ISSN 2522-9818 (print)
ISSN 2524-2296 (online) *Innovative technologies and scientific solutions for industries. 2023. No. 4 (26)*

УДК 681.326 DOI: https://doi.org/10.30837/ITSSI.2023.26.122

О. Шкіль, Д. Рахліс, І. Філіпенко, В. Корнієнко

ПРОЄКТУВАННЯ ТА САМОДІАГНОСТИКА КІБЕРФІЗИЧНИХ ПРИСТРОЇВ КЕРУВАННЯ НА ПЛАТФОРМІ *SoC*

Предметом дослідження в статті є моделі, методи та процедури проєктування та самодіагностики автоматних моделей пристроїв логічного керування, реалізованих в *SoC*. **Об'єкт роботи** – процедури автоматизованого проєктування та діагностування цифрових пристроїв на технологічній платформі *SoC*. **Метою дослідження** є розроблення моделей і процедур проєктування та самотестування в циклі автоматизованого проєктування автоматних систем логічного управління на технологічній платформі *SoC*, що суттєво підвищить надійність їх функціювання. У статті вирішуються такі **завдання**: розгляд процедур взаємодії процесорного ядра з програмованою логікою у складі *SoC*; удосконалення процедур проєктування та тестування програмно-апаратних систем на основі *SoC*; подальший розвиток процедур автоматизованого проєктування, верифікації та діагностування кіберфізичних систем логічного управління з використанням мов програмування та мов опису апаратури; реалізація процедури апаратного самотестування керуючих автоматів на технологічній платформі *SoC*. Упроваджуються такі **методи**: синтез керуючих автоматів на основі графових моделей, імплементація моделей керуючих автоматів мовою програмування *C* з використанням автоматного шаблону, діагностичний експеримент способом обходу графа переходів автомата. **Досягнуті результати.** На основі аналізу процедур взаємодії процесорного ядра та програмованої логіки на обраній платформі *SoC* спроєктовано модель кіберфізичної системи логічного управління. Практичну реалізацію виконано на базі стеку інструментальних засобів САПР *Vivado/Vitis/Vitis HLS*. Реалізовано метод апаратного самотестування керуючих автоматів на технологічній платформі *SoC ZYNQ-7000*. **Висновки**. У статті проаналізовано принципи проєктування вбудованих кіберфізичних систем, що реалізуються в системах на кристалі. Розглянуто принципи побудови систем верифікації та вбудованої самодіагностики систем на кристалі, що містять програмну й апаратну частини. Розроблені методи апробовано на моделі пристрою логічного керування світлофором на технологічній платформі *SoC FPGA* сімейства *ZYNQ-7000* фірми *Xilinx*. Керуючий автомат Мура реалізовано у блоці *PL* мовою програмування *C*, а операційний автомат – у блоці *PS*. Під час організації процесу самодіагностики здійснено неруйнівний діагностичний експеримент способом обходу всіх дуг графа переходів, починаючи з початкової вершини. Тестером у цьому разі був операційний автомат, еталонні логічні та часові значення якого зберігалися в пам'яті блока *PS*. Візуальне спостереження за виконанням діагностичного експерименту здійснювалося за допомогою панелі світлодіодів плати *ZedBoard*.

Ключові слова: кіберфізичні системи; вбудовані системи; логічне управління; проєктування систем на кристалі; *FPGA*; САПР; самотестування *SoC*; мова програмування *C*.

Вступ

Вбудовані цифрові пристрої та системи за останні 50 років зазнали значних змін не тільки в технологічному, а й в архітектурному аспекті. Якщо впродовж 60–70 років минулого століття вбудованою цифровою системою було прийнято називати спеціалізовані пристрої керування, спроєктовані на логічних інтегральних схемах малого й середнього ступеня інтеграції, то сучасне розуміння вбудованої системи визначає її у вигляді замовної надвеликої інтегральної схеми (НВІС), що за своєю сутністю є спеціалізованою обчислювальною системою, технологічно виготовленою у вигляді системи на кристалі. Останні роки з'явилося розуміння, що більшість проблем вбудованих систем є наслідком складнощів в умовах взаємодії з фізичними процесами, а не через обмеженість ресурсів. У 2006 р. було запропоновано термін "кіберфізичні системи" (*Cyber-Physical Systems, CPS*), що передбачає інтеграцію обчислень із фізичними процесами. Поняття позначає очікуваний у найближчому майбутньому якісний перехід у сприйнятті вбудованих систем і методів їх проєктування. Сутність *CPS* полягає в тому, що проєктування об'єкта керування та системи управління для цього об'єкта мають виконуватися в єдиному ключі, в єдиному комплексі інструментальних засобів, що тісно взаємодіють [1].

Система на кристалі (*System on Chip, SoC*) побудована на єдиному кристалі, у якій інтегруються такі елементи, як процесор (процесори, зокрема спеціалізовані), пам'ять, кілька периферійних пристроїв, спеціалізовані обчислювальні блоки та їх з'єднання. Усе згадане вище становить оптимальний набір для деякого заздалегідь відомого

ISSN 2522-9818 (print)
Сучасний стан наукових досліджень та технологій в промисловості. 2023. № 4 (26) ISSN 2524-2296 (online)

функціонала, наприклад, оброблення та передачі відеоінформації. Вислів "система на кристалі" не є насправді терміном. Це поняття відбиває загальну тенденцію до підвищення рівня інтеграції за допомогою інтеграції функцій [2].

SoC часто застосовуються як компоненти вбудованих систем, і для них використовуються одні й ті самі методи проєктування [3]. Ідеалізований маршрут проєктування *SoC* містить такі кроки: 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 програмного забезпечення (ПЗ), верифікація реалізації.

Сучасні вимоги до проєктування *SoC* привели до появи таких нових підходів, як сумісне проєктування (співпроєктування) та сумісна верифікація (співверифікація) [4]. Співпроєктування – це процес паралельного проєктування апаратних і програмних засобів, за умови якого оцінюється доцільність вибору апаратної або програмної реалізації певного фрагмента проєкту. Цей процес також дає проєктувальникам змогу побачити, як система могла б працювати з поділом обчислень між апаратурою та ПЗ. Співверифікація – це аналіз спільної роботи програмного забезпечення та апаратних засобів *SoC* з метою визначення, чи вони функціюватимуть правильно разом. З іншого боку, під час цього процесу аналізується, чи однаково вирішуватимуться особливі завдання проєкту в умовах апаратної чи програмної реалізації. Результатом роботи є фіксація обраного варіанта реалізації.

Однією з основних проблем проєктування контролепридатних цифрових пристроїв є реалізація їх вбудованого самотестування. Самотестування програмно-апаратних систем на кристалі призначене для виявлення та діагностики потенційних помилок апаратного й програмного забезпечення всередині самого *SoC*. Запускаючи комплексне самотестування, розробники вбудованих систем можуть заздалегідь виявляти й усувати будь-які проблеми, здатні поставити під загрозу загальну надійність і функціональність системи. Ці тести охоплюють значну кількість різних сфер, зокрема пам’ять, порти введення-виведення, таймери, периферійні пристрої та навіть процесорну частину.

У багатьох галузях, особливо в тих, що пов’язані з критично важливими щодо безпеки застосунками, діють суворі нормативні стандарти, яких необхідно дотримуватись. Дотримання стандартів є необхідним забезпеченням якості та надійності вбудованих систем. Тому самотестування відіграє вирішальну роль у виконанні цих вимог. Самотестування *SoC* допомагає в процесі налагодження, надаючи цінну інформацію про стан *SoC* і пов’язані з ним компоненти. Залучивши комплексні механізми самотестування, розробники можуть продемонструвати, що їх системи пройшли ретельне тестування та відповідають необхідним нормативним критеріям.

Аналіз останніх досліджень і публікацій

У роботі [5] визначено роль і місце систем логічного управління в класі реактивних систем, що активуються у відповідь на зовнішні події. Для опису частини керування систем логічного управління застосовують кінцеві автомати, зазвичай у формі моделей Мура. Для подання керуючих автоматів у системах логічного управління використовують графи переходів, що є не лише візуальним зображенням алгоритму роботи автомата, але і його повною математичною моделлю. У цій праці також запропоновано методику розроблення автоматних систем логічного управління з огляду на реальний час та оброблення зовнішніх подій.

Поняття часового автомата (*timed automata*) як засіб опису систем реального часу розглянуто в роботі [6]. Граф переходів автомата доповнюється скінченним набором таймерів, що приймають дійсні значення. Кожен таймер скидається на нуль у момент переходу та збільшує своє значення з кожним тактом автомата. З кожним переходом пов’язано часове обмеження (*clock constraint*), воно означає, що цей перехід може відбутися лише в разі, коли поточні значення таймера відповідають цьому обмеженню. З кожною позицією пов’язане обмеження на таймери, яке називається інваріантом; система може перебувати в цій позиції лише доти, доки виконується її інваріант.

У праці [7] досліджуються методи тестування, що беруть до уваги часові характеристики технічної системи. Для опису поведінки системи використовується модель часового кінцевого автомата *TFSM (Timed Finite State Machine).* У процесі розроблення тестів для часових автоматів розглядається модель *TFSM*, що зважає на тайм-аут (*timeout*) у станах і затримки вихідних сигналів щодо реалізації переходу в стан. Водночас береться до уваги, що якщо протягом тайм-ауту не надійшло

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2023. No. 4 (26)

жодного вхідного сигналу, то автомат переходить у наступний стан.

Загалом модель часового автомата передбачає три типи часових параметрів: тайм-аут у станах, часове обмеження на прийом вхідних сигналів і час оброблення вхідного сигналу, тобто затримка вихідного сигналу щодо вхідного. У цьому разі можуть досліджуватися часові автомати з меншою кількістю параметрів. У роботі [8] розглядаються завдання мінімізації часових автоматів, перевірки їх еквівалентності та синтезу тестів для перевірки.

У праці [9] порушуються проблеми побудови апаратних подієвих систем логічного управління реальним часом на програмованих логічних інтегральних схемах (ПЛІС). Алгоритм управління реалізується на основі моделі часового автомата (*timed FSM*), поданого темпоральним графом переходів (*temporal state diagram*). Побудова моделі пристрою керування виконана мовою опису апаратури *VHDL* у формі трипроцесного шаблону з обробленням зовнішньої події. Функціональна верифікація моделі здійснювалася з використанням інструментальних засобів *Active-HDL*, синтез схемної реалізації виконаний на технологічній платформі ПЛІС *Spartan 3E* інструментальними засобами САПР *Xilinx ISE*. Проаналізовані апаратні витрати на схемну реалізацію пристрою керування.

У роботі [10] для реалізації систем логічного управління реального часу була запропонована модель часового автомата Мура з тайм-аутами та часовими обмеженнями з використанням додаткового лічильника для зберігання значення часової змінної. На основі цієї моделі був розроблений темпоральний граф переходів для системи управління дорожнім світлофором. На його основі розроблена двопроцесна *VHDL*-модель часового автомата Мура. За допомогою середовища *Xilinx ISE* виконані верифікація, синтез та імплементація розробленої *VHDL*-моделі. Синтез і моделювання до та після імплементації здійснювалися для мікросхем *CPLD XC9572XL-10-TQ100* та *FPGA XC3S500E-5fg320*. Результати синтезу та моделювання схеми після імплементації підтвердили працездатність і коректність розробленої *VHDL*-моделі.

У монографії [11] описано методи синтезу базових компонентів вбудованих цифрових пристроїв. Особливу увагу приділено синтезу вбудованих цифрових пристроїв з мікропрограмним управлінням. Подано аспекти впровадження елементів надійності на початкових стадіях проєктування цифрових пристроїв, зокрема й методи захисту цифрового контенту вбудованих систем. Розглянуті принципи побудови самотестованих цифрових пристроїв на основі вбудованих засобів для формування тестових послідовностей, аналізу реакцій із подальшим їх порівнянням з еталонними значеннями.

Деталі використання високорівневого синтезу (*High Level Synthesis, HLS*) для *FPGA* подано в роботі [12], докладно наведено приклади використання *HLS* інструментів та їх специфічні оптимізації для *FPGA*-платформ. Розглянуто використання *HLS* для розв'язання задач у сферах кодування інформації, оброблення зображень і аудіо в реальному часі.

У публікації [13] порушується проблема високопродуктивного потокового генерування випадкових чисел у діапазоні рівномірного й нормального розподілу в ПЛІС. Робота зосереджена на легкій реалізації, придатній для широкого спектра ПЛІС. Спочатку розглядаються наявні типи модулів генерації випадкових чисел і описано побудову розробленого генератора. Його розподілено на дві частини: реалізація потокового генератора рівномірних чисел і потоковий генератор гауссівських чисел на основі кумулятивного розподілу. У роботі модулі реалізовані за допомогою мови синтезу високого рівня (*C/C++*), протилежно до типових підходів на рівні мов опису апаратури (*HDL*).

У досліджені [14] описуються інструменти високорівневого синтезу, що дають змогу розробляти спеціалізовані апаратні прискорювачі (*HWacc*). Проте етап верифікації все ще є найдовшим етапом у життєвому циклі розробки. На відміну від програмної індустрії, в інструментів *HLS* відсутні фреймворки тестування, які могли б охоплювати весь процес розроблення, особливо етап верифікації на бортовому рівні згенерованого опису рівня регістрових передач (*Register Transfer Level, RTL*). У статті запропоновано фреймворк для верифікації модулів, створених на основі *HLS*, з використанням реконфігурованої системи та контейнерів (*Docker*) з метою автоматизації процесу верифікації та збереження чистого середовища для тестування, зробивши тестову систему перехідною між різними етапами розроблення.

У дослідженні [15] розглядається використання потужних інтегральних схем на ринку для різноманітних застосувань із різними вимогами й завантаженнями обчислень. Особлива увага в роботі приділяється системам, що застосовуються в галузі високопродуктивних обчислень

Сучасний стан наукових досліджень та технологій в промисловості. 2023. № 4 (26)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

(*High Performance Computing*). У цій галузі основними аспектами, що потрібно брати до уваги, є продуктивність (за визначенням) та споживання енергії (оскільки витрати на операційну діяльність переважають витрати на закупівлю).

У праці [16] оцінюється можливість використання *HLS* для ефективної імплементації загального коду на різних цільових платформах. Порівнювалися такі платформи: *ASIC* (система зі спеціалізованими інтегральними мікросхемами) та *FPGA* (програмовані логічні інтегральні схеми). Можливість використовувати одну кодову базу для обох платформ дозволить прототипування на *FPGA* та швидкий цикл розроблення. Для оцінювання придатності використання коду створюється фільтр із кінцевою імпульсною характеристикою.

Огляд публікацій доводить актуальність створення нових технологій проєктування та самодіагностики систем на кристалі.

Мета цього дослідження – розроблення моделей і процедур проєктування та самотестування в циклі автоматизованого проєктування автоматних систем логічного управління на технологічній платформі *SoC*, що суттєво підвищить надійність їх функціювання.

Для досягнення поставленої мети необхідно виконати такі завдання:

– розглянути процедури взаємодії процесорного ядра з програмованою логікою в складі *SoC*;

– удосконалити процедури проєктування та тестування програмно-апаратних систем на основі *SoC*;

– удосконалити процедури автоматизованого проєктування, верифікації та діагностування кіберфізичних систем логічного управління з використанням мов програмування та мов опису апаратури;

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

Апаратна платформа *SoC Zynq-7000*

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 такі функціональні блоки: процесорну підсистему, що містить процесорний модуль, інтерфейси пам'яті, периферійні інтерфейси, міжблокові інтерфейси та інтерфейси до програмованої логіки, а також програмовану логіку. 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 крізь ділянку загальної пам'яті способом передачі повідомлень [17].

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

Платформа *Xilinx ZYNQ-7000* має архітектуру, де поєднано систему оброблення (*processing system, PS*) та програмовану логіку (*programmable logic, PL*). Взаємодія двох систем відбувається з допомогою інтерфейсу *AMBA* на основі протоколу *AXI.*

AXI – це двоточковий інтерфейс, розроблений для високопродуктивних і швидкодіючих мікроконтролерних систем. Протокол *AXI* оснований на каналі "точка-точка", що передбачає спільне використання шини та дає змогу збільшити пропускну здатність і зменшити затримку. *AXI* є найпопулярнішим серед усіх міжз'єднань інтерфейсу *AMBA*. Він забезпечує взаємодію різних блоків усередині кожного чипа. Механізм квитування гарантує, що інформація передається чітко й безперебійно. Це дає змогу різним компонентам взаємодіяти без будь-яких перешкод.

Маршрут проєктування *SoC*

Процес проєктування вбудованих систем, що реалізуються на базі кристалів програмованої логіки з архітектурою *FPGA* і розширюваних обчислювальних платформ сімейства *Zynq-7000 AP SoC*, загалом передбачає шість етапів [17].

1. Розроблення проєкту мікропроцесорної системи. На цьому етапі визначається архітектура

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2023. No. 4 (26)

системи й розподіл функцій щодо виконання обчислень між апаратною та програмною частиною із використанням обраних критеріїв та метрик (енергоспоживання, швидкість реакції системи, оптимальність виконання блоків). Також додається необхідність вибору цільового процесора або *SoC* для подальших етапів проєктування.

2. Проєктування апаратної платформи системи, яка розробляється. Етап передбачає формування проєкту апаратної платформи та визначення IP-ядер, необхідних для конкретного завдання. Наступними завданнями на цьому етапі є реалізація апаратної платформи на базі обраного чипа та її верифікація.

3. Підготовка системних програмних засобів нижнього (апаратного) рівня. На цьому етапі відбуваються розроблення, налагодження первинного завантажувача системи (*First Stage BootLoader, FSBL*) та формування пакету підтримки плати (*Board Support Package, BSP*). Під час розроблення *FSBL* можуть бути розгорнуті базові засоби для оновлення (*Over-The-Air update, OTA*), що спрощує подальші етапи тестування системи. Створення *BSP* передбачає формування необхідного пакету драйверів системи та їх тестування / інтеграцію / адаптацію для обраного набору периферії.

4. Формування основного програмного забезпечення системи, що розробляється. На цьому етапі формується архітектура ПЗ системи та пишеться код застосунків з одночасним виконанням типового ітераційного створення ПЗ, орієнтованої на тестування (*Test Driven Development, TDD*).

5. Комплексне моделювання та налагодження. Під час цього етапу виконується інтеграція розроблених програмних і апаратних компонентів на цільовій платформі. Комплексне налагодження містить як перевірку інтеграції компонентів, так і налагодження граничних станів системи (режим сну, вихід із режиму сну, перехід у режим енергозбереження, навантажувальне тестування та перевірки безпеки системи загалом).

6. Генерація завантажувального образу та розгортання розробленої системи. Для обраного варіанта завантаження системи генерується образ, який може містити такі компоненти: образ для прошивання енергонезалежної вбудованої пам’яті (*Embedded Multimedia Memory Card, eMMC*), образ для розгортання системи із застосуванням завантажувача *NFS boot (Network File System)* для розробників, генерація та прошивання *fuse*-конфігурації для захисту системи.

Інструментальні засоби

Після вибору варіанта конфігурації процесорного блоку, що забезпечує необхідну продуктивність у процесі виконання функцій вбудованої системи, можна перейти безпосередньо до проєктування її апаратної платформи.

Розроблення вбудованого програмного забезпечення здійснюється з використанням *Vitis IDE*. Узагальнений процес проєктування ПЗ на базі *Vitis IDE* є типовим циклом для вбудованої платформи. Основна частина – це конфігурація платформи для цільової плати та генерація *Software Development Kit (SDK)*. Після цього етапу можна скористатися *Application Programming Interface (API)* для роботи в режимі *baremetal* або з додаванням *FreeRTOS*.

Вбудоване середовище розробки *Vitis IDE* містить готові базові приклади роботи з *Zynq-7000*, серед яких застосунок тесту *DDR* пам’яті, перевірка *LwIP* мережного стеку. Після створення апаратної частини у *Vivado* та генерації необхідних файлів доцільно перевірити цілісність пам’яті та виконати застосунок *memory_test*, результати якого будуть доступні в консолі послідовного порту, до якого підключено плату. Зазначимо, що для програмування цільової платформи може бути використаний як офіційний *Xilinx Platform Cable*, так і *Xilinx Virtual Cable Protocol*, що дає змогу прошити й налагодити віддалений пристрій.

Узагальнене проєктування IP-ядер виконується в середовищі *Vivado IDE* із застосуванням як готових блоків, так і користувацьких ядер. Після створення IP-ядра можна контролювати версії та, відповідно, оновлювати блоки, а також інтегрувати ядра через процес *System Block Designer*, де можна налаштувати параметри блоку та інтегрувати його разом до *PL*-частини. Одним із доступних варіантів генерації IP-ядер є використання *Vitis HLS* для роботи з високорівневим синтезом. Також може бути застосований класичний підхід з *Verilog/VHDL*-описом і відповідним *testbench*.

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

Процес верифікації IP-блоку передбачає написання відповідного *testbench* та інтеграцію в *System Block Design*. Зазначимо, що після

Сучасний стан наукових досліджень та технологій в промисловості. 2023. № 4 (26)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

проходження верифікації та збирання IP-блок стає доступним у репозиторії проєкту, звідки може бути доданий до системи та налаштований. Основними параметрами налаштування можуть бути як константи, що змінюються для конкретного дизайну, так і параметри *AXI*-шини.

У типовому процесі дизайну для платформи *ZYNQ* доцільно використовувати інструментарій *Vitis HLS* для синтезу IP-блоків і *Vivado* – для проєктування системи. Написання програмного забезпечення для *ARM* (*PS*-частини) здійснюється в середовищі *Vitis IDE*. На рис. 1 зображений типовий маршрут використання інструментів *Xilinx*. Цей процес є реалізацією окремих IP-блоків у середовищі *Vitis HLS* та написання відповідних *testbench* для них, після чого імпорт до середовища *Vivado*, де інтегруються блоки IP з *PS*-частиною *ZYNQ*. Після синтезу та отримання бітстриму для конфігурації *FPGA*-частини *ZYNQ* виконується операція *Export Hardware including bitstream* для отримання *xsa*-файлу. Далі, відповідно до створеної архітектури системи, можна або згенерувати образ *Linux* із використанням *petalinux* на базі *Yocto Framework* та *xsa*-файлу, або імпортувати отриманий *xsa*-файл до *Vitis IDE* й застосовувати *baremetal SDK* від *Xilinx*. Також можна проводити налагодження в середовищі *Vitis IDE* для крос-компільованого застосунку на базі *PetaLinux*.

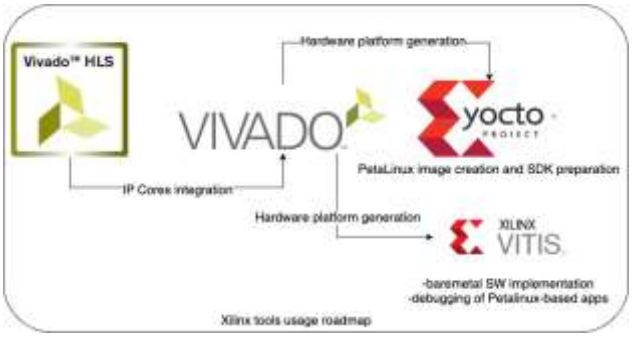

Рис. 1. Маршрут використання інструментів *Xilinx* для платформи *ZYNQ*

Пристрій логічного керування дорожнім світлофором

Розглянемо алгоритм функціювання дорожнього світлофора, що працює у двох режимах: денному й нічному. Множина вхідних сигналів $X = \{Onn,\ St,\ Btn\}$, де $Onn = \{0,\ 1\}$ – сигнал увімкнення дорожнього світлофора та запуску нічного циклу; $St = \{0,\ 1\}$ – сигнал запуску денного циклу роботи світлофора; $Btn = \{0,\ 1\}$ – сигнал увімкнення зеленого світла на пішохідному переході. Отже, Onn і St є оповіщувальними сигналами, а Btn – подією.

Множина вихідних сигналів для світлофорів $Y = \{R1,\ YRG,\ YGR,\ G1,\ R2,\ G2\}$, де $R1$ – сигнал увімкнення червоного світла на основній дорозі; $G1$ – сигнал увімкнення зеленого світла світлофора на основній дорозі; YRG – сигнали увімкнення жовтого світла на основній дорозі (за умови зміни $R - G$); YGR – сигнали увімкнення жовтого світла на основній дорозі (у разі зміни $G - R$); $R2$ – сигнал увімкнення червоного світла на пішохідному переході; $G2$ – сигнал увімкнення зеленого світла на пішохідному переході.

Інтерфейс системи управління дорожнім світлофором зображений на рис. 2.

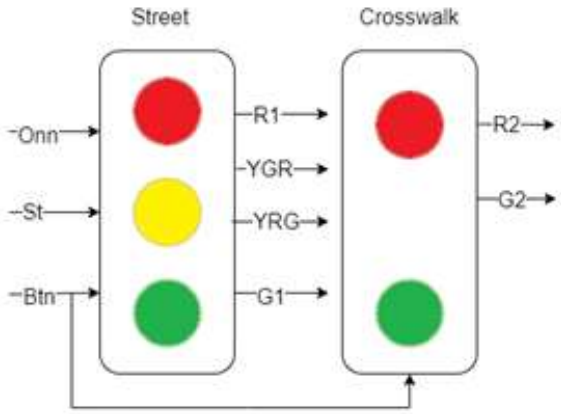

Рис. 2. Інтерфейс системи управління дорожнім світлофором

Визначимо стани керуючого автомата:

– стан $a1$ – увімкнення автомата, вихідних сигналів немає, затримка to_1;

– стан $a2$ – жовтий за умови зміни $G - R$, виходи $\{YGR,\ R1,\ R2\}$, затримка to_2;

– стан $a3$ – червоний на дорозі, зелений на переході, виходи $\{R1,\ G2\}$, затримка to_3;

– стан $a4$ – жовтий за умови зміни $R - G$, виходи $\{YRG,\ R1,\ R2\}$, затримка to_2;

– стан $a5$ – зелений на дорозі, червоний на переході, виходи $\{G1,\ R2\}$, затримка to_3;

– стан $a6$ – увімкнення зеленого на переході та червоного на дорозі по кнопці Btn, затримка to_6,

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2023. No. 4 (26)

виходи $\{R1, G2\}$ із затримкою появи t_d, затримка у стані to_6;

– стан $a7$ – горить тільки жовтий, затримка to_1.

Наведемо алгоритм роботи світлофора. Під час увімкнення пристрою керування $(Onn = 1)$ запускається нічний цикл роботи світлофора, відбувається миготіння жовтого світла на основній дорозі $(a_1 - a_7)$, світлофор на пішохідному переході не працює. Після запуску денного циклу $(St = 1)$ реалізується така система переходів: $a_2 - a_3 - a_4 - a_5 - a_2$. У стані a_5, коли на основній дорозі горить зелений, визначено вікно прийому (*time constraint*) t_c для зовнішньої події Btn (натиснення кнопки переходу). У процесі оброблення цієї події керуючий автомат переходить у стан a_6. У цьому разі на основній дорозі загоряється червоний, а на пішохідному переході затримується червоний, а зелений загоряється із затримкою t_d (час на підготовку до переходу). За період t_c може бути прийнятий тільки один сигнал (зовнішня подія) Btn; $t0$ – початок вікна; t_c, t_d – затримка формування вихідних сигналів; t_c – вікно прийому зовнішньої події.

Отже, для системи управління дорожнім світлофором може бути побудований темпоральний граф переходів часового автомата Мура (рис. 3).

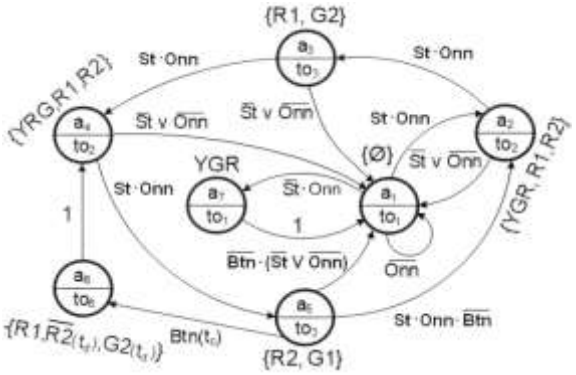

Рис. 3. Темпоральний граф переходів автомата Мура для системи управління дорожнім світлофором

На рис. 4 подана часова діаграма функціювання часового автомата Мура для системи управління дорожнім світлофором у денному циклі роботи, яка, власне, є специфікацією пристрою, що проєктується.

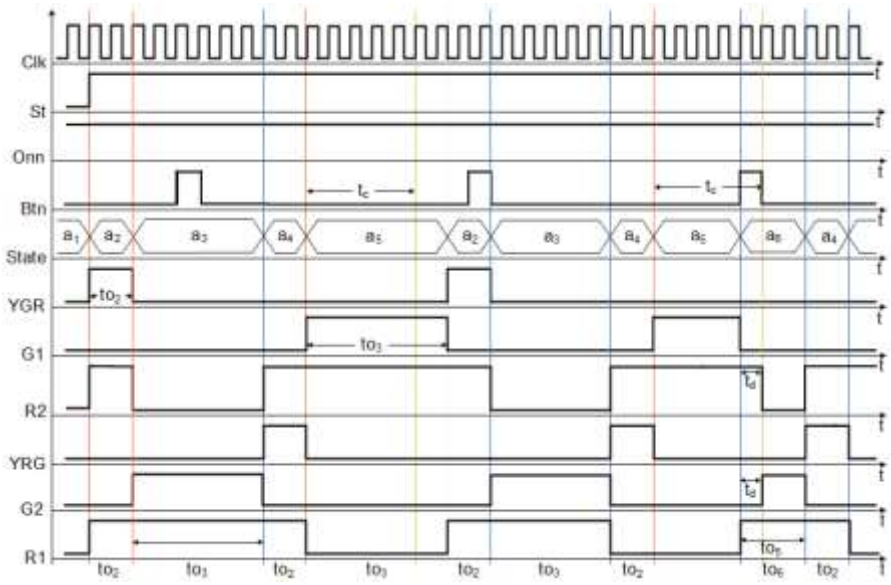

Рис. 4. Часова діаграма автомата Мура для системи управління дорожнім світлофором

Реалізація в *SoC*

У разі наявності графової моделі Мура для автоматного пристрою керування функціональне покриття специфікації може бути зведено до обходу всіх станів автомата Мура та реалізації всіх переходів у кожному стані. Під час проведення верифікації автоматних моделей реалізується так званий "неруйнівний" діагностичний експеримент (ДЕ), у процесі проведення якого тест (обхід графа) починається з початкової вершини графа переходів і повертається до неї. В організації ДЕ із самодіагностики пристрою керування на *SoC* як тестер використовується операційний автомат (ОА),

Сучасний стан наукових досліджень та технологій в промисловості. 2023. № 4 (26)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

що імітує сигнали *Onn*, *St* та *Btn*. Еталони зберігаються у пам'яті *PS* у вигляді значень $R1$, *YGR*, *YRG*, $G1$, $R1$, $G2$ у відповідні проміжки часу, а проміжки часу у вигляді констант T_i також зберігаються в пам'яті *PS*. Шляхи обходу графа переходів зберігаються в пам'яті *PS* і реалізуються способом задання параметра t у певні проміжки часу.

Керуючий автомат (КА) виробляє вихідні сигнали $R1$, *YGR*, *YRG*, $G1$, $R1$, $G2$, формує з них бінарний вектор та передає його до ОА. Там цей вектор порівнюється з еталоном і результат порівняння крізь блок введення / виведення передається на контрольний світлодіод для візуального спостереження. Крім того, сигнали $R1$, *YGR*, *YRG*, $G1$, $R1$, $G2$ безпосередньо з КА крізь блок введення / виведення *PL* передаються на світлодіодну панель вихідних сигналів для візуального спостереження. Узагальнений вигляд системи наведено на рис. 5.

Операційний автомат у режимі проведення ДЕ імітує видачу вхідних сигналів для КА.

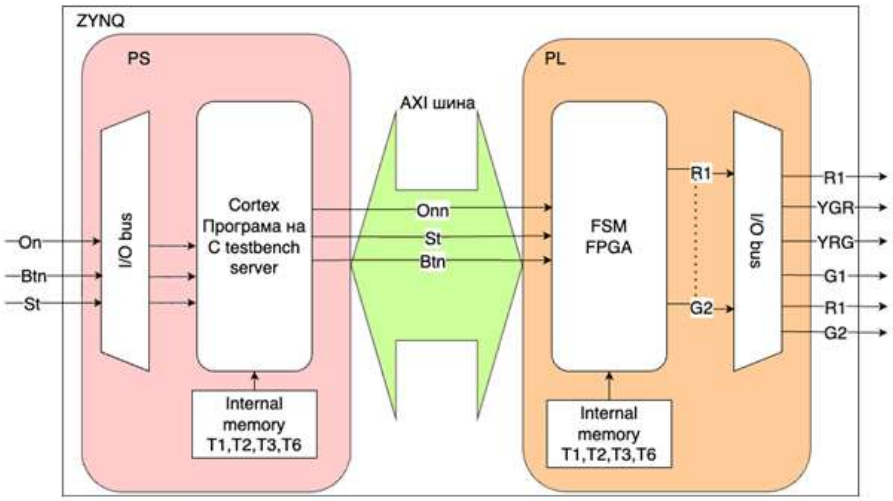

Рис. 5. Реалізація пристрою керування в *SoC ZYNQ-7000*

Побудова моделі пристрою у САПР *Vivado*

У середовищі *Vitis HLS* із підтримкою *High Level Synthesis* логіку КА реалізовано з використанням *pragma*-директив компілятора, що дають змогу призначити конкретний інтерфейс, що буде застосовано під час синтезу IP-блоку. Перед процесом синтезу необхідно вказати *top-level-function*, яка має бути використана для синтезу, й очікуваний *clock*-період блоку. Окремо потрібно визначити модель *SoC*, яка може бути застосована для *co-simulation* під час налагодження IP-блоку. Усі моделі логіки роботи *SoC* як блоку *PS*, так і блоків *PL* та *AXI* виконані мовою програмування *C*. Реалізація логіки *FSM* виконана у форматі автоматного шаблону. Як приклад наведемо фрагмент реалізації стану $a5$ у коді моделі *FSM* (рис. 6).

Після розроблення IP-блоку *FSM* виконується його синтез та експорт як IP до середовища *Vivado*. Інтеграція блоку до середовища *Vivado* виконана з допомогою типового процесу додавання користувальницького репозиторію IP-блоків та блоку на *Block Design*. Усі розроблені блоки з використанням технології *Vitis HLS* окремо позначаються на діаграмі. Також система містить *ZYNQ Processing System*, IP-ядро генерації сигналу *RESET* та IP-блок взаємодії з *AXI*-шиною *AXI Interconnect*. Блок *status leds* використано для візуального відображення інформаційних сигналів (рис. 7).

```
case fsm_states::A5:
  if (input_signals & Btn && m_counter >
     constraint_a5_l - 1 && m_counter <
     constraint_a5_h)
     (m_next_state = fsm_states::A6;)
  else if (m_counter < T3 - 1)
         {m_next_state = fsm_states::A5;
          ++m_counter;}
  else if (!(input_signals & OnSignal))
         {m_next_state = fsm_states::A1;}
  else if (!(input_signals & St))
         {m_next_state = fsm_states::A1;}
  else {m_next_state = fsm_states::A2; }
 break;
```

Рис. 6. Реалізація логіки перебування в стані $a5$ та переходів між відповідними станами

ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

Реалізація логіки діагностування *FSM* виконана на *ARM*-частині з використанням мови програмування *C*. *Vitis HLS* дає змогу застосувати базову логіку з *testbench* IP-блоку. Налагодження програмного забезпечення *PS*-частини виконується в середовищі *Vitis IDE* з використанням *Xilinx Platform Cable*, що робить можливим взаємодію з *ZYNQ* з допомогою *JTAG*-інтерфейсу.

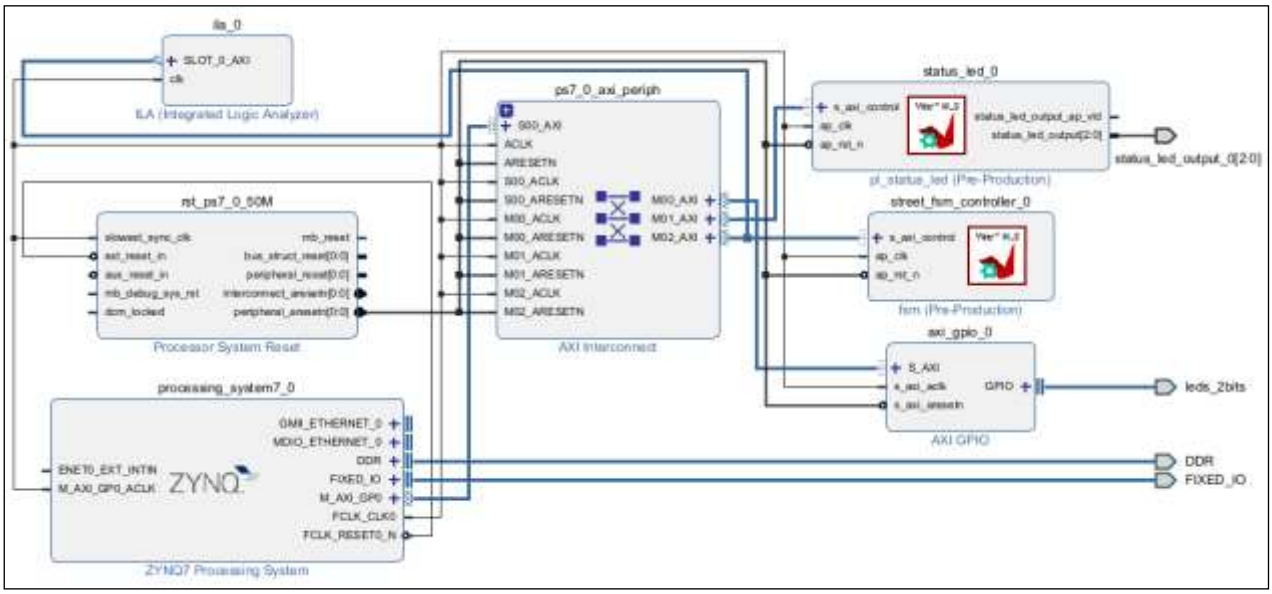

Рис. 7. Діаграма розробленого проєкту *SoC* у середовищі *Vivado ISE*

Діагностичний експеримент

В організації процесу самодіагностики реалізується неруйнівний діагностичний експеримент (ДЕ) способом обходу всіх дуг графа переходів керуючого автомата, починаючи з початкової вершини (рис. 8).

Під час виконання неруйнівного діагностичного експерименту було отримано часові діаграми з вбудованого логічного аналізатора *IP-Core Xilinx Integrated Logic Analyzer*.

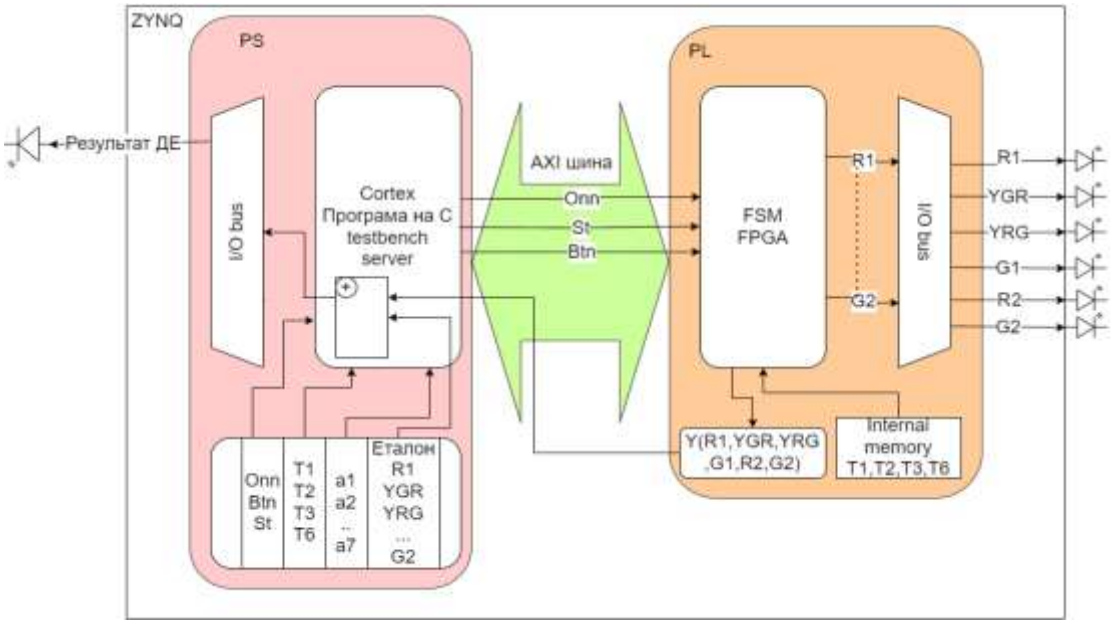

Рис. 8. Реалізація ДЕ пристрою керування для *SoC Zynq-7000*

Сучасний стан наукових досліджень та технологій в промисловості. 2023. № 4 (26)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

Для перевірки коректності оброблення транзакцій *AXI*-шини було виконано під'єднання аналізатора до відповідної шини обміну в системі. У разі використання *AXI*-шини виходи закодовано через зсув одиниці у відповідному 32-бітному регістрі. Результати проведення ДЕ показані способом реалізації типового циклу роботи *FSM* $a_2 - a_3 - a_4 - a_5 - a_2$.

Відповідно, у стані $a3$ активними сигналами керування є $R1$ та $G2$. Їх об'єднання дорівнює $R1 \mid G2$ або $1 << 0 \mid 1 << 5$, що у *HEX* дорівнює 0×21. Відповідні дані отримані по AXI-шині на лінії *streetlight _fsm_bd_i/ps7_0_axi_periph_M02_AXI_RDATA*[31:0].

На рис. 9 наведено стан виходів автомату для стану $a3$ $(R1 = 1,\ G1 = 1)$, отриманий за допомогою вбудованого логічного аналізатора.

Таким самим чином була отримана діаграма для стану $a5$ після подачі відповідних тестових сигналів з боку *PS*-частини системи. На рис. 10 наведено діаграму з логічного аналізатора для стану $a5$.

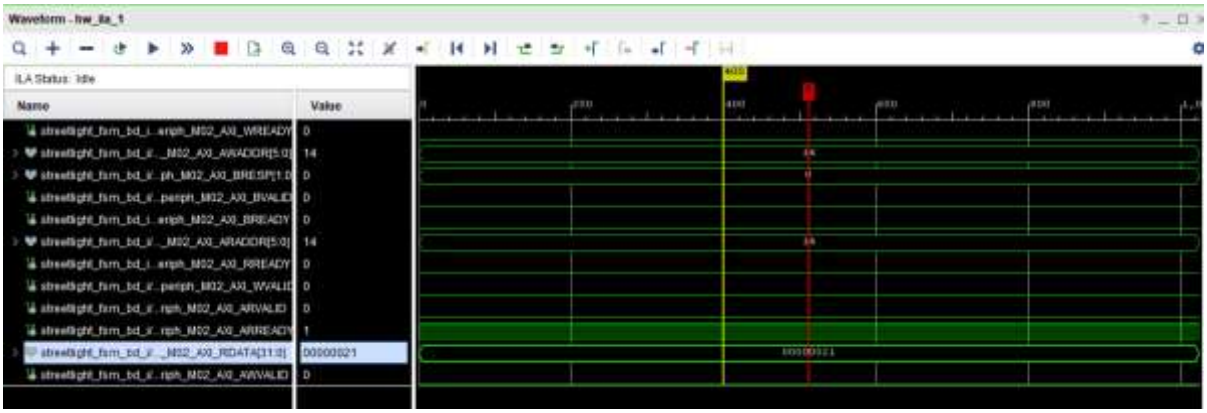

Рис. 9. Часова діаграма з логічного аналізатора сигналів керування для стану $a3$

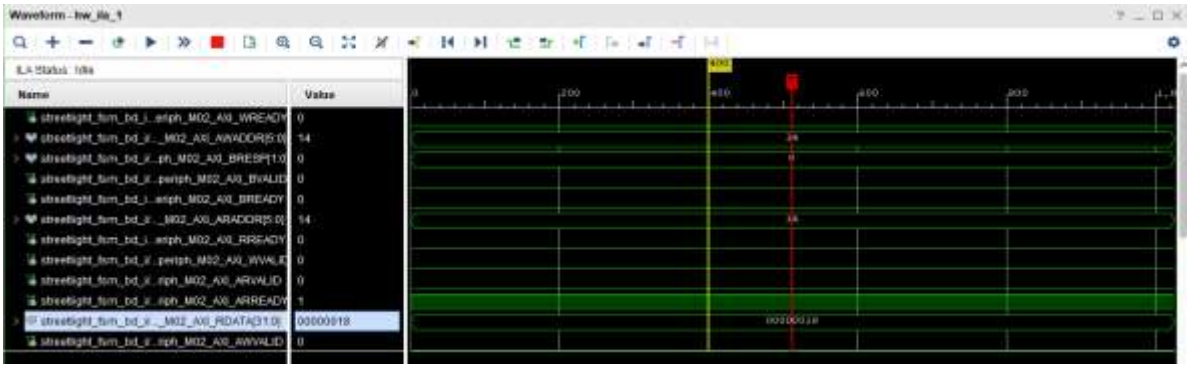

Рис. 10. Часова діаграма з логічного аналізатора сигналів керування для стану $a5$

Порівняння отриманих часових діаграм з еталонною специфікацією (рис. 4) показало повний збіг, що підтверджує коректну працездатність спроєктованого пристрою.

Висновки та перспективи подальшого розвитку

Унаслідок виконання роботи порушено питання побудови системи самодіагностики кіберфізичної системи логічного управління, реалізованої на технологічній платформі системи на кристалі. Проаналізовано принципи побудови вбудованих кіберфізичних систем, які реалізуються в системах на кристалі. Розглянуто методи й маршрути проєктування систем на кристалі, що містять програмну та апаратну частини. Описано принципи побудови систем верифікації та вбудованої самодіагностики *SoC*.

Технологічною платформою обрано програмовану систему на кристалі *FPGA* сімейства *ZYNQ-7000* фірми *Xilinx Inc*. Платформа *Xilinx ZYNQ-7000* має архітектуру, де поєднано систему оброблення *PS* і програмовану логіку *PL*. Взаємодія двох систем відбувається з допомогою інтерфейсу *AMBA* на основі

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) *Innovative technologies and scientific solutions for industries. 2023. No. 4 (26)*

протоколу *AXI*. Як налагоджувальну плату використано недорогу загальнодоступну плату *ZedBoard*, оснащену пристроєм *XC7Z020 Zynq*. Сформульовано загальну послідовність етапів проєктування системи на кристалі на платформі *ZYNQ-7000*. Практичну реалізацію виконано на базі стеку інструментальних засобів САПР *Vivado/Vitis/Vitis HLS*. Це дає змогу виконати наскрізне проєктування як частини, що синтезується з боку *PL*, так і користувацького програмного забезпечення з боку *PS*. Розроблені методи апробовано на моделі пристрою логічного керування світлофором. Як модель пристрою керування розглядається часовий автомат Мура, а його описом є темпоральній граф переходів. Під час реалізації на платформі *ZYNQ-7000* керуючий автомат реалізований у блоці *PL* мовою програмування *C*, а операційний автомат – у блоці *PS*.

В організації самодіагностики реалізується неруйнівний діагностичний експеримент способом обходу всіх дуг графа переходів, починаючи з початкової вершини. Тестером у цьому разі є операційний автомат, еталонні логічні та часові значення зберігаються в пам'яті блоку *PS*. Візуальне спостереження за виконанням ДЕ здійснюється на панелі світлодіодів плати *ZedBoard*.

Наукова новизна роботи полягає в подальшому розвитку моделей та методів проєктування й самодіагностики автоматних пристроїв логічного керування, реалізованих на технологічній платформі систем на кристалі.

Перспективи подальшого розвитку досліджень у цьому напрямі полягають у розробленні моделей розподілу обчислень у *SoC* між блоками *PL* та *PS* залежно від критеріїв замовника.

Список літератури

1. Lee E. A., Seshia S. A. Introduction to embedded systems: A cyber-physical system approach. UC Berkeley, 2011. 499 p. URL: https://ptolemy.berkeley.edu/books/leeseshia/releases/LeeSeshia_DigitalV1_08.pdf

2. Greaves D. Modern system-on-chip design on ARM. Arm Education Media, 2021. 608 p. URL: https://armkeil.blob.core.windows.net/developer/Files/pdf/ebook/arm-modern-soc.pdf

3. Teich J. Hardware software codesign: the past, the present, and predicting the future. *Proc. IEEE*, Vol. 100. 2012. P. 1411–1430. DOI: 10.1109/JPROC.2011.2182009

4. Bailey B., Martin G., Piziali A. ESL design and verification: A prescription for electronic system level methodology. *Morgan Kaufmann Publishers Inc*. 2007. P. 113–138. DOI: 10.1016/B978-012373551-5/50068-X

5. Shalyto A. A. Software automation design: algorithmization and programming of problems of logical control. Journal of Computer and System Sciences International. Vol. 39, No. 6. 2000. P. 899–916. URL: https://www.researchgate.net/publication/297443303_Software_automaton_design_Algorithmization_and_programming_of_problems_of_logical_control

6. Alur R., Dill D.L. A theory of timed automata. *Theoretical Computer Science*. Vol.126, № 2. 1994. P. 183–235. DOI: 10.1016/0304-3975(94)90010-8

7. Zhigulin M. Yevtushenko N., Maag S., Cavalli A. R. FSM-based test derivation strategies for systems with time-outs. *Proceedings of the 11th International Conference on Quality Software (QSIC 2011)*, Madrid, Spain, 2011. P. 141–149. DOI:10.1109/QSIC.2011.30

8. El-Fakih K., Yevtushenko N., Simão A. A practical approach for testing timed deterministic finite state machines with single clock. *Science of Computer Programming*. Elsevier. Vol. 80. 2014. P. 343–355. DOI: 10.1016/j.scico.2013.09.008

9. Miroschnyk M., Shkil A., Kulak E., Rakhlis D., Filippenko I., Hoha M., Malakhov M., Serhiienko V. Design of real-time logic control system on FPGA. *Proceedings of 2019 IEEE East-West Design & Test Symposium (EWDTS'19)*, September 13–16, Batumi, Georgia, 2019. P. 488–491. DOI: 10.1109/EWDTS.2019.8884387

10. Miroschnyk M. A., Shkil A. S., Kulak E. N., Rakhlis D. Y., Miroshnyk A. M., Malahov N. V. Design timed FSM with VHDL Moore pattern. *Radio Electronics, Computer Science, Control*. 2020. №2 (53). P. 137–148. DOI: 10.15588/1607-3274-2020-2-14

11. Мірошник М. А. Проектування діагностичної інфраструктури обчислювальних систем та пристроїв на ПЛІС: монографія. Х.: ХУПС, 2012. 188 с. URL: http://lib.kart.edu.ua/bitstream/123456789/7162/1/%D0%9D%D0%B0%D0%B2%D1%87%D0%B0%D0%BB%D1%8C%D0%BD%D0%B8%D0%B9%20%D0%BF%D0%BE%D1%81%D1%96%D0%B1%D0%BD%D0%B8%D0%BA.pdf

12. Chi Y., Guo L., Lau J., Choi Y. K., Wang J., Cong J. Extending high-level synthesis for task-parallel programs. *Proceedings of 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 28 February – 2 March 2021. 225 p. DOI: 10.1145/3431920.3439470

13. Gniazdowski T. et al. High-performance lightweight HLS generator module of normally distributed random numbers in FPGAs. *Electronics*. Vol. 12, No. 22. 2023. 4667 p. DOI: 10.3390/electronics12224667

Сучасний стан наукових досліджень та технологій в промисловості. 2023. № 4 (26)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

14. Caba J., Rincón F., Barba J., De la Torre J. A., López J. C. FPGA-based solution for on-board verification of hardware modules using HLS. *Electronics*. Vol. 9, No. 12. 2020. 2024 p. DOI: 10.3390/electronics9122024

15. Roozmeh M. High performance computing via high level synthesis, *Xilinx FPGA*. PhD thesis. Turin, 2018. 125 p. DOI: 10.6092/POLITO/PORTO/2710706

16. Heyden M. High Level Synthesis for ASIC and FPGA: master thesis. Sweden, Lund, 2023. 47 p. URL: https://lup.lub.lu.se/student-papers/record/9126669/file/9128084.pdf (дата звернення: 10.11.2023).

17. Crockett L. H., Elliot R. A., Enderwitz M. A., Stewart R. W. The Zynq Book: Embedded Processing with the ARM Cortex-A9 on the Xilinx Zynq-7000. All Programmable SoC. Strathclyde Academic Media, 2014. P. 484. URL: https://is.muni.cz/el/1433/jaro2015/PV191/um/The_Zynq_Book_ebook.pdf (дата звернення: 17.09.2023).

References

1. Lee E. A., Seshia S. A. "Introduction to embedded systems: A cyber-physical system approach, Berkeley". 2011. 499 p. available at: https://ptolemy.berkeley.edu/books/leeseshia/releases/LeeSeshia_DigitalV1_08.pdf

2. Greaves D. "Modern system-on-chip design on ARM, Arm Education Media". 2021. 608 p. available at: https://armkeil.blob.core.windows.net/developer/Files/pdf/ebook/arm-modern-soc.pdf

3. Teich, J. (2012), "Hardware/software codesign: the past, the present, and predicting the future", *Proc. IEEE*, Vol. 100. P. 1411–1430. DOI: 10.1109/JPROC.2011.2182009

4. Bailey, B., Martin, G., Piziali, A. (2007), "ESL design and verification: A prescription for electronic system level methodology", *Morgan Kaufmann Publishers Inc.*, P. 113–138. DOI: 10.1016/B978-012373551-5/50068-X

5. Shalyto, A. A. "Software automation design: algorithmization and programming of problems of logical control", Journal of Computer and System Sciences International, Vol. 39, No. 6, 2000. P. 899–916. available at: https://www.researchgate.net/publication/297443303_Software_automaton_design_Algorithmization_and_programming_of_problems_of_logical_control

6. Alur, R., Dill, D.L. (1994), "A theory of timed automata", *Theoretical Computer Science*, Vol.126, № 2, P. 183–235. DOI: 10.1016/0304-3975(94)90010-8

7. Zhigulin, M. Yevtushenko, N., Maag, S., Cavalli, A. R. (2011), "FSM-based test derivation strategies for systems with time-outs". *Proceedings of the 11th International Conference on Quality Software (QSIC 2011)*, Madrid, Spain, P. 141–149. DOI:10.1109/QSIC.2011.30

8. El-Fakih, K., Yevtushenko, N., Simão, A. (2014), "A practical approach for testing timed deterministic finite state machines with single clock", *Science of Computer Programming*, Elsevier, Vol. 80, P. 343–355. DOI: 10.1016/j.scico.2013.09.008

9. Miroschnyk, M., Shkil, A., Kulak, E., Rakhlis, D., Filippenko, I., Hoha, M., Malakhov, M., Serhiienko, V. (2019), "Design of real-time logic control system on FPGA". *Proceedings of 2019 IEEE East-West Design & Test Symposium (EWDTS'19)*, September 13-16, Batumi, Georgia, P. 488–491. DOI: 10.1109/EWDTS.2019.8884387

10. Miroschnyk, M. A., Shkil, A. S., Kulak, E. N., Rakhlis, D. Y., Miroshnyk, A. M., Malahov, N. V. (2020), "Design timed FSM with VHDL Moore pattern", *Radio Electronics, Computer Science, Control*, №2 (53), P. 137–148. DOI: 10.15588/1607-3274-2020-2-14

11. Miroschnyk, M. A. (2012), "Design of diagnostic infrastructure of computing systems and devices on FPGA" ["Proektuvannya diagnosty`chnoyi infrastruktury` obchy`slyuval`ny`x sy`stem ta pry`stroyiv na PLIS"], monograph, Kharkiv, 331 p. available at: http://lib.kart.edu.ua/bitstream/123456789/7162/1/%D0%9D%D0%B0%D0%B2%D1%87%D0%B0%D0%BB%D1%8C%D0%BD%D0%B8%D0%B9%20%D0%BF%D0%BE%D1%81%D1%96%D0%B1%D0%BD%D0%B8%D0%BA.pdf

12. Chi, Y., Guo, L., Lau, J., Choi, Y. K., Wang, J., Cong, J. (2021), "Extending high-level synthesis for task-parallel programs". *Proceedings of 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 225 p. DOI: 10.1145/3431920.3439470

13. Gniazdowski, T. et al. (2023), "High-performance lightweight HLS generator module of normally distributed random numbers in FPGAs", *Electronics*, Vol. 12, No. 22, 4667 p. DOI: 10.3390/electronics12224667

14. Caba, J., Rincón, F., Barba, J., De la Torre, J. A., López, J. C. (2020), "FPGA-based solution for on-board verification of hardware modules using HLS", *Electronics*, Vol. 9, No. 12, P. 2024. DOI: 10.3390/electronics9122024

15. Roozmeh, M. (2018), "High performance computing via high level synthesis". *Xilinx FPGA*. PhD thesis, Turin, 125 p. DOI: 10.6092/POLITO/PORTO/2710706

16. Heyden, M. "High Level Synthesis for ASIC and FPGA": master thesis. Sweden, Lund, 2023. 47 p. available at: https://lup.lub.lu.se/student-papers/record/9126669/file/9128084.pdf (last accessed: 10.11.2023).

17. Crockett, L. H., Elliot, R. A., Enderwitz, M. A., Stewart, R. W. "The Zynq Book: Embedded Processing with the ARM Cortex-A9 on the Xilinx Zynq-7000. All Programmable SoC, Strathclyde Academic Media", 2014. P. 484. available at: https://is.muni.cz/el/1433/jaro2015/PV191/um/The_Zynq_Book_ebook.pdf (last accessed: 17.09.2023).

Надійшла 04.12.2023

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) *Innovative technologies and scientific solutions for industries. 2023. No. 4 (26)*

Відомості про авторів / About the Authors

Шкіль Олександр Сергійович – кандидат технічних наук, доцент, Харківський національний університет радіоелектроніки, доцент кафедри автоматизації проєктування обчислювальної техніки, Харків, Україна; e-mail: oleksandr.shkil@nure.ua; ORCID ID: https://orcid.org/0000-0003-1071-3445

Рахліс Дарія Юхимівна – кандидат технічних наук, доцент, Харківський національний університет радіоелектроніки, доцент кафедри автоматизації проєктування обчислювальної техніки, Харків, Україна; e-mail: dariia.rakhlis@nure.ua; ORCID ID: https://orcid.org/0000-0002-6652-1840

Філіпенко Інна Вікторівна – кандидат технічних наук, Харківський національний університет радіоелектроніки, доцент кафедри автоматизації проєктування обчислювальної техніки, Харків, Україна; e-mail: inna.filippenko@nure.ua; ORCID ID: https://orcid.org/0000-0002-3584-2107

Корнієнко Валентин Русланович – Харківський національний університет радіоелектроніки, аспірант кафедри автоматизації проєктування обчислювальної техніки, Харків, Україна; e-mail: valentyn.korniienko1@nure.ua; ORCID ID: https://orcid.org/0000-0001-7070-5127

Shkil Alexander − PhD (Engineering Sciences), Associate Professor, Kharkiv National University of Radio Electronics, Associated Professor of Design Automation Department, Kharkiv, Ukraine.

Rakhlis Dariia − PhD (Engineering Sciences), Associate Professor, Kharkiv National University of Radio Electronics, Associated Professor of Design Automation Department, Kharkiv, Ukraine.

Filippenko Inna − PhD (Engineering Sciences), Kharkiv National University of Radio Electronics, Associated Professor of Design Automation Department, Kharkiv, Ukraine.

Korniienko Valentyn – Kharkiv National University of Radio Electronics, PhD Student of Design Automation Department, Kharkiv, Ukraine.

DESIGN AND SELF-DIAGNOSTICS OF CYBERPHYSICAL CONTROL DEVICES ON *SoC* PLATFORM

The subject of research in this article is models, methods, and procedures for designing and self-diagnosing automated models of logic control devices implemented in *SoC*s. **The object of work** is the procedures for automated design and diagnosis of digital devices on the *SoC* technology platform. **The aim of the study** is to develop models and procedures for designing and self-testing in the cycle of automated design of automatic logic control systems on the *SoC* technology platform, which will significantly increase the reliability of their operation. The article solves the following **tasks**: consideration of the procedures for interacting the processor core with programmable logic as part of the *SoC*; improvement of the procedures for designing and testing software and hardware systems based on *SoC*; further development of procedures for automated design, verification, and diagnosis of cyber-physical logic control systems using programming languages and hardware description languages; implementation of the procedure for hardware self-testing of control automata on the *SoC* technology platform. The following **methods** are implemented: synthesis of control automata based on graph models, implementation of control automata models in the *C* programming language using an automata template, diagnostic experiment by traversing the automata transition graph. **Results** achieved. Based on the analysis of the procedures for the interaction of the processor core and programmable logic on the selected *SoC* platform, a model of a cyber-physical logic control system was designed. The practical implementation was carried out on the basis of the *Vivado/Vitis/Vitis HLS* CAD toolkit. The method of hardware self-testing of control automata on the technological platform of SoC *ZYNQ-7000* was implemented. **Conclusions.** The article analyzes the principles of designing embedded cyber-physical systems implemented in system-on-chip. The principles of building verification systems and embedded self-diagnostics of system-on-chip systems containing software and hardware are considered. The developed methods are tested on a model of a traffic light logic control device on the *SoC FPGA* platform of the *ZYNQ-7000* family by *Xilinx*. The Moore's control automaton is implemented in the *PL* block in the *C* programming language, and the operational automaton is implemented in the *PS* block. During the organization of the self-diagnosis process, a non-destructive diagnostic experiment was performed by traversing all arcs of the transition graph, starting from the initial vertex. In this case, the tester was an operational automaton, the reference logic and time values of which were stored in the memory of the *PS* unit. Visual observation of the diagnostic experiment was carried out using the LED panel of the *ZedBoard* board.

Keywords: cyber-physical systems; embedded systems; logic control; system-on-chip design; *FPGA*; CAD; *SoC* self-testing; *C* programming language.

Бібліографічні описи / Bibliographic descriptions

Шкіль О. С., Рахліс Д. Ю., Філіпенко І. В., Корнієнко В. Р. Проєктування та самодіагностика кіберфізичних пристроїв керування на платформі *SoC*. *Сучасний стан наукових досліджень та технологій в промисловості*. 2023. № 4 (26). С. 122–134. DOI: https://doi.org/10.30837/ITSSI.2023.26.122

Shkil, A., Rakhlis, D., Filippenko, I., Korniienko, V. (2023), "Design and self-diagnostics of cyberphysical control devices on *SOC* platform", *Innovative Technologies and Scientific Solutions for Industries*, No. 4 (26), P. 122–134. DOI: https://doi.org/10.30837/ITSSI.2023.26.122