



SAPIENTIA
ERDÉLYI MAGYAR
TUDOMÁNYEGYETEM

TUDOMÁNYOS DIÁKKÖRI DOLGOZAT

Fejlesztői környezet és In-Circuit Debugger tervezése és kivitelezése RISC processzorra

Szerző: Székely Áron

Automatika és alkalmazott
informatika szak, II évf.

Témavezető: Dr. Bakó László

egyetemi docens

SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM

**FEJLESZTŐI KÖRNYEZET ÉS IN-CIRCUIT DEBUGGER TERVEZÉSE ÉS
KIVITELEZÉSE RISC PROCESSZORRA**

**DESIGN AND IMPLEMENTATION OF DEVELOPER ENVIRONMENT
AND IN-CIRCUIT DEBUGGER FOR RISC PROCESSOR**

Székely Áron

Témavezető:

Dr. Bakó László

Kézirat lezárva: 2024 április 14.

Abstract

The use of microcontrollers is almost inevitable in the development of modern electronic devices: they are generally cheaper than circuits assembled from discrete elements, and numerous emerging problems can be solved through software updates. However, along with these advantages come disadvantages: it is harder to understand their internal operations.

The aim of my paper is to design and implement an assembly development environment for an existing RISC processor. The interface is accessible on the web and uses standard web technologies (HTML, JavaScript, CSS), allowing the interface to be used on any device with a web browser. The interface is suitable for firmware development, testing, and exporting the complete firmware as VHDL file. Testing can be performed with a simulator expandable with virtual IO modules or with a processor built on an FPGA and an In-Circuit Debugger. In both testing methods, the program execution can be interrupted at any time (manually or based on breakpoints), instructions can be stepped through, and the internal state of the processor can be observed and modified.

Kivonat

Modern elektronikai eszközök fejlesztésében szinte elkerülhetetlen a mikrovezérlők használata: általában olcsóbbak, mint egy diszkrét elemekből összerakott áramkör és számtalan felbukkanó problémát meg lehet oldani szoftver frissítésekkel. De ezekkel az előnyökkel jönnek hátrányok is: nehezebb átlátni a belső működésüket.

Dolgozatom célja egy assembly fejlesztői környezet tervezése és kivitelezése egy meglévő RISC processzorhoz. A felület weben érhető el és standard webes technológiákat használ (HTML, JavaScript, CSS), ez lehetővé teszi a felület használatát akármilyen web böngészővel rendelkező eszközön. A felület alkalmas firmware fejlesztésére, tesztelésére és exportálására VHDL állományként. A tesztelést lehet végezni egy virtuális IO egységekkel bővíthető szimulátorral vagy egy FPGA-n kialakított processzor és In-Circuit Debuggerrel. Mind a két tesztelési módszer során bármelyik pillanatban meg lehet szakítani a program futását (manuálisan vagy breakpointok alapján), lehet utasításonként lépni és meg lehet figyelni és módosítani a processzor belső állapotát.

Tartalomjegyzék

1	Bevezető.....	1
2	Irodalmi áttekintés	1
2.1	Szimulátorok	1
2.2	In-Circuit Debuggerk	1
3	Célkitűzések.....	2
4	Elméleti megalapozás	2
4.1	Használt webes technológiák.....	2
4.2	Processzor architektúra	3
5	Gyakorlati megvalósítás.....	4
5.1	Felhasználói felület és szimulátor megvalósítása	4
5.1.1	Osztályok leírása	5
5.1.2	Virtuális IO egységek címezése	7
5.1.3	Felhasználói felület funkciói.....	8
5.1.4	Hardverrel való kommunikáció megvalósítása.....	9
5.2	In-Circuit Debugger megvalósítása	10
5.2.1	Debug funkciók tervezése.....	10
5.3	Kommunikációs protokoll megtervezése.....	11
5.3.1	Fizikai és kommunikációs réteg.....	11
5.3.2	Applikáció réteg.....	11
5.4	Kommunikációs protokoll implementálása	12
5.4.1	Fizikai és kommunikációs réteg.....	12
5.4.2	Applikáció réteg.....	12
6	Mérési eredmények.....	14
7	Összefoglalás	15
7.1	Továbbfejlesztési lehetőségek.....	15
8	Irodalomjegyzék	16
9	Mellékletek	17
9.1	Memóriatérkép.....	17
9.2	Bit térkép.....	18

Ábrajegyzék

1. ábra: Dolgozat kivitelezéséhez használt processzor architektúrája	3
2. ábra: Webapplikáció felépítése	4
3. ábra: A processzor 2,3,4 címeinek a regisztertömb 1,2,3 címeire térképezése	8
4. ábra: A processzor 0,1,4 címeinek térképezése a regisztertömb 0,1,4 címeire	8
5. ábra: Képernyőkép az webapplikációról, egy példaprogram hibakeresése közben...	9
6. ábra: QR kód a webes felület eléréséhez	9
7. ábra: Kérés feldolgozó automata diagramja	13
8. ábra: Szimuláció a 2000-2001 címtartomány lekéréséről.....	14
9. ábra: Szimuláció a 4000-es cím írásáról	14
10. ábra: Szimuláció a 4000-es cím írás utáni olvasásáról	14
11. ábra: Processzor és In-Circuit Debugger erőforrás használata egy Pynq-Z2 fejlesztési lapon.....	14

1 Bevezető

Modern elektronikai eszközök fejlesztésében szinte elkerülhetetlen a mikrovezérlők használata: általában olcsóbbak, mint diszkrét elemekből ugyanarra a célra tervezett áramkör, miközben kevesebb helyet foglalnak a nyomtatott áramkörön és kevesebb energiát is fogyasztanak. De ezekkel az előnyökkel jönnek hátrányok is: nehezebb átlátni a belső működésüket.

Ezt az akadályt megkönnyítik a szimulátorok és az In-Circuit Debuggerek amik lehetővé teszik ennek a belső állapotnak a megfigyelését és módosítását, ezzel megkönnyítik a szoftveres hibák keresését, ami a fejlesztési ciklus gyorsulásához vezet.

2 Irodalmi áttekintés

Már léteznek hasonló megoldások szimulátorokra ugyanúgy, mint In-Circuit Debuggerekre, a következőkben ezeket mutatom be.

2.1 Szimulátorok

A Microchip által fejlesztett AVR mikrokontrollerekhez használható az AVR® Simulator: „Az AVR® Szimulátor egy szoftver szimulátor a Microchip AVR eszközökhöz, ahol a felhasználó futtathat és hibakeresést végezhet anélkül, hogy bármilyen hardvert használna. Szimulálja a CPU-t, beleértve az összes utasítást, megszakításokat, valamint a legtöbb on-chip I/O modult.” [1]

A nyílt forrású QEMU projekt képes rengeteg különböző processzor architektúra szimulálására. Képes a széleskörűen használt x86 és ARM architektúrákat szimulálni ugyanúgy, mint az újabb RISC-V architektúrát. [2]

2.2 In-Circuit Debuggerek

In-Circuit Debuggerekhez használt kommunikációs standard a JTAG (Joint Test Action Group), de más rendszereket is kifejlesztettek:

A Microchip Technology kis lábszámú mikrokontrolleriek számára kifejlesztett egy úgy nevezett debugWire protokollt, amin keresztül egy vezetékkel használva lehet hibakeresést végezni bármely a protokollt támogató mikrokontrolleren. [3]

Az ARM Limited kifejlesztette a Serial Wire Debug protokollt amik keresztül lehet ARM processzorokkal hibakeresést végezni. [4]

3 Célkitűzések

A célom ezzel a dolgozattal egy RISC processzor architektúrához kifejleszteni egy környezetet, ami lehetővé teszi:

- Programok írását
- Programok tesztelését szimuláció segítségével
- Programok tesztelését FPGA áramkörön megvalósított processzoron

Ezzel megkönnyítve a szoftverfejlesztést az adott architektúrára.

4 Elméleti megalapozás

Az célkitűzések elérése érdekében a következő technológiákat és meglévő processzor architektúrát használtam a dolgozatomban.

4.1 Használt webes technológiák

Webes applikációk felépítéséhez elkerülhetetlen a HTML (Hyper-Text Markup Language), a CSS (Cascading Style Sheets), és a Javascript használata. Ezek a technológiák kezdetileg csak egyszerű weboldalak alkotására voltak alkalmasak, de manapság, az említett technológiák továbbfejlesztése miatt, a rájuk épített alkalmazások képesek versengeni natív applikációkkal.

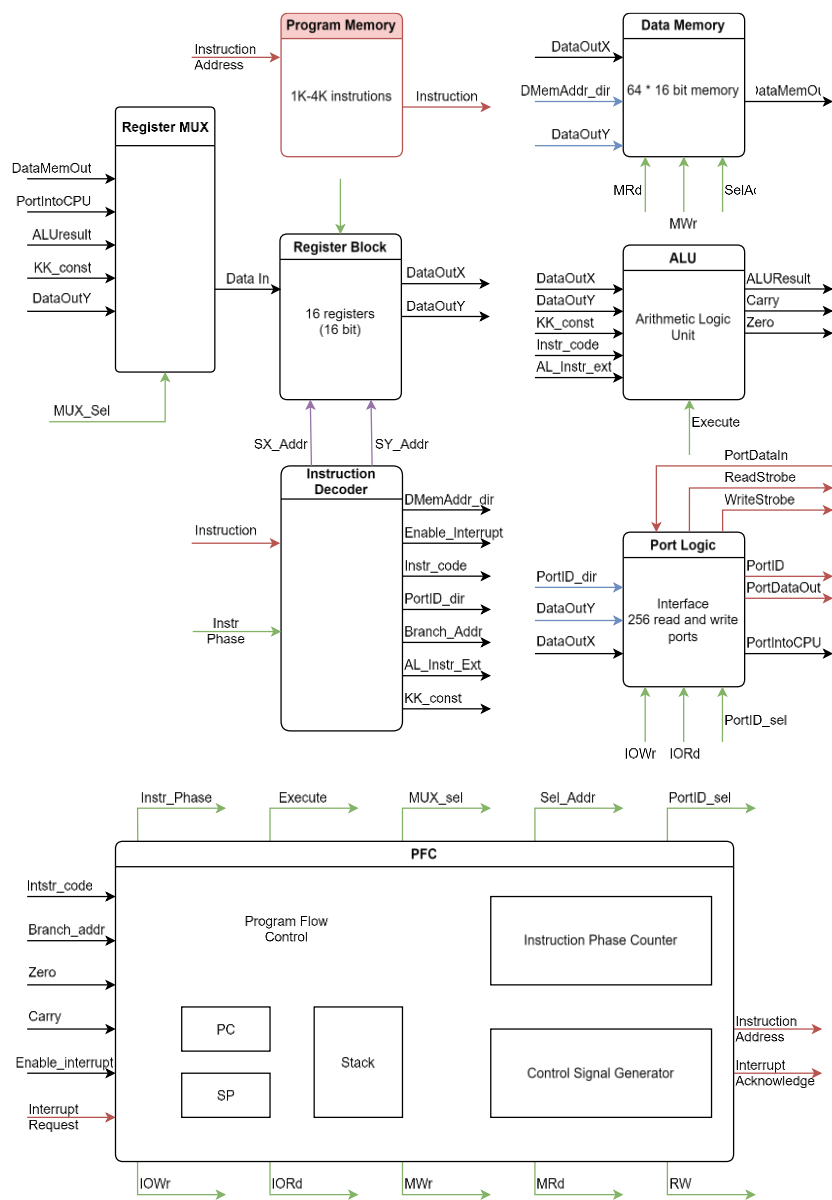
A HTML egy jelölő nyelv, lehetővé teszi egy Document Object Model (DOM) alapján leírni egy weboldalt. A DOM egy fa típusú adat struktúrába rendezi el az oldalt alkotó elemeket.

A CSS egy stílusleíró nyelv, lehetővé teszi a DOM elemek kinézetének és pozicionálásának meghatározását.

A Javascript egy programozási nyelv, weboldalak interaktivitását biztosítja. Lehetővé teszi különböző felhasználói interakciókhoz való függvények regisztrálását, amik lefutnak amennyiben a felhasználó az adott interakciót elvégzi. Képes megfigyelni és módosítani DOM felépítését és a benne lévő összes alkotó elemet is.

4.2 Processzor architektúra

A dolgozat megvalósítására az egyetemi órákon tervezett és megvalósított processzor architektúrát használtam, ami a Xilinx által kifejlesztett PicoBlaze [5] architektúrára alapul. A 1. ábrán látható a processzor felépítése.



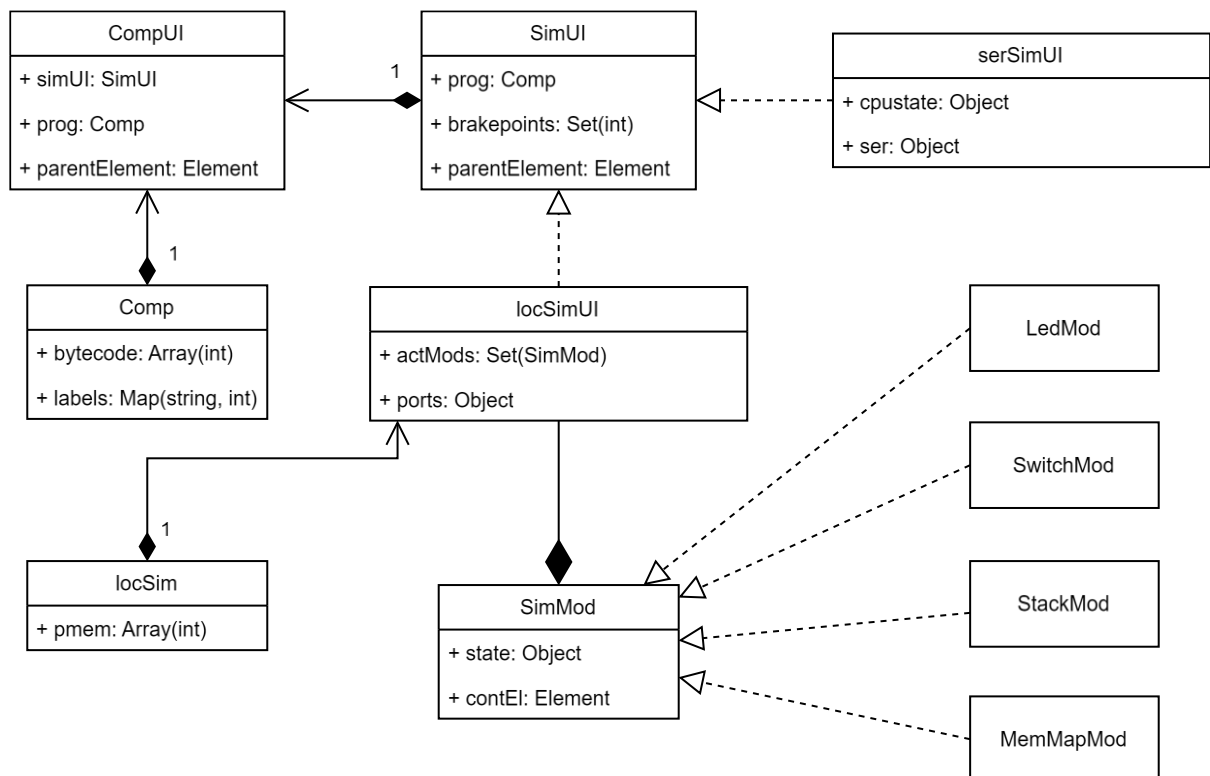
1. ábra: Dolgozat kivitelezéséhez használt processzor architektúrája

5 Gyakorlati megvalósítás

A dolgozat célkitűzései elérése érdekében két különböző alprojektet kellett teljesíteni: a felhasználói felület megvalósítását és az In-Circuit debugger megvalósítását. Ezek után meg kellett tervezni egy kommunikációs protokollt, amit mind a két alprojektben valósítani kellett.

5.1 Felhasználói felület és szimulátor megvalósítása

A szimulátort egy webapplikáció formájában valósítottam meg. Ez lehetővé teszi a felület használatát akármilyen böngészővel rendelkező eszközön, némi limitációkkal (lásd 5.1.4). Az applikáció felépítése a 2. ábrán látható.



2. ábra: Webapplikáció felépítése

5.1.1 Osztályok leírása

5.1.1.1 CompUI

Ez az osztály felelős a szerkesztő felhasználói felületének legenerálásához és az onnan érkező interakciók lekezeléséhez. A legenerált gombok funkciói:

- Delete - Kitörli a szerkesztőt és az összes általa létrehozott gyerek objektumot
- Compile: összeszereli a programot, legenerálja a megadott szimulátort vagy már létező szimulátor esetén frissíti azt, hibás forráskód esetén hibát ad ki
- Export: A legutolsó összeszerelt kódot exportálja VHDL állományként
- Show bin: Előhoz/elrejt egy panelt, amin az összeszerelt kód bináris reprezentációban megfigyelhető

Összeszereléskor az osztály létrehoz egy új Comp objektumot megadva a forrás kódot. Amennyiben sikeres ez, létrehoz egy szimulátort vagy már meglévő szimulátor esetén frissíti a bináris kódot benne

5.1.1.2 Comp

Immutabilis osztály. Létrehozásakor feldolgozza a megadott forráskódot, legenerálja a bináris kódot és címke térképet. Ezen kívül nyújt metódusokat bináris kód visszaalakítására olvasható kóddá, felhasználva a legenerált címke térképet.

5.1.1.3 SimUI

Ez az osztály felelős a szimulátor legenerálásáért és kezeléséért. A legenerált felhasználói felületen a processzor összes belső regisztere látható és módosítható. A programmemóriában automatikusan breakpoint-tal vannak bejelölve azok a sorok, amik egymagukban végtelen ciklushoz vezetnek. Ez az osztály lehetővé teszi a gyerek osztályai számára, hogy a tőle jobbra lévő felületen a saját felhasználói felületüket legenerálják. Egymagában ez az osztály nem kerül létrehozásra, hanem csak a gyerek osztályain keresztül.

5.1.1.4 locSim

A locSim osztály felelős a processzor működését lemodellezni, szimulálni. Tárolja a processzor belső állapotát és metódusokat nyújt a processzor futtatására, visszaállítására. Ezen kívül elfogad egy visszahívást amint meghív port művelet elvégzése során.

5.1.1.5 locSimUI

Ez az osztály gyerek osztálya a SimUI osztálynak. Létrehozza egy locSim objektumot és kommunikációs csatornaként szerepel a SimUI által legenerált felhasználói felület és a szimulátor között. Ezen kívül a SimUI által szolgáltatott szabad területre legenerál egy modul generátort. A modul generátor lehetőséget ad virtuális IO egységek parametrikus legenerálására. Legenerálás után a legenerált modul a generátor alá kerül az oldalon és hozzáadódik a locSimUI actMods gyűjteményéhez. A locSimUI osztály kezeli a virtuális IO egységek címezését, erről bővebben a 5.1.2 szekcióban.

5.1.1.6 SimMod

Ez a modul felelős a virtuális IO egységek legenerálásáért és magas szintű kezeléséért. Egymagában nincsen létrehozva csak a gyerek osztályai által. Felelős az IO egység felhasználói felület keretének a legenerálásáért, ami tartalmazza a nevét és port konfigurációját. Ezen kívül ez a modul felelős a modul címváltoztatásainak követésére. Bővebben a virtuális IO egységek címezéséről a 5.1.2 szekcióban.

5.1.1.7 Virtuális IO egységek leírása

5.1.1.7.1 LedMod

Konfigurálható 1-16 db virtuális LED-el. Egy írható és olvasható regisztert tartalmaz, aminek az alsó n értékét a LED-eken mutatja ki.

5.1.1.7.2 SwitchMod

Konfigurálható 1-16 db virtuális kapcsolóval. Egy olvasható regisztert tartalmaz, aminek az alsó n bitjét a kapcsolók állapota adja meg, a többi bit mindig '0'.

5.1.1.7.3 StackMod

Konfigurálható egy mélységgel. Ez a modul két regisztert tartalmaz: a Push/Pop egy belső kupac adat struktúrához nyújt hozzáférést, a regiszterbe való íráskor rátevéődik a beírt érték a kupacra és olvasáskor lekerül a legfelső érték a kupacról. A másik regiszter a Size, csak olvasható, visszatéríti a kupac jelenlegi méretét.

Ezen kívül ez a modul még képes megszakítás kéréseket generálni a processzor fele: az „Enable interrupts” jelölőnégyzetet kijelölve a modul üres állapotban lévő olvasásra és teli állapotban lévő írásokra generál megszakításokat.

5.1.1.7.4 MemMapMod

Konfigurálható egy n mérettel. Ez a modul egy regiszter tömböt reprezentál, aminek (alapértelmezetten) szekvenciális címekre vannak térképezve a regiszterei. A Data regisztere egy regiszter tömb, hozzáférésehez bővebben a 5.1.2 szakaszban.

5.1.1.8 serSimUI

Ez az osztály gyerek osztálya a SimUI osztálynak. Felelős az in-Circuit debugger és a felhasználói felület között biztosítani a kapcsolatot. Periodikusan lekérdezi a hardveres processzor állapotát, ami alapján frissíti a felhasználói felületet és a felhasználói felületről érkező akciókat is továbbítja a hardver felé. A SimUI által adott szabad területen legenerálja a soros kapcsolat módosítására és megfigyelésére alkalmas felületet.

5.1.2 Virtuális IO egységek címezése

A virtuális modulok megcímezését a modulok fejlécében lehetséges, a következő szintaxis alapján:

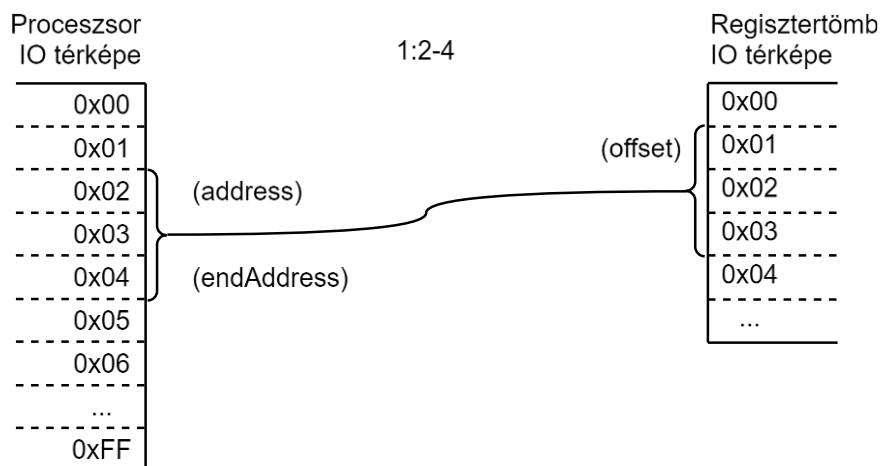
rw	offset:	address	-endAddress
----	---------	---------	-------------

Hol:

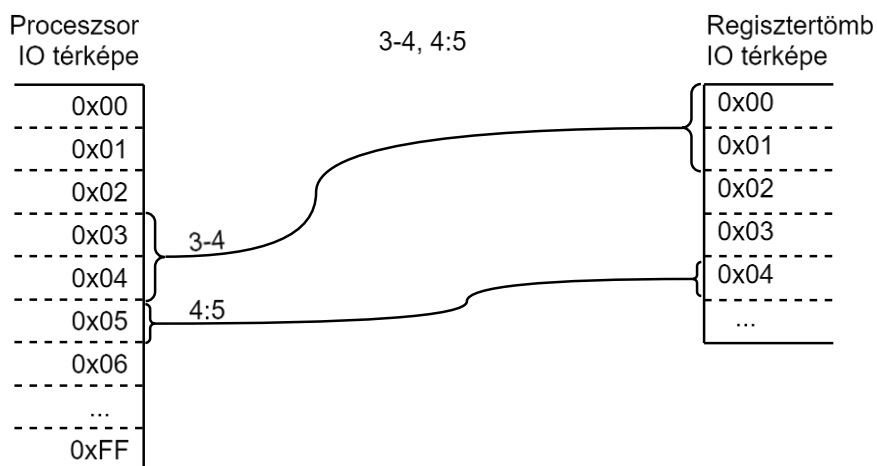
- address – természetes szám, a kívánt port címe, regiszter tömb esetén az első elem címe
- [rw] – 'r' vagy 'w', opcionális, megadja a kívánt címet csak írásra, vagy csak olvasásra szeretnénk használni, alapértelmezetten írásra és olvasásra is regisztrálja a címet
- [endAddress] – természetes szám, opcionális, abban az esetben kell megadni, ha egy regiszter tömböt címezünk meg, megadja a regiszter tömb végső címét
- [offset] – természetes szám, opcionális, abban az esetben kell megadni, ha egy regiszter tömböt címezünk meg, megadja a kezdeti cím eltolást a regiszter tömbben

Több szabály vesszővel való elválasztásával több címre rá lehet rendelni ugyanazt a regisztert. Ha egy olyan regiszterhez rendelünk olvasásra vagy írásra címet, ami nem képes arra az operációra akkor azt a címet arra az operációra figyelmen kívül haddja.

A szintaxis megértését a 3. ábrán és a 4. ábrán látható példák segítik:



3. ábra: A processzor 2,3,4 címeinek a regisztertömb 1,2,3 címére térképezése



4. ábra: A processzor 3,4,5 címeinek térképezése a regisztertömb 0,1,4 címére

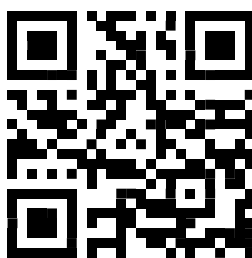
5.1.3 Felhasználói felület funkciói

Kezdetileg az oldal tartalma csak két gomból áll, „New compiler” és „New hardware connection”. Ezekre a gombokra rákattintva lehet elérni az oldal többi funkcióját. Bármelyik opció kiválasztása után be lehet gépelni a forrás kódot és létrehozni egy hibakeresésre alkalmas környezetet a „Compile” gombra rákattintva. A hibakeresés függetlenül attól, hogy hardveren vagy szimulátor alapján fut, ugyanúgy lehetséges visszaállítani, megállítani, utasításonként lépni és megszakítás kérést küldeni a processzornak. A processzor összes regiszterének az értéke

módosítható megállított állapotban a kívánt regiszterre való kattintással. Brakepoint-ok beállíthatók a kívánt cím elé kattintva.

5. ábra: Képernyőkép az webapplikációról, egy példaprogram hibakeresése közben

A felület elérhető a következő címen: <https://nblazesim.zertsu.com/>



6. ábra: QR kód a webes felület eléréséhez

5.1.4 Hardverrel való kommunikáció megvalósítása

A hardverrel való kommunikáció a serSimUI osztályban van megvalósítva. Megvalósításához a Web Serial API-t használtam, és mivel ez még egy kísérleti technológia nem minden böngésző támogatja: jelenleg csak a Google Chrome és más Chromium alapú böngészők támogatják (Microsoft Edge, Opera, etc.). Egy nem támogatott böngészőben is lehet használni az applikációt, csak a hardveres hibakeresést nem lehet elvégezni.

A soros port kezeléséhez implementáltam egy mutex zárat. Az aszinkron kommunikációs metódus megvárja ameddig szabad lesz a soros port, elküldi a kérést és a válasz bejövetele után visszatéríti azt.

5.2 In-Circuit Debugger megvalósítása

Egy In-Circuit Debugger megvalósításához szükség van egy rendszerre, ami lehetővé teszi a processzor bármelyik regiszterének megcímezését. Ezt egy memória térképpel oldottam meg: minden regiszterhez rendeltem egy 16 bites címet. Ebből az első 4 a processzoron bejüli részeket címezi meg és a maradék 12 a részeken bejül adja meg a címet, részletekért lásd 9.1 mellékletet.

5.2.1 Debug funkciók tervezése

A 0 című modul speciális mivel nem a processzor része, hanem a Debug funkciókat implementálja. Lehetővé teszi a processzor megállítását, utasításonkénti léptetését és breakpoint-ok beállítását.

A négy darab hardveres breakpoint-ot a 0001-0004 cím tartományban lehet elérni, bekapcsolásukhoz beléjük kell írni a breakpoint kívánt címét és kikapcsoláshoz a programmemória címtartományából kívül eső címet kell írni.

És a 0000 címen lévő Status/CMD regiszter két különböző szerepet tölt be: olvasáskor megadja a processzor futási állapotát (1 – fut, 0 – megállítva) és íráskor különböző utasításokat hajt végre:

- 0000 – Stop
- 0001 – Run
- 0002 – Step
- 0003 – Trigger Interrupt
- 0004 – Reset

5.3 Kommunikációs protokoll megtervezése

5.3.1 Fizikai és kommunikációs réteg

A webapplikáció és FPGA közötti kommunikációhoz RS232 soros portot használok.

5.3.2 Applikáció réteg

A kommunikációs standard megtervezéshez WozMon nevezetű programot használtam inspirációként. Ez a program kezdett el futni az Apple 1 számítógépen beindítás után. Egy egyszerű szintaxissal lehetővé tette a processzor memóriatérképén lévő összes cím terminálon keresztüli olvasását és írását. [6]

A kommunikáció mester-szolga alapú: a webapplikáció (mester) kiküld kéréseket amire az In-Circuit Debugger (szolga) válaszol, minden kommunikáció ASCII karaktereken keresztül történik, bináris adatok hexadecimális reprezentációban vannak közvetítve. A mester a CR (13) ASCII karakterrel jelzi a kérése végét és a szolga a LF (10) ASCII karakterrel jelzi a válasza végét. Ez a protokoll lehetővé teszi (megfelelő dokumentációval ellátva) az In-Circuit Debugger használatát egy egyszerű soros port terminál segítségével.

5.3.2.1 Olvasási kérések

A tervezett protokoll két különböző olvasás kérést tesz elérhetővé:

Cím	CR
-----	----

Egy cím megadásával a szolga az azon a címen lévő regiszter tartalmával válaszol.

Kezdeti cím	Végző cím	CR
-------------	-----------	----

Egy kezdőcím és egy végző cím megadásával a szolga a két cím közötti zárt intervallumból való regiszterek tartalmával válaszol

5.3.2.2 Írási kérések

A tervezett protokoll két különböző írási kérést tesz elérhetővé:

Kezdeti cím	‘.’	Adat	CR
-------------	-----	------	----

Egy kezdeti cím, egy ‘.’ (kettőspont) karakter és maximum 14 szónak megfelelő adat megadásával a szolga beírja a megadott adatokat a kezdeti címmel kezdődő tartományba, üres válasszal válaszol.

‘.’	Adat	CR
-----	------	----

Egy ‘.’ (kettőspont) karakter és maximum 15 szónak megfelelő adat megadásával a szolga beírja az előző írás kérés végső címe utáni tartományba a megadott adatokat, üres válasszal válaszol.

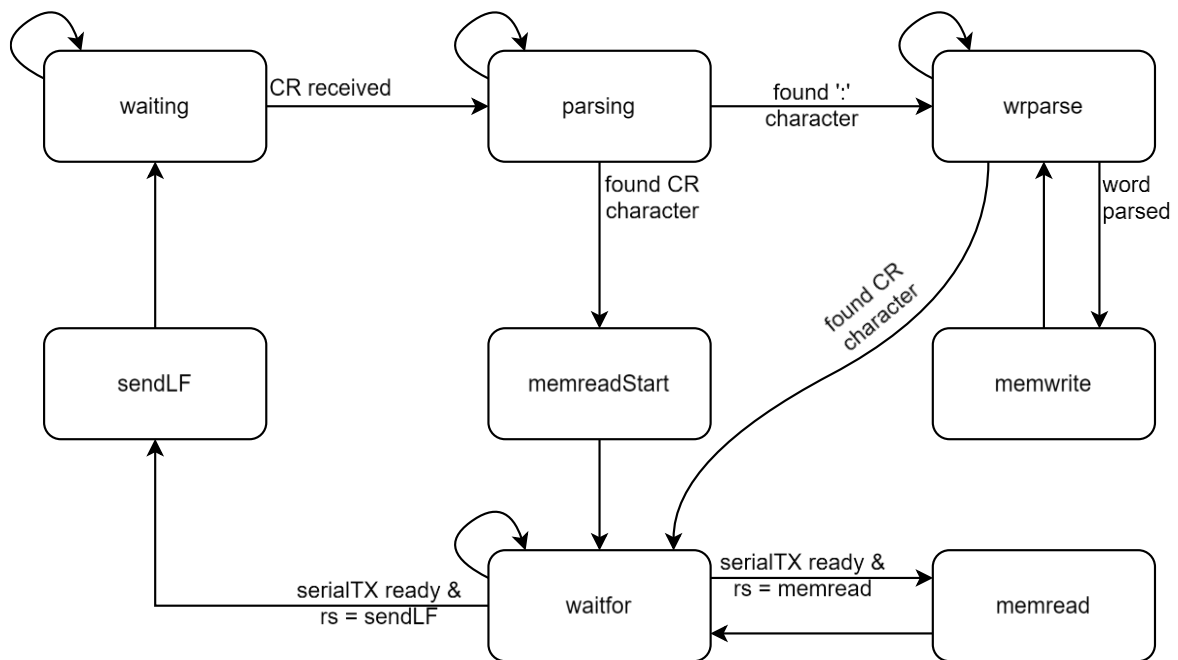
5.4 Kommunikációs protokoll implementálása

5.4.1 Fizikai és kommunikációs réteg

A soros küldő és vevő modulokat egy interneten elérhető példamodulokkal valósítottam meg. [7] A soros vevő modul egy jel egy órajelre való magas logikai szintbe emelésével jelzi egy új byte beérkezését és a soros küldő egység egy jel alapján indítsa a küldést és egy másik jelen megadja az aktivitási állapotát.

5.4.2 Applikáció réteg

A kérések feldolgozására egy véges állapotú automatát használtam. A bejövő kérést a modul egy 64 bájtos adatpufferben tárolja. A 7. ábrán látható az automata működése.



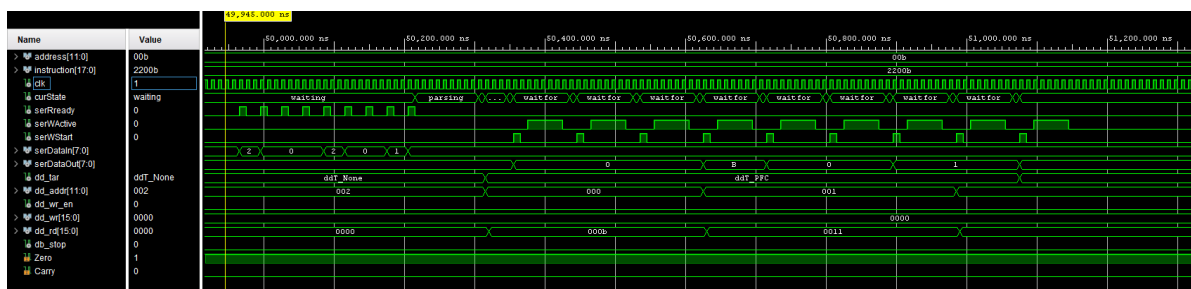
7. ábra: Kérés feldolgozó automata diagramja

Magyarázat az állapotokhoz:

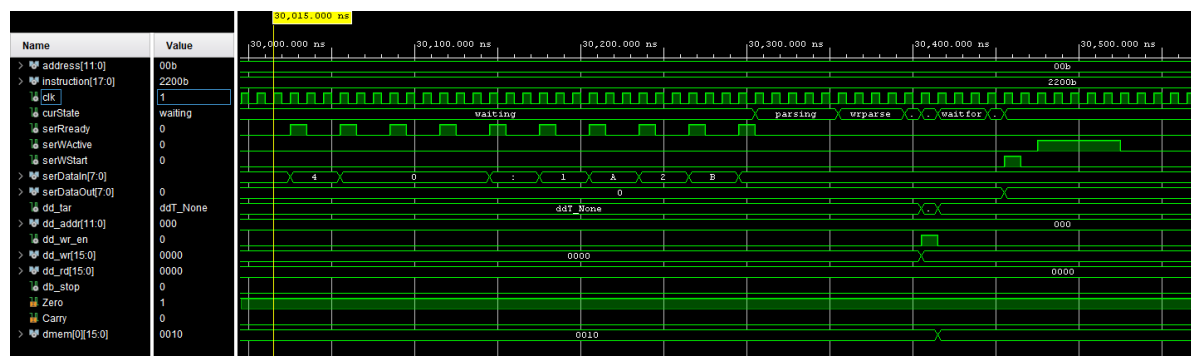
- waiting: olvas a soros portról egy pufferbe, vár egy kérés végére, CR karakter megérkezésekor átlép parsing állapotba
- parsing: értelmezi a kérést, értelmezi a kellő címeket és áttér wrparse vagy memreadStart állapotba a kérés típusától függően
- memReadStart: kiadja az olvasási kérelmet a processzor felé, és átlép waitfor állapotba, beállítva a returnStatus jelet memread-re
- waitfor: várakozik 3 óra jelet és megvárja a serialTX inaktivitását, ezután átlép a returnState jelben tárolt állapotba
- memread: kiolvassa a kért értékeket és kiküldi azokat a soros porton, növeli eggyel a lekért cím értéket és waitfor-ba való átlépés előtt beállítja a returnState jelet sendLF-re amennyiben befejezte az olvasást
- wrparse: értelmezi az értékeket a pufferben, amennyiben sikeresen kiolvasott egy értéket átlép a memwrite állapotba és ha elérte a kérés végét átlép a waitfor állapotba, beállítva a returnState jelet sendLF-re
- memwrite: egy órajelig bekapcsolja az írás jelet a processzorhoz, ezzel beírva a kívánt értéket a kívánt címre
- sendLF: soros porton kiküld egy LF karaktert

A kérés beolvasása egy 64 karakteres pufferbe történik, ez lelimitálja az egy írás kérésben megadható adatok számát.

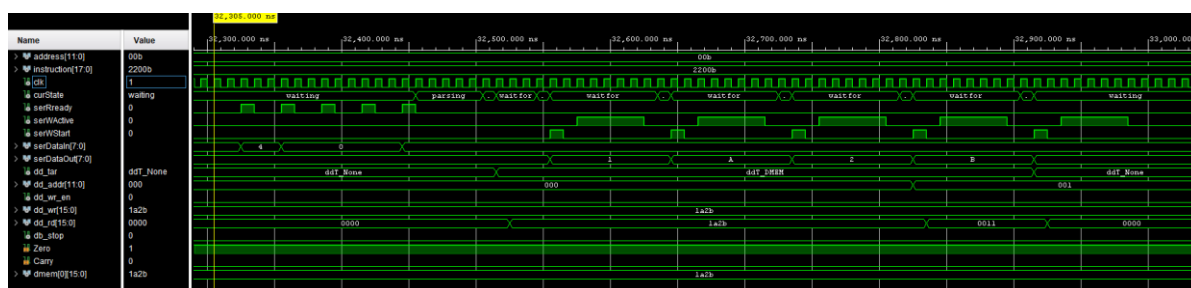
6 Mérési eredmények



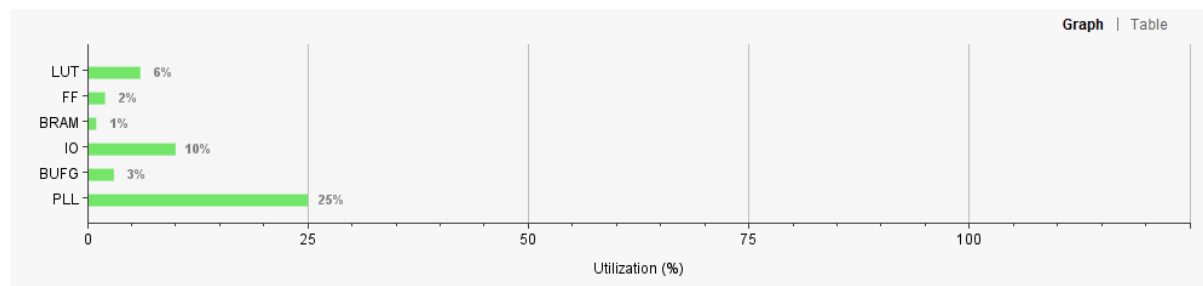
8. ábra: Szimuláció a 2000-2001 címtartomány lekéréséről



9. ábra: Szimuláció a 4000-es cím írásáról



10. ábra: Szimuláció a 4000-es cím írás utáni olvasásáról



11. ábra: Processzor és In-Circuit Debugger erőforrás használata egy Pynq-Z2 fejlesztési lapon

7 Összefoglalás

A dolgozatom kivitelezése során sikeresen végrehajtottam a következő feladatokat:

- Egy assembly program összeszerelő megvalósítása.
- Egy processzor szimulátor megvalósítása.
- Egy felhasználói felület tervezése és megvalósítása.
- Egy In-Circuit Debugger tervezése és megvalósítása.
- Egy kommunikációs protokoll tervezése és megvalósítása.

7.1 Továbbfejlesztési lehetőségek

Ezt a dolgot mind szoftveres, mint hardveres szempontból sok irányba tovább lehet fejleszteni. Szoftveres szempontból, lehet bővíteni a virtuális IO egységek könyvtárát több modullal, mint például 7 szegmenses kijelzőkkel, PS/2 egér és billentyűzet interfészekkel, vagy soros port interfészekkel. A kódszerkesztővel való interakció jobbá tehető automatikus szintaxis kiemeléssel és automatikus összeszerelés előtti hibaellenőrzés és figyelmeztetéssel.

Hardveres szempontból pedig ki lehetne fejleszteni In-Circuit Debuggerrel összekapcsolt IO egységeket, amik hasonlóan jelenhetnek meg a felhasználói felületen, mint a szimulátor által létrehozott modulok. És 5.4.2 szakaszban tárgyalt véges állapotú automatát csővezetékessítve el lehetne hanyagolni a puffer használatát, ez felgyorsítaná írás kérések esetén a válaszidőt és a kérés hossz limitációt is kiküszöbölné.

8 Irodalomjegyzék

- [1] „AVR® Simulator,” 2020. [Online]. Available: <https://www.microchip.com/content/dam/mchp/documents/parked-documents/AVR-Simulator-UserGuide-DS50003042A.pdf>. [Hozzáférés dátuma: 14 04 2024].
- [2] The QEMU Project Developers, „QEMU’s documentation,” 2024. [Online]. Available: <https://www.qemu.org/docs/master/about/index.html>. [Hozzáférés dátuma: 14 04 2024].
- [3] Microchip, „MPLAB® PICKIT™ 5 IN-CIRCUIT DEBUGGER USER'S GUIDE,” 2022. [Online]. Available: <https://onlinedocs.microchip.com/oxy/GUID-8D61C0B9-A97F-4F4D-99F8-1D7424264C2A-en-US-1/GUID-2221F90B-9A21-474E-B1B0-9688B9A9EC00.html>. [Hozzáférés dátuma: 14 04 2024].
- [4] Arm Limited, „ARM Debug Interface v5 Architecture Specification,” 2024. [Online]. Available: <https://developer.arm.com/documentation/ih0031/a/The-Serial-Wire-Debug-Port--SW-DP-/Introduction-to-the-ARM-Serial-Wire-Debug--SWD--protocol>. [Hozzáférés dátuma: 14 04 2024].
- [5] Advanced Micro Devices, „PicoBlaze 8-bit Microcontroller,” 2024. [Online]. Available: <https://www.xilinx.com/products/intellectual-property/picoblaze.html>. [Hozzáférés dátuma: 14 04 2024].
- [6] S. Bergmans, „The Woz Monitor,” 31 12 2023. [Online]. Available: <https://www.sbprojects.net/projects/apple1/wozmon.php>. [Hozzáférés dátuma: 14 04 2024].
- [7] R. Merrick, „UART, Serial Port, RS-232 Interface,” 2022. [Online]. Available: <https://nandland.com/uart-serial-port-module/>. [Hozzáférés dátuma: 14 04 2024].

9 Mellékletek

9.1 Memóriatérkép

Az In-Circuit Debugger segítségével elérhető regiszterek táblázata.

	Debug	CPU	PFC	REG	DMEM
	0	1	2	3	4
000	Status/CMD	Control Signals	PC	s0	dmem[0]
001	BP1	InstCode+AluExt	Flags	s1	dmem[1]
002	BP2	AluRes	StackP	s2	dmem[2]
003	BP3	BranchAddr		s3	dmem[3]
004	BP4	RegDataIn		s4	dmem[4]
005		RegOutX		s5	dmem[5]
006		RegOutY		s6	dmem[6]
007		RegXYAddr+KKonst		s7	dmem[7]
008		DmemAddr		s8	dmem[8]
009		PortSrobesID		s9	dmem[9]
00A		PortDataOut		sA	dmem[10]
00B		PortDataIn		sB	dmem[11]
00C				sC	dmem[12]
00D				sD	dmem[13]
00E				sE	dmem[14]
00F				sF	dmem[15]
010			stack[0]		dmem[16]
011			stack[1]		dmem[17]
...			...		
02E			stack[30]		dmem[46]
02F			stack[31]		dmem[47]
030					dmem[48]
...					
03E					dmem[62]
03F					dmem[63]

9.2 Bit térkép

A 9.1 mellékletben lévő több jelet tartalmazó regiszterek bit térképe.

Regiszter	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CPU:ControlSig	IOWr	IORd	MWr	MRd			PortID_sel	RW	SelAddr	Mux_sel			Execute	Inst_phase		
PFC:Flags+etc							intRq	INTreset	INTs	RETURNS	CALLs	JUMPs		IntEn	CF	ZF
CPU: RegXYAddr+Kkonst	Kkonst								RegXAddr				RegYAddr			
CPU:InstCode+ALUExt					ALU_ext						Instr_code					
CPU: PORT			WrStrobe	RdStrobe					PORT_ID							