Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра програмних систем і технологій

УДК 004.23

ВИПУСКНА КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

Тема: Розширення можливостей алгоритмічних мов низького рівня для вбудованих систем

Спеціальність 121 “Інженерія програмного забезпечення”

ПОЯСНЮВАЛЬНА ЗАПИСКА

Студент

Воронцов Віталій Ігорович

____

Науковий керівник

док. тех. наук, професор

Шевченко Віктор Леонідович

____

Допускається до захисту з питань нормоконтролю

Завідувач кафедри

док. тех. наук, професор

Бичков Олексій Сергійович

____

КИЇВ

2022

Рішенням Екзаменаційної комісії

випускна кваліфікаційна робота студента

____

захищена з оцінкою

____

Голова Екзаменаційної комісії

професор, доктор техн. наук Бондарчук Андрій

Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій
Спеціальність 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ:
Завідувач кафедри програмних систем і технологій
________________(О.С.Бичков)
“____”_________________ 20___р.

ЗАВДАННЯ
НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

__________________Воронцову Віталію Ігоровичу__________________
(прізвище, ім’я, по батькові)

1. Тема випускної кваліфікаційної магістерської роботи “Розширення можливостей алгоритмічних мов низького рівня для вбудованих систем” затверджені наказом вищого навчального закладу від „__”_____20__р. №____

2. Строк здачі студентом закінченої роботи ______________________________
3. Вихідні дані до роботи імплементація алгоритму обчислення функції sin(x) у вигляді інструкцій процесора, розробка та вбудовування інструкцій обчислень 16-бітної posit арифметики додавання, множення та ділення, опис можливостей підвищення швидкодії обчислень.
4. Зміст пояснювальної записки (перелік питань, що їх належить розробити)
1. Обґрунтувати актуальність розробки інструкцій для MIPS процесору.
2. Дослідити процесорне ядро MIPSfpga, інтерфейс інтеграції інструкцій користувача.
3. Дослідити формат чисел posit та обрати алгоритм обрахування функції sin(x).
4. Розробка модулів та вбудовування їх у процесор.
5. Результати досліджень і розробки, аналіз результатів розробленого рішення.

5. Перелік графічного матеріалу (з точним забезпеченням обов’язкових креслень)
1. Рис. 1.1 Блок-схема процесора MIPSfpga
2. Рис. 1.2 Місце CorExtend в RTL ієрархії ядра
3. Рис. 1.3 Формат сигналу UDI_ir_e
4. Рис. 1.4 точне значення та наближення для 16-розрядного квантування
5. Рис. 1.5 Різниця між точними значеннями функції та наближенням
6. Рис. 1.6 Графік наближення поліному
7. Рис. 1.7 Формат числа posit
8. Рис. 2.1 Підключення плати DE2-115 до плати Bus Blaster v3c через інтерфейс JTAG.
9. Рис. 2.2 Вікно інтерфейсу проекту в Quartus Prime
10. Рис. 2.3 Інтерфейс програматора ПЛІС в середовищі розробки.
11. Рис. 2.4 Зміна ProductId на платі Bus Blaster v3c
12. Рис. 2.5 Вигляд середовища розробки
13. Рис. 2.6 Формат інструкції UDI

1. Рис. 2.7 Початок виконання інструкції UDI2
2. Рис. 2.8 Завершення виконання інструкції UDI2
3. Рис. 2.9 Початок виконання інструкції ділення posit
4. Рис. 2.10 Завершення виконання інструкції ділення posit
5. Рис. 2.11 Початок виконання інструкції множення posit
6. Рис. 2.12 Завершення виконання інструкції множення posit
7. Рис. 2.13 Початок виконання інструкції додавання posit
8. Рис. 2.14 Завершення виконання інструкції додавання posit
9. Рис. 2.15 Розрахунок синуса 37°
10. Рис. 2.16 Схема розробленого модуля UDI з posit арифметикою

6. Консультанти з роботи із зазначенням розділів роботи, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Загальні теоретичні відомості: розділ 1	В.Л. Шевченко		
Практична частина: розділ 2	В.Л. Шевченко		

7. Дата видачі завдання ____________

Керівник ____________ В.Л. Шевченко
(підпис) (розшифровка підпису)

Завдання прийняв до виконання ____________ В.І. Воронцов
(підпис) (розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

Номер і назва етапів роботи	Термін виконання етапів роботи	Примітка
1. Вивчення та аналіз наявних інструкцій		виконано
2. Опис алгоритму обрахування функції sin(x)		виконано
3. Пошук методів підвищення точності результатів та швидкодії		виконано
4. Вивчення та аналіз чисел posit		виконано
5. Розробка модуля обрахування функції sin(x) та інтеграція у MIPS процесор		виконано
6. Розробка та вбудовування інструкцій обчислень 16-бітної posit арифметики додавання, множення та ділення в процесор		виконано
7. Аналіз результатів дослідження та розробленого рішення.		виконано
8. Оформлення пояснювальної записки		виконано

Студент – магістр ________________ В.І. Воронцов
(підпис) (розшифровка підпису)

Керівник роботи ________________ В.Л. Шевченко
(підпис) (розшифровка підпи

АНОТАЦІЯ

Випускна кваліфікаційна магістерська робота: 91 сторінок, 23 рисунки, 6 таблиць, 5 додатків, 11 джерел.

Тема: Розширення можливостей алгоритмічних мов низького рівня для вбудованих систем.

Об'єкт досліджень: процесорне ядро MIPSfpga, що є версією комерційного ядра microAptiv UP на архітектурі MIPS32.

Мета роботи: підвищення швидкодії виконання процесорних інструкцій на прикладі функції sin(x) з використанням формату чисел float32, додати арифметичні операції posit арифметики у MIPS-сумісний процесор, які можна використати для підвищення швидкодії відносно формату float32 та подальше впровадження інструкцій.

Предмет дослідження: блок CorExtend в ядрі MIPSfpga, що дозволяє додавати команди, визначені користувачем.

Результати дослідження: Розроблені та інтегровані у процесор процесорні інструкції обрахування функції sin(x) використовуючи стандарт чисел з плаваючою комою IEEE754 у форматі 32 біт на архітектурі MIPS та інструкції обчислень 16-бітної posit арифметики додавання, множення та ділення. Для розробки була використана мова опису апаратури Verilog.

Висновки: Результати роботи можна використати для модернізації існуючих інструкцій та реалізації нових процесорних інструкцій, використовуючи розроблені модулі. Пропоновані 16-бітні posit числа рекомендується використовувати у задачах, в яких не потрібна висока точність, але критичним є швидкодія.

BUS BLASTER V3C, COREXTEND, CYCLONE IV, DE2-115, FPGA, MIPS, MODELSIM, QUARTUS PRIME, UDI, POSIT, АСЕМБЛЕР, МОВИ ОПИСУ АПАРАТУРИ, ПРОЦЕСОР.

АННОТАЦИЯ

Випускна кваліфікаційна магістерська робота: 91 страниц, 23 рисунка, 6 таблиц, 5 приложений, 11 источников.

Тема: Расширение возможностей алгоритмических языков низкого уровня для встраиваемых систем.

Об'єкт исследований: процесорное ядро MIPSfpga, что является версией коммерческого ядра microAptiv UP на архитектуре MIPS32.

Цель роботи: повышение быстродействия выполнения процессорных инструкций на примере функции sin(x) с использованием формата чисел float32, добавить арифметические операции posit арифметики в MIPS-совместимый процессор, которые можно использовать для повышения быстродействия относительно формата float32 и последующего внедрения инструкций.

Предмет исследования: блок CorExtend в ядре MIPSfpga, позволяющий добавлять команды, определяемые пользователем.

Результати исследования: Разработаны и интегрированы в процессорные инструкции вычисления функции sin(x) используя стандарт чисел с плавающей запятой IEEE754 в формате 32 бит на архитектуре MIPS и инструкции вычислений 16-битной posit арифметики сложения, умножения и деления. Для разработки был использован язык описания аппаратуры Verilog.

Выводы: Результати работы могут быть использованы для модернизации существующих инструкций и реализации новых процессорных инструкций, используя разработанные модули. Предлагаемые 16-битные posit числа рекомендуется использовать в задачах, в которых не требуется высокая точность, но критическим является быстродействие.

BUS BLASTER V3C, COREXTEND, CYCLONE IV, DE2-115, FPGA, MIPS, MODELSIM, QUARTUS PRIME, UDI, POSIT, АССЕМБЛЕР, ЯЗЫКИ ОПИСАНИЯ АППАРАТУРЫ, ПРОЦЕССОР.

ANNOTATION

Graduation qualifuing master's thesis: 91 pages, 23 figures, 6 tables, 5 applications, 11 sources.

Topic: Expanding the capabilities of low-level algorithmic languages for embedded systems.

The research object: processor core MIPSfpga, which is a version of the commercial microAptiv UP core based on the MIPS32 architecture.

The purpose of work: Increase the perfomance of processor instructions on the example of function sin(x) using float32 format, add arithmetic operations of posit arithmetic in MIPS-compatible processor, which can be used to increase performance relative to float32 format and further implementation of instructions.

The research subject: a CorExtend block in the MIPSfpga core that allows to add user defined instructions.

The result of the research: Developed and integrated into processor instructions for computation the sin(x) function using the IEEE754 floating point number standard in 32-bit format on the MIPS architecture and instructions for computation 16-bit posit arithmetic for addition, multiplication and division. Verilog hardware description language was used for development.

Conclusions: The results of the work can be used to upgrade existing instructions and implement new processor instructions using the developed modules. The proposed 16-bit posit numbers are recommended for use in tasks where high precision is not required, but perfomance is critical.

BUS BLASTER V3C, COREXTEND, CYCLONE IV, DE2-115, FPGA, MIPS, MODELSIM, QUARTUS PRIME, UDI, POSIT, АСЕМБЛЕР, МОВИ ОПИСУ АПАРАТУРИ, ПРОЦЕСОР.

ЗМІСТ

ВСТУП 10

1 Загальні теоретичні відомості 13

1.1 Опис ядра MIPSfpga 13

1.2 UDI та інтерфейс CorExtend 15

1.3 Асемблер MIPS 17

1.4 Огляд мови Verilog HDL 18

1.4.1 Загальний огляд 18

1.4.2 Формати чисел 19

1.4.3 Стан гонитви, собливості присвоєння 21

1.4.4 Процес розробки на мові Verilog 24

1.5 Розрахунок функції sin(x) 27

1.6 Огляд формату Posit 29

2 Практична частина 32

2.1 Початок роботи 32

2.2 Середовище розробки на базі Visual Studio Code 35

2.3 Інтеграція sin(x) в MIPS процесор 37

2.4 Тестування інструкції sin(x) у середовищі ModelSim 39

2.5 Тестування інструкцій posit числення у середовищі ModelSim 42

2.5.1 Ділення posit чисел 42

2.5.2 Множення posit чисел 45

2.5.3 Додавання posit чисел 48

2.6 Тестування на платі DE2-115 51

2.7 Аналіз результатів 52

ВИСНОВКИ 54

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 56

ДОДАТКИ 58

Додаток А 58

Додаток Б 62

Додаток В 65
Додаток Г 67
Додаток Д 68
Додаток Е 68

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

MIPS	Microprocessor without interlocked pipelined stages	Архітектура скорочених наборів інструкцій
MIPS32	32-bit version of mips architecture	32-розрядна версія архітектури mips
UDI	User defined instructions	Інструкції визначені користувачем
MDU	Multiply / divide unit	Блок множення / ділення
MMU	Memory management unit	Блок управління пам'яттю
BIU	Buffer interface unit	Блок шинного інтерфейсу
GPR	General-purpose register	Блок регістрів загального призначення
RTL	Register transfer level	Рівень регістрових передач
EDA	Electronic design automation	Програми проєктування електронних систем
DSP	Digital signal processor	Процесор цифрової обробки сигналів
FPGA	Field-programmable gate array	Програмована користувачем вентильна матриця
ПЛІС (CPLD)	Complex programmable logic device	Програмована логічна інтегральна схема
OpenOCD	Open on-chip debugger	Відлагоджувач OpenOCD
САПР (CAD)	Computer-aided design	Система автоматизованого проектування

ВСТУП

Існують багато алгоритмів, які використовують одні й ті самі функції. Прикладом таких функцій є тригонометричні. Тригонометричні функції застосовуються в багатьох алгоритмах цифрової обробки сигналів, наприклад перетворення Хартлі, перетворення Фур'є та у комп'ютерній графіці.

Виконання даних операцій тільки при програмній реалізації відносно повільне. Якщо написати процесорні інструкції що будуть сприйматися процесором як власні, тобто будуть в конвеєрі процесора, то швидкодія даних операцій зросте. Використавши новий формат арифметичних інструкцій posit, можна покращити швидкодію арифметичних операцій.

Тому було прийнято рішення розробити апаратну реалізацію обрахування функції sin(x), додати арифметичні операції posit числення на основі сучасного комерційного процесору MIPSfpga та протестувати на платі DE2-115.

Це необхідно для задач, де обмежена потужність процесора, наприклад вбудовані системи. Це рішення не універсальне, а спеціалізоване.

На основі даної роботи можна вбудовувати нові інструкції. Вбудувавши posit арифметику, можна отримати серйозне збільшення швидкодії процесора. За аналогічною схемою до реалізації даних інструкцій можна працювати у напрямку реалізації односторонніх функцій, які необхідні для передачі інформації по відкритому каналу.

Актуальність роботи полягає в тому, що існують багато алгоритмів, які використовують одні й ті самі функції. Прикладом таких функцій є тригонометричні. Тригонометричні функції застосовуються в багатьох алгоритмах цифрової обробки сигналів, наприклад перетворення Хартлі, перетворення Фур'є та у комп'ютерній графіці.

Виконання даних операцій тільки при програмній реалізації відносно повільне. Недоліком даних рішень є те, що при програмувані використовуючи стандартні інструкції процесора швидкодія не буде найкращою. Якщо написати процесорні інструкції що будуть сприйматися процесором як власні, тобто

будуть в конвеєрі процесора, то швидкодія даних операцій зросте. Використавши новий формат арифметичних інструкцій posit, можна покращити швидкодію арифметичних операцій.

Мета наукової роботи - підвищення швидкодії виконання процесорних інструкцій на прикладі функції sin(x) з використанням формату чисел float32, додати арифметичні операції posit арифметики у MIPS-сумісний процесор, які можна використати для підвищення швидкодії відносно формату float32 та подальше впровадження інструкцій.

Об'єкт досліджень – процесорне ядро MIPSfpga, що є версією комерційного ядра microAptiv UP на архітектурі MIPS32.

Предмет досліджень - блок CorExtend в ядрі MIPSfpga, що дозволяє додавати команди, визначені користувачем.

Завдання дослідження, які дозволяють досягти поставленої мети

1. Аналіз блоку CorExtend в ядрі MIPSfpga
2. Аналіз FPGA плати Altera DE2-115
3. Дослідження формату float32
4. Дослідження методів обрахування фукнції sin(x)
5. Дослідження posit-арифметики

При виконанні досліджень та прийнятті рішень використовувались такі **методи та підходи:** теорія створення алгоритмічних мов, алгебра логіки, теорія фунцій, теорія posit числення, теорія float числення, прикладна теорія цифрових автоматів.

Наукова новизна:

1. Вперше запропонована інструкція для обрахування функції sin(x) за допомогою полінома Тейлора, яка була впроваджена у процесорне ядро MIPSfpga, та протестована у пакеті ModelSim та на FPGA платі Altera DE2-115.
2. Вперше використана posit арифметика для розробки процесорної інструкції обрахування sin(x), операції додавання, множення та ділення.

Практична цінність роботи: Отримані інструкції обрахування sin(x), операції додавання, множення та ділення використовуючи posit арифметику. Результати роботи можна використати для подальшого покращення інструкцій та реалізації нових процесорних інструкцій.

Стислий опис результатів дослідження: В ході роботи було розглянуто можливості та особливості роботи плати Altera DE2-115, Bus Blaster v3c, робота процесора MIPSfpga, інтерфейс CorExtend та способи програмування FPGA Cyclone IV.

Розроблено процесорну інструкцію обрахування функції sin(x) використовуючи стандарт чисел з плаваючою комою IEEE754 у форматі 32 біт на архітектурі MIPS. Додані інструкції обчислень 16-бітної posit арифметики додавання, множення та ділення. Для розробки була використана мова опису апаратури Verilog. Інструкції були протестовані і промодельовані у середовищі ModelSim, протестовані на платі Altera DE2-115.

Апробація результатів випускної кваліфікаційної роботи: результати дослідження представлені та схвалені на другому турі Всеукраїнського конкурсу студентських наукових робіт у 2020/2021 н.р. зі спеціальності «Комп'ютерна інженерія». Робота посіла перше місце (додаток Г).

Впровадження результатів дослідження засвідчене відповідним актом (додаток Д).

Публікації:

1) Воронцов В.І., Шевченко В.Л., Розширення системи команд MIPS асемблера (синус) - 7-ма Східно-Европейська конференція Математичні та програмні технології Internet of Everything, м.Київ, 2020. – с. 66.
2) Воронцов В.І., Шевченко В.Л., Інтеграція функції синуса в MIPS процесор - 7-ма Східно-Европейська конференція Математичні та програмні технології Internet of Everything, м.Київ, 2020. – с. 67.

РОЗДІЛ 1 ЗАГАЛЬНІ ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1 Опис ядра MIPSfpga

MIPS (Microprocessor without Interlocked Pipelined Stages) - це архітектура скорочених наборів інструкцій (RISC), розроблена компанією MIPS Computer Systems, тепер MIPS Technologies, що базується в Сполучених Штатах.

Існує кілька версій MIPS: включаючи MIPS I, II, III, IV і V; а також п'ять випусків MIPS32/64 (для 32- та 64-розрядної реалізації відповідно). Ранні архітектури MIPS були лише 32-бітовими; 64-розрядні версії були розроблені пізніше. Станом на листопад 2019 року поточна версія MIPS - це MIPS32/64, випуск 6. MIPS32 і MIPS64 визначають як набір регістрів управління, так і набір команд.

Ядро MIPSfpga є версією комерційного ядра microAptiv UP. Процесори microAptiv знаходять широке застосування в комерційних виробах, включаючи пристрої промислової і офісної автоматизації, автомобільної та споживчої електроніки, засоби бездротового зв'язку. Ядро MIPSfpga було спроектовано з використанням однієї з мов опису апаратури - Verilog. Воно являє собою конфігуроване (програмоване) ядро (soft core), оскільки воно описано на мові Verilog, а не фізично реалізовано у вигляді мікросхеми (або її частини).

Опис ядра MIPSfpga складається з приблизно 12 тисяч рядків мови Verilog.

Ядро MIPSfpga ліцензовано виключно для некомерційного освітнього використання. На рисунку 1.1 зображена блок-схема процесора MIPSfpga

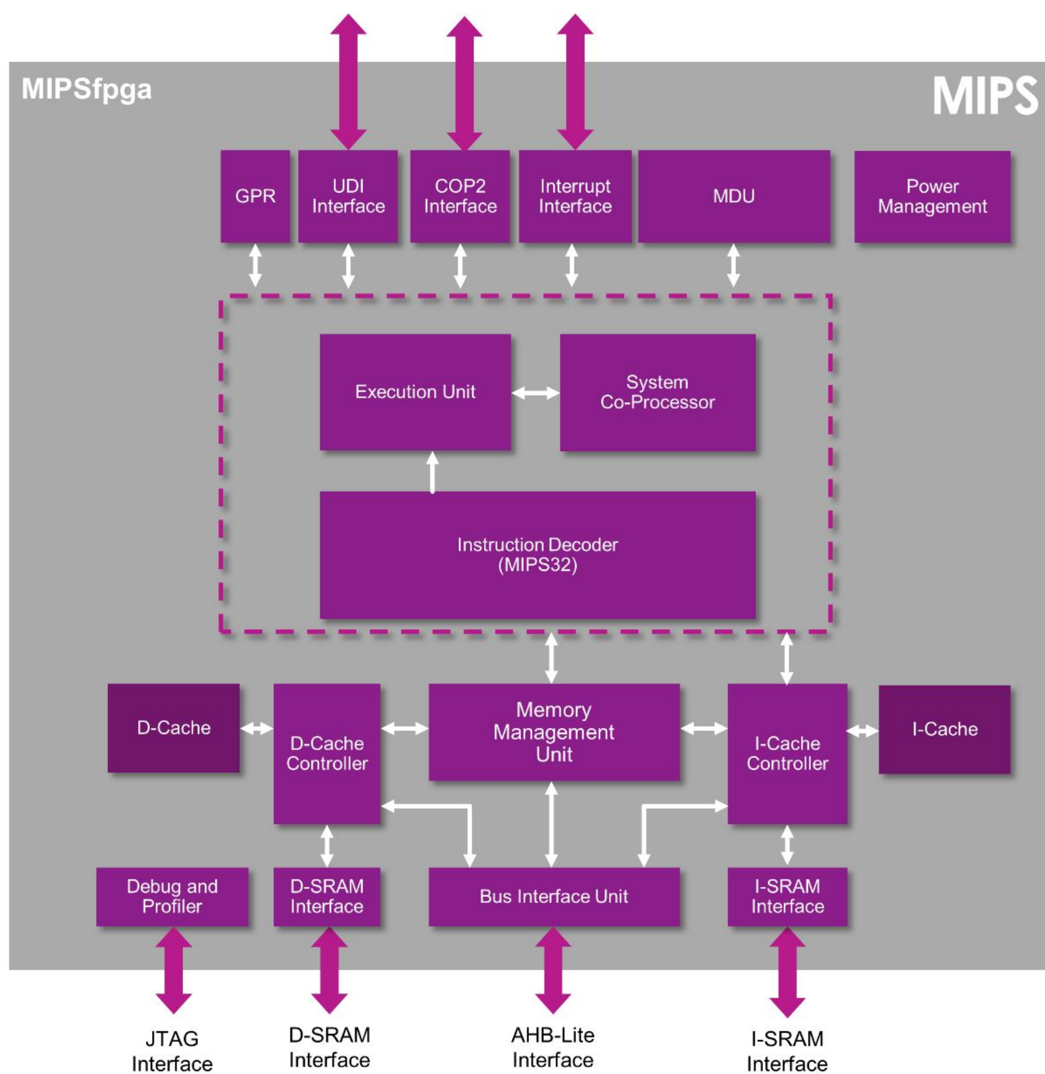


Рис. 1.1 Блок-схема процесора MIPSfpga

1.2 UDI та інтерфейс CorExtend

UDI (user defined instructions) дозволяє створювати власні процесорні інструкції. Ці інструкції конвеєр сприймає як свої власні. Це дозволяє пришвидшити швидкодію, наприклад виконавши певні дії за один такт замість кількох тактів. Додати їх можна за допомоги інтерфейсу CorExtend.

На рис. 1.2 зображено Місце CorExtend в RTL ієрархії ядра m14k microAptiv.

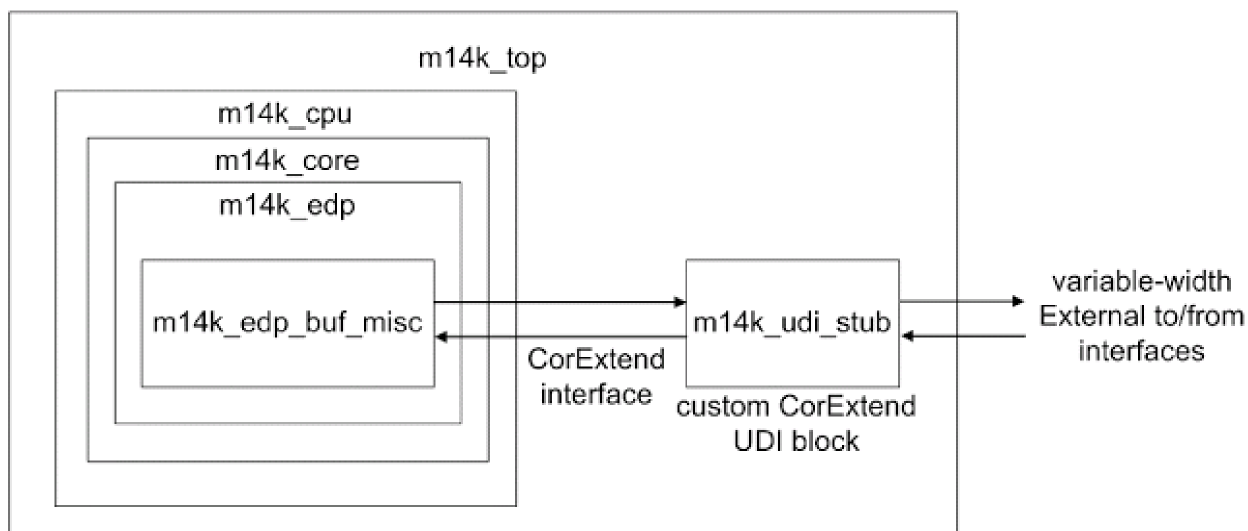


Рис. 1.2 Місце CorExtend в RTL ієрархії ядра

Всі сигнали рівня m14k_cpu, включаючи CorExtend UDI, описані в документі MIPS32 microAptiv UP Processor Core Family Integrators Guide[4] (Таблиця 2.3 Signal Descriptions for m14k cpu Level). Краще дивитися там, але для наочності нижче приведена витримка звідти виключно з сигналами CorExtend UDI, що нас цікавлять на даний момент. У табл. 1.1 відображений скорочений список сигналів CorExtend UDI.

Таблиця 1.1

Скорочений список сигналів CorExtend UDI

Назва сигналу	**Тип**	**Опис**
UDI_ir_e[31:0]	Out	Повне слово інструкції. Хоча модуль отримує rs і rt операнди, передається

		інструкція цілком, щоб була можливість передавати дані в полях адрес операндів. Зверніть увагу, що той, хто буде реалізовувати власний UDI блок, повинен самостійно декодувати поля Opcode і Function field.
UDI_rs_e[31:0]	Out	Операнд rs.
UDI_rt_e[31:0]	Out	Операнд rt.
UDI_rd_m[31:0]	In	32-бітний результат виконаної інструкції, доступний на стадії M (Memory fetch).
UDI_wrreg_e[4:0]	In	Адреса регістра для запису результату виконання user-defined інструкції. Також передається в MPC.
UDI_stall_m	In	Сигналізує, що UDI блок виконує багатотактову інструкцію і повинен зупинити конвеєр перед записом в регістр загального призначення. повинен бути встановлений в 0 для однотактових інструкцій. Сигнал стадії M.

На рис. 1.3 зображений формат сигналу UDI_ir_e. Номер регістру Rd вказується в нашому випадку в бітах [15-11]

31 26	25 21	20 16	15 6	5 0
Major opcode 011100	R_S	R_T	user-interpretable	Function field 01xxxx

Рис. 1.3 Формат сигналу UDI_ir_e

Для того щоб процесор сприймав UDI необхідно додати код (на мові Verilog), який описаний у практичній частині роботи.

1.3 Асемблер MIPS

Мова асемблера - це зручне для сприйняття людиною представлення рідної мови комп'ютера. Кожна інструкція мови асемблера задає операцію, яку необхідно виконати, а також операнди, які будуть використані під час виконання

Найбільш часта операція, яка виконується комп'ютером, - це сума. У наступному прикладі показаний код, який складає змінні b і c і записує результат в змінну a. Кожен приклад спочатку написаний на мові високого рівня (використовується синтаксис C, C ++ і Java), а потім переписаний на мові асемблера MIPS. Перша частина інструкції асемблера, add, називається мнемонікою і визначає, яку операцію потрібно виконати. Операція здійснюється над b і c, операндами-джерелами, а результат записується в a, операнд-призначення.

Код на мові високого рівня:

```
a = b + c;
```

Код на мові асемблера MIPS:

```
add $s0, $s1, $s2  # a = b + c
```

1.4 Огляд мови Verilog HDL

1.4.1 Загальний огляд

Verilog був розроблений, щоб спростити процес і зробити HDL більш надійним і гнучким. Сьогодні Verilog є найпопулярнішим HDL, який використовується і практикується у всій напівпровідникової промисловості.

HDL був розроблений для покращення процесу проектування, дозволяючи інженерам описати потрібну функціональність апаратного забезпечення та дозволити інструментам автоматизації перетворити цю поведінку в фактичні апаратні елементи, такі як комбінаційні схеми та послідовна логіка. Комбінаційні схеми визначаються як незалежні від часу схеми, які не залежать від попередніх входів для створення будь-якого виходу. Послідовна логіка залежить від тактових циклів, від поточних, а також минулих вхідних даних для створення будь-якого виходу.

Verilog підтримує дизайн на багатьох рівнях абстракції, наприклад:

- Поведінковий рівень;
- Рівень регістрових передач;
- Рівень логічних вентилів.

Поведінковий рівень описує поведінку системи за допомогою паралельних алгоритмів. Кожен алгоритм є послідовним, що означає, що він складається з набору виконуваних інструкцій одна за одною. Основними елементами є функції, завдання та блоки.

Конструкції, що використовують рівень регістрових передач, визначають характеристики схеми за допомогою операцій і передачі даних між регістрами.

Характеристики системи описуються логічними зв'язками та їхніми властивостями часу на рівні логічних вентилів. Усі сигнали є дискретними. Вони можуть мати лише певні логічні значення (`0', `1', `X', `Z`). Операції, які можна використовувати, є попередньо визначеними логічними примітивами (базовими вентилями). Моделювання рівня логічних вентилів може бути не

найкращою ідеєю для логічного проектування. Код цього рівня генерується за допомогою таких інструментів, як інструменти синтезу, а його список зв'язків використовується для моделювання на рівні логічних елементів.

Лексичні конструкції у Verilog подібні до мови програмування C. Вихідні текстові файли мови Verilog – це потік лексичних лексем. Лексична лексема може складатися з одного або кількох символів, і кожен окремий символ міститься саме в одному лексемі. Маркерами можуть бути ключові слова, коментарі, числа, пробіли або рядки. Усі рядки мають закінчуватися крапкою з комою. Дана мова є чутливою до регістру, усі ключові слова пишуться з нижнього регістру.

Мова підтримує коментарі. Існує два типи представлення коментарів, а саме:

- Однорядкові коментарі: починаються з маркера // і закінчуються закінченням строки;
- Багаторядкові коментарі: починаються з маркера /* і закінчуються лексемою */.

1.4.2 Формати чисел

Ми можемо вказати постійні числа в двійковому, десятковому, шістнадцятковому або вісімковому форматі. Від'ємні числа представлені у вигляді доповняльного коду. Знак питання '?' є альтернативою Verilog для символу z, якщо він використовується в числі. Символ підкреслення '_' допустимий у будь-якому місці числа, але він ігнорується як перший символ.

Verilog HDL дозволяє вказувати цілі числа як:

- Розмірні чи нерозмірні числа (розмір без розміру становить 32 біти).
- В основі десяткового, шістнадцяткового, двійкового чи вісімкового.
- Основні та шістнадцяткові цифри (a,b,c,d) не чутливі до регістру.
- Допускаються пробіли між основою, розміром і значенням.

Синтаксис виглядає наступним чином:

<розмір>'<основа><значення> , наприклад 8’b00001111.

Verilog підтримує числа з плаваючою комою, а саме константи та змінні. Мова перетворює дійсні числа в цілі шляхом округлення. Дійсні числа округляються до найближчого цілого числа при приписуванні цілого числа.

Дійсні числа можуть бути вказані як у десятковому, так і в науковому вигляді:

1) < значення >.< значення >
2) < мантиса >E< показник >

Verilog підтримує знакові та беззнакові типи чисел, але з певними обмеженнями. У мові C у нас немає типів int та uint, щоб визначити, чи є число ціле число зі знаком чи ціле без знака. Будь-яке число, яке не має префікса від’ємного знака, є додатним. Або непрямим способом буде беззнаковим. Від’ємні числа можна вказати, поставивши знак мінус перед розміром постійного числа, таким чином стаючи числами зі знаком. Verilog внутрішньо представляє негативні числа у форматі доповнювального коду. Для знакової арифметики можна додати необов’язковий специфікатор зі знаком.

Від’ємні числа вказуються шляхом розміщення знака мінус (-) перед розміром числа. Знак мінус між базовим форматом і числом заборонено.

1.4.3 Стан гонитви, собливості присвоєння

Стандарт IEEE Verilog [11] визначає: які інструкції мають гарантований порядок виконання, а які оператори не мають гарантованого порядку виконання. Умова перегонів (стан гонитви, race conditions) Verilog виникає, коли два або більше операторів, виконання яких заплановано на одному і тому самому етапі моделювання, дадуть різні результати, коли порядок виконання операторів змінюється, як це дозволяє стандарт IEEE Verilog.

Щоб уникнути стану гонитви, важливо розуміти планування блокуючих і неблокуючих присвоєнь Verilog.

Оператором блокуючого присвоєння є знак рівності ‘=’. Призначення блокування отримує свою назву, тому що блокуюче призначення має оцінити аргументи RHS і завершити призначення без переривання будь-яким іншим оператором Verilog. Кажуть, що присвоєння "блокує" інші призначення, доки поточне присвоєння не буде завершено. Єдиним винятком є присвоєння блокування із затримками часу на виразі з лівого боку оператора блокування, що вважається поганим стилем кодування.

Виконання блокуючих призначень можна розглядати як одноетапний процес:

Оцініть рівняння правого боку та оновіть вираз лівої сторони блокування без переривання будь-яким іншим оператором Verilog.

Блокуюче присвоєння "блокує" кінцеві присвоєння в тому самому блоці завжди від виконання до завершення поточного присвоєння.

Проблема з блокуванням призначень присвоєння, коли змінна правої стороти одного присвоєння в одному процедурному блоці також є змінною лівої сторони іншого присвоювання в іншому процедурному блоці, і обидва рівняння заплановано виконувати на тому самому кроці часу моделювання, наприклад, на тому самому фронті тактового сигналу. Якщо завдання блокування не впорядковано належним чином, може виникнути стан гонитви.

Якщо блокування призначено для виконання на одному і тому самому етапі часу, порядок виконання невідомий.

Для ілюстрації можна використати наступний код:

```
module module1 (y1, y2, clk, rst);
output y1, y2;
input clk, rst;
reg    y1, y2;

always @(posedge clk or posedge rst)
if (rst) y1 = 0;
else   y1 = y2;

always @(posedge clk or posedge rst)
if (rst) y2 = 1;
else   y2 = y1;
endmodule
```

Лістинг 1. Приклад блокуючих присвоєнь

Відповідно до стандарту IEEE Verilog, два блоки завжди можна запланувати в будь-якому порядку. Якщо перший блок завжди виконується першим після скидання, обидва y1 і y2 набудуть значення 1. Якщо другий блок завжди виконується першим після скидання, обидва блоки y1 і y2 набудуть значення 0. Це чітко представляє стан гонитви у Verilog.

Неблокуючий оператор присвоєння такий самий, як оператор «менше або дорівнює» “<=". Неблокуюче присвоєння отримує свою назву, тому що присвоювання оцінює вираз правої частини неблокуючого оператора на початку тимчасового кроку і планує оновлення лівої частини на кінець часового кроку. Між оцінкою виразу правої частини та оновленням виразу лівої частини можна оцінювати й оновлювати інші оператори Verilog, а також можна оцінювати вираз правої частини інших неблокуючих присвоєнь Verilog та планувати оновлення лівої частини операторів. Неблокуюче присвоєння не блокує обрахування інших операторів Verilog.

Виконання неблокуючих присвоєнь можна розглядати як двоетапний процес:

1) Оцініть праву частину виразу неблокуючих операторів на початку кроку часу.
2) Оновіть ліву частину виразу неблокуючих операторів наприкінці часового кроку.

Неблокуючі присвоєння виконуються лише для реєстрації типів даних і тому дозволені всередині процедурних блоків, таких як початкові блоки та “always” блоки.

Для ілюстрації можна використати наступний код:

```
module module2 (y1, y2, clk, rst);
output y1, y2;
input clk, rst;
reg    y1, y2;

always @(posedge clk or posedge rst)
if (rst) y1 <= 0;
else    y1 <= y2;

always @(posedge clk or posedge rst)
if (rst) y2 <= 1;
else    y2 <= y1;
endmodule
```

Лістинг 2. Приклад неблокуючих присвоєнь

Відповідно до стандарту IEEE Verilog, два блоки завжди можна запланувати в будь-якому порядку. Незалежно від того, який блок завжди починається першим після скидання, обидва неблокуючі вирази правої частини буде оцінено на початку кроку часу, а потім обидві неблокуючі змінні лівої частини виразу оновлюватися в кінці того ж етапу часу. З точки зору користувачів, виконання цих двох неблокуючих оператори відбуваються паралельно.

1.4.4 Процес розробки на мові Verilog

Через складну природу сучасних мікросхем неможливо створити щось з нуля, і в багатьох випадках багато компонентів будуть використані повторно.

Наприклад, компанія A потребує певного модуля для взаємодії з іншими модулями в автомобілі. Вони можуть або придбати конструкцію даного модуля в іншої компанії, щоб заощадити час і зусилля, або витратити ресурси на її створення.

Непрактично спроектувати таку систему з основних конструкційних блоків, таких як тригери та CMOS-транзистори.

Замість цього розробляється поведінковий опис для аналізу дизайну з точки зору функціональності, продуктивності та інших проблем високого рівня за допомогою мови опису обладнання, наприклад Verilog або VHDL.

Зазвичай це робить цифровий дизайнер і схожий на програміста високого рівня, який володіє навичками цифрової електроніки. Коли проект RTL буде готовий, його потрібно перевірити на функціональну правильність.

Наприклад, очікується, що процесор DSP видаватиме транзакції шини з отриманням інструкцій з пам'яті і знає, що це відбудеться, як очікувалося. На цьому етапі потрібна функціональна перевірка, яка виконується за допомогою симуляторів EDA, які можуть моделювати проект і застосовувати до нього різні стимули. Це робота інженера з верифікації.

Щоб заощадити час і досягти функціонального завершення, як проекти, так і групи верифікації працюють паралельно, де дизайнери випускають версію RTL. Команда верифікації розробляє середовище тестового стенду та тестові випадки, щоб перевірити функціональність цієї версії RTL.

Якщо будь-який з цих тестів не вдається, це може свідчити про проблему з дизайном, і на цьому елементі дизайну буде виникнути помилка. Цю помилку потрібно буде виправити в наступній версії випуску RTL від команди дизайнерів. Цей процес триває до тих пір, поки не з’явиться хороший рівень впевненості у функціональній коректності конструкції.

На наступному етапі цей дизайн перетворюють на апаратну схему з реальними елементами, такими як комбінаційні схеми та тригери. Цей етап називається синтезом.

Інструменти логічного синтезу дозволяють конвертувати опис RTL у HDL у список зв’язків схем на рівні логічних вентилів. Цей список мереж є описом схеми в термінах вентилів і з'єднань між ними.

Інструменти логічного синтезу забезпечують відповідність списку зв’язків специфікаціям часу, площі та потужності. Як правило, вони мають доступ до різних технологічних вузлів процесів і бібліотек цифрових елементів і можуть робити інтелектуальні розрахунки, щоб відповідати всім цим різним критеріям.

Ці бібліотеки отримані від напівпровідникових фабрик, які надають характеристики даних для різних компонентів, таких як час наростання або спаду для тригерів, час введення-виведення для комбінаційних вентилів тощо.

Список зв’язків схем на рівні шлюза перевіряється на логічну еквівалентність RTL. Іноді перевірка на рівні шлюза виконується, коли перевірка певних елементів виконується ще раз, різниця в тому, що цього разу це відбувається на рівні воріт і нижчому рівні абстракції. Час симуляції, як правило, повільніший через величезну кількість елементів, що беруть участь у проектуванні, і інформацію про затримку.

Потім список зв’язків схем вводиться до фізичного процесу проектування, де виконуються розміщення і трасування за допомогою інструментів EDA. Це вибере та помістить стандартні комірки в рядки, визначить введення та виведення, створить різні металеві шари та розмістить буфери.

Після завершення цього процесу створюється макет, який зазвичай відправляється на виготовлення. Цим етапом зазвичай займається команда фізичного проектування, яка добре знайома з технологічним вузлом і деталями фізичної реалізації.

Зразок мікросхеми буде виготовлено тією ж компанією з виробництва напівпровідників або надіслано сторонній стороні, наприклад всім відомий завод TSMC. Цей зразок тепер проходить процес перевірки після кремнію, коли

інша команда інженерів запускає різні шаблони тестування. Це складніше налагодити під час перевірки після реалізації в кремнії, ніж перед цим шляхом, просто тому, що рівень видимості внутрішніх вузлів чіпа різко знижується. Через те, що мільйон тактових циклів завершився б за секунду, відстеження до точного часу помилки займе багато часу. Якщо на цьому етапі будуть виявлені якісь реальні проблеми або помилки в дизайні, це потрібно буде виправити в RTL, повторно перевірити, а також виконати всі наступні кроки.

Незважаючи на те, що в потоці проектування є кілька етапів, велика частина проектної діяльності зазвичай зосереджена на оптимізації та перевірці опису RTL схеми. Важливо відзначити, що хоча інструменти EDA доступні для автоматизації процесів, неправильне використання призведе до неефективного проектування. Отже, дизайнер повинен робити свідомий вибір під час процесу проектування.

1.5 Розрахунок функції sin(x)

У розрахунку функції sin(x) є деяка особливість. Аргумент, як правило, визначається лише для першого квадранта, тобто $0 \le x \le \pi/2$, а інші значення квадрантів обчислюються через

$$\sin(x) = -\sin(-x)$$

$$sin(x) = sin(\pi/2 - x)$$

Рис. 1.4 показує точне значення та наближення для 16-розрядного квантування

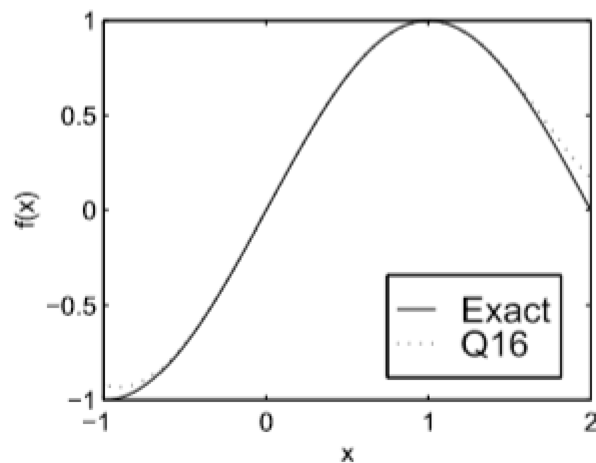


Рис. 1.4 точне значення та наближення для 16-розрядного квантування

На Рис. 1.5 відображається похибка, тобто різниця між точними значеннями функції та наближенням

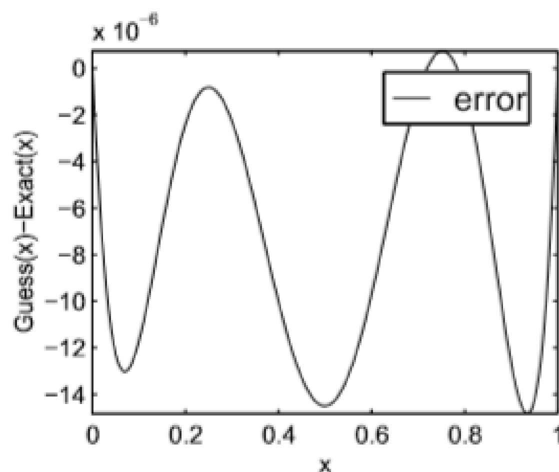


Рис. 1.5 Різниця між точними значеннями функції та наближенням

Є декілька способів обчислення синуса. Наприклад, поліном Чебишева або поліном Тейлора. Проблема поліному Чебишева полягає в тому, що він визначений лише для діапазону x ∈ [−1,1]. [2]

В той же час, у полінома Тейлора такої проблеми немає. Тому для обрахування синуса було обрано даний метод:

$$sin(x) = \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k+1)!} x^{2k+1}$$

При цьому використали 4 доданки:

$$\sin(x) \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!}$$

На рис. 1.6 зображений Графік наближення поліному

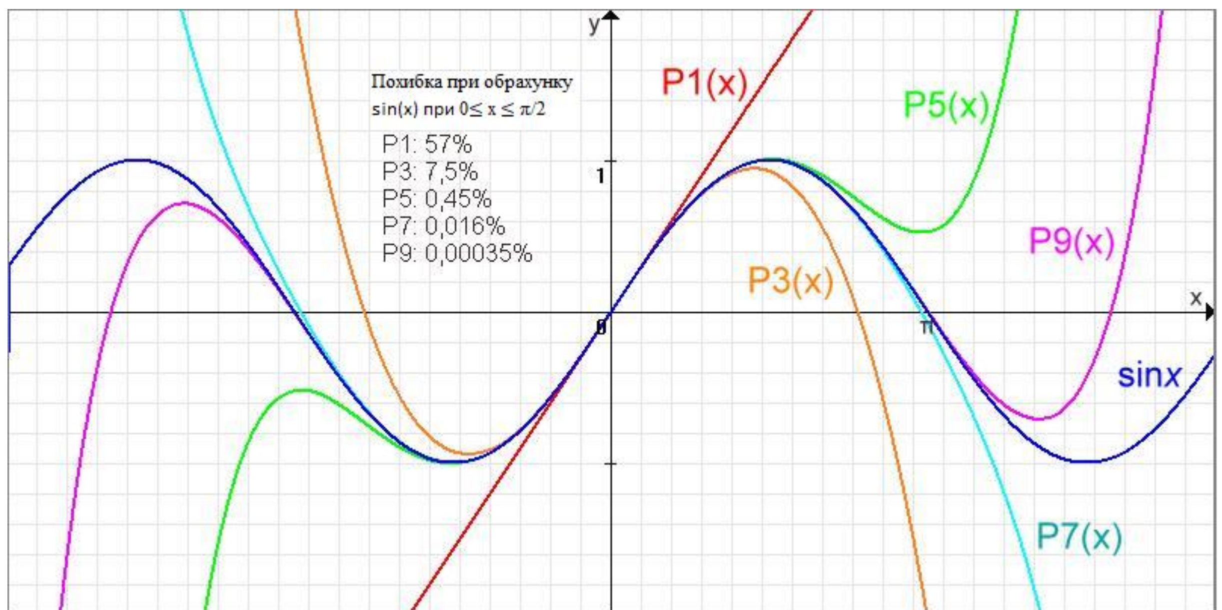


Рис. 1.6 Графік наближення поліному

При необхідності завжди можна збільшити кількість доданків у результуючому коді.

1.6 Огляд формату Posit

Основна відмінність у формуванні числа posit перед float полягає у тому, що число posit має змінну довжину дробової частини (мантиси), а також з'являється додаткове поле режиму числа, яке теж має змінну довжину. Лише поля знаку та експоненти мають фіксоване значення, визначається в залежності від реалізації.

MSB (старший розряд) LSB (молодший розряд)

Sign (S, знак)	**Regime (R,режим)**	**Exponent (E,експонента)**	**Fraction (F,дробова частина)**
1 біт	rs бітів	es бітів	n-rs-es-1 бітів

Рис. 1.7 Формат числа posit

Стандарт posit диктує наступний розмір поля експоненти:

$$es = \log_2 \frac{n}{2^3},$$

де n – кількість бітів.

Мінімальний розмір числа у форматі posit складає 8 біт.

Мантиса має ті ж властивості, що й в типі float, проте можливо що біти режиму заповнять весь допустимий простір і буде отримано пусту мантису, тобто її значення дорівнює 0. Також існує ціла частина – так званий віртуальний біт, який завжди рівний одиниці.

Оскільки при розмірі формату 8 біт, експонента буде нульовою, за множник будуть відповідати біти режиму. Біти режиму - це послідовність однакових знаків, яка закінчується знаком протилежної величини. Таким чином можна зрозуміти де закінчилось поле режиму та почалась експонента. Знаючи фіксовану довжину експоненти, можна дізнатись де починається дробова частина.

У табл. 1.2 відображена розшифровка бітів режиму в число k.

Таблиця 1.2

Розшифровка бітів режиму в число k

R	**k**	**R**	**k**
10	0	01	-1
110	1	001	-2
1110	2	0001	-3

Поле режиму дає досить специфічне значення для числа загалом. Саме режим визначає додатній або від'ємний степінь числа, а експонента грає менш впливову роль, регулюючи певний діапазон значень, обмежений її бітами.

Для визначення значення режиму використовується наступна формула:

$$Regime = useed^{k}$$

$$useed = 2^{2^{es}}$$

В даному випадку кількість біт експоненти визначає базис режиму, який підноситься у степінь k, що отриманий з режимних бітів. Приклад формування числа k зображено в таблиці 1.2.

Залежність базису режиму (useed) від кількості біт експоненти показано в таблиці 1.3.

Таблиця 1.3

Залежність базису (useed) від кількості біт експоненти (es)

es	0	1	2	3	4
useed	2	4	16	256	65536

Таким чином, posit-число можна порахувати за наступною формулою:

$$x = (-1)^{S} \times useed^{k} \times 2^{e} \times (1 + \sum_{i=1}^{fn-1} b_{fn-1-i} 2^{-i}),$$

де fn – кількість бітів дробової частини, b – певний біт дробової частини, e – значення експоненти.

В posit арифметиці існує лише один режим округлення: округлити до найближчого, прив’язати до найближчого парного, таке, яким є за замовчуванням режим округлення для IEEE 754. Якщо програміст або користувач відчуває потребу в інших трьох режимах, які підтримуються float (округлення вниз, округлення вгору, округлення до нуля), це означає, що програма вимагає числа valid (режим точного числа у стандарті posit, являє собою діапазон між 2 значеннями), а не posit або, можливо, просто правильного інструменту налагодження.

Аналіз типу posit:

До явних переваг можна віднести:

- Динамічний режим полів, який дозволяє досягнути кращої точності в певному діапазоні;
- простота апаратної реалізації;
- відсутність понять переповнення або втрати значимості;
- мала кількість виключень, а саме NaN (не є числом), нескінченність, underflow;
- переповнення ніколи не відбувається, окрім випадку 1/0;
- на відміну від float, має лише 1 значення NaN, коли у float їх кількість може досягати 9 квадрильйонів;
- підтримка доповняльного коду, що дозволяє виконувати порівняння чисел формату posit як знакові цілі числа;
- єдина система правил округлення.

Недоліки:

- в деяких випадках доведеться виконувати додаткові операції віднімання для порівняння чисел;
- на сьогодняшній день немає апаратної реалізації у комерційних продуктах;
- формат числа недостатньо досліджений відносно float чисел.

РОЗДІЛ 2 ПРАКТИЧНА ЧАСТИНА

2.1 Початок роботи

Було обрано плату Altera DE2-115 через те що вона має достатній функціонал для виконання поставленої задачі та є в наявності. Розміщений на ній FPGA Cyclone IV має достатню кількість логічних елементів, а саме 115000.

На рис. 2.1 зображено підключення плати DE2-115 до плати Bus Blaster v3c через інтерфейс JTAG.

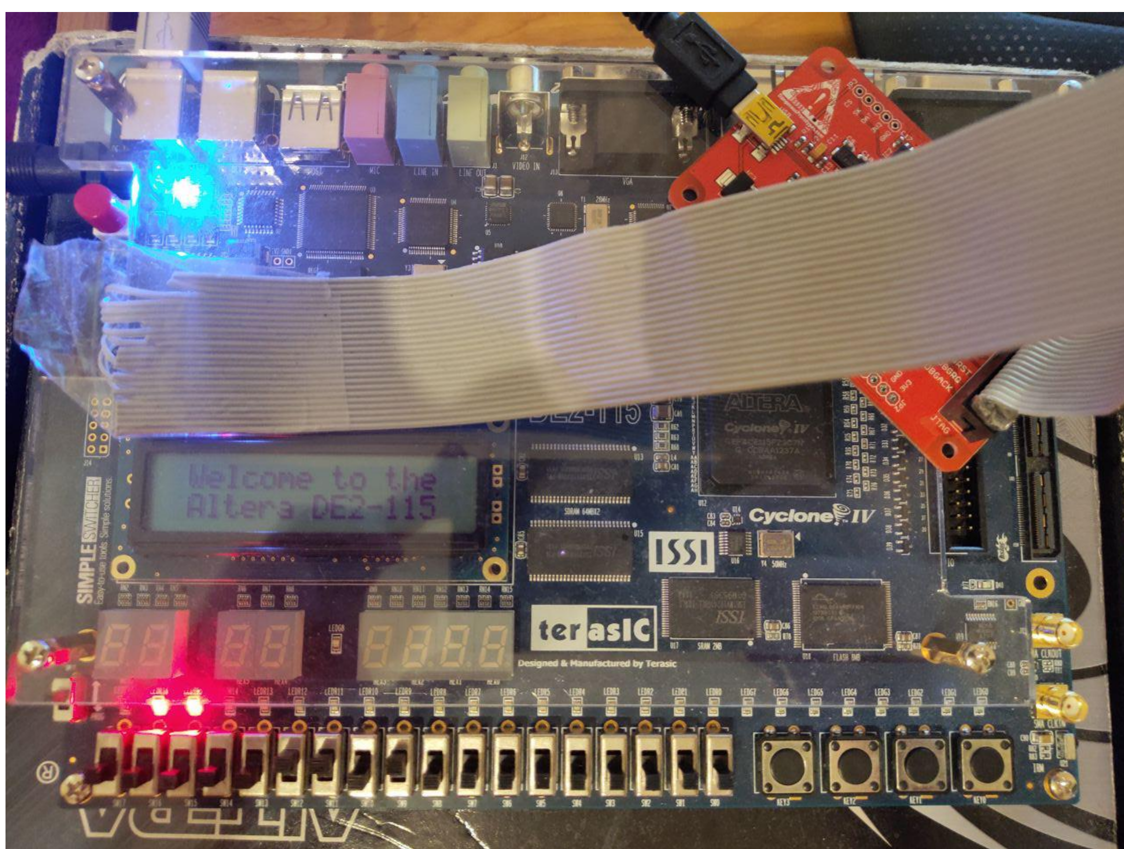


Рис. 2.1 Підключення плати DE2-115 до плати Bus Blaster v3c через інтерфейс JTAG.

Для программування FPGA Cyclone IV (рівень регістрової передачі) використовується Quartus Prime. Bus Blaster v3c використовується для програмування та налагоджування мовою асемблера або C за допомоги ПЗ OpenOCD та Codescape SDK.

FPGA програмується в середовищі Quartus Prime. На рис. 2.2 зображено вікно інтерфейсу проекту в Quartus

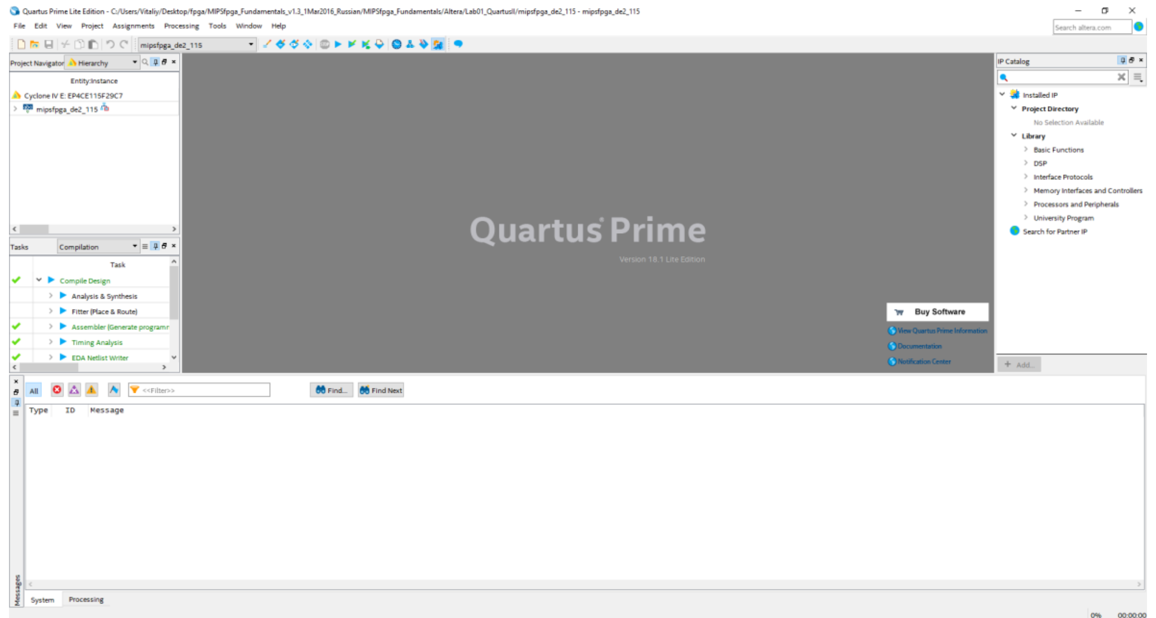


Рис. 2.2 Вікно інтерфейсу проекту в Quartus Prime

Коли проект скомпільовано, необхідно зайти у вікно програматора та почати процес програмування на рівні RTL. На Рис. 2.3 зображено інтерфейс програматора ПЛІС в середовищі розробки.

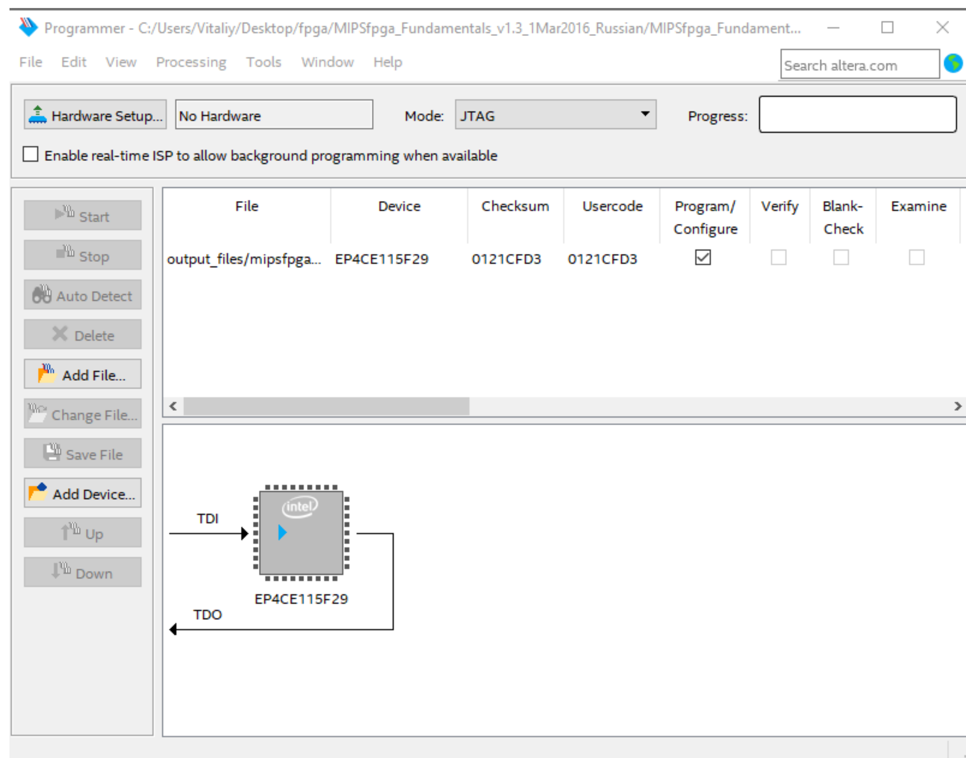


Рис. 2.3 Інтерфейс програматора ПЛІС в середовищі розробки.

На момент компіляції в середовищі розробки встановлюється початкова программа машинним кодом, що буде виконуватись при перезапуску ядра MIPS (Key0 на платі). Файл називається ram_reset_init.ini. Приклад його вмісту буде наведений у практичній частині.

При прошивці FPGA, можна змінювати вміст пам'яті, відповідно програму, що виконується, за допомогою Bus Blaster v3c та відповідного ПЗ. Для використання Bus Blaster v3c необхідно змінити його ProductId в EEPROM пам'яті.

Використовуючи ПЗ FT_Prog змінимо вищезгадане значення з 6010 на 7780 та максимальний струм з 100мА на 500мА. Після вищезгаданих дій можна встановлювати драйвери WinUSB за допомоги програми Zadig.exe на цей девайс та він буде готовий до роботи. На рис. 2.4 зображена зміна ProductId на платі Bus Blaster v3c

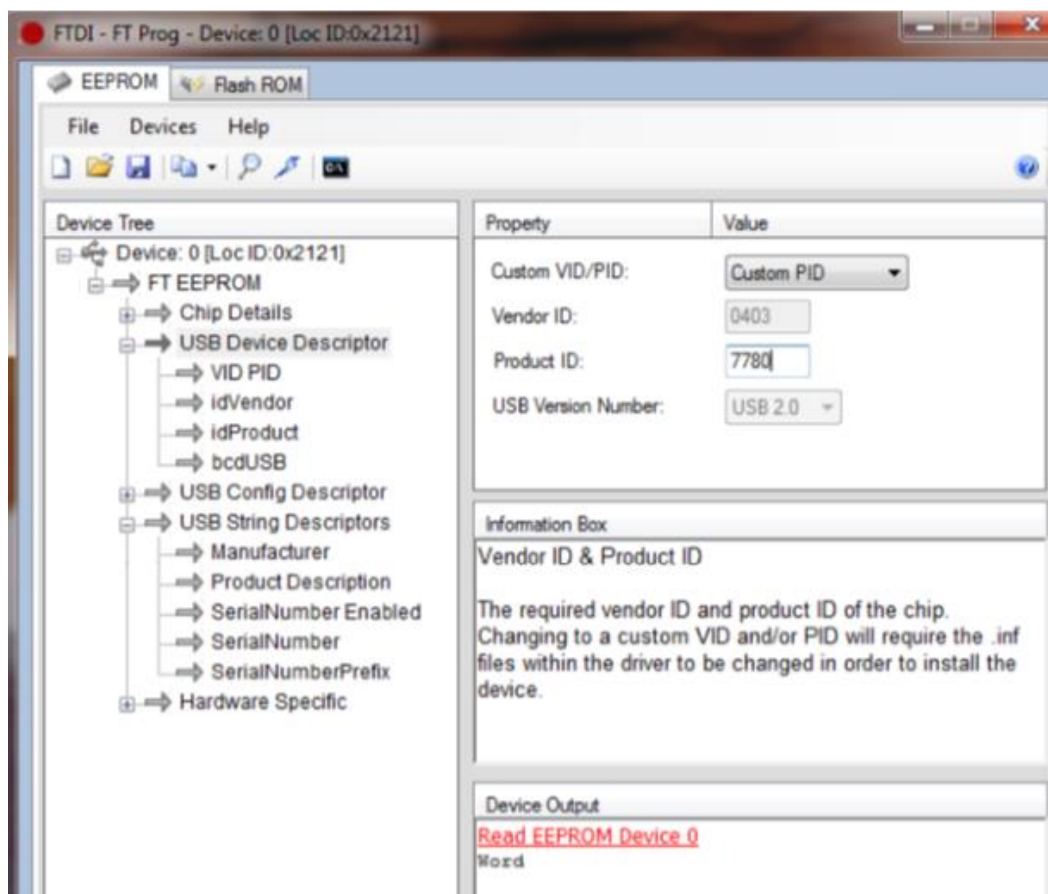


Рис. 2.4 Зміна ProductId на платі Bus Blaster v3c

2.2 Середовище розробки на базі Visual Studio Code

Початком роботи слугував приклад CExample з проекту MIPSfpga.

Налаштування взаємодії плагіну на C/C++ з компілятором та відлагоджувачем GCC з підтримкою інструкцій MIPS була використана конфігурація c_cpp_properties.json, у якій вказується шлях до відлагоджувача та режим компілятора IntelliSense msvc-x64. Загальна структура файлів зображена на рис. 2.5

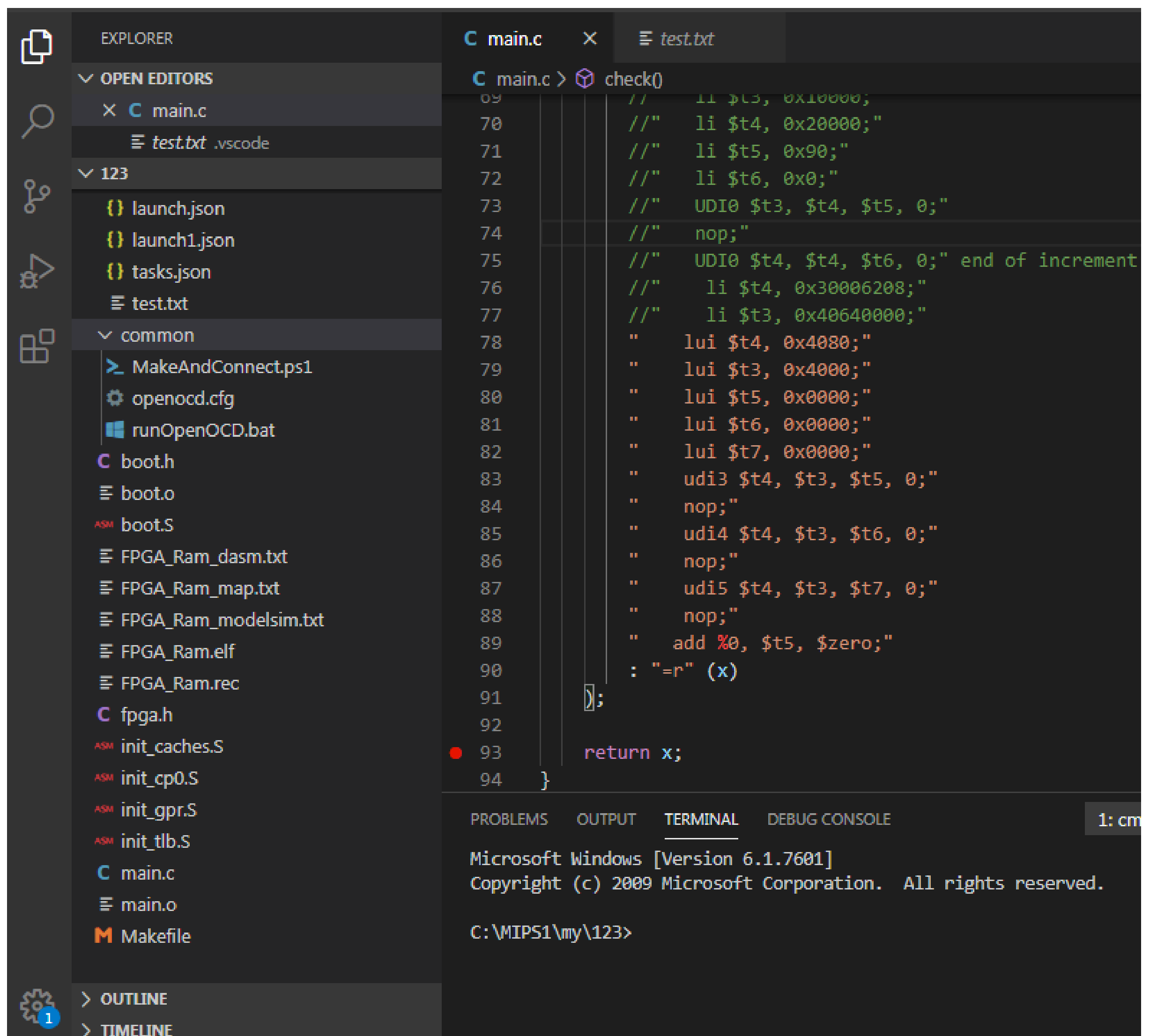


Рис. 2.5 Вигляд середовища розробки

В файлі launch.json знаходяться 2 профілі взаємодії – MIPSFPGA Load та MIPSFPGA Attach. При виборі Load програма виконає компіляцію та

завантажиться в процесор, після цього процесор увійде в режим відлагодження. При виборі опції Attach процесор входить у режим відлагодження з існуючою в ньому програмою.

В конфігурації необхідно вказувати шлях до файлу програми MIPS (*.elf). Шляхи не являються динамічними, тобто програма має лежати у чітко визначеному місці, оскільки це не підтримується OpenOCD.

За підключення до плати з завантаженим процесором через інтерфейс JTAG відповідає сервер OpenOCD, конфігурація до якого береться з файлу openocd.cfg.

У результаті ми отримали інструменти для компілювання та відлагодження.

Саме так і реалізоване виконання інструкцій процесора для програміста, який буде використовувати дане рішення. Все це відбувалось на віртуальній машині, яку при необхідності можна клонувати.

2.3 Інтеграція sin(x) в MIPS процессор

Був написаний модуль обрахування синуса, алгоритм якого у зрозумілому вигляді виглядає наступним чином:

*y = x1 * 2 / 3.14159265;*

*y2 = y * y;*

*y3 = y * y2;*

*y5 = y3 * y2;*

*y7 = y5 * y2;*

*sum = 1.57 * y - 0.645 * y3 +*

*0.08 * y5 - 0.00472 * y7;*

Даний модуль використовує float32 додавання та множення. Сам модуль продемонстрований у додатку А.

Після того як отримано необхідний вихідний Verilog-код, необхідно перейти до етапу інтеграції в процесор.

Система MIPS FPGA має спеціальний модуль, що називається UDI та розшифровується як інструкції, визначені користувачем. Це ефективний механізм для розширення функціоналу в MIPS-процесорі. Даний модуль підтримує до 7 додаткових інструкцій. В даній інтеграції використовується лише 5.

Створено новий модуль, що називається m14k_udi_seleqz. Використовується формат інструкції SPECIAL2, що має 3 аргументи та спеціальне поле для визначення сценарію використання інструкції, тобто кожну інструкцію можна поділити на ще більше інструкцій за допомогою коду в спеціальному полі. Оскільки використано всього 4 інструкції, потреби в цьому не було, тому постійно в спеціальне поле передається значення 0. Формат інструкції представлено на рис. 2.6.

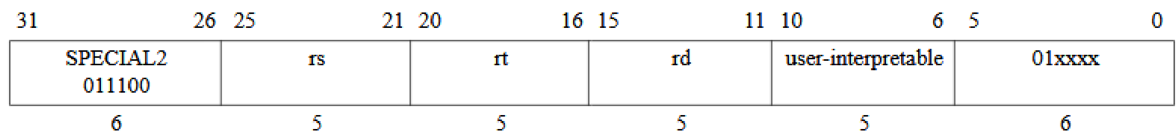


Рис. 2.6 Формат інструкції UDI

Далі приведена скорочена структура модуля m14k_udi_seleqz

1: udi_add_e = код вибору UDI3
2: udi_mul_e = код вибору UDI4
3: udi_div_e = код вибору UDI5
4: udi_sin_e = код вибору UDI2
5: UDI_ri_e = (udi_add_e || udi_mul_e || udi_div_e || udi_sin_e) ? 1'b0 : 1'b1; -> перевірка чи інструкція UDI вірна
6: Зберігання результату отриманого на стадії E для стадії M.
7: UDI_stall_m = udi_sin_m ? (udi_sin_m && ~ UDI_stall_m_inv_sin) : (udi_add_m || udi_mul_m || udi_div_m) && ~(UDI_stall_m_inv_add || UDI_stall_m_inv_mul || UDI_stall_m_inv_div); -> встановлення багатотактності інструкції
8: assign UDI_present = 1'b1; -> увімкнення UDI
9: Створення екземплярів UDI модулів
10: Запис значення з виділеного модуля до udi_res_e
за допомогою мультиплексору

Код вибору інструкції починається від значення 6’o20, тобто 010000 до 6’o26, тобто 010110. Важливо розуміти, що наразі інструкція призначена для UDI модуля та має формат SPECIAL2, тобто UDI_ir_e[31:26]==6'o34, де 6'o34 = 011100.

Далі приведений приклад реалізації вибору інструкції UDI.

assign udi_sin_e = UDI_irvalid_e && UDI_ir_e[31:26]==6'o34 && UDI_ir_e[5:0]==6'o22;

2.4 Тестування інструкції sin(x) у середовищі ModelSim

Для симуляції використовувався наступний ініціацілізуючий файл (ram_reset_init.ini):

3c013f25 - lui $1, 0x3f25;

342c5150 - ori $t4, $1,5150;

3c0d0000 - lui $t5, 0x0;

718B6812 - udi2 $t4, $t3, $t5, 0;

00000000 - nop;

Пояснення до інструкцій розглядається у табл. 2.1.

Таблиця 2.1

Використані інструкції асемблеру

lui $1, 0x3f25	Завантаження старших 16 біт аргумента x в регістр $1
ori $t4, $1,5150	Завантаження в регістр $t4 аргумента x
lui $t5, 0x0	Завантаження в регістр $t5 число 0
udi2 $t4, $t3, $t5, 0	Знаходження синусу аргумента $t4, результат в регістр $t5
nop	Нульова операція (branch delay)

На рис. 2.7 зображено початок виконання інструкції UDI2.

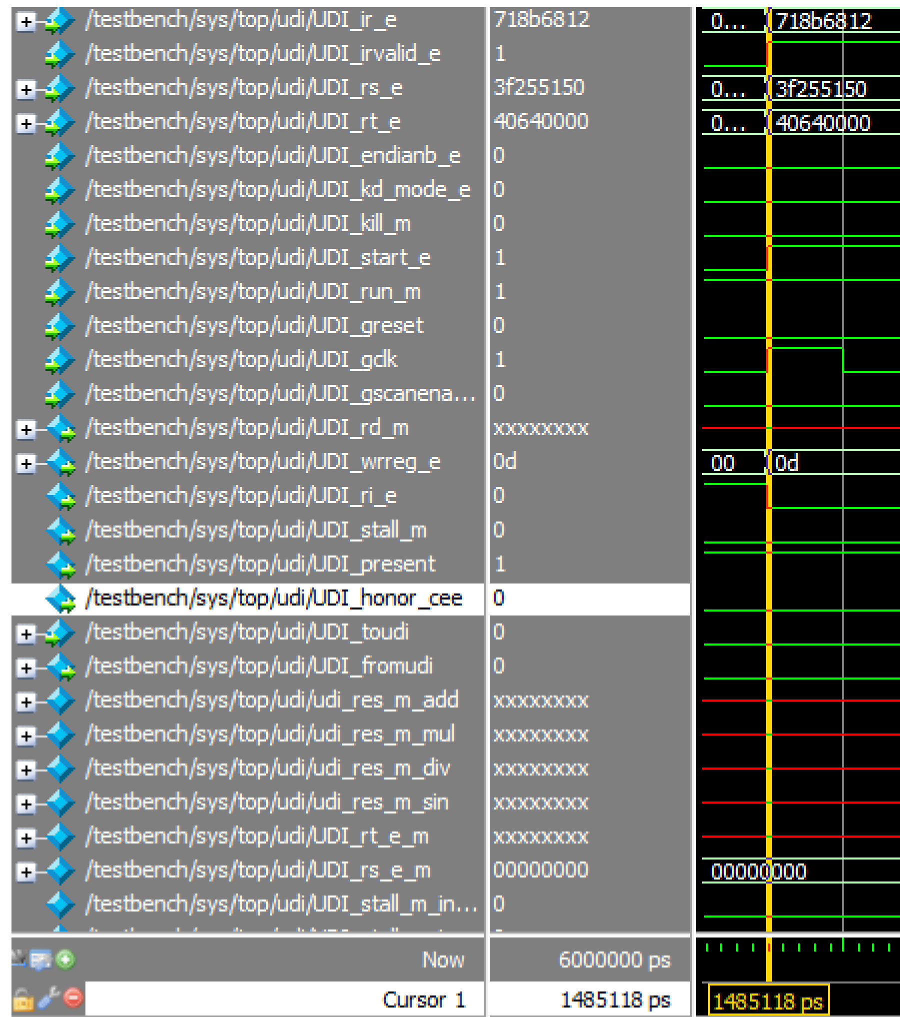


Рис. 2.7 Початок виконання інструкції UDI2

Аргуметом x є сигнал */testbench/sys/top/udi/UDI_rs_e*, в даному випадку він дорівнює $3f255150_{16}$, тобто 0.645772 радіан, або 37 градусів. Початок інструкції відбувся на момент часу 1485000 пікосекунд.

На рис. 2.8 зображено завершення виконання інструкції UDI2.

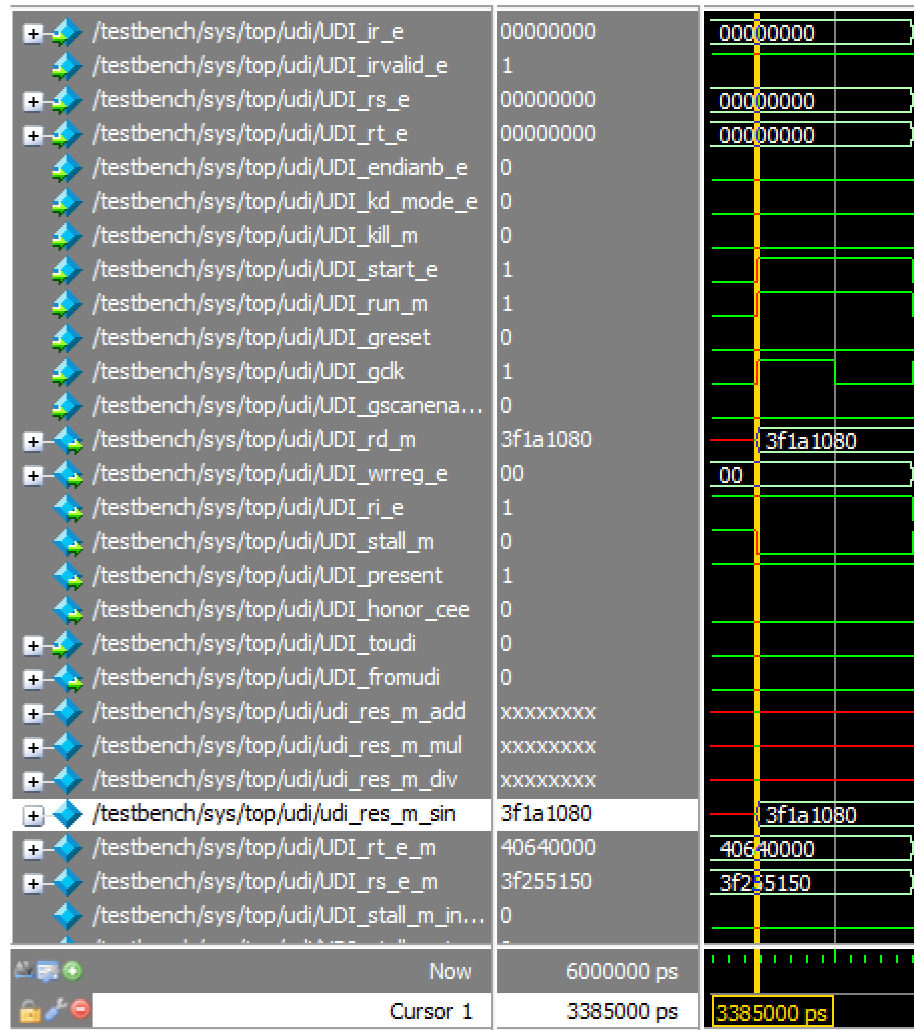


Рис. 2.8 Завершення виконання інструкції UDI2

Сигнал */testbench/sys/top/udi/UDI_rd_m* є результатом виконання інструкції, з якого на стадії M конвеєра процесора записується у регістр загального призначення $t5. На виході маємо $3f1a1080_{16}$. У результаті операції sin(0.645772) було отримано число 0.601814270. Інструкція завершилася на момент часу 3385000 пікосекунд. У середовищі ModelSim час одного такту займає 10000 пікосекунд. Час виконання інструкції вийшов 190 тактів. Для порівняння, операція множення float32 виконується за 14 тактів, ділення близько 100.

2.5 Тестування інструкцій posit числення у середовищі ModelSim

2.5.1 Ділення posit чисел

Було розроблено та протестовано інструкції додавання, множення та ділення 16 бітних чисел використовуючи формат чисел posit.

На рисунку 2.9 зображений початок виконання інструкції ділення, а саме 0x718b6813.

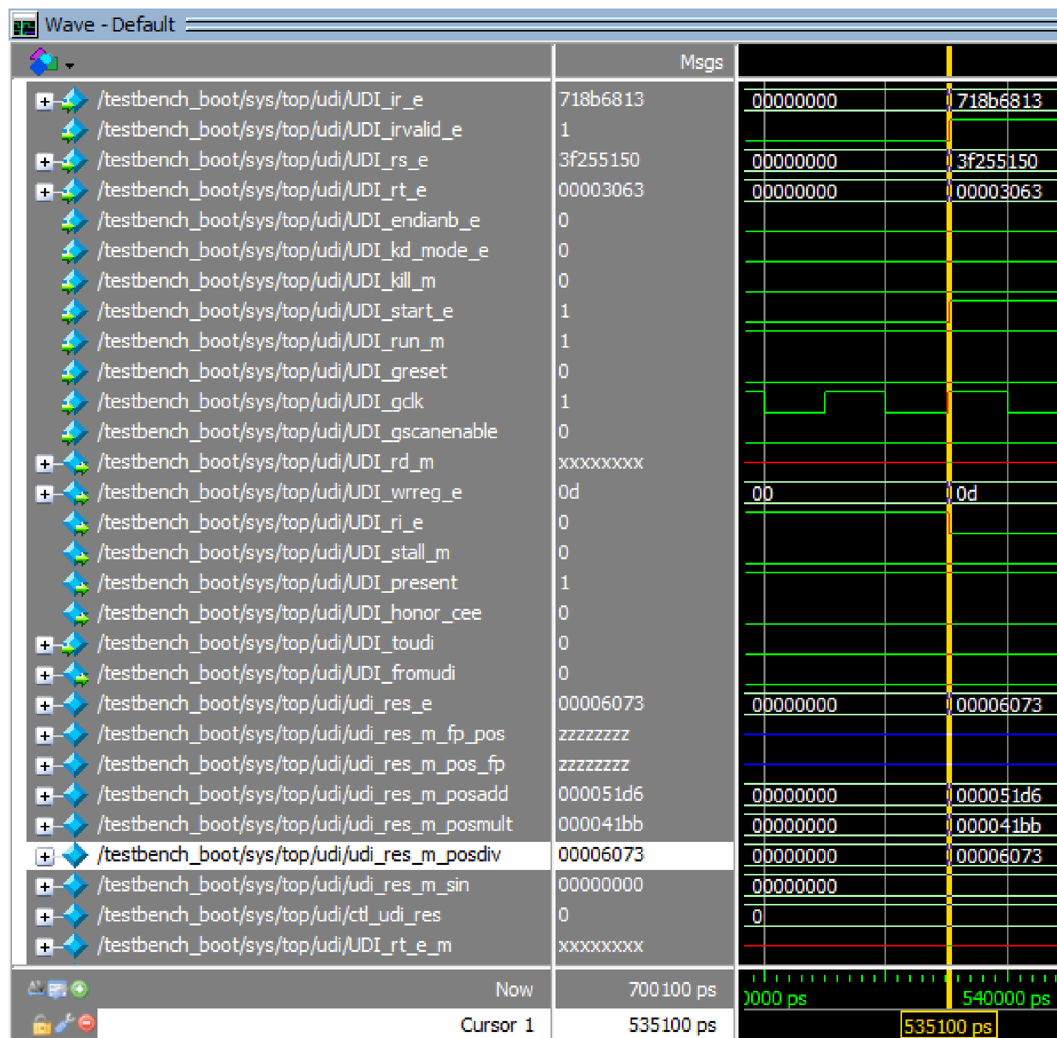


Рис. 2.9 Початок виконання інструкції ділення posit

Сигнал /testbench/sys/top/udi/udi_res_m_posdiv є попереднім результатом роботи модуля ділення. Інструкція ділить перші(молодші) 16 біт з сигналу /testbench/sys/top/udi/UDI_rs_e на перші 16 біт з сигналу /testbench/sys/top/udi/UDI_rt_e. Дані інтерпретуються як posit числа. У нашій

архітектурі використовується little endian, що визначає останній біт числа як молодший.

Сигнал /testbench/sys/top/udi/udi_res_e являється виходом мультиплексору, який відповідає за вивід результату до регістру, з якого буде зчитаний результат інструкції. Він приймає значеня відповідно до управляючого сигналу /testbench/sys/top/udi/ctl_udi_res, який дорівнює 0. При поточному керуючому модулю інструкцій користувача, це є результатом ділення.

На рисунку 2.10 зображено завершення виконання інструкції ділення, з результатом 0x00006073.

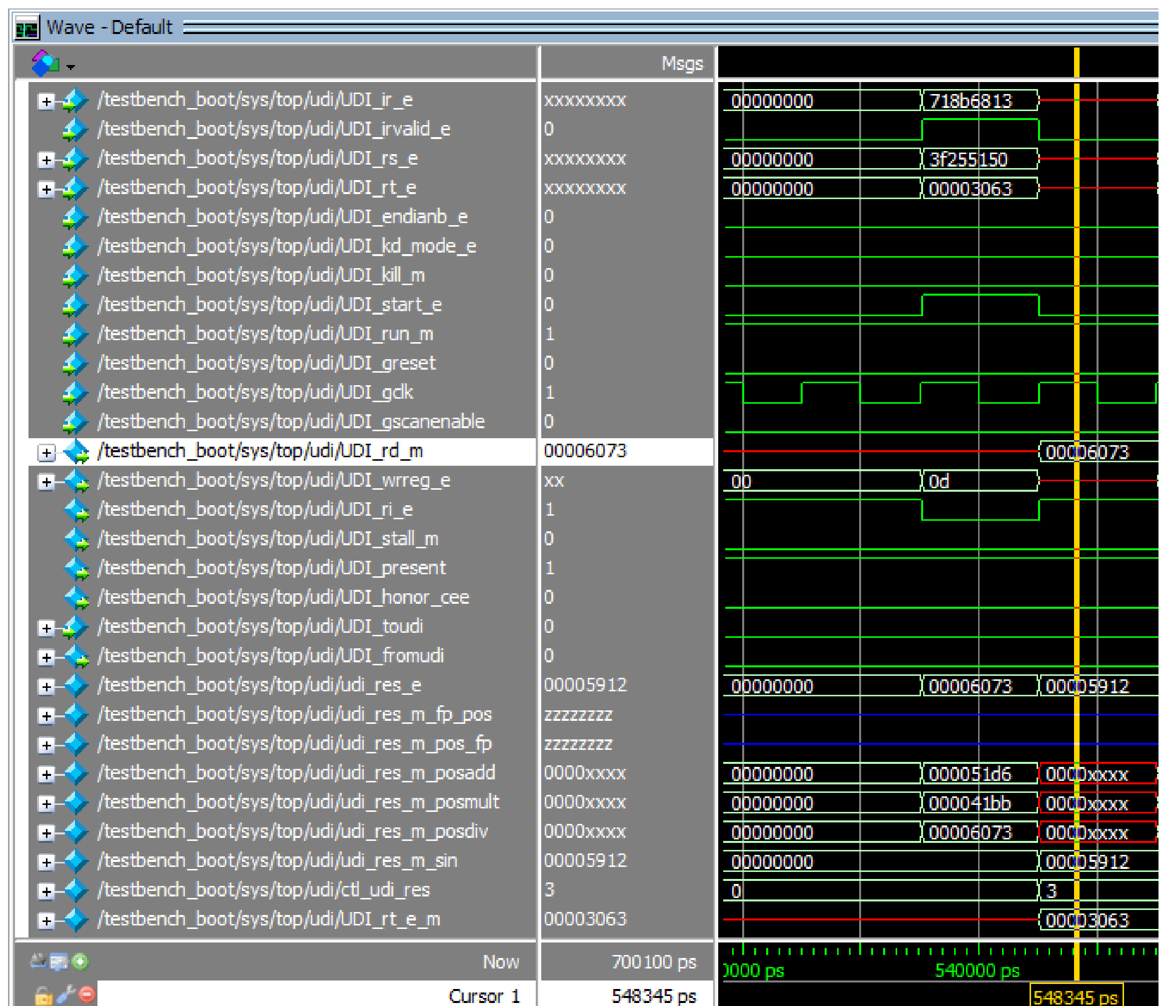


Рис. 2.10 Завершення виконання інструкції ділення posit

Сигнал /testbench/sys/top/udi/UDI_rd_m є результатом роботи інструкції ділення. Значення даного регістру буде записано інструкцією у відповідний результуючий регістр, заданий в декларації інструкції при переході конвеєра процесора з E-Stage (виконання) до M-Stage (операції з пам’яттю). В даному

випадку цей регістр $t5. Список регістрів наведений у теоретичній частині роботи.

Розберемо результати. Сигнал /testbench/sys/top/udi/UDI_rs_e представляє собою ділене, а саме 0x5150. В бінарному вигляді 0101000101010000_2, що при розмірі експоненти рівній 1 розшифровується як 2.1640625.
Сигнал /testbench/sys/top/udi/UDI_rt_e представляє собою дільник, а саме 0x3063. В бінарному вигляді 0011000001100011_2, що при розмірі експоненти рівній 1 розшифровується як 0.5120849609375.

Результат /testbench/sys/top/udi/UDI_rd_m рівний 0x6073. В бінарному вигляді 0110000001110011_2, що при розмірі експоненти рівній 1 розшифровується як 4.224609375. Для порівняння абсолютне значення результату дорівнює 4.22598331347.

Інструкція виконалася за 1 такт. Виконання інструкції ділення float32 займає приблизно 100 тактів.

2.5.2 Множення posit чисел

На рисунку 2.11 зображений початок виконання інструкції ділення, а саме 0x718b6816.

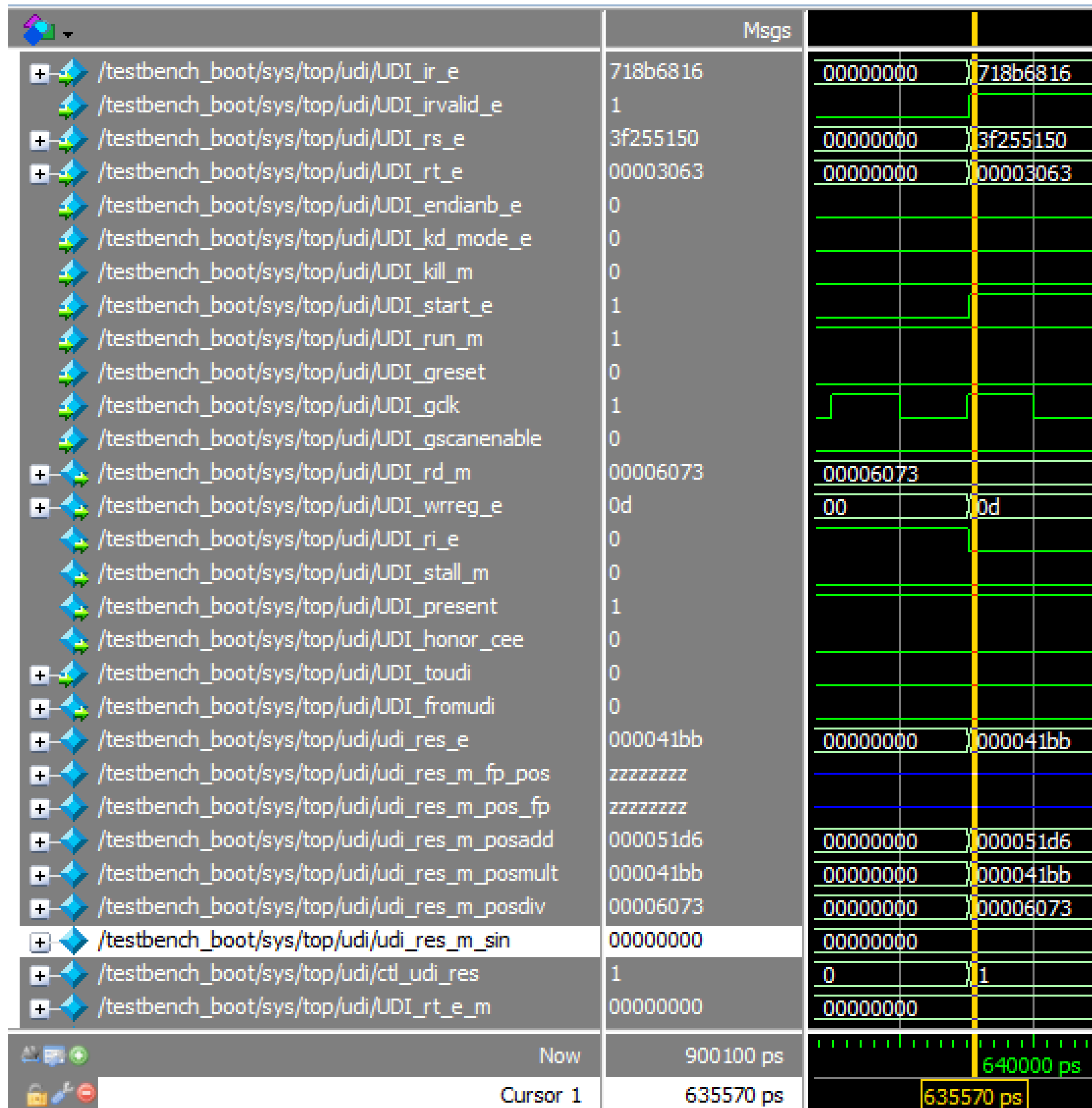


Рис. 2.11 Початок виконання інструкції множення posit

Сигнал */testbench/sys/top/udi/udi_res_posmult* є попереднім результатом роботи інструкції множення. Інструкція виконує операцію множення з першими 16 біт з сигналу */testbench/sys/top/udi/UDI_rs_e* та першими 16 біт з сигналу */testbench/sys/top/udi/UDI_rt_e.*

Сигнал */testbench/sys/top/udi/udi_res_e* це знову ж таки вихід мультиплексору. Він приймає значеня відповідно до управляючого сигналу

/testbench/sys/top/udi/ctl_udi_res, який дорівнює 1 в даній інструкції. При поточному керуючому модулю інструкцій користувача, це є результатом множення.

На рисунку 2.12 зображено завершення виконання інструкції ділення, з результатом 0x000041bb.

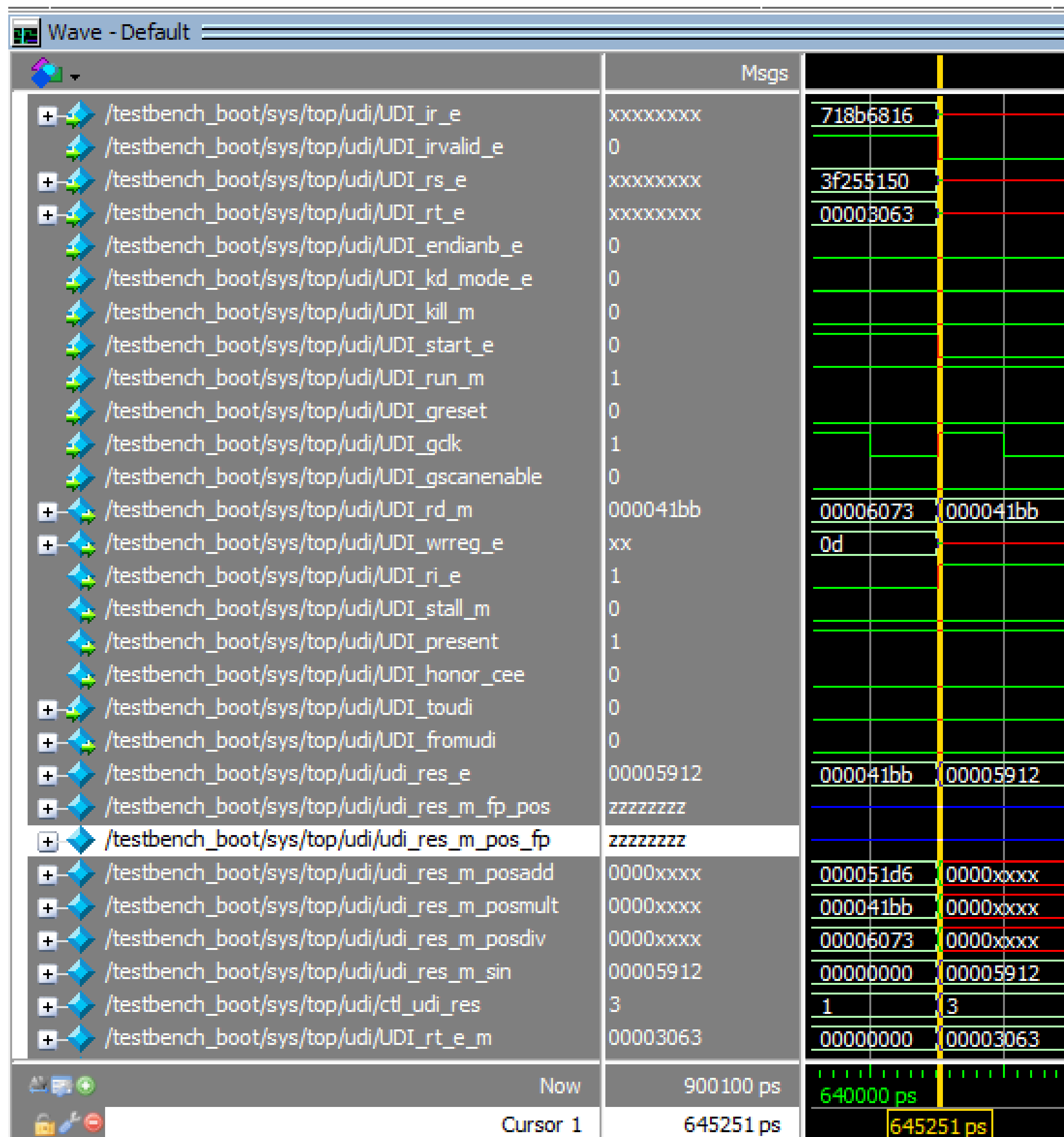


Рис. 2.12 Завершення виконання інструкції множення posit

Сигнал */testbench/sys/top/udi/UDI_rd_m* є результатом роботи інструкції множення. Значення даного регістру буде записано у регістр $t5.

Розберемо результати. Сигнал */testbench/sys/top/udi/UDI_rs_e* представляє собою множене, а саме 0x5150. В бінарному вигляді 0101000101010000_2, що при розмірі експоненти рівній 2 розшифровується як 4.65625.

Сигнал */testbench/sys/top/udi/UDI_rt_e* представляє собою множник, а саме 0x3063. В бінарному вигляді 0011000001100011_2, що при розмірі експоненти рівній 2 розшифровується як 0.2620849609375.

Результат */testbench/sys/top/udi/UDI_rd_m* рівний 0x41bb. В бінарному вигляді 0100000110111011_2, що при розмірі експоненти рівній 2 розшифровується як 1.21630859375. Для порівняння абсолютне значення результату дорівнює 1.22033309937.

Інструкція виконалася за 1 такт. Виконання інструкції множення float32 займає 14 тактів.

2.5.3 Додавання posit чисел

На рисунку 2.13 зображений початок виконання інструкції ділення, а саме 0x718b6812.

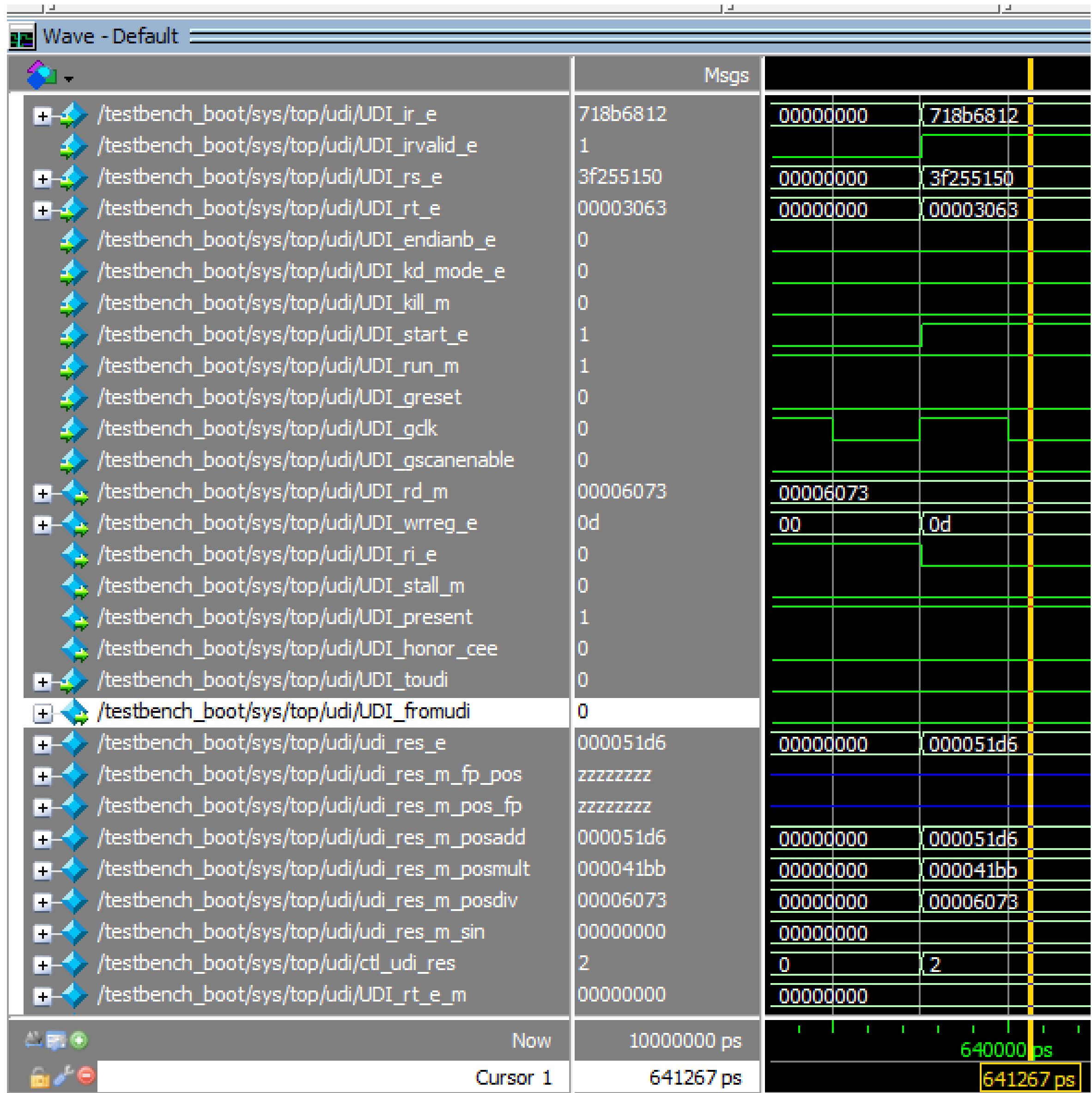


Рис. 2.13 Початок виконання інструкції додавання posit

Сигнал */testbench/sys/top/udi/udi_res_posadd* є результатом роботи інструкції додавання. Інструкція додає перші 16 біт з сигналу */testbench/sys/top/udi/UDI_rs_e* та перші 16 біт з сигналу */testbench/sys/top/udi/UDI_rt_e.*

Сигнал /testbench/sys/top/udi/udi_res_e приймає значеня відповідно до управляючого сигналу /testbench/sys/top/udi/ctl_udi_res, який дорівнює 2 в даній

інструкції. При поточному керуючому модулю інструкцій користувача, це є результатом додавання.

На рисунку 2.14 зображено завершення виконання інструкції додавання, з результатом 0x000051d6.

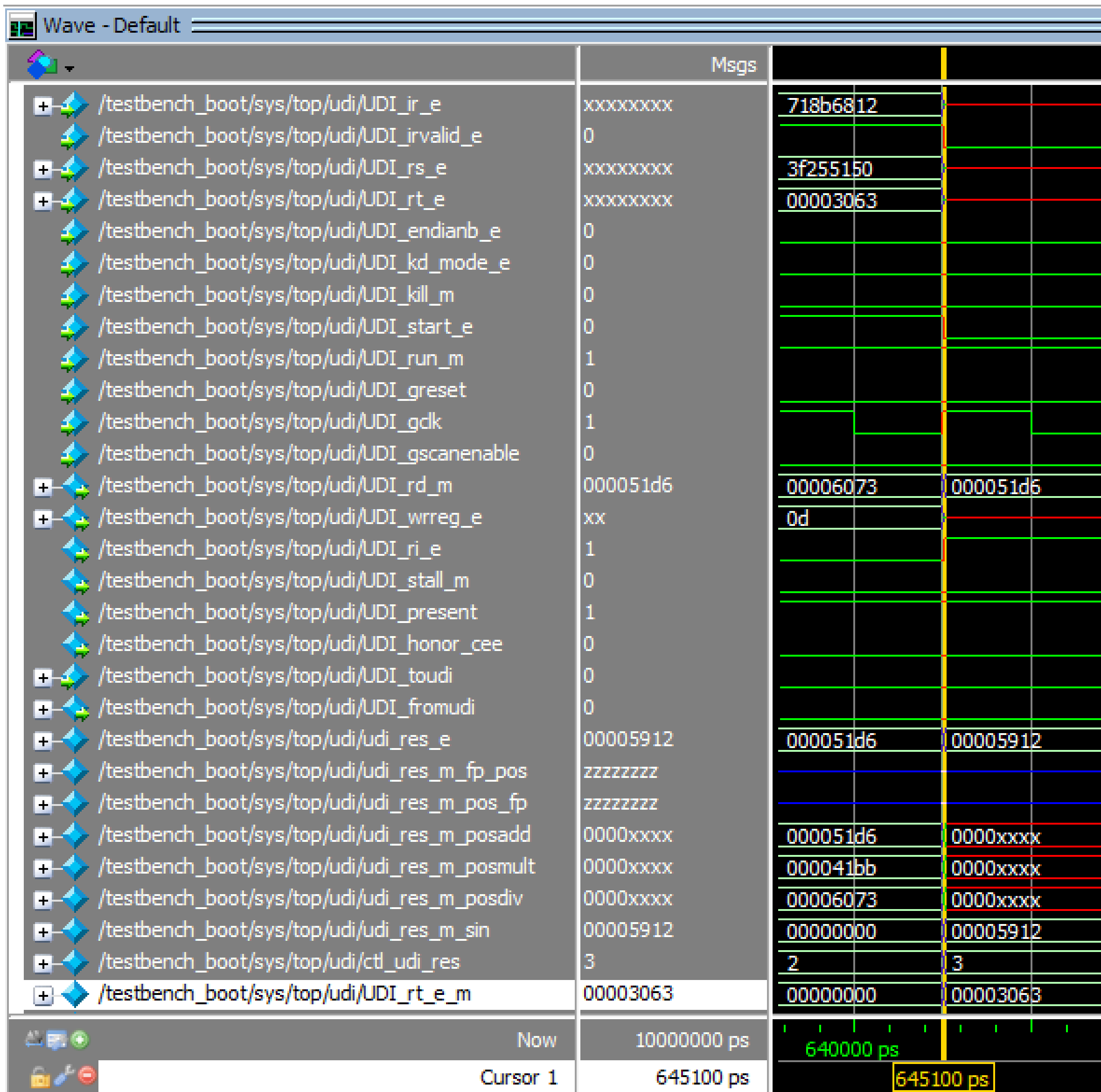


Рис. 2.14 Завершення виконання інструкції додавання posit

Сигнал */testbench/sys/top/udi/UDI_rd_m* є результатом роботи інструкції додавання. Значення даного регістру буде записано у регістр $t5.

Розберемо результати. Сигнал */testbench/sys/top/udi/UDI_rs_e* представляє собою множене, а саме 0x5150. В бінарному вигляді 0101000101010000_2, що при розмірі експоненти рівній 2 розшифровується як 4.65625.

Сигнал */testbench/sys/top/udi/UDI_rt_e* представляє собою дільник, а саме 0x3063. В бінарному вигляді 0011000001100011_2, що при розмірі експоненти рівній 2 розшифровується як 0.2620849609375.

Результат */testbench/sys/top/udi/UDI_rd_m* рівний 0x51d6. В бінарному вигляді 0101000111010110_2, що при розмірі експоненти рівній 2 розшифровується як 4.91796875. Для порівняння абсолютне значення результату дорівнює 4.91833496094.

Інструкція виконалася за 1 такт. Виконання інструкції множення float32 займає 14 тактів.

2.6 Тестування на платі DE2-115

Для тестуванні на платі було застосоване IDE описане у пункті 2.2. Було проведено ряд тестів, результати яких відображені у табл. 2.2

Таблиця 2.2

Результати тестів виконаних на платі

x		x, що подається на вхід	результат, що зберігається в регістрі		розрахований sin(x)	точний sin(x)
°	рад	система числення				
		16	10	16		
0	0	00000000	0	00000000	0	0
13	0.226893	3e6856a3	1046894983	3e665987	0.224950894	0.22495105
30	0.523599	3f060a96	1056964591	3effffef	0.499999493	0.5
37	0.645772	3f255150	1058672768	3f1a1080	0.601814270	0.60181502
69	1.20428	3f9a25d9	1064238638	3f6efe2e	0.933565974	0.93358042
87	1.51844	3fc25c3e	1065328270	3f7f9e8e	0.998513102	0.99862953
90	1.5708	3fc90ff9	1065350570	3a7aa5aa	0.999842286	1

Вище наведена таблиця демонструє результати обрахування функції. Для прикладу візьмемо наступні дані: аргументом є $3f255150_{16}$ у форматі float32, що розшифровується як 0.645772 радіан (37 градусів).

На виході маємо число 1058672768_{10}, що дорівнює $3f1a1080_{16}$. У результаті отримано число sin(0.645772) = 0.601814270. На рисунку 2.15 зображено розрахунок синуса у середовищі програмування.

Рис. 2.15 Розрахунок синуса 37°

2.7 Аналіз результатів

В результаті роботи було отримано 2 версії процесора:

1) з модулем float32 та функцією sin(x);
2) з реалізацією posit числення.

Для тестування функції sin(x) було взято серію випадкових аргументів та перевірено коректність результатів. Після компіляції були отримані цікаві результати, результат, що очікувався повинен займати менше площі на кристалі та бути компактнішим. Для тестування posit числення було використано 2 типа 16-бітних posit числення: для множення и додавання з експонентою рівною 2, для ділення з експонентою рівною 1. У табл. 2.3 наведено порівняння результатів компіляції. Інструкції posit числення працюють корректно, в роботі наведені часткові випадки обрахувань.

Таблиця 2.3

Порівняння результатів компіляції з UDI модулями та без

Параметр\Модуль	Без модулів	Sin(x), Float32	Posit
Загальна кількість логічних елементів	15320	24075	19368
Кількість регістрів	7487	11847	7463
Перемножувачі	0	70	4

Кількість регістрів значно зросла у реалізації інструкції sin(x) та Floating Point порівняно з реалізацією чистого процесору без модулів. Однак велику кількість елементів можна зумовити тим, що у FPGA-матриць досить специфічне представлення логічних елементів, важко спрогнозувати що матиме більшу площу в кінцевій реалізації. Також варто зазначити, що в цій версії доданий FPU на float32

На рисунку 2.16 наведено схему розробленого модуля UDI

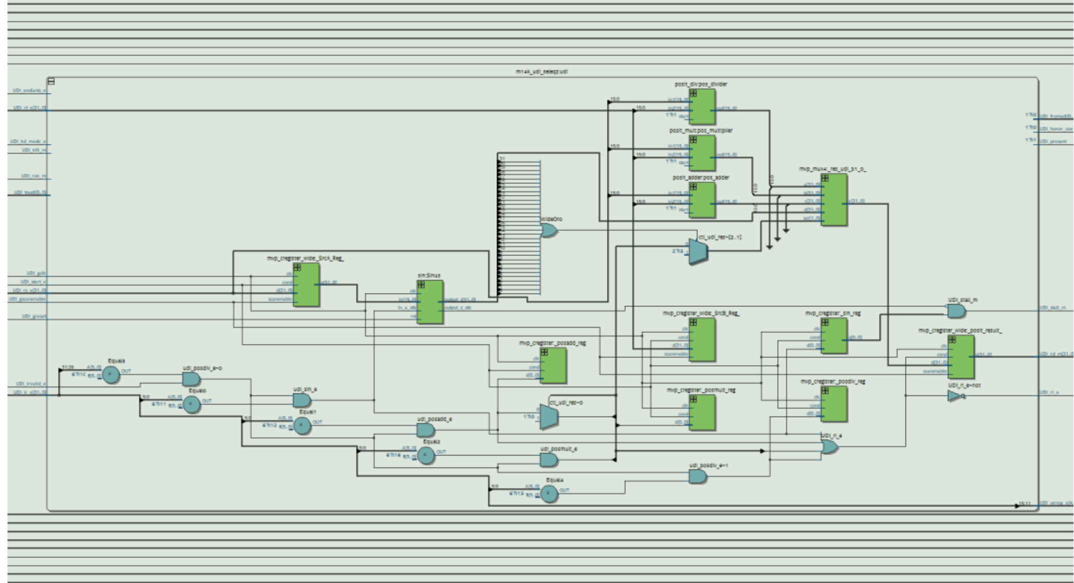


Рис. 2.16 Схема розробленого модуля UDI з posit арифметикою

Приріст швидкодії у випадку sin(x) з числами float32 близько 3 разів відносно програмної реалізації. Доцільність даної версії процесора сумнівна.

Однак, друга версія процесора показала значно кращі результати. Кількість логічних елементів збільшується не критично відносно голої версії процесора. Кількість регістрів та перемножувачів майже не змінилась, на відміну від першої версії процесора. Дані інструкції posit числення показали себе відмінно з точки зору швидкодії, у окремих випадках досягаючи приросту у 100 разів (у випадку ділення) відносно float32. При множенні та додаванні приріст склав 14 разів. Усі інструкції виконались за 1 такт, що в цілому може при збільшенні тактової частоти процесора сильно збільшити приріст відносно float32, оскільки конкретний час виконання в межах 1 такту визначити неможливо. Але є недолік у вигляді невисокої точності, близько 3 знаків після коми в середньому. Проте змінюючи розмір експоненти можна підібрати, якщо знати приблизні межі чисел, в яких відбувається обрахування в конкретній задачі. Це і є однією з переваг posit числення, а саме плаваючий розмір експоненти, що при 16 бітному численні може призвести до вкрай високого росту швидкодії в тих задачах, які цього потребують, при відносно допустимій похибці.

ВИСНОВКИ

Результати роботи можна використати для модернізації існуючих інструкцій та реалізації нових процесорних інструкцій, використовуючи розроблені модулі. Пропоновані 16-бітні posit числа рекомендується використовувати у задачах, в яких не потрібна висока точність, та дані для арифметичних операцій є самі по собі не точними, або представляють числа невеликого діапазону. Максимально точний діапазон можна регулювати величиною експоненти в posit числах. Також умовою є те, що результат задачі буде кращий від ітерацій виконання арифметичних операцій. Прикладом даних задач є певна частина операцій, які виконує GPU або CPU при задачах машинного навчання, де більша кількість ітерацій може покращити результати. Дане рішення не можна використати при точних математичних розрахунках, проте можна використати якщо результати можуть мати похибку, що буде у межах похибки результатів обчислень використовуючи posit16 числа з оптимальною заданою кількістю біт експоненти. Прикладом таких задач є система переміщення і орієнтації в просторі дрону, розрахунок необхідного кута нахилу корпусу.

Наукова новизна:

1. Вперше запропонована інструкція для обрахування функції sin(x) за допомогою полінома Тейлора, яка була впроваджена у процесорне ядро MIPSfpga, та протестована у пакеті ModelSim та на FPGA платі Altera DE2-115.
2. Досягнуто підвищення швидкодії виконання процесорної інструкції функції sin(x) з використанням формату чисел float32, додано арифметичні операції posit арифметики у MIPS-сумісний процесор для покращення поточної реалізації інструкції обрахування sin(x). При реалізації даної інструкції, використовуючи posit арифметику, швидкість збільшиться приблизно у 14 разів або більше, в залежності від тактової частоти.

3. Використаний поліном Тейлора для реалізації інструкції обрахування функції sin(x), яка була впроваджена у процесорне ядро MIPSfpga, та протестована у пакеті ModelSim та на FPGA платі Altera DE2-115.
4. Розроблено процесорні інструкції обрахування операцій додавання, множення та ділення використовуючи posit арифметику.
5. Результати роботи можна використати для подальшої модернізації інструкцій та реалізації нових процесорних інструкцій.
6. В ході роботи було виявлено, що обрахування інструкцій додавання на posit арифметиці виконується у 14 разів швидше ніж на float32. У випадку ділення приріст досягав 100 разів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Дэвид М. Харрис, Сара Л. Харрис. Цифровая схемотехника и архитектура компьютера второе издание: навчальний посібник. Morgan Kaufman, 2015. 1621 с
2. Uwe Meyer-Baese. Digital Signal Processing with Field Programmable Gate Arrays Fourth Edition: навчальний посібник. Springer, 2005. 930 с.
3. Інструкція користувача плати DE2-115 [Електронний ресурс]: – Режим доступу до ресурсу: https://www.intel.com/content/dam/altera-www/global/en_US/portal/dsn/42/doc-us-dsnbk-42-1404062209-de2-115-user-manual.pdf
4. MIPS32® MicroAptiv™ UP Processor Core Family Integrator's Guide Micro Aptiv Integrator's MD0094 [Електронний ресурс]: – Режим доступу до ресурсу: https://usermanual.wiki/Document/MicroAptiv20UP20Integrators20Guide20MD00941.2060483040
5. MIPSfpga Getting Started Guide v1.3 [Електронний ресурс]: – Режим доступу до ресурсу: https://www.mips.com/downloads/mipsfpga-getting-started-guide-v1-3-russian
6. Intel Quartus Prime Standard Edition User Guide: Getting Started [Електронний ресурс]: – Режим доступу до ресурсу: https://www.intel.com/content/www/us/en/programmable/documentation/yoq1529444104707.html
7. MIPS Instruction Set [Електронний ресурс]: – Режим доступу до ресурсу: https://www.dsi.unive.it/~gasparetto/materials/MIPS_Instruction_Set.pdf
8. Bus Blaster v3 design overview [Електронний ресурс]: – Режим доступу до ресурсу: http://dangerousprototypes.com/docs/Bus_Blaster_v3_design_overview
9. Posit Standard Documentation [Електронний ресурс]: – Режим доступу до ресурсу: https://posithub.org/docs/posit_standard.pdf

10. Posit lookup table [Електронний ресурс]: – Режим доступу до ресурсу: https://posithub.org/widget/lookup
11. IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language. IEEE Computer Society: стандарт IEEE Std 1364-1995, 2001.

ДОДАТКИ

Додаток А

Далі приведені модулі на мові Verilog. Модуль m14k_udi_seleqz.v являється модулем, що процесор сприймає як керуючий UDI інструкціями. Цей модуль використовує всі написані інструкції.

Модуль m14k_udi_seleqz.v

```
`include "m14k_const.vh"
module m14k_udi_seleqz(
       UDI_ir_e,
       UDI_irvalid_e,
       UDI_rs_e,
       UDI_rt_e,
       UDI_endianb_e,
       UDI_kd_mode_e,
       UDI_kill_m,
       UDI_start_e,
       UDI_run_m,
       UDI_greset,
       UDI_gclk,
       UDI_gscanenable,
       UDI_rd_m,
       UDI_wrreg_e,
       UDI_ri_e,
       UDI_stall_m,
       UDI_present,
       UDI_honor_cee,
       UDI_toudi,
       UDI_fromudi);

        /* Inputs */
    input [31:0]    UDI_ir_e;       // full 32 bit Spec2 Instruction
    input           UDI_irvalid_e;  // Instruction reg. valid signal.
    input [31:0]    UDI_rs_e;       // edp_abus_e data from register file
    input [31:0]    UDI_rt_e;       // edp_bbus_e data from register file
       input        UDI_endianb_e;      // Endian - 0=little, 1=big
       input        UDI_kd_mode_e;      // Mode - 0=user, 1=kernel or debug

    input           UDI_kill_m;  // Kill signal
    input           UDI_start_e; // mpc_run_ie signal to start the UDI.
    input           UDI_run_m;       // mpc_run_m signal to qualify kill_m.
```

```
    input           UDI_greset;  // greset signal to reset state machine.
    input           UDI_gclk;    // Clock
    input           UDI_gscanenable;

        /* Outputs */
    output [31:0]   UDI_rd_m;           // Result of the UDI in M stage
    output [4:0]    UDI_wrreg_e;        // Register File
    output          UDI_ri_e;           // Illegal Spec2 Instn.
    output          UDI_stall_m;        // Stall the pipeline. E stage signal
       output             UDI_present;        // Indicate whether UDI is implemented
       output             UDI_honor_cee;        // Indicate whether UDI has local state


    // external UDI signals
  input  [`M14K_UDI_EXT_TOUDI_WIDTH-1:0] UDI_toudi; // External input to UDI module
  output  [`M14K_UDI_EXT_FROMUDI_WIDTH-1:0] UDI_fromudi; // Output from UDI module to
external system

// BEGIN Wire declarations made by MVP
wire [`M14K_UDI_EXT_FROMUDI_WIDTH-1:0] /*[0:0]*/ UDI_fromudi;
// END Wire declarations made by MVP

wire   [31:0] udi_res_e;
wire   [31:0] udi_res_m_fp_pos;
wire   [31:0] udi_res_m_pos_fp;
wire   [31:0] udi_res_m_posadd;
wire   [31:0] udi_res_m_posmult;
wire   [31:0] udi_res_m_posdiv;
wire   [31:0] udi_res_m_sin;
wire   [1:0]  ctl_udi_res;
wire   [31:0] UDI_rt_e_m;
wire   [31:0] UDI_rs_e_m;
wire                udi_fp_pos_e;
wire                udi_pos_fp_e;
wire                udi_posadd_e;
wire                udi_posmult_e;
wire                udi_posdiv_e;
wire                udi_sin_e;
wire                udi_sin_m;
wire                UDI_stall_m_inv_sin;
assign UDI_fromudi[`M14K_UDI_EXT_FROMUDI_WIDTH-1:0] =
{`M14K_UDI_EXT_FROMUDI_WIDTH{1'b0}};

// This module is a dummy module which reflects that no user defined SPEC2
// instructions have been implemented. So it just sets ri_e to 1
// to signal that any spec2 instn is illegal.
```

```
// Inactive value for outputs and no connect for inputs

mvp_cregister_wide #(32) _SrcB_Reg_(UDI_rt_e_m[31:0], UDI_gscanenable, UDI_start_e,
UDI_gclk, UDI_rt_e[31:0]);
mvp_cregister_wide #(32) _SrcA_Reg_(UDI_rs_e_m[31:0], UDI_gscanenable, UDI_start_e,
UDI_gclk, UDI_rs_e[31:0]);//new

assign udi_sin_e = UDI_irvalid_e && UDI_ir_e[31:26]==6'o34 && UDI_ir_e[5:0]==6'o21;
mvp_cregister #(1) _sin_reg(udi_sin_m, UDI_start_e, UDI_gclk, udi_sin_e);
assign udi_posadd_e = UDI_irvalid_e && UDI_ir_e[31:26]==6'o34 && UDI_ir_e[5:0]==6'o22; //
Set to 1 when valid UDI2 instruction is at the IR.
mvp_cregister #(1) _posadd_reg(udi_posadd_m, UDI_start_e, UDI_gclk, udi_posadd_e);
                           // Register udi_mul
assign udi_posmult_e = UDI_irvalid_e && UDI_ir_e[31:26]==6'o34 && UDI_ir_e[5:0]==6'o26;
       // Set to 1 when valid UDI6 instruction is at the IR.
mvp_cregister #(1) _posmult_reg(udi_posmult_m, UDI_start_e, UDI_gclk, udi_posmult_e);

assign udi_posdiv_e = UDI_irvalid_e && UDI_ir_e[31:26]==6'o34 && UDI_ir_e[5:0]==6'o23; //
Set to 1 when valid UDI6 instruction is at the IR.
mvp_cregister #(1) _posdiv_reg(udi_posdiv_m, UDI_start_e, UDI_gclk, udi_posdiv_e);

//assign UDI_ri_e = 1'b1;           // Illegal Spec2 Instn.
assign UDI_ri_e = (udi_posadd_e || udi_posmult_e || udi_sin_e || udi_posdiv_e) ? 1'b0 :
1'b1;  // Set to 1 unless a valid UDI0, UDI1 or UDI2 instruction is at the IR.

//assign UDI_rd_m = 32'b0;                // Result = 0
mvp_cregister_wide #(32) _posit_result_(UDI_rd_m[31:0], UDI_gscanenable, ~UDI_ri_e,
UDI_gclk, udi_res_e[31:0]);      // Register the result calculated at the E-Stage to the
M-Stage.

//assign UDI_wrreg_e = 5'b0;         // No writing into register.
assign UDI_wrreg_e = UDI_ir_e[15:11];   // Destination register for the new UDI
instruction.

// assign UDI_stall_m = udi_sin_m ? (udi_sin_m && ~ UDI_stall_m_inv_sin) : (udi_posadd_m
|| udi_posmult_m);
assign UDI_stall_m = udi_sin_m && ~(UDI_stall_m_inv_sin);
assign UDI_present = 1'b1;
assign UDI_honor_cee = 1'b0;

// NEW FUNCTIONALITY:

posit_adder pos_adder(
       .in1(UDI_rs_e[15:0]),
       .in2(UDI_rt_e[15:0]),
       .start(1'b1),
       .out(udi_res_m_posadd[15:0]),
```

```
        .inf(), .zero(), .done());

posit_mult pos_multiplier(
        .in1(UDI_rs_e[15:0]),
        .in2(UDI_rt_e[15:0]),
        .start(1'b1),
        .out(udi_res_m_posmult[15:0]),
        .inf(), .zero(), .done());

posit_div pos_divider(
        .in1(UDI_rs_e[15:0]),
        .in2(UDI_rt_e[15:0]),
        .start(1'b1),
        .out(udi_res_m_posdiv[15:0]),
        .inf(), .zero(), .done());

sin Sinus(
        .in(UDI_rs_e_m[15:0]),
    .in_a_stb(udi_sin_e),
    .clk(UDI_gclk),
    .rst(UDI_greset),
    .output_z(udi_res_m_sin),
    .output_z_stb(UDI_stall_m_inv_sin)
);

assign ctl_udi_res[1:0] = udi_res_m_sin ? 2'b11 : udi_posmult_e ? 2'b01 : udi_posadd_e ?
2'b10 : 2'b00;

mvp_mux4 #(32) _res_udi_31_0_(udi_res_e[31:0], ctl_udi_res[1:0], udi_res_m_posdiv,
udi_res_m_posmult, udi_res_m_posadd, udi_res_m_sin);

assign udi_res_m_posadd[31:16] = 4'h0;
assign udi_res_m_posmult[31:16] = 4'h0;
assign udi_res_m_posdiv[31:16] = 4'h0;

//verilint 240 on  // Unused input
endmodule       // m14k_udi_stub
```

Додаток Б

Використане обладнання

FPGA плата Altera DE2-115

Плата Altera DE2-115 має безліч функцій, які дозволяють користувачам реалізувати широкий спектр розроблених схем, від простих схем до різних мультимедійних проектів.[3]

На платі DE2-115 надано таке обладнання:

- FPGA Altera Cyclone® IV 4CE115 FPGA
- Пристрій послідовної конфігурації Altera - EPCS64
- USB Blaster (на платі) для програмування; підтримуються і режими програмування JTAG і Active Serial (AS)
- 2 МБ SRAM
- Два 64MB SDRAM
- 8MB флеш-пам'ять
- Кард-рідер
- 4 натисні кнопки
- 18 кнопок-перемикачів
- 18 червоних світлодіодів
- 9 зелених світлодіодів
- генератор 50 МГц для джерел тактових частот
- 24-bit CD-quality audio CODEC з лінійними входом та виходом та вхід мікрофона
- VGA DAC (8-бітний швидкісний потрійний ЦАП) з роз'ємом VGA
- телевізійний декодер (NTSC / PAL / SECAM) та роз'єм для входу телевізора
- 2 гігабітні Ethernet PHY з роз'ємами RJ45
- USB Host/Slave контролер з роз'ємами USB типу A і типу B
- Трансивер RS-232 та 9-контактний роз'єм

- Роз'єм для миші / клавіатури PS / 2
- ІЧ-приймач
- 2 роз'єми SMA для зовнішнього та вхідного сигналу тактового сигналу
- Один 40-контактний Expansion Header із захисними діодами
- Один роз'єм High Speed Mezzanine Card (HSMC)
- LCD-модуль 16x2

На додаток до цих апаратних функцій плата DE2-115 має програмну підтримку стандартних інтерфейсів вводу-виводу та засоби управління панеллю для доступу до різних компонентів. Також програмне забезпечення передбачено для підтримки ряду демонстрацій, що ілюструють передові можливості плати DE2-115.

Для використання плати DE2-115 користувач повинен ознайомитись з програмним забезпеченням Quartus II(Quartus Prime). На рисунку 1 зображена fpga плата.

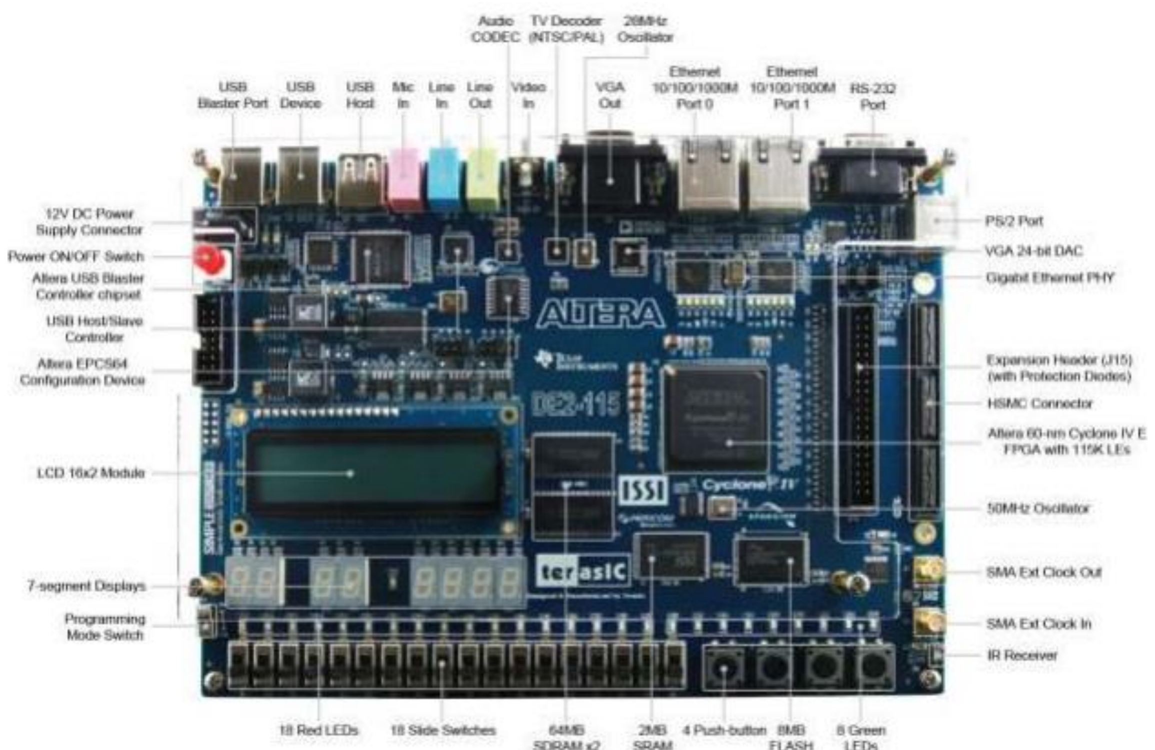


Рис. Вигляд плати de2-115

Налагоджувач Bus Blaster v3c

Bus Blaster v3c[8] - це експериментальний, швидкісний відлагоджувач JTAG від Dangerous Prototypes. Завдяки буферу для перепрограмування, просте оновлення через USB робить Bus Blaster сумісним з багатьма різними типами відлагоджувача JTAG в найпопулярнішому програмному забезпеченні з відкритим кодом.

- В основі FT2232H зі швидкісним USB 2.0
- Буферизований інтерфейс працює з пристроями від 3,3 до 1,8 вольт
- Буфер перепрограмування сумісний з декількома типами налагоджувача
- Сумісна з ПЗ OpenOCD, urJTAG
- Повинна підтримувати послідовне налагодження з проводами, якщо вони є
- Плата Mini-CPLD(Програмована логічна інтегральна схема): самопрограмована, з додатковими пінами.

На рисунку 1.5 зображена плата Bus Blaster v3c.

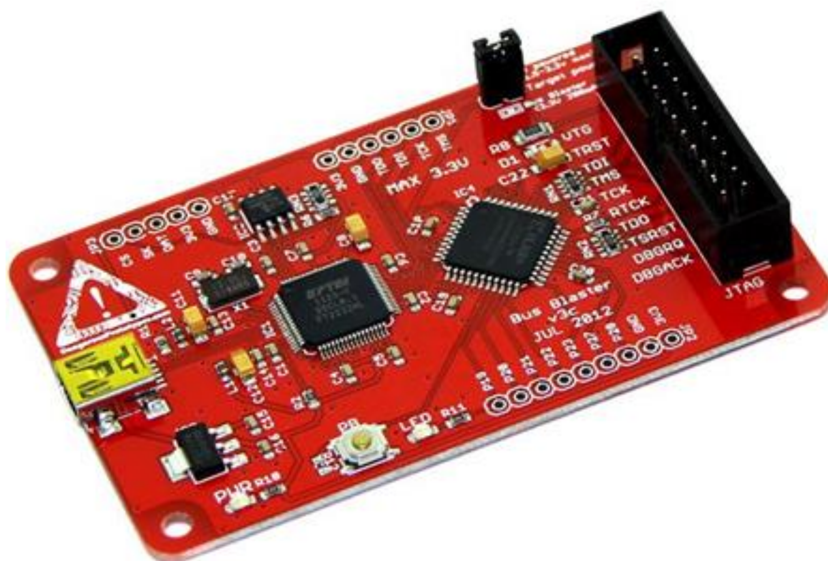


Рис. 1.5 Плата Bus Blaster v3c

Додаток В

Використане програмне забезпечення

Intel Quartus Prime

Intel Quartus Prime[6] - це середовище компанії Intel для програмування та розробки під ПЛІС. До придбання компанією Intel компанії Altera інструментом називався Altera Quartus II. Quartus Prime дозволяє аналізувати і синтезувати HDL конструкції, що дозволяє розробнику складати свої проекти, виконувати часовий аналіз, тестувати RTL діаграми, імітує реакцію дизайну на різні подразники, і налаштувати цільовий пристрій на програміста. Quartus включає в себе реалізацію VHDL та Verilog для опису апаратного забезпечення, візуального редагування логічних схем та моделювання векторних сигналів.

ModelSim

ModelSim — багатомовне середовище опису та моделювання електронного обладнання за допомогою Mentor Graphics, VHDL, Verilog і SystemC.

Забезпечує розробку описів алгоритмів роботи цифрових пристроїв, може бути підключене до систем проектування на ПЛІС.

В даній роботі використовується для моделювання роботи процесора MIPSfpga та перевірки розроблених інструкцій.

OpenOCD

OpenOCD - це безкоштовне програмне забезпечення налагодження мікросхем, внутрішньосистемне програмування та інструмент тестування типу периферійного сканування для різних систем ARM, MIPS та RISC-V.

Open On-Chip Debugger (OpenOCD) має на меті забезпечити налагодження, внутрішньосистемне програмування та тестування граничного сканування для вбудованих цільових пристроїв.

Це робиться за допомогою адаптера налагодження, який представляє собою невеликий апаратний модуль, який допомагає забезпечити правильний вид електричної сигналізації до налагодженої цілі. Вони потрібні, оскільки хост налагодження (на якому працює OpenOCD) зазвичай не має вбудованої підтримки для такої сигналізації або з'єднувача, необхідного для підключення до цільового пристрою.

MIPS SDK

Codescape MIPS SDK - це високо інтегрований набір інструментів для налагодження та розробки програм на комп’ютерних системах де виконується виконання інструкцій напряму без втручання операційної системи та GNU Linux на тренажерах MIPS та плат розробки. Створений безпосередньо MIPS, SDK підтримує новітні архітектурні розробки в ядрах MIPS до MIPSR6 ISA з розширеннями MSA.

Додаток Г

Диплом переможця Всеукраїнського конкурсу студентських робіт у 2020/2021 н.р.:

КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

Міністерство освіти і науки України

ОДЕСЬКА НАЦІОНАЛЬНА АКАДЕМІЯ ХАРЧОВИХ ТЕХНОЛОГІЙ

ДИПЛОМ

І СТУПЕНЯ

нагороджується

Воронцов Віталій Ігорович

здобувач освіти Київського національного університету імені Тараса Шевченка
переможець ІІ туру
Всеукраїнського конкурсу студентських наукових робіт у 2020/2021 н.р.
зі спеціальності «Комп'ютерна інженерія»

Науковий керівник – професор, к.т.н. Шевченко Віктор Леонідович

Голова Галузевої конкурсної комісії,
Проректор з наукової роботи
Одеської національної
академії харчових технологій

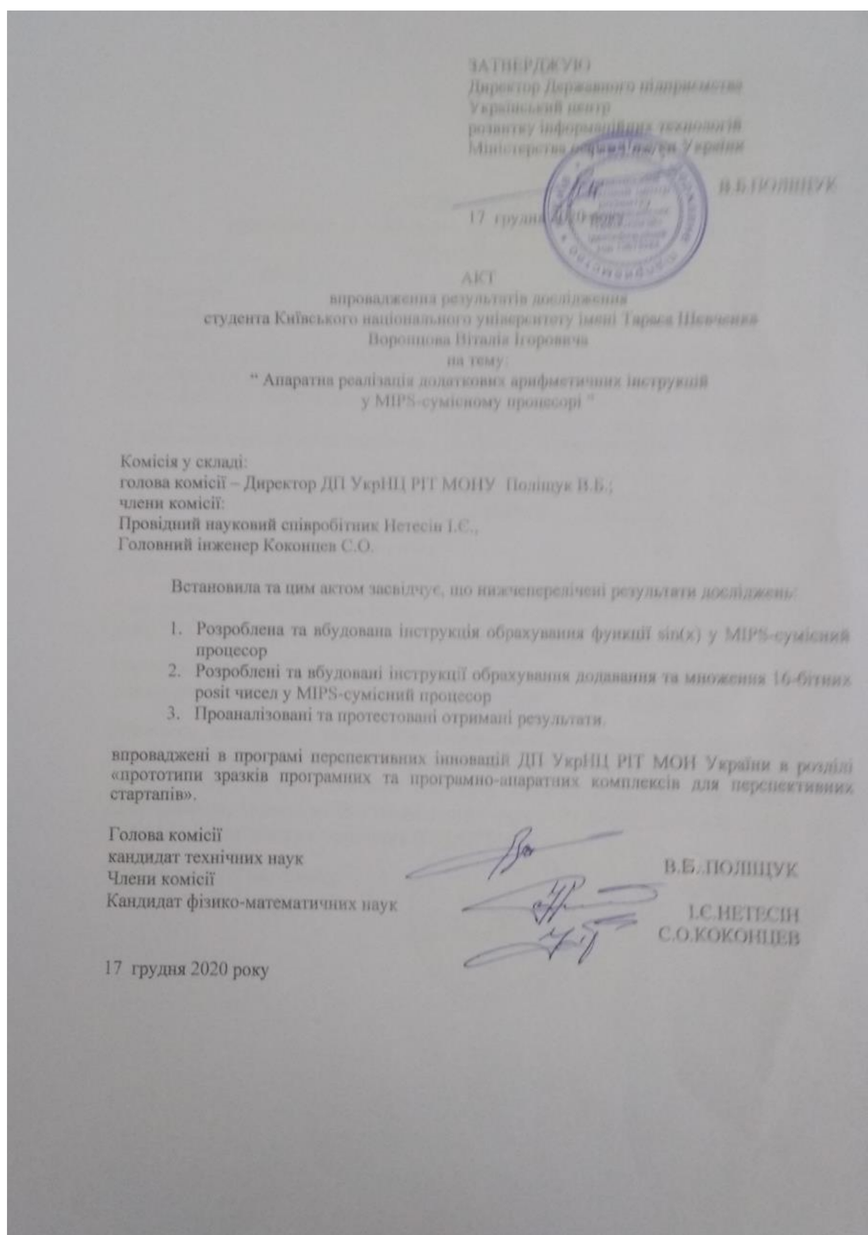
Наталя ПОВАРОВА

Одеса, 2021

Рис. Диплом переможця

Додаток Д

Акт впровадження:



ЗАТВЕРДЖУЮ
Директор Державного підприємства
Український центр
розвитку інформаційних технологій
Міністерства [illegible] науки України

В.Б.ПОЛІЩУК

17 грудн[illegible] року

АКТ
впровадження результатів дослідження
студента Київського національного університету імені Тараса Шевченка
Воронцова Віталія Ігоровича
на тему:
" Апаратна реалізація додаткових арифметичних інструкцій
у MIPS-сумісному процесорі "

Комісія у складі:
голова комісії – Директор ДП УкрНЦ РІТ МОНУ Поліщук В.Б.;
члени комісії:
Провідний науковий співробітник Нетесін І.Є.,
Головний інженер Коконцев С.О.

Встановила та цим актом засвідчує, що нижчеперелічені результати досліджень:

1. Розроблена та вбудована інструкція обрахування функції sin(x) у MIPS-сумісний процесор
2. Розроблені та вбудовані інструкції обрахування додавання та множення 16-бітних posit чисел у MIPS-сумісний процесор
3. Проаналізовані та протестовані отримані результати.

впроваджені в програмі перспективних інновацій ДП УкрНЦ РІТ МОН України в розділі «прототипи зразків програмних та програмно-апаратних комплексів для перспективних стартапів».

Голова комісії
кандидат технічних наук — В.Б. ПОЛІЩУК
Члени комісії
Кандидат фізико-математичних наук — І.Є.НЕТЕСІН
С.О.КОКОНЦЕВ

17 грудня 2020 року

Рис. Акт впровадження

Додаток Е

SAD

Taras Shevchenko National University of Kyiv

Expanding the capabilities of low-level algorithmic languages for embedded systems Software Architecture Document (SAD)

CONTENT OWNER: Vorontsov Vitalii

DOCUMENT NUMBER:	RELEASE/REVISION:	RELEASE/REVISION DATE:
• 1	• 1.0v	• 22.04.2022
•	•	•
•	•	•
•	•	•
•	•	•
•	•	•

Table of Contents

1 Documentation Roadmap 3
1.1 Document Management and Configuration Control Information 3
1.2 Purpose and Scope of the SAD 3
1.3 How the SAD Is Organized 4
1.4 Stakeholder Representation 5
1.5 Viewpoint Definitions 5
1.5.1 Logical Viewpoint Definition 6
1.5.1.1 Abstract 6
1.5.1.2 Stakeholders and Their Concerns Addressed 6
1.5.1.3 Elements, Relations, Properties, and Constraints 6
1.5.1.4 Language(s) to Model/Represent Conforming Views 6
1.5.1.5 Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria 6
1.5.1.6 Viewpoint Source 6
1.5.2 Physical Viewpoint Definition 7
1.5.2.1 Abstract 7
1.5.2.2 Stakeholders and Their Concerns Addressed 7
1.5.2.3 Elements, Relations, Properties, and Constraints 7
1.5.2.4 Language(s) to Model/Represent Conforming Views 7
1.5.2.5 Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria 7
1.5.2.6 Viewpoint Source 7
1.6 How a View is Documented 7
1.7 Relationship to Other SADs 8
1.8 Process for Updating this SAD 8
2 Architecture Background 9
2.1 Problem Background 9
2.1.1 System Overview 9
2.1.2 Goals and Context 9
2.1.3 Significant Driving Requirements 9
2.2 Solution Background 10
2.2.1 Architectural Approaches 10

2.2.2 Analysis Results 10
2.2.3 Requirements Coverage 10
2.2.4 Summary of Background Changes Reflected in Current Version 10
2.3 Product Line Reuse Considerations 10
3 Views 11
3.1 Use-case View 12
3.1.1 View Description 12
3.1.2 View Packet Overview 12
3.1.3 Architecture Background 12
3.1.4 Variability Mechanisms 12
3.1.5 View Packets 12
3.2 Architecture Layers View 13
3.2.1 View Description 13
3.2.2 View Packet Overview 13
3.2.3 Architecture Background 13
3.2.4 Variability Mechanisms 13
3.2.5 View Packets 13
3.2.5.1 View packet 1 13
3.2.5.1.1 Primary Presentation 13
3.2.5.1.2 Element Catalog 13
3.2.5.1.2.1 Elements 13
3.2.5.1.2.2 Relations 13
4 Relations Among Views 14
4.1 General Relations Among Views 14
4.2 View-to-View Relations 14
5 Referenced Materials 15
6 Directory 16
6.1 Index 16
6.2 Glossary 16
6.3 Acronym List 17
7 Sample Figures & Tables 18

List of Figures

Figure 1, 2: Architecture Layers View presentation 19

List of Tables

Table 1: 6

1 Documentation Roadmap

1.1 Document Management and Configuration Control Information

CONTENTS OF THIS SECTION: This section identifies the version, release date, and other relevant management and configuration control information associated with the current version of the document. Optional items for this section include: change history and an overview of significant changes from version to version.

- Revision Number: <<1>>
- Revision Release Date: *<< 22.04.2022>>*
- Purpose of Revision: << Outline viewpoints/ views of stakeholders>>
- Scope of Revision: *<<Viewpoints Definitions, Views>>*

1.2 Purpose and Scope of the SAD

CONTENTS OF THIS SECTION: This section explains the SAD's overall purpose and scope, the criteria for deciding which design decisions are architectural (and therefore documented in the SAD), and which design decisions are non-architectural (and therefore documented elsewhere).

This SAD specifies the software architecture for **expanding the capabilities of low-level algorithmic languages for embedded systems.** All information regarding the software architecture may be found in this document, although much information is incorporated by reference to other documents.

What is software architecture? The software architecture for a system[1] is the structure or structures of that system, which comprise software elements, the externally-visible properties of those elements, and the relationships among them [Bass 2003]. "Externally visible" properties refers to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on. This definition provides the basic litmus test for what information is included in this SAD, and what information is relegated to downstream documentation.

Elements and relationships. The software architecture first and foremost embodies information about how the elements relate to each other. This means that architecture specifically omits certain information about elements that does not pertain to their interaction. Thus, a software architecture is an *abstraction* of a system that suppresses details of elements that do not affect how they use, are used by, relate to, or interact with other elements. Elements interact with each other by means of interfaces that partition details about an element into public and private parts. Software architecture is concerned with the public side of this division, and that will be documented in this SAD accordingly. On the other hand, private details of elements—details having to do solely with internal implementation—are not architectural and will not be documented in a SAD.

Multiple structures. The definition of software architecture makes it clear that systems can and do comprise more than one structure and that no one structure holds the irrefutable claim to being the architecture. The neurologist, the orthopedist, the hematologist, and the dermatologist all take a different perspective on the structure of a human body. Ophthalmologists, cardiologists, and podiatrists concentrate on subsystems. And the kinesiologist and psychiatrist are concerned with different aspects of the entire arrangement's behavior. Although these perspectives are pictured differently and have very different properties, all are inherently related; together they describe the architecture of the human body. So it is

[1] Here, a system may refer to a system of systems.

with software. Modern systems are more than complex enough to make it difficult to grasp them all at once. Instead, we restrict our attention at any one moment to one (or a small number) of the software system's structures. To communicate meaningfully about an architecture, we must make clear which structure or structures we are discussing at the moment—which *view* we are taking of the architecture. Thus, this SAD follows the principle that documenting a software architecture is a matter of documenting the relevant views and then documenting information that applies to more than one view.

For example, all non-trivial software systems are partitioned into implementation units; these units are given specific responsibilities, and are the basis of work assignments for programming teams. This kind of element will comprise programs and data that software in other implementation units can call or access, and programs and data that are private. In large projects, the elements will almost certainly be subdivided for assignment to sub-teams. This is one kind of structure often used to describe a system. It is a very static structure, in that it focuses on the way the system's functionality is divided up and assigned to implementation teams.

Other structures are much more focused on the way the elements interact with each other at runtime to carry out the system's function. Suppose the system is to be built as a set of parallel processes. The set of processes that will exist at runtime, the programs in the various implementation units described previously that are strung together sequentially to form each process, and the synchronization relations among the processes form another kind of structure often used to describe a system.

None of these structures alone is *the* architecture, although they all convey architectural information. The architecture consists of these structures as well as many others. This example shows that since architecture can comprise more than one kind of structure, there is more than one kind of element (e.g., implementation unit and processes), more than one kind of interaction among elements (e.g., subdivision and synchronization), and even more than one context (e.g., development time versus runtime). By intention, the definition does not specify what the architectural elements and relationships are. Is a software element an object? A process? A library? A database? A commercial product? It can be any of these things and more.

These structures will be represented in the views of the software architecture that are provided in Section 3.

Behavior. Although software architecture tends to focus on structural information, *behavior of each element is part of the software architecture* insofar as that behavior can be observed or discerned from the point of view of another element. This behavior is what allows elements to interact with each other, which is clearly part of the software architecture and will be documented in the SAD as such. Behavior is documented in the element catalog of each view.

1.3 How the SAD Is Organized

CONTENTS OF THIS SECTION: This section provides a narrative description of the major sections of the SAD and the overall contents of each. Readers seeking specific information can use this section to help them locate it more quickly.

This SAD is organized into the following sections:

- **Section 1 ("Documentation Roadmap") provides information about this document and its intended audience**. It provides the roadmap and document overview. Every reader who wishes to find information relevant to the software architecture described in this document should begin by reading Section 1, which describes how the document is organized, which stakeholder viewpoints are represented, how stakeholders are expected to use it, and where information may be found. Section 1 also provides information about the views that are used by this SAD to communicate the software architecture.

- **Section 2 (“Architecture Background”) explains why the architecture is what it is.** It provides a system overview, establishing the context and goals for the development. It describes the background and rationale for the software architecture. It explains the constraints and influences that led to the current architecture, and it describes the major architectural approaches that have been utilized in the architecture. It includes information about evaluation or validation performed on the architecture to provide assurance it meets its goals.
- **Section 3 (“Views”) and Section 4 (“Relations Among Views”) specify the software architecture**. Views specify elements of software and the relationships between them. A view corresponds to a viewpoint (see Section 1.5), and is a representation of one or more structures present in the software (see Section 1.2).
- **Sections 5 (“Referenced Materials”) and 6 (“Directory”) provide reference information for the reader.** Section 5 provides look-up information for documents that are cited elsewhere in this SAD. Section 6 is a *directory*, which is an index of architectural elements and relations telling where each one is defined and used in this SAD. The section also includes a glossary and acronym list.

1.4 Stakeholder Representation

This section provides a list of the stakeholder roles considered in the development of the architecture described by this SAD. For each, the section lists the concerns that the stakeholder has that can be addressed by the information in this SAD.

Each stakeholder of a software system—customer, user, project manager, coder, analyst, tester, and so on—is concerned with different characteristics of the system that are affected by its software architecture. For example, the user is concerned that the system is reliable and available when needed; the customer is concerned that the architecture can be implemented on schedule and to budget; the manager is worried (in addition to cost and schedule) that the architecture will allow teams to work largely independently, interacting in disciplined and controlled ways. The developer is worried about strategies to achieve all of those goals. The security analyst is concerned that the system will meet its information assurance requirements, and the performance analyst is similarly concerned with it satisfying real-time deadlines.

This information is represented as a matrix, where the rows list stakeholder roles, the columns list concerns, and a cell in the matrix contains an indication of how serious the concern is to a stakeholder in that role. This information is used to motivate the choice of viewpoints chosen in Section 1.5.

1.5 Viewpoint Definitions

CONTENTS OF THIS SECTION: This section provides a short textual definition of a viewpoint and how the concept is used in this SAD. The section describes viewpoints that may be used in the SAD. The specific viewpoints will be tailored by the organization.

The SAD employs a stakeholder-focused, multiple view approach to architecture documentation, as required by ANSI/IEEE 1471-2000, the recommended best practice for documenting the architecture of software-intensive systems [IEEE 1471].

As described in Section 1.2, a software architecture comprises more than one software structure, each of which provides an engineering handle on different system qualities. A *view* is the specification of one or more of these structures, and documenting a software architecture, then, is a matter of documenting the relevant views and then documenting information that applies to more than one view [Clements 2002].

ANSI/IEEE 1471-2000 provides guidance for choosing the best set of views to document, by bringing stakeholder interests to bear. It prescribes defining a set of viewpoints to satisfy the stakeholder

community. A viewpoint identifies the set of concerns to be addressed, and identifies the modeling techniques, evaluation techniques, consistency checking techniques, etc., used by any conforming view. A view, then, is a viewpoint applied to a system. It is a representation of a set of software elements, their properties, and the relationships among them that conform to a defining viewpoint. Together, the chosen set of views show the entire architecture and all of its relevant properties. A SAD contains the viewpoints, relevant views, and information that applies to more than one view to give a holistic description of the system.

The remainder of Section 1.5 defines the viewpoints used in this SAD. The following table summarizes the stakeholders in this project and the viewpoints that have been included to address their concerns.

Table 1: Stakeholders and Relevant Viewpoints

Stakeholder	**Viewpoint(s) that apply to that class of stakeholder's concerns**
Software developer	Logical Viewpoint
Hardware engineer	Physical Viewpoint, Logical Viewpoint

1.5.1 Logical Viewpoint Definition

1.5.1.1 Abstract

Defines views, applicable to the developer.

1.5.1.2 Stakeholders and Their Concerns Addressed

Software developer - performance, functionality.
Hardware developer - security, performance, data consistency, extendability.

1.5.1.3 Elements, Relations, Properties, and Constraints

Use-cases, Models, instructions

1.5.1.4 Language(s) to Model/Represent Conforming Views

UML, Visual elements

1.5.1.5 Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria

1.5.1.6 Viewpoint Source

1.5.2 Physical Viewpoint Definition

1.5.2.1 Abstract

Defines physical implementation, applicable to the hardware engineer.

1.5.2.2 Stakeholders and Their Concerns Addressed

Hardware engineer - security, performance, deployment

1.5.2.3 Elements, Relations, Properties, and Constraints

Project consists of 3 primary components:

- MIPS processor developed on Hardware Description Languages
- Developed user-defined instructions on Verilog
- FPGA board

1.5.2.4 Language(s) to Model/Represent Conforming Views

Project built using VHDL, Verilog, System Verilog and C++.

1.5.2.5 Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria

1.5.2.6 Viewpoint Source

1.6 How a View is Documented

Section 3 of this SAD contains one view for each viewpoint listed in Section 1.5. Each view is documented as a set of view packets. A view packet is the smallest bundle of architectural documentation that might be given to an individual stakeholder.

Each view is documented as follows, where the letter *i* stands for the number of the view: 1, 2, etc.:

- Section 3.i: Name of view.
- Section 3.i.1: View description. This section describes the purpose and contents of the view. It should refer to (and match) the viewpoint description in Section 1.5 to which this view conforms.
- Section 3.i.2: View packet overview. This section shows the set of view packets in this view, and provides rationale that explains why the chosen set is complete and non-duplicative. The set of view packets may be listed textually, or shown graphically in terms of how they partition the entire architecture being shown in the view.
- Section 3.i.3: Architecture background. Whereas the architecture background of Section 2 pertains to those constraints and decisions whose scope is the entire architecture, this section provides any architecture background (including significant driving requirements, design approaches, patterns, analysis results, and requirements coverage) that applies to this view.
- Section 3.i.4: Variability mechanisms. This section describes any architectural variability mechanisms (e.g., adaptation data, compile-time parameters, variable replication, and so forth) described by this view, including a description of how and when those mechanisms may be exercised and any constraints on their use.

- Section 3.i.5: View packets. This section presents all of the view packets given for this view. Each view packet is described using the following outline, where the letter *j* stands for the number of the view packet being described: 1, 2, etc.

− Section 3.i.5.j: View packet #j.

− Section 3.i.5.j.1: Primary presentation. This section presents the elements and the relations among them that populate this view packet, using an appropriate language, languages, notation, or tool-based representation.

− Section 3.i.5.j.2: Element catalog. Whereas the primary presentation shows the important elements and relations of the view packet, this section provides additional information needed to complete the architectural picture. It consists of the following subsections:

  − Section 3.i.5.j.2.1: Elements. This section describes each element shown in the primary presentation, details its responsibilities of each element, and specifies values of the elements' relevant *properties*, which are defined in the viewpoint to which this view conforms.

  − Section 3.i.5.j.2.2: Relations. This section describes any additional relations among elements shown in the primary presentation, or specializations or restrictions on the relations shown in the primary presentation.

1.7 Relationship to Other SADs

CONTENTS OF THIS SECTION: This section describes the relationship between this SAD and other architecture documents, both system and software. For example, a large project may choose to have one SAD that defines the system-of-systems architecture, and other SADs to define the architecture of systems or subsystems. An embedded system may well have a *system* architecture document, in which case this section would explain how the information in here traces to information there.

If none, say "Not applicable."

Not applicable.

1.8 Process for Updating this SAD

CONTENTS OF THIS SECTION: This section describes the process a reader should follow to report discrepancies, errors, inconsistencies, or omissions from this SAD. The section also includes necessary contact information for submitting the report. If a form is required, either a copy of the blank form that may be photocopied is included, or a reference to an online version is provided. This section also describes how error reports are handled, and how and when a submitter will be notified of the issue's disposition.

Not applicable.

2 Architecture Background

2.1 Problem Background

CONTENTS OF THIS SECTION: The sub-parts of Section 2.1 explain the constraints that provided the significant influence over the architecture.

The general purpose of the project is to increase the execution speed of CPU instructions suck as the sin (x) function using the float32 number format, add arithmetic posit operations to the MIPS-compatible processor to improve the current implementation of the sin (x) calculation instruction. Further implementation of instructions can be used to make significant improvements on performance compared to conventional solutions.

2.1.1 System Overview

CONTENTS OF THIS SECTION: This section describes the general function and purpose for the system or subsystem whose architecture is described in this SAD.

System provides new instructions for processor and can be used on C++ MIPS compiler. General purpose of this project is to create a new instructions and improve performance in embedded systems in specific scenarios.

2.1.2 Goals and Context

CONTENTS OF THIS SECTION: This section describes the goals and major contextual factors for the software architecture. The section includes a description of the role software architecture plays in the life cycle, the relationship to system engineering results and artifacts, and any other relevant factors.

The goal is to implement user instructions in processor and make this instructions usable from ++ environment.

2.1.3 Significant Driving Requirements

CONTENTS OF THIS SECTION: This section describes behavioral and quality attribute requirements (original or derived) that shaped the software architecture. Included are any scenarios that express driving behavioral and quality attribute goals, such as those crafted during a Quality Attribute Workshop (QAW) [Barbacci 2003] or software architecture evaluation using the Architecture Tradeoff Analysis Method[2]SM (ATAM[SM]) [Bass 2003].

1. Analysis of the CorExtend block in the MIPSfpga kernel

2. FPGA analysis of the Altera DE2-115 board

3. Research of float32 format

4. Investigation of methods for calculating the function sin (x)

5. Research of posit-arithmetic

2.2 Solution Background

CONTENTS OF THIS SECTION: The sub-parts of Section 2.2 provide a description of why the architecture is the way that it is, and a convincing argument that the architecture is the right one to satisfy the behavioral and quality attribute goals levied upon it.

2.2.1 Architectural Approaches

CONTENTS OF THIS SECTION: This section provides a rationale for the major design decisions embodied by the software architecture. It describes any design approaches applied to the software architecture, including the use of architectural styles or design patterns, when the scope of those approaches transcends any single architectural view. The section also provides a rationale for the selection of those approaches. It also describes any significant alternatives that were seriously considered and why they were ultimately rejected. The section describes any relevant COTS issues, including any associated trade studies.

Hardware on-board development and simulator was used as a prime approach for developing the architecture of the system under design.

2.2.2 Analysis Results

CONTENTS OF THIS SECTION: This section describes the results of any quantitative or qualitative analyses that have been performed that provide evidence that the software architecture is fit for purpose. If an Architecture Tradeoff Analysis Method evaluation has been performed, it is included in the analysis sections of its final report. This section refers to the results of any other relevant trade studies, quantitative modeling, or other analysis results.

Instructions for calculating sin (x), addition and multiplication operations using posit arithmetic are obtained. The results can be used to further improve instructions and implement new processor instructions.

2.2.3 Requirements Coverage

CONTENTS OF THIS SECTION: This section describes the requirements (original or derived) addressed by the software architecture, with a short statement about where in the architecture each requirement is addressed.

In order for the solution to function it needs:

- a hardware programmable hardware, e.g. FPGA, processors with built-in FPGA
- reasonable processing power and enough quantity of logic elements
- C++ compiler installed

2.2.4 Summary of Background Changes Reflected in Current Version

CONTENTS OF THIS SECTION: For versions of the SAD after the original release, this section summarizes the actions, decisions, decision drivers, analysis and trade study results that became decision drivers, requirements changes that became decision drivers, and how these decisions have caused the architecture to evolve or change.

2.3 Product Line Reuse Considerations

Not Applicable.

3 Views

CONTENTS OF THIS SECTION: The sub-parts of Section 3 specify the views corresponding to the viewpoints listed in Section 1.5.

This section contains the views of the software architecture. A view is a representation of a whole system from the perspective of a related set of concerns [IEEE 1471]. Concretely, a view shows a particular type of software architectural elements that occur in a system, their properties, and the relations among them. A view conforms to a defining viewpoint.

Architectural views can be divided into three groups, depending on the broad nature of the elements they show. These are:

- Module views. Here, the elements are modules, which are units of implementation. Modules represent a code-based way of considering the system. Modules are assigned areas of functional responsibility, and are assigned to teams for implementation. There is less emphasis on how the resulting software manifests itself at runtime. Module structures allow us to answer questions such as: What is the primary functional responsibility assigned to each module? What other software elements is a module allowed to use? What other software does it actually use? What modules are related to other modules by generalization or specialization (i.e., inheritance) relationships?
- Component-and-connector views. Here, the elements are runtime components (which are principal units of computation) and connectors (which are the communication vehicles among components). Component and connector structures help answer questions such as: What are the major executing components and how do they interact? What are the major shared data stores? Which parts of the system are replicated? How does data progress through the system? What parts of the system can run in parallel? How can the system's structure change as it executes?
- Allocation views. These views show the relationship between the software elements and elements in one or more external environments in which the software is created and executed. Allocation structures answer questions such as: What processor does each software element execute on? In what files is each element stored during development, testing, and system building? What is the assignment of the software element to development teams?

These three kinds of structures correspond to the three broad kinds of decisions that architectural design involves:

- How is the system to be structured as a set of code units (modules)
- How is the system to be structured as a set of elements that have run-time behavior (components) and interactions (connectors) ?
- How is the system to relate to non-software structures in its environment (such as CPUs, file systems, networks, development teams, etc.)?

Often, a view shows information from more than one of these categories. However, unless chosen carefully, the information in such a hybrid view can be confusing and not well understood.

The views presented in this SAD are the following:

Name of view	Viewtype that defines this view	Types of elements and relations shown		Is this a module view?	Is this a component-and-connector view?	Is this an allocation view?
Use-case	Logical Viewporint	Text	Text	No	No	No
Architecture Layers	Physical Viewpoint	Visual	Visual	Yes	No	No

3.1 Use-case View

CONTENTS OF THIS SECTION: For each view documented in this SAD, the sub-parts of Section 3.1 specify it using the outline given in Section 1.6. This part of the template assumes you are using view packets to divide up a view into management chunks. If not, then see the note in Section 1.6 as to what outline to use for each view.

3.1.1 View Description

Describes scenarios of use of functionality of the system, conforming to Logical Viewpoint.

3.1.2 View Packet Overview

This solution is using in embedded systems. Should be used after hardware preparation by calling developed instruction from compiler in run-time.

3.1.3 Architecture Background

View is developed with the driving requirements in mind.

3.1.4 Variability Mechanisms

Main variability mechanisms are graphically represented and outlined as configuration flow.

3.1.5 View Packets

CONTENTS OF THIS SECTION: For each view packet in the view, this section describes it using the outline given in Section 1.6.

3.2 Architecture Layers View

CONTENTS OF THIS SECTION: For each view documented in this SAD, the sub-parts of Section 3.1 specify it using the outline given in Section 1.6. This part of the template assumes you are using view packets to divide up a view into management chunks. If not, then see the note in Section 1.6 as to what outline to use for each view.

3.2.1 View Description

Describes the layers of composition, of which the architecture is composed.

3.2.2 View Packet Overview

This view has been divided into the following view packets for convenience of presentation:

View Packet 1

3.2.3 Architecture Background

View is developed with the driving requirements in mind. It complies with the concerns separation principle.

3.2.4 Variability Mechanisms

Not applicable.

3.2.5 View Packets

CONTENTS OF THIS SECTION: For each view packet in the view, this section describes it using the outline given in Section 1.6.

3.2.5.1 View packet 1

3.2.5.1.1 Primary Presentation

Visual presentation of architecture layers and relations between them.

3.2.5.1.2 Element Catalog

Figure 1, Figure 2

3.2.5.1.2.1 Elements

Architecture layer

3.2.5.1.2.2 Relations

Utilize flow

4 Relations Among Views

Each of the views specified in Section 3 provides a different perspective and design handle on a system, and each is valid and useful in its own right. Although the views give different system perspectives, they are not independent. Elements of one view will be related to elements of other views.

4.1 General Relations Among Views

CONTENTS OF THIS SECTION: This section describes the general relationship among the views chosen to represent the architecture. Also in this section, consistency among those views is discussed and any known inconsistencies are identified.

Each View described the system from the point of view.

4.2 View-to-View Relations

CONTENTS OF THIS SECTION: For each set of views related to each other, this section shows how the elements in one view are related to elements in another.

5 Referenced Materials

CONTENTS OF THIS SECTION: This section provides citations for each reference document. Provide enough information so that a reader of the SAD can be reasonably expected to locate the document.

Morgan Kaufan, 2015	David M. Harris, Sarah L. Harris *Digital Design and Computer Architecture*
Springer, 2005	Uwe Meyer-Baese *Digital Signal Processing with Field Programmable Gate Arrays Fourth Edition*
Clements, 2002	Clements, Bachmann, Bass, Garlan, Ivers, Little, Nord, Stafford *Documenting Software Architectures: Views and Beyond*
IMAGINATION TECHNOLOGIES PROPRIETARY / CONFIDENTIAL	*MIPS32® MicroAptiv™ UP Processor Core Family Integrator's Guide Micro Aptiv Integrator's MD0094*
IEEE 1471	ANSI/IEEE-1471-2000, *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, 21 September 2000.

6 Directory

6.1 Index

CONTENTS OF THIS SECTION: This section provides an index of all element names, relation names, and property names. For each entry, the following are identified:

- the location in the SAD where it was defined
- each place it was used

Ideally, each entry will be a hyperlink so a reader can instantly navigate to the indicated location.

6.2 Glossary

CONTENTS OF THIS SECTION: This section provides a list of definitions of special terms and acronyms used in the SAD. If terms are used in the SAD that are also used in a parent SAD and the definition is different, this section explains why.

Term	Definition
software architecture	The structure or structures of that system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [Bass 2003]. "Externally visible" properties refer to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on.
view	A representation of a whole system from the perspective of a related set of concerns [IEEE 1471]. A representation of a particular type of software architectural elements that occur in a system, their properties, and the relations among them. A view conforms to a defining viewpoint.
view packet	The smallest package of architectural documentation that could usefully be given to a stakeholder. The documentation of a view is composed of one or more view packets.
viewpoint	A specification of the conventions for constructing and using a view; a pattern or template from which to develop individual views by establishing the purposes and audience for a view, and the techniques for its

	creation and analysis [IEEE 1471]. Identifies the set of concerns to be addressed, and identifies the modeling techniques, evaluation techniques, consistency checking techniques, etc., used by any conforming view.

6.3 Acronym List

API	Application Programming Interface; Application Program Interface; Application Programmer Interface
ATAM	Architecture Tradeoff Analysis Method
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
CORBA	Common object request broker architecture
COTS	Commercial-Off-The-Shelf
EPIC	Evolutionary Process for Integrating COTS-Based Systems
FPGA	Field-programmable gate array
IEEE	Institute of Electrical and Electronics Engineers
KPA	Key Process Area
MIPS	Microprocessor without Interlocked Pipelined Stages
OO	Object Oriented
ORB	Object Request Broker
OS	Operating System
QAW	Quality Attribute Workshop
RUP	Rational Unified Process
SAD	Software Architecture Document
SDE	Software Development Environment
SEE	Software Engineering Environment
SEI	Software Engineering Institute Systems Engineering & Integration Software End Item
SEPG	Software Engineering Process Group
SLOC	Source Lines of Code
UML	Unified Modeling Language
UDI	User defined instructions

7 Sample Figures & Tables

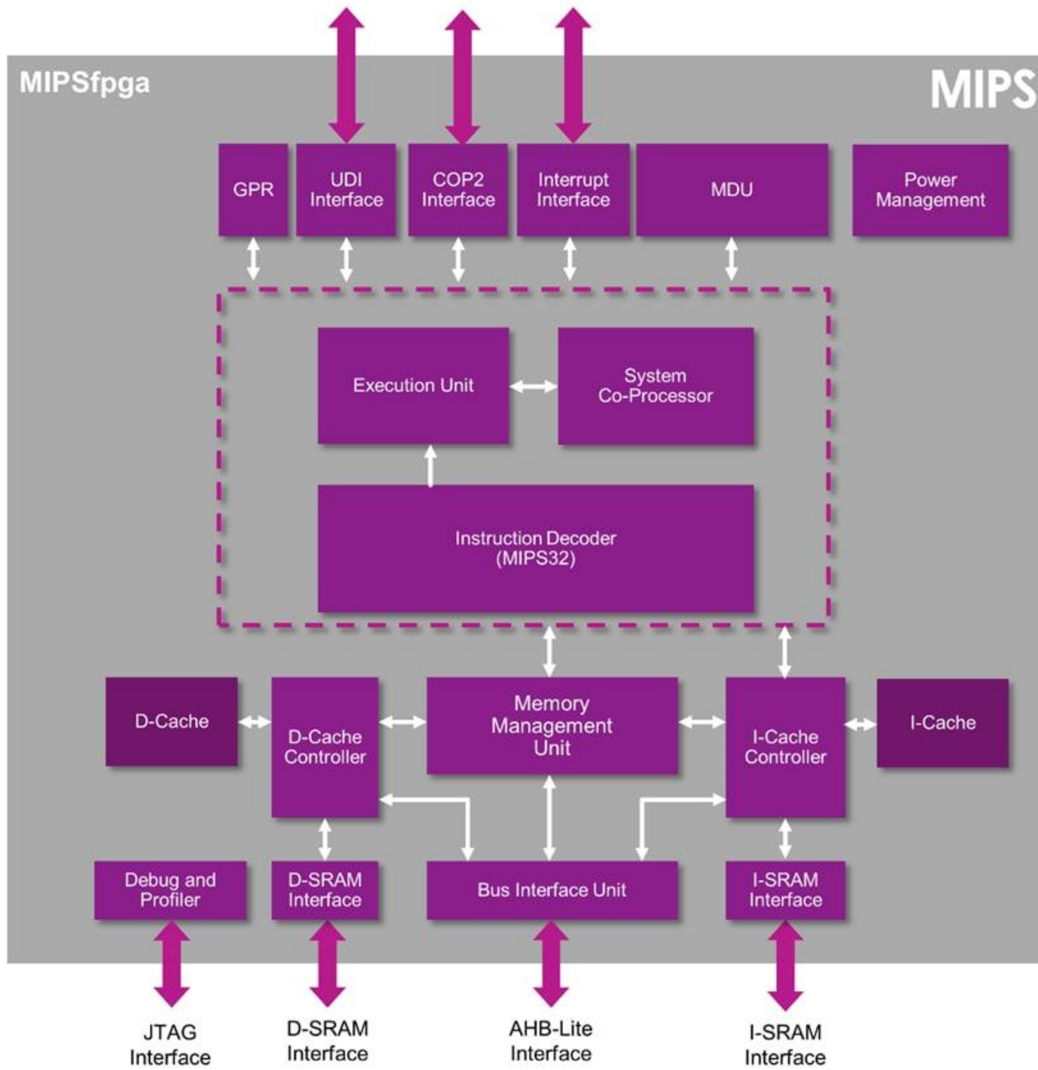


Figure 1: Visual presentation of architecture logical layers and relations between them

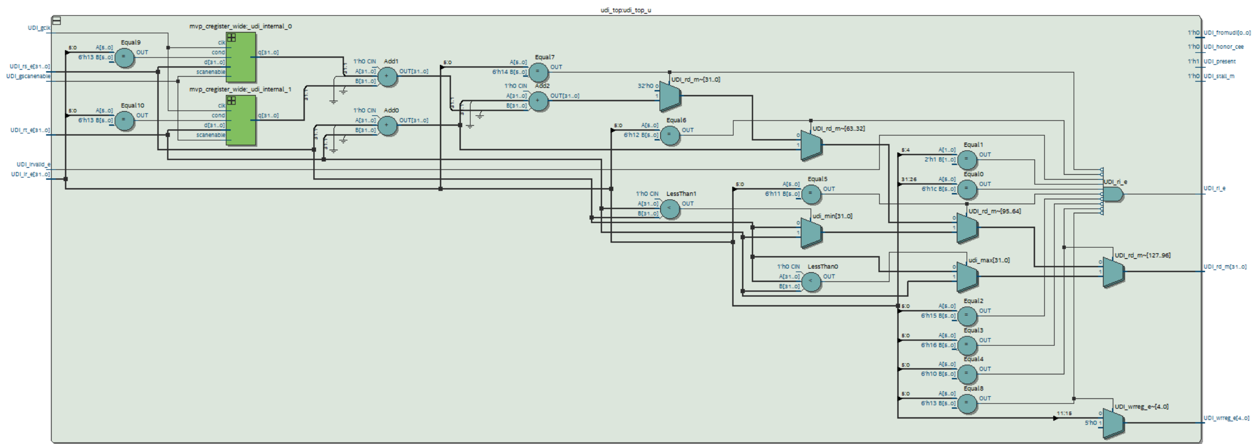


Figure 2: Logical implementation of one of the developed instructions