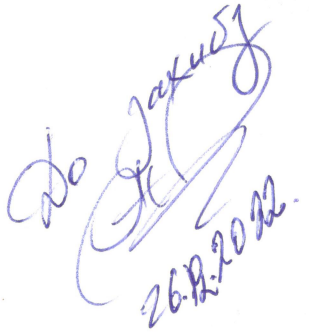
Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет «Комп'ютерні технології і системи»
(назва факультету)

Кафедра «Електронні обчислювальні машини»
(повна назва кафедри)



Пояснювальна записка

до кваліфікаційної роботи
магістра
(ступінь вищої освіти)

на тему: Розробка та дослідження процесора виконання математичних операцій над комплексними числами з використанням ПЛІС. Операції додавання та віднімання і дослідний стенд для їх тестування.
за освітньою програмою Комп'ютерна інженерія
зі спеціальності: 123 Комп'ютерна інженерія
(шифр і назва спеціальності)

Виконав: студент групи: КС2121

	(підпис студента)	Дмитро ЛАЗОРЕНКО (Ім'я ПРІЗВИЩЕ)
Керівник:	(підпис)	доцент Володимир ШАПОВАЛОВ (посада, Ім'я ПРІЗВИЩЕ)
Нормоконтролер:	(підпис)	доцент Володимир ШАПОВАЛОВ (посада, Ім'я ПРІЗВИЩЕ)

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Дніпро – 2022 рік

Ministry of Education and Science of Ukraine
Ukrainian State University of Science and Technologies

Faculty «Computer technologies and systems»
(faculty)

Department «Electronic computers»
(department)

Explanatory Note
to Qualification Work

Master's
(higher education degree)

on the topic: Development and research of a processor for performing mathematical operations on complex numbers using FPGA. Operations of addition and subtraction and a research stand for their testing.

according to educational curriculum Computer Engineering

in the Speciality: 123 Computer Engineering
(speciality and its code)

Done by the student of the group: KC2121 Dmitro Lazorenko
(name, surname)

Scientific Supervisor: docent Volodymyr Shapovalov
(position, name, surname)

Normative controller : docent Volodymyr Shapovalov
(position, name, surname)

Dnipro – 2022

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: Комп'ютерні технології і системи
Кафедра: ЕОМ
Рівень вищої освіти: Другий (магістерський)
Освітня програма: Комп'ютерна інженерія
Спеціальність: 123 Комп'ютерна інженерія
(шифр та назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри ЕОМ
_____ Ігор ЖУКОВИЦЬКИЙ
(підпис) (Ім'я ПРІЗВИЩЕ)
Дата 30.09.2022

З А В Д А Н Н Я

на кваліфікаційну роботу магістра
(ступінь вищої освіти)

студенту Лазоренку Дмитру Вікторовичу
(Прізвище, Ім'я По батькові)

1. Тема роботи: Розробка та дослідження процесора виконання математичних операцій над комплексними числами з використанням ПЛІС. Операції додавання та віднімання і дослідний стенд для їх тестування.

Керівник роботи: Шаповалов Володимир Олександрович, к. т. н., доцент
(Прізвище, Ім'я, По батькові, науковий ступінь, вчене звання)

затверджені наказом від " 14 " 09 2022 р. № 845ст

2. Строк подання студентом роботи: 19.12.2022 р.
3. Вихідні дані до роботи: Для розробки та дослідження процесора використати засоби високорівневого проектування пакету Matlab/Simulink, додаток HDL Coder і САПР цифрових пристроїв.
4. Зміст пояснювальної записки (перелік питань, які потрібно опрацювати):
4.1 Аналітична частина: Зробити огляд засобів високорівневого проектування цифрових систем, CISC та RISC процесорів та можливостей їх використання для обробки комплексних чисел.
4.2 Основна частина: Розробка структури процесора, пам'яті команд, оперативної пам'яті, пристрою управління за допомогою засобів високоріневого проектування Matlab/Simulink, в тому числі М-функцій. Розробка формату команд для обробки комплексних чисел. Тестування процесору в пакеті Matlab/Simulink з використанням програми, записаної в пам'ять команд в машинних кодах. Генерація VHDL-коду в додатку HDL Coder і тестування процесору за допомогою САПР цифрових пристроїв.

5. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Завдання видав: (підпис консультанта, дата)	Завдання прийняв: (підпис студента, дата)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Огляд процесорів та засобів проектування	20.04.2022	10%
2	Розробка структури процесору	10.05.2022	10%
3	Розробка функціональних вузлів процесору в пакеті Matlab/Simulink	10.10.2022	45%
4	Генерування VHDL-коду в HDL Coder та аналіз цього коду	20.10.2022	5%
5	Тестування процесору в пакеті Matlab/Simulink	17.11.2022	15%
6	Моделювання процесору в САПР цифрових пристроїв	8.12.2022	10%
7	Оформлення кваліфікаційної роботи, підготовка презентації та доповіді	18.12.2022	5%
8	Подання кваліфікаційної роботи до кафедри	19.12.2022	
9	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	27.12.2022	

Студент _____ Дмитро ЛАЗОРЕНКО
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник роботи _____ Володимир ШАПОВАЛОВ
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи магістра: 82 с., 51 рис., 5 табл., 3 додатки, 13 джерел.

Об’єкт розробки – процесор виконання математичних операцій додавання та віднімання над комплексними числами з використанням ПЛІС.

Мета роботи – розробка та дослідження процесора виконання математичних операцій додавання та віднімання над комплексними числами з використанням ПЛІС та стенду для його тестування.

Методи дослідження включають в себе використання модельно орієнтованого проектування та моделювання в програмному пакеті Matlab/Simulink, генерацію у HDL Coder VHDL-опису процесора та його тестування в САПР Qesta Sim.

В ході виконання кваліфікаційної роботи проведено огляд спроектованих процесорів за допомогою модельно-орієнтованого програмування. Розглянуті етапи проектування процесору за допомогою пакету програм MatLab/Simulink. Було виконане проектування основних компонентів процесору обробки комплексних чисел та реалізована робота з пам’яттю для виконання інструкцій додавання та віднімання комплексних чисел. Проведені дослідження розробленного процесору разом з пам’яттю шляхом створення Simulink-моделі системи та подальшого її моделювання. Була написана тестова програма мовою Matlab для тестування і отримано VHDL опис системи, проведено синтез та веріфікацію спроектованого процесору.

Результати роботи можуть стати основою для створення процесору з розширеним набором команд, використані методи високорівневого проектування можуть значно прискорити розробку процесорів.

Ключові слова: ПРОЦЕСОР, RISC, ПЛІС, КОМПЛЕКСНІ ЧИСЛА, MATLAB, Simulink, VHDL, САПР

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ 5

ВСТУП 6

1 ОГЛЯД ПРОЦЕСОРІВ ТА ЗАСОБІВ ПРОЕКТУВАННЯ 8

1.1 Функціональні вузли обчислювальних пристроїв.....8

1.2 Огляд архітектури процесорів CISC та RISC.....8

1.3 Можливості програмного пакету Matlab/Simulink при проектуванні цифрових схем.....9

1.4 Мови опису апаратури та особливості VHDL.....12

1.5 Особливості використання модельно орієнтованого проектування в Matlab/Simulink 14

1.6 Операції додавання та віднімання комплексних чисел та деякі можливості їх використання в Matlab 19

1.7 Огляд існуючих реалізацій процесорів з використанням модельно орієнтованого проектування.....21

1.8 Розробка маршруту проектування.....22

2 ПРОЕКТУВАННЯ ПРОЦЕСОРУ В ПРОГРАМНОМУ ПАКЕТІ MATLAB/SIMULINK.....24

2.1 Верхній рівень проекту.....24

2.2 Пристрої пам'яті розробляємого процесору 24

2.3 Розробляємий процесор в пакеті Simulink.....28

2.4 Пристрої управління процесору.....29

2.5 Тракт даних процесору.....32

2.5.1 Аналізатор інструкцій та формати команд процесору.....32

2.5.2 АЛП процессору та вибір формату даних.....35

2.5.3 Керуючі сигнали.....36

3 МОДЕЛЮВАННЯ ПРОЦЕСОРУ В ПАКЕТІ MATLAB/SIMULINK.....40

4 ГЕНЕРАЦІЯ VHDL КОДУ ТА TESTBENCH ЗА ДОПОМОГОЮ MATLAB/SIMULINK.....46

4.1 Використання САПР Questa Sim за допомогою технології HDL Verifier.....46

4.2 Використання технології HDL Verifier.....52

ВИСНОВКИ.....57

ПЕРЕЛІК ПОСИЛАНЬ.....58

ДОДАТКИ

ДОДАТОК А. VHDL-код процесору……..60

ДОДАТОК Б. VHDL код для блоків пам’яті …….78

ДОДАТОК В ……84

ПЕРЕЛІК СКОРОЧЕНЬ

АЛП – арифметико-логічний пристрій;

ОЗП – оперативно запам’ятовуючий пристрій;

ПЗП – постійний запам’ятовуючий пристрій;

САПР – система автоматизованого проектування;

ПЛІС (FPGA) – програмуєма логічна інтегральна схема;

DSP – Digital signal processor, цифровий сигнальний процессор;

HDL – Hardware description language, мова опису апаратури;

ВСТУП

При розробці сучасних різноманітних пристроїв цифрової обробки інформації на основі ПЛІС все більше використовуються засоби автоматизації і високорівневого проектування, які дають можливість суттєво зменшити терміни, підвищити надійність і спростити проектування. Засоби високорівневого проектування апаратури вбудовують в популярні мови високорівневого програмування і пакети автоматизації математичних обчислень.

Використання модельно орієнтованого проектування в програмному пакеті Matlab/Simulink може у рази прискорити час розробки складних технічних систем, в тому числі і обчислювальних систем, що містять в якості обчислювачів ПЛІС, за рахунок ранього знаходження помилок проектування, можливості модулювання, автоматичної генерації коду та повторного використання попередніх напрацювань.

При роботі в Simulink, на початковому етапі потрібно розробити системну модель, на якій буде перевірено виконання вимог технічного завдання. Отримавши на цьому етапі первинний опис системи, необхідно виконати розробку і деталізацію функціональних вузлів і алгоритмів за допомогою інструментів, бібліотек Simulink і функцій MATLAB. У процесі зміни та деталізації функціональних вузлів і алгоритмів виконується безперервна верифікація щодо еталонної роботи пристрою.

Розглянуті принципи модельно орієнтованого проектування, налагодження функціоальних вузлів і алгоритмів на моделях, які з точністю до біта і такту відповідають згенерованому коду HDL, дозволяють скоротити час верифікації HDL коду і впровадження апаратури у інформаційні системи.

Представлена робота складається зі вступу, чотирьох основних розділів та висновків.

У першому розділі проведено огляд основних принципів проектування апаратури і зокрема процесорів. Розглянуто можливості прискорення проектування цифрових систем за допомогою об'єктно і модельно орієнтованого проектування. Був проведений огляд та зроблені висновки з досвіду проектування процесорів за допомогою Matlab/Simulink. Розглянуті особливості виконання математичних операцій над комплексними числами.

У другому розділі наведена структура розроблювальної системи, створена в

програмному пакеті Matlab/Simulink. Детально описано взаємодію її функціональних вузлів та керуючих сигналів його управління, наведено систему команд розробленого процесору.

У третьому розділі проведено симулювання розробленого процесору у Matlab/Simulink.

У четвертому розділі було згенеровано VHDL код та тест бенч для косимуляції процесора в САПР Qesta Sim та порівняні результати обчислених значень в Matlab/Simulink і САПР Qesta Sim.

1 ОГЛЯД ПРОЦЕСОРІВ ТА ЗАСОБІВ ПРОЕКТУВАННЯ

1.1 Функціональні вузли обчислювальних пристроїв

Сучасні комп’ютерні системи є універсальними пристроями, які мають спільні особливості побудови своєї роботи. Кожна така система має постійну та оперативну пам'ять, різноманітні обчислювальні блоки й керуючі пристрої.

Постійна пам’ять (ПП) є енергонезалежною, й зберігає програми для обчислення та необхідні вхідні дані.

Оперативна пам’ять (ОП) є енергозалежною та набагато швидшою за постійну пам’ять. В неї завантажуються дані, які необхідні для поточних обчислень.

Обчислювальні блоки виконують цифрові обчислення. Універсальні комп’ютери завжди мають арифметико-логічний пристрій для виконання базових арифметичних та логічних операцій. Також бувають пристрої для обчислень з плаваючою комою (підтримка нецілих чисел), цифрові сигнальні процесори (для обробки аналогових сигналів) та інші спеціалізовані блоки.

Керуючий пристрій організує обчислювальний процес, керуючи іншими блоками системи. Робота процесору – це повторюване виконання так званого «машинного циклу». Машинний цикл, як правило, складається з декількох стадій: вибірка команди з пам’яті, дешифрація команди, вибірка операндів, операція з операндами, запис результатів [1,2].

1.2 Огляд архітектури процесорів CISC та RISC

Всі цифрові процесори загального призначення розділяють на CISC і RISC процесори. CISC (Complex Instruction Set Computer) – комп’ютер з комплексною (складною) системою команд. RISC (Reduced Instruction Set Computer) – комп’ютер з системою команд зменшеної складності. Архітектура RISC-процесорів заснована на тому, що вони мають простий, можливо навіть надлишковий формат команд. Це забезпечує нескладну , через це, швидку дешифрацію команд й адрес.

Основні особливості архітектури RISC:

1) Найпростіша команда повинна виконуватися за один такт.

2) Операції над операндами виконуються у регістровій пам’яті. Це зроблено для того, щоб проміжні результати довше затримувались у регістрах і не переписувалися до ОЗП, щоб мінімізувати кількість обмінів між регістрами та ОП.

3) Простий, зазвичай надлишковий формат команд, який включає код операції та поля адрес безпосередніх операндів, забезпечує нескладну і швидку дешифрацію команд і адрес.

У таблиці 1 приведено порівняння основних характерстик процесорів RISC та CISC архітектур.

Таблиця 1 – Порівняняя RISC та CISC архітектур [3]

Характеристики	MIPS(RISC)	X86(CISC)
Регістровий файл	32 регістри	8, деякі обмежені по використанню
Кількість операндів	3 (2 джерела, 1 призначення)	2 (1 джерело, 1 жерело/призначення
Розташування операндів	Регістри або безпосередні операнди	Регістри або безпосередні операнди чи пам'ять
Розмір операндів	32 біти	8, 16, 32 біти
Коди умов	нема	присутні
Типи команд	прості	прості та складні
Розмір команд	фіксований, 32 біти	змінний, 1-15 байт

1.3 Мови опису апаратури та особливості VHDL

Мови опису апаратури (HDL) – дуже важливі інструменти розробників сучасної цифрової електроніки. Використання SystemVerilog або VHDL, дозволяє розробляти цифрові системи набагато швидше, ніж за традиційним креслення принципових схем. Цикл налагодження теж зазвичай набагато коротший, так як зміни полягають у редагуванні тексту, а не стомлюючему перепідключенні проводів на схемі. Однак з використанням HDL цикл налагодження може бути набагато довше, якщо погано уявляти, яку апаратуру описує написаний код. Мови опису апаратури використовуються і для симуляції, і для синтезу. Логічна симуляція – потужний спосіб протестувати

систему на комп'ютері, перш ніж вона перетвориться на апаратуру. Симулятори дозволяють перевірити значення сигналів у системі, які можуть бути недоступні для вимірювання на реальному електричної схеми. Логічний синтез перетворює код на HDL в цифрові логічні схеми.[3]

Оскільки схеми стають дедалі складнішими, мають місце дві паралельні тенденції:

- перехід до більш загальних форм вхідних даних, таких як опис поведінки системи,
- використання комп’ютерних систем автоматизації проектування.

Яку б функцію не виконувала система, вона має отримувати деякі вхідні дані і виводити деякі вихідні результати. Іншими словами, система має спілкуватись із середовищем.

Комунікаційна частина системи називається інтерфейсом. Системний інтерфейс описується у VHDL через блок інтерфейсу (entity) або просто інтерфейс, який є базовим елементом проектування будь-якої системи. Як неможливо створити систему без інтерфейсу, так і неможливо створити VHDL-систему без блоку інтерфейса.

Для досягнення певної функціональності дані повинні якимось чином перетворюватись всередині системи. Ця трансформація даних і вивід очікуваних функцій виконується внутрішньою частиною системи, або тілом системи, яке називається архітектурою (architecture).

Специфікація інтерфейсу

Оскільки починати будь-який проект необхідно з аналізу його середовища, не дивно, що блок інтерфейсу (entity), який описує інтерфейс між системою та її середовищем, є головною частиною кожного опису.

Не може існувати VHDL-специфікації якоїсь системи без декларації інтерфейсу. Крім того, все, що описано в інтерфейсі, автоматично стає видимим для усіх інших елементів проекту, зв’язаних з цим інтерфейсом. Більше того, ім’я системи завжди співпадає з іменем її інтерфейсу.

Склад інтерфейсу

Блок інтерфейсу забезпечує специфікацію інтерфейсу системи і звичайно складається з трьох елементів: заголовку інтерфейсу, параметрів системи (наприклад, ширина шини даних процесора), з'єднання, за допомогою яких інформація передається в систему та з неї (входи та виходи системи).

Два ключових елементи довільного інтерфейсу (параметри та з'єднання) у кожному інтерфейсі декларуються окремо:

- усі параметри декларуються як generics і передаються в тіло системи,
- з'єднання, через які передаються дані, називаються портами (ports) – вони формують другу частину інтерфейсу

Типи архітектурних описів

Кожна система може бути описана як в термінах її функціональних можливостей, (поведінки), так і в термінах її структури, відповідно тому, яка інформація про систему потрібна. Засоби синтезу працюють з обидвома типами архітектурного опису: перший, який задає очікуване функціонування, має бути специфікований і формалізований, після чого він трансформується в структурний стиль, який більш придатний для засобів синтезу.

Поведінковий опис

Функціональний опис визначає те, що система, як очікується, буде робити. Наприклад, необхідно описати, як деяка система відреагує на вхідні сигнали від кнопок пульта дистанційного керування.

Поведінкова специфікація взагалі – це опис виходів як реакцій на вхідні дані.

Необхідно зазначити, що поведінковий опис не відповідає, як ці функціональні можливості системи повинні бути реалізовані.

```
entity Set is
    port (ON,OFF : one-bit input
    ...
    )
end entity Set;
architecture Control of Set is
```

```
        ...
        if ON then turn on the system
        if OFF then turn off the system
        ...
end architecture Control;
```

Структурний опис

Структурний опис не торкається функціонування системи, а замість цього визначає компоненти, що повинні використовуватися, і як вони повинні бути з'єднані, щоби досягти очікуваних результатів. Іншими словами, він описує внутрішню структуру системи.

Структурний проект набагато легше синтезувати, ніж поведінковий, тому що він будується на конкретних фізичних компонентах. Однак, створення структурного опису системи дещо складніше, оскільки вимагає детального знання компонентів і принципів їх дії для підвищення його ефективності.

```
entity Set is
        port (ON,OFF : one-bit input
        ...
        )
end entity Set;
architecture Control of Set is
        ...
        U1,U5  -> U2  -> U4
        U1  -> U3-> U5
        U3  -> U5-> U2
        U2  -> U4
        ...
end architecture Control; [4]
```

1.4 Можливості програмного пакету Matlab/Simulink при проектуванні цифрових схем

Matlab – це потужний програмний пакет для технічних обчислень. В цій середі використовується своя високорівнева мова для програмування та розробки алгоритмів, Matlab дає широкий спектр можливостей для аналізу, візуалізації та чисельних розрахунків.

Matlab розширяєтья за допомогою програмних пакетів для обробки сигналів, зображень, статистики, оптимізації, символьної математики та інше.Їх можна додати при інсталяції Matlab або за допомогою зв'язу з офіційним сайтом компанії «MathWorks.

Simuink – це провідна середа для системного моделювання, симуляції та верифікації систем зв'язку, електронних систем та систем управління. Simulink – це графічна середа, яка використовує блок-схеми для системного моделювання і верифікації. Цей пакет дозволяю розробляти цифрові, аналогові та змішані системи, які містять дискретну, непреривну та подієву логіку. Є можливість оперувати числами з плаваючою та фіксаваною крапкою з використанням можливостей Matlab, блоків Simulink, написаного С-коду, а також діаграм станів StateFlow.

Simulink, як і Matlab, розширяється та пропонує рішення для систем управління, систем обробки сигналів, відео та зображень, систем зв'язку, радіочастотних систем та інших областей з використанням системного інжинірингу. Використовуючи Simulink можна генерувати C/C++ код для цифрових сигнальних процесорів DSP , генерувати VHDL або Verilog код для ПЛІС за допомогою HDL-coder для подальшого його впровадження в апаратуру. Додатки пакету Matlab/Simulink представлені на рисунку 1.1.

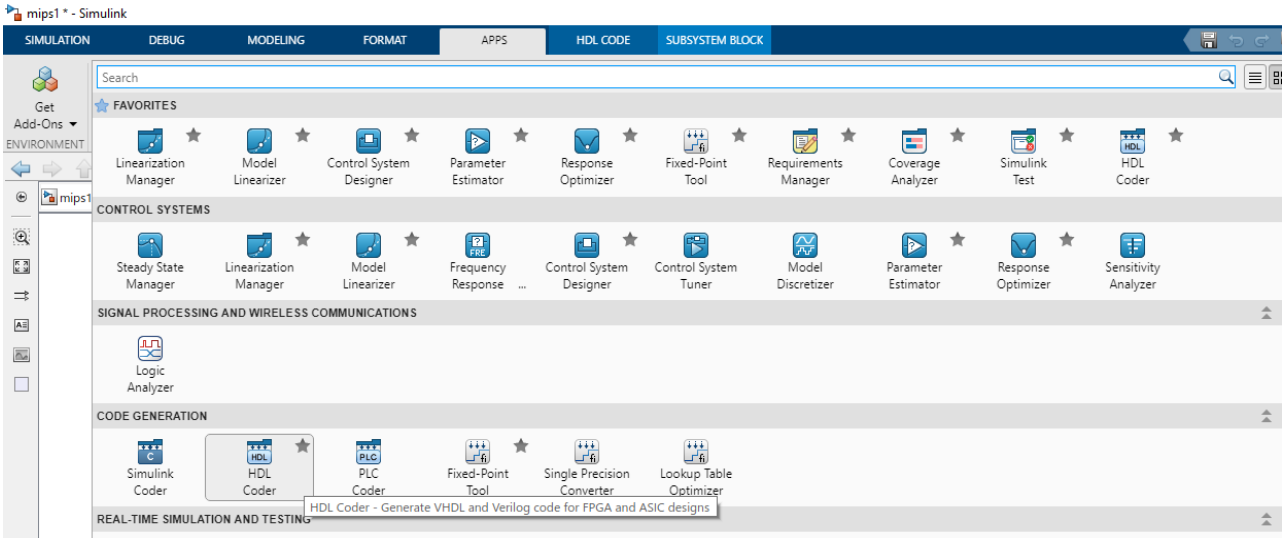


Рисунок 1.1 – Додатки пакету Matlab/Simulink

1.5 Особливості використання модельно орієнтованого проектування в Matlab

HDL Coder – автоматичний генератор синтезованого Verilog або VHDL коду з підмножини функцій MATLAB, моделей Simulink та діаграм Stateflow. Відповідні піктограми представлені на рисунку 1.2. Моделі Simulink знаходяться у бібліотеці компонентів зображеній на рисунку 1.3, вона містить синтезовані блоки на мові Matlab, використання вже готових блоків значно пришвидшує процес проектування.

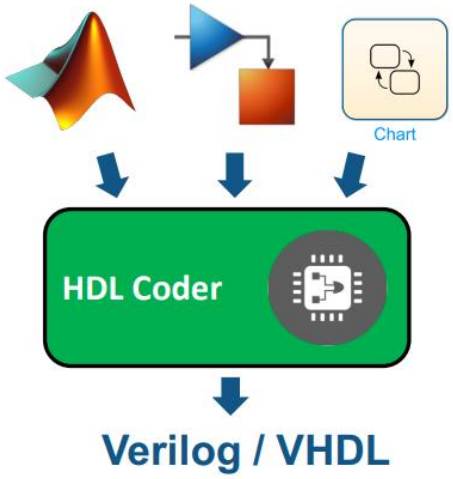


Рисунок 1.2 – Піктограми HDL Coder, функцій MATLAB, моделей Simulink та діаграм Stateflow

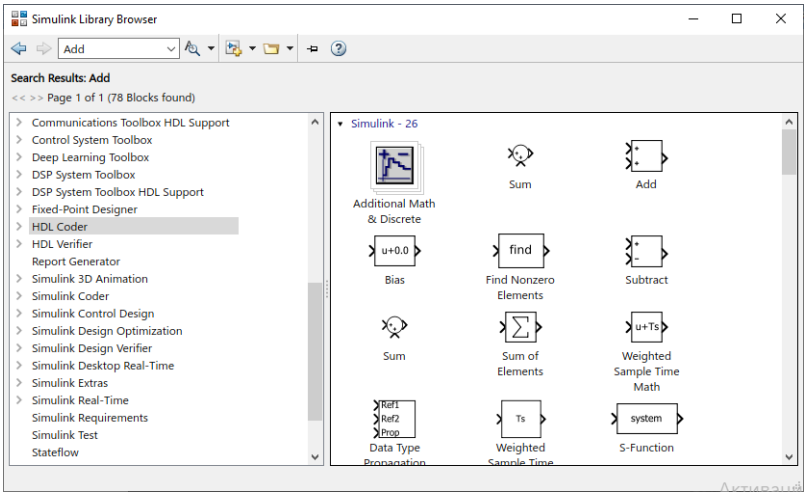


Рисунок 1.3 – Бібліотека компонентів в Simulink

Для того щоб писати функції потрібно познайомитись з структурою її опису:

function [y1,...,yN] = myfun(x1,...,xM) – оголошує функцію з ім'ям myfun, яка приймає вхідні дані x1,...,xM і повертає вихідні параметри y1,...,yN. Цей оператор повинен бути першою функцією, що виконується. Допустимі імена функцій починаються з літерного символу і можуть містити літери, числа або символи нижнього підкреслення. Ім'я файлу має збігатися з ім'ям першої функції у файлі. У файлі скрипта, який містить команди та функціональні визначення. Функції мають бути наприкінці файлу. Файли скрипта не можуть мати те саме ім'я як функція у файлі. Файли можуть містити кілька локальних або вкладених функцій. Для зручності читання краще за все використовувати ключове слово end, щоб вказати на кінець кожної функції у файлі. Ключове слово end потрібно коли: будь-яка функція файлу містить вкладену функцію.

Приклад використання піктограм Stateflow та M-функцій зображені на рисунку 1.4.

Функція зображана на рис. 1.5 має назву ALU, п’ять вхідних значень: controls, X1_R, X2_R Y1_R,Y2_R та два вихідні: Zr_L, Zi_L.

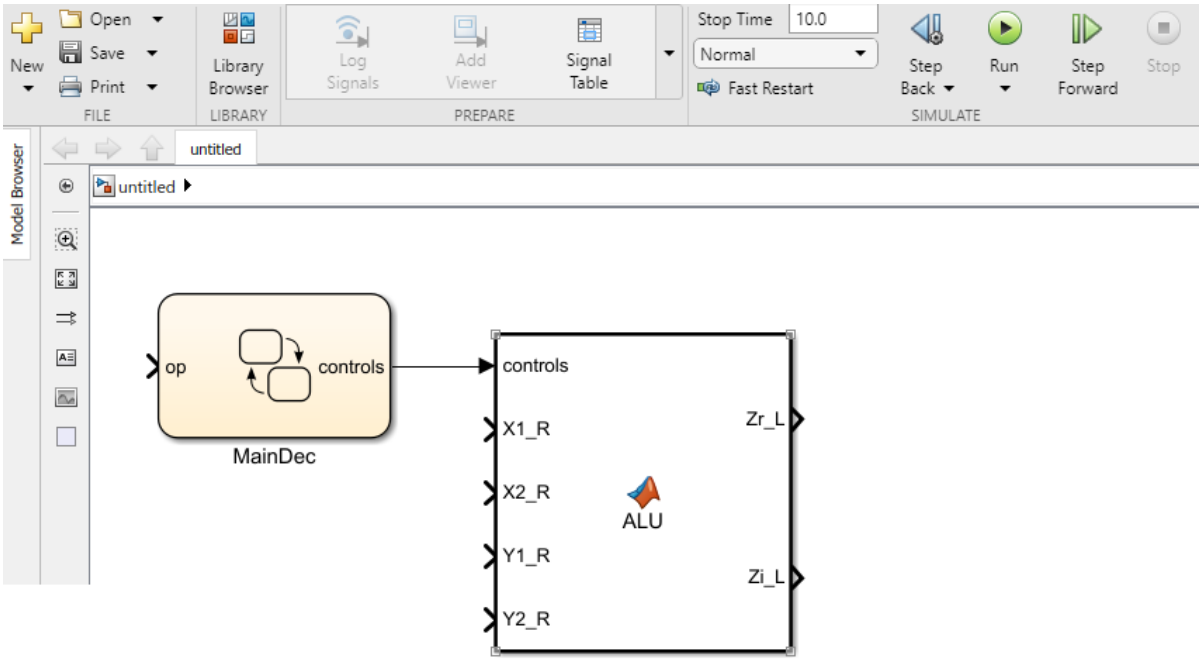


Рисунок 1.4 – Приклад діаграми станів та M-функції в середовищі Simulink

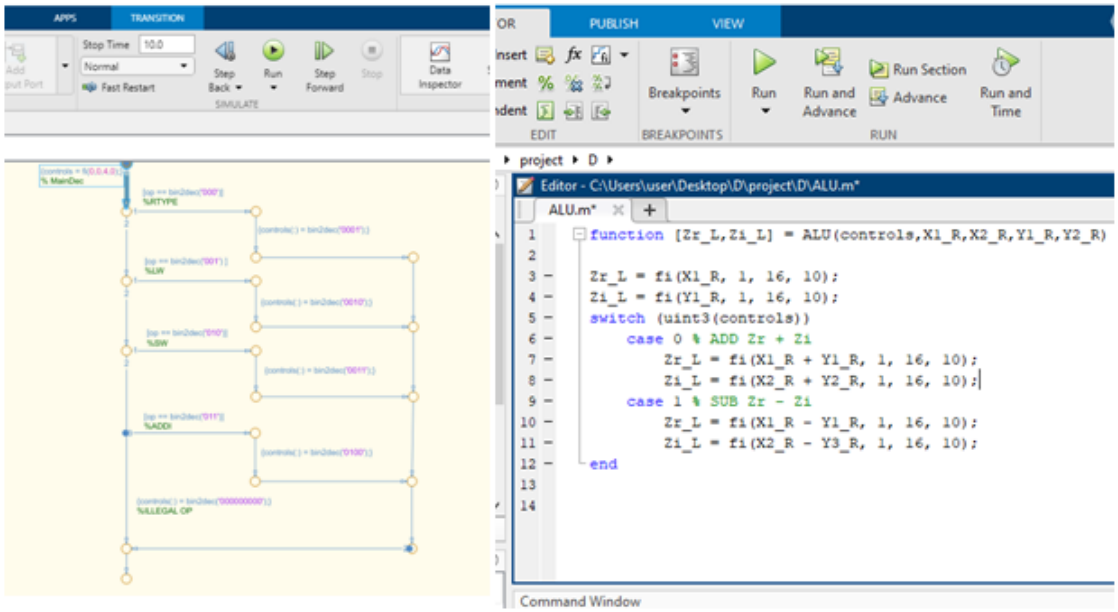


Рисунок 1.5 – Приклад реалізації діаграми станів та функції Matlab

У Simulink є інструмент для перегляду конфігурації проекту у вигляді ієрархічної таблиці – Model explorer. В цих таблицях можна швидко знайти конфігурацію потрібного компонента схеми та відредагувати його, меню цього додатку зображено на рисунку 1.6.

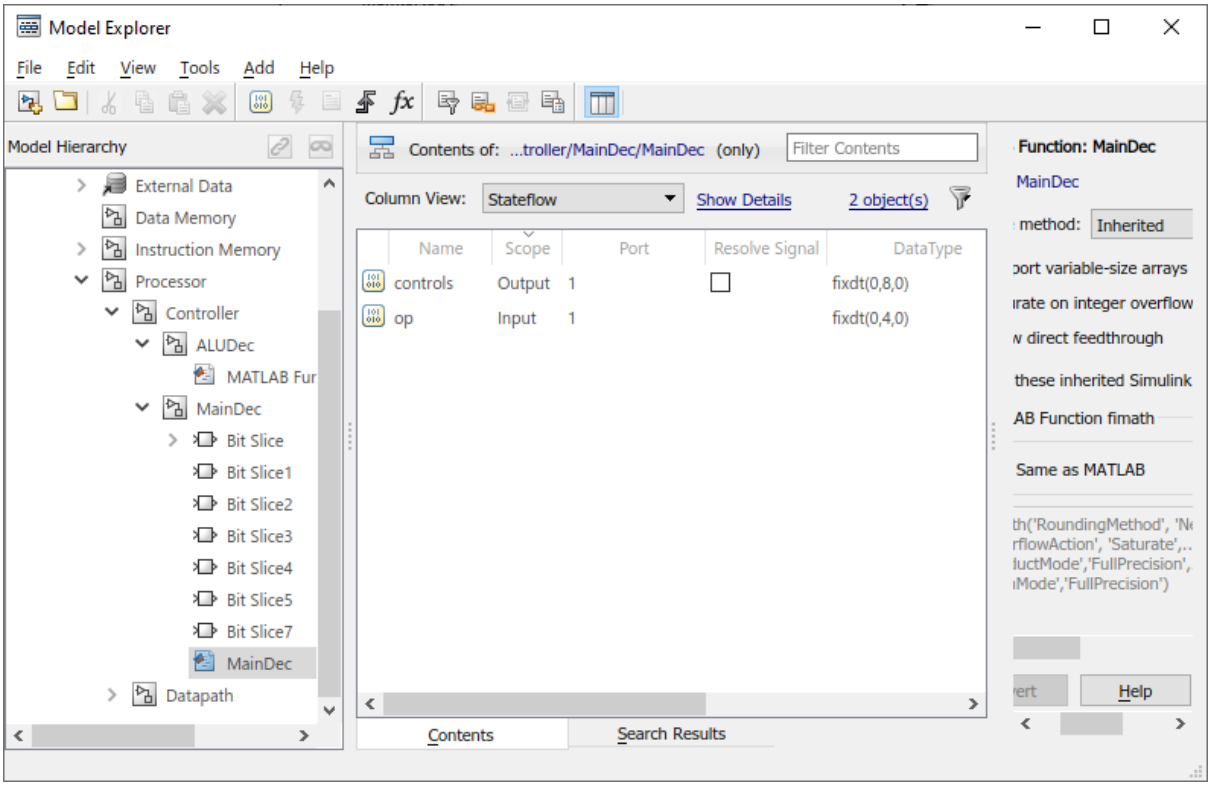


Рис. 1.6 – Меню додатку Model explorer у середовищі Simulink

Також є інструмент аналізу, який дозволяє відслідковувати покрокову роботу потрібних сигналів для розуміння як працює побудована схема. Для цього потрібно обрати необхідне з'єднання та промаркувати потрібні сигнали за допомогою піктограми Log signal яка зображена на рисунку 1.7 та запустити модель на симуляцію, після якої потрібно запустити інструмент Data inspector зображений на рисунку 1.8.

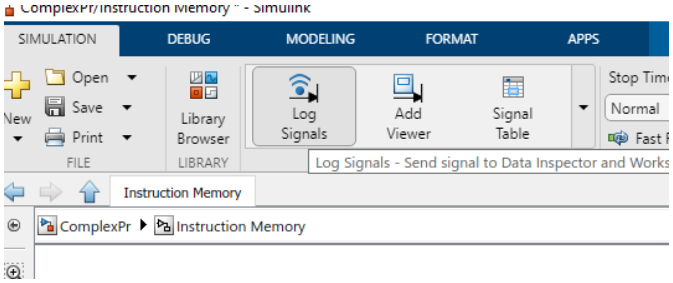


Рисунок 1.7 – Піктограма Log signal

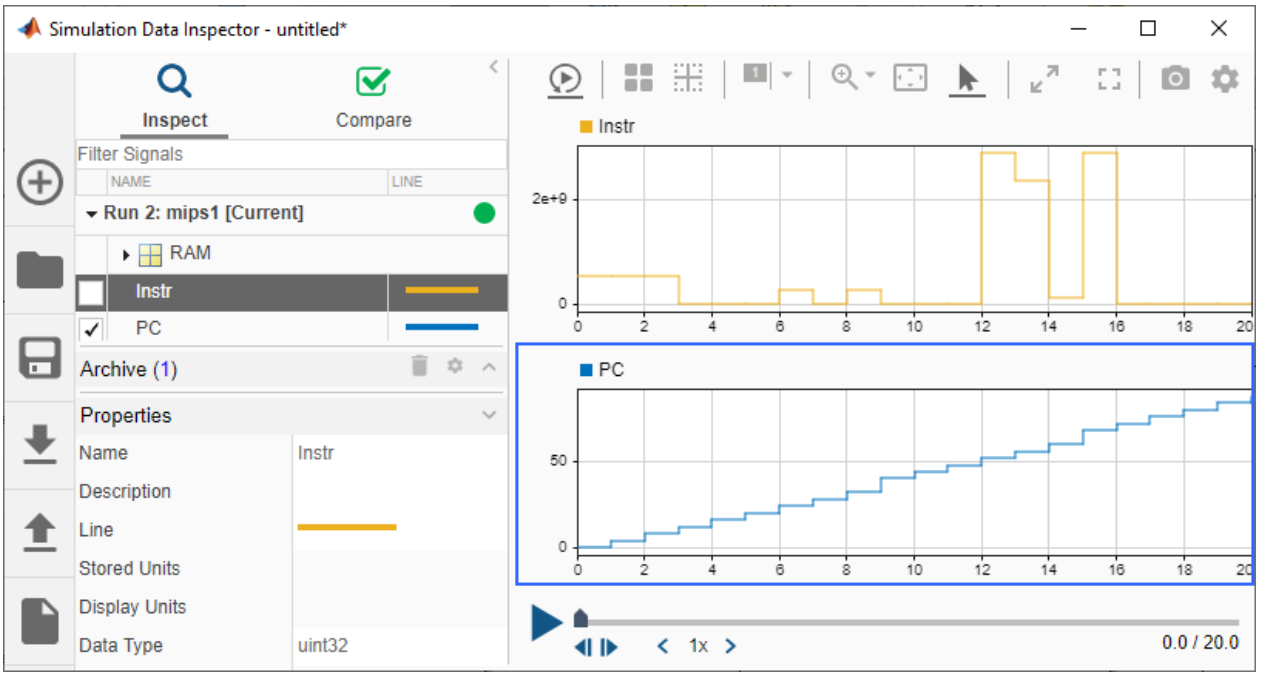


Рис. 1.8 – Відслідковування потрібних сигналів в Data inspector

Після того, як ми впевнились що побудована модель працює коректно ми можемо генерувати HDL код. Для автоматичного генерування HDL коду потрібно в додатку Simulink обрати HDL Coder та зробити основні налаштування такі як: необхідна мова опису стиль Verilog/VHDL, стиль опису, параметри автоматичної оптимізації та інше, вікно основних налаштувань HDL

Coder зображені на рисунку 1.9. Після генерування автоматично відкривається звіт, який містить: налаштування які ми обрали, звіт з використання ресурсів, по трасуванню, який показує як кожен блок моделі відповідає відповідному рядку коду. Згенерований код формується окремо для кожного блоку моделі, сам код є досить цілісним та якісним, самостійне редагування не рекоменується, досвідчені проектувальники рекомендують редагувати саму схему з якої генерувався код, приклад звіту згенерованого коду зображений на рисунку 1.10. Наступним кроком є верифікація цього коду в зручному для розробника HDL сімуляторі.

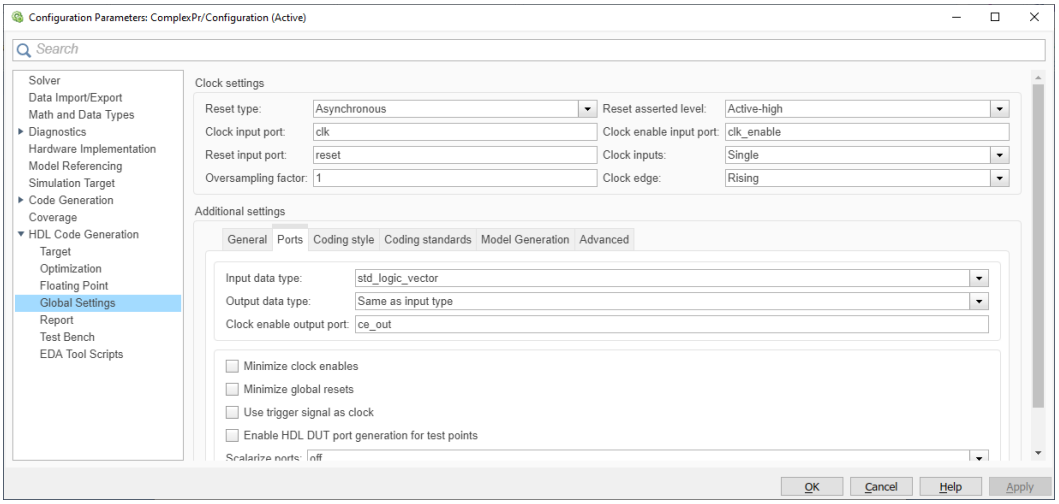


Рисунок 1.9 – Вікно основних налаштувань HDL Coder

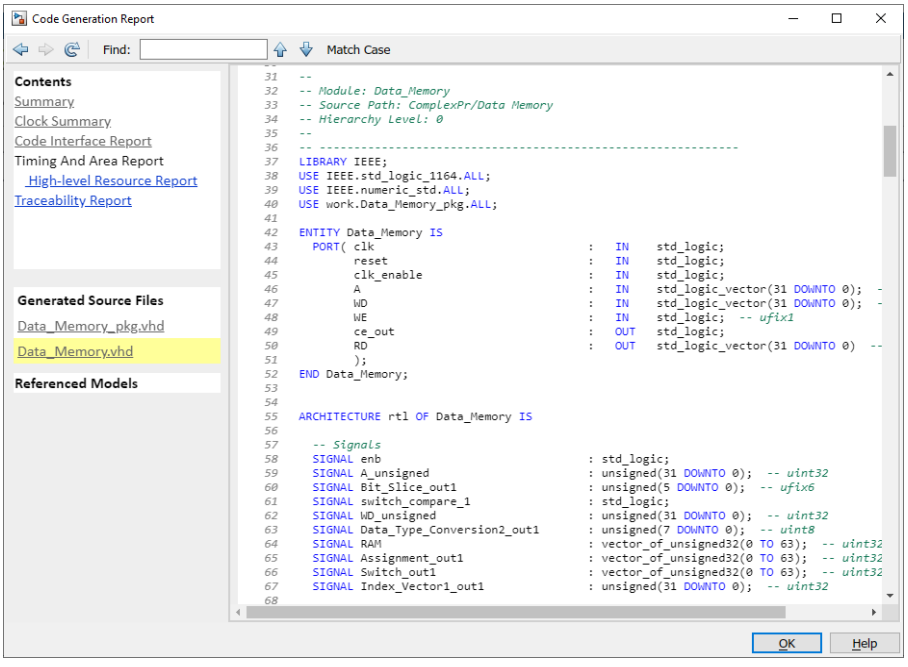


Рисунок 1.10 – Звіт генерації коду в HDL Coder

Для верифікації використовується технологія HDL Verifier, яка дозволяє проводити косимуляцію, тобто створити зв'язок між симулятор HDL і MATLAB або Simulink. Ця технологія працює за допомогою майстру косимуляції Cosimulation Wizard, який за допомогою графічного інтерфейсу користувача GUI дозволяє налаштувати зв'язок між моделю та HDL симулятором.

Технологія HDL Verifier підримує наступні симулятори: «Vivado» компанії Xilinx, «ModelSimor» та «Questa Sim» фірми Mentor Graphics, Xcelium компанії Cadence. [5]

1.6 Операції додавання та віднімання комплексних чисел та деякі можливості їх використання в Matlab

Комплексні числа використовуються при описі багатьох завдань фізики та техніки. Основні розділи класичного математичного аналізу набувають повної ясності і закінченості лише за використання комплексних чисел, чим зумовлюється центральне місце, яке займає теорією функцією комплексного змінного.

Комплексними числами (x,y) можна називати пари дійсних чисел x та y, для яких правила виконання арифметичних операцій додавання та віднімання визначені наступним чином.

Означення комплексного числа і уявної одиниці

Число x+y*i, де x і y – будь-які дійсні числа, «i» уявна одиниця, називається комплексним числом (x– дійсна частина, y— уявна частина комплексного числа).

Число, квадрат якого дорівнює −1, позначають латинською буквою «i» і називають уявною одиницею.

Тобто, для символу і виконується наступна рівність

$$i*i=i^2=-1.$$

Два комплексних числа $x+yi$ і $c+di$ рівні між собою тоді і тільки тоді, коли $x=c$ і $y=d$,тобто, коли рівні їх дійсні частини і коефіцієнти при уявних частинах.

Поняття «більше» і «менше» для комплексних чисел не має сенсу. Ці числа за величиною не порівнюють. Тому не можна, наприклад, сказати, яке з двох комплексних чисел більше $11i$ чи $3i$, $2+11i$ чи $5+2i$.

Дії над комплексними числами:

Додавання комплексних чисел

Нехай дано два комплексні числа $z_1 = x_1 + y_1 i$ і $z_2 = x_2 + y_2 i$

Сумою двох комплексних чисел $x_1 + y_1 i$ і $x_2 + y_2 i$ називається комплексне число $(x_1 + y_2) + (x_1 + y_2)i$ дійсна частина якого і коефіцієнт при уявній частині дорівнюють відповідно сумі дійсних частин і коефіцієнтів при уявних частинах додатків.

Приклади додавання комплексних чисел:

1. $(-3+5i)+(4-8i)=(-3+4)+(5-8)i=1-3i$
2. $(2+3i)+(6-3i)=(2+6)+(3-3)i=8-0i=8$
3. $(10-3i)+(-10+3i)=(10-10)+(-3+3)i=0-0i=0$

Віднімання комплексних чисел

Різницею двох комплексних чисел $x_1 + y_1 i$ і $x_2 + y_2 i$ називається комплексне число $(x_1 - x_2) + (y_1 - y_2)i$

Приклади віднімання комплексних чисел:

1. $(-5+2i)-(3-5i)=(-5-3)+(2-(-5))i=-8+7i$
2. $(6+7i)-(6-5i)=(6-6)+(7+5)i=12i$ [6].

Деякі синтезовані блоки для обробки комплексних чисел з бібліотеки Simulink:

Complex to Magnitude-Angle (виділення з комплексного числа амплітуди та фази) – блок, що дозволяє отримати з вхідної комплексної величини амплітудно-фазові характеристики сигналу

Complex to Real-Imag (виділення з комплексного числа дійсної та уявної частини) – блок, який служить для виділення з комплексного числа дійсної та уявної частини.

Real-Imag to Complex (перетворення дійсної та уявної частини в комплексне

число) – блок, що дозволяє отримати з дійсної і уявної частини комплексне число зображений на рисунку 1.11. На його вхід подається реальна та уявна частини або тільки одна на вибір, а інша задається у самому блоці. Після перетворення, дві пари комплексних чисел надходять на суматор та результати обчислень надходять на дисплей [7].

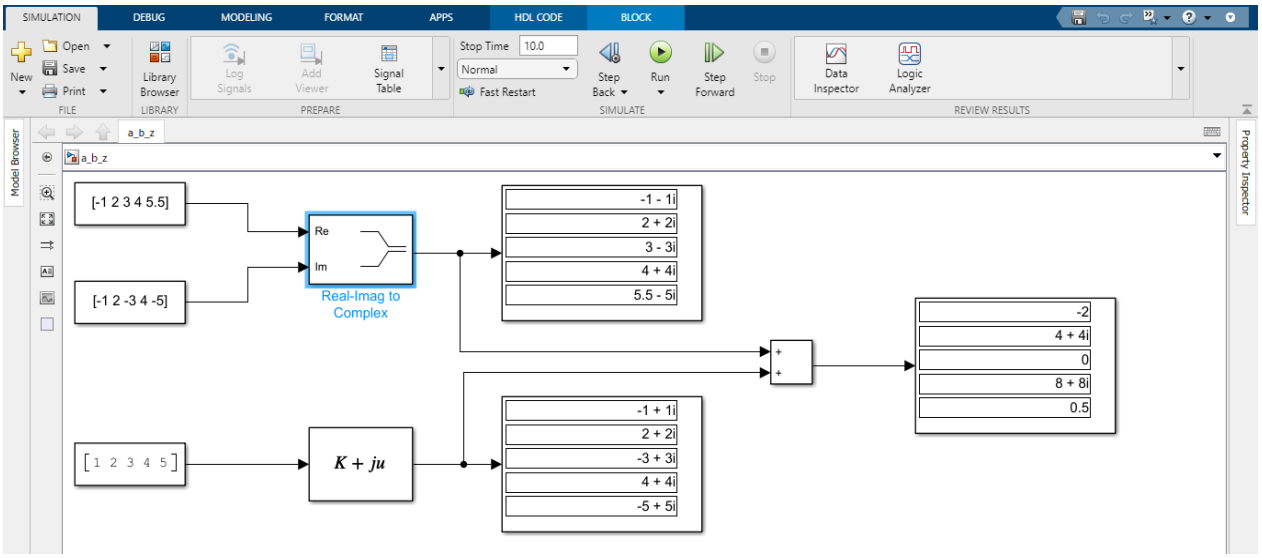


Рисунок 1.11 – Приклад використання блоку Real-Imag to Complex

1.7 Огляд існуючих реалізацій процесорів з використанням модельно орієнтованого проектування

У роботах [8,9,10,11,12] були розглянуті можливості системи візуально-імітаційного моделювання MATLAB/Simulink для проектування мікропроцесорних ядер з подальшою реалізацією у базисі ПЛІС. В цих працях автори реалізовували процесор на освнові кінцевого автомату описанного М-функціями.

Проектований процесор в цих роботах складався з наступних блоків: керуючий автомат, який керував роботою процесору, пам'ять програм – ПЗП процесора, АЛП процесора, два регістри загального призначення блоки RegisterA та RegisterB, блок PC_Inc, який необхідний для забезпечення команд переходів, таких як JMP, JMPZ, CALL та RET, лічильник команд – блок PC, регістр інструкцій – Instruction_Reg, в цих статях автори поступово

покращували системи команд процесору.

В деяких роботах арифметичні операції над операндами, були представленими у форматі з фіксованою комою, що давало більш високу точність. Програми в усіх цих працях були записані у ПЗП за допомогою опису М-функціями. Відмічалось про ефективність використання пакету MATLAB/Simulink з HDL Coder для прискорення процесу розробки моделей мікропроцесорних ядер.

Мною було прийнято рішення, опираючись на досвід реалізації в цих роботах опису основних блоків процесору, реалізувати однотактний процесор описаний у [3] для операцій додавання та віднімання комплексних чисел .

1.8 Розробка маршруту проектування

У кваліфікаційній роботі будуть використовуватись програмні пакети MATLAB/Simulink та проводитись верифікація за допомогою технології HDL Verifier у САПР QestaSim. Для розробки пристрою потрібно скласти маршрут проектування. Складений маршрут наведено на рисунку 1.12 .

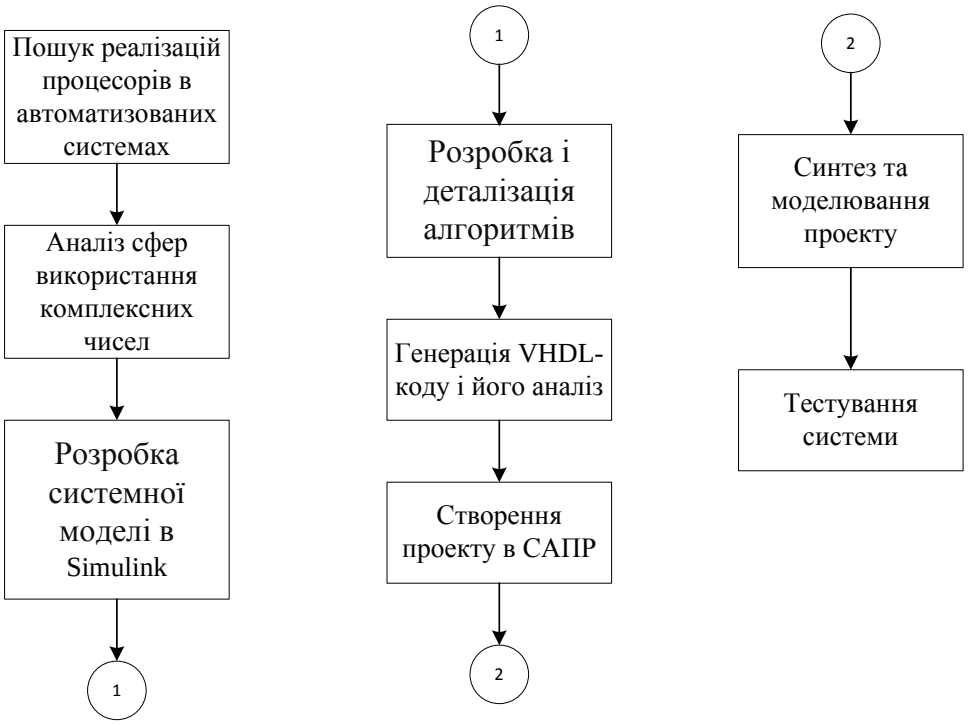


Рисунок 1.12 – Маршрут проектування системи

Цей маршрут включає у себе послідовність дій при проектуванні системи.

Спочатку проводився пошук та аналіз вже спроектованих процесорів в системах модельно-орієнтованого проектування. Після аналізується опарації додавання та віднімання комплексних чисел та формується формати команд для цих дій. Потім треба розпочинати проектування процесору та блоків пам'яті в пакеті Simulink та поступово за допомогою інструментів цього пакету вдосконалювати та деталізувати його, після чого потрібно за допомогою додатку HDL Coder згенерувати VHDL. Наступний етап – це створення віртуального стенду в САПР Qesta Sim за допомогою технології HDL Verifier для тестування та перевірку на наявність помилок.

2 ПРОЕКТУВАННЯ ПРОЦЕСОРУ В MATLAB/SIMULINK

2.1 Верхній рівень проекту

Компоненти у середовищі Simuink представлені у вигляді блочних діаграм. На рисунку 2.1 зображений верхній рівень розробляємого пристрою, на якому знаходиться процесор, який взаємодіє з блокими пам’яті команд ПЗП (Instruction memory) та з ОЗП(Data memory), які необхідні для його тестування. У цих блоках можна побачити вхідні та вихідні порти пристрою та типи даних якими вони оперують. Основна частина компонентів процесору описана за допомогою М-функцій, ПЗП реалізован М-функцією, ОЗП побудована з блоків бібліотеки компонентів Simulink. Опис блоків, що реалізовані за допомогою М-функцій буде приводитись під їхніми позначенням.

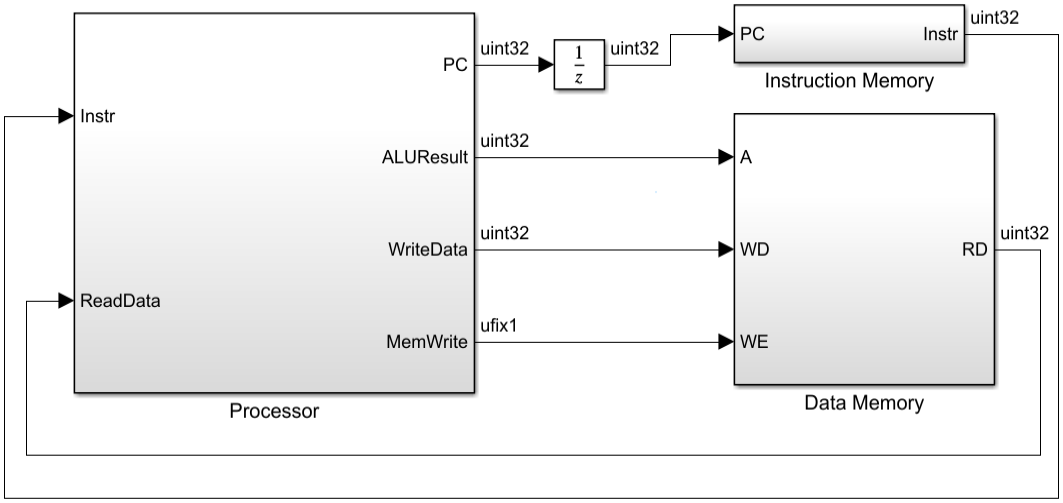


Рисунок 2.1 – Верхній рівень у додатку Simulink

2.2 Пристрої пам’яті розробляємого пристрою.

Проектування процесору доцільно розпочинати з елементів, які зберігають його стан при роботі, ці елементи включають:

- Постійний запам'ятовуючий пристрій, який реалізован М-файлом зображений на рисунку 2.2 – це пам'ять для зберігання інструкцій, які виконує процесор. Пам'ять інструкцій має один порт для читання, на адресний вхід PC подається 32-бітна адреса команди, після чого на виході RD з'являється 32-бітове

число тобто інструкція, прочитана з пам'яті за цією адресою.

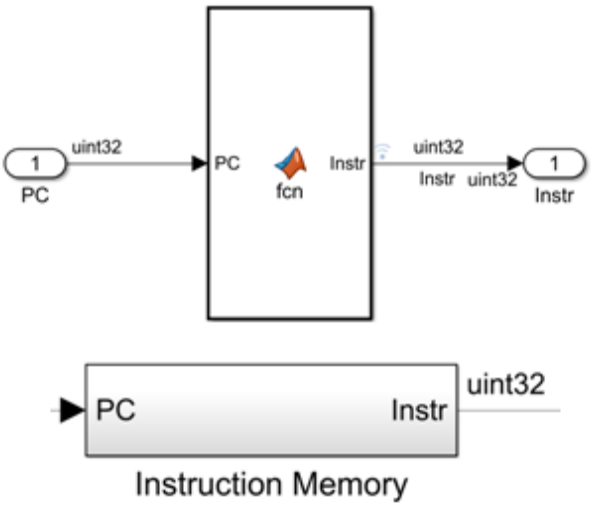


Рисунок 2.2 – Пам'ять інструкцій в Simulink

М-функція ПЗП:

```
function Instr = fcn(PC)
persistent RAM;
if isempty(RAM)
    RAM = fi(zeros(64, 1), 0, 32, 0);
    RAM(1) = hex2dec('30200007');
    RAM(2) = hex2dec('32300006');
    RAM(3) = hex2dec('30400005');
    RAM(4) = hex2dec('30600003');
    RAM(5) = hex2dec('02346781');
    RAM(6) = hex2dec('30a0000a');
    RAM(7) = hex2dec('30b00005');
    RAM(8) = hex2dec('30c0000b');
    RAM(9) = hex2dec('30d00006');
    RAM(10) = hex2dec('0abcdef2');
    RAM(11) = hex2dec('30000028');
    RAM(12) = hex2dec('20700008');
    RAM(13) = hex2dec('2080000c');
```

```
    RAM(14)  = hex2dec('1090000c');
end
Instr = RAM(bitsliceget(PC,8,3)+1);
```

- Оперативно запам’ятовуючий пристрій побудований з стандартних блоків Simulink зображений на рисунку 2.3 – це пам'ять для зберігання даних. Ця пам'ять має один порт для читання/запису, якщо сигнал дозволу запису WE дорівнює одиниці, то дані зі входу WD записуються в комірку пам'яті за адресою A, яка обчислюється в АЛП, по позитивному фронту тактового сигналу, якщо сигнал дозволу запису дорівнює нулю, дані за адресою A подаються на вихід RD регістрового файлу.

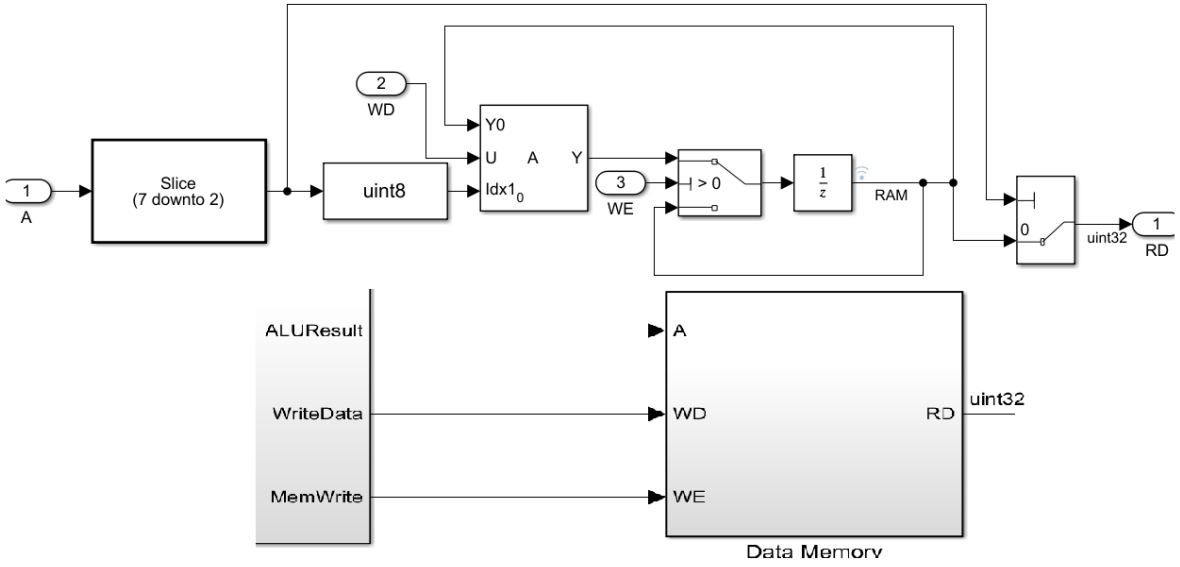


Рисунок 2.3 – Оперативно запам’ятовуючий пристрій в Simulink

- Лічильник команд (англ. program counter, PC) зображений на рисунку 2.4 – це регістр процесора, за допомогою якого визначається яка команда програми буде виконуватись процесором наступною. Його вихід PC містить адресу поточної команди, його вхід PC_in містить адресу наступної команди. Одночасно з виконанням команди процесор має обчислити адресу наступної команди PC_in, оскільки команди 32-бітові, тобто чотирибайтні, то адреса наступної команди дорівнює PC + 4.

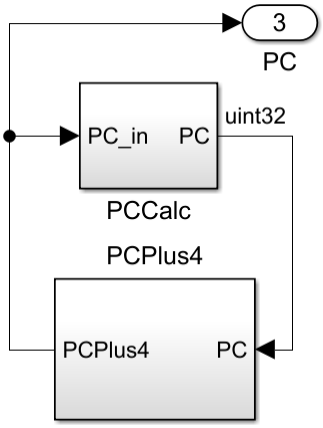


Рисунок 2.4 – Лічильник команд в Simulink

- Регістровий файл зображений на рисунку 2.5 він знаходяться в процессорі та реалізовані парою М-функцій: одна для запису в регістри, інша для читання – це видимі програмісту регістри, які призначені для зберігання операндів для арифметичних інструкцій.

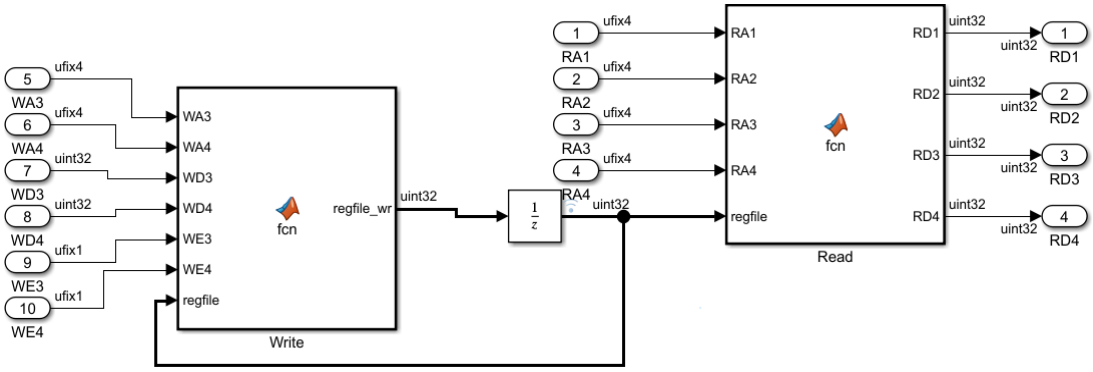


Рисунок 2.5 – Регістровий файл в Simulink

М-функція для запису в регістровий файл

```
function regfile_wr = fcn(WA3,WA4,WD3,WD4,WE3,WE4,  regfile)
regfile_wr = regfile;
    if (WE3)
        regfile_wr(int32(WA3)+1)  = WD3;
    end
    if (WE4)
        regfile_wr(int32(WA4)+1)  = WD4;
    end
```

```
end
```

М-функція для читання з регістрового файлу

```
function [RD1,RD2,RD3,RD4]  = fcn(RA1,RA2,RA3,RA4,  regfile)

RD1 = regfile(int32(RA1)+1);
RD2 = regfile(int32(RA2)+1);
RD3 = regfile(int32(RA3)+1);
RD4 = regfile(int32(RA4)+1);
End
```

Регістровий файл, містить 16 регістрів по 32 біти кожен, має 4 порти читання та два порти запису даних (результатів розрахунків). Порти читання мають чотирьох бітні входи адреси WA3 та WA4, кожен з яких визначає один з $2^4 = 16$ регістрів як джерело даних для команди. Кожен з чотирьох портів RA1-4 читає 32-бітове значення з регістру і подає його на виходи RD1-4 відповідно.

Порти запису (WD3, WD4) отримують чотирьох бітну адресу регістру в який потрібно записати результат на адресні порти (WA3,WA4), після цього 32-бітні числа надходять на вхід даних (WD3,WD4), сигнали дозволу запису WE3,WE4 та тактовий сигнал. Якщо сигнали дозволу запису дорівнюють одиниці, то регістровий файл записує дані у вказані регістри по позитивному фронту тактового сигналу.

Потім між цими елементами пам'яті потрібно розташувати комбінаційні схеми, що обчислюють новий стан процесору на основі поточного стану. Команда читається із тієї частини пам'яті, де знаходиться програма, команди завантаження та збереження потім читають або пишуть дані в іншу частину пам'яті. Тому зручно розділити пам'ять на дві менші за розміром частини, щоб одна містила команди, а інша – дані.

2.3 Розробляємий процесор в Simulink

На рисунку 2.6 зображений процесор в середовищі Simulink. Його доцільно розділити на дві умовні частини: тракт даних (Datapath) зображений на рисунку

2.7 та пристрій управління (Controller) дивись рисунок 2.8. Тракт даних складається з аналізатора інструкцій, лічільнику команд, регістрового файлу, АЛП та мультиплексорів, які керують роботою цих блоків в залежності від керуючих сигналів, які надходять від контролера.

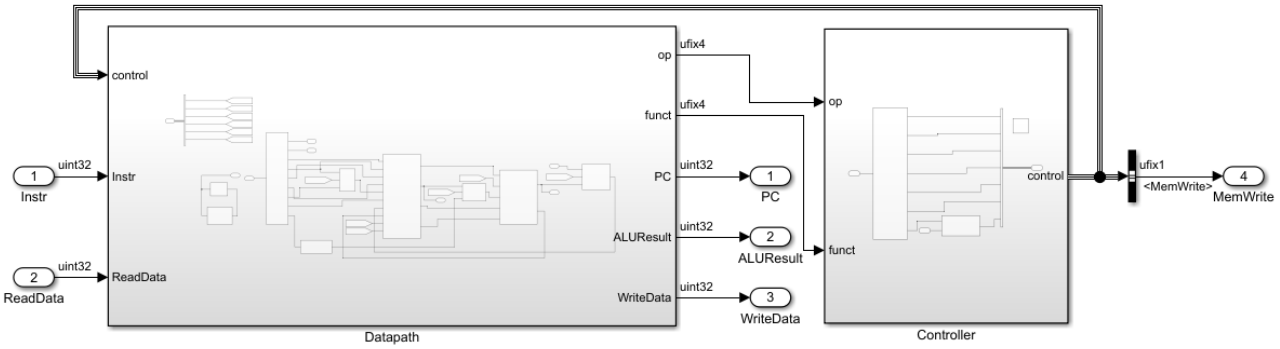


Рисунок 2.6 – Процессор у середовищі Simulink

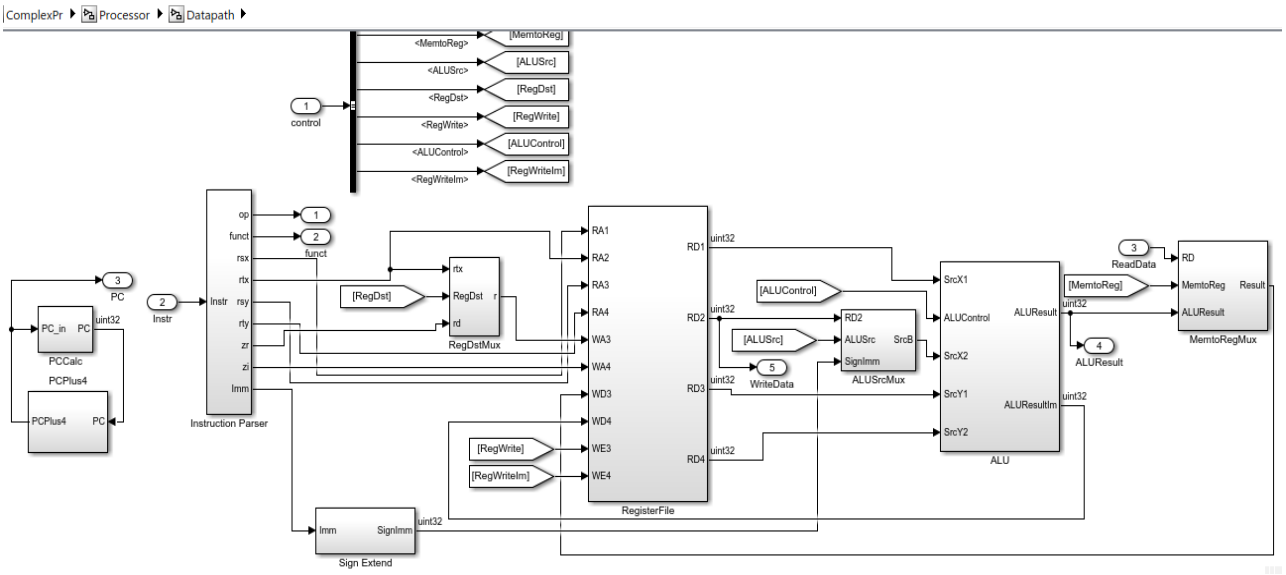


Рисунок 2.7 – Тракт даних

2.4 Пристрій управління процесору

Пристрій управління (controller) який зображений на рис. 2.8 реалізований двома М-функціями, формує сигнали, що управляють роботою процесору на основі полів opcode та funct, присутніх у кожній команді Instr31:28 та Instr3:0 відповідно.

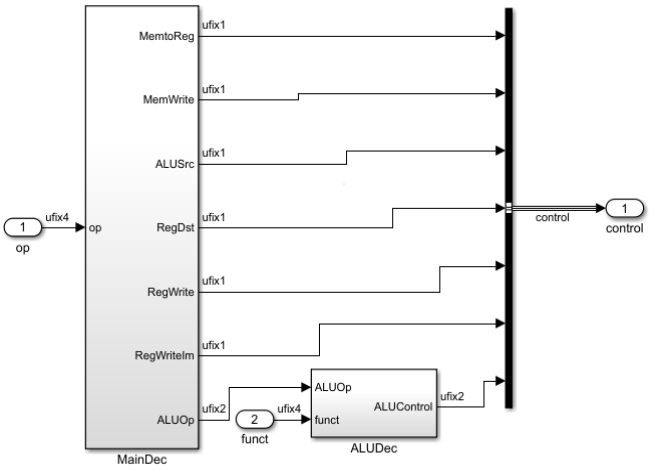


Рисунок 2.8 – Пристрі управління процесору

М-функція головного декодеру:

```
function controls = fcn(op)
controls = fi(0,0,8,0);
switch op
   case bin2dec('0000')
      %для команд compexadd/complexsub
      controls(:) = bin2dec('11100010');
   case bin2dec('0001')
      %LW
      controls(:) = bin2dec('10010100');
   case bin2dec('0010')
      %SW
      controls(:) = bin2dec('00011000');
   case bin2dec('0011')
      %ADDI
      controls(:) = bin2dec('10010000');
   otherwise
      %ILLEGAL OP
      controls(:) = bin2dec('000000000');
end
```

М-функція декодеру АЛП:

```
function ALUControl = fcn(ALUOp,funct)
ALUControl = fi(0,0,2,0);
switch ALUOp
  case bin2dec('00')       %для команд I-типу (lw/sw/addi)
    ALUControl(:) = bin2dec('01');
  otherwise
      switch funct
          case bin2dec('1000')
          %add
          ALUControl(:) = bin2dec('01');
          case bin2dec('0001')
          %complexadd
          ALUControl(:) = bin2dec('10');
          case bin2dec('0010')
          %complexsub
           ALUControl(:) = bin2dec('11');
          otherwise
          %ILLEGAL OP
          ALUControl(:) = bin2dec('00');
      end
end
```

У таблиці 2.1зображена таблиця істинності для основного дешифратора, який обчислює значення більшості виходів на основі поля opcode. Він також формує двобітний сигнал ALUOp. Дешифратор АЛП (ALU decoder) використовує ALUOp разом із полем funct для обчислення стану ALUControl, який надходить у АЛП для декодування операції. Розшифрування значень сигналів АЛП наведено у таблиці 2.2.

Таблиця 2.1 – Таблиця істинності основного дешифратора

Command	opcode	RWrite	RWriteIm	RegDst	ALUSrc	MemWrite	MemtoReg	AluOp
Complextype	0000	1	1	1	0	0	0	10
Lw	0001	1	0	0	1	0	1	00
Sw	0010	0	0	0	1	1	0	00
addi	0011	1	0	0	1	0	0	00

Таблиця 2.2 – Таблиця істинності дешифратора АЛП

ALUOp	funct	ALUControl	Операція
00	x	01	addi
xx	1000	01	add
xx	0001	10	complexadd
xx	0010	11	complexsub

2.5 Тракт даних процесору

2.5.1 Аналізатор інструкцій та формати інструкцій процесору

В тракт даних надходить 32-бітна інструкція (Instr), яка розбивається в залежності від значень полів команди в тракті даних в блоці Instruction Parser (аналізатор інструкцій), який сображений на рисунку 2.9.

Інструкції процесору поділяються на два типи: інструкції для комплексних чисел «complex-type», до них входять комплексне додавання та віднімання: complexadd та complexsub відповідно, та інструкції роботи з пам'яттю «I-type»: lw (load word), sw (save word) та addi(addimmidiate). Для того щоб виконати операцію додавання чи віднімання комплексних чисел потрібно 4 регістри джерела Rsx, Rtx, Rsy, Rty та два регістри результатів Zr та Zi.

Формат команди I-type зображений в таблиці 2.3 використовують в якості операндів два регістри та один безпосередній операнд (константу), 32-бітна інструкція складається з чотирьох полів: opcode, rsx,rtx та imm. Поле imm (скор. від англ. immediate) містить 16-бітну константу, операція визначається виключно полем opcode. Операнди задані у трьох полях: rsx, rtx та imm, поля rs та imm

завжди використовуються як операнди-джерела. Поле rtx у командах addi та lw містять номер регістра-призначення, в sw – номер регістра-джерела.

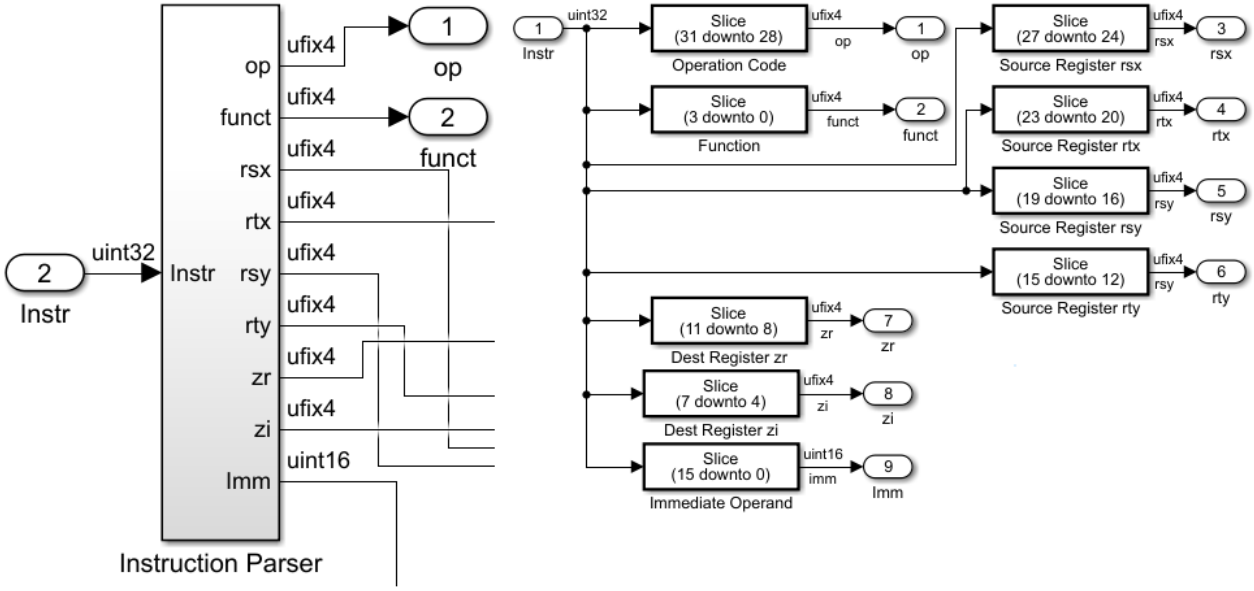


Рис 2.9 – аналізатор інструкцій у середовищі Simulink

Таблиця 2.3 – Поля інструкції I-типу

Opcode	RSX	RTX	Imm
по 4 біти кожне поле			16 бітна константа

Приклад зображений нижче демонструє використання команд I-типу які записані в ПЗП у шестадцятирічній системі. Команди цього типу визначаються лише полем Opcode, тобто першими чотирма бітами. В першій інструкції вказано що команда виконує занесення операнду, тобто додавання його до регістра вказаному в полі RTX в данній команді це другий регістр. У другій інструкції виконується занесення операнду в регістр вказаному в полі RTX тобто третій регістр, значення якого обчислюється додаванням до регістру вказаному в полі RSX, тобто до другого регістру значення вказаного в полі Imm.

Інструкція SW записує значення регістру вказаного в полі RTX в ОЗП по адресу який обчислюється шляхом додавання базової адреси вказаної в полі RSX поля Imm, значення регістру R3 дорівнює hex(d), тобто адреса запису розраховується наступним чином: d+a і дорівнює hex(17), тобто записується в

комірку пам'яті hex(15-18).

Інструкція LW зчитує з ОЗП число яке знаходиться за адресою вказаною у полі RSX+Imm і записує у регістр вказаний за адресою в полі RTX.

RAM(1) = hex2dec('30200007'); % addi R2=7

RAM(2) = hex2dec('32300006'); % addi R3=R2+7

RAM(3) = hex2dec('2330000a'); % sw R7,R2+a запис в ОЗП у 6 комірку

RAM(4) = hex2dec('1380000a'); % lw читаємо з ОЗП за адресою R3+a та записуємо у R9

Інструкції Complex-type використовують 4 регістри в якості джерела: RSX, RTX, RSY, RTY та два регістри призначення: Zr, Zi, в таблиці 2.4 показано формат команди для цього типу. 32-бітна команда складається з восьми полів, розмір кожного поля дорівнює чотирьом бітам.

Таблиця 2.4 – поля інструкцій для команд з комплексними числами

Opcode	RSX	RST	RSY	RTY	ZR	ZI	Funct
розмір всіх полів 4 біти							

Операції над комплексними числами закодовані двома полями: opcode та funct, в усіх командах цих командах opcode дорівнює нулю. Операції цього типу контролюються виключно полем funct, для котрого таблиця істинності зображена в таблиці 2.2.

В ПЗП RAM(5) записана інструкція додавання комплексних чисел. В ній вказана дреса дійсної частини яка записана у полях RSX, RTX і уявною яка записана в полях RSY, RTY результати додавання дійсної частини записуються в регістр який вказаний в полі Zr, уявної в Zi.

Для інструкцій комплексного віднімання діють ті ж правила що і для додавання комплексних чисел , вони розрізняються лише полем Funct.

Приклад використання команд для комплексних чисел наведений нижче.

RAM(1) = hex2dec('30200007'); % addi R2=7

RAM(2) = hex2dec('32300006'); % addi R3=R2+6

RAM(3) = hex2dec('30400005'); % addi R4=5

```
RAM(4) = hex2dec('30600003');  % addi R6=3
RAM(5) = hex2dec('02346781');  % Zr + Zi = (x1+ x2,y1 + y2) complexadd R7 + R8 = (R2 + +R3,R4 + R6)
RAM(6) = hex2dec('30a0000a');  % addi Ra=a
RAM(7) = hex2dec('30b00005');  % addi Rb=5
RAM(8) = hex2dec('30c0000b');  % addi Rc=b
RAM(9) = hex2dec('30d00006');  % addi Rd=6
RAM(10) = hex2dec('0abcdef2'); % Zr-Zi=(x1 - x2,y1 - y2) complexsub Re - Rf=(Ra -Rb,Rc--Rd)
```

2.5.2 АЛП процесору та вибір точності обчислень.

АЛП представлений М-функцією яка зображена на рисунку 2.10 та має чотири 32-бітних вхідних порти, дворозрядний керуючий сигнал ALUControl та два 32-бітних результати ALUResult та ALUResultIm, в залежності від керуючого сигналу виконуються необхідні операції. В АЛП передбачено три операції: додавання, комплексне додавання та віднімання.

Операція додавання необхідна для команд роботи з пам’яттю (lw та sw), для цих операцій виконується обчислення адрес шляхом додавання до базової адреси необхідного зсуву, для цього на вхід SrcX1 подається базова адеса, а на SrcX2 зсув – на виході з АЛП виходить 32-бітовий результат ALUResult.

Операція додавання також необхідна для команди addi, ця команда додає до вказаного регістру безпосередній операнд.

Для команд комплекних дій необхідно виконувати дії над дійсною та уявною частинами. Інструкції Complexadd та Complexsub читають з регістрового файлу одразу чотири операнди, на SrcX1 та SrcX2 надходять дійсні частини комплекного числа, а на SrcY1 та SrcY2 – уявні. Після обчислень результат обчислень над дійсною частиною виходить з ALUResult, над уявною з ALUResultIm.

Через те, що АЛП задіян при обчисленні адрес, виконувати дії над дробами стає неможливо – необхідно додавати окремий АЛП для адрес. Було вирішено не ускладнювати пристрій та виконувати дії над цілими комплексними

числами.

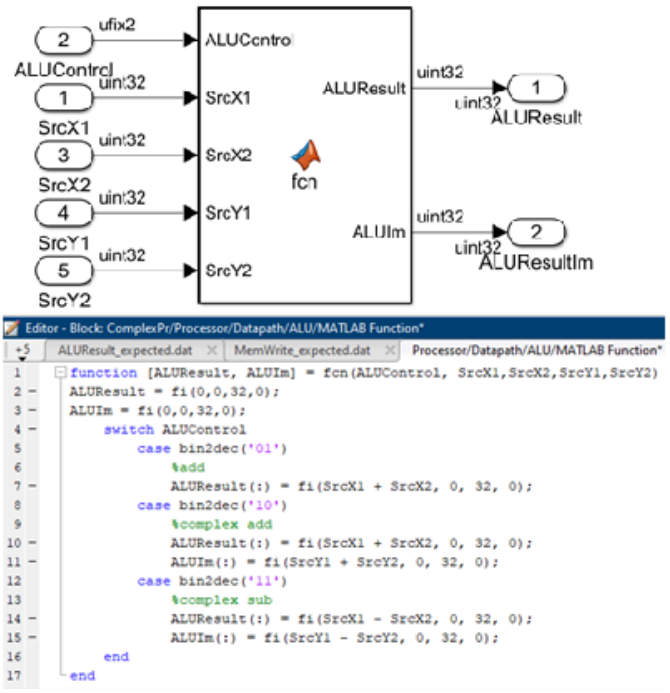


Рисунок 2.10 – М-функція АЛП

2.5.3 Керуючі сигнали

Всі керуючі сигнали, крім MemWrite, який керує запис в ОЗП, надходять в тракт даних по восьмирозрядній шині Control, яка зображена на рисунку 2.11.

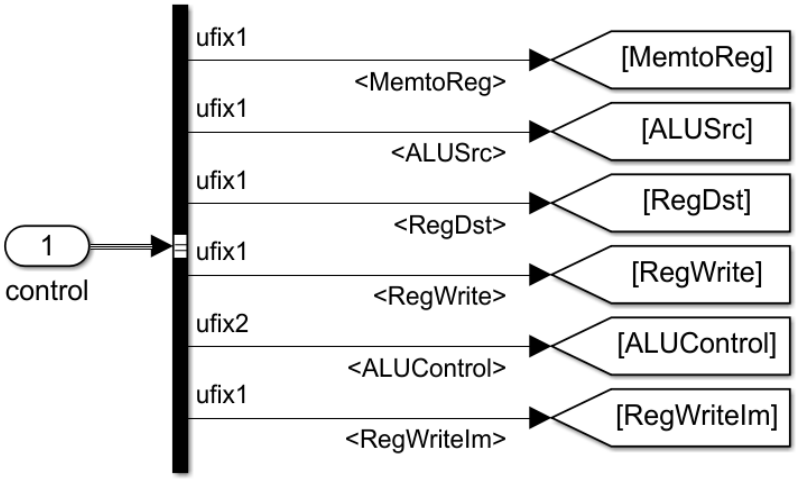


Рисунок 2.11 – Керуючі сигнали, які надходять в тракт даних

Сигнал MemtoReg – подає сигнал на мультиплексор, який зображений на рисунку 2.12, він керує вихідними портоми АЛП – ALUResult та ReadData (читання з ОЗП в регістровий файл). Для команд Complex-type потрібно

записувати в регістровий файл значення ALUResult.

MemtoReg дорівнює нулю для інструкцій типу Complex-type – у цьому випадку вихідний порт мультиплексора Result приймає значення ALUResult. Для команди lw сигнал MemtoReg дорівнює одиниці, а Result приймає значення ReadData. Для інструкції Sw значення MemtoReg не відіграє ніякої ролі, тому що Sw нічого в регістровому файлі не пише.

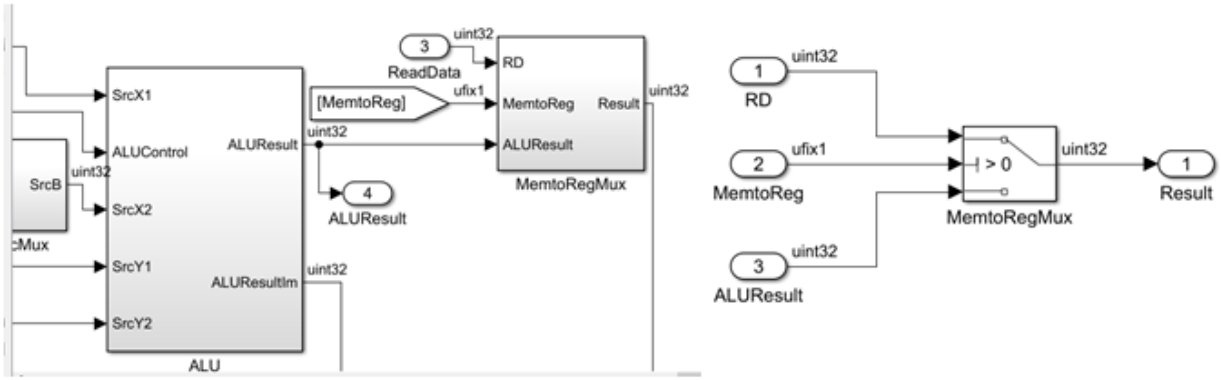


Рисунок 2.12 – Керуючий сигнал MemtoReg

Сигнал ALUSrc (від англ. source – джерело) – подає сигнал на мультиплексор, який зображений на рисунку 2.13 , він керує вхідним портом АЛП – SrcX2, на який може надходити SignImm або вихід RD2 регістрового файлу. ALUSrc дорівнює нулю для інструкцій Complex-type – в цьому випадку на вхід АЛУ подається значення із регістрового файлу. Для інструкцій lw та sw ALUSrc дорівнює одиниці, і на вхід АЛУ подається SignImm.

SignImm – це операція знакового розширення, представлена М-функцією яка зображена на рисунку 2.14. Інструкція Lw також потребує зміщення (offset) – число, яке буде додано до базової адреси. Зміщення передається як безпосередній операнд (immediate), тобто перебуває безпосередньо в полі Instr15:0, займаючи молодші 16 біт. Так як 16-бітове число може бути як позитивним, так і негативним, то над ним має бути виконана операція знакового розширення до 32 біт. Отримане розширене 32-бітове значення називається SignImm. Знакове розширення полягає в тому, що знаковий біт (він же старший біт) розширюваного числа просто копіюється у всі старші біти розширеного числа, а саме SignImm15:0 = Instr15:0 та SignImm31:16 = Instr15.

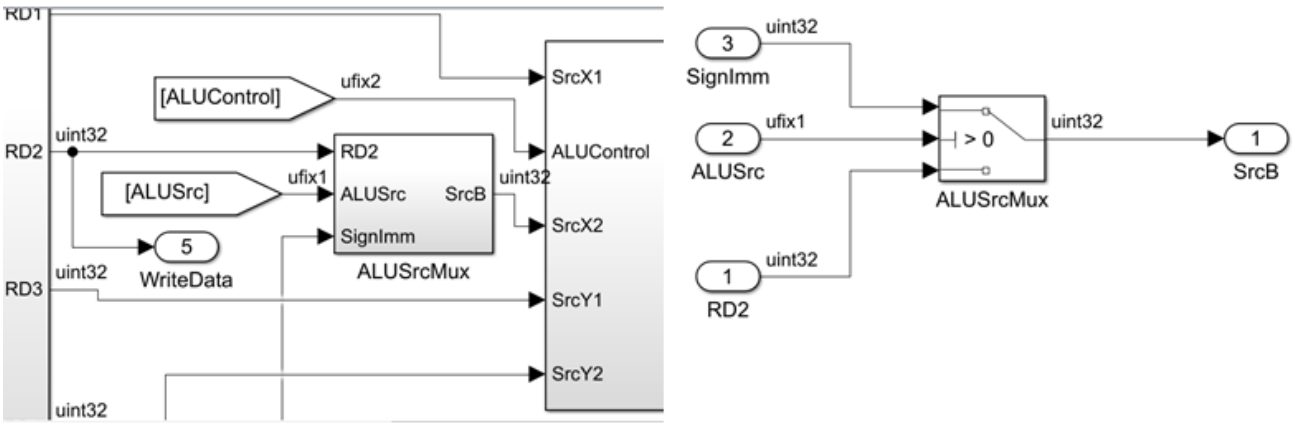


Рисунок 2.13 – Керуючий сигнал ALUSrc

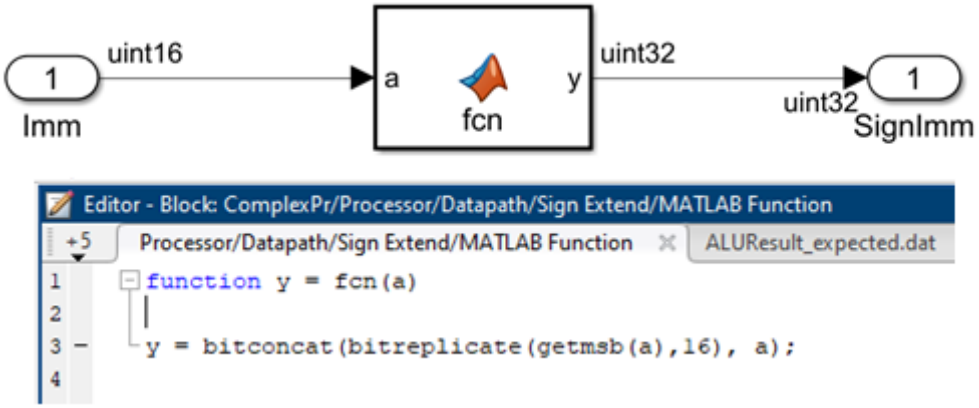


Рисунок 2.14 – Операція розширення знаку

Сигнал RegDst (від англ. destination – призначення) – подає сигнал управління на мультиплексор RegDstMux, який зображении на рисуну 2.15. RegDst дорівнює одиниці для інструкцій типу Complex-type – у цьому випадку вихід приймає значення поля Zr (Instr11:8). Для команди Lw сигнал RegDst дорівнює нулю, а вихід набуває значення поля Rtx (Instr23:20). Для інструкції Sw значення RegDst не відіграє жодного ролі, тому що Sw нічого в регістровий файл не пише.

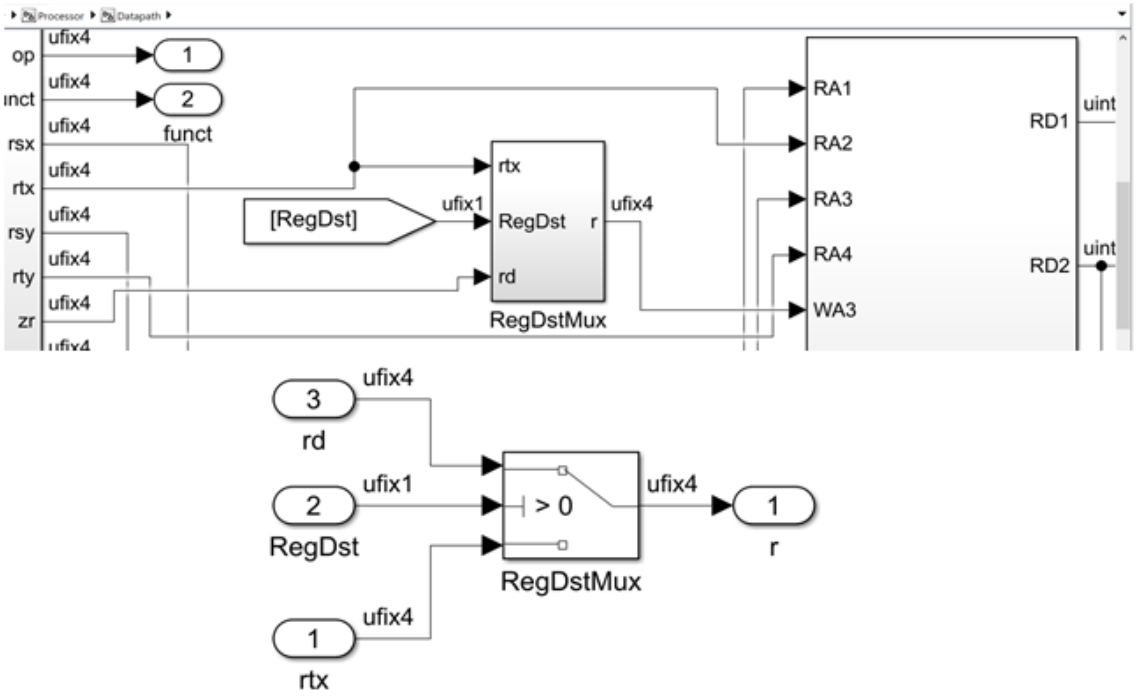


Рисунок 2.15 – Керуючий сигнал RegDst

3. МОДЕЛЮВАННЯ ПРОЦЕСОРУ В СИСТЕМІ MATLAB/SIMULINK

Під час реалізації процесору в Simulink була необхідність перевіряти, як працює та чи інша команда, як надходить той чи інший сигнал. Для цього в ROM пам'ять записувались тестуючі програми, маркувались необхідні лінії зв’язку компонентів та виконувалось моделювання за допомогою інструменту «Logic Analyzer». Коли робота процесора була приближена до «еталонного зразку», була написана тестувальна програма. Для того, щоб протестувати команди комплексних дій необхідно занести в регістровий файл чотири операнди за допомогою інструкції «addi», потім необхідно виконати операції комплекного додавання та вінімання над завантаженими числами. Потім буде протестована команда завантаження в ОЗП та завантаження з ОЗП в регістровий файл.

Модельний час в Simulink обрано 20 секунд, один такт процесору умовно дорівнєю одній секунді. Треба розуміти, що модельний час, який вказується при моделюванні в Logic Analyzer, не відображає реальної роботи процесора, тобто ми не можемо побачити частостні характеристики, які нас інтересують при тестуванні реального процесору. Ці релузьтати не можуть бути взяті як істені. Simulink – це високорівневий пакет для проектування. Хоча є можливість підключати ПЛІС до Matlab через технологію «FPGA in the Loop»(FIL), але ця технологія у данній роботі не використовувалась.

FIL забезпечує канал зв'язку для надсилання та отримання даних між Simulink та FPGA. Цей канал може бути JTAG, Ethernet або PCI Express. Зв'язок між Simulink та FPGA суворо синхронізується, щоб забезпечити надійне середовище верифікації.

Тестова програма процесору:

```
RAM(1) = hex2dec('30200007');   % addi R2=7
RAM(2) = hex2dec('32300006');   % addi R3=R2+6
RAM(3) = hex2dec('30400005');   % addi R4=5
```

```
RAM(4) = hex2dec('30600003');   % addi R6=3
RAM(5) = hex2dec('02346781');   % Zr+Zi=(x1+x2,y1+y2) complexadd R7+R8=(R2+R3,R4+R6)
RAM(6) = hex2dec('30a0000a');   % addi Ra=a
RAM(7) = hex2dec('30b00005');   % addi Rb=5
RAM(8) = hex2dec('30c0000b');   % addi Rc=b
RAM(9) = hex2dec('30d00006');   % addi Rd=6
RAM(10) = hex2dec('0abcdef2');  % Zr-Zi=(x1-x2,y1-y2) complexsub Re-Rf=(Ra-Rb,Rc-Rd)
RAM(11) = hex2dec('30000028');  % addi R0=28
RAM(12) = hex2dec('20700008');  % sw R7 R0+8 зсув на hex(28+8)=40+8=48dec     48/4=12
RAM(13) = hex2dec('2080000c');  % sw R8 R0+c          hex(28+c)=40+12=52dec     52/4=13
RAM(14) = hex2dec('1090000c');  % lw читаємо з ОЗП за адресом R0+c  40+12=52 та
записуємо в R9
```

З вище зазначеної тестової програми можна зробити висновок що процесор повинен здійснити 14 тактів для виконання цієї програми, на кожному такті подається інструкція та збільшується значення PC. Вікно додатку Data Inspector, яке демонструє виконання 14 тактів зображено на рисунку 3.1.

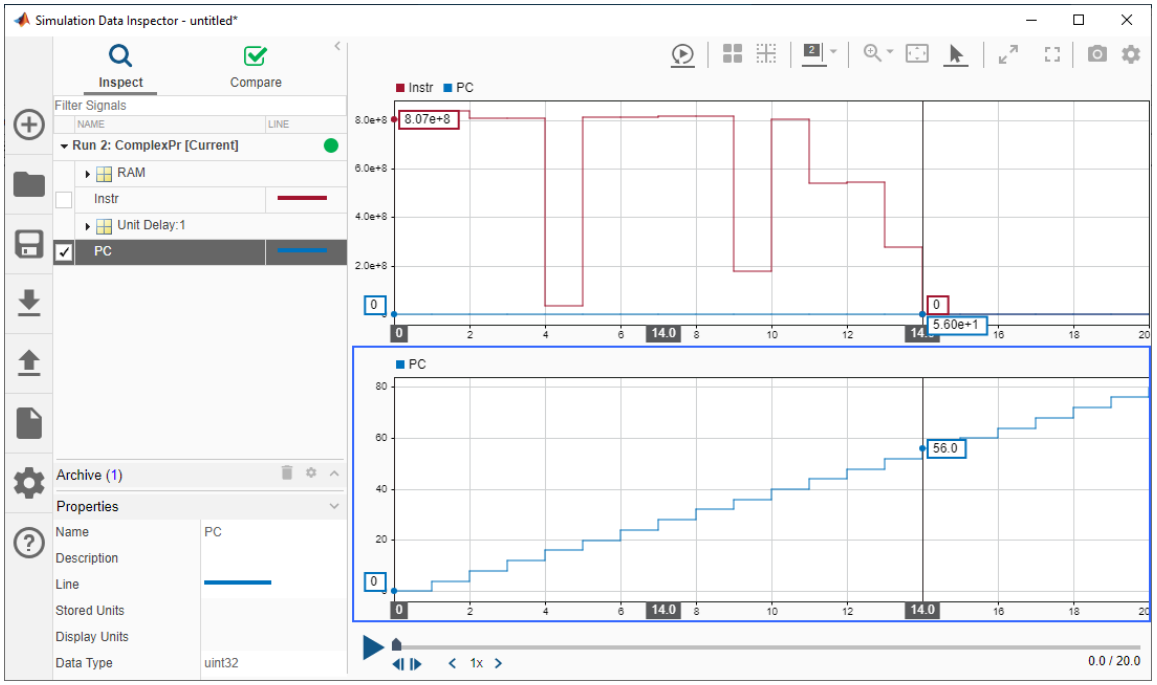


Рисунок 3.1 – Інструкції та значення PC в інструменті Data Inspector

Для виконання перший чотирьох операцій занесення операндів в регістровий файл, які знаходяться нижче, процесор повинен здійснити чотири такти, четвертий занесений операнд повинен відображатись на п’ятому такті

процесору. В першій інструкції вказано, що команда виконує занесення операнду, тобто додавання його до регістра вказаному в полі RTX в данній команді це другий регістр. У другій інструкції виконується занесення операнду в регістр вказаному в полі RTX, тобто третій регістр, значення якого обчислюється додаванням до регістру вказаному в полі RSX, тобто до другого регістру значення вказаного в полі Imm. Третя та четверта – подібні першій інструкції.

На рисунку 3.2 зображено вміст регістрового файлу після 4 такту.

```
RAM(1) = hex2dec('30200007');   % addi R2=7
RAM(2) = hex2dec('32300006');   % addi R3=R2+6
RAM(3) = hex2dec('30400005');   % addi R4=5
RAM(4) = hex2dec('30600003');   % addi R6=3
```

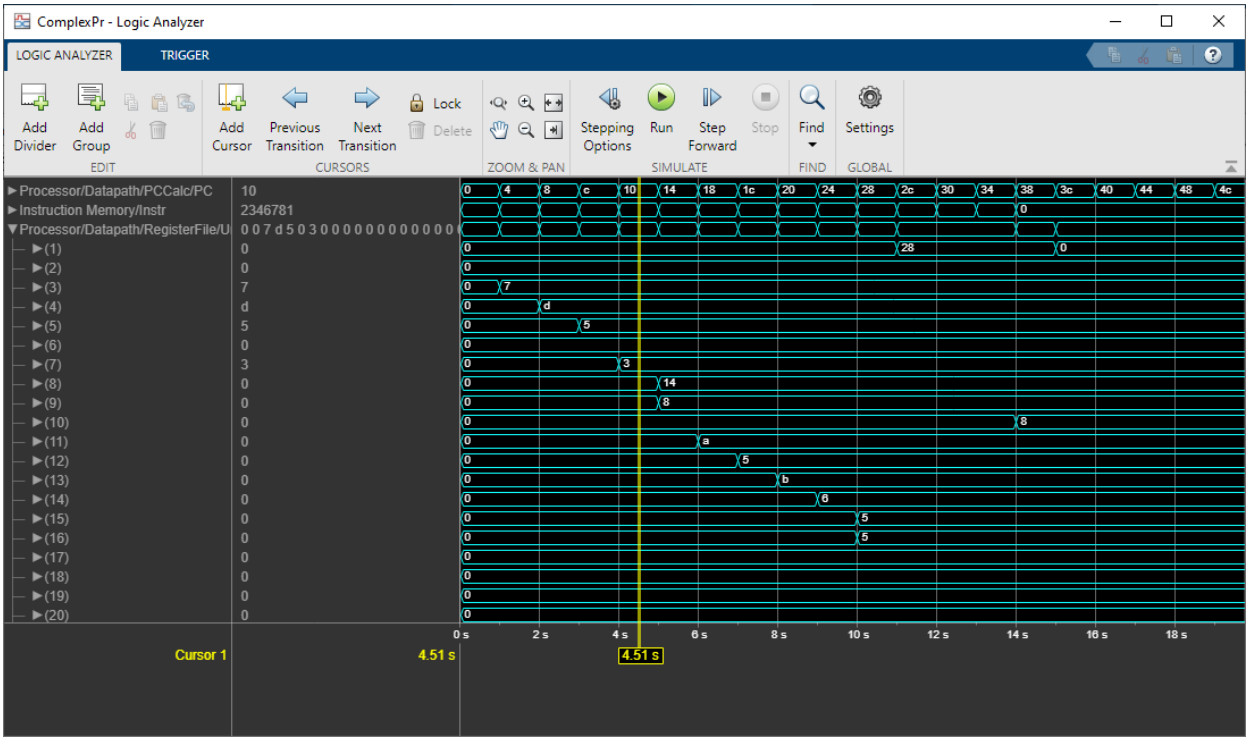


Рисунок 3.2 – Занесення перших чотирьох операндів

Після занесення виконується інструкція комплексного додавання над регістрами R2, R3, R4, R6 та занесення результатів в регістри R7 та R8, результати виконання комплексного додавання созбражені на рисунку 3.3, всі числа представлені в шістнадцятирічній системі.

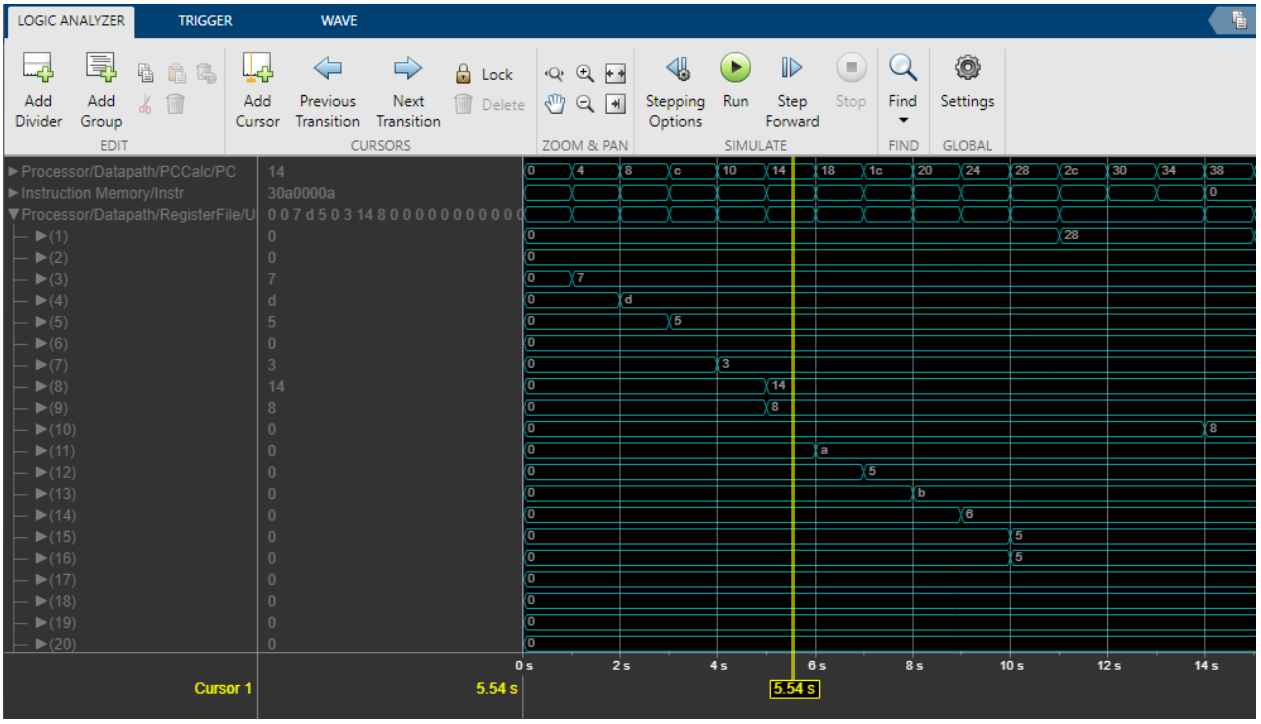


Рисунок 3.3 – Результати комплексного додавання

Потім заноситься друга пара комплексних чисел в регістри, які будуть використовуватись при комплексному відніманні. На рисунку 3.4 зображено занесення другої пари числ в регістри за адресами Ra. Rb, Rc, Rd. Після виконується операція комплексного віднімання над цими регістрами та запису результатів у Re, Rf результат цієї інструкції зображені на рисунку 3.5.

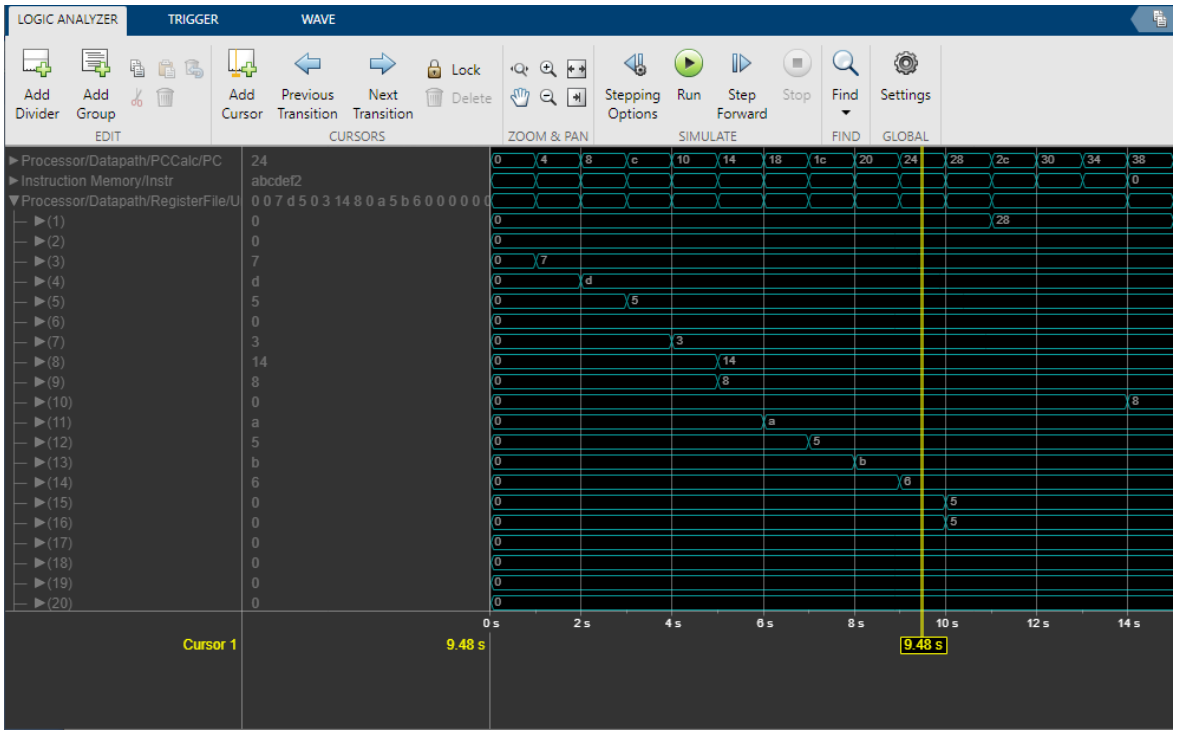


Рисунок 3.4 – Занесення другої пари чисел

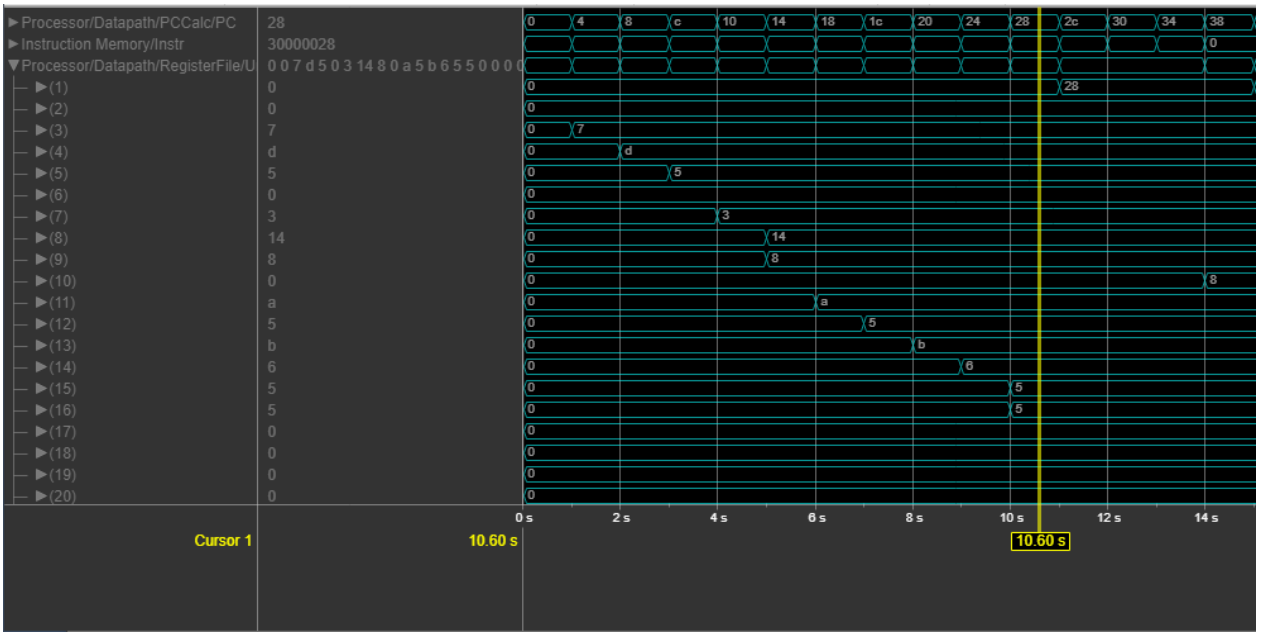


Рисунок 3.5 – Результати віднімання комплексних чисел

Після виконання основних операцій було записано число hex(28) в R0, він буде використовуватись як базовий адрес. І виконуються інструкції запису в ОЗП з R7 та R8 за адресами hex(R0+8) та hex(R0+c) відповідно, дивись рисунок 3.6.

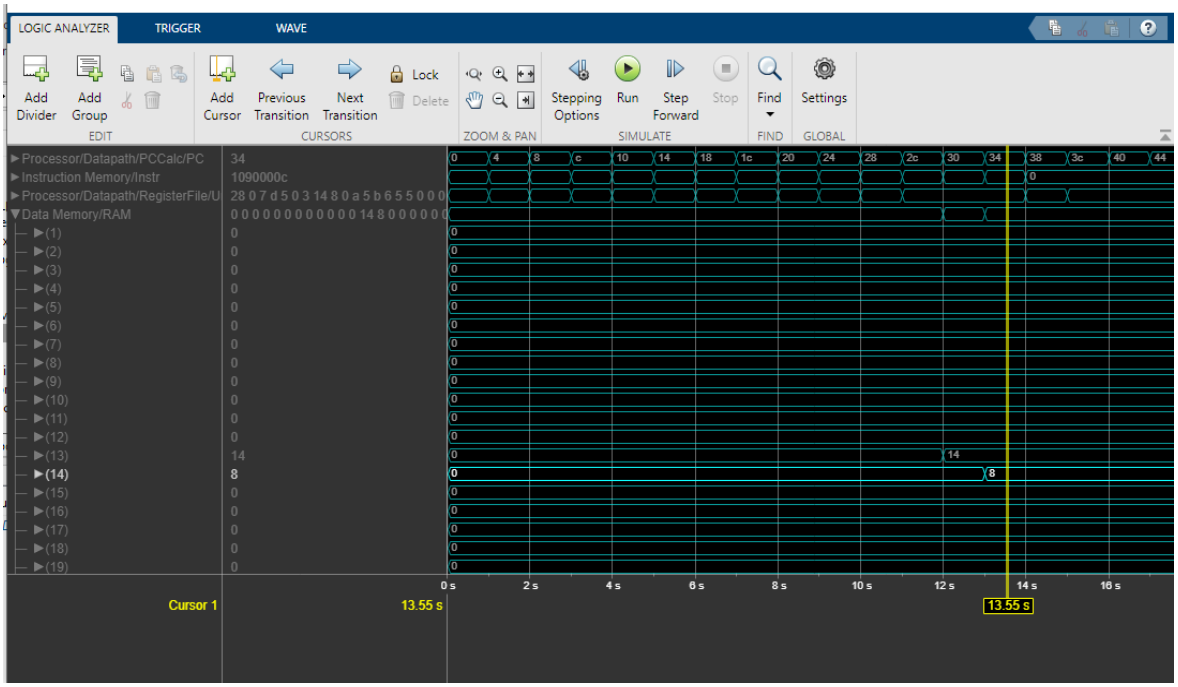


Рисунок 3.6 – Занесення результатів додавання в ОЗП

Потім виконується інструкція запису в R9 вмісту комірки ОЗП за адресою

R0+c , результати виконання інструкції зображені на рисунку 3.7.

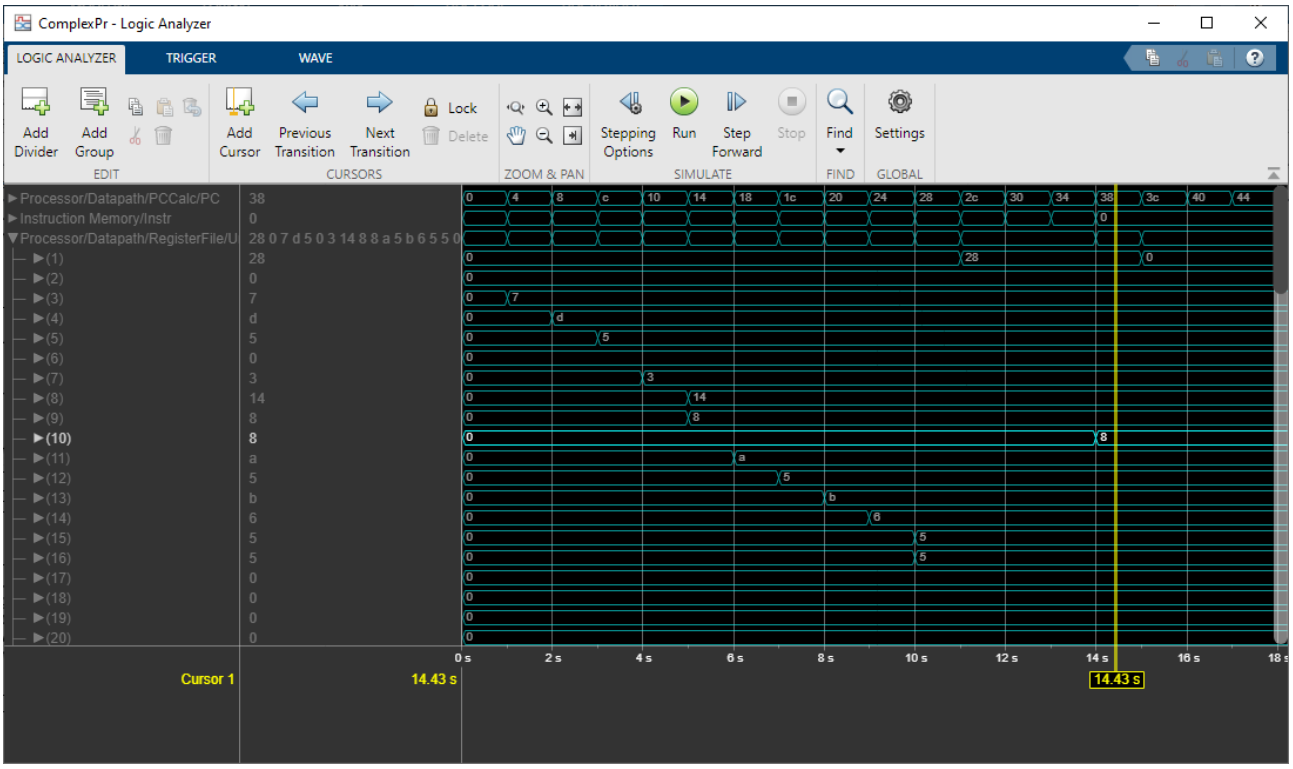


Рисунок 3.7 – Запис в регістр з ОЗП

4. ГЕНЕРАЦІЯ VHDL КОДУ ТА TESTBENCH ЗА ДОПОМОГОЮ MATLAB/SIMULINK

4.1 Робота з HDL Coder.

Після того, як я впевнився, що процесор працює коректно можна приступити до генерації VHDL коду. Для цього потрібно в меню налаштування обрати необхідну мову опису апаратури VHDL або Verilog та обрати необхідну модель ПЛІС. HDL Coder може автоматично здійснювати оптимізацію коду, яку потребує проект, для досягнення необхідної тактової частоти.

Після компіляції VHDL коду для процесору відкривається звіт в якому ми можемо побачити налаштування які були обрані дивись рисунок 4.1, детальний звіт по кількості необхідних ресурсів зображений на рисунку 4.2 та сторінка на якій вказані вхідні та вихідні порти пристрою дивись рисунок 4.3.

VHDL код процесору наведений у додатку А.

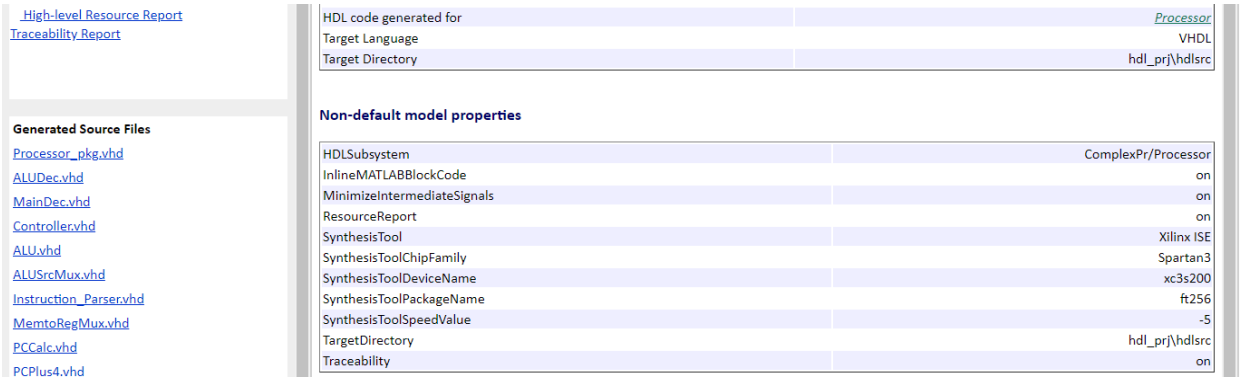


Рисунок 4.1 – Обрані налаштування HDL Coder

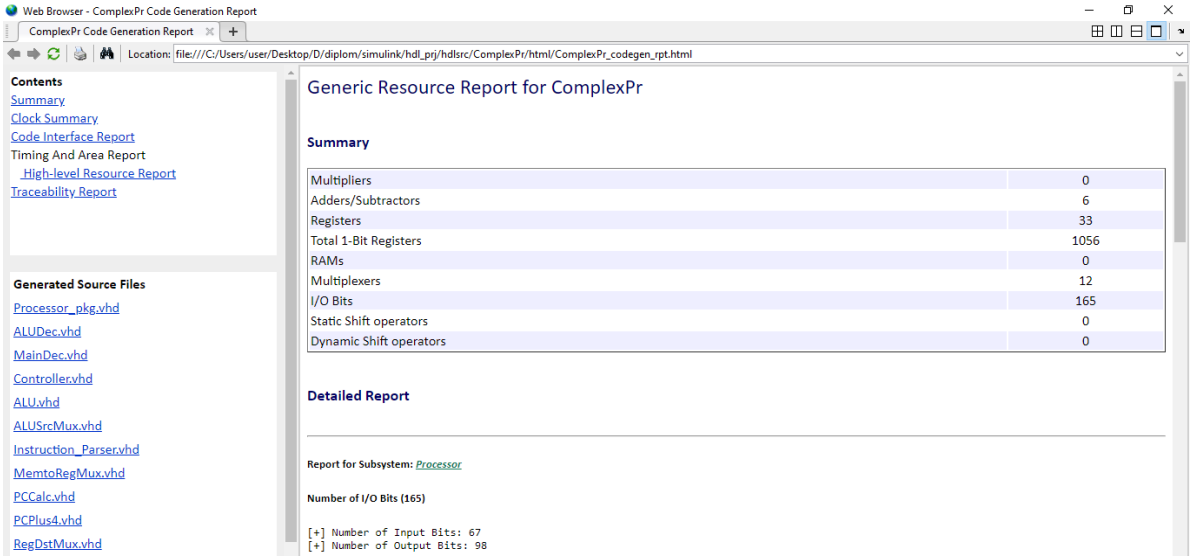


Рисунок 4.2 – Звіт по необхідним ресурсам ПЛІС

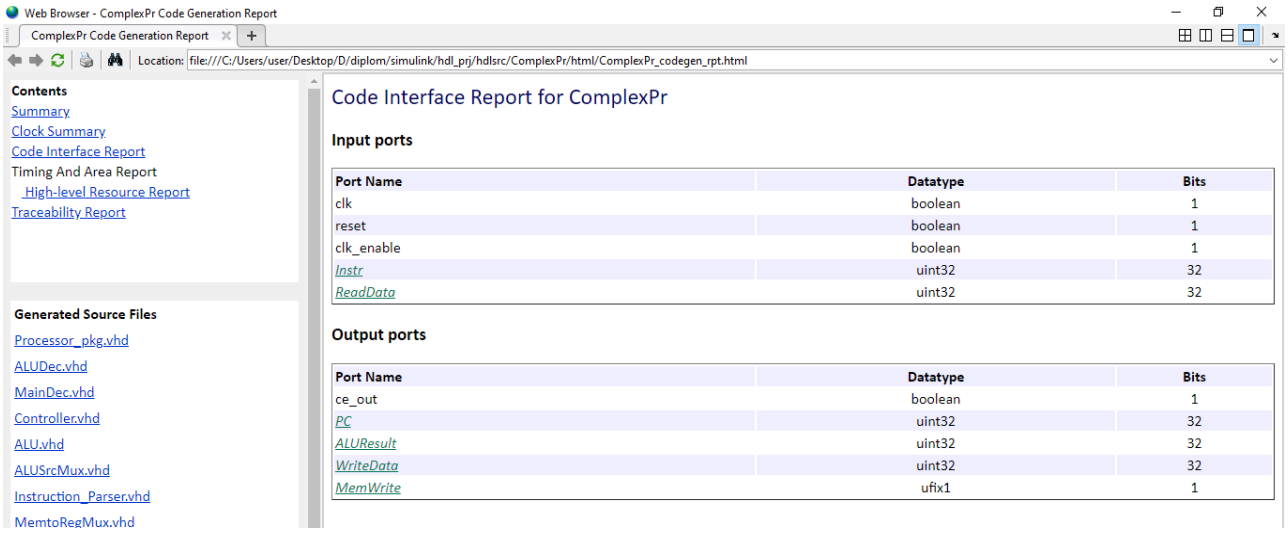


Рисунок 4.3 – Вхідні та вихідні порти пристрою

У лівій частині звіту знаходяться файли з розширенням .vhd, для кожного модуля пристрою. Згенерований код достовірно відображає розроблений в Matlab/Simuink пристрій і, як правило, не потребує значного редагування. Приклад фрагменту звіту HDL Coder для верхнього рівня процесору зображений на рисунку 4.4.

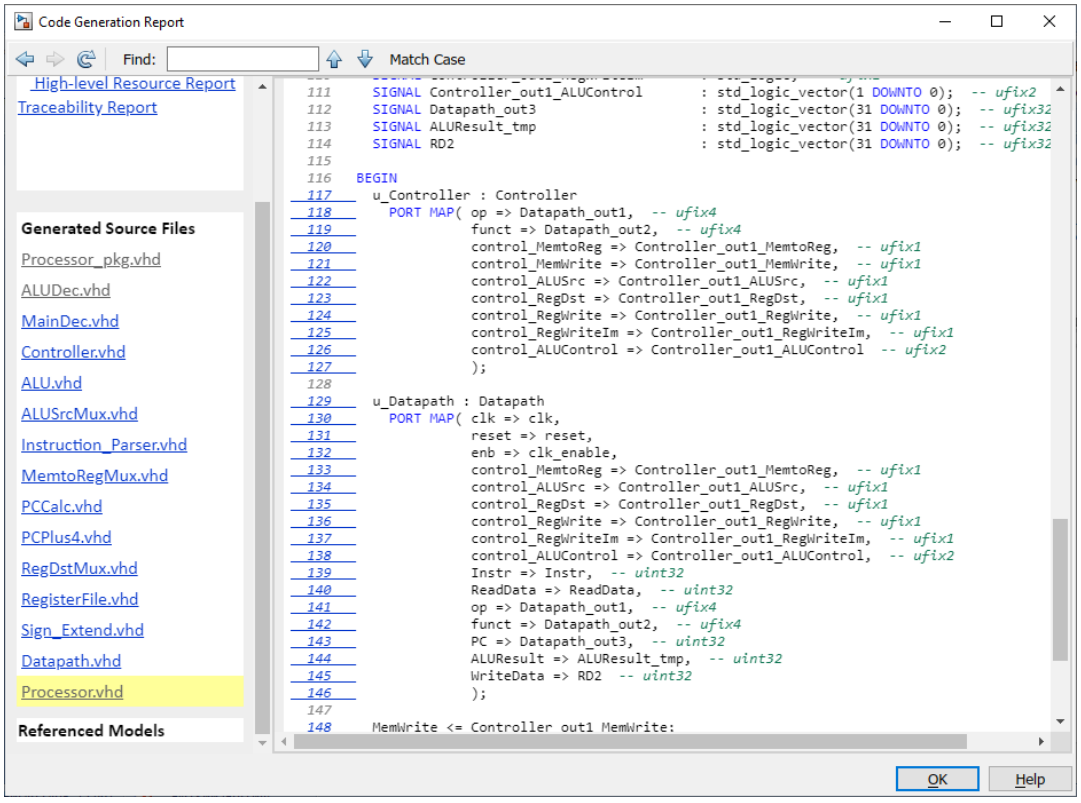


Рисунок 4.4 – Приклад фрагменту звіту HDL Coder для процесору

Для процесору згенерувалось 15 файлів VHDL-коду. Основним є файл з

назвою блоку процесору «processor». Він згенерований у структурному стилі, в ньому описані зв’язки між контролером та трактом даних, а також порти. Контролер також описан за допогою структурного стилю, декодери які в ньому знаходяться, описують логіку всього пристрою за допогою конструкції case в паралельному операторі Process. Нижче представлений код для дешифратора АЛП.

```
MATLAB_Function_output : PROCESS (ALUOp_unsigned, funct_unsigned)
 BEGIN
  CASE ALUOp_unsigned IS
   WHEN "00" =>
    ALUControl_tmp <= to_unsigned(16#1#, 2);
   WHEN OTHERS =>
    CASE funct_unsigned IS
     WHEN "1000" =>
      ALUControl_tmp <= to_unsigned(16#1#, 2);
     WHEN "0001" =>
      --complexadd
      ALUControl_tmp <= to_unsigned(16#2#, 2);
     WHEN "0010" =>
      --complexsub
      ALUControl_tmp <= to_unsigned(16#3#, 2);
     WHEN OTHERS =>
      ALUControl_tmp <= to_unsigned(16#0#, 2);
    END CASE;
  END CASE;
 END PROCESS MATLAB_Function_output;
```

В тракті даних описується всі зв'язоки внутрішніх компонентів процесору. Всі компоненти тракту даних мають блоки опису Entity, в яких описуються порти пристрою, і блоки Arhitecture, де декларуються їх сигнали і наведені паралельні оператори. АЛП реалізован паралельним оператором Process, в якому аналізується керуючий сигнал ALUControl за допомогою case. В регістровоу файлі аналізуються сигнали дозволу запису WE, синхросигнал Clk та сигнал сбросу Reset. Нижче продемонстрована частина цього VHDL-коду,

створеного поведінковим стилем.

```
Write_output : PROCESS (Unit_Delay_out1, WA3_unsigned, WA4_unsigned,
WD3_unsigned, WD4_unsigned, WE3,WE4)
  BEGIN
    Write_out1 <= Unit_Delay_out1;
    IF WE3 /= '0' THEN
      Write_out1(to_integer(WA3_unsigned)) <= WD3_unsigned;
    END IF;
    IF WE4 /= '0' THEN
      Write_out1(to_integer(WA4_unsigned)) <= WD4_unsigned;
    END IF;
  END PROCESS Write_output;
  Unit_Delay_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      Unit_Delay_out1 <= (OTHERS => to_unsigned(0, 32));
    ELSIF clk'EVENT AND clk = '1' THEN
      IF enb = '1' THEN
        Unit_Delay_out1 <= Write_out1;
      END IF;
    END IF;
  END PROCESS Unit_Delay_process;
```

Після генерації коду є можливість навігації, тобто, якщо потрібно побачити як той чи інший блок реалізован мовою VHDL, потрібно його вибрати та обрати пункт меню Navigate to Code (дивись рисунки 4.5 та 4.6).

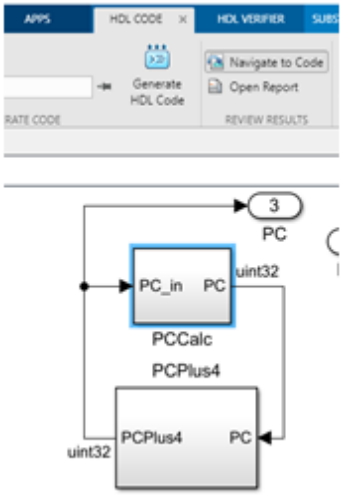
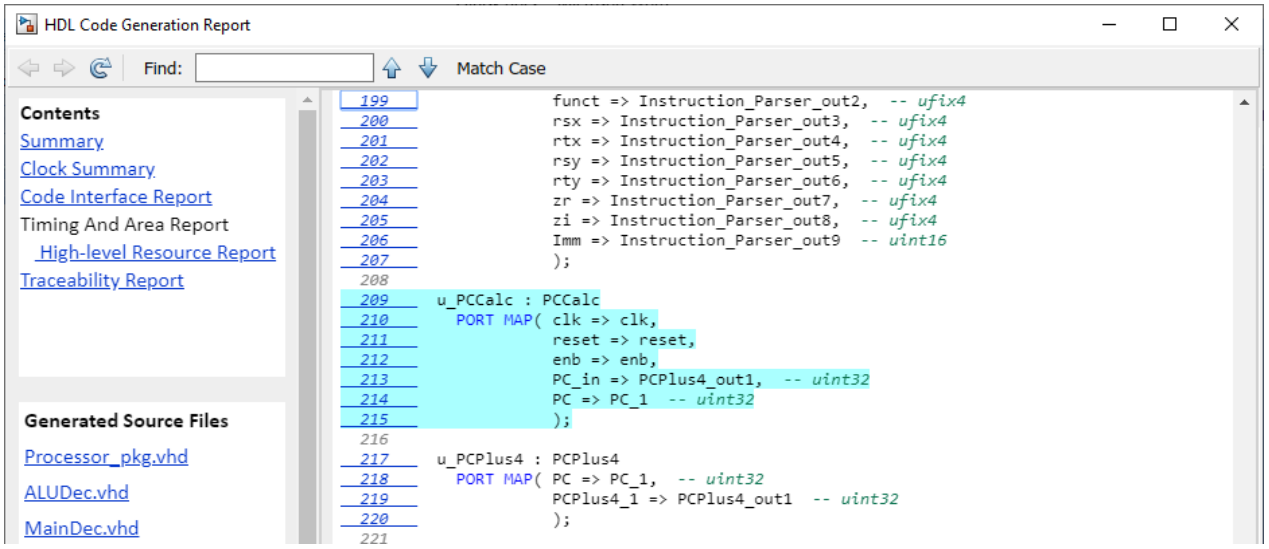
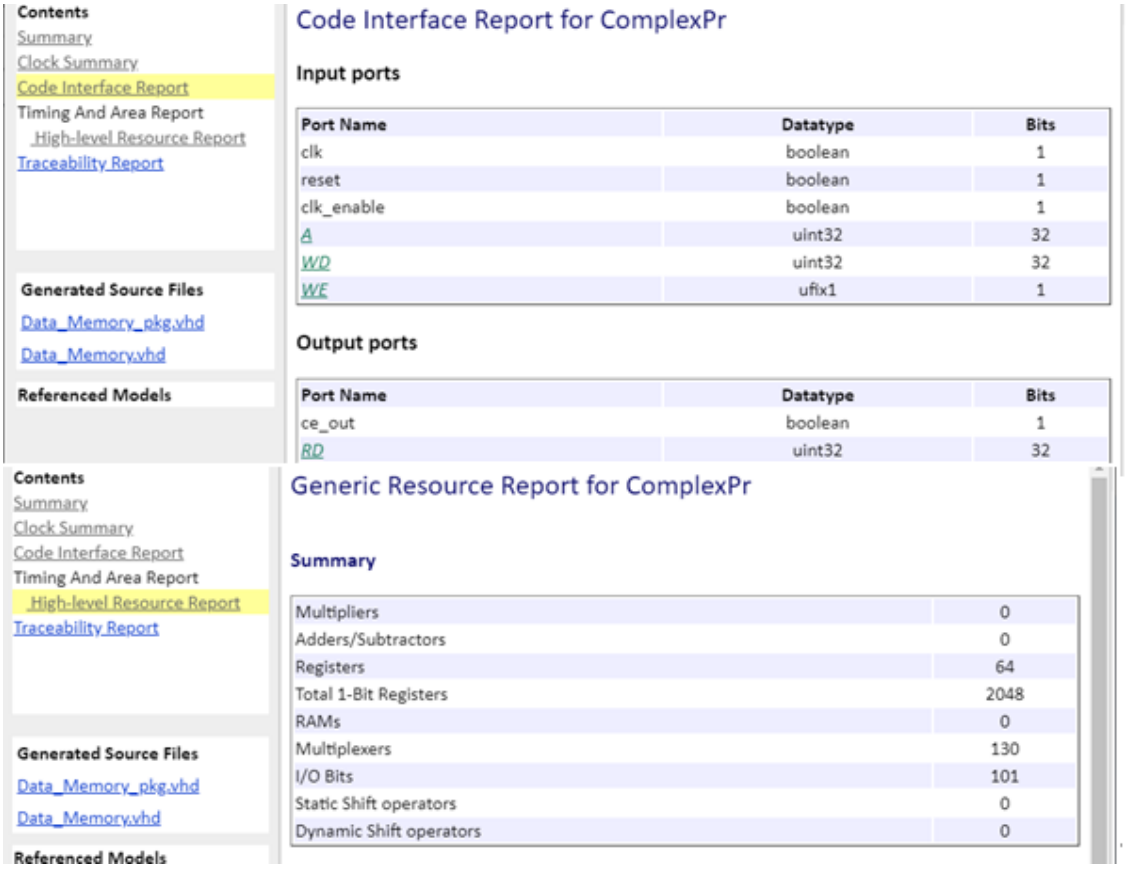


Рисунок 4.5 – Вибір пункту Navigate to Code



Рисуно 4.6 – Навігація по згенерованому коду

Після генерування коду для процесору, потрібно згенерувати VHDL код для ОЗП і ПЗП. Частина звіту зображена на рисунках 4.7 та 4.8. VHDL код для блоків пам’яті, розроблюваної системи, наведений у додатку Б.



Contents
Summary
Clock Summary
Code Interface Report
Timing And Area Report
High-level Resource Report
Traceability Report

Generated Source Files
Data_Memory_pkg.vhd
Data_Memory.vhd

Referenced Models

Code Interface Report for ComplexPr

Input ports

Port Name	Datatype	Bits
clk	boolean	1
reset	boolean	1
clk_enable	boolean	1
A	uint32	32
WD	uint32	32
WE	ufix1	1

Output ports

Port Name	Datatype	Bits
ce_out	boolean	1
RD	uint32	32

Contents
Summary
Clock Summary
Code Interface Report
Timing And Area Report
High-level Resource Report
Traceability Report

Generated Source Files
Data_Memory_pkg.vhd
Data_Memory.vhd

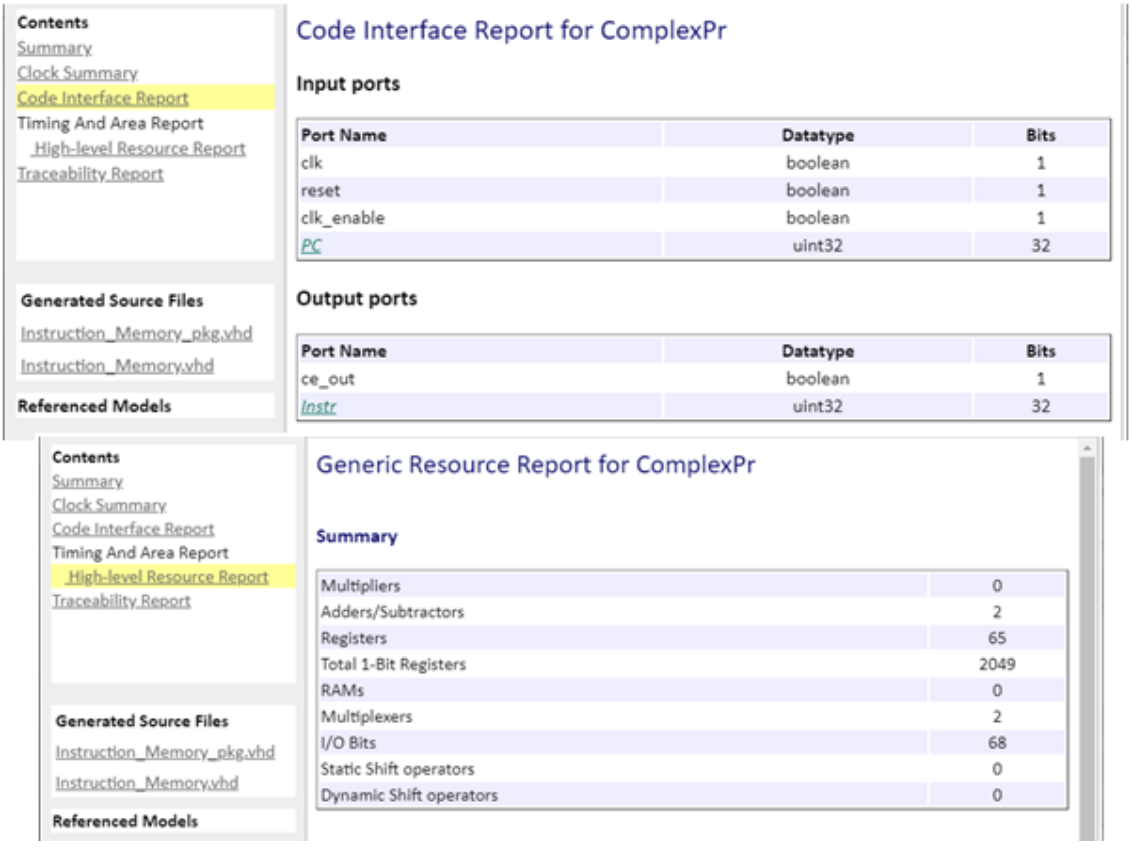
Referenced Models

Generic Resource Report for ComplexPr

Summary

Multipliers	0
Adders/Subtractors	0
Registers	64
Total 1-Bit Registers	2048
RAMs	0
Multiplexers	130
I/O Bits	101
Static Shift operators	0
Dynamic Shift operators	0

Рисунок 4.7 – Звіт для ОЗП



Contents
Summary
Clock Summary
Code Interface Report
Timing And Area Report
High-level Resource Report
Traceability Report

Generated Source Files
Instruction_Memory_pkg.vhd
Instruction_Memory.vhd

Referenced Models

Code Interface Report for ComplexPr

Input ports

Port Name	Datatype	Bits
clk	boolean	1
reset	boolean	1
clk_enable	boolean	1
PC	uint32	32

Output ports

Port Name	Datatype	Bits
ce_out	boolean	1
Instr	uint32	32

Contents
Summary
Clock Summary
Code Interface Report
Timing And Area Report
High-level Resource Report
Traceability Report

Generated Source Files
Instruction_Memory_pkg.vhd
Instruction_Memory.vhd

Referenced Models

Generic Resource Report for ComplexPr

Summary

Multipliers	0
Adders/Subtractors	2
Registers	65
Total 1-Bit Registers	2049
RAMs	0
Multiplexers	2
I/O Bits	68
Static Shift operators	0
Dynamic Shift operators	0

Рисунок 4.8 – Звіт для ПЗП

4.2. Використання технології HDL Verifier

Після генерації в HDL Coder косимуляції – HDL Cosimulation автоматично генерується проект косимуляції зображений на рисунку 4.9 Зв'язок між Simulink та Questa Sim відбувається через спільну пам'ять. На рисунку 4.10 показано порти обміну між Matlab/Simulink та Qesta Sim. В кожний блок надходять результати симуляції з Simuink та Qesta Sim для порівняння значень, блоки порівняння результатів для значення ALUResult зображені на рисунку 4.11.

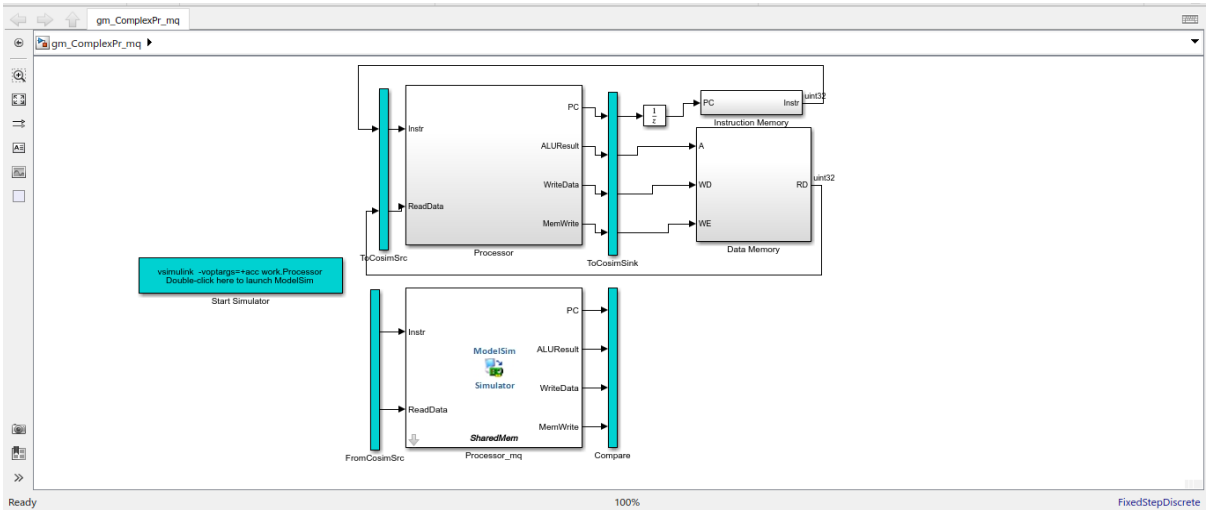


Рисунок 4.9 – Зв'язок між Simulink та Questa Sim

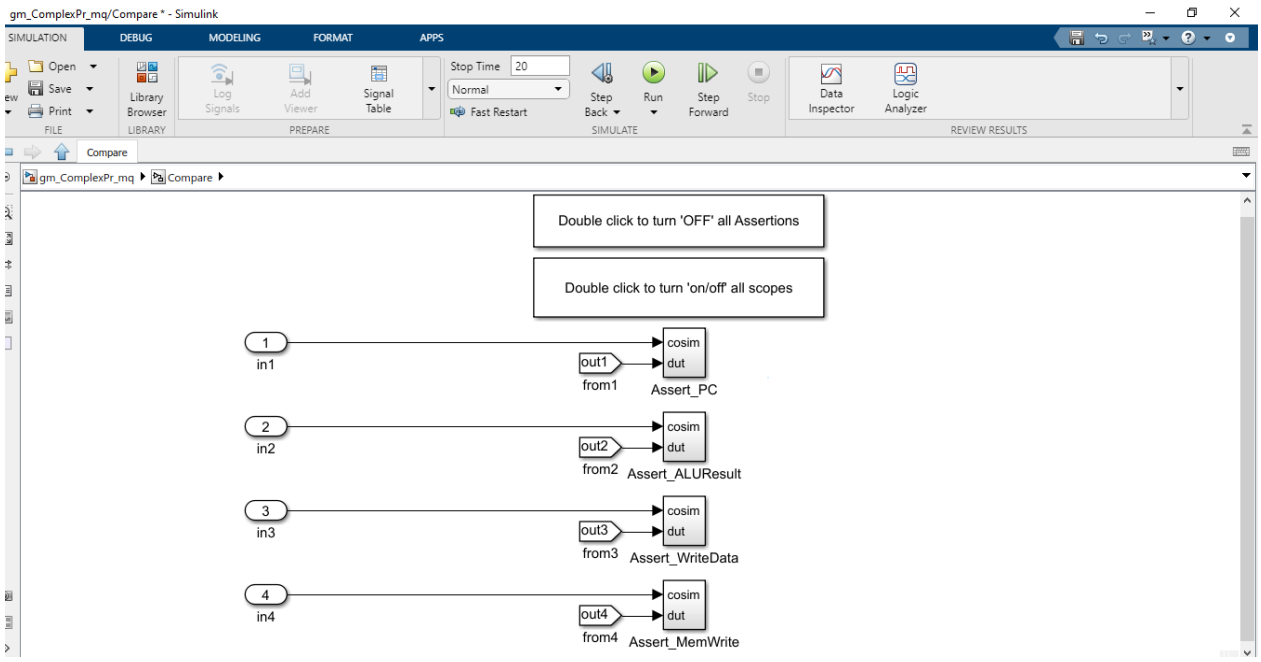


Рисунок 4.10 – Порти обміну між Matlab/Simulink та Qesta Sim

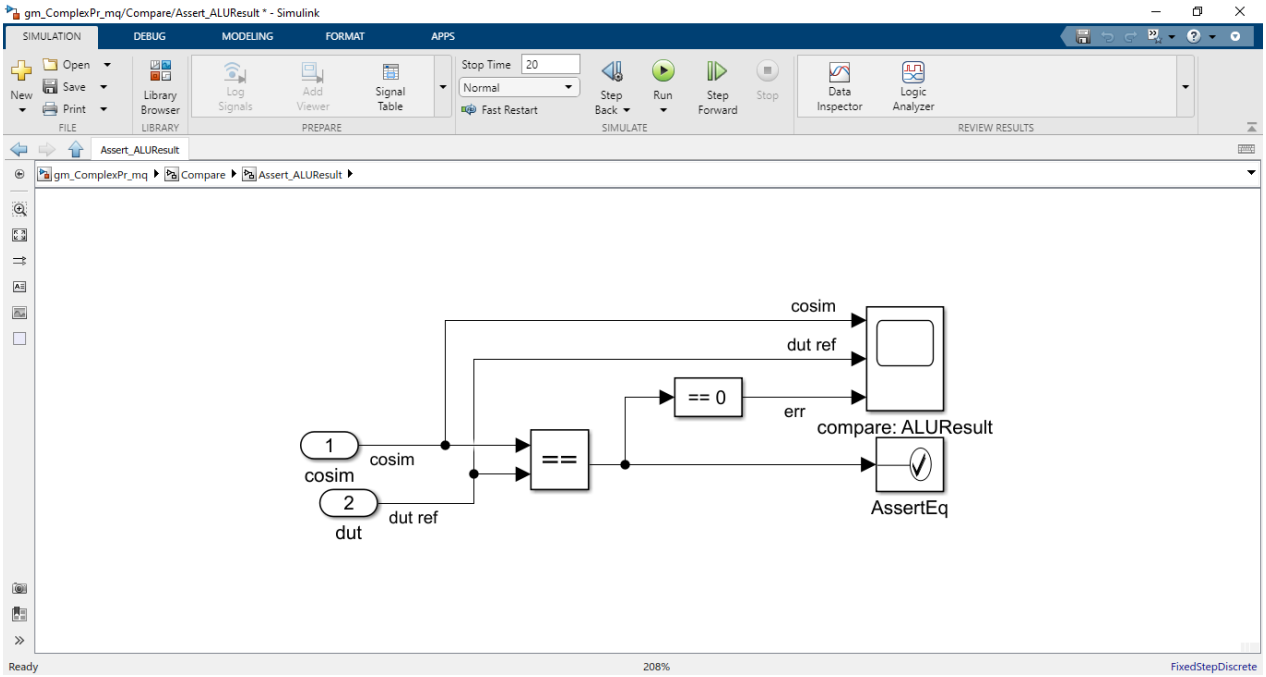


Рисунок 4.11 – Блоки порівняння результатів для значення ALUResult

Ми можемо встановити залежність між часов симуляції в Simuink та Questa Sim, вікно налаштувань зображено на рисунку 4.12, в данному прикладі виконується косимуляція з залежністю одна секунда в Simulink відповідає десяти наносекундам у Qesta Sim.

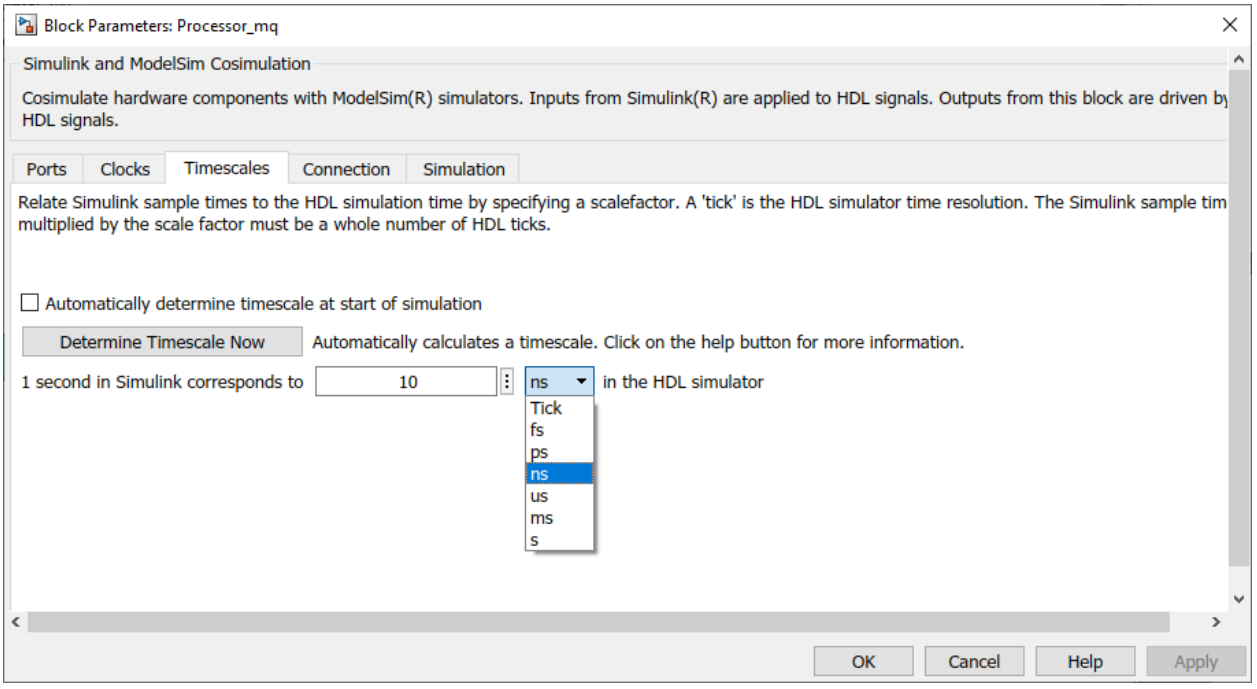


Рисунок 4.12 – Налаштування залежність часу симуляції

Спочатку потрібно запустити Qesta Sim за допомогою блоку Start Simulator, після чого завантажується САПР з автоматично створеним проектом який зображений на рисунку 4.13.

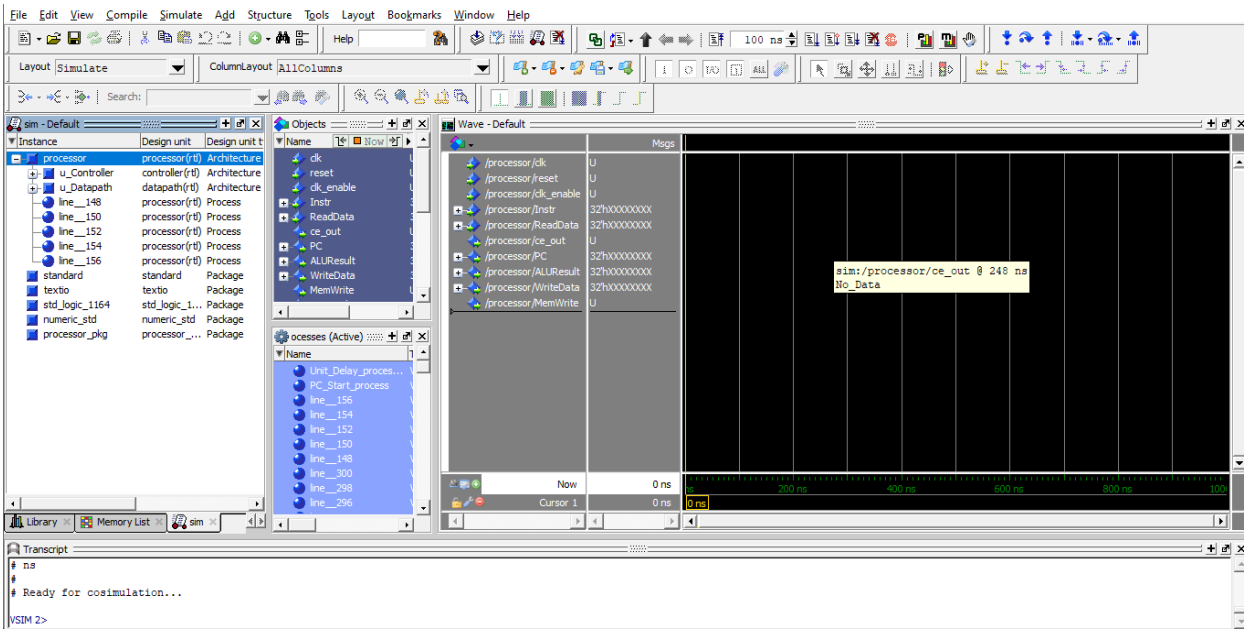


Рисунок 4.13 – Проект у Qusta Sim

Після запуску косимуляції можна більш детально подивитись як проходять перехідні процеси, подивитись за роботою сигналів CLK та RESET, які сигнали задіяни при виконанні той чи іншої команди в Qesta Sim дивись рисунок 4.14 та подивитись наявність помилок в результатах косимуляції, які відображаються на осцилографах. Результати помилок для значень ALUResult продемонстровані на рисунку 4.15, для MemWrite на рисунку 4.16 та для WriteData на рисунку 4.17.

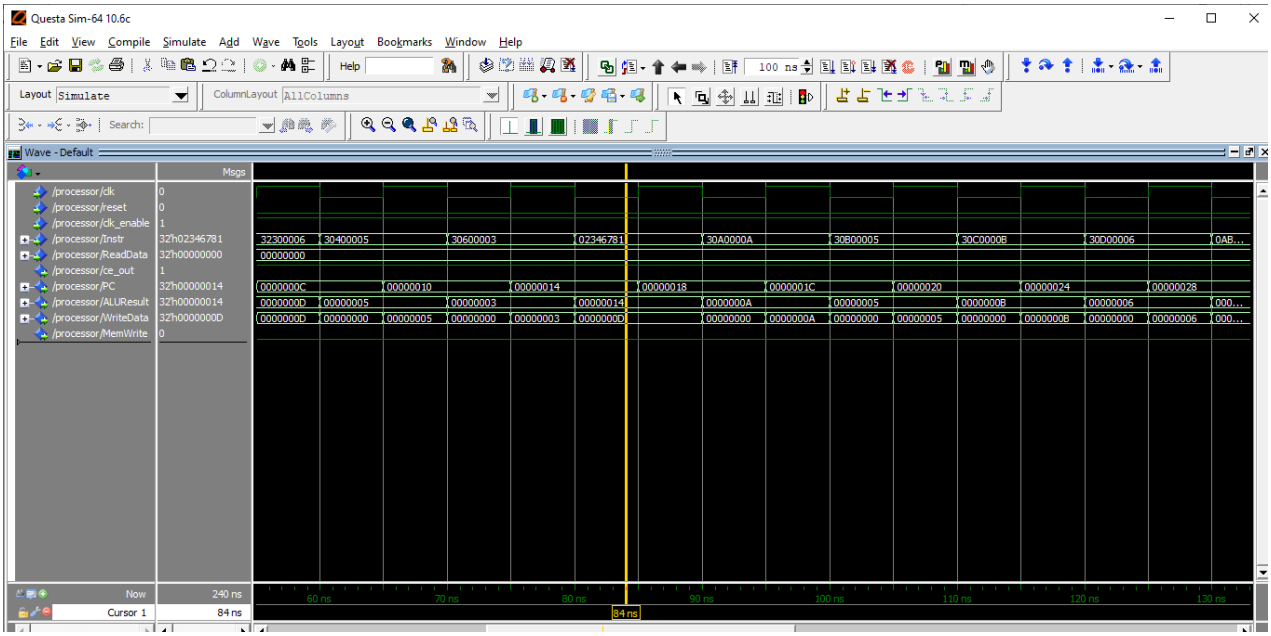


Рисунок 4.14 – Верифікація в Questa Sim

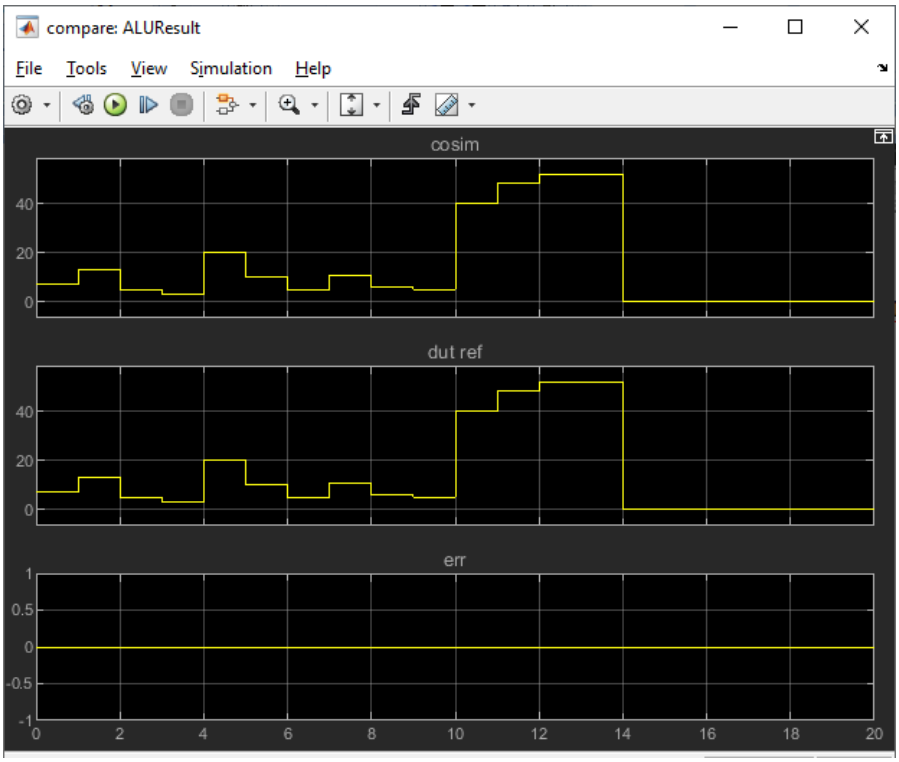


Рисунок 4.15 – Вікно помилок для ALUResult

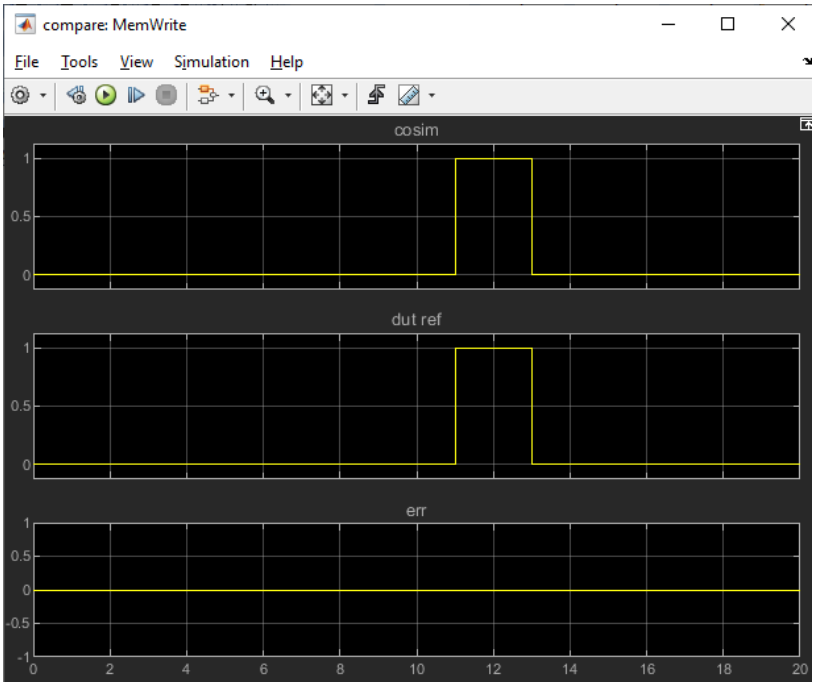


Рисунок 4.16 – Вікно помилок для MemWrite

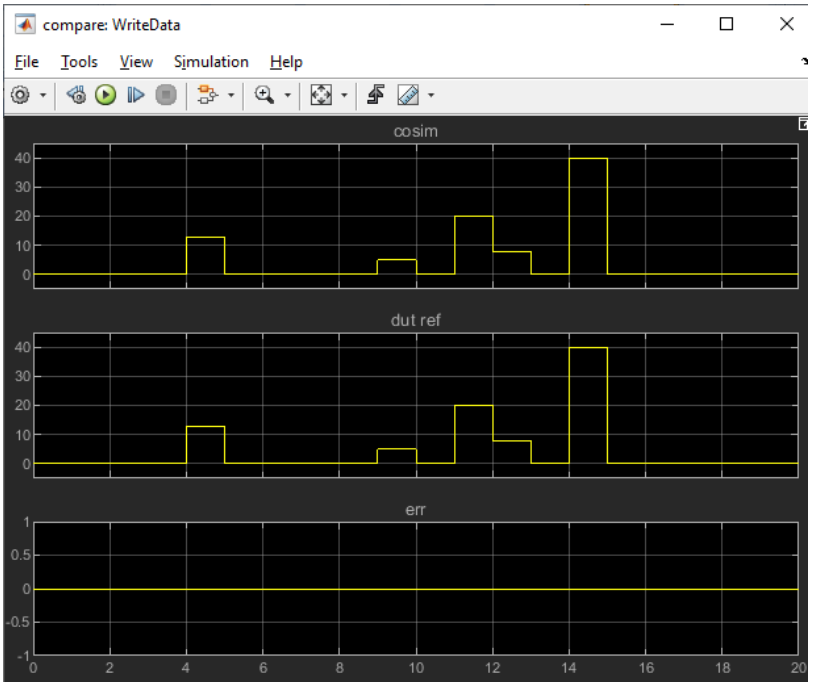


Рисунок 4.17 – Вікно помилок для WriteData

ВИСНОВКИ

У кваліфікаційній роботі було розроблено та реалізовано мовою VHDL однотактовий RISC-процесор для виконання дій над комплексними числами та проведено його тестування. Під час реалізації процесору були розглянуті високорівневі технології проектування, які надає пакет програм Matlab/Simuink. Ці технології дозволяють суттєво прискорити і спростити проектування. Також при розробці були використані система автоматизованого проектування цифрових пристроїв, мова опису апаратури VHDL і ПЛІС.

Реалізований на ПЛІС процесор можна використовувати для проведення обчислень додавання та віднімання комплексних чисел. При необхідності проект у пакеті Matlab/Simulink можна достатньо легко модифікувати та деталізувати.

Для тестування процесору була спроектована ROM пам'ять, в якій зберігається програма в машинних кодах для виконання процесором. Тестування було проведено шляхом виконання симуляції з відображенням можливих помилок при виконанні операцій додавання та віднімання комплексних чисел.

Проект можна покращувати далі, наприклад: розширити кількість математичних команд над комплексними числами, додати команди переходів, що дало б змогу писати більш складні програми, реалізувати можливість виконання дій над числами з плаваючою комою, модифікувати мікроархітектуру процесору тощо.

Результати кваліфікаційної роботи апробовані на XVI Міжнародній науково-практичній конференції «СУЧАСНІ ІНФОРМАЦІЙНІ І КОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ НА ТРАНСПОРТІ, В ПРОМИСЛОВОСТІ ТА ОСВІТІ» [13].

ПЕРЕЛІК ПОСИЛАНЬ

1. Сергиенко А. М., Корнейчук В. И. Микропроцессорные устройства на программируемых логических ИС : Киев : ЧП «Корнейчук», 2005. 108 с.

2. Сергиенко А. М. VHDL для проектирования вычислительных устройств : Киев : ЧП «Корнейчук», ООО «ТИД «ДС», 2003. 208 с.

3. Harris D. M., Harris S. L. Digital Design and Computer Architecture :NewYork: «Morgan Kaufmann», 2013. 1662 с.

4. Хомич С.В., курс лекцій з дисципліни: Мови опису апаратних засобів. – Інститут Комп'ютерної Техніки та Автоматики, 2005. – 129с.

5. Get Started with Simulink Hdl Cosimulation [Електронний ресурс] // Режим доступу до ресурсу: https://www.mathworks.com/help/hdlverifier/ug/get-started-simulink-cosimulation-hdl.html.

6. Реализация арифметических операций с комплексными числами на ПЛИС [Текст] / В. Н. Опанасенко, А. Н. Лисовый, В. Г. Сахарин // Технология и проектирование в электронной аппаратуре, 2008. №5. – С.24-30.

7. Дьяконов В. П. MATLAB. Полный самоучитель : Москва : «ДМК Прес», 2012. - 768 с.

8. Строгонов А. В., Буслов А. И. Проектирование учебного процессора для реализации в базисе ПЛИС с использованием системы MATLAB/ Simulink // Компоненты и технологии. 2009. № 5.

9. Строгонов А. В. Проектирование учебного процессора с фиксированной запятой в системе Matlab/Simulink / А. В. Строгонов // Компоненты и технологии. 2009. № 7.

10. Строгонов А. В., Цыбин С. А., Буслов А. И. Проектирование микропроцессорных ядер с использованием приложения StateFlow системы MATLAB/Simulink // Компоненты и технологии. 2010. № 1.

11. Тарасов И. Проектирование конфигурируемых процессоров на базе ПЛИС. Часть I // Компоненты и технологии. 2006. № 2.

12. Тарасов И. Проектирование конфигурируемых процессоров на

базе ПЛИС. Часть II // Компоненты и технологии. 2006. № 3.

13. Лазоренко Д. В., Шаповалов В.О. Особливості реалізації в ПЛІС процесора обробки комплексних чисел і стенду для його тестування // Тези XVI-ої Міжнародної науково-практичної конференції «СУЧАСНІ ІНФОРМАЦІЙНІ І КОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ НА ТРАНСПОРТІ, В ПРОМИСЛОВОСТІ ТА ОСВІТІ». – 2022. – с. 28.

Додатки

Додаток А

VHDL-код верхнього рівня процесору

```
-- Rate and Clocking Details
-- -------------------------------------------------------
-- Model base rate: 1
-- Target subsystem base rate: 1
--
--
-- Clock Enable  Sample Time
-- -------------------------------------------------------
-- ce_out       1
-- -------------------------------------------------------
--
--
-- Output Signal                 Clock Enable  Sample Time
-- -------------------------------------------------------
-- PC                       ce_out       1
-- ALUResult                ce_out       1
-- WriteData                ce_out       1
-- MemWrite                 ce_out       1
-- Module: Processor
-- Source Path: ComplexPr/Processor
-- Hierarchy Level: 0
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY Processor IS
 PORT( clk                        : IN    std_logic;
       reset                      : IN    std_logic;
       clk_enable                 : IN    std_logic;
       Instr                      : IN    std_logic_vector(31 DOWNTO 0); -- uint32
       ReadData                   : IN    std_logic_vector(31 DOWNTO 0); -- uint32
       ce_out                     : OUT   std_logic;
       PC                         : OUT   std_logic_vector(31 DOWNTO 0); -- uint32
       ALUResult                  : OUT   std_logic_vector(31 DOWNTO 0); -- uint32
       WriteData                  : OUT   std_logic_vector(31 DOWNTO 0); -- uint32
       MemWrite                   : OUT   std_logic -- ufix1
       );
END Processor;


ARCHITECTURE rtl OF Processor IS

 -- Component Declarations
 COMPONENT Controller
   PORT( op                       : IN    std_logic_vector(3 DOWNTO 0); -- ufix4
         funct                    : IN    std_logic_vector(3 DOWNTO 0); -- ufix4
         control_MemtoReg         : OUT   std_logic; -- ufix1
         control_MemWrite         : OUT   std_logic; -- ufix1
         control_ALUSrc           : OUT   std_logic; -- ufix1
         control_RegDst           : OUT   std_logic; -- ufix1
         control_RegWrite         : OUT   std_logic; -- ufix1
         control_RegWriteIm       : OUT   std_logic; -- ufix1
         control_ALUControl       : OUT   std_logic_vector(1 DOWNTO 0)  -- ufix2
         );
```

```
 END COMPONENT;

 COMPONENT Datapath
  PORT( clk                    :  IN    std_logic;
        reset                  :  IN    std_logic;
        enb                    :  IN    std_logic;
        control_MemtoReg       :  IN    std_logic; -- ufix1
        control_ALUSrc         :  IN    std_logic; -- ufix1
        control_RegDst         :  IN    std_logic; -- ufix1
        control_RegWrite       :  IN    std_logic; -- ufix1
        control_RegWriteIm     :  IN    std_logic; -- ufix1
        control_ALUControl     :  IN    std_logic_vector(1 DOWNTO 0); -- ufix2
        Instr                  :  IN    std_logic_vector(31 DOWNTO 0); -- uint32
        ReadData               :  IN    std_logic_vector(31 DOWNTO 0); -- uint32
        op                     :  OUT   std_logic_vector(3 DOWNTO 0); -- ufix4
        funct                  :  OUT   std_logic_vector(3 DOWNTO 0); -- ufix4
        PC                     :  OUT   std_logic_vector(31 DOWNTO 0); -- uint32
        ALUResult              :  OUT   std_logic_vector(31 DOWNTO 0); -- uint32
        WriteData              :  OUT   std_logic_vector(31 DOWNTO 0)  -- uint32
        );
 END COMPONENT;

 -- Component Configuration Statements
 FOR ALL : Controller
   USE ENTITY  work.Controller(rtl);

 FOR ALL : Datapath
   USE ENTITY  work.Datapath(rtl);

 -- Signals
 SIGNAL  Datapath_out1               : std_logic_vector(3 DOWNTO 0);  -- ufix4
 SIGNAL  Datapath_out2               : std_logic_vector(3 DOWNTO 0);  -- ufix4
 SIGNAL  Controller_out1_MemtoReg    : std_logic;  -- ufix1
 SIGNAL  Controller_out1_MemWrite    : std_logic;  -- ufix1
 SIGNAL  Controller_out1_ALUSrc      : std_logic;  -- ufix1
 SIGNAL  Controller_out1_RegDst      : std_logic;  -- ufix1
 SIGNAL  Controller_out1_RegWrite    : std_logic;  -- ufix1
 SIGNAL  Controller_out1_RegWriteIm  : std_logic;  -- ufix1
 SIGNAL  Controller_out1_ALUControl  : std_logic_vector(1 DOWNTO 0);  -- ufix2
 SIGNAL  Datapath_out3               : std_logic_vector(31 DOWNTO 0);  -- ufix32
 SIGNAL  ALUResult_tmp               : std_logic_vector(31 DOWNTO 0);  -- ufix32
 SIGNAL  RD2                         : std_logic_vector(31 DOWNTO 0);  -- ufix32

BEGIN
 u_Controller : Controller
   PORT MAP( op => Datapath_out1,  -- ufix4
             funct => Datapath_out2,  -- ufix4
             control_MemtoReg => Controller_out1_MemtoReg,  -- ufix1
             control_MemWrite => Controller_out1_MemWrite,  -- ufix1
             control_ALUSrc => Controller_out1_ALUSrc,  -- ufix1
             control_RegDst => Controller_out1_RegDst,  -- ufix1
             control_RegWrite => Controller_out1_RegWrite,  -- ufix1
             control_RegWriteIm => Controller_out1_RegWriteIm,  -- ufix1
             control_ALUControl => Controller_out1_ALUControl  -- ufix2
             );

 u_Datapath : Datapath
   PORT MAP( clk => clk,
             reset => reset,
             enb => clk_enable,
             control_MemtoReg => Controller_out1_MemtoReg,  -- ufix1
             control_ALUSrc => Controller_out1_ALUSrc,  -- ufix1
             control_RegDst => Controller_out1_RegDst,  -- ufix1
```

```
        control_RegWrite => Controller_out1_RegWrite,  -- ufix1
        control_RegWriteIm => Controller_out1_RegWriteIm,  -- ufix1
        control_ALUControl => Controller_out1_ALUControl,  -- ufix2
        Instr => Instr,  -- uint32
        ReadData => ReadData,  -- uint32
        op => Datapath_out1,  -- ufix4
        funct => Datapath_out2,  -- ufix4
        PC => Datapath_out3,  -- uint32
        ALUResult => ALUResult_tmp,  -- uint32
        WriteData => RD2  -- uint32
        );

  MemWrite <= Controller_out1_MemWrite;

  ce_out <= clk_enable;

  PC <= Datapath_out3;

  ALUResult <= ALUResult_tmp;

  WriteData <= RD2;

END rtl;
```

VHDL-код контролеру

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY Controller IS
  PORT( op                        : IN    std_logic_vector(3 DOWNTO 0);  -- ufix4
        funct                     : IN    std_logic_vector(3 DOWNTO 0);  -- ufix4
        control_MemtoReg          : OUT   std_logic;  -- ufix1
        control_MemWrite          : OUT   std_logic;  -- ufix1
        control_ALUSrc            : OUT   std_logic;  -- ufix1
        control_RegDst            : OUT   std_logic;  -- ufix1
        control_RegWrite          : OUT   std_logic;  -- ufix1
        control_RegWriteIm        : OUT   std_logic;  -- ufix1
        control_ALUControl        : OUT   std_logic_vector(1 DOWNTO 0)  -- ufix2
        );
END Controller;

ARCHITECTURE rtl OF Controller IS
  -- Component Declarations
  COMPONENT MainDec
    PORT( op                      : IN    std_logic_vector(3 DOWNTO 0);  -- ufix4
          MemtoReg                : OUT   std_logic;  -- ufix1
          MemWrite                : OUT   std_logic;  -- ufix1
          ALUSrc                  : OUT   std_logic;  -- ufix1
          RegDst                  : OUT   std_logic;  -- ufix1
          RegWrite                : OUT   std_logic;  -- ufix1
          RegWriteIm              : OUT   std_logic;  -- ufix1
          ALUOp                   : OUT   std_logic_vector(1 DOWNTO 0)  -- ufix2
          );
  END COMPONENT;

  COMPONENT ALUDec
    PORT( ALUOp                   : IN    std_logic_vector(1 DOWNTO 0);  -- ufix2
          funct                   : IN    std_logic_vector(3 DOWNTO 0);  -- ufix4
          ALUControl              : OUT   std_logic_vector(1 DOWNTO 0)  -- ufix2
          );
  END COMPONENT;
```

```
 -- Component Configuration Statements
 FOR ALL : MainDec
   USE ENTITY  work.MainDec(rtl);

 FOR ALL : ALUDec
   USE ENTITY  work.ALUDec(rtl);

 -- Signals
 SIGNAL  MainDec_out1              : std_logic;  -- ufix1
 SIGNAL  MainDec_out2              : std_logic;  -- ufix1
 SIGNAL  MainDec_out3              : std_logic;  -- ufix1
 SIGNAL  MainDec_out4              : std_logic;  -- ufix1
 SIGNAL  MainDec_out5              : std_logic;  -- ufix1
 SIGNAL  MainDec_out6              : std_logic;  -- ufix1
 SIGNAL  MainDec_out7              : std_logic_vector(1 DOWNTO 0);  -- ufix2
 SIGNAL  ALUControl                : std_logic_vector(1 DOWNTO 0);  -- ufix2

BEGIN
 u_MainDec : MainDec
   PORT MAP( op => op,  -- ufix4
         MemtoReg => MainDec_out1,  -- ufix1
         MemWrite => MainDec_out2,  -- ufix1
         ALUSrc => MainDec_out3,  -- ufix1
         RegDst => MainDec_out4,  -- ufix1
         RegWrite => MainDec_out5,  -- ufix1
         RegWriteIm => MainDec_out6,  -- ufix1
         ALUOp => MainDec_out7  -- ufix2
         );

 u_ALUDec : ALUDec
   PORT MAP( ALUOp => MainDec_out7,  -- ufix2
         funct => funct,  -- ufix4
         ALUControl => ALUControl  -- ufix2
         );

 control_MemtoReg <= MainDec_out1;

 control_MemWrite <= MainDec_out2;

 control_ALUSrc <= MainDec_out3;

 control_RegDst <= MainDec_out4;

 control_RegWrite <= MainDec_out5;

 control_RegWriteIm <= MainDec_out6;

 control_ALUControl <= ALUControl;

END rtl;
```

VHDL-код докедору АЛП

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY  ALUDec IS
  PORT( ALUOp                  :  IN   std_logic_vector(1 DOWNTO 0);  -- ufix2
       funct                   :  IN   std_logic_vector(3 DOWNTO 0);  -- ufix4
       ALUControl              :  OUT  std_logic_vector(1 DOWNTO 0)  -- ufix2
       );
END ALUDec;
```

```
ARCHITECTURE rtl OF ALUDec IS

 -- Signals
 SIGNAL ALUOp_unsigned          : unsigned(1 DOWNTO 0); -- ufix2
 SIGNAL funct_unsigned          : unsigned(3 DOWNTO 0); -- ufix4
 SIGNAL ALUControl_tmp          : unsigned(1 DOWNTO 0); -- ufix2

BEGIN
 ALUOp_unsigned <= unsigned(ALUOp);

 funct_unsigned <= unsigned(funct);

 MATLAB_Function_output : PROCESS (ALUOp_unsigned, funct_unsigned)
 BEGIN
  --MATLAB Function 'Processor/Controller/ALUDec/MATLAB Function'
  CASE ALUOp_unsigned IS
   WHEN "00" =>
    --add (for lw/sw/addi)
    ALUControl_tmp <= to_unsigned(16#1#, 2);
   WHEN OTHERS =>
    --R-type instructions
    CASE funct_unsigned IS
     WHEN "1000" =>
      --add
      ALUControl_tmp <= to_unsigned(16#1#, 2);
     WHEN "0001" =>
      --complexadd
      ALUControl_tmp <= to_unsigned(16#2#, 2);
     WHEN "0010" =>
      --complexsub
      ALUControl_tmp <= to_unsigned(16#3#, 2);
     WHEN OTHERS =>
      --ILLEGAL OP
      ALUControl_tmp <= to_unsigned(16#0#, 2);
    END CASE;
  END CASE;
 END PROCESS MATLAB_Function_output;

 ALUControl <= std_logic_vector(ALUControl_tmp);

END rtl;
```

VHDL-код основного докедору

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY MainDec IS
 PORT( op                : IN  std_logic_vector(3 DOWNTO 0); -- ufix4
       MemtoReg          : OUT std_logic; -- ufix1
       MemWrite          : OUT std_logic; -- ufix1
       ALUSrc            : OUT std_logic; -- ufix1
       RegDst            : OUT std_logic; -- ufix1
       RegWrite          : OUT std_logic; -- ufix1
       RegWriteIm        : OUT std_logic; -- ufix1
       ALUOp             : OUT std_logic_vector(1 DOWNTO 0) -- ufix2
       );
END MainDec;

ARCHITECTURE rtl OF MainDec IS
```

```
  -- Signals
  SIGNAL op_unsigned              : unsigned(3 DOWNTO 0); -- ufix4
  SIGNAL MainDec_out1             : unsigned(7 DOWNTO 0); -- uint8
  SIGNAL ALUOp_tmp                : unsigned(1 DOWNTO 0); -- ufix2

BEGIN
  op_unsigned <= unsigned(op);

  MainDec_1_output : PROCESS (op_unsigned)
  BEGIN
    --MATLAB Function 'Processor/Controller/MainDec/MainDec'
    CASE op_unsigned IS
      WHEN "0000" =>
        --RTYPE
        MainDec_out1 <= to_unsigned(16#E2#, 8);
      WHEN "0001" =>
        --LW
        MainDec_out1 <= to_unsigned(16#94#, 8);
      WHEN "0010" =>
        --SW
        MainDec_out1 <= to_unsigned(16#18#, 8);
      WHEN "0011" =>
        --ADDI
        MainDec_out1 <= to_unsigned(16#90#, 8);
      WHEN OTHERS =>
        --ILLEGAL OP
        MainDec_out1 <= to_unsigned(16#00#, 8);
    END CASE;
  END PROCESS MainDec_1_output;


  MemtoReg <= MainDec_out1(2);
  MemWrite <= MainDec_out1(3);
  ALUSrc <= MainDec_out1(4);
  RegDst <= MainDec_out1(5);
  RegWrite <= MainDec_out1(7);
  RegWriteIm <= MainDec_out1(6);
  ALUOp_tmp <= MainDec_out1(1 DOWNTO 0);
  ALUOp <= std_logic_vector(ALUOp_tmp);

END rtl;
```

VHDL-код АЛП

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY ALU IS
  PORT( SrcX1                 : IN  std_logic_vector(31 DOWNTO 0); -- uint32
        ALUControl            : IN  std_logic_vector(1 DOWNTO 0); -- ufix2
        SrcX2                 : IN  std_logic_vector(31 DOWNTO 0); -- uint32
        SrcY1                 : IN  std_logic_vector(31 DOWNTO 0); -- uint32
        SrcY2                 : IN  std_logic_vector(31 DOWNTO 0); -- uint32
        ALUResult             : OUT std_logic_vector(31 DOWNTO 0); -- uint32
        ALUResultIm           : OUT std_logic_vector(31 DOWNTO 0) -- uint32
        );
END ALU;


ARCHITECTURE rtl OF ALU IS
```

```
  -- Signals
  SIGNAL ALUControl_unsigned          : unsigned(1 DOWNTO 0);  -- ufix2
  SIGNAL SrcX1_unsigned               : unsigned(31 DOWNTO 0);  -- uint32
  SIGNAL SrcX2_unsigned               : unsigned(31 DOWNTO 0);  -- uint32
  SIGNAL SrcY1_unsigned               : unsigned(31 DOWNTO 0);  -- uint32
  SIGNAL SrcY2_unsigned               : unsigned(31 DOWNTO 0);  -- uint32
  SIGNAL MATLAB_Function_out1         : unsigned(31 DOWNTO 0);  -- uint32
  SIGNAL MATLAB_Function_out2         : unsigned(31 DOWNTO 0);  -- uint32

BEGIN
  ALUControl_unsigned <= unsigned(ALUControl);

  SrcX1_unsigned <= unsigned(SrcX1);

  SrcX2_unsigned <= unsigned(SrcX2);

  SrcY1_unsigned <= unsigned(SrcY1);

  SrcY2_unsigned <= unsigned(SrcY2);

  MATLAB_Function_output : PROCESS (ALUControl_unsigned, SrcX1_unsigned, SrcX2_unsigned,
SrcY1_unsigned,
      SrcY2_unsigned)
  BEGIN
    --MATLAB Function 'Processor/Datapath/ALU/MATLAB Function'
    MATLAB_Function_out1 <= to_unsigned(0, 32);
    MATLAB_Function_out2 <= to_unsigned(0, 32);
    CASE ALUControl_unsigned IS
      WHEN "01" =>
        --add
        MATLAB_Function_out1 <= SrcX1_unsigned + SrcX2_unsigned;
      WHEN "10" =>
        --complex add
        MATLAB_Function_out1 <= SrcX1_unsigned + SrcX2_unsigned;
        MATLAB_Function_out2 <= SrcY1_unsigned + SrcY2_unsigned;
      WHEN "11" =>
        --complex sub
        MATLAB_Function_out1 <= SrcX1_unsigned - SrcX2_unsigned;
        MATLAB_Function_out2 <= SrcY1_unsigned - SrcY2_unsigned;
      WHEN OTHERS =>
        NULL;
    END CASE;
  END PROCESS MATLAB_Function_output;


  ALUResult <= std_logic_vector(MATLAB_Function_out1);

  ALUResultIm <= std_logic_vector(MATLAB_Function_out2);

END rtl;
```

VHDL-код мультиплексора ALUSrcMux

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY ALUSrcMux IS
  PORT( RD2                       :  IN    std_logic_vector(31 DOWNTO 0);  -- uint32
        ALUSrc                    :  IN    std_logic;  -- ufix1
        SignImm                   :  IN    std_logic_vector(31 DOWNTO 0);  -- uint32
        SrcB                      :  OUT   std_logic_vector(31 DOWNTO 0)  -- uint32
```

```
      );
END ALUSrcMux;


ARCHITECTURE  rtl OF ALUSrcMux IS

  -- Signals
  SIGNAL  switch_compare_1              : std_logic;
  SIGNAL  RD2_unsigned                  : unsigned(31 DOWNTO 0);  -- uint32
  SIGNAL  SignImm_unsigned              : unsigned(31 DOWNTO 0);  -- uint32
  SIGNAL  ALUSrcMux_out1                : unsigned(31 DOWNTO 0);  -- uint32

BEGIN
  switch_compare_1 <= '1' WHEN ALUSrc > '0' ELSE
     '0';
  RD2_unsigned <= unsigned(RD2);
  SignImm_unsigned <= unsigned(SignImm);
  ALUSrcMux_out1 <= RD2_unsigned WHEN switch_compare_1 = '0' ELSE
     SignImm_unsigned;
  SrcB <= std_logic_vector(ALUSrcMux_out1);
END rtl;
```

VHDL-код аналізатора інструкцій

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY Instruction_Parser IS
 PORT( Instr                : IN    std_logic_vector(31 DOWNTO 0);  -- uint32
       op                   : OUT   std_logic_vector(3 DOWNTO 0);  -- ufix4
       funct                : OUT   std_logic_vector(3 DOWNTO 0);  -- ufix4
       rsx                  : OUT   std_logic_vector(3 DOWNTO 0);  -- ufix4
       rtx                  : OUT   std_logic_vector(3 DOWNTO 0);  -- ufix4
       rsy                  : OUT   std_logic_vector(3 DOWNTO 0);  -- ufix4
       rty                  : OUT   std_logic_vector(3 DOWNTO 0);  -- ufix4
       zr                   : OUT   std_logic_vector(3 DOWNTO 0);  -- ufix4
       zi                   : OUT   std_logic_vector(3 DOWNTO 0);  -- ufix4
       Imm                  : OUT   std_logic_vector(15 DOWNTO 0)  -- uint16
       );
END Instruction_Parser;


ARCHITECTURE rtl OF Instruction_Parser IS

 -- Signals
 SIGNAL Instr_unsigned          : unsigned(31 DOWNTO 0);  -- uint32
 SIGNAL op_tmp                  : unsigned(3 DOWNTO 0);  -- ufix4
 SIGNAL funct_tmp               : unsigned(3 DOWNTO 0);  -- ufix4
 SIGNAL rsx_tmp                 : unsigned(3 DOWNTO 0);  -- ufix4
 SIGNAL rtx_tmp                 : unsigned(3 DOWNTO 0);  -- ufix4
 SIGNAL rsy_tmp                 : unsigned(3 DOWNTO 0);  -- ufix4
 SIGNAL rsy_1                   : unsigned(3 DOWNTO 0);  -- ufix4
 SIGNAL zr_tmp                  : unsigned(3 DOWNTO 0);  -- ufix4
 SIGNAL zi_tmp                  : unsigned(3 DOWNTO 0);  -- ufix4
 SIGNAL imm_1                   : unsigned(15 DOWNTO 0);  -- uint16

BEGIN
 Instr_unsigned <= unsigned(Instr);
 op_tmp <= Instr_unsigned(31 DOWNTO 28);
 op <= std_logic_vector(op_tmp);
 funct_tmp <= Instr_unsigned(3 DOWNTO 0);
 funct <= std_logic_vector(funct_tmp);
 rsx_tmp <= Instr_unsigned(27 DOWNTO 24);
 rsx <= std_logic_vector(rsx_tmp);
 rtx_tmp <= Instr_unsigned(23 DOWNTO 20);
 rtx <= std_logic_vector(rtx_tmp);
 rsy_tmp <= Instr_unsigned(19 DOWNTO 16);
 rsy <= std_logic_vector(rsy_tmp);
 rsy_1 <= Instr_unsigned(15 DOWNTO 12);
 rty <= std_logic_vector(rsy_1);
 zr_tmp <= Instr_unsigned(11 DOWNTO 8);
 zr <= std_logic_vector(zr_tmp);
 zi_tmp <= Instr_unsigned(7 DOWNTO 4);
 zi <= std_logic_vector(zi_tmp);
 imm_1 <= Instr_unsigned(15 DOWNTO 0);
 Imm <= std_logic_vector(imm_1);

END rtl;
```

VHDL-код мультиплексора MemtoRegMux

```
LIBRARY IEEE;
```

```
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY  MemtoRegMux IS
 PORT( RD                        :  IN    std_logic_vector(31 DOWNTO 0);  -- uint32
      MemtoReg                   :  IN    std_logic;  -- ufix1
      ALUResult                  :  IN    std_logic_vector(31 DOWNTO 0);  -- uint32
      Result                     :  OUT   std_logic_vector(31 DOWNTO 0)  -- uint32
      );
END MemtoRegMux;


ARCHITECTURE  rtl OF MemtoRegMux IS
 -- Signals
 SIGNAL  switch_compare_1             : std_logic;
 SIGNAL  ALUResult_unsigned           : unsigned(31 DOWNTO 0);  -- uint32
 SIGNAL  RD_unsigned                  : unsigned(31 DOWNTO 0);  -- uint32
 SIGNAL  MemtoRegMux_out1             : unsigned(31 DOWNTO 0);  -- uint32

BEGIN
 switch_compare_1 <= '1' WHEN MemtoReg > '0' ELSE
    '0';
 ALUResult_unsigned <= unsigned(ALUResult);
 RD_unsigned <= unsigned(RD);
 MemtoRegMux_out1 <= ALUResult_unsigned WHEN switch_compare_1 = '0' ELSE
    RD_unsigned;
 Result <= std_logic_vector(MemtoRegMux_out1);

END rtl;
```

VHDL-код мультиплексора RegDstMux

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY  RegDstMux IS
 PORT( rtx                       :  IN    std_logic_vector(3 DOWNTO 0);  -- ufix4
      RegDst                     :  IN    std_logic;  -- ufix1
      rd                         :  IN    std_logic_vector(3 DOWNTO 0);  -- ufix4
      r                          :  OUT   std_logic_vector(3 DOWNTO 0)  -- ufix4
      );
END RegDstMux;


ARCHITECTURE  rtl OF RegDstMux IS

 -- Signals
 SIGNAL  switch_compare_1             : std_logic;
 SIGNAL  rtx_unsigned                 : unsigned(3 DOWNTO 0);  -- ufix4
 SIGNAL  rd_unsigned                  : unsigned(3 DOWNTO 0);  -- ufix4
 SIGNAL  RegDstMux_out1               : unsigned(3 DOWNTO 0);  -- ufix4

BEGIN

 switch_compare_1 <= '1' WHEN RegDst > '0' ELSE
    '0';
 rtx_unsigned <= unsigned(rtx);
 rd_unsigned <= unsigned(rd);
 RegDstMux_out1 <= rtx_unsigned WHEN switch_compare_1 = '0' ELSE
    rd_unsigned;
 r <= std_logic_vector(RegDstMux_out1);
```

```
END rtl;
```

VHDL-код лічильника команд

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY PCCalc IS
 PORT( clk                    : IN  std_logic;
       reset                  : IN  std_logic;
       enb                    : IN  std_logic;
       PC_in                  : IN  std_logic_vector(31 DOWNTO 0); -- uint32
       PC                     : OUT std_logic_vector(31 DOWNTO 0) -- uint32
       );
END PCCalc;

ARCHITECTURE rtl OF PCCalc IS

 -- Signals
 SIGNAL PC_in_unsigned        : unsigned(31 DOWNTO 0); -- uint32
 SIGNAL PC_tmp                : unsigned(31 DOWNTO 0); -- uint32

BEGIN
 PC_in_unsigned <= unsigned(PC_in);
 PC_Start_process : PROCESS (clk, reset)
 BEGIN
  IF reset = '1' THEN
   PC_tmp <= to_unsigned(0, 32);
  ELSIF clk'EVENT AND clk = '1' THEN
   IF enb = '1' THEN
    PC_tmp <= PC_in_unsigned;
   END IF;
  END IF;
 END PROCESS PC_Start_process;
 PC <= std_logic_vector(PC_tmp);

END rtl;
```

VHDL-код обчислення лічильника

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY PCPlus4 IS
 PORT( PC                     : IN  std_logic_vector(31 DOWNTO 0); -- uint32
       PCPlus4_1              : OUT std_logic_vector(31 DOWNTO 0) -- uint32
       );
END PCPlus4;


ARCHITECTURE rtl OF PCPlus4 IS

 -- Signals
 SIGNAL PC_unsigned           : unsigned(31 DOWNTO 0); -- uint32
 SIGNAL PCInc_out1            : unsigned(31 DOWNTO 0); -- uint32

BEGIN
 PC_unsigned <= unsigned(PC);
```

```
  PCInc_out1 <= PC_unsigned + to_unsigned(4, 32);

  PCPlus4_1 <= std_logic_vector(PCInc_out1);

END rtl;
```

VHDL-код регістрового файлу

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
USE work.Processor_pkg.ALL;

ENTITY RegisterFile IS
 PORT( clk                : IN  std_logic;
       reset              : IN  std_logic;
       enb                : IN  std_logic;
       RA1                : IN  std_logic_vector(3 DOWNTO 0); -- ufix4
       RA2                : IN  std_logic_vector(3 DOWNTO 0); -- ufix4
       RA3                : IN  std_logic_vector(3 DOWNTO 0); -- ufix4
       RA4                : IN  std_logic_vector(3 DOWNTO 0); -- ufix4
       WA3                : IN  std_logic_vector(3 DOWNTO 0); -- ufix4
       WA4                : IN  std_logic_vector(3 DOWNTO 0); -- ufix4
       WD3                : IN  std_logic_vector(31 DOWNTO 0); -- uint32
       WD4                : IN  std_logic_vector(31 DOWNTO 0); -- uint32
       WE3                : IN  std_logic; -- ufix1
       WE4                : IN  std_logic; -- ufix1
       RD1                : OUT std_logic_vector(31 DOWNTO 0); -- uint32
       RD2                : OUT std_logic_vector(31 DOWNTO 0); -- uint32
       RD3                : OUT std_logic_vector(31 DOWNTO 0); -- uint32
       RD4                : OUT std_logic_vector(31 DOWNTO 0)  -- uint32
       );
END RegisterFile;


ARCHITECTURE rtl OF RegisterFile IS

 -- Signals
 SIGNAL RA1_unsigned          : unsigned(3 DOWNTO 0); -- ufix4
 SIGNAL RA2_unsigned          : unsigned(3 DOWNTO 0); -- ufix4
 SIGNAL RA3_unsigned          : unsigned(3 DOWNTO 0); -- ufix4
 SIGNAL RA4_unsigned          : unsigned(3 DOWNTO 0); -- ufix4
 SIGNAL WA3_unsigned          : unsigned(3 DOWNTO 0); -- ufix4
 SIGNAL WA4_unsigned          : unsigned(3 DOWNTO 0); -- ufix4
 SIGNAL WD3_unsigned          : unsigned(31 DOWNTO 0); -- uint32
 SIGNAL WD4_unsigned          : unsigned(31 DOWNTO 0); -- uint32
 SIGNAL Unit_Delay_out1       : vector_of_unsigned32(0 TO 31); -- uint32 [32]
 SIGNAL Write_out1            : vector_of_unsigned32(0 TO 31); -- uint32 [32]
 SIGNAL Read_out1             : unsigned(31 DOWNTO 0); -- uint32
 SIGNAL Read_out2             : unsigned(31 DOWNTO 0); -- uint32
 SIGNAL Read_out3             : unsigned(31 DOWNTO 0); -- uint32
 SIGNAL Read_out4             : unsigned(31 DOWNTO 0); -- uint32

BEGIN

 RA1_unsigned <= unsigned(RA1);
 RA2_unsigned <= unsigned(RA2);
 RA3_unsigned <= unsigned(RA3);
 RA4_unsigned <= unsigned(RA4);
 WA3_unsigned <= unsigned(WA3);
 WA4_unsigned <= unsigned(WA4);
```

```
  WD3_unsigned <= unsigned(WD3);

  WD4_unsigned <= unsigned(WD4);
  Write_output : PROCESS (Unit_Delay_out1, WA3_unsigned, WA4_unsigned, WD3_unsigned,
WD4_unsigned, WE3,
      WE4)
  BEGIN
    --MATLAB Function 'Processor/Datapath/RegisterFile/Write'
    --Zero based address into one based MATLAB indexing of vectors
    Write_out1 <= Unit_Delay_out1;
    IF WE3 /= '0' THEN
      Write_out1(to_integer(WA3_unsigned)) <= WD3_unsigned;
    END IF;
    IF WE4 /= '0' THEN
      Write_out1(to_integer(WA4_unsigned)) <= WD4_unsigned;
    END IF;
  END PROCESS Write_output;

  Unit_Delay_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      Unit_Delay_out1 <= (OTHERS => to_unsigned(0, 32));
    ELSIF clk'EVENT AND clk = '1' THEN
      IF enb = '1' THEN
        Unit_Delay_out1 <= Write_out1;
      END IF;
    END IF;
  END PROCESS Unit_Delay_process;

  --MATLAB Function 'Processor/Datapath/RegisterFile/Read'
  --Zero based address into one based MATLAB indexing of vectors
  Read_out1 <= Unit_Delay_out1(to_integer(RA1_unsigned));
  Read_out2 <= Unit_Delay_out1(to_integer(RA2_unsigned));
  Read_out3 <= Unit_Delay_out1(to_integer(RA3_unsigned));
  Read_out4 <= Unit_Delay_out1(to_integer(RA4_unsigned));
  RD1 <= std_logic_vector(Read_out1);
  RD2 <= std_logic_vector(Read_out2);
  RD3 <= std_logic_vector(Read_out3);
  RD4 <= std_logic_vector(Read_out4);

END rtl;
```

VHDL-код розширення знаку

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY Sign_Extend IS
  PORT( Imm                          :   IN    std_logic_vector(15 DOWNTO 0);  -- uint16
        SignImm                      :   OUT   std_logic_vector(31 DOWNTO 0)  -- uint32
        );
END Sign_Extend;

ARCHITECTURE rtl OF Sign_Extend IS

  -- Signals
  SIGNAL Imm_unsigned                : unsigned(15 DOWNTO 0);  -- uint16
  SIGNAL MATLAB_Function_out1        : unsigned(31 DOWNTO 0);  -- uint32
  SIGNAL c                           : std_logic;  -- ufix1

BEGIN
```

```
    Imm_unsigned <= unsigned(Imm);

    --MATLAB Function 'Processor/Datapath/Sign Extend/MATLAB Function'
    c <= Imm_unsigned(15);
    MATLAB_Function_out1 <= unsigned'(c & c & c & c & c & c & c & c & c & c & c & c & c & c & c & c)
 & Imm_unsigned;
    SignImm <= std_logic_vector(MATLAB_Function_out1);
END rtl;
```

VHDL-код тракту даних

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY Datapath IS
 PORT( clk                 : IN  std_logic;
       reset               : IN  std_logic;
       enb                 : IN  std_logic;
       control_MemtoReg    : IN  std_logic; -- ufix1
       control_ALUSrc      : IN  std_logic; -- ufix1
       control_RegDst      : IN  std_logic; -- ufix1
       control_RegWrite    : IN  std_logic; -- ufix1
       control_RegWriteIm  : IN  std_logic; -- ufix1
       control_ALUControl  : IN  std_logic_vector(1 DOWNTO 0); -- ufix2
       Instr               : IN  std_logic_vector(31 DOWNTO 0); -- uint32
       ReadData            : IN  std_logic_vector(31 DOWNTO 0); -- uint32
       op                  : OUT std_logic_vector(3 DOWNTO 0); -- ufix4
       funct               : OUT std_logic_vector(3 DOWNTO 0); -- ufix4
       PC                  : OUT std_logic_vector(31 DOWNTO 0); -- uint32
       ALUResult           : OUT std_logic_vector(31 DOWNTO 0); -- uint32
       WriteData           : OUT std_logic_vector(31 DOWNTO 0) -- uint32
       );
END Datapath;


ARCHITECTURE rtl OF Datapath IS

 -- Component Declarations
 COMPONENT Instruction_Parser
  PORT( Instr              : IN  std_logic_vector(31 DOWNTO 0); -- uint32
        op                 : OUT std_logic_vector(3 DOWNTO 0); -- ufix4
        funct              : OUT std_logic_vector(3 DOWNTO 0); -- ufix4
        rsx                : OUT std_logic_vector(3 DOWNTO 0); -- ufix4
        rtx                : OUT std_logic_vector(3 DOWNTO 0); -- ufix4
        rsy                : OUT std_logic_vector(3 DOWNTO 0); -- ufix4
        rty                : OUT std_logic_vector(3 DOWNTO 0); -- ufix4
        zr                 : OUT std_logic_vector(3 DOWNTO 0); -- ufix4
        zi                 : OUT std_logic_vector(3 DOWNTO 0); -- ufix4
        Imm                : OUT std_logic_vector(15 DOWNTO 0) -- uint16
        );
 END COMPONENT;

 COMPONENT PCCalc
  PORT( clk                : IN  std_logic;
        reset              : IN  std_logic;
        enb                : IN  std_logic;
        PC_in              : IN  std_logic_vector(31 DOWNTO 0); -- uint32
        PC                 : OUT std_logic_vector(31 DOWNTO 0) -- uint32
        );
 END COMPONENT;

 COMPONENT PCPlus4
```

```
  PORT( PC                    :  IN    std_logic_vector(31 DOWNTO 0);  -- uint32
        PCPlus4_1             :  OUT   std_logic_vector(31 DOWNTO 0)  -- uint32
        );
END COMPONENT;

COMPONENT RegDstMux
  PORT( rtx                   :  IN    std_logic_vector(3 DOWNTO 0);  -- ufix4
        RegDst                :  IN    std_logic;  -- ufix1
        rd                    :  IN    std_logic_vector(3 DOWNTO 0);  -- ufix4
        r                     :  OUT   std_logic_vector(3 DOWNTO 0)  -- ufix4
        );
END COMPONENT;

COMPONENT Sign_Extend
  PORT( Imm                   :  IN    std_logic_vector(15 DOWNTO 0);  -- uint16
        SignImm               :  OUT   std_logic_vector(31 DOWNTO 0)  -- uint32
        );
END COMPONENT;

COMPONENT ALUSrcMux
  PORT( RD2                   :  IN    std_logic_vector(31 DOWNTO 0);  -- uint32
        ALUSrc                :  IN    std_logic;  -- ufix1
        SignImm               :  IN    std_logic_vector(31 DOWNTO 0);  -- uint32
        SrcB                  :  OUT   std_logic_vector(31 DOWNTO 0)  -- uint32
        );
END COMPONENT;

COMPONENT MemtoRegMux
  PORT( RD                    :  IN    std_logic_vector(31 DOWNTO 0);  -- uint32
        MemtoReg              :  IN    std_logic;  -- ufix1
        ALUResult             :  IN    std_logic_vector(31 DOWNTO 0);  -- uint32
        Result                :  OUT   std_logic_vector(31 DOWNTO 0)  -- uint32
        );
END COMPONENT;

COMPONENT RegisterFile
  PORT( clk                   :  IN    std_logic;
        reset                 :  IN    std_logic;
        enb                   :  IN    std_logic;
        RA1                   :  IN    std_logic_vector(3 DOWNTO 0);  -- ufix4
        RA2                   :  IN    std_logic_vector(3 DOWNTO 0);  -- ufix4
        RA3                   :  IN    std_logic_vector(3 DOWNTO 0);  -- ufix4
        RA4                   :  IN    std_logic_vector(3 DOWNTO 0);  -- ufix4
        WA3                   :  IN    std_logic_vector(3 DOWNTO 0);  -- ufix4
        WA4                   :  IN    std_logic_vector(3 DOWNTO 0);  -- ufix4
        WD3                   :  IN    std_logic_vector(31 DOWNTO 0);  -- uint32
        WD4                   :  IN    std_logic_vector(31 DOWNTO 0);  -- uint32
        WE3                   :  IN    std_logic;  -- ufix1
        WE4                   :  IN    std_logic;  -- ufix1
        RD1                   :  OUT   std_logic_vector(31 DOWNTO 0);  -- uint32
        RD2                   :  OUT   std_logic_vector(31 DOWNTO 0);  -- uint32
        RD3                   :  OUT   std_logic_vector(31 DOWNTO 0);  -- uint32
        RD4                   :  OUT   std_logic_vector(31 DOWNTO 0)  -- uint32
        );
END COMPONENT;

COMPONENT ALU
  PORT( SrcX1                 :  IN    std_logic_vector(31 DOWNTO 0);  -- uint32
        ALUControl            :  IN    std_logic_vector(1 DOWNTO 0);  -- ufix2
        SrcX2                 :  IN    std_logic_vector(31 DOWNTO 0);  -- uint32
        SrcY1                 :  IN    std_logic_vector(31 DOWNTO 0);  -- uint32
        SrcY2                 :  IN    std_logic_vector(31 DOWNTO 0);  -- uint32
        ALUResult             :  OUT   std_logic_vector(31 DOWNTO 0);  -- uint32
```

```
      ALUResultIm              :  OUT  std_logic_vector(31 DOWNTO 0)  -- uint32
      );
 END COMPONENT;

 -- Component Configuration Statements
 FOR ALL : Instruction_Parser
  USE ENTITY  work.Instruction_Parser(rtl);
 FOR ALL : PCCalc
  USE ENTITY  work.PCCalc(rtl);
 FOR ALL : PCPlus4
  USE ENTITY  work.PCPlus4(rtl);
 FOR ALL : RegDstMux
  USE ENTITY  work.RegDstMux(rtl);
 FOR ALL : Sign_Extend
  USE ENTITY  work.Sign_Extend(rtl);
 FOR ALL : ALUSrcMux
  USE ENTITY  work.ALUSrcMux(rtl);
 FOR ALL : MemtoRegMux
  USE ENTITY  work.MemtoRegMux(rtl);
 FOR ALL : RegisterFile
  USE ENTITY  work.RegisterFile(rtl);

 FOR ALL : ALU
  USE ENTITY  work.ALU(rtl);

 -- Signals
 SIGNAL  Instruction_Parser_out1        : std_logic_vector(3 DOWNTO 0);  -- ufix4
 SIGNAL  Instruction_Parser_out2        : std_logic_vector(3 DOWNTO 0);  -- ufix4
 SIGNAL  Instruction_Parser_out3        : std_logic_vector(3 DOWNTO 0);  -- ufix4
 SIGNAL  Instruction_Parser_out4        : std_logic_vector(3 DOWNTO 0);  -- ufix4
 SIGNAL  Instruction_Parser_out5        : std_logic_vector(3 DOWNTO 0);  -- ufix4
 SIGNAL  Instruction_Parser_out6        : std_logic_vector(3 DOWNTO 0);  -- ufix4
 SIGNAL  Instruction_Parser_out7        : std_logic_vector(3 DOWNTO 0);  -- ufix4
 SIGNAL  Instruction_Parser_out8        : std_logic_vector(3 DOWNTO 0);  -- ufix4
 SIGNAL  Instruction_Parser_out9        : std_logic_vector(15 DOWNTO 0);  -- ufix16
 SIGNAL  PCPlus4_out1                : std_logic_vector(31 DOWNTO 0);  -- ufix32
 SIGNAL  PC_1                        : std_logic_vector(31 DOWNTO 0);  -- ufix32
 SIGNAL  RegDst                      : std_logic;  -- ufix1
 SIGNAL  RegDstMux_out1                 : std_logic_vector(3 DOWNTO 0);  -- ufix4
 SIGNAL  MemtoReg                    : std_logic;  -- ufix1
 SIGNAL  RegWrite                    : std_logic;  -- ufix1
 SIGNAL  RegWriteIm                  : std_logic;  -- ufix1
 SIGNAL  ALUControl                  : unsigned(1 DOWNTO 0);  -- ufix2
 SIGNAL  ALUControl_1                : unsigned(1 DOWNTO 0);  -- ufix2
 SIGNAL  ALUSrc                      : std_logic;  -- ufix1
 SIGNAL  y                           : std_logic_vector(31 DOWNTO 0);  -- ufix32
 SIGNAL  RD2                         : std_logic_vector(31 DOWNTO 0);  -- ufix32
 SIGNAL  ALUSrcMux_out1                 : std_logic_vector(31 DOWNTO 0);  -- ufix32
 SIGNAL  ALUResult_tmp               : std_logic_vector(31 DOWNTO 0);  -- ufix32
 SIGNAL  MemtoRegMux_out1               : std_logic_vector(31 DOWNTO 0);  -- ufix32
 SIGNAL  ALUIm                       : std_logic_vector(31 DOWNTO 0);  -- ufix32
 SIGNAL  RD1                         : std_logic_vector(31 DOWNTO 0);  -- ufix32
 SIGNAL  RD3                         : std_logic_vector(31 DOWNTO 0);  -- ufix32
 SIGNAL  RD4                         : std_logic_vector(31 DOWNTO 0);  -- ufix32

BEGIN
 u_Instruction_Parser : Instruction_Parser
  PORT MAP( Instr => Instr,  -- uint32
          op => Instruction_Parser_out1,  -- ufix4
          funct => Instruction_Parser_out2,  -- ufix4
          rsx => Instruction_Parser_out3,  -- ufix4
          rtx => Instruction_Parser_out4,  -- ufix4
          rsy => Instruction_Parser_out5,  -- ufix4
```

```
        rty => Instruction_Parser_out6,  -- ufix4
        zr => Instruction_Parser_out7,  -- ufix4
        zi => Instruction_Parser_out8,  -- ufix4
        Imm => Instruction_Parser_out9 -- uint16
        );

u_PCCalc : PCCalc
  PORT MAP( clk => clk,
        reset => reset,
        enb => enb,
        PC_in => PCPlus4_out1,  -- uint32
        PC => PC_1  -- uint32
        );
u_PCPlus4 : PCPlus4
  PORT MAP( PC => PC_1,  -- uint32
        PCPlus4_1 => PCPlus4_out1  -- uint32
        );
u_RegDstMux : RegDstMux
  PORT MAP( rtx => Instruction_Parser_out4,  -- ufix4
        RegDst => RegDst,  -- ufix1
        rd => Instruction_Parser_out7,  -- ufix4
        r => RegDstMux_out1  -- ufix4
        );
u_Sign_Extend : Sign_Extend
  PORT MAP( Imm => Instruction_Parser_out9,  -- uint16
        SignImm => y  -- uint32
        );
u_ALUSrcMux : ALUSrcMux
  PORT MAP( RD2 => RD2,  -- uint32
        ALUSrc => ALUSrc,  -- ufix1
        SignImm => y,  -- uint32
        SrcB => ALUSrcMux_out1  -- uint32
        );
u_MemtoRegMux : MemtoRegMux
  PORT MAP( RD => ReadData,  -- uint32
        MemtoReg => MemtoReg,  -- ufix1
        ALUResult => ALUResult_tmp,  -- uint32
        Result => MemtoRegMux_out1  -- uint32
        );
u_RegisterFile : RegisterFile
  PORT MAP( clk => clk,
        reset => reset,
        enb => enb,
        RA1 => Instruction_Parser_out3,  -- ufix4
        RA2 => Instruction_Parser_out4,  -- ufix4
        RA3 => Instruction_Parser_out5,  -- ufix4
        RA4 => Instruction_Parser_out6,  -- ufix4
        WA3 => RegDstMux_out1,  -- ufix4
        WA4 => Instruction_Parser_out8,  -- ufix4
        WD3 => MemtoRegMux_out1,  -- uint32
        WD4 => ALUIm,  -- uint32
        WE3 => RegWrite,  -- ufix1
        WE4 => RegWriteIm,  -- ufix1
        RD1 => RD1,  -- uint32
        RD2 => RD2,  -- uint32
        RD3 => RD3,  -- uint32
        RD4 => RD4  -- uint32
        );
u_ALU : ALU
  PORT MAP( SrcX1 => RD1,  -- uint32
        ALUControl => std_logic_vector(ALUControl_1),  -- ufix2
        SrcX2 => ALUSrcMux_out1,  -- uint32
        SrcY1 => RD3,  -- uint32
```

```
            SrcY2 => RD4,  -- uint32
            ALUResult => ALUResult_tmp,  -- uint32
            ALUResultIm => ALUIm   -- uint32
            );

        RegDst <= control_RegDst;
        MemtoReg <= control_MemtoReg;
        RegWrite <= control_RegWrite;
        RegWriteIm <= control_RegWriteIm;
        ALUControl <= unsigned(control_ALUControl);
        ALUControl_1 <= ALUControl;
        ALUSrc <= control_ALUSrc;
        op <= Instruction_Parser_out1;
        funct <= Instruction_Parser_out2;
        PC <= PCPlus4_out1;
        ALUResult <= ALUResult_tmp;
        WriteData <= RD2;

END rtl;
```

Додаток Б

VHDL-код ОЗП

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
USE work.Data_Memory_pkg.ALL;

ENTITY Data_Memory IS
 PORT( clk              : IN  std_logic;
       reset            : IN  std_logic;
       clk_enable       : IN  std_logic;
       A                : IN  std_logic_vector(31 DOWNTO 0);  -- uint32
       WD               : IN  std_logic_vector(31 DOWNTO 0);  -- uint32
       WE               : IN  std_logic;  -- ufix1
       ce_out           : OUT std_logic;
       RD               : OUT std_logic_vector(31 DOWNTO 0)  -- uint32
       );
END Data_Memory;


ARCHITECTURE rtl OF Data_Memory IS

 -- Signals
 SIGNAL enb                   : std_logic;
 SIGNAL A_unsigned            : unsigned(31 DOWNTO 0);  -- uint32
 SIGNAL Bit_Slice_out1        : unsigned(5 DOWNTO 0);  -- ufix6
 SIGNAL switch_compare_1      : std_logic;
 SIGNAL WD_unsigned           : unsigned(31 DOWNTO 0);  -- uint32
 SIGNAL Data_Type_Conversion2_out1  : unsigned(7 DOWNTO 0);  -- uint8
 SIGNAL RAM                   : vector_of_unsigned32(0 TO 63);  -- uint32 [64]
 SIGNAL Assignment_out1       : vector_of_unsigned32(0 TO 63);  -- uint32 [64]
 SIGNAL Switch_out1           : vector_of_unsigned32(0 TO 63);  -- uint32 [64]
 SIGNAL Index_Vector1_out1    : unsigned(31 DOWNTO 0);  -- uint32

BEGIN
  A_unsigned <= unsigned(A);
  Bit_Slice_out1 <= A_unsigned(7 DOWNTO 2);

  switch_compare_1 <= '1' WHEN WE > '0' ELSE
    '0';
  WD_unsigned <= unsigned(WD);
  Data_Type_Conversion2_out1 <= resize(Bit_Slice_out1, 8);
  enb <= clk_enable;

  Assignment_out1(0) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#00#, 8) ELSE
    RAM(0);
  Assignment_out1(1) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#01#, 8) ELSE
    RAM(1);
  Assignment_out1(2) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#02#, 8) ELSE
    RAM(2);
  Assignment_out1(3) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#03#, 8) ELSE
    RAM(3);
  Assignment_out1(4) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#04#, 8) ELSE
    RAM(4);
  Assignment_out1(5) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#05#, 8) ELSE
    RAM(5);
  Assignment_out1(6) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#06#, 8) ELSE
    RAM(6);
  Assignment_out1(7) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#07#, 8) ELSE
    RAM(7);
```

```
  Assignment_out1(8) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#08#, 8) ELSE
     RAM(8);
  Assignment_out1(9) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#09#, 8) ELSE
     RAM(9);
  Assignment_out1(10) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#0A#, 8) ELSE
     RAM(10);
  Assignment_out1(11) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#0B#, 8) ELSE
     RAM(11);
  Assignment_out1(12) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#0C#, 8) ELSE
     RAM(12);
  Assignment_out1(13) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#0D#, 8) ELSE
     RAM(13);
  Assignment_out1(14) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#0E#, 8) ELSE
     RAM(14);
  Assignment_out1(15) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#0F#, 8) ELSE
     RAM(15);
  Assignment_out1(16) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#10#, 8) ELSE
     RAM(16);
  Assignment_out1(17) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#11#, 8) ELSE
     RAM(17);
  Assignment_out1(18) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#12#, 8) ELSE
     RAM(18);
  Assignment_out1(19) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#13#, 8) ELSE
RAM(19);
  Assignment_out1(20) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#14#, 8) ELSE
     RAM(20);
  Assignment_out1(21) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#15#, 8) ELSE
     RAM(21);
  Assignment_out1(22) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#16#, 8) ELSE
     RAM(22);
  Assignment_out1(23) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#17#, 8) ELSE
     RAM(23);
  Assignment_out1(24) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#18#, 8) ELSE
     RAM(24);
  Assignment_out1(25) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#19#, 8) ELSE
     RAM(25);
  Assignment_out1(26) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#1A#, 8) ELSE
     RAM(26);
  Assignment_out1(27) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#1B#, 8) ELSE
     RAM(27);
  Assignment_out1(28) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#1C#, 8) ELSE
     RAM(28);
  Assignment_out1(29) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#1D#, 8) ELSE
     RAM(29);
  Assignment_out1(30) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#1E#, 8) ELSE
     RAM(30);
  Assignment_out1(31) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#1F#, 8) ELSE
     RAM(31);
  Assignment_out1(32) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#20#, 8) ELSE
     RAM(32);
  Assignment_out1(33) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#21#, 8) ELSE
     RAM(33);
  Assignment_out1(34) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#22#, 8) ELSE
     RAM(34);
  Assignment_out1(35) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#23#, 8) ELSE
     RAM(35);
  Assignment_out1(36) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#24#, 8) ELSE
     RAM(36);
  Assignment_out1(37) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#25#, 8) ELSE
     RAM(37);
  Assignment_out1(38) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#26#, 8) ELSE
     RAM(38);
  Assignment_out1(39) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#27#, 8) ELSE
```

```
    RAM(39);
Assignment_out1(40) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#28#, 8) ELSE
    RAM(40);
Assignment_out1(41) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#29#, 8) ELSE
    RAM(41);
Assignment_out1(42) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#2A#, 8) ELSE
    RAM(42);
Assignment_out1(43) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#2B#, 8) ELSE
    RAM(43);
Assignment_out1(44) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#2C#, 8) ELSE
    RAM(44);
Assignment_out1(45) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#2D#, 8) ELSE
    RAM(45);
Assignment_out1(46) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#2E#, 8) ELSE
    RAM(46);
Assignment_out1(47) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#2F#, 8) ELSE
    RAM(47);
Assignment_out1(48) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#30#, 8) ELSE
    RAM(48);
Assignment_out1(49) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#31#, 8) ELSE
    RAM(49);
Assignment_out1(50) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#32#, 8) ELSE
    RAM(50);
Assignment_out1(51) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#33#, 8) ELSE
    RAM(51);
Assignment_out1(52) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#34#, 8) ELSE
    RAM(52);
Assignment_out1(53) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#35#, 8) ELSE
    RAM(53);
Assignment_out1(54) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#36#, 8) ELSE
    RAM(54);
Assignment_out1(55) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#37#, 8) ELSE
    RAM(55);
Assignment_out1(56) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#38#, 8) ELSE
    RAM(56);
Assignment_out1(57) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#39#, 8) ELSE
    RAM(57);
Assignment_out1(58) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#3A#, 8) ELSE
    RAM(58);
Assignment_out1(59) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#3B#, 8) ELSE
    RAM(59);
Assignment_out1(60) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#3C#, 8) ELSE
    RAM(60);
Assignment_out1(61) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#3D#, 8) ELSE
    RAM(61);
Assignment_out1(62) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#3E#, 8) ELSE
    RAM(62);
Assignment_out1(63) <= WD_unsigned WHEN Data_Type_Conversion2_out1 = to_unsigned(16#3F#, 8) ELSE
    RAM(63);


        Switch_out1 <= RAM WHEN switch_compare_1 = '0' ELSE
          Assignment_out1;

        Unit_Delay_process : PROCESS (clk, reset)
        BEGIN
          IF reset = '1' THEN
            RAM <= (OTHERS => to_unsigned(0, 32));
          ELSIF clk'EVENT AND clk = '1' THEN
            IF enb = '1' THEN
              RAM <= Switch_out1;
            END IF;
          END IF;
```

```
END PROCESS Unit_Delay_process;

Index_Vector1_out1 <= RAM(0) WHEN Bit_Slice_out1 = to_unsigned(16#00#, 6) ELSE
   RAM(1) WHEN Bit_Slice_out1 = to_unsigned(16#01#, 6) ELSE
   RAM(2) WHEN Bit_Slice_out1 = to_unsigned(16#02#, 6) ELSE
   RAM(3) WHEN Bit_Slice_out1 = to_unsigned(16#03#, 6) ELSE
   RAM(4) WHEN Bit_Slice_out1 = to_unsigned(16#04#, 6) ELSE
   RAM(5) WHEN Bit_Slice_out1 = to_unsigned(16#05#, 6) ELSE
   RAM(6) WHEN Bit_Slice_out1 = to_unsigned(16#06#, 6) ELSE
   RAM(7) WHEN Bit_Slice_out1 = to_unsigned(16#07#, 6) ELSE
   RAM(8) WHEN Bit_Slice_out1 = to_unsigned(16#08#, 6) ELSE
   RAM(9) WHEN Bit_Slice_out1 = to_unsigned(16#09#, 6) ELSE
   RAM(10) WHEN Bit_Slice_out1 = to_unsigned(16#0A#, 6) ELSE
   RAM(11) WHEN Bit_Slice_out1 = to_unsigned(16#0B#, 6) ELSE
   RAM(12) WHEN Bit_Slice_out1 = to_unsigned(16#0C#, 6) ELSE
   RAM(13) WHEN Bit_Slice_out1 = to_unsigned(16#0D#, 6) ELSE
   RAM(14) WHEN Bit_Slice_out1 = to_unsigned(16#0E#, 6) ELSE
   RAM(15) WHEN Bit_Slice_out1 = to_unsigned(16#0F#, 6) ELSE
   RAM(16) WHEN Bit_Slice_out1 = to_unsigned(16#10#, 6) ELSE
   RAM(17) WHEN Bit_Slice_out1 = to_unsigned(16#11#, 6) ELSE
   RAM(18) WHEN Bit_Slice_out1 = to_unsigned(16#12#, 6) ELSE
   RAM(19) WHEN Bit_Slice_out1 = to_unsigned(16#13#, 6) ELSE
   RAM(20) WHEN Bit_Slice_out1 = to_unsigned(16#14#, 6) ELSE
   RAM(21) WHEN Bit_Slice_out1 = to_unsigned(16#15#, 6) ELSE
   RAM(22) WHEN Bit_Slice_out1 = to_unsigned(16#16#, 6) ELSE
   RAM(23) WHEN Bit_Slice_out1 = to_unsigned(16#17#, 6) ELSE
   RAM(24) WHEN Bit_Slice_out1 = to_unsigned(16#18#, 6) ELSE
   RAM(25) WHEN Bit_Slice_out1 = to_unsigned(16#19#, 6) ELSE
   RAM(26) WHEN Bit_Slice_out1 = to_unsigned(16#1A#, 6) ELSE
   RAM(27) WHEN Bit_Slice_out1 = to_unsigned(16#1B#, 6) ELSE
   RAM(28) WHEN Bit_Slice_out1 = to_unsigned(16#1C#, 6) ELSE
   RAM(29) WHEN Bit_Slice_out1 = to_unsigned(16#1D#, 6) ELSE
   RAM(30) WHEN Bit_Slice_out1 = to_unsigned(16#1E#, 6) ELSE
   RAM(31) WHEN Bit_Slice_out1 = to_unsigned(16#1F#, 6) ELSE
   RAM(32) WHEN Bit_Slice_out1 = to_unsigned(16#20#, 6) ELSE
   RAM(33) WHEN Bit_Slice_out1 = to_unsigned(16#21#, 6) ELSE
   RAM(34) WHEN Bit_Slice_out1 = to_unsigned(16#22#, 6) ELSE
   RAM(35) WHEN Bit_Slice_out1 = to_unsigned(16#23#, 6) ELSE
   RAM(36) WHEN Bit_Slice_out1 = to_unsigned(16#24#, 6) ELSE
   RAM(37) WHEN Bit_Slice_out1 = to_unsigned(16#25#, 6) ELSE
   RAM(38) WHEN Bit_Slice_out1 = to_unsigned(16#26#, 6) ELSE
   RAM(39) WHEN Bit_Slice_out1 = to_unsigned(16#27#, 6) ELSE
   RAM(40) WHEN Bit_Slice_out1 = to_unsigned(16#28#, 6) ELSE
   RAM(41) WHEN Bit_Slice_out1 = to_unsigned(16#29#, 6) ELSE
   RAM(42) WHEN Bit_Slice_out1 = to_unsigned(16#2A#, 6) ELSE
   RAM(43) WHEN Bit_Slice_out1 = to_unsigned(16#2B#, 6) ELSE
   RAM(44) WHEN Bit_Slice_out1 = to_unsigned(16#2C#, 6) ELSE
   RAM(45) WHEN Bit_Slice_out1 = to_unsigned(16#2D#, 6) ELSE
   RAM(46) WHEN Bit_Slice_out1 = to_unsigned(16#2E#, 6) ELSE
   RAM(47) WHEN Bit_Slice_out1 = to_unsigned(16#2F#, 6) ELSE
   RAM(48) WHEN Bit_Slice_out1 = to_unsigned(16#30#, 6) ELSE
   RAM(49) WHEN Bit_Slice_out1 = to_unsigned(16#31#, 6) ELSE
   RAM(50) WHEN Bit_Slice_out1 = to_unsigned(16#32#, 6) ELSE
   RAM(51) WHEN Bit_Slice_out1 = to_unsigned(16#33#, 6) ELSE
   RAM(52) WHEN Bit_Slice_out1 = to_unsigned(16#34#, 6) ELSE
   RAM(53) WHEN Bit_Slice_out1 = to_unsigned(16#35#, 6) ELSE
   RAM(54) WHEN Bit_Slice_out1 = to_unsigned(16#36#, 6) ELSE
   RAM(55) WHEN Bit_Slice_out1 = to_unsigned(16#37#, 6) ELSE
   RAM(56) WHEN Bit_Slice_out1 = to_unsigned(16#38#, 6) ELSE
   RAM(57) WHEN Bit_Slice_out1 = to_unsigned(16#39#, 6) ELSE
   RAM(58) WHEN Bit_Slice_out1 = to_unsigned(16#3A#, 6) ELSE
   RAM(59) WHEN Bit_Slice_out1 = to_unsigned(16#3B#, 6) ELSE
   RAM(60) WHEN Bit_Slice_out1 = to_unsigned(16#3C#, 6) ELSE
```

```
    RAM(61) WHEN Bit_Slice_out1 = to_unsigned(16#3D#, 6) ELSE
    RAM(62) WHEN Bit_Slice_out1 = to_unsigned(16#3E#, 6) ELSE
    RAM(63);

 RD <= std_logic_vector(Index_Vector1_out1);
 ce_out <= clk_enable;

END rtl;
```

VHDL-код ПЗП

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
USE work.Instruction_Memory_pkg.ALL;

ENTITY Instruction_Memory IS
 PORT( clk                : IN  std_logic;
       reset              : IN  std_logic;
       clk_enable         : IN  std_logic;
       PC                 : IN  std_logic_vector(31 DOWNTO 0);  -- uint32
       ce_out             : OUT std_logic;
       Instr              : OUT std_logic_vector(31 DOWNTO 0)  -- uint32
       );
END Instruction_Memory;

ARCHITECTURE rtl OF Instruction_Memory IS

 -- Signals
 SIGNAL enb                : std_logic;
 SIGNAL PC_unsigned        : unsigned(31 DOWNTO 0);       -- uint32
 SIGNAL Instr_tmp          : unsigned(31 DOWNTO 0);       -- uint32
 SIGNAL RAM                : vector_of_unsigned32(0 TO 63);  -- ufix32 [64]
 SIGNAL RAM_not_empty      : std_logic;
 SIGNAL RAM_next           : vector_of_unsigned32(0 TO 63);  -- ufix32 [64]
 SIGNAL RAM_not_empty_next : std_logic;

BEGIN
 PC_unsigned <= unsigned(PC);

 enb <= clk_enable;

 MATLAB_Function_process : PROCESS (clk, reset)
 BEGIN
  IF reset = '1' THEN
   RAM_not_empty <= '0';
   RAM <= (OTHERS => to_unsigned(0, 32));
  ELSIF clk'EVENT AND clk = '1' THEN
   IF enb = '1' THEN
    RAM <= RAM_next;
    RAM_not_empty <= RAM_not_empty_next;
   END IF;
  END IF;
 END PROCESS MATLAB_Function_process;

 MATLAB_Function_output : PROCESS (PC_unsigned, RAM, RAM_not_empty)
  VARIABLE RAM_temp : vector_of_unsigned32(0 TO 63);
  VARIABLE add_temp : unsigned(6 DOWNTO 0);
  VARIABLE sub_cast : signed(31 DOWNTO 0);
 BEGIN
  RAM_temp := RAM;
  RAM_not_empty_next <= RAM_not_empty;
  --MATLAB Function 'Instruction Memory/MATLAB Function'
```

```
            IF ( NOT RAM_not_empty) = '1' THEN
              --aligned by words
              --64 elements
              RAM_not_empty_next <= '1';
              RAM_temp(0) := to_unsigned(807403527, 32); -- addi R2=7
              RAM_temp(1) := to_unsigned(842006534, 32); -- addi R3=R2+6
              RAM_temp(2) := to_unsigned(809500677, 32);    -- addi R4=5
              RAM_temp(3) := to_unsigned(811597827, 32); -- addi R6=3
              RAM_temp(4) := to_unsigned(36988801, 32); -- Zr+Zi=(x1+x2,y1+y2) compadd
R7+R8=(R2+R3,R4+R6)
              RAM_temp(5) := to_unsigned(815792138, 32); -- addi Ra=a
              RAM_temp(6) := to_unsigned(816840709, 32); -- addi Rb=5
              RAM_temp(7) := to_unsigned(817889291, 32); -- addi Rc=b
              RAM_temp(8) := to_unsigned(818937862, 32); -- addi Rd=6
              RAM_temp(9) := to_unsigned(180150002, 32); -- Zr-Zi=(x1-x2,y1-y2) compsub Re-Rf=(Ra-Rb,Rc-Rd)
                   RAM_temp(10) := to_unsigned(805306408, 32);
              -- addi R0=28
              RAM_temp(11) := to_unsigned(544210952, 32); -- sw R7 R0+8 hex(28+8)=40+8=48dec     48/4=12
              RAM_temp(12) := to_unsigned(545259532, 32); -- sw R8 R0+b      hex(28+c)=40+12=52dec    52/4=13

              RAM_temp(13) := to_unsigned(277872652, 32);      -- lw R0+c  40+12=52 R9
            END IF;
            add_temp := resize(PC_unsigned(7 DOWNTO 2), 7) + to_unsigned(16#01#, 7);
            sub_cast := signed(resize(add_temp, 32));
            Instr_tmp <= RAM_temp(to_integer(sub_cast - 1));
            RAM_next <= RAM_temp;
          END PROCESS MATLAB_Function_output;
          Instr <= std_logic_vector(Instr_tmp);
          ce_out <= clk_enable;

        END rtl;
```

Додаток В

Особливості реалізації в ПЛІС процесора обробки комплексних чисел і стенду для його тестування

Лазоренко Д. В., Шаповалов В. О., Український державний університет науки і технологій

При розробці сучасних різноманітних пристроїв цифрової обробки інформації на основі ПЛІС все більше використовуються засоби автоматизації і високорівневого проектування, які дають можливість суттєво зменшити терміни, підвищити надійність і спростити проектування. Таки засоби розробляють найбільш потужні по випуску мікросхем фірми Intel (Altera), Amd (Xilinx), орієнтуючись на використання в нових системах мікросхем власної розробки, які часто є окремими потужними адаптуємими системами під різні застосування і, слід відміти, достатньо дорогими. Також засоби високорівневого проектування апаратури вбудовують в популярні мови високорівневого програмування. Наприклад, мову Python доповнили фреймворком MyHDL, який дозволяє провести розробку цифрового пристрою і його моделювання з використанням цієї мови. На етапі конфігурування ПЛІС необхідно компілювати код з мови Python в одну із традиційних мов проектування апаратури – VHDL або Verylog.

Суттєвим питанням при реалізації проектів в ПЛІС крім опису і моделювання (Behavioral і Post-Route Simulation) є тестування пристрою, реалізованого в ПЛІС. Таке тестування іноді потребує багато часу і додаткових ресурсів.

Розглянемо особливість процесора обробки комплексних чисел. Його відмінність від звичайного процесора полягає в тому, що при виконанні арифметичних операцій над двома комплексними операндами необхідно обробляти, по суті, чотири операнди – дві пари операндів дійсних і уявних частин комплексних чисел. З метою оптимізації швидкодії процесора в ньому використана RISC-архітектура. Реалізація в ПЛІС дає можливість виконувати таку обробку паралельно. Також в процесорі з метою розширення його функціональних можливостей передбачено виконання операцій над дійсними числами.

Для розробки процесора обробки комплексних чисел було використано програмний пакет Matlab з вбудованим інтерактивним середовищем Simulink для моделювання та аналізу динамічних систем, включаючи цифрові. Були задані команди, формати команд і даних, була розроблена функціональна схема процесора і М-функції для опису функціонування блоків (арифметико-логічного пристрою, пристрою управління, пам’яті команд, блоку регістрів, оперативної пам’яті). В пам'ять команд була записана програма і проведено моделювання. Після відладки роботи процесора в середовище Simulink було автоматично програмою HDL Coder, вбудованою в пакет MATLAB, сгенеровано VHDL код. Далі в САПР фірми Xilinx можна проводити синтез схеми, моделювання, реалізацію пристрою в ПЛІС.

Для тестування процесору, реалізованого в ПЛІС (апаратного засобу), можна використовувати великий набір чисел, який повинен охоплювати увесь діапазон можливих значень операндів в різних комбінаціях. Ці значення можуть змінюватись (задаватись) за якім ось законом, для якого відома закономірність зміни значень виходів (результатів) з метою наочної фіксації невірних значень, наприклад, можна значення різних оперендів синхронно змінювати по закону гармонічних сигналів і на виходах здобувати також значення гармонічних сигналів з новими параметрами. Ці значення операндів можна подавати на відповідні вхідні порти (якщо такі є), або вставляти їх в команди, сформувавши таким чином тестуючи програму. Можна також для тестування в тієї ж ПЛІС створити постійну пам'ять, в яку на етапі конфігурування для тестування записати точні значення результатів для певних комбінацій значень операндів. Далі при виконанні програми в процесорі слід фіксувати кількість невірних результатів.