Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет __Комп'ютерної інженерії та управління_____
(повна назва)

Кафедра Автоматизації проектування обчислювальної техніки_____
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти ____другий (магістерський)_____
(рівень вищої освіти)

Методи апаратно-програмної верифікації моделей керуючих автоматів в системах на кристалі
(тема)

Виконав: студент 2 курсу, групи СКСм-22-1

_____Кур'янов А.І._____
(прізвище, ініціали)

Спеціальність *123 _Комп'ютерна інженерія*

Тип програми ____*освітньо-професійна*_____
(освітньо-професійна або освітньо-наукова)

Освітня програма_____

*Спеціалізовані комп'ютерні системи*_____
(повна назва освітньої прграми)

Керівник роботи ___доц.Шкіль О.С.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Чумаченко С.В.
(підпис) (прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Факультет_____Комп'ютерної інженерії та управління _____

Кафедра_____Автоматизації проектування обчислювальної техніки_____

Рівень вищої освіти____другий (магістерський)_____

Спеціальність_____123_ Комп'ютерна інженерія _____
(шифр і назва)

Тип програми _____Освітньо-професійна_____
(освітньо-професійна або освітньо-наукова)

Освітня програма _____Спеціалізовані комп'ютерні системи_____
(повна назва

ЗАТВЕРДЖУЮ:

Зав. кафедри ___  ___
(підпис)

«__02__»___09____ 2023 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові_____Кур'янову Артему Ігоровичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи (проекту)_ Методи апаратно-програмної верифікації моделей керуючих автоматів в системах на кристалі_____
Methods of Hardware and Software Verification of Models of Control FSM in Systems on Chip_____

затверджена наказом по університету від «__03_»__11__2023р. № _1282 Ст_____

2. Термін подання студентом роботи до екзаменаційної комісії _____15.01.2023_____

3. Вихідні дані до роботи (проекту)_____
Налагджувальна плата ZebBoard з кристалом ZYNQ-7000 фирмы Xilinx Inc._
Мова програмування С_____
Мови опису апаратури VHDL, Verilog_____
САПР XILINX ISE, Vivado, Vitis_____
_____

Перелік питань, що потрібно опрацювати у роботі _____
Методи проектування систем на кристалі (SoC)_____
Кіберфізичні системи на платформі SoC_____
Взаємодія процесорного ядра та ПЛІС у складі SoC_____
Методи діагностування SoC_____
Верифікація моделей кінцевих автоматів SoC у реальному часі_____

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 21 слайд _____
_____
_____
_____

6. Консультанти розділів роботи (проекту)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

7. Дата видачі завдання 02.09.2023_____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи (проекту)	Термін виконання етапів роботи	Примітка
1	Видача теми проекту, узгодження і затвердження теми	02.09.2023-08.09.2023	
2	Аналіз проблемної галузі, постановка задачі, вибір інструментальних засобів	09.09. 2023-15.09. 2023	
3	Аналіз технологій проектування та діагностування систем на кристалі	16.09.2023-29.09.2023	
4	Вибір налагоджувальної плати та інструментарію для реалізації проекту	30.09. 2023-20.10.2023	
5	Реалізація проекту керуючого автомату в SoC	20.10. 2023-20.11. 2023	
6	Фізична реалізація проекту в налагоджувальній платі	20.11. 2023-10.12. 2023	
7	Проведення ДЕ	10.12. 2023-25.12. 2023	
8	Оформлення пояснювальної записки	25.12. 2023-12.01. 2024	
9	Захист проекту	12.01. 2024-25.01. 2024	

Студент _____
(підпис)

Керівник роботи (проекту) _____ ___доц.Шкіль О.С_____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка містить 79 сторінок, 27 рисунків, та 12 джерел за переліком посилань.

КІНЦЕВИЙ АВТОМАТ, ГРАФ ПЕРЕХОДІВ, САПР, МОВА ОПИСУ АПАРАТУРИ, МОВА ПРОГРАМУВАННЯ, СИСТЕМА НА КРИСТАЛІ

В кваліфікаційній роботі розглянуті питання побудови системи самодіагностики кіберфізичної системи логічного управління, реалізованої на технологічній платформі системи на кристалі. В якості технологічної платформи обрано програмовану систему на кристалі FPGA сімейства ZYNQ-7000 фірми Xilinx Inc. В якості налагоджувальної плати використана недорога плата ZedBoard, яка оснащена пристроєм XC7Z020 Zynq. Практична реалізація виконана на базі стеку інструментальних засобів САПР Vivado/Vitis/Vitis HLS з використанням мови програмування С.

Розроблені методи апробовані на моделі пристрою керування тепловентилятором. При реалізації на платформі ZYNQ-7000 керуючий автомат реалізований в блоку PL, а операційний автомат в блоку PS. При організації процесу самодіагностики реалізується неруйнівний діагностичний експеримент шляхом обходу всіх дуг графа переходів починаючи з початкової вершини. В якості тестера при цьому виступає операційний автомат, еталонні логічні та часові значення зберігаються пам’яті блоку PS. Візуальне спостереження за виконанням ДЕ здійснюється на панелі світлодіодів плати ZedBoard.

Наукова новизна роботи полягає у подальшому розвитку методів самодіагностики автоматних пристроїв логічного керування, реалізованих на технологічній платформі систем на кристалі.

ABSTRACT

The explanatory note contains: 79 pages, 27 figures, 12 sources according to the list of links.

FINITE STATE MACHINE, STATE DIAGRAM, CAD, HARDWARE DESCRIPTION LANGUAGE, PROGRAMMING LANGUAGE, SYSTEM ON CHIP

In the qualification work, the construction issues of self-diagnosis system of a cyber-physical logical control system, implemented on the technological platform of a system-on-chip, are considered. A programmable system-on-chip an FPGA of the ZYNQ-7000 family of Xilinx Inc. was chosen as a technological platform. An inexpensive ZedBoard board equipped with an XC7Z020 Zynq device was used as a debug board. The practical implementation is based on the stack of CAD's tools Vivado/Vitis/Vitis HLS using programming language C.

The developed methods were tested on a model of the fan heater control device. During implementation on the ZYNQ-7000 platform, the control FSM is implemented in the Programmable Logic (PL) block, and the operating FSM in the Processing System (PS) block. During organization of the self-diagnosis process, a non-destructive diagnostic experiment is implemented by bypassing all arcs of the state diagram starting from the initial node. In this case, the operating FSM acts as a tester, the reference logic and time values are stored in the memory of the PS block. Visual observation of the diagnostic experiment execution is carried out on the LED panel of the ZedBoard board.

The scientific novelty of the work consists in the further development of self-diagnosis methods of automatic logic control devices, implemented on the technological platform of system-on-chip.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ВбС – вбудовані системи;

ГСА – граф-схеми алгоритмів;

ДЕ – діагностичний експеримент;

ІУС – інформаційно-управляючі системи;

НВІС – надвеликі інтегральні схеми;

КА – керуючий автомат;

ОА – операційний автомат;

ОС – обчислювальні системи;

ПЗ – програмне забезпечення;

ПЛІС – програмовані логічні інтегральні схеми;

САПР – система автоматизованого проектування;

СЛУ – системи логічного управління;

ЦП – цифровий пристрій;

AMBA – Advanced Microcontroller Bus Architecture (стандартна специфікація міжз'єднань на кристалі);

ARM – Advanced RISC Machine (32-бітна процесорна архітектура);

ASIC – Application-specific integrated circuit (інтегральна схема для специфічного застосування);

ASIP – Application-specific instruction-set processor (проблемно-орієнтований процесор, заказна мікросхема);

AXI Advanced eXtensible Interface;

CPS – Cyber-Physical System (кіберфізичні системи);

ESL – Electronic System Level (технологія проектування SoC);

FPGA – field-programmable gate array (різновид ПЛІС);

FSM – finite state machine (кінцевий автомат);

HLS – High-Level Synthesis (вісокорівневий синтез);

HW – hardware (апаратне забезпечення);

HDL – hardware description language (мова опису апаратури);

IDE – Integrated Development Environment (интегрированная среда разработки);

IP-core – intellectual property core (IP-ядра, готові блоки для проектування ПЛІС);

PS – processing system (процесорний блок SoC);

PL – programmable logic (блок програмовованої логіки SoC);

RAM – Random Access Memory (оперативна пам’ять);

ROM – Read Only Memory (постійне запам’ятовуючий пристрій);

RTL – Register Transfer Level (рівень регістрових передач);

RTOS – Real-time operating system (операційна система реального часу);

SoC – System of Chip (система на кристалі);

SW – software (програмне забезпечення);

UUT – Unit Under Test (об’єкт діагностування);

VHDL – very high speed integrated circuits HDL (одна з мов опису апаратури);

VLIW – very long instruction word (архітектура процесорів з декількома обчислювальними пристроями);

VLSI – very-large-scale integration (схеми надвеликого рівня інтеграції).

ВСТУП

Вбудовані цифрові пристрої та системи за останні 50 років зазнали значних змін не тільки в технологічному, а й в архітектурному сенсі. Якщо в 60-70 роки минулого століття вбудованою цифровою системою прийнято було називати спеціалізовані пристрої управління, спроектовані на логічних інтегральних схемах малого та середнього ступеня інтеграції, то сучасне розуміння вбудованої системи визначає її у вигляді замовної надвеликі інтегральні схеми (НВІС), що по своїй суті представляє спеціалізовану обчислювальну систему, технологічно виготовлену у вигляді системи на кристалі (SoC – System of Chip).

Проектування сучасних систем, що вбудовуються, засноване на застосуванні високотехнологічних САПР цифрових пристроїв, що вимагає від розробників глибоких знань не тільки цифрової схемотехніки та архітектур обчислювальних систем, але й знання методів синтезу спеціалізованих пристроїв з мікропрограм ним та логічним управлінням, знання високорівневих мов проектування та методів контролепридатного проектування. Процес проектування сучасних вбудованих цифрових пристроїв і систем – це процес створення власних та використання стандартних цифрових компонентів інтелектуальної власності, які є не лише схемотехнічними описами, але по суті є повноцінними проектними документаціями з функціонального та параметричного моделювання, верифікації та виготовлення із застосуванням конкретних технологій. Масове поширення мов проектування цифрових пристроїв, формування світового ринку компонентів інтелектуальної власності визначило проблему що реконфігуруються, але й визначив нові завдання, такі як забезпечення самодіагностики вбудованих SoC. Стрімкий прогрес у технологіях реалізації програмованих логічних інтегральних схем не тільки відновив інтерес інженерів і науковців до обчислень, що реконфігуруються, але й визначив нові завдання, такі як забезпечення самодіагностики вбудованих SoC.

Таким чином, розробка методів верифікації моделей цифрових пристроїв на технологічній платформі SoC на різних стадіях циклу проектування є актуальною та технічно значущою задачею.

Об’єкт дослідження у роботі – процедури автоматизованого проектування та діагностування цифрових пристроїв на технологічній платформі SoC.

Предмет дослідження – моделі, методи та процедури проектування та самодіагностики автоматних моделей пристроїв логічного керування, реалізованих в SoC.

Мета роботи – розробка моделей та процедур проектування,верифікації та самотестування в циклі автоматизованого проектування автоматних систем логічного управління на технологічній платформі SoC.

Для досягнення поставленої мети необхідно вирішити такі задачі:

– розглянути процедури взаємодії процесорного ядра з програмованою логікою у складі системи на кристалі;

– провести аналіз існуючих методик проведення верифікації на різних етапах автоматизованого проектування цифрових систем;

– провести подальший розвиток процедур проектування та тестування програмно-апаратних систем на основі SoC;

– провести подальший розвиток процедур автоматизованого проектування, верифікації та діагностування кіберфізичних систем логічного управління з використанням мов програмування;

– реалізувати процедури апаратного самотестування керуючих автоматів на технологічній платформі SoC.

1 ТЕХНОЛОГІЇ ПРОЕКТУВАННЯ ТА ДІАГНОСТУВАННЯ СИСТЕМ НА КРИСТАЛІ

1.1 Особливості побудови кіберфізичних систем на платформі SoC

Спочатку, електронні обчислювальні машини створювалися як інструмент виконання громіздких і трудомістких обчислень. Однак з часом сфера застосування ЕОМ розширилася на багато інших сфер людської діяльності, що піддаються формалізації та алгоритмічному опису. У тому числі, в 60-ті роки XX століття з'явилися так звані інформаційно-управляючі системи (ІУС), функцією яких стало управління реальними об'єктами.

Наприкінці 70-х, завдяки появі інтегральних схем і мікропроцесорів, стало можливим наближати ІУС безпосередньо до об'єктів управління або навіть вбудовувати їх у ці об'єкти. Так з'явилося нове поняття – вбудовані системи (ВбС). Бурхливий розвиток обчислювальної техніки протягом останніх десятиліть, що полягав, зокрема, в мініатюризації електронних пристроїв та в зростанні їх обчислювальних потужностей, призвело до збільшення кількості таких систем. Фактично, ВбС тепер є абсолютною більшістю серед усіх існуючих обчислювальних систем (ОС).

Найбільш загально ВбС можна визначити, як спеціалізовану ОС, яка в силу розв'язуваної задачі безпосередньо взаємодіє з фізичними об'єктами та процесами.

Наступним після появи ВбС етапом еволюції ІУС і їх природним розвитком стали просторово-розподілені вбудовані системи, в яких зв'язки між компонентами є слабкими, або, інакше кажучи, інтенсивність обміну даними в рамках одного обчислювального процесу значно вища за інтенсивність обміну даними між різними обчислювальними процесами. Такі системи стали доступними в міру здешевлення елементної бази, підвищення її інтеграції, і навіть зростання надійності обчислювальних приладів. Завдяки

перерахованим факторам з'явилася можливість встановлювати окремі, відносно незалежні ЕОМ у різні місця об'єкта управління, поєднуючи всі обчислювальні вузли в єдину контролерну мережу.

Діапазон реалізації ВбС дуже великий. До нього потрапляють і прості пристрої, які виконують будь-які елементарні функції, і найскладніші розподілені ієрархічні системи, у тому числі управляючі критично важливими об'єктами. ВбС знаходять широке застосування в побутовій електроніці, промисловій автоматиці, на транспорті, телекомунікаційних системах, медичному устаткуванні, у військовій та аерокосмічній техніці, в інших областях.

Сфера застосування ВбС постійно розширюється і в тому чи іншому вигляді ці системи найближчим часом проникнуть у всі сфери діяльності людини. При цьому всупереч широкому поширенню ВбС, основним об'єктом вивчення науки в комп'ютерній області тривалий час була обробка інформації в стандартних ОС загального призначення. Незважаючи на те, що ВбС використовувалися з 1970-х, протягом основної частини своєї історії вони розглядалися просто як маленькі комп'ютери.

Останні роки з'явилося розуміння, що важливі проблеми у вбудованих системах відбуваються через взаємодії з фізичними процесами, а не через наявність обмежених ресурсів. У 2006 році було введено термін «кіберфізичні системи» (CPS – Cyber-Physical System,), що передбачає інтеграцію обчислень із фізичними процесами. Поняття відображає очікуваний у найближчому майбутньому якісний перехід у сприйнятті ВбС та методів їх проектування. Суть CPS полягає в тому, що проектування об'єкта управління та системи управління для цього об'єкта повинні виконуватися в єдиному ключі, в єдиному комплексі, інструментальних засобів, що тісно взаємодіють. Крім того, концепція CPS, як правило, передбачає об'єднання в мережу з іншими пристроями, тісну взаємодію та обмін інформацією з цими пристроями.

Таким чином, з часом еволюція поняття «обчислювальні системи, що

вбудовуються» змінювалася наступним чином:

1) інформаційно-управляючі системи, 60-ті роки (ІУСта ІУК);

2) вбудовані обчислювальні системи (embedded systems) – ВбС (ES), кінець 70-х років;

3) розподілені вбудовані системи управління (networked embedded control systems/розподілені інформаційно-управляючі системи) – NECS/РІУС, кінець 90-х років;

4) Cyber Physical Systems – кіберфізичні системи, приблизно з 2006 р.

На рис.1.1 наведено приклад типової структури CPS. Вона складається з об'єкта управління, «фізичної» частини – тієї частини, яка не реалізується обчислювальною технікою чи цифровими мережами, і може включати механіку, біологічні чи хімічні процеси навіть людей-операторів. Управління цією частиною CPS здійснюють одна або кілька платформ, які складаються з датчиків, виконавчих механізмів, одного або кількох процесорів і, можливо, однієї або кількох операційних систем. Мережева структура забезпечує механізми взаємозв'язку платформ, утворюючи разом із ними «кібернетичну» частину CPS [1].

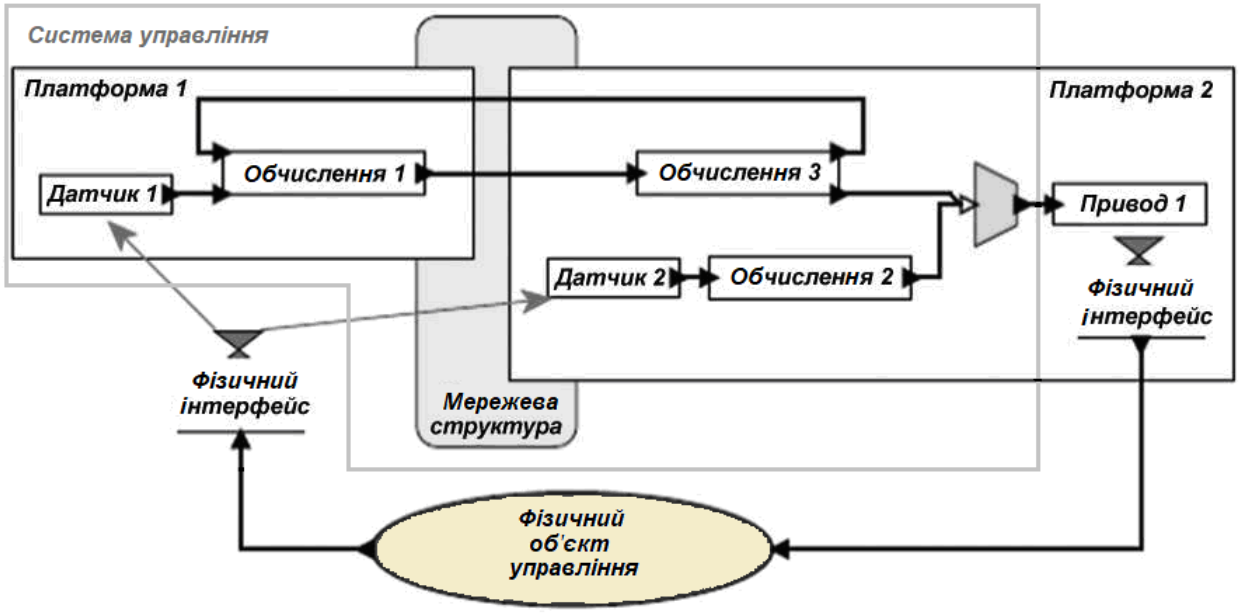

Рисунок 1.1 – Типова структура кіберфізичної системи.

Проектування ВбС спричиняє деякі певні проблеми, які є рідкісними в універсальних обчисленнях і звичайних настільних комп'ютерах. Наприклад, в універсальному програмному забезпеченні (ПЗ) час, який потрібно, щоб виконати завдання, є лише показником продуктивності, а не коректності написання ПЗ. Не є некоректним, якщо більше часових ресурсів витрачається на розв'язання задачі – це менш зручно.

У ВбС час, який потрібно, щоб виконати завдання, може бути критичним для правильного функціонування системи, адже фізичні процеси, що відбуваються в реальному світі, рідко є процедурними, і зволікання в обробці якихось подій для критичних областей може спричинити непоправні наслідки.

Принципова відмінність інформаційних систем від систем реального часу, до яких, як правило, належать ВбС, полягає в трактуванні параметра «реакція вхід-вихід»: «The right answer late is wrong» («Правильна відповідь пізніше, ніж треба – неправильна»).

Функції ВбС, як і будь-якої системи – чи то біологічний організм, людське суспільство або штучно створений з якоюсь метою об'єкт - можна звести до якоїсь основної базової задачі або набору таких. Система повинна встигати обробляти зміни значень своїх входах і виставляти актуальні значення управляючих виходів відповідно до закладеними алгоритмами, впливаючи цим на довкілля і підлеглий об'єкт (об'єкти) управління. Таким чином, здійснюється управління динамікою процесів у навколишньому середовищі за принципом зворотного зв'язку: за рахунок вимірювань стану контрольованих процесів та організації дій, що впливають на ці процеси через деякі виконавчі механізми.

Крім того, може бути необхідність одночасного виконання кількох дій, кожна з яких є важливою – адже процесів, які необхідно контролювати, часто буває більше одного. Виходячи з цього – паралелізм внутрішньо притаманний ВбС. Підхід, коли кожне завдання чекає своєї черги, і проводиться простий послідовний перебір по порядку, здебільшого

неприйнятний у алгоритмах управління.

Виходячи з цього, при проектуванні ВбС необхідно домагатися максимально швидкої та паралельної з погляду зовнішнього середовища поведінки ВС, незважаючи на послідовну природу більшості сучасних комп'ютерів. Бажано, щоб ВбС функціонувала так, ніби є повністю паралельною системою. Багато з технічних проблем при розробці та аналізі вбудованого програмного забезпечення мікропроцесорних систем якраз і відбуваються внаслідок потреби поєднувати послідовні обчислення з безперервними процесами, що одночасно змінюються у зовнішньому світі.

Але часові показники є єдиною відмінною здатністю ВбС. Найчастіше такі системи застосовуються у відповідальних галузях, що тягне за собою підвищені вимоги до надійності та всебічного захисту проектованих пристроїв. Якщо виходить з ладу бортовий комп'ютер автомобіля або літака, медичне обладнання, блок керування ліфтом, світлофором або яким-небудь верстатом – це може спричинити шкоду людям, навколишньому середовищу, чиємусь майну.

Тому для забезпечення достатньої надійності роботи пристроїв потрібно враховувати багато факторів: як природні впливи, так і наслідки людської діяльності. Причому останнє включають випадкові чинники і цілеспрямовані шкідливі впливу – вандалізм, спроби несанкціонованого доступу, злому пристроїв. Враховуючи, що кіберфізичний підхід до створення ВбС, що набирає чинності, передбачає в більшості випадків підключення до мережі інтернет, а також те, що поняття «інтернет речей», коли кожен пристрій підключається до глобальної мережі і стає доступним ззовні, вже стає реальністю – з'являється все більше мішеней для ймовірних атак ззовні та все гостріше постає проблема забезпечення безпеки.

Крім зовнішніх чинників, які впливають роботу системи вже під час її функціонування, обов'язково слід враховувати можливість виникнення помилок під час процесу створення продукту. Будь-який, навіть найдосвідченіший інженер, може припускатися помилок, і що складніший

виріб, то їх ймовірність більше. Це можуть бути баги в коді, неправильне трасування доріжок на платі, невірно підібрані електронні елементи і не в останню чергу концептуальні помилки всього проекту загалом. Бувають помилки в інструментальних засобах для розробки програмного та апаратного забезпечення і в бібліотеках, що поставляються виробниками. Крім того закуплені електронні компоненти можуть виявитися бракованими. Проблеми можуть виникати на етапі збирання пристроїв, при їх монтажі на об'єктах тощо. На рис.1.2 показаний взаємний вплив компонентів ВбС та зовнішніх факторів, які впливають на надійність системи в цілому.

Рисунок 1.2 – Взаємний вплив компонентів ВбС та зовнішніх факторів, що позначається на складності та надійності.

Таким чином процес створення ВбС, який зазвичай враховує спочатку

розробку, потім виготовлення, супроводжується процесами верифікації і тестування. Перші, дослідні зразки перевіряються на відповідність вимогам технічного завдання. Потім при налагодженому виробництві проводиться лише швидка перевірка працездатності нових виробів за деякими характерними ознаками.

Концептуально, як таке завдання забезпечення надійності і захищеності невіддільне і безпосередньо випливає із завдання забезпечення функціональності обчислювальної системи, тобто на те, що зазвичай прописується в технічному завданні явно. Грубо кажучи – система має працювати. Яким би не було її призначення, в чому б не виявлялася ця робота – управління якимось об'єктом, збирання інформації з датчиків, просто миготіння лампочками – система повинна штатно функціонувати у всьому допустимому діапазоні зовнішніх впливів, не тільки в ідеальних умовах. Якщо вдалося домогтися правильної поведінки системи та змусити її спрацювати один раз, це не означає, що отримано працездатний пристрій, що відповідає вимогам.

Також, якщо говорити про надійність, завжди варто пам'ятати: можливість безвідмовної роботи системи (якщо для працездатності всієї системи в цілому потрібна працездатність всіх складових, що найчастіше вірно для ВбС) дорівнює добутку можливостей безвідмовної роботи кожного її елемента. Це спричиняє два наслідки.

Перше те, що максимально надійне є максимально простим. Будь-яке нарощуванні функціональності здійснюється за рахунок збільшення обсягу коду, додавання нових елементів тощо. Одночасно збільшується ймовірність відмови.

Другий наслідок – те, що ВбС, що складаються з великої кількості тісно взаємодіючих «елементів» (якими можна виділити ПЗ, у тому числі операційні системи, бібліотеки та драйвера, інтегральні схеми, всі електронні елементи, що встановлюються на платах, із зв'язками між ними, датчики, виконавчі механізми, з'єднувальні кабелі тощо), апріорі мають великий

потенціал для виникнення помилок при проектуванні та відмов – вже в процесі роботи.

Як правило, досить складними є вже хоча б окремо ПЗ та апаратна частина, при їх комбінуванні – йде вибухове зростання складності. Навіть при повністю налагоджених окремо ПЗ та апаратурі, у їхній спільній роботі можуть виникати непередбачувані ефекти, викликані взаємодією цих базових елементів.

Так як ВбС, згідно з визначенням та призначенням, часто вбудовуються в об'єкти управління – на них найчастіше накладаються численні обмеження. Наприклад, як додаткові умови можуть бути задані максимальні розміри або навіть форма виконання конструкції пристрою, температурні режими роботи, максимально допустиме енергоспоживання і т.д. Зниження енергоспоживання взагалі є одним з найпоширеніших пунктів багатокритеріальної оптимізації, яка проводиться при проектуванні ВбС.

Крім цього, завжди стоїть завдання мінімізації вартості виробництва кінцевих пристроїв. Так як ВбС у більшості випадків є не стандартними, а замовними пристроями, що виробляються на замовлення, зниження вартості комплектуючих, технологій виготовлення, накладних витрат на проектування безпосередньо впливають на вартість і, отже, конкурентоспроможність кінцевих пристроїв, що виставляються на ринок або надаються замовнику.

Таким чином, розробка ВбС вимагає великих знань у різних галузях – розуміння обчислювальних систем, програмного забезпечення, мереж та фізичних процесів, уміння оцінювати їхню спільну динаміку. Щоб змусити ВбС працювати з фізичними процесами, потрібне технічно складне, низькорівневе проектування.

Система на кристалі (System on Chip, SoC) – система, побудована на єдиному кристалі, в якій інтегруються такі елементи, як процесор (процесори, у тому числі спеціалізовані), деякий обсяг пам'яті, ряд периферійних пристроїв та спеціалізованих обчислювальних блоків та їх з'єднання. Все вищезгадане складає оптимальний набір для деякого

заздалегідь відомого функціонала, наприклад, обробки та передачі відеоданих. Вираз "система на кристалі" не є, строго кажучи, терміном. Це поняття відбиває загальну тенденцію до підвищення рівня інтеграції рахунок інтеграції функцій [2].

SoC часто використовуються як компоненти ВбС і в граничних випадках фактично ними ж і є, для ВбС і SoC використовуються одні і тіж методи проектування. Реалізація у вигляді SoC виключає необхідність застосування численних окремих інтегральних схем та реалізації інтерфейсів зв'язку між ними. Від звичайних мікроконтролерів SoC відрізняється специфікою організації, спрямованої рішення конкретних завдань, тоді як мікроконтролер реалізує універсальний набір функцій. Також на відміну від мікроконтролерів є те, що в SoC зазвичай використовуються потужніші процесори, придатні для запуску великовагових операційних систем (Linux, Windows), які найчастіше вимагають для ефективної роботи великих банків зовнішньої пам'яті та розгорнуту периферію. У деяких реалізаціях мікросхеми SoC мають контакти як на нижній частині для з'єднання з платою, так і на верхній - для встановлення додаткових блоків пам'яті.

Типова система, що вбудовується, побудована на базі SoC, може містити такі набори інтерфейсів і контролерів, як:

- системна шина та контролери шин LPC/ISA, PCI, PCMCIA;
- контролери управління NOR/NAND Flash, SDRAM, SRAM, DDR;
- контролер Ethernet;
- послідовні інтерфейси UART, SPI/SSP/uWire, RS-232, RS-422/RS-485, CAN;
- бездротові інтерфейси WiFi/IEEE802.11, ZigBee, Bluetooth, IrDA;
- інтерфейси підтримки Flash-карт пам'яті: SD/MMC, CompactFlash, MemoryStick;
- контролер LCD STN/TFT/OLED;
- контролер матричної клавіатури;
- модулі бездротової передачі даних GSM/GPRS, CDMA;

- модулі прийому сигналів супутникових навігаційних систем GPS;
- апаратні підтримки плаваючої точки, шифрування, DRM тощо;
- аудіо- та відео-інтерфейси;
- джерела опорної частоти;
- регулятори напруги та стабілізатори живлення.

Переваги "систем на кристалі" перед класичними "системами на друкованій платі".

1. Мініатюризація. Як правило, пристрій, створений на базі SoC, складається з однієї (максимум 2-х) НВІС та обмеженого набору дискретних компонентів, які з технологічних причин не можуть бути інтегровані всередину ІС.

2. Зниження споживаної потужності. НВІС типу "система на кристалі" виготовляються за технологією "глибокого субмікрона" до 0.35 мкм і нижче, що дозволяє знизити напругу живлення і, як наслідок, суттєво зменшити споживану потужність.

3. Підвищення надійності. Об'єднання декількох компонентів (IP-блоків) на одній пластині кремнію дозволяє істотно зменшити кількість паяних сполук.

4. Зниження вартості великих партій. ВбС, виконаних з урахуванням SoC, при налагодженому виробництві є набагато дешевшими рішеннями, ніж звичайні системи на платах.

5. Спрощення монтажу. При використанні технології SoC потрібно встановлювати менше корпусів на плату, як правило, сам обчислювальний елемент.

Також, необхідно відзначити, що процесу проектування апаратури зазвичай супроводжує процес створення програмного забезпечення, драйверів, що дозволяють «оживити» створювану мікросхему. Крім того, продуктивність приладів класу SoC значною мірою залежить від ефективності взаємодії всіх вбудованих компонентів та від ефективності їх взаємодії із зовнішнім, щодо приладу, світом. Насамперед це з різницею у

швидкодії вбудованих компонентів, особливо організації інтерфейсів.

За всіх переваг технології проектування та налагодження SoC як єдиної складної системи є набагато більш трудомістким і складним процесом, ніж проектування її компонентів у вигляді окремих мікросхем. Виходячи з цього, однією з основних проблем при розробці SoC є боротьба зі складністю та труднощі аналізу процесів, що протікають всередині мікросхеми.

Крім цього, враховуючи високу вартість виправлення помилок у кінцевих реалізаціях, з одного боку, значну частку процесу проектування SoC завжди займає верифікація та всебічне тестування продукту, а з іншого – шукаються шляхи внесення змін до закінченого і вже випущеного з виробництва виробу. Останній варіант привабливий ще й тим, що може збільшити гнучкість пристрою, що дозволяє краще налаштовувати його під конкретну програму.

1.2 Технологічний маршрут проектування SoC

Ідеалізований процес проектування SoC можна розбити на вісім кроків:

- створення специфікації та моделювання;
- попередній аналіз перед поділом на апаратні та програмні частини;
- поділ на апаратні та програмні частини;
- аналіз, налагодження та усунення виявлених помилок після поділу;
- верифікація після поділу;
- реалізація апаратури;
- реалізація ПЗ;
- верифікація реалізації.

Перший крок – написання специфікацій та моделювання. Він є розробкою документів, що описують систему або концепцію проекту та його обмеження, а також їх трансляцію в різні моделі, що виконуються або декларуються. Специфікації та обмеження на цьому етапі пишуться звичайною мовою, що забезпечує деяку свободу трактування і не обмежує

якоюсь конкретною реалізацією моделі. Контроль складності проекту починається вже на рівні специфікації і полягає у розподілі системи на підсистеми та компоненти, навіть якщо вони не передбачаються для конкретних реалізацій [3].

Для ESL-проектування (Electronic System Level) неприйнятним є традиційний підхід, коли для розробки ПЗ потрібна досить стабільна апаратна платформа – апаратура та ПЗ проектуються паралельно для того, щоб уникнути непорозумінь на фінальному етапі інтеграції. Для збереження вимог до кінцевого продукту у процесі проектування використовується система відстеження вимог. В ідеалі модель ПЗ повинна бути створена першою, щоб аналіз до поділу на апаратну та програмну частини виявив вимоги ПЗ до апаратури. Типові мови для створення специфікацій: MATLAB M-Code, SystemC, SystemVerilog тощо [4].

Аналіз перед поділом на апаратні та програмні частини – процес дослідження спектра алгоритмічних компромісів. Його метою є знаходження деякої золотої середини між продуктивністю, займаним на кристалі обсягом, енергоспоживанням, складністю реалізації та часом, необхідним для доведення розробки до деякого остаточного виду, прийнятного для кінцевого користувача. У ході аналізу порівнюються витрати та передбачуваний позитивний ефект при виборі різних варіантів реалізації системи. Для їхньої оцінки можна використовувати різні сценарії та методи – статичний аналіз, платформно-орієнтоване проектування, динамічний аналіз, алгоритмічний аналіз тощо.

Статичний аналіз може використовуватися за відсутності специфікації, що виконується. Платформно-орієнтоване проектування – інтенсивне повторне використання існуючих елементів, які зазвичай представляють у вигляді чорних ящиків, які без шкоди функціональним вимогам надають кращу продуктивність, енергоспоживання або інші переваги порівняно з деякими стандартними аналогами. Щоправда, цей підхід має недолік – при його застосуванні команда розробників мимоволі починає скочуватися в

традиційне проектування. Динамічний аналіз дає можливість отримати і більш точну інформацію, яку можна отримати тільки з моделей, що виконуються. Симуляція цих моделей дозволяє оцінювати затримки та пропускні можливості обчислювальних трактів, вплив політик планування, обчислювальні та комунікаційні накладні витрати на обчислювальні операції. На додаток до аналізу інформації, одержуваної зі статичного та динамічного аналізу, потрібен розгляд моделей обчислень за допомогою алгоритмічного аналізу. Його мета – оцінка потреб у ресурсах, і навіть оцінка таких експлуатаційних параметрів системи, як завантаження вузлів, вимоги до каналів передачі, ймовірності виникнення помилкових бітів у потоках даних.

Поділ – процес, у ході якого виробляється вибір, які алгоритми (чи його частини), задані у специфікації, реалізовувати як запущеного на процесорах програмного забезпечення (також, цьому етапі вибираються типи процесорів), які – як чисто апаратних компонентів. Відбувається поділ алгоритмів між апаратними та програмними частинами. Процес поділу можна розглядати в наступному порядку: функціональна декомпозиція, архітектурна декомпозиція, поділ, поділ апаратної частини, поділ програмної частини, визначення частин, що реконфігуруються, і реалізація комунікацій.

У функціональній декомпозиції за відправну точку береться специфікація, настільки вільна від артефактів реалізації, наскільки це можливо, і дозволяє отримати уявлення про можливості розпаралелювання на рівні додатків. Ці вимоги можна задовольняти двома способами: використанням функціональної паралельної мови для створення виконуваних специфікацій або послідовного, але має підтримку інструментів автоматичного вилучення паралелізму. Приклад паралельної мови – Simulink від MathWorks. Застосування другого підходу, незважаючи на його велику привабливість, є досить складним завданням і не має універсальних рішень.

В архітектурному описі прийнято використовувати елементарні блоки, кожен із яких реалізує якусь функцію: зберігання даних, обчислення, зв'язки й з іншими блоками. Деякі з цих блоків – фіксовані чи конфігуровані

процесори, довільні чи стандартні апаратні модулі, шини, блоки пам'яті, операційні системи та сервіси, API, стеки протоколів. Вони використовуються для відображення функціонального опису загальну архітектуру, часто – платформу. У найпростіших випадках початкове відображення функцій на конкретні архітектурні елементи найчастіше відбиває кінцеве обране апаратне поділ. Проте, у складніших проектах не всі функції відображаються безпосередньо на якісь конкретні апаратні компоненти, а можуть відповідати, наприклад, об'єктам об'єктно-орієнтованого програмного забезпечення.

Після уточнення або відображення абстрактних моделей на архітектуру потрібно поділити апаратні та програмні компоненти. Апаратний поділ може бути реалізований у вигляді одного або кількох процесорів розподіленої системи.

Програмні модулі складаються не тільки з операційної системи, бібліотек та проміжного ПЗ (middleware), але і з додатків, які найчастіше беруть на себе основне функціональне навантаження.

Обчислювачі, що реконфігуруються, покликані заповнити прірву між апаратурою і програмним забезпеченням, дозволяючи домогтися компромісів у продуктивності, вартості та енергоспоживання.

Апаратні обчислювальні елементи можуть бути реконфігуровані одного разу, коли система випускається, раз на якусь кількість років, для усунення багів і поліпшень, або раз на кілька годин, щоб адаптуватися під поточне завдання.

Однією з найвідповідальніших завдань при поділі є реалізація комунікацій, оскільки від вибраних протоколів залежить безліч аспектів системи, таких, наприклад, як продуктивність та надійність. Існують два підходи до реалізації комунікаційних архітектур: використання шаблонних рішень (template instantiation) та синтез інтерфейсів. Перший спосіб полягає у підборі одного або кількох існуючих варіантів рішень під конкретні вимоги. Другий можливий тільки при виконанні наступних вимог: середня пропускна

спроможність передавальних та приймаючих інтерфейсів приблизно однакові.

Після поділу слідує обов'язковий і важливий етап аналізу та верифікації. Він складається із двох кроків. Перший полягає у реалізації алгоритмів, які призначаються до виконання на процесорах і написані однією з мов програмування, тобто. у тому компіляції для конкретних процесорних архітектур. Другий – у моделюванні на поведінковому рівні призначених для апаратної реалізації алгоритмів за допомогою однієї з мов опису апаратури. Після цих двох кроків йде перевірка відповідності отриманої моделі вимогам специфікації, в ході чого може з'ясуватися, що необхідно знову повернутися до етапу поділу для оптимізації системи, що розробляється.

Зазвичай застосовуються такі аналізи: функціональність, продуктивність, інтерфейси, енергоспоживання, займане місце, вартість і придатність для налагодження.

Функціональний аналіз, який зазвичай відноситься до функціональної верифікації, потрібен на цьому кроці проектування для розуміння вимог до розміру різних елементів зберігання. Початковий аналіз продуктивності проводиться для валідації розподілу та визначення розмірів апаратних та програмних елементів. Аналіз інтерфейсів дозволяє виявити рано, задовго до реалізації, підходить чи ні конкретний варіант комунікаційних інтерфейсів модуля. Аналіз енергоспоживання вигідний цьому етапі у разі, коли модель має властивості скоріш фізичної реалізації, ніж программной.Анализ займаного місця використовує метрики складності з метою оцінки вимог до апаратурі по місяцю. Аналіз вартості за допомогою евристик розраховує вартість проектування, виробництва, підтримки та повного життєвого циклу виробу. Нарешті, аналіз придатності для налагодження перевіряє складність та ризик функціональних помилок передбачуваної реалізації апаратури, щоб з'ясувати, які можливості контрольованості та спостереження вимагаються від зовнішніх інтерфейсів фізичної реалізації чіпа.

Верифікація після етапу поділу повинна виявити, чи збереглася

очікувана поведінка компонентів системи виробу, що розробляється в уточненій моделі, отриманої після поділу. Це перший крок верифікації, наступний це вже верифікація реалізації. В обох випадках поведінка моделі на найвищому рівні абстракції має бути порівняна з поведінкою моделі на нижньому. При цьому є деякі ефекти, які відсутні в грубіших моделях і з'являються тільки після деталізації.

Для вирішення цієї та інших проблем робиться три кроки: планування верифікації, реалізація оточення верифікації та аналіз результатів верифікації.

Планування верифікації – перерахування в документі, можливо, у форматі, придатному для читання будь-якими інструментами автоматизації, проблем, що підлягають верифікації та їх вирішенню.

Коли область інтересу для верифікації визначається, а план верифікації та функціональна специфікація для верифікаційного оточення написана, створюється саме верифікаційне оточення, зазвичай із використанням високорівневих мов верифікації (High-Level Verification Languages), можливо навіть аспектно-орієнтованих.

Аналіз коректності результатів має виявляти загальні джерела помилок для прогонів тестів. Пошук причин - другий з трьох кроків налагодження, що полягає в знаходженні багів, їх діагностиці та усуненні. Робота з знаходження багів виконується генерацією тестів та перевіркою аспектів динамічного оточення верифікації чи перевіркою моделі статичного оточення. Аналіз повноти покриття покликаний виявити, чому існують конкретні діри у покритті цільової області тестами.

Реалізація апаратури – процес створення моделей, які можуть бути синтезовані у моделі вентильного рівня. Апаратні моделі зазвичай описуються лише на рівні передач сигналів між регістрами (RTL-моделі), і навіть більш високому поведінковому рівні. До цих пір існує безліч виборів, що проводяться на цьому етапі переходу від поведінкового опису до RTL-рівня, таких як поділ ресурсів та додавання конвеєрів, які можуть вплинути

на продуктивність системи, тому деякий аналіз все одно завжди буде потрібно навіть для перевірки згенерованих RTL-рішень .

Існує п'ять основних варіантів реалізації апаратури, які доступні на даному етапі: процесори, що розширюються (extensible), DSP співпроцесори, спеціалізовані VLIW співпроцесори, ASIP і ASIC/FPGA.

Розширюваний процесор – це центральний процесор загального призначення, який може бути адаптований під конкретну програму.

DSP – процесор, оптимізований до виконання алгоритмів цифрової обробки сигналів.

Спеціалізований VLIW співпроцесор оптимізований у кількості та типі функціональних блоків для виконання конкретного алгоритму.

ASIP використовують спеціалізовані тракти обробки даних та кінцеві автомати для реалізації конкретного алгоритму.

При створенні апаратури можуть використовуватися повністю специфіковані IP блоки, що налаштовуються (зазвичай рахунок параметрів) RTL компоненти, і спеціалізована апаратура, що повністю створюється з нуля. IP блоки, придатні для повторного використання, просто додаються до проекту, часто з деякими змінами та доопрацюваннями.

Процеси проектування для ASIC та FPGA дуже схожі. В обох випадках результат – RTL код, який реалізує потік даних крізь функціональні модулі, де він контролюється мультиплексорами, які контролюються кінцевими автоматами.

Реалізація ПЗ надає людино-машинний інтерфейс, що дозволяє використовувати апаратну частину, і є найвищими елементами проекту. ПЗ пишеться для вже існуючої апаратури з метою задоволення всіх вимог до продукту, включаючи компенсацію апаратних багів та спроби виправити можливу нестачу продуктивності.

Крім оцінки обсягу обчислень, також важливо оцінювати обсяг пам'яті, що використовується, і її пропускну здатність. Кодування інструкцій (на кшталт RISC або CISC) безпосередньо впливає на розмір інструкцій та

програмної шини, обсяг займаної скомпільованим кодом пам'яті, енергоспоживання на вибір інструкцій. Крім того, вплив на продуктивність надають шаблони доступу до сучасних SDRAM, оптимізованих для послідовного доступу всередині блоків. Якщо є певний ступінь свободи під час реалізації шаблонів звернення до адресів операндів, їх бажано оптимізувати для мінімізації затримок.

Мови програмування та інструменти розробки, які використовуються при створенні програмного забезпечення в стилі ESL, схожі на звичайні, але з деякими особливостями. З продовжує бути найбільш широко використовуваною мовою, Embedded C++ бачиться перспективним варіантом, так як надає можливості об'єктно-орієнтованої мови без зниження продуктивності, характерної для повної версії C++. Проектування для верифікації ("Design-For-Verification, DFV"), що враховує простоту виявлення, діагностики та виправлення багів – має бути однією з першочергових явних вимог щодо створення моделі для реалізації проектних рішень.

1.3 Технології сумісного проектування для SoC

Якщо ще зовсім недавно схеми БІС і НВІС вже вміщали більшість ІС, що знаходяться на одній друкованій платі, то тепер один кристал SoC вміщає в собі кілька НВІС (таких як мікропроцесор, пам'ять, блоки вводу/виводу). Завдання, що постають перед розробником, що збирається використовувати SoC, зовні не відрізняються від завдань, які вже розглянуті для МПС і ПЛІС. Основна відмінність полягає в тому, що в одному кристалі має бути спроектована і апаратура, і програмне забезпечення.

Сучасна вимога зменшення часу, необхідного для випуску кінцевого продукту, змушує шукати шляхи скорочення часу виконання будь-яких окремих етапів проектування, так і усунення причин, що викликають необхідність ітераційних повернень до попередніх етапів проектування.

Традиційна процедура проектування, наведена на рис.1.3, характеризувалася суттєвим запізненням розробки програм щодо розробки апаратури.

1 – проектування HW
2 – побудова прототипів HW
3 – налаштування HW
4 – проектування SW
5 – побудова коду програми SW
6 – налаштування SW

Рисунок 1.3 – Традиційна процедура проектування SoC

Труднощі проектування SoC пов'язані як з необхідністю виконувати проекти в стислі терміни, так і з складністю систем, що постійно зростає. За оцінками, середній обсяг сучасних проектів перевищує 50 тис. вентилів і має постійну тенденцію подвоюватися кожні два роки, і в остаточній вартості проекту все більшу роль відіграють розроблені програмні та "зашиті" в ПЗУ апаратні ресурси. Очікується, що у найближчі кілька років проектами, що містять понад 100 тис. вентилів та МП-ядро (core-based), займатимуться понад 200 тис. розробників. Зростання складності сучасних проектів як щодо об'ємів HW (у кількості логічних вентилів), так і щодо об'ємів SW (у рядках програмного коду), робить проблему підтримки цього напрямку актуальною.

Як вже зазначалося вище, поєднання в одному кристалі програмного та апаратного виконання алгоритмів призводить не до арифметичного складання їх потенційних можливостей. Якщо першому етапі появи кристалів типу SoC основну увагу приділялося мобільності периферійної частини МПС, і це висувалося як основного гідності SoC, далі стало зрозуміло, що змінюється ставлення й інших складових МПС. Основним фрагментом, визначальним інтелектуальність, стає процесор, а вміст пам'яті, не периферія як така, а пам'ять її конфігурації. Процесор, а точніше його конкретна організація, перестає бути домінуючим чинником, визначальним

продуктивність системи. Поведінку чи структуру процесора у сучасних системах можна легко змінити чи просто поміняти процесор цілком. Перетрансляція, а точніше емуляція нового типу процесора (якщо це Softcore) може дозволити без значних часових втрат працювати з програмою, початково розробленої іншого процесора. При цьому властивість перенесення написаних мовами С і С++ програм є фактором, що істотно полегшує подібні модифікації.

Оскільки метою будь-якого підходу до проектування є спроби спрощення складання, а за потреби і коригування структури системи, то природним буде бажання побудови системи за модульним принципом. Природно, що цей же прийом розробники електронної апаратури спробували зберегти і під час переходу від розміщення елементів у формі корпусів ІС на друкованій підкладці до розміщення та трасування віртуальних компонентів у площині кристала. Об'єднання блоків, у тому числі і віртуальних компонентів, незалежно від їхнього типу (апаратних HW-core, програмних SW-core, схемотехнічних firm-core) здійснюється за допомогою шин. Модульність створює передумови і для переміщення рішень в галузі апаратного підходу в програмні рішення або навпаки.

Розглянуті вище проблеми та сучасні вимоги до проектування систем і, насамперед, типу SoC призвели до появи таких нових підходів до проектування, як поєднане (сумісне) проектування, поєднана (сумісна) верифікація та поєднане (сумісне) моделювання [3].

Со-проектування (сумісне проектування) – процес паралельного проектування апаратних та програмних засобів, при якому здійснюється оцінка доцільності вибору апаратної або програмної реалізації певного фрагмента проекту. Цей процес також дає проектувальникам можливість побачити (на абстрактному рівні), як система могла б працювати з даним поділом апаратних засобів процесора та програмного забезпечення.

Со-верифікація (сумісна верифікація) – аналіз спільної роботи програмного забезпечення та апаратних засобів процесора з метою

визначення, чи вони функціонуватимуть правильно разом. З іншого боку, під час цього процесу аналізується, чи однаково вирішуватимуться специфічні завдання проекту при апаратній чи програмній реалізації. Результатом роботи є фіксація обраного варіанта реалізації.

Со-симуляція (сумісне моделювання) – спільне моделювання роботи апаратних та програмних засобів, при якому апаратні засоби ЕОМ описані, наприклад, у формі мовних моделей (зазвичай це VHDL або Verilog-моделі).

Можливі два трактування поняття сумісного проектування: або це весь процес проектування, а со-верифікація і симуляція його окремі підетапи, або це початковий етап проектування, результатом якого є декомпозиція проблеми, а со-верифікація і симуляція це наступні і самостійні етапи проектування. Загалом завдання сучасного проектування SoC – виконувати кожен етап швидше, з більш надійними та достовірними (що не потребують перегляду та ітераційних повернень) результатами. Питання контролю проекту стає одним із вузлових. Якщо вдається добре протестувати кожен етап проекту та отримати оцінки його ефективності, то різко зменшується кількість ітераційних процедур, в со-верифікації (рис.1.4.).

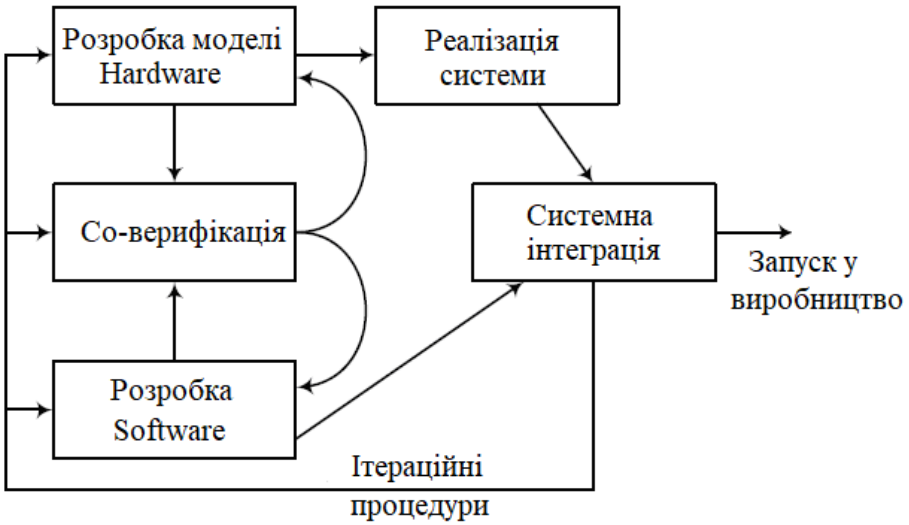

Рисунок 1.4 – Варіанти верифікації апаратних та програмних рішень

Проблеми та методи їх вирішення, що спираються на ідеї сумісного

проектування, при збігу їх загальних підходів відрізняються та виявляються різними для різних етапів проектування. Вузловою проблемою проектування мікропроцесорних систем завжди була проблема ухвалення рішення про поділ функцій між програмною та апаратною частинами системи. Від успішності віднесення розв'язуваного фрагмента системного завдання до тієї чи іншої реалізації часто залежить успіх чи неуспіх проекту. На початковому етапі проектування потрібно правильно (ефективно, за кінцевою оцінкою) розділити задачі, що вирішуються між апаратною та програмною реалізаціями. Переміщення фрагмента від однієї реалізації до іншої на заключних стадіях проектування може взагалі виявитися неможливим, а частіше призводить до істотних зрушень остаточної готовності проекту через необхідність повернення до попередніх етапів проектування. Основний підхід для початкового етапу проектування - пошук способу специфікації проекту, що підходить для опису апаратної, і програмної частини. Наприкінці цього етапу проектувальника цікавлять питання організації со-верифікаційної процедури, яка дозволила б оцінити правильність розбиття системи.

На етапі проектування окремих складових системи, що проектується, сумісне проектування означає паралельне проектування HW- та SW- засобів із взаємними та координованими змінами тих кроків розробки, де передбачено їхню взаємодію. Необхідність перерозподілу функцій або дій між HW і SW може бути доцільною на будь-якому кроці цього етапу проектної процедури, такий перерозподіл може бути дуже корисним при складанні частин проекту та його моделей. На заключних етапах розробки HW і SW необхідно переконатися, що отримані правильно функціонуючі частини. Основний метод тут со-верифікація. Якщо спільне HW/SW-моделювання виконується досить швидко, цей процес може використовуватися як основа со-проектування та со-верифікації.

Завданням комплексного налагодження HW та SW є швидке усунення помилок проектування. На цьому етапі найскладнішою стає стикування двох

різних підходів. Тут у проектувальників, які репрезентують протилежні сторони (HW і SW), з'являються суперечності в трактуванні причин і джерел відхилень характеристик системи від запланованих або очікуваних. Локалізація причини відхилення та способи її усунення – SW. Зручніше, щоб стикування відбулося якомога раніше, і налагодження виконувалося засобами, однаково ефективними і для апаратури, і для програмного забезпечення. Не менш важливо, щоб ці засоби були однаково зручними для розробників HW та SW.

Питання розбиття або перерозподілу завдань між HW та SW можуть виникати навіть на етапі вбудовування проекту в ІС або на етапах налагодження готового виробу. Проте що пізніше, то складніше і дорожче дається такий перерозподіл. Ідея сумісного проектування в SoC призводить до двох основних напрямків вирішення цього глобального завдання.

1. Відкладення питання виділення конкретного виконавця (для програмної чи апаратної реалізації) певного поведінкового фрагмента системи на якомога більш віддалений момент за умови, що опис поведінки фрагмента існує і може використовуватися для моделювання, а можливо навіть для синтезу фрагмента. Програмна реалізація може бути досить компактною. Після реалізації основних функцій ядра МП-системи додавання ще однієї програми, як правило, дуже слабко впливатиме на обсяг пам'яті, вже закладеної в систему, і не вимагатиме заміни типу використовуваної БІС пам'яті. Навіть істотна модернізація функціонування системи (при правильно закладених ресурсах модернізації) не потребує змін конструкції друкованої плати при переході до наступного більш ємного корпусу пам'яті.

2. Можливість не приймати остаточне рішення про поділ функцій до якомога пізнішого моменту. Це дозволяє відсувати вирішення питання доцільної фізичної реалізації фрагмента більш пізній термін. Таку можливість дає реалізація нетипової частини проекту у формі ВІС програмованої логіки. Перерозподіл функцій між апаратною та програмною частинами системи може бути виконано будь-якої миті часу життя

спроектованої системи. Граничним варіантом є реалізація у формі кристала типу SoC, коли конструктивна реалізація захована всередині одного кристала. SoC виявляються чудовою технічною основою сумісного проектування – оскільки системи виявляються у значних межах конструктивно інваріантними до апаратних та програмних рішень.

Перехід від одного способу реалізації до іншого вимагає істотної конструктивної переробки. Технологічною основою сумісного проектування є САПР, що інтегрують у собі засоби проектування та налагодження апаратно-програмних систем. Крім того, робота як самих САПР, так і проектувальників суттєво спрощується, якщо можливе використання інваріантних способів опису апаратної та програмної реалізації.

З наведених визначень ясна загальна постановка проблеми, проектна процедура у разі може мати вигляд, наведений на рис.1.5, проте питання у тому, як технічно організувати цю справді поєднану процедуру.

1 – проектування HW
2 – побудова та налагодження моделі HW
3 – побудова прототипів HW
4 – налаштування HW
5 – проектування SW
6 – побудова коду програми SW
7 – налаштування SW

Рисунок 1.5 – Процедура сумісного проектування

Ціла низка причин гальмує рішення у такій загальній постановці. Немає технічної бази. У проектах ще досить рідко використовуються сучасні кристали SoC. Проектувальники змушені спиратися на старі САПР, нові САПР виявляються дуже дорогими. Немає єдиних проектних засобів першого рівня (системного) інваріантних до реалізації. Нові технології проектування вимагають суттєвих змін і у мисленні самих проектувальників. Проектуванням складних апаратно-програмних систем традиційно

займаються фахівці трьох профілів: системні інженери, інженери програмісти та інженери-схемотехніки. Методи та засоби автоматизації, які використовуються цими розробниками, суттєво відрізняються – програмісти у своїй роботі більше уваги приділяли питанням ефективного опису програмного забезпечення з погляду його поведінки, тоді як схемотехніки традиційно тяжіли до засобів, що дозволяють ефективно описувати структуру проектованої системи.

Перегляд позицій, як видно з подальшого матеріалу, необхідний майже всюди. Доцільно зосередити зусилля на вирішенні завдань за чотирма ключовими напрямками:

– створення нових засобів та методології проектування;

– систематичне повторне використання базових елементів проектів;

– використання передових інструментів САПР;

– запровадження нових та модернізація існуючих стандартів з метою підвищення їх ефективності.

Нові проектні підходи були потрібні не тільки через величезну складність проектованих схем (десятки мільйонів вентилів), а й через такі проблеми, як часова кореляція між логічними та фізичними областями або достовірність верифікації таких великих систем. Питання конструкторсько-технологічної реалізації проекту у формі сучасних ВІС стають значущими вже для перших кроків проектування. Багато неявно видимих зв'язків між фізичним і схемотехнічним втіленнями проекту виявляються дуже суттєвими. Використання ідей сумісного проектування для SoC потребує вирішення двох основних проблем. Першою проблемою є введення таких засобів опису проекту, які б дозволили згладити протиріччя між HW- і SW-реалізаціями. Другою проблемою (спосіб вирішення якої виявляється значною мірою пов'язаною з першою) є розробка та використання нової техніки верифікацій та моделювання. Існуючі досі моделюючі кошти неспроможна за прийнятні терміни обробити послідовність подій у складних системах чи можуть давати спотворену інформацію. Виконання процедур,

що моделюють, на традиційних засобах може зайняти більше року. Потрібно введення певних додаткових кроків у верифікаційну процедуру, щоб якнайшвидше виявити ті проектні помилки, які здатні проявитися тільки після реалізації проекту у формі кристала, якщо вони залишаються непоміченими раніше.

До сучасних проектних засобів відносяться, перш за все, нові САПР з новими можливостями та побудовані на нових принципах. Однак не слід очікувати, що наявність добрих САПР вирішить усі проблеми. Від проектувальників вимагається відмова від застарілих методів та засобів та оволодіння новими. Розуміння описів, створюваних автоматичними проектними засобами, необхідне сучасного кваліфікованого розробника.

Завдання поведінки вузла, виконане однією з мов опису апаратури, або опис того ж блоку, але вже на рівні регістрових передач, повинні сприйматися проектувальником так само, як і схемотехнічне завдання структури цього вузла. Проектні засоби, що використовуються для розробок SoC, повинні не тільки органічно об'єднати два раніше розділені проектні маршрути HW і SW, а й включати нові проектні засоби. Слід враховувати, що вартість таких коштів буде на початкових етапах дуже значною.

1.4 Вбудоване самотестування в SoC

Самотестування програмно-апаратних систем на кристалі призначене для виявлення та діагностики потенційних помилок апаратного та програмного забезпечення всередині самого SoC. Запускаючи комплексні самотестування, розробники систем, що вбудовуються, можуть заздалегідь виявляти і усувати будь-які проблеми, які можуть поставити під загрозу загальну надійність і функціональність системи. Ці тести охоплюють широкий спектр областей, включаючи пам'ять, порти введення-виведення, таймери, периферійні пристрої і навіть процесорну частину. Мета – виявити будь-які аномалії чи невідповідності, які можуть виникнути в процесі

експлуатації, та вжити відповідних заходів для забезпечення надійної роботи системи.

У критично важливих для безпеки програмах, таких як автомобільні системи, авіоніка та медичне обладнання, наслідки відмови SoC можуть бути серйозними, призводячи до потенційно небезпечних для життя ситуацій. Самотестування SoC виступає як превентивний захід зі зниження ризику катастрофічних збоїв. Регулярно виконуючи самотестування, розробники та експлуатаційники можуть виявляти та усувати несправності до того, як вони переростуть у критичні, тим самим підвищуючи загальну безпеку та цілісність вбудованої системи.

У багатьох галузях, особливо в тих, які пов'язані з критично важливими з точки зору безпеки застосунками, діють суворі нормативні стандарти, яких необхідно дотримуватись. Дотримання цих стандартів необхідне забезпечення якості та надійності вбудованих систем. Самотестування SoC грає вирішальну роль виконанні цих вимог. Включивши комплексні механізми самотестування, розробники можуть продемонструвати, що їх системи пройшли ретельне тестування та відповідають необхідним нормативним критеріям.

На етапах розробки та обслуговування вбудованих систем виявлення основної причини будь-яких проблем чи несправностей може виявитися складним завданням. Самотестування SoC допомагає в процесі налагодження, надаючи цінну інформацію про стан SoC та пов'язані з ним компоненти. При виявленні несправності результати самотестування можуть допомогти розробникам ефективніше визначити джерело проблеми, прискорюючи процес усунення неполадок та скорочуючи загальний час розробки та обслуговування.

Вбудовані системи призначені для роботи протягом тривалих періодів часу, які часто охоплюють кілька років або навіть десятиліть. Протягом усього життєвого циклу цих систем на їхню продуктивність можуть впливати різні фактори, такі як умови довкілля, застарілі компоненти та оновлення

програмного забезпечення. Самотестування SoC відіграє вирішальну роль у забезпеченні довговічності та надійності вбудованих систем, дозволяючи періодично перевіряти будь-які відхилення від очікуваної поведінки. Контролюючи стан SoC з часом, розробники можуть активно вирішувати проблеми та здійснювати необхідне обслуговування чи заміну, цим продовжуючи термін служби системи.

Самотестування SoC є невід'ємною частиною сучасних вбудованих систем, надаючи розробникам важливий інструмент для підвищення надійності, запобігання катастрофічним збоям, відповідності нормативним стандартам, полегшенню налагодження та підтримки управління життєвим циклом продукту. Включивши у свої проекти комплексні механізми самотестування, розробники можуть забезпечити надійність та надійність своїх вбудованих систем у широкому спектрі галузей. Оскільки технології продовжують розвиватися, самотестування SoC залишиться життєво важливим компонентом, що гарантуватиме оптимальну продуктивність та безпеку вбудованих систем у все більш пов'язаному світі.

Проектування контролепридатних цифрових пристроїв (ЦП) базується на двох основних концепціях: спостережуваності та керованості. Під спостережуваністю розуміють можливість трансляції логічних значень внутрішніх ліній цифрової схеми її вихідні порти (зовнішні виходи). Керованість визначає можливість встановлювати потрібні логічні значення на внутрішніх полюсах цифрової схеми за допомогою її вхідних портів (зовнішніх контактів). Представлені концепції дозволяють здійснювати подачу тестових сигналів, що виробляються генератором тестових послідовностей (ГТП), який підключається до вхідних портів схеми, що тестується. У свою чергу, пристрій аналізу реакцій схеми на тестові впливи підключається до вихідних портів. Однією з основних проблем проектування контролепридатних цифрових пристроїв є використання додаткових зовнішніх висновків для підключення тестового обладнання. У разі реалізації ЦП на платформі SoC наявність додаткових контактів може бути лімітована

або взагалі неприйнятна. Для подолання цього обмеження широко застосовуються два підходи:

– використання послідовних каналів передачі тестових даних та прийому реакцій від пристрою;

– реалізація вбудованого самотестування цифрових пристроїв.

Перший підхід заснований на проектуванні внутрішньої схемотехніки ЦП з використанням зсувних регістрів, що дозволяють реалізовувати подачу тестових сигналів та отримання реакцій за мінімальної кількості додаткових виходів. Самотестування цифрових пристроїв, не виключаючи використання першого методу, полягає у реалізації схем ГТП та аналізатора спільно з функціональним ядром цифрового пристрою. При цьому мінімальна кількість додаткових зовнішніх виводів може дорівнювати двом: вхід ініціалізації процедури самотестування та вихід результату тестування (рис. 1.6).

Рисунок 1.6 – Узагальнена схема вбудованого самотестування

Проектування цифрових пристроїв, що самотестуються, ґрунтується на

застосуванні вбудованих засобів для формування тестових послідовностей та аналізу реакцій, що формуються на проміжних лініях і виходах цифрових пристроїв. Процедура аналізу здійснюється шляхом стиснення реакцій пристрою в короткі ключові слова з подальшим порівнянням з еталонними значеннями. Ідея самотестування базується на таких основних засадах:

– генератор тестових сигналів (послідовностей) вбудовується в цифровий пристрій, що тестується;

– вихідні реакції на тестові сигнали зберігаються цифровим пристроєм, що тестується, у вигляді компактних характеристик – ключових слів (сигнатур, синдромів, контрольних сум тощо);

– проведення процедури самотестування полягає лише в ініціалізації тестування та аналізі його результату;

– для побудови цифрових пристроїв, що самотестуються, необхідно використовувати мінімальну кількість додаткових зовнішніх виходів пристрою і незначний обсяг додаткової апаратури.

Проектування самотестованих пристроїв полягає не тільки у застосуванні вбудованих генераторів тестів та аналізаторів вихідних реакцій, а також у проектуванні пристрою з урахуванням його тестування в автономному режимі. Для цих цілей широко використовуються методи контролепридатного проектування, що застосовуються на практиці побудови сучасних цифрових пристроїв.

Таким чином проектування контролепридатних цифрових пристроїв є реалізація можливості окремого тестування комбінаційних схем та схем пам'яті пристрою. Цифровий пристрій може бути представлений як сукупність всіх комбінаційних підсхем, об'єднаних в єдину схему комбінаційну спільно з множиною елементів пам'яті, які утворюють послідовні підсхеми пристрою. Незважаючи на велику різноманітність методів контролепридатного проектування, найбільше застосування знаходять методи проектування послідовнісних цифрових схем, що використовують ідею сканування станів елементів пам'яті [5].

Серед всього наявного різноманіття методів контролепридатного проектування слід виділити такі.

1. Метод сканування елементів пам'яті. Сутність даного методу полягає у внесенні до структури цифрового пристрою додаткових мультиплексорів, що дозволяють перемикати входи пам'яті пристрою. Таким чином, в режимі тестування мультиплексори об'єднують усі елементи пам'яті пристрою в єдиний регістр зсуву, що дозволяє досягти додаткових можливостей його тестування. Всі елементи пам'яті тестуються ізольовано від комбінаційної частини. Наступний стан пристрою може бути встановлений незалежно від його поточного стану. Крім того, вихідні значення комбінаційної частини пристрою, що надходять на елементи пам'яті, можуть легко спостерігатися шляхом сканування станів елементів пам'яті.

2. Метод наскрізного зсувного регістру. Цей метод називається LSSD (Level-Sensitive Scan Design) і є стандартною технікою проектування цифрових пристроїв, запропонованої фірмою IBM.

3. Метод довільного сканування (Random Access Scan Technique). Цей метод є одним із різновидів методу сканування станів елементів пам'яті. Подібно до класичної схеми методів сканування, даний метод також дозволяє порівняно просто встановлювати кожен елемент пам'яті в необхідний стан і спостерігати його на зовнішніх контактах ЦП. Відмінністю є використання цих цілей елементів пам'яті з довільною адресацією, що дозволяє незалежно встановлювати або аналізувати стан будь-якого з них.

4. Метод сканування граничних елементів пам'яті. Подальшим розвитком ідеї сканування станів елементів пам'яті є метод проектування цифрових схем, що реалізуються з архітектурою сканування граничних елементів пам'яті (Boundary-Scan Architecture). Основна ідея методу граничного сканування полягає в створенні в режимі тестування зсувного регістру елементів пам'яті ЦП, розташованих між її зовнішніми контактами і функціональним ядром.

2 ТЕХНОЛОГІЧНА ПЛАТФОРМА РЕАЛІЗАЦІЇ SoC

2.1 Процесорні платформи сімейства Zynq-7000 AP SoC

Досягнення сучасної напівпровідникової промисловості дають змогу виготовляти інтегральні схеми цифрових пристроїв із надвеликим ступенем інтеграції. Виробники ПЛІС типу FPGA збільшують не тільки число і щільність компонування логічних блоків, але й здійснюють розміщення на програмованому кристалі таких функціональних блоків, як мікропроцесорні ядра, вбудовані пристрої, блоки арифметичних прискорювачів, інтерфейсні контролери стандартного обладнання тощо, перетворюючи цим ПЛІС в програмовані системи на кристалі.

Одним із прикладів програмованої системи на кристалі є FPGA сімейства ZYNQ-7000 фірми Xilinx Inc. Для побудови SoC різного призначення платформа Zynq-7000 EPP містить наступні функціональні блоки: процесорну підсистему, що включає процесорний модуль, інтерфейси пам'яті, периферійні інтерфейси, міжблокові інтерфейси та інтерфейси до програмованої логіки, а також програмовану логіку. При цьому процесор ARM Cortex A9 MPCore має вбудовану пам'ять, багатий набір периферійних пристроїв, інтерфейси до зовнішньої пам'яті. Взаємодія між двома процесорами може здійснюватися за допомогою міжпроцесорних переривань через область загальної пам'яті шляхом передачі повідомлень [6].

Інтегральна схема ZYNC-7000 EPP (Extensible Processing Platform) виконана за технологічним процесом SRAM 28 нм і є FPGA з впровадженими додатковими функціональними блоками. Крім 350 тисяч логічних блоків (для FPGA Z-7045) кристал включає багатоядерний блок обчислювачів, побудований на базі двох ядер процесора ARM Cortex-A9, 2 Мбіт вбудованої багатопортової пам'яті, контролери зовнішніх динамічних оперативних запам'ятовуючих пристроїв (DDR2 і DDR3), контролери пристроїв flash-

пам'яті (NAND і NOR), два вбудовані блоки високошвидкісних 12-розрядних аналого-цифрових перетворювачів, вбудований контролер інтерфейсу PCIe, вбудовані контролери спеціалізованих інтерфейсів таких, як I2C, USB 2 Ethernet, UART, CAN, SPI тощо.

Програмована логіка містить конфігуровані логічні блоки (CLB), конфігуровані двопортові блоки пам'яті (BRAM), комірки цифрової обробки сигналів з 25×18-бітним помножувачем, 48-бітним акумулятором і суматором (DSP48E1), АЦП XADC, керовані блоки формування, конфігурований блок шифрування та автентифікації, конфігурований блок вводу-виводу (Select IO). (рис. 2.1).

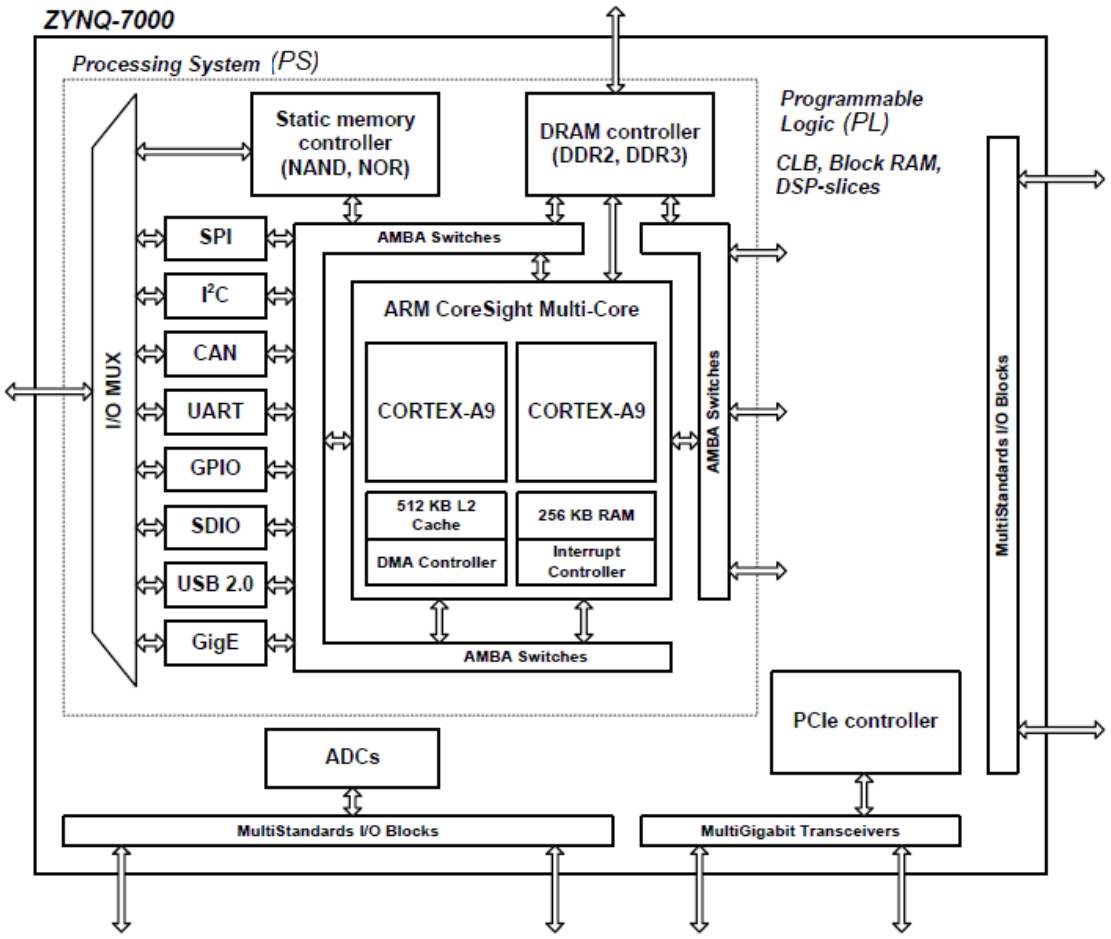

Рисунок 2.1 – Архітектура SoC Xilinx ZYNQ-7000

Xilinx ZYNQ-7000 платформа має архітектуру, де поєднано PS (processing system) – систему обробки та PL (programmable logic) -

програмовану логіку. Взаємодія двох систем використовується через інтерфейс AMBA на основі протоколу AXI.

Однією з популярних платформ розробки для роботи з Zynq є ZedBoard. Назва «Zed» означає Zynq Evaluation and Development. Загальний вигляд плати ZedBoard з елементами інтерфейсу представлений на рис. 2.2.

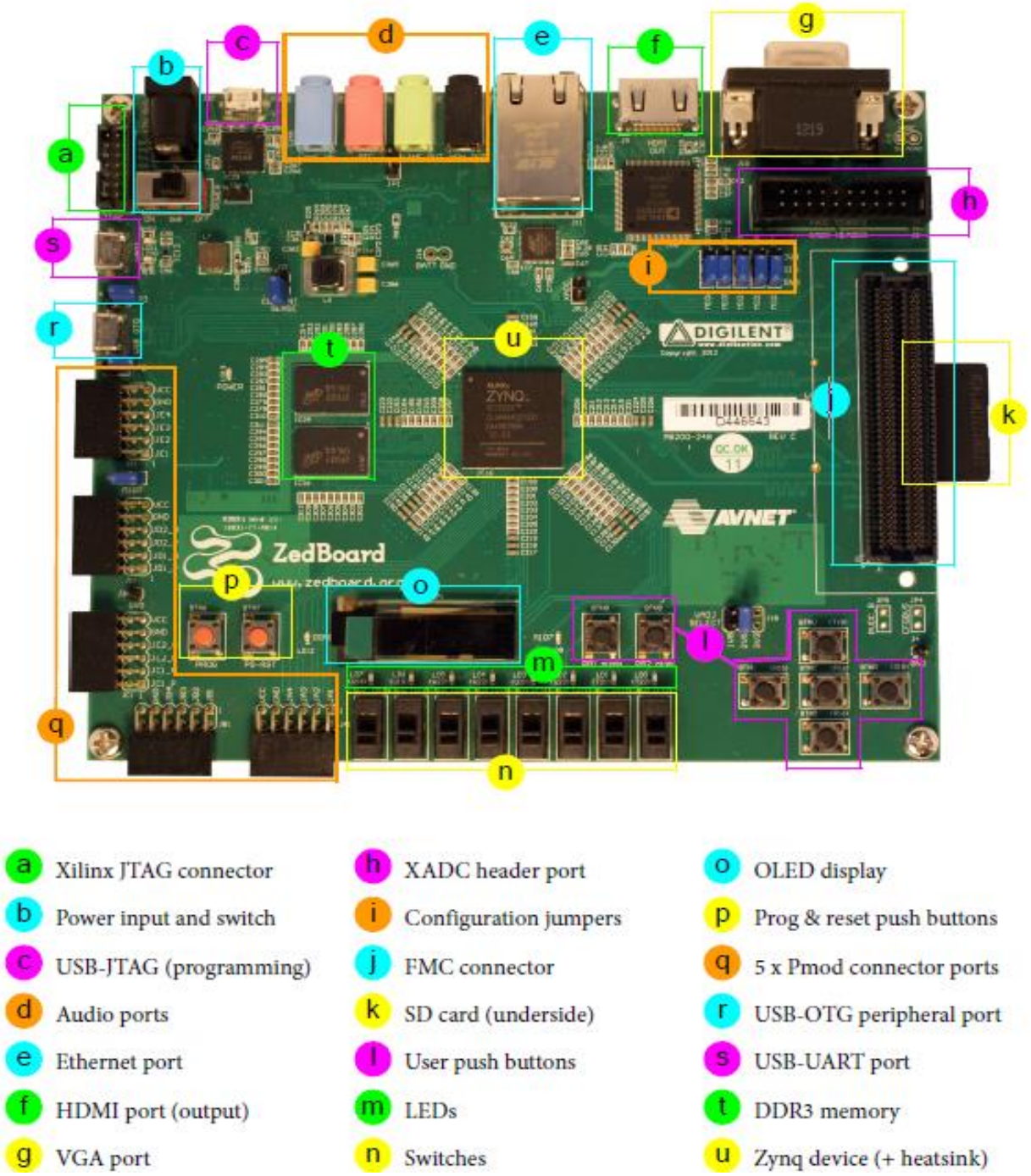

Рисунок 2.2 – Загальний вигляд плати ZedBoard

ZedBoard – це недорога загальнодоступна плата, оснащена пристроєм XC7Z020 Zynq. Це спільне підприємство Xilinx, Avnet (дистриб'ютор) і Digilent (виробник плат). Незважаючи на те, що ZedBoard є платформою розробки для промисловості, він також орієнтований на студентів, учених і любителів, пропонує спеціальні матеріали для нових користувачів Zynq і орієнтований на навчання новачків [7].

2.2 Цифровий інтерфейс AXI Bus

Привабливість Zynq-7000 полягає не тільки у властивостях його складових частин, PS та PL, але й у можливості використовувати їх у тандемі для формування закінчених інтегрованих систем. Ключовим фактором у цьому відношенні є набір ретельно визначених міжз'єднань та інтерфейсів AXI, що утворюють міст між двома частинами. AXI (Advanced eXtensible Interface), що означає розширений розширюваний інтерфейс, є протоколом інтерфейсу, визначений ARM як частина стандарту AMBA (Advanced Microcontroller Bus Architecture, розширена архітектура шини мікроконтролера). Багато пристроїв та IP-блоки, що випускаються сторонніми виробниками та розробниками, засновані на цьому стандарті [7].

Розширена архітектура шини мікроконтролера ARM (Advanced Microcontroller Bus Architecture, AMBA) – це відкрита стандартна специфікація міжз'єднань на кристалі для підключення та управління функціональними блоками в конструкціях система на кристалі (SoC). Це полегшує розробку багатопроцесорних конструкцій із великою кількістю контролерів та компонентів з архітектурою шини. З моменту свого створення AMBA, незважаючи на назву, вийшла далеко за межі мікроконтролерів. Сьогодні AMBA широко використовується в ряді частин ASIC та SoC, включаючи процесори, що використовуються у сучасних портативних мобільних пристроях, таких як смартфони. AMBA – зареєстрована торгова марка ARM Ltd..

AMBA була представлена ARM в 1996 році. Першими шинами AMBA були Advanced System Bus (ASB) та Advanced Peripheral Bus (APB). У другій версії AMBA 2 в 1999 році ARM додала високопродуктивну шину AMBA (AHB), яка є протоколом з одним фронтом тактової частоти. У 2003 році ARM представила третє покоління AMBA 3, включаючи AXI для досягнення ще більш високої продуктивності міжз'єднання та Advanced Trace Bus (ATB) як частина вбудованого рішення CoreSight для налагодження та трасування. У 2010 році були представлені специфікації AMBA 4, починаючи з AMBA 4 AXI4, а потім у 2011 році розширивши загальносистемну узгодженість за допомогою AMBA 4 AXI Coherency Extensions (ACE). У 2013 році було представлено специфікацію AMBA 5 з високошвидкісним транспортним рівнем та функціями, призначеними для зменшення навантаження.

Ці протоколи сьогодні є стандартом де-факто для архітектур із вбудованими процесорами, оскільки вони добре документовані та можуть використовуватись без ліцензійних відрахувань.

AXI - це двоточковий інтерфейс, розроблений для високопродуктивних та швидкодіючих мікроконтролерних систем. Протокол AXI заснований на каналі «точка-точка», що виключає спільне використання шини та дозволяє збільшити пропускну здатність та зменшити затримку. AXI є найпопулярнішим серед усіх міжз'єднань інтерфейсу AMBA. Він забезпечує основу у тому, як різні блоки всередині кожного чіпа взаємодіють друг з одним. Механізм квитування гарантує, що дані передаються чітко та безперебійно. Це дозволяє різним компонентам взаємодіяти без будь-яких перешкод.

Передача даних між Master та Slave пристроями відбувається відповідно до специфікації протоколу AXI:

– ведучий (Master) та ведений (Slave) пристрої повинні встановити «міжз`єднання» (handshake) для автентифікації дійсних сигналів;

– передача сигналу, що управляє, здійснюється в окремі фази;

– є окремі канали передачі сигналу;

– безперервна передача може бути досягнута за допомогою пакетного зв'язку.

Протокол AXI визначає 5 каналів (рис. 2.3):

– 2 використовують для транзакцій читання:

прочитати адресу, прочитати дані;

– 3 використовуються для транзакцій запису:

записати адресу, записати дані, записати відповідь.

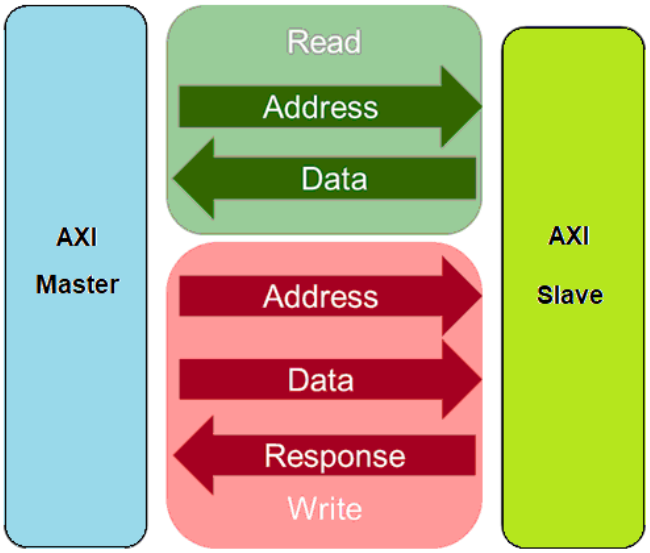

Рисунок 2.3 – Структура протоколу AXI

Транзакції читання AXI вимагають кількох передач двома каналами читання:

– спочатку канал читання адреси передається від ведучого до веденого для встановлення адреси та деяких сигналів керування;

– потім дані на цю адресу передаються від веденого пристрою до ведучого по каналу даних читання.

Важливо, що на одну адресу може здійснюватися кілька передач даних. Цей тип транзакції називається пакетним.

Транзакції запису AXI вимагають кількох передач 3 каналами читання.

– спочатку канал запису адреси передається від ведучого до веденого

для встановлення адреси та деяких керуючих сигналів;

– потім дані за цією адресою передаються від ведучого до веденого по каналу запису даних;

– нарешті, відповідь на запис відправляється від веденого пристрою до ведучого по каналу відповіді на запис, щоб вказати, чи була передача успішною (рис.2.4).

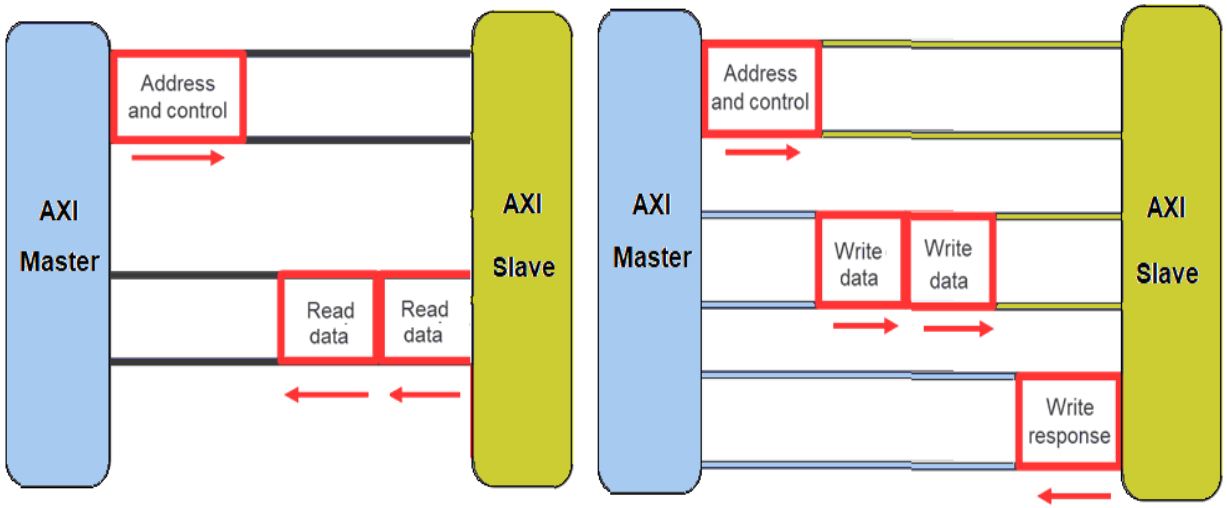

Рисунок 2.4 – Транзакції читання та запису протоколу AXI

2.3 Засоби автоматизованого проектування та етапи розробки вбудованих систем на SoC Zynq-7000

Об'єднання в одному кристалі апаратного двоядерного процесорного блоку з архітектурою ARM Cortex-A9 та програмованої логіки FPGA останнього покоління надає розробникам широкі можливості для проектування вбудованих систем різного призначення з динамічно конфігурованою структурою. Програмовані системи на кристалі All Programmable System-On-Chip (AP SoC) сімейства принципово відрізняються від ПЛІС з апаратними мікропроцесорними ядрами PowerPC сімейств Virtex-4 FX і Virtex-5 FXT, які виробляє компанія Xilinx, як в архітектурному відношенні, так і організацією взаємодії програмованої логіки та

процесорного ядра. У ПЛІС цих сімейств мікропроцесорне ядро може функціонувати лише після завершення процедури конфігурування основних логічних ресурсів. У кристалах обчислювальних платформ сімейства Zynq-7000 AP SoC, що розширюються, апаратний процесорний блок є основним компонентом, який доступний до конфігурування програмованої логіки і може використовуватися незалежно від логічних ресурсів.

Важливою перевагою кристалів процесорних платформ, порівняно з ПЛІС перелічених вище сімейств є істотно менша вартість, що дозволяє ефективно використовувати їх для серійної реалізації вбудованих систем різного рівня складності та функціонального призначення. Активному поширенню програмованих систем на кристалі сімейства Zynq-7000 AP SoC сприяє також надання фірмою Xilinx широкого спектру інструментальних налагоджувальних комплектів та комплексів засобів автоматизованого проектування, що дозволяють у короткі терміни виконувати всі етапи розробки вбудованих мікропроцесорних систем, включаючи налагодження апаратної частини та програмного забезпечення.

Процес проектування вбудованих систем, що реалізуються на базі кристалів програмованої логіки з архітектурою FPGA і розширюваних обчислювальних платформ сімейства Zynq-7000 AP SoC, в загальному випадку включає наступні етапи:

– розробка проекту мікропроцесорної системи;

– проектування апаратної платформи системи, що розробляється;

– підготовка системних програмних засобів нижнього (апаратного) рівня;

– формування основного програмного забезпечення системи, що розробляється;

– комплексне моделювання та налагодження;

– генерація завантажувального образу та розгортання розробленої системи.

У більш наочному вигляді послідовність виконання перерахованих

етапів розробки вбудованих мікропроцесорних систем на основі ПЛІС з архітектурою FPGA і кристалів обчислювальних платформ сімейства Zynq-7000 AP SoC, що розширюються, представлена на рис. 2.5. Також наведено скорочений зміст кожного з етапів проектування [8].

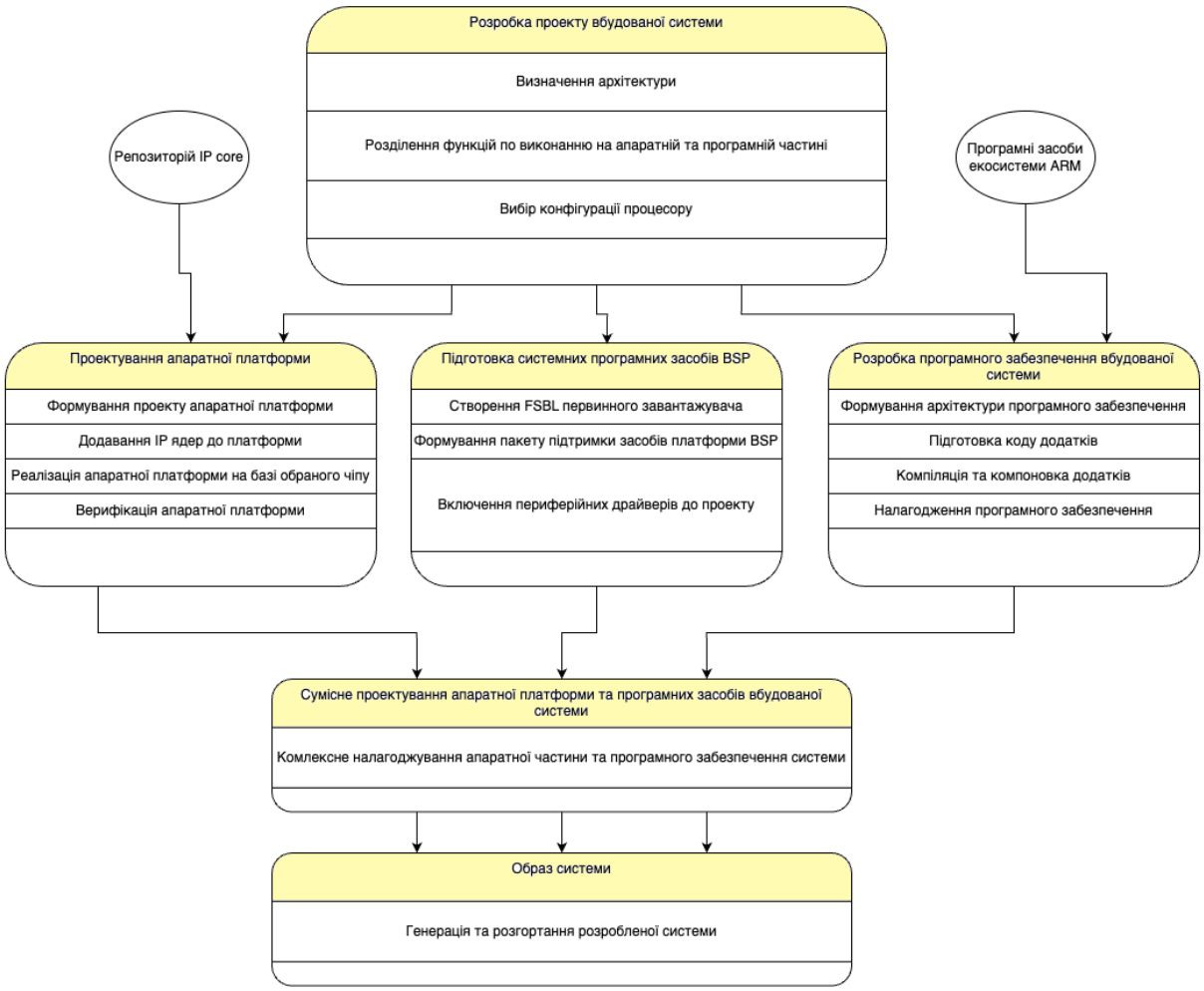

Рисунок 2.5 – Основні етапи проектування вбудованої системи на базі SoC Xilinx ZYNQ 7000

Процесорна система PS кристалів обчислювальних платформ сімейства Zynq-7000 AP SoC, що розширюються, підтримує чотири варіанти конфігурації: одноядерний; двоядерний симетричний мультипроцесорний (Symmetric Multi-Processing, SMP); двоядерний асиметричний мультипроцесорний (Asymmetric Multi-Processing, AMP); двоядерний

віртуальний симетричний чи асиметричний мультипроцесорний.

При виборі одноядерного варіанта конфігурації процесорного блоку PS для завантаження операційної системи та прикладного ПЗ використовується лише одне ядро ARM Cortex-A9 MPCore. Цей варіант можна застосовувати і для автономного програмного забезпечення, яке функціонує без операційної системи. При використанні одноядерного варіанта конфігурації процесорної системи PS друге ядро процесорного блоку перетворюється на вимкнений стан, наприклад, за допомогою технології Clock Gating, тим самим забезпечується зниження рівня енергоспоживання.

Двоядерний симетричний мультипроцесорний варіант конфігурації процесорної системи дозволяє використовувати ту саму операційну систему одночасно для двох ядер ARM Cortex-A9 MPCore. У цьому випадку керування всіма процесами, які виконують ці ядра, здійснюється засобами операційної системи. Даний варіант конфігурації процесорної системи можна застосовувати лише для операційних систем, що підтримують режим SMP, наприклад Linux.

Двоядерний асиметричний мультипроцесорний варіант конфігурації процесорного блоку дозволяє встановити для кожного з ядер ARM Cortex-A9 MPCore різні операційні системи. Іншими словами, кожне ядро процесорного блоку при використанні цього варіанта функціонує під керуванням власної операційної системи..

Двоядерний віртуальний симетричний або асиметричний мультипроцесорний варіант конфігурації відрізняється від розглянутих вище варіантів SMP та AMP використанням спеціального програмного інструменту управління операційними системами – гіпервізора (hypervisor).

Після вибору варіанта конфігурації процесорного блоку, що забезпечує необхідну продуктивність при виконанні функцій системи, що вбудовується, можна перейти безпосередньо до проектування її апаратної платформи.

Розробка вбудованого програмного забезпечення здійснюється з використанням Vitis IDE. Узагальнений процес проектування ПЗ на базі Vitis

IDE являє собою типовий цикл розробки для вбудованої платформи. Основна частина це конфігурування платформи для цільової плати та генерація SDK. Після цього етапу є можливим використати API для роботи у режимі baremetal або з додаванням FreeRTOS. На Рисунку 2.6 наведено вікно конфігурації платформи та board support package.

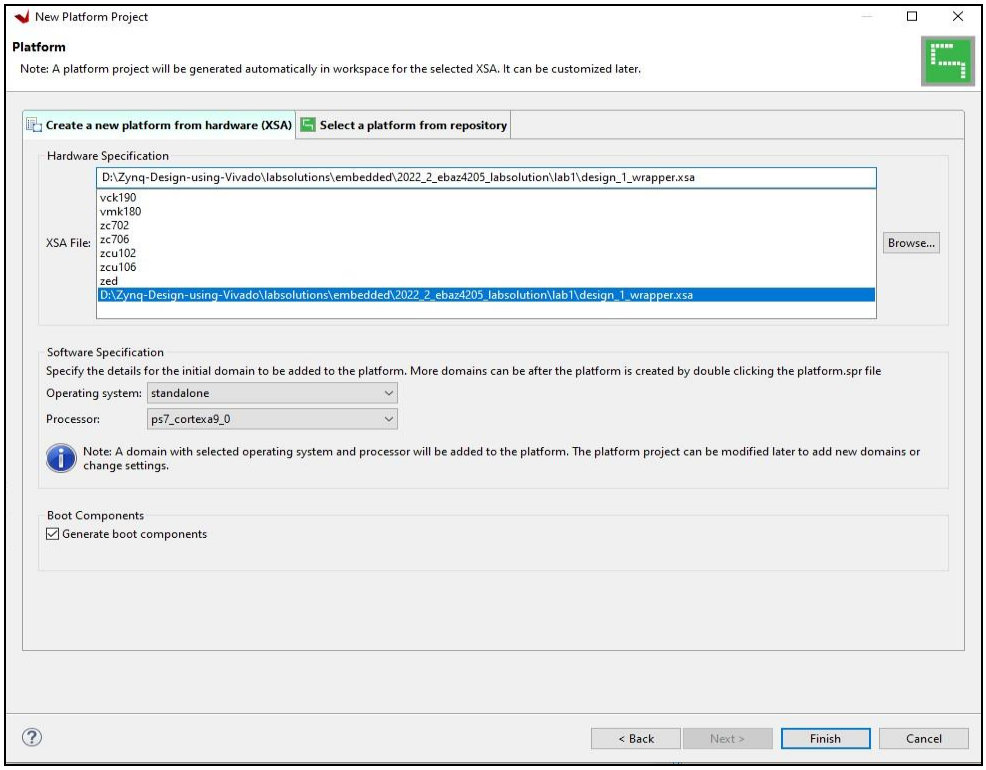

Рисунок 2.6 – Вікно конфігурації та генерації SDK/board support package

Вбудоване середовище розробки Vitis IDE містить готові базові приклади роботи з Zynq-7000, серед яких додаток тесту DDR пам’яті, перевірка LwIP мережевого стеку. Після створення апаратної частини у Vivado та генерації необхідних файлів є доцільним пройти перевірку цілісності пам’яті та виконати додаток memory_test, результати якого будуть доступні у консолі послідовного порту, де підключено плату. Зазначимо, що для програмування цільової платформи може бути використано як офіційний Xilinx Platform Cable, так і Xilinx Virtual Cable Protocol, що дозволяє виконувати прошивку та налагодження віддаленого пристрою .

На рис. 2.6 наведено структуру проекту для ARM-частини Xilinx

ZYNQ-7000 для перевірки DDR пам’яті та її налаштувань. Прошивка та налагодження відбуваються через вбудовані інтерфейси програмування. Результати виконання тесту наведено на рис. 2.7, де зазначені регіони пам’яті що були протестовані та регіони адресації.

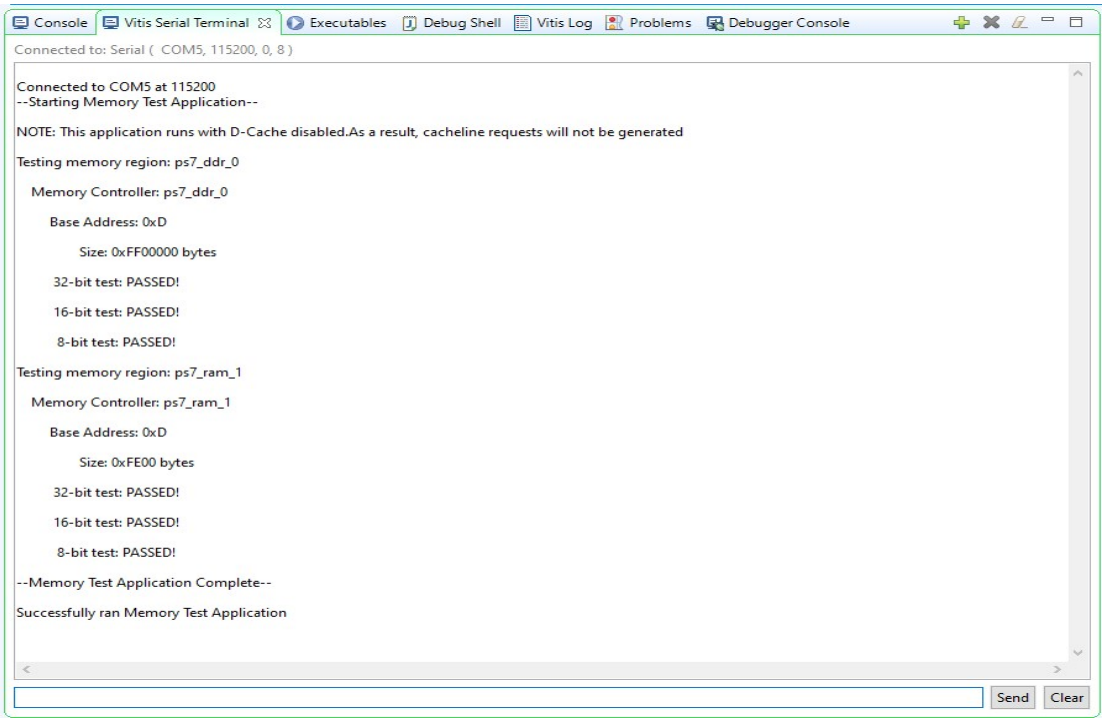

Рисунок 2.7 – Результати тестування пам’яті додатком mem_test для платформи Xilinx ZYNQ.

Узагальнений процес проектування IP-ядер виконується у середовищі Vivado IDE з використанням як готових блоків, так і користувацьких ядер. Після створення IP-ядра є можливість виконувати як контроль версій і відповідно оновлювати блоки, так і інтегрувати ядра через процес System Block Designer, де можна налаштувати параметри блоку та інтегрувати його разом до PL частини. Одним з доступних варіантів генерації IP ядер є використання Vitis HLS Synthesis для роботи з високорівневим синтезом. Також може бути використаний класичний підхід з Verilog/VHDL описом і відповідним testbench. На рисунку 2.8 наведено основне вікно створення нового IP-ядра у середовищі Vivado.

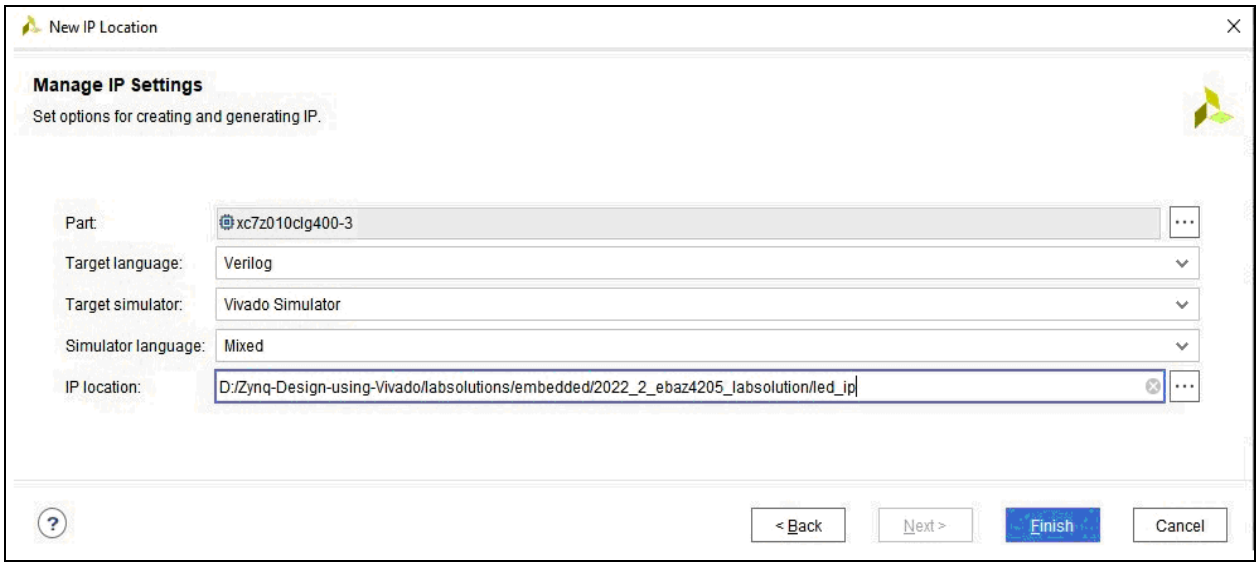

Рисунок 2.8 – Вікно додавання нового користувацького IP-ядра до системи.

Після створення IP-ядра необхідно під'єднати основні комунікаційні інтерфейси, а саме AXI шину та вхідні/вихідні порти. Для цього необхідно модифікувати згенерований шаблон на мові опису апаратури Verilog/VHDL.

У лістингу 2.1 наведено приклад декларації додавання користувацького блоку до шини AXI4.

Лістинг2.1 – Декларація підключення блоку до шини AXI4

```
1  led_user_logic  # (
2      .LED_WIDTH(LED_WIDTH)
3    )
4   U1(
5       .S_AXI_ACLK(S_AXI_ACLK),
6       .slv_reg_wren(slv_reg_wren),
7       .axi_awaddr(axi_awaddr[C_S_AXI_ADDR_WIDTH-1:ADDR_LSB]),
8       .S_AXI_WDATA(S_AXI_WDATA),
9       .S_AXI_ARESETN(S_AXI_ARESETN),
10      .LED(LED)
11    );
```

Після інсталювання блоку необхідно реалізувати базовий функціонал. Як приклад, наведемо реалізацію блоку виводу на LED індикатори обраної плати налагодження. У лістингу 2.2 наведено реалізацію користувацького блоку на мові опису Verilog.

Лістинг 2.2 - Користувацький блок на шині AXI4

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////
// Module Name: led_user_logic
//////////////////////////////////////////////////////////////////

module led_user_logic # (
          parameter LED_WIDTH = 3
          )
          (
   input S_AXI_ACLK,
   input slv_reg_wren,
   input [2:0] axi_awaddr,
   input [31:0] S_AXI_WDATA,
   input S_AXI_ARESETN,
   output reg [LED_WIDTH-1:0] LED
   );
  always @( posedge S_AXI_ACLK )
  begin
   if ( S_AXI_ARESETN == 1'b0 )
      LED <= 4'b0;
   else
     if (slv_reg_wren && (axi_awaddr == 3'h0))
        LED <= S_AXI_WDATA[LED_WIDTH-1:0];
     end
endmodule
```

Процес валідації IP блоку включає до себе написання відповідного testbench та інтеграції у System Block Design. Зазначимо, що після проходження процесу валідації та збирання IP стає доступним у репозиторії проекту, звідки може бути доданий до системи та налаштований. Основними параметрами налаштування можуть бути як константи, що можуть бути змінені для конкретного дизайну, так і параметри AXI шини. На рис. 2.9 наведено вікно конфігурації параметрів IP блока у середовищі Vivado.

Доданий до block design IP модуль буде доступний з автоматично згенерованим драйвером у середовищі Vitis де буде додана частина викликів, що необхідні для взаємодії блоку з AXI шиною. Як приклад, у лістингу 2.3 наведено фрагмент header файлу для обраного IP блоку, де можна побачити автоматично означений адрес пристрою на шині та згенеровану функцію запису даних до блоку.

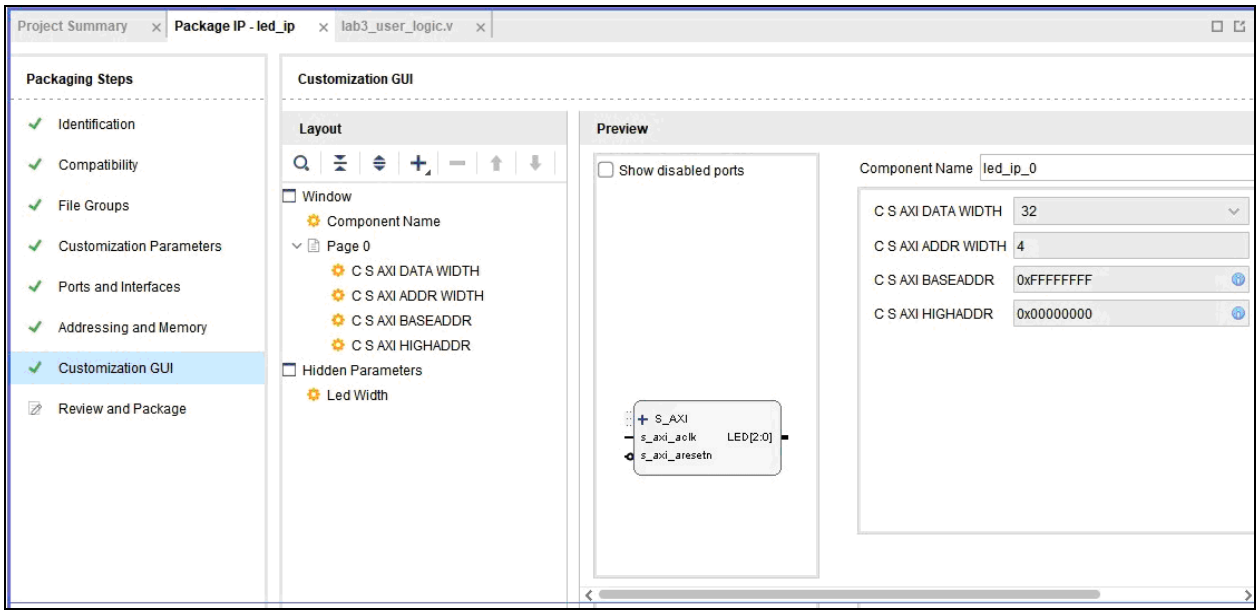

Рисунок 2.9 – Вікно конфігурації користувацького IP-ядра.

Лістинг 2.3 – Фрагмент згенерованого виклику запису до IP-блоку

```
/**
* Write a value to a LED_IP register. A 32 bit write is performed.
 * If the component is implemented in a smaller width, only the least
 * significant data is written.
 *
 * @param   BaseAddress is the base address of the LED_IP device.
 * @param   RegOffset is the register offset from the base to write to.
 * @param   Data is the data written to the register.
*
 * @note
 * C-style signature:
 *       void LED_IP_mWriteReg(Xuint32 BaseAddress, unsigned RegOffset,    Xuint32 Data)
*/
#define LED_IP_mWriteReg(BaseAddress, RegOffset, Data) \
        Xil_Out32((BaseAddress) + (RegOffset), (Xuint32)(Data))
```

Якщо на PS частині розгорнуто дистрибутив на базі Petalinux-Yocto based Linux то процес взаємодії з IP блоком може виконуватись через devmem, де виконується звертання до фізичного адресного простору з userspace-частини. У лістингу 2.4 наведено фрагмент роботи з синтезованим BRAM з використанням devmem та mmap.

Лістинг 2.4 – Фрагмент роботи з синтезованим BRAM

```
 unsigned int bram_size = 0x8000;
   off_t bram_pbase = 0xA0000000; // physical base address
   u64 *bram64_vptr;
   int fd;
   // Map the BRAM physical address into user space getting a virtual address for it
   if ((fd = open("/dev/mem", O_RDWR | O_SYNC)) != -1) {
      bram64_vptr = (u64 *)mmap(NULL, bram_size, PROT_READ|PROT_WRITE,
MAP_SHARED, fd, bram_pbase);
      // Write to the memory that was mapped, use devmem from the command line of Linux to
verify it worked
      // it could be read back here also
      bram64_vptr[0] = 0xDEADBEEFFACEB00C;
      close(fd);
   }
```

2.4 Апаратне забезпечення самотестування пристрою логічного керування на SoC

Розглянемо апаратну реалізацію самотестування пристрою логічного керування на базі SoC. Практична реалізація виконується на базі стеку інструментів Vivado/Vitis/Vitis HLS [9]. Це дозволяє виконати наскрізне проектування як частини, що синтезується з боку Programmable Logic, так і користувацького програмного забезпечення з боку Processing System.

Процес розробки IP блоку у середовищі VitisHLS починається з додавання файлів testbench та користувацьких файлів для синтезу блоків. Зазначимо, що технологія високорівневого синтезу активно розвивається як з боку Xilinx(AMD) так і Altera(Intel) для задач моделювання комп'ютерного зору та машинного навчання.

Для проведення діагностичного експерименту (ДЕ) визначимо, що реалізація керуючого FSM виконується на стороні PL та верифікація реалізується шляхом обходу графу переходів на стороні PS у вигляді користувацького програмного забезпечення. Комунікація між блоками PS-PL виконана з використанням AXI шини. Для блоку FSM був реалізований testbench у середовищі Vitis HLS та отримані результати синтезу блоку.

Зберігання еталонів реалізовано на стороні PS частини у ROM сегменті

користувацького ПЗ для тестування. Генерацію тестів виконано шляхом реалізації обходу графу за допомогою надсилання відповідних запитів на PL частину. На рис. 2.10 наведено діаграму розробленого проекту у середовищі Vivado де виконано поєднання стандартних IP блоків та розроблених у середовищі Vitis.

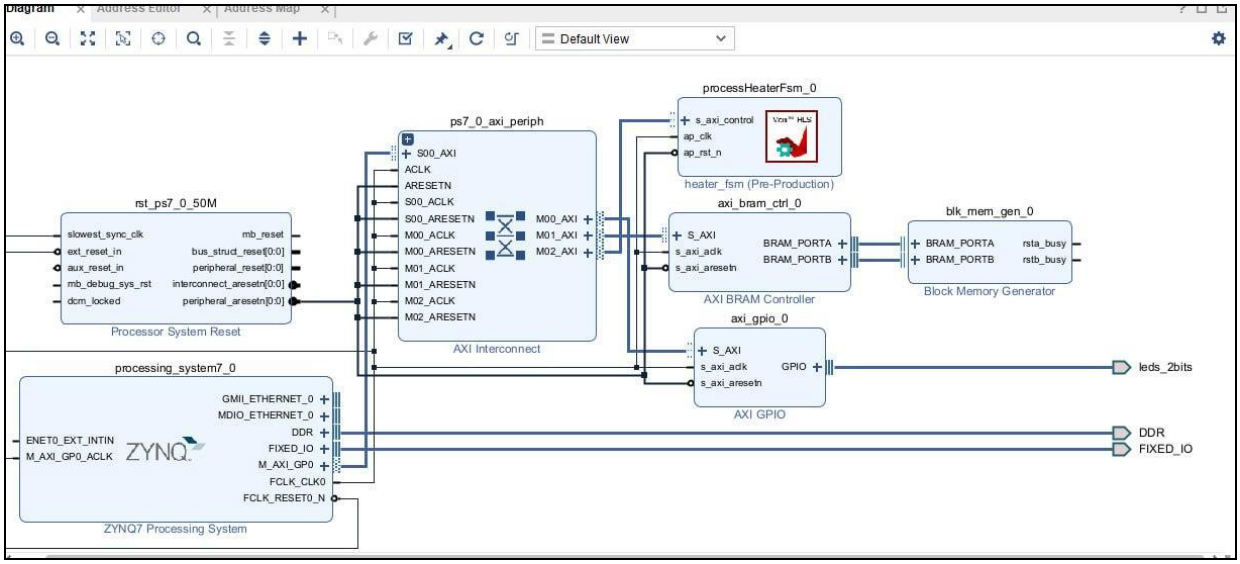

Рисунок 2.10 – Блок-діаграма розробленого проекту на базі Vivado та Vitis HLS зі зберіганням тестових еталонів у ROM сегменті користувацького ПЗ.

Реалізація блоку з боку Vitis HLS представляє опис блоку, що має бути синтезований на мовах C/C++ з проміжною компіляцією результатів з використанням LLVM за технологією, запропонованою в [10]. Зазначимо, що не всі конструкції можуть бути коректно синтезовані чи імплементовані, тому є необхідність звертатися до Vitis High Level Synthesis User Guide(UG1399) за детальним роз’ясненням можливих несинтезованих конструкцій. Основною перевагою розробки з використанням Vitis HLS є можливість виконувати реалізацію алгоритму на високорівневих мовах програмування C/C++ та сумісну симуляцію. Окремою перевагою є автоматична генерація підключення блоку до AXI4-Lite/AXI Stream/GPIO за рахунок pragma-директив HLS-компілятору. Зовнішній інтерфейс IP модуля задається через спеціалізовані компілятор-специфічні директиви pragma.

Фрагмент реалізації інтерфейсу FSM наведено у лістингу 2.5. Директивами pragma задано, які аргументи синтезованого HLS компоненту будуть доступні через інтерфейс AXI та відповідно оброблені.

Лістинг 2.5 – Фрагмент реалізації інтерфейсу FSM синтезованого на PL частині

```
#include "heater_fsm.hpp"

HeaterFsm fsm;

void processHeaterFsm(int sensorIn, int &controlOut, bool extClock) {
	#pragma HLS INTERFACE ap_ctrl_none port=return
	#pragma HLS INTERFACE s_axilite port=sensorIn
	#pragma HLS INTERFACE s_axilite port=controlOut
	#pragma HLS INTERFACE s_axilite port=extClock

	fsm.handleSensorInputs(sensorIn, extClock);
	controlOut = fsm.getControlSignals();
}
```

HLS дозволяє використовувати testbench, написаний на мовах C/C++, таким чином є можливість провести базову симуляцію та верифікацію блоку. З доступних операторів є можливість як використовувати assert для порівняння значень, так і використовувати оператори вводу/виводу до файлового потоку. Розглянемо фрагмент testbench який виконується на стороні Vitis HLS забезпечення. У лістингу 2.6 наведено фрагмент testbench-файлу який використовується для перевірки коректності блоку у симуляції Vitis.

Лістинг 2.6 – Фрагмент реалізації testbench для обходу графу FSM у режимі симуляції середовища Vitis HLS

```
	// Simulate transition to A3
	assert(fsm.getCurrentState() == FsmStates::A2);
	processHeaterFsm(OnnSignal | Tm, controlOutput, false);
	processHeaterFsm(OnnSignal | Tm, controlOutput, true);
	assert(fsm.getCurrentState() == FsmStates::A3);
	assert(controlOutput == (Y1 | Y3));

	// Simulate transition to A4
```

```
assert(fsm.getCurrentState() == FsmStates::A3);
processHeaterFsm(!OnnSignal, controlOutput, false);
processHeaterFsm(!OnnSignal, controlOutput, true);
assert(fsm.getCurrentState() == FsmStates::A4);
assert(controlOutput == Y2);

//Simulate transition to A0
assert(fsm.getCurrentState() == FsmStates::A4);
for (int it = 0; it < T4; ++it) {
        processHeaterFsm(0, controlOutput, false);
        processHeaterFsm(0, controlOutput, true);
}
processHeaterFsm(0, controlOutput, false);
processHeaterFsm(0, controlOutput, true);
```

Після виконання testbench є можливим підготувати та експортувати готовий блок до середовища Vivado, таким чином додавши його до дизайну.

Синтез блоку виконується як один з етапів design-flow у середовищі vitis, після чого експортується до Xilinx Vivado де задаються основні параметри блоку і виконується підключення до AXI шини. На рис. 2.11 наведено результати синтезу та використання ресурсів SoC XC7Z010CLG400.

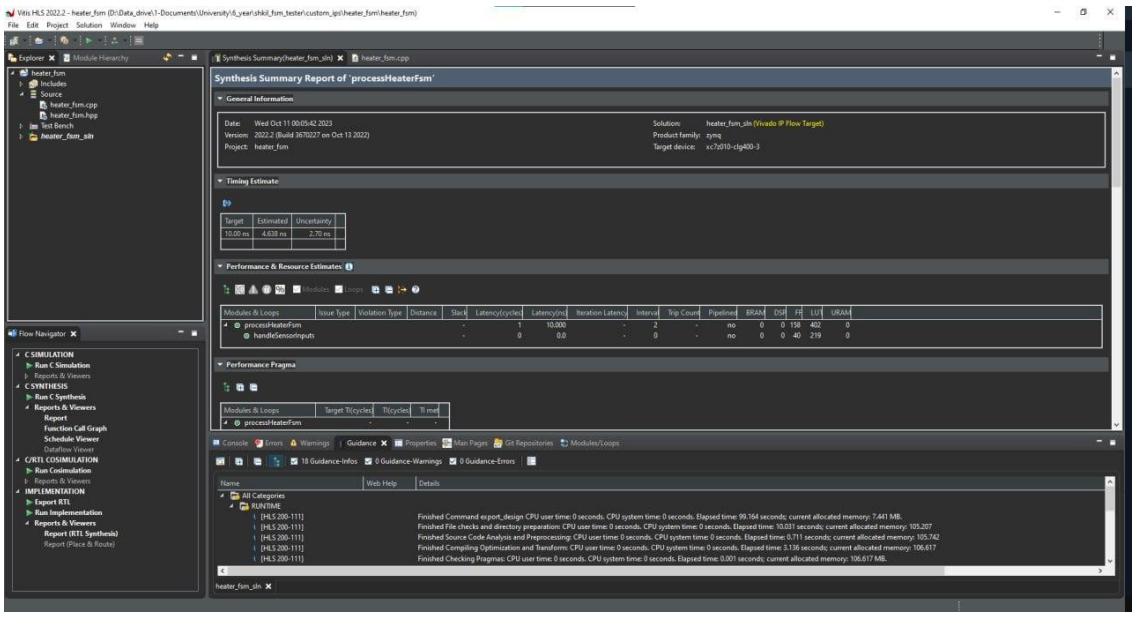

Рисунок 2.11 – Результати виконання HLS Design flow та утилізація ресурсів XC7Z010CLG400 у середовищі Vitis HLS.

Після експортування IP блоку до середовища Vivado та стандартний

процес додавання блоків до проекту виконується додавання з'єднань до AXI та зовнішніх інтерфейсів і генерація SDK і драйверів для користувацьких пристроїв AXI. У поточному дизайні для повного обходу графу станів керуючого автомату було згенеровано API для читання-запису вхідних сигналів за допомогою AXI та отримання сигналів керування за цією шиною. З боку testbench server було додано перевірку обхіду станів після тестової взаємодії з FSM та перевірки очікуваних сигналів керування. У рамках поточної ітерації testbench server реалізовано як bare-metal застосунок для ARM-частини ZYNQ. Розглянемо фрагмент сгенерованого SDK для PS частини та перевірку очікуваних станів після тестової взаємодії, де виконується звертання по AXI шині до користувацького блоку (PL).

Лістинг 2.7 – Фрагмент згенерованого SDK для взаємодії з користувацьким блоком AXI шини

```
void XProcessheaterfsm_Set_sensorIn(XProcessheaterfsm *InstancePtr, u32 Data) {
    Xil_AssertVoid(InstancePtr != NULL);
    Xil_AssertVoid(InstancePtr->IsReady == XIL_COMPONENT_IS_READY);

    XProcessheaterfsm_WriteReg(InstancePtr->Control_BaseAddress,
XPROCESSHEATERFSM_CONTROL_ADDR_SENSORIN_DATA, Data);
}

u32 XProcessheaterfsm_Get_sensorIn(XProcessheaterfsm *InstancePtr) {
    u32 Data;
    Xil_AssertNonvoid(InstancePtr != NULL);
    Xil_AssertNonvoid(InstancePtr->IsReady == XIL_COMPONENT_IS_READY);
    Data = XProcessheaterfsm_ReadReg(InstancePtr->Control_BaseAddress,
XPROCESSHEATERFSM_CONTROL_ADDR_SENSORIN_DATA);
    return Data;
}

u32 XProcessheaterfsm_Get_controlOut(XProcessheaterfsm *InstancePtr) {
    u32 Data;
    Xil_AssertNonvoid(InstancePtr != NULL);
    Xil_AssertNonvoid(InstancePtr->IsReady == XIL_COMPONENT_IS_READY);
    Data = XProcessheaterfsm_ReadReg(InstancePtr->Control_BaseAddress,
XPROCESSHEATERFSM_CONTROL_ADDR_CONTROLOUT_DATA);
    return Data;
}
```

Реалізація testbench серверу представляє собою baremetal застосунок, зроблений з використанням Xilinx SDK. Типовою є перевірка вихідних

сигналів FSM за допомогою вбудованого функціоналу мов C/C++, а саме assert, що дозволяє під час розробки та налагодження мати можливість отримувати результат проходження тестів. Зазначимо, що технічно можливим є розгортання testbench server у вигляді Linux userspace застосунку та використанням вже існуючих фреймворків тестування у вигляді GTest/CppUnit для отримання докладного звіту по проходженню тесту і процентного відношення до повного тестового покриття.

3 ПРОЕКТУВАННЯ ТА САМОТЕСТУВАННЯ ПРИСТРОЮ в SoC

3.1 Пристрій логічного керування

Реалізацію розроблених процедур самодіагностування пристроїв логічного керування, реалізованих в SoC розглянемо на прикладі пристрою керування тепловентилятором.

Об’єктом проектування є система управління тепловентилятором. Об’єктом управління є тепловентилятор, алгоритм роботи якого наступний. Спочатку тепловентилятор знаходиться у режимі очікування. Після включення режиму нагріву (сигнал Onn) починається розгін вентилятора до необхідної швидкості протягом часу T_1. Після цього включається нагрівальний елемент та виробляється тепле повітря. Через проміжок часу T_2 відбувається опитування датчика температури та, якщо температура не досягла критичного рівня, продовжується нагрів. Якщо температура досягла критичного рівня $t_к$, тепловентилятор переходить в режим охолодження (працює тільки вентилятор) та відбувається опитування датчика температури через проміжок часу T_3. При цьому слід враховувати, що $T_3 >> T_2$, що обумовлено значно меншою швидкістю охолодження, ніж при нагріванні. Якщо температура знизилася, знов включається нагрівальний елемент та продовжує вироблятися тепле повітря. В такому режимі тепло вентилятор працює доти, поки включений режим нагріву.

При виключенні режиму нагріву у будь-який момент часу, тепловентилятор переходить у режим остаточного (повного) охолодження, працює тільки вентилятор протягом часу T_4, і після цього тепловнтилятор остаточно вимикається та переходить у режим очікування.

На рис.3.1 наведена структура системи автоматичного управління тепловентилятором, де КА – керуючий автомат, ТВ – тепловентилятор, ДТ – аналогово-цифровий датчик температури, $t_к$ – критична температура,

tm – сигнал перевищення критичної температури.

При цьому слід враховувати наступне. У відповідності до принципів проектування пристроїв керування виконавчі пристрої, датчики, сенсори та перетворювачі відносяться до об’єкту управління, а при верифікації моделі та налагодженні прототипу керуючого автомату (при проведенні ДЕ) їх вихідні сигнали імітуються у алфавіті {0, 1}.

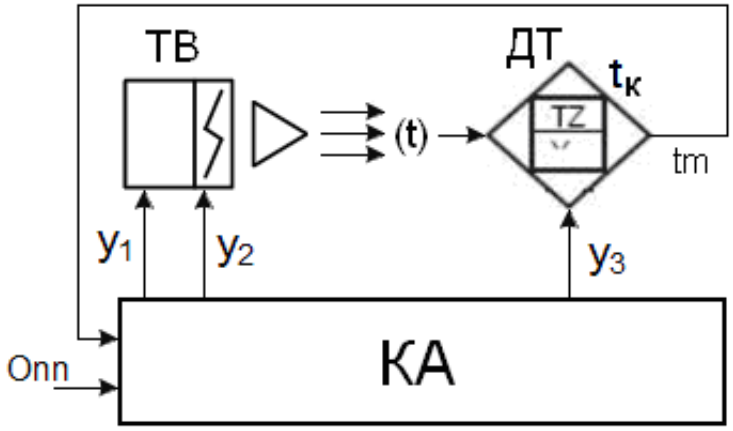

Рисунок 3.1 – Система автоматичного управління тепловентилятором

У відповідності до методики проектування автоматних систем логічного управління визначимо технічні стани роботи тепловентилятора.

1. Тепловентилятор виключений (режим очікування).
2. Режим включення живлення та розгону вентилятора.
3. Режим нагріву та подачі теплого повітря.
4. Режим охолодження нагрівального елементу.
5. Режим повного охолодження нагрівального елементу.

Визначимо вхідні та вихідні сигнали для тепловентилятора, а також часові параметри його роботи:

Onn = {0, 1} – вхідний сигнал включення живлення тепловентилятора;

St = {0, 1} – вхідний сигнал включення режиму нагріву;

y_1 = {0, 1} – вихідний сигнал включення вентилятора;

$y_2 = \{0, 1\}$ – вихідний сигнал включення нагрівального елементу;

$y_3 = \{0, 1\}$ – вихідний сигнал опитування датчика температури;

tm = {0, 1} – вихідний сигнал датчика температури, вхідний для керуючого автомата (tm = 1 при $t > t_к$, tm = 0 при $t \leq t_к$, де t – температура теплого повітря, $t_к$, – критична температура теплого повітря, вище якої нагрів не здійснюється);

T_1=3 сек – проміжок часу для розгону вентилятора до необхідної швидкості;

T_2=0,5 сек – проміжок часу між опитуваннями датчика температури при нагріві;

T_3=5 сек – проміжок часу між опитуваннями датчика температури при охолодженні;

T_4=20 сек – проміжок часу для повного охолодження нагрівального елементу.

Визначимо стани керуючого автомата. Виходячи з того, що всі технічні стани тепловентилятора є стійкими, аварійні стани не розглядаються, в розщепленні технічних станів немає потреби, кожному з технічних станів відповідає стан керуючого автомата $\{a_0, a_1, a_2, a_3, a_4\}$. Перелічимо стани керуючого автомата та відповідні вхідні та вихідні сигнали у кожному стані

1. Стан a_0 (тепловентилятор виключений), Onn=0, $y_1 = y_2 = y_3 = 0$.

2. Стан a_1 (режим включення та розгону вентилятора), Onn=1, tm=0. y_1=1, затримка T_1.

3. Стан a_2 (режим нагріву та подачі теплого повітря), Onn=1, $y_1 = y_2 = y_3 = 1$, затримка опитування T_2.

4. Стан a_3 (режим охолодження нагрівального елементу), Onn=1, $y_1 = y_3 = 1$, $y_2 = 0$, затримка T_3 .

5. Стан a_4 (режим повного охолодження нагрівального елементу), Onn=0, $y_1 = 1$, $y_2 = y_3 = 0$, затримка T_4 .

Для реалізації керуючого автомата (КА) будемо використовувати модель часового автомата Мура без вхідних обмежень та вихідних затримок,

таймаути автомата Мура будуть реалізовані шляхом реалізації затримок у станах автомата Мура. Граф переходів КА наведений на рис. 3.2.

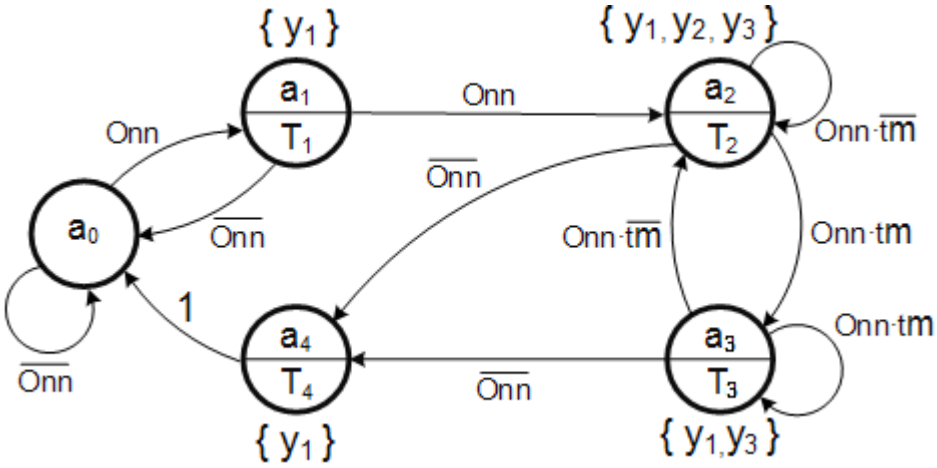

Рисунок 3.2 – Граф переходів КА тепловентилятора

Як було зазначено у підрозділі 2.4 основною перевагою розробки з використанням Vitis HLS є можливість виконувати реалізацію алгоритму на високорівневих мовах програмування C/C++ з подальшим перетворення коду у HDL-модель, яка використовується для синтезу автомата у блоку PL. На основі наведеного на рис. 3.2 графа переходів КА може бути побудована модель керуючого автомата на мові програмування С у стилі автоматного шаблону.

Повна реалізація моделі керуючого FSM на мові програмування С для подальшого синтезу з використанням Vitis HLS наведена у Додатку Б, а на лістингу 3.1 показана частина моделі, яка демонструє використання автоматного шаблону на мові програмування С [12]. Модель на мові програмування С інструментальними засобами САПР Vitis HLS перетворюється в VHDL-модель КА та модель взаємодії PL та PS через AXI шину, які наведені у додатку В.

Лістинг 3.1 – Фрагмент реалізації моделі FSM IP на мові програмування С

```
enum class FsmStates { A0, A1, A2, A3, A4};
```

```
class HeaterFsm {
public:
        void handleSensorInputs(int sensorIn, bool extClock) {
                const bool isRisingEdge = (extClock == true)
                                && (m_prevClockSignal == false);
                m_prevClockSignal = extClock;
                if (isRisingEdge) { m_currentState = m_nextState; }
. . .
                switch (m_currentState) {
                case FsmStates::A0:
                        if (sensorIn & OnnSignal) { m_nextState = FsmStates::A1;}
                        else { m_nextState = FsmStates::A0; m_counter = 0; }
                        break;
                case FsmStates::A1:
                        if (m_counter < T1 - 1)
                                { ++m_counter; m_nextState = FsmStates::A1;}
                        else if (!(sensorIn & OnnSignal))
                                {m_nextState = FsmStates::A0; }
                        else { m_counter = 0; m_nextState = FsmStates::A2; }
                        break;

        void processControlSignals() {
                switch (m_currentState) {
                case FsmStates::A0: m_controlOutputs = 0;
                        break;
                case FsmStates::A1: m_controlOutputs = Y1;
                        break;
                case FsmStates::A2: m_controlOutputs = Y1 | Y2 | Y3;
                        break;
                case FsmStates::A3: m_controlOutputs = Y1 | Y3;
                        break;
                case FsmStates::A4: m_controlOutputs = Y2;
                        break;
                }
        }
```

Як згадувалося у підрозділі 2.4 керуючий автомат (КА) реалізований у блоку програмованої логіки (PL) кристала Zynq-7000, а операційний автомат (ОА) – у процесорній системі (PS) (рис. 3.3). В операційному автоматі функціонує програма на мові С, яка зв’язує КА з зовнішнім середовищем. ОА приймає та нормує сигнал включення Onn, а також вихідний сигнал від датчика температури t та перетворює його у вхідний с сигнал для керуючого автомата tm = {0, 1} з урахуванням критичної температури $t_к$. ОА взаємодіє з постійною пам’яттю, де зберігаються часові і температурні константи {T_1, T_2,

T_3, T_4, $t_к$}. Сигнали Onn та tm передаються до блоку PL через AXI шину. В свою чергу ОА отримує сигнал опитування датчика температури (y3) та передає його безпосередньо до датчика температури (ДТ). Керуючий автомат прошивається у програмовану логіку (FPGA) та реалізує алгоритм управління тепловентилятором (рис. 3.2). Вихідні сигнали управління y1 та y2 через блок ввода/вивода передаються до вентилятора та нагрівального елемента.

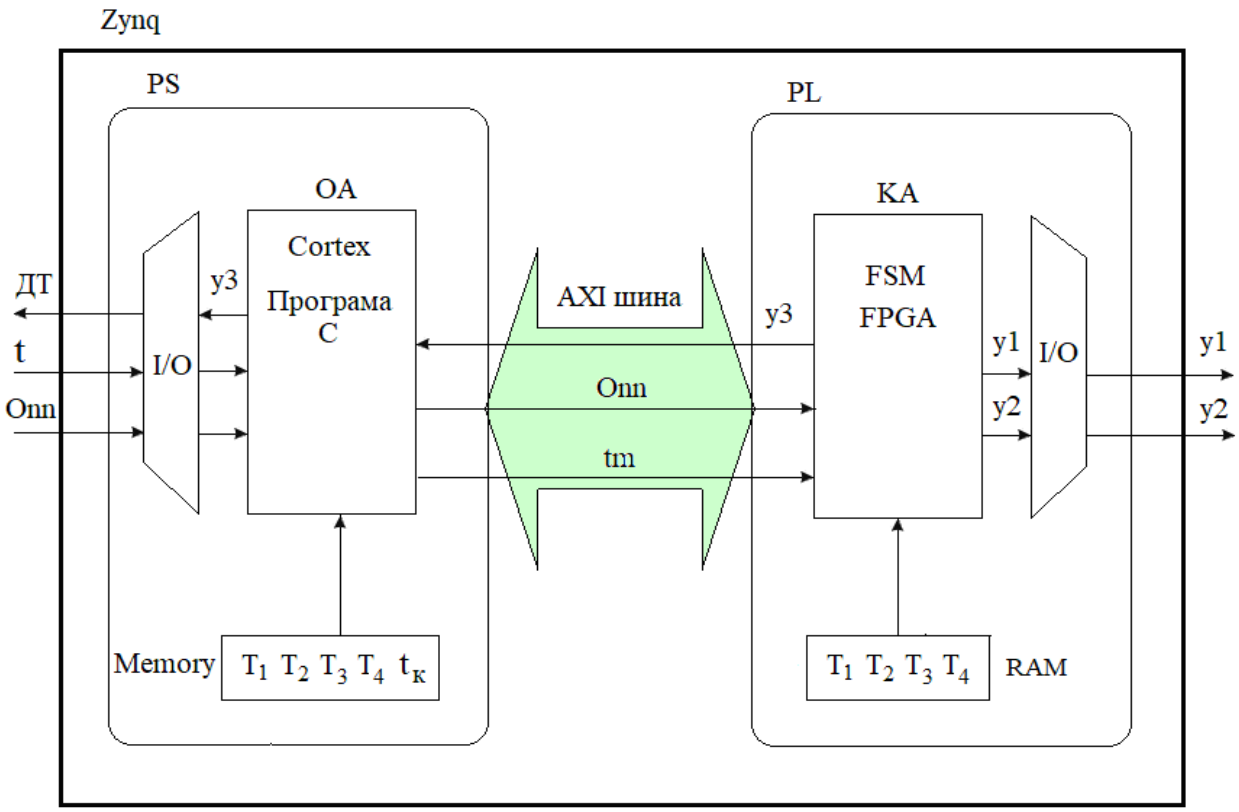

Рисунок 3.3 – Реалізація пристрою керування в SoC Zynq 7000

3.2 Програмне та апаратне забезпечення діагностичного експерименту

У випадку наявності графової моделі Мура для автоматного пристрою керування функціональне покриття специфікації може бути зведено до обходу всіх станів автомата Мура та реалізації усіх переходів у кожному стані. При проведенні верифікації автоматних моделей реалізується так званий «неруйнівний» діагностичний експеримент (ДЕ) при проведенні якого

тест (обхід графа) починається з початкової вершини графа переходів і повертається до неї. Такий ДЕ гарантує досяжність всіх станів керуючого автомата та реалізацію всіх переходів у кожному стані [11]. На основі графа переходів (рис. 3.2) будується дерево рішень (рис. 3.4), а по ньому – множина шляхів обходу графа переходів. В таблиці 3.1 наведений перелік всіх шляхів обходу графа переходів, які реалізують неруйнівний діагностичний експеримент.

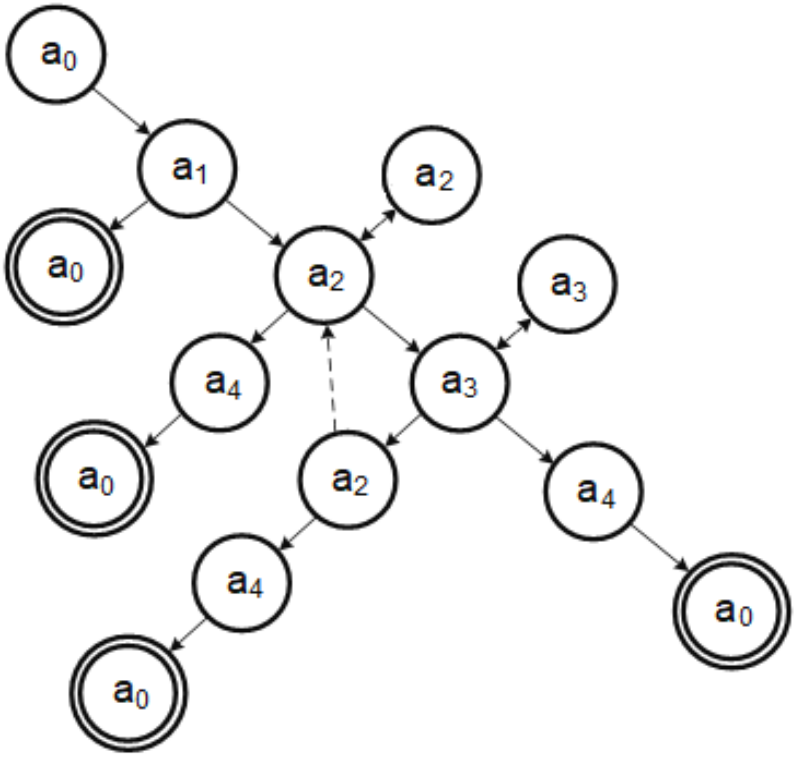

Рисунок 3.4 – Дерево рішень для побудови неруйнівного ДЕ

Таблиця 3.1 – Шляхи обходу графа переходів

$a_0 - a_1 - a_0$
$a_0 - a_1 - a_2 - a_2 - a_4 - a_0$
$a_0 - a_1 - a_2 - a_3 - a_3 - a_2 - a_4 - a_0$
$a_0 - a_1 - a_2 - a_3 - a_4 - a_0$

При організації ДЕ по самодіагностиці пристрою керування на SoC в якості тестера використовується ОА, який імітує сигнали Onn та tm. Сигнал tm імітується за рахунок імітації зовнішньої температури t з урахуванням часових параметрів T_i. Еталони зберігаються у пам'яті PS у вигляді значень y1, y2, y3 у відповідні проміжки часу, а проміжки часу у вигляді констант T_i. теж тех. Зберігаються у пам'яті PS. Шляхи обходу графа переходів зберігаються у пам'яті PS і реалізуються шляхом завдання параметру t у відповідні проміжки часу.

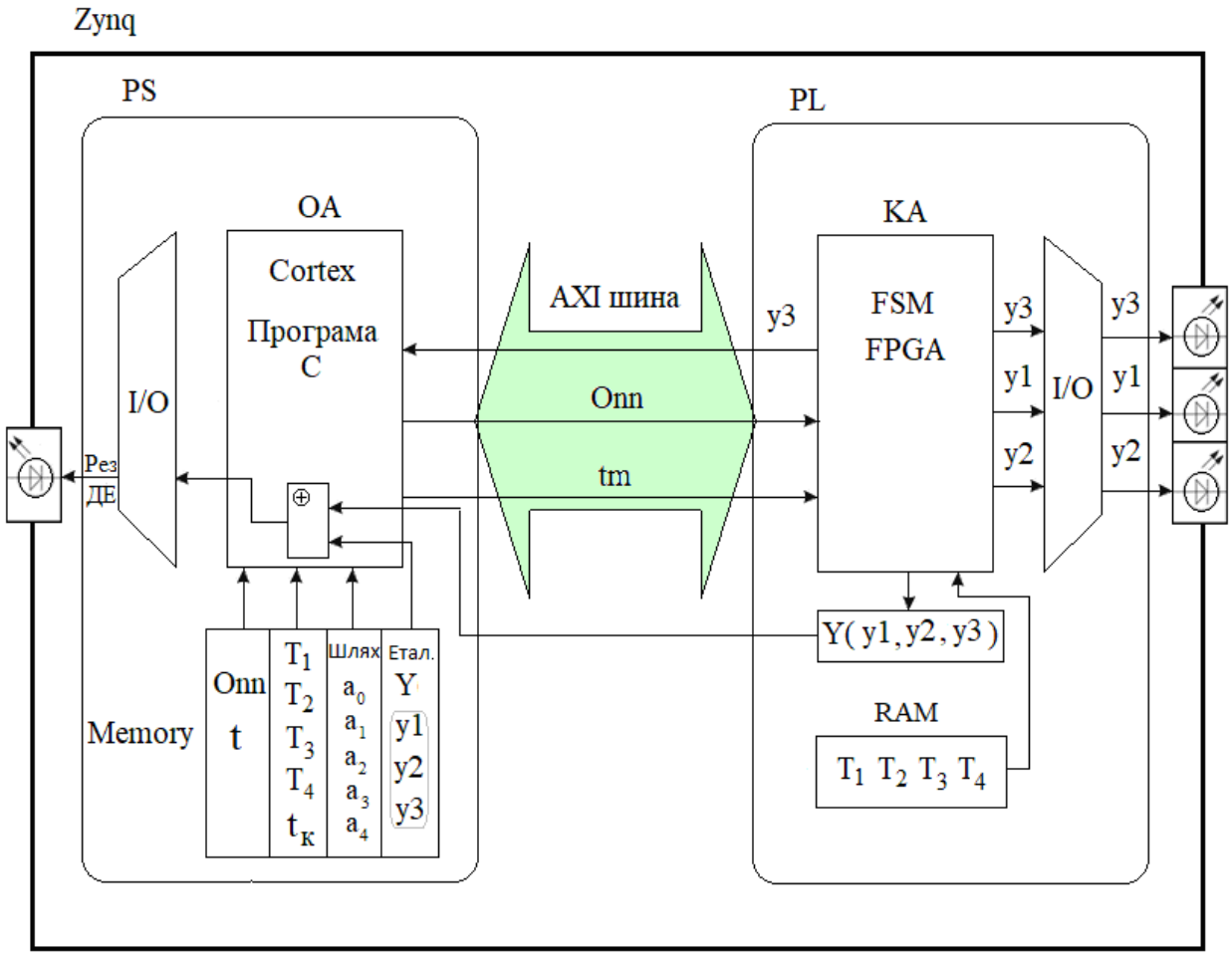

Рисунок 3.5 – Реалізація ДЕ пристрою керування для SoC Zynq

Керуючий автомат виробляє вихідні сигнали y1, y2, y3, формує з них бінарний вектор та передає його до ОА. Там цей вектор порівнюється з еталоном і результат порівняння через блок вводу/виводу передається на контрольний світлодіод для візуального спостереження. Крім того сигнали

y1, y2, y3 безпосередньо з КА через блок вводу/виводу PL передаються на світлодіодну панель вихідних сигналів для візуального спостереження.

Операційний автомат в режимі проведення ДЕ імітує видачу вхідних сигналів для КА. Фрагмент коду моделі ОА на мові програмування С для проходження станів КА наведено у лістингу 3.2. де використано автоматично згенерований bare-metal драйвер для AXI периферії.

Лістинг 3.2 – Фрагмент обходу графу переходів FSM з перевірками вихідних сигналів керування з використанням Xilinx SDK

```
void toggle_clock(XProcessheaterfsm* pInstance) {
	assert(pInstance != NULL);
	XProcessheaterfsm_Set_extClock(&instance, 0x00);
	XProcessheaterfsm_Set_extClock(&instance, 0x01);
}

void perform_a0_a1_transition(XProcessheaterfsm* pInstance) {
	uint32_t fsm_outputs = XProcessheaterfsm_Get_controlOut(&instance);
	XProcessheaterfsm_Set_sensorIn(&instance, OnnSignal);
	toggle_clock(&instance);
	fsm_outputs = XProcessheaterfsm_Get_controlOut(&instance);
	assert(fsm_outputs == Y1);
}

void perform_a1_a2_transition(XProcessheaterfsm* pInstance) {
	uint32_t fsm_outputs = XProcessheaterfsm_Get_controlOut(&instance);
	XProcessheaterfsm_Set_sensorIn(&instance, OnnSignal);
	for (int i = 0; i < T1; ++i) {
		toggle_clock(pInstance);
	}
	fsm_outputs = XProcessheaterfsm_Get_controlOut(&instance);
	assert(fsm_outputs == (Y1 | Y2 | Y3));
}
```

Одним з варіантів розширення поточної реалізації є використання вбудованого IP блоку Block Memory Generator та збереження очікуваних станів вихідних сигналів одразу у ROM частині з подальшим зчитуванням з боку ARM-baremetal testbench серверу. Для обміну з блоком ROM є можливим використання AXI BRAM Controller IP блоку.

3.3 Самотестування пристрою логічного керування на SoC у реальному часі

Діагностичний експеримент виконано на базі апаратної платформи EBAZ4205 з модулем розширення. Ця налагоджувальна плата є одним з представників сімейства ZedBoard На платі налагодження змонтований ZYNQ7010 з підключеними інтерфейсами програмування та зовнішньою периферією. Процес прошивки testbench серверу виконується за допомогою JTAG-адаптеру з середовища Vitis. На рисунку 3.6 зображено початок діагностичного експерименту з підключенням до ARM частини та конфігурацією FPGA відповідним бітстримом.

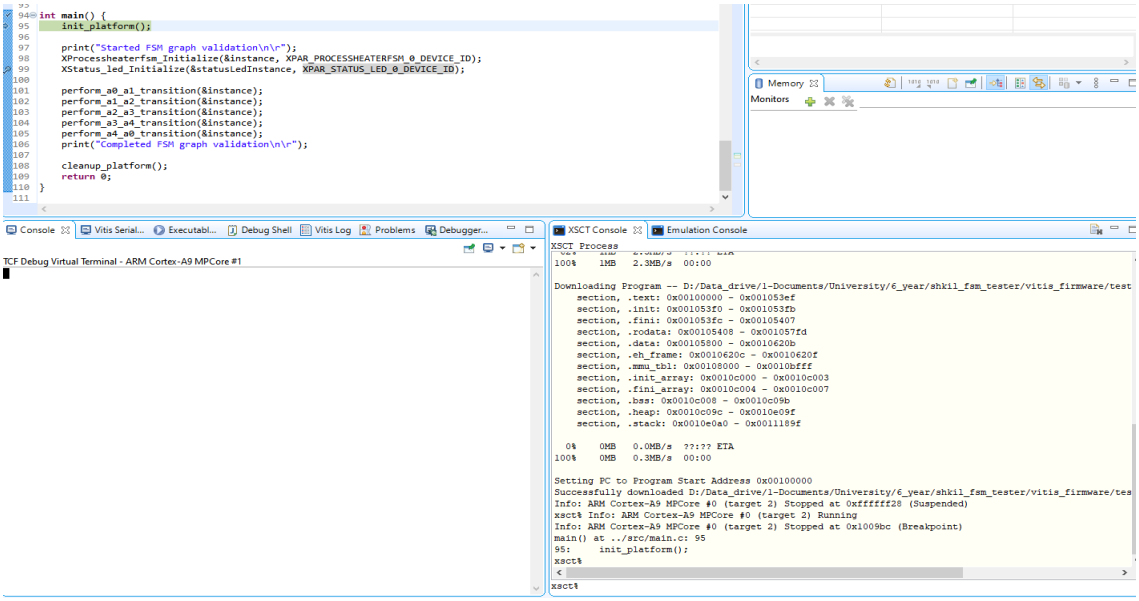

Рисунок 3.6 – Початок діагностичного експерименту на платформі EBAZ4205

Продемонструємо проведення ДЕ по обходу графа переходів керуючого автомата в основному режимі роботи $a_0 - a_1 - a_2 - a_3 - a_4 - a_0$. Перелічимо режими роботи тепловентилятора, які верифікуються: режим очікування (a_0) – режим розгону (a_1) – режим нагріву (a_2) – режим охолодження (a_3) – режим остаточного охолодження (a_4) – режим очікування(a_0).

Початок діагностичного експерименту та завершення супроводжуються відповідними повідомленнями у UART-консоль яка є доступною у Vitis Serial Monitor. Візуалізації станів виходів автомату виконується за допомогою виведення y_i на світлодіоди, які під'єднані до блоку PL. Перехід між станами здійснюється за допомогою подання відповідних сигналів через згенерований API до AXI-шини. На рис. 3.7 наведена візуалізація вихідних сигналів КА у різних станах на платі налагоджувальній платі ZebBoard (EBAZ4205).

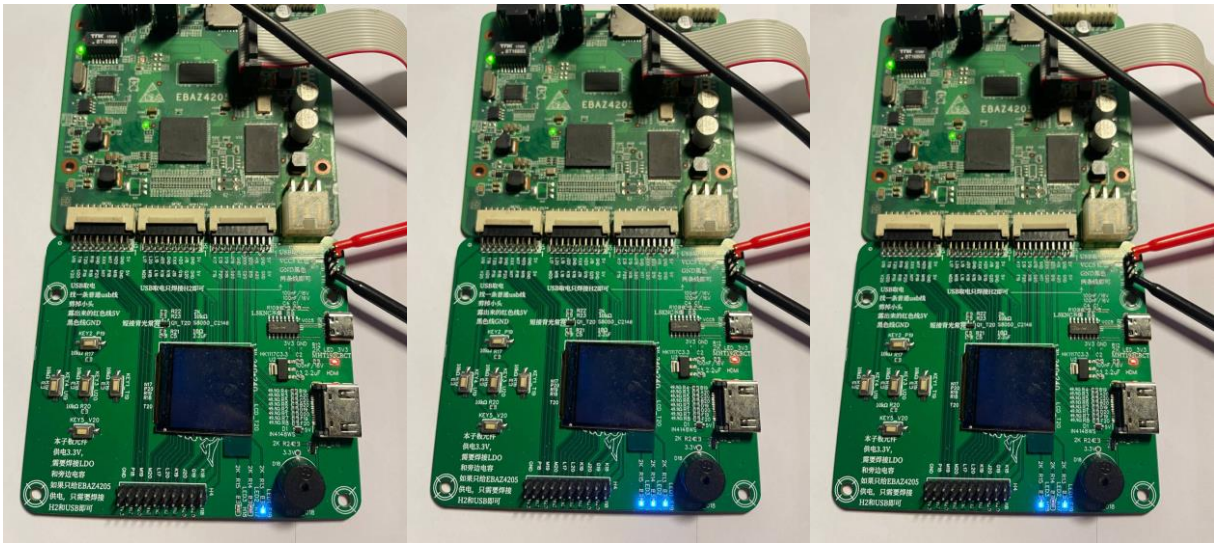

Рисунок 3.7 – Візуалізація вихідних сигналів у станах a_1, a_2, a_3 на платі EBAZ4205 під час проведення діагностичного експерименту

Для коректної візуалізації обхід графу станів реалізовано з затримкою 1 секунда між подачею тестових сигналів. З метою перевірки коректності транзакцій станів вихідних сигналів було використано Integrated Logic Analyzer IP-ядро з підключеними сигналами до AXI шини. Integrated Logic Analyzer (ILA) є вбудованим інструментом Vivado IDE який представлений у вигляді синтезованого IP-ядра та окремого інструменту який стає доступним після конфігурації FPGA. Перед запуском чипу є необхідним додати сигнали, що мають бути проаналізовані до відповідних портів аналізатору. Вигляд на діаграмі системи ILA наведено на рис. 3.8 разом з прикладом шин,

що підключені для діагностування.

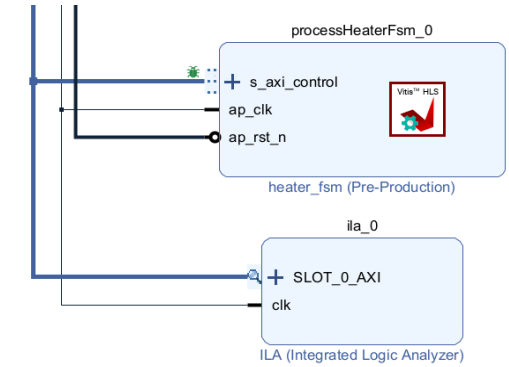

Рисунок 3.8 – Вигляд вбудованого логічного аналізатора Integrated Logic Analyzer IP з під'єднанням до AXI шини блоку

З метою перевірки було проведено другий етап неруйнівного експерименту з діагностуванням обміну між PS та PL частиною для обраних IP блоків. На рисунках 3.9 та 3.10 наведено перехід автомату до стану A1 з виставленням сигналу y1 та перехід зі стану A1 до стану A2 відповідно

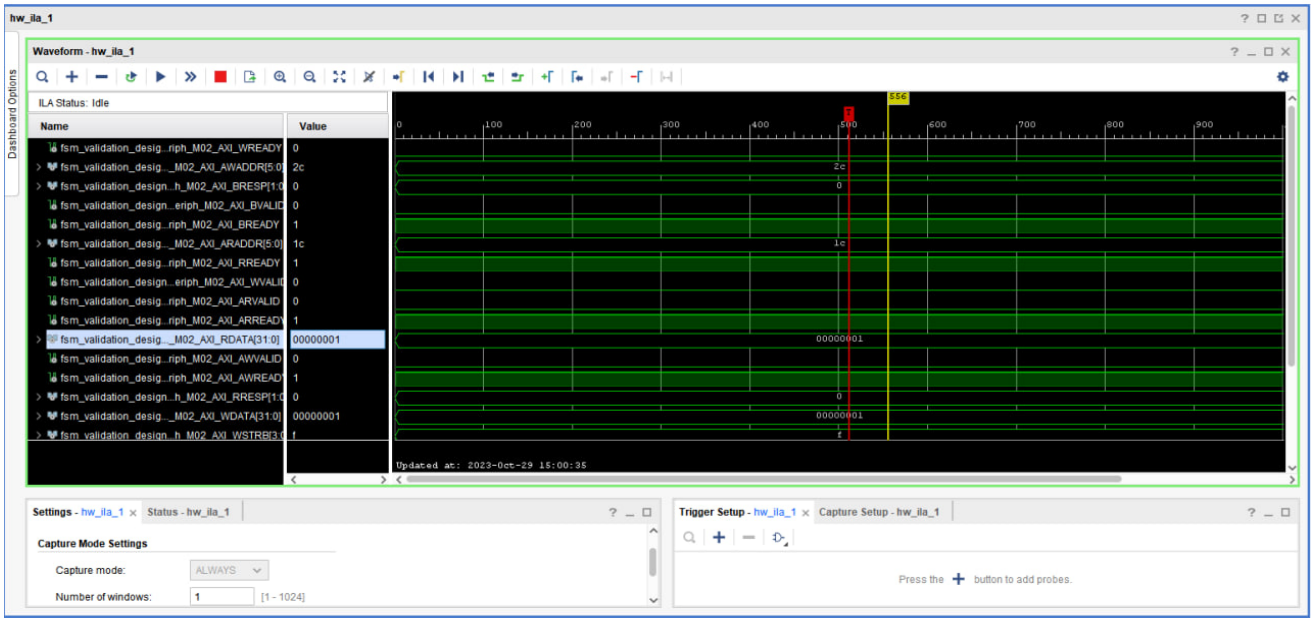

Рисунок 3.9 – Фіксація обміну по AXI шині після переходу автомату зі стану A0 до стану A1

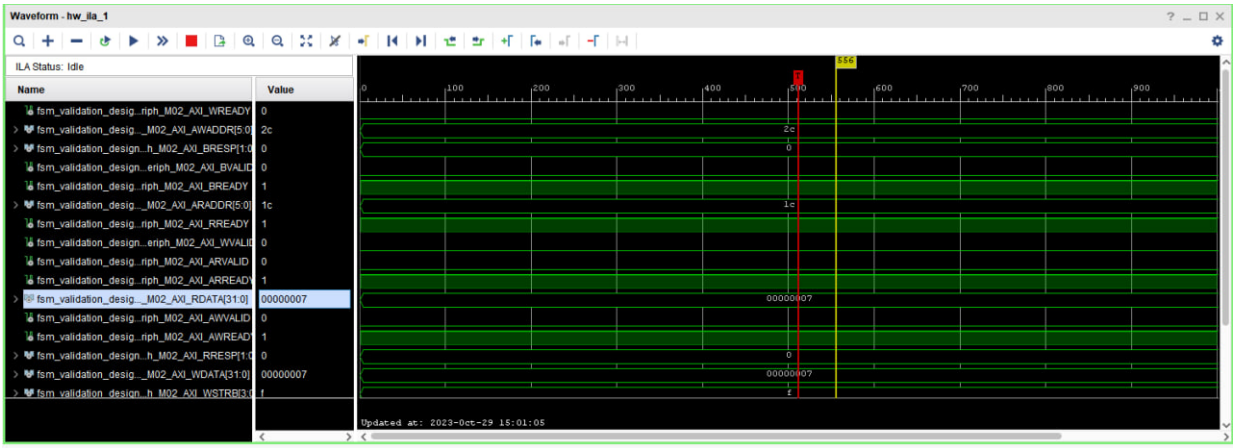

Рисунок 3.10 – Фіксація обміну по AXI шині після переходу автомату зі стану A1 до стану A2

Верифікація коректності отриманих значень під час діагностики виконується на стороні testbench серверу за допомогою вбудованого макросу мов C\C++ assert. Таким чином кожна взаємодія автоматично перевіряє отриманий стан виходів після подання тестових сигналів. У Лістингу 3.3 наведено фрагмент обходу для переходу між станами A2 та A3. Встановлення статусних світлодіодів виконується наприкінці ДЕ.

Лістинг 3.3 – Фрагмент тестової взаємодії з використанням AXI шини та макросу assert для перевірки коректного встановлення вихідних сигналів КА

```
void perform_a2_a3_transition(XProcessheaterfsm* pInstance) {
        uint32_t fsm_outputs = XProcessheaterfsm_Get_controlOut(&instance);
        XProcessheaterfsm_Set_sensorIn(&instance, OnSignal);
        toggle_clock(pInstance);
        assert(fsm_outputs == (Y1 | Y2 | Y3));

        XProcessheaterfsm_Set_sensorIn(&instance, OnSignal | Tm);
        toggle_clock(pInstance);
         fsm_outputs = XProcessheaterfsm_Get_controlOut(&instance);
        assert(fsm_outputs == (Y1 | Y3));
        XStatus_led_Set_status_led_value(&statusLedInstance, fsm_outputs);
```

Отримані результати показали коректність представленої моделі та методики організації самотестування.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи розглянуті питання побудови системи самодіагностики кіберфізичної системи логічного управління, реалізованої на технологічній платформі системи на кристалі.

Проведений аналіз принципів побудови вбудованих кіберфізичних систем, які реалізуються в системах на кристалі. Розглянуті методи та маршрути проектування систем на кристалі, які складаються з програмної та апаратної частини. Розглянуті принципи побудови систем верифікації та вбудованої самодіагностики SoC.

В якості технологічної платформи обрано програмовану систему на кристалі FPGA сімейства ZYNQ-7000 фірми Xilinx Inc. Платформа Xilinx ZYNQ-7000 має архітектуру, де поєднано PS (processing system) – систему обробки та PL (programmable logic) - програмовану логіку. Взаємодія двох систем використовується через інтерфейс AMBA на основі протоколу AXI. В якості налагоджувальної плати використана недорога загальнодоступна плата ZedBoard, яка оснащена пристроєм XC7Z020 Zynq.

Сформульована загальна послідовність етапів проектування системи на кристалі на платформі ZYNQ-7000. Практична реалізація виконана на базі стеку інструментальних засобів САПР Vivado/Vitis/Vitis HLS. Це дозволяє виконати наскрізне проектування як частини, що синтезується з боку Programmable Logic (PL), так і користувацького програмного забезпечення з боку Processing System (PS) на мові програмування C.

Розробленні методи апробовані на моделі пристрою логічного керування тепловентилятором. В якості моделі пристрою керування розглядається часовий автомат Мура, а його описом є темпоральній граф переходів. При реалізації на платформі ZYNQ-7000 керуючий автомат реалізований в блоку PL, а операційний автомат в блоку PS.

При організації процесу самодіагностики реалізується неруйнівний діагностичний експеримент шляхом обходу всіх дуг графа переходів починаючи з початкової вершини. В якості тестера при цьому виступає операційний автомат, еталонні логічні та часові значення зберігаються пам’яті блоку PS. Візуальне спостереження за виконанням ДЕ здійснюється на панелі світлодіодів плати ZedBoard.

Наукова новизна роботи полягає у подальшому розвитку методів самодіагностики автоматних пристроїв логічного керування, реалізованих на технологічній платформі систем на кристалі

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Быковский С.В. Сопряжённое проектирование встраиваемых систем (Hardware/Software Co-Design). Часть 1: Учебное пособие / С.В. Быковский, Я.Г. Горбачев, А.О. Ключев, А.В. Пенской, А.Е. Платунов. – СПб: Университет ИТМО, 2016. – 108 с.

2. Greaves, D. Modern System-on-Chip Design on Arm / David Greaves. – Arm Education Media, 2021. – 608 p. ISBN 978-1-911531-37-1.

3. Быковский С.В. Сопряжённое проектирование встраиваемых систем (Hardware/Software Co-Design). Часть 2: Учебное пособие / С.В. Быковский, Я.Г. Горбачев, А.О. Ключев, А.В. Пенской, А.Е. Платунов. – СПб: Университет ИТМО, 2016. – 105 с.

4. Bailey B., Martin G., Piziali A. ESL Design and Verification // ESL Des. Verif., 2007. – 113-138 p.

5. Мірошник М.А. Автоматизація проектування вбудованих систем і програмних засобів на ПЛІС мовою опису апаратури: навч. посібник / М.А. Мірошник, М.С. Курцев. – Харків: УкрДУЗТ, 2021. – 331 с.

6. The Zynq Book. Embedded Processing with the ARM® Cortex®-A9 on the Xilinx® Zynq®-7000 All Programmable SoC [Electronic resource] – Zynqbook.com – Mode of access: http://www.zynqbook.com/ – Date of access: 17.09.2023.

7. AXI Basics 1 - Introduction to AXI [Electronic resource] – Adaptive SoC & FPGA Support – Mode of access: https://support.xilinx.com/s/article/1053914? – Date of access: 10.09.2023.

8. Палагин А.В. Особенности проектирования компьютерных систем на кристалле ПЛИС / А.В. Палагин, Ю.С. Яковлев // Математичні машини і системи. – 2017. – № 2. – С. 3-14. ISSN 1028-9763.

9. Advanced Micro Devices - Vitis HLS [Electronic resource] – AMD Xilinx – Mode of access: https://www.xilinx.com/products/design-tools/vitis/vitis-hls.html – Date of access: 20.09.2023.

10.Шкіль О. Проєктування та самодіагностика кіберфізичних пристроїв керування на платформі SoC / О. Шкіль, Д. Рахліс, І. Філіпенко, В. Корнієнко // Сучасний стан наукових досліджень та технологій в промисловості. – 2023. – № 4 (26). – С. 122-134.

11.Шкиль, А.С. Обнаружение ошибок проектирования в HDL-моделях конечных автоматов с использованием синхронизирующих последовательностей / А.С. Шкиль, М.А. Мирошник, Э.Н. Кулак А.С. Гребенюк, Д.Е. Кучеренко // Радиоэлектроника и информатика. – 2016. – № 3(74). – С. 39-46.

12.Шалыто А. А. Автоматное программирование / Н.И. Поликарпова, А.А. Шалыто. – Спб.: Питер, 2008. – 168 с.