

TDK dolgozat

Kicsák Ádám

Debreceni Egyetem
Informatikai Kar
Informatikai Rendszerek és Hálózatok tanszék

Tudományos Diákköri Dolgozat

Téma: Hardveresen megvalósított szemafor modul

Témavezető:
Dr. Végh János
egyetemi tanár

Szerző:
Kicsák Ádám
Programtervező informatikus
Msc, II. évf.

Debrecen, 2014. 05. 03.

Tartalomjegyzék

1. Bevezetés.....	4
1.1 Célkitűzés.....	6
2. A használt fogalmak és előzmények.....	7
2.1. A szemafor előző hardveres implementációja.....	8
3. Saját hardveres szemafor modul implementációm.....	9
3.1. A szemaforok állapotai.....	9
3.2. A modul interfésze.....	10
3.3. A modul működése.....	11
3.3.1. A szemafor inicializálása.....	11
3.3.2. A lefoglalás művelet.....	11
3.3.3. A felszabadítás művelet.....	12
3.3.4. A szemafor használaton kívül helyezése.....	12
3.3.5. Alapállapot.....	12
3.3.6. Hibajelzések.....	12
4. Tesztelés és szimuláció.....	14
4.1. A tesztesetek.....	14
4.2. Tesztelés a fejlesztőkörnyezetben.....	15

4.3. A lefoglalt erőforrások és a modul sebessége.....	16
4.3.1. A lefoglalt terület vizsgálata.....	17
4.3.2. A sebesség és az időzítés vizsgálata.....	19
5. Összefoglalás.....	21
6. Ábrajegyzék.....	22
7. Táblázatjegyzék.....	23
8. Irodalomjegyzék.....	24
9. Köszönetnyilvánítás.....	26

1. Bevezetés

Az ún. újra konfigurálható hardver elemek megjelenése nemcsak technológiai szempontból jelentett újdonságot, hanem megváltoztatta a hardver és szoftver felosztás fogalmát, valamint a közöttük húzódó határvonalat. Az ötlet ugyan még az 1960-as években felmerült, és az első programozható eszközök már az 1970-es években piacra kerültek, az igazán komolyan használható FPGA-k inkább csak az 1990-es évek óta használatosak. A használati igény hatására nagyon komoly fejlődésnek indultak mind a fejlesztő szoftver segédeszközök, mind a hardver gyártási technológia. Mára a fejlett FPGA eszközök és fejlesztő környezeteik már komplett processzorok vagy egyéb digitális elektronikai egységek megépítésére is alkalmassá váltak, és lehetőséget biztosítanak az olcsó prototípus készítésre is [FPGAPrototyping]. A fejlődés eredményeként az FPGA kikerült a csak mérnökök által és csak eszközök építésére használható kategóriából, és a megkezdte hódító útját számításokat végző eszközként is [ReconfComputing]

Számos egyéb következménye mellett, a hagyományos (Neumann vagy Harvard) felépítésű számítógépek működésének optimalizálása, újabb egységek kutatása és fejlesztése is jelentősen könnyebbé vált, mivel a fejlesztők viszonylag egyszerűen tudtak kiegészítő eszközöket, új processzorokat vagy akár összetett számítógépes rendszereket is fejleszteni. Lehetővé vált a rendszerek szimulációs vizsgálata, és a tényleges hardver prototípus fejlesztése is. A számítógép gyártók folyamatos gyártásfejlesztésének eredményeként megjelentek a több központi egységet, vagy egy teljes számítógépes rendszert egyetlen tokban tartalmazó áramkörök (System on a chip) [SoC]. Az ilyen irányban kutatók pedig a gyártóktól függetlenül elkezdhettek kipróbálni új ötleteket, mint pl. a mikro-szalakat használó utasítás végrehajtás megvalósítása [MicroThread], vagy a több processzoros rendszerek működésének jobb összehangolása [AppleCore].

A kutatási eredmények alapján nyilvánvalóvá vált, hogy elértük a hagyományos architektúrák által megengedett legnagyobb órajel frekvenciát [ClockRate]. Világossá vált az is, hogy nem minden esetben a legjobb megoldás az újrakonfigurálható eszközök használata. Napjaink

egyik támogatott irányzat a számítási teljesítmény további növelésére egyfajta hibrid architektúra kutatása [HeteroComputing], ahol mindkét technológia saját legjobb képességei alapján járul hozzá a közös sikerhez. Az ötletek és próbálkozások skálája igen széles. Kapható olyan tan-panel [ZedBoard], amelyik a hagyományos (Neumann) és újrakonfigurálható hardverelemek együttes használatát teszi lehetővé, és létezik olyan operációs rendszer [BORPH], amelyik kizárólag újrakonfigurálható hardveren futva, egyenrangúként kezeli az ún. szoft processzoron futó hagyományos (szoftverben implementált) és hardverben implementált folyamatokat.

A mai számítógépes alkalmazások egyre inkább bonyolult operációs rendszereken futnak melyek az alkalmazások számára szolgáltatásokat biztosítanak. A szolgáltatásokhoz a hasznos utasításokon kívül nagy mennyiségű szolgálati utasítás is tartozik, ami jelentősen csökkenti a számítógép hasznos utasítás végrehajtási kapacitását. A gyakran használt szolgáltatások igénybevétele esetén jelentősen gyorsítható a végrehajtás, ha csökkenteni tudjuk a szolgálati utasítások számát. A kutatások már elég régóta feltárták a szoftveresen elérhető egyszerűsítési lehetőségeket (lásd pl. [ExceptionHandling]), manapság – az újra konfigurálható számítástechnika eszközeinek egyre érettebbé válásával – pedig előtérbe került a hardveresen elérhető egyszerűsítések kutatása.

A folyamat eredményeként már eddig is számos speciális hardver gyorsító elem (pl. eszköz vezérlők, kommunikációs egységek) jelent meg és részben be is épült a processzor közvetlen közelébe. A hardveres egyszerűsítések előnye, hogy lehetőség nyílik arra, hogy a Neumann-architektúra kötöttségeitől (részben) megszabadulva, egyszerű és hatékony megoldásokat kínáljunk egy-egy probléma megoldására. Hátrányuk, hogy a kész megoldást össze kell hangolni a jelenleg domináns Neumann processzorok és a rajtuk futó operációs rendszerek működésével. Az eddigi fejlesztések többsége egy-egy speciális hardver eszközhöz kapcsolódik, és csak nagyon kevés számú támogatja egy általános célú operációs rendszer működését.

1.1 Célkitűzés

Kutatásom során azt vizsgáltam meg, hogy az operációs rendszer egy gyakran használt, de viszonylag egyszerű, könnyen függetleníthető feladatának (a közös erőforrások védelmére elterjedten használt ún. szemafor) hardveres megvalósításával lehet-e jelentős teljesítményjavulást elérni, azaz egy kitűzött területen lehet-e jelentősen javítani az operációs rendszer teljesítőképességét úgy, hogy a feladatot Neumann-architektúrán való szokásos megvalósítástól eltérően egy kombinációs hálózattal végeztetem el. Fontosnak érzem biztosítani az elkészült kombinációs hálózat hibátlan működését, így figyelmet fordítottam az átgondolt teszteseteken alapuló szimulációs tesztelésére. Egy újrakonfigurálható eszköz erőforrásai meghatározott kereteken belül vannak, emiatt fontosnak éreztem az elkészült kombinációs hálózat erőforrás igényeit is megvizsgálni.

2. A használt fogalmak és előzmények

Az operációs rendszerek (valós idejű változatban is) működése ma már alapvető egyetemi tananyag, lásd pl. [TanebaumOPR], [GrehanRT]. A használt terminológia nem mindig egységes, ezért nagyon röviden, és főként az általam használt terminológia ismertetésére felidézem az operációs rendszerek erőforrás kezelésére és különösen a szemaforokra vonatkozó alapfogalmakat.

A szemafor a mai hardvereknek, és a rajtuk futó operációs rendszernek egy fontos eleme. Az operációs rendszer az erőforrásokat szemaforokkal védi, hogy azokat a futó folyamatok egymást kölcsönösen kizárva használhassák.

A szemafor szoftveres megvalósítása esetén a szemafor gyakorlatilag egy olyan változó, ami a memóriában van tárolva. A szemaforon két művelet végezhető, a P művelet és a V művelet. Amikor egy folyamat hozzá szeretne férni az adott erőforráshoz, akkor az operációs rendszer a P művelettel megpróbálja lefoglalni az adott erőforrást. Ha sikerül (a szemafor előzőleg szabad volt), akkor a szemafor állapotát átállítja foglalt állapotra. Ekkor a folyamat hozzáférhet az erőforráshoz, és a többi folyamat ilyenkor az erőforrást nem érheti el, várniuk kell annak felszabadulására. Amikor a folyamat befejezte az erőforrás használatát, akkor az operációs rendszer a V művelet segítségével felszabadítja az erőforrást, így lehetőséget biztosítva arra, hogy ahhoz más folyamatok is hozzáférhessenek.

A szemafor szoftveres kezelése elég sok erőforrást igényelhet a rendszerben. Gondoskodni kell arról, hogy a szemafor állapotát tároló változón, így az általa foglalt memóriaterületen a P és a V művelet atomi legyen. Ez akár azzal is járhat, hogy a szemaforműveletek idejére a megszakítások le legyenek tiltva. Továbbá minden egyes szemaforművelet alatt a processzornak be kell tölteni és egyes esetekben ki is kell írni a szemafor változóját. Ezek az utasítások természetesen memória olvasással, memória írással járnak, amik lefoglalhatják a rendszersínt. Figyelni kell továbbá a szemafor által felhasznált memóriaterület védelmére is.

Egy hardveres megoldás esetén az atomi műveletek sokkal könnyebben kivitelezhetőek. Ezekben a megoldásokban ahelyett, hogy egy meglévő hardver architektúrára valamilyen szoftveres megoldás lenne tervezve, maga a hardver tervezése olyan, hogy a különböző kombinációs logikai hálózaton keresztülhaladva több művelet is elvégezhető legyen akár egy órajel ciklus alatt.

Így megtervezhető és kivitelezhető lehet a hardverben egy olyan szemafor, amelynek a működéséhez nem szükséges a rendszersín használata, és a megszakítások letiltása sem.

2.1. A szemafor előző hardveres implementációja

Az irodalom tanulmányozása során bukkantam rá arra a közleményre [HWSemaphore], amelyről első pillantásra úgy tűnt, hogy lényegében az általam kitűzött célt valósította meg. Valójában a közleményben szereplő rendszer egésze a kitűzött céloktól összetettebb feladatot lát el, hiszen a kezeli a folyamatok prioritását is, valamint tudja tárolni, hogy melyik folyamat melyik szemafor felszabadulására várakozik.

A közleményben ugyan nem említik konkrétan a rendszer kapacitását, de a címvonalak hat bites szélességéből, és a folyamat várakozási mátrixának 8×8 -as méretéből arra következtetek, hogy a rendszer 64 szemaforot képes kezelni egy időben.

A rendszer szimulációin látható, hogy a különböző szemafor műveletek több órajel ciklus alatt kerülnek végrehajtásra.

3. Saját hardveres szemafor modul implementációm

A szemafor modult a Xilinx ISE 9.1i fejlesztőeszköz [XilinxISE] segítségével, Verilog hardverleíró nyelven [Verilog2001] készítettem el.

A modul feladata, hogy egy időben meghatározott mennyiségű szemafort tudjon kezelni. Minden egyes szemafort tudjon inicializálni, lefoglalni (P művelet), felszabadítani (V művelet), és használaton kívül helyezni, valamint tudjon tájékoztatást adni hibák bekövetkezéséről is.

Az inicializálás és használaton kívül helyezés műveletek nem kapcsolódnak szorosan a szemafor működéséhez, de ezekkel a műveletekkel lehet nyilvántartani, hogy egy hardveres szemafor mikor van használatban, vagy használaton kívül. A két művelet funkciójában megfelel a szoftveres megoldás által használt memóriaterület lefoglalásnak és felszabadításnak. A hardveres megoldásban viszont a két művelet csak egy jelzőbit átállításával jár.

A modul az említett négy szemaforművelet mindegyikét egy órajel ciklus alatt végzi el, és az esetleges hibák előfordulásáról is ez idő alatt ad tájékoztatást.

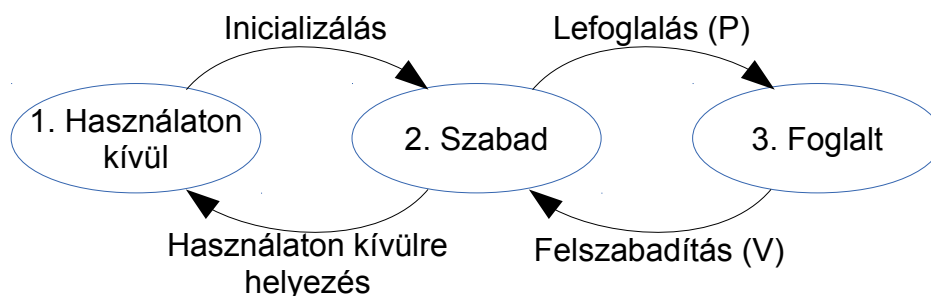
A modulnak van egy paramétere, a COUNT, ami modul kapacitását határozza meg, azaz hogy működés közben maximum hány szemafor lehet egyszerre használatban, ezzel a paraméterrel a modul egyszerűen igazítható a rendszerigényekhez. Az említett paramétert már a modul létrehozásakor meg kell adni, és csak az FPGA konfigurálásakor módosítható.

3.1. A szemaforok állapotai

A modulban az egyes szemaforoknak három állapota lehet:

1. Használaton kívül
2. Szabad
3. Foglalt

Az állapotok között a négy szemaforművelettel lehet lépkedni. A műveletek és az általuk generált állapotátmeneteket az 1. Ábra szemlélteti.



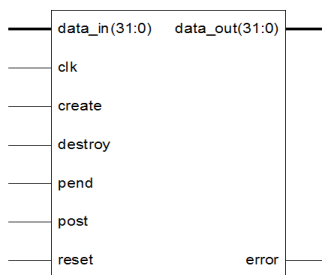
1. Ábra: A szemafor állapotai

3.2. A modul interfésze

A modul kimeneteit és bemeneteit a 2. Ábra szemlélteti. A modulnak szüksége van egy órajel bemenetre, és egy alapállapotba hozó bemenetre. Ezeket a feladatokat a `clk`, és az `rst` vonalak látják el.

A négy szemafor művelet elvégzésére négy vezérlővonalat alakítottam ki. Ezek a `create`, a `destroy`, a `pend` és a `post` vezérlő vonalak, melyek a négy szemaforműveletnek felelnek meg.

A modulba való adatbevitelre és adatkivitelre a `data_in` és a `data_out` 32 bites adatbuszok szolgálnak.



2. Ábra: A szemafor modul interfésze

Hiba bekövetkezésének jelzésére létrehoztam az `error` státuszvonalat, amely mindig az utoljára elvégzett művelet eredményéről ad tájékoztatást. Ha egy művelet elvégzése után ez a vonal aktív, akkor a műveletet valamilyen hiba miatt nem sikerült végrehajtani. Ebben az esetben az adatkimeneten megjelenő értékek adnak tájékoztatást a hiba jellegéről.

3.3. A modul működése

A modul szemaforonként két bitet használ az állapot tárolására. Az első bit azt jelzi, hogy az adott szemafor használatban van-e. A második bit a szemafor foglaltságáról ad információt, amennyiben használatban van a szemafor. A modul a kért műveletet a `clk` bemenet felfutó élére végzi el.

3.3.1. A szemafor inicializálása

Egy szemafor inicializálásához a modul `create` vonalát logikai igaz értékre kell állítani. Inicializálás folyamán a modul megkeresi az első használaton kívüli szemafort, használatba veszi azt, és szabad állapotra állítja. A modul a `data_out` kimenetére helyezi az újonnan létrejött szemafor azonosítóját. Amennyiben nincs szabad szemafor, hiba keletkezik, amit a modul az `error` kimenetével jelez.

3.3.2. A lefoglalás művelet

A szemafor lefoglalását (P művelet) a `pend` vonal segítségével lehet kérni. Lefoglaláskor a modul `data_in` bemenetére kell helyezni a lefoglalni kívánt szemafor azonosítóját, melyet a szemafor inicializálásakor a modul a `data_out` kimenetére helyezett.

Lehetséges hibák:

- A kapott azonosítóval még nincs inicializálva szemafor
- A szemafor által védett erőforrás jelenleg foglalt

3.3.3.A felszabadítás művelet

A szemafor felszabadítását (V művelet) a `post` vonal segítségével lehet kérni. Felszabadításkor a modul `data_in` bemenetére kell helyezni a felszabadítani kívánt szemafor azonosítóját.

Lehetséges hibák:

- A kapott azonosítóval még nincs inicializálva szemafor
- A szemafor által védett erőforrás jelenleg szabad

3.3.4. A szemafor használaton kívül helyezése

A használaton kívül helyezést a `destroy` vonal segítségével lehet kérni. A művelet elvégzéséhez a modul `data_in` bemenetére kell helyezni a használaton kívülre helyezni kívánt szemafor azonosítóját. Egy szemafort csak akkor lehet használaton kívülre helyezni, ha az általa védett erőforrás már felszabadult.

Lehetséges hibák:

- A kapott azonosítóval még nincs inicializálva szemafor
- A szemafor által védett erőforrás jelenleg foglalt

3.3.5. Alapállapot

A modul alapállapotba hozása is az órajellel szinkronban történik, amennyiben az `rst` bemeneten logikai igaz érték jelenik meg. A modul alapállapotba hozásakor az összes szemafor használaton kívülre kerül. A művelet végrehajtásakor hiba nem keletkezhet.

3.3.6 Hibajelzések

Amennyiben valamilyen hiba történt a kapott parancs végrehajtása közben, a modul logikai igaz értéket helyez az `error` kimenetére, és a `data_out` kimenetén a hiba jellegéről

információt ad. A kimeneten a hibajelzések az 1. Táblázat szerinti formátumban jelennek meg. Az egyes bitek logikai igaz értéke jelzi a különböző hibákat.

Bit sorszáma	Jelentés
31 – 11	Mindig logikai hamis
10	Létrehozás sikertelen, mert nincs több használaton kívüli szemafor
9	Használaton kívül helyezés sikertelen mert a szemafor foglalt
8	Lefoglalás vagy felszabadítás sikertelen
7 – 2	Mindig logikai hamis
1	Érvénytelen művelet. Ez a hiba akkor fordulhat elő, ha egyszerre több szemaforművelet vonala logikai igaz értékű
0	Érvénytelen azonosító. Ez a hiba azt jelzi, ha a kapott azonosítóval nincs még inicializálva szemafor.

1. Táblázat: A hibakód formátuma a data_out kimeneten

A hibajelzést nem kell törölni, a modul tudja folytatni a működését egy hiba bekövetkezése után is, és végre tudja hajtani a következő műveletet az órajel bemenet soron következő felfutó élére.

4. Tesztelés és szimuláció

A modul tesztelése azért szükséges, hogy a helyes működés igazolható legyen. A tesztek szimulálják a modul működését, megvizsgálják a modul kimenetén megjelenő jeleket és belső állapot helyességét is. A tesztelés alatt különböző tesztesetek kerülnek lefuttatásra és kiértékelésre. A modul működésének teljes tesztje másképp nem lehetséges, hiszen csak a tesztesetek körütekintő összeállításával van lehetőség olyan körülmények előállítása, és a modulnak a körülményekre adott válaszána megfigyelésére, amilyen körülmények valós használat esetében lehetséges, hogy nem is fordulnak majd elő.

4.1. A tesztesetek

A tesztelési folyamatra a modul nyolc szemafor kezelésére lett paraméterezve (COUNT = 8). A különböző tesztesetek tesztelik a négy szemaforműveletet, és kikényszerítik az összes lehetséges hiba bekövetkezését is.

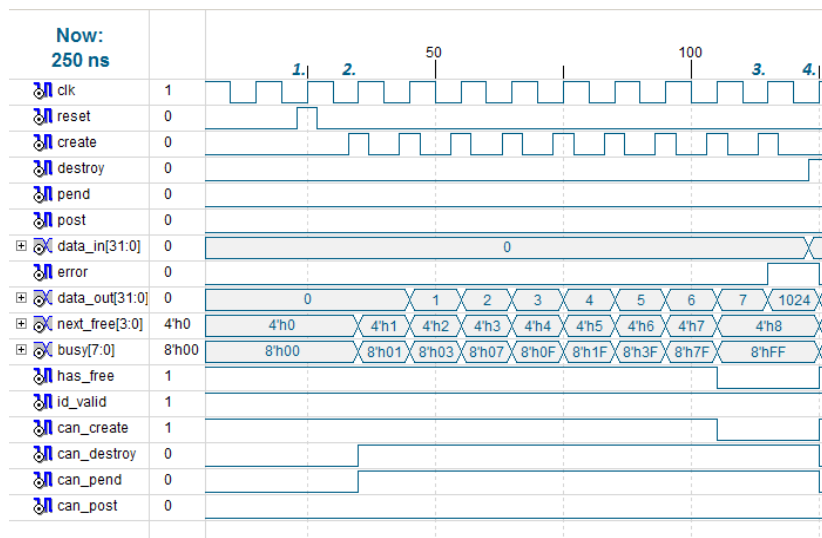
A folyamat úgy került összeállításra, hogy minden teszteset az előző teszteset által hagyott állapotból kiindulva végezze el a műveleteket. Ha egyik teszteset hibásan fut le, az egész tesztfolyamat megáll. A folyamat alatt a következő esetek kerülnek tesztelésre:

1. A modul alapállapotba hozása
2. Mind a nyolc szemafor inicializálása
3. Egy újabb szemafor inicializálása
4. Egy véletlenszerűen kiválasztott szemafor használaton kívül helyezése
5. A fenti szemafor lefoglalása, felszabadítása, használaton kívül helyezése
6. Szemafor inicializálása
7. Az inicializált szemafor lefoglalása
8. Lefoglalás újra
9. A lefoglalt szemafor felszabadítása
10. Felszabadítás újra

11. Vezérlő vonalak érvénytelen állapotba helyezése
12. Érvénytelen szemaforazonosító használata
13. A modul alapállapotba hozása

4.2. Tesztelés a fejlesztőkörnyezetben

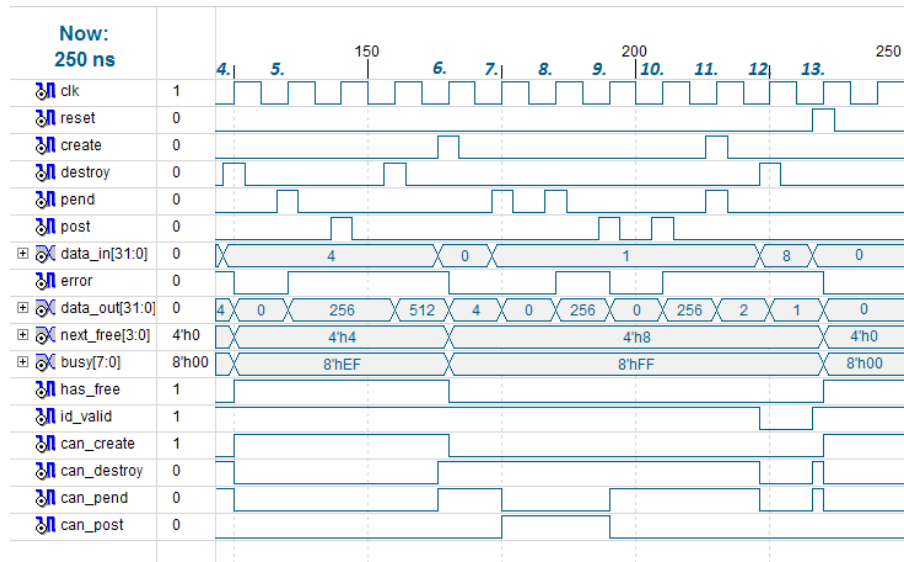
A teszteléshez a Xilinx ISE fejlesztőkörnyezet szimulációs eszközét használtam. A teszteseteket egy Verilog Test Fixture fájlban programoztam le, mely az előző fejezetben leírtak szerint, pontról pontra végzi a tesztelést. A tesztfolyamatról készült szimulációt a 3. Ábra és a 4. Ábra szemlélteti.



3. Ábra: A szimuláció eredménye az 1-4. teszteset futása alatt

Az ábrák fejlécében a dőlt félkövér betűk a teszteset sorszámát jelölik. A clk, reset, create, destroy, pend, és post vonalak, valamint a data_in busz a szemafor modul bemenetei, a data_out busz és az error vonal a modul kimenetei. A többi megfigyelt vonal belső regiszter vagy kombinációs logikai hálózat kimenete.

A busy busz bitjei arról adnak információt, hogy melyik szemafor van használatban. A has_free vonal logikai 1 értéke azt jelenti, hogy még van használaton kívüli szemafor. Az id_valid vonal értéke a data_in busz bemenetre elhelyezett azonosító érvényességéről ad információt



4. Ábra: A szimuláció eredménye az 5-13. tesztet futása alatt

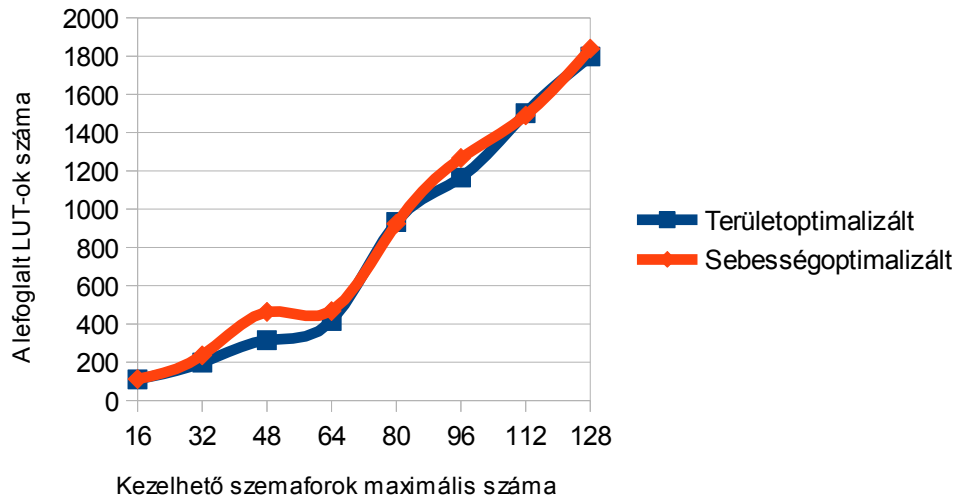
A can_create, can_destroy, can_pend, és can_post vonalak a négy szemafor művelet az adott időpillanatban való végrehajthatóságáról adnak információt.

4.3. A lefoglalt erőforrások és a modul sebessége

A vizsgálathoz a LOGSYS Spartan-3E FPGA kártyát [LOGSYS] használtam, melyen a Xilinx Spartan3E sorozatából a XC3S250E-4TQ144C chip található meg. A szintetizálást terület és sebesség optimalizáltan is elvégeztem. A modullal kezelhető szemaforok maximális számát 16 és 128 közötti intervallumon növeltem 16-osával.

4.3.1. A lefoglalt terület vizsgálata

A lefoglalt területet az 5. Ábra szemlélteti. A diagramon jól látható a 16 – 64 és a 80 – 128 szemaforot megvalósító szintetizálások között egy töréspont.



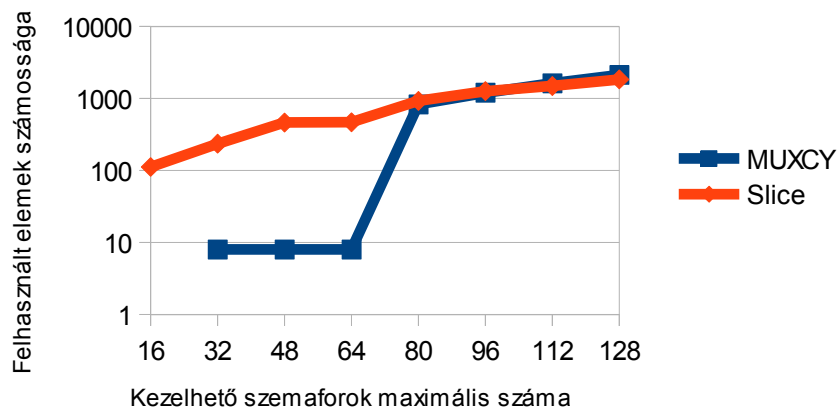
5. Ábra: A felhasznált Slice-k száma a szemafor kapacitás függvényében

A töréspont mind a terület optimalizált, mind a sebesség optimalizált szintetizálás eredményében jelen van. Megvizsgáltam, hogy a jelenség oka lehet-e az, hogy a logikai háló bonyolultsága elért egy olyan szintet, amikor már bizonyos tárolókat többszörösen helyez le a szintetizáló. A szintetizálást megismételve a regiszter többszörözés tiltásával sebesség optimalizált esetben kevesebb tárolót használt fel a szintetizálást során, viszont területre optimalizálva nem látszik változás, így ez nem magyarázza a töréspontot a diagramon. A vizsgálat eredménye a 2. Táblázaton látható.

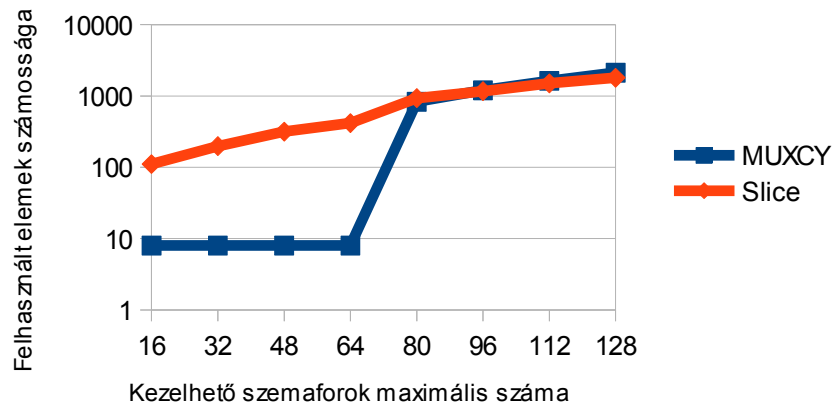
	Felhasznált tároló	
Optimalizációs cél	Sebesség	Terület
Regisztertöbbszörözés engedélyezett	358	160
Regisztertöbbszörözés tiltva	171	160

2. Táblázat: A felhasznált tárolók száma 80 szemafor kapacitás mellett különböző esetekben

A valódi ok az, hogy bár a 64 szemafor megvalósító, és a 80 szemafor megvalósító Verilog kód szempontjából bizonyos regiszterek és kombinációs hálózatok csak ki lesznek bővíve, a generált logikai hálózatok felépítése nem biztos, hogy hasonló lesz. A szintetizálás eredményében ez úgy jelentkezett, hogy a felhasznált slice-k száma és a felhasznált multiplexerek száma közötti arány jelentősen megváltozott. Ahogy a 6. Ábra és a 7. Ábra szemlélteti, 80 szemaforkapacitás mellett több mint százszor annyi MUXCY típusú multiplexer került felhasználásra, mint 64 szemaforkapacitás esetén. A MUXCY multiplexerek segítségével az FPGA-ban egy nagyon jó teljesítményű hálózat állítható össze, ám ez a megoldás nem a legjobb a területkihasználás szempontjából [Multiplexer].



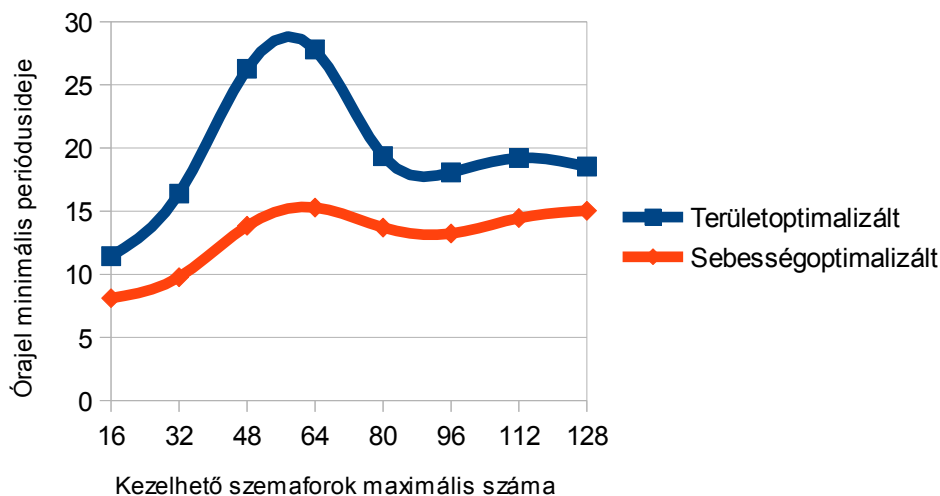
6. Ábra: MUXCY és slice-k aránya sebesség optimalizált esetben



7. Ábra: MUXCY és slice-k aránya terület optimalizált esetben

4.3.2. A sebesség és az időzítés vizsgálata

A modul sebességét a 8. Ábra szemlélteti. A diagramon itt is megjelenik egy töréspont a 16 – 64 és a 80 – 128 szemafor megvalósító szintetizálások között, mind a terület optimalizált, mind a sebesség optimalizált eredményében.



8. Ábra: Az órajel periódus ideje a szemaforkapacitás függvényében

A tárolók többszörös lehelyezése azzal is jár, hogy a több tároló kimenetére kötött hálózatok között megoszlik a terhelés. Minél kisebb egy logikai hálózaton a terhelés, annál hamarabb jut át rajta a jel. A szintetizálást megismételve a regiszter többszörözés tiltásával az órajel minimális periódusideje sebesség optimalizált esetben megnövekedett. Az eredmény a 3. Táblázaton látható.

Optimalizációs cél	Órajel periódus	
	Sebesség	Terület
Regisztertöbbszörözés engedélyezett	13,704ns	19,353ns
Regisztertöbbszörözés tiltva	14,200ns	19,353ns

3. Táblázat: Minimális órajel periódus 80 szemaforkapacitás mellett különböző esetekben

Elmondható, hogy a regiszter többszörözés a sebesség optimalizált esetben javít a sebességen, de nem magyarázza a terület optimalizált eset sebességjavulását.

A valódi ok itt is a MUXCY használatában rejlik. Egy MUXCY multiplexereken alapuló multiplexerhálózat használatával a hálózat nagy része a túlsordulás logika vonalára kerül, ami elősegíti a logikai blokkok közvetlen és jó hatásfokú összekapcsolását [Multiplexer]

5. Összefoglalás

A dolgozat egy hardverben megvalósított szemafor modult mutat be. A tesztesetek hibamentes futása alapján elmondható, hogy a szemafor modul az elvártaknak megfelelően működik.

Ugyan a közleményen [HWSemaphore] megjelent rendszer több funkciót lát el mint a saját implementáció, hiszen szemaforok mellett egyben kezeli a folyamatokat is, de a dolgozat célját a saját – csak szemaforokat kezelő – implementáció használata és vizsgálata tette elérhetővé.

Mivel az elkészült modulban csak szemaforkezelő került implementálásra, a szemaforkezelés összekapcsolható tetszőleges más hardveres vagy szoftveres megoldással.

Míg a közleményben megjelent szemafor modul skálázhatóságáról nem esik szó, addig a saját szemafor megvalósításom által egy időben kezelt szemaforok maximális száma a COUNT paraméterén keresztül beállítható, így a modul képessége, és területfoglalása a befogadó rendszer igényeihez igazítható.

A modul zárt abból a szempontból, hogy a szemaforok állapotához kívülről nem férhet hozzá semmi- A modul, és a szemaforok állapotára csak a szemaforműveletekkel, és a modul alapállapotba való hozásával lehet hatást gyakorolni.

Az elkészült szemafor modul egy konkrét rendszerbe való behelyezésére több megoldás is elképzelhető, akár a feldolgozó egységben is megjelenhet saját gépi szintű utasításokkal, vagy akár perifériaként is ki- és beviteli utasításokkal vezérelve.

A modult egy rendszerben elhelyezve jelentős terhelést vesz le a feldolgozó egységről, hiszen már csak a szemafor modult kell használnia a szemaforok kezelésére, nem szükséges figyelmet fordítani a műveletek atomi végrehajtására, hiszen erről a modul gondoskodik. A memória védelme sem szükséges, hiszen a modul nem használja a rendszer memóriáját, saját regisztereiben tárolja a szemaforok állapotát.

Összességében elmondható, hogy a modul használata felgyorsítja egy rendszer működését, hiszen rengeteg járulékos művelet elvégzése alól mentesül a rendszer a futása közben.

6. Ábrajegyzék

1. Ábra: A szemafor állapotai.....	10
2. Ábra: A szemafor modul interfésze.....	10
3. Ábra: A szimuláció eredménye az 1-4. teszteset futása alatt.....	15
4. Ábra: A szimuláció eredménye az 5-13. teszteset futása alatt.....	16
5. Ábra: A felhasznált Slice-k száma a szemafor kapacitás függvényében.....	17
6. Ábra: MUXCY és slice-k aránya sebesség optimalizált esetben.....	18
7. Ábra: MUXCY és slice-k aránya terület optimalizált esetben.....	18
8. Ábra: Az órajel periódus ideje a szemafor kapacitás függvényében.....	19

7. Táblázatjegyzék

1. Táblázat: A hibakód formátuma a data_out kimeneten.....13
2. Táblázat: A felhasznált tárolók száma 80 szemaforkapacitás mellett különböző esetekben 17
3. Táblázat: Minimális órajel periódus 80 szemaforkapacitás mellett különböző esetekben...19

8. Irodalomjegyzék

[DesignWarrior] Clive „Max” Maxfield: The Design Warrior's Guide to FPGAs. Newness/Elsevier 2004 Burlington, USA. ISBN 978-0-7506-7604-5.

[FPGAprototyping] Pong P. Chu: Prototyping By Verilog Examples, Wiley, 2008, ISBN: 978-0470185322

[ReconfComputing] S. Hauck and André DeHon: Reconfigurable Computing. Morgan Kaufmann/Elsevier 2008 Burlington, USA. ISBN 978-0-12-370522-8.

[SoC] System on a chip

http://en.wikipedia.org/wiki/System_on_a_chip

[MicroThread] [M. Daněk](#), [L Kafka](#), [L Kohout](#), [J Sikora](#), [R Bartosinski](#): UTLEON3: Exploring Fine-Grain Multi-Threading in FPGAs. Springer, 2012. ISBN: 978-1-46-142409-3

[AppleCore] <http://www.apple-core.info/>

[ClockRate] V. Agarwal, M.S. Hrishikesh, S.W. Keckler and D Burger: Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures, in Proceedings of the 27th Annual International Symposium on Computer Architecture, 2000, URL http://static.usenix.org/event/hotpar09/tech/full_papers/chandy/chandy_html

[CompDensity] J. Williams, A. D. George, J. Richardson, K. Gosrani, S. Suresh, Computational density of fixed and reconfigurable multi-core devices for application acceleration, in: Proceedings of Reconfigurable Systems. Summer Institute, Urbana, IL, Jul. 2008., 2008.

[HeteroComputing] http://cordis.europa.eu/projects/rcn/80670_en.html

[ZedBoard] <http://zedboard.org/>

[BORPH] BORPH: An Operating System for FPGA-Based Reconfigurable Computers
<http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-92.html>

[ExceptionHandling] C. A. Thekkath, H. M. Levy: ASPLOS-VI Proceedings 1994: Hardware and software support for efficient exception handling. URL
www.thekkath.org/Documents/traps.pdf

[TanebaumOPR] A. S. Tanenbaum, A. S. Woodhull: Operációs rendszerek. Panem- Prentice Hall. Budapest, 1999. ISBN 963 545 189 X

[GrehanRT] R. Grehan, R. Moote, I. Cyliax: Real-Time Programming. Addison-Wesley/Longman. 1998, Reding, USA. ISBN: 0-201-48540-0

[HWSemaphore] Yan LI, Ping-Ping Gu and Xian-Xhan Wang: The implementation of Semaphore Management in Hardware Real Time Operating System. Information Technology Journal 10(1) 158-163, 2011.

[XilinxISE] ISE Classics – WebPACK – Earlier Software Releases
<http://www.xilinx.com/webpack/classics/wpclassic/>

[Verilog2001] Verilog® HDL Quick Reference Guide
http://www.sutherland-hdl.com/online_verilog_ref_guide/verilog_2001_ref_guide.pdf

[LOGSYS] LOGSYS SPARTAN-3E FPGA KÁRTYA
http://logsys.mit.bme.hu/sites/default/files/page/2009/09/LOGSYS_SP3E_FPGA_Board.pdf

[Multiplexer] White Paper: Xilinx FPGA Families, Multiplexer Selection
http://www.xilinx.com/support/documentation/white_papers/wp274.pdf

9. Köszönetnyilvánítás

Köszönettel tartozom Dr. Végh János témavezetőmnek, aki végig segítőkészen állt rendelkezésemre, és a kutatással, illetve a dolgozattal kapcsolatosan hasznos tanácsokkal látott el.