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_____перший (бакалаврський)_____
(рівень вищої освіти)

Модель багатофункціонального годинника
на базі SoC Zynq7000
(тема)

Виконав: студент IV курсу, групи КІУКІ-21-7

_____Чухно Д.С._____
(прізвище, ініціали)

Спеціальність *123 «Комп'ютерна інженерія»*

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

Освітня програма_____

_____*Комп'ютерна інженерія*_____
(повна назва освітньої прграми)

Керівник ____доц. Рахліс Д.Ю._____
(посада, прізвище, ініціали)

Допускається до захисту

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

2025 р.

Харківський національний університет радіоелектроніки

Факультет_____Комп'ютерної інженерії та управління _____

Кафедра_____Автоматизації проектування обчислювальної техніки_____

Рівень вищої освіти___ _____перший (бакалаврський)_____

Спеціальність___ _____123 Комп'ютерна інженерія_____
(шифр і назва)

Тип програми _____ ____освітньо-професійна_____
(освітньо-професійна або освітньо-наукова)

Освітня програма _____Комп'ютерна інженерія_____
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«06»__травня____2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві_____Чухно Денису Сергійовичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи «Модель багатофункціонального годинника на базі SoC Zynq7000» затверджена наказом університету від "21" травня 2025 р. № 403 Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії_____18.06.2025_____
3. Вихідні дані до роботи _____

Плата Ebaz4205 + плата розширення
SoC Zynq7000, 4-розрядний 7-сегментний дисплей
Мова VHDL та C
4-розрядний 7-сегментний дисплей, резистори, кабелі, п'єзодинамік

4. Перелік питань, що потрібно опрацювати в роботі _____

Огляд існуючих архітектур та вибір апартної бази для реалізації
Модель багатофункціонального годинника
Вибір компонентів для реалізації
Створення прототипу багатофункціонального годинника
Програмування алгоритму роботи PL частини SoC в Vivado
Програмування алгоритму роботи PS частини SoC в Vitis
Функціональна верифікація RTL-моделі та фізичне тестування прототипу

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) _____
19 слайдів

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

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

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Видача теми роботи, узгодження і затвердження теми	06.05.2025 – 10.05.2025	*виконано*
2	Аналіз проблемної галузі, постановка задачі, вибір інструментальних засобів	11.05.2025 – 13.05.2025	*виконано*
3	Розробка моделі пристрою	14.05.2025 – 17.05.2025	*виконано*
4	Прототипування моделі	18.05.2025 – 25.05.2025	*виконано*
5	Розробка алгоритму роботи та налагодження	25.05.2025 – 05.06.2025	*виконано*
6	Тестування пристрою	06.06.2025 – 07.06.2025	*виконано*
7	Оформлення пояснювальної записки	08.06.2025 – 13.06.2025	*виконано*
8	Перевірка виконаного проекту керівником, допуск до захисту	14.06.2025 – 17.06.2025	*виконано*
9	Захист проекту	18.06.2025	

Дата видачі завдання ____06.05.2025_____

Здобувач _____
(підпис)

Керівник роботи _____ _____доц. Рахліс Д.Ю.____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи містить 80 сторінок, 78 рисунків, 8 таблиць, 3 додатки та 9 джерел посилання.

ВБУДОВАНА СИСТЕМА, СПІЛЬНЕ ПРОЄКТУВАННЯ, СИСТЕМИ-НА-КРИСТАЛІ, СИСТЕМА ОБРОБКИ, ПРОГРАМОВАНА ЛОГІКА, VHDL, C.

Метою кваліфікаційної роботи є спільне HW/SW проєктування моделі багатофункціонального ручного годинника на базі SoC Zynq 7000 з використанням САПР Vivado Design Suite and Vitis Unified IDE.

У роботі розглянута методологія спільного проєктування апаратного та програмного забезпечення для вбудованих систем на базі SoC, зокрема використання інструментів Vivado Design Suite та Vitis Unified IDE від фірми AMD. В якості приклада було створено модель багатофункціонального ручного годинника на базі SoC Zynq-7000, який включає як апаратну, так і програмну частину. Проєкт для FPGA частини було описано на мові VHDL з використанням готового IP ядра для поділу частоти синхросигналу. Паралельно створено проєкт на мові C, що описує керування IPS дисплеєм. Функціональна верифікація VHDL проєкта через Testbench показала коректність алгоритма.

Прототип пристрою було зібрано на базі плати EBAZ 4205 з використанням плати розширення з IPS дисплеєм, п’єзодинаміком та кнопками на борту, безпаєчній платі та додаткових 4-бітових 7-сегментних дисплеях на ній. Результати експериментів підтвердили коректність прототипу та самої моделі багатофункціонального годинника і, в цілому, доцільність використання САПР Vivado Design Suite and Vitis Unified IDE для спільного проєктування вбудованих систем.

ABSTRACT

Explanatory note of qualification work contains 80 pages, 78 figures, 8 tables, 3 appendices and 9 references.

EMBEDDED SYSTEM, HW/SW CO-DESIGN, SOC, PROCESSING SYSTEM, PROGRAMMABLE LOGIC, VHDL, C.

The purpose of the qualification work is the joint HW/SW co-design of a model of a multifunctional wristwatch based on the SoC Zynq 7000 using CAD Vivado Design Suite and Vitis Unified IDE.

The work considers the methodology of co-design of hardware and software for embedded systems based on SoC, in particular, the use of the Vivado Design Suite and Vitis Unified IDE tools from AMD. As an example, a model of a multifunctional wristwatch based on the SoC Zynq-7000 was created, including hardware and software parts. The project for the FPGA part was described using VHDL and the built-in IP core to divide the clock frequency in Vivado. The control of the IPS display was created using the C language in Vitis. Functional verification of the VHDL project through Testbench showed the correctness of the algorithm.

The prototype of the device was assembled based on the EBAZ 4205 board using an expansion board with an IPS display, a piezo speaker, buttons on board, a breadboard, and two additional 4-bit 7-segment displays.

The results of the experiments confirmed the correctness of the prototype and the model of the multifunctional watch itself and, in general, the feasibility of using the Vivado Design Suite and Vitis Unified IDE CAD for the co-design of embedded systems.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ОС – операційна система;

BRAM – Block Random Access Memory – блочна пам’ять з довільним доступом;

CLB – Configurable Logic Blocks – конфігуровані логічні блоки;

DSP – digital signal processor – процесор цифрової обробки сигналів;

FPGA – field programmable gate array – програмована користувачем вентильна матриця;

HW – hardware – апаратне забезпечення;

IoT – internet of things – інтернет речей;

LUT – Look-Up Tables – таблиця відповідності;

MCU – microcontroller – мікроконтролер;

MMCM – Mixed-Mode Clock Manager – модуль управління синхронізацією;

MPU – microprocessor – мікропроцесор;

PL – programmable logic – програмована логіка;

PLL – Phase-Locked Loop – фазове автопідлаштування частоти;

PS – processing system – система обробки;

SoC – System-on-Chip – система-на-кристалі;

SW – software – програмне забезпечення;

XSA – Xilinx Support Archive – архів підтримки Xilinx.

ВСТУП

Цифрові вбудовані системи – це спеціалізовані комп’ютерні системи, які інтегровані в інші пристрої та призначені для виконання конкретних функцій або завдань у складі більшого пристрою чи системи. Сьогодні у світі використовуються мільярди пристроїв з вбудованими системами в різних галузях, включаючи медичне та промислове обладнання, транспортні системи та військову техніку. Багато споживчих пристроїв – від цифрових годинників до кухонної техніки та автомобілів – також містять вбудовані системи. Вбудовані системи є малими за розміром, швидкими, потужними та розробленими для дуже конкретних завдань. На відміну від універсальних систем, які можуть виконувати багато функцій, вбудовані системи значно дешевші у виробництві для більшості застосувань і часто мають кращі показники надійності, споживання енергії, компактності та інших функціональних і технічних характеристик. Проєктування таких систем має низку особливостей, пов’язаних із жорсткими обмеженнями за розміром, вартістю, продуктивністю, а також з необхідністю забезпечення роботи в реальному часі.

До основних методології проєктування вбудованих систем можна віднести наступні.

1. V-подібна модель, в основі якої лежить чітка послідовність етапів: від вимог до тестування. Кожен етап розробки має відповідний етап валідації. Така модель забезпечує високу якість і контроль, тому добре підходить для критично важливих систем (авіація, медицина).

2. Каскадна модель – послідовний підхід: аналіз → проєктування → реалізація → тестування. Застосовується для невеликих і стабільних проєктів, через обмежену гнучкість при внесенні змін.

3. Ітеративна модель, де розробка відбувається поступово, з розбиттям на окремі функціональні модулі. Це дозволяє тестувати й удосконалювати

систему на кожному етапі. Підходить для складних систем із високою невизначеністю на початковому етапі проєктування.

4. Методологія швидкого прототипування дозволяє отримати робочу моделі пристрою з подальшим вдосконаленням. Це дозволяє зменшити ризики, виявити недоліки ще до повної реалізації. Часто використовується з платформами типу Arduino, STM32, Raspberry Pi.

5. Проектування на основі моделей передбачає застосування спеціалізованого програмного забезпечення (наприклад, MATLAB/Simulink) для створення функціональної моделі системи. Дає змогу автоматично генерувати код для мікроконтролерів. Застосовується у високоточних системах, таких як, автомобільні системи чи робототехніка.

6. Agile/Lean методології, також можуть бути адаптовані до вбудованих систем. Акцент на швидку розробку, адаптацію до змін, часті ітерації. Використовується часто в IoT-пристроях, де важлива швидкість виводу на ринок.

7. Co-проєктування (Hardware-Software Co-Design) – одночасне проєктування апаратної та програмної частин. Цей метод забезпечує оптимальне розподілення функцій між мікроконтролером, периферією та прошивкою. Підходить для реалзіації SoC, FPGA, DSP-систем.

У сучасних умовах розвитку технологій вбудовані системи стають дедалі складнішими, а вимоги до продуктивності, енергоефективності, вартості та гнучкості – все жорсткішими. У цьому контексті методологія Hardware-Software Co-Design стає особливо актуальною і вигідною.

Таким чином, об’єктом дослідження є методологія спільного проєктування апаратного та програмного забезпечення вбудованих систем, предметом – автоматизація спільного проєктування апаратного та програмного забезпечення вбудованих систем за допомогою САПР фірми AMD (Vivado та Vitis) на прикладі моделі багатофункціонального годинника.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ, ПОСТАНОВКА ЦІЛІ ТА ЗАДАЧ

Методологія спільного проєктування апаратного (hardware, HW) та програмного (software, SW) забезпечення (HW/SW co-design) при розробці вбудованих цифрових систем дедалі більше застосовуються у таких галузях, як автомобільна, аерокосмічна, телекомунікаційна та споживча електроніка. При виборі варіанту реалізації вбудованої системи важливо враховувати вимоги до продуктивності, енергоспоживання, вартості та складності розробки [1].

1.1 Огляд існуючих архітектур та вибір апартної бази для реалізації

До найпоширеніших апаратних платформ для реалізації вдубованих систем відносять наступні.

1. Архітектура на основі мікроконтролера (MCU), чіпа, що поєднує в собі процесор, пам’ять та периферійні інтерфейси (наприклад, AVR (Atmega328), STM32, PIC, ESP32). Перевагами такої архітектури є низька вартість, мала споживана енергія та простота. Застосовуються для реалізації побутової техніки, сенсорних модулів, IoT.

2. Архітектура на основі мікропроцесора (MPU), де він виконує лише обчислення та керування, решта компонентів (пам’ять, контролери) підключаються окремо (наприклад, ARM Cortex-A, Intel Atom). Перевагами такої архітектури є вища продуктивність, можливість використання ОС (Linux, Android). Застосовуються для реалізації інтелектуальних та мультимедійних пристроїв, медичного обладнання.

3. Системи-на-кристалі (SoC), які поєднують процесор, графічне ядро, пам’ять, інтерфейси та периферію на одному чипі (наприклад, Broadcom, Qualcomm Snapdragon, Apple M1, ESP32). Основні переваги – це

компактність, енергоефективність та потужність. Застосовуються у мобільних пристроях, розумних годинниках, IoT, камерах.

4. Програмовані логічні інтегральні схеми типу FPGA, які можна перепрограмувати для реалізації будь-якої цифрової логіки (наприклад, Xilinx, Intel/Altera, Lattice). Перевагами є паралельна обробка даних, реальна швидкодія та гнучкість. Застосовується в аерокосмічній галузі, телекомунікаціях, відеообробці, високошвидкісних контролерах.

5. Цифрові сигнальні процесори (DSP), які оптимізовані для математичних обчислень із сигналами (аудіо, відео, сенсори) (наприклад, Texas Instruments DSP, Analog Devices Blackfin). Перевагами є висока ефективність у задачах фільтрації та обробки сигналів. Застосовується при реалізації звукових системи, систем обробки зображень, радіозв'язку.

6. Гібридні архітектури SoC та FPGA, що поєднують ARM-процесор та FPGA в одному рішенні (наприклад, Xilinx Zynq-7000, Intel SoC FPGA). Перевагами є висока обчислювальна здатність та гнучкість. Застосовується в робототехніці, автомобільних системах, машинному навчані та в системах реального часу.

Серед розглянутих апаратних платформ для реалізації вбудованих систем саме гібридна архітектура SoC + FPGA вирізняється найбільшою гнучкістю, функціональністю та потенціалом для складних застосувань [2]. Її головна перевага полягає в поєднанні високопродуктивного ARM-процесора, здатного ефективно керувати системними процесами, з перепрограмовуваною логікою FPGA, яка дозволяє реалізовувати апаратне прискорення критичних операцій або нестандартні периферійні інтерфейси.

1.2 Гібридна архітектури SoC для реалізації вбудованих систем

Гібридні SoC, на відміну від класичних SoC, поєднують програмовані логічні блоки (FPGA) та процесор (зазвичай ARM або RISC-V) (рис. 1.1).

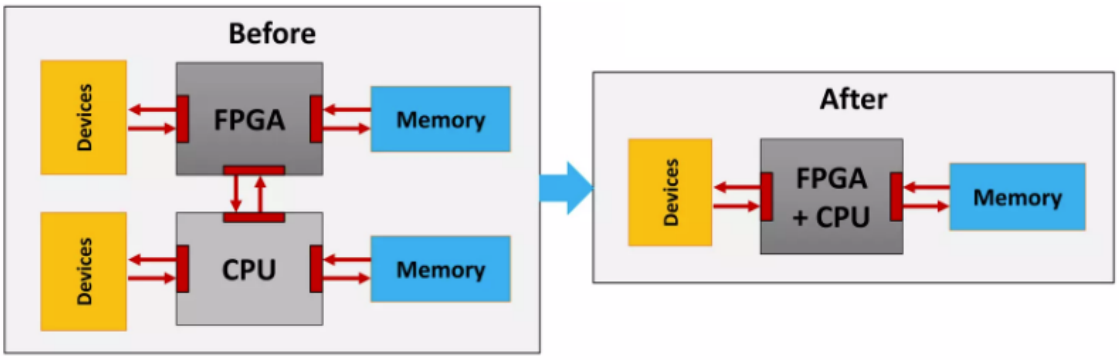


Рисунок 1.1 – Порівняння класичних та гібрідних SoC

Такий підхід дозволяє ефективно реалізовувати обчислення як у програмному середовищі, так і в апаратному, завдяки чому досягається гнучкість, масштабованість і висока продуктивність вбудованої системи.

До популярних гібридних SoC можна віднести наступні:

– Xilinx Zynq-7000 / UltraScale+ MPSoC;
– Intel SoC FPGA (Cyclone V, Arria V);
– Microchip PolarFire SoC;
– Versal ACAP (з вбудованими AI-прискорювачами).

Вони використовуються в якості сучасною універсальної платформи, яка дозволяє створювати високоефективні, гнучкі та адаптивні вбудовані системи. Вони забезпечує ідеальні умови для HW-SW co-design, що робить її оптимальним вибором для критичних застосувань, де важливі швидкодія, точність і надійність.

У цій роботі в якості апаратної платформи використовується гібридна SoC Xilinx Zynq-7000, а саме чип ZYNQ XC7Z010-1CLG400I на базі плати EBAZ4205.

Архітектура Zynq [3] складається з двох основних частин, як показано на рисунку 1.2: системи обробки даних (PS), сформованої навколо двоядерного процесора ARM Cortex-A9, та програмованої логіки (PL), яка еквівалентна FPGA. Вона також має вбудовану пам'ять, різноманітні периферійні пристрої та високошвидкісні комунікаційні інтерфейси.

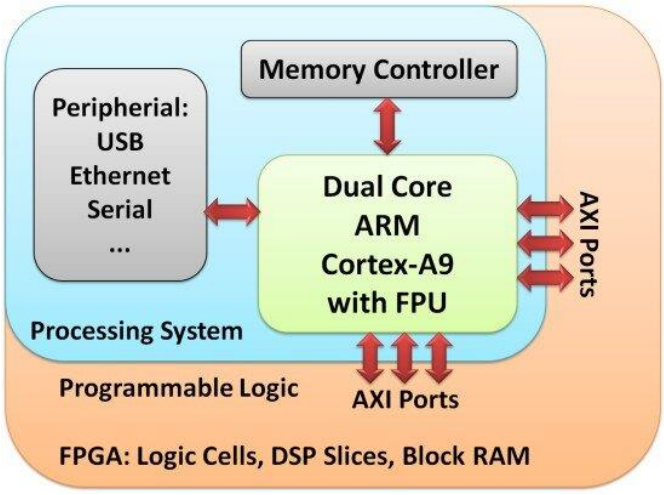


Рисунок 1.2 – Архітектура Zynq-7000

Програмована логіка в Zynq-7000 базується на FPGA-архітектурі Xilinx 7-ї серії (Artix-7 або Kintex-7) в залежності від моделі чипа.

Ключові компоненти структури PL:

– конфігуровані логічні блоки (CLB) містять LUT (Look-Up Tables), регістри та мультиплексори, які реалізують користувацькі цифрові схеми (рис. 1.3);

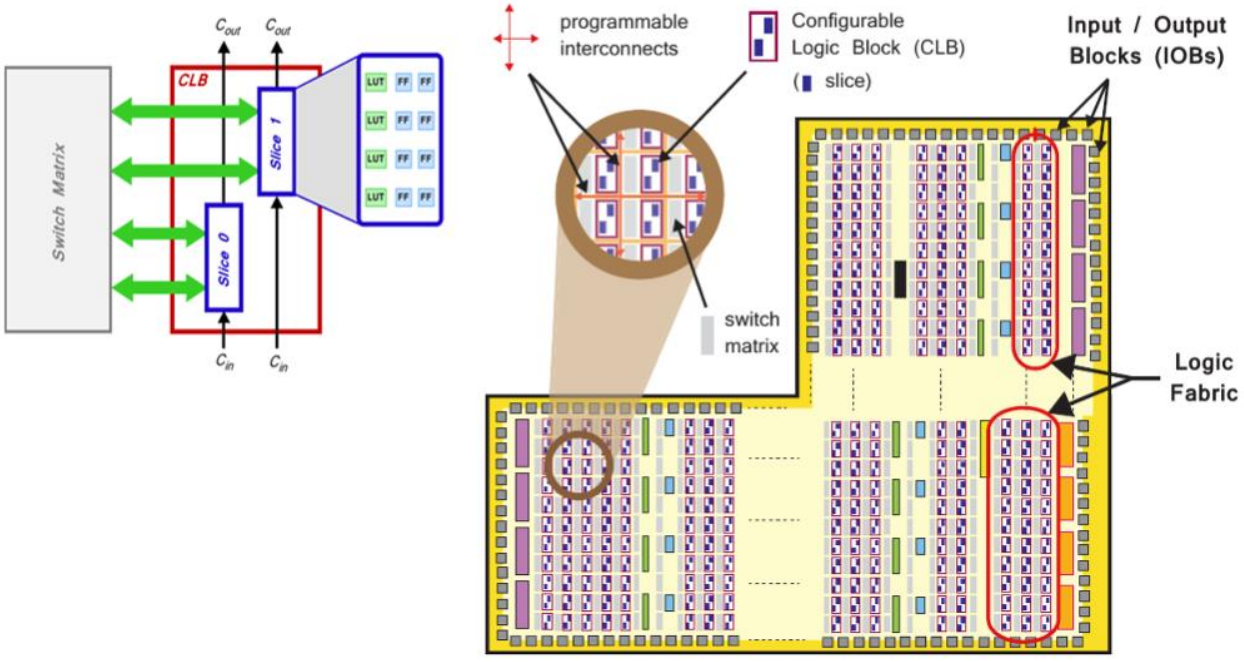


Рисунок 1.3 – Загальна структура FPGA

– DSP-блоки (DSP Slices), що призначені для цифрової обробки сигналів, зображення, відео, AI-алгоритмів;

– Block RAM (BRAM) – вбудована швидка пам’ять, яку можна використовувати як однопортову або двопортову пам'ять, FIFO або кеш для тимчасового збереження даних у PL без звернення до основної пам’яті;

– MMCM та PLL блоки, що керують формуванням тактових сигналів різної частота та дають змогу створити незалежні тактові домени;

– входи/виходи (I/O Banks) дозволяють підключення зовнішніх пристроїв.

Основні компоненти PS частини представлені на рисунку 1.4.

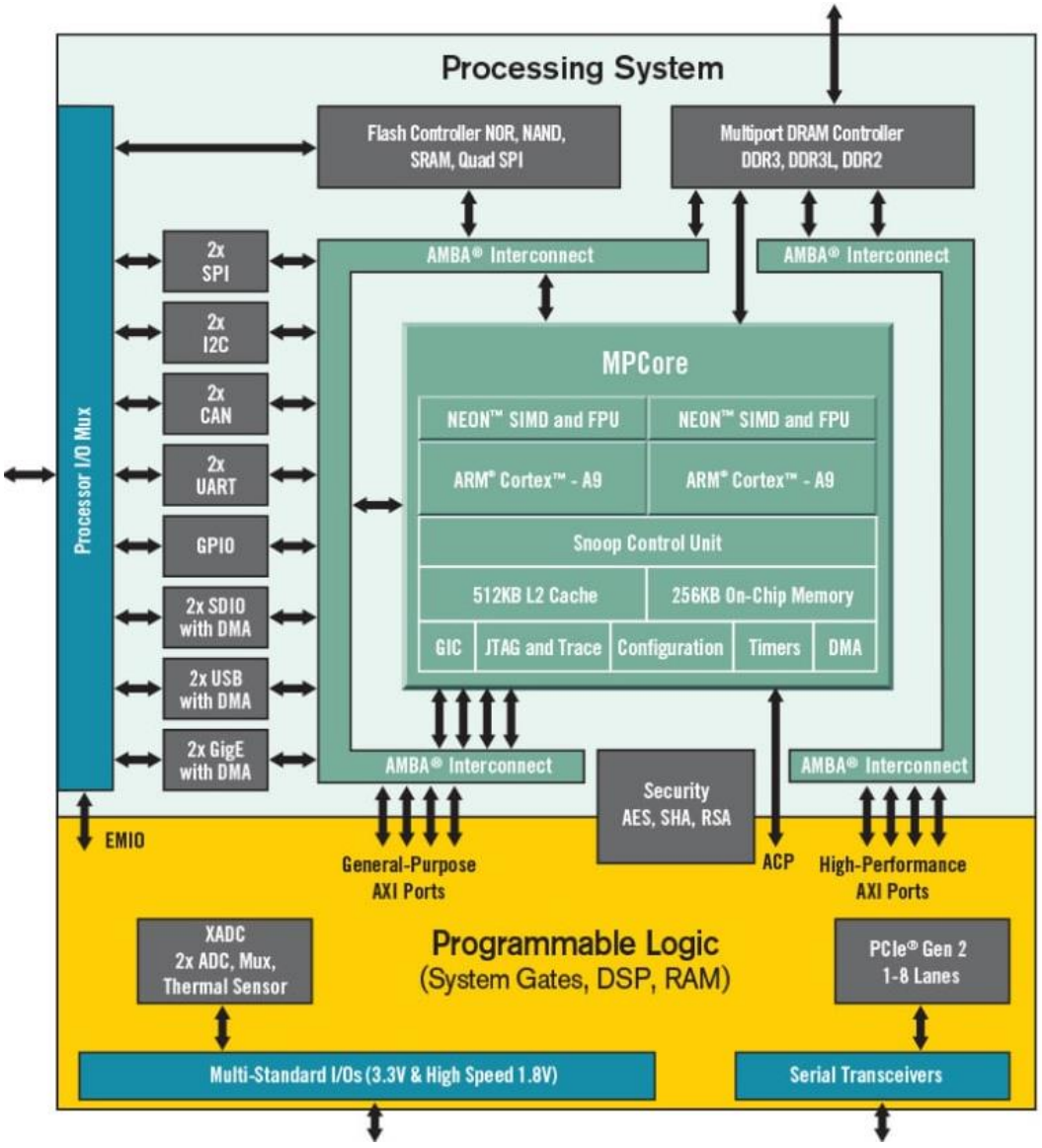


Рисунок 1.4 – Загальна структура PS частини Zynq-7000

ARM Cortex-A9 – двоядерний процесор, який підтримує обчислення з високою продуктивністю. Він може працювати на частотах до 1 ГГц, залежно від конкретної моделі Zynq. Система підтримує підключення до динамічної пам'яті DDR3, що забезпечує високу швидкість обміну даними з процесором. Cortex-A9 має вбудовану кеш-пам'ять (L1 і L2), яка прискорює обробку даних. Окрім того, PS частина має різні периферійних інтерфейси, такі як UART, I2C, SPI, GPIO, USB, Ethernet. Ці інтерфейси використовуються для підключення до зовнішніх пристроїв або для організації комунікацій із іншими мікросистемами. До інтерфейсів для зв'язку з PL частиною відносяться AXI інтерфейси (AXI GP, AXI HP, AXI ACP). Ці інтерфейси використовуються для з'єднання між процесором ARM і FPGA-логікою, що дозволяє реалізувати високошвидкісні та паралельні обчислення.

1.3 Особливості со-проєктування вбудованих систем на базі SoC

Саме використання SoC, як інтегрованої схеми, що включає всі основні компоненти, необхідні для функціонування вбудованої системи, на одному кристалі, дозволяє в повній мірі скористатися перевагами спільного проєктування апаратного та програмного забезпечення, забезпечуючи тим самим високу ефективність, зменшення часу на розробку та гнучкість. На рисунку 1.5 представлено загальний вигляд спільного проектування апаратного/програмного забезпечення, на якому еліпси позначають сутності даних/інформації в проектуванні системи, а квадрати – процеси, дії або діяльність проектування системи. Спільне проектування все ще є відносно новою, швидкозмінною галуззю, тому немає єдиного встановленого стандарту щодо того, як це слід робити, і існує багато варіацій.

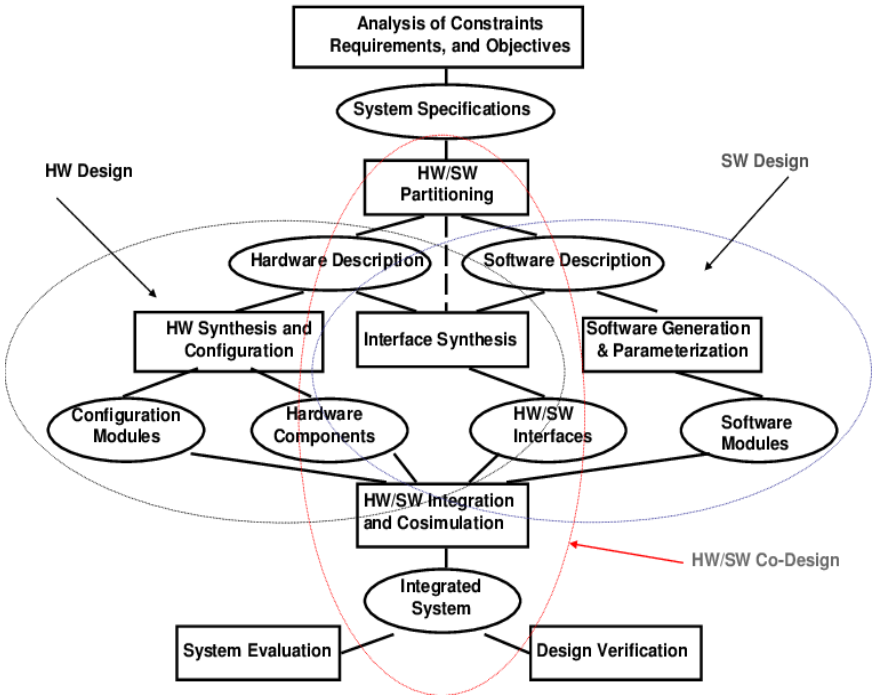


Рисунок 1.5 – Hardware-Software co-проєктування

У традиційному проєктуванні апаратні та програмні частини розробляються окремо, що часто призводить до неузгодженості між ними. У методології со-проєктування апаратна частина і програмне забезпечення проектуються паралельно, що дозволяє досягти кращої інтеграції між компонентами. Це дозволяє розробникам знижувати час виходу на ринок, оскільки програмісти можуть розробляти програмне забезпечення одночасно з розробкою апаратної частини.

Процес спільного проєктування починається з незалежного від архітектури опису функціональності запланованої системи, аналізу обмежень та вимог до системи, а також формулювання цілей.

Потім система специфікується за допомогою концептуальної моделі (що стосується функціональності та поведінки) або мови програмування (наприклад, VHDL, Verilog, тощо), яка потім компілюється у внутрішнє представлення, таке як опис потоку керування даними. Ця специфікація/опис служить єдиним системним представленням, яке може представляти апаратне або програмне забезпечення. Функціональне (або архітектурне) розділення апаратного/програмного забезпечення виконується на цьому

єдиному представленні. Після завершення цього кроку синтезується апаратне, програмне забезпечення та пов'язані з ними інтерфейси.

Процес розділення є ітеративним, і якщо оцінювання не відповідає необхідним цілям, генерується та оцінюється інший розділ апаратного/програмного забезпечення. Головна задача, що стоїть перед розробниками вбудованих систем на SoC, полягає в тому, що треба проаналізувати архітектуру конкретного кристалу, для того щоб обрати оптимальне рішення щодо розподілу функцій між двома секціями (рис. 1.6).

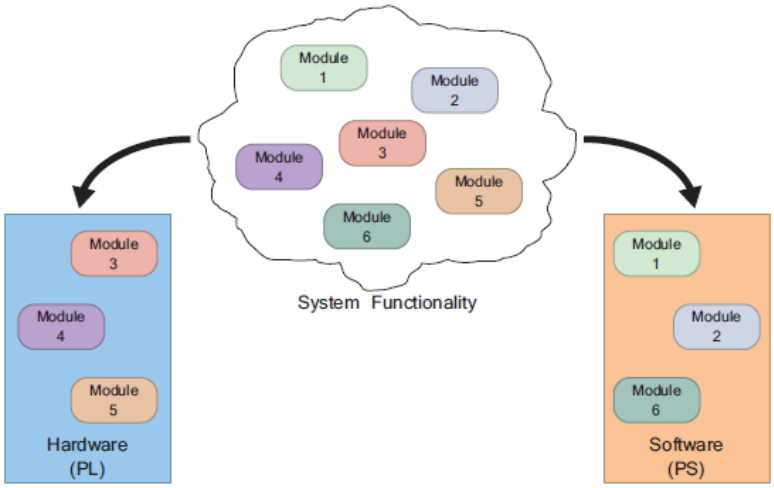


Рисунок 1.6 – Ідея апаратного/програмного розділення

Для спільного проєктування використовуються спеціалізовані інструменти, які дозволяють моделювати апаратні компоненти разом з програмним забезпеченням, а потім перевіряти їх на працездатність.

Для створення проектів на SoC фірми Xilinx (зараз AMD) використовуються два інструмента: Vivado Design Suite та Vitis Unified IDE (рис. 1.7). Таким чином, Vivado пропонує апаратно-орієнтований підхід до проєктування обладнання [4], тоді як Vitis пропонує програмно-орієнтований підхід до розробки як апаратного, так і програмного забезпечення [5].

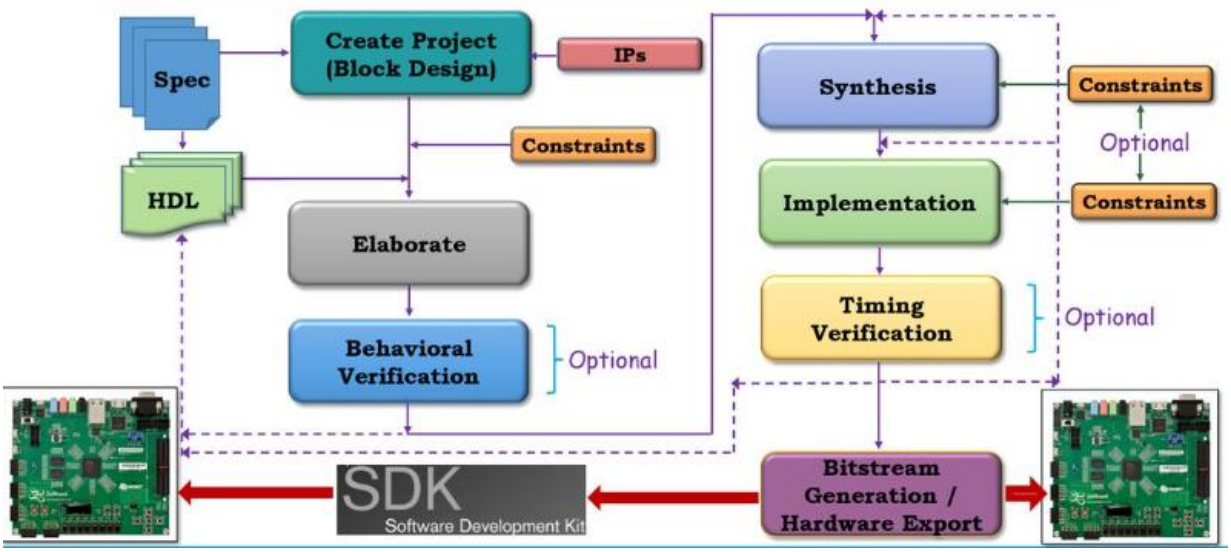


Рисунок 1.7 – Процес проєктування вбудованих систем SoC фірми Xilinx

Перший етап включає визначення специфікацій та вимог до системи. Потім, на етапі проектування системи, різні завдання (функції) призначаються або програмованій логіці (PL), або системі обробки (PS) у процесі, відомому як розподіл завдань. Цей етап є вирішальним, оскільки загальна продуктивність системи залежить від призначення завдань/функцій найбільш підходящій технології – апаратній чи програмній.

Після цього проводиться розробка та тестування апаратного та програмного забезпечення.

Розробка логіки на PL частині полягає у розробці власних IP-блоків на VHDL/Verilog або/та використанні готових IP-блоків з бібліотеки Vivado. Візуально створюється система, додається процесор (PS), логіка (PL), з'єднуються блоки через шини та інтерфейси (наприклад, AXI), налаштовуються адреси та зв’язки.

Далі виконується синтез, тобто запускається процес переведення схеми з логічного опису у форму, яку можна реалізувати на FPGA. Vivado перевіряє правильність та відповідність проєкту.

Далі виконується імплементація, коли Vivado розміщує логіку на фізичних елементах кристалу (LUT, FF, DSP, пам’ять) і проводить

трасування зв’язків. Перевіряється, чи система вкладається в часові обмеження (таймінги).

Після успішної імплементації генерується бітовий файл (.bit), який завантажується в PL частину FPGA. Також створюється файл опису платформи (.hwh), потрібний для програмної розробки.

Для експорту апаратної платформи створюється спеціальний файл (.xsa), що описує апаратну конфігурацію. Цей файл потім використовується в середовищі Vitis для написання програмної частини.

Vitis не створює апаратну частину, а імпортує її з Vivado (XSA-файл).

Далі необхідно розробити C або C++ програму, яка буде виконуватися на: ARM-процесорі – як standalone-програма (bare-metal), або під керуванням ОС (Linux, FreeRTOS).

У програмі можна керувати периферією, взаємодіяти з PL, обробляти дані. Опційно можна додати апаратні прискорювачі. Для цього використовують систему Vitis HLS (High-Level Synthesis), в якій можна написати частину алгоритму на C/C++ і перетворити його в апаратну логіку (HDL), що буде інтегруватися у PL як прискорювач (Hardware Kernel).

Після конфігурації Vitis компілює код для PS, генерує драйвери для IP, що знаходяться у PL, створює повну програму для запуску на SoC, збирає SD-образ для запуску з карти пам’яті (опційно).

Також можна завантажити програму у SoC через JTAG (у режимі debug), або з SD-карти/Flash у звичайному режимі.

Підтримується повноцінне налагодження коду, перегляд змінних, використання точок зупину тощо.

1.4 Мета та постановка завдання

Мета роботи – спільне HW/SW проєктування моделі багатофункціонального ручного годинника на базі SoC Zynq 7000 з використанням САПР Vivado Design Suite and Vitis Unified IDE.

Згідно з поставленою мети необхідно вирішити наступні задачі:

1) проаналізувати особливості спільного HW/SW проєктування вбудованих систем на базі гібридних SoC;

2) проаналізувати структуру SoC Xilix Zynq7000;

3) розробити модель багатофункціонального годинника, розподіливши частину функціонала для реалізації між PL та PS частинами SoC;

4) обрати додаткові апартні елементи для створення прототипу;

5) зібрати прототип багатофункціонального годинника на базі SoC;

6) в середовищі Vivado розробити VHDL модель для реалізації на FPGA;

7) розробити Testbench для функціональної верифікації VHDL моделей всіх компонентів, проаналізувати часові діаграми;

8) побудувати блочну діаграму, додати необхідні IP блоки та користувальницький IP блок (розроблену VHDL модель), згенерувати XSA-файл для завантаження у Vitis;

9) розробити проєкт у середовищи Vitis на базі згенерованого XSA-файла для роботи з IPS дисплеєм;

10) розробити бібліотеку шрифтів для виводу українських та англійських літер, цифер та символів на екран;

11) додати до основної програми сигнали, що поступають через інтерфейс AXI GPIO для координації роботи PL та PS частини;

12) завантажити програму у SoC через JTAG та перевірити працездатність створеного прототипу.

2 СТВОРЕННЯ МОДЕЛІ БАГАТОФУНКЦІОНАЛЬНОГО ГОДИННИКА НА БАЗІ SoC

В якості прикладу вбудованої системи для HW/SW со-проєктування у середовищі Vivado та Vitis було обрано багатофункціональний годинник з функцією секундоміра. Розглянемо більш детально саму модель.

2.1 Модель годинника та загальний алгоритм її роботи

Розглянемо загальну концепцію запропонованої моделі багатофункціонального годинника на базі SoC Zynq7000 (рис. 2.1).

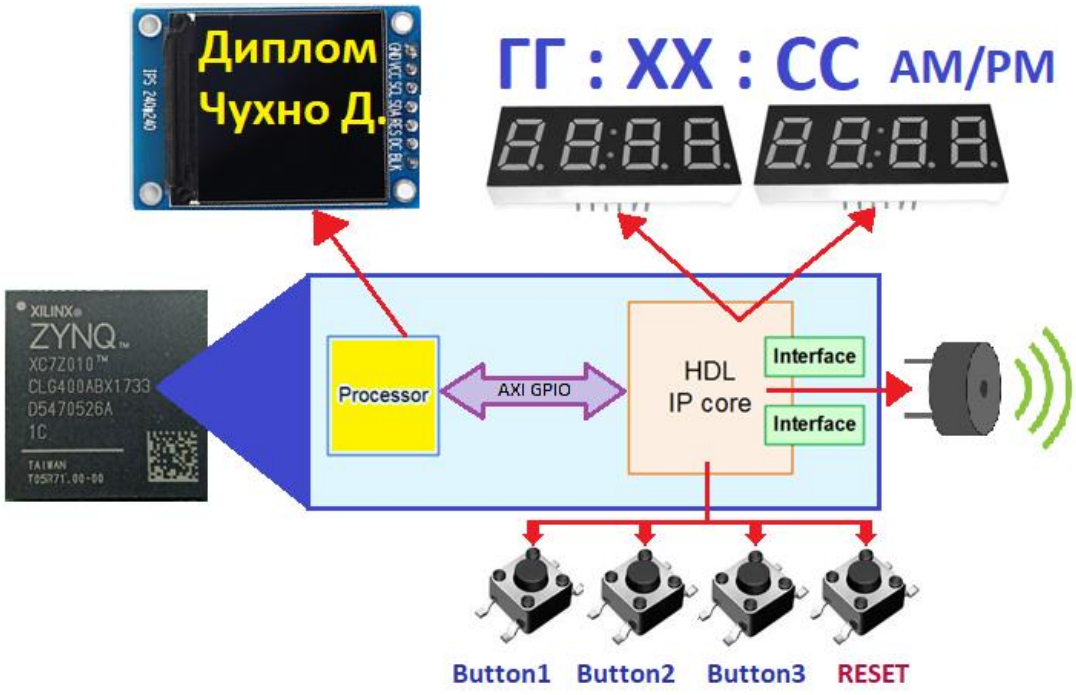


Рисунок 2.1 – Загальна модель реалізації годинника на базі SoC

На етапі проєктування будь-якої вбудованої системи на базі гібридних SoC головною задачею є розподіл різних завдань (функції) між PL та PS частинами. Як показано на рисунку 2.1 основні функції будуть реалізовуватися на PL, включно з виводом на семи-сегментні дисплеї,

аналізом кнопок та керуванням п’єзодинаміком. А за вивід додаткової інформації на IPS дисплей буде відповідати процесор на PS частині.

Також стає питання раціонального використання існуючих портів плати для керування годинником та вибір додаткових апаратних блоків, що будуть розглядатися в наступному розділі.

Розглянемо алгоритм роботу прототипу багатофункціональног годинника, що запропоновано в проєкті (рис. 2.2).

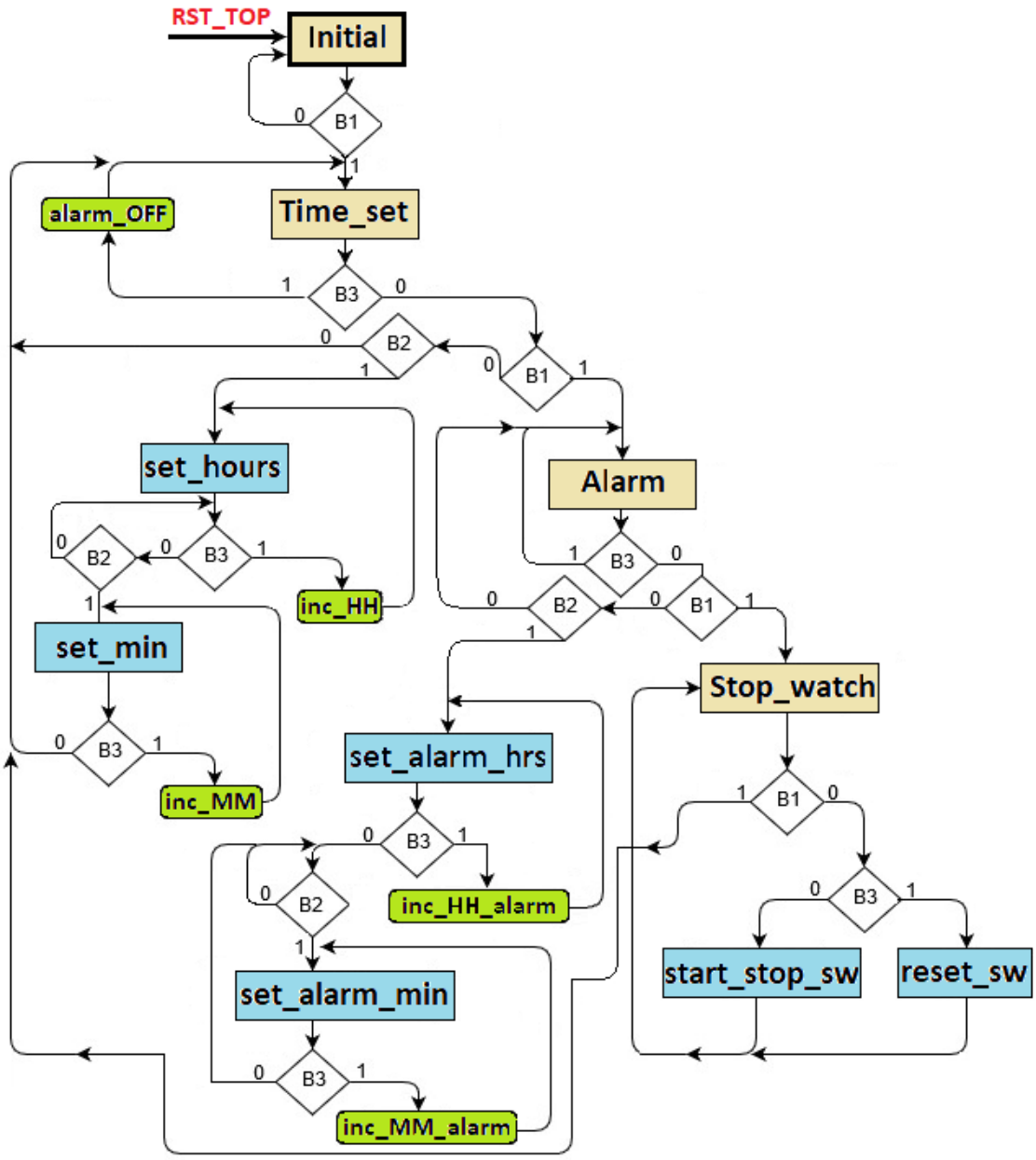


Рисунок 2.2 – Граф-схема роботи багатофункціонального годинника

На рисунку 2.2 видно, що запронована модель годинника буде мати три основні режими роботи: режим налаштування годинника (Time), режим налаштування будильника (Alarm) та режим секундоміра (Stop_watch) та декілька додаткових.

Налаштування годинника буде відбуватися за допомоги трьох кнопок (B1, B2 та B3):

– функція кнопки B1 – переключення між режимами Time → Alarm → Stop_watch →знову Time;

– функції кнопок B2 та B3 залежать від режиму.

Розглянемо детально кожен з режимів роботи багатофункціонального годинника.

1. Режим налаштування годинника (Time). Дисплей показує час в 12-годинному форматі, а також значок, що показує ранок (AM) чи вечір (PM), використовуючи формат ГГ:ХХ:СС А/Р. Якщо в цьому режимі спрацював будильник, його можна вимкнути, натиснувши кнопку B3.

Натискання B2 змінює перключає на режим «Встановити години» (set_hours) або «Встановити хвилини» (set_min) і назад у режим часу (time).

2. В режимах «Встановити години» (set_hours) або «Встановити хвилини» (set_min) кожне натискання B3 збільшує години або хвилини на 1.

3. Режим будильника (Alarm). Дисплей показує час, на який було встановлено будильник та значок ранку (AM) чи вечора (PM), використовуючи формат ГГ:ХХ А/Р. Натискання B2 змінює режим на «Встановити години будильника» (set_alarm_hours) або «Встановити хвилини будильника» (set_alarm_min), а потім назад на «Будильник» (Alarm).

4. В режимах «Встановити години будильника» (set_alarm_hours) або «Встановити хвилини будильника» (set_alarm_min) кожне натискання B3 збільшує години або хвилини будильника на 1. Після того, як будильник почне дзвонити, він дзвонитиме 50 секунд, а потім вимкнеться. Його також можна вимкнути вручну, натиснувши B3 у режимі часу (Time).

5. Режим секундоміра (Stop_watch). Дисплей відображає час секундоміра у форматі MM:CC.cc (де cc – соті частки секунди). Натискання B2 запускає лічильник часу, повторне натискання B2 зупиняє його, потім натискання B2 перезапускає його і так далі. Натискання B3 скидає секундомір. Після запуску секундоміра він продовжуватиме працювати, навіть якщо наручний годинник перебуває в режимі часу або будильника!

2.2 Програмна реалізація на PL частині SoC у середовищі Vivado

Основний функціонал багатофункціонального годинника був реалізовано на PL частині мовою VHDL [6] у середовищі Vivado.

Розглянемо структурну схему VHDL-моделі (рис. 2.3).

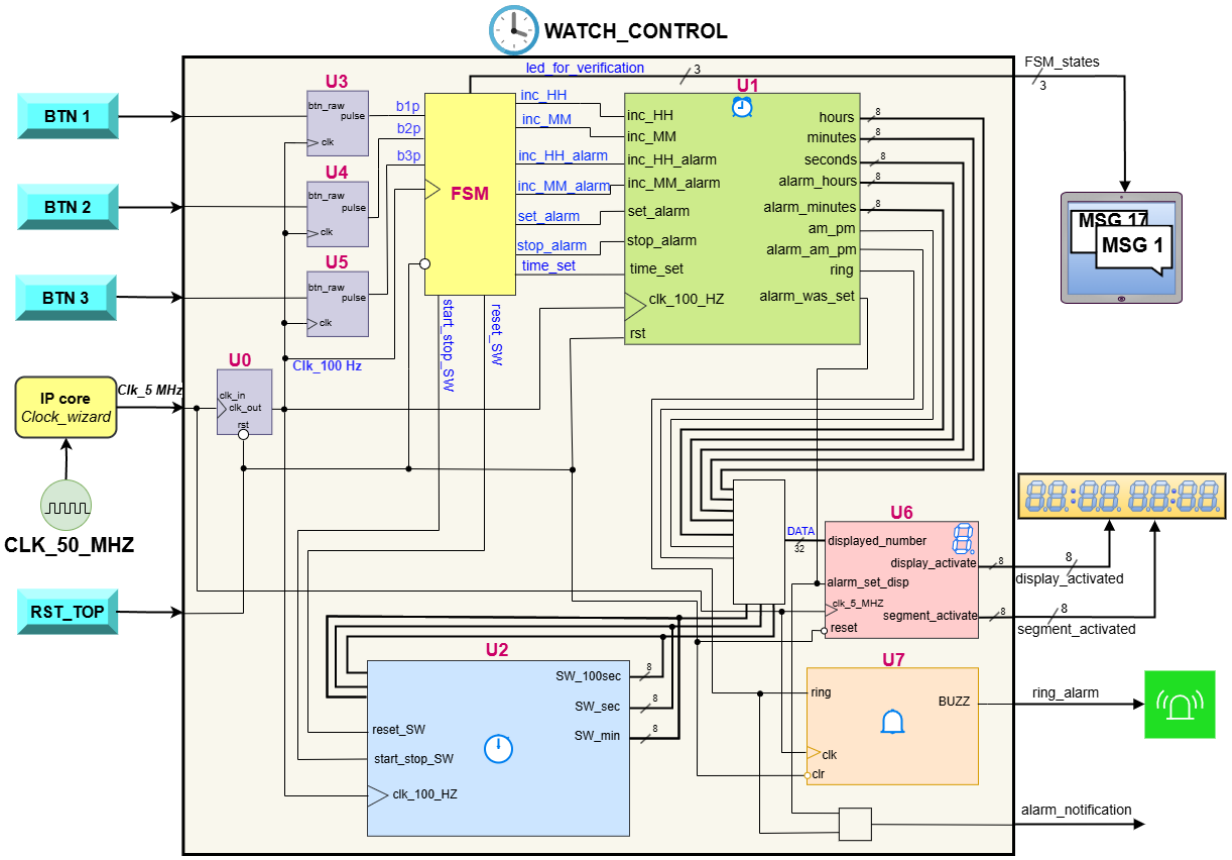


Рисунок 2.3 – Структурна схема VHDL-моделі

Згідно з цією структурною схемою керування годинником відбувається в моделі верхнього рівня (watch_control), основний алгоритм (рис. 2.2) якого

описаний у вигляді кінцевого автомата Мілі [7]. Граф переходів представлено не рисунку 2.4.

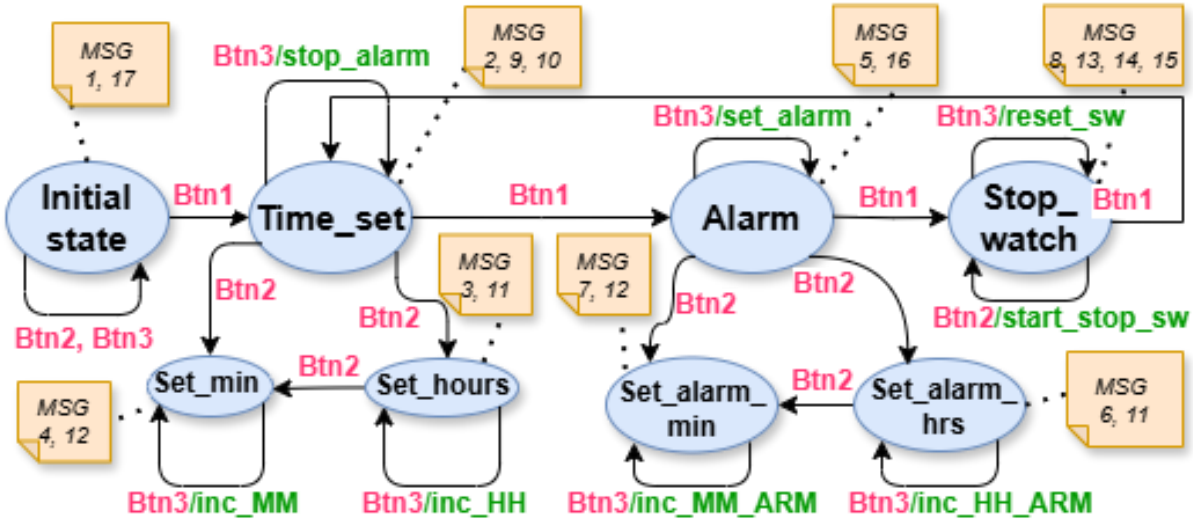


Рисунок 2.4 – Граф переходів автомата

Таким чином, необхідно створити 8 станів, кожен з яких буде відповідати за конкретний режим роботи годинника, включно з початковим.

Також треба передбачити сигнал скидання (RST_TOP) та сигнал (MSG), що буде передаватися з PL частини на PS частину і відповідати за координацію роботи основного блоку керування годинником та блоку керування IPS дисплеєм.

Модель верхнього рівня (WATCH_control) складається з 7 компонентів U1 – U7.

1. Блок U0 (DIV_5MHZ_to_100HZ) відповідає за поділ частоти синхросигнала для його зменшення від 5 MHz до 100 Hz.

2. Блок U1 (WATCH) відповідає за все, що стосується годинника та будильника. Це змішана модель, яка має поведінкову частину, що відповідає за формування всіх сигналів для керування лічильниками, та структурну частина, яка в свою чергу також складається в 5-х блоків, а саме:

– блоки U1_1, U1_2 та блок U1_4 (CTR_60) – це лічильники хвилин/секунд для годинника/будильника;

– блок U1_3 та U1_5 (CTR_12) – це лічильник годин для годинника /будильника.

3. Блок U2 (STOPWATCH) відповідає за все, що стосується секундоміра. Це змішана модель, яка має поведінкову частину, що відповідає за формування всіх сигналів для керування лічильниками, та структурну частина, яка в свою чергу складається в 3-х блоків, а саме:

– блоки U2_1, U2_2 (CTR_60) – це лічильники хвилин/секунд для секундоміра;

– блок U2_3 (CTR_100) – це лічильник сотен секунд.

4. Блоки U3 – U5 (ANTI_BOUNCED_PULSE) відповідають за антибрязкотну схему кожної кнопки.

5. Блок U6 (SEVEN) відповідає за керування семи-сегментними дисплеями.

6. Блок U7 (BUZZER) відповідає за формування мелодії п’єзодинаміка при спрацьовуванні будильника.

На рисунку 2.5 показана ієірархія файлів в проєкті в Vivado.

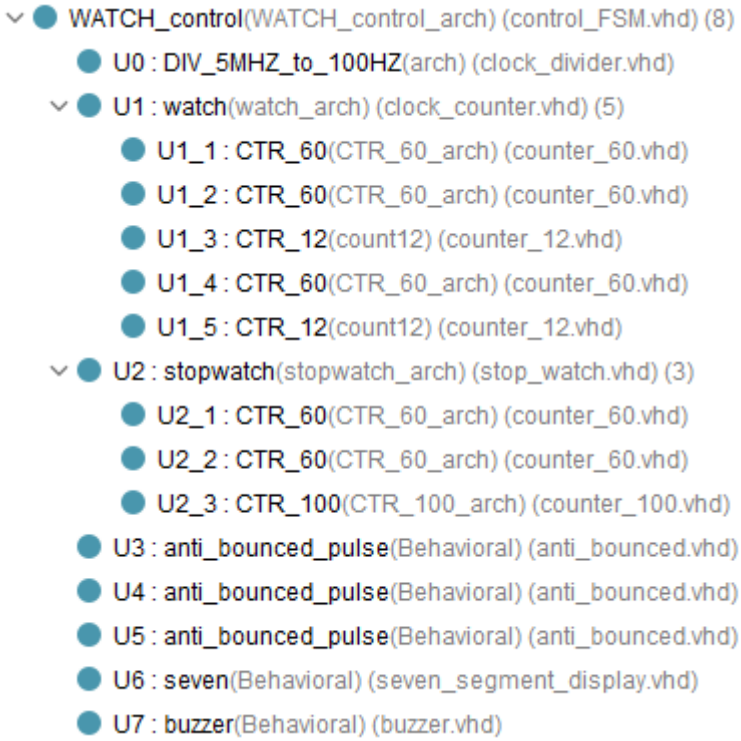


Рисунок 2.5 – Ієрархія файлів в проєкті в Vivado

Повний VHDL-код для кожного компонента, включно з моделлю верхнього рівня представлено у додатку А.

Далі розроблена VHDL модель буде додана в якості користувальницького IP-блока для блочної діаграми проєкта.

2.2.1 Створення файлу обмежень

Xilinx Design Constraints (XDC) – це текстовий файл, який використовується у процесі розробки для ПЛІС зокрема для:

– задання часових обмежень (timing constraints);

– призначення сигналів до фізичних виводів (pin assignments);

– інших фізичних обмежень для реалізації апаратного дизайну.

Для створення такого файлу у дереві проєкту (Sources) клацнемо правою кнопкою миші на "Constraints" → Add Sources. Оберемо "Add or Create Constraints". Натиснемо "Create File", оберемо XDC та напишемо будь-яке ім’я. Після створення файл з’явиться в проєкті, і його можна редагувати прямо у вбудованому редакторі Vivado. Важливо зазначити, що ви можете знайти XDC-файл для вашої плати. Цей файл містить списки всіх виводів FPGA, підключених до фізичних виводів на платі, організовані за групами зовнішніх компонентів для зручності. Розкоментуйте потрібні вам контакти, видаливши початковий символ #, і переконайтеся, що get_ports названо за іменами ваших входів/виходів. Відредагуйте файл і натисніть кнопку «Зберегти».

Для керування годинником потрібно підключити кнопки, генератор тактового сигналу.

Лістинг 2.6 – Фрагмент XDC файл

```
set_property -dict {PACKAGE_PIN N18 IOSTANDARD LVCMOS33} [get_ports CLK_50MHZ]
set_property -dict {PACKAGE_PIN V20 IOSTANDARD LVCMOS33} [get_ports RST_TOP]

set_property -dict {PACKAGE_PIN U19 IOSTANDARD LVCMOS33} [get_ports Button1]
set_property -dict {PACKAGE_PIN U20 IOSTANDARD LVCMOS33} [get_ports Button2]
set_property -dict {PACKAGE_PIN T19 IOSTANDARD LVCMOS33} [get_ports Button3]
```

.....

```
set_property PACKAGE_PIN T20 [get_ports {GPIO_0_0_tri_io[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {GPIO_0_0_tri_io[0]}]
set_property PACKAGE_PIN R18 [get_ports {GPIO_0_0_tri_io[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {GPIO_0_0_tri_io[1]}]
set_property PACKAGE_PIN N17 [get_ports {GPIO_0_0_tri_io[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {GPIO_0_0_tri_io[2]}]
set_property PACKAGE_PIN R19 [get_ports {GPIO_0_0_tri_io[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {GPIO_0_0_tri_io[3]}]
set_property PACKAGE_PIN P20 [get_ports {GPIO_0_0_tri_io[4]}]
set property IOSTANDARD LVCMOS33 [get ports {GPIO_0_0_tri_io[4]}]
```

2.2.2 Створення блочної діаграми

Для створення блочного проєкту обираємо в меню в IP Integrator пункт Create Block Design, вводимо будь-яке ім’я (наприклад, MY_BD.bd). Цей файл з’явиться у папці Design Sources. А в головному вікні відкриється пусте поле. Тут ми будемо створювати блочний проєкт, додаючи готові IP блоки, а також ті, що ми описали на VHDL (кнопка + Add IP).

Розглянемо всі блоки, які знадобляться для цього проєкту детальніше.

1. ZYNQ7 Processing System – готовий IP блок (hard IP core), що діє як логічне з’єднання між PS та PL, допомагаючи інтегрувати спеціальні та вбудовані IP блоки. У полі пошуку ввели proc, щоб знайти цей блок (рис. 2.6). Двічі клацнув IP блок буде додано до діаграми.

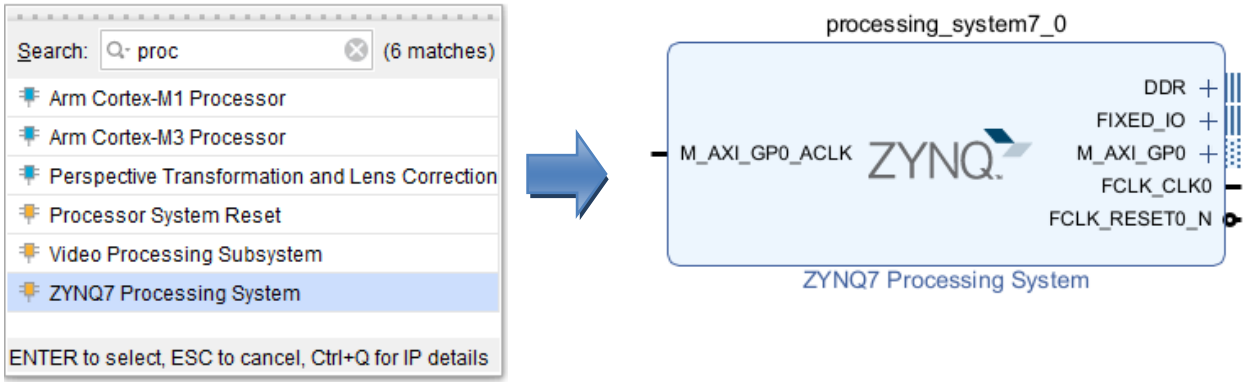


Рисунок 2.6 – Додавання блоку ZYNQ7 Processing System

Подвійне клацання на цьому IP блоці відкриває вікно майстра налаштувань (рис. 2.7).

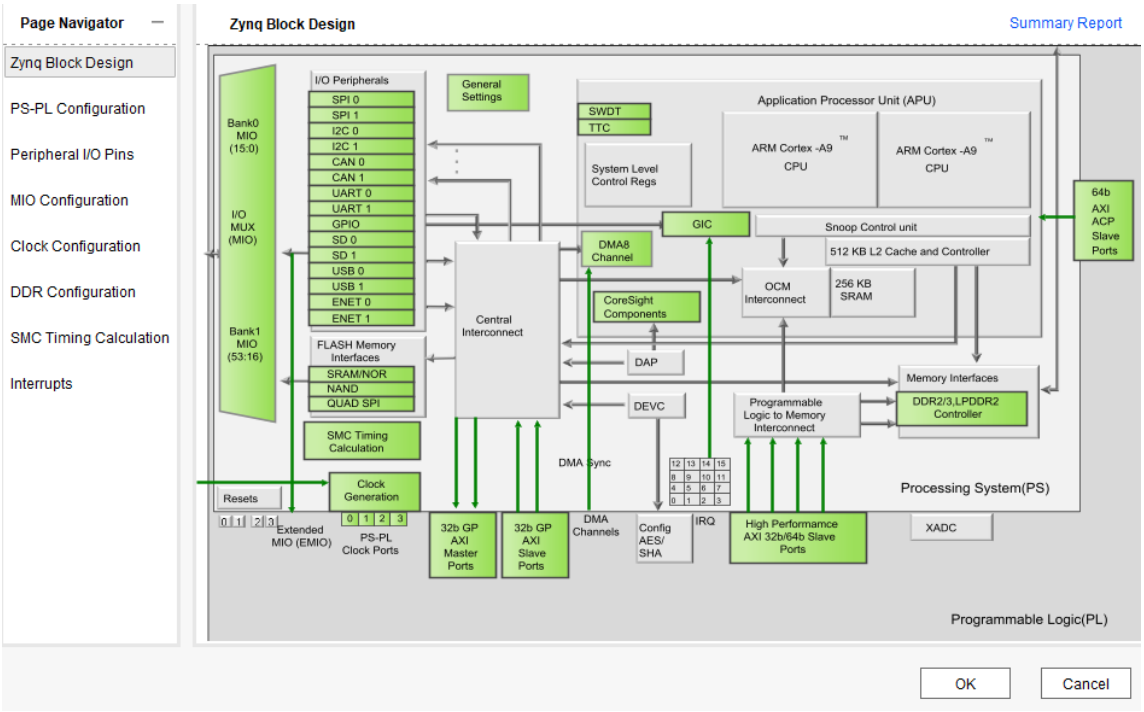


Рисунок 2.7 – Вікно майстра налаштувань блоку ZYNQ7 Processing System

Знаходимо опцію Clock Configuration, щоб встановити необхідну тактову частоту, що потрібна для PL частини з 4-х існуючих (вкладка PL Fabric Clocks). Обираємо для FCL_CLK0 частоту 50MHz.

Знаходимо опцію DDR Configuration, щоб налаштувати пам’ять. На вкладці DDR Controller Configuration згідно зі специфікацією до плати EBAZ4205 обираємо в якості компонента пам’яті (Memory Part) MT41K128M16JT-125 розмірністю (Effective DRAM Bus width) 16 bit.

Знаходимо опцію MIO Configuration, щоб налаштувати порти для керування LCD дисплеєм. На вкладці I/O Peripherals обираємо GPIO → EMIO GPIO (Width): 5. Таким чином ми налаштували п’ять GPIO портів через EMIO для керування наступними параметрами LCD дисплея: підсвічування (Backlight, BL), сигнал передачі даних чи команд (Data/Command, D/C), сигнал синхронізації (SPI clock, SCL),дані (SPI data, SDA), скидання (Reset, RES). Всі інші галочки прибираємо. Після завершення всіх налаштувань, тиснемо кнопку OK. Збережіть параметри за замовчуванням у спливаючих параметрах, натисніть «OK».

Запускаємо автоматизацію блочної діаграми, натискаючи під панеллю інструментів кнопку (Run Block Automation). Щоб завершити налаштування системи обробки Zynq7:

– підключаємо сигнал FCLK_CLK0 до сигналу M_AXI_GP0_ACLK: для цього клікаємо на перший сигнал і з'являється курсор у вигляді олівця; тягнемо лінію до другого сигналу, який підсвічується зеленою галочкою;

– робимо порти GPIO_0 зовнішніми: для цього клікаємо правою кнопкою миші і обираємо в меню опцію Make External.

В результаті наш IP блок буде виглядати наступним чином (рис. 2.8).

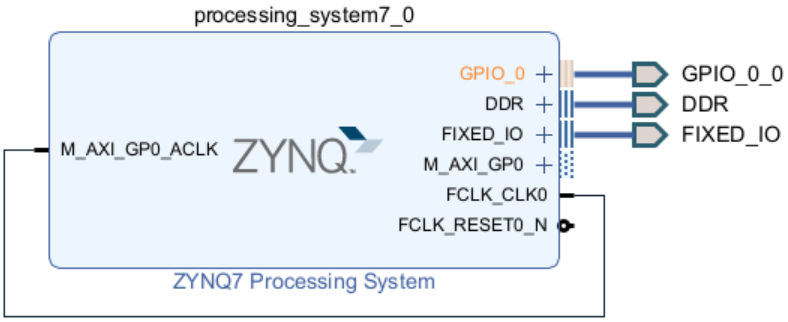


Рисунок 2.8 – Блок ZYNQ7 Processing System

2. AXI GPIO – готовий IP блок (hard IP core), який часто використовується в FPGA-системах для взаємодії з фізичними пристроями, такими як кнопки, світлодіоди, датчики, дисплеї та інше. У полі пошуку ввели назву, щоб знайти цей блок (рис. 2.9). Двічі клацнув IP блок буде додано до діаграми.

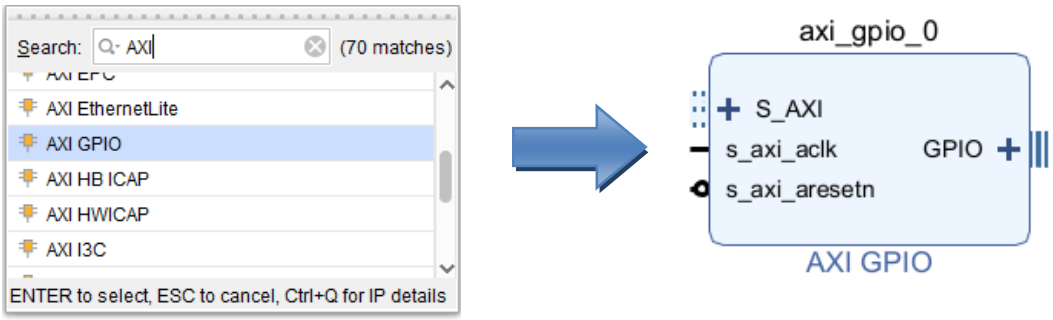


Рисунок 2.9 – Додавання блоку AXI GPIO

Натиснувши один раз на блок, відкриються його властивості, де можна поміняти назву цього блока (наприклад на axi_gpio_for_display).

AXI GPIO можна налаштувати з двома каналами:

– Канал 2 для прийому даних з RTL моделі (вхід).

– Канал 1 для передачі даних на дисплей (вихід).

Двічі клацнув на цей IP блок відкриється вікно налаштувань IP блока (рис. 2.10).

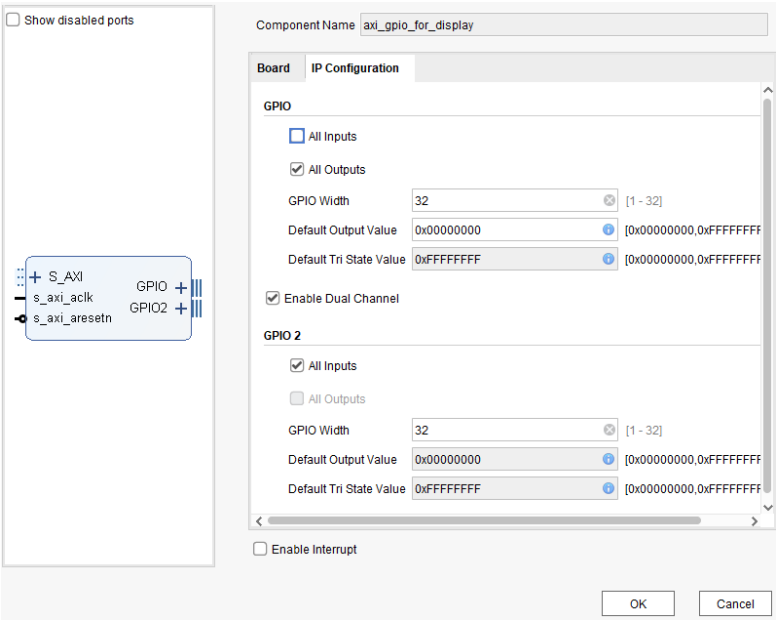


Рисунок 2.10 – Налаштування блока AXI GPIO

Для першого каналу оберемо наступні параметри:

– ширина шини (GPIO Width): 32;

– напрямок пінів (GPIO Direction): output, так як ми передаємо дані, то піни повинні бути виходами.

– переривання (Interrupt Enable): не відмічати галочкою, тому що для простого передавання даних цей параметр зазвичай не потрібен.

Оберемо опцію Enable Duo Chanel, щоб налаштувати другий канал:

– ширина шини (GPIO Width): 32;

– напрямок пінів (GPIO Direction): input, так як ми передаємо дані з RTL моделі;

– переривання (Interrupt Enable): не відмічати галочкою.

Блок AXI GPIO необхідно підключити до процесора. Для цього натискаємо під панеллю інструментів кнопку (Run Block Automation).

У діалоговому вікні, яке з'явиться, перегляньте кожну вкладку, щоб переглянути підключення і переконайтеся, що позначено прапорці S_AXI та GPIO. Натисніть OK, щоб запустити автоматизацію підключення та підключити блоки AXI GPIO до вашого процесора (рис. 2.11).

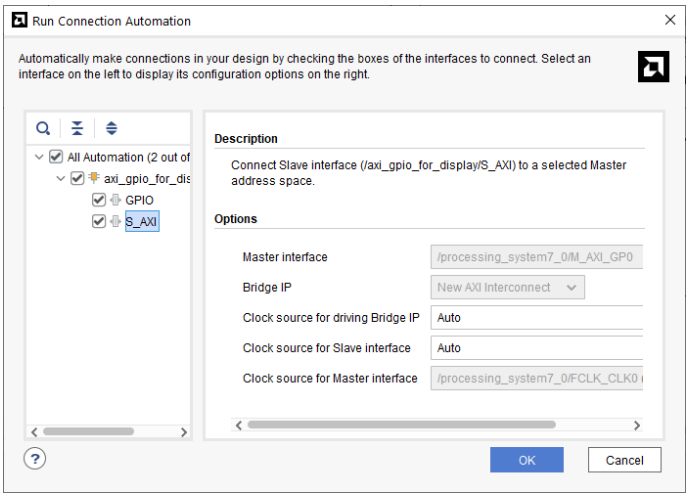
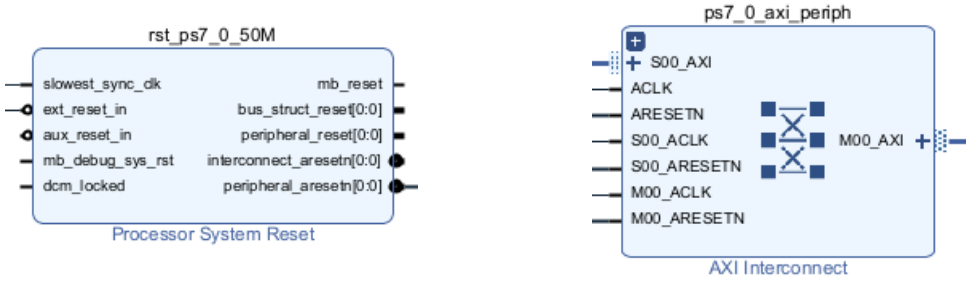


Рисунок 2.11 – Автоматизація підключення блока AXI GPIO до процесора

3. Автоматично до блочної діаграми додаються два блоки (рис. 2.12):



(а) (б)

Рисунок 2.12 – Блоки Processor System Reset (а) та блок AXI Interconnect (б)

Processor System Reset – це IP-блок, який використовується в Vivado для ініціалізації і скидання різних компонентів системи, таких як, програмованої логіки та периферійні пристрої.

AXI Interconnect – це IP-блок в Vivado, який використовується для з'єднання різних компонентів вашої системи, що використовують протокол AXI. Блок AXI Interconnect дозволяє забезпечити правильну маршрутизацію даних між Master (компонент, який ініціює транзакцію) і Slave (компонент, який обробляє транзакцію), зберігаючи ефективність та швидкість передачі даних у всій системі. У малоймовірному випадку, коли Vivado не вдасться правильно призначити адреси для кожного AXI GPIO блока, підключеного до вашого процесора, можливо, вам доведеться вручну встановити їхні адреси. Якщо це станеться, під час перевірки дизайну блоку з’являться помилки, і бітовий потік не зможе бути згенерований. Доступ до редактора адрес можна отримати через меню Window →Adress Editor. У робочому вікні на відповідній вкладці перевіряємо наступні показники (рис. 2.13).

Рисунок 2.13 – Редактора адрес для AXI GPIO блока

Після призначення адрес вручну конструкцію блоку слід повторно перевірити. Щоб перевірити блочний дизайн і переконатися, що немає помилок, клікаємо правою кнопкою миші на білому просторі вікна діаграми та обираємо опцію Validate Design чи відповідну кнопка на панелі ( ). Відкриється діалогове вікно повідомленням, що перевірка пройшла успішно. Натисніть OK, щоб закрити повідомлення.

До цієї базової блочної моделі необхідно додати створену VHDL модель у вигляді користувальницького IP-блока. Для цього необхідно у вікні «Sources» клікнути правою кнопкою миші на ТОП-моделі (Watch_control) і обрати в контекстному меню пункт «Add Module to Block Design» (рис. 2.14).

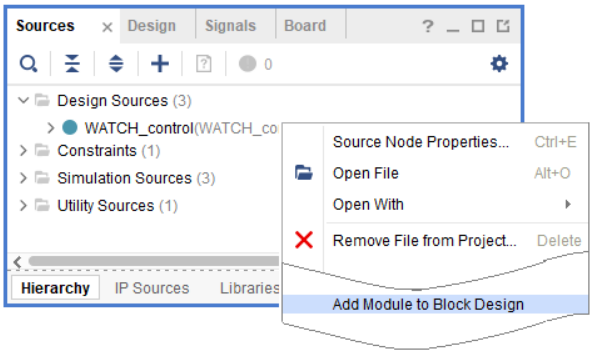


Рисунок 2.14 – Додавання користувацького RTL-блок до блочної моделі

Після цього у вікні діаграми з'явиться RTL-блок з іменем WATCH_control_0.

Останній кроком додаємо до створеної моделі IP блок для керування частотою синхросигнала. Для того, щоб додати готовий IP-блок [8] треба клікнути правою кнопкою миші на пустому місці діаграми або натиснути на кнопку + на панелі і обрати Add IP. В випадаючому списку можна ввести повну назву IP блока , що нас цікавить (Clock Wizard), або кілька літер. В цьому списку з’явиться необхідний нам блок. Подвійним кліком на його імені додаємо його до нашого проєкта (рис. 2.15).

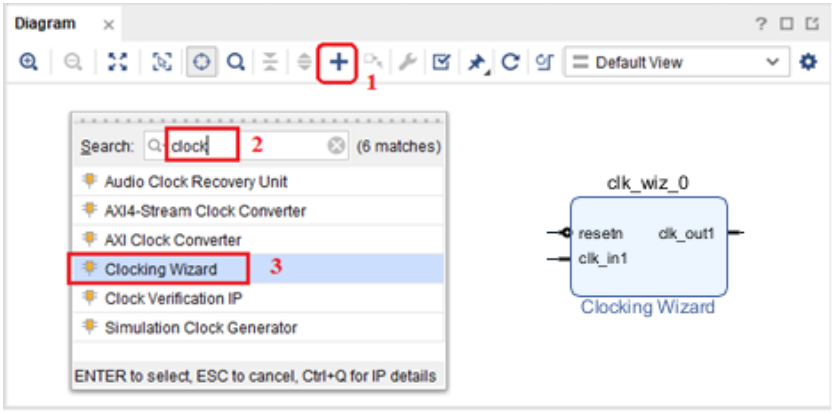
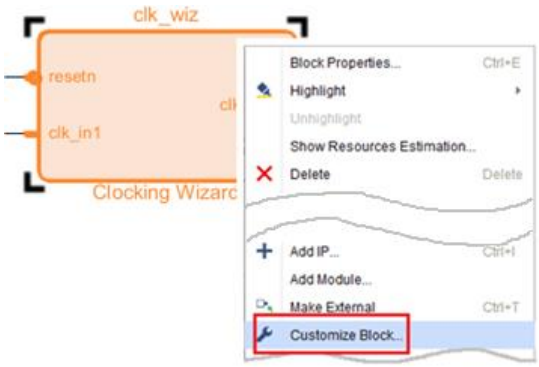


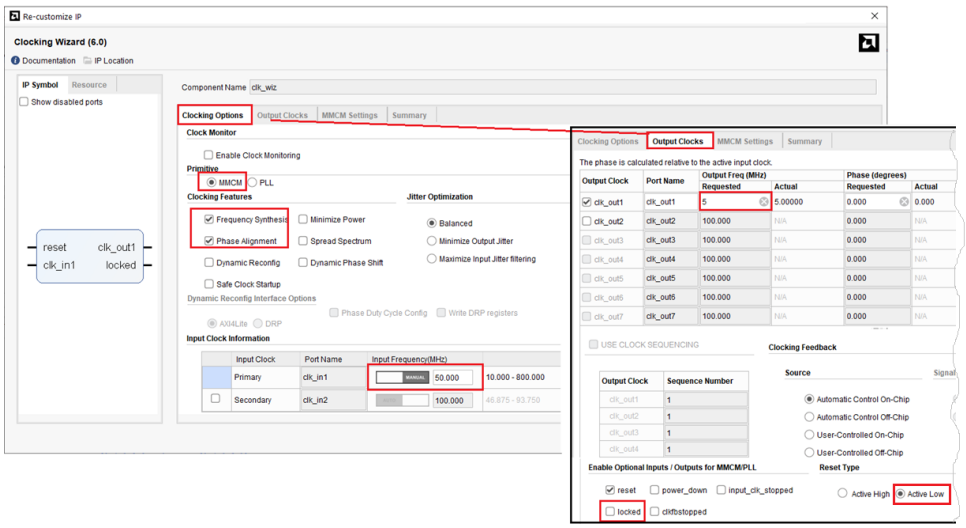
Рисунок 2.15 – Додавання готового IP блока Clock Wizard

Тепер його треба налаштувати. Клацніть правою кнопкою миші на блоці Clock Wizard та виберіть у контекстному меню пункт «Customize block» або просто двічі клацніть на ньому (рис. 2.16, а).

На вкладці «Clocking Options tab» залиште налаштування за замовчуванням для «Primitive (MMCM)», «Clocking Features» – Frequency Synthesis and Phase Alignment та «Jitter» – Balanced. Ці налаштування є найтиповішими для більшості тактових сигналів. На панелі «Input Clock Information panel» введіть вхідну тактову частоту вашої системи (50 МГц) та залиште значення за замовчуванням для Jitter (UI) та Source (Single ended clock capable pin). Далі перейдіть на вкладку «Output Clocks». Потім поставте прапорець, щоб увімкнути вихідний тактовий сигнал і встановити його вихідну частоту 5 МГц (рис. 2.16, б).



(а)



(б)

Рисунок 2.16 – Налаштування готового IP блока Clock Wizard

Далі натисніть «ОК». Залишилося під’єднати всі блоки. Створення з’єднань в IP Integrator є простим. Коли ви переміщуєте курсор поблизу

інтерфейсу або контактного порту на IP-блоці, він змінюється на значок олівця (). Потім ви можете клацнути на порт, утримувати ліву кнопку миші та намалювати з'єднання з будь-яким портом іншого блока. Під час створення з'єднань поруч із будь-яким підходящим місцем призначення з'являється зелена позначка (), яка виділяє доступні варіанти з'єднання для сигналу або інтерфейсу. Щоб зв'язати сигнали із зовнішніми портами на діаграмі, спочатку виберіть з'єднання контакту, шини або інтерфейсу, клацніть правою кнопкою миші та виберіть «**Make External**» у контекстному меню. Готова блочна модель представлена на рисунку 2.17.

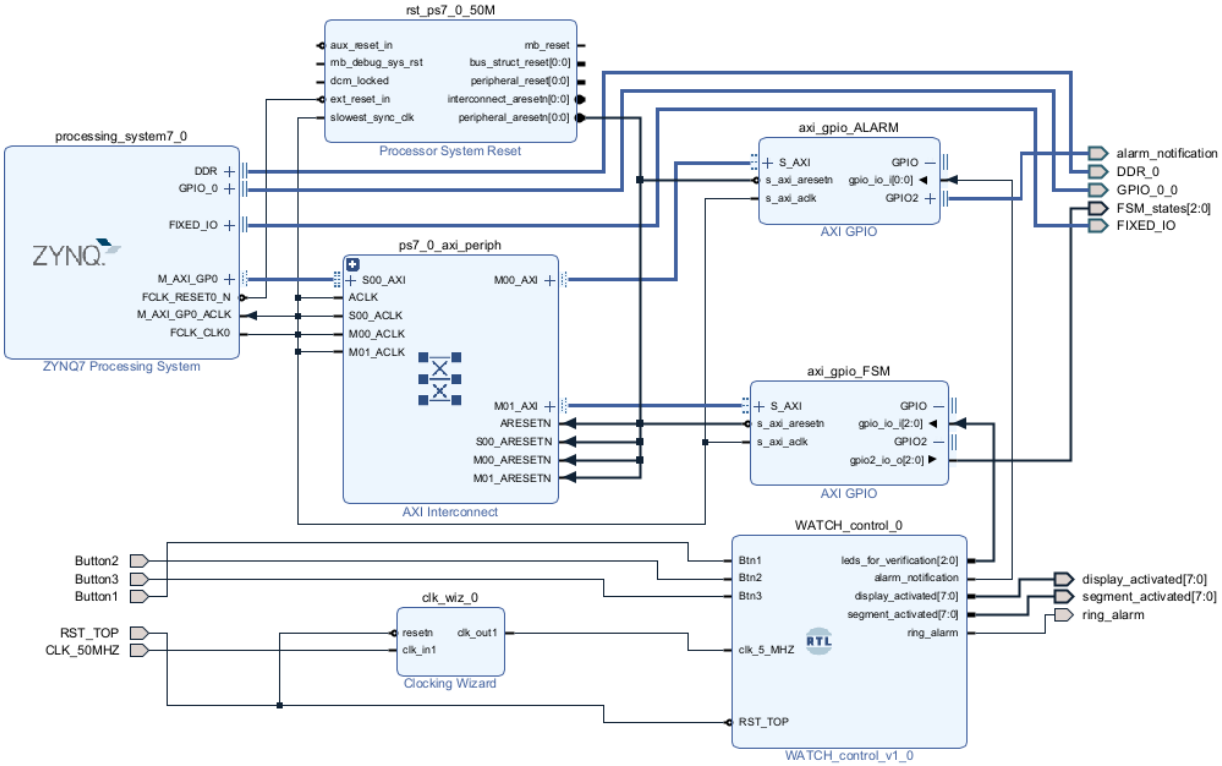


Рисунок 2.17 – Блочна модель проєкта

Після створення ієрархічного блоку та додавання потрібних IP-блоків знову запускаємо Validate Design (кнопка або F6).

Готову модель готуємо для подальшого використання шляхом генерування HDL Wrapper файл.

HDL Wrapper файл – це файл, що автоматично створюється на мові опису апаратури (VHDL або Verilog, в залежності від того, на якій ові писався сам проєкт) і який обгортає один або кілька внутрішніх модулів, інкапсулюючи їхню функціональність. Основна мета такого файлу – забезпечити зручний та узгоджений інтерфейс для взаємодії з цими модулями або для їх інтеграції у вищі рівні ієрархії апаратного дизайну (наприклад, у SoC). Для генерації цього коду у вікні Sources знайдемо свій блочний проєкт (MY_BD), клікнемо правою кнопкою миші по ньому та оберимо пункт «Create HDL Wrapper» в контекстному меню. З'явиться вікно з двома варіантами: «Copy and allow user edits» (дозволити редагувати вручну) та «Let Vivado manage wrapper and auto-update» (автоматичне оновлення). Виберимо другий варіант і натиснимо кнопку «OK» (рис. 2.18).

Рисунок 2.18 – Створення HDL Wrapper файла

Тепер цей файл з’явиться у вікні Design Sources (рис. 2.19).

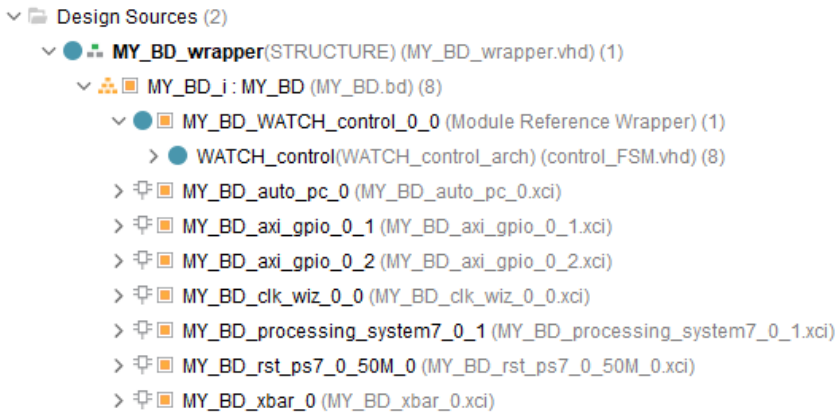


Рисунок 2.19 – Ієрархія файлі в проєкті Vivado

Щоб проаналізувати модель необхідно встановили його як Top Model та запустили RTL Analysis → Schematic. Відкриється згенерована RTL схема проєкта (рис. 2.20). Вона дуже велика, тому наведемо тільки її загальний вид.

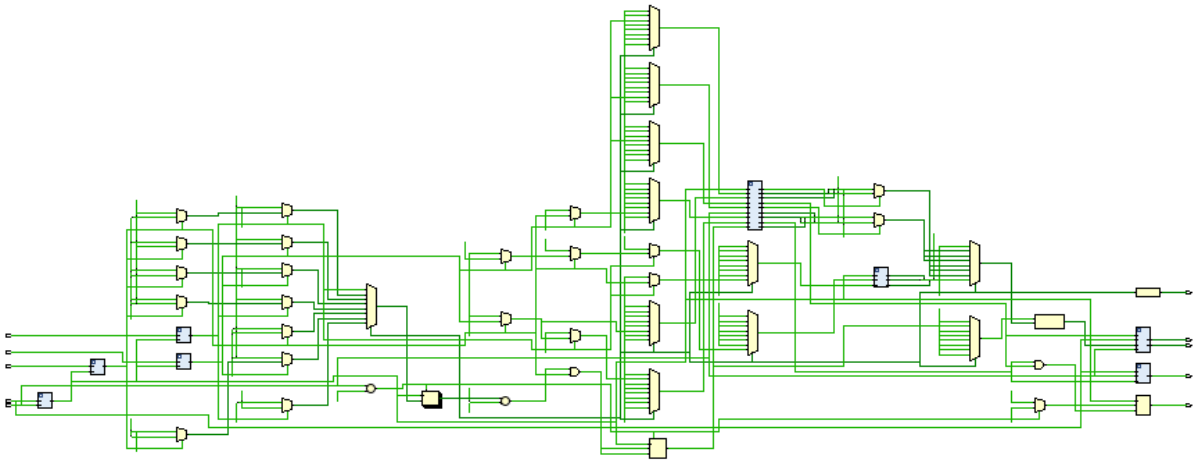
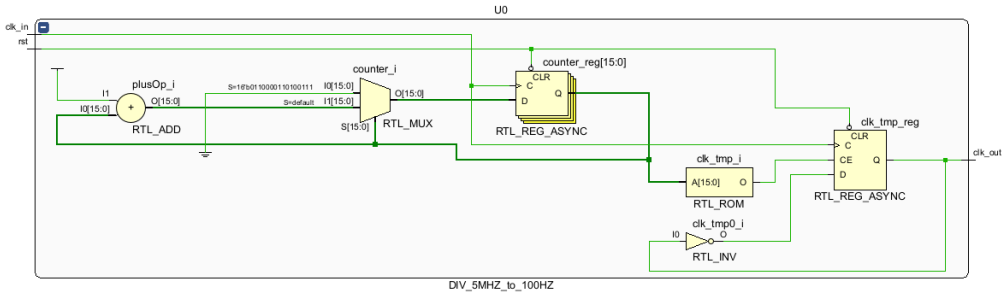


Рисунок 2.20 – Загальний вид RTL схеми

Але якщо наблизити, то можна розкрити кожен блок і проаналізувати схемну реалізацію. Наприклад, схема блока U0 (подільника частоти) представлена на рисунку 2.21, а, а на рисунку 2.21, б показана RTL схема блока U4 (антибрязкотної схеми).



(а)

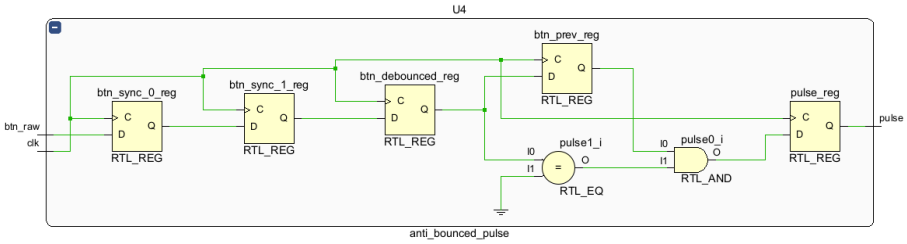
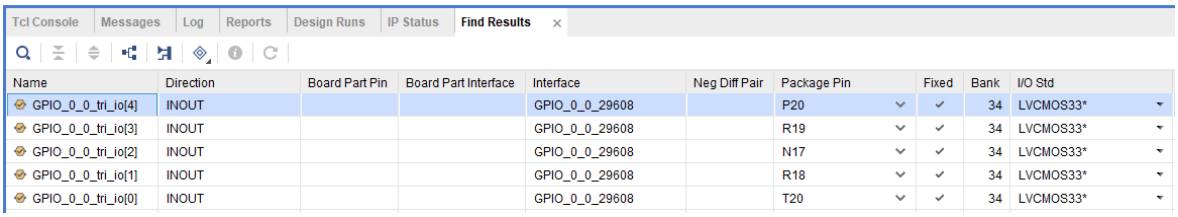


Рисунок 2.21 – RTL схема блока U0 (а) та U4

На панелі оберемо I/O ports (рис. 2.22, а) та налаштуємо порти для роботи з IPS екраном (рис. 2.22, б).

(а)



(б)

Рисунок 2.22 – Налаштування портів для роботи з IPS екраном

Це можна зробити і шляхом додавання опису цих портів безпосередньо до XDC файлу, так як ми це зробили для опису кнопок, світлодіодів, генератора тактового сигналу і т.п.

При збереженні проєкта з’явиться вікно, в якому треба обрати опцію Select an existing file (my_constraints.xdc), створений нами раніше і натиснути OK (рис. 2.23).

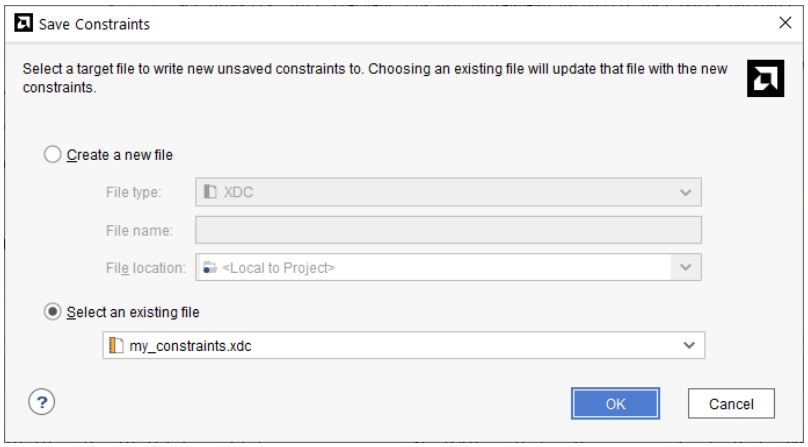
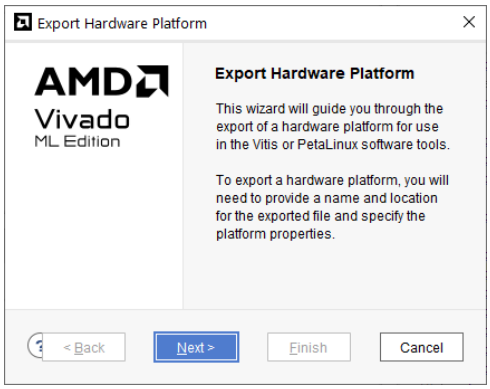


Рисунок 2.23 – Підключені порти до існуючого XDC файла

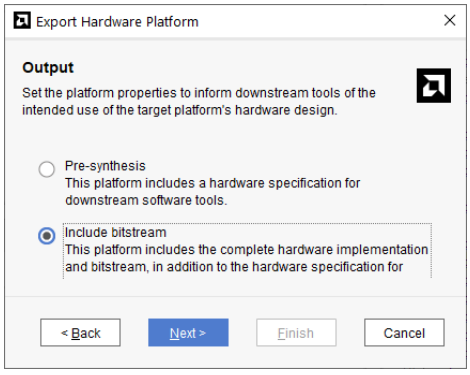
Виконаємо експорт розробленої VHDL моделі через меню File → Export → Export Hardware. Далі необхідно обрати платформу для експорту

апаратних засобів. У Vivado і Vitis існує два основні типи платформ для експорту апаратних засобів: фіксовані (Fixed Platforms) та розширювальні (Extensible Platforms).

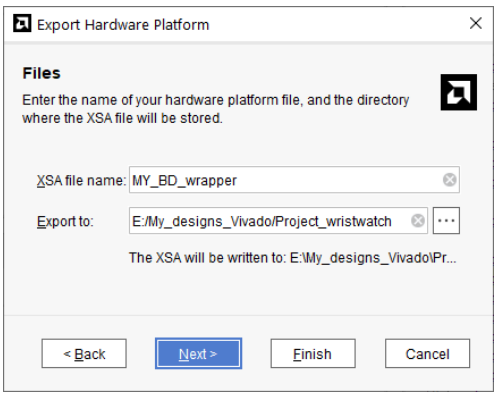
Розширювальні платформи в свою чергу дозволяють Vitis додавати нові апаратні компоненти або прискорювачі до існуючого дизайну. Для того, що обрати такий тип треба повернутися до налаштуваннь проєкту і на вкладці «General» поставити галочку напроти пункту «Project is an extensible Vitis platform». Фіксовані платформи містять повний набір апаратних файлів для певного пристрою, таких як FPGA або SoC. Вони визначають повністю завершену апаратну конфігурацію (наприклад, інтегрований процесор, периферія, пам'ять тощо). Коли ви експортуєте фіксовану платформу з Vivado, вона включає всі налаштування для конкретного пристрою, і ці файли потім імпортуються в Vitis, де ви можете додавати програмне забезпечення до цього готового апаратного дизайну. Важливо, що в такій платформі не можна додавати нові апаратні компоненти або прискорювачі, оскільки вона вже має завершену апаратну структуру. Ми будемо використовувати саме фіксовану платформу, тому нічого в налаштуваннях проєкту не міняємо. Просто слідуємо підказкам майстра екпорту платформи (рис. 2.24). Зверніть увагу, що на рисунку 2.24 (б) обрано пункт «including bitstream».

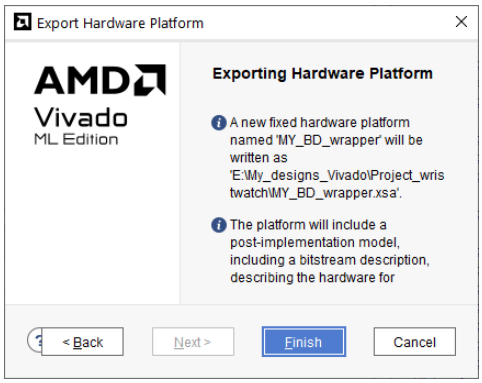


(а)



(б)





(в) (г)

Рисунок 2.24 – Процес експорту апаратних засобів

2.2.3 Інтерфейс між PL та PS частинами SoC

Процесорний модуль спілкується із зовнішнім світом та програмованою логікою за допомогою портів, об'єднаних у групи: MIO (multiplexed I/O); EMIO (extended multiplexed I/O); GP (General-Purpose Ports); HP (High-Performance Ports); ACP (Accelerator coherency port).

Розглянемо їх трохи детальніше.

1. Порти MIO (всього їх 54) підключені до виводів процесора. За допомогою цих портів можуть бути підключені такі периферійні пристрої процесорного модуля: USB-контролер – 2 шт.; Gigabit Ethernet контролер – 2 шт.; SD/SDIO контролер – 2 шт.; UART – 2 шт.; CAN – 2 шт; I2C – 2 шт; SPI – 2 шт.

2. EMIO є прокиданням периферійного пристрою з процесорного модуля в програмовану логіку і підключення периферійного пристрою до виводів програмованої логіки. Однак не всі периферійні пристрої можна перенести через EMIO, якісь не переносяться зовсім (USB), якісь із зміною функціональності (Ethernet через EMIO має інтерфейс GMII для підключення зовнішньої мікросхеми фізичного рівня). Між процесорним модулем та програмованою логікою по порту EMIO можна підключити 192 сигнали (64 входи та 128 виходів).

До з'єднань MIO та EMIO підключені мультиплексори, які вибирають режим їх роботи. Проаналізуємо реалізація GPIO через MIO and EMIO.

Частина процесорної системи (PS) Zynq 7000 має багато вбудованих контролерів IOP, причому кожен контролер забезпечує власний драйвер, доступний у формі коду C, що дозволяє користувачам інтегрувати зовнішні IOP з PS без додаткових витрат. Важливо зазначити, що ці контролери доступні лише в частині PS, тому IOP можуть спілкуватися лише з цією частиною Zynq 7000 SoC. PS Zynq може безпосередньо взаємодіяти із зовнішніми IOP через MIO. Однак кількість контактів MIO обмежена. Xilinx також пропонує своє рішення, представляючи контакти EMIO. Отже, EMIO можна використовувати для використання контролерів IOP, доступних у PS, для здійснення прямого зв'язку між PS і PL або для взаємодії із зовнішніми IOP через PL у випадку, якщо всі контакти MIO зайняті. З визначення GPIO зрозуміло, що їхні функції та напрямки вводу-виводу можна повністю конфігурувати. Їх застосування включає, але не обмежується моніторингом і керуванням іншими схемами. Периферійний пристрій GPIO забезпечує програмне забезпечення зі спостереженням і керуванням до 54 контактів пристрою через модуль MIO. Він також забезпечує доступ до 64 входів з програмованої логіки і 128 виходів до PL через інтерфейс EMIO.

GPIO організований у чотири банки регістрів, які групують пов'язані сигнали інтерфейсу: Bank0: 32-розрядний банк, що керує контактами MIO[31:0]; Bank1: 22-розрядний банк, що керує контактами MIO[53:32]; Bank2: 32-розрядний банк, що контролює сигнали EMIO[31:0]; Bank3: 32-розрядний банк, що контролює сигнали EMIO[63:32].

3. Порти GP використовують протокол AXI4. Протокол передбачає роботу з даними шириною до 32 біт. За одну транзакцію здійснюється передача/прийом даних не більше 8*32 біт (розмір boost транзакції дорівнює 8). Використовується, як правило, для управління блоками в програмованій логіці або низькошвидкісному обміні. Існує 2 порти Master і 2 порти Slave.

4. Порти HP використовують протокол AXI4. Протокол передбачає роботу з даними шириною 32 або 64 біти. Протокол підтримує boost транзакції. За одну транзакцію здійснюється передача/прийом довільного

розміру даних. Використовується, як правило, для організації високошвидкісного обміну даними між процесорним модулем та програмованою логікою. Існує 4 порти Slave.

5. Порт ACP використовує протокол AXI4. Протокол передбачає роботу з даними 64 біта. За одну транзакцію здійснюється передача/прийом довільного розміру даних. Існує один порт Slave.

Розглянемо більш детально інтерфейс AXI, який є основною формою з'єднання між елементами PS і PL Zynq. Він забезпечує з'єднання з високою пропускною здатністю та низькою затримкою між обома частинами пристрою. Існує 3 типи інтерфейсів AXI:

– AXI4 (для високопродуктивного відображення пам'яті);

– AXI4-Lite (зв'язок із відображенням пам'яті з низькою пропускною здатністю);

– AXI4 -Stream (для високошвидкісної потокової передачі даних).

AXI GPIO підтримує лише AXI4-Lite. AXI інтерфейс побудований зв'язку Master-Slave, між якими є п'ять незалежних каналів: канал зчитування адреси; канал зчитування даних; канал запису адреси; канал запису даних; канал запису відповіді. Адресні канали використовуються для надсилання адресної та контрольної інформації під час виконання основного «рукостискання» між Master і Slave. Канали даних – це місце, де розміщується інформація, якою потрібно обмінюватися. Master зчитує дані з Slave пристрою та записує дані в нього. Інформація про відповідь на зчитування розміщується в каналі даних на зчитування, тоді як інформація про відповідь на запис має спеціальний канал. Кожен обмін даними називається транзакцією. Таким чином Master може перевірити, чи транзакція запису завершена. Транзакція включає адресу та контрольну інформацію, надіслані дані, а також будь-яку інформацію відповіді. Фактичні дані надсилаються пакетами, які містять кілька передач.

2.3 Програмна реалізація на PS частині SoC у середовищі Vitis

2.3.1 Створення проєкту та налаштування системи у середовищі Vitis

Для створення проєкту в середовищі Vitis обираємо Open Workspace і папку, де будемо зберігати проєкт. Далі тиснемо Create Platform Component, вводимо будь-яке ім’я (My_platform) (рис. 2.25). Тиснемо Next.

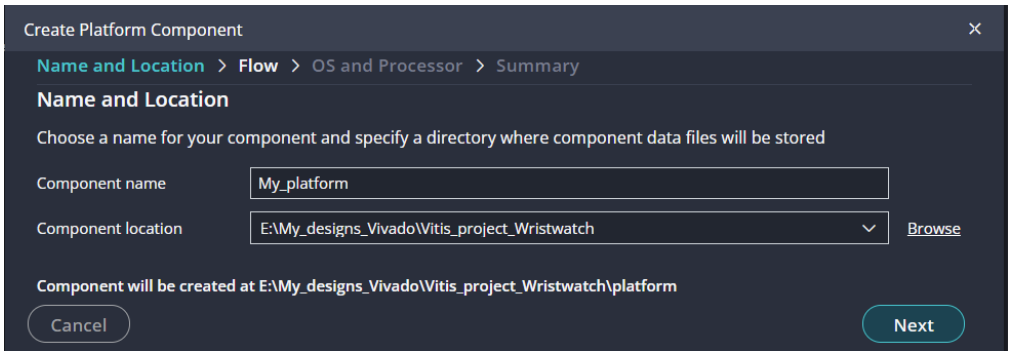


Рисунок 2.25 – Створення проєкту Vitis

У вікні, що відкриється, обираємо Hardware Design, тиснемо Browse і в папці Vivado проєкта шукаємо відповідний файл (MY_BD_wrapper.xsa) (рис. 2.26). Тиснемо Next.

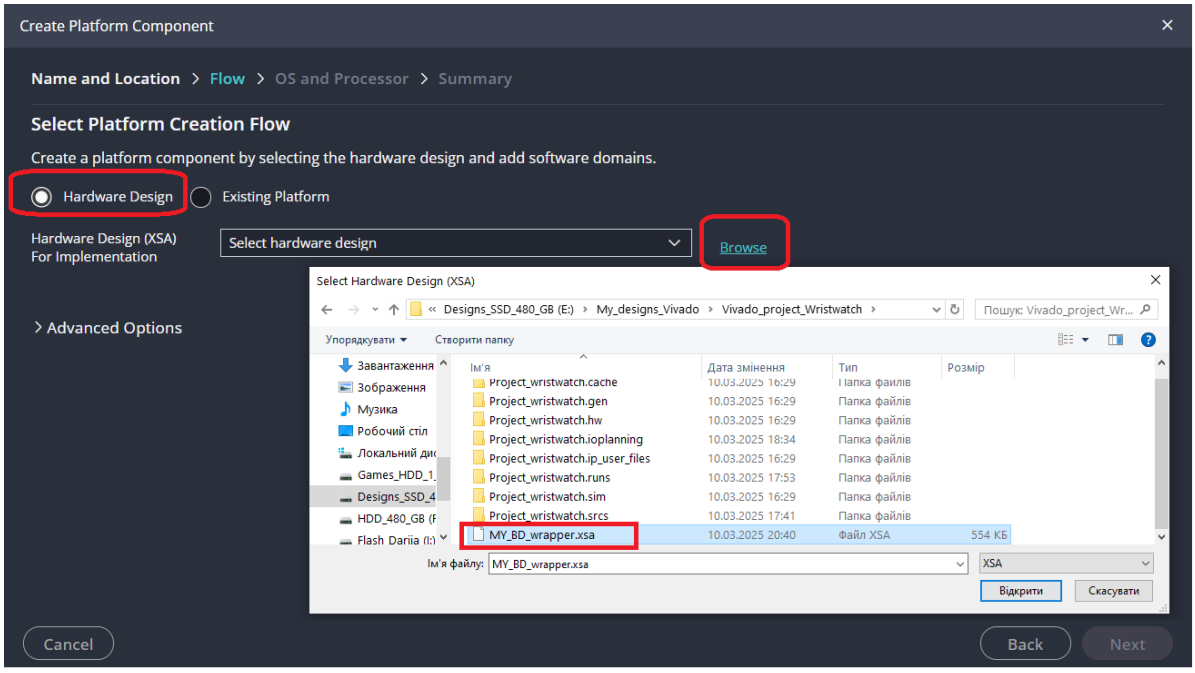


Рисунок 2.26 – Вибір апаратної платформи, створенної у Vivado

В наступному вікні обираємо в якості операційної системи Standalone, а в якості процесора – ps7_cortex9_0. Ця платформа (також відома як «Bare-metal») – це тип операційної системи, яка працює без додаткового операційного середовища, як у випадку з традиційними (Linux, Windows тощо). У контексті розробки на FPGA (Field Programmable Gate Array) або SoC (System on Chip), Standalone зазвичай означає платформу, де програмний код працює безпосередньо на процесорі або на мікроконтролері, який є частиною цієї платформи. Тиснемо Next. Наступне вікно показує підсумок. Натискаємо Finish .

В меню File обираємо New Component → From examples. У відповідному меню шукаємо приклад Hello World. На вкладці Hello World, що відкрилася, тиснемо кнопку Create Application Component from Template. Ця опція дозволяє швидко створити програмний застосунок (application) для обраної апаратної платформи на основі готового шаблону (template). Відкривається серія вікон для покрокового створення застосунку:

1. Спочатку обираємо будь-яке ім’я (PS_side) та папку з Vitis проєктом (рис. 2.27, а). Натискаємо Next.

2. Далі обираємо створену раніше платформу (My_platform) (рис. 2.27, б). Натискаємо Next.

3. Наступне вікно дозволяє обрати домен (standalone_ps7_cortexa9_0). Домен – це певний сегмент апаратних ресурсів, на якому буде виконуватимеся наше програмне забезпечення. У нашому випадку, standalone_ps7_cortexa9_0 вказує на процесор ARM Cortex-A9 на платформі Zynq-7000, який працює без операційної системи або в «bare-metal» режимі (рис. 2.27, в). Просто натискаємо Next.

4. Наступне вікно показує резюме. Натискаємо Finish.

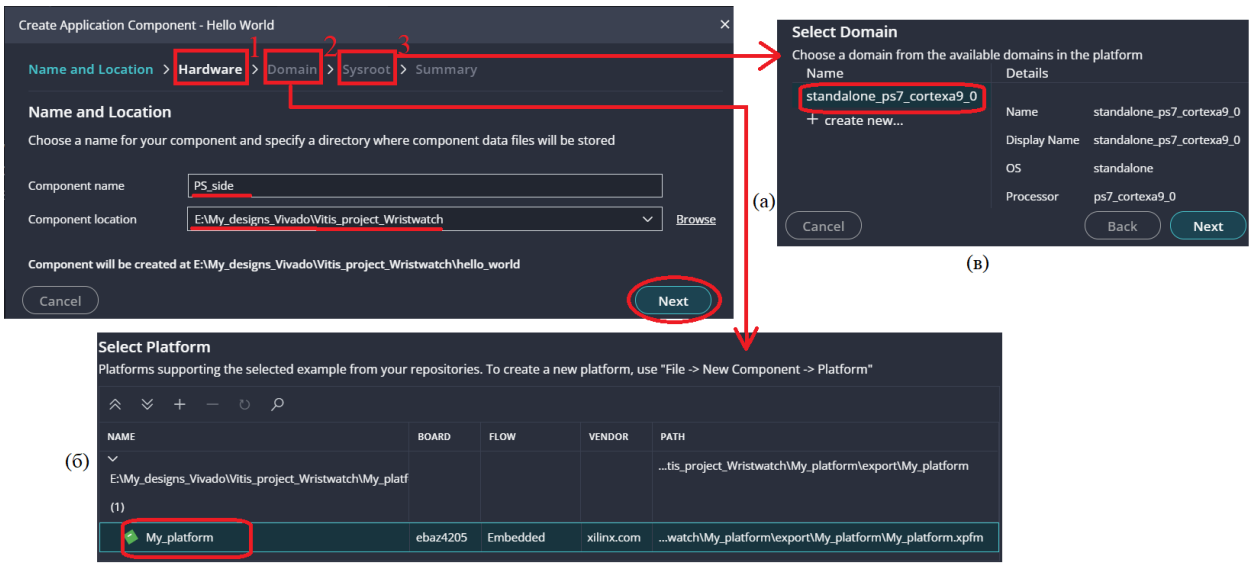
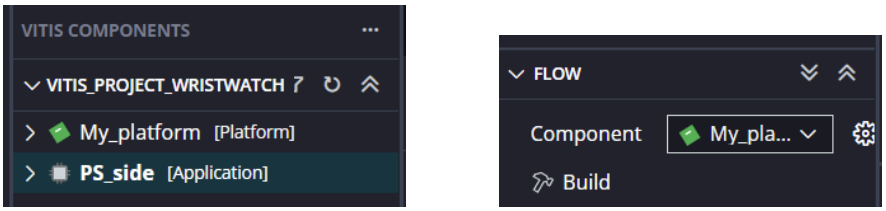


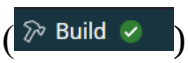
Рисунок 2.27 – Створення застосунка на основі готового шаблону

В вікні VITIS COMPONENTS з’явиться створена платформа та застосунок (рис. 2.28, а). Обираємо платформу (My_platform), а в вікні FLOW натискаємо на кнопку Build (рис. 2.28, б).



(а) (б)

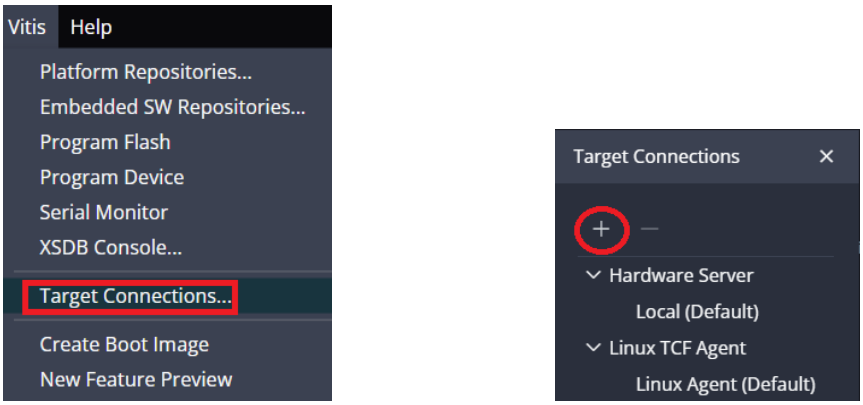
Рисунок 2.28 – Запуск побудови

Чекаємо на повідомлення про успішну генерацію платформи. Біля кнопки Build має з’явитися зелена відмітка (

). Те саме робимо для застосунку (PS_side) і запускаємо збирання (компіляції) програмного коду.

В вікні VITIS COMPONENTS обираємо наш застосунок (PS_side) → папка Sources → src. Подвійний клік по файлу helloworld.c відкриває файл для редагування. Тут ми пишемо код на С для керування дисплеєм, куди додаємо необхідні заголовочні файли, в тому числі той, що створимо самі для шрифтів (п. 2.3.2). Назву файлу helloworld.c перейменуємо на main.c. Як тільки код основного файлу створено та описаний заголовочний файл для

шрифтів, зберігаємо проєкт та заново збираємо програмний код (команда Build). Для перевірки створеного проєкту підключаємо апаратне забезпечення до комп'ютера (заздалегідь зібрана схема на базі SoC, див. розділ 3). Перевіряємо підключення у Vitis через меню Vitis → Target Connection (рис. 2.29, а).



(а) (б)

Рисунок 2.29 – Перевірка підключення плати

У вікні, що відкрилося, натискаємо на + (рис. 2.29, б). В поле Name вводимо будь-яке ім'я (наприклад назву SoC – EBAZ4205), в полі Type обираємо Hardware Server, в полі Host – localhost (тому що плата підключена безпосередньо до нашого комп'ютера), в полі Port залишаємо значення 3121 (default). Далі тиснемо кнопку Test Connection (рис. 2.30).

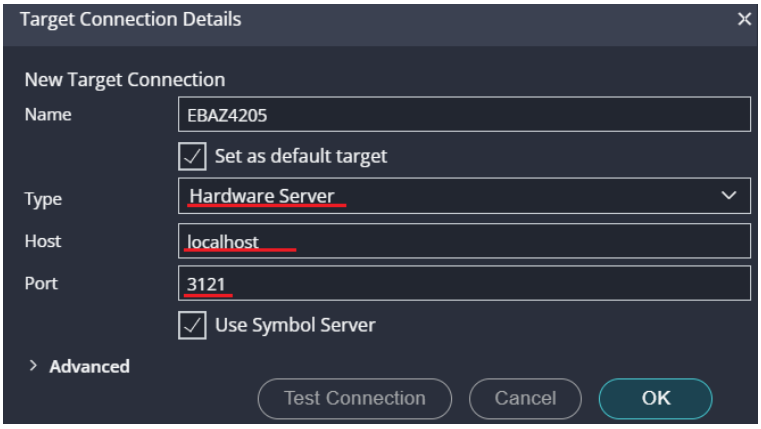
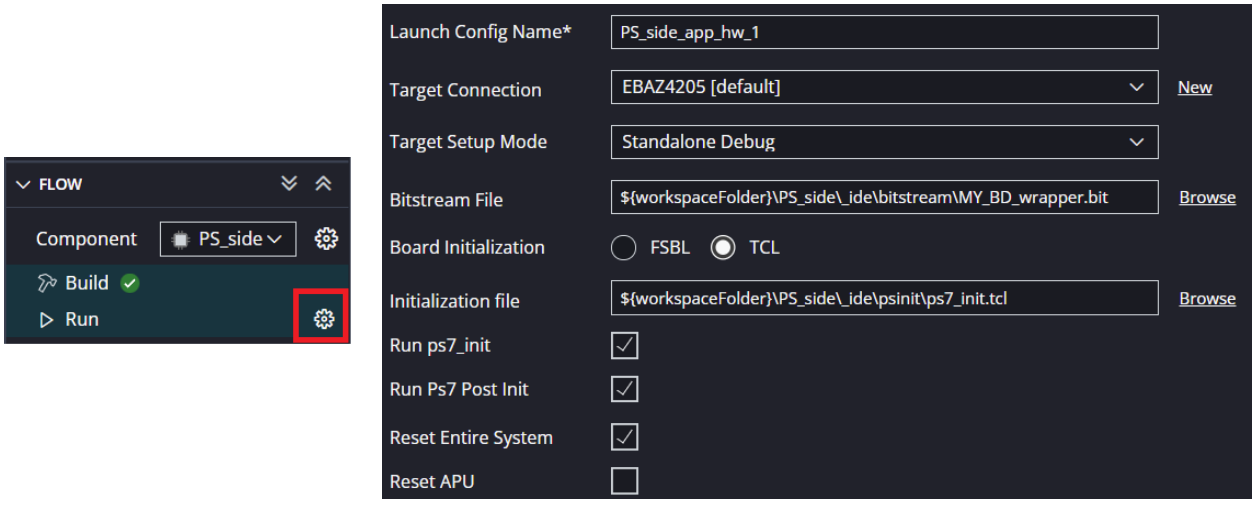


Рисунок 2.30 – Детальна інформація про підключену плату

Якщо з’єднання встановлено коректне, то отримаємо відповідне повідомлення. Тиснемо OK і закриваємо вікно. Далі програмуємо FPGA частину SoC: меню Vitis →Program Device.

Далі запускаємо сам застосунок (PS_side). Для цього в меню FLOW навпроти опції Run клікаємо на кнопку Settings ( ) (рис. 2.31, а). Відкриється файл налаштувань (рис. 2.32, б).



(а) (б)

Рисунок 2.31 – Налаштування програмного компонента

Можна ввести нове ім’я файлу, а можна залишити ім’я за замовчуванням (PS_side_app_hw_1). Обов’язково перевіряємо, що в полі Target Connection обрана наша апаратна платформа (плата EBAZ4205). Далі запускаємо Debug в меню FLOW (при необхідності цей процес можна зупинити командою Stop). Далі обираємо в меню FLOW команду Run.

Таким чином ми завантажуємо код як у PL, так і у PS частину SoC.

2.3.2 Реалізація драйвера керування IPS дисплеєм

IPS дисплей в данному проєкті використовується для виводу інформації для полегшення комунікації з користувачем багатофункціонального годинника. Керування здійснюється окремо через драйвер, який нам і треба написати.

IPS дисплей, на відміну від LED дисплея, є графічним, а не символьним. Тому виведення тексту на ньому потребує формування зображення символів у вигляді пікселів. Це означає, що кожна літера тексту малюється програмно за допомогою шрифтів і графічних функцій, а не виводиться як готовий символ із вбудованої пам’яті. Для виводу латинських літер, базових символів та цифр будемо використовувати ASCII формат, а для відображення кирилиці – Unicode. У відповідності з таблицею кодування ASCII для подання одного символу виділялося 7 бітів. Набір з 7 біт міг зберігати 2^7 = 128 різних значень. Трохи пізніше з’явилася розширена версія ASCII, що стала підмножиною системи кодування Windows-1251. Windows-1251 – це 8-бітне кодування, яке розширює стандарт ASCII, додавши підтримку додаткових символів, необхідних для кирилиці та інших мов (рис. 2.32). Перші 128 значень (від 0 до 127) постійні і формують так звану основну частину таблиці, куди входять десяткові цифри, букви латинського алфавіту (великі і малі), розділові знаки (крапка, кома, дужки та ін.), а також пропуск і різні службові символи (табуляція, перенесення рядка та ін.). Значення від 128 до 255 формують додаткову частину таблиці, де прийнято кодувати символи національних алфавітів.

Dec	Hex	Символ	Dec	Hex	Символ	Dec	Hex	Символ	Dec	Hex	Символ	Dec	Hex	Символ	Dec	Hex	Символ	Dec	Hex	Символ	Dec	Hex	Символ
0	0	спец. NOP	32	20	спец. SP (Пробел)	64	40	@	96	60	`	128	80	Ъ	160	A0		192	C0	А	224	E0	а
1	1	спец. SOH	33	21	!	65	41	A	97	61	a	129	81	Ѓ	161	A1	Ў	193	C1	Б	225	E1	б
2	2	спец. STX	34	22	"	66	42	B	98	62	b	130	82	‚	162	A2	ў	194	C2	В	226	E2	в
3	3	спец. ETX	35	23	#	67	43	C	99	63	c	131	83	ѓ	163	A3	J	195	C3	Г	227	E3	г
4	4	спец. EOT	36	24	$	68	44	D	100	64	d	132	84	„	164	A4	¤	196	C4	Д	228	E4	д
5	5	спец. ENQ	37	25	%	69	45	E	101	65	e	133	85	…	165	A5	Ґ	197	C5	Е	229	E5	е
6	6	спец. ACK	38	26	&	70	46	F	102	66	f	134	86	†	166	A6	¦	198	C6	Ж	230	E6	ж
7	7	спец. BEL	39	27	'	71	47	G	103	67	g	135	87	‡	167	A7	§	199	C7	З	231	E7	з
8	8	спец. BS	40	28	(	72	48	H	104	68	h	136	88	€	168	A8	Ё	200	C8	И	232	E8	и
9	9	спец. Tab	41	29	)	73	49	I	105	69	i	137	89	‰	169	A9	©	201	C9	Й	233	E9	й
10	0A	спец. LF	42	2A	*	74	4A	J	106	6A	j	138	8A	Љ	170	AA	Є	202	CA	К	234	EA	к
11	0B	спец. VT	43	2B	+	75	4B	K	107	6B	k	139	8B	‹	171	AB	«	203	CB	Л	235	EB	л
12	0C	спец. FF	44	2C	,	76	4C	L	108	6C	l	140	8C	Њ	172	AC	¬	204	CC	М	236	EC	м
13	0D	спец. CR	45	2D	-	77	4D	M	109	6D	m	141	8D	Ќ	173	AD		205	CD	Н	237	ED	н
14	0E	спец. SO	46	2E	.	78	4E	N	110	6E	n	142	8E	Ћ	174	AE	®	206	CE	О	238	EE	о
15	0F	спец. SI	47	2F	/	79	4F	O	111	6F	o	143	8F	Џ	175	AF	Ї	207	CF	П	239	EF	п
16	10	спец. DLE	48	30	0	80	50	P	112	70	p	144	90	ђ	176	B0	°	208	D0	Р	240	F0	р
17	11	спец. DC1	49	31	1	81	51	Q	113	71	q	145	91	‘	177	B1	±	209	D1	С	241	F1	с
18	12	спец. DC2	50	32	2	82	52	R	114	72	r	146	92	’	178	B2	І	210	D2	Т	242	F2	т
19	13	спец. DC3	51	33	3	83	53	S	115	73	s	147	93	“	179	B3	і	211	D3	У	243	F3	у
20	14	спец. DC4	52	34	4	84	54	T	116	74	t	148	94	”	180	B4	ґ	212	D4	Ф	244	F4	ф
21	15	спец. NAK	53	35	5	85	55	U	117	75	u	149	95	•	181	B5	µ	213	D5	Х	245	F5	х
22	16	спец. SYN	54	36	6	86	56	V	118	76	v	150	96	–	182	B6	¶	214	D6	Ц	246	F6	ц
23	17	спец. ETB	55	37	7	87	57	W	119	77	w	151	97	—	183	B7	·	215	D7	Ч	247	F7	ч
24	18	спец. CAN	56	38	8	88	58	X	120	78	x	152	98	[illegible]	184	B8	ё	216	D8	Ш	248	F8	ш
25	19	спец. EM	57	39	9	89	59	Y	121	79	y	153	99	™	185	B9	№	217	D9	Щ	249	F9	щ
26	1A	спец. SUB	58	3A	:	90	5A	Z	122	7A	z	154	9A	љ	186	BA	є	218	DA	Ъ	250	FA	ъ
27	1B	спец. ESC	59	3B	;	91	5B	[	123	7B	{	155	9B	›	187	BB	»	219	DB	Ы	251	FB	ы
28	1C	спец. FS	60	3C	<	92	5C	\	124	7C	|	156	9C	њ	188	BC	ј	220	DC	Ь	252	FC	ь
29	1D	спец. GS	61	3D	=	93	5D	]	125	7D	}	157	9D	ќ	189	BD	S	221	DD	Э	253	FD	э
30	1E	спец. RS	62	3E	>	94	5E	^	126	7E	~	158	9E	ћ	190	BE	s	222	DE	Ю	254	FE	ю
31	1F	спец. US	63	3F	?	95	5F	_	127	7F		159	9F	џ	191	BF	ї	223	DF	Я	255	FF	я

Рисунок 2.32 – Розширена версія стандарта ASCII

У 1991 році був створений стандарт кодування символів Unicode (уніфіковане кодування) – промисловий стандарт, розроблений для забезпечення цифрового представлення символів усіх писемностей світу та спеціальних символів. Таблиця символів Unicode знімає обмеження на кодування символів лише одним байтом. Натомість, використовується 17 просторів, кожен з яких визначає 65536 кодів і дає можливість описати максимум $17 \cdot 2^{16} = 1\,114\,112$ різних символів.

Unicode має декілька реалізацій, з яких найпоширенішими є UTF (Unicode Transformation Format) – формат перетворення Юнікоду; UCS (Universal Character Set) – універсальна таблиця символів.

Методи кодування знаків таблиці Unicode, тобто перетворення номерів осередків таблиці Юнікод у бінарні коди, складають кодовий простір, що складається з трьох кодів сімейства UTF: UTF-8, UTF-16 та UTF-32. Число після UTF визначає кількість бітів, що виділені під один символ, а число після UCS – кількість байтів. UTF-8 – стандарт, який став найпоширенішим для інтернаціональних кодувань, що реалізовує представлення Unicode, сумісне з 8-бітовим кодуванням тексту. UTF-8 є системою кодування зі змінною довжиною кодування символів. Це означає, що UTF-8 для кодування символів використовує від 1 до 4 байт на символ UTF-8. Саме сумісність UTF-8 з ASCII та використання змінного розміру байт, що веде до раціональне використання пам'яті, є головними перевагами алгоритму. Текст, що складається тільки з символів з номером менше 128 (символи латинського алфавіту, розділові знаки і керуючі символи), при записі в UTF-8 перетворюється на звичайний текст ASCII. Решта символів Unicode зображується послідовностями завдовжки від 2 до 4 байтів (табл. 2.1):

– 1 байт: традиційний ASCII;

– 2 байти: охоплює арабську, івритську та більшість європейських писемностей;

– 3 байти: відноситься до базової багатомовної площини (BMP);

– 4 байти: враховує всі символи Unicode.

Таблиця 2.1 – Утворення UTF кодів символів

Кількість байтів	Діапазон Unicode		UTF			
			Байт 1	Байт 2	Байт 3	Байт 4
1	U+000	U+007F	0xxxxxxx			
2	U+0080	U+07FF	110xxxxx	10xxxxxx		
3	U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
8	U+10000	U+1FFFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

Слід зазначити, що кодпоінти Unicode позначаються як U+xxxx, де xxxx – шістнадцятковий код символу. За префіксом U+ ми можемо визначити, що мають на увазі саме кодпоінт Unicode.

Для кодування кириличних символів достатньо двох байтів. Наприклад, буква Б має код 193 (Win-1251), що відповідає 0x0411 в Unicode. Процес кодування в UTF-8 показано на рисунку 2.33.

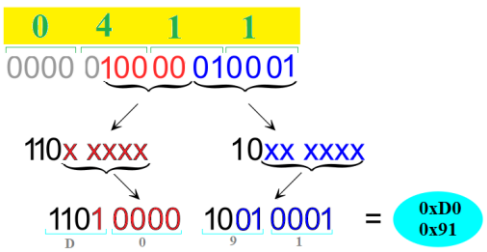


Рисунок 2.33 – Принцип кодування UTF-8, що використовує 2 байти на представлення символа

На рисунку 2.34 наведено відповідність кодування UTF-8 та розширеного ASCII (Win-1251).

	UTF-8	Win-1251
А	0xD0, 0x90	192 0xC0
Б	0xD0, 0x91	193 0xC1
В	0xD0, 0x92	194 0xC2
Г	0xD0, 0x93	195 0xC3
Д	0xD0, 0x94	196 0xC4
Е	0xD0, 0x95	197 0xC5
Ж	0xD0, 0x96	198 0xC6
З	0xD0, 0x97	199 0xC7

	UTF-8	Win-1251
И	0xD0, 0x98	200 0xC8
Й	0xD0, 0x99	201 0xC9
К	0xD0, 0x9A	202 0xCA
Л	0xD0, 0x9B	203 0xCB
М	0xD0, 0x9C	204 0xCC
Н	0xD0, 0x9D	205 0xCD
О	0xD0, 0x9E	206 0xCE
П	0xD0, 0x9F	207 0xCF

	UTF-8	Win-1251
Р	0xD0, 0xA0	208 0xD0
С	0xD0, 0xA1	209 0xD1
Т	0xD0, 0xA2	210 0xD2
У	0xD0, 0xA3	211 0xD3
Ф	0xD0, 0xA4	212 0xD4
Х	0xD0, 0xA5	213 0xD5
Ц	0xD0, 0xA6	214 0xD6
Ч	0xD0, 0xA7	215 0xD7

	UTF-8	Win-1251
Ш	0xD0, 0xA8	216 0xD8
Щ	0xD0, 0xA9	217 0xD9
Ъ	0xD0, 0xAA	218 0xDA
Ы	0xD0, 0xAB	219 0xDB
Ь	0xD0, 0xAC	220 0xDC
Э	0xD0, 0xAD	221 0xDD
Ю	0xD0, 0xAE	222 0xDE
Я	0xD0, 0xAF	223 0xDF

Рисунок 2.34 – UTF-8 кодування кирилиці

На цьому рисунку зеленим кольором показані коди в 16-річній системі числення і додатково для кодування Win-1251 жовтим кольором вказані десяткові еквіваленти для простоти орієнтації по таблиці.

Різниця в кодах складає 0x30. Перевірите це можна, взявши за приклад будь-яку літеру і порівнявши молодший байт коду UTF-8 та код Win-1251.

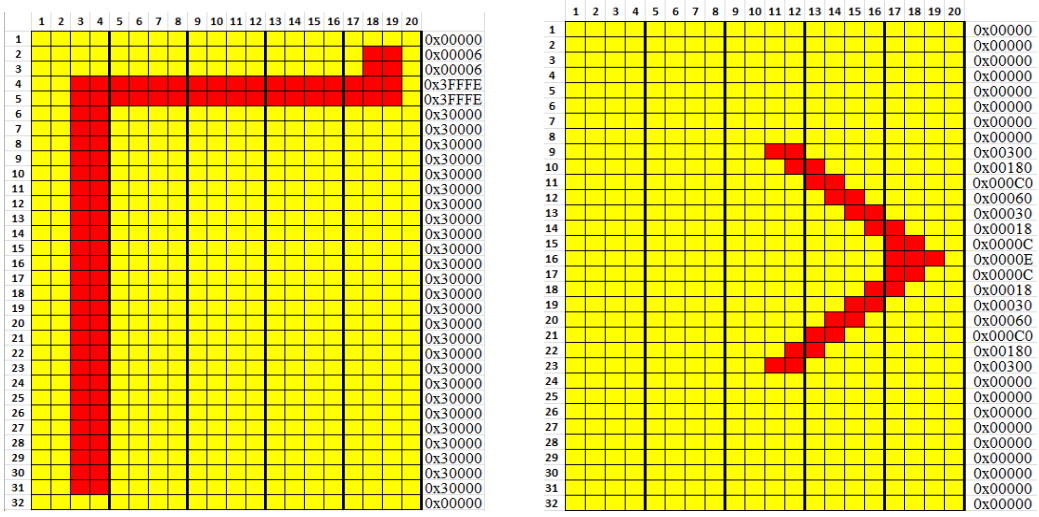
Наприклад, код літери А у кодуванні Win-1251 дорівнює 0xC0, а код у кодуванні UTF-8 дорівнює у молодшому байті 0x90. Тоді 0xC0 – 0x90 = 0x30.

Такі букви, яких не має в кирилиці (І, Ї, Є, Ґ) також мають свої коди (таблиця 2.2).

Таблиця 2.2 – UTF-8 кодування символів українського алфавіту І, Ї, Є, Ґ

	UTF-8	Win-1251
Є	0xD0, 0x84	170 0xAA
І	0xD0, 0x86	178 0xB2
Ї	0xD0, 0x87	175 0xAF
Ґ	0xD2, 0x90	165 0xA5

Ця інформація і лягла в основу створеної бібліотеки шрифтів, де кожна буква була описана намальована графічно у Exel. Розмір кожної літери мав вписуватися у прямокутних 32*20 пікселей. Наприклад, на рисунку 2.35 (а) показана літера Ґ, а на рисунку 2.35 (б) – символ >.



(а) (б)

Рисунок 2.35 – Графічне представлення літери Ґ та символа >

В середовищі Vitis створимо заголовочний файл, де ми будемо описувати шрифти (font_32_20.h). Назва файлу вказує на розмір букв 32*20 (рис. 2.36). На основі графічних преставлень, був створений масив з типом даних *const uint32_t font[][32]*, де 32 – це висота сітки.

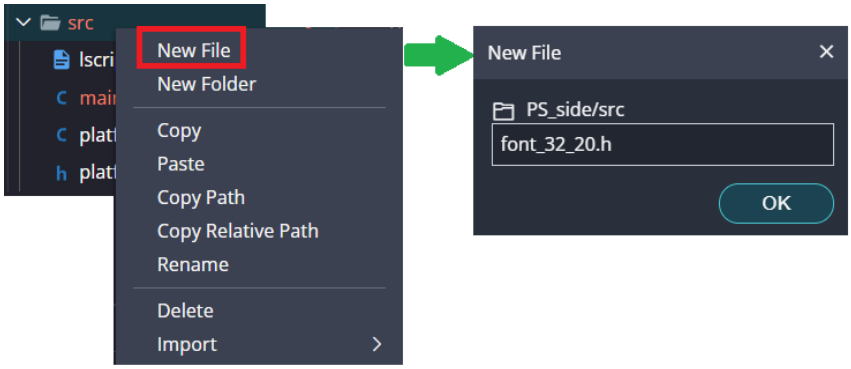


Рисунок 2.36 – Створення заголовчного файлу

На лістингу 2.1 показан фрагмент коду для латинської літери A, української літери Ґ та символа >.

Лістинг 2.1 – Фрагмент бібліотеки шрифтів

```
#include "xplatform_info.h"
#include <stdlib.h>
const uint32_t font[][32] = {
//Латинські букви
//0 символ A
 {0x00000, 0x00F00, 0x01F80, 0x06060, 0x06060, 0x0C030, 0x0C030, 0x0C030,
  0x18018, 0x18018, 0x3000C, 0x3000C, 0x3000C, 0x60006, 0x60006, 0x7FFFE,
  0x7FFFE, 0x60006, 0x60006, 0x60006, 0x60006, 0x60006, 0x60006, 0x60006,
  0x60006, 0x60006, 0x60006, 0x60006, 0x60006, 0x60006, 0x60006, 0x00000},
.....................................................................
//40 символ Ґ
 {0x00000, 0x00006, 0x00006, 0x3FFFE, 0x3FFFE, 0x30000, 0x30000, 0x30000,
  0x30000, 0x30000, 0x30000, 0x30000, 0x30000, 0x30000, 0x30000, 0x30000,
  0x30000, 0x30000, 0x30000, 0x30000, 0x30000, 0x30000, 0x30000, 0x30000,
  0x30000, 0x30000, 0x30000, 0x30000, 0x30000, 0x30000, 0x30000, 0x00000},
.....................................................................
//78 символ >
  {0x00000, 0x00000, 0x00000, 0x00000, 0x00000, 0x00000, 0x00000, 0x00000,
  0x00300, 0x00180, 0x000C0, 0x00060, 0x00030, 0x00018, 0x0000C, 0x0000E,
  0x0000C, 0x00018, 0x00030, 0x00060, 0x000C0, 0x00180, 0x00300, 0x00000,
  0x00000, 0x00000, 0x00000, 0x00000, 0x00000, 0x00000, 0x00000, 0x00000}
};
```

Повний код бібліотеки шрифтів представлено в додатку А.

Сам драйвер керування дисплеєм був написан на основі відомого драйвера ST7789. Цей драйвер забезпечує підтримку SPI-інтерфейсу, керування кольоровою палітрою у форматі RGB565, а також реалізує функції ініціалізації дисплея, виводу пікселів, прямокутників, ліній та тексту в режимі реального часу [9]. Але шрифт базового драйвера не підходив під наш розмір екрана. Крім того він не підримував український алфавіт. Тому були внесені відповідні правки в код та підключено створену бібліотеку шрифтів `#include "font_32_20.h"`. Повний код драйвера керуванням дисплеєм представлено в додатку А. В цілому, було створено багато функцій, які для зручності опису розіб'ємо на декілька категорій (табл. 2.3)

Таблиця 2.3 – Функції кервування дисплеєм

Системні функції ініціалізації	
init_alarm_gpio()	ініціалізує GPIO інтерфейс для сигналу будильника як вхід
init_gpio_fsm()	ініціалізує GPIO для зчитування стану автомата (FSM) як вхід
Lcd_Gpio_Init()	налаштовує EMIO виводи для керування LCD-дисплеєм (як GPIO виходи)
init_timer()	ініціалізує системний таймер (рекомендовано Vitis)
get_ms()	повертає поточний час у мілісекундах із глобального таймера (XTime)
Затримки	
delay_spi_nop()	дуже коротка затримка (1 ітерація циклу), для SPI-сигналів
delay(unsigned volatile int i)	затримка у програмі (примітивна, на циклі).
Керування дисплеєм	
spi_send(unsigned char dat)	виводить один байт даних у SPI-протоколі через бітову емуляцію
LCD_WR_DATA8(u8 dat)	виводить 8-бітне значення як дані у LCD
LCD_WR_REG(u8 dat)	виводить 8-бітне значення як команду у LCD
LCD_WR_DATA(u16 dat)	виводить 16-бітне значення як дані у LCD
Address_set(unsigned int x1, y1, x2, y2)	встановлює область екрану для виведення пікселів
LCD_background(u16 color)	заливає весь екран заданим кольором
Lcd_Init()	ініціалізує LCD-дисплей, надсилаючи всі потрібні команди
Робота з шрифтами	
get_char_bitmap(unsigned char c)	повертає бітову мапу символу з таблиці шрифтів font, підтримує латиницю, кирилицю та деякі знаки
draw_char(int x, y, unsigned char c, uint32_t color)	малює окремий символ на дисплеї в заданих координатах
draw_text(int x, y, char *text, uint16_t color)	малює рядок тексту, підтримує ASCII та українську кирилицю (перекодування з UTF-8)
Повідомлення (MSG1 – MSG15)	

Загальна граф-схема головної функції представлена на рисунку 2.37.

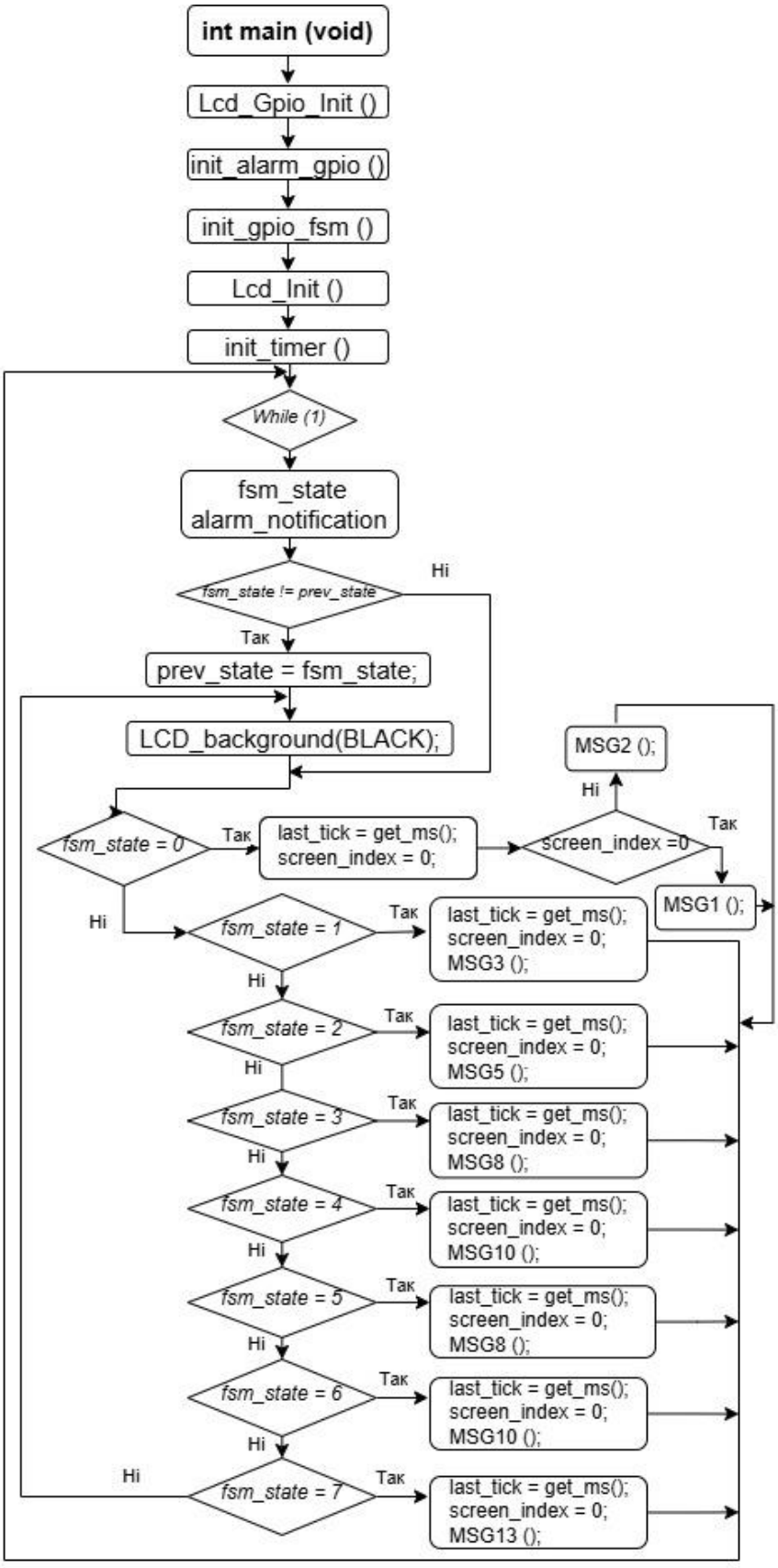


Рисунок 2.37 – Загальна граф-схема функції main ()

3 ПРОТОТИП МОДЕЛІ БАГАТОФУНКЦІОНАЛЬНОГО ГОДИННИКА

Для візуалізації майбутнього прототипу створимо спочатку схему (рис. 3.1). Цю схему було побудовано на платформі EasyEDA – це онлайн-сервіс для проєктування електронних схем та друкованих плат. Хоча вибір систем на кристалі (SoC) тут обмежений, доступні користувацькі бібліотеки з нестандартними компонентами. Крім того, можна легко додавати периферійні елементи, такі як дисплеї, світлодіоди, кнопки, резистори, п’єзодинаміки тощо, з доступної бібліотеки.

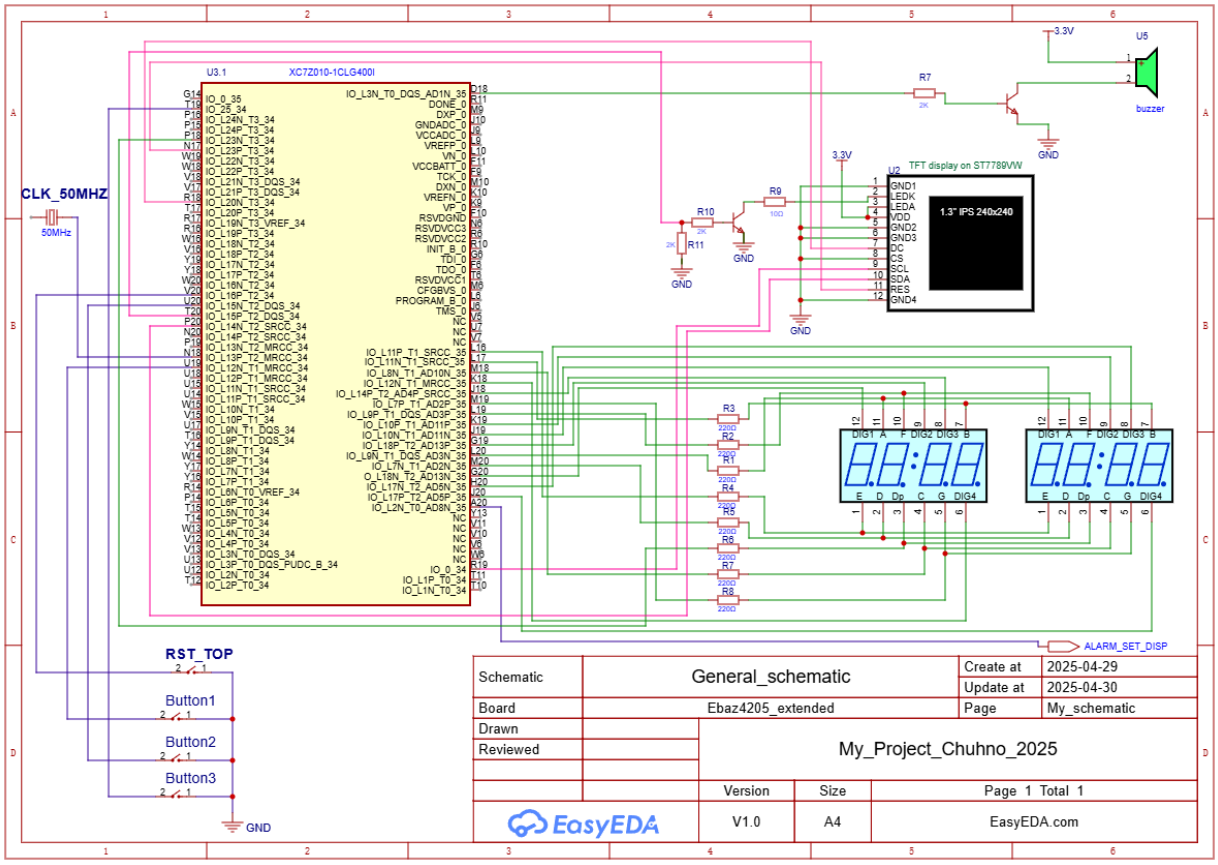


Рисунок 3.1 – Схема для збору прототипа

В якості аппаратної бази оберемо наступні компоненти:

- плата Ebaz4205 на базі SoC Zynq-7000;
- плата розширення з екраном, п’єзодинаміком та кнопками;

- 7-сегментні дисплеї;
- резистори;
- JTAG програматор та кабелі «папа–мама», «папа–папа».

Розглянемо основні компоненти детальніше.

3.1 Вибір компонентів та збор прототипа

3.1.1 Плата Ebaz4205 та плата розширення

Плата EBAZ4205 – це бюджетна плата на базі Xilinx Zynq-7000, яка спочатку використовувалась як контролер у майнерах Ebit E9+ BTC (рис. 3.2). Зараз вона популярна серед ентузіастів FPGA та розробників вбудованих систем завдяки своїй низькій вартості та широким можливостям.

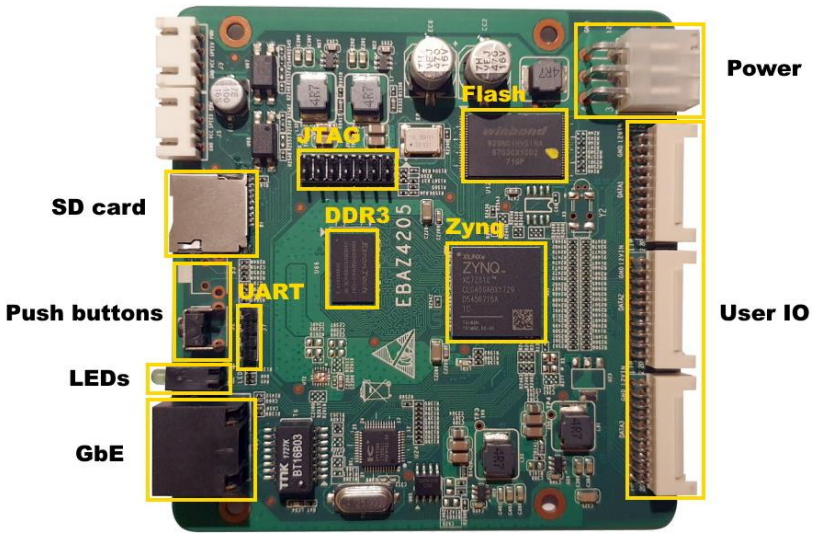


Рисунок 3.2 – Плата EBAZ4205

Основні характеристики: SoC ZYNQ XC7Z010-1CLG400I (двоядерний ARM Cortex-A9 @ 666 МГц + FPGA Artix-7 з 28 тис. логічних елементів); оперативна пам'ять: 256 МБ DDR3 (модель MT41K128M16); постійна пам'ять: 128 МБ NAND Flash; інтерфейси: 10/100 Мбіт Ethernet (PHY IP101GA), UART, JTAG, слот microSD; три 20-контактних роз’єма, які підключаються до PL частині; живлення: 5В через роз'єм або USB.

У якості програматора використаємо китайський Platform Cable USB II, модель DLC10 та відповідний USB кабель тип A-тип B. Основним недоліком

плати Ebaz4205 є відсутність перифірії для контролю. Використаємо китайську плату, на борту якої є кнопки, п’єзодинаміки, діоди. Вона підключається через три 20-контактних роз’єма до плати Ebaz4205, а також до комп’ютера за допомогою USB кабеля тип А-тип С (рис. 3.3).

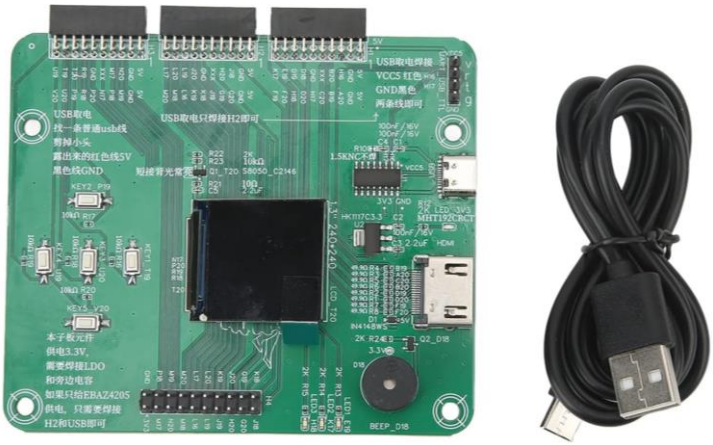


Рисунок 3.3 – Плата розширення

Підключення плати Ebaz4205 до плати розширення показано на рисунку 3.4.

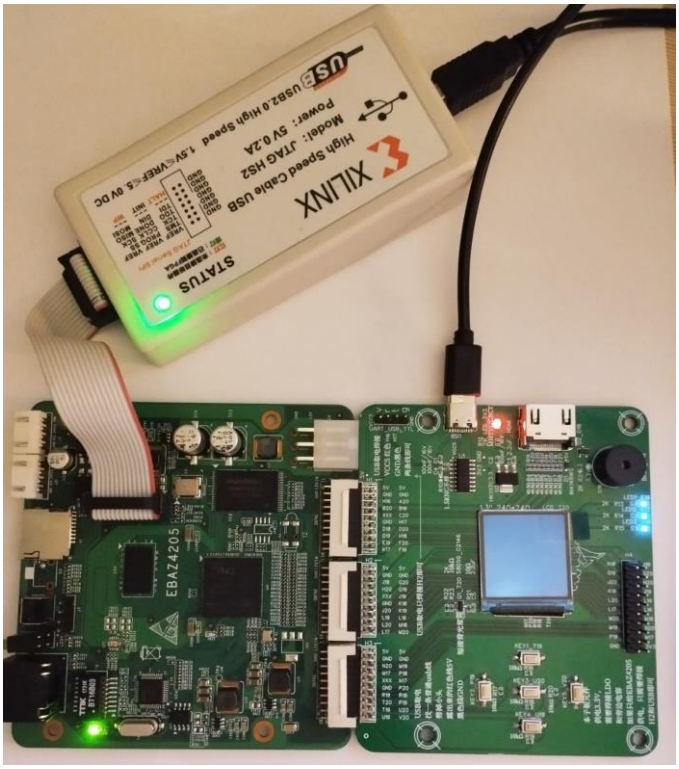


Рисунок 3.4 – Підключення плати Ebaz4205 до плати розширення

3.1.2 IPS дисплей

На базі цієї плати є IPS дисплей. Це недорогий, але потужний дисплей з роздільною здатністю 240 x 240 пікселів і розміром лише 1,3 дюйма. Кольори дисплея RGB (рис. 3.5). Матриця IPS – різновид матриць TFT. IPS технологія виконання матриці рідкокристалічного (на тонкоплівкових транзисторах) екрана, коли кристали розташовані паралельно один одному уздовж єдиній площині екрана, а не спірально. За відсутності напруги молекули рідких кристалів, не повертаються. На практиці найважливіша відмінність IPS-матриці від TN-TFT-матриці полягає в підвищеному рівні контрастності за рахунок практично ідеального відображення чорного кольору. Картинка виходить більш чіткою.

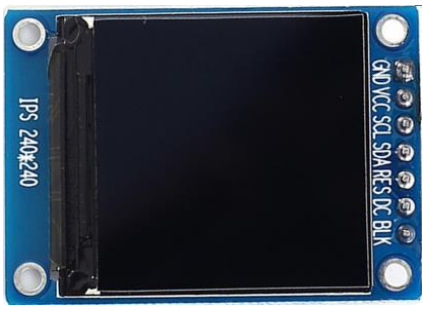


Рисунок 3.5 – IPS дисплей на базі драйвера ST7789

Цей дисплей працює на базі драйвера ST7789 та має послідовний периферійний інтерфейс 4-бітовий SPI. 12-пінова версія – це повноцінний модуль (рис. 3.6).

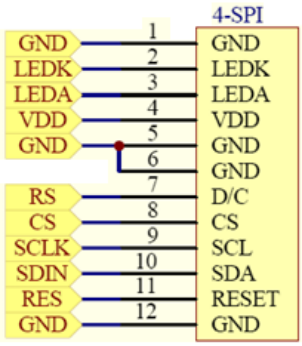


Рисунок 3.6 – 12-піновий SPI інтерфейс

Але цей дисплей має спрощену 7-пінову версію, опис кожного виводу і відповідний інтерфейс наведено в таблиці 3.1.

Таблиця 3.1 – Опис пінів IPS екрана

№	12-піновий SPI	7-піновий SPI
1	GND (земля)	GND (земля)
2	LEDK (LED катод)	BL (backlight) підсвічювання
3	LEDA (LED анод)	
4	VDD (живлення 5V)	VCC (живлення 5V або 3.3V)
5	GND (земля)	–
6		
7	D/C (керуючий сигнал для перемикання введення команд чи даних)	DC (керуючий сигнал для перемикання введення команд чи даних)
8	CS (пін вибору чіпа)	CS (постійно в 0)
9	SCL (послідовний CLK)	CLK (послідовний CLK)
10	SDA (вхід даних SPI інтерфейсу)	SDA (вхід даних SPI інтерфейсу)
11	Reset (скидання дисплея, активний по 0)	RES (скидання, активний по 0)
12	GND (земля)	–

У цьому проєкті SPI-інтерфейс реалізовано програмно – без апаратного SPI-контролера, шляхом керування GPIO-виводами PS через EMIO:

- пін даних (SDA) – 58 пін;
- тактовий імпульс (SCL) – 57 пін;
- дані або команда (CD / DC) – 55 пін;
- скидання дисплея (RES) – 56 пін;
- підсвічування (BLK) – 54 пін.

3.1.3 Електронний п’єзодинамік

Електронні п’єзодинаміки (базери) створюють звук і поділяються на два типи: активні та пасивні. Активний базер має вбудований генератор, завдяки чому може видавати звук, підключаючись лише до постійного струму (DC). Пасивний базер, навпаки, не має вбудованого генератора, тому для створення звуку потребує подачі змінного струму (AC) або аудіосигналу

певної частоти. На платі розширення встановлено пасивний базер (рис. 3.7). Перевага пасивних базерів полягає в тому, що вони дозволяють змінювати висоту або тон звуку, тобто їх можна запрограмувати на відтворення широкого діапазону частот або музичних нот.

Рисунок 3.7 – Пасивний п’єзодинамік

Розглянемо частоти музичних нот у 4-й октаві (табл. 3.2).

Таблиця 3.2 – Музичні ноти на їх частоти

Назва ноти		Частота (Hz)
Do	C	261.626
Do sharp	C#	277.183
Re	D	293.665
Re sharp	D#	311.127
Mi	E	329.628
Fa	F	349.228
Fa sharp	F#	369.994
Sol	G	391.995
Sol sharp	G#	415.305
La	A	440
La sharp	A#	466.164
Si	B	493.883

Висоту тону пасивного базера можна керувати, змінюючи частоту сигналу, який подається на нього. FPGA здатна генерувати сигнал широтно-імпульсної модуляції (PWM) у вигляді прямокутної хвилі через свій цифровий вихід.

PWM-сигнал має два основні параметри, які визначають його характеристики:

- частота – визначає, як часто повторюється імпульс за секунду (і, відповідно, частоту звуку),
- скважність (duty cycle) – визначає співвідношення часу, коли сигнал перебуває в активному (високому) стані, до загального періоду.

Щоб розрахувати ширину імпульсу для кожної ноти 4-ї октави, враховуючи частоту годинника 5 МГц, скористаємося такими розрахунками:

– частота ноти Do 261.626Hz => 5MHz/261.626Hz = 19111;

– частота ноти Re 293.665Hz => 5MHz/293.665Hz = 17026;

– частота ноти Mi 329.628Hz => 5MHz/329.628Hz = 15169;

– частота ноти Fa 349.228Hz => 5MHz/349.228Hz = 14317;

– частота ноти Sol 391.995Hz => 5MHz/391.995Hz = 12755;

– частота ноти note La 440Hz => 5MHz/440Hz = 11364;

– частота ноти note La sharp 466.164Hz => 5MHz/466.164Hz = 10726;

– частота ноти Si 493.883Hz => 5MHz/493.883Hz = 10124.

Таким чином затримку і термінах тактового сигналу можна порахувати настйпним чином (табл. 3.3).

Таблиця 3.3 – Затримки в термінах CLK (5MHz)

Затримки в термінах CLK (5MHz)	
0.5 s	$0.5 \cdot (5 \cdot 10^6) \simeq 2 \cdot 10^6$
1 s	$1 \cdot (5 \cdot 10^6) = 5 \cdot 10^6$
2 s	$2 \cdot (5 \cdot 10^6) = 10 \cdot 10^6$
3 s	$3 \cdot (5 \cdot 10^6) = 15 \cdot 10^6$
4 s	$4 \cdot (5 \cdot 10^6) = 20 \cdot 10^6$
5 s	$5 \cdot (5 \cdot 10^6) = 25 \cdot 10^6$

3.1.4 Резистор

Обов'язково підключати резистор до світлодіодів (LED), коли ви їх використовуєте в електричних схемах. Резистори потрібні для наступного.

1. Обмежити струм: світлодіоди працюють при певному струмі, зазвичай від 10 до 30 мА, залежно від типу діода. Без резистора струм може бути занадто великим, що призведе до перегріву і пошкодження діода.

2. Захистити світлодіод: якщо ви підключаєте світлодіод без резистора, він може отримати занадто великий струм, і це може викликати його пошкодження або навіть згоряння.

3. Забезпечити стабільну роботу: Резистор допомагає забезпечити стабільну роботу діода, особливо в умовах зміни напруги на живленні, що може вплинути на струм.

В проєкті для обмеження струму були використані резистори 200 Ом, потужність 0,25Вт (рис. 3.8).

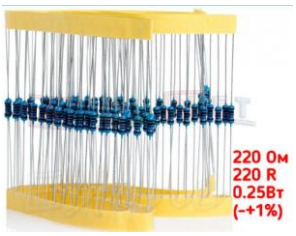


Рисунок 3.8 – Резистори 200 Ом

3.1.5 Семисегментний дисплей

7-сегментний дисплей чи індикатор – це цифровий дисплей, призначений для відображення числової інформації. Світлодіоди, розташовані у формі цифр, формують добре видимий дисплей. 7-сегментні дисплеї доступні в різних кольорах (червоний, синій і зелений) і розмірах (від 0,56 до 6,5 дюймів). Іноді від двох до чотирьох 7-сегментних дисплеїв упаковують разом, щоб утворити великий дисплей.

Кожен із семи світлодіодів називається сегментом, тому що під час освітлення сегмент утворює частину цифрової цифри (як десяткової, так і шістнадцяткової), яка має відображатися. Додатковий 8-й світлодіод іноді використовується в тому самому корпусі і дозволяє вказувати десяткову крапку (DP), коли два або більше 7-сегментних дисплеїв з’єднані разом для відображення чисел більше десяти (рис. 3.9).

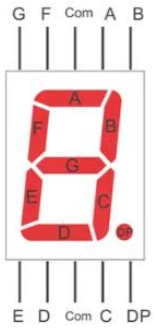


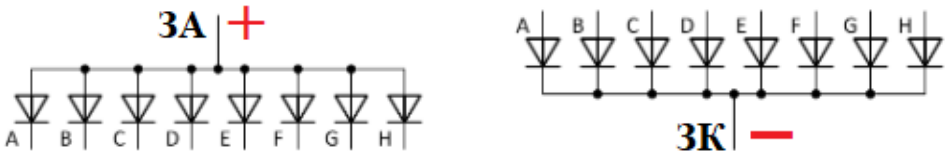
Рисунок 3.9 – Виводи 7-сегментного дисплея

Як видно з рисунку 3.9, дисплей має 10 виводів: два загальних (Common, Com) та 8 – для керування (A, B, C, D, E, F, G, DP).

Розглянемо, як формуються цифри на 7-сегментному індикаторі. Візуальне зображення цифри створюється завдяки комбінації увімкнених сегментів. Кожній цифрі від 0 до 9 відповідає певна послідовність логічних 0 і 1. Залежно від підключення анода і катода світлодіодів 7-сегментні дисплеї діляться на два типи.

1. Дисплей із загальним анодом (ЗА) (рис. 3.10, а) – усі анодні висновки з'єднані, а катодні – залишаються відкритими. Щоб використовувати цей тип дисплея, необхідно підключити анод до джерела живлення 5V, а катод – до землі (GND)

2. Дисплей із загальним катодом (ЗК) (рис. 3.10, б) – всі катодні виводи з'єднані між собою, а анодні – залишаються відкритими. Щоб використовувати цей тип дисплея, потрібно під'єднати катод до землі (GND), а анод – до джерела живлення 5V.

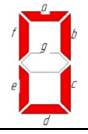
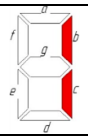
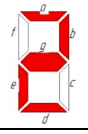
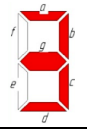
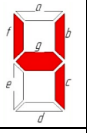
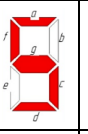
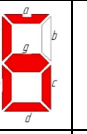
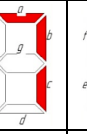
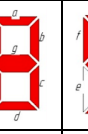
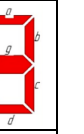


(а) (б)

Рисунок 3.10 – Дисплей із загальним анодом (а) та із загальним катодом (б)

Розглянемо 7-сегментний дисплей із загальним катодом, де сигнал керування кожного сегмента є високим. Кодування показано в таблиці 3.4.

Таблиця 3.4 – Кодування цифр для 7-сегментного індикатора із ЗК

Цифра	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Зображення										
A	1	0	1	1	0	1	1	1	1	1
B	1	1	1	1	1	0	0	1	1	1
C	1	1	0	1	1	1	1	1	1	1
D	1	0	1	1	0	1	1	0	1	1
E	1	0	1	0	0	0	1	0	1	0
F	1	0	0	0	1	1	1	0	1	1
G	0	0	1	1	1	1	1	0	1	1

В даному проєкті буде використовуватися два чотирирозрядних 7-сегментних індикатора з додатковою двокрапкою між 2 розрядами із загальним катодом фірми KLM модель KL4041-BB (рис. 3.11).

Рисунок 3.11 – Чотирирозрядний 7-сегментний індикатор

Цей індикатор має наступні характеристики: кількість виводів – 12; кількість сегментів – 28+6; розмір індикатора – 40,5x16x7 мм; тип підключення сегментів – загальний катод; фон: чорний; колір свічення: синій; тип індикації: динамічна.

Схема підключення чотирирозрядного 7-сегментного індикатора з загальним катодом представлена на рисунку 3.12. На рисунку 3.12 видно, що усі однойменні аноди розрядів (A/B/C/D/E/F/G/DP) з’єднані і підключені до

одного з портів (11/7/4/2/1/10/5/3), а катоди кожного з розрядів (DIG.1, DIG.2, DIG.3, DIG.4) – на один з інших портів відповідно (12/9/8/6).

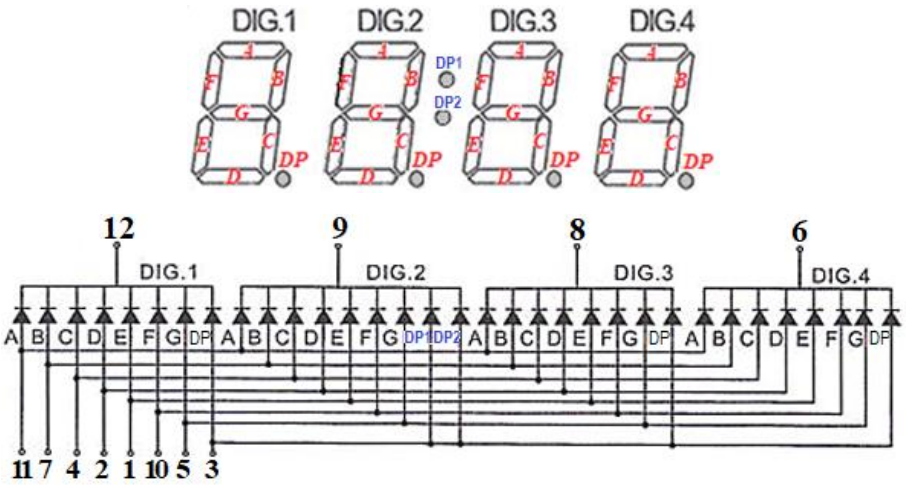


Рисунок 3.12 – Схема підключення чотирирозрядного 7-сегментного індикатора з загальним катодом

Це і є динамічна індикація. Таким чином, нам необхідно всього 8+4=12 виводів портів. Індикація здійснюється шляхом швидкого циклічного відображення кожного розряду числа з відповідним активним катодом. Хоча індикатори по черзі відображають цифри, завдяки тому, що час перемикання менший за час реакції людського ока (приблизно 0,04 с), ми сприймаємо, що всі індикатори світяться одночасно. Розпіновка чотирирозрядного 7-сегментного дисплея представлена на рисунку 3.13. Виводи D1 – D4 використовуються для підключення розрядів (катодів) DIG.1, …, DIG.4 відповідно; A,B, C, …G – для підключення сегментів (анодів) кожного дисплею; DP – для підключення точок (анод).

Рисунок 3.13 – Піни 7-сегментного індикатора із загальним катодом

3.1.6 Технічна реалізація прототипу

Повністю зібраний прототип представлено на рисунку 3.14. Для підключення 7-сегментних дисплеїв знадобилася монтажна плата без пайки (Breadboard), резистори та відповідні кабелі. Вона дозволила легко вставляти й з'єднувати електронні компоненти за допомогою спеціальних отворів і внутрішніх провідників.

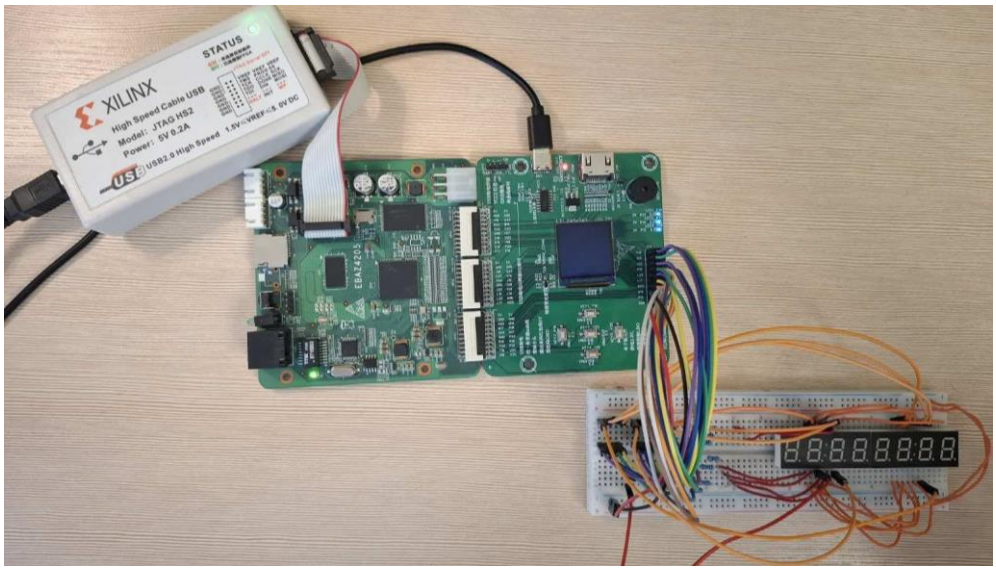


Рисунок 3.14 – Прототип багатофункціонального годинника

Підключення двох чотирирозрядних 7-сегментних індикаторів з загальним катодом до плати розширення показано в таблиці 3.5.

Таблиця 3.5 – Підключення дисплеїв до плати

Виводи дисплея / Виводи плати	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
	E	*D*	*DP*	*C*	*G*	*D4*	*B*	*D3*	*D2*	*F*	*A*	*D1*
до дисплея № 1	L16	M20	P18	M18	M19	K18	L17	J18	G19	L19	L20	G20
до дисплея № 2						J20		H20	K19			J19

3.2 Функціональна верифікація RTL-модуля

Для перевірки коректності розробленої VHDL-моделі був створений Testbench-код якого наведено у додатку А (лістинг А. 11). Треба взяти до

уваги деякі особливості моделювання синхронних схем. На вхід TOP моделі поступає сигнал CLK безпосередньо з виходу IP блока, що зменшує частоту синхросигналу на платі з 50 MHz до 5 MHz. Основні блоки запропонованої моделі працюють по сигналу CLK, частота якого має бути 100 Hz. Це означає, що до моделі необхідно було додати і ще один подільник частоти (*DIV_5HZ_to_100HZ*) з коефіцієнтом ділення $5*10^6$. Але такий коефіцієнт довго перевіряти на часовій діаграмі. Тому для зручності аналізу діаграм цей коефіцієнтом зменшено до 5 (рис. 3.15).

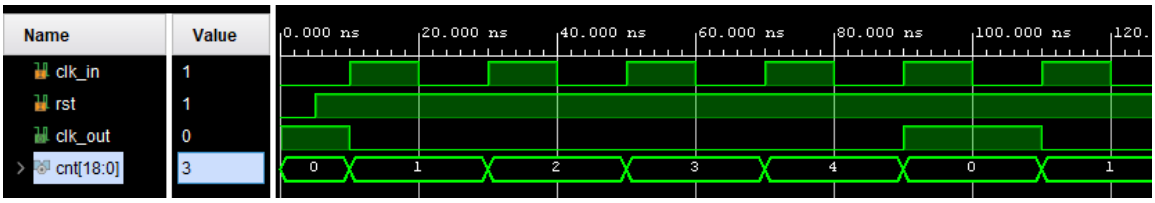


Рисунок 3.15 – Часова діаграма роботи подільника частоти *U0: DIV_5HZ_to_100HZ* (з коєфіцієнтом ділення 5)

Далі наведені часові діаграми основних блоків запропонованої моделі.

На рисунку 3.16 та 3.17 показано часову діаграму роботи лічильника, що рахує до 60 (*U1_1:CTR_60*).

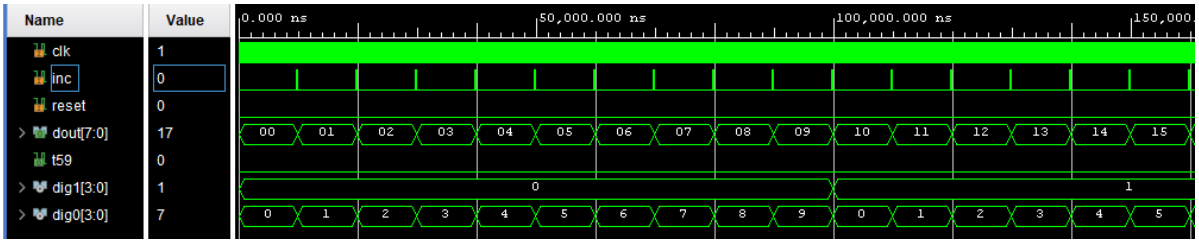


Рисунок 3.16 – Часова діаграма роботи лічильника секунд *U1_1: CTR_60* (з 00)

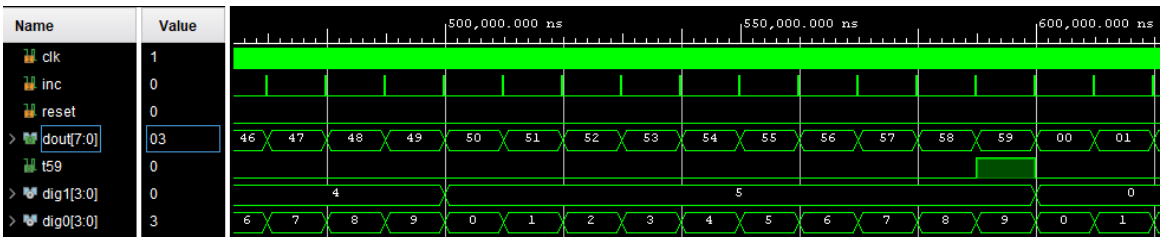


Рисунок 3.17 – Часова діаграма роботи лічильника секунд *U1_1: CTR_60* (до 59)

На рисунку 3.18 та 3.19 показано часову діаграму роботи лічильника, що рахує до 60 (*U1_2:CTR_60*).

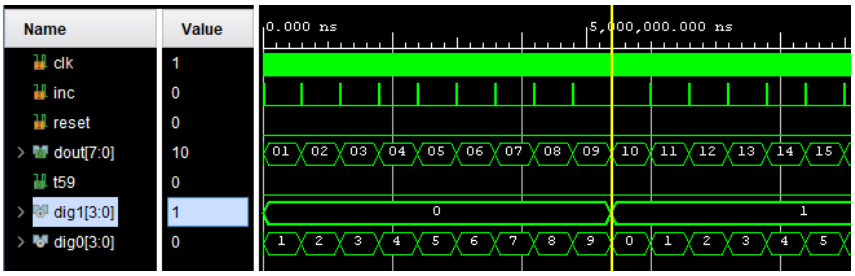


Рисунок 3.18 – Часова діаграма роботи лічильника хвилин *U1_2: CTR_60* (з 00)

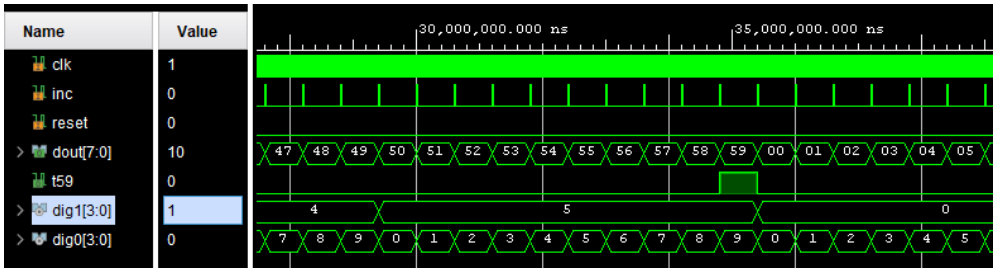


Рисунок 3.19 – Часова діаграма роботи лічильника хвилин *U1_2: CTR_60* (до 59)

Обидва блоки працюють коректно, але з різними часовими характеристика. Це пояснюється тим, що блок U1_1 є лічильником секунд, а U1_2 – хвилин. Об'єднавши дві часові діаграми можна побачити цю залежність: як тільки один лічильник дорахує до 59 секунд, в наступному циклі інший лічильник збільшить кількість хвилин з 01 до 02 (рис. 3.20).

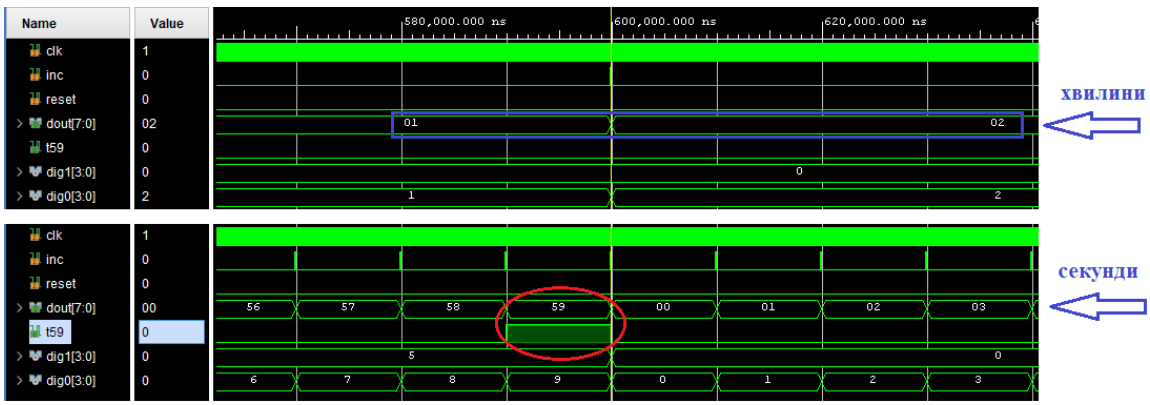


Рисунок 3.20 – Часова діаграма *U1_1* та *U1_2*

Проаналізуємо часові діаграми лічильника хвилин (*U1_2*) та лічильника хвилин (*U1_3:CTR_12*). Ситуація аналогічно попередній: як тільки один лічильник дорахує до 59 хвилин, в наступному циклі інший лічильник збільшить кількість годин з 01 до 02 (рис. 3.21).

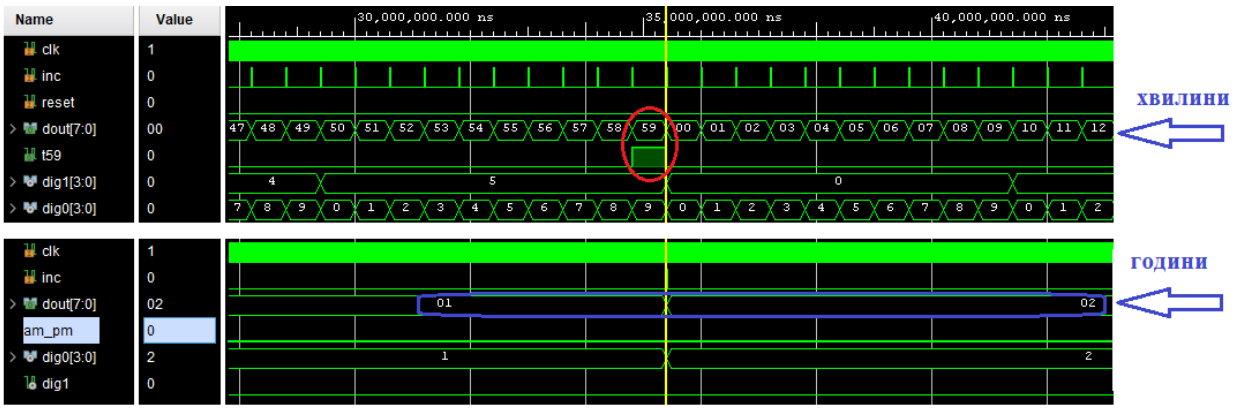


Рисунок 3.21 – Часова діаграма *U1_2* та *U1_2*

Крім того, треба перевірити формування сигналу *am_pm* (*CTR_12*). На рисунку 3.22 видно, що кожні 12 годин (дня чи ночі), сигнал *am_pm* переключається на протилежний.

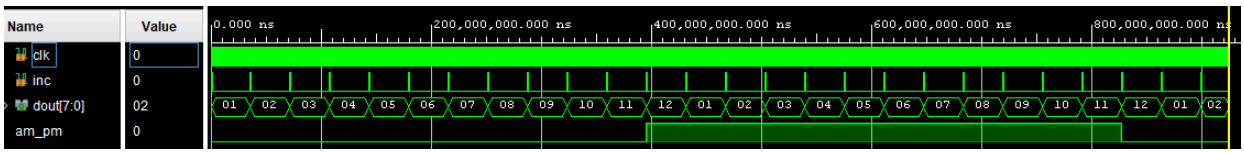
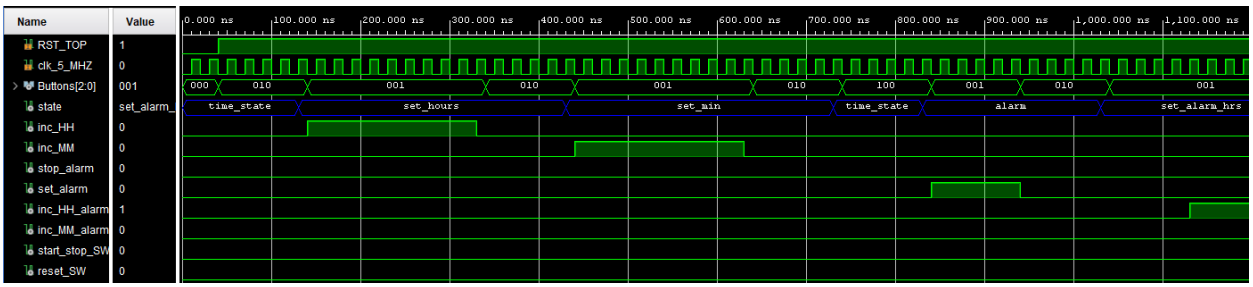
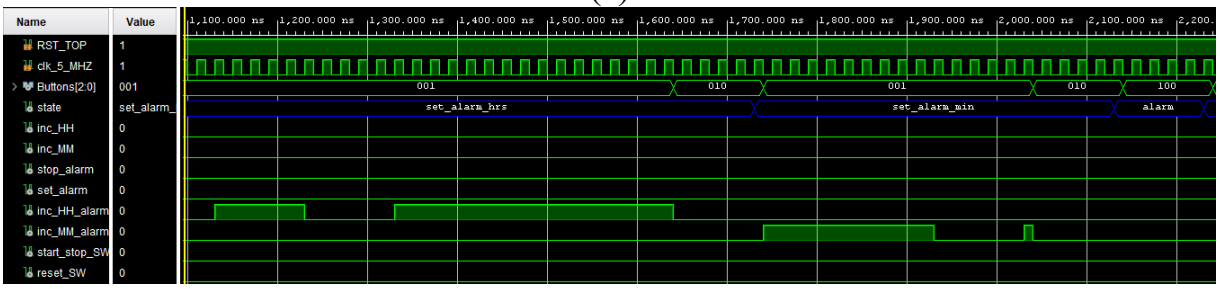


Рисунок 3.22 – Часова діаграма роботи лічильника годин *U1_3:CTR_12*

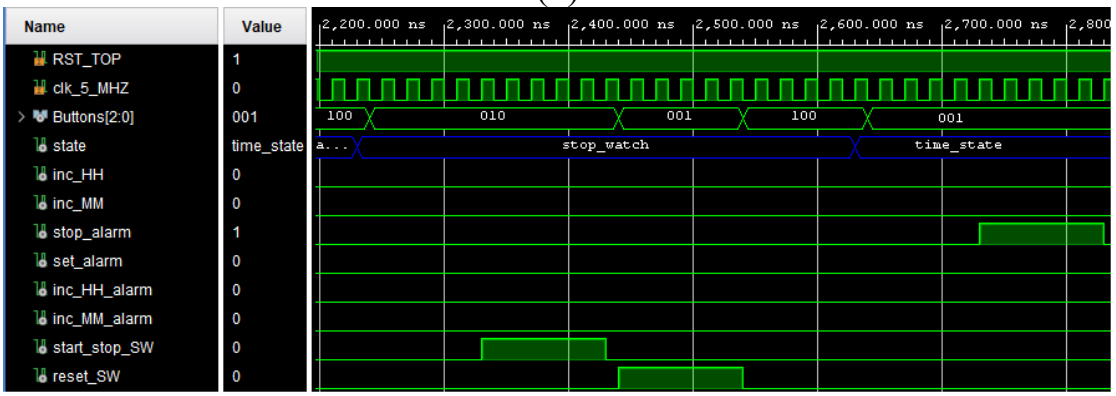
На рисунку 3.23 показані часові діаграми роботи автомата Мілі, що відповідала за контроль роботи запропонованої моделі годинника (*Control_FSM*). Для перевірки всіх переходів (*time_state, set_min, set_hours, alarm, set_alarm_hrs, set_alarm_min, stop_watch*) та виходів (*inc_HH, inc_MM, stop_alarm, set_alarm, inc_HH_alarm, inc_MM_alarm, start_stop_SW, reset_SW*) ми включали кнопки (шина *Buttons[2:0]*) згідно з графом переходів (рис. 2.4).



(а)



(б)



(в)

Рисунок 3.23 – Часова діаграма роботи автомата Мілі: з 0 до 1100 ns (а), з 1100 ns до 2200 ns (б) та з 2200 ns до 2800 ns

Проаналізуємо один з переходів більш детально, наприклад *time_set →set_hours→set_min* (рис. 3.24).

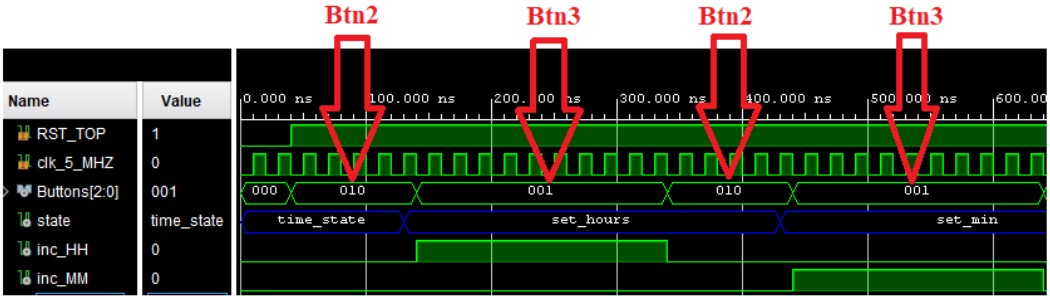


Рисунок 3.24 – Аналіз переходу *time_set →set_hours→set_min*

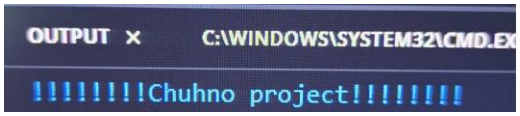
Знаходячись у початковому стані *time_set* і змоделювавши натискання на кнопку *Btn2* (*Buttons=“010”*) автомат переходить до стану *set_hours*. Знаходячись у цьому стані і змоделювавши натискання на кнопку *Btn3* (*Buttons=“001”*) автомат залишається в стані *set_hours*, при цьому генеруючи вихідний сигнал *inc_HH*. Для переходу до стану *set_min* змоделюємо натискання на кнопку *Btn2*. Знаходячись у цьому стані і змоделювавши натискання на кнопку *Btn3* (*Buttons=“001”*) автомат залишається в стані *set_min*, при цьому генеруючи вихідний сигнал *inc_MM*.

Перевіримо блоку усунення брязкоту сигналу (*ANTI_BOUNCED*) при натисканні кнопок (Btn1, Btn2, Btn3). На рисунку 3.25 видно, що сигнал S зсувається. Стабільним вважається сигнал на виході O.

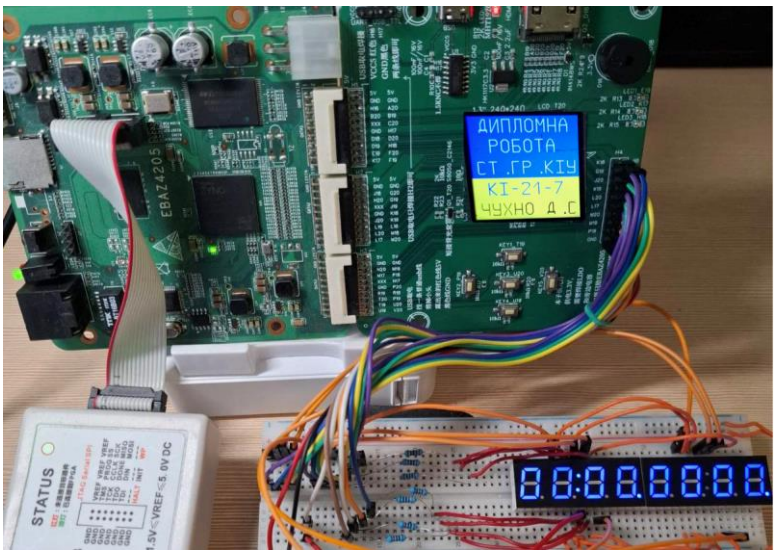
Рисунок 3.25 – Часова діаграма роботи блока усунення брязкоту сигналу *U3*, *U4* та *U5*

4.3 Тестування прототипу

Після програмування пристрою в консолі з’являється відповідних надпис (рис. 3.26, а), загораються 7-сегментні диплеї і показують 0. Також на екрані плати розширення ми бачимо екран привітання «Дипломна робота ст.гр. КІУКІ-21-7 Чухно Д.С.» (рис. 3.26, б).



(а)



(б)

Рисунок 3.26 – Екран привітання

Через 5 секунд надпис на екрані змінюється на інший «Годинник на Zynq7000. Старт → кнопка 1». Це означає, що треба натиснути на кнопку 1, щоб перейти до налаштувань годинника (рис. 3.27).

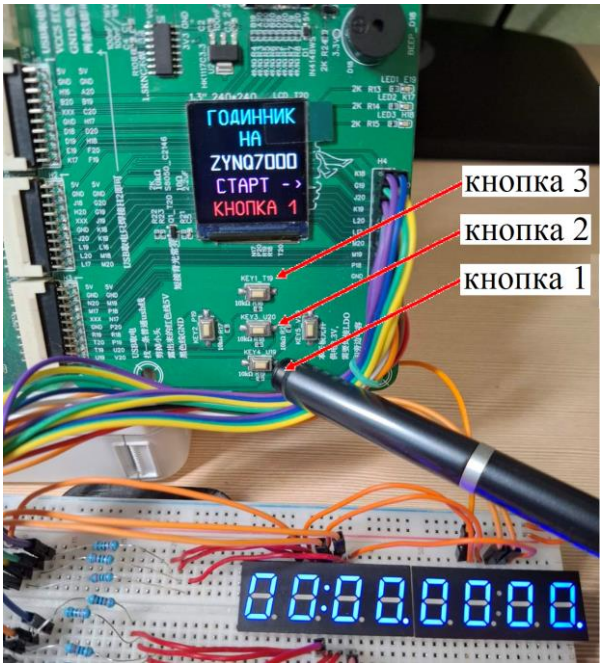


Рисунок 3.27 – Початок роботи

Далі ми попадаємо в режим налаштування годинника. На екрані ми бачимо повідомлення з підказками, яку кнопку треба натиснути для

налаштування годин (рис. 3.28, а) та хвилин (рис. 3.28, б), а саме кнопка 2 для входу в режим налаштування, а кнопка 3 збільшує години/хвилини на 1 при кожному натисканні. Для того, щоб перейти до PM, просто клацаємо на кнопку 3 до 12. Тобто режими змінюються кожні 12 годин.

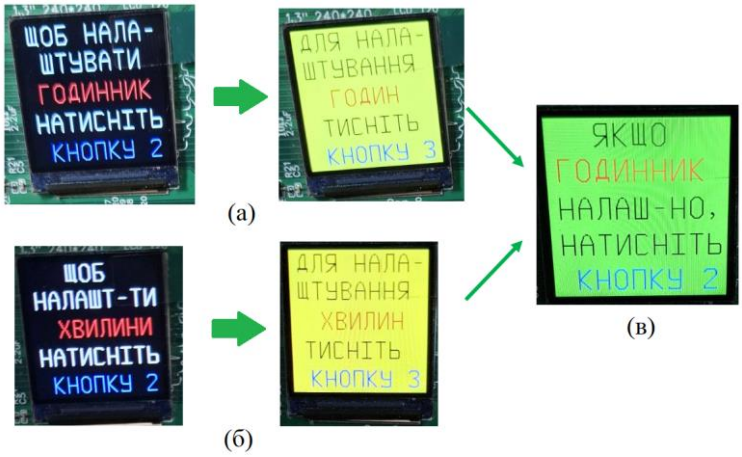


Рисунок 3.28 – Налаштування годин та хвилин годинника

Коли годинник налаштовано ми отримаємо відповідне повідомлення і натиснувши знову на кнопу 2 виходимо з режиму налаштування (рис. 3.29, в). Годинник налаштовано в 12-годинному форматі на 2:37, символ А означає АМ. Перевіряємо його роботу, порівнюючи з часом на телефоні.

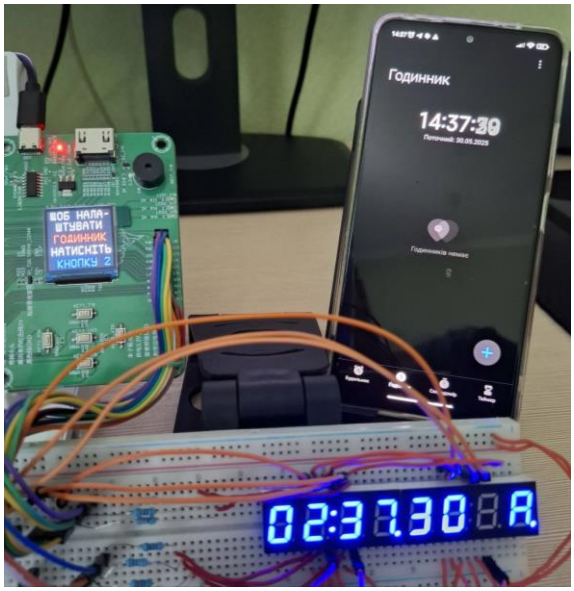


Рисунок 3.29 – Повністю налаштований годинник

Розроблена модель дозволяє додати будильник. Щоб перейти до налаштувань натискаємо на кнопку 1, про що періодично сигналізує повідомлення на екрані (рис. 3.30, б). Далі ми попадаємо в режим налаштування будильника і на 7-сегментному індикаторі бачимо всі 0 (рис. 3.30, а). На екрані ми бачимо повідомлення з підказками, яку кнопку треба натиснути для налаштування годин (рис. 3.30, в) та хвилин (рис. 3.30, г), а саме кнопка 2 для входу в режим налаштування, а кнопка 3 збільшує години/хвилини на 1 при кожному натисканні. Коли будильник налаштовано ми отримаємо відповідне повідомлення і натиснувши знову на кнопу 2 виходимо з режиму налаштування (рис. 3.30, д).

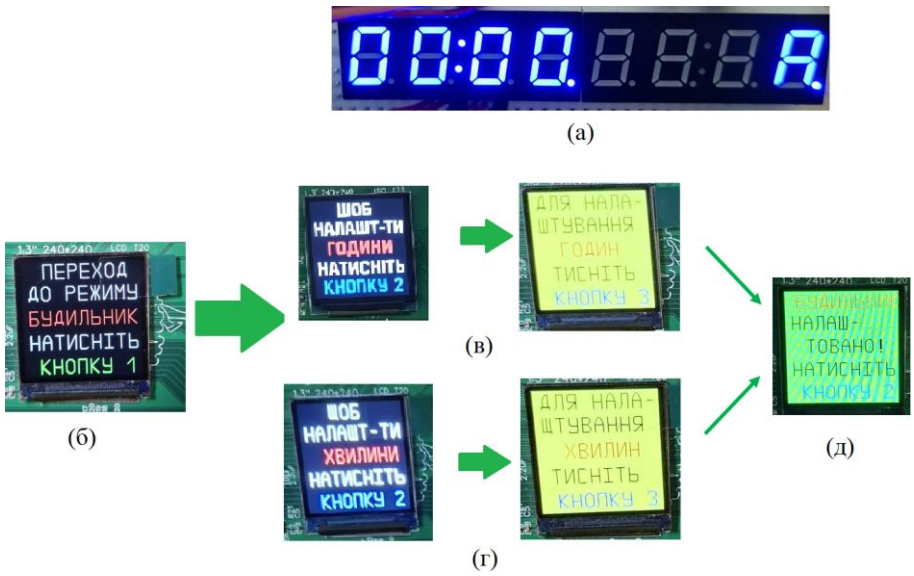
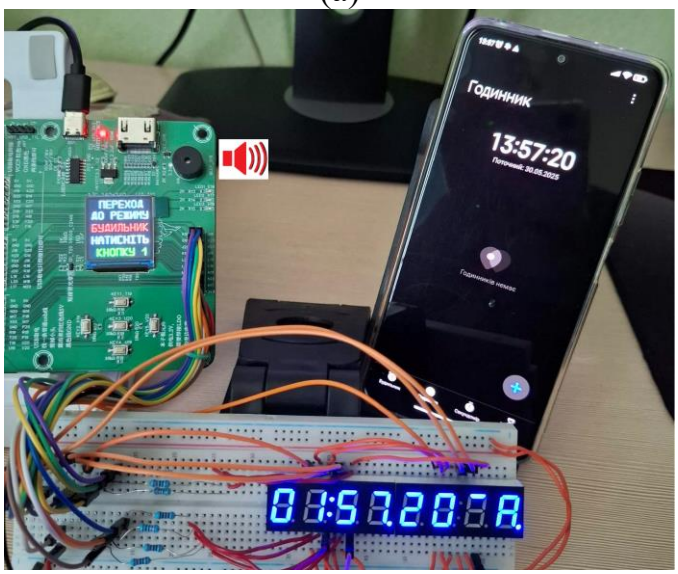


Рисунок 3.30 – Налаштування годин та хвилин будильника

Для перевірки роботи будильник налаштували на 1:58 (рис. 3.31, а). На екрані з’являється відмітка на передостанньому індикаторі  (рис. 3.31, б). У зазначений час будильник спрацював, про що сигналізує п’єзодинамік (грає музика), а на екрань з’являється відповідне повідомлення (рис. 3.31, в).

(а)



(б)

(в)

Рисунок 3.31 – Повністю налаштований годинник з будильником на 1:58

Якщо користувач не натисне на кнопку 3, то музика сама автоматично припиниться через хвилину.

Для переходу в режим секундоміра і назад необхідно натиснути на кнопку 1 (рис. 3.32).

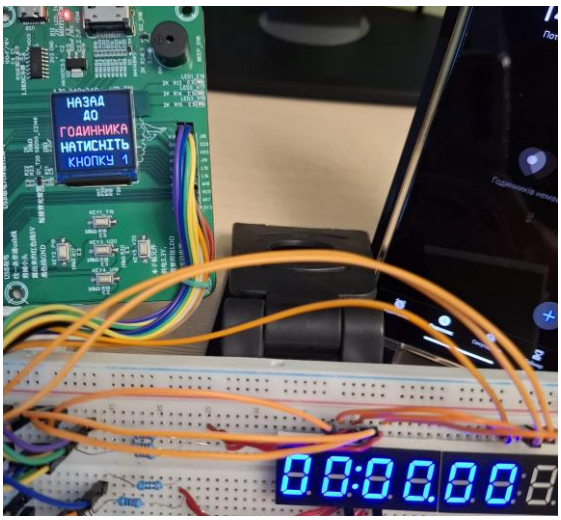


Рисунок 3.32 – Режим секундоміра

Для запуск чи зупинки секундомері необхідно натиснути на кнопку 2 (рис. 3.33, а), а для скидання секундомір треба спочатку зупинити (кнопка 2), а потім скинути (кнопка 3), про що нагадує відповідне повідомлення на екрані (рис. 3.33,б).

(а)

(б)

Рисунок 3.33 – Повідомлення для керування секундоміром

Для перевірки коректності роботи запускаємо на телефоні секундомір і намагаємося запустити його паралельно з секундоміром на платі (рис. 3.34). Розбіжність приблизно в 10 сотих секунди пояснюється як раз тим, що важко натискати одночасно. В цілому секундомір працює абсолютно коректно.

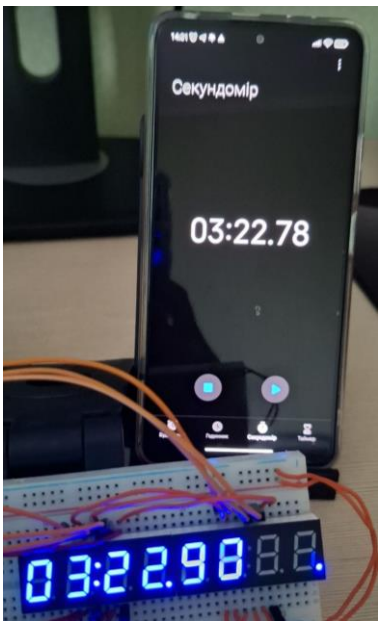


Рисунок 3.34 – Робота секундоміра

Відеодоказ працездатності проєкта представлено у презентації.

ВИСНОВКИ

На відміну від традиційного процесу проектування, де апаратне та програмне забезпечення розробляються ізольовано та інтегруються пізніше в життєвому циклі проекту, спільне проєктування дозволяє паралельно розробляти ці компоненти.

Vivado Design Suite – це потужний набір програмних інструментів, що забезпечує повний цикл проєктування, включаючи етапи створення вихідних описів проекту на мовах опису апаратних засобів (VHDL/Verilog), синтезу, моделювання, розміщення і трасування в кристалі, конфігурації чіпів та внутрішнього апаратного налагодження. А Vitis Unified IDE, в свою чергу, підтримує розробку програмного забезпечення на C, дозволяючи інтеграцію із системами на кристалі.

Запропонована модель багатофункціонального ручного годинника була перевірена на платі SoC EBAZ4205, доукомплектованої платою розширення. Результат програмування показав коректність моделі та, в загальному, доцільність використання САПР Vivado та Vitis для спільного проєктування апаратного та програмного забезпечення вбудованих систем.

Наукова новизна визначається використанням методології спільного проєктування апаратного та програмного забезпечення для розробки вбудованих систем на базі SoC, зокрема застосуванням сучасних інструментів, таких як Vivado Design Suite та Vitis Unified IDE. У подальших дослідженнях можна зосередитись на розширенні можливостей інтеграції програмно-апаратних рішень для більш складних і спеціалізованих застосувань, таких як обробка сигналів, машинне навчання або інші ресурсоємні завдання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Шкіль О.С., Рахліс Д.Ю., Філіпенко І.В., Корнієнко В.Р., Рожнова Т.Г. Автоматизоване проєктування вбудованих систем цифрового оброблення сигналів на платформі SOC // Сучасний стан наукових досліджень та технологій в промисловості. 2024. No 1 (27). С. 72–83. DOI: https://doi.org/10.30837/ITSSI.2024.27.192

2. Shkil O., Filippenko O., Rakhlis D, Filippenko I, Kornienko V. Analysis of the implementation efficiency of digital signal processing systems on the technological platform SoC Zynq 7000 // Radioelectronic and Computer Systems. 2024. No. 4(112). P. 168–177. DOI: https://doi.org/10.32620/reks.2024.4.14

3. Crockett, L., 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. UK: Elsevier, 2014. 484 p.

4. Vivado design suite user guide: design flows overview. USA: AMD, 2022. 96 p. URL: https://www.xilinx.com/support/documents/sw_manuals /xilinx2022_2/ug904-vivado-implementation.pdf (access date: 06.05.2025)

5. Vitis Unified Software Platform Tutorials Landing Page. USA: AMD, 2024. URL: https://docs.amd.com/r/2024.1-English/Vitis-Tutorials-Getting-Started/Vitis-Introduction-and-Getting-Started (access date: 07.05.2025)

6. Merrick R. Getting Started with FPGAs: digital circuit design, Verilog, and VHDL for beginners, USA: No Starch Press, 2023. 320 p.

7. Miroshnyk M., Shkil O., Rakhlis D., Filippenko I., Kulak E. Verification of FPGA control systems by analyzing the correctness of state diagrams // DESSERT'2020: Proc. of international conf., 14-18 may, 2020. Kyiv, Ukraine, 2020. P. 1–5. DOI: https://doi.org/10.1109/dessert50317.2020.9125067

8. Vivado design suite user guide: designing IP subsystems using IP Integrator. USA: AMD, 2022. 289 p. URL: https://www.xilinx.com/support/documents/sw_manuals/xilinx2022_1/ug994-vivado-ip-subsystems.pdf (access date: 08.05.2025).

9. Мірошник М.А., Шкіль О.С., Рахліс Д.Ю., Пшеничний К.Ю., Мірошник А. Модель обробки подій для моделювання пристроїв логічного керування реального часу // Збірник наукових праць «Вісник ЧДТУ». 2023. № 2. С. 50–57. DOI: https://doi.org/10.24025/2306-4412.2.2023.274840