

Saját tervezésű, RISC processzor megvalósítása és  
tesztelése FPGA áramkörön

Sapientia

Erdélyi Magyar Tudományegyetem, Marosvásárhely

Gáll János

Dr. Bakó László

2021

---

**UNIVERSITATEA „SAPIENTIA” DIN CLUJ-NAPOCA**  
**FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE,**  
**TÎRGU-MUREȘ**  
**SPECIALIZAREA CALCULATOARE**

**Proiectarea și dezvoltarea unui  
procesor de tip RISC cu testare pe  
circuit FPGA**  
**PROIECT DE DIPLOMĂ**

**Coordonator științific:**

**Ș.l.dr.ing. Bakó László**

**Absolvent:**

**Gáll János**

**2021**

---

### Declarație

Subsemnata/ul Gáll János, absolvent(ă) al/a specializării Calculatoare, promoția 2021 cunoscând prevederile Legii Educației Naționale 1/2011 și a Codului de etică și deontologie profesională a Universității Sapientia cu privire la furt intelectual declar pe propria răspundere că prezenta lucrare de licență/proiect de diplomă/disertație se bazează pe activitatea personală, cercetarea/proiectarea este efectuată de mine, informațiile și datele preluate din literatura de specialitate sunt citate în mod corespunzător.

Localitatea, Târgu Mureș

Data: 05.07.2021

Absolvent

Semnătura.....

UNIVERSITATEA SAPIENTIA  
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE DIN TÂRGU-MUREȘ  
SECȚIA CALCULATOARE

VIZAT DECAN  
Dr. Domokos József

VIZAT SEF DEPARTAMENT  
Dr. Szabó László-Zsolt

## TEMĂ PROIECT DE DIPLOMĂ

Conducătorul temei:  
Dr. Bakó László

Candidat : **GÁLL JÁNOS**  
Anul absolvirii: 2021

### 1. Conținutul proiectului

a) Tema proiectului de diplomă: *Proiectarea și dezvoltarea unui procesor de tip RISC cu testare pe circuit FPGA*

b) Problemele principale care vor fi tratate în proiect:

- Implementarea în limbaj VHDL - prin tehnica descrierii funcționalităților la nivel de registre - al unui procesor de tip RISC cu structură pipe-line, cu arhitectură (parțial) compatibilă cu procesorul PicoBlaze livrat de firma Xilinx.

- Se vor implementa metode de testare și comparație a vitezei de execuție a celor două procesoare, respectiv capacitatea acestora de comunicație cu echipamente periferice.

- A se implementa un software de emulare a celor două procesoare pentru a permite studierea funcționării componentelor acestora.

- Studentul va implementa un program de asamblare pentru generarea codului mașină (înglobat într-un fișier VHDL care va fi memoria de program a procesoarelor) din programele scrise cu ajutorul setului de instrucțiuni definit în prealabil.

c) Desene obligatorii:

- Schema bloc a procesorului realizat respectiv a componentelor acestuia

- Scheme de conectare a componentelor hardware

- Scheme UML ale programelor realizate

d) Softuri obligatorii

- Programe VHDL ce vor implementa componentele procesorului respectiv conectarea acestora

- Emulator al celor două procesoare

- Program de asamblare pentru generarea modulelor de memorie de instrucțiuni pentru cele două procesoare

### Bibliografie recomandată:

1. Mikroprocesszorok, mikroszámítógép elemek / Madarász László, Kecskeméti Főiskola Műszaki Főiskolai Kar, 1998, 004.3/MAD.

2. A. Tanenbaum: Számítógép architektúrák. (Arhitectura calculatoarelor) Bp., Panem Könyvkiadó, 2004.

3. Gamma, Erich, Helm, Richard, Johnson, Ralph, Vlissides, John, Programtervezési minták, Kiskapu 2004.

**Termene obligatorii de consultații:** - săptămânal

**Locul practicii:** Universitate

Primit la data de: 15. 03. 2020

Termen de predare: 06. 07. 2021

Semnătura șefului de catedră

Semnătura îndrumătorului științific

Semnătura candidatului

## Extras

Disertația prezintă un procesor conceput și implementat pentru un circuit FPGA, pe care l-am modelat pe procesorul Xilinx PicoBlaze. Vă prezint pașii de proiectare, construcție și testare.

În zilele noastre, tehnologia computerelor se dezvoltă extrem de rapid, cu reproiectarea și optimizarea diverselor elemente. Procesorul este una dintre cele mai importante componente ale unui computer care execută instrucțiuni și controlează întregul sistem.

Există mai multe tipuri de arhitecturi de procesor, cum ar fi arhitecturile Harvard și Neumann. Am construit procesorul prezentat în disertație pe principiile arhitecturii Harvard. Cu toate acestea, am folosit și o structură pipeline pentru a folosi mai bine procesorul. Acest lucru vă permite să efectuați operațiuni mult mai repede.

În timpul proiectării procesorului, am luat în calcul și strategia de proiectare RISC, a cărei esență este că procesorul are un număr mic de instrucțiuni simple. Acest lucru face ca proiectarea să fie mai simplă și mai rapidă.

Scopul disertației este de a compara cele două procesoare, PicoBlaze și procesorul pe care l-am proiectat și dezvoltat. În ceea ce privește performanța de calcul și performanța controlului proceselor. Testele efectuate pot fi împărțite în două grupe, teste aritmetice și teste de eșantionare a semnalului analogic.

În timpul testelor aritmetice, am examinat factorizarea și generarea de secvențe numerice Fibonacci pentru ambele procesoare. În timpul eșantionării analogice a semnalelor, procesoarele au trebuit să reproducă semnale sinusoidale și dreptunghiulare de diferite frecvențe.

**Cuvinte cheie:** FPGA, RISC, emulator de procesor, compilator, conversie analog-digitală

**SAPIENTIA ERDÉLYI MAGYAR  
TUDOMÁNYEGYETEM  
MAROSVÁSÁRHELYI KAR  
SZÁMÍTÁSTECHNIKA SZAK**

**Saját tervezésű, RISC processzor  
megvalósítása és tesztelése FPGA  
áramkörön  
DIPLOMADOLGOZAT**

**Témavezető:**

**Dr. Bakó László, egyetemi adjunktus**

**Végzős hallgató:**

**Gáll János**

**2021**

## Kivonat

A dolgozat egy FPGA áramkörre tervezett és megvalósított processzort mutat be amelyet a Xilinx PicoBlaze processzorról mintáztam. Illetve a tervezés, kivitelezés és tesztelés lépéseit.

Napjainkban a számítógépes technológia rendkívül gyorsan fejlődik, különböző elemek újratervezéssel, optimalizálásával. A processzor a számítógép egyik legfontosabb alkotóeleme, amely végrehajtja az utasításokat és irányítja az egész rendszert.

Több típusú processzor architektúra is létezik, mint például Harvard és Neumann architektúra. A dolgozatomban bemutatott processzort a Harvard architektúra elveire építettem. Ugyanakkor alkalmaztam egy kétfázisú csövezeték is, a processzor jobb kihasználásának érdekében. Így sokkal gyorsabban képes a műveletek elvégzésére.

A processzor tervezése során figyelembe vettem a RISC tervezési stratégiát is amelynek lényege abban áll, hogy a processzor kevés számú és egyszerű utasításokkal rendelkezik. Ennek köszönhetően a tervezés egyszerűbb és gyorsabb.

A dolgozat célja összehasonlítani a két processzort, a PicoBlaze és az általam tervezett és fejlesztett processzort. Számítási teljesítmény és folyamat vezérlési teljesítmény szempontjából. Az elvégzett tesztek két csoportba lehet osztani, aritmetikai tesztek és analóg jel mintavételezési tesztek.

Aritmetikai tesztek során mindkét processzor esetén a FaktORIZÁCIÓT és a Fibonacci számsorozat generálást vizsgáltam. Analóg jel mintavételezés során, a processzoroknak különböző frekvenciájú szinusz és négyszögjelet kellett reprodukálniuk.

**Kulcsszavak:** FPGA, RISC, Processzor emulátor, kompilátor, analóg-digitális konverzió

# Abstract

The dissertation presents a processor designed and implemented for an FPGA circuit, which I modeled on the Xilinx PicoBlaze processor. In this paper, I present the steps of design, construction, and testing.

Nowadays, computer technology is developing extremely fast, with redesigning and optimizing various elements. The processor is one of the most important components of a computer that executes instructions and controls the entire system.

There are several types of processor architectures, such as Harvard and Neumann architectures. I built the processor presented in my dissertation on the principles of Harvard architecture. However, I also used a two-phase piping to make better use of the processor. This allows you to perform operations much faster.

During the design of the processor, I also took into account the RISC design strategy, the essence of which is that the processor has a small number and simple instructions. This makes planning easier and faster.

The aim of the dissertation is to compare the two processors, PicoBlaze and the processor I designed and developed. In terms of computational performance and process control performance. The tests performed can be divided into two groups, arithmetic tests and analog signal sampling tests.

During arithmetic tests, I examined Factorization and Fibonacci number sequence generation for both processors. During analog signal sampling, processors had to reproduce sine and square signals of different frequencies.

**Keywords:** FPGA, RISC, Processor emulator, compiler, analog-to-digital conversion



# Tartalomjegyzék

<b>1. Bevezető</b>	<b>13</b>
<b>2. Célkitűzések</b>	<b>15</b>
<b>3. Szakirodalom áttekintése</b>	<b>17</b>
3.1. Design & Analysis of 16 bit RISC Processor Using low Power Pipelining . . . . .	17
3.2. 16-Bit RISC Processor Design for Convolution Application . . . . .	18
3.3. FPGA Prototyping of a RISC Processor Core for Embedded Applications . . . . .	18
3.4. Implementation of RISC Processor on FPGA . . . . .	19
<b>4. Technológiai áttekintés</b>	<b>20</b>
4.1. Hardver . . . . .	20
4.1.1. FPGA . . . . .	20
4.1.2. VHDL . . . . .	21
4.2. Assembly fordító . . . . .	21
4.2.1. Java . . . . .	21
4.3. Processzor emulátor . . . . .	22
4.3.1. C# . . . . .	22
4.3.2. .NET keretrendszer . . . . .	22
4.3.3. Windows Forms . . . . .	22
<b>5. Tervezés</b>	<b>23</b>
5.1. A rendszer architektúra . . . . .	23

5.2.	A processzor tervezése . . . . .	23
5.2.1.	A processzor utasításkészlete . . . . .	26
5.2.2.	A processzor felépítése . . . . .	27
5.3.	A fordítóprogram tervezése . . . . .	38
5.4.	Az emulátor tervezése . . . . .	40
<b>6.</b>	<b>Hardveres tesztek</b>	<b>45</b>
6.1.	Kommunikáció az analóg-digitál, digitál-analóg interfészekkel . . . . .	46
6.2.	Ki/Bemeneti műveletek tesztelése, analóg jel mintavételezéssel . . . . .	47
6.2.1.	Színusz jel mintavételezése . . . . .	47
6.2.2.	Négyszög jel mintavételezése . . . . .	50
6.3.	Aritmetikai műveletvégzési teljesítményét felmérő tesztek . . . . .	52
6.3.1.	Faktorizáció, sorozatos összeadással . . . . .	53
6.3.2.	Faktorizáció, beépített szorzó áramkörrel . . . . .	54
6.3.3.	Fibonacci számsorozat generálása 8 biten . . . . .	54
6.3.4.	Fibonacci számsorozat generálása 16 biten . . . . .	55
<b>7.</b>	<b>Eredmények</b>	<b>56</b>
7.1.	Aritmetika műveletvégzési teljesítmény felmérő tesztek kiértékelése . . . . .	56
7.1.1.	Faktorizációs tesztek kiértékelése . . . . .	57
7.1.2.	Fibonnacsi tesztek kiértékelése . . . . .	58
7.2.	A processzor erőforrás igényének kiértékelése . . . . .	60
<b>8.</b>	<b>Összefoglalás</b>	<b>61</b>
8.1.	Továbbfejlesztési lehetőségek . . . . .	61
8.2.	Köszönetnyilvánítás . . . . .	62
	<b>Irodalomjegyzék</b>	<b>62</b>
<b>A.</b>	<b>Függelék</b>	<b>65</b>

# Ábrák jegyzéke

5.1. Az utasításszerkezet felbontása . . . . .	26
5.2. Az utasítás betöltő egység részletes tömbvázlata . . . . .	28
5.3. Az adatmemória tömbvázlata . . . . .	31
5.4. I/O műveletek órajel felbontása . . . . .	32
5.5. I/O modul szerkezeti felépítése . . . . .	32
5.6. Az ALU ram műveleteinek órajelszükséglete . . . . .	34
5.7. Az ALU szerkezeti tömbvázlata . . . . .	34
5.8. Az utasítások csoportosítása . . . . .	35
5.9. Utasítások órajel-szükséglete (betöltési és végrehajtási fázissal) . . . . .	36
5.10. Aritmetikai utasítások órajelszükséglete . . . . .	37
5.11. Példa általános órajel-szükségletre . . . . .	38
5.12. Az fordító osztálydiagrammja . . . . .	39
5.13. Fordítóprogram futása . . . . .	40
5.14. Az emulátor osztálydiagrammja . . . . .	41
5.15. pBlaze IDE utasítás szimulátor . . . . .	42
5.16. Saját processzor emulátor . . . . .	43
5.17. Emulátor használati eset diagramja . . . . .	44
6.1. AD interfész működése . . . . .	46
6.2. PicoBlaze, 100hz szinusz jel . . . . .	48
6.3. Saját processzor, 100hz szinusz jel . . . . .	48
6.4. PicoBlaze, 1Khz szinusz jel . . . . .	48

6.5. Saját processzor, 1Khz szinusz jel . . . . .	48
6.6. PicoBlaze, 10Khz szinusz jel . . . . .	49
6.7. Saját processzor, 10Khz szinusz jel . . . . .	49
6.8. PicoBlaze, 50Khz szinusz jel . . . . .	50
6.9. Saját processzor, 50Khz szinusz jel . . . . .	50
6.10. PicoBlaze, 100Khz szinusz jel . . . . .	50
6.11. Saját processzor, 100Khz szinusz jel . . . . .	50
6.12. PicoBlaze, 5khz négyszög jel . . . . .	51
6.13. Saját processzor, 5khz négyszög jel . . . . .	51
6.14. PicoBlaze, 50khz négyszög jel . . . . .	51
6.15. Saját processzor, 50khz négyszög jel . . . . .	51
6.16. PicoBlaze, 100khz négyszög jel . . . . .	52
6.17. Saját processzor, 100khz négyszög jel . . . . .	52
6.18. PicoBlaze, faktorizáció összeadással . . . . .	53
6.19. Saját, faktorizáció összeadással . . . . .	53
6.20. Saját processzor, faktorizáció beépített szorzó áramkörrel . . . . .	54
6.21. PicoBlaze, fibonacci számsor generálás 8 biten . . . . .	55
6.22. Saját processzor, fibonacci számsor generálás 8 biten . . . . .	55
6.23. PicoBlaze, fibonacci számsor generálás 16 biten . . . . .	55
6.24. Saját processzor, fibonacci számsor generálás 16 biten . . . . .	55
7.1. Faktorizáció kiértékelése . . . . .	57
7.2. Fibonacci 8bit, eredmények . . . . .	58
7.3. Fibonacci 16bit, eredmények . . . . .	59
7.4. Erőforrás használat, PicoBlaze . . . . .	60
7.5. Erőforrás használat, PicoBlaze . . . . .	60

# Táblázatok jegyzéke

5.1. A két processzor szerkezeti összehasonlítása . . . . .	25
7.1. FaktORIZÁCIÓ MIPS ÉRTÉKE . . . . .	57
7.2. Fibonacci MIPS értéke . . . . .	59

# 1. fejezet

## Bevezető

A mai rendkívül gyorsan fejlődő világunkban elengedhetetlen a számítógépes eszközök mindennapi használata. A számítógépek egyik legfontosabb alkotóeleme a processzor, ez a komponens felelős a műveletek gyors és helyes elvégzéséért. Napjainkban már nagyon nagy sebességre képesek a processzorok. A zsebünkben lévő telefon már 1000-szer nagyobb számítási kapacitással rendelkezik mint az a processzor ami a holdraszállás algoritmusát hajtotta végre [1].

Napjainkban a processzorok családja igen csak széles. Rengeteg különböző processzor közül választhatunk, mindegyik más és más feladat elvégzésére van kiélezve. Vannak amelyek kevés energia felvétellel működnek, mint például a telefonokban, található processzorok. Néhány processzor 128 maggal képes rengeteg művelet párhuzamos elvégzésére, ilyenek találhatóak a szervergépekben. A processzorok két fő architektúra családba oszthatóak, Harvard- és Neumann architektúra.

Ugyanakkor a processzorok megkülönböztethetőek az utasítások száma szerint is. A RISC (Reduced Instruction Set Computing) processzorok kevés és egyszerű utasítással rendelkeznek, ezért ezek gyorsan végrehajthatóak. Ilyen típusú processzorok a telefonokban található ARM processzorok is. Egy másik változata a CISC (Complex Instruction Set Computing). CISC típusú processzorok nagy számú és komplex utasítással rendelkeznek. Előnye, hogy mivel több utasítást tartalmaz, a program kód rövidebb és könnyebben átlátható. Hátránya a RISC struktúrával szemben az, hogy hiába rövidebb az utasítás kód, a komplex műveletek elvégzése hosszabb lesz, illetve a processzor mérete is nagyobb, amely így több energiát is fogyaszt. A tervezés nehézségét figyelembe véve a RISC egy könnyebben és gyorsabban megvalósítható processzor, hiszen jóval egyszerűbb a processzor felépítése, ellentétben

egy CISC típusú processzorral szemben.

Napjainkban egyik legfontosabb tervezési minta a csövezeték elv, ennek köszönhetően egy processzor, egyszerre képes több utasításon is dolgozni.

A processzorok tervezését nagyban megkönnyíti egy FPGA (Field-programmable gate array) lap használata. Az FPGA egy olyan eszköz amely programozható blokkokat tartalmaz, illetve a blokkok közötti összeköttetést. Ennek köszönhetően az FPGA chip belsejében könnyedén és gyorsan képesek vagyunk különböző komplex áramkörök megvalósítására és tesztelésére.

A dolgozatomban egy általam implementált processzort fogok bemutatni. Amely megvalósítása során a Harvard architektúra elveit használtam, mely működése abban áll, hogy az adat és a program memória külön sínre használ, így az utasítások mérete eltérő lehet az adatsín méretétől [2]. A processzor a RISC tervezés elveire épül, csupán néhány egyszerű és gyorsan végrehajtható utasítást tartalmaz.

## 2. fejezet

### Célkitűzések

A projekt fő célja összehasonlítást készíteni a Xilinx PicoBlaze processzor és az általam fejlesztett Harvard architektúrára épült processzor között, számítási valamint folyamat vezérlési teljesítmény szempontjából.

A cél megvalósításának érdekében az első feladatom a saját tervezésű processzorom alkotóelemeinek a megtervezése és implementálása VHDL viselkedés leíró nyelv használatával és egy FPGA lap segítségével.

A tervezés után következik a processzor moduljainak implementálása és tesztelése a helyes működés érdekében.

Ezek után következhet a komponensek összekapcsolása, majd a kész rendszer tesztelése, különböző szimulációk segítségével.

Következő lépés a két különböző processzor összehasonlítása aritmetikai művelet végzéssel illetve analóg jel mintavételezés segítségével.

Fontosnak tartottam, hogy a processzor fejlesztés egy kiterjedtebb folyamatát valósítsam meg, így a processzor mellé készítettem egy fordító programot, JAVA nyelven, amely a processzor Assembly kódjából képes a processzor által futtatható bytekodót generálni. Fontos, hogy a fordító program képes legyen könnyedén az általam megvalósított processzor és a Xilinx PicoBlaze processzor nyelvére is fordítani. Ugyanakkor szeretnék egy egyszerű hiba kijelzési modót is implementálni, amely kijelzi a lehető legtöbb hibát a kódban, illetve amelyek megoldása elvégezhető a program által, azokat ne jelezze ki mint hiba, hanem oldja meg fordítás közbe, ezzel megkönnyítve a felhasználók dolgát.



Ugyanakkor a programkódok fejlesztésének érdekében egy emulátort is szerettem volna implementálni C# nyelven. A program segítségével sokkal könnyebben lehet majd a programokat tesztelni, hiszen nem szükséges egy FPGA lap. Az emulátor képes lesz szimulálni a processzor teljes belső szerkezetét, így megfigyelhető lesz minden regiszter értéke minden utasítás után.

Az emulátor segítségével szeretnék több processzort összekapcsolni, így szimulálható lenne egy kommunikáció a két processzor között, ezzel bemutatva a különböző kommunikációs protollokat.

A processzor további tesztelése során szeretnék hang felvételt készíteni és visszajátszani egy fejhallgató segítségével. Illetve megfigyelni, hogy melyik lenne az a legmagasabb baud rate mivel a processzor még képes kommunikálni 50Mhz-es órajel esetén.

A projekt egy nagyobb részt körül fog a processzorok témában, így akár használható lesz az egyetemi oktatásban is, hiszen a processzor tervezés és implementálás lépései kimutathatóak a VHDL kód használatával. Illetve az Assembly programozásba is betekintést lehet nyerni a fordító program segítségével.

## 3. fejezet

### Szakirodalom áttekintése

A világhálón való keresés során több tanulmányt is találtam amely processzor implementálást mutat be, különböző szempontok és architektúrák szerint. A következőkben bemutatok néhány hasonló témájú processzorokról szóló tanulmányt.

#### 3.1. Design & Analysis of 16 bit RISC Processor Using low Power Pipelining

Az általam tervezett processzorhoz hasonló architektúrát alkalmaztak 2015-ben a Galgotias Egyetem tanulói [3].

Egy 16 bites RISC processzort készítettek Harvard architektúra elveire alapozva. Hasonlóan az általam fejlesztett processzorhoz ők is szintén egy két fázisú csövezetékessítést alkalmaztak. Az általuk fejlesztett processzor 33 különböző utasítás végrehajtására volt képes.

Számukra a legfontosabb a processzor áramfelvétele volt egy fontos tényező. Verilog nyelv használatával igyekeztek a fogyasztást minimálisra csökkenteni egy XILINX KINTEX XC7K1607-3fbg676 FPGA lap használatával.

A tanulmányuk során arra jutottak, hogy a fogyasztás nagyban függ a Verilog nyelvből szintetizált kódtól, amely minden futás során egy kissé változó eredményt ad.

### 3.2. 16-Bit RISC Processor Design for Convolution Application

Szintén egy RISC típusú processzort terveztek a SSN College of Engineering, Kalavakkam-i egyetemen, 2011-ben [4].

A tanulmányban egy 16 bites RISC nem csővezetékesített processzort terveztek load/store és Neumann architektúra alkalmazásával. A processzor összesen 27 utasítást támogatott amely bővíthető 32-re is, ezek aritmetikai és logikai utasítások, ugrási utasítások, load/store utasítások illetve halt utasítás. A megvalósítás Verilog-HDL nyelv használatával történt. A processzort úgy tervezték, hogy képes legyen bármelyik utasítást egyetlen órajel alatt végrehajtani.

Ebben a tanulmányban is a processzor energiafelvételének a minimalizálása volt fontos a tervezők számára.

A szimulációk során az is kiderült, hogy a processzort 200 Mhz-es órajelen is képes működni.

A tanulmány során arra a következtetésre jutottak, hogy a processzor alkalmazható jelfeldolgozásra is.

### 3.3. FPGA Prototyping of a RISC Processor Core for Embedded Applications

Egy 2001-es tanulmányban amelyet Michael Gschwind és Valentina Salapura készített. Egy általuk tervezett RISC típusú processzort terveztek és valósítottak meg [5].

A processzor a csővezeték elveire épült egy állapotautomata segítségével. A processzort modulárisan tervezték, hogy könnyen lehessen új utasításokkal bővíteni.

A processzor képes logikai és aritmetikai műveletek elvégzésére, tömbökkel való műveletvégzésre, illetve egyszerűbb fuzzy műveletek elvégzésére.

A tanulmány során több különböző szimulációs tesztet hajtottak végre, mint az órajel szimulálás, amikor órajel ütemekre vizsgálták a processzor helyes működését. Funkcionális szimuláció során készítettek egy felületet amelyben figyelni tudták a processzor jelenlegi állapotát illetve az utasítások utáni állapotát.

A következő részben egy fizikai FPGA-ra töltötték fel a processzort, majd ennek segítségével

végeztek különböző fizikai teszteket.

A tanulmányban arra jutottak, hogy az általuk fejlesztett processzor megfelel egy alap processzornak amely könnyedén bővíthető, így egyszerűen átalakítható egy applikáció specifikus processzorrá.

### **3.4. Implementation of RISC Processor on FPGA**

Ebben a tanulmányban Pravin S. Mane, Indra Gupta és M. K. Vasantha valósítottak meg egy 16 bites RISC processzort, amely egy 5 fázisú csővezeték elvén működött [6].

A processzor egy ki és egy bemeneti portot tartalmaz illetve 6 hardveres megszakítást képes megkülönböztetni.

Adatelérés szempontjából Load/Store architektúrát alkalmaztak. A processzor 37 darab 16 bites utasítással rendelkezik. Egy utasítás szerkezete 5 részből áll. Az első 4 bit az utasítás kódja, ezt követi a cél register címe, majd az első és a második regiszter címe, végül pedig az utolsó 3 bitt az utasítás különböző funkcióit jelenti.

A következő részben különböző általuk írt programokat futtattak a tervezett processzoron, majd ezek eredményét vizsgálták. Ebből kiderül, hogy az általuk fejlesztett processzor képes egy órajel alatt egy utasítást végrehajtani. Illetve a maximális órajel 26 Mhz egy Xilinx Spartan 2-es FPGA-t használva.

## 4. fejezet

# Technológiai áttekintés

A célkitűzéseknek megfelelően kerestem a megvalósításhoz szükséges technológiákat. A következőkben ezekről szeretnék röviden beszélni.

### 4.1. Hardver

#### 4.1.1. FPGA

Az FPGA (field-programmable gate array) egy olyan félvezető eszköz, amely logikai blokkokat tartalmaz. Ezek a blokkok programozható logikai modulokat és azok programozható összeköttetéseit tartalmazzák [7].

A projekt során egy Digilent Nexys 2 FPGA lapot használtam. Választásom azért erre az eszközre esett, mert eleget tett a projekt minimális követelményeinek, illetve az egyetem már rendelkezett vele.

A Digilent Nexys 2 FPGA lap a következő specifikációkkal rendelkezik [8]:

- 1200K-gate Xilinx Spartan 3E FPGA
- USB2-based
- 16 MB Micron PSDRAM&16MB
- Xilinx Platform Flash

- 50 MHz oszcillátor
- 60 FPGA I/O csatlakozó
- USB-powered
- 8 darab led, 4 darab digitális 7 szegmenses kijelző, 4 db. nyomógomb, 8 darab csúszókapcsoló

### 4.1.2. VHDL

A VHDL egy az áramkörök specifikus leírására szolgáló nyelv, más néven hardverleíró nyelv. Segítségével könnyedén lehet az FPGA-ban található blokkokat és összeköttetéseket programozni [9].

A processzor implementálása során a VHDL nyelvet használtam hiszen ebben már volt tapasztalatom illetve több dokumentációt is találtam róla, ugyanakkor az összehasonlítani kívánt Xilinx processzor is ebben a nyelvben volt megírva.

## 4.2. Assembly fordító

A következőkben a fordítóprogram fejlesztése során használt technológiákról szeretnék beszélni.

### 4.2.1. Java

A Java egy általános célú, objektumorientált programozási nyelv. A java alkalmazások általában bájt-kód formájában vannak tárolva, de akár natív gépi kódra is lehet fordítani. A bájtkódot egy Java virtuális gép (JVM) futtatja, amely vagy egy interpreterként végrehajtja a bájtkódot, vagy natív gépi kódot állít elő belőle, amelyet majd az operációs rendszer futtat [10].

A fordító program megírása során a Java nyelvet választottam, mert napjainkban egy nagyon elterjedt nyelv illetve a futási sebessége megközelíti egy C program sebességét. Ugyanakkor rendkívül könnyen és gyorsan lehet benne programozni, hiszen rengeteg könyvtár áll a felhasználó rendelkezésére. Ugyanakkor egy másik nagy előnye a hordozhatóság, napjainkban szinte minden számítógép rendelkezik JVM (Java virtual machine) programmal amely segítségével bármilyen operációs rendszeren futtatható a fordítóprogram.

## 4.3. Processzor emulátor

### 4.3.1. C#

A C# a Microsoft által a .NET keretrendszerhez fejlesztett objektumorientált nyelv [11]. A nyelv alapja a C++ és a Java volt. A nyelv a .NET keretrendszer alapnyelve, átlátható és gyors, lehetőséget biztosít ahhoz, hogy a .NET keretrendszer alá különböző alkalmazásokat készítsünk [12].

A nyelv néhány lényegesebb jellemzői [12].:

- Professzionális, nagy programok, akár rendszerprogramok írására is alkalmas.
- Nincs destruktork, helyette a keretrendszer szemétgyűjtési algoritmus van. Ha szükséges az osztály Dispose metódusa újradefiniálható
- Függvényparaméterek lehetnek: érték, referencia (ref) és output(out).

### 4.3.2. .NET keretrendszer

A .Net keretrendszer gyors alkalmazásfejlesztést tesz lehetővé, ugyanakkor képes platformfüggetlen alkalmazások fejlesztésére is. A keretrendszer eszköztára nagyon széles, lehet kliens illetve szerveroldali alkalmazásokat is készíteni, adatbázisok kezelése [13].

### 4.3.3. Windows Forms

A Windows Forms egy nyílt és ingyenes forráskódú grafikus felhasználói felület könyvtár. A .Net keretrendszer egyik alrész. Lehetővé teszi, hogy különböző platformokon egységes felületek tervezését és megvalósítását. Ezek az alkalmazások egy eseményvezérelt programozási mintával valósulnak meg, amelyben a .Net keretrendszer segít. Gyors és egyszerű tervezést biztosít a programozók számára C# nyelvben [14].

A fentebb említett C# és .Net keretrendszer technológiákat használva valósítottam meg a processzor emulátort illetve egy még Béta fázisban lévő felhasználói felületet.

# 5. fejezet

## Tervezés

### 5.1. A rendszer architektúra

A teljes rendszer három fő részből áll, az FPGA lapon megtervezett és implementált processzor, a Java nyelven írt fordító program, illetve egy processzor emulátor amely rendelkezik felhasználó felülettel.

### 5.2. A processzor tervezése

Az általam tervezett és megvalósított processzort a Xilinx PicoBlaze soft core processzorról mintáztam, a Harvard architektúra elveit alkalmazva [15].

A processzorok általános tervezési lépései:

- Digital Logic Level
- Micro-architecture level
- Instruction level architecture
- Operating System
- Assembly Level
- Problem Oriental language level



A Harvard architektúra elült processzorok esetén a tervezés két részre osztható: az utasítások meghatározása és a feladataik definiálása, a mikroarchitektúra komponensek tervezése és implementálása. Jelen esetben az utasítások 18 bitből állnak, amelyből az első 6 bit az utasítás kódja, a következő 12 pedig utasítástól függően változik, mint a regiszterek címe, értékadás és elágazási biteket definiálnak. [16]

Egy általános processzor a következő komponenseket tartalmazza:

- Általános célú regisztertömb: két utasítás között képes az értékek tárolására
- Utasítás dekódoló egység: felbontja a beérkező utasításokat
- Aritmetikai/Logikai egység: az aritmetikai és logikai műveletek elvégzésért felelős
- Ki/Bemeneti egység: kapcsolatot teremt a processzor és a külső perifériák között
- Vezérlő egység: az utasításoknak megfelelően vezérlő jeleket biztosít a processzor többi komponensének

A RISC egy tervezési stratégia, ami a csökkentett számú és egyszerű utasítások alkalmazását teszi lehetővé. Az első RISC típusú gép 1980-ban jelent meg, amikor ténylegesen meg tudták valósítani, azt a tervet hogy egy chipben elkészítettek egy processzort. A RISC stratégia magába foglalja az utasítások hosszának rövidítését, a nagyméretű regisztertárat mivel így több utasítást tudnak eltárolni. Az ilyen típusú processzoroknál egy nagy hátrány van, a bonyolultabb műveleteket csak több lépésben tudja megoldani [17].

Az általam fejlesztett processzor a következő komponensekből épül fel:

- Regisztertömb
- Utasítás betöltő és dekódoló egység
- Aritmetikai és logikai egység (ALU)
- Ki/Bemeneti egység (I/O)
- Block Ram

Utasítás betöltő és dekódoló egység beolvassa az utasításokat és eldönti, hogy a beolvasott utasítás elágazást kepző vagy ugrási utasítás-e (CALL, JUMP, RETURN). Ha a beolvasott utasítás ugrási utasítás akkor a feltételnek megfelelően elvégzi az előre definiált utasítást. Amennyiben nem ugrás/elágazás kepző utasítás érkezett akkor az utasítást továbbküldi az Aritmetikai és logikai egységnek illetve a Ki/Bemeneti egységnek amelyek elvégzik az utasítás rájuk eső részét.

A processzor megvalósítása során alkalmaztam a csővezeték elvet is amely abban áll, hogy a processzor egyszerre több utasításon is képes dolgozni párhuzamosan így jobban kihasználva a komponenseket, ez által egy utasítássorozat kevesebb órajel alatt is képes végrehajtani.

Az analóg jel mintavételezés során szükség volt analóg-digitál és digitál-analóg interfészek implementálására is amelyel kapcsolatot teremtenek a processzor és a külső analóg-digitál és digitál-analóg konverterek között.

Az általam VHDL nyelven fejlesztett processzort - melynek tömbvázlata a dolgozat mellékletében látható - a Xilinx PicoBlaze processzorról mintáztam. Hasonló utasításkészletet használtam illetve a ki és bemeneti struktúra is megegyezik a két processzor esetében.

Az alábbi táblázatban megfigyelhető a két processzor közötti különbségek és hasonlóságok:

	PicoBlaze	Saját processzor
Utasítások mérete	18 bit	18 bit
Verem memória	32x10 bit	32x10 bit
Regisztertömb mérete	16x8 bit	16x16
Adatmemória mérete	64x8 bit	1024x16 bit
Szorzó áramkör	0 db	1 db

5.1. táblázat. A két processzor szerkezeti összehasonlítása

Mint ahogyan a 5.1. táblázatban is megfigyelhető a két processzor ugyanakkora utasítás- és verem mérettel rendelkezik. Ugyanakkor különbség figyelhető meg a regiszter tömb mérete között. Az általam fejlesztett processzor 16 bites számokkal képes műveletet végezni, míg a PicoBlaze processzor csupán 8 bites számokkal képes dolgozni így a regisztertömb mérete kétszer nagyobb. Adatmemória szempontjából is jelentős különbség figyelhető meg, hiszen az általam tervezett processzor 32x-er na-

gyobb adat memóriával rendelkeznek. Szintén egy jelentős különbség a két processzor között, hogy az általam implementált processzor tartalmaz egy beépített szorzó áramkört is amely jelentősen csökkenti a szorzási műveletekhez szükséges időt.

A processzor tervezése során fontosnak találtam, hogy a lehető legkevesebb órajelre legyen szükség egy utasítás elvégzésére, ezzel tovább gyorsítva a processzort. Ennek köszönhetően a processzort alkotó legtöbb modul akár önmagában is el tudja végezni az utasítás rá jutó részét. Így nem szükséges egy külön vezérlő egység amely a többi modul szinkronizálását és vezérlését végzi.

Az adatmozgató, aritmetikai és logikai utasításokat az Aritmetikai és logikai egység végzi. Az adatok gyors elérése érdekében a modul magába foglalja a 16 darab 16 bites regisztert amelyek az adatok ideiglenes tárolását végzik. Így az Aritmetikai és logikai egység képes egyetlen órajelen alatt kiolvasni a regiszter tartalmát elvégezni egy egyszerűbb műveletet majd az eredményt eltárolni a megfelelő regiszterbe.

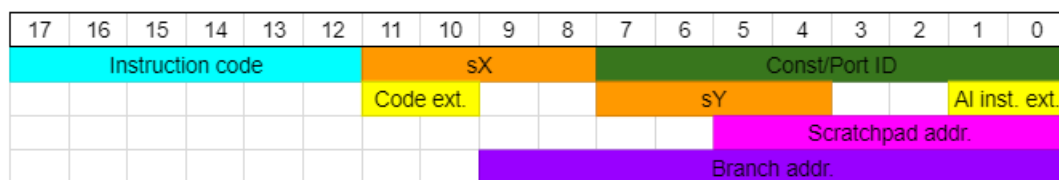
### 5.2.1. A processzor utasításkészlete

Az általam tervezett processzor a RISC tervezési stratégiát alkalmazva kevés és egyszerű utasítást tartalmaz. Összesen 50 darab.

Ezeket az utasításokat három csoportba sorolhatjuk:

- Adatmozgató utasítás (10 db)
- Aritmetikai-logikai utasítás (28 db)
- Ugrási/függvényhívási utasítás (12 db)

Az utasítások szerkezete az 5.1 ábrán figyelhető meg:



5.1. ábra. Az utasításszerkezet felbontása

Ahogy az 5.1 ábrán is látható, az utasítás első hat bitje minden esetben az utasítás kódja a következő bittek az utasítástól függően változnak.

Az utasítások Assembly szerkezete:

[cimke:] <utasítás neve> {flag} {változó1}, {változó2}

- < > kötelező
- [ ] opcionális
- { } utasításfüggő

Példa utasítás sorozatra:

Listing 5.1. Assembly kód példa

```
LOAD S0, 00
LOAD S1, 05
LOAD S2, 08

loop:
ADD S0, S1
SUB S2, 01
JUMP NZ, loop

OUTPUT S0, 01
```

A fenti utasítássorozat kiszámolja az S1 és S2 regiszterek szorzatát, az eredményt eltárolja az S3 regiszterbe majd kiküldi a 01-es porton egy külső perifériára.

## 5.2.2. A processzor felépítése

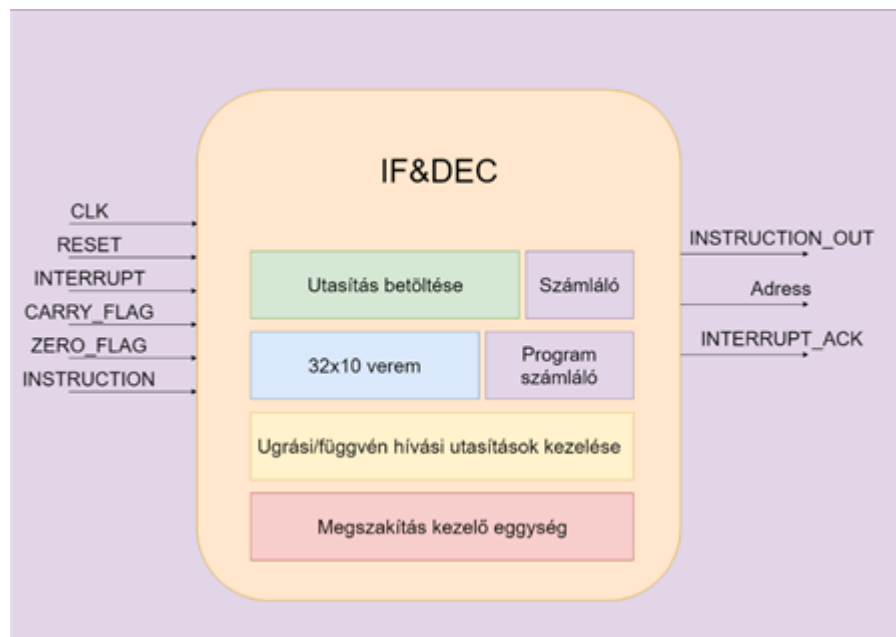
Az általam tervezett processzor a következő főbb elemekből áll:

- Utasítás betöltő és dekódoló egység: Kiolvassa az utasítást a program memóriából majd az Aritmetikai és logikai egység által szolgáltatott jelző bitek segítségével eldönti, hogy ugrás hajtódik-e végre, ha igen akkor a programszámlálót elmenti a verem memóriába. Ezután továbbküldi az utasítást az Aritmetikai és logikai egységnek illetve a Ki/Bemeneti egységnek. Magába foglalja a 32 mélységű verem memóriát, amelyben a programszámláló mentődik el, ugrás vagy megszakítás esetén.

- Aritmetikai és logikai egység (ALU): az adatmozgatás és az aritmetikai illetve logikai műveletek helyes elvégzéséért felelős egység. Tartalmazza a regisztertömböt melynek köszönhetően gyorsan képes hozzáférni az adatokhoz. Ugyanakkor a Zéró és Túlszordulást jelző bittekért is felelős. Közvetlen kapcsolatban áll az adat memóriával.
- Ki/Bemeneti egység: kapcsolatot teremt a külső perifériák és processzor között, az adatokat egy 16 bites adatsínen továbbítja a belső regiszterekbe.
- Adatmemória: Az FPGA Block Ram modulját felhasználva valósítottuk meg, így 1024x16 bitnyi adatot képes tárolni.

### Utasítás betöltő és dekódoló egység

Az utasítások betöltéséért felelős modul, magába foglalja a programszámláló vermet amelybe ugrás illetve megszakítás esetén eltárolódik a programszámláló értéke, szerkezete a 5.2 ábrán látható.



5.2. ábra. Az utasítás betöltő egység részletes tömbvázlata

Az utasítás betöltő és dekódoló egység a következő feladatokat látja el:

- Utasítások betöltése a programmemóriából

- Utasítások ütemezése
- Függvényhívási és ugrási utasítások kezelése
- Megszakítások kezelése

Minden órajelciklus elején ellenőrzi a processzor állapotát, ha a processzor képes új feladatot fogadni akkor megnézi, hogy történt-e megszakítás ha igen akkor működésbe lép a megszakítást kezelő rutin. A programszámlálót elmenti a verembe, majd a megszakítás rutin címére állítja be. Ha nem történt megszakítás akkor betölti a következő utasítást. Majd beállít egy belső számlálót amely tárolja, hogy hány órajelciklusra van szüksége a processzornak az adott utasítás végrehajtásához. A számláló értéke minden órajelciklusban eggyel csökken, ha eléri a 0 értéket a processzor készen áll a következő utasítás betöltésére, vagy a következő megszakítás kezelésére. Ha ugrási vagy függvényhívási utasítás érkezett azokat elvégzi és a programszámlálót beállítja a megfelelő értékre.

Listing 5.2. Assembly kód példa

1. LOAD S0 , 12	2
2. SUB S0 , 12	3
3. CALL Z , ZERO	5
LOOP:	
4. JUMP LOOP	4
ZERO:	
5. LOAD S1 , 01	6
6. RETURN	4

A fenti utasítás sorozatot figyelve, első részben betölti az S0 regiszterbe a 12 es értéket, majd a második utasításban ki is vonja ugyanazt az értéket. Így Az S0 regiszter tartalma 0. A 3. utasítás egy feltételes függvényhívási utasítás, amely akkor hajtódik végre, ha az előző utasítás műveleti eredménye 0 volt. Ennek következtében a verembe elmenti a következő utasítás címét a 4-et, majd a program számlálót beállítja 5-re. A következő órajel esetén a programsorozat már a 5. utasítástól fog folytatódni. A RETURN utasítás kilép a függvényből és visszaállítja a lementett utasítás címet, így a program a CALL függvény után fog folytatódni a 4. utasítással, amely egy végtelen ciklus. [16]

## Adatmemória

A PicoBlaze processzorhoz képest egy nagyobb adatmemóriát használtam, mivel az Xininx processzor 8 bites számokkal képes dolgozni addig az általam fejlesztett processzor 16 bites számokat képes

a registereiben tárolni. Így már dupla az eltárolható adatok mennyisége register szinten, növeltem az adatmemória szavak számát, az eredeti PicoBlaze processzor 64 mélységű volt addig az általam fejlesztett processzor 1024 szavat volt képes eltárolni.

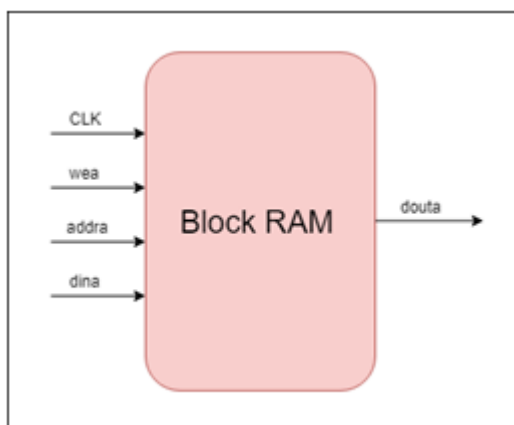
Az első próbálkozásom során regisztereket használva próbáltam megvalósítani az adómemóriát. Ez egy gyorsabb memóriát eredményez, így hasonlóan az ALU-ban használt regiszter tömbhöz, az adatokat gyorsan, két órajel alatt tudjuk mozgatni, írási és olvasási műveleteken keresztül.

Viszont ez a megoldás egy nagy hátránnyal rendelkezett, mivel regisztereket használtam az adatmemória megvalósítása során ezért ez nagy erőforrás igénytel járt így az FPGA 30-31%-át használta a processzor.

A fenti megoldás gyors volt, viszont rengeteg erőforrást használt az FPGA kapacitásából, ezért más módszert alkalmaztam, amely lassabb, de jelentősen lecsökkenti erőforrásigényt.

A következőkben az adatmemória megvalósításához az FPGA-n található Block Ram modulokat használtam amelyek képesek eltárolni a szükséges adatmennyiséget és nem foglal olyan értékes erőforrást amelyet másra is lehetne használni. Mivel ezt a dedikált modult használtam, a megoldás lassúnak bizonyult, 3 órajelre volt szükség egy írás vagy olvasás végrehajtásához, viszont jelentősen kevesebb erőforrást használt.

Mivel a processzor 18 bites utasításokkal dolgozik. Direkt címzés esetén, amely első 6 bitje maga az utasítás, a következő 4, hogy melyik regiszterbe tölti a memória tartalmát vagy éppen a regiszter tartalmát tölti a memóriába, az utolsó 8 bit, az adatmemória címe. Így direkt címzést alkalmazva csupán az első 64 memóriacímen lévő adatokat lehet megcímezni. Viszont regiszter címzés esetén mind az 1024 rekeszben található adat, gond nélkül elérhető, mind írási, mind olvasási művelet esetén.



5.3. ábra. Az adatmemória tömbvázlata

Ahogy az 5.3 ábrán is látható az adatmemória négy bemeneti jelet foglal magába (CLK, wea, addra, dina) és egy kimeneti jelet tartalmaz (douta). Minden felfutó órajelre képes egy adat írására vagy olvasására a memóriából. Az addra egy 10 bites jel, amely használatával meg lehet címezni az 1024 szó valamelyikét. A wea vezérlő jel segítségével irányíthatjuk, hogy éppen írni vagy olvasni akarunk a memóriából, wea = 0 esetén olvasunk és a douta (16 bites) kimenetre továbbítja a memóriában tárolt értéket, wea = 1 esetén a dina (16 bites) bemenetre küldött értéket tárolja el a megfelelő címen.

### Ki/Bementi egység

Az I/O kapcsolatot teremt a külső perifériák és a processzor között.

Az I/O utasítások az rendkívül időigényesek, mivel, ezek használják a processzorban a legtöbb modult és várniuk kell a külső perifériára is az adatok továbbítása érdekében.

I/O utasítás végrehajtása:

Mind írási és olvasási művelet esetén az első órajel alatt betöltjük az utasítást a program memóriába.

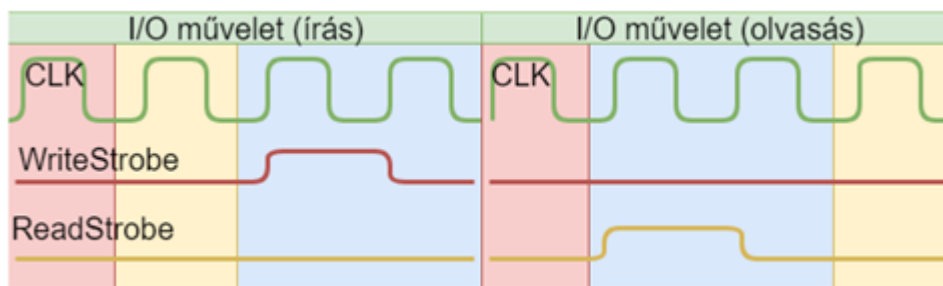
Írási utasítás esetén(5.4 ábra):

- A második órajel alatt kiolvasásra kerül az adat a regiszter tömbből.
- A harmadik, illetve negyedik órajele alatt, egyesre állítódik a WriteStrobe jelző bit, majd kiíródik az adat.

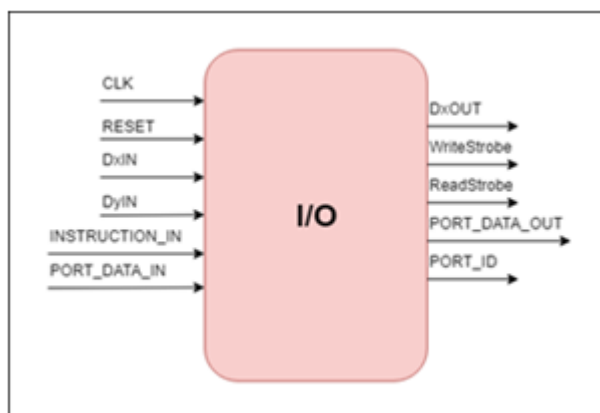


Olvasási utasítás esetén (5.4 ábra):

- A második és harmadik órajel esetén egyesre állítódik a ReadStrobe jelző bit, és beolvasásra kerül a perifériáról érkező adat.
- A negyedik órajel alatt, a beolvasott érték elmentésre kerül a megfelelő regiszterbe.



5.4. ábra. I/O műveletek órajel felbontása



5.5. ábra. I/O modul szerkezeti felépítése

Az I/O egység kapcsolatban áll az Aritmetikai és Logikai egységgel mivel abban található a regiszter tömb. Ennek köszönhetően minden I/O művelet esetén az ALU szolgáltatja a kimenő adatokat.

A 5.5 ábrán megfigyelhető a modul szerkezeti felépítése ki és bemeneti portjai. Bemenetként megkapja a regiszterek értékeit, DxIn, DyIn, az utasítást. Olvasás esetén a PORT\_DATA\_IN en érkezik a bemenet a külső perifériától. A kimeneti portjain pedig kiküldi az értéket, a port számot, valamint, hogy írási vagy olvasási művelet van és a beolvasott értéket a regiszterbe.

Négy utasítással használható: direkt címzéssel és regiszter címzéssel, egyaránt írási és olvasási műveletek.

### **Aritmetikai logikai egység**

Az aritmetikai és logikai műveletek végrehajtásáért felelős egység, a processzor egyik legnagyobb modulja.

Az egységnek három fő szerepe van:

- Aritmetikai és logikai műveletek végzése
- Kommunikáció az adatmemóriával
- Kommunikáció az I/O egységgel

Célom az aritmetikai műveletek gyors elvégzése volt, ezért a 16x16 bites regiszter tömböt, beépítettem az Aritmetikai és Logikai egységbe, így nem képes a modul nagyon magas órajelen működni, viszont képes egyetlen egy órajel alatt kiolvasni a regiszter értékét, elvégezni az adott aritmetikai műveletet, majd az eredményt elmenteni, a megfelelő regiszterben. Az általam használt FPGA lap 50 MHz el működik, ezen a frekvencián a processzor gond nélkül működik. [16]

Mivel az ALU tartalmazza a regisztertömböt ezért ez az egység vezérli az adat memóriával való kommunikációt, a STORE és a FETCH utasítások esetén. Három vezérlő jelen keresztül (Ram\_address, Ram\_out, Ram\_write).

Store művelet esetén:

STORE S4, 44 Az utasítás befejezéséhez három órajel periódusra van szükség (5.6. Ábra), az első az utasítás beolvasása a program memóriából, majd továbbküldése az Aritmetikai és Logikai egységnek. Második órajel periódus alatt kiolvasódik az S4 regiszter tartalma, és megjelenik az ALU Ram\_out kimenetén és a Ram\_address kimeneten az 44 érték, amely meghatározza, hogy melyik cellába fog beíródni az érték. A harmadik órajel alatt történik az adat mentése az adat memóriába. [16]

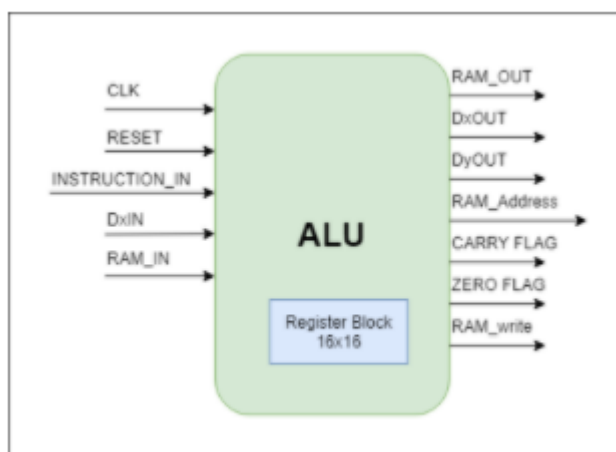


5.6. ábra. Az ALU ram műveleteinek órajelszükséglete

Az Aritmetikai és Logikai egység egy másik fontos feladata, az adatok továbbítása a Ki/Bemeneti egységnek port műveletek esetén. Mivel tartalmazza a regisztertömböt, ezért fontos a két modul közötti kommunikáció.

Egy port művelet elvégzéséhez négy órajelre van szükség. Az első az utasítás beolvasása a program memóriából, a második a regiszter tartalmának továbbítása a Ki/Bemeneti egység felé, a harmadik és negyedik órajel alatt történik a kiírás a perifériára.

Az Aritmetikai és logikai egység szerkezeti felépítése a 5.7 ábrán látható. Megfigyelhető több kimenet is, hiszen ez a modul tart kapcsolatot a regisztertömb és a Ki/Bemeneti egység valamint az adatmemória között.



5.7. ábra. Az ALU szerkezeti tömbvázlata

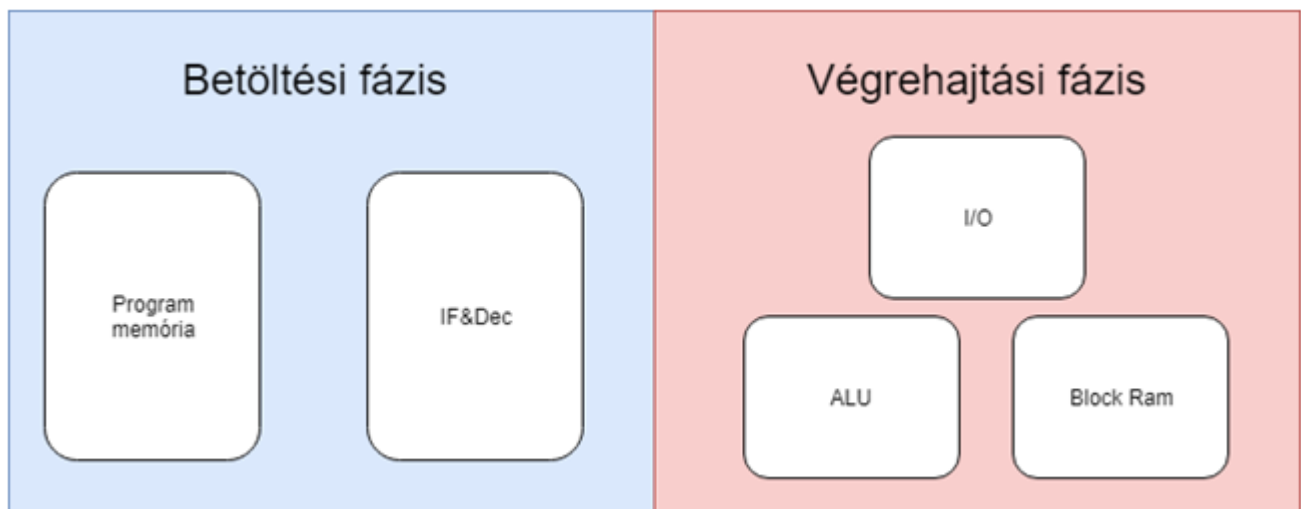
## A csővezeték elv alkalmazása a megépített processzorban

A processzor hatékonyságának növelésének érdekében (pipeline) csővezeték elvét alkalmaztam, így egyes moduloknak nem kell egymásra várniuk, hanem párhuzamosan végezhetik a saját utasításaikat.

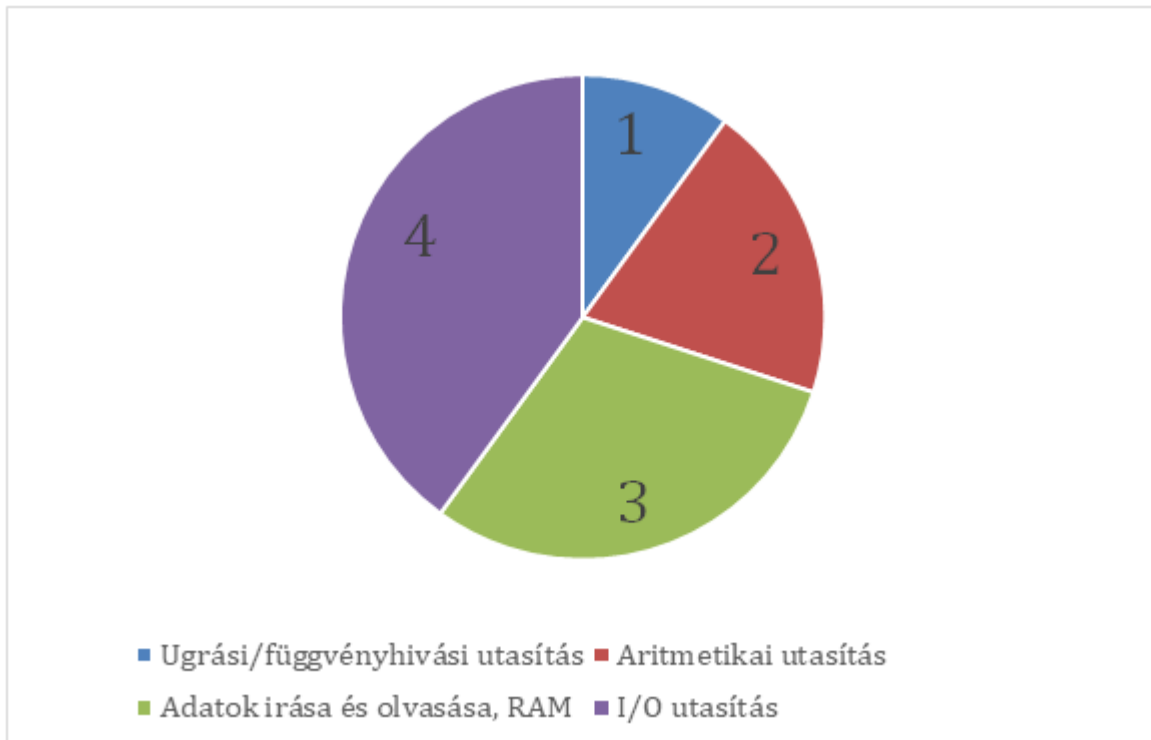
Az utasítások feldolgozása két fő részre osztható (5.8 Ábra, 5.9 Ábra):

- Az utasítás beolvasása a program memóriából és ha van ugrás akkor azok elvégzése.
- A második lépés az utasítás végrehajtása, és az eredmény eltárolása.

A processzor a legtöbb utasítást két órajel alatt képes végrehajtani.



5.8. ábra. Az utasítások csoportosítása



5.9. ábra. Utasítások órajel-szükséglete (betöltési és végrehajtási fázissal)

A csővezetékes struktúra alkalmazásának köszönhetően, mindig két utasítás elvégzése történik a processzorban. Az egyik a betöltési fázisban, a másik a végrehajtási fázisban van. Ennek következtében, az utasítások elvégzési sebessége jelentősen megnő. Például, 10 aritmetikai utasításhoz, csupán 9 órajelre van szükség. Tehát  $n$  egymást követő aritmetikai utasítás elvégzéséhez  $n+1$  órajelciklusra van szükség (5.10 Ábra). [16]

Órajel/ Utasítás	1	2	3	4	5	6	7
I. utasítás	Betöltés	Végrehajtás					
II. utasítás		Betöltés	Végrehajtás				
III. utasítás			Betöltés	Végrehajtás			
IV. utasítás				Betöltés	Végrehajtás		
V. utasítás					Betöltés	Végrehajtás	

5.10. ábra. Aritmetikai utasítások órajelszükséglete

A nagyobb órajelszükségletű utasítások esetén a processzor vár, mivel nem képes két utasítás egyszerre használni ugyanazt az erőforrást, egy adott utasítás sorozat esetén: aritmetikai → I/O → aritmetikai → RAM → aritmetikai, az utasítások végrehajtása órajel lépésekre bontva a 5.11 Ábrán látható. [16]

Órajel/ Utasítás	1	2	3	4	5	6	7	8	9
Aritmetikai utasítás	Betöltés	Végrehajtás							
I/O Utasítás		Betöltés	Végrehajtás	Végrehajtás	Végrehajtás				
Aritmetikai utasítás					Betöltés	Végrehajtás			
Ram utasítás						Betöltés	Végrehajtás	Végrehajtás	
Aritmetikai utasítás								Betöltés	Végrehajtás

5.11. ábra. Példa általános órajel-szükségletre

Jelen példában (5.11. Ábra) az utasítások egymás utáni végrehajtásához 13 órajel lépésre lenne szükség, viszont a csővezeték elvét alkalmazva ezt a processzor képes elvégezni 9 órajel ütem alatt. [16]

### 5.3. A fordítóprogram tervezése

A projekt során fontos volt egy hatékony és gyors fordítóprogram fejlesztése amely képes mindkét processzor nyelvére egyaránt fordítani. A Xilinx PicoBlaze processzorhoz társul egy fordító program is, kcpsm.exe. Viszont ez a program már elavult csak DOS emulátor használatával lehetséges a futtatása, illetve, csak egyetlen egy hibát képes kijelezni a fordítani kívánt kódból.

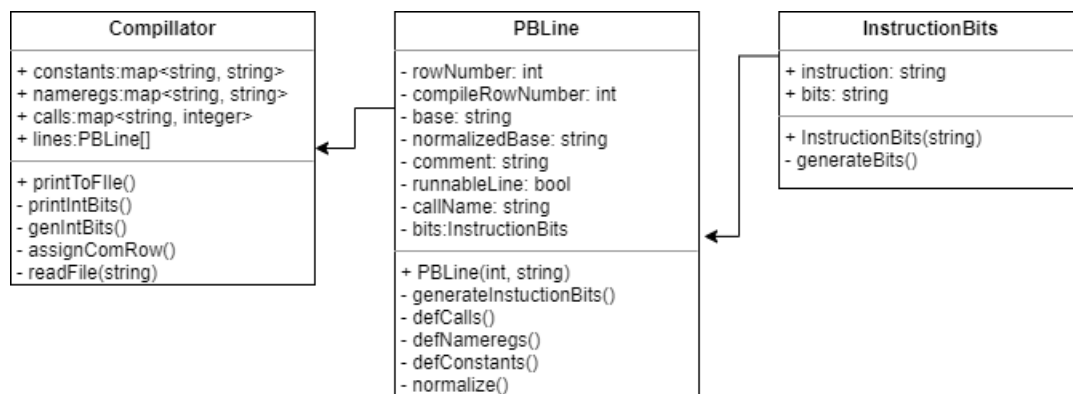
Ezek a megszorítások nagyban lassíthatják egy alkalmazás fejlesztését. Így fontosnak tartottam, hogy az általam írt fordító program képes legyen szinte bármilyen operációs rendszeren futni, illetve a lehető legtöbb hibát egyszerre kijelezni.

Választásom ezért a Java nyelv használatára esett, hiszen. Az általam írt kódot bármilyen gépen lehet futtatni abban az esetben ha az a számítógép képes egy Java virtuális gép futtatására.

A program rendkívül egyszerű, az indítás után megjelenik egy konzol ablak amely egy állomány nevét várja. Ha az állomány létezik akkor megnyitja és elvégzi a fordítást, amennyiben hibát talál jelzi azokat, ellenkező esetben egy out.vhdl fájlba menti a lefordított kódot.

Különbség figyelhető meg a két fordító viselkedése között. Az általam fejlesztett fordítóprogram rugalmas, ha talál egy hibát amely megoldható azt megoldja anélkül, hogy a fordítás sikertelen lenne, ilyen hibák az esetleges vessző, zárójel kihagyása, illetve már elnevezett regiszterek alapértelmezett nevének a használata.

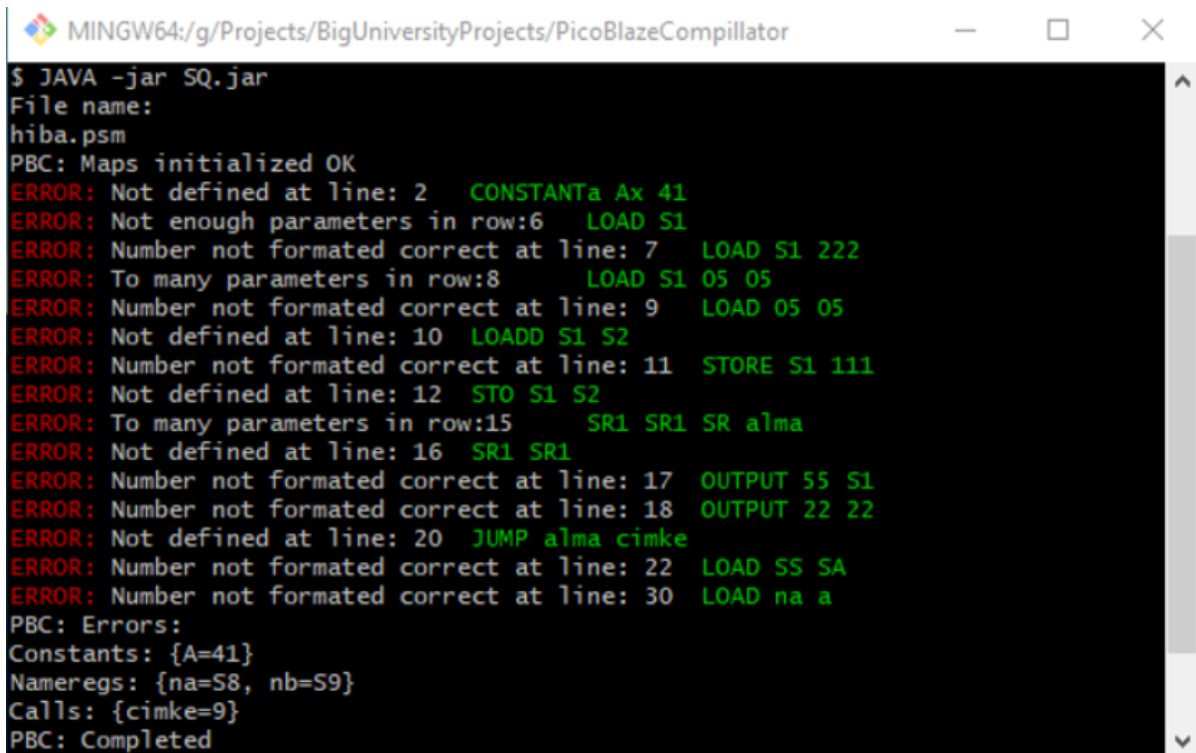
Ugyanakkor az általam fejlesztett program sokkal gyorsabban volt képes elvégezni a kód fordítását, így akár integrálni is lehet egy felületbe mely valós időben várakozás nélkül képes a kód futtatására, ezt felhasználva valósítottam meg későbbiekben a processzor emulátort.



5.12. ábra. Az fordító osztálydiagrammja

Az 5.12 megfigyelhető a fordítóprogram osztály diagramra. Látható, hogy egy fő komponensből áll. Amely tartalmaz egy másik osztályt is, a PBLLine osztály ez felelős egy utasítás értelmezéséért. Különválasztja a kommenteket a címkéket és a kódot. Majd ha ez megtörtént akkor az InstructionBits osztály segítségével ki generálódik a bájtkód. Ha nem volt hiba a fordítás közben akkor a fő osztály kiíró függvénye hívódik meg, amely elmenti a generált kódot egy VHDL fájlba.





```
MINGW64:/g/Projects/BigUniversityProjects/PicoBlazeCompillator
$ JAVA -jar SQ.jar
File name:
hiba.psm
PBC: Maps initialized OK
ERROR: Not defined at line: 2    CONSTANTa Ax 41
ERROR: Not enough parameters in row:6    LOAD S1
ERROR: Number not formatted correct at line: 7    LOAD S1 222
ERROR: To many parameters in row:8    LOAD S1 05 05
ERROR: Number not formatted correct at line: 9    LOAD 05 05
ERROR: Not defined at line: 10    LOADD S1 S2
ERROR: Number not formatted correct at line: 11    STORE S1 111
ERROR: Not defined at line: 12    STO S1 S2
ERROR: To many parameters in row:15    SR1 SR1 SR alma
ERROR: Not defined at line: 16    SR1 SR1
ERROR: Number not formatted correct at line: 17    OUTPUT 55 S1
ERROR: Number not formatted correct at line: 18    OUTPUT 22 22
ERROR: Not defined at line: 20    JUMP alma cimke
ERROR: Number not formatted correct at line: 22    LOAD SS SA
ERROR: Number not formatted correct at line: 30    LOAD na a
PBC: Errors:
Constants: {A=41}
Nameregs: {na=S8, nb=S9}
Calls: {cimke=9}
PBC: Completed
```

5.13. ábra. Fordítóprogram futása

Az 5.13 Ábrán látható az általam fejlesztett fordító program futtatása. A program működése rendkívül egyszerű, az indítás után csak a fordítani kívánt állomány nevét kell beírni a megjelenő konzol ablakba. Ha hibát talált az állományba akkor kiírja a hibákat, ha az állomány helyes akkor a fordítás eredményét kimentí egy VHDL típusú fájlba.

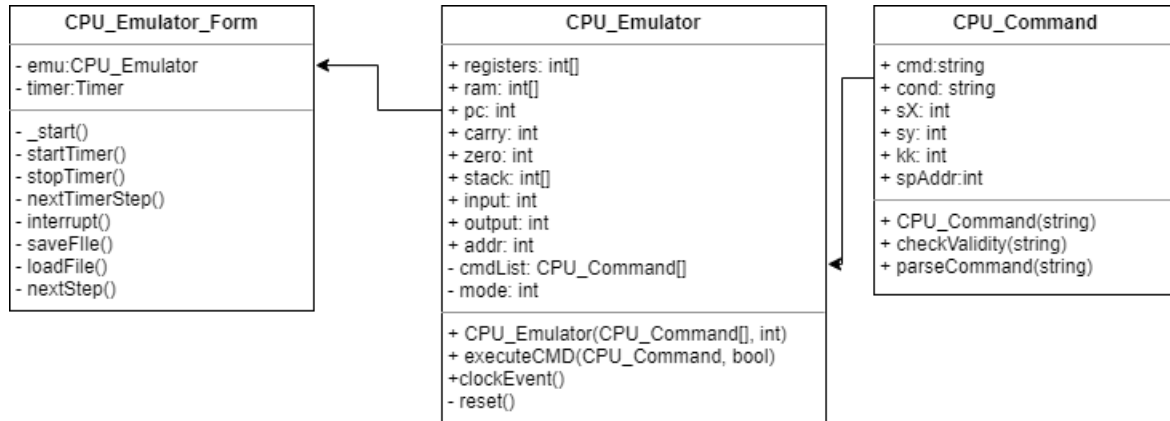
## 5.4. Az emulátor tervezése

A projekt utolsó része egy emulátor fejlesztése. Ennek a programnak a feladata egy processzor működésének a szimulálása egy számítógép használatával. Így könnyedén megfigyelhetők a processzor belső regisztereinek a tartalma két utasítás között. Illetve lehetőség van az általunk fejlesztett programok gyors hiba kezelésére. Hiszen egy egyszerűbb teszt elvégzéséhez nincs szükség egy FPGA lapra illetve a lassú kompilálásra sem. Csupán az assembly kódot kell megírni a többi a program intézi.

Az emulátort C# nyelven írtam, mivel a .Net keretrendszer segítségével rendkívül könnyedén lehet

felhasználói felületet készíteni.

Az emulátort a pBlaze IDE utasítás szimulátorról mintáztam. Viszont ez a program már nem elérhető letöltésre az interneten ezért csak a róla található fotókat használtam útmutatónak.



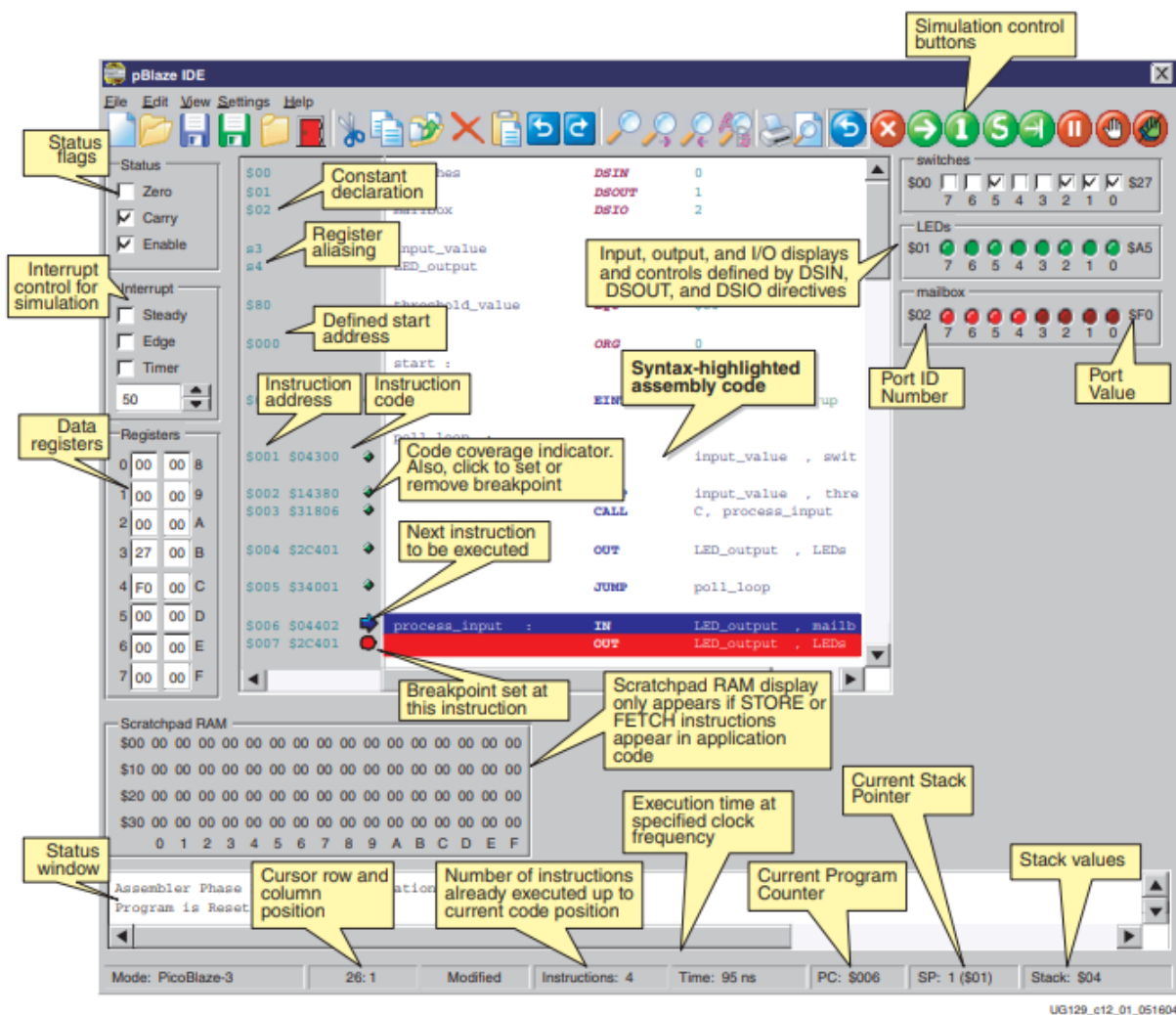
5.14. ábra. Az emulátor osztálydiagrammja

Az 5.14 ábrán megfigyelhető az emulátor osztálydiagrammja. Az emulátor három részből áll.

Egy felület amely kezeli a felhasználó inputjait és kapcsolatot teremt a processzor emulátor maggal.

Egy **CPU\_Emulator** modul amely, szimulálja a processzor belső regisztereinek az értékeit. Ez történhet automatikus előre meghatározott időintervallumonként, vagy a felhasználó által lépésenként haladva.

Illetve egy **CPU\_Command** osztály amely egy egysoros karakterláncból állít elő az emulátor számára futtatható kódot.



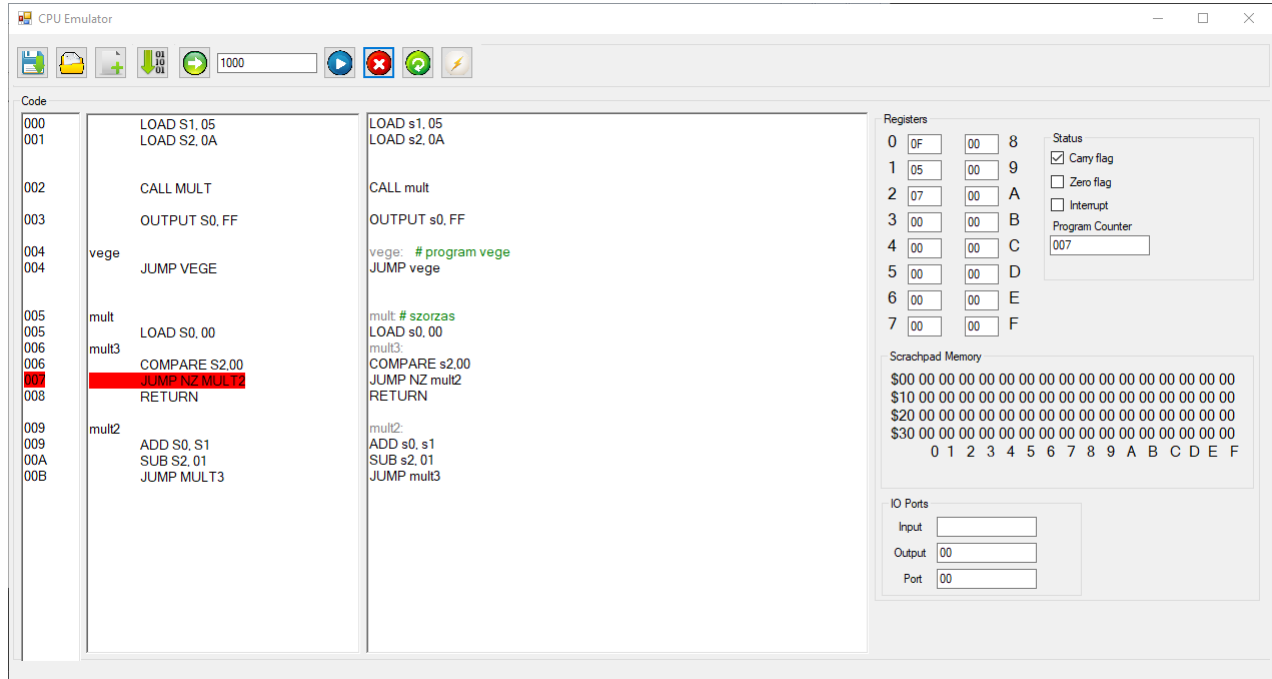
5.15. ábra. pBlaze IDE utasítás szimulátor

Az 5.15 Ábrán látható egy képe a pBlaze emulátor kezelőfelületéről. Megfigyelhető, hogy a felső eszközsáv kissé zsúfolt, a legtöbb gombd nem teljesen egyértelmű. Ugyanakkor megfigyelhetőek rajta, az emulált processzor státusz bitjei, mint például a zero értéket jelző bit illetve a túlcsordulást jelző bitt is. Egy másik fontos rész amely szintén látható a képen az a megszakítások vezérlése. Az egyik legfontosabb a regiszterek belső tartalmának a kijelzése, itt látható a képen a 16 darab 16 bites regiszter jobb oldalt, illetve alatta a 64 16 bites értéket tartalmazó memória.

Látható a képen, hogy a felhasználó által írt kód szét van választva a fordító által generált kódtól,

így könnyebb csupán a kép által generált kódot vizsgálni.

Az általam írt emulátor is rendelkezik a fent említett tulajdonságokkal. Ugyanakkor fontosnak tartottam az egyszerűséget és a felület letisztultságát.



5.16. ábra. Saját processzor emulátor

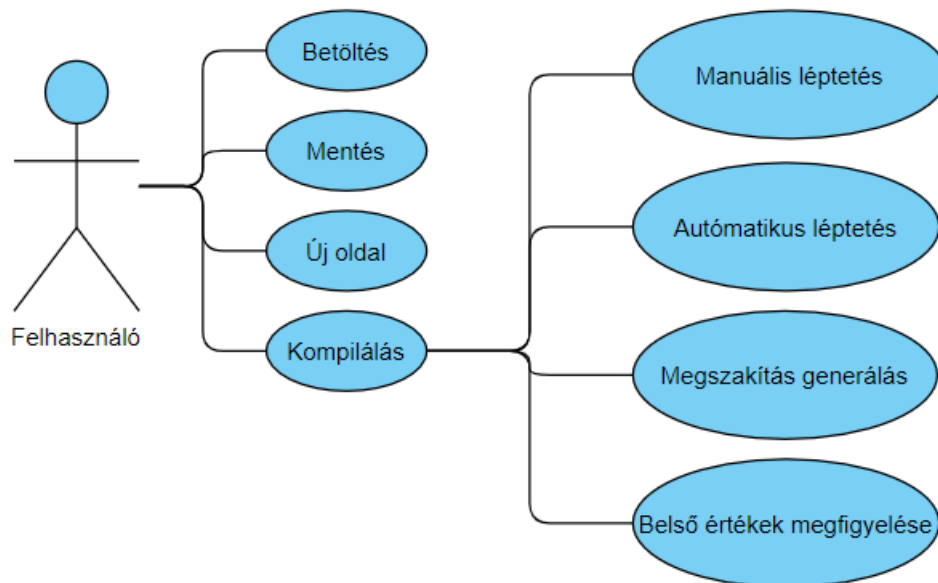
Az 5.16 Ábrán látható az általam tervezett és fejlesztett processzor emulátor. A képen látható, hogy az eszközsáv letisztult csupán pár elemet tartalmaz. Az első a mentés, erre kattintva el tudjuk menteni az assembly kódunkat egy szöveges fájlba. A következő a fájl megnyitása, ennek a használatával vagyunk képesek a már mentett kódot újra letölteni és futtatni. A harmadik egy új üres fájlt hoz létre. A negyedik gomb használatával kompiláljuk a jelenlegi kódot. Ha a kompilálás sikerül megjelenik a menü további része amely segítségével lehet futni a kódot soronként vagy egy megadott időegység alatt. Az utolsó gomb a megszakítás küldésért felelős. Ezt a gombot megnyomva az emulált processzor a következő órajel során elkezd a megszakítás rutin végrehajtását. Az emulátor esetén ekkor az INT címkéhez ugrik a programban a programszámláló.

Jobb oldalon megfigyelhetők a processzor belső regiszterei mint, a zero értéket és túlcscordulást jelző bitek, illetve ha megszakítás érkezett a processzor számára akkor az Interrupt előtt lévő dobozban

szintén megjelenik egy pipa. Ugyanakkor megfigyelhető a programszámláló és a regiszterek értéke is.

A kód a pBlaze emulátorhoz hasonlóan szintén két részre van osztva. Az elsőben megjelenik az utasítás címe és maga az utasítás. A másodikban pedig a felhasználó által írt és formázott kód, amely tartalmazza a kommenteket is.

Az emulátor működése során a programszámlálóban található érték szerint a neki megfelelő utasítás sora vörösre változik ezzel jelezve a következő utasítást.



5.17. ábra. Emulátor használati eset diagramja

Az 5.17 Ábrán bemutatja az emulátor használatát. Az indítást követően a felhasználó, négy különböző dolgot csinálhat. Betölt egy mentett állomány, elment egy állományt, új állomány hoz létre, vagy kompilálja a jelenlegi állományt.

A kompilálást követően manuálisan léptetheti az utasításokat. Illetve egy időzített léptetést is beállíthat. Az felhasználó bármikor küldhet egy megszakítást a processzornak. A szimulálás bármely szakaszában megfigyelheti az emulált processzor belső regisztereiben található értékeket.

## 6. fejezet

### Hardveres tesztek

A kutatás céljának megfelelően, a Xilinx PicoBlaze processzor teljesítményét hasonlítottam össze az általam fejlesztett és implementált processzorral. A cél elérésének érdekében, különféle teszteket végeztem mindkét processzoron. A tesztek közé tartoznak: aritmetikai művelet-igényes programok futtatása (ilyenek a faktORIZÁCIÓ és a Fibonacci számok generálása), analóg jel mintavételezése, analóg-digitál és digitál-analóg interfészek felhasználásával.

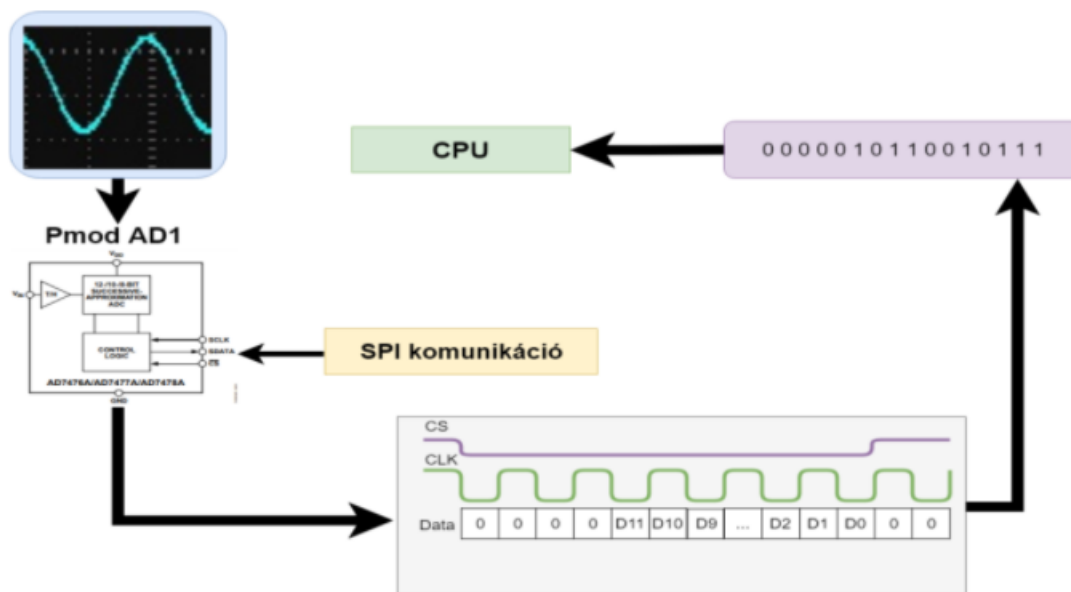
Az nagy aritmetikai erőforrást igénylő programokat folyamatosan futtattam újra és újra. Minden egyes futtatás előtt az, FPGA egyik kimenetét aktiváltam egy kis időre (magas impulzus), ezzel jelezve a program futásának kezdetét, majd minden iteráció után hasonló jelet generáltunk. Így egy oszcilloszkóp segítségével ki lehet számolni, hogy mennyi időbe telik a program végrehajtása, mindkét processzor esetén.

Az analóg jel mintavételezés során, jelgenerátort használva, különböző frekvenciájú és alakú (szinusz, négyszög) jelet vezettem az analóg-digitál átalakító bemenetére, amely egy hardveresen implementált analóg-digitál vezérlő egységbe küldte tovább a jelet. Azt beolvasta a processzor majd hasonló elven kiküldte a digitál-analóg interfésznek, amely a digitális jelet visszaalakítja egy analóg jellé, melyet oszcilloszkóp segítségével mértem és elemeztem.

## 6.1. Kommunikáció az analóg-digitál, digitál-analóg interfészekkel

Az adatok továbbítása a processzor és a konverterek között SPI (Serial Peripheral Interface) kommunikáció segítségével valósult meg. Az általam használt FPGA lap 50 MHz-es órajelen működik. Viszont az analóg-digitál, digitál-analóg konverterek maximális órajele csupán 20MHz, amelyen még képesek a helyes működésre. Ezért az eredeti 50 MHz-es órajelet a negyedére osztottam, így a modulok, 13,5 MHz-en működik.

A moduloknak egy 16 bites adat soros továbbításához 20 órajelre van szükségük.



6.1. ábra. AD interfész működése

A 6.1 ábrán látható az analóg-digitál konverter működése. Első lépésben a jelgenerátor által előállított szinusz jelet rávezettem a PMOD AD1 interfész bemenetére. Majd SPI kommunikáció során 20 órajel alatt a konverter továbbküldi a 16 bites értéket a processzor fele. Digitál-analóg konverzió esetén pont a fordítottja történik.

Mindkét processzor tesztelése során hasonló elvet alkalmaztam, különbség volt, hogy az FPGA-n hardveresen implementált vezérlő egység a Xilinx PicoBlaze processzor esetén nem egy 16 bit-es

értéket ad tovább, hanem 2 darab 8 bites értéket, mivel a processzor csak 8 bitet képes kezelni. Ennek következtében a Xilinx PicoBlaze processzornak kétszer több írási és olvasási műveletre volt szükség a jelek olvasására és megjelenítésére.

A tesztelés során az SPI kommunikáció hardveresen volt megvalósítva az FPGA lapon, így a processzorok csak a ki és bemeneti portműveleteket végezték.

## **6.2. Ki/Bemeneti műveletek tesztelése, analóg jel mintavételezésével**

Ebben a tesztben a Digilent PmodAD1 és PmodDA2 interfészek felhasználásával, analóg jel mintavételezést végeztünk meg. Az interfészek analóg jelet digitalizálnak egy 16 bites értéké, illetve digitális jelből állítanak elő analóg jelet.

A Xilinx PicoBlaze modul nem képes 16 bites számok kezelésére ezért a hardveres megvalósítás során két írási és olvasási műveletre volt szükség egy mintavétel beolvasásához és kiírásához. Az általam fejlesztett processzor 16 bites regiszterekkel rendelkezik, így elég volt csupán egy írási és olvasási művelet, a feladat elvégzéséhez.

Az analóg jel mintavételezést két csoportba osztottam. Szinuszos jel és négyszögjel mintavételezése. Mindkét jel esetén a mintavételezést elvégeztem alacsony és magasabb frekvenciákon is.

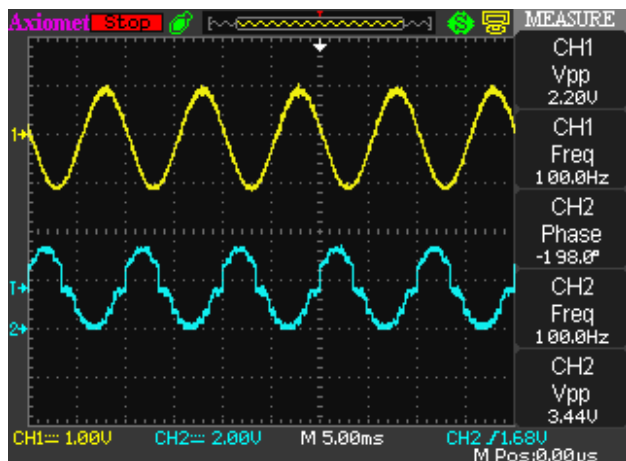
### **6.2.1. Szinuszos jel mintavételezése**

A szinuszos jel mintavételezését több különböző frekvencián is elvégeztem, 1 KHz-től egészen 100 KHz-ig. A következőkben ezekről szeretnék beszélni.

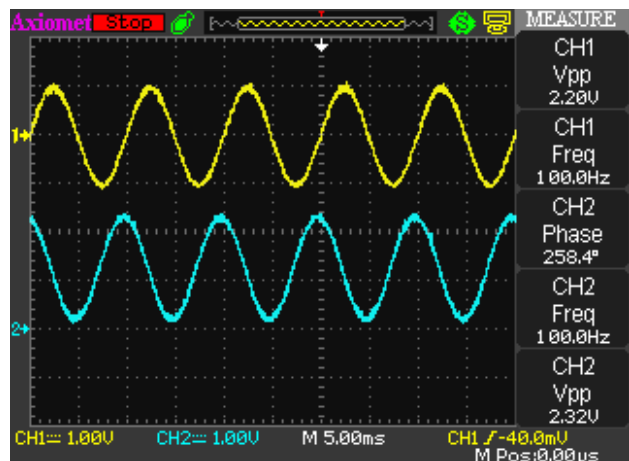
#### **Szinuszos jel mintavételezése 100Hz esetén**

A 6.2 és 6.3 ábrákon megfigyelhető a szinuszos jel mintavételezése mindkét processzor esetén 100Hz-es frekvencián. A sárga jel a jelgenerátor által szolgáltatott jel, amelyet mintavételezett mindkét processzor, a kék, pedig a processzorok által szolgáltatott jel.





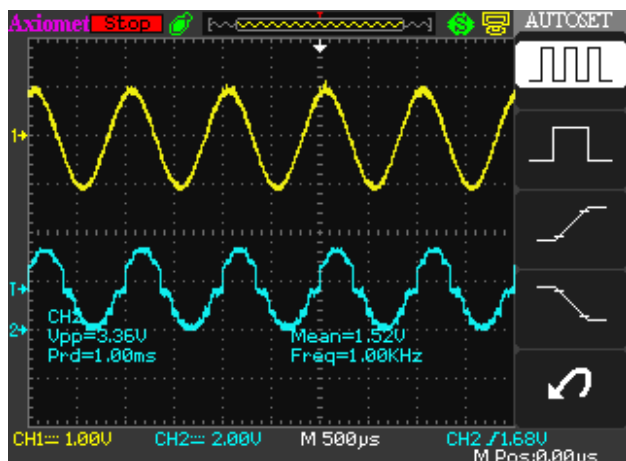
6.2. ábra. PicoBlaze, 100hz szinusz jel



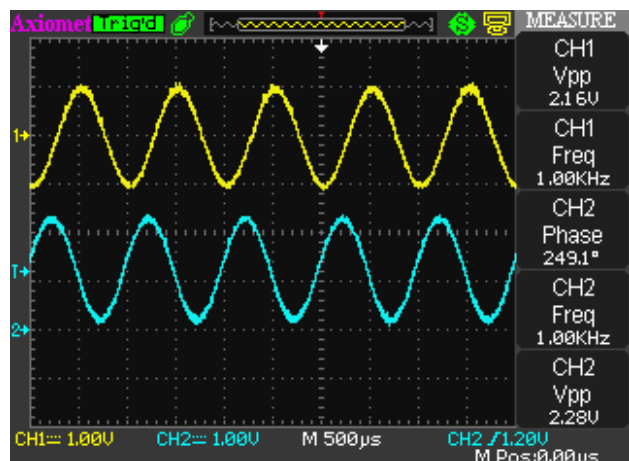
6.3. ábra. Saját processzor, 100hz szinusz jel

A 6.4 ábrán a Xilinx PicoBlaze processzor által visszaadott jel látható. Megfigyelhető, hogy enyhén töredezett. Ez annak a következménye, hogy processzor nem képes 16 bites értékeket kezelni ezért kétszer több írási és olvasási művelet szükséges a feladat végrehajtásához.

### Szinusz jel mintavételezése 1Khz esetén



6.4. ábra. PicoBlaze, 1Khz szinusz jel



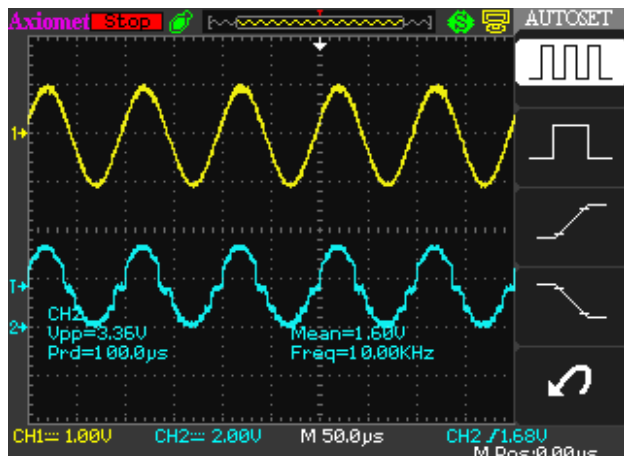
6.5. ábra. Saját processzor, 1Khz szinusz jel

A 6.4 és 6.5 ábrákon megfigyelhető a szinusz jel mintavételezése mindkét processzor esetén 1Khz-es frekvencián.

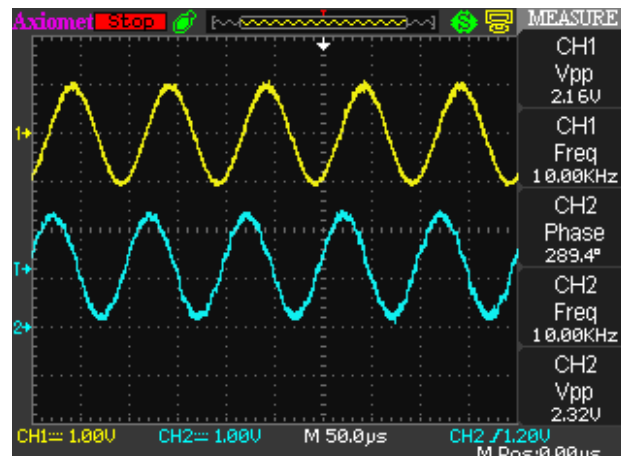
Az általam fejlesztett processzor láthatóan pontosan képes reprodukálni a mintavételezett szinusz

jelet.

### Színusz jel mintavételezése 10Khz esetén



6.6. ábra. PicoBlaze, 10Khz színusz jel



6.7. ábra. Saját processzor, 10Khz színusz jel

A 6.6 és 6.7 ábrákon megfigyelhető 10Khz-es frekvenciájú színusz jel mintavételezése mindkét processzor esetén.

Mint ahogyan 1Khz esetén itt is megfigyelhető a 6.6 ábrán, hogy a PicoBlaze processzor jele enyhén töredezett, a kétszer több írási és olvasási művelet következtében.

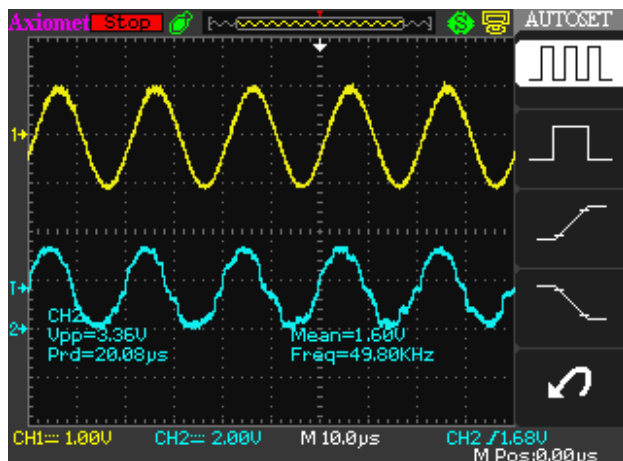
Az általam fejlesztett processzor 10Khz esetén is képes elfogadható színusz jelet reprodukálni, ahogyan ez a 6.7 ábrán is látható.

### Színusz jel mintavételezése 50Khz esetén

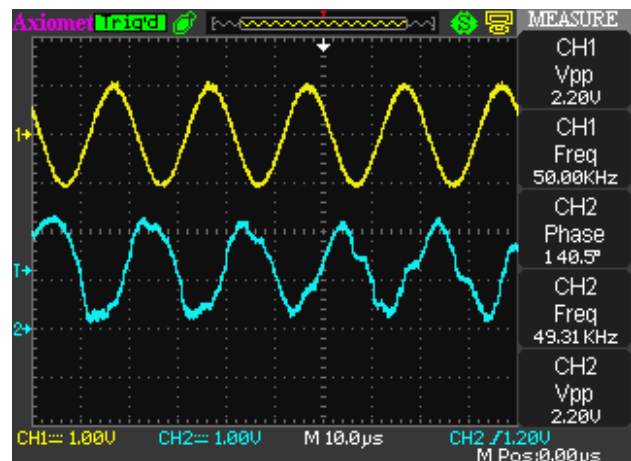
Színusz jel mintavételezését elvégeztem 50Khz esetén is ahogyan az a 6.8 és 6.9 ábrákon is látható. 50Khz esetén már mindkét processzor esetében a jelek töredezték, de még megfigyelhető egy enyhén eltorzult színusz minta.

### Színusz jel mintavételezése 100Khz esetén

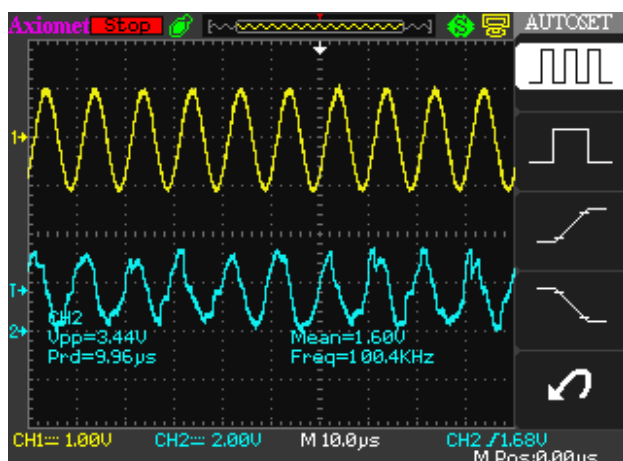
Az általam tesztelt legmagasabb frekvencia a 100Khz. A teszt eredményei megfigyelhetőek a 6.10 és 6.11 ábrákon. Látható, hogy ilyen magas frekvenciákon, sem a Xilinx PicoBlaze processzora, sem az



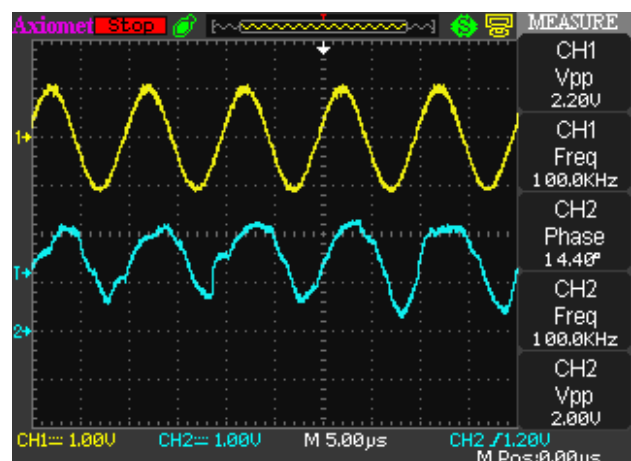
6.8. ábra. PicoBlaze, 50Khz szinusz jel



6.9. ábra. Saját processzor, 50Khz szinusz jel



6.10. ábra. PicoBlaze, 100Khz szinusz jel

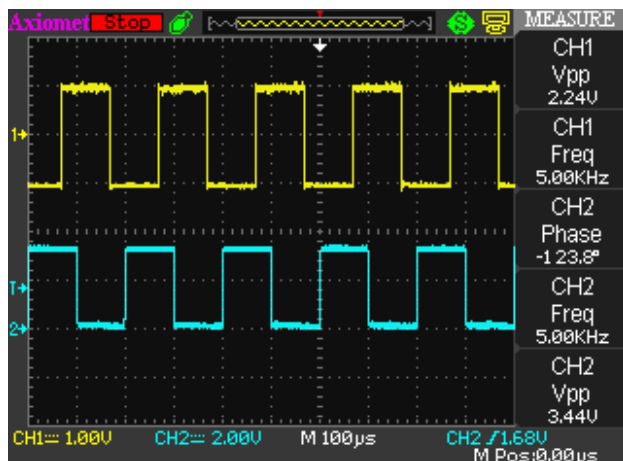


6.11. ábra. Saját processzor, 100Khz szinusz jel

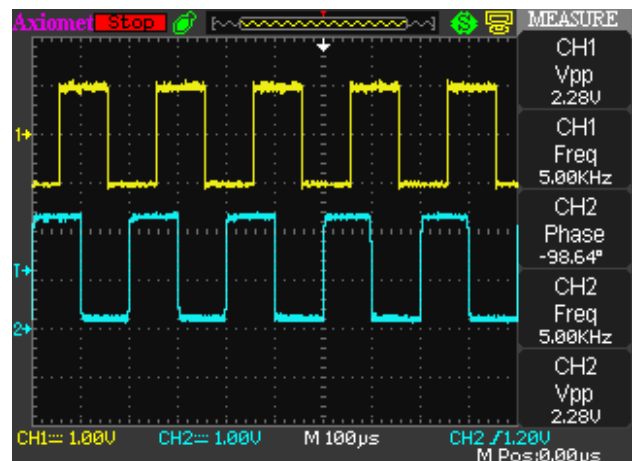
általam fejlesztett processzor nem képes elfogadható szinusz jel reprodukálására.

### 6.2.2. Négyzög jel mintavételezése

A négyzög jel mintavételezését több különböző frekvencián is elvégeztem, 5Khz-től egészen 100Khz-ig. A következőekben ezekről szeretnék beszélni.



6.12. ábra. PicoBlaze, 5khz négyzög jel



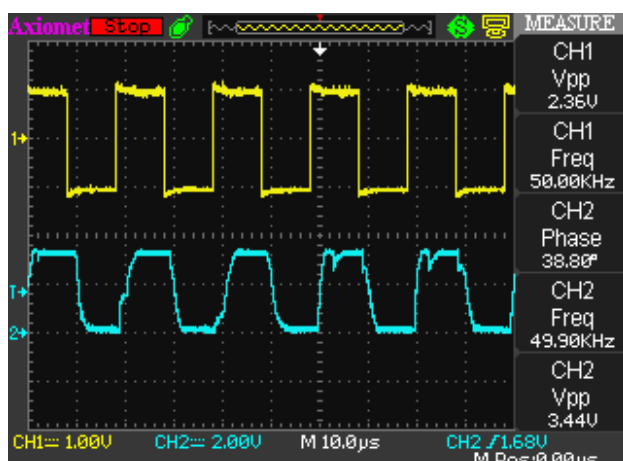
6.13. ábra. Saját processzor, 5khz négyzög jel

### Négyzög jel mintavételezése 5khz esetén

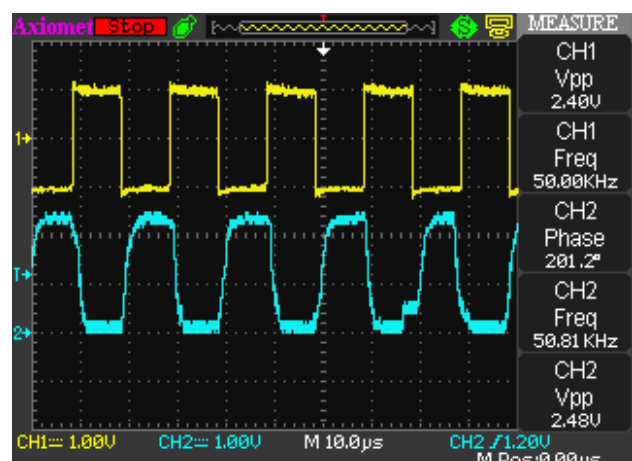
A 6.12 és 6.13 ábrákon megfigyelhető a négyzög jel mintavételezése mindkét processzor esetén 5Khz-es frekvencián. A sárga jel a jelgenerátor által szolgáltatott jel, amelyet mintavételezett mindkét processzor, a kék, pedig a processzorok által szolgáltatott jel.

Megfigyelhető, hogy alacsony frekvencián, 5Khz, mindkét processzor képes a négyzög jel helyes reprodukálására.

### Négyzög jel mintavételezése 50khz esetén



6.14. ábra. PicoBlaze, 50khz négyzög jel

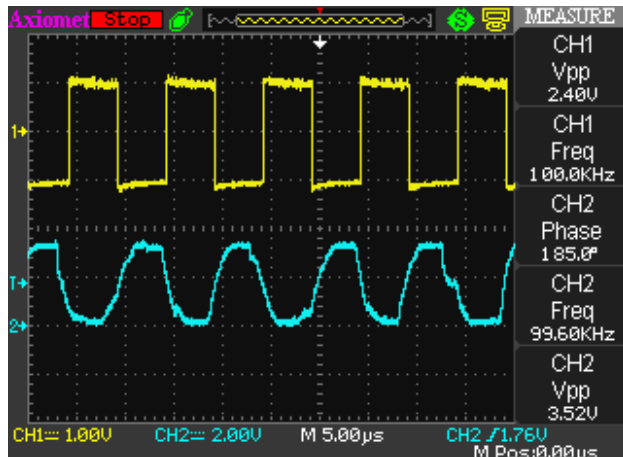


6.15. ábra. Saját processzor, 50khz négyzög jel

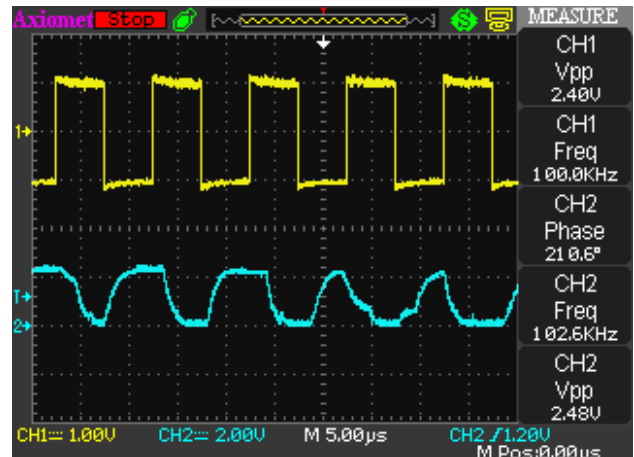
A 6.14 és 6.15 ábrákon megfigyelhető a négyzög jel mintavételezése mindkét processzor esetén 50Khz-es frekvencián.

Az ábrákon megfigyelhető, hogy 50Khz környékén mindkét processzor által szolgáltatott jel enyhén torz, de még egy elfogadható négyzög jel a mintavételezett és visszaadott jel.

### Négyzög jel mintavételezése 100khz esetén



6.16. ábra. PicoBlaze, 100khz négyzög jel



6.17. ábra. Saját processzor, 100khz négyzög jel

A 6.16 és 6.17 ábrákon megfigyelhető a négyzög jel mintavételezése mindkét processzor esetén 100Khz-es frekvencián.

Az ábrákon megfigyelhető, hogy 100Khz környékén mindkét processzor által szolgáltatott jel nagyon torz. Ilyen magas frekvenciákon már egyik processzor sem képes egy elfogadható négyzög jel reprodukálására.

## 6.3. Aritmetikai műveletvégzési teljesítményét felmérő tesztek

A processzorokat nem csupán jel mintavételezési teszteknek vettem alá. Kíváncsi voltam az aritmetikai számítási kapacitásra is, ezért a következő tesztek is elvégeztem a Xilinx PicoBlaze és az általam fejlesztett processzorral.

Elvégzett aritmetikai tesztek:

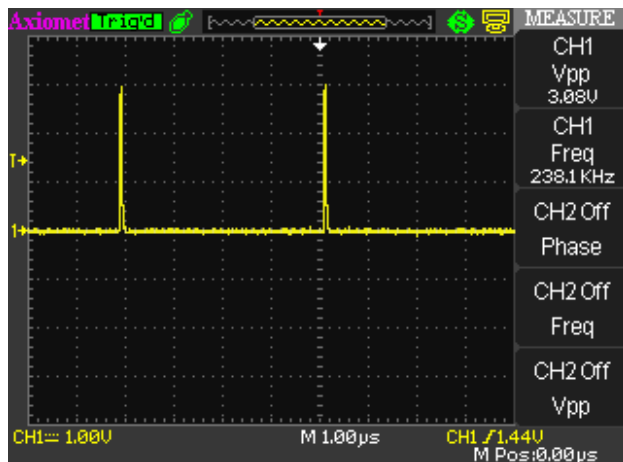
- FaktORIZÁCIÓ, sorozatos összeadással
- FaktORIZÁCIÓ, beépített szorzó áramkörrel
- Fibonacci számsorozat generálása 8 biten
- Fibonacci számsorozat generálása 16 biten

### 6.3.1. FaktORIZÁCIÓ, sorozatos összeadással

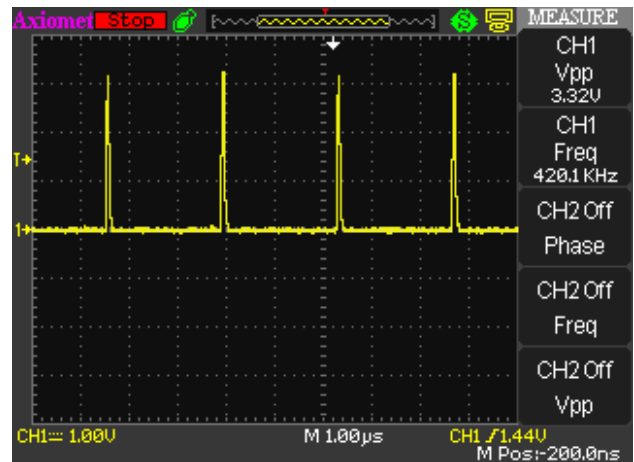
Az első aritmetikai teszt a FaktORIZÁCIÓ volt. Mivel a Xilinx PicoBlaze nem tartalmaz szorzó áramkört ezért a szorzást szoftveresen valósítottam meg folyamatos összeadással. Ez nem a leghatékonyabb, de ez nem is volt fontos, hiszen mindkét processzor ugyanazt a kódot fogja futtatni, így nem az algoritmusokat fogom versenyeztetni hanem a két processzort összemérni aritmetikai műveletvégzés szempontjából.

A Xilinx PicoBlaze processzor 8 bites számok kezelésre képes ezért a legnagyobb érték kiszámolt érték 5 faktoriális volt.

A tesztek kivitelezése során mindkét processzorra ugyanazt a kódot töltöttem fel. A kód folyamatosan elvégezte újra és újra a faktORIZÁCIÓT, minden fután elején egy jelet küldtem az oszcilloszkópra. Így megfigyelhető a program futása két impulzus jel között.



6.18. ábra. PicoBlaze, faktORIZÁCIÓ összeadással



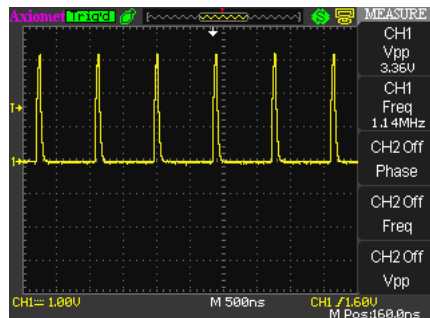
6.19. ábra. Saját, faktORIZÁCIÓ összeadással

A 6.18 és 6.19 ábrákon megfigyelhető a Xilinx PicoBlaze és az általam fejlesztett processzor eredménye a faktorizáció során.

Az 6.18 és 6.19 ábrákról leolvasható, hogy a PicoBlaze processzor 4 mikroszekundum alatt volt képes kiszámolni az 5 faktoriálisan, míg az általam fejlesztett processzornak elég volt csupán 2.2 mikroszekundum.

### 6.3.2. Faktorizáció, beépített szorzó áramkörrel

A faktorizációt elvégeztem a beépített szorzó áramkör segítségével is, az eredmény a 6.20 ábrán látható. Az ábráról kiderül, hogy az 5 faktoriális kiszámításához a processzornak csupán 900 nanoszekundumra volt szüksége.

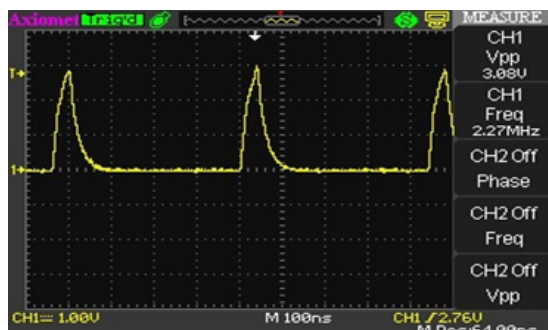


6.20. ábra. Saját processzor, faktorizáció beépített szorzó áramkörrel

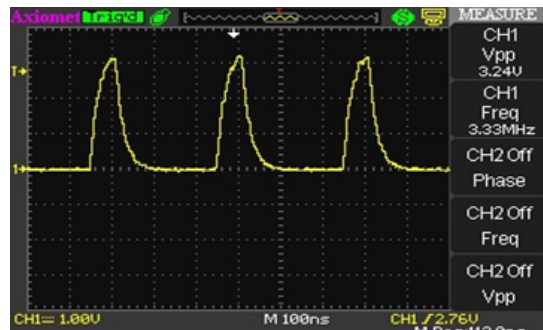
### 6.3.3. Fibonacci számsorozat generálása 8 biten

A Fibonacci számok generálása során, mindkét processzorom ugyanazt az programkódot futtattuk újra és újra, megfigyelve az elvégzéshez szükséges időt. A PicoBlaze 8 bites jellege miatt, csupán a legnagyobb 8 biten ábrázolható Fibonacci számig (233) generáltam a számokat, mindkét processzor esetében.

A 6.21 és 6.22 ábrákon megfigyelhető a Xilinx PicoBlaze és az általam fejlesztett processzor eredménye a Fibonacci számsorozat generálás során. Az ábrákról leolvasható, hogy a Xilinx PicoBlaze processzor 420 nanoszekundum alatt végezte el a műveletet, míg az általam fejlesztett processzornak elég volt csupán 300 nanoszekundum is.



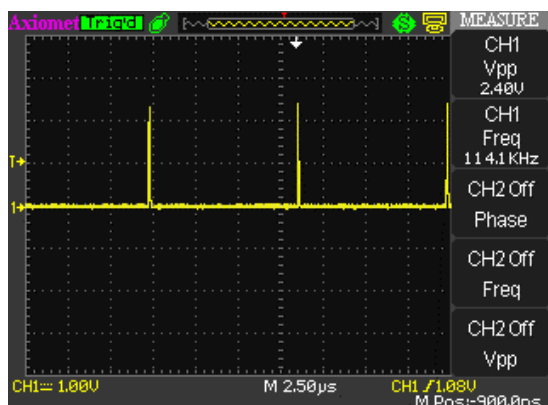
6.21. ábra. PicoBlaze, fibonacci számsor generálás 8 biten



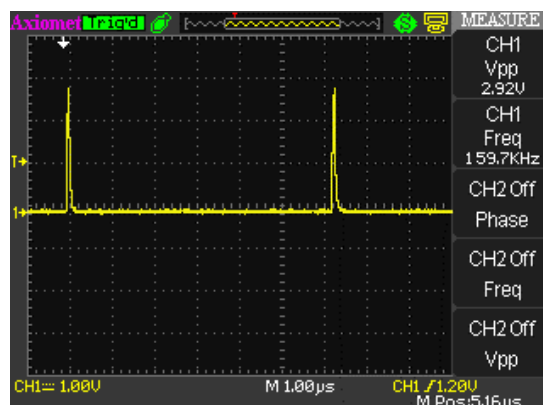
6.22. ábra. Saját processzor, fibonacci számsor generálás 8 biten

### 6.3.4. Fibonacci számsorozat generálása 16 biten

Fibonacci számok generálást 16 biten is elvégeztem mindkét processzor esetén. Fontosnak tartottam, hogy ne az algoritmusokat versenyeztessük, hanem a két processzort mérjem össze ezért mindkét processzoron ugyanaz a kód futott. Így a PicoBlaze esetén két regisztert használtam egy 16 bites szám tárolására. Az általam fejlesztett processzor esetén szintén két regiszterben tároltam egy számot, így a processzor 32 biten végezte a műveleteket.



6.23. ábra. PicoBlaze, fibonacci számsor generálás 16 biten



6.24. ábra. Saját processzor, fibonacci számsor generálás 16 biten

A 6.21 és 6.22 ábrákról leolvasható, hogy a Xilinx PicoBlaze processzor 9 mikroszekundum alatt végezte el a feladatot, míg az általam fejlesztett processzor csupán 6.1 mikroszekundum alatt végezte el ugyanazt a feladatot.



## 7. fejezet

# Eredmények

A következő részben a kutatásom során elért eredményeket szeretném bemutatni az elvégzett tesztek kiértékelésével. A tesztek során összehasonlításra került a Xilinx PicoBlaze processzora és az általam fejlesztett Harvard architektúrájú processzor.

A tesztek két csoportba osztoztam, aritmetikai művelet végzési tesztek és analóg jel mintavételezési tesztek.

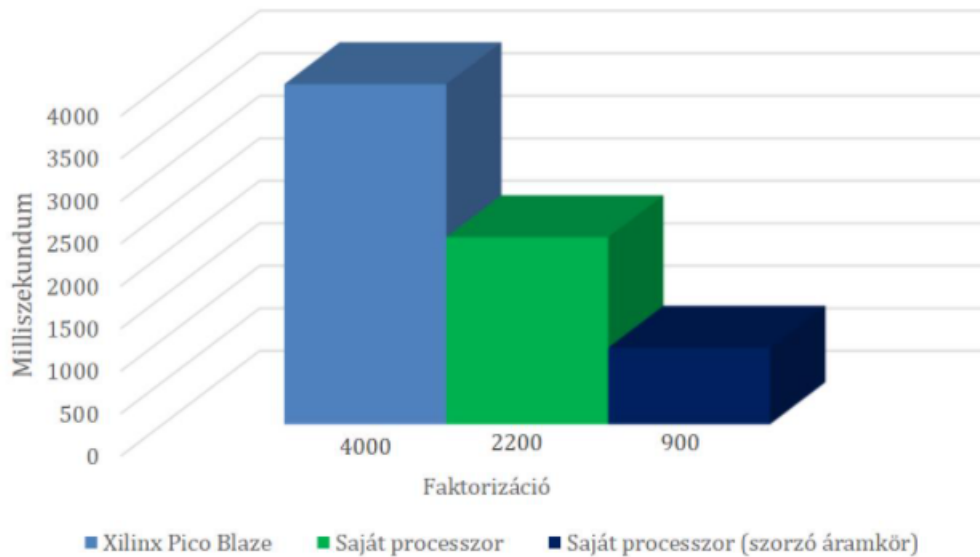
### 7.1. Aritmetika műveletvégzési teljesítmény felmérő tesztek kiértékelése

Ebben a részben az általam elvégzett aritmetikai tesztek kiértékeléséről fogok beszélni.

A tesztek során mindkét processzoron ugyanaz a kód futott annak érdekében, hogy a processzorok teljesítményét mérjük össze.

Hiszen például a szorzó áramkör használatával az általam fejlesztett processzor bármilyen szorzási műveletet sokkal gyorsabban képes elvégezni.

### 7.1.1. FaktORIZÁCIÓS TESZTEK KIÉRTÉKELÉSE



7.1. ábra. FaktORIZÁCIÓ KIÉRTÉKELÉSE

A FaktORIZÁCIÓS tesztek eredményét összesítése a 7.1 ábrán látható. Megfigyelhető, hogy az általam fejlesztett processzor lényegesen gyorsabban tudta elvégezni a rábízott feladatot. A beépített szorzó áramkör segítségével 4-szer gyorsabban volt képes végrehajtania a faktORIZÁCIÓT mint a Xilinx PicoBlaze processzor.

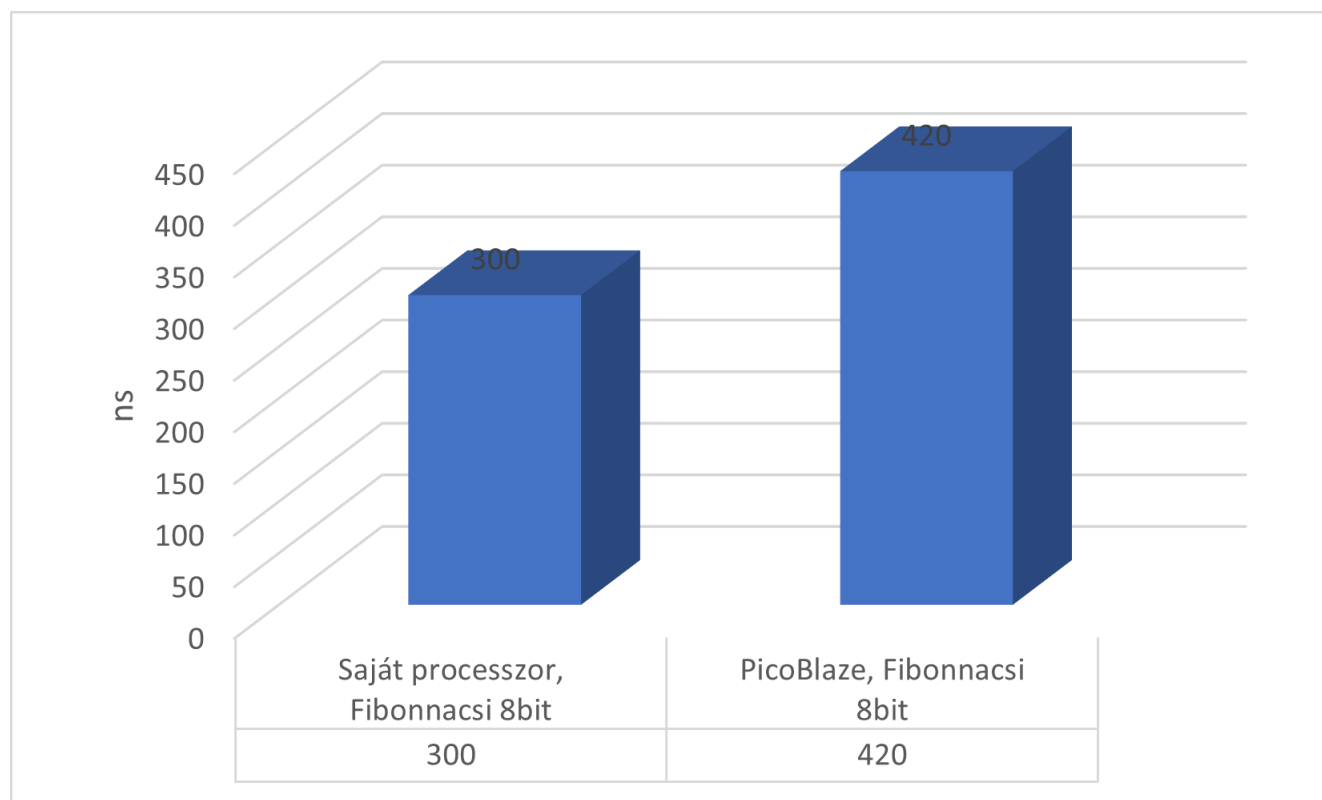
Az aritmetikai műveletek elvégzésében az általam fejlesztett processzor átlagosan fele annyi idő alatt képes volt elvégezni ugyanazt a művelet és nagyobb számokkal volt képes dolgozni. Jelentős sebesség növekedést eredményez a beépített szorzó áramkör, a sorozatos összeadással szemben.

A programkódból és a végrehajtási időből kiszámoltam a processzorok MIPS (Million instruction per second) értékét is amely a 7.1 táblázaton látható. A táblázatból kiderül, hogy az általam fejlesztett processzor 56%-al volt gyorsabb mint a PicoBlaze processzor.

PicoBlaze	Saját processzor
19,5 MIPS	34,4 MIPS

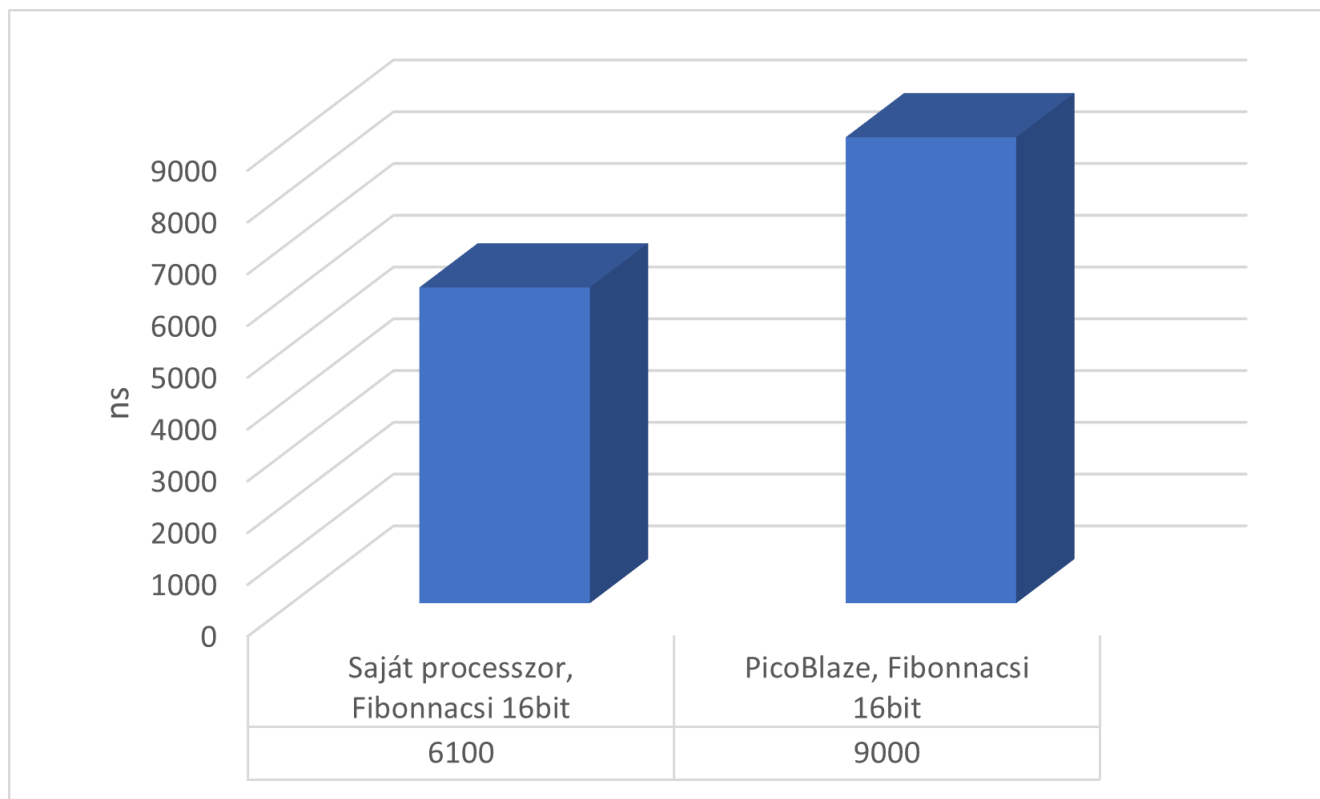
7.1. táblázat. FaktORIZÁCIÓ MIPS ÉRTÉKE

### 7.1.2. Fibonnacsi tesztek kiértékelése



7.2. ábra. Fibonacci 8bit, eredmények

A 8 bites Fibonacci számsorozat generálás eredményei a 7.2 ábrán láthatóak. Megfigyelhető, hogy az általam tervezett processzor kevesebb idő alatt volt képes elvégezni a faktORIZÁCIÓS feladatot. Ez a nagy sebességnövekedés a csővezeték elv segítségével volt lehetséges.



7.3. ábra. Fibonacci 16bit, eredmények

A 16 bites Fibonacci számsorozat generálás esetén hasonló eredmény figyelhető meg mint 8 bit esetén, ahogyan ez a 7.3 ábrán is látható.

PicoBlaze	Saját processzor
23,81 MIPS	33,84 MIPS

7.2. táblázat. Fibonacci MIPS értéke

A programkódból és a végrehajtási időből kiszámoltam a processzorok MIPS (Million instruction per second) értékét is amely a 7.2 táblázaton látható. A táblázatból kiderül, hogy az általam fejlesztett processor 70%-al volt gyorsabb mint a PicoBlaze processzor.

## 7.2. A processzor erőforrás igényének kiértékelése

Az általam fejlesztett processzor képes volt a jel mintavételezésre. Illetve jobban teljesít az aritmetikai művelet végzése esetén. Viszont egy nagy hátránya, hogy több erőforrást használ mint a Xilinx PicoBlaze processzor.

Ez annak a következménye, hogy a regiszterek mérete nagyobb, szorzó áramkör is tartalmaz. Viszont a legjelentősebb hogy a teljes processzor viselkedése leírással volt tervezve. Míg a PicoBlaze esetén strukturális VHDL leírás volt használva.

Az alábbi ábrákon (7.4, 7.5) megfigyelhető a két processzor erőforrás szükséglete, megfigyelhető, hogy az általam fejlesztett processzor az FPGA 18%-át használta míg a PicoBlaze processzor csupán 2%-ot foglal.

Device Utilization Summary			
Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	184	17,344	1%
Number of 4 input LUTs	310	17,344	1%
Number of occupied Slices	205	8,672	2%
Number of Slices containing only related logic	205	205	100%
Number of Slices containing unrelated logic	0	205	0%
Total Number of 4 input LUTs	314	17,344	1%
Number used as logic	242		
Number used as a route-thru	4		
Number used for Dual Port RAMs	16		
Number used for 32x1 RAMs	52		
Number of bonded IOBs	37	250	14%
Number of RAMB16s	1	28	3%
Number of BUFMUXs	3	24	12%
Average Fanout of Non-Clock Nets	3.79		

7.4. ábra. Erőforrás használat, PicoBlaze

Device Utilization Summary			
Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	876	17,344	5%
Number of 4 input LUTs	2,887	17,344	16%
Number of occupied Slices	1,604	8,672	18%
Number of Slices containing only related logic	1,604	1,604	100%
Number of Slices containing unrelated logic	0	1,604	0%
Total Number of 4 input LUTs	2,966	17,344	17%
Number used as logic	2,887		
Number used as a route-thru	79		
Number of bonded IOBs	61	250	24%
Number of RAMB16s	2	28	7%
Number of BUFMUXs	3	24	12%
Number of MULT18X18SIOs	2	28	7%
Average Fanout of Non-Clock Nets	3.84		

7.5. ábra. Erőforrás használat, PicoBlaze

## 8. fejezet

# Összefoglalás

A projekt során sikerült, megtervezni és implementálni egy Harvard architektúrájú processzort. Megvalósult a két processzor összehasonlítása, az általam fejlesztett processzor és a Xilinx PicoBlaze processzor között. A testekből kiderül, hogy az általam fejlesztett processzor képes felvenni a versenyt a PicoBlaze processzorral. Illetve néhány kategóriában még jobban is teljesít, mint például az aritmetikai műveletvégzés.

Sikerült megvalósítani egy fordító programot is, amely mindkét processzor nyelvére képes egyaránt fordítani. Szinte bármilyen gépen gond nélkül futtatható, ugyanakkor mivel több hibát képes egyszerre kifejezni ezért felgyorsítja a programok fejlesztését és javítását.

Megvalósításra került a processzor emulátor is amelynek segítségével rendkívül könnyedén és gyorsan lehet az Assembly kódot tesztelni, egy FPGA nélkül.

### 8.1. Továbbfejlesztési lehetőségek

A dolgozat során a célok egy nagy részét sikerült megvalósítani, viszont még rengeteg lehetőséget rejt magában a projekt. Ilyen ötleteket szeretnék megemlíteni:

- Különböző tesztek FPGA segítségével, mint hang felvétel és visszajátszás.
- Összehasonlítás más processzorokkal.
- Emulátor tovább fejlesztése, grafikus felület, több processzor szimulálása

## 8.2. Köszönetnyilvánítás

Köszönetet szeretnék mondani tanáromnak dr. Bakó Lászlónak, a Sapientia EMTE egyetem adjunktusának, akinek segítségével nem valósulhatott volna meg ez a dolgozat. Illetve köszönetet szeretnék mondani a Sapientia EMTE Marosvásárhely-i karának aki biztosította az FPGA, lapot illetve az eszközöket amelyek segítségével történtek a tesztek elvégzése.

# Irodalomjegyzék

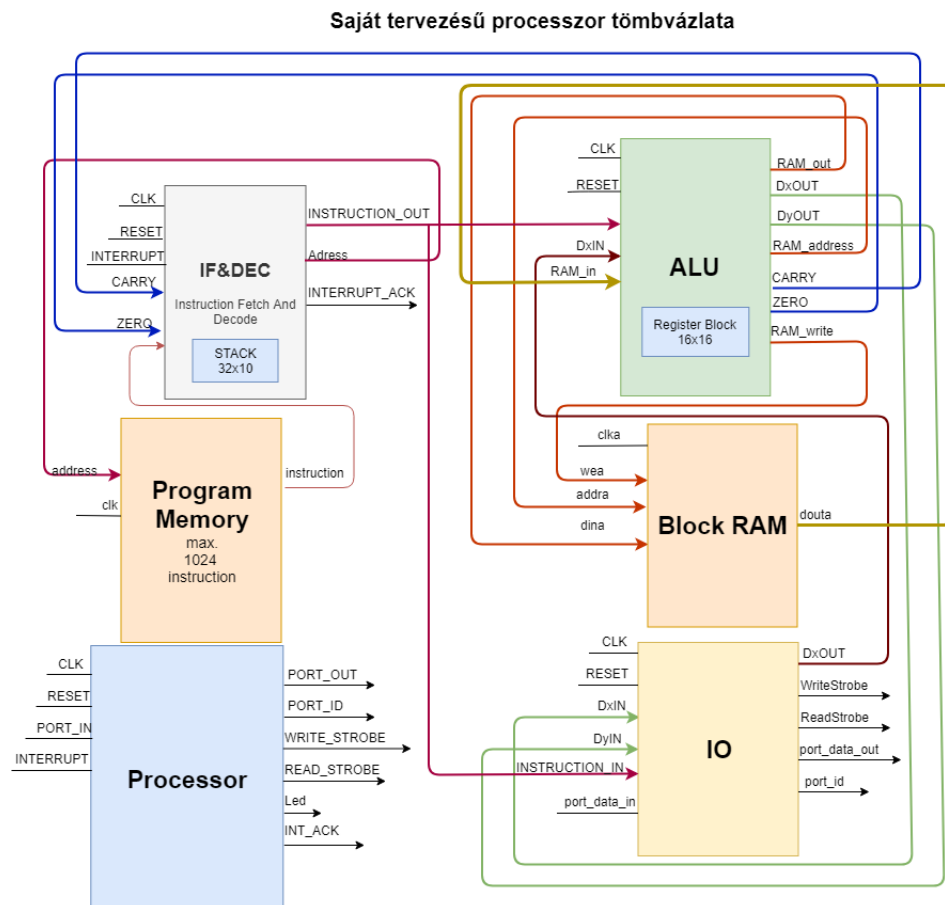
- [1] Frank O'Brien, „Defining computer "power",” in *The Apollo Guidance Computer: Architecture and Operation*, Springer Science Business Media, 2010.
- [2] Maurer, Ward Douglas, „The effect of the harvard architecture on the teaching of assembly language.” in *The effect of the Harvard architecture on the teaching of assembly language.*, pp. 79–90, Journal of Computing Sciences in Colleges 20.5, 2005.
- [3] Priyanka Trivedi, Rajan Prasad Tripathi, „Design analysis of 16 bit risc processor using low power pipelining,” in *Design analysis of 16 bit RISC processor using low power pipelining*, IEEE, 2015.
- [4] Samiappa Sakthikumaran, S. Salivahanan, V. S. Kanchana Bhaaskaran, „16-bit risc processor design for convolution application,” in *16-Bit RISC processor design for convolution application*, IEEE, 2011.
- [5] M. Gschwind, V. Salapura, D. Maurer, „Fpga prototyping of a risc processor core for embedded applications,” in *FPGA prototyping of a RISC processor core for embedded applications*, IEEE, 2001.
- [6] Pravin S. Mane, Indra Gupta, M. K. Vasantha, „Implementation of risc processor on fpga,” in *Implementation of RISC Processor on FPGA*, IEEE, 2006.
- [7] „Field-programmable gate array.” Available at "[https://hu.wikipedia.org/wiki/Field-programmable\\_gate\\_array](https://hu.wikipedia.org/wiki/Field-programmable_gate_array)", [Online; accessed 20.06.2021].



- [8] „Digilent nexys2 board reference manual,” in *Digilent Nexys2 Board Reference Manual*, Digilent, 2012.
- [9] „Vhdl.” Available at "<https://hu.wikipedia.org/wiki/VHDL>", [Online; accessed 20.06.2021].
- [10] „Java (programozási nyelv).” Available at "[https://hu.wikipedia.org/wiki/Java\\_\(programozási\\_nyelv\)](https://hu.wikipedia.org/wiki/Java_(programozási_nyelv))", [Online; accessed 20.06.2021].
- [11] „C sharp.” Available at "[https://hu.wikipedia.org/wiki/C\\_Sharp](https://hu.wikipedia.org/wiki/C_Sharp)", [Online; accessed 20.06.2021].
- [12] „A c programozási nyelv.” Available at "<http://nyelvek.inf.elte.hu/leirasok/Csharp/index.php?chapter=1>", [Online; accessed 20.06.2021].
- [13] „.net keretrendszer.” Available at "[https://hu.wikipedia.org/wiki/.NET\\_keretrendszer](https://hu.wikipedia.org/wiki/.NET_keretrendszer)", [Online; accessed 20.06.2021].
- [14] „Windows forms.” Available at "[https://en.wikipedia.org/wiki/Windows\\_Forms](https://en.wikipedia.org/wiki/Windows_Forms)", [Online; accessed 20.06.2021].
- [15] Ken Chapman, „Picoblaze 8-bit microcontroller for virtex-e and spartan-ii/iie devices.” Available at "[https://www.xilinx.com/support/documentation/application\\_notes/xapp213.pdf](https://www.xilinx.com/support/documentation/application_notes/xapp213.pdf)", [Online; accessed 20.06.2021].
- [16] Nagy Serbán Tünde, Gáll János, „Saját tervezésű, risc processzor megvalósítása és tesztelése fpga áramkörön, tudományos diákköri konferencia,” in *Saját tervezésű, RISC processzor megvalósítása és tesztelése FPGA áramkörö*, 2019.
- [17] „Reduced instruction set computing.” Available at "[https://hu.wikipedia.org/wiki/Reduced\\_Instruction\\_Set\\_Computing](https://hu.wikipedia.org/wiki/Reduced_Instruction_Set_Computing)", [Online; accessed 20.06.2021].

# A. függelék

## Függelék



UNIVERSITATEA SAPIENTIA DIN CLUJ-NAPOCA  
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE, TÎRGU-MUREȘ  
SPECIALIZAREA CALCULATOARE

Vizat decan  
Conf. dr. ing. Domokos József

Vizat director departament  
Ș.l. dr. ing Szabó László Zsolt