Műszaki Tudományos Közlemények vol. 21. (2024) 97–106.
DOI: Magyar: https://doi.org/10.33895/mtk-2024.21.15
Angol: https://doi.org/10.33894/mtk-2024.21.15

BEÁGYAZOTT RENDSZEREK ADAPTÍV ALKALMAZÁSÁNAK VIZSGÁLATA

EXAMINATION OF EMBEDDED SYSTEM ADAPTIVE CONFIGURABILITY

Vásárhelyi József,[1] Batók Roland,[2] Drótos Dániel[3]

[1] *Miskolci Egyetem, Gépészmérnöki és Informatikai Kar, Automatizálási és Infokommunikációs Intézet. Miskolc, Magyarország, jozsef.vasarhelyi@uni-miskolc.hu*

[2] *Miskolci Egyetem, Gépészmérnöki és Informatikai Kar, Automatizálási és Infokommunikációs Intézet. Miskolc, Magyarország, roland.bartok@uni-miskolc.hu,*

[3] *Miskolci Egyetem, Gépészmérnöki és Informatikai Kar, Automatizálási és Infokommunikációs Intézet. Miskolc, Magyarország, daniel.drotos@uni-miskolc.hu*

Abstract

Field Programmable Gate Arrays or FPGA circuits have been the focus of applied research since their appearance. Following the introduction of these circuits in 1980, their year-by-year development and the expansion of application areas have opened up new areas of research. With the appearance of system-on-chip, SOC (System on Chip) solutions and adaptive processing units, they occupy an important role in industrial applications. The question is: when, why and what hardware is to change dynamically during system operation? Whether it is necessary to expand computing resources by expanding new circuit blocks or by inserting soft-core processor(s), this can be determined by examining the computing load of the embedded system during operation. This article presents the tests that were performed on the processors of the embedded systems, in order to determine the computational load of the processor and how the interrupt handling can be accelerated. The article also presents the examination of fuzzy interpolation.

Keywords*: embedded systems, dynamic function exchange, FPGA, robot control, fuzzy interpolation.*

Összefoglalás

A programozható, logikai kapcsolómátrixok vagy FPGA- (Field Programmable Gate Array) áramkörök a megjelenésük óta az alkalmazott kutatások középpontjába kerültek. Az áramkörök 1980-as bevezetése óta évről évre történő fejlődésük és az alkalmazási területek bővülése új kutatási területeket nyitott meg. A *rendszer a lapkán* (SOC = System on Chip) megoldások és az adaptív feldolgozóegységek megjelenésével a hardver adaptív módosítása már ipari alkalmazásokban is megjelenik. Kérdés, hogy működés közben mikor kell, mikor szükséges, mikor érdemes módosítani a számítási algoritmust megvalósító áramkört? Kell-e bővíteni számítási erőforrásokat újabb áramköri tömbök bővítésével, vagy lágy magos processzor(ok) beillesztésével? Ezt a számítási terhelés működés közbeni vizsgálatával lehet megállapítani. Jelen cikk bemutatja azokat a vizsgálatokat, amelyeket a beágyazott rendszerek processzorain végeztünk el, annak érdekében, hogy megállapítható legyen a processzor számítási terhelése, a megszakításkezelés gyorsítása. A cikkben bemutatásra kerül a fuzzy interpoláció vizsgálata is, annak érdekében, hogy eldönthető legyen, milyen hardvert kell alkalmazni, amikor a robot irányítását fuzzy interpolációval valósítjuk meg.

Kulcsszavak*: beágyazott rendszerek, dinamikus függvény csere, FPGA, robotirányítás, fuzzy interpoláció.*

1. Bevezetés

A programozható, logikai kapcsolómátrixok vagy FPGA- (Field Programmable Gate Array) áramkörök az alkalmazott kutatások középpontjába kerültek már megjelenésük pillanatában. Az alkalmazási területek, amelyek annak idején még a kétszintű logikai hálózatok megvalósítására korlátozódtak, mára a digitalizáció korszakában a telekommunikáció, valós idejű beágyazott rendszerek, mesterséges intelligencia, képfeldolgozás és -működés közben megvalósuló, módosítható hardver (adaptív működés) területeken alkalmazhatók.

A beágyazott rendszerek egyik fontos paramétere a disszipált teljesítmény. Ezen tervezési paraméter csökkentése több módszerrel is lehetséges, például a mikrovezérlő frekvenciájának csökkentésével, viszont ez csökkenti a számítási sebességet is. A legjobb megoldás, amikor vizsgáljuk a processzor terheltségét, és ennek alapján végezzük el a szükséges módosításokat. A megoldások között lehetséges a megszakítási mechanizmus újfajta megközelítése vagy az irányító algoritmus vizsgálata és hardveres implementációja. Ez pedig a következő kérdéseket veti fel: kell-e bővíteni számítási erőforrásokat újabb áramköri tömbök bővítésével vagy lágy magos processzor(ok) beillesztésével?

Jelen cikk az FMTÜ 2024 konferencián elhangzott plenáris előadás bővített változata. Bemutatja a beágyazott rendszerekben használt processzor számítási terhelésének vizsgálatát, majd két példa bemutatásával – megszakításkezelés és fuzzy interpoláció állapotgép-megvalósítással – szemlélteti az adaptív működés lehetséges alkalmazását.

2. Irodalom

2.1. Processzorterhelés-vizsgálatok

A Neumann János által 1945-ben megjelentett, "The First Draft of the Report on the Edvac" [1], megfogalmazott számítógépes modellnek az évek során rengeteg cikk fogalmazta meg minden előnyét és hátrányát. A legnagyobb gondot az okozta, hogy a processzorgyártók elhagyták a Neumann által meghatározott modellben az adatok tárolásánál használt plusz jelzőbitet, amelynek az lett volna a feladata, hogy jelezze a tárolt adat programkódját vagy információját (adat). A Neumann-elvek a következők [2]:

– teljesen elektronikus felépítésű számítógép,
– kettes számrendszer alkalmazása,
– aritmetikai egység (univerzális Turing-gép),
– központi vezérlőegység,
– belső program- és adattárolás [2].

A Neumann-modell működése akkor a leghatékonyabb, amikor egy processzor egyetlen feladatot hajt végre [3]. Ez a megközelítés általában a beágyazott rendszerekre igaz. Egy beágyazott rendszerben a processzornak jól meghatározott feladata van, főleg, amikor a működéséhez nem használ operációs rendszert. A modellel akkor jelentkeznek a gondok, amikor a feladatok (taskok) száma különbözik a gépben jelen lévő processzorok számától. Sajnos, a szoftverfejlesztők még mindig ragaszkodnak a klasszikus Neumann-modellhez, ezért az operációs rendszer elrejti a hardverréteget a felhasználó elől, ezért úgy tűnik, mintha a processzor rendelkezésre állna minden task számára [4].

Az FPGA-technológia megjelenése óta kétfajta processzortípusról beszélünk: lágy magos és kemény magos. Az előbbi leírása hardver leíró nyelven történik (VHDL vagy Verilog), és beillesztik az FPGA-tervbe, míg az utóbbit integrálják a félvezető lapkára.

A beágyazott rendszerek számára, amelyeket kis erőforrással rendelkező FPGA-áramkörökkel valósítanak meg, különösen fontos a processzor vagy mikrovezérlő terhelésének vizsgálata az alkalmazói program végrehajtása során.

Programozható logikai mezők (FPGA) alkalmazása esetén különböző módszerekkel illeszthetünk be egy lágy magos processzort. A leggyakoribb az előregyártott (IP) alapú beillesztés, de használható a regisztertranszferszint- (RTL-) módszer is. Az RTL-lágymagosprocesszorhasználat azért előnyös, mivel ily módon a központi egység (CPU) belső jelei hozzáférhetők, és a processzorterhelés-vizsgálat megvalósítható. Ezzel ellentétben az IP-alapú tervezés esetén a processzor belső jelei nem elérhetők, így a számítási terhelésvizsgálat nem lehetséges.

A processzor számítási teljesítményének növelése FPGA-alapú rendszerek esetében egyéb kisegítő, gyorsító áramkörök, társprocesszorok használatával lehetséges. Ezen áramkörök működés közbeni beillesztése a dinamikus függvénycserével lehetséges, ily módon a processzor számítási terhelése csökkenthető.

A processzor működés közbeni terhelésvizsgálata a rendszer jeleinek vizsgálatán alapul. Az összes jel figyelésével megállapíthatók a hardverhibák [5]. A módszer teszteli a hardverműveleteket, és következtetéseket von le a gép hibás működésére vonatkozóan.

Egy másik vizsgálati módszert [6] a tervezési fázisban használtak, annak érdekében, hogy meghatározzák a szükséges hardvererőforrásokat, amelyek a szoftver optimális működéséhez szükségesek egy beágyazott rendszerben [7]

Zhao és társai egy szoftvereszköz segítségével határozták meg a futó program hibáit. Ehhez egy célrendszert terveztek [4].

A rendszertervezés és terhelésvizsgálat során a következő szempontokat kell figyelembe venni [8]:

- a mérési módszer független a szoftver működésétől, amelyet a beágyazott processzor végrehajt;
- a mérési módszer megvalósítása nem módosíthatja a mikroprocesszor architektúráját;
- a mérésnek valós idejű adatokat kell szolgáltatnia, azaz azonnal kell érzékelnie a rendszerben (processzorban) történő változásokat.

A fenti szempontok alapján megállapítható, hogy a processzor számítási terhelésének mérése független kell, hogy legyen a rendszer hardverétől.

2.2. Processzormegszakítás végrehajtásának folyamata

Amint említettük, a Neumann-modell akkor működik jól, ha csakis egyetlenegy feladatot lát el a mikroprocesszor. A beágyazott rendszerekben a processzornak egyszerre több feladatot is el kell végeznie. Ezt pedig csak úgy tudja, ha a feladatok között megosztja a processzoridőt. Ez pedig csak megszakításkezeléssel lehetséges [9].

A megszakítás kiszolgálása (Interrupt Service Routine – ISR) mint plusz feladat jelenik meg a főprogram mellett. A processzornak pedig ezt a feladatot is el kell végeznie [10].

Az ISR-t a processzor nem tudja azonnal végrehajtani, mivel a megszakításelfogadásban van egy lappangási idő, amely a megszakításkérés és a megszakítás kiszolgálása között eltelik. Ennek több oka is van:

- a processzor kritikus kódrészletet hajt végre – megszakításkérés letiltása;
- a processzor egy elsőbbséggel rendelkező task-ot (feladatot) hajt végre;
- a processzor egy hosszú utasítást hajt végre, amelyet nem lehet megszakítani.

A megszakítás elfogadásának időtartama is változhat. Ennek egyik oka, hogy az utasításátlapolásos processzoroknál ki kell üríteni a pipeline tartalmát. Ennek időtartama a pipeline utasításainak típusától függ. A másik ok az, hogy az ISR (megszakítás kezelő program) után helyre kell állítani az előző program állapotát. A mentés időtartama attól függ, hogy az ISR hány regisztert használ. A megszakítás elfogadásának időtartama tehát nem állandó, a késés ingadozik.

Amikor egy processzor megszakítási jelet kap, és ha a megszakítás engedélyezett, akkor a processzor felfüggeszti a futási folyamatot, és megkezdi a kivétel kiszolgálását. A megszakítási kezelő rutin (ISR) végrehajtása után a processzor visszatér a felfüggesztett feladathoz.

ARM-processzorok esetében az ISR kivételes eseményként kezelendő, és az ARM-dokumentumok a megszakítást kivételnek írják le [11]. Ezt az alábbiakban ismertetjük röviden:

- Ha ISR-esemény következik be, és a kivételkezelést engedélyeztük, a processzor felfüggeszti az aktuálisan végrehajtott utasítást (az úgynevezett hosszú utasításokat eldobja, és visszatérés után újraindítja a végrehajtásukat),
- majd a kontextusregisztereket (8 regiszter, mindegyik 32 bites) menti a veremmutató által megadott veremtárba (ARM MSP – Main Stack Pointer vagy PSP – Process Stack Pointer esetén). A mentett regiszterek a következők: xPSR (Program Status Register), PC (Program Counter – tartalmazza a visszatérési címet, LR (Link Register R14) és regiszterek: R12, R3, R2, R1, R0.
- Ezt követően a processzor handler üzemmódba kapcsol.
- Ezután a programszámlálóba (PC) kerül az ISR-kezdőcím, és az LR betölti a visszatérési címet.
- A következő lépésben az ISR száma betöltődik az IPSR-be (Interrupt Program Status Register).
- Végül megkezdődik a megszakításkezelő rutin végrehajtása.

A fentiekben leírt folyamat felemészti a processzoridőt. Fontos figyelembe venni az ISR-kéréstől a kivételkezelés elindulásáig eltelt időt. Ha a megszakítás ciklikusan ismétlődik, akkor az ISR válaszideje megnövekszik a megszakításkezelés elkezdésének lappangási idejével. A megszakításkezelés kiszolgálásának ideje szempontjából kritikus külső események késleltethetik a kiszolgáló rutin indulását.

Mennyi idő szükséges a regiszterek mentésének elvégzéséhez („rezsiköltséghez”), és a program PUSH/POP utasításai végrehajtásához? Ilyenkor a processzor a regisztermentés/-visszaállítás címével foglalkozik (veremtárműveletek). Ha a veremtár külső memóriában van, akkor ez a „rezsiidő” hosszabb lesz (külsőmemória-műveletek végrehajtása).

Kiszámítható, egy adott processzorra, az adott órajel frekvenciával, mekkora lehet a megszakítások maximális ismétlési gyakorisága:

$$F_{MaxInt} = F_{CPU} / (C_{ISR} + C_{Overhead}), \quad (1)$$

ahol az F_{MaxInt} a megszakítás maximális ismétlési gyakorisága, az F_{CPU} a vezérlő órajel frekvenciája, a C_{ISR} a megszakításkezelő rutin (ISR) elvégzéséhez szükséges órajel ciklusainak száma, a $C_{Overhead}$ pedig a regiszterek mentésére és visszaállítására (azaz a programállapot mentésére és visszaállítására) fordított órajelciklusok száma.

Gyakori megszakításkérés teljesítésével, a processzor az ISR futtatását végzi. Ezért kevesebb időt fordít a fő program végrehajtására. Az ISR teljes végrehajtásához szükséges idő (U_i):

$$U_i = (F_i (C_{ISR} + C_{Overhead})) / F_{CPU}, \quad (2)$$

ahol F_i a megszakítások gyakorisága.

A fentiekből pedig az következik, hogy a fő program végrehajtása látszólag sokkal alacsonyabb órajel-frekvenciával történik:

$$F_{main} = (1 - U_i) * F_{CPU}, \quad (3)$$

ahol F_{main} a fő program végrehajtásának látszólagos gyakorisága.

3. Processzor terhelésének vizsgálata

A processzor működése során megkülönböztetünk két állapotot: amikor a processzor futtatja a programokat – ezt nevezzük terhelt állapotnak; míg lehetségesek olyan állapotok, amikor a processzornak nem kell aktívnak lennie – ezt nevezzük terheletlen állapotnak. A terheletlen állapotnál megkülönböztetjük:

- az alvó üzemmódot (sleep state), amikor a processzor nem végez számítási műveletet, és úgy tűnik mintha ki lenne kapcsolva – a processzor disszipált teljesítménye nagyon kicsi;
- a tétlen állapot (idle state), amikor a processzor a program végrehajtása közben várakozik valamilyen eseményre.

A processzor aktív állapotában, a szoftveraktivitástól függően ,a rendszer különböző részei nagyobb mértékben aktívak, mint a processzor egyéb részei, például az aritmetikai egység – amikor matematikai számításokat végez, vagy a memóriaillesztő egység a ki-be meneti műveletek elvégzése során. Ennek alapján a következő processzorműködési kategóriákat különböztetjük meg:

- alap (terheletlen állapot);
- számításintenzív,
- ki-be műveletvégzés.

Ezt a fajta osztályozást a processzor különböző részeinek tényleges terhelésének mérésével végeztük. A mérések segítségével azonosítottuk az intenzívebben használt processzorrészeket.

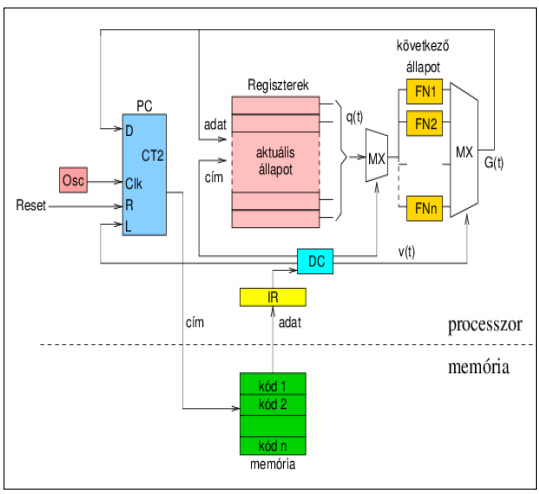
3.1. Mérési módszer

A mérési módszert az alkalmazott processzor belső felépítése segíti (**1. ábra**). A processzor egy olyan állapotgép, amelyben a tárolóelemek (az ábrán: Regiszterek) biztosítják a rendszer aktuális állapotának megőrzését. Az aktuális állapot (q) az órajel ütemében változik. Az új állapot értékét az úgynevezett gerjesztőhálózat (az ábrán FN1, FN2, stb.) állítja elő, az aktuális állapot és az esetleges külső jelek (v) függvényében. A v, G, q értékei az idő t függvényei (ebben az esetben az állapotot szabályozó órajel változik). Mivel a tárolt érték a gerjesztési értékké válik egy órajel-késleltetés után **[8]**, ezért:

$$q(t) = G(t-1) \quad (4)$$

A külső jelek befolyásolják a G függvény értékeit, így nem szükséges közvetlenül tesztelni őket. A G értékek egy órajel-késleltetéssel kerülnek át a q (következő állapot – next state) értékekre, így a mérés akár a G, akár a q jelek vizsgálatával elvégezhető.

A CPU egyes összetevőinek működését a hozzájuk tartozó memória értékének változása jelzi. Tehát a vizsgálat szempontjából releváns q jelek mérhető jellemzője, a q jelek változási gyakorisága, vagyis a jel frekvenciája. Az osztályozás azonban nem a jelfrekvenciák pillanatnyi értékein alapul, hanem a frekvencia időbeli változásán, a frekvencia időfüggvényén: $fr(t)$.



1. ábra. *A processzor felépítésének tömbvázlata*

3.2. A tesztrendszer ismertetése

Az alkalmazás, amelyen a processzorterhelés mérését elvégeztük, egy olyan FPGA-áramkörön megvalósított beágyazott rendszer, amely egy, a Nap mozgását követő eszközt vezérel.

Az alkalmazás egy napelemrendszer (**2. ábra**), amely a Nap mozgásának függvényében mindig az optimális működést állítja be, azaz a maximális felületet mutatja a Nap felé, amely merőleges a napsugarakra.

A vezérlőszoftver az idő és a földrajzi koordináták függvényében kiszámítja a napsugarak pillanatnyi tájolását (vízszintes és északi szögét). A kiszámított koordináták alapján beállítja a napelempanelt, hogy merőleges legyen a napsugarakra, azaz a motorok vezérlésével mozgatja a panelt. A számítás eredményétől függő mozgatás (panelek beállása) során a szoftver nagyon sok I/O-műveletet végez, ugyanez a helyzet a kezelőfelület kijelzőjének a frissítése során is.

3.3. Mérési eredmények

A továbbiakban a mérési eredményeket a **[8]** cikk alapján ismertetjük.

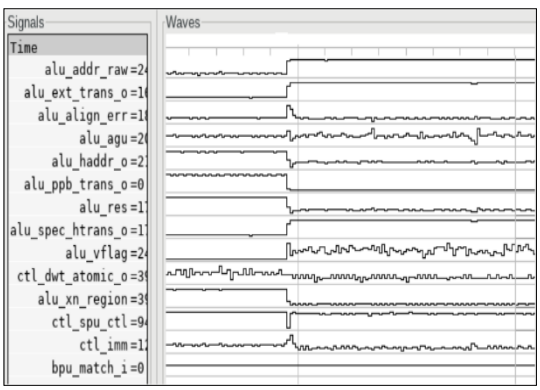
A rendszer modellezését FPGA-n megvalósított beágyazott rendszerrel végeztük, RTL-alapú ARM M0+ processzorral. A vizsgálat során a rendszermodellt hardverszimulátor segítségével futtatjuk, amely az áramköri elemek ki- és bemenetein megjelenő jelek értékeit elmenti egy úgynevezett „dump”-állományba, amelynek a tartalmát megfelelő szoftverekkel vizsgáltuk.

Az $fr(t)$ frekvenciajel időbeli változását egy dedikált szoftver állítja elő, amely feldolgozza a szimulációs eredmények kimeneti állományát. A feldolgozó szoftverbe épített algoritmus megfelel a frekvenciaméréshez használt digitális áramkör működésének. A hardverszimuláció során nagy adatmennyiség keletkezik, ezért a feldolgozószoftverbe beépítettünk egy szűrő algoritmust, amely a vizsgálat szempontjából nem releváns adatokat el tudja távolítani. A modell leírási módja miatt például több jel duplikáltja valamely másik jelnek, így az értékeik azonosak. Ezeknek a jeleknek az eltávolításával a feldolgozandó adatmennyiség csökkenthető. Az eredmények grafikus ábrázolásával ellenőrizhető, hogy mely frekvenciajel-változások mutatnak érzékelhető eltérést valamilyen jellemzőben abban az időpontban, amikor az alkalmazásszoftver megváltoztatja a processzor számítási állapotát (terhelését), azaz amikor a számítási algoritmus elindul vagy leáll. A **3. ábra** néhány mért jel változását mutatja be az alkalmazói program számítási részének elindulása pillanatában. A **3. ábrán** látható jelváltozást vizsgálva például az *alu_ext_trans_o* jel esetében észrevehető, hogy jelentős amplitúdóváltozást mutat, ami azt jelenti, hogy a processzor elkezdi a számítást.

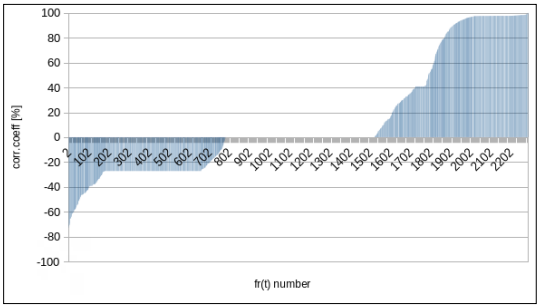
A modellezés és szimulációs eredmények alapján kiválaszthatók azok a jelek, amelyek alkalmasak a központi egység számítási terhelésének mérésére. A modell esetében ismert, hogy mikor végez az alkalmazói szoftver olyan számítást, amely a CPU terhelését okozza. Az $fr(t)$ függvények statisztikai jellemzőinek vizsgálatával megállapítható, hogy melyik jel mutat nagyobb mértékű korrelációt a detektálandó processzorviselkedéssel.

A **4. ábra** mutatja a vizsgált processzorjeleknek a számítási üzemmóddal való korrelációjának eloszlását. Látható, hogy a jelek egy része jól korrelál az üzemmóddal, így alkalmas lehet a számítási időszakok jelzésére.

2. ábra. *A teszt rendszerképe*



3. ábra. *Processzorterheltség-vizsgálat jeleinek változását mutató grafikus ábrázolás*



4. ábra. *A CPU jeleinek korrelációja a számítási üzemmóddal*

Ezek közül a jelek közül azt célszerű a méréshez felhasználni, amelynek a változását a mérést végző áramkör a legkönnyebben tudja detektálni.

A beágyazott rendszer struktúrájának módosítása után a kiválasztott jeleket kivezettük a processzorból. Ezen jeleket pedig a mérőrendszer segítségével elemeztük. A jelek frekvenciamérése után egy amplitúdóváltozást figyelő modul segítségével megállapítható a jelváltozás amplitúdója. Az amplitúdóváltozásokból pedig következtethető a processzor számítási terhelése.

4. Megszakításkezelés gyorsítása

A megszakításkezelés kiszolgálására összeállítottuk az **5. ábra** szerinti kísérleti rendszert. A kísérletben ARM-lágymagos- (ARM M0) processzorokat) implementáltunk az FPGA-áramkörben. A kísérletben a processzoroknak a következő feladatot kell megoldani: az időzítő periféria megszakítást generál egy adott frekvencián (10 kHz a kísérleti rendszerben), a kivételkezelő program számolja a megszakítások számát, azaz növeli egy számláló értékét (szoftveróra). A fő program megjeleníti a számláló értékét egy kijelzőn.

A P3 processzor hagyományos módon működik. Kezeli a megszakítást, azaz időzített feladatot hajt végre, majd a számláló értékét kiküldi a kijelzőre. A P3 processzoron futó hagyományos megoldás futási paramétereit hasonlítjuk össze egy, a feladatot párhuzamosan (P1 és P2) végrehajtó rendszerrel. A P1 processzor a fő programot futtatja, míg a P2 processzor a megszakításkezelést hajtja végre. Amikor a kivételkezelés megtörtént, a P2 processzor „sleep”-, azaz alvó állapotba kerül.

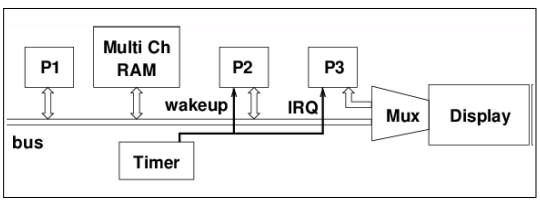
A megszakításjel hatására a P2 processzor éled és végrehajtja a kivételkezelő programot.

Ennek a processzornak a működését úgy terveztük meg, hogy elkerülje a szokásos megszakításkezelési folyamat végrehajtása során felmerülő problémákat, például a kiszolgálás elindulásához szükséges idő ingadozását. Ezért az időzítő periféria jelét nem az ISR-jel-megszakítási kérelem bemenetére csatlakoztattuk. A processzor felfüggesztett állapotban van (alvó üzemmódban). Az időzítő által generált perifériajel csatlakozik a processzor „ébresztését” indító (WakeUp) bemenetre. Amikor a jel indítja a processzort, akkor megtörténik az ISR kiszolgálása, amely ezúttal a P2 fő programjaként szerepel. Mivel az ébredés az ISR-jel felfutó élére történik (a processzor felfüggesztése a program elején van), azonnal megkezdődik az „ISR” kiszolgálása. Nincs szükség a megszakításelfogadás folyamatára. Nincs szükség a fő program állapotának mentésére, mert a processzornak az ISR a fő programja. Ezért a $C_{Overhead} = 0$ az (1) és (2)-es egyenletekben.

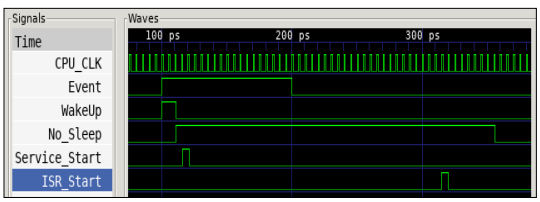
Az „ISR”-kiszolgálás végén a program visszaugrik a program elejére, ahol egy utasítás felfüggeszti a processzort, így az alvó üzemmódban várja a következő ISR-jelet. A periféria megszakításjelét a processzormegszakítás kezelő áramköre tárolja a szolgáltatás megkezdéséig.

A párhuzamos megoldás szimulációjának eredményét, összehasonlítva a hagyományos megoldással, a **6. ábra** mutatja. Az esemény- (Event-) jel a periféria megszakítási kérését jelzi, ez az impulzus az időzítő perifériáról érkezik 10 kHz-es frekvenciával.

A *WakeUp* jel ébreszti a P2 processzort. A *No_sleep* jelet a processzor generálja, és jelzi, hogy folytatja a programvégrehajtást. A *Service_Start* jelet a P2 processzor az „ISR-kiszolgálás” első utasításával generálja, amely jelzi, hogy a szolgáltatás elindult.



5. ábra. *Megszakításkezelés párhuzamosításának tömbvázlata*



6. ábra. *Megszakításkezelés párhuzamosításának idődiagramja*

Az *Event* jel hagyományos módon megszakítja a P3-processzor-programot. A P3 processzor ISR-funkciójának első utasítása generálja az *ISR_Start* jelet. Ezen indikátorjel segítségével összehasonlíthatjuk a két ISR kezdőpontját, az egyiket, amelyet a P2-n hajt végre, és a másikat, amelyet a P3-on hajt végre. Látható, hogy a kivételkezelési folyamat miatt (regiszterek mentése stb.) a P3 processzor később kezdi meg az ISR-folyamatot, mint a P2 processzor.

5. Fuzzy interpoláció

A valós környezetben mozgó robotnak különféle helyzetekben kell döntést hoznia, az egyes helyzetekre megfelelően kell reagálnia. A viselkedésalapú irányítás esetén a szakértő által megadott viselkedések érvényesek adott helyzetek esetén. A helyzeteket a robot az érzékelői által szolgáltatott adatok és a belső állapota alapján ismeri fel. Minden meghatározó körülményhez tartozik egy-egy viselkedés, amelyet a robot végrehajt **[12]**, **[13]** és **[14]**.

Abban esetben, amikor a viselkedésmodellek fuzzy logikával kerülnek megvalósításra, a fuzzy állapotgépnek köszönhetően az összes lehetséges viselkedés részt vesz a robot irányításában, de különböző mértékben. Azok a viselkedések lesznek a legnagyobb hatással a robot mozgására, reakcióira, amelyek az adott megfigyeléshalmazba legjobban illeszkednek **[12–14]**.

Mivel a valós környezetben mozgó robotot nagyon sokfajta hatás érheti, előfordulhat, hogy egy-egy megfigyeléshalmazhoz nem tartozik viselkedés. Ebben az esetben a robot döntésképtelenné válhat. Újabb fuzzy szabályok hozzáadásával ez javítható, de túl sok szabály számítása nagy számítási erőforrást igényel. Erre az esetre ad megoldást a fuzzy-szabály-interpoláció **[12–14]**.

Az interpolációs eljárásnak köszönhetően elég a szakértőnek a legmeghatározóbb szabályokat megadni, és a köztes esetekben az interpolációs eljárás fogja a kimeneti értékeket szolgáltatni **[12–14]**.

A szakértői tudásbázis használatával a rendszer akkor is közel megfelelően viselkedik, amikor nincs az adott helyzetre viselkedésszabálya. A fuzzy állapotgép a FIVE (Fuzzy Interpolation in Vague Environment) fuzzy interpolációs eljárást alkalmazza **[12–14]**.

A viselkedésalapú irányítás egyik alkalmazási példája az etorobotika, ahol a robot valamilyen élőlény viselkedését utánozza.

5.1. A FIVE-módszer

Az autonóm robotok döntéshozatali folyamata többféle érzékelőből származó információ és a belső állapot alapján történik. Ezek a bemeneti adatok képezik a fuzzy szabályok megfigyeléseit, más néven antecedenseit, míg a fuzzy szabályok kimeneteit konzekvenseknek nevezzük. Egy fuzzy szabály, „HA … AKKOR …”-formájú implikáció formájában írható fel. Az azonos kimenetet befolyásoló fuzzy szabályok összességét fuzzy szabálybázisnak nevezzük **[13][15][16]**.

A klasszikus fuzzy rendszerekben a stabilitás érdekében a szabályokat úgy kell kialakítani, hogy minden lehetséges bemeneti kombinációhoz tartozzon egy szabály. Ez biztosítja, hogy a szabályok teljesen lefedjék az állapotteret. Azonban a megoldandó feladat méretének növekedésével a szabályok száma exponenciálisan nőhet, ami jelentős számítási igényt eredményez. Továbbá, a nagyszámú szabály előállítása gyakran problematikus, mivel a szakértők nem mindig tudják teljes mértékben lefedni a problémát a matematikai bonyolultság vagy információhiány miatt. Ennek következtében a teljesen lefedő szabályrendszer helyett gyakran ritka szabálybázissal dolgozunk, amelyben előfordulhat, hogy egyes megfigyelésekhez nem tartozik szabály, így a rendszer kimenete bizonytalan lesz.

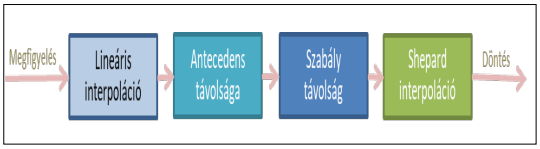
A ritka szabályrendszerek gyakoribbak, mint a teljes lefedést biztosító szabályrendszerek.

A fuzzy interpolációs módszerek megoldást kínálnak a ritka szabályrendszerek problémájára. Ezek a módszerek a hiányzó szabályokhoz tartozó döntéseket a meglévő szabályok alapján számítják ki. Ennek köszönhetően a szakértőnek elegendő csak a legfontosabb szabályokat megadnia, csökkentve ezzel a szabályok számát és a rendszer bonyolultságát, valamint az interpolációs módszertől függően a számítási igényt is.

A FIVE- (Fuzzy Interpolation in a Vague Environment) módszer a szabályokat egy homályos térben helyezi el, ahol értelmezhető a szabályok közötti euklideszi távolság. A szabályok közötti távolság alapján különböztethetők meg egymástól: nagyobb távolság nagyobb különbözőséget, míg kisebb távolság hasonlóbb szabályokat jelent.

A szabálytávolsággal súlyozott konzekvenseket felhasználva, a Shepard-interpoláció inverz távolságokkal számítva hozza létre a szabálybázis döntését. A FIVE-módszer számítási lépéseit az alábbiakban ismertetjük (lásd **7. ábra**):

– A megfigyelés a környezetből vagy a belső állapotból származó információ.



7. ábra. *A FIVE hardveres gyorsító számítási lépései*

– Lineáris interpoláció segítségével rendeli hozzá a megfigyeléshez azt az értéket, amely megadja, hogy az adott fuzzy halmaznak milyen mértékben tagja a megfigyelés értéke (μ).
– A szabályhoz tartozó antecedens és a megfigyelés μ értékének különbsége adja meg az antecedenstávolságot.
– A szabálytávolság az antecedenstávolságok euklideszi távolsága.
– A Shepard-interpoláció a szabálytávolságokat használja fel a döntés, azaz a konzekvens előállításához.

5.2. A FIVE-hardver felépítése

A hardveres megvalósítással szembeni legfontosabb követelmény a működés közbeni hangolhatóság és paraméterezhetőség. Ez azt jelenti, hogy az áramkör működése közben annak leállítása nélkül módosíthatók a FIVE paraméterei, beleértve az antecedens- és konzekvensértékeket.

Ennek érdekében a paramétereket regiszterekben tárolják, amelyekhez olyan felület készült, amely lehetővé teszi, hogy egy órajelciklus alatt megváltoztatható legyen az egyes modulok belső adata.

Az áramkör egyszerű csatlakoztatása a külvilághoz változtatható bitszélességű kimeneti és bemeneti buszokon keresztül történik. Ez a buszrendszer lehetővé teszi az AXI-interfésszel való kompatibilitást is, ami különösen hasznos a FIVE-hardver integrálásában a processzoros rendszerhez.

A skálázható felbontás moduláris felépítéssel és paraméterezhető bitszélességgel valósult meg. Az egyes modulok száma a szabályok által igényelt mennyiség szerint változtatható, így rugalmasan alkalmazkodik a rendszer követelményeihez.

Fontos szempont a kód hordozhatósága az FPGA-platformok között, így a Verilog nyelven írt kódban nem kerültek felhasználásra gyártó, specifikus elemek. Ez biztosítja, hogy a kód különböző platformokon is alkalmazható legyen.

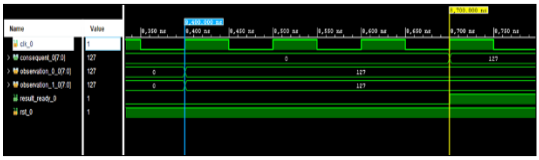
Az áramkör generálható az FBDL- [17] (Fuzzy Behavior Description Language) nyelvből. A modulok kapcsolódása és paraméterei az FBDL-nyelven leírt szabálybázisokból nyerhetők ki, ami egyszerűsíti a rendszer tervezését és implementációját.

A FIVE-módszer négy számítási lépésen keresztül képes az eredmények előállítására, amelyek két interpolációs számításból és két távolságszámításból állnak. Az adatfolyam feldolgozását a **7. ábra** szemlélteti, amely egy szabálybázis számítási lépéseit tartalmazza. Az ábrán látható, hogy a Shepard interpolációs modul kivételével az egyes modulok száma a szabálybázisban található szabályok, antecedensek és megfigyelések számától függ. A lineáris interpolációk számát a megfigyelések száma határozza meg, míg az antecedenstávolság-számítások száma az egyes szabályok antecedenseinek számától függ. Minden szabályhoz egy szabálytávolság-számító modul tartozik, és a Shepard-interpoláció számítása egyetlen modullal valósul meg szabálybázisonként.

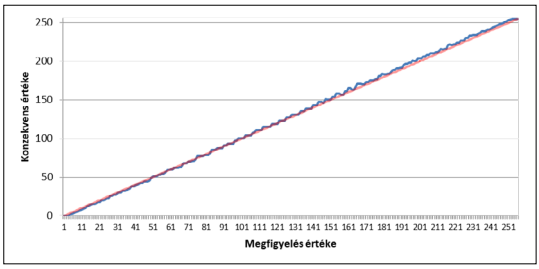
A lineáris interpoláció számítása a végső megvalósításban "Universe" néven szerepel. A kísérleti megvalósítás Vivado High Level Synthesis eszközzel is elkészült, az alap-μFRI-könyvtár [18] módosítás nélküli és optimalizált változatainak felhasználásával.

A FIVE FPGA megvalósítása egy szinkron számítási láncot képez. Az áramkör minden egysége a bemenő adatok megváltozására reagálva állítja elő a kimeneti értéket. A hardver változtatható bitszélességgel működik, alapértelmezés szerint 8 bites előjel nélküli, egész számokat használ a bemenetein és kimenetein. A Verilog paraméterezési lehetőségei révén a bitszélesség változtatható, és a paraméterek alapján az érintett sínrendszerek szélessége átszámítható. A számításhoz szükséges adatok szintén paraméterben adhatók meg. A Verilog kód az FBDL (Function Block Diagram Language) alapján készült, és a paraméterek segítségével az értékek megváltoztatása újragenerálás nélkül lehetséges.

Az egyes számítási egységek és a befoglaló egység is állandó és változó részekből épül fel. Azok a részek, amelyek száma a szabályok és szabálybázisok tulajdonságai alapján változik (szabályok száma, megfigyelések száma, antecedensek száma stb.), a kódban kiemelve, az FBDL-ből előállítható formában kerültek a Verilog kódba. A számítási egységek szerinti felosztás biztosítja a pipeline működést. Az egyes egységek tartalmaznak egy belső bemeneti és egy kimeneti tárolóregisztert, ahol a beérkező adat és az eredmény tárolható. A jelenlegi terv szerint minden egység egy órajelen belül ad eredményt, kivéve a Universe-egységet, amely két lépésben állítja elő az eredményt.



8. ábra. *A FIVE hardveres gyorsító késleltetése*



9. ábra. *A FIVE hardveres gyorsító hibája*

5.3. Eredmények

A rendszer szimulációs eredményei a Xilinx Vivado 2018.1 verzióval kerültek előállításra. A teljes számítási lánc késleltetése 4 órajel, ahogy az a **8. ábrán** látható. A rendszer előjel nélküli egész számokkal képes működni, és maximális működési frekvenciája 4 MHz. A FIVE-gyorsító pipeline-működésre képes, linearitási hibáját a **9. ábra** szemlélteti. A szimuláció alapján a legnagyobb abszolút eltérés a várt értéktől 6, ami a 8 bites tartományon 2,34%-nak felel meg.

A FIVE-módszer FPGA-n történő alkalmazásához egy Verilog hardver leíró nyelven készült váz is elkészült, amely alkalmas arra, hogy az FBDL (Fuzzy Behavior Description Language) alapján generálható hardverelemként gyorsítsa a FIVE-módszer számítását. Ez az elem alkalmazható önállóan vagy CPU-val társítva, gyorsító segédáramkörként. A FRI_FIVE-modulok összekapcsolásával összetett viselkedést megvalósító fuzzy viselkedésautomaták hozhatók létre.

A FIVE IP struktúrája lehetővé teszi a Xilinx Adaptive Platformon történő implementálását is, mint szoftveres környezetben működő hardveres gyorsítóegységet. Ebben az esetben a számítási egységek FIVE IP AXI-n keresztül csatlakoznak a processzoros rendszerhez, ahol az irányító rendszer szoftveres modulja végzi az adatgyűjtési és irányítási feladatokat. A rendszerről részletes leírást a **[19]** cikk tartalmaz.

Szakirodalmi hivatkozások

[1] J. Neumann:, *First Draft of a Report on the EDVAC*. Moore School of Electrical Engineering, University of Pennsylvania, 1945.

[2] W Aspray: *John von Neumann and the Origins of Modern Computing*. MIT Press, Cambridge, 1990, p. 34-48.

[3] Drótos D., Vásárhelyi J.: *Interrupt Driven Parallel Processing*. In: 20th International Carpathian Control Conference (ICCC), Krakow-Wieliczka, Poland, 2019, 1–4. https://doi.org/10.1109/CarpathianCC.2019.8765909

[4] G. Zhao, S. Hassan, Y. Zou, D. Truong, T. Corbin: *Predicting Performance Anomalies in Software Systems at Run-Time*. ACM Transactions. Software. Engineering. Methodology. 30/3. Article 33 (July 2021). https://doi.org/10.1145/3440757

[5] H. Sayadi, N. Patel, S. Manoj P.D., A. Sasan, S. Rafatirad, H. Homayoun: Ensemble Learning for Effective Run-Time Hardware-Based Mal Ware Detection: A Comprehensive Analysis and Clas-sification, 55th ACM/ESDA/IEEE Design Automation Conference (DAC), 2018, pp. 1-6, https://doi.org/10.1109/DAC.2018.8465828

[6] J. Muskens, M. Chaudron: *Prediction of Run-Time Resource Consumption in Multi-Task Component-Based Software Systems*. In: Crnkovic I., Stafford J.A., Schmidt H.W., Wallnau K. (eds) *Component-Based Software Engineering*. CBSE 2004. Lecture Notes in Computer Science, vol. 3054. Springer, Berlin, Heidelberg 2004. https://doi.org/10.1007/978-3-540-24774-6 16

[7] B. H. Fletcher: *FPGA Embedded Processors*. In: Embedded Training Program Embedded Systems Conference San Francisco 2005 ETP-367, 2005, 37.

[8] D. Drótos, J. Vásárhelyi: *Microprocessor Load Measurement in an Embedded System*. 24th International Carpathian Control Conference (ICCC), Miskolc–Szilvásvárad, Hungary, 2023. 110–113. https://doi.org/10.1109/ICCC57093.2023.10178929

[9] G. M. Amdahl, Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities. In: AFIPS Conference, Proceedings, Vol. 30. 483–485. https://doi.org/10.1145/1465482.1465560

[10] U. Vishkin: *Is Multicore Hardware for General-Purpose Parallel Processing Broken?* Comm. ACM, 57, p. 35 (2014) The Future of Computing Performance: Game Over or Next Level?, https://www.nap.edu/read/12980/chapter/1, 31/08/2017. utoljára látogatott: 2024. 06. 06.

[11] A. N. Sloss, D. Symess, C. Wright: *ARM System Developer's Guide Designing and Optimizing System Software*. Elsevier, Morgan Kauffmann Publishers, ISBN: 1-55860-874-5, 2004, 702.

[12] Kovács S., Kóczy L.: *Application of the Approximate Fuzzy Reasoning Based on Interpolation in the Vague Environment of the Fuzzy Rulebase in the Fuzzy Logic Controlled Path Tracking Strategy of Differential Steered AGVs*. In: International Conference on Computational Intelligence, 1997. 456–467.

[13] Kovács S.: *Extending the Fuzzy Rule Interpolation FIVE by Fuzzy Observation*. In: Computational Intelligence, Theory and Applications, Springer, 2006. 485–497.

[14] S. Kovács, M. Gácsi, D. Vincze, P. Korondi, Á. Miklósi: *A Novel, Ethologically Inspired HRI Model Implementation: Simulating Dog-Human Attachment*. In: 2nd International Conference on Cognitive Infocommunications, 2011. 1–4.

[15] T. Tompa, S. Kovács: *Applying Expert Heuristic as an A Priori Knowledge for FRIQ-Learning*. Acta Polytechnica Hungarica, 17. (2020) 27–45.

[16] T. Tompa, S. Kovács, D. Vincze, Mihoko Niitsuma: *Demonstration of Expert Knowledge Injection in Fuzzy Rule Interpolation Based Q-Learning*. In: IEEE/SICE International Symposium on System Integration (SII), 2021. 843–844.

[17] I. Piller, D. Vincze, S. Kovács: *Declarative Language for Behaviour Description*. In: Emergent Trends in Robotics and Intelligent Systems. Springer, 2015. 103–112.

[18] R. Bartók, J. Vásárhelyi: *Two Methods for Autonomous Robot Obstacle Sensing and Application Programming Interface for Fuzzy Rule Interpolation*. 18th International Carpathian Control Conference (ICCC). IEEE, 2017.

[19] R. Bartók, J. Vásárhelyi: *Design of a FPGA Accelerator for the FIVE Fuzzy Interpolation Method*. International Journal of Computer Applications in Technology, 68/4. (2022) 321–331.