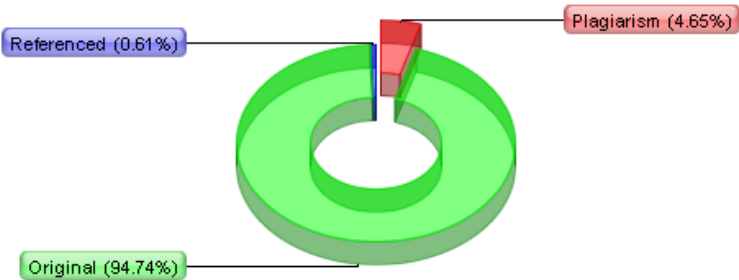


Детектор Плагиата v. 2215 - Отчёт оригинальности: 18.11.2024 13:41:39

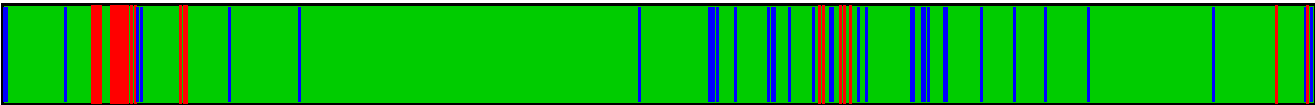
Проанализированный документ: Каневський Сергій Олексійович\_ІПЗ.docx  
Лицензия: ВОЛОДИМИР МАТІЄВСЬКИЙ\_License2

Тип поиска: Поиск переписанного  
Язык: Uk  
Тип проверки: Интернет  
ТЭЕ и кодировка: DocX n/a

Детальный анализ тела документа:  
Диаграмма соотношения частей:



Граф распределения зон:



Источники плагиата: 11

2%	391	1. <a href="http://ni.biz.ua/9/9_17/9_17289_klassifikatsiya-po-arhitekturnomu-priznaku.html">http://ni.biz.ua/9/9_17/9_17289_klassifikatsiya-po-arhitekturnomu-priznaku.html</a>
1%	264	2. <a href="https://learn.ztu.edu.ua/pluginfile.php/317431/mod_folder/content/0/2.2_Лекція_2_Класифікація_цифрових_IC.doc">https://learn.ztu.edu.ua/pluginfile.php/317431/mod_folder/content/0/2.2_Лекція_2_Класифікація_цифрових_IC.doc</a>
1%	248	3. <a href="https://studfile.net/preview/5203878/page:2/">https://studfile.net/preview/5203878/page:2/</a>

Детали обработанных ресурсов: 452 - ОК / 12 - Ошибок

Важные замечания:

Википедия:	Google Книги:	Сервисы платных работ:	Античит:
[не обнаружено]	[не обнаружено]	[не обнаружено]	Обнаружено соккрытие!

Античит-отчет UACE:

- 1. Статус: Анализатор Включен Нормализатор Включен сходство символов установлено на 100%
- 2. Обнаруженный процент загрязнения UniCode: 5,6% с лимитом: 4%
- 3. Процент нераспознанных символов после нормализации: 3%
- 4. Все подозрительные символы будут отмечены фиолетовым цветом: Abcd...
- 5. Найдены невидимые символы: 0

Рекомендации по оценке:

Особое внимание следует уделить анализу этого отчета! Предполагается, что этот документ содержит значительное количество символов, чуждых языку документа. Это прямое указание на то, что автор

документа использовал специальное программное обеспечение\онлайн-веб-сервис, чтобы эффективно скрыть текст в попытке избежать обнаружения потенциального плагиата. Настоятельно рекомендуется передать это дело на более высокий уровень! В случае сомнений обращайтесь: в службу поддержки Детектора плагиата!

Алфавитная статистика и анализ символов:

 Активные ссылки (URL-адреса, извлеченные из документа):

URL не найдены

 Исключённые ресурсы:

URL не найдены

 Включённые ресурсы:

URL не найдены

Детальний аналіз документа:

Міністерство освіти і науки України Державний заклад

Цитування: 0,03%

id: 1

«Луганський національний університет імені Тараса Шевченка»

Навчально-науковий інститут математики та інформаційних технологій Кафедра інформаційних технологій та систем Каневський Сергій Олексійович АНАЛІЗ ВАРІАНТІВ ВИКОРИСТАННЯ ПЛІС-ТЕХНОЛОГІЙ ДЛЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ ПОБУДОВИ ОНТОЛОГІЙ кваліфікаційна робота здобувача вищої освіти другого (магістерського) рівня освітньої програми

Цитування: 0,01%

id: 2

«Мультимедійні системи»

за спеціальністю 121 Інженерія програмного забезпечення Особистий підпис \_\_\_\_\_  
Сергій КАНЕВСЬКИЙ Науковий керівник \_\_\_\_\_ Світлана ДОНЧЕНКО, асистент кафедри інформаційних технологій та систем В.о. завідувача кафедри \_\_\_\_\_ Микола СЕМЕНОВ, кандидат педагогічних наук, доцент кафедри інформаційних технологій та систем Полтава – 2025 АНОТАЦІЯ Каневський С. О. Тема: Аналіз варіантів використання ПЛІС-технологій для автоматизованої системи побудови онтологій. Спеціальність: 123 Комп'ютерна інженерія. Установа: ЛНУ імені Тараса Шевченка, 2025р. Магістерська робота містить: 98 с., 26 рис., 3 табл., 30 джерел. Об'єкт дослідження – автоматизовані системи побудови онтологій. Предмет дослідження – методи і засоби архітектурно-структурної організації автоматизованої системи на базі ПЛІС. Мета роботи - аналіз варіантів використання ПЛІС-технологій для автоматизованої системи побудови онтологій. Результати роботи. У роботі проведено аналіз принципів побудови та функціонування систем верифікації реконфігуруємих пристроїв на основі програмуємих логічних схем (ПЛІС), та виконано аналіз особливостей знання – орієнтованих інформаційних систем. На його основі запропоновано підхід до побудови онтологічної бази знань, яка функціонує в складі онтологокеруємої інформаційної системи. Ключові слова: ПЛІС, САПР, [FPGA](#), БІС, КОМПОНЕНТИ ОНТОЛОГІЇ. [ANNOTATION Kanevskyi Serhii Theme: Analysis of options for using FPGA technologies for an automated ontology building system. Speciality: 121 Computer Engineering. Institution: Luhansk Taras Shevchenko National University \(LTSNU\), 2025 year. Master's work of: 98 p., 26 im, 3 table, 30 sources. A research object is Automated ontology building systems. The article of research is Methods and means of architectural and structural organization of an automated system based on FPGAs. An aim of work is Analysis of options for using FPGA technologies for automated ontology building. Job performances. The analysis of the principles of constructing and functioning of reconfigurable devices verification systems based on programmable logic circuits \(FPGA\) is carried out and the analysis of knowledge features - oriented information systems is performed. On its basis, an approach to the construction of an ontological knowledge base is proposed, which functions as part of an ontologic information system. Keywords. FPGA, CAD, FPGA, BIS, COMPONENTS OF ONTOLOGY.](#) ЗМІСТ ВСТУП6 РОЗДІЛ 1. СТАН В ОБЛАСТІ ПРОЄКТУВАННЯ СИСТЕМ НА КРИСТАЛІ9 1.1.Огляд елементної бази систем на одному кристалі9 1.1.1. Класифікація цифрових інтегральних мікросхем9 1.1.2. Класифікація систем на одному кристалі13 1.1.3. Переваги та недоліки ПЛІС в якості основи реалізації систем на одному кристалі15 1.2. Основні проблеми, що виникають при проектуванні систем на одному кристалі і можливі шляхи їх вирішення17 1.2.1. Різні підходи і методології проектування окремих вузлів17 1.2.2. Види ядер багаторазового використання20 1.3. Огляд сучасних САПР ПЛІС21 1.3.1. Засоби проектування систем на одному кристалі на основі багаторазово використовуваних блоків фірми [Xilinx](#)21 1.3.2. Засоби проектування систем на одному кристалі на основі ПЛІС фірми [ALTERA](#)22 1.3.3. Засоби проектування систем на одному кристалі фірми [ATMEL](#)24 1.3.4. Програмні засоби фірми [Mentor Graphics](#)29 1.3.5. Маршрут проектування цифрових пристроїв в базисі ПЛІС31 1.4. Висновки до розділу34 РОЗДІЛ 2. РОЗРОБКА МЕТОДИКИ ПРОЄКТУВАННЯ СКЛАДНИХ ЯДЕР В БАЗИСІ ПЛІС35 2.1. Методика проектування складного ядра35 2.2. Маршрут проектування складного ядра40 2.3. Загальні правила проектування43 2.4.Правила створення ядер, використовуючи мови опису апаратури [VHDL](#)47 2.5. Висновки до розділу50 РОЗДІЛ 3. ВИКОРИСТАННЯ ПЛІС - ТЕХНОЛОГІЙ ДЛЯ АВТОМАТИЗОВАНОЇ ПОБУДОВИ ОНТОЛОГІЙ51 3.1. Системно-онтологічний аналіз предметної області51 3.1.1. Загальний підхід до проектування55 3.1.2. Побудова компонент онтології59 3.2. Системи верифікації

на основі реконфігурованих пристроїв 70 3.2.1. Системи цифрового проектування [HOT](#) 70 3.2.2. Моделююча система [System Explorer](#) 73 3.2.3. Система [FUSE](#) 78 3.2.4. Система [XtremeDSP](#) 80 3.3. Висновки до розділу 88 ВИСНОВКИ 90 СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 92 ВСТУП

Однією з важливих гілок сучасного розвитку інтелектуальних інформаційних систем є онтологізовані інформаційні системи (ОКІС). Побудова останніх тісно пов'язана з розробкою теоретичних основ і методологіями проектування, що включають формальний підхід, фундаментальні принципи та механізми, узагальнену архітектуру і структуру системи, формальну модель і методологію проектування онтології предметної області (ПрО), формальну модель подання знань, узагальнені алгоритми процедур обробки знань та ін. В свою чергу, кожна з перерахованих складових загальної методології проектування являє собою складну інформаційно-алгоритмічну структуру, наприклад, розробка онтології ПрО тісно пов'язана з концептуалізацією онтологічних категорій, розробкою і вдосконаленням ієрархічних структур сутностей на всіх рівнях, побудовою формальної системи аксіом і обмежень. Комплексне вирішення зазначених завдань проектування повинно підвищити роль онтологічних (концептуальних) знань при вирішенні конкретних завдань в прикладних областях. Компонентами онтологічної бази знань є онтологія об'єктів, онтологія процесів і онтологія завдань, для яких розроблені схеми моделей і методика проектування. СОП і відповідна технологія його реалізації припускають аналіз ПДС, витяг, уявлення і обробку предметних знань. Питання автоматизації придбання знань, зокрема з безлічі природно-мовних об'єктів, залишається як і раніше актуальним. Реконфігурована обробка даних ([reconfigurable computing](#)), є новітньою технологією, яка використовує кристали ПЛІС типу [FPGA \(Field Programmable Gate Array\)](#) для апаратної реалізації алгоритмів [1]. На відміну від традиційної Фон-Неймановської архітектури, що забезпечує універсальне рішення для всіх алгоритмів, архітектура, заснована на кристалах [FPGA](#), дозволяє розробнику проектувати структуру для кожного окремого алгоритму. Логічна щільність або кількість вентилів кристалів [FPGA](#) визначає, якої складності алгоритм може бути реалізований. Алгоритм може також бути розбитий на фрагменти, які багаторазово передаються в той же самий пристрій, приводячи до техніки, званої реконфігурацією під час виконання. Таким чином, технологія

Цитування: 0,01%

id: 3

"[reconfigurable computing](#)"

представляє зміни парадигми в проектуванні комп'ютера. Проблема об'єднання апаратних і програмних засобів є більш значущою, ніж збільшення логічної щільності і розширення області додатків. На відміну від систем проектування минулого нові платформи повинні задовольняти потребам і навичкам фахівців - інженерів-розробників апаратних і програмних засобів. Використання таких платформ робить можливим верифікацію проєктів в реальному масштабі часу за рахунок об'єднання розроблюваних апаратних засобів системи з навколишнім середовищем. Об'єкт дослідження - автоматизовані системи побудови онтологій. Предмет дослідження - методи і засоби архітектурно - структурної організації автоматизованої системи на базі ПЛІС. Мета роботи - аналіз варіантів використання ПЛІС - технологій для автоматизованої побудови онтологій. Для досягнення даної мети в роботі ставляться і вирішуються такі завдання: Дослідження сучасного процесу проектування в САПР ПЛІС і елементного базису ПЛІС. Дослідження існуючих методик проектування ядер багаторазового використання в базисі ПЛІС, що застосовуються в сучасних системах САПР. Розробка методики проектування складних цифрових ядер багаторазового використання в базисі ПЛІС. Аналіз особливостей знання-орієнтованих інформаційних систем. Аналіз принципів побудови і функціонування сучасних систем верифікації реконфігурованих пристроїв на основі програмованих логічних інтегральних схем (ПЛІС). У першому розділі наведено огляд засобів та ініціатив, спрямованих на реалізацію систем на одному кристалі, в тому числі і в базисі ПЛІС. Наводиться класифікація СОК. Розглядаються основні проблеми, що виникають при проектуванні систем на одному кристалі і можливі шляхи їх вирішення. Виділяється важливість застосування ядер багаторазового використання як найважливішого елемента при створенні СОК. Обґрунтовується необхідність створення методики проектування складних ядер багаторазового використання. У другому розділі розглядаються теоретичні питання проектування ядер багаторазового використання. Пропонується методика проектування складних ядер. Описується маршрут проектування складних ядер багаторазового використання в базисі ПЛІС. Виділяються загальні правила проектування і правила, що враховують реалізацію на мові [VHDL](#). Виділяються критерії ефективності ядер.

У третьому розділі проведено аналіз особливостей знання-орієнтованих інформаційних систем. На його основі запропоновано загальний підхід до побудови онтологічної бази знань, що функціонує в складі онтологізованої інформаційної системи, який названий системно-онтологічним. Виконано аналіз принципів побудови і функціонування сучасних систем верифікації реконфігурованих пристроїв на основі програмованих логічних інтегральних схем (ПЛІС). РОЗДІЛ 1.

СТАН В ОБЛАСТІ ПРОЄКТУВАННЯ СИСТЕМ НА КРИСТАЛІ 1.1.Огляд елементної бази систем на одному кристалі 1.1.

 Обнаружен Плагиат: **0,9%** <https://learn.ztu.edu.ua/pluginfile.php...>

id: 4

1. Класифікація цифрових інтегральних мікросхем Елементну базу електронної апаратури обробки інформації та її зберігання складають інтегральні схеми (IC). Залежно від типу сигналів засоби обробки інформації поділяють на цифрові, аналогові та цифроаналогові. Класифікація цифрових IC [6] наведена нижче (рис. 1.1.). Рис. 1.1. Класифікація цифрових інтегральних мікросхем Перш за все, цифрові IC (IC - [integrated circuit](#)) поділяють по типу функцій, тобто орієнтована дана IC на масове споживання або на конкретне замовлення, звідси впливає поділ на стандартні і спеціалізовані IC. Стандартні функції, які реалізуються в мікросхемах, купуються споживачем як готові вироби і виробляються масовими партіями, що дозволяє витратити великі кошти на їх проєктування, оскільки його вартість розкладається на велике число виробів. Стандартні IC традиційних видів мають практично жорстку внутрішню структуру, і споживач не може впливати на характер їх функціонування. Спеціалізовані IC ([ASIC - Applications Specific Integrated Circuit](#)), мають індивідуальний характер функціонування, їх доводиться розробляти (проєктувати) на конкретне замовлення. Проєктування IC - процес складний і дорогий, тому зрозуміле прагнення в максимально можливій мірі будувати апаратуру на основі стандартних IC.

Мікросхеми, програмовані користувачами, відкрили нову сторінку в історії сучасної мікроелектроніки та обчислювальної техніки. Вони зробили БІС / НВІС, призначені для вирішення спеціалізованих завдань, стандартною продукцією електронної промисловості (що на малюнку позначено пунктирною стрілкою) з усіма що впливають з цього позитивними наслідками: масове виробництво, зниження вартості мікросхем, термінів розробки і виходу на ринок продукції на їх основі. Виконаємо класифікацію ПЛІС за такими ознаками: по архітектурі; за рівнем інтеграції та однорідності / гібридності;

 Обнаружен Плагиат: **1,65%** [http://ni.biz.ua/9/9\\_17/9\\_17289\\_klas...](http://ni.biz.ua/9/9_17/9_17289_klas...) + 4 ресурсів!

id: 5

У класифікації за першою ознакою (рис. 1.2.) ПЛІС розділені на три основні класи [6,31]. Рис. 1.2. Класифікація ПЛІС з архітектури Першими з подібних мікросхем з'явилися [SPLD](#) (не показано на рисунок 1.2, оскільки в даний час вже не використовуються), [Simple Programmable Logic Devices](#), тобто прості програмовані логічні пристрої. За архітектурою ці ПЛІС діляться на підкласи програмованих логічних матриць ПЛМ ([PLA, Programmable Logic Arrays](#)) і програмованої матричної логіки ПМЛ ([PAL, Programmable Arrays Logic](#), або [GAL, Generic Array Logic](#)). Обидва ці підкласи мікросхем реалізують диз'юнктивні нормальні форми (ДНФ) переключальних функцій, а їх основними блоками є дві матриці: матриця елементів І і матриця елементів АБО, включені послідовно. У складних програмованих логічних схемах [CPLD \(Complex Programmable Logic Devices\)](#) кілька блоків, подібних ПМЛ ([PLA, Programmable Logic Arrays](#)), об'єднуються засобами програмованої комутаційної матриці. Основними блоками є дві матриці: матриця елементів І і матриця елементів АБО, включені послідовно. У [CPLD](#) можуть входити сотні блоків і десятки тисяч еквівалентних вентилів. Архітектури [CPLD](#) розробляються фірмами [Altera](#), [Atmel](#), [Lattice Semiconductor](#), [Cypress Semiconductor](#), [Xilinx](#) і ін. Впливаючи на програмовані з'єднання комутаційної матриці і ПМЛ, що входять до складу [CPLD](#), можна реалізувати необхідну схему. Мікросхеми програмовані користувачами вентильних матриць [FPGA \(Field Programmable Gate Arrays\)](#) в своїй основі складаються з великого числа конфігурованих логічних блоків, розташованих по рядках і стовпцях у вигляді матриці, і трасувань ресурсів, що забезпечують їх з'єднання. (рис. 1.3.). Рис. 1.3. Структурна схема [FPGA](#) В архітектурі [FPGA](#) явно простежується велика схожість з архітектурою БМК (базових матричних кристалів). Різниця в тому, що [FPGA](#), що надходить в розпорядження споживача, має вже готові, стандартні, хоча і не запрограмовані, трасувальні ресурси, які не залежать від конкретного споживача. Отримання конкретного проєкту на базі [FPGA](#), як і на основі інших ПЛІС, реалізується впливом на програмовані з'єднання, в ході якого забезпечується замкнутий стан одних ділянок і розімкнутий - інших. Звертатися до виробника [FPGA](#) при



цьому не потрібно.

Класифікація за другою ознакою наведена нижче (рис. 1.4.) [6,30]. Рис. 1.4. Класифікація ПЛІС за рівнем інтеграції Термін

Цитирования: 0,02%

id: 6

"система на програмованому кристалі"

(SOPC - System On Programmable Chip)

Обнаружен Плагиат: 0,32% [http://ni.biz.ua/9/9\\_17/9\\_17289\\_klas...](http://ni.biz.ua/9/9_17/9_17289_klas...) + 3 ресурсів!

id: 7

відноситься до ПЛІС найбільшого рівня інтеграції, що містить мільйони еквівалентних вентилів. Такий високий рівень інтеграції досягається тільки за допомогою найсучасніших технологічних процесів (малі топологічні норми проектування, безліч шарів металізації і т. д.). На основі прогресивних технологічних процесів забезпечується одночасно високий рівень інтеграції і висока швидкодія БІС / НВІС. В результаті стає можливою інтеграція на одному кристалі цілої високопродуктивної системи.

Обнаружен Плагиат: 0,12% <https://present5.com/lekciya-12-z-na...>

id: 8

Класифікація за рівнем інтеграції (рис. 1.4.) відображає ситуацію останніх років - бурхливе зростання рівня інтеграції ПЛІС і виділення з них класу

Цитирования: 0,02%

id: 9

"системи на кристалі".

Клас систем на кристалі (SOPC) поділяється на підкласи однорідних і блокових рішень. В однорідних SOPC всі компоненти системи виконуються однаковими апаратними засобами, що забезпечує можливість їхнього програмування. Для створення систем використовуються попередньо розроблені

Цитирования: 0,02%

id: 10

"одиниці інтелектуальної власності"

(ІР-ядра), тобто готові модулі, які формують різні частини системи. Всі компоненти системи є синтезованими і можуть бути розміщені в різних зонах кристала. Розробка ІР-ядер стала важливою сферою для багатьох компаній, які пропонують широкий асортимент рішень, хоча придбання таких ядер зазвичай потребує значних витрат. Проектувальник може розміщувати необхідні йому блоки на кристалі за допомогою цих ІР-ядер. Блокові SOPC використовують апаратні ядра — спеціалізовані зони на кристалі, призначені для виконання конкретних функцій. У цих зонах створюються блоки постійної структури, розроблені за методологією ASIC, що передбачає використання макросхем типу БМК або схем із стандартними осередками. Такі ядра оптимізовані для заданих функцій і не можуть перепрограмовуватися. Спеціалізовані апаратні ядра займають меншу площу на кристалі порівняно з універсальними програмованими рішеннями, що покращує характеристики схеми, зокрема швидкодю, однак призводить до підвищення вартості ПЛІС. 1.1.2. Класифікація систем на одному кристалі Необхідно виділити різні види систем на одному кристалі. Основною їх відмінністю один від одного є технологія виробництва кристала (рис. 1.5.). Це пов'язано, з тим, що весь кристал проводиться за єдиним технологічним процесом (літографією). Крім того, системи, що містять в своєму складі осередки ПЛІС, як правило, є реконфігурованими. Всі системи на кристалі ділять на цифрові, аналогові і змішані. В даний час найбільшого поширення набули цифрові СОК. Аналогові і змішані СОК поширені менше. Це пов'язано з більш трудомістким процесом проектування і виготовлення таких мікросхем. Рис. 1.5. Загальна класифікація систем на одному кристалі Цифрові СОК в свою чергу можна розділити по апаратному базису використовуваних мікросхем. Повністю замовні мікросхеми в даній класифікації відсутні через занадто великої складності проектування таких кристалів. Для проектування такої системи були б невиправдано великі час і матеріальні витрати. Спеціалізовані мікросхеми (ASIC) найпоширеніше рішення для масового випуску продукції. Вони дозволяють випускати кристали з досягненням максимальних частотних характеристик при малій площі кристала і невеликим тепловиділенням. Спеціалізовані мікросхеми випускаються для масового ринку. При випуску обмежених партій виробів або навіть штучних примірників стає вигідніше використовувати мікросхеми, що містять в своєму складі ПЛІС. Їх можна виділити в окрему групу реконфігурованих СОК. Методології проектування спеціалізованих мікросхем і ПЛІС близькі. Виготовлення одного тестового кристала ASIC коштує сотні тисяч доларів, тоді як

**FPGA** дозволяє необмежену кількість разів перепрограмувати кристал без матеріальних витрат. Це призвело до того, що багато розробників спочатку проводять верифікацію на макетній платі з **FPGA** кристалом, а потім починають виготовлення **ASIC**. Використання ядер багаторазового використання при проектуванні систем на одному кристалі пов'язано з основною проблемою - призначення ядра для певної технології, одного виробника. Це особливо актуально при проектуванні **ASIC** і повністю замовлених мікросхем. В результаті проектувальнику доводиться вибирати з вузького кола ядер, що надаються невеликою групою фірм. Це обмеження частково пом'якшується при використанні ядер що розробляються для **FPGA**. Якщо вони мають синтезовані вихідні коди на мовах **VHDL** або **Verilog**, то їх можна використовувати в більшості систем проектування **FPGA**. Але важливо враховувати що оптимізований код під певну серію кристалів неможливо буде синтезувати для іншої серії, і потрібно буде переписувати код. З іншого боку стандартний код буде не оптимальний, якщо важливі такі параметри як швидкодія та площа, яку займає кристал. Як правило, розробникам ядер багаторазового використання доводиться заздалегідь розробляти різні версії одних і тих же ядер, але під різні платформи. Сучасні засоби синтезу розвиваються такими темпами, що незабаром таку оптимізацію повинна буде виконувати САПР автоматично.

1.1.3. Переваги та недоліки ПЛІС в якості основи реалізації систем на одному кристалі ПЛІС розглядаються в даний час як найбільш перспективна елементна база для побудови цифрової апаратури різноманітного призначення.

 **Обнаружен Плагиат: 0,2%** <https://studfile.net/preview/5203878/...> id: 11

Перспективність ПЛІС базується на ряді їх переваг, до числа яких можна віднести перераховані нижче, справедливі для ПЛІС взагалі, безвідносно до їх конкретних різновидів. Універсальність і пов'язаний з нею високий попит з боку споживачів, що забезпечує масове виробництво

даного класу мікросхем. Низька вартість, обумовлена масовим виробництвом і високим відсотком виходу придатних мікросхем при їх виробництві внаслідок досить регулярної структури.

 **Обнаружен Плагиат: 0,1%** <https://studfile.net/preview/5203878/...> id: 12

Висока швидкодія і надійність як наслідок реалізації на базі передових технологій та інтеграції складних пристроїв на одному кристалі.

Різноразоманітність конструктивного виконання, оскільки зазвичай одні й ті ж кристали поставляються в різних корпусах. Різноразоманітність у виборі напруги живлення і параметрів сигналів введення / виводу, а також режимів зниження

 **Обнаружен Плагиат: 0,22%** <https://studfile.net/preview/5203878/...> id: 13

потужності, що особливо важливо для портативної апаратури з автономним живленням. Простота модифікації проектів на будь-яких стадіях їх розробки. Наявність різноразоманітних, добре розвинених і ефективних програмних засобів автоматизованого проектування. Малий час проектування і налагодження проектів, а також виходу продукції на ринок.

Рисунок 1.6 показує різницю в часових витратах при проектуванні одного пристрою в базисі ПЛІС і базисі спеціалізованих мікросхем. Рис. 1.6. Порівняння витрат часу на проектування пристроїв в базисі **ASIC** і **FPGA** Для нових варіантів ПЛІС з динамічно реконфігурованими структурами, крім важливих з загальних позицій властивостей, слід назвати і додаткову специфічну рису: можливість побудови на їх базі нових класів апаратури - динамічно реконфігурованих пристроїв. Недоліком ПЛІС може виявитися їх енергоспоживання в додатках, де це критичний параметр. У цьому випадку застосування замовних і спеціалізованих мікросхем вигідніше. Так само в ряді випадків зберігається небезпека несанкціонованого зчитування конфігурації ПЛІС. Нове покоління **FPGA** провідних виробників (**Xilinx**, **Altera** і ін.) дозволяє реалізувати повноцінну систему на програмованому кристалі. До складу **FPGA** включаються вбудовані процесорні ядра (**Power PC** в **Virtex II Pro Xilinx** і **NIOS** в **Stratix Altera**), спеціалізовані арифметичні блоки для ЦГЗ додатків, високошвидкісні послідовні інтерфейси, блоки пам'яті різної конфігурації та призначення. Відповідно зростають і вимоги до засобів проектування таких кристалів.

1.2. Основні проблеми, що виникають при проектуванні систем на одному кристалі і можливі шляхи їх вирішення

1.2.1. Різні підходи і методології проектування окремих вузлів

Виділяються 2 типи постачальників ядер багаторазового використання: внутрішні (**internal IP providers**), що використовують власні напрацювання; зовнішні (**external IP providers**). Для зовнішніх постачальників продаж їх інтелектуальної власності у вигляді ядер - головний

напрямок бізнесу. В даний час більшість внутрішніх постачальників - це групи проєктувальників у великих компаніях, що відповідають за випуск продукції. У майбутньому вони повинні виділятися в окремі підрозділи ([IP](#) групи) для більш ефективного використання можливостей багаторазово використовуваних блоків (ядер). Існуючі засоби САПР були спочатку призначені на проєктування, виходячи з конкретної технології виготовлення, і спрямовані на один клас пристроїв ([ASIC](#), [DSP](#), мікроконтролери та ін.). При проєктуванні систем на кристалі виникає необхідність об'єднувати кошти і вести спільну розробку. Це, перш за все, необхідно для повноцінного моделювання на різних етапах проєктування не тільки окремих елементів системи, а й всієї системи в цілому. Крім того, тут же встає питання про те, що можуть використовуватися різні методології проєктування і кардинально відрізнятися маршрути проєктування таких частин. Виникає потреба узгодження різних мов (моделі на мовах високого рівня, [VHDL](#) опис, програмний код для мікроконтролера і т.д.). Необхідно мати фахівця, відповідального за координацію всього проєкту, що має дуже високу кваліфікацію - системного інтегратора. Часто виникає проблема оптимізації коду під конкретні платформи (наприклад, однієї фірми) і неможливість його інтеграції в проєкт іншої платформи (фірми). Для цього потрібна часткова або повна переробка проєкту, що можна порівняти з часом і ресурсами зі створенням його з нуля. Для просування технології проєктування на основі багаторазово використовуваних ядер в 1996 році був утворений інтернаціональний союз компаній - [VSIA](#), головною метою якого був розвиток відкритих стандартів зв'язку та тестування функціональних блоків систем. [VSIA](#) визначає, розробляє, затверджує, тестує і підтримує відкриті стандарти стосовно форматам даних, методологій тестування, інтерфейсів. Союз [VSIA](#) розробляє специфікації щодо внутрішнього проєктування блоків інтелектуальної власності, функціональної архітектури компонентів підсистеми, процесів виготовлення кристалів, алгоритмів або технологій засобів САПР. Члени [VSIA](#) представляють чотири ключові групи: компанії розробника САПР, компанії що займаються виготовленням мікросхем, незалежні розробники ядер, і об'єднують їх системні компанії. Кожен член спілки має свої власні інтереси і продукти, але вони також розуміють, що тільки за підтримки загальних стандартів і описів інтелектуальної власності вони можуть повністю використовувати можливості проєктування систем на основі ядер (рис. 1.7.). Рис. 1.7. Участь [VSIA](#) в процесі взаємодії постачальників програмних ядер і проєктувальників систем на одному кристалі [VSIA](#) був розроблений документ

Цитування: 0,02%

id: 14

«[Virtual Socket Interface](#)»

([VSI](#)), що передбачує безліч стандартів. Метою даного документа стало визначити чи рекомендувати безліч апаратних і програмних інтерфейсів, форматів і методів проєктування для створення функціональних блоків, які могли б бути ефективно і правильно інтегровані, верифіковані та протестовані спільно з іншими блоками на одному чіпі [30]. 1.2.2. Види ядер багаторазового використання На сьогоднішній день виділяють три типи багаторазово використовуваних блоків (ядер): Програмні (м'які, [soft](#)) - блоки, представлені у формі синтезованого [HDL](#) (мови опису апаратури). Вони є найбільш гнучкими. Однак недоліком таких ядер вважається непередбачуваність в термінах робочих характеристик, таких як тимчасові параметри, яку займає площу, споживана потужність. Також великий ризик, пов'язаний з правами на інтелектуальну власність, тому що для інтеграції блоку в систему потрібно [RTL](#) ([Register Transfer Level](#) - рівень реєстрових передач) код. Віртуальні (стійкі, [firm](#)) - компоненти, структурно і топологічно оптимізовані по продуктивності і займаній площі за допомогою синтезу топологічної структури та розміщення, можливо з використанням певного класу технологічних бібліотек. Рівень деталізації варіюється від розміщення областей [RTL](#) підблоків щодо прокладених трас, до параметричного синтезу для отримання повністю розміщеного списку з'єднань ([Netlist](#)). Віртуальні ядра є компромісом між програмними і апаратними. Вони більш гнучкі й зручні ніж апаратні, більш передбачувані по продуктивності і займаній площі, ніж програмні. Віртуальні компоненти включають комбінацію синтезованого [RTL](#) опису, спираються на технологічні бібліотеки, деталізують топологічну структуру і мають повний або частковий список міжз'єднань ([Netlist](#)). Коли представлено повний опис ([Netlist](#)) передбачається, що тестова логіка вже присутня, і що список тестів буде супроводжуватися з проєктом. Віртуальні ядра не включають в себе трасування. Захищеність інтелектуальної власності вище, ніж у програмних блоків, якщо відсутня [RTL](#) опис. Апаратні (жорсткі, [hard](#)) - ядра, оптимізовані по споживаній потужності, розміру або продуктивності і засновані на



конкретній технології. Зразки містять список повністю розміщених міжз'єднань з трасуванням, і оптимізовані з використанням специфічних технологічних бібліотек. Апаратні блоки зазвичай представляються у форматі [GDSII](#). Вони більш передбачувані за характеристиками, але менш гнучкі і зручні, тому що є технологічно залежними. Захищеність даних ядер найбільш висока, тому що не потрібно [RTL](#) опис. [15] 1.3. Огляд сучасних САПР ПЛІС 1.3.1. Засоби проектування систем на одному кристалі на основі багаторазово використовуваних блоків фірми [Xilinx](#) В даний час вже велика кількість готових ядер є для реалізації різних системних функцій безпосередньо в кристалах програмованої логіки фірми [Xilinx](#). Ці ядра доступні як безпосередньо від самої [Xilinx](#), так і від сторонніх фірм. Повна інформація про ядра [Xilinx](#) доступна за допомогою мережі [Internet](#) зі спеціалізованої служби [IP Center](#) за адресою [www.xilinx.com/ipcenter](http://www.xilinx.com/ipcenter). Продукти [LogiCORE](#) поширюються безпосередньо фірмою [Xilinx](#) і включають ядра системних інтерфейсів (таких як [PCI](#)), функції цифрової обробки сигналів ([DSP](#)) і велику кількість інших модулів, таких як помножувачі, суматори, блоки пам'яті та інші. Ці ядра оптимізовані для [FPGA](#) і [CPLD](#) кристалів [Xilinx](#) і забезпечують максимальну продуктивність, при цьому займаючи меншу площу. Модулі [AllianceCORE](#) продаються і підтримуються сторонніми виробниками (в основному розробниками комунікаційних пристроїв). Вони теж оптимізовані під різні пристрої [Xilinx](#). Така практика підтримується фірмою [Xilinx](#), яка бачить в цьому можливість розширення доступності високоякісних ядер за допомогою взаємодії між [Xilinx](#) і незалежних сторонніх розробників ядер. В даний час існують різні продукти [AllianceCORE](#), починаючи з процесорів і стандартних периферійних контролерів і закінчуючи функціями [ATM](#). [Xilinx](#) тісно працює зі своїми партнерами і здійснює каталогізацію продуктів для підвищення якості та доступності продаваних ядер. Будь-яке ядро має задовольняти мінімальним набором вимог до того як воно отримає мітку [AllianceCORE](#). [CORE Generator](#) - програмний засіб фірми [Xilinx](#), що надає користувачеві можливість автоматично впроваджувати ядра в свій проект. Це простий програмний засіб здатний генерувати гнучкі, високо продуктивні ядра з високим ступенем передбачуваності і дозволяє користувачам завантажувати нові ядра з [Internet](#) сайту фірми [Xilinx](#). І [Xilinx](#), і незалежні сторонні розробники [IP](#) можуть проектувати ядра для системи [CORE Generator](#), яка одночасно служить в якості служби каталогізації і є системою доставки для всіх проектувальників, які використовують [Xilinx](#). Основні особливості програми [CORE Generator](#): простий, інтуїтивно зрозумілий інтерфейс - вибір ядра, вибір його параметрів і автоматична генерація ядра; сумісний з [VHDL](#), [Verilog](#) і схемним проектуванням; оптимальне розміщення ядер; продуктивність не залежить від розмірів [FPGA](#) кристала; продуктивність залишається постійною при додаванні нових ядер; поведінкова модель [VHDL](#) і специфікація для кожного ядра; передбачуваний результат; підтримуються платформи [PC](#) і робочі станції [28]. 1.3.2. Засоби проектування систем на одному кристалі на основі ПЛІС фірми [ALTERA](#) Фірма [ALTERA](#) виробляє мікросхеми програмованої логіки, засоби проектування та програмування для них. Використовуючи НВІС і САПР фірми [ALTERA](#) можливо: реалізувати цифрову частину проекту будь-якої складності на одному кристалі; скоротити час проектування, повністю виключивши етап макетування і почавши розробку друкованої плати одночасно з розробкою НВІС; вносити зміни на будь-якій стадії проектування або в закінчений проект без повторного розведення друкованої плати; забезпечити роботу системи з тактовою частотою понад 250 МГц; прискорити і спростити процес проектування цифрового пристрою, використовуючи настроювані бібліотечні модулі типових функціональних вузлів; вводити проект в схемном вигляді або використовуючи текстовий опис, скористатися наявними можливостями інших засобів САПР; зменшити вартість розробки. Для проектування цифрових пристроїв фірма [ALTERA](#) пропонує САПР з різними функціональними можливостями, які працюють на різних платформах: [UNIX](#) та [PC](#). Повнофункціональний САПР [MAX + PLUSII BASELINE](#) пропонується безкоштовно і дозволяє: використовувати НВІС логічною ємністю до 16 тисяч еквівалентних логічних вентилів; здійснювати графічне (схемне) введення, текстове введення на мові [VHDL](#); використовувати для вхідних файлів формат [EDIF](#) і формат [.sch \(Oread\)](#); використовувати широкий набір готових функціональних пристроїв, що настроюються користувачем (множильні та ділильні пристрої, [FIFO](#) і т.д.); проводити часовий аналіз затримок сигналів і тактових частот; виконувати функціональне і тимчасове моделювання на тестах, заданих у вигляді тимчасових діаграм, що забезпечує достовірність результатів проектування; за допомогою програматора або пристроїв [BITBLASTER](#) і [BYTEBLASTER](#) програмувати НВІС, в тому числі програмувати і перепрограмувати мікросхеми прямо на платі, використовуючи технологічний роз'єм [16]. 1.3.3. Засоби проектування систем на одному кристалі фірми

[ATMEL](#) Корпорація [Atmel](#) (США) добре відома на світовому ринку електронних компонентів і є одним з визнаних світових лідерів в розробці і виробництві складних виробів сучасної мікроелектроніки. [System Designer](#) - пакет для розробки пристроїв на основі [Atmel FPSLIC](#) - призначений для створення пристроїв на одному кристалі. При цьому мається на увазі, що система на кристалі - це набір незалежних вузлів (ядер) і пам'яті, інтерфейс між якими не є визначеним і може бути обраний розробником

” Цитування: 0,01%

id: 15

"під задачу".

[FPSLIC](#) фірми [Atmel](#) представляє собою два незалежних ядра - ядро мікроконтролера [AVR](#) і ядро [FPGA](#). Гнучкість архітектури [FPSHC](#) дозволяє реалізовувати різні варіанти обміну даними між ядрами. Робоче вікно [System Designer](#) надає графічне зображення процесу розробки, представлене у вигляді етапів, із зазначенням послідовності їх виконання. Гнучка архітектура [FPSLIC](#) і широкий набір вбудованих в [System Designer](#) засобів розробки дозволяє проводити незалежне проектування програмної (ядро [AVR](#)) і апаратної (ядро [FPGA](#)) частин проекту. При цьому в процесі проектування розробник користується звичними засобами як для написання програм (C, асемблер), так і для створення мовного опису апаратури ([VHDL](#), [Verilog HDL](#)). Головною відмінною рисою [System Designer](#) є наявність в його складі засобів спільної (тобто одночасної для апаратної і програмної частин) програмної верифікації проекту. Перевага спільної верифікації проекту, тобто налагодження за допомогою моделювання програм, в порівнянні з налагодженням проекту безпосередньо в реальній розробці, полягає в набагато більш ранньому виявленні помилок як в програмній, так і в апаратній частинах проекту. Це дозволяє вносити зміни в проект на більш ранній стадії, і таким чином уникати багаторазового повторення процедури синтезу, розміщення і розводки, які, як правило, є найбільш тривалими з усього процесу проектування. Таким чином, спільна верифікація дозволяє істотно скоротити час розробки. Концепцію [FPSLIC](#) фірми [Atmel](#) можна розглядати як перший реальний крок до злиття двох шляхів розвитку складних універсальних мікросхем (мікропроцесорів і програмованої логіки) в єдине ціле. [FPSLIC](#) є аббревіатурою [Field Programmable System Level Integration Circuits](#), що приблизно можна перекласти як Програмовані Користувачем Мікросхеми Системного Рівня Інтеграції. Мікросхеми [FPSLIC](#) спроектовані як сімейство стандартних кристалів, яке буде розвиватися з плином часу. Першими з'являться мікросхеми, що базуються на ядрі [AVR RISC](#) мікроконтролерів [Atmel](#). Перш за все, [Atmel](#) помістила на кристал модуль стандартного [FPGA](#) сімейства AT40K, забезпечивши розробнику програмовану апаратну платформу для реалізації його власних макросів і проектів безпосередньо в [FPSLIC](#). Поруч з блоком [FPGA](#) знаходиться [AVR](#)-мікроконтролер. Тут вперше стандартне ядро [AVR](#) виконує команди з [SRAM](#), що значно підвищує швидкість його роботи. Для забезпечення додаткової ефективності при виконанні [DSP](#)-додатків до ядра [AVR](#) доданий блок апаратного множення. Поруч з мікро контролером розміщений набір фіксованих периферійних вузлів: два [UART](#), три таймера-лічильника (два 8-розрядних і один 16-розрядний) і два порти введення / виводу. Доданий апаратний інтерфейс 12C, дозволяє [AVR](#) обмінюватися даними з зовнішньою конфігураційною [EEPROM](#), використовуваною для програмування [FPSLIC](#). Блок фіксованої логіки, розміщений між [AVR](#) і [FPGA](#), дозволяє використовувати масив [FPGA](#) для реалізації додаткових програмованих периферійних вузлів в реальному проекті, причому ці нові периферійні пристрої будуть доступні в загальному адресному просторі пам'яті [AVR](#). Архітектура загального масиву статичної пам'яті [SRAM](#) всередині кристалу [FPSLIC](#) була реалізована таким чином, щоб забезпечити розробнику максимальну гнучкість у розподілі адресного простору. Об'єм пам'яті програм становить 10Kx16, пам'яті даних - 4Kx8. Крім цих фіксованих масивів пам'яті, на кристалі є додатковий блок пам'яті 6Kx16, який може використовуватися або як додаткова пам'ять програм, або як додаткова пам'ять даних, в залежності від призначення програми та бажання розробника. Якщо, наприклад, додаток пишеться на мові C, то потрібна додаткова пам'ять програм; якщо реалізується будь-якої алгоритм [DSP](#), то зазвичай потрібна додаткова пам'ять даних. Конфігурація розподілу пам'яті зберігається в спеціальному регістрі конфігурації системи, який знаходиться в контролері пам'яті [FPSLIC](#). Регістр конфігурації завантажується при включенні харчування разом з [SRAM](#) програм, [SRAM](#) даних і [FPGA](#) з зовнішньої конфігураційної [EEPROM](#). Ще однією особливістю архітектури є те, що [AVR](#) може безпосередньо реконфігурувати [FPGA](#) на кристалі. Ця обставина є дуже важливою з огляду на вимоги постійного розвитку портативних реконфігурованих систем. Виділимо три головні переваги [FPSLIC](#), отримані в результаті

розміщення ядра [AVR](#), блоку [FPGA](#) і масиву статичної пам'яті на одному кристалі. Перше - підвищений ступінь інтеграції, новий якісний рівень кінцевого виробу. Енергоспоживання, в порівнянні з варіантом системи, реалізованим на дискретних компонентах, знизилося вдвічі. Низький рівень споживання енергії [FPSLIC](#) передбачає його використання в різноманітній портативній апаратурі. Друге - продуктивність. Ядро [AVR](#), реалізоване в складі [FPSLIC](#) по технологічному процесу [SRAM / ASIC](#), здатне працювати з продуктивністю понад 30 [MIPS](#). Третє - значне скорочення часу виходу на ринок нових проєктів і розробок. Комбінуючи апаратний ([FPGA](#)) і програмний ([AVR](#)) підходи до створення складних універсальних мікросхем в межах площі одного кремнієвого кристала, [Atmel](#) розробила також набір інструментальних засобів підтримки розробок, які дозволяють розробнику проєктувати і верифікувати як апаратну, так і програмну частини проєкту одночасно. Об'єднання стандартних вузлів [AVR](#) і [FPGA](#) на одному кристалі дозволяє безпосередньо використовувати в [FPSLIC](#) коди проєктів, які вже були реалізовані на базі [AVR](#) або [FPGA](#). Перше сімейство [FPSLIC](#), що містить [AVR](#) і [FPGA](#), має позначення AT94Kxx. У ньому анонсовані три мікросхеми з ємністю [FPGA](#) 40000, 20000 і 10000 еквівалентних вентилів - AT94K40, AT94K20 і AT94K10, відповідно. Розглянемо реалізовану архітектуру [FPSLIC](#). AT94K - це сучасний виріб мікроелектроніки, що представляє в своїй основі стандартну мікросхему [FPGA](#) серії AT40K з інтегрованим на кристал блоком статичної пам'яті [SRAM](#) і апаратним ядром [AVR](#)-мікроконтролера. Архітектура [FPGA](#) збережена тут такий же, як і у мікросхем версії [AL](#) сімейства AT40K. Масив [FPGA](#) виконаний за технологічними нормами 0,35 мкм. Він має поліпшену структуру міжз'єднань, додаткові регістри в елементах введення / виведення, доступ до тактування безпосередньо з самої [FPGA](#) і розподілену швидкісну двопортову оперативну пам'ять [FreeRAMIM](#). Логічний елемент являє собою восьмикутний осередок з 8 зв'язками з 8 сусідніми осередками. Масив [FPGA](#) має 8 глобальних ліній тактування. Шість з них є зовнішніми, а дві - внутрішніми, причому їх джерелом є ядро [AVR](#). Одна з внутрішніх ліній тактування з'єднана з системної тактової шиною мікроконтролера, а інша може бути програмним чином підключена до одного з восьми різних джерел тактового сигналу, які генеруються всередині [AVR](#) - виходи таймерів / лічильників, сторожового таймера і так далі. Весь масив [FPGA](#) в [FPSLIC](#) динамічно реконфігуруємо, це має хороші показники статичного і динамічного енергоспоживання, що робить його придатним для реалізації мобільних додатків. [AVR RISC](#)-мікроконтролер має в системі команд близько 120 інструкцій. Доданий блок апаратного множення для підтримки проєктів і підсистем, заснованих на [DSP](#). Очікувана продуктивність ядра [AVR](#) - більше 30 [MIPS](#) при тактовій частоті 40 МГц, що досягається за рахунок вибірки команд з швидкої пам'яті [SRAM](#). Архітектура [AVR](#) оптимізована для розробки додатків на мові C, є регістровий файл на 32 регістра. Мікроконтролер також підтримує кілька режимів зниженого енергоспоживання. Для периферійних пристроїв, специфікованих користувачем в масиві [FPGA](#), реалізована додаткова підтримка. Мікроконтролер має можливість записувати інформацію безпосередньо в [FPGA](#) через механізм введення / виведення зі спеціалізованим адресним простором. Шина даних [AVR](#) також безпосередньо з'єднана з масивом програмованої логіки, так що є можливість отримувати доступ до адресованого периферійного пристрою в [FPGA](#). У мікроконтролері апаратно підтримується 16 додаткових векторів переривань від [FPGA](#). Всі вони розподілені в загальному просторі векторів переривань [AVR](#) з різними рівнями пріоритету. Є також чотири окремих зовнішніх переривання в [AVR](#). Мікросхеми сімейства AT94K допускають використання декількох типів джерел зовнішньої опорної частоти. Напруга живлення становить 3,3 В. Мікросхеми [FPSLIC](#) виготовляються по 0,35-мкм технології. Нові мікросхеми повністю сумісні по розташуванню і призначенню зовнішніх висновків з мікросхемами [FPGA](#) сімейств [Atmel](#) AT40K, [Xilinx](#) 4000, 5200 і [SPARTAN](#) і є також [PCI](#)-сумісними. Тому всі засоби підтримки розробок, вже розроблені і наявні для [FPGA](#), можуть бути використані і для роботи з мікросхемами [FPSLIC](#). Це, наприклад, комплект макетних плат [ATDH40M](#) і новий, недавно випущений набір розробника [ATSTK40](#). Що ж стосується аналогових або аналого-цифрових компонентів, то поки тут є певні технологічні проблеми, так як це пов'язано з впровадженням [BiCMOS](#)-технології. Паралельно з цим [Atmel](#) буде приділяти велику увагу створенню і розвитку бібліотеки стандартних ядер багаторазового використання у вигляді готових макросів, які потім можуть розміщуватися на кристалі [FPSLIC](#) [9].

### 1.3.4. Програмні засоби фірми [Mentor Graphics](#)

Компанія [Mentor Graphics](#) однією з перших серйозно зайнялася розробкою систем проєктування для [FPGA](#), СОК на їх основі. В даний час це один із стратегічних напрямків її розвитку. Відмітна особливість продуктів [Mentor Graphics](#) - незалежність від конкретного виробника, що дозволяє проєктувати системи, засновані на комбінації [FPGA](#) різних



виробників, оптимально використовуючи переваги кожного з них. У разі необхідності переходу на іншу технологію або інший тип кристалів систему досить легко перепроектувати. Все це неможливо при використанні засобів проектування, пропонованих виробниками [FPGA](#), які працюють тільки зі своїми кристалами. Проектування починається з розробки специфікації архітектури системи на мовах високого рівня C, C++, [System C](#), [System Verilog](#), [MATLAB](#) і верифікації отриманої моделі за допомогою програми [ModelSim](#). На цьому етапі приймається рішення про апаратний або програмно-апаратний спосіб реалізації. У разі вибору апаратного способу реалізації можна безпосередньо переходити до створення [RTL](#) описів ([VHDL](#) або [Verilog](#)) проєкту. Рис. 1.8. Маршрут проектування [FPGA](#) компанії [Mentor Graphics](#). Це можна робити вручну або скористатися програмою поведінкового синтезу [Precision 3 Synthesis](#), яка орієнтована в основному на синтез систем цифрової обробки сигналів. Вона дозволяє, маючи опис алгоритмів на C або C++, автоматично отримати код, який синтезується [VHDL](#) або [Verilog](#). Слід зазначити, що отриманий на виході [RTL](#) код може бути використаний як для синтезу [FPGA](#), так і для синтезу [ASIC](#). Але слід враховувати, що отриманий код є, як правило, надмірним і не оптимальним за основними параметрами (швидкодії, займаної площі кристала). Якщо прийнято рішення про програмно-апаратний спосіб реалізації та вибрано процесорне ядро, процес проектування поділяється на розробку вбудованої програми і створення апаратної частини проєкту. Апаратна частина за допомогою пакета [Platform Express](#) верифікується на рівні шинних інтерфейсів. Програмна частина налагоджується в комплексному інтегрованому середовищі, що складається з системи програмування [code lab](#), відладчика [XRAY](#) і операційних систем реального часу [VRTX](#) і [Nucleus](#). Процес комплексної верифікації системи можна істотно прискорити за допомогою пакета програмно-апаратної верифікації (або віртуального прототипу) [Seamless CVE](#). Ядро [Seamless CVE](#) управляє взаємодією інтерпретатора системи команд вбудованого процесора і [RTL](#) моделювання апаратної частини, забезпечуючи спільне моделювання та налагодження всієї програмно-апаратної системи на віртуальному прототипі, не вдаючись до створення макета. На ранніх стадіях розробки можуть бути виявлені і усунені помилки взаємодії розроблюваного програмного забезпечення та апаратури, виправлення яких на подальших етапах потребує великих витрат, а іноді просто неможливо без повного повторного проектування. Ядро маршруту проектування - пакет [FPGA Advantage](#) - включає три основних модуля: [HDL Designer](#), [ModelSim](#) і [Precision RTL / Physical Synthesis](#), які при необхідності можуть використовуватися автономно або в інших маршрутах проектування. Цьому сприяє те, що основним форматом передачі даних між етапами на цій стадії проектування є [RTL](#) опису на [VHDL](#), [Verilog](#) або їх комбінації [17].

### 1.3.5. Маршрут проектування цифрових пристроїв в базисі ПЛІС

До кінця 1990-х років проектування схем в основному виконувалося за допомогою графічних редакторів і бібліотек стандартних логічних елементів, що включали базові логічні примітиви (логічні елементи, прості комбінаційні і послідовні вузли, аналоги стандартних ІС низької та середньої інтеграції). Сьогодні ж у проектуванні переважає використання мов опису апаратури для реалізації алгоритмів на ПЛІС. У сучасних системах автоматизованого проектування (САПР) підтримуються як стандартизовані мови опису апаратури (такі як [VHDL](#) і [Verilog HDL](#)), так і фірмові мови, розроблені виробниками ПЛІС для врахування архітектурних особливостей своїх продуктів. Багато великих компаній-виробників САПР для інтегральних схем активно розробляють програмне забезпечення, сумісне з ПЛІС різних виробників. Це дозволяє створювати алгоритми, які можуть бути реалізовані на ПЛІС не тільки різних сімейств, а й різних брендів, що спрощує перенесення алгоритмів і прискорює процес розробки. Прикладами таких САПР є [FPGA Express](#) від [Synopsys](#), [OrCAD Express](#) від [OrCAD](#), а також продукти компаній [VeryBest](#), [Aldec](#), [Cadence Design Systems](#) та інших. Класичний процес проектування цифрових пристроїв на основі ПЛІС із використанням сучасних САПР можна представити у вигляді схеми, показаної нижче (рис. 1.9.). З представленого тут маршруту видно, що класичний підхід до проектування цифрових пристроїв в базисі ПЛІС істотно розширився, до нього додалися нові можливості (на малюнку виділено). При цьому в сучасному підході велика увага приділяється проблемі верифікації розробки, де головним інструментом служить моделювання. Рис. 1.9. Сучасний маршрут проектування цифрових пристроїв

Основною проблемою в моделюванні раніше була обмежена обчислювальна потужність комп'ютерів, на яких встановлювалися системи моделювання. Продуктивність сучасних комп'ютерів (навіть персональних) настільки зросла, що стало можливо моделювання складних систем (що містять велику кількість елементів) за прийнятні часові терміни, при цьому не тільки збільшена адекватність результатів моделювання, а й швидкість і простота отримання результатів. Так само треба

відзначити, що після кожного етапу моделювання можливе повернення на будь-який з попередніх рівнів. Таким чином, досягається мета ітераційного процесу розробки. У сучасних САПР ПЛІС отримання опису пристрою на вентиляльному рівні по його опису на рівні реєстрових передач, а так же отримання конфігурації за описом пристрою на вентиляльному рівні є процесом, що виконується практично в автоматичному режимі (з мінімальним втручанням користувача до якого вдаються тільки в разі крайньої необхідності). Рис. 1.10. Різні маршрути проєктування систем на кристалі в базисі ПЛІС Як зазначалося раніше, використання ядер багаторазового використання при проєктуванні пристроїв призводить до відмови від традиційного маршруту проєктування і заміни деяких його частин на абсолютно нові. Так, якщо при традиційному проєктуванні, кожен окремий блок системи розробляється і окремо тестується, то в новому маршруті ці етапи відсутні (рис. 1.10.). Необхідно враховувати, що якщо крім стандартної настройки використовується зміна вихідних кодів ядра, то в такому випадку етап верифікації окремих блоків необхідний. 1.4. Висновки до розділу Досліджено стан мікроелектронної промисловості в області створення систем на одному кристалі. Виділена класифікація цифрових інтегральних мікросхем і систем на одному кристалі на їх основі. Досліджено стан в області сучасних наскрізних САПР використовуваних при проєктуванні цифрових систем на одному кристалі. В ході проведених досліджень були виявлені сімейства ПЛІС, що володіють найбільш розвиненими функціональними структурами, перспективні для застосування в нових пристроях. РОЗДІЛ 2.

**РОЗРОБКА МЕТОДИКИ ПРОЄКТУВАННЯ СКЛАДНИХ ЯДЕР В БАЗИСІ ПЛІС 2.1. Методика проєктування складного ядра** Дамо характеристику складним ядрам як складним технічним системам (СТС). Цілеспрямованість. Складне ядро як СТС створюється для досягнення певної мети. Так як воно функціонує в складі і включає її в надсистему, то метою можна вважати усунення протиріч, що виникають в цій надсистемі. Якщо в якості СТС розглядається складне ядро, то надсистемою є САПР, для якої воно створюється. Протиріччя може полягати, наприклад, у невідповідності трудомісткості або термінів проєктування вимогам, що пред'являються. Цілісність і роздільність. Складне ядро являє собою цілісне утворення, що складається з пов'язаних між собою елементів. Цілеспрямованість складного ядра обумовлює його розуміння як єдиного цілого, що функціонує в конкретних умовах надсистеми і складається з взаємодіючих в інтересах досягнення мети частин, різноякісних, але сумісних. Сукупність стійких зв'язків між елементами системи називається її структурою. Часто структуру розуміють більш широко, включаючи в це поняття і сукупність елементів системи, тобто її склад. Структуру системи зручно описувати графом, вершини якого відповідають її елементам, а ребра (в орієнтованому графі - дуги) - зв'язкам між ними. Рис. 2.1. Приклад ієрархічної структури складного ядра Ієрархічність. Складне ядро може бути представлено не тільки як елемент надсистеми, що знаходиться на більш високому рівні ієрархії, а й як сукупність елементів, які є підсистемами і належать більш низького рівня ієрархії. Підсистеми також можуть бути розділені на частини (декомпозиція). Продовжуючи декомпозицію до рівня елементів, подальше членування яких недоцільно, отримуємо багаторівневу ієрархічну структуру (рис.2.1.). Ієрархічна структура полегшує завдання забезпечення сумісності елементів ядра за рахунок її декомпозиції на завдання сумісності підсистем і сумісності елементів кожної підсистеми. Багатоаспектність. Складне ядро характеризується різними групами властивостей (аспектами), які необхідно враховувати при його проєктуванні й описі. Аспекти описів тісно пов'язані між собою, однак при вирішенні конкретної проєктної задачі на перший план зазвичай виступає один з них. Опис складного ядра, виконаний в будь-якому аспекті, називається її поданням. Основними уявленнями системи є: функціональне, морфологічне та процесне. На базі цих уявлень можуть бути побудовані більш складні, що використовують і конкретизують, поняття декількох основних уявлень. Кожна постанова характеризується своєю структурою. Більш того, під структурою складної технічної системи зазвичай розуміється структура одного з її уявлень. Це призводить до полідекомповованості – неоднозначності декомпозиції на підсистеми і ускладнює створення складного ядра. Зокрема, складне ядро можна декомпонувати на забезпечення (технічне, програмне та ін.) і підсистеми, які проєктують, на обслуговуючі. Розвиток. Складне ядро розвивається, тобто змінює свої функції, структуру, внутрішні процеси протягом усього життєвого циклу - проєктування, виготовлення, експлуатації, ліквідації. На кожному етапі життєвого циклу ядро має відмінні один від одного описи по кожному аспекту. Причинами змін в ядрі є зміни його зовнішнього середовища, в першу чергу надсистеми і продукції, що випускаються елементною базою, а також обмежені терміни і



вартість проектування, що не дозволяє відразу реалізувати в ядрі необхідні функції і зумовлює введення його в дію чергами. Перераховані властивості дозволяють сформулювати основне протиріччя процесу проектування, що виникає при описі складного ядра, - між необхідністю отримання цілісного опису, що відображає цілісність системи, і її складністю, що полягає в ієрархічності, натхненні, розвитку. Для вирішення цієї суперечності необхідно забезпечити комплексний характер створюваних описів, що враховує складність в вищевказаному сенсі. Лише ґрунтуючись на цілісному описі системи, можна з упевненістю її проектувати. Проектування складного ядра багаторазового використання, аналогічно проектуванню складних обчислювальних пристроїв. При цьому визначають основні блоки. Такими блоками можуть бути чисто логічні схеми, схеми пам'яті, схеми, виконують певні арифметичні операції, схеми з повторюваною структурою та ін. Кожен блок повинен бути спроектований з огляду на його основні властивості. Це призводить до необхідності використання при проектуванні різних підходів. У наступних розділах будуть розглянуті особливості проектування конкретних видів блоків проектного ядра. Як правило, будь-яка розробка ведеться на основі раніше спроектованих блоків (не обов'язково ядер). У такому випадку при складанні загальної структури розробки ядра необхідно враховувати можливість їх використання. При цьому дані використовуваних блоків повинні бути модернізовані до рівня ядер. Це треба враховувати, оскільки в ряді випадків більш доцільно вести проектування заново, не задіявши старі блоки (наприклад, якщо ми бажаємо отримати більш широкий діапазон застосування ядра). При складанні узагальненої ієрархічної структури ядра у вигляді дерева все листя буде відповідати певному типу (рис. 2.1.). Тип буде вказувати на характер блоку з урахуванням подальшої апаратної реалізації. Рисунок 3.1 показує приклад такого дерева ієрархічної структури. Блоки А, В, С, [D](#) - блоки різної природи (характеру). Наприклад, А - блок є операцією множення, В - блок - осередки пам'яті, С - блок - операція піднесення до ступеня, [D](#) - блок виконує логічні операції. Всі позначення на такій схемі повинні бути прокоментовані в проектній документації. Буквені позначення можуть бути введені довільно, проте можливо використовувати й інші способи позначення відповідних блоків. Щоб виділити блоки ядра, що розробляється, необхідно мати його програмну модель на системному рівні. У такій моделі ми можемо простежити основні складові частини, але для цього модель повинна бути максимально наближена до апаратної реалізації. У програмній моделі бажано використовувати обмежену розрядність для всіх даних. Виконання арифметичних виразів, бажано будувати з огляду на можливі апаратні обмеження. Наприклад, операцію множення за можливості замінювати операцією додавання. Необхідно використовувати кроки виконання операцій, аналогічні виконуваним в апаратурі, враховувати можливі затримки у виконанні і загальну протяжність обчислень. Модель проектного пристрою може бути написана або на мовах програмування (С, С++, [Pascal](#) і т.д.) або мовами опису апаратури ([VHDL](#), [Verilog](#)). Останнім часом все більшої популярності набирають змішані мови ([SystemC](#), [HandleC](#) і ін.) вироблені на базі мови С і наближені до апаратних засобів. Такий підхід дозволяє більш чітко відслідковувати особливості поведінки моделі і змінювати структуру проекту на самих ранніх стадіях. Дуже часто, особливо при проектуванні пристроїв цифрової обробки сигналів, проектоване ядро має повторювані блоки, що відрізняються незначно. Якщо участь даних блоків в обчислювальному процесі розділена в часі, то має сенс використовувати один загальний настроюваний блок. Це скоротить апаратні витрати в кристалі, що в свою чергу може привести до зменшення енергоспоживання. Такий же підхід має сенс використовувати, якщо необхідно виконувати складні обчислення. Наприклад, всі множення можна виконувати не на окремих помножувачах, а на їх сильно обмеженій підмножині. Особливо це актуально, якщо використовуються вбудовані блоки множення сучасних кристалів ПЛІС, наприклад [Virtex II](#) фірми [Xilinx](#). Аналогічно використовуються великі блоки пам'яті. Але такі питання необхідно враховувати на самих ранніх стадіях проектування ядра, оскільки вони, як правило, призводять до значних модернізацій в моделі. Якщо блок ядра можна параметризувати, то бажано залишати таку можливість доступною. У програмній моделі такі блоки можна використовувати як функції, що перетворюють дані в залежності від вхідних параметрів. В апаратній моделі на мові [VHDL](#) для цього існують спеціальні конструкції. Налаштовувана константа [generic](#) кодує певну властивість об'єкта проекту. Вона використовується, наприклад, для завдання розрядності ліній зв'язку, кодування структури модельованого пристрою. При вставці компонента через інтерфейс зв'язування налагоджувальних констант, налаштування об'єкта верхнього рівня передається об'єктам нижнього рівня. Якщо необхідно неодноразово повторити один або кілька паралельних

операторів, то використовують оператор [generate](#). Дані оператори враховуються автоматизованими системами проектування, як на стадії моделювання, так і на стадії синтезу. При проектуванні ядер часто виникає ситуація, коли частина блоків не може бути параметризована автоматично. Наприклад, таблиця косинусів-синусів для різних варіантів розмірності перетворення Фур'є може бути швидко сформована програмно, але важко піддається параметризації в ПЛІС. Тому при проектуванні таких ядер бажано використовувати готові, заздалегідь сформовані набори таблиць для всього діапазону можливих значень розмірності задачі, які прораховані програмно і підставляються в модель на стадії впровадження ядра в проєкт. Якщо це неможливо, тоді можна використовувати спеціальні методи інтерполяції, але в такому випадку необхідно точно розрахувати похибку одержуваного результату. Методику проектування складного ядра в загальному випадку можна представити у вигляді наступних кроків: Складання узагальненої структури ядра. Виділення основних блоків, а також виділення серед них апаратно залежних і апаратно незалежних. Визначення способу реалізації кожного блоку. Проектування кожного блоку з урахуванням апаратного базису і правил проектування. 2.2. Маршрут проектування складного ядра Візьмемо за основу загальний маршрут проектування обчислювальних пристроїв (рис. 1.10.). Як і в загальному маршруті будуть виділятися дві найважливіші стадії проектування: розробка функціональної моделі ядра на системному рівні і розробка моделі ядра на рівні регістрових передач. Решта стадій проектування можуть бути автоматизовані. Необхідно враховувати, що для повноцінного проектування необхідно проходити повний маршрут, включаючи стадії макетної верифікації. Це дозволить з упевненістю гарантувати належне виконання функцій майбутнього ядра. Перетворимо даний маршрут з урахуванням основних особливостей проектування ядер, викладених в попередньому розділі. Необхідно додати стадію розробки повної документації, а так само використовувати для перевірки проєкту макетування, що відображено на схемі (рис. 2.2.). В даному маршруті необхідно деталізувати два етапи: розробку функціональної моделі пристрою на системному рівні і розробку моделі пристрою на рівні регістрових передач. Першим і найбільш важливим кроком проектування складних ядер можна назвати розбиття його на окремі функціональні блоки. Дані блоки повинні бути пов'язані між собою строго визначеними інтерфейсами. Виділення залежить від складності ядра, їх ролі, а так само від планованого діапазону застосування. Необхідно враховувати апаратний базис, для якого проєктується ядро. Якщо ядро планується використовувати для різного апаратного базису, то необхідно заздалегідь планувати можливі варіанти реалізації. Рис. 2.2. Маршрут проектування ядер У загальному випадку при проектуванні ядер використовуваних для цифрової обробки сигналів в базисі новітніх ПЛІС можна використовувати наступну структуру дерева вибору апаратних реалізацій окремих блоків (рис. 2.3.). Так, якщо нам необхідно реалізувати в ядрі операцію множення, ми можемо вибрати реалізацію на процесорі, на вбудованих в кристал апаратних помножувачах, на стандартній логіці, або залишити можливість (надати відповідні інтерфейси) виконати операцію на зовнішньому пристрої. Аналогічно обираються реалізації схеми з пам'яттю і проста логіка. При створенні ядра можна передбачити різні комбінації реалізації окремих блоків. Це залежить, перш за все, від області застосування ядра, від обраних в якості основи кристалів і їх апаратних можливостей. Проектувальнику, котрий використовує дане ядро, необхідно дати можливість вибору реалізації окремих блоків, так як йому доводиться ділити весь кристал на різні ядра, і він працює в обмеженому просторі апаратних можливостей кристала. Дана схема (рис. 2.3.) не покриває всіх можливих варіантів реалізації блоків, які можуть виникнути під час проектування ядра. Вона є лише прикладом. Відповідно завдання проектувальника перед початком процесу проектування скласти карту можливих способів реалізації на підставі апаратних можливостей застосовуваних кристалів. З огляду на побудовану карту можна виділяти функціональні блоки розроблюваного складного ядра. Таким чином, маршрут проектування складних ядер повинен враховувати можливість розбиття проєктованого ядра на складові блоки, їх розробку, верифікацію і подальше сполучення, як на рівні функціональних моделей, так і на рівні регістрових передач (рис. 2.4.). Рис. 2.3. Дерево вибору апаратних реалізацій блоків ядра Якщо розробка ведеться на основі вже готових, розроблених раніше, блоках (наприклад, більш дрібних ядер), то маршрут проектування може бути аналогічний загальному маршруту проектування з використанням ядер. Зміни стосуватимуться в першу чергу розробки більш повної документації, розробки тестів і макетування різного роду пристроїв, що використовують дане ядро. Рис. 2.4. Модифікований маршрут проектування ядер 2.3. Загальні правила проектування У ряді

робіт [3, 13, 14, 26] були запропоновані правила проєктування, яких бажано дотримуватися при проєктуванні різних класів обчислювальних пристроїв. З огляду на можливості сучасних систем САПР, і в тому числі можливості систем синтезу в даній роботі були систематизовані правила, які стосуються проєктування ядер багаторазового використання. Виділяються правила загального характеру і правила, пов'язані із застосуванням мови VHDL. Основна увага приділяється правилам, які необхідно застосовувати при проєктуванні складних ядер. Спочатку розглянемо види документації, що використовується на першій стадії проєктування відповідно до описаного маршруту.

1. Специфікація. Перед початком безпосереднього HDL кодування, треба перевірити існуючі ядра і написати специфікацію. Це призведе до наступних переваг: чисте визначення того, що ядро має робити і які стандарти використовувати; визначення профілів розробників при формуванні групи людей, зайнятої в проєктуванні. По суті ядро багаторазового використання - це чорний ящик, тому специфікація повинна концентруватися тільки на інтерфейсі даного чорного ящика. Будь-який бажаний використовувати дане ядро повинен прочитати цю специфікацію, тоді як для зміни ядра необхідно прочитати також і проєктні інструкції. Для специфікації бажано використовувати шаблон специфікації.

2. Проєктна документація. Якщо над ядром працює група розробників, проєктна документація повинна бути написана до початку безпосереднього програмування, це умова того, що блоки що розробляються будуть здатні працювати спільно без додаткових витрат на їх інтеграцію. Проєктна документація важлива тому, що: досягається розуміння того, як повинні працювати і взаємодіяти один з одним внутрішні блоки ядра; з'являється можливість роботи групи людей над різними частинами ядра; можливість подальшої розробки та вдосконалення іншими; спрощується верифікація та усунення несправностей.

3. Структура директорій. Для спрощення інтеграції різних ядер в систему на одному кристалі необхідно використовувати задану, суворо обумовлену структуру директорій, але вона обов'язково має бути присутня в проєктній документації та документації користувача даного ядра. Якщо ядро вимагає додаткових директорій необхідно додавати їх, враховуючи дані угоди в типовій структурі. Наприклад, дуже часто вихідні тексти для утиліт і програмних тестів вимагає створення декількох піддиректорій. Далі слід привести ряд правил, які використовуються на стадії розробки моделі ядра або окремих блоків на системному рівні.

1. Рекомендується писати наочні коментарі. Необхідно дотримуватися тенденції писати коментарі до кожного нового призначення або блоку.

2. Якщо ядро складне і складається з декількох ієрархічних підмодулів, то рекомендується створювати модуль верхнього рівня тільки для зв'язки більш низьких, і не використовувати в ньому логіку.

3. Бажано зберігати одне і те ж ім'я сигналу в різних ієрархіях. Далі наведемо ряд правил, які використовуються на стадії розробки моделі ядра на рівні регістрових передач.

1. Не рекомендується змішувати логіку, що працює по верхньому рівню сигналу і нижньому рівню сигналу. Бажано дотримуватися тільки логіки одного верхнього рівня.

2. Скидання (Reset). Рекомендується створювати асинхронне скидання, що працює по верхньому рівню сигналу для всіх тригерів. Таке скидання повинно бути синхронізоване з тактовим генератором на верхньому рівні СОК і не повинно вимагати мультиплексування на кожному вході тригера. Рекомендується, що б під час скидання всі двонаправлені порти були б в змозі входу.

3. Синхросигнали. Сигнали, які перетинають різні області тактування повинні бути дискретизовані до і після перетину таких областей (необхідний дворазовий вибіркового контроль) для запобігання нестабільних станів. Рекомендується не використовувати стробований синхросигнал, якщо немає чіткого розуміння правильного шляху стробування синхросигналу і наслідків для подальшого тестування і верифікації. Рекомендується не використовувати синхросигнал або сигнал скидання як дані або як сигнал запуску, чи не використовувати дані як синхросигнал або сигнал скидання. Бажано використовувати мінімальну кількість областей синхросигналів в одному ядрі.

4. Шини. Строго рекомендується порівнювати шини однакової довжини. Рекомендується починати індексувати шину з 0. Якщо це правило порушується, потрібно вказувати спосіб індексування у всіх випадках використання шин. Рекомендується використовувати перетворення MSB в LSB (старший значущий розряд в молодший значущий розряд). При цьому біт 0 повинен бути молодшим значущим (LSB), Рекомендується намагатися проєктувати з мінімальною кількістю з'єднань в інтерфейсі ядра, чи не робити шини ширше, ніж це необхідно. Якщо можливо, робити шину даних звуженою, а натомість збільшувати шину адреса.

5. Тристабільні схеми. Рекомендується зазвичай уникати використання внутрішніх тристабільних сигналів. Однак вони рекомендуються для внутрішнього моніторингу.

6. Пам'ять. Рекомендується використовувати синхронні одно і

двох портів групі блоку пам'яті. 7. Кодування для синтезу. Рекомендується використовувати синхронний метод проектування. Це дозволяє уникнути проблем з синтезом, тимчасовою верифікацією і при моделюванні. Строго рекомендується не використовувати елементів затримки. Це призводить до проблем синтезу і проблем тимчасової верифікації. 8. Рекомендується всі зовнішні входи / виходи ядра буферизувати. Це запобігає появі довгих тимчасових шляхів і дозволяє дотримуватися тимчасових обмежень. Це також дозволяє легше проводити верифікацію всієї системи на одному кристалі. Однак в деяких випадках не можна використовувати буфери на виходах як, наприклад, в разі сполучення з шиною [PCI](#). Рекомендується також синхронізувати внутрішні інтерфейси ядра. Рекомендується уникати використання регістрів-клямок ([latches](#)). Бажано уникати використання тригерів з негативним перепадом синхроімпульса. 9. Порти введення / виводу. Рекомендується називати порти ядра спираючись на угоди показані нижче (Таблиця 2.1) або аналогічні з обов'язковим занесенням в проектну документацію. Це спрощує процес інтеграції СОК. Таблиця 2.1. Порти введення / виводу ядра

Порт	Опис
*_i	Вхідний порт ядра
*_o	Вихідний порт ядра
*_io	Двухсторонній порт ядра
*_clk_i	Порт вхідного синхросигналу ядра
*_clk_o	Порт вихідного синхросигналу ядра
*_rst_i	Вхідний порт сигналу скидання
*_rst_o	Вихідний порт сигналу скидання

Не варто використовувати інші аббревіатури крім \*\_clk\_\* і \*\_rst\_\* для позначення синхросигналу і сигналу скидання. Наприклад, не треба використовувати \*reset\* або \*clock\*. У такому випадку можуть виникнути проблеми при синтезі ядра. Рекомендується використовувати \*n для позначення сигналів працюють за низьким рівнем, і не використовувати для цього інше позначення, типу \*\_\_. Використання \*\_ для вказівки того, що сигнал працює по нижньому рівню можливо в мові [Verilog](#), але не в [VHDL](#). 2.4. Правила створення ядер, використовуючи мови опису апаратури [VHDL](#)

Мова опису апаратури [VHDL](#), як і будь-який інший мову має ряд особливостей. В першу чергу це відноситься до інтерпретації цієї мови на стадії синтезу. Тому необхідно дотримуватися ряду правил і рекомендацій, що забезпечують правильне функціонування розроблюваного ядра, а так само важливим є

Цитування: 0,01%

id: 16

«читаність»

коду, що важливо при модернізації ядра. Наведемо далі ряд правил, яких бажано дотримуватися для отримання універсального легко інтегрованого в проект замовника ядра. Ці правила були систематизовані щодо проектування складних ядер. Дані правила не є обов'язковими для застосування, але їх порушення може ускладнити в багатьох випадках інтеграцію в кінцевому проекті. Можуть змінитися характеристики в реалізації. Крім того, правила враховують можливість модернізації ядер. В даних правилах враховуються як особливості самої мови [VHDL](#), так і можливості сучасних систем автоматизованого синтезу для ПЛІС. Ці правила відносяться до стадії розробки моделей блоків на рівні регістрових передач. 1. Необхідно використовувати тип [std logic](#) для зовнішніх портів. Не можна призначати сигналам невідоме значення. У деяких випадках можуть створюватися некоректні результати при моделюванні і синтезі. Використовувати значення за замовчуванням або ініціалізацій для сигналів і змінних можна, хоча і не бажано, тільки для моделювання, але не синтезу ([variable](#) B: [INTEGER](#) = 0;). Таке призначення може привести до розбіжностей при моделюванні і реалізації. Необхідно використовувати сигнал скидання для завдання всіх сигналів і змінних. 2. Не можна використовувати порти з буфером для читання вихідних значень. Натомість потрібно додавати інші змінні або сигнали з тим же вихідним значенням. Це пов'язано з тим, що порти буферного типу не можуть бути з'єднані з портами іншого типу і тому даний тип [buffer](#) поширюється на порти всього проекту. 3. Рекомендується використовувати визначення компонент і констант для кожного ядра в одиночному модулі. Потрібно намагатися писати один [VHDL](#) елемент проекту в одному файлі. Назва файлу повинна бути таке ж, як ім'я елемента. Рекомендується намагатися використовувати реалізацію ([instantiation](#)) на ім'я (відповідному), а не по розміщенню. Це спрощує налагодження і читаність коду. Рекомендується використовувати конфігурацію для визначення назв ([entities](#)), архітектур ([architectures](#)) і компонент ([components](#)), наприклад, для визначення такого яскраво вираженого уявлення. Це дозволяє проводити трасування різних архітектур на основі одного файлу. Крім того, це можна використовувати при переході з верхнього рівня проектування на більш низький. 4. Бажано не змішувати в проекті різні стандарти [VHDL](#) (наприклад, не змішувати конструкції [VHDL](#) 87 і [VHDL](#) 93). 5. Бажано компілювати кожен блок в окрему бібліотеку. Рекомендується користуватися константами і налаштованими параметрами ([generics](#)) для розміру буферів,



ширини шин і всіх інших параметрів компонента. Це покращує читаність коду і дозволяє застосовувати його багаторазове використання. 6. Кодування для синтезу. Строго рекомендується читати зі змінних, потім писати в них (читання перед записом) якщо немає повної впевненості в своїх діях. Якщо змінні спочатку записуються, а потім зчитуються, то це призводить до створення довгої комбінації логіки і тригерів-засувки (або регістрів-клямок). Це впливає з того, що змінні отримують своє значення відразу ж, а не так як сигнали. Крім того, необхідно включати всі сигнали, які зчитуються всередині комбінованого процесу в його список чутливості. Це робиться для запобігання появи небажаних тригерів-засувки. 7. Рекомендується уникати використання довгих виразів [if-then-else](#), а натомість використовувати оператор [case](#). Це запобігає появі декодерів високого пріоритету і робить код більш легко читаним. 8. Рекомендується використовувати сигнал [clock enable](#) (PE), як це показано нижче, з одним синхросигналом на процес і не використовувати два різних процеси, один для синхронізації, а інший для комбінаторної логіки. Це пов'язано з тим, що деякі системи синтезу самі знаходять сигнал PE і призначають його. В іншому випадку ніжка PE не буде задіяна, що призведе до появи додаткової логіки. Це стандартна ситуація для проектування [FPGA](#). 10. Бажано писати тестові послідовності в двох частинах, одна частина для генерації даних і перевірки, а друга частина для генерування часових параметрів протоколів шинних інтерфейсів і їх перевірки. Це дозволяє ізолювати дані (перевірка результатів) від перевірки шинних взаємодій і спростити зміну протоколу перевірки при збереженні внутрішньої логіки. 11. Заголовок файлу. Рекомендується використовувати стандартний заголовок на початку кожного файлу. Тема містить базову інформацію про проєкт, файли, автора, ліцензії та ін. Всі пункти заголовка повинні бути обумовлені в проєктній документації. 2.5. Висновки до розділу 3 урахуванням проведених досліджень, описаних у другому розділі, на основі стандартного маршруту проектування обчислювальних пристроїв була запропонована методика розробки складних цифрових ядер багаторазового використання. Розроблено модифікований маршрут проектування складних ядер багаторазового використання. Виділено загальні правила проектування складних ядер і правила, яких необхідно дотримуватися при проектуванні ядер в базисі ПЛІС. Визначено основні критерії ефективності проєктованих ядер. РОЗДІЛ 3. ВИКОРИСТАННЯ ПЛІС - ТЕХНОЛОГІЙ ДЛЯ АВТОМАТИЗОВАНОЇ ПОБУДОВИ ОНТОЛОГІЙ 3.1. Системно-онтологічний аналіз предметної області Згідно [22] під аналізом розуміється вид дослідження, при якому реальний або мислимий об'єкт розчленовується на складові частини (елементи) і досліджуються ці елементи і зв'язки між ними. Аналіз предметної області (ПдО) представляє особливий вид наукової діяльності, в результаті якої будується інтерпретаційна модель предметних знань (в широкому сенсі). У процесі аналізу останні діляться на симетричні і прагматичні знання, концептуальні складові яких представляють онтологічні знання ПдО. Деякі ідеї по розробці методології проектування онтології ПдО беруть свій початок в літературі з об'єктно-орієнтованого підходу (ООП), яке з'явилося як технологія програмування великих програмних продуктів [2]. Однак розробка онтологій як ієрархічної структури понять (концептів) відрізняється від проектування об'єктів як класів і відносин в об'єктно-орієнтованому програмуванні. Останній зосереджується головним чином на методах опису класів - програміст приймає проєктні рішення, засновані на операційних властивостях класу, тоді як розробник онтології приймає ці рішення, ґрунтуючись на структурних властивостях класу. В результаті структура понять і відносини між поняттями в онтології відрізняються від структури класів об'єктів подібної ПдО в об'єктно-орієнтованій програмі [3]. Крім того, при розробці онтології внутрішній зміст поняття есплікується завжди, в той час як в об'єктно-орієнтованому програмуванні найчастіше застосовується метод інкапсуляції як спосіб обмеження доступу до внутрішнього вмісту об'єкта. Системний підхід до пізнання орієнтує аналітика на розгляд будь-якої ПдО з позицій закономірностей системного цілого і взаємодії складових його частин. Системність знань виходить з багаторівневої ієрархічної організації будь-якої сутності, тобто всі об'єкти, процеси і явища можна розглядати як безліч дрібніших підмножин (ознак, деталей) і, навпаки, будь-які об'єкти можна (і потрібно) розглядати як елементи більш високих класів узагальнень. Початок 90-х років минулого століття вважається зародженням парадигми комп'ютерних онтологій. Вона була сформульована як спроба згладити (і по можливості усунути) все частіше проявляються різного роду протиріччя при функціонуванні та впровадженні інтелектуальних систем з використанням баз знань предметних областей. Яскравим представником таких систем на той час були експертні системи (ЕС). Вони успішно і ефективно функціонували в межах одного



колективу і на рівні комерційних зразків. У розробників постійно виникало питання:

” Цитування: 0,04%

id: 17

"Як забезпечити просування цих зразків до кінцевого користувача?".

Були запропоновані розробки оболонок -

” Цитування: 0,01%

id: 18

"порожніх"

ЕС і ряд інших нововведень. Але вони не мали вирішального значення. Напрошувався висновок, що для ефективного функціонування ЕС на найважливішому етапі

” Цитування: 0,01%

id: 19

"життєвого циклу"

- функціонування у кінцевого користувача при вирішенні реальних завдань - необхідно до кожної ЕС

” Цитування: 0,01%

id: 20

"докласти"

експерта у відповідній ПДО. Необхідність присутності експерта пояснювалася, зокрема, швидкоплинністю зміни знань у багатьох предметних областях і відповідно необхідністю оновлення бази знань ЕС в

” Цитування: 0,01%

id: 21

"реальному часі".

У тих же ПДО, де знання мали відносної інваріантністю, ЕС продовжували ефективно функціонувати. Сказане вище і ряд інших чинників

” Цитування: 0,01%

id: 22

"підштовхнули"

вчених до розробки парадигми комп'ютерних онтологій, основні принципи якої були сформульовані в [4]. 1. Дохідливість, ясність ([Clarity](#)). Терміни (і поняття) онтології повинні відображати реальну дійсність. Їх символічні позначення (знаки) повинні формуватися на основі загальноприйнятих правил в семіотики і повинні висловлювати загальноприйняті смисли реальних об'єктів. У свою чергу, ці смисли витягуються із загальноприйнятих визначень термінів (понять), зафіксованих в тлумачних словниках, різних глосаріях ПДО. Судження, що входять до визначення, формалізуються на основі формального загальноприйнятого апарату у вигляді тотожно істинних логічних аксіом. 2. Обґрунтованість, зв'язність ([Coherency](#)). Формування початкового набору понять онтології і їх додаток має бути обґрунтованим, визначеним, в першу чергу, вимогами передбачуваної сукупності вирішуваних завдань. Логічні аксіоми початкового набору понять повинні бути повинні суперечити одна одній. Для цього повинен бути передбачений механізм логічного висновку, який, в тому числі, перевіряє на несуперечливість додаються аксіоми і виводяться в онтології затвердження. 3. Можливість розширення ([Extendibility](#)). Ядром онтології є спочатку введені (спроєктовані) поняття і описують їх аксіоми. В онтології повинен бути передбачений механізм розширення (обмеження) спільно використовуваних словників понять без порушення цілісності системи. 4. Мінімальний вплив кодування ([Minimal encoding bias](#)). У онтологічній системі (ОНС) повинен бути реалізований принцип спільного використання онтологій, який передбачає: специфікацію онтології на рівні уявлення, а не символічного ко-дирования; запис такої специфікації на загальноприйнятому і платформонезависимості мовою опису онтологій можна передати для використання будь-якого програмного агента. 5. Мінімальні онтологічні зобов'язання ([Minimal ontological commitment](#)). Цей принцип перегукується з принципами обґрунтованості і розширюваності / обмеження. Важливо, щоб безліч понять онтології відображало концептуальну структуру ПДС, відносно стабільну протягом

” Цитування: 0,01%

id: 23

"життєвого циклу"

ОНС. А остання надавала можливість розширення або спеціалізації окремих гілок онтологічного графа (он-тографа, ОГ). Відділення концептуальних знань від знань, виражених фактами, є стратегією побудови ОНС, а точніше - онтологічних баз знань. Відомі

загальні сукупності методів, принципів, процедур і атрибутів системного аналізу як наукового пізнання в будь-який ПдО [5, 6 та ін.]. При конкретизації ПдО конкретизуються і засоби системного аналізу. Парадигма комп'ютерних онтологій, що розвивається у взаємодії із засобами і методами системного аналізу, поклала початок розвитку нової гілки засобів і методів системного аналізу ПдО - системно-онтологічного аналізу (або підходу). Центральною ідеєю системно-онтологічного підходу (СОП) є розробка онтологічних засобів підтримки рішення прикладних задач - поліфункціональної онтологічної системи. Така система (точніше, її концептуальна частина) описується кортежем (1) і являє онтологію ПдО, що складається з онтології об'єктів і онтології процесів, і онтологію завдань [7, 8]. ОнС= ОПдО(ОО,ОП),ОЗ (1) На рис. 3.1 представлена схема компонентів проблемного простору (ПрП) і відповідні йому онтології. ПрП - це модель всіх таких аспектів або компонент ПдО, з якими пов'язані (опосередковано або безпосередньо) знання, що вимагаються при вирішенні різних завдань в цій ПдО. Будь-яке ПрП складається з двох блоків: інваріантної (щодо незмінною) частини і безлічі змінюваних частин, відповідних окремим завданням. У складі інваріантної частини, наприклад в методології [SMEЕ](#) ([Structured Methodology for Elicitation of Expertise](#)), виділяють сім типів компонент: об'єкти, інструменти, оператори, операції, кінцеві продукти, побічні продукти і обмеження [9]. Ці типи компонент - суть поняття, які добре групуються в онтології об'єктів і процесів, представлених на рис.3.1. На ньому прийнято такі позначення: ОО - онтологія безлічі об'єктів (понять, концептів) ПДС, яка розглядається як ієрархічна структура класів, підкласів та елементів класів; ОП - онтологія безлічі процесів ПдО, яка розглядається як ієрархічна структура процесів, підпроцесів, дій і операцій; ОЗ - онтологія сукупності завдань, які можуть бути поставлені і вирішені в ПдО. Розглядається як ієрархічна структура завдань, підзадач, процедур і операторів. Рис.3.1. Схема компонентів проблемної області Отже, головним завданням дослідження є розробка формалізованої методики проектування онтологічних знань ПдО, що складаються з онтології об'єктів, онтології процесів і онтології задач. Така методика є важливою складовою при побудові онтологизировать систем знань предметних областей. 3.1.1. Загальний підхід до проектування Відома методологія структурного аналізу і проектування ([SADT](#)) складних систем в довільній предметній області [11]. Ця методологія породила сімейство методик (і відповідних стандартів) [IDEF](#) ([Integrated DEFinition](#)), орієнтованих на розробку моделей ПдО і акцентують увагу на якомусь конкретному аспекті проектування [12, 13]. Зокрема: методика [IDEF0](#) рекомендована для змістовного аналізу і функціонального проектування складних систем управління, в тому числі і програмного забезпечення. Однак в останньому випадку важко визначити її переваги в порівнянні з методологією ООП. Опис об'єктів і процесів в методиці виконується у вигляді ієрархічної сукупності діаграм з лаконічним описом функцій. Блоки на діаграмах виражають функції, тому їх назви - дієслова або віддієслівні іменники; методика [IDEF1](#) х призначена для інформаційного моделювання, заснована на концепції

Цитування: 0,01%

id: 24

"сутність-зв'язок".

Зазвичай відправним пунктом для розробки інформаційної моделі є [IDEF0](#)-модель; методика [IDEF3](#) описує поведінкові аспекти конкретних програм, розглядає послідовність виконання і причинно-наслідкові зв'язки між ситуаціями і подіями для структурного представлення знань про ПдО. Якщо [IDEF0](#) пов'язана з функціональними аспектами і відповідає на питання

Цитування: 0,02%

id: 25

"Що робить система?",

То в [IDEF3](#) дета-лизируются [IDEF0](#) -функції. Ця модель відповідає на питання

Цитування: 0,02%

id: 26

"Як система це робить?"

; методика [IDEF 5](#) призначена для онтологічного аналізу ПдО, аналізу основних термінів і понять, що використовуються для опису об'єктів і процесів, меж використання, а також взаємозв'язків між ними. Служить для ефективного дослідження і документування: словника термінів, які використовуються при описі характеристик об'єктів і процесів, що мають відношення до ПдО, точних і однозначних визначень всіх термінів і класифікації логічних взаємозв'язків між ними. Для кожної з методик сімейства [IDEF](#) розроблені етапи та

стадії побудови моделі ПдО, мови і діаграми представлення результатів. Аналіз отриманих в результаті застосування методик [IDEF](#) моделей ПдО і їх описів з точки зору формалізації та комп'ютерної обробки показав, що, по суті, вони (функціональні моделі та опису) представ-ляють собою змістовне технічне завдання на проектування ОнС, що включає онтологію об'єктів (сутностей) , онтологію процесів і онтологію завдань. Опис ж самої ОнС має дещо інші цілі. Вона орієнтована на комп'ютерне подання за допомогою стандартизованих мов предметних знань з метою широкого використання спільнотою користувачів. Отже: методики [IDEF](#) і системно-онтологічний підхід використовують одні і ті ж безлічі сутностей ПрП, акцентуючи увагу на різних сукупностях характеристик і атрибутів; кінцеві цілі цих двох підходів відрізняються: для [IDEF](#) - функціонально-блокова модель, а для СОП - онтолого-змістова модель. Виконаний вище аналіз дозволяє стверджувати, що методики [IDEF](#) і системно-онтологічний підхід вирішують різні завдання, але зі значним

Цитування: **0,01%**

id: 27

"перекриттям"

деяких функцій. Вельми очевидно це проявляється для пари

Цитування: **0,03%**

id: 28

"онтологія процесів поведінкова модель ПдО ".

Насправді в ООП (як і в онтології задач) основними понятійним одиницями є класи об'єктів (завдань) і методи їх опису (рішення). Очевидно, етапи проектування онтологічних систем виходять з прийнятих в методології [SADI](#) фаз проектування складних систем: аналіз - визначення того, що система буде робити; проектування - визначення підсистем та їх взаємодію; реалізація - розробка підсистем окремо, об'єднання - з'єднання підсистем в єдине ціле; тестування - перевірка роботи системи; установка - введення системи в дію; 6) функціонування - використання системи. 1. Попередній аналіз предметної області У всі методології включений етап попереднього аналізу ПдО або складання змістовного ТЗ на проектування [11-18 і др.]. Цей етап (як і процес проектування бази знань ПдО в цілому) має складний аналітичний характер і полягає в багаторазовому абстрагуванні, в результаті якого з усього різноманіття сторін і властивостей сутностей предметної області виділяються найбільш істотні, релевантні конкретним завданням. Знання предметної області, розуміння суті відбуваються в ній процесів, законів, правил і обмежень, які керують її розвитком, є необхідною умовою успішного вирішення завдань, що стоять перед дослідником. Більш того, наявність таких знань є необхідною умовою постановки, формулювання цих завдань, без чого неможливе саме рішення [19]. Онтологічні системи покликані зробити знання колективним надбанням широкого кола осіб, дати потужний інструмент для фіксації, придбання і обробки знань, перевірки їх на несуперечливість, повноту і т.п. Крім того, складається систематизоване уявлення знань про ПдО, виявляються джерела формування елементів множин і процедур, завдань, які виконуються в аналізованій ПдО. Складається і документується словник термінів ПдО. Етап попереднього аналізу ПдО включає: обґрунтований вибір точного (і достатнього) фрагмента ПрП, щодо якого будуть ставитися і вирішуватися завдання користувача; вибір методів і процедур системно-онтологічного аналізу, якими, зокрема, можуть бути абстрагування і конкретизація, композиція і декомпозиція, структурування, кластеризація і класифікація, тестування і верифікація; складання детального словника термінів і його розбиття на підмножини термінів-об'єктів, термінів-процесів і термінів, які іменують завдання і методи. Якщо проблемне простір являє складну систему, то слід розглянути питання про попередньому етапі проектування на основі методик [IDEF](#), які доповнюють описані вище кроки проектування. Як правило, методика зводиться до алгоритму, який носить ітеративний характер. Для процесу розробки необхідно передбачити ряд

Цитування: **0,01%**

id: 29


"контрольних точок"

для перевірки отриманих результатів на відповідність обраним критеріям. Зазначені критерії повинні співвідноситися з заданими критеріями на проектування бази знань ПдО, так як створення останньої є метою для розробників. Оптимальний результат, як правило, залежить від ступеня опрацювання передбачуваних додатків і варіантів використання онтології. 2. Онтологія об'єктів ПдО Під онтологією об'єктів предметної області розуміється кортеж чотирьох множин:  $OO = X, R, F, A(D, Rs)$ , (2) де -  $X = \{x_1, x_1, ..., x_i, ..., x_n\}$ ,  $i = 1, n$ ,  $n =$

Card X - кінцева множина концептів (понять) заданої ПДС; -  $R = \{r_1, r_2, \dots, r_k, r_m\}$ ,  $R$ :  $x_1 \times x_2 \times \dots \times x_n$ ,  $k = 1, m$ ,  $m = \text{Card} R$ , -

 **Обнаружен Плагиат: 0,37%** <http://dspace.nbuv.gov.ua/bitstream/...> + 3 ресурсів! id: 30

кінцева множина семантично значущих відносин між концептами ПдО. Вони визначають тип взаємозв'язку між поняттями. У загальному випадку, відносини ділять на загальнозначущі (у тому числі виділяють, як правило, відносини часткового порядку) і конкретні відносини заданої ПдО; -  $F: X \times R$  - кінцева множина функцій інтерпретації, заданих на концептах і / або відносинах; -  $A$  - кінцева множина аксіом, які використовуються для запису завжди істинних висловлювань (визначень і обмежень

 **Обнаружен Плагиат: 0,35%** <http://dspace.nbuv.gov.ua/bitstream/...> id: 31

Комп'ютерна онтологія є (формальним) виразом концептуальних знань про предметну область та за своєю значимістю порівнянна з базою знань інтелектуальної інформаційної системи, а її побудова є специфічною формою людської творчості. Творчий процес можна уявити сукупністю операцій-процедур з судженнями, твердженнями, поняттями і відносинами між ними. А останні є фундаментом, основою для побудови складової частини наукової теорії - онтологічної бази знань в заданій предметній області. При цьому такі

знання описуються в декларативній формі. Онтологія визначає загальновживані, семантично значущі


 **Цитирования: 0,02%** id: 32

"понятійні одиниці знань",

якими оперують дослідники і розробники знання-орієнтованих інформаційних систем. Вона відокремлює

 **Цитирования: 0,01%** id: 33

"статичні"

 **Цитирования: 0,01%** id: 34

"динамічні"

компоненти знань ПдО від операціональних знань. На відміну від знань, закодованих в алгоритмах, онтологія забезпечує їх уніфіковане і багаторазове використання різними дослідницькими групами, на різних комп'ютерних платформах при вирішенні різних завдань. 3.1.2. Побудова компонент онтології Нагадаємо деякі відомі визначення, безпосередньо пов'язані з побудови множин концептуальної моделі ПдО або її онтології [20, 21]. Поняття є цілісна сукупність суджень, в яких щось стверджується про відмітні ознаки досліджуваної сутності, ядром якої є судження (чи затвердження) про найбільш загальних і в той же час істотних ознаках цієї сутності. Кожне поняття характеризується обсягом і змістом. Обсяг і зміст поняття - дві взаємопов'язані сторони поняття. Обсяг - клас узагальнених в понятті предметів, зміст - сукупність (зазвичай істотних) ознак, за якими вироблено узагальнення і виділення предметів в даному понятті. Обсяг поняття є визна-ляючим при формуванні ієрархічної структури відповідного онтографа, а зміст - при аксіоматизації його (ОГ) вершин.

 **Обнаружен Плагиат: 0,26%** <http://dspace.nbuv.gov.ua/bitstream/...> id: 35

Всі поняття (або концепти) діляться на ряд класів (по семантичній залежності). В залежності від відображення виду або роду предметів - на видові і родові поняття. В залежності від відображення частини або цілого предметів - на поняття-частини і поняття-цілі. В залежності від кількості відображуваних предметів - на одиничні і загальні поняття. В залежності від відображення предмета або властивості, абстрагованого від предмета, - на конкретні поняття і абстрактні поняття. Онтологія ПдО -

 **Обнаружен Плагиат: 0,06%** <http://dspace.nbuv.gov.ua/bitstream/...> id: 36

це концептуальна модель реального світу і її поняття повинні відображати цю реальність.

Побудова множини  $X$  вважається найбільш важливим моментом при розробці онтології ПдО. Воно повинно бути обов'язково не пустим. Співвідношення між Card X, Card R Card F



характеризують онтологію за функціональною ознакою. Для добре опрацьованих предметних областей за основу безлічі елементів  $\{x_i\}$  може бути взято вміст відповідних тлумачних словників. В іншому випадку слід скласти повний список термінів, в якому вказати (причому перетин обсягів і змістів понять в такому попередньому списку не суттєво):

 **Обнаружен Плагиат: 0,12%** <http://dspace.nbuuv.gov.ua/bitstream/...> id: **37**

чим є кожен термін - поняттям-класом предметів або конкретним поняттям; вказати для кожного терміна можливі суттєві відносини з іншими термінами зі списку;

описати можливі суттєві властивості понять. Відомо, що в будь-якій предметній області існують терміни-синоніми. Для них в онтології відводиться тільки одне поняття, в аксіомах якого може бути вказаний синонімічний ряд термінів. Іншими словами, синоніми одного і того ж поняття не уявляють різні класи. Далі слід уточнити і визначити остаточний список класів-понять, імена яких будуть входити в розроблювану онтологію і бути вершинами онтографа. Також слід прийняти єдині правила присвоювання імен поняттям і їх властивостями. Потім, можливо, слід повторити деякі фрагменти процесу аналізу ПдО (з прив'язкою до складеного списку понять), виконані на попередньому етапі. Відзначимо, що в число зазначених вище

 **Цитирования: 0,01%** id: **38**

"контрольних точок"

(точок входу ітерації) має бути включено завершення розробки будь-якого компонента онтології. В результаті має бути отриманий повний список істотних для заданої ПдО (і передбачуваних додатків) понять і їх машинно-інтерпретовані формулювання. Побудова множини  $R$  також засновано на результатах попереднього етапу аналізу ПдО. По суті, потрібно встановити для кожного елемента  $x_i \in X$  семантичне відношення  $R_k$  з елементом  $X_j \in X$ :  $x_i R_k x_j$ ,  $i, j = 1, N$ ,  $i \neq j$ ,  $k = 1, m$ . Іншими словами, необхідно побудувати безліч ребер, що зв'язують вузли спрямованого онтографа. Як вузлів онтографа виступає безліч понять ПдО. Вершиною (або вершинами) онтографа є родові поняття, яке не має надкласа, а найнижчий рівень представляють конкретні поняття, тобто не мають видових понять в заданій ПдО. На практиці безліч  $R$  на початковому етапі представляють деяким узагальненим ставленням

 **Цитирования: 0,02%** id: **39**

"вище - нижче".

Відомо кілька підходів для розробки ієрархії класів: процес низхідній розробки, процес висхідній розробки і комбінований процес розробки. Останній найбільш часто використовується розробниками, так як він є більш природним, спочатку оперує поняттями середнього рівня, до яких найчастіше звертаються розробники. Потім ці поняття узагальнюються і обмежуються. На закінчення даного підетапи слід співвіднести розроблені класи і їх ієрархії з результатами попереднього аналізу ПдО. Зокрема, уточнюються залежно для конкретних пар  $(x_i, x_j)$ . В процесі співвіднесення (і побудови ієрархії) слід враховувати, що [3]: прямі підкласи в ієрархії повинні розташовуватися на одному рівні узагальнення; клас може бути підкласом декількох класів, і тоді він може успадковувати властивості від усіх цих класів; якщо клас має тільки один прямий підклас, то, можливо, при моделюванні допущена помилка або онтологія неповна; якщо у даного класу є більш дюжини (іноді говорять про число 7) підкласів, то, можливо, необхідні додаткові проміжні класи; в онтології число класів співвідноситься з числом передбачуваних додатків. І слід пам'ятати, що не існує єдино правильної ієрархії класів. Описана побудова онтографа є спеціальним видом класифікації понять ПдО - онтологічної класифікацією. Побудова множин  $F$  і  $A$ . Відомі методики розробки онтології [3, 7, 14 -18] пропонують різні тлумачення цих множин. Залежно від функціональної орієнтації проєктованої онтології безлічі  $F$  і  $A$  можуть інтерпретуватися по-різному:  $A \in F$  - множина аксіом тотожна множині функцій інтерпретацій. У цьому випадку встановлюються істотні зв'язки між розробляються компонентами онтології і варіантами її використання. Основним призначенням такої онтології є однозначна інтерпретація понять, що входять в онтологію, спільнотою користувачів;  $A \neq F$  - безліч аксіом не тотожне безлічі функцій інтерпретацій. В аксіомах задаються а) базові функції (підмножина  $F$ ) або б) додаткові ставлення (які не є елементами безлічі  $R$ ) між поняттями, обмеження і умови, які аналізуються в машині виведення ОНБ і використовуються в процесі вирішення завдань;  $A \neq F$  - безліч аксіом не



тотожне безлічі  $F$ . Функції інтерпретації розглядаються як спеціальний вид відносин на безлічі понять  $F$ :  $x_1 \times x_2 \times \dots \times x_{n-1} = x_n$ . У цьому випадку встановлюються істотні зв'язки між уже раз-прабованими компонентами онтології і сукупністю завдань передбачуваного приложення (додатків). Онтології з таким поданням  $E$  використовуються в питально-відповідних системах, в яких результатом є одне зі значень двоелементною безлічі {істина, брехня} або ім'я предиката. В кінцевому рахунку, незалежно від того, яка з цих формулювань буде прийнята, ефективність розробленої онтології буде визначатися кінцевими результатами додатків. У визначенні концептуальної моделі онтології об'єктів безліч аксіом  $A$  складається з безлічі визначень  $D$  і безлічі обмежень  $RS_i$  для поняття  $X_i$ . Визначення записуються у вигляді тотожно істинних висловлювань, які можуть бути взяті, зокрема, з тлумачних словників ПдО. У них можуть бути вказані додаткові взаємозв'язку понять  $X_i$  з поняттями  $X_j$ . У безлічі обмежень  $RS_i$  можуть бути задані обмеження на інтерпретацію відповідних понять  $X_i$ . Крім того, з повного списку відібраних в онтологію термінів не всі уявляють поняття. Існують терміни (наприклад, рольові), які відповідають властивостям певних класів-понять. Такі властивості слід прив'язати до опису самого загального класу, що володіє ними. А підкласи цього класу будуть наслідувати вказане властивість (звичайно, якщо між ними встановлено певний стосунок часткового порядку). Властивості понять мають певні значення, такі як тип значень, потужність значень, дозволені значення (для даного класу) і інші. Наприклад, потужність значень можна описати: з одиничною потужністю, потужністю без обмежень і потужністю з деяким допустимим інтервалом. На основі побудованих множин кортежу можна синтезувати концептуальну модель ПдО, наприклад, за допомогою відомого інструментального кошти **Protege** і отримати формальне опис розробленої онтології на одній з мов опису онтологій, а також графічне представлення онтографа. 3. Онтологія процесів ПдО Синонімами онтологій об'єктів і процесів є відповідно статична і динамічна онтології ПдО. У науково-технічній літературі, коли говорять про онтологію ПдО, то мається на увазі її статична складова. Саме компоненти останньої найбільш розроблені, як в літературі з філософії, так і в конкретних описах ряду предметних областей. Під поведінковим описом сутностей-процесів найчастіше розуміють розробку певних бізнес-процесів. Їх результатом є графічні діаграми і природно-мовні опису процесів. Розробка ж бази знань не є прямою метою зазначених методик. Тому методики розробки онтології процесів практично невідомі. Хоча слід зазначити, що в деяких відомих онтологіях верхнього рівня (наприклад, онтологія Дж. Сови, **SUMO** і **Mikrokosmos**) сутність поняття

” Цитування: 0,01%

id: 40

"Процес"

розглянута досить детально [22-25]. На рис. 3.2 представлений синтезований онтограф (на основі аналізу зазначених онтологій), який представляє схему початкового розвитку поняття

” Цитування: 0,01%

id: 41

"Процес",

причому тільки тієї її частини, яка відповідає процесам в науково-технічних предметних областях. Гілки онтографа

” Цитування: 0,01%

id: 42

"Соціальний процес",

” Цитування: 0,01%

id: 43

"Матеріальний процес"

і їм подібні не розглядаються. Категорія Процес розглядається як Дійсність і Подієвість, на відміну від категорії Об'єкт, який характеризується як Дійсність і Тривалість [26]. В першу чергу Процес розглядається як часопов'язувальна категорія і потім поділяється за видами змін, наявністю початкових і кінцевих точок і т.д. Далі Процес підрозділяється на безперервний і дискретний. Перший з них характеризується наявністю експліцитно початкової і кінцевої точок або як без явної вказівки цих точок. Другий вид процесу вказує, що зміни відбуваються дискретними кроками, названими подіями, які чергуються з періодами спокою, названими станами. Наведена схема початкового розвитку онтології процесів не відображає всіх характеристик (підстав розгалужень в онтографі) категорії

” Цитування: 0,01%

id: 44

"Процес",

навіть для тієї її частини, що представлена на рис.3.2. Рис.3.2. Початковий розвиток онтології процесів На рис.3.3. представлена загальна схема онтології процесу ПдО, в якій категорія

Цитування: 0,01%

id: 45

"Процес"

представлена онтографом з  $r$  рівнями тощо підпроцесами (ПП) на кожному рівні. Передостанній рівень представлений множинами дій, на які розбивається кожен ПП попереднього рівня. У свою чергу, кожна дія на останньому ( $r$ -ом) рівні розбивається на послідовність операцій Огр. Рис.3.3. Загальна схема онтології процесу Зв'язки між підпроцесами для сусідніх рівнів відповідають відносинам

Цитування: 0,01%

id: 46

"Ціле-частина",

а всередині кожного рівня - деякої змішаною формою організації з'єднань. На рис.3 показаний окремий випадок такої організації - паралельний. Подальший розвиток (конкретизація) онтології процесів можливо, коли задана конкретна предметна область і відповідне проблемне простір, а в більш вузькому сенсі - конкретні ознаки розгалужень (умови ініціювання ПП, умови закінчення ПП і обмеження) в онтографі. Обсяг динамічних знань  $W$  в предметній області можна оцінити через характеристики (параметри) їх формально-онтологічних уявлень. Зокрема, при поданні онтографом (без урахування складності функцій інтерпретації підпроцесів) величина  $W$  може характеризуватися числом вершин ОГ. Для ОГ, представленого на рис.3, це число може бути виражено формулою де - ПП - підпроцес (вершина ОГ), Д - дія (вершина ОГ на  $(p-1)$ -ом рівні), О - операція (вершина ОГ на  $p$ -ом рівні);  $k = 1, p, p$  - кількість рівнів в онтографі процесу;  $i = 1, p, p$  - кількість підпроцесів на  $k$ -ом рівні;  $i = 1, t, t$  - кількість підпроцесів в  $g$ -ом підпроцесі  $(k-1)$ -го рівня. 3. Онтологія завдань ПрП У текстових описах (специфікаціях) цільових завдань виділяються набір об'єктів і набір процесів (методів), необхідних і достатніх для виконання конкретних цільових завдань. Можна виділити деякий уніфіковане (поповнюється) безліч базових завдань (типових фрагментів завдань), на основі яких за допомогою певних логічних послідовностей конструювати складніші завдання. Таким чином, рішення прикладної задачі включає, в тому числі, взаємодія трьох онтологій: онтології об'єктів, онтології процесів і онтології базових задач [7, 8]. Полем конструктора, що описує схеми типів вирішуваних завдань, є деяка семантична вісь з протилежними сторонами

Цитування: 0,01%

id: 47

"Вибір"

i

Цитування: 0,01%

id: 48

"Побудова",

концепти яких описують просту і складну схеми рішення задач. Під простою схемою (схема

Цитування: 0,01%

id: 49

"Вибір"

) розуміється така схема, в якій для вирішення завдання онтологічної системі необхідно зробити вибір із сукупності відомих схем рішень. Коли вибір зроблений, то відомі метод, процес і його стану, тобто рішення задачі стає тривіальним. Протилежністю простої схеми є складна схема (схема

Цитування: 0,01%

id: 50

"Побудова"

), коли всі складові процесу рішення задачі невідомі. На практиці, найчастіше, зустрічаються деякі проміжні схеми рішення задач. Наприклад, схеми, в яких початкові і цільові стану, сукупності методів задаються в явному вигляді. Загальною стратегією вирішення складних (непростих) завдань є багаторівнева декомпозиція вихідної задачі до того рівня, на якому отримані підзадачі є простими. При цьому онтологічні знання є активною основою процесу декомпозиції. Онтологічні знання, що описують деякий ПрП,

можна розділити на наступні компонентні знання: типи вхідних і вихідних даних, інструменти, оператори (людина або комп'ютерна програма) і операції (дії оператора або розв'язувача завдань) [9]. Для реалізації підходу необхідно розробити уніфікований мову уявлення онтологічних знань і інструментальну середу як набір спеціалізованих і універсальних базових операцій, які керують процесом рішення. Необхідно також розробити Формирователь завдань, який здійснює вибір засобів і методів формування структури завдання на основі базових операцій. Схема моделі онтології задач описується кортежем (4) де ОЗПрП - узагальнена задача проблемного простору, що складається з  $p$  завдань, які, в свою чергу, складаються з  $q = 1, Q$  фрагментів кожна. Кожен фрагмент представлений процедурою, реалізованої на безлічі  $r = 1, R$  операцій кожна;  $M$  - безліч методів вирішення завдань;  $PЗ$  - вирішувач завдань. На рис.3.4. представлена схема онтології задач. Одна з переваг онтологічного підходу, зокрема, ієрархічного уявлення полягає в тому, що складне завдання великої розмірності розбивається на послідовно вирішуються групи завдань малої розмірності. Онтологія завдань в якості понять містить типи вирішуваних завдань, а відносини цієї онтології, як правило, специфікують декомпозицію задач на підзадачі. На рис.3.5. представлений алгоритм методики проєктування онтології ПдО і онтології задач. Передбачається, що безліч функцій інтерпретації в моделях онтологій тотожне безлічі аксіом. На малюнку прийняті наступні скорочення: КТ - контрольні точки; ПрП - проблемний простір; ТС - тлумачний словник; Т-О, Т-П - терміни-об'єкти і терміни-процеси. Одна з переваг онтологічного підходу, зокрема, ієрархічного уявлення полягає в тому, що складне завдання великої розмірності розбивається на послідовно вирішуються групи завдань малої розмірності. Онтологія завдань в якості понять містить типи вирішуваних завдань, а відносини цієї онтології, як правило, специфікують декомпозицію задач на підзадачі. На рис.3.5. представлений алгоритм методики проєктування онтології ПдО і онтології задач. Передбачається, що безліч функцій інтерпретації в моделях онтологій тотожне безлічі аксіом. На малюнку прийняті наступні скорочення: КТ - контрольні точки; ПрП - проблемне простір; ТС - тлумачний словник; Т-О, Т-П - терміни-об'єкти і терміни-процеси. Рис.3.4. Схема онтології задач Рис. 3.5. Граф-схема алгоритму проєктування онтологій

3.2. Системи верифікації на основі реконфігурованих пристроїв

3.2.1. Системи цифрового проєктування [HOT](#) Фірма [Virtual Computer Corporation \(VCC\)](#) [4] на основі системного підходу об'єднала сучасну технологію проєктування реконфігурованих компонент з набором сучасних інструментальних засобів програмного забезпечення в єдину систему цифрового проєктування - [Hardware Object Technology \(HOT\)](#). Ця система спільного проєктування програмного та апаратного забезпечення містить всі компоненти, необхідні для реалізації конфігурується обробки даних в одному пакеті. Використовуючи мови [HDL \(hardware description languages\)](#), систему [HOT](#) і співпроцесорні плати з вбудованим контролером шини [PCI](#), проєктувальник може використовувати механізм

Цитування: 0,01%

id: 51

"reconfigurable computing"

для прискорення верифікації алгоритму, емуляції проєкту і швидкого моделювання. Верифікація проєктів в реальному масштабі часу використовує реальні дані при конфігуруванні апаратних засобів. Система проєктування [HOT II](#) є засобом для оцінки, моделювання та макетування проєктів, що використовують кристали [FPGA](#) в якості обчислювального елемента, вбудований програмований таймер і інші елементи системи (структура приведена на рис. 3.6). Комбінація цих елементів доступна через системну ініціалізацію апаратних і програмних засобів для користувача проєкту. Менеджер конфігурацій ([Configuration Cache Manager - CCM](#)) дозволяє включати апаратні засоби проєкту в програму прикладного програмного забезпечення і динамічно завантажувати проєкт під час виконання програми. Інтерфейс [HOT II](#) реалізує інтерфейс шини [PCI](#), контролер якої формується за допомогою генератора [LogiCORE](#) і завантажується в [FPGA](#), а також спеціальний внутрішній інтерфейс [VCC - HOS \(Hardware Operating System Interface\)](#) для зв'язку з двома повністю незалежними 32-розрядними банками пам'яті і [CCM](#). [CCM](#) може конфігурувати [FPGA](#) з двох джерел пам'яті конфігурації, наявних на платі, - [Flash](#) і [RAM](#). Рис.3.6. Структура системи [HOT II](#) Під час реконфігурації доступ до плати заборонений драйвером. [Cache](#) - пам'ять конфігурацій об'ємом 128 КБ може підтримувати до 3 конфігурацій кристала, при цьому користувач має можливість завантажувати пам'ять конфігурації через шину [PCI](#). Файл конфігурації проєкту перетворюється в код програмного елемента, що завантажується в плату [HOT II](#) командами прикладної програми. [PCI](#) плата

HOT II має дві незалежні шини (кожна з яких містить 32 розряду даних і 24 розряди адреси). Для кожної з цих двох шин є коннектор введення-виведення. Стандартна конфігурація HOT2 містить кристал серії [Spartan XCS40-4](#), що включає контролер шини [PCI](#) і призначену для користувача конфігуровані частина; кристал [XC95108-15](#) для призначених для користувача реалізацій; швидкодіючу пам'ять [SRAM](#) об'ємом 1 МБ, організовану як два незалежних 32-бітових банку; [Flash](#) - пам'ять конфігурації (128кб); [Cache](#) - пам'ять конфігурації (128кб); зовнішні роз'єми для плат розширення. На платі є джерела живлення напругою 3,3 В і 5В. Розширена конфігурація HOT2 -XL містить кристал [XC4062XLT-1](#) (замість кристала [XCS40-4](#)); швидкодіючу пам'ять [SRAM](#) (4МБ), організовану як два незалежних 32 -бітових банку; [Flash](#) - пам'ять конфігурації (2МБ); [Cache](#) - пам'ять конфігурації (512Кб). Інші параметри аналогічні стандартної версії. Віртуальні інструментальні засоби - [Virtual Workbench \(VW\)](#) призначені для прискореної реалізації нових проєктів на кристалах [FPGA](#) сімейства [Virtex](#) фірми [Xilinx](#). Стандартна конфігурація (VW-300) містить користувальницький кристал [XCV-300](#); кристал [XC9500](#), службовець в якості контролера конфігурацій; [Flash](#) - пам'ять об'ємом 2МБ; пам'ять [SDRAM](#) об'ємом 64МБ. Інструментальних засобів проєктування для кристалів [Virtex](#) доступні асинхронний послідовний порт [RS-232](#); вісім знакових індикаторів ([Character Display](#)); восьмипозиційні [DIP](#)-перемикачі; вісім світлодіодів; [JTA](#)-коннектор; робоче місце для макетування; 10 роз'ємів для введення-виведення або плат розширення, які реалізують аналогоцифрове або цифро-аналогове перетворення, а також канали зв'язку між окремими компонентами.

3.2.2. Моделююча система [System Explorer](#) Вироби фірми [Aptix](#) включають сімейство [System Explorer](#) - реконфігурованих емульованих систем, набір додаткових моделюють апаратних засобів і комплект програмного забезпечення [5, 6]. За допомогою реконфігурованих апаратних засобів є можливість створення високоефективної функціональної моделі проєкту

Цитування: 0,02%

id: 52

"система в кристалі"

- [SoC \(System-on-Chip\)](#). Є дві моделі апаратних засобів [System Explorer](#) - [MP3C](#) і [MP4](#). Обидві моделі використовують методологію блокового моделювання, за допомогою якої користувач створює реконфігурованих дослідний зразок своєї системи на базі кристалів [FPGA](#) та інших системних компонентів. Модель [MP3C](#) забезпечує максимальні можливості для обробки потоків даних конвеєрного типу і прийнята в якості платформи для верифікації цифрових систем. Архітектура моделі [MP4](#) призначена для проєктів з більш складними типами даних, наприклад, мережевих систем і мультимедіа. Комплект інструментального програмного забезпечення [APTIX](#) служить для виконання і перевірки проєктів користувачів на платформах [MP3C](#) і [MP4](#). Як діалогового ієрархічного інструменту для групування списку з'єднань між [FPGA](#) і іншими компонентами системи використовується утиліта [Logic AggreGATER](#). Програмне забезпечення [Explorer](#) формує список з'єднань для прийнятої платформи [MP3C](#) / [MP4](#), конфігурує модель і забезпечує автоматизовану налагодження апаратних засобів. До складу додаткових моделюють апаратних засобів входять 11 модулів, які представляють собою кристали [Virtex 2000E](#) фірми [Xilinx](#), кристал [FLEX 10K250](#) фірми [Altera](#), плати розширення пам'яті різних обсягів, аналого-цифровий і цифро-аналоговий перетворювачі, кабелі, з'єднувачі і т.д. Сімейство виробів [System Explorer](#) побудовано на двох ключових доповнюють один одного технологіях, реалізованих відповідно у вигляді програмованої користувачем схемної плати ([Field Programmable Circuit Board](#) - [FPCB](#)) і програмованого користувачем компонента комутації ([Field Programmable Interconnect Component](#) - [FPIC](#)). [FPIC](#) є програмований кристал в 1024-контактному корпусі, що забезпечує до 1000 доступних зовнішніх сполучних точок. Даний кристал реалізує двонаправлені сигнальні ланцюга. При цьому тимчасова затримка між контактами вхід-вихід становить всього 5нс. Він ідеально підходить для розробки макетних і дослідних зразків, коли для перевірки нових схемних ідей потрібно динамічна внутрісистемна реконфігурація. Схемна плата [FPCB](#) забезпечує програмовані міжз'єднання між блоками (кристалами) проєкту. [FPCB](#) реалізує принцип модульної компоновки, що полягає в тому, що блоки системи, представлені компонентами на платах розширення, включаються в конфігурується поле на [FPCB](#). Конфігурується поле являє собою набір сполучних панелей ([socket](#)), призначених для з'єднання модулів [FPGA](#) і компонентів системи між собою. Кожен висновок сполучної панелі з'єднаний з одним з висновків кристала [FPIC](#), програмування якого визначає з'єднання між макетованими компонентами. Вироби фірми [Aptix](#) використовують структурні опису блоків на системному рівні і список



з'єднань [ASIC](#) на вентиляльному рівні для перетворення елементів системи в стандартні компоненти [FPGA](#). Програмне забезпечення [Aptix](#) генерує дані для програмування макетованого зразка, а також управляє інтерфейсами з налагоджувальними інструментами, такими як логічні аналізатори. Система [Explorer](#) сумісна як з внутрішньої методологією проєктування замовника, так і з системами автоматизованого проєктування електронних виробів ([ESDA](#)), а також набором різних засобів синтезу. Модульна методологія верифікації, підтримувана системою, передбачає створення прототипу, що спрощує налагодження і оперативну реалізацію змін проєкту. Ієрархічна організація прототипу проєкту створюється за принципом

Цитування: 0,02%

id: 53

"блок за блоком".

Відображення проєктів на логіку [FPGA](#) виконується за допомогою програмного забезпечення [Design Pilot](#) (рис. 3.7). [Design Pilot](#) відображає замовну логіку на цільовій кристал [FPGA](#) з автоматичним ієрархічним блоковим поділом і подальшим групуванням. Допускається безпосереднє управління відображенням логіки за допомогою графічного інтерфейсу користувача ([GUI](#)). Програмне забезпечення [Explorer](#) 2000 відображає проєкт в фізичні компоненти системи макетування і автоматично встановлює зв'язку верхнього рівня ієрархії за допомогою [FPGA](#)-кристалів. Реалізація замовної логіки в [FPGA](#) здійснюється як шляхом автоматичного синтезу проєкту, так і вибором наявних ядер типу [CPU](#), [DSP](#), блоків або мікросхем пам'яті. Прототип автоматично конфігурується з урахуванням забезпечення вимог проєкту [SoC](#). Після конфігурації прототипу [Explorer](#) 2000 автоматизує апаратне налагодження. Рис. 3.7. Відображення ієрархічного проєкту Система [Explorer](#) включає також інтерфейс моделювання з пакетом [ModelSim](#) (мова опису [Verilog](#) / [VHDL](#)). Розробники можуть розміщувати макетовані модулі в системі і комбінувати (об'єднувати) модулі [FPGA](#) з іншими компонентами, включаючи [IP](#) блоки. Схеми, макетовані в системі [Explorer](#), зазвичай працюють на частотах від 5 до 20 МГц, що дозволяє виконувати верифікацію практично в реальному масштабі часу для багатьох застосувань. Швидкодія шини і системи введення-виведення може досягати 50 МГц або вище. Налагодження частин апаратних засобів проєкту виконується за допомогою наявного в наявності логічного аналізатора. Налагодження програмного забезпечення реалізується шляхом використання внутрішньосхемною емуляції. Конфігуровані проєкти використовують програмне забезпечення [Logic AggreGATEr](#) для ієрархічного відображення файлів проєктування, виконаних на мові [Verilog](#) або представлених в форматах [EDIF](#) і [XNF](#), в макетовані елементи. Платформа [MP3CF](#). Система [Explorer MP3CF](#) оптимізована для макетування [DSP](#)-подібних проєктів з конвеєрної організацією і помірними вимогами для зв'язків між макетованими компонентами. Архітектура [MP3CF](#) забезпечує максимальні характеристики для прототипів, що включають компоненти макетування типу [CPU](#), [DSP](#), карти пам'яті та інші з зумовленими контактами. Рис. 3.8. Архітектура з'єднань системи [Explorer MP3CF](#) Платформа [MP3CF](#) ефективно використовується для побудови швидкодіючих прототипів пристроїв бездротового зв'язку і обробки цифрових зображень. Архітектура міжз'єднань показана на рис. 3.8. Поле макетування має 1920 контактних точок з програмованими зв'язками і в даний час підтримує до 12 [FPGA](#). Платформа [MP4CF](#) (рис.3.9) оптимізована для цілей макетування систем і пристроїв, що мають велику ширину внутрішніх шин і велику довжину зв'язків. Архітектура [MP4CF](#) забезпечує максимальну гнучкість для прототипів, що включають більше, в порівнянні з [MP3CF](#), число [FPGA](#) і тільки кілька компонентів макетування з зумовленими контактами (типу [CPU](#) або [DSP](#)). Платформа [MP4CF](#) використовується для побудови прототипів мережевих систем і мультимедіа. Рис. 3.9. Архітектура з'єднань системи [Explorer MP4CF](#) Поле макетування має 2880 сполучних точок з програмованими зв'язками і підтримує до 20 [FPGA](#). Методологія блочного проєктування передбачає поетапну розробку системи, починаючи з високорівневою розробки алгоритму, з подальшою блоковою реалізацією і потім трансляцією функціональних блоків в елементи програмного забезпечення і апаратні засоби. Кожен блок елементного рівня повинен бути розроблений і перевірений на більш детальному рівні, в кінцевому рахунку вся система повинна бути верифікована в реальному середовищі, щоб гарантувати необхідні функціональні можливості. Значимість блокової методології макетування складається в тому, що вона інтуїтивно зіставляє паралелі процесу створення проєкту, який використовується великими групами проєктувальників, забезпечує механізм ітераційного відображення і верифікації проєктних блоків по відношенню до прототипу і паралельно з процесом розробки проєкту, а сам [IP](#)



проект допускає багаторазове використання для SoC. 3.2.3. Система FUSE Прискорити процес розробки проектів можна за допомогою використання спеціалізованої мови DIME script, що забезпечує взаємодію з навколишнім середовищем при реконфігурованню обробці [7]. Ця мова підтримується програмним забезпеченням для виробів фірми Nallatech через систему FUSE (Field Upgradeable Systems Environment), як показано на рис.3.10. Рис. 3.10. Архітектура системи проектування і верифікації DIME script містить простий набір команд, який скорочує набір апаратних засобів при використанні складних інтерфейсів програмування.

Цитування: 0,02%

id: 54

"Plug & Play"

програмне забезпечення системи забезпечує гнучкість і масштабованість управління і конфігурації кристалів FPGA на модулях і материнських платах. Обмін даними між материнською платою і Host комп'ютером реалізується поруч інтерфейсів, таких, як GUI та API (Application Programming Interface). Потужний інтерфейс API допускає швидку розробку прикладного програмного забезпечення для систем на основі FPGA. Система FUSE підтримує широкий діапазон мов через доступні інтерфейси API, включаючи C / C ++, Java, MATLAB і DIME script. FUSE забезпечує зв'язок між основними засобами розробки, такими, як VHDL для FPGA і C ++ для DSP-процесорів з інструментальними засобами високого рівня, такими, як System C, System Generator для MATLAB, безпосередньо MATLAB і Handel C Celoxica. Масштабоване сімейство виробів DIME-II включає материнські плати, модулі розширення, програмований устаткування і програмне забезпечення. Програмне забезпечення FUSE фірми Nallatech сумісно з операційними системами Windows, Linux і VxWorks. Система FUSE призначена для використання в наступних програмах: реконфігуровані обчислювальні системи; системне тестування / налагодження платформ DIME; розробка прикладного програмного забезпечення для FPGA-додатків; безпосередня інтеграція апаратних засобів і програмного забезпечення для реконфігурованих комп'ютерів на базі кристалів FPGA. Інтегроване середовище розробки IDE (Integrated Development Environment) являє собою засіб взаємодії з системою FUSE. Застосовуючи IDE, користувач має можливість виявити будь-які FUSE - сумісні плати в системі і здійснити перегляд / конфігурація логічних ресурсів кристалів на платі, використовуючи засоби FPGA Explorer. Система FUSE допускає формування на материнській платі DIME / DIME-II наступних конфігурацій кристалів: одного користувальницького кристала FPGA; всіх кристалів FPGA на материнській платі і кристалів FPGA на будь-яких модулях, що підключаються до материнської плати; всіх кристалів FPGA на безлічі материнських плат в системі і всіх кристалів FPGA на модулях, закріплених за цими материнськими платами. Материнські плати підтримуються будь-якою версією FUSE (для Windows, Linux або VxWorks) в залежності від індивідуальних вимог. Плата BenNUEY забезпечує доступ до масштабованої системі модуля DIME-II. Призначений для користувача кристал Virtex-II призначений виключно для проектів користувача. Другий кристал (Spartan-II) забезпечує попереднє конфігурування програмованого обладнання (контролер шини PCI). Плата BenNUEY має високий рівень зв'язності між кристалами FPGA і трьома слотами модуля DIME, які забезпечують гнучку можливість розширення за рахунок підключення стандартних модулів. Це дає можливість створювати повністю замовні системи на базі кристалів FPGA з використанням стандартних виробів. Модуль BenPRO як частина масштабується сімейства DIME-II може бути легко інтегрований в системи за допомогою материнських плат DIME-II, програмного забезпечення і програмованого обладнання. Модуль BenPRO містить кристали Virtex-II PRO (XC2VP7 або XC2VP20), які мають вбудовані PowerPC-процесори, допускаючи, таким чином, що один DIME-II модуль включає до 4 апаратних процесорів. Він може забезпечувати ширину смуги частот введення-виведення близько 25 Гбіт / сек, використовуючи до 4 приймачів введення-виведення типу Rocket на додаток до стандартного вводу-виводу. Для підтримки проміжних даних і пам'яті програм включені 2 швидкодіючих банку пам'яті (DDR SDRAM або ZBT SRAM). Це дає можливість вибрати відповідну архітектуру пам'яті для алгоритму або необхідну архітектуру платформи. DIME-II надає можливість користувачам жорстко з'єднати 4 модуля BenPRO на cPCI-платі типу BenERA, що забезпечує 8 PowerPC-процесорів на одній платі. Із застосуванням FUSE ця система може бути розширена до декількох плат, що призводить до безмежного числа процесорів в межах всієї системи. 3.2.4. Система XtremeDSP Фірмами Xilinx і Nallatech представлена система XtremeDSP Development Kit для високоефективної розробки систем DSP [8], яка служить професійної платформою розробки проектів на базі кристалів Virtex-II.

Дана система містить платформу розробки апаратних засобів, програмні засоби [FUSE](#) фірми [Nallatech](#), джерела живлення і кабелі. [XtremeDSP Kit](#) включає інструментальні засоби типу [MATLAB](#) / [SIMULINK](#) фірми [MathWorks](#), [System Generator](#) для [DSP](#), а також пакети програмного забезпечення: [ISE](#) фірми [Xilinx](#), [Synplify Pro](#) фірми [Synplicity](#) і [FPGA Advantage](#) фірми [Mentor Graphics](#). Блок - схема процесу проектування представлена на рис. 3.11. Дана платформа містить здвоєні 14-розрядні ЦАП, що працюють з швидкодією до 160 [Mega-samples / sec \(MSPS\)](#), здвоєні 14-розрядні АЦП, що працюють з швидкодією до 65 [MSPS](#), що забезпечує високоефективну цифрову обробку сигналів на базі кристалів [FPGA](#). Інтерфейс з платформою апаратних засобів реалізується через шини [PCI](#) або [USB](#) з наданням можливості користувачеві вибору інтерфейсу. Рис. 3.11. Блок-схема процесу верифікації [DSP](#) - додатків [System Generator](#) для програмного забезпечення [DSP](#) платформи [System Generator](#) (розроблений спільно фірмами [Xilinx](#) і [MathWorks](#)) дозволяє автоматично переносити моделі, створені в середовищі [MATLAB](#) / [Simulink](#), в середу фізичного синтезу проєктів для [FPGA](#) [9]. Використання [System Generator](#) скорочує шлях від концепції проєкту до працюючих апаратних засобів завдяки таким властивостям: Простота. Проєкти подаються на відповідному рівні абстракції. Зміни в проєкті автоматично транслюються в відповідні зміни трансльованих блоків. Гнучкість. Моделі потоків даних, традиційні мови опису апаратних засобів ([VHDL](#) і [Verilog](#)) і функції, отримані на основі мови програмування [MATLAB](#), можуть моделюватися разом і синтезуватися в апаратні засоби. Швидкодія. Моделювання за допомогою [System Generator](#) виконується значно швидше, ніж на традиційних [HDL](#)-симуляторах. При цьому результати більш прозорі для аналізу. Потужність. [System Generator](#) реалізований в рамках системи [Simulink](#) (фірма [MathWorks](#)) і є розширенням набору інструментальних засобів [Simulink](#) і [MATLAB](#). [System Generator](#) включає підтримку для [ModelSim](#), яка дозволяє користувачам імпортувати [HDL](#)-код і надає можливість проєктувальнику апаратних засобів реалізувати моделювання на системному рівні. Точність. Результати моделювання точно відображають ті помилки, які можуть мати місце в апаратних засобах. [System Generator](#) надає набір блоків, реалізований у вигляді функцій, до складу якого входять параметризовані модулі математичних, логічних і [DSP](#)-функцій, модулі для взаємодії з [Simulink](#), спеціальні конструкції для роботи з програмним забезпеченням [Xilinx](#). За допомогою утиліт [Netlister](#) і [Mapper](#), [Simulink](#) модель транслюється в [VHDL](#)-опис. [System Generator](#) також містить [Testbench Generator](#), який дозволяє формувати з тестових впливів і реакцій модельованого об'єкта тестові вектори для [VHDL](#)-симулятора. Пакет проєктування [ISE](#) фірми [Xilinx](#) містить засіб синтезу функціональних блоків високого рівня ([IP](#)-ядер) [COREGen](#), яке дозволяє автоматично генерувати широкий спектр функціональних вузлів: від сумматорів і лічильників до цифрових фільтрів і блоків спектрального аналізу. Вихідні дані генератора представлені в форматі [EDIF](#) і описом ядра на мові [VHDL](#) / [Verilog](#) і / або у вигляді макроелементи для САПР [Xilinx Foundation](#). Всі модулі [COREGen](#) оптимізовані під архітектуру використовуваного кристала ПЛІС. Доступні еталонні тести показують, що швидкодія проєктів на базі кристалів [FPGA](#) серії [Virtex-II](#) приблизно в 100 разів вище, ніж швидкодія промислових процесорів [DSP](#). В результаті один кристал [FPGA](#) може замінити кілька процесорів [DSP](#). У табл. 3.1 наведені порівняльні характеристики за швидкодією при реалізації деяких блоків [DSP](#) стандартними засобами та на основі кристалів серії [Virtex](#). Таблиця 3.1 Порівняльні оцінки за швидкодією

Функція	Швидкодія
Промислове швидкодіюче ядро процесора <a href="#">DSP</a>	Проект на базі Кристала <a href="#">Virtex-E-08</a>
Проект на базі кристала <a href="#">Virtex-II</a>	Множення з накопиченням (8x8 біт) за секунду. 4,4 128 600
<a href="#">Multiply and accumulate per second (MAC)</a>	(млрд.)
<a href="#">FIR</a> фільтр: 256 - відгалужень з лінійної фазою 17 160 180 16-бітові дані і коефіцієнти	<a href="#">Mega-samples / sec (MSPS)</a> <a href="#">FFT</a> -1024 точки (16-бітові комплексні числа) (мкс) 7,7 4,1 1

Одним з широко використовуваних засобів верифікації є пакет [ChipScope Pro](#), розроблений фірмою [Xilinx](#) [10], який складається з трьох модулів: [ChipScope Pro Core Generator](#), [ChipScope Pro Core Inserter](#) і [ChipScope Pro Analyzer](#), що забезпечують перевірку кристалів серій [Virtex](#), [Virtex-E](#), [Virtex-II](#), [Virtex-II Pro](#), [Spartan-II](#), [Spartan-IIe](#) і [Spartan-III](#). Взаємодія [PC](#) і перевіряється кристала [11] здійснюється через порт [JTAG](#) ([IEEE Std. 1149.1](#)). На рис. 3.12 представлена блок-схема з'єднання [PC](#) з верифікованим кристалом [2]. Рис. 3.12. Блок-схема з'єднання [PC](#) з верифікованим кристалом

Призначення модулів [ChipScope Pro Core Generator](#) призначений для формування списків ланцюгів і шаблонів (функцій), які генерують вихідні коди для певного набору вхідних параметрів ядер (готових технічних рішень), до яких відносяться: інтегрований контролер ([Integrated Controller](#)) - [ICON](#); інтегрований логічний аналізатор ([Integrated Logic Analyzer](#)) з трасувальником ядра ([Agilent Trace Core](#)) - [ILA](#) / [ATC](#); інтегрований шинний аналізатор ([Integrated Bus Analyzer](#)) для шини [IBM CoreConnect On-Chip Peripheral Bus-IBA](#) /

[OPB](#); інтегрований шинний аналізатор ([Integrated Bus Analyzer](#)) для шини [CoreConnect Processor Local Bus-IBA / PLB](#); віртуальний вхід / вихід ([Virtual Input / Output - VIO](#)). [Core Inserter](#) автоматично включає ядра [ICON](#), [ILA](#) і [ILA / ATC](#) в синтезується користувачем проєкт. [Analyzer](#) забезпечує конфігурація кристала, початкову установку і індикацію виконання програм при роботі з ядрами [ILA](#), [ILA / ATC](#), [IBA / OPB](#), [IBA / PLB](#) і [VIO](#). Ядро [ICON](#) пов'язано із спеціальними висновками системи периферійного сканування. Користувач має можливість за допомогою [Core Generator](#) сформувати ядра [ICON](#), [ILA](#), [ILA / ATC](#), [IBA / OPB](#), [IBA / PLB](#) і [VIO](#) і потім розмістити отримані блоки в проєкті. В результаті чого буде згенерований код [HDL](#). Можливо також включення ядер [ICON](#), [ILA](#) і [ILA / ATC](#) безпосередньо в список ланцюгів синтезованого проєкту за допомогою програми [Core Inserter](#). Далі виконуються процедури компоновки і трасування проєкту в кристалі, які реалізуються засобами системи [Xilinx ISE Foundation](#). Користувач може завантажити сформований файл конфігурації з блоками перевірки в кристал і провести аналіз проєкту за допомогою програми [ChipScope Pro Analyzer](#). В рамках [XtremeDSP](#) ініціативи для прискорення процесу розробки і макетування систем в області [DSP](#) фірма [Xilinx](#) співпрацює з численними партнерами, використовуючи розроблені ними платформи на базі кристалів ПЛІС. Наприклад, [ADM-XRC](#), [ADC-RC](#) фірми [Alpha Data Parallel Systems](#); [Wildcard](#), [Firebird](#) фірми [Annapolis Micro Systems](#); [AVR32 / BVR32](#) фірми [Delanco Spry](#); [Chameleon](#) фірми [Catalina Research](#); [Dime / Dime-II](#) фірми [Nallatech](#). Розглянуті системи призначені для спільного проєктування програмного та апаратного забезпечення. Вони містять апаратну частину (Реконфігуровані плати) та комплекс інструментальних програмних засобів, необхідних для верифікації проєктів методом моделювання. Інтеграція реконфігурованої плати та інструментальних засобів проєктування створює зручну платформу для налагодження і доопрацювання апаратних і програмних засобів. Використання таких платформ робить можливим верифікацію проєктів в реальному масштабі часу за рахунок об'єднання розроблених апаратних засобів системи з зовнішнім середовищем. Використовуючи замовні спеціалізовані плати розширення, можна додавати як цифрові (збільшення логічної складності проєкту), так і аналогові компоненти до системи (розширення функціональних можливостей). Більшість систем розглянутого класу підтримують широкий діапазон мов через доступні інтерфейси і забезпечують зв'язок між основними засобами розробки, такими, як [VHDL](#) для [FPGA](#) і [C++](#) для [DSP](#)-процесорів з інструментальними засобами високого рівня, [System C](#), [System Generator](#) для [MATLAB](#), безпосередньо [MATLAB](#) і [Handel C Celoxica](#). Їх можливості тягнуться до повної інтеграції з інструментальними засобами високорівневого проєктування на кристалах ПЛІС, протоколів промислового стандарту і прикладного програмного забезпечення користувача. У таблиці 3.2 наведені основні властивості систем [HOT](#), [System Explorer](#), [FUSE](#) і [Xtreme DSP](#), що мають як загальні, так і відмінні ознаки, що відносяться до можливості модулів розширення, інтерфейсу, наявності програмованих матриць зв'язку, програмного забезпечення. У таблиці також наведені області застосування зазначених систем. Таблиця 3.2 Властивості систем верифікації Тип системи (фірма-виробник) Модулі розширення Інтерфейс Програмовані матриці зв'язків Програмне забезпечення Область застосування [HOT \(VCC\)](#) + [PCI - C Reconfigurable Computing](#), модифікація проєктів [FPGA](#) через Інтернет [System Explorer \(Aptix\)](#) + [PCI](#), [LVDS\\* FPCB](#) + [EPIC Logic AggreGATER](#), [Design Pilot](#), [Explorer 2000 DSP](#), [Multimedia](#), [Network System](#), [SoC FUSE \(Nallatech\)](#) + [PCI](#), [Compact PCI - C/C++](#), [Java](#), [MatLab](#), [DIME Script Reconfigurable Computing Xtreme DSP \(Xilinx + Nallatech\)](#) - [PCI](#), [USB](#) - [MatLab & Simulink](#), [System Generator](#), [ChipScopePro DSP](#), [Multimedia](#) Система [HOT](#) як модулі розширення містить сопроцессорные плати з вбудованим контролером шини [PCI](#), а також інструментальні засоби для налагодження проєкту шляхом безпосередньої модифікації файлу конфігурації. Налагодження проєкту проводиться в реальному масштабі часу під управлінням програм, що використовують мову

Цитування: 0,01%

id: 55

"C".

Система застосовується в [Reconfigurable Computing](#), а також при модифікації проєктів [FPGA](#) через мережу Інтернет. [System Explorer](#) в якості апаратних засобів містить модулі, що представляють собою кристали ПЛІС, плати розширення пам'яті, АЦП, ЦАП. В кристали ПЛІС проводиться завантаження [IP](#)-модулів, що виконують функції обробки даних, цифрової обробки сигналів і т.д. Зв'язок верифікованих модулів з робочою станцією ([host](#)-комп'ютером) здійснюється через [PCI](#)-інтерфейс і інтерфейсний кабель типу [LVDS](#) з високою пропускну здатністю. З'єднання модулів між собою проводиться за допомогою



програмованих компонент [FPCB](#) і [EPIC](#). Програмне забезпечення містить ряд компонентів, до яких відносяться [Logic AggreGATEr](#), [Design Pilot](#), [Explorer](#) 2000. Система використовується для верифікації проєктів, що розробляються в якості [SoC](#) для [DSP](#), [Multimedia](#), [Network System](#). Система [FUSE](#) містить материнські плати з модулями розширення, на які встановлюються кристали ПЛІС. Зв'язок апаратури з [host](#)-комп'ютером здійснюється через інтерфейси [PCI](#) або [Compact PCI](#). Система підтримує широкий діапазон мов через доступні інтерфейси [API](#), включаючи [C / C ++](#), [Java](#), [MATLAB](#) і [DIME script](#). Використовується в багатьох додатках, пов'язаних з [Reconfigurable Computing](#). Система [Xtreme DSP](#) містить платформу розробки апаратних засобів і програмні засоби [FUSE](#) фірми [Nallatech](#). Інтерфейс з платформою апаратних засобів реалізується через шини [PCI](#) або [USB](#). До складу програмних засобів входять інструментальні засоби [MATLAB](#) / [SIMULINK](#) фірми [MathWorks](#), [System Generator](#) для [DSP](#), а також пакети програмного забезпечення, що забезпечують розробку і верифікацію проєктів в кристалах ПЛІС. Система використовується для верифікації проєктів в області [DSP](#) і [Multimedia](#). Найбільш потужною, серед розглянутих, представляється система [XtremeDSP Development Kit](#) для високоефективної розробки [DSP](#), що містить засоби верифікації - пакет [ChipScope Pro](#), розроблений фірмою [Xilinx](#). Система [System Explorer](#) фірми [Aptix](#) включає до свого складу потужні програмовані засоби комутації, реалізовані відповідно у вигляді програмованої користувачем схемної плати [FPCB](#) і програмованого користувачем компонента комутації [FPIC](#). Виконаний аналіз дозволяє зробити висновок, що в даний час не існує єдиної системи для верифікації проєктів, що реалізуються для різних областей застосування. Так, якщо система [System Explorer](#) фірми [Aptix](#) підтримує широкий спектр проєктів в таких областях, як засоби цифрової обробки даних, мультимедіа і т.д., то система [XtremeDSP](#), представлена фірмами [Xilinx](#) і [Nallatech](#), є більш спеціалізованою. Ця система дозволяє враховувати як різні аспекти предметної області (цифрова обробка сигналів), так і можливості архітектури кристалів, які використовуються для реалізації проєктів. До переваг системи [ChipScope Pro](#) слід віднести простоту роботи і можливість формування в процесі роботи досить широкої номенклатури ядер, розроблених для використовуваних кристалів.

3.3. Висновки до розділу

Виконано аналіз особливостей знання-орієнтованих інформаційних систем. На його основі запропоновано загальний підхід до побудови онтологічної бази знань, що функціонує в складі онтологізованого інформаційної системи, який названий системно-онтологічним. Компонентами онтологічної бази знань є онтологія об'єктів, онтологія процесів і онтологія завдань, для яких розроблені схеми моделей і методика проєктування. СОП і відповідна технологія його реалізації припускають аналіз ПДС, витяг, уявлення і обробку предметних знань. Виконано аналіз принципів побудови і функціонування сучасних систем верифікації реконфігурованих пристроїв на основі програмованих логічних інтегральних схем (ПЛІС) таких відомих фірм, як [Virtual Computer Corporation \(HOT\)](#), [Nallatech \(FUSE\)](#), [Aptix \(System Explorer\)](#), [Xilinx \(XtremeDSP Development Kit\)](#). Дані системи, реалізуючи інтеграцію конфігурується плати та інструментальних засобів проєктування, є засобом для оцінки, настройки та макетування проєктів, які використовують кристали ПЛІС інші компоненти в якості обчислювальних елементів системи.

ВИСНОВКИ

У магістерській роботі були розглянуті варіанти використання ПЛІС - технологій для автоматизованої побудови онтологій. В процесі виконання магістерської роботи були проведені дослідження в області архітектурно - структурної організації автоматизованої системи на базі ПЛІС. Відповідно до проведеним дослідженням були вирішені наступні завдання:

Виконано аналіз принципів побудови і функціонування сучасних систем верифікації реконфігурованих пристроїв на основі програмованих логічних інтегральних схем (ПЛІС) таких відомих фірм, як [Virtual Computer Corporation \(HOT\)](#), [Nallatech \(FUSE\)](#), [Aptix \(System Explorer\)](#), [Xilinx \(XtremeDSP Development Kit\)](#). Дані системи, реалізуючи інтеграцію конфігурується плати та інструментальних засобів проєктування, є засобом для оцінки, настройки та макетування проєктів, які використовують кристали ПЛІС та інші компоненти в якості обчислювальних елементів системи. Досліджено стан мікроелектронної промисловості в області створення систем на одному кристалі. Виділена класифікація цифрових інтегральних мікросхем і систем на одному кристалі на їх основі. Досліджено стан в області сучасних наскрізних САПР використовуваних при проєктуванні цифрових систем на одному кристалі. В ході проведених досліджень були виявлені сімейства ПЛІС, що володіють найбільш розвиненими функціональними структурами, перспективні для застосування в нових пристроях. З урахуванням проведених досліджень, описаних в першому розділі, на основі стандартного маршруту проєктування обчислювальних пристроїв була запропонована методика розробки складних цифрових ядер




багаторазового використання. Розроблено модифікований маршрут проектування складних ядер багаторазового використання. Виділено загальні правила проектування складних ядер і правила, яких необхідно дотримуватися при проектуванні ядер в базисі ПЛИС. Визначено основні критерії ефективності проектування ядер. Виконано аналіз особливостей знання-орієнтованих інформаційних систем. На його основі запропоновано загальний підхід до побудови онтологічної бази знань, що функціонує в складі онтологізованої інформаційної системи, який названий системно-онтологічним. Компонентами онтологічної бази знань є онтологія об'єктів, онтологія процесів і онтологія завдань, для яких розроблені схеми моделей і методика проектування. СОП і відповідна технологія його реалізації припускають аналіз ПДО, витяг, уявлення і обробку предметних знань. СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ Андон Ф.И., Яшунин Л.Е., Резниченко В.И. Логические модели интеллектуальн<sub>ых</sub> информационн<sub>ых</sub> систем. - К.: Наук. думка, 1999. - 397 с. Андреев А.М., Березкин Д.В., Симаков К.В. Особенности проектирования модели и онтологии предметной области для поиска противоречий в правов<sub>ых</sub> электронн<sub>ых</sub> библиотеках. - Доступно на [www.inteltec.ru/publish/articles/textan/RCDL2004.shtml](http://www.inteltec.ru/publish/articles/textan/RCDL2004.shtml). Бухтеев А. Метод<sub>ы</sub> и средства проектирования систем на кристалле. - [CHIPINFO](http://CHIPINFO), 09 марта 2005г

 **Обнаружен Плагиат: 0,1%** <http://dspace.nbuu.gov.ua/bitstream/...> + 2 ресурсов! id: 56

Гаврилова Т.А., Хорошевский В.Ф. Баз<sub>ы</sub> знаний интеллектуальн<sub>ых</sub> систем. - СПб.: Питер, 2001. - 384 с.

Дейв Криста, Тони Джонсон Методология в<sub>ы</sub>сокоуровневого проектирования устройств на базе [FPGA](http://chipnews.gaw.ru/html.cgi/archiv_i/99_03/stat-38.htm). [http://chipnews.gaw.ru/html.cgi/archiv\\_i/99\\_03/stat-38.htm](http://chipnews.gaw.ru/html.cgi/archiv_i/99_03/stat-38.htm) Дж. Ф. Уэйкерли Проектирование цифров<sub>ых</sub> устройств. В 2 томах. - М. Постмаркет, 2002. Добров Б.В., Лукашевич Н.В., Невзорова О.А., Федун Б.Е. Метод<sub>ы</sub> и средства автоматизированного проектирования прикладной онтологии // Изв. РАН. Теория и систем<sub>ы</sub> управления. М.: 2004. - № 2. - С.58-68. Згуровский М.З., Панкратова Н.Д. Системн<sub>ый</sub> анализ: проблем<sub>ы</sub>, методология, приложения. - К.: Наук. думка, 2005. - 744 с. Зотов В. [pBlaze IDE](http://pblaze.com) - интегрированная среда разработки и отладки программного обеспечения встраиваем<sub>ых</sub> 8-разрядн<sub>ых</sub> микропроцессорн<sub>ых</sub> систем, реализуем<sub>ых</sub> на основе ПЛИС фирм<sub>ы</sub> [Xilinx](http://www.xilinx.com). Компонент<sub>ы</sub> и технологии. 2006. № 3, 4. Зотов В. Директив<sub>ы</sub> и сообщения об ошибках интегрированной сред<sub>ы</sub> разработки и отладки программного обеспечения встраиваем<sub>ых</sub> систем [pBlaze IDE](http://pblaze.com). Компонент<sub>ы</sub> и технологии. 2006. № 5. Зотов В. Проектирование цифров<sub>ых</sub> устройств на основе ПЛИС фирм<sub>ы</sub> [Xilinx](http://www.xilinx.com) в САПР [WebPack ISE](http://www.webpack.com). М.: Горячая линия - Телеком, 2003. Зотов В. Разработка и отладка программного обеспечения встраиваем<sub>ых</sub> 8-разрядн<sub>ых</sub> микропроцессорн<sub>ых</sub> систем на основе ПЛИС фирм<sub>ы</sub> [Xilinx](http://www.xilinx.com) в среде [pBlaze IDE](http://pblaze.com). Компонент<sub>ы</sub> и технологии. 2006. № 6. Зотов В.Ю. Проектирование встраиваем<sub>ых</sub> микропроцессорн<sub>ых</sub> систем на основе ПЛИС фирм<sub>ы</sub> [Xilinx](http://www.xilinx.com). М.: Горячая линия - Телеком. 2006. Капитанов В., Мистюков В., Довгань С. Контроллер [PCI](http://www.pci.com) интерфейса на ПЛИС фирм<sub>ы</sub> [Xilinx](http://www.xilinx.com). - [http://chipnews.gaw.ru/html.cgi/archiv/00\\_02/stat-14.htm](http://chipnews.gaw.ru/html.cgi/archiv/00_02/stat-14.htm) Клещёв А.С., Артемьева И.Л. Отношения между онтологиями предметн<sub>ых</sub> областей. Ч.1. // Информационн<sub>ый</sub> анализ, Выпуск 1, С. 2, 2002. - С.4-9. Кривченко И. Системная интеграция в микроэлектронике — [FPSLICM](http://www.fpsli.com). -, [http://chipnews.gaw.ru/html.cgi/archiv/00\\_03/stat-4.htm](http://chipnews.gaw.ru/html.cgi/archiv/00_03/stat-4.htm) Кузелин М.О., Кн<sub>ы</sub>шев Д.А., Зотов В.Ю. Современн<sub>ые</sub> семейства ПЛИС фирм<sub>ы</sub> [Xilinx](http://www.xilinx.com). Справочное пособие. М.: Горячая линия - Телеком, 2004. Куликов К.В. Основн<sub>ые</sub> проблем<sub>ы</sub> проектирования систем на одном кристалле. - Нов<sub>ые</sub> методологии проектирования изделий микроэлектроники ([New design methodologies](http://www.newdesignmethodologies.com)): Материал<sub>ы</sub> МНТК., г.Владимир. 10-11 декабря 2004г. - ВлГУ, 2004. - С. 35-37. Ланцов В.Н., Морозов М.А. Проектирование систем на кристалле ([System-on-a-Chip](http://www.system-on-a-chip.com)). - Владимир, ВлГУ, 1999 Лохов А., Рабоволук А., Средства проектирования [FPGA](http://www.fpga.com) компании [Mentor Graphics](http://www.mentor.com). - Электроника: Наука, Технология, Бизнес 4/2004 Лохов А., Средства проектирования СБИС компании [Mentor Graphics](http://www.mentor.com). - Электроника: Наука, Технология, Бизнес 7/2003 Мальцев П.П., Гарбузов Н.И., Шарапов А.П., Кн<sub>ы</sub>шев Д.А. Программируем<sub>ые</sub> логические ИМС на КМОП-структурах и их применение. - М.: Энергоатомиздат. - 1998. Марка Д. А., МакГоуэн К. Методология структурного анализа и проектирования. - М.:

 **Цитирования: 0,01%** id: 57

"МетаТехнология",

1993. - 239 с. Мистюков Владимир, Володин Павел, Капитанов Владимир Однокристалльная реализация алгоритма БПФ на ПЛИС фирм<sub>ы</sub> [Xilinx](http://www.xilinx.com). - Компонент<sub>ы</sub> и технологии, № 10, 2000 Невзорова О.А. Онтолингвистические систем<sub>ы</sub>: методологические основ<sub>ы</sub> построения // Научная сессия МИФИ-2007. Сборник научн<sub>ых</sub> трудов. Том 3. Интеллектуальн<sub>ые</sub> систем<sub>ы</sub> и

технологии. М., 2007. - С. 84-85. Палагин А.В., Яковлев Ю.С. Системная интеграция средств компьютерной техники. - Винница:

” Цитирования: **0,01%** id: **58**

«УНІВЕР-СУМ-Вінниця»,

2005. - 680 с.

Обнаружен Плагиат: **0,06%** <http://dspace.nbuv.gov.ua/handle/123...> id: **59**

Палагін О.В., Петренко М.Г. Модель категоріального рівня мовно-онтологічної картини світу

// Математичні машини і системи. - 2006. - №3. - С. 91-104. Попович А. ПЛИС [ACTEL](#) - платформа для

” Цитирования: **0,02%** id: **60**

«Систем на кристалле»

бортовой аппаратур<sup>ы</sup>. - Электроника: Наука, Технология, Бизнес, 2004 №4 Сергиенко [А.М.](#) [VHDL](#) для проектирования [вычислительных](#) устройств. - К ЧП

” Цитирования: **0,01%** id: **61**

«Корнейчук»,

ООО «ТИД «ДС», 2003 — 208 с Стешенко В.Б, Школа разработки аппаратур<sup>ы</sup> цифровой обработки сигналов на ПЛИС. Занятие 1. Обзор [элементной баз<sup>ы</sup>](#), - [Chip News](#). 1999. № 8. С. 2-6.

Заявление об ограничении ответственности:

Этот отчет должен быть правильно истолкован и проанализирован квалифицированным специалистом, который несет ответственность за оценку!

Любая информация, представленная в этом отчете, не является окончательной и подлежит ручному просмотру и анализу. Пожалуйста, следуйте инструкциям: [Рекомендации по оценке](#)

Детектор Плагиата - Ваше право на оригинальность! ☐ SkyLine LLC