezérlőszavak számának a csökkentésére is, hiszen ha két különböző folyamathoz tartozó vezérlőszó-szekvenciák csak a statikus utasításdekóder által generált jelekben különböznek egymástól, akkor a vezérlő egység felől nézve a két folyamat ekvivalens, így a hozzájuk tartozó vezérlési mezőket elég egyszer eltárolnunk a mikroprogram-tárban. Ennek a lehetőségnek a kiaknázására szolgálnak a *4.3.1. pontban* ismertetett *"mester folyamat"* és *"átirányított folyamat"* fogalmak.

A teljes vezérlési rendszer tehát mikroprogramozott esetben három fő részből áll, a mikroprogram-tárból, a mikrosequencer-ből és a statikus utasítás dekóderből. FPGA technológián mindhárom összetevő többféleképpen valósítható meg. Az egyes implementációs lehetőségeket, azok előnyeit és hátrányait foglalja össze a *4-6. táblázat*.

Komponens	Funkció	Megvalósítás
Mikroprogram-tár	Az egyes folyamatokhoz tartozó vezérlőszó-szekvenciákat tárolja.	**Blokk RAM**: Általános célú FPGA erőforrást nem igényel, a kritikus út hosszát megnövelheti. A tár méretéből adódóan általában a blokk RAM az optimális választás.
		LUT: Elhelyezés és huzalozás szempontjából rugalmasabban optimalizálható. Erőforrásigénye nagyobb.
		FF + Mux: A kimenet aszinkronná tehető, így a sebesség megnövekszik, de erőforrásigénye rendkívül nagy.
Mikrosequencer	A külső feltételeknek és a belső állapotjeleknek megfelelően kiválasztja futásra a soron következő folyamatot.	**Blokk RAM**: Általános célú FPGA erőforrást nem igényel, a kritikus út hosszát megnövelheti.
		LUT: Nagyobb erőforrásigény, de a kritikus út hosszát általában nem befolyásolja.
Statikus utasítás dekóder	A folyamat során változatlan vezérlő jelek állítja elő.	**Blokk RAM**: Általános célú FPGA erőforrást nem igényel, a kritikus út hosszát megnövelheti. Nagy méretű logikai hálózatok helyettesítésekor előnyös. Használatához a kimenetet szinkronná kell alakítani.
		LUT: A logikai hálózat lehet aszinkron, ami gyorsabb működést eredményez, de erőforrásigénye nagy.

4-6. táblázat - A folyamatvezérelt processzorok vezérlési rendszerének elemei

Huzalozott vezérlő egység

A folyamatvezérelt processzorok huzalozott vezérlő egysége formailag a *2-5. ábrán* látható VHDL leírással egyezik meg. Az egyes folyamatok minden vezérlőszavához az állapotgép egy-egy állapotát rendeljük. Az első utasítás felhozását, illetve a dekódolást (folyamatkiválasztást) szintén külön állapotok végzik. A mikroprogramozott vezérlővel ellentétben dekódoláskor ez esetben nem a soron következő utasítás első mikroutasításának kezdőcímét, hanem az utasításhoz tartozó állapotsorozat legelső elemét állapítjuk meg, és a végrehajtást a megfelelő állapotban folytatjuk. A műveletvégzés várakoztatása és engedélyezése egyszerűbben, egy feltételvizsgálat beiktatásával valósul meg (*4-14. ábra*).

```
...
when s_mul2 => MULTIPLIERRqst <= '1';
               State <= s_mul3;
when s_mul3 => if (MULTIPLIERRdy = '1') then
                 MX_GPAddr_A <= "01"; LD_GP <= '1';
                 MULTIPLIERRqst <= '0'; State <= s_mul4;
               else State <= s_mul3; end if;
when s_mul4 => LD_IR <= '1'; MX_GPAddr_A <= "11";
               MX_GPDataIn <= "1000"; State <= s_mul5;
...
```

4-14. ábra - Folyamat várakoztatása huzalozott vezérlős rendszerben

Az *"s_mul2"* állapotban egy kérés (*"MULTIPLIERRqst"*) jelet generálunk a szorzó áramkör felé, majd átlépünk az *"s_mul3"* állapotba. Az *"s_mul3"* állapotból csak akkor lépünk át az *"s_mul4"* állapotba, ha a szorzás eredménye előállt, vagyis a szorzó áramkör a *"MULTIPLIERRdy"* vonalon jelzi, hogy a műveletet elvégezte.

Látható, hogy a folyamatvezérlés rendszere lényegesen egyszerűbb a *2.2.1. pontban* bemutatott Wilkes-féle modellnél. Az ilyen típusú mikroprocesszorok tervezésére újonnan kifejlesztett szoftver eszközt a *4.3.2. pontban* részletesen ismertetem.

4.3. Folyamatvezérelt mikrogépek viselkedésének leírása

A klasszikus ARTL leírás általános jellegű, segítségével szinte bármilyen adatfeldolgozást végző digitális rendszer leírható. Az alábbi szakaszban bemutatom azokat az ARTL leíró nyelvet érintő fejlesztéseket, melyek lehetővé teszik, hogy a *4.2. pontban* ismertetett folyamatvezérelt mikroprocesszorok viselkedését a lehető legpontosabban írjuk le. A nyelv e formailag kötöttebb, nyelvi elemeit tekintve bővebb változatát **A^{2}RTL**-nek (ASIP ARTL - *Application Specific Instruction-set Processor ARTL*) nevezem. A rendszer A^{2}RTL modellje a egy kiegészítő leírást, az ún. MPD (Microprogram Definition) leírást is tartalmazza, melynek a vezérlő egység szintézise során lesz fontos szerepe.

4.3.1. PCM-specifikus nyelvi elemek az ARTL-ben

Az ARTL leírás újonnan bevezetett nyelvi elemeit a *4-7. táblázat* foglalja össze.

Új nyelvi elem		Funkció
"instruction" típusú álnév		A mikroprocesszor operációs kódjainak definiálása.
Speciális jeltípusok	cssid_c	"Control Signal Static ID - Controller": A statikus utasítás dekóder vezérlőegység által előállított bemeneti jele.
	cssid_d	"Control Signal Static ID - Datapath": A statikus utasítás dekóder műveletvégző egység által előállított bemeneti jele.
	fs_cnti	"Forced Step - Control Input": A felfüggesztett µPC léptetést újraindító külső vezérlő jel.

	fs_ss	"Forced Step - Status Signal": A felfüggesztett µPC léptetést újraindító állapotjel.
Folyamat (process)		Egy gépi utasítás végrehajtását vagy egy kivételt leíró utasításblokk.
Mester folyamat		Olyan folyamat, melyhez saját mikroprogram-tárbeli kezdőcímet rendelünk.
Átirányított folyamat		Bizonyos folyamatokhoz, az ún. "átirányított folyamatokhoz" nem rendelünk egyedi kezdőcímet a mikroprogram-tárban, hanem azokat egy másik, "mester folyamatnak" nevezett folyamat kezdőcímére irányítjuk.
Esemény		Egy folyamaton belül egyidejűleg végrehajtott műveletek halmaza.
Számábrázolás	hexadecimális	`#h'8"AE"h`
	decimális	Előjeles: `#d'8"-82"s`
		Előjel nélküli: `#d'8"174"u`

4-7. táblázat - Az ARTL nyelvbe bevezetett új nyelvi elemek

"instruction" típusú álnév: Az *"instruction"* típusú álnevek a mikroprocesszor egyes gépi utasításainak megnevezéséhez (mnemonikjaihoz) rendelnek számértéket (operációs kódot). Használatukra a *4-15. ábra* mutat példát.

```
...
instruction mov:      #d'6"0"u;
instruction movi:     #d'6"1"u;
...
if    (ss IR[20:15] = @mov)  => $ mov instruction execution
elsif (ss IR[20:15] = @movi) => $ movi instruction execution
...
```

4-15. ábra - "instruction" típusú álnevek használata

Speciális jeltípusok: "cssid_c", "cssid_d": A *4.2.1. pontban* leírtak szerint a statikus utasításdekóder bemenetei a vezérlő egység által előállított vezérlő jelek, valamint a műveletvégző egység által előállított állapotjelek megfelelően kiválasztott halmaza. Az A^{2}RTL leírásban lehetőség van ezek pontos specifikálására. A *"cssid_d"* típusú jeleket a leírás meghatározott pontján felsorolással adhatjuk meg, a *"cssid_c"* jelek pedig a klasszikus ARTL leírás *"cs"* (*control signal*) típusú jeleiből képezhetők a típus átírásával (*4-16. ábra*).

```
...
$ Datapath-generated StaticID control signals
cssid_d IR[20:15];
cssid_d CondReg[8:0];
...
$ Behavioral descripton
...
cssid_c IntrFromCont <= '0';
...
```

4-16. ábra - "cssid_c" és "cssid_d" jelek definiálása az A²RTL leírásban

Folyamat: Az A²RTL leírás feltétel kifejezésekből és az azok teljesülése esetén végbemenő műveleteket leíró értékadás-blokkokból áll. Ezek a műveletek a *4.2. pontban* definiált folyamatok. A folyamat leírása a *4-17. ábrán* látható módon történik. A folyamatvezérelt processzorok a futás első lépéseként mindig a programmemória első elemét olvassák be, és hajtják végre, így a legelső folyamat mindig ugyanaz, a tervezőnek erre nincs befolyása. Ez a folyamat az ún. *Read First Instruction* - **RFI** folyamat, mely a *4-18. ábrán* látható.

```
$ ------------------------------------------------------------------------------------------
elsif (ss IR[20:15] = @and) =>  $ and instruction execution ------------------------------
  process AND => ADD:
    event 1: { IP <= IPAdd.Output(IP,#d'15"1"u);
               GP[IR[14:10]] <= ALU.Result(@ALU_and,GP[IR[9:5]],GP[IR[4:0]],-);
               CondReg[0] <= CondReg[0]; CondReg[1] <= CondReg[1]; CondReg[2] <= CondReg[2];
               CondReg[3] <= ALU.Zout(@ALU_and,GP[IR[9:5]],GP[IR[4:0]],-);
               CondReg[4] <= CondReg[4]; CondReg[5] <= CondReg[5]; CondReg[6] <= CondReg[6];
               CondReg[7] <= CondReg[7]; CondReg[8] <= CondReg[8]; }
    event 2: { IR <= DataFromPrMem; }
  endprocess;
$ ------------------------------------------------------------------------------------------
```

4-17. ábra - Folyamat definiálása az A²RTL nyelvben

```
$ Read first instruction ------------------------------------------------------------
if (cnti Run = "1") => process RFI: event 1: { IR <= DataFromPrMem; } endprocess;
```

4-18. ábra - "RFI" - Read First Instruction folyamat

Mester folyamat, átirányított folyamat: A *4.2.1. pontban* leírtak alapján a statikus utasításdekódernek köszönhetően bizonyos folyamatok a vezérlő egység szempontjából ekvivalensek. Ekkor csupán az egyik folyamatvezérlő jeleit tároljuk a mikroprogram-tárban, vagy huzalozott esetben csak az egyik folyamathoz tartozó állapotokat valósítjuk meg a vezérlést végző állapotgépben. Azokat a folyamatokat, melyek saját mikroprogram-tárbeli kezdőcímmel rendelkeznek *"mester folyamatnak"*, amelyekhez pedig egy mester folyamat kezdőcímét rendeljük *"átirányított folyamatnak"* nevezzük. Az átirányításnak kétféle esete lehetséges. Ha az átirányított folyamat ARTL leírása különbözik a mester folyamat leírásától, akkor az átirányítástól függetlenül az átirányított folyamatot le kell írni. Ha azonban a mester

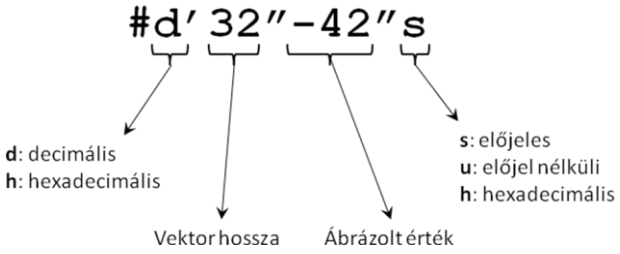
és az átirányított folyamat műveletei teljesen azonosak, úgy a folyamat leírását elég egyszer szerepeltetni (*4-19. ábra*).

```
$ ------------------------------------------------------------------------------
elsif (ss IR[20:15] = @b) =>    $ b instruction execution ----------------------
  process BRANCH:
    event 1: { IP <= IR[14:0]; }
    event 2: { IR <= DataFromPrMem; }
  endprocess;
$ ------------------------------------------------------------------------------
elsif (ss IR[20:15] = @bc and ss CondReg[4] = "1") => process BC_HIT => BRANCH;
$ ------------------------------------------------------------------------------
```

4-19. ábra - A folyamatátirányítás az A^{2}RTL nyelvben

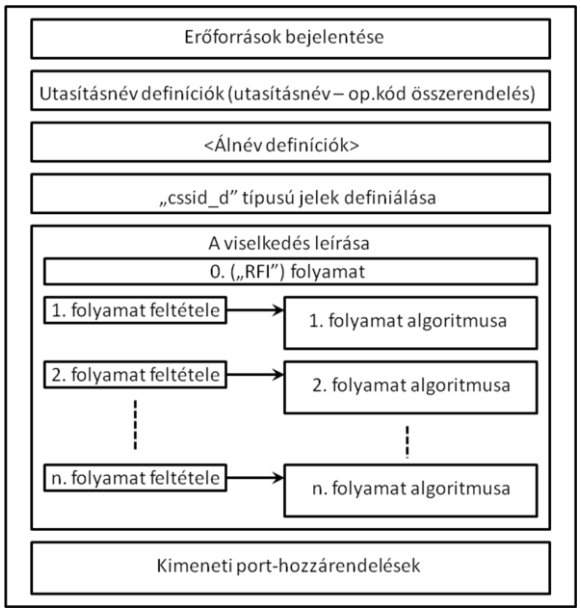
Esemény: Az esemény fogalom lehetővé teszi az egyidejűség kezelését. Az egy eseményen belül lévő értékadások jobbértéke egyszerre értékelődik ki. Ha egy balérték egy eseményen belül többször fordul elő, akkor az érintett értékadások szekvenciálisan hajtódnak végre. Ennek eredményeképpen a legutolsó értékadás kivételével mindegyik értékadás hatástalan lesz. Az operátorok hívása esetén a helyzet az előbbiekhez hasonló. Ha egy operátort többször hívunk egyazon eseményen belül, akkor az annak kimeneteihez rendelt ideiglenes változók csak a legutolsó hívás eredményét képesek tárolni.

Számábrázolás: Nagy szószélességű mikroprocesszorok tervezése esetén a számok kezelését megkönnyítendő a *4-20. ábrán* látható decimális és hexadecimális számformátumok bevezetésére került sor. A konstansok kezelése a korábbi ARTL leírás esetében is hasznos lehet. Természetesen bináris számok ábrázolására is van lehetőség.



4-20. ábra - Konstansok ábrázolása az A^{2}RTL nyelvben

Egy rendszer A^{2}RTL modellje a korábbi ARTL modellhez képest szerkezetileg is kötöttebb. A leírás felépítése a *4-21. ábrán* látható.



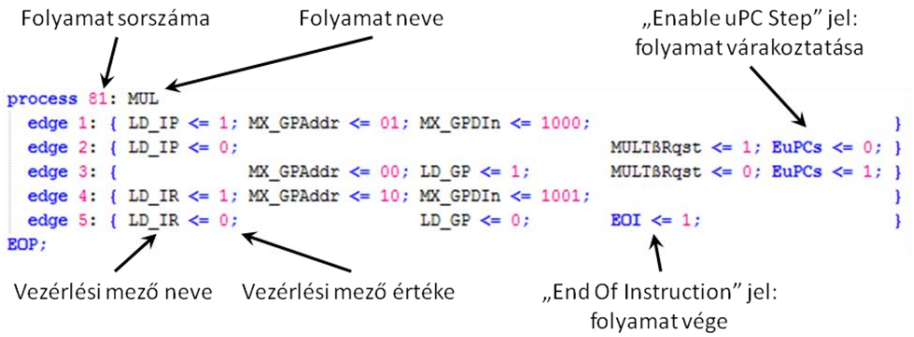
4-21. ábra - Az A^{2}RTL leírás szerkezete

A különböző A^{2}RTL modellek által leírt rendszerek egymáshoz nagyon hasonlóak. A műveletvégző egységet felépítő erőforrások - az operátorok kivételével - gyakorlatilag azonosak, csak paramétereikben különböznek egymástól. A vezérlőszavak előállításának módja is azonos lesz adott vezérlő egység típus esetén. Megállapítható tehát, hogy a különböző A^{2}RTL leírások a különböző alkalmazásorientált utasításkészletű tárolt programú gépek egyazon konstrukció szerinti megvalósítását írják le. Ez a konstrukció nem más, mint a *4.2. pontban* ismertetett folyamatvezérelt mikroprocesszor modell.

4.3.2. Az MPD (*Microprogram Definition*) leírás

A *4.2.1. pontban* bemutatott vezérlési rendszer megtervezését teszi lehetővé az ún. *Microprogram Definition* - **MPD** leírás, melyben az egyes folyamatok egyedi vezérlőszavait definiálhatjuk. Az MPD leírás az A^{2}RTL leírást egészíti ki a rendszervezérlő egységére vonatkozó információkkal. A *4.4.3. pontban* ismertetésre kerülő ARTL2VHDL modellgeneráló szoftver az MPD leírás alapján a vezérlő egység VHDL modelljét automatikusan állítja elő.

Egy folyamat MPD leírására mutat példát a *4-22. ábra*.



4-22. ábra - "hso"-t hívó folyamat MPD leírása

Az egyes folyamatok ”élekre” (*edge*) osztva definiálhatók, utalva arra, hogy az egy élnél beállított vezérlési mezők a vezérlő egység órajelének egyazon felfutó élénél fognak megjelenni a vezérelt objektum bemenetén. Az MPD leírásban nem szerepel minden vezérlő jel, mely a műveletvégző egység működésének pontos definiálásához szükséges. Az MPD leírásban *nem* szereplő jelek előállításáról a statikus utasításdekóderben kell gondoskodni. A legelső folyamat (RFI - *Read First Instruction*) az A^{2}RTL leírásnál látottakhoz hasonlóan az MPD leírásban is előre definiált (*4-23. ábra*).

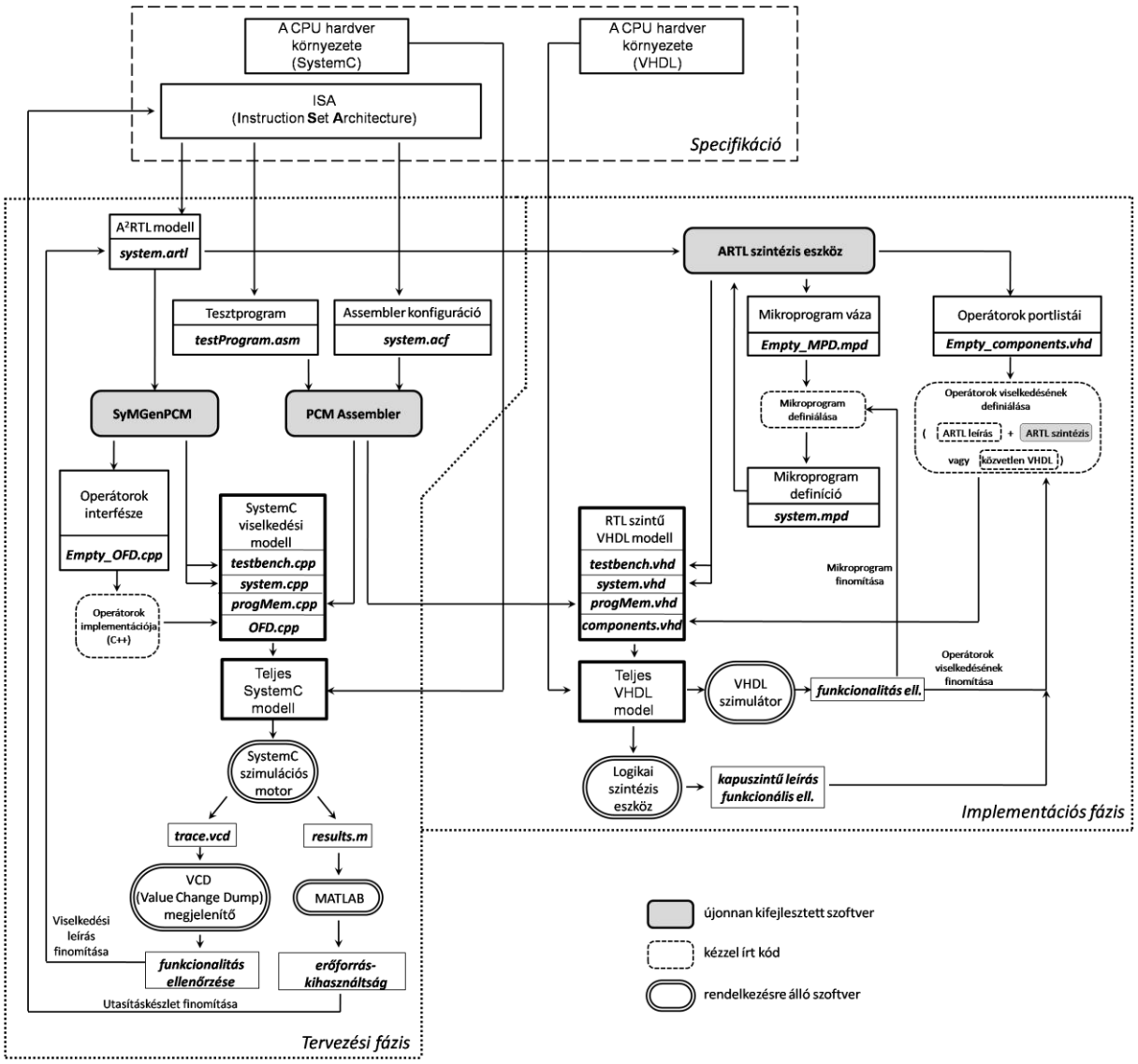
```
process 1: RFI
  edge 1: { reset                }
  edge 2: { LD_IR <= 1;          }
  edge 3: { LD_IR <= 0; EOI <= 1; }
EOP;
```

4-23. ábra - "RFI" - Read First Instruction folyamat MPD leírása

Az első élnél szereplő *reset* kulcsszóval a vezérlő egység által előállított vezérlési mezők mindegyike inaktív állapotba hozható. Látható, hogy az MPD leírás semmilyen korlátozást nem tartalmaz az egyes utasításokat leíró vezérlőszó-szekvenciák hosszára vonatkozóan. Tetszés szerint előállíthatunk tehát akár vertikális, akár az összes lehetséges erőforrást párhuzamosan vezérlő horizontális mikrokódot. A vertikális jellegű mikroprogram előállítása lényegesen egyszerűbb, és gyorsabb, ugyanakkor a kész mikroprocesszor műveletvégzési sebessége ez esetben természetesen alacsony lesz. Horizontális mikrokód esetén a helyzet fordított; a folyamatok definiálása sok időt vesz igénybe, az utasítás-végrehajtás azonban a lehető leggyorsabb lehet. A tervezés korai fázisában célszerű egy vertikális jellegű mikrokód előállítása, így a rendszer gyorsan működőképes állapotba hozható, a műveletvégző egység tehát tesztelhető. Ha szükséges, a mikroprogram-tár tartalma utólag is lecserélhető egy sebességre optimalizált mikrokódra. Huzalozott vezérlő egység esetén ugyan nincs mikroprogram-tár, de a későbbiekben látni fogjuk, hogy a teljes vezérlő egység automatikusan generálható a bemutatott leíró fájlokból, így a módszer ez esetben is használható.

4.4.ASIP ARTL tervezési keretrendszer

Az alábbi pontban egy általam kifejlesztett tervezési keretrendszer kerül ismertetésre, mely alkalmazásorientált utasításkészletű mikroprocesszorok és azok speciális célú hardver egységeinek megtervezését egyazon környezetben teszi lehetővé. (*4-24. ábra*).



4-24. ábra - ASIP ARTL tervezési keretrendszer

A keretrendszer a *4.1. pontban* ismertetett ARTL leíró nyelvből, a *4.2. pontban* bemutatott folyamatvezérelt mikrogép modellből és néhány szoftver eszközből áll, melyek az utasításkészlet–mikroarchitektúra pár erőforrás-kihasználtság szempontjából való optimalizálásában nyújtanak segítséget. A rendszer tartalmaz továbbá egy FPGA technológiára optimalizált szintézis eljárást, melynek kimenete egy szintetizálható, RTL szintű leírás. Az alkalmazásorientált mikroprocesszor létrehozása két fő fázisra bontható. A

tervezési fázisban egy előzetes utasításkészletet megvalósító magas szintű, SystemC nyelvű modellt állítunk elő, mely logikai szintézisre ugyan nem alkalmas, de képes tesztprogramok futtatására, ezen kívül statisztikai vizsgálatokat végez a hardver erőforrások kihasználtságára vonatkozóan. E vizsgálatok eredménye alapján egyaránt finomíthatjuk az utasításkészletet és a mikroarchitektúrát. A tervezés fázisa bonyolult, iteratív feladat, hiszen az utasításkészlet változtatása a futó tesztprogram változtatását is megköveteli. Ha az optimális, de legalábbis a követelményeknek megfelelő szoftver–utasításkészlet–mikroarchitektúra hármas elkészült, következhet az implementációs fázis, melyben a megtervezett utasításkészletet megvalósító RTL szintű modellt szintén gépi úton állítjuk elő.

A keretrendszer egyes elemeinek rövid leírását a *4-8. táblázat* és a *4-9. táblázat* tartalmazza.

A^{2}RTL model (system.artl): ASIP ARTL modell: Az alkalmazásközpontú mikrogép ARTL leírása (*4.3. pont*).

Elem	Leírás
Tesztprogram (testProgram.asm)	Assembly nyelven megfogalmazott tesztprogram.
Assembler konfiguráció (system.acf)	A különböző architektúrákra írt assembly nyelvű programokat egyazon assembler fordító alakítja a megfelelő gépi kóddá. Ennek elvégzéséhez a fordítónak ismernie kell az adott architektúrára jellemző utasításszerkezeteket. Az egyes utasítások pontos felépítését az ún. ACF (*Assembler Configuration File*) fájlban definiálhatjuk (*4.4.2. pont*).
PCM Assembler	Konfigurálható assembler fordító, mely az assembly nyelvű programkódból a programmemória SystemC és VHDL modelljét állítja elő (*4.4.2. pont*).
SystemC viselkedési modell	A mikroprocesszor viselkedési szintű modelljét (*4.4.1. pont*), a programmemória modelljét és az ezeket példányosító testbench modult tartalmazza.
SyMGenPCM	*SystemC Model Generator for Process-Controlled Machines*. Modelltarnszformációs szoftver, mely a rendszer ARTL leírását egy ciklushelyes, viselkedési szintű SystemC leírássá alakítja (*4.4.1. pont*).
Empty_OFD.cpp	A rendszer ARTL leírása nem teljes funkcionális leírás. Az adatmanipulációt végző erőforrások fekete doboz modellel definiáltak, melyek pontos viselkedése ezen a ponton még nem ismert. A SystemC modell tesztelése előtt ezen operátorok funkcióját C++ nyelven fogalmazhatjuk meg.
SystemC szimulációs motor	A SystemC osztálykönyvtárhoz tartozó szimulációs motor, mely lehetővé teszi, hogy tesztprogramokat futtassunk a SyMGenPCM szoftver által előállított modellen. E szimuláció eredménye egy VCD (*Value Change Dump*) fájl és egy MATLAB környezetben feldolgozható szöveges állomány, mely az erőforrás-kihasználtságra vonatkozó mérések

	eredményeit tartalmazza (MATLAB m-file).
VCD megjelenítő	A szimulációs motor által generált VCD fájl egy szöveges formátumú leíró fájl, mely az egyes adattároló elemek szimuláció során felvett értékeit írja le. A VCD megjelenítő ebből a szöveges reprezentációból egy könnyen nyomon követhető grafikus hullámformát állít elő. A hullámformából az ARTL leírás funkcionális helyességére következtethetünk.
MATLAB	A SystemC viselkedési modell a tesztprogram futtatása során a hardver erőforrások kihasználtságára vonatkozó vizsgálatokat végez. E vizsgálat eredményeit MATLAB környezetben feldolgozható és megjeleníthető formában szolgáltatja. Az így kapott információk segítségünkre lehetnek az utasításkészlet finomításában.

4-8. táblázat - Az ASIP ARTL keretrendszer tervezési fázisa

Elem	Leírás
RTL szintű VHDL modell	A mikroprocesszor RTL szintű modelljét, a programmemória modelljét és az ezeket példányosító testbench modult tartalmazza.
ARTL szintézis eszköz	Szoftver eszköz, mely a rendszer ARTL modelljéből annak szintetizálható VHDL modelljét állítja elő. Segítségével egyaránt generálható a mikroprocesszor műveletvégző egységének szerkezeti jellegű leírása, a huzalozott és a mikroprogramozott vezérlő egység (*4.4.3. pont*).
Empty_MPD.mpd	MPD (*Microprogram Definition*) leírás. A mikroprocesszor vezérlő egységének automatikus generálásához szükséges leírás, mely a mikrokódot könnyen olvasható és szerkeszthető formában tartalmazza (*4.3.2. pont*).
Empty_components.vhd	Akárcsak a SystemC modell esetén, úgy a VHDL modellben is definiálnunk kell az adatmanipulációt végző erőforrások (operátorok) pontos viselkedését, ezúttal azonban VHDL nyelven. A komplex funkciót ellátó egységek („hand-shake operátorok” lásd: *4.1.1. pont*) VHDL modelljének elkészítéséhez szintén használható az ARTL szintézis szoftver.
VHDL szimulátor	A VHDL szimulátor segítségével modellünk funkcionális verifikációját végezhetjük el.
Logikai szintézis eszköz	Ha az RTL szintű modell helyességéről a szimulátor segítségével meggyőződtünk, akkor a logikai szintézis szoftveren a sor, hogy a rendszer logikai szintű leírását előállítsa. Ilyen szoftver eszköz pl. a Xilinx ISE XST, vagy a Mentor Graphics Leonardo Spectrum. Bár az itt bemutatott eljárással előállított leírások FPGA technológiára optimalizáltak, a logikai szintézist tetszőleges ASIC technológiára is elvégezhetjük, ekkor azonban meglehetősen magas erőforrásigényre kell számítanunk.

4-9. táblázat - Az ASIP ARTL keretrendszer implementációs fázisa

4.4.1. SyMGenPCM (*SystemC Model Generator for PCMs*) modelltranszformációs szoftver

A SyMGenPCM modelltranszformációs szoftver egy C++ nyelven implementált alkalmazás, mely a rendszer ARTL leírásából egy viselkedési szintű, ciklushelyes SystemC modellt állít elő. E SystemC modell és a SystemC szimulációs motor együttesen már alkalmas arra, hogy különböző tesztprogramokat futtassunk a fejlesztés alatt álló mikroprocesszoron, így ellenőrizve az ARTL modell helyességét, valamint az utasításkészlet és a mikroarchitektúra kihasználtságát. A generált SystemC viselkedési modell szerkezetét a *4-10. táblázat* tartalmazza. A táblázat a C++ forrásállomány szerkezetét is tükrözi.

SystemC modell eleme	Példa

Portlista	`sc_in < bool > Step;`
	`sc_in < bool > Reset;`
	`// control inputs ---------------`
	`sc_in < sc_lv<1> > Run;`
	`// control outputs --------------`
	`sc_out < sc_lv<1> > IMCE;`
	`sc_out < sc_lv<1> > Abort;`
	`// data inputs ------------------`
	`sc_in < sc_lv<8> > DataFromPrMem;`
	`sc_in < sc_lv<9> > ConfigPort;`
	`...`

Felsorolt típus létrehozása az állapotgép egyes állapotainak reprezentálásához: A SystemC modell egyetlen állapotgépet tartalmaz, melynek egyes állapotai az ARTL leírás egy-egy eseményének feleltethetők meg.	`// state machine declaration -------------------------`
	`enum FSMStates { WaitForRun,RFI,ProcessSelect,`
	`IMCE_0,`
	`LDI_SECONDSTEP_0,LDI_SECONDSTEP_1,`
	`JUMP_SECONDSTEP_0,JUMP_SECONDSTEP_1,`
	`... };`

Konstansok definiálása: Az ARTL leírás utasításkódjainak és álneveinek SystemC-beli reprezentációinak létrehozása.	`// constants ----------------------`
	`#define inst_ldi "00000"`
	`#define inst_lda "00001"`
	`#define inst_sda "00010"`
	`...`
	`#define als_DeassertPCR "000"`
	`#define als_LoadImmediate2ACC "100"`
	`#define als_LoadJumpAddress "101"`
	`...`

Belső változók és tömbök deklarálása: Az ARTL leírás "reg", "sprf" és "dprf" típusú erőforrásainak reprezentációi.	`// declaration of registers and register files --`
	`sc_lv<8> IR;`
	`sc_lv<8> IP;`
	`sc_lv<9> ACC;`
	`...`
	`sc_lv<9> Stack[4];`
	`sc_lv<9> GPRF[8];`
	`...`

Az ARTL leírás operátorait reprezentáló függvények deklarációi.	`// operator function declarations --`
	`void IPAdd(const sc_lv<8>& Input1,`
	`const sc_lv<8>& Input2,`

	sc_lv<8>& Output); void SPAdd(const sc_lv<2>& Input1, const sc_lv<2>& Input2, sc_lv<2>& Output); ...
Ideiglenes változók deklarálása, melyek az operátorok kimeneteinek értékét tárolják.	// temporary variables for operator outputs -- sc_lv<8> IPAdd_Output; sc_lv<2> SPAdd_Output; sc_lv<9> STConcatenate_Output; ...
Ideiglenes változók a regiszterek értékeinek átmeneti tárolására. Az egyidejűség kezelésében van szerepük.	// temporary variables for concurrent statements -- unsigned Tmp0; unsigned Tmp1; unsigned Tmp2; ...
A statisztikai vizsgálatok során használt adatszerkezetek deklarálása.	// statistical analysis utilities ------------ std::string pathM; std::ofstream mOut; std::map<std::string,unsigned> processEntries; std::map<std::string,unsigned> operatorEntries; std::map<std::string,unsigned> regWrites; std::map<std::string,unsigned> regReads; std::map<std::string,unsigned> regFileWrites; std::map<std::string,unsigned> regFileReads; std::set<unsigned> addressSet_Stack; std::set<unsigned> addressSet_GPRF; ...

A SystemC modul viselkedését definiáló függvény (szál).	void FunctionThread() {
	while (1) {
	if (Reset.read() == true) { ... }
	...
	...
	}

A viselkedést leíró állapotgép megvalósítása. Minden ARTL-beli eseményhez egy-egy állapotot rendelünk. A "ProcessSelect" állapot a műveletvégzés dekódolási fázisának felel meg. Az ARTL leírás feltételeinek megfelelő kifejezések kiértékelése során dől el, hogy a végrehajtás melyik állapotban folytatódjon. A SystemC modul viselkedését definiáló szál része.	switch (State) {
	case WaitForRun: if (Run.read() == "1")
	State = RFI; }
	else State = WaitForRun;
	break;
	case RFI: IR = DataFromPrMem.read();
	State = ProcessSelect; break;
	case ProcessSelect:
	if (ControlRegister(3,3) == "1" &&
	ControlRegister(4,4) == "1") {
	State = IMCE_0;
	processEntries["IMCE"]++; }
	else if (PCR(2,0) == "100") {
	State = LDI_SECONDSTEP_0;
	processEntries["LDI_SECONDSTEP"]++; }
	else if (PCR(2,0) == "101") {
	State = JUMP_SECONDSTEP_0;
	processEntries["JUMP_SECONDSTEP"]++; }
	else if (IR(7,3) == inst_ldi) {
	State = LDI_0;
	processEntries["LDI"]++; }
	...

Az egyes ARTL-beli eseményeket leíró állapotok. A SystemC modul viselkedését definiáló szál része.	case JUMP_SECONDSTEP_0:
	// generate right-side expressions ----------
	Tmp0 = IR.to_uint(); regReads["IR"]++;
	Tmp1 = static_cast< sc_lv<3> >("000").to_uint();
	// assign ----------
	regWrites["IP"]++;

	`IP = static_cast< sc_lv<8> >(Tmp0);`
	`regWrites["PCR"]++;`
	`PCR = static_cast< sc_lv<3> >(Tmp1);`
	`// select next state ----------`
	`State = JUMP_SECONDSTEP_1;`
	`break;`
	`...`

"ABORT" folyamat leírása, a statisztikai vizsgálatok eredményének kiírása.
A SystemC modul viselkedését definiáló szál része.
	`case ABORT_0:`
	`// generate right-side expressions ----------`
	`Tmp0 = static_cast< sc_lv<1> >("1").to_uint();`
	`// assign ----------`
	`Abort.write(static_cast< sc_lv<1> >(Tmp0));`
	`// select next state ----------`
	`State = ProcessSelect;`
	`State = Error;`
	`std::cout<<"Generate M-file to: ";`
	`std::cin>>pathM;`
	`mOut.open(pathM.c_str());`
	`mOut<<"clear all;"<<endl;`
	`mOut<<"close all;"<<endl;`
	`mOut<<"processEntries.IDs = [";`
	`for(std::map<std::string,unsigned>::iterator p =`
	`processEntries.begin();p !=`
	`processEntries.end();++p) {`
	`mOut<<"'"<<(*p).first<<"'";`
	`if ( ++p != processEntries.end() )`
	`mOut<<","; else mOut<<"];"<<endl;`
	`p--;`
	`}`
	`...`

Változók hozzárendelése a kimeneti portokhoz.
A SystemC modul viselkedését definiáló szál része.
	`Addr2PrMem.write(IP);`
	`Vizbe.write(ControlRegister(0,0));`
	`Vizki.write(ControlRegister(1,1));`
	`Futes.write(ControlRegister(2,2));`
	`Forog.write(ControlRegister(3,3));`
	`Vissza.write(ControlRegister(4,4));`
	`Cmotor.write(ControlRegister(5,5));`
	`...`

A SystemC modul konstruktora a modul viselkedését leíró szál megadásával és az érzékenységi listával.	`// constructor ----------`
	`SC_CTOR(WMPCM) {`
	`SC_THREAD(FunctionThread); sensitive << Step.pos() <<`
	`Reset.pos();`
	`...`

A kimeneti portok és a belső változók inicializálása. A SystemC modul konstruktorának része.	`// initialize outputs (cntos and oports) --`
	`IMCE.initialize("0");`
	`Abort.initialize("0");`
	`Addr2PrMem.initialize("00000000");`
	`...`
	`// initialize variables --`
	`State = WaitForRun;`
	`IR = 0;`
	`for(unsigned i = 0;i<4;i++) Stack[i] = 0;`
	`Tmp0 = 0;`
	`Tmp1 = 0;`
	`processEntries["RFI"] = 0;`
	`...`
	`operatorEntries["SPAdd"] = 0;`
	`...`
	`regWrites["IR"] = 0;`

4-10. táblázat - A SyMGenPCM szoftver által generált SystemC modell szerkezete

Az egyidejűség kezelése a SystemC modellben

A *4.3. pontban* leírtak szerint az ARTL leírás egy-egy eseményén belül elhelyezkedő értékadások egymással párhuzamosan hajtódnak végre. Lássuk, hogy ez a párhuzamosság miként valósul meg a kizárólag szekvenciálisan végrehajtható utasításokat tartalmazó C++ (SystemC) modellben.

Az ARTL leírás folyamatainak egyes eseményeihez a SystemC modellbeli állapotgép egy-egy állapotát rendeljük. Egy állapot három részre tagolódik (*4-11. táblázat*).

Az ARTL leírás egy eseménye:

```
event 1: { IP <= IPAdd.Output(IP,#d'8"1"u);
           GPRF[IR[2:0]] <= ACC; }
```

Az állapot eleme	Példa

Jobbérték típusú kifejezések kiértékelése: A kiértékelés eredményét egy ideiglenes változóban tároljuk. Egy-egy kifejezés kiértékelése több függvényhívást is igényelhet, ha egy regisztertömb címbemenetére vagy egy operátor bemeneti portjára egy másik operátor kimeneti portját kapcsoljuk. Ebben a lépésben történik azon balértékként szereplő regisztertömb-hivatkozások feldolgozása is, melyekben a regisztertömb címét egy operátor állítja elő.	`// generate right-side expressions --`
	`IPAdd(IP,`
	`  static_cast< sc_lv<8> >("00000001"),`
	`  IPAdd_Output);`
	`Tmp0 = IR(2,0).to_uint();`
	`Tmp1 = ACC.to_uint();`

Értékadás: A jobbérték típusú kifejezések kiértékelése során a kapott eredményeket ideiglenes változókban tároltuk. Ebben a lépésben ezeket az ideiglenes változókat hozzárendeljük az esemény megfelelő balértékeihez.	`// assign --`
	`IP = IPAdd_Output;`
	`GPRF[Tmp0] = static_cast<sc_lv<9> >(Tmp1);`

Következő állapot kijelölése: Egy esemény végrehajtása után a műveletvégzés az aktuális folyamat következő eseményével folytatódik. Ha az adott folyamatnak nincs több eseménye, akkor a következő állapot a "*ProcessSelect*", mely a mikroprocesszor dekódolás állapotának felel meg.	`// select next state --`
	`State = SDA_1;`
	`break;`

4-11. táblázat - A SystemC modell állapotgépének egy állapota

A párhuzamosság kezelésének fenti mechanizmusából következik két szabály, melyeket be kell tartanunk az ARTL leírás megfogalmazásakor:

1. Egyazon regiszter nem szerepelhet egy eseményen belül kétszer balértékként, mert ebben az esetben az állapot második szakaszában kizárólag a legutolsó értékadás érvényesülne.
2. Egy adott operátor kimenetére egy eseményen belül csak egyszer hivatkozhatunk, mert az operátorok kimeneteinek ideiglenes tárolására kimenetenként csak egy-egy változó áll rendelkezésre.

Operátorok funkciójának definiálása a SystemC modellben

A SyMGenPCM szoftver által generált SystemC modell nem tartalmazza az ARTL leírásban előforduló operátorok pontos funkcióját, hiszen azt maga az ARTL modell sem tartalmazza. Ezek az operátorok függvényekként jelennek meg a SystemC modellben, melyek deklarációit a rendszer automatikusan generálja, definícióikat azonban a felhasználónak kell elkészíteni. A rendszer előnyös tulajdonsága, hogy a tervezendő alkalmazásorientált mikrogép esetlegesen bonyolult, speciális célú utasításokat végrehajtó alrendszereinek funkcióját ezen a ponton C++ nyelven fogalmazhatjuk meg. A *4-25. ábrán* egy ARTL leírásbeli operátor és annak SystemC-beli megfelelője látható.

```
resource IPAdd: ao (Input1[8],Input2[8])(Output[8]);

void IPAdd(const sc_lv<8>& Input1,const sc_lv<8>& Input2,sc_lv<8>& Output);
```

4-25. ábra - ARTL-beli operátor deklaráció és az annak megfelelő SystemC függvénydeklaráció

A SystemC-beli operátornak megfelelő függvénynek nincs visszatérési értéke. Ennek magyarázata, hogy általános esetben az operátornak több kimenete is van. Ez csak úgy valósítható meg, ha az operátor a paraméterlistáján adja vissza a megfelelő kimeneteket, a mellékhatásként megváltoztatott változókat pedig referenciaként veszi át. (A bemeneti paramétereket szintén referenciaként adjuk át a függvénynek, de ennek csak a futási idő szempontjából van jelentősége. Ezzel a megoldással nincs szükség a nagyméretű SystemC objektumok veremre másolására.)

Statisztikai vizsgálatok a SystemC modellben

Az egyes gépi utasítások funkcionális leírásának helyessége nyilvánvalóan szükséges a rendszer hibátlan működéséhez, a SyMGenPCM szoftver azonban ennél többet is kínál. A tervezési folyamat végterméke egy olyan mikroprocesszor modell, mely kifejezetten egy adott problémakörben felmerülő gyakori feladat hatékony megoldását célozza. Hatékonyság alatt jelen esetben elsősorban a hardver erőforrások hatékony kihasználását értjük. Még a tervezési

folyamat korai szakaszában szükséges lehet olyan információ kinyerése, mely a futó program és a hardver összhangjára utal, arra, hogy a futó szoftver mennyire hatékonyan használja fel a rendelkezésre álló erőforrásokat. Ezt az információt a SyMGenPCM szoftver által generált SystemC viselkedési modell egy sor beépített, az erőforrások kihasználtságát monitorozó adatszerkezet formájában szolgáltatja (*4-12. táblázat*).

C++ STL-beli adatszerkezet	Leírás
`std::map<std::string,unsigned> processEntries;`	Asszociatív tömb, mely az egyes folyamatok azonosítóját, és az éppen futó tesztprogramban való előfordulásuk számát tartalmazza.
`std::map<std::string,unsigned> operatorEntries;`	Asszociatív tömb, mely az egyes operátorok azonosítóját, és az éppen futó tesztprogramban való hívásuk számát tartalmazza.
`std::map<std::string,unsigned> regWrites;`	Asszociatív tömb, mely az egyes regiszterek azonosítóit tartalmazza és azt, hogy értékük hányszor frissült a tesztprogram futása során.
`std::map<std::string,unsigned> regReads;`	Asszociatív tömb, mely az egyes regiszterek azonosítóit tartalmazza és azt, hogy hány olyan értékadás futott le a tesztprogram futtatása során, melyben az adott regiszter jobbértékként szerepel.
`std::map<std::string,unsigned> regFileWrites;`	Asszociatív tömb, mely az egyes regisztertömbök azonosítóit tartalmazza és azt, hogy hány alkalommal hajtódott végre írás művelet az adott regisztertömbön a tesztprogram futása során.
`std::map<std::string,unsigned> regFileReads;`	Asszociatív tömb, mely az egyes regisztertömbök azonosítóit tartalmazza és azt, hogy hány olyan értékadás futott le a tesztprogram futtatása során, melyben az adott regisztertömb jobbértékként szerepel.
`std::set<unsigned> addressSet_Stack;`	Az egyes regisztertömbökhöz rendelt halmaz, mely azokat a címeket tartalmazza, amelyeken írás művelet hajtódott végre a tesztprogram futása során. Segítségével felmérhető a belső tárolók kapacitás-kihasználtsága.

4-12. táblázat - Statisztikai vizsgálatokhoz használt adatszerkezetek

A rendszer a mérési eredményeket MATLAB környezetben feldolgozható formában (MATLAB m-fájl) bocsátja rendelkezésünkre (*4-26. ábra*).

```
clear all;
close all;
processEntries.IDs = ['ABORT','ADD','AND','CALL',...]
processEntries.Values = [1,3558,8,2,2,...]
operatorEntries.IDs = ['ACCConcatenate','ALU','IPAdd',...]
operatorEntries.Values = [42,18718,29551,...];
regWrites.IDs = ['ACC','ControlRegister','IP',...];
regWrites.Values = [14037,19512,35596,...];
regReads.IDs = ['ACC','ControlRegister','IP',...];
regReads.Values = [661,16260,0,...];
regFileWrites.IDs = ['GPRF','Stack'];
regFileWrites.Values = [661,4];
regFileReads.IDs = ['GPRF','Stack'];
regFileReads.Values = [646,8];
addressSet_Stack = [0];
capacity_Stack = 4;
addressSet_GPRF = [0,1,2,3,4,5,6,7];
capacity_GPRF = 8;
```

4-26. ábra - A statisztikai vizsgálatok eredményét tartalmazó MATLAB m-file

4.4.2. PCM Assembler

Tárolt programú gépek tervezésekor a funkció ellenőrzése utasításszekvenciák, tesztprogramok futtatásával történik. A fejlesztés korai fázisában e tesztprogramok előállítása nehéz feladat, hiszen az éppen fejlesztés alatt álló és ebből adódóan folyamatosan változó utasításkészlethez nem létezik assembler fordító. A *PCM Assembler* olyan C++ nyelven íródott, a legalapvetőbb assembler fordító funkciókat megvalósító alkalmazás, mely tetszőleges utasításkészletre írt assembly programot képes a tervezés alatt álló mikroprocesszor gépi kódjára fordítani. Az esetlegesen megváltozó utasításkészlethez egy speciális utasításkészlet-definíciós fájl segítségével - egy konfigurációs folyamat során - gyorsan alkalmazkodik. Az utasításkészlet szerkezetét leíró fájl az ún. ACF (*Assembler Configuration File*) fájl, mely az összes gépi utasítás pontos szerkezetét leírja. Az ACF fájlban tárolt információkat a *4-13. táblázat* foglalja össze.

Az ACF leírás eleme	Szintaktika
A rendszer egyedi azonosítójának definíciója	`SID ExampleCore`
A programmemória címbuszának mérete	`AL 10`
Egy utasítás hossza (a programmemória adatbuszának mérete)	`WL 10`
Az operációs kód hossza	`OCL 6`
Utasítás szerkezetének definíciója: <Operációs kód> <mnemonik> (<1.szóban helyet foglaló paraméterek>)+... + (<n.szóban helyet foglaló paraméterek>), ahol (<n.szóban helyet foglaló paraméterek>): (<1.paraméter felső index:1.paraméter alsó index>,...,<k.paraméter felső index:k.paraméter alsó index>)	`0h add (3:0) + (9:6,4:1)`

4-13. táblázat - Az ACF fájlban tárolt információk

Az assembler szoftver az ACF fájl beolvasása után képes az abban leírt szerkezettel rendelkező utasításkészletre megírt assembly nyelvű programokat gépi kódra fordítani (*4-27. ábra*).

```
                ; ...
                EXCEPTION_HANDLER_SELECT:
                  push 0d       ; push r0 into the stack
                  rexc 0d       ; r0 = exception code
                  cmpi 0d,42d   ; if exception code = 42 then call DIVNULL_EXCEPTION_HANDLER
                  bz   DIVNULL_EXCEPTION_HANDLER
                  cmpi 0d,2d    ; if exception code = 2 then call ARITH_OVERFLOW_EXCEPTION_HANDLER
                  bz   ARITH_OVERFLOW_EXCEPTION_HANDLER
                  movi 0d,FFh   ; r0 = FFh
                  wrp  0d,0d    ; port A = r0
                  pop  0d       ; pop r0 from the stack
                  abort         ; stop the program ( unknown exception )

                DIVNULL_EXCEPTION_HANDLER:
                  movi   0d,D0h   ; r0 = D0h
                  wrp    0d,0d    ; port A = r0
                  abort           ; stop the program ( division by zero exception )

                ARITH_OVERFLOW_EXCEPTION_HANDLER:
                  movi   0d,A0h   ; r0 = A0h
                  wrp    0d,0d    ; port A = r0
                  abort           ; stop the program ( arithmetic overflow exception )
@32767          JMP_2_MAIN_ISR:   b  EXCEPTION_HANDLER_SELECT ; jump to main ISR
```

4-27. ábra - A PCM Assembler-rel feldolgozható asm fájl részlete

A fordítás kimenete a programmemória SystemC és VHDL nyelvű modellje. A PCM Assembler szolgáltatásai közé tartozik továbbá az implicit fordítási cím megadás, a címkekezelés (abszolút- és IP-relatív címzéssel), a több (tetszőleges) bájtos utasítások és a megjegyzések kezelése.

4.4.3. ARTL2VHDL modellgeneráló szoftver

A digitális technikában sok módszer ismert adatfeldolgozást végző rendszerek magas szintű szintézisére [2], vagyis arra, hogy egy magas - például algoritmus - szintű leírásból egy szerkezeti leírást automatikusan állítsunk elő. Az ARTL2VHDL modellgeneráló szoftver is ezt a funkciót látja el azzal a különbséggel, hogy nem csak konkrét feladatot ellátó, ún. célhardver szintetizálására képes, hanem az ARTL mellett a szintén magas elvonatkoztatási szintű A^2RTL leírást feldolgozva általános vagy speciális célú tárolt programú gép szerkezeti leírását is generálja.

A tervezés menete – Design Flow

Az ARTL2VHDL modellgeneráló szoftver egy C++ nyelven íródott alkalmazás. A szoftver működése a tervezési folyamaton (*design flow*) keresztül kerül bemutatásra.

A magas szintű leírás feldolgozását a program elemi lépések sorozatára bontja. Az egyes lépéseket a *4-14. táblázat* foglalja össze.

Elemi lépés megnevezése	Leírás
1. Munkakönyvtár beállítása, ARTL fájl megadása	A modellezett rendszert leíró fájlokat tartalmazó könyvtár elérési útját adhatjuk meg. A generált kimeneti fájlok szintén ebbe a könyvtárba kerülnek.
2. Erőforráslista felépítése	Az ARTL/A^2RTL leírásban bejelentett erőforrásokról lista készül, mely az erőforrás adatait tartalmazza (pl. regiszter esetén annak mérete, operátor esetén a be- és kimenetek neve és mérete stb.).
3. Értékadások kiválasztása	A leírásban előforduló értékadásokat egy listába rendezzük. Ez a lista a műveletvégző egységre vonatkozó információk nagy részét tartalmazza.
4. Értékadások kiírása fájlba	Opcionális lépés, elsősorban ellenőrzésre szolgál. A kimeneti fájl *"balérték <= jobbérték"* formában tartalmazza a leírás összes értékadását. A kimeneti fájl a munkakönyvtárba kerül.

5. Operátorok hozzáadása	Operátort tartalmazó kifejezés esetén ebben a lépésben újabb értékadások jönnek létre, melyek az operátorok bemeneteire vonatkozó jelhozzárendeléseket és az operátor által visszaadott érték megfelelő tároló elem bemenetére juttatását írják le.
	Pl.:
	`Reg1 <= Add.Output(Reg2,Reg3);` ⟶
	`Add(Input1) <= Reg2;`
	`Add(Input2) <= Reg3;`
	`Reg1 <= Add(Output);`
	A lépés során egymásba ágyazott operátorokat tartalmazó kifejezések esetén az egymásba ágyazottság szintje csak

	eggyel csökken, így ez a lépés többször is elvégzendő.

6. Regisztertömb-címzés feloldása	A regisztertömbök címzésének feloldása során két új értékadás keletkezik. Az egyik a regisztertömb címbemenetére a címet képző kifejezést kapcsolja, a másik pedig vagy a regisztertömb kimenetét juttatja egy másik erőforrás bemenetére, vagy egy másik erőforrás kimenetét kapcsolja a regisztertömb adatbemenetére. A két utóbbi értékadás közül mindig csak az egyik jön létre attól függően, hogy a regisztertömbre való hivatkozás az értékadás jobb- vagy balértéke volt.
	Pl.:
	`RF[Reg2] <= Reg1;` ⟶
	`RF(Address) <= Reg2;`
	`RF(DataIn) <= Reg1;`
	A regisztertömbökre és operátorokra vonatkozó kifejezések egymásba ágyazhatósága miatt a lépés többszöri végrehajtására lehet szükség.

7. Portnevek hozzáadása a regiszterekhez	A feldolgozásnak ebben a szakaszában már minden *"reg"*-től különböző típusú erőforrásra való hivatkozás *"erőforrásnév(portnév)"* formájú. A regiszterek portneveit ebben a lépésben fűzzük azok neve mögé. Pl.: `Reg1 <= Reg2;` ⟶ `Reg1(DataIn) <= Reg2(DataOut);`

8. Multiplexerek beiktatása	Ha egy erőforrás bemenetére több másik erőforrás kimenetét kell kapcsolni, akkor - FPGA technológiáról lévén szó - multiplexert kell beiktatni.
	Pl.:
	`Reg1(DataIn) <= Reg2(DataOut);` ⟶
	`Reg1(DataIn) <= Reg3(DataOut);`
	`MX_Reg1(Input0) <= Reg2(DataOut);`
	`MX_Reg1(Input1) <= Reg3(DataOut);`
	`Reg1(DataIn) <= MX_Reg1(Output);`

9. Components.vhd fájl generálása	Az ARTL/A^{2}RTL leírásban szereplő operátorok VHDL modellje nem definiált. Ezek az operátorok általában egyszerű aritmetikai műveleteket írnak le, de lehet közöttük bonyolultabb funkcionális elem is (pl. ALU). A szintézis során generálódó *"Components.vhd"* nevű állomány a leírásban előforduló operátorok portlistáit, valamint az adattároló elemek előre definiált VHDL modelljét tartalmazza. A kimeneti fájl a munkakönyvtárba kerül.
10. Vezérlő- és állapotjelek, Empty_MPD.mpd generálása	Az ARTL/A^{2}RTL leírás alapján egy listát generálunk a rendszer vezérlő- és állapotjeleiről. Ezek egy része implicit módon van megadva (*"ss"* állapotjel), mások pedig erőforrástípushoz kötöttek (pl. regiszter *"load"* jel, multiplexer kiválasztó jel). Tárolt programú gép tervezése esetén ebben a lépésben

	történik az *"Empty_MPD.mpd"* fájl generálása is, mely egy üres MPD leírást tartalmaz. A^{2}RTL leírás esetén ez az üres MPD fájl a mester folyamatok vázát tartalmazza. A vezérlőszavak *4.2.2. pontban* bemutatott módon való megszerkesztésével jutunk a végleges MPD leíráshoz. A folyamatok váza *n* számú *"reset"* éllel definiált, ahol az *n* számot a tervező adhatja meg. Az n szám a folyamatok pontos definiálása során nem változhat meg. Ha az előre becsült értéknél kevesebb élre van szükség egy adott folyamathoz, akkor a kihasználatlan élek *"reset"* állapotban maradhatnak.
11. Vezérlő- és állapotjelek kiírása fájlba	Opcionális lépés. Az MPD fájl megszerkesztéséhez nagy segítség lehet a rendszer összes vezérlő- és állapotjelének fájlba írása. A kimeneti fájl a munkakönyvtárba kerül.
12. Végleges MPD fájl megadása	A 10. lépésben generált üres MPD fájl kitöltésével nyert MPD leírás nevének megadása. A fájlt célszerű a munkakönyvtárba másolni.
13. Statikus ID generálása	A *4.2.1. pontban* ismertetett statikus utasítás dekóder áramkör portlistájának generálása. Az áramkör VHDL megvalósítása többféle lehet, az általa előállítandó jelek listáját a szoftver a végleges MPD leírásból állapítja meg: az MPD leírásban *nem* szereplő jelek képezik a statikus utasításdekóder kimeneteit.
14. Mikroprogram-tár VHDL modelljének generálása	A végleges MPD leírás alapján a szoftver előállítja a mikroprogramozott vezérlő egység részét képező mikroprogram-tár VHDL modelljét. Ebben a lépésben van lehetőség az egyes vezérlési mezők inaktív állapotának definiálására, melyre az MPD leírás feldolgozása során van szükség (lásd *4.3.2. pont*: *"reset kulcsszó"*).
15. MicroSequencer generálása	A mikroprogramozott vezérlő egységben lévő, az egyes folyamatok kiválasztását (dekódolást), valamint a mikroprogram-tár címzését végző állapotgép VHDL modelljének generálása szintén a végleges MPD leírás alapján történik.
16. Vezérlő egység VHDL modelljének generálása	A vezérlő egység VHDL modelljének automatikusan generált része az ARTL és az A^{2}RTL modellek esetén különböző (*4-15. táblázat*).
17. Műveletvégző egység VHDL modelljének generálása	A szoftver által generált műveletvégző egység egy RTL szintű, szerkezeti jellegű leírás, mely az ARTL/A^{2}RTL leírásban bejelentett erőforrások és a szükséges összeköttetések megvalósítása céljából beépített multiplexerek példányosítását tartalmazza.
18. TopLevel modul VHDL modelljének generálása	A TopLevel modul a vezérlő- és a műveletvégző egység példányosítását tartalmazza.

4-14. táblázat - ARTL2VHDL szintézis lépései

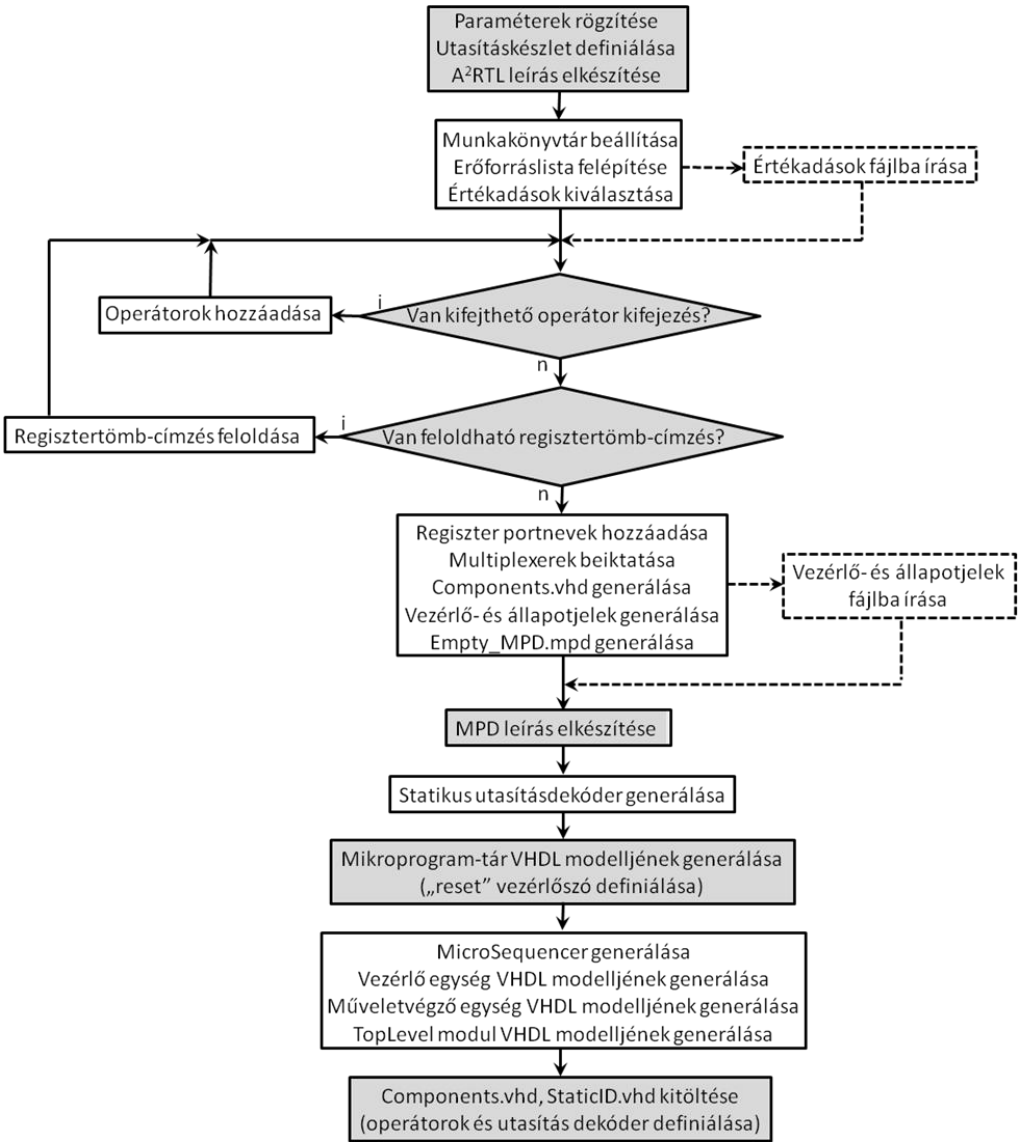
A szintézishez szükséges lépések és az előállított kimeneti fájlok egyaránt függnek a feldolgozott leírás minőségétől (ARTL vagy A^{2}RTL) és a modellezett rendszer jellegétől

(speciális célú adatfeldolgozást végző rendszer vagy tárolt programú gép). E paraméterek alapján három különböző tervezési folyamatot különböztethetünk meg (*4-15. táblázat*).

<table>
<tr><th>Szintézis bemenete</th><th colspan="2">Szükséges lépések</th><th>Szintézis kimenete</th></tr>
<tr><td>Klasszikus ARTL leírás speciális célú rendszerről.</td><td colspan="2">1 - 11. ; 16 - 18.</td><td>- Műveletvégző egység RTL szintű, szerkezeti jellegű leírása
- Vezérlő egység portlistája</td></tr>
<tr><td rowspan="2">Klasszikus ARTL leírás tárolt programú rendszerről.</td><td>Huzalozott vezérlővel</td><td>1 - 13. ; 16 - 18.</td><td rowspan="2">- Műveletvégző egység RTL szintű, szerkezeti jellegű leírása
- Vezérlő egység portlistája</td></tr>
<tr><td>Mikroprogramozott vezérlővel</td><td>1 - 18.</td></tr>
<tr><td rowspan="2">A^{2}RTL leírás folyamatvezérelt mikroprocesszorról.</td><td>Huzalozott vezérlővel</td><td>1 - 13. ; 16 - 18.</td><td>- Műveletvégző egység RTL szintű, szerkezeti jellegű leírása
- Vezérlő állapotgép viselkedési jellegű leírása</td></tr>
<tr><td>Mikroprogramozott vezérlővel</td><td>1 - 18.</td><td>- Műveletvégző egység RTL szintű, szerkezeti jellegű leírása
- Mikroprogramozott vezérlő egység RTL szintű, szerkezeti jellegű leírása
- MicroSequencer állapotgép viselkedési jellegű leírása</td></tr>
</table>

4-15. táblázat - A szintézishez szükséges lépések és a folyamat kimenete az egyes rendszertípusok esetén

Látható, hogy a szoftver összes funkcióját a folyamatvezérelt mikroprocesszorok A^{2}RTL leírás alapján való szintézise során használhatjuk ki. Ezt a tervezési folyamatot a *4-28. ábrán* látható folyamatábra szemlélteti. A szürke háttérrel jelzett mezők azokat a lépéseket jelölik, melyek végrehajtásához a tervező beavatkozása szükséges.



4-28. ábra - Design flow A²RTL modellel specifikált tárolt programú gép esetén

A tervezési folyamat utolsó lépése, hogy az A²RTL leírásban fekete doboz modellel reprezentált operátorok és a statikus utasításdekóder viselkedését definiáljuk. Ehhez a szoftver által generált "Components.vhd", illetve "StaticID.vhd" állományok kitöltése szükséges. A *4-29. ábrán* látható egy összeadó operátor VHDL modelljének szoftver által generált és a tervező által definiálandó (kiemelt) része.

```
library ieee;
use ieee.std_logic_1164.all;
-- additional library declarations ------------
use ieee.std_logic_unsigned.all
----------------------------------------------
entity STSPAdd is
port (Input1: in  std_logic_vector(7 downto 0);
      Input2: in  std_logic_vector(7 downto 0);
      Output: out std_logic_vector(7 downto 0));
end STSPAdd;
----------------------------------------------
architecture Behavior of STSPAdd is

  -- constants and types ----------------------

  -- components -------------------------------

  -- signals ----------------------------------

begin

  -- functionality ----------------------------
  Output <= Input1 + Input2;
  ----------------------------------------------

end Behavior;
----------------------------------------------
```

4-29. ábra - Összeadó modul a Components.vhd fájlban

A kevert elvonatkoztatási szintű leírások esetén nem szokatlan, hogy egyes funkcionális egységek viselkedését alacsonyabb szinten kell megfogalmaznunk [9]. A probléma mérsékelhető lenne, ha a gyakran használt operátorok - összeadás, kivonás, összefűzés - alacsony szintű leírását előre definiálnánk. Ebben az esetben az ARTL leírás kiegészíthető lenne egy "implicit operátorhívás" funkcióval, melyekhez a szintézis során előre definiált HDL modellt rendelhetnénk.

5. AZ ÚJ ELJÁRÁSSAL TERVEZETT PROCESSZOROK KIÉRTÉKELÉSE

Az alábbi pontban a dolgozatban tárgyalt tervezési módszer és szoftverrendszer segítségével készült eszközök közül három, funkció és összetettség tekintetében nagyon eltérő rendszer kerül ismertetésre. Az első ezek közül a *Taylor_FXD* aritmetikai társprocesszor, a második egy átlagos bonyolultságú mosógép vezérlésére alkalmas eszköz (*WMPCM*), a harmadik pedig egy általános célú utasításkészletet megvalósító 8 bites mikroprocesszor (*LAZARUS v1.4*). Ezek közül a *Taylor_FXD* nevű, függvénytani problémák megoldására alkalmas aritmetikai társprocesszor speciális funkciói részletesen is bemutatásra kerülnek a következő szakaszban.

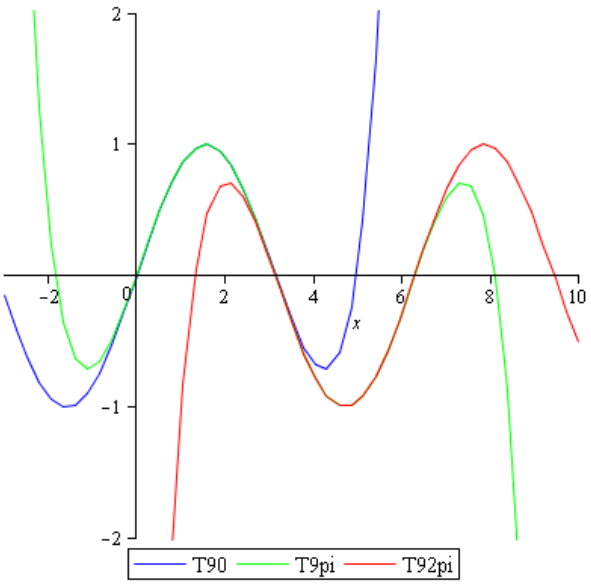
5.1. Példa alkalmazásorientált utasításkészletre

A *Taylor_FXD* kifejezetten társprocesszor jellegű áramkör, utasításkészlete mindössze 27 gépi utasításból áll, melyek között szerepelnek általános jellegű adatmozgató és aritmetikai műveletek és speciális célú utasítások, melyek a mikroprocesszort bizonyos gyakori trigonometriai függvények - sőt, akár tetszőleges, sorba fejthető függvény - helyettesítési értékeinek Taylor-soros közelítéssel való kiszámítására különösen alkalmassá teszik. E speciális gépi utasításokat az *5-1. táblázat* foglalja össze. A mikroprocesszor nevében szereplő "*FXD*" jelzés egy beépített fixpontos osztó egységre utal.

Gépi utasítás	Funkció	Megvalósítás	Pontosság
exp <arg>	y = exp(x)	9. fokú Taylor-polinomos közelítés 4 különböző bázispont körül	0.35% a [-13;6.5) tartományon
ln <arg>	y = ln(x)	1. és 9. fokú Taylor-polinomos közelítés 9 különböző bázispont körül	1.5% a [0.036;7.3) tartományon
sin <arg>	y = sin(x)	9. fokú Taylor-polinomos közelítés 3 különböző bázispont körül	3.5 ppm a számábrázolás által lehetővé tett tartományon: [-2048 .. 2048)
cos <arg>	y = cos(x)	9. fokú Taylor-polinomos közelítés 3 különböző bázispont körül	3.5 ppm a számábrázolás által lehetővé tett tartományon: [-2048 .. 2048)
tsm <arg> (Taylor-series Member)	$y = c*(x-a)^{unsigned(n)}$		A számábrázolás pontossága: ($2^{-20} = 9.5367x10^{-7}$)

5-1. táblázat - A Taylor_FXD aritmetikai társprocesszor speciális utasításai

Az *5-1. táblázatban* felsorolt speciális utasításokat kivétel nélkül egy a mikroprocesszor ARTL leírásában "hso"-ként (hand-shake operator) megjelenő, és szintén ARTL szintézis eljárással elkészített funkcionális egység, a TAU (Taylor Arithmetic Unit) végzi. Ez az aritmetikai egység a különböző függvények helyettesítési értékét azok Taylor-polinomjának adott helyen való kiszámításával végzi. A Taylor-polinomok jellegzetessége, hogy a közelített függvényt csak egy bázispont környezetében közelítik adott hibahatár alatt. Ez azt eredményezi, hogy egy adott bázisponthoz tartozó Taylor-polinom a függvényt csak az értelmezési tartomány egy viszonylag szűk tartományán helyettesítheti. A TAU egység a különböző trigonometriai függvényeket nem egyetlen Taylor-polinommal, hanem az argumentumtól függően más-más bázispontú és fokszámú polinommal közelíti. A bázispontok száma és elhelyezkedése egyaránt függ az adott függvénytől (Általánosan igaz, hogy a "gyorsan változó" függvényeket nehéz Taylor-sorral közelíteni). A szinusz függvény három különböző bázispontú Taylor-polinommal való közelítését szemlélteti az *5-1.ábra*.



5-1. ábra - Szinusz függvény közelítése Taylor-polinomjaival

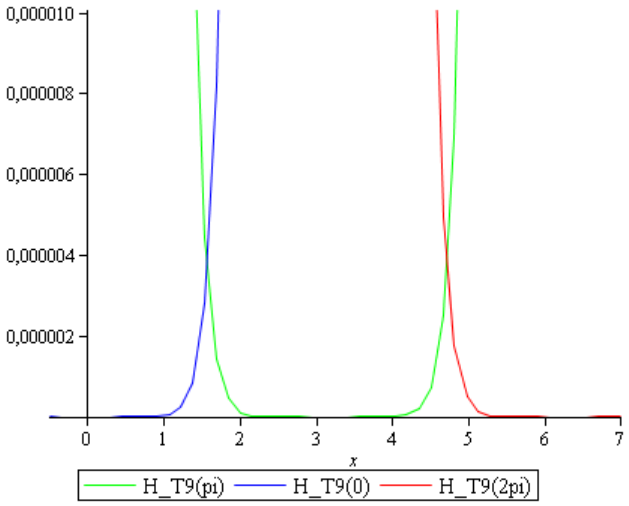
Az *5-1. ábra* három függvénye közül az argumentum értéke alapján választunk, így elérhető, hogy a relatív hiba - a közelített függvényérték valódi függvényértéktől való eltérésének aránya a valódi függvényértékhez képest (*1. képlet*) - alacsony maradjon.

Az *5-2. ábra* a szinusz függvény *5-1. ábrán* látható Taylor-polinomos közelítéseinek relatív hibáját ábrázolja az argumentum függvényében.

$$H\_T9_{bp}(x) = |sin(x) - T9_{bp}(x)| / |sin(x)|, \qquad (1)$$

ahol $H\_T9_{bp}(x)$*:* a 9. fokú, *bp* bázispontú Taylor-polinom relatív hibája

$T9_{bp}(x)$: a 9. fokú, *bp* bázispontú Taylor-polinom helyettesítési értéke



5-2. ábra - A szinusz függvény Taylor-polinomjainak relatív hibája az argumentum függvényében

Az *5-1. táblázatban* látható, hogy több bázispont felvételével is korlátozott az a tartomány, melyen a függvény megfelelő pontossággal közelíthető (Ez alól kivételt képeznek a periodikus függvények). Ha egy függvény - jelen esetben nem kell, hogy trigonometriai függvény legyen, tetszőleges, sorba fejthető függvény lehet - adott tartományon való közelítésére van szükségünk, használhatjuk a TAU egység "tsm" (Taylor-series Member) utasítását. Ebben az esetben a tartomány belsejében elhelyezkedő bázispont, és az adott függvény Taylor-polinomjának megfelelő számú együtthatóját kell eltárolnunk a memóriában. A Taylor-polinom egy tagjának kiszámítását egy lépésben elvégezhetjük a "tsm" utasítással, így a függvény helyettesítési értéke is könnyen és gyorsan számítható.

5.2. Eredmények

Az *5-2. táblázat* a három, a dolgozatban bemutatott tervezési módszerrel készült mikroprocesszor legfontosabb konstrukciós szintű jellemzőit foglalja össze, az *5-3. táblázat* pedig tartalmazza az egyes mikroprocesszorok szintézise során a logikai szintézis szoftver által szolgáltatott információkat.

	Taylor_FXD	**WMPCM**	**LAZARUS v1.4**
Gépi utasítások száma	27	26	58
Speciális funkció / utasítás	Függvénytani feladatok gyors megoldására alkalmas utasításkészlet.	Átlagos bonyolultságú mosógép vezérlésére alkalmas különleges utasítások. Pl.:	Általános célú 8 bites vezérlő. Különleges utasítások: szorzás, maradékos osztás

		„vízbe 0/1”: szivattyú indítása / leállítása	
Belső regiszterek	32x32 bit általános célú regisztertömb, 4x32 bit speciális célú regiszter a "tsm" utasításhoz	8x9 bit általános célú regisztertömb	32x8 bit általános célú regisztertömb
Stack	-	4x9 bit állapot-stack	0-126x8 bit adat-stack, 0-126x24 bit állapot-stack
Utasításmutató	10 bit, max. 1024 utasítás	8 bit, max. 256 utasítás	15 bit, max. 32.768 utasítás
Adatmemória	4 kbyte szavanként címezhető lineáris címtartomány	-	8 kbyte lineáris címtartomány, 32 db 256 byte-os szegmens, 0. szegmens: 256x8 bit SCRATCHPAD memória
I/O portok	-	1x9 bit vízszint jelző, 1x9 bit hőfok jelző,1x1 bit szivattyúmotor vezérlőbit, 1x1 bit lúgszivattyú-motor vezérlőbit, 1x1 bit fűtőszál vezérlőbit, 1x2 bit mosógépmotor vezérlő, 1x1 bit centrifugamotor vezérlőbit	2x8 bit bemeneti, 2x8 bit kimeneti port
Aritmetika, számábrázolás	32 bites fixpontos számábrázolás: egészrész: 12 bit	Előjel nélküli egész	Előjeles kettes komplemens kódú és előjel nélküli egész aritmetika
Megszakítások	-	-	1 db szintérzékeny megszakítási vonal
Egyéb	-	-	Kivételkezelés (szoftveres megszakítás)
Tervezésre fordított idő	≈ 70 munkaóra	≈ 25 munkaóra	≈ 40-50 munkaóra

5-2. táblázat - Az új mikroprocesszorok konstrukciós jellemzői

		Taylor_FXD	WMPCM	LAZARUS v1.4	
Vezérlő típusa		μ-programozott	μ-programozott	huzalozott	μ-programozott
Erőforrás-igény	Spartan 3E slice	2559-3296	207-259	747-1217	599-1068
	Blokk RAM	2-0	3-0	7-0	7-0
Maximális működési frekvencia (MHz)		11-13	70-77	71-95	73-95
Átlagos művelet-végzési sebesség a maximális órajelfrekvencián (MIPS)		4-5	23-26	25.3	16.5

5-3. táblázat - Az új mikroprocesszorok szintézise során nyert információk

Az *5-3. táblázat* alapján látható, hogy az erőforrásigény és a maximális működési frekvencia is széles tartományban változhat egyazon eszköz esetén. Ez annak köszönhető, hogy a szintézis eljárás során előálló RTL szintű modell nagy rugalmasságot biztosít a logikai szintézis tekintetében (*4.2.1. pont, 4-6. táblázat*). A logikai szintézist végző szoftver konfigurációjától függ tehát, hogy az egyes RTL szintű komponensek (főként a memóriák és az állapotgépek) milyen FPGA erőforrás felhasználásával kerülnek megvalósításra.

6. ÖSSZEFOGLALÁS

A dolgozat a tárolt programú gépek vezérlő egységének, nevezetesen a véges állapotgéppel leírt, ún. huzalozott vezérlő és a különböző mikroprogramozott architektúrák megvalósítási lehetőségeinek, valamint az alkalmazásorientált mikrogépek modellezésének és szintézisének áttekintése alapján egy új architektúra leíró nyelvet (ARTL) ismertet, amely egyaránt alkalmas speciális célú funkcionális egységek és tárolt programú gépek viselkedésének modellezésére. Kifejlesztésre került továbbá egy, az új leíró nyelvhez illeszkedő elvont mikroprocesszor modell, amely tetszőleges alkalmazásorientált utasításkészlet implementációjára alkalmas. Az így létrehozott mikroprocesszor típus tervezésének megkönnyítése érdekében elkészült egy újelvű, több szoftverkomponensből álló, C++ nyelven implementált tervezési keretrendszert, amely az elvégzett vizsgálatokkal igazoltan hatékony eszköz az ARTL leíró nyelv segítségével specifikált tárolt programú rendszerek hardver és szoftver összetevőinek megtervezéséhez és verifikációjához.

A szerző az általa kifejlesztett tervezési módszert, az ARTL leíró nyelvet, a folyamatvezérelt mikroprocesszor modellt és a tervezést segítő szoftverrendszert alkalmazhatósága ill. helyes működése, valamint a korábbi hasonló rendszerekkel szembeni előnye szempontjából részletesen tesztelte és segítségével több különböző mikroprocesszor tervet készített. Ezek közül a dolgozatban három, az utasításkészlet jellege és az összetettség szempontjából is különböző rendszer került bemutatásra, melyek igazolják, hogy a kifejlesztett tervezési eljárással egyaránt megvalósíthatók általános célú, egyszerű vezérlési feladatokra alkalmas és bonyolult adatfeldolgozási problémák hatékony megoldását célzó utasításkészletek.

A szerző igazolta, hogy a kifejlesztett tervezési eljárás és a hozzá elkészített szoftverrendszer a műszaki előnyökön túl magát a tervezési eljárást is felgyorsítja, így egyszerűbb mikroprocesszoros rendszerek a hardver és a szoftver együttes tervezése mellett a korábbi módszereknél egyszerűbben, ezért elvileg gyorsabban elkészíthetők és logikai szintézisre alkalmas formára hozhatók.

7. FELHASZNÁLT IRODALOM

[1] Gary D. Hachtel, Fabio Somenzi, *"Logic Synthesis and Verification Algorithms"*, University of Colorado, Kluwer Academic Publishers 1996, Dordrecht

[2] Coussy, P., Meredith, M., Gajski, D.D., Takach, A., *"An Introduction to High-Level Synthesis"*, Lab.-STICC, Univ. de Bretagne-Sud, France, IEEE Design & Test of Computers, vol. 26, no. 4., July-Aug. 2009

[3] Pong P. Chu, *"RTL Hardware Design Using VHDL"*, Cleveland State University, John Wiley & Sons, Inc. 2006, Hoboken, New Jersey, pp. 9-12.

[4] Németh Gábor, Horváth László: *Számítógép-architektúrák*. Budapest, Akadémiai Kiadó. 1993. pp. 14-19.

[5] Morris Mano, M., Kime, Charles R.: *Logic and Computer Design Fundamentals*, Pearson Prantice Hall, (2004), pp. 390-392.

[6] Ing-Jer Huang, *Member, IEEE,* Alvin M. Despain, *Member, IEEE, "Synthesis of Application Specific Instruction Sets"*, IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems, vol. 14, no. 6, JUW 1995, pp. 663-675.

[7] J. Sato *et* al., "*An integrated design environment for application specific integrated processor*" ICCD '91. Proceedings, 1991 IEEE Internationasl Conference on Computer Design: VLSI in Computers and Processors, 1991, pp. 414-417.

[8] C.-M. e. a. Kyung, "*Metacore: An application specific DSP development system*," in Proc. of the Design Automation Conference *(DAC),* Jun. 1998. pp. 800-803.

[9] Hoffmann, A.; Schliebusch, O.; Nohl, A.; Braun, G.; Wahlen, O.; Meyr, H.; "*A Methodology for the Design of Application Specific Instruction Set Processors (ASIP) Using the Machine Description Language LISA*", International Conference on Computer Aided Design, 2001 ICCAD 2001. IEEE/ACM pp. 625-630.

[10] Fauth, A.; Van Praet, J.; Freericks, M.; "*Describing Instruction Set Processors Using nML*", European Design and Test Conference, 1995, ED&TC 1995, Proceedings, pp. 503-507.

[11] Barbacci, M.R.; "*Instruction Set Processor Specifications For Simulation, Evaluation and Synthesis*", Design Automation, 1979. 16th Conference, pp. 64-72.

[12] Pees, S.; Hoffmann, A.; Zivojnovic, V.; Meyr, H.; "*LISA - Machine Description Language for cycle-accurate models of programmable DSP architectures*", Design Automation Conference, 1999. Proceeding. 36th, pp. 933-938.

[13] Hadjiyiannis, G.; Hanono, S.; Devadas, S.; "ISDL: "*An Instruction Set Description Language For Retargetability*", Design Automation Conference, 1997. Proceeding. 34th, pp. 299-302.

[14] Halambi, A.; Grun, P.; Ganesh, V.; Khare, A.; Dutt, N.; Nicolau, A.; "*EXPRESSION: A Language For Architecture Exploration through Compiler/Simulator Retargetability*", Design, Automation and Test In Europe Conference and Exhibition 1999. Proceedings, pp. 485-490.

[15] Hoffmann, A.; Kogel, T.; Nohl, A.; Braun, G.; Schliebusch, O.; Wahlen, O.; Wieferink, A.; Meyr, H.; "*A novel methodology for the design of application-specific instruction-set processors (ASIPs) using a machine description language*", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume 20, Issue 11, 2001. pp. 1338-1354.

[16] Barbacci, M.R.; "*A Comparison of Register Transfer Languages for Desribing Computers and Digital Systems*", IEEE Transactions on Computers, Volume: C-24, Issue: 2, 1975, pp. 137-150.

[17] Erényi István, Dr. Vajda Ferenc: *Mikroprocesszoros rendszerek fejlesztése*. Budapest, Műszaki Könyvkiadó. 1981. 330 - 334. o.

[18] *Digilent Nexys2 Board Reference Manual*, Revision: June 21, 2008, www.digilentinc.com

[19] SystemC osztálykönyvtár 2.2.0., http://www.systemc.org/downloads/standards, *2011.09.25.*

8. FÜGGELÉK

8.1. Elágazást tartalmazó folyamat megvalósítása folyamatvezérelt mikroprocesszor modellel

A *4.2. pontban* bemutatott mikroprocesszor modell nem alkalmas folyamaton belüli elágazás közvetlen megvalósítására, csak a folyamat felfüggesztésére és újraindítására van lehetőség. A folyamaton belüli elágazással egyenértékű működés az elágaztatni kívánt folyamat felbontásával és egy speciális feltételregiszter beiktatásával lehetséges. Tekintsük a *8-1. ábrán* látható folyamatot:

```
if ( Condition1 ) then
  1.: MAR = IR[7:0]
  2.: ACC = DataMemory[MAR]
  3.: if (ACC == 0x00) then RegA = 0x00
      else RegA = 0xFF end if
  4.: IP++
  5.: IR = ProgramMemory[IP]
end if
```

8-1. ábra - Elágazást tartalmazó folyamat

Mivel a folyamatba nem tudunk feltételvizsgálatot beiktatni, a *8-1. ábra* folyamata helyett a következő feltétel-folyamat párokat definiáljuk:

```
if ( PCR = 0x1 && ACC = 0x00 ) then
  1.: RegA = 0x00
  2.: IP++
  3.: IR = ProgramMemory[IP]
  4.: PCR = 0x0
end if
```

```
if ( PCR = 0x1 && ACC != 0x00 ) then
  1.: RegA = 0xFF
  2.: IP++
  3.: IR = ProgramMemory[IP]
  4.: PCR = 0x0
end if
```

.....

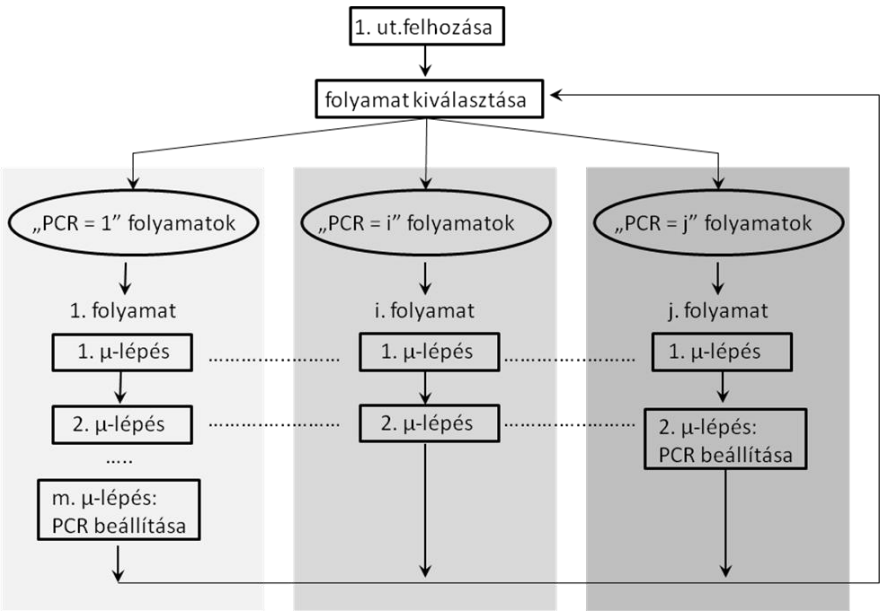
.....

```
if ( Condition1 ) then
  1.: MAR = IR[7:0]
  2.: ACC = DataMemory[MAR]
  3.: PCR = 0x1
end if
```

8-2. ábra - Elágazást tartalmazó folyamat felbontása

Az elágazást tartalmazó folyamatot úgy bontottuk fel, hogy az eredeti folyamat elágazást megelőző része, valamint a feltételvizsgálat különböző kimeneteihez tartozó eseménysorozatok egy-egy külön folyamatba kerültek. Ez utóbbiak végrehajtása csak akkor következhet be, ha a PCR (Process Control Register) értéke 0x1, amit az eredeti folyamat elágazást megelőző részéhez rendelt folyamat állít be. Hasonló módon oldható meg a több bájtos utasítások kezelése is.

Tegyük fel, hogy a PCR egy olyan több bites regiszter, amely a mikroprocesszor összes folyamatának feltételében szerepel. Ebben az esetben a folyamatkiválasztás (dekódolás) műveletet a hagyományos megoldásnál sokkal rugalmasabban tarthatjuk kézben. A PCR regiszter értékének egy adott folyamaton belüli beállításával már az adott folyamat végrehajtása alatt kiválaszthatjuk, hogy a következő ciklusban a többi folyamat közül melyek szerepeljenek a kiválasztható folyamatok között (*8-3. ábra*).



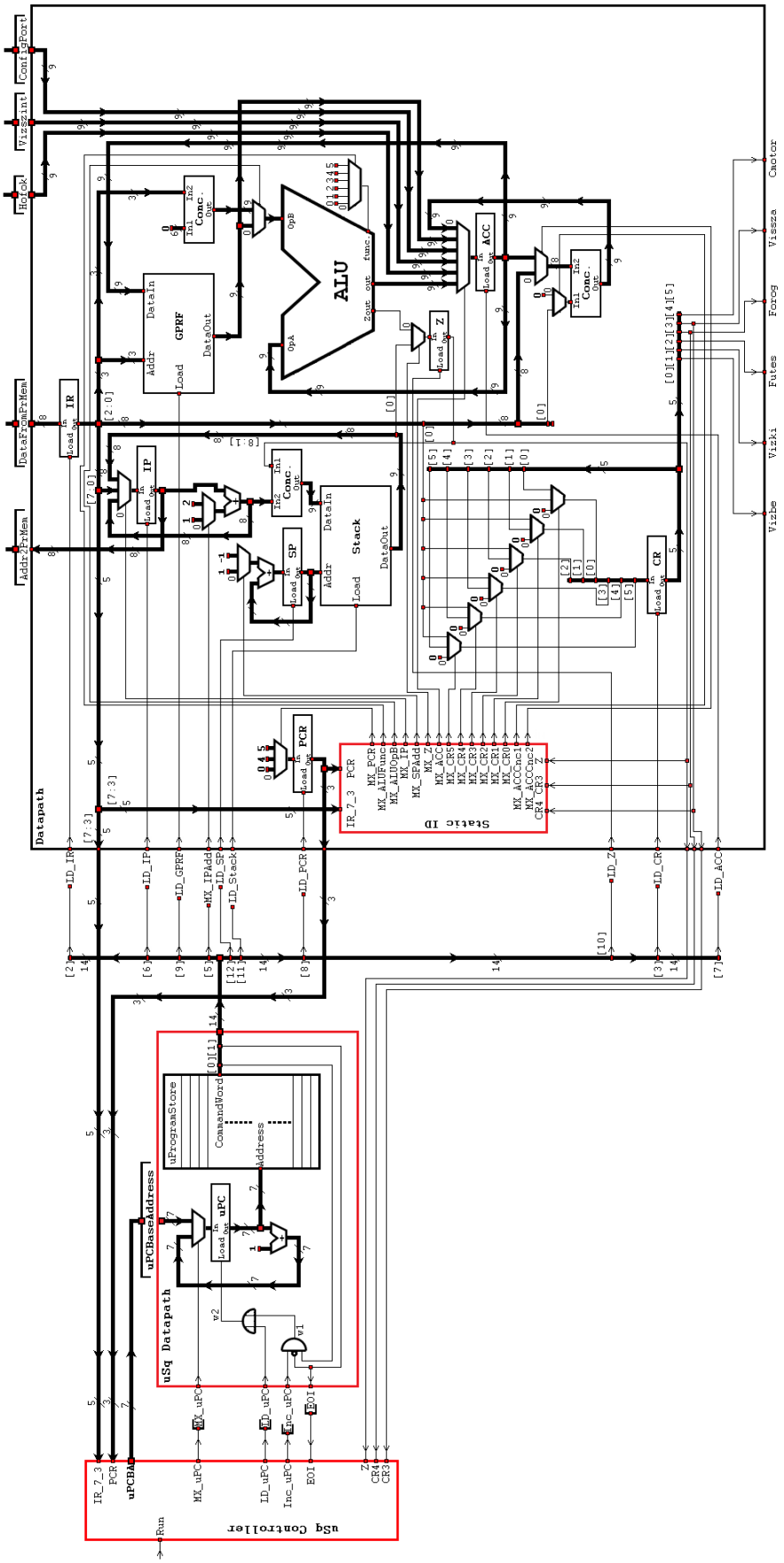
8-3. ábra - Rugalmas folyamatvezérlés a folyamatvezérelt mikroprocesszorban

8.2.A dolgozatban használt rövidítések

Rövidítés	Jelentés	Magyarázat
ARTL	Algorithmic Register Transfer Language	Újonnan kifejlesztett hardver leíró nyelv, mely a modellezett rendszer viselkedését regiszterek közötti adatáramlás algoritmikus jellegű megfogalmazásával definiálja.
A^2RTL	ASIP (Application-specific Instruction-set Processor) ARTL	~ leírás: Az ARTL nyelv kiterjesztése új nyelvi konstrukciókkal, melyek lehetővé teszik a folyamatvezérelt mikrogépek (PCM) viselkedésének hatékony leírását.
		~ modell: Egy folyamatvezérelt mikrogép A^2RTL modellje annak A^2RTL leírásából és egy kiegészítő, ún. MPD (Microprogram Definition) leírásból áll, melynek a mikroprogram megtervezésében van szerepe.
ADL	Architecture Description Language	Architektúra leíró nyelv. Általában regiszter-átviteli szintű, modellező nyelv.
ALU	Arithmetic-logic Unit	Aritmetikai-logikai egység. A tárolt programú gépek műveletvégző egységének legfontosabb elemeként hivatkozik rá az irodalom. Feladata - ahogy neve is mutatja - aritmetikai és logikai műveletek elvégzése.
ASIP	Application-specific Instruction-set Processor	Olyan tárolt programú mikroprocesszor, melyet utasításkészlete egy bizonyos szoftver vagy egy bizonyos problémakörben előforduló problémák hatékony megoldására teszi alkalmassá.
DSP	Digital Signal Processor/Processing	Digitális jelfeldolgozó processzor, vagy digitális jelfeldolgozás.
FPGA	Field Programmable Gate Array	A felhasználás helyén programozható logikai áramkör.
FSM	Finite State Machine	Véges állapotgép.
LUT	Look Up Table	Általános célú FPGA erőforrás. Tetszőleges 4 vagy 6 bemenetű, egy kimenetű logikai függvény megvalósítására alkalmas funkcionális egység.
MPD	Microprogram Definition	A folyamatvezérelt mikroprocesszorok (PCM) mikroprogramjának leírására szolgáló nyelv.
PCM	Process-controlled Machine/Microprocessor	Folyamatvezérelt mikroprocesszor. Tetszőleges utasításkészlet implementációjára alkalmas, FPGA technológiára optimalizált elvont mikrogép modell.
RTL	Register-transfer Level	Regiszter-átviteli szint. Eredetileg mikroprocesszorok műveletvégző egységének megtervezéséhez kifejlesztett elvonatkoztatási szint, mely a rendszert kisebb-nagyobb, egymással kommunikáló

Rövidítés	Jelentés	Magyarázat
		funkcionális egységek halmazaként írja le.
SyMGenPCM	SystemC Model Generator for Process-controlled Machines/Microprocessors	C++ nyelven implementált alkalmazás, mely egy folyamatvezérelt mikrogép A^2RTL leírásából annak viselkedési szintű SystemC modelljét képes előállítani.
VHDL	Very High Speed Integrated Circuit Hardware Description Language	A VHDL egy olyan általános hardvermodellező nyelv, amely a digitális áramkörök különböző elvonatkoztatási szinteken történő, egységes leírására alkalmas.

8.3.A WMPCM mikroprocesszor RTL szintű kapcsolási rajza



8.4.A LAZARUS v1.4 mikroprocesszor RTL szintű kapcsolási rajza