МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ПРОЕКТУВАННЯ СИСТЕМ НА КРИСТАЛІ.
ЛАБОРАТОРНИЙ ПРАКТИКУМ.

Навчальний посібник

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня магістра
за освітньою програмою «Інформаційно-обчислювальні засоби електронних систем»
спеціальності 172 Телекомунікації та радіотехніка

Укладачі: О. І. Антонюк, Д. Ю. Лебедев

Електронне мережне навчальне видання

Київ
КПІ ім. Ігоря Сікорського

2022

Рецензент *Сафронов П.С.*, к.т.н, доцент,
кафедра електронних пристроїв та систем

Відповідальний редактор *Лисенко О.І.* старший викладач
кафедра конструювання електронно-обчислювальної апаратури

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 6 від 24.06.2022 р.)
за поданням Вченої ради факультету електроніки
(протокол № 5/22 від 31.05.2022 р.)

Викладено вимоги до виконання лабораторних робіт з дисципліни «Проектування систем на кристалі». Посібник містить короткі теоретичні відомості та завдання, щодо виконання лабораторних робіт у Quartus Prime 18.1 Lite із платою Terasic DE1-SoC. Описано використання інструменту Platform Designer Tool, який дозволяє розробнику створювати системи на кристалі із бібліотечних компонентів. Наведений матеріал лежить в основі ряду лабораторних робіт, метою яких є формування у студентів компетентностей, що мають дозволити їм виконувати аналіз та синтез систем на кристалі.

Посібник призначений для здобувачів ступеня магістра за спеціальністю 172 Телекомунікації та радіотехніка та може бути також корисним для студентів та аспірантів радіотехнічних та телекомунікаційних спеціальностей, а також для інженерно-технічних спеціалістів, що працюють в галузі цифрової електроніки.

Реєстр. № 21/22-850. Обсяг 4 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Перемоги, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів
і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

Зміст

ПЕРЕДМОВА

Даний посібник призначений для забезпечення виконання лабораторних робіт з дисципліни «Проектування систем на кристалі» для здобувачів ступеня магістра.

Дана дисципліна вивчається з метою опанування методів розробки систем на кристалі з використанням високорівневого проектування. На лабораторних заняттях студенти навчаються проектувати пристрої з вбудованими та синтезованими процесорними ядрами, виконувати їх тестування на відповідність специфікації та аналізувати результати.

Предметом навчальної дисципліни є системи на кристалі Cyclone V фірми IntelFPGA та САПР для синтезу цифрових систем Platform Designer.

Навчальна дисципліна «Проектування систем на кристалі» вивчається паралельно з навчальними дисциплінами «Системи комп'ютерного зору» і «Основи нейромережних технологій» і створює базу для вивчення дисциплін з вибіркового циклу, таких як «Сучасні методи синтезу обчислювальних пристроїв», «Комп'ютерні технології проектування електронних засобів» і «Радіоелектронні обчислювальні засоби на основі цифрових сигнальних процесорів». Метою навчальної дисципліни є формування у студентів здатності:

- грамотного і правильного застосування досягнень сучасної цифрової техніки (систем на кристалі);
- проводити оцінку та вибір апаратних та програмних засобів цифрової обробки сигналів;
- розробляти проекти для ПЛМ;
- виконувати налагодження та супроводження різноманітних цифрових систем обробки інформації;
- розробляти та реалізовувати заходи по збільшенню надійності пристроїв цифрової обробки сигналів.

1. Лабораторна робота № 1.
Перші кроки у Platform Designer Tool

1. Вступ

Platform Designer Tool – це інструмент, який дозволяє розробнику створювати системи на кристалі із бібліотечних компонентів, наприклад - пам’яті, інтерфейсів введення-виведення, таймерів, процесорного ядра, обробників відео/аудіо потоків, тощо. У даному описі розглянуто роботу у Quartus Prime 18.1 Lite із платою Terasic DE1-SoC.

Технологію розробки апаратної системи проілюстровано послідовністю покрокових інструкцій, щодо використання середовища **Platform Designer Tool** (попередня назва – **Qsys**) спільно з програмним забезпеченням **Quartus® Prime**. Останнім кроком процесу розробки є налаштування розробленого пристрою на платі з мікросхемою FPGA та запуск прикладної програми.
Малюнки в роботі були отримані з використанням середовища **Quartus Prime** версії 18.1. Для інших версій програмного забезпечення ці малюнки можуть трохи відрізнятися [1].

2. Плати Intel® FPGA серії DE

Для цієї роботи необхідно мати плату Intel серії DE, наприклад, таку, що показана на рис. 1.

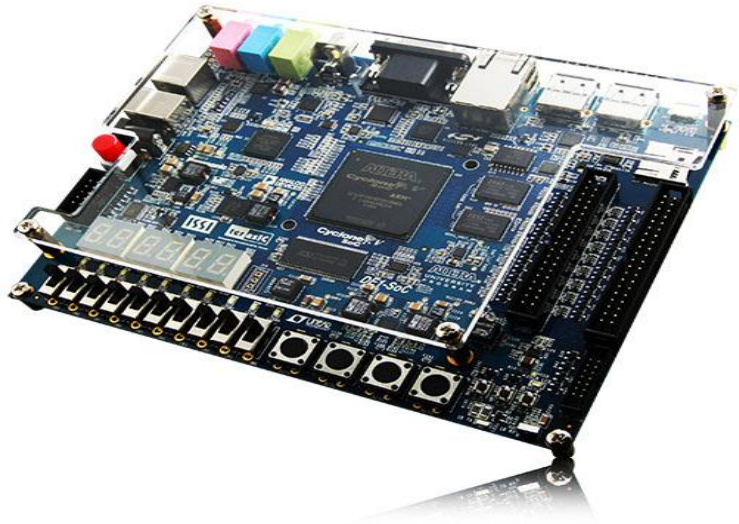

Рис. 1. Плата DE1-SoC кампанії Intel

На рис. 1 зображена плата DE1-SoC, на якій встановлено мікросхему FPGA Cyclone® V. Плата має багато додаткових ресурсів, таких як мікросхеми пам'яті, повзунки-перемикачі, кнопки, світлодіоди, аудіовхід / вихід, відеовхід (з декодером NTSC / PAL) та відео вихід (VGA). Вона оснащена кількома типами

послідовних вхідних / вихідних з'єднань, включаючи USB-порт для підключення плати до персонального комп'ютера. У цій роботі буде використано лише невелику частину ресурсів: мікросхему FPGA, повзунки-перемикачі, світлодіоди та порт USB, який підключається до комп'ютера.

3. Приклад апаратної цифрової системи

У цій роботі буде використовуватися проста апаратна система, яка показана на рис. 2. Вона складається з вбудованого процесорного ядра Nios II, що являє собою програмний процесорний модуль, описаний мовою опису апаратури [2]. Процесор Nios II може бути частиною більшої системи, а сама система може бути реалізована в мікросхемі Intel FPGA за допомогою програмного забезпечення Quartus Prime.

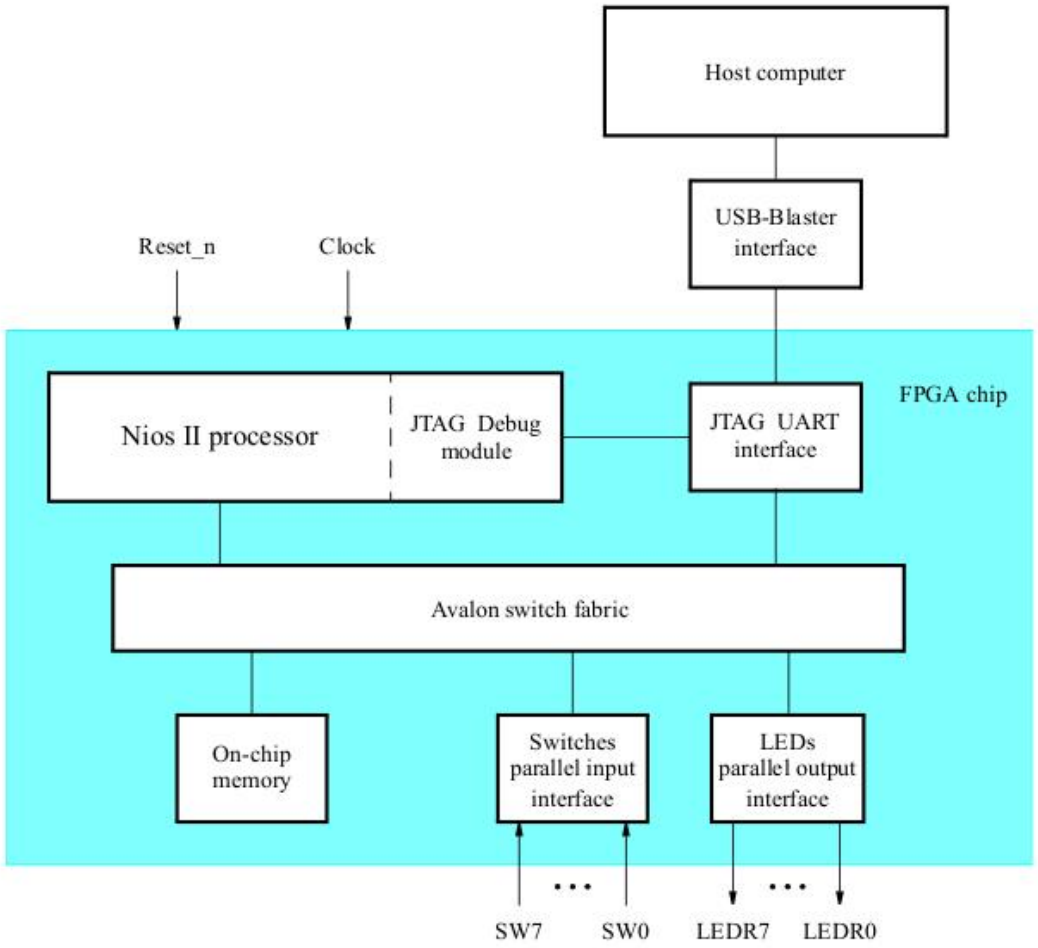

Рис. 2. Приклад системи з ядром Nios II

Як показано на рис. 2, процесор Nios II підключається до інтерфейсів пам'яті та вводу / виводу за допомогою системи зв'язку, яка має назву **комутатор шини Avalon**. Ця шина генерується автоматично середовищем Qsys. Компонент пам'яті в наведеної системи буде реалізовано за допомогою

вбудованої пам'яті, що є в мікросхемі FPGA. Інтерфейси вводу / виводу, які підключаються до повзункових перемикачів та світлодіодів, буде реалізовано за допомогою стандартних модулів, які присутні у середовищі Qsys. Спеціальний інтерфейс JTAG UART використовується для підключення до мікросхеми, та забезпечує обмін USB-посиланнями з хост-комп'ютером, до якого підключена плата DE. Цей пристрій та його супутнє програмне забезпечення мають назву USB-Blaster.

Інший модуль, що зветься JTAG Debug, забезпечує можливість керування хост-комп'ютера процесором Nios II. Це дає можливість виконувати такі операції, як завантаження програм для Nios II в пам'ять, запуск та зупинку виконання цих програм, встановлення точок переривання та аналізу вмісту комірок пам'яті та регістрів Nios II.

Оскільки всі частини ядра Nios II, які реалізовані у мікросхемі FPGA, визначаються за допомогою мови опису апаратури, досвідчений користувач може написати такий код для реалізації будь-якої частини. Але, це було б обтяжливе і трудомістке завдання. Натомість існує можливість використання бібліотеки середовища Qsys для реалізації необхідної системи, просто вибираючи відповідні компоненти та вказуючи параметри, необхідні для пристосування кожного компонента до загальних вимог системи. Хоча, в цьому прикладі, ілюструються можливості середовища Qsys для реалізації дуже простої системи, той же підхід може використовуватися для проектування складних систем.

Система, яка зображена на рис. 2, призначена для реалізації простого завдання. Вісім перемикачів на платі DE1-SoC, SW7 - 0, використовуються для включення або вимкнення восьми світлодіодів LEDR7 - 0. Щоб вирішити це завдання, восьмибітна послідовність, яка відповідає поточному стану перемикачів, повинна бути надіслана до вихідного порту для активації світлодіодів. Це відбувається за допомогою програми, яка виконується процесором Nios II та зберігається в пам'яті мікросхеми. Потрібна безперервна робота програми, для відповідної індикації зміни стану перемикачів.

Далі буде розглянуто процес проектування пристрою, зображеного на рис. 2, з використанням середовища Qsys. Після призначення контактів FPGA, для реалізації з'єднань між паралельними інтерфейсами перемикачів та світлодіодів плати DE1-SoC, проект компілюється. Використовуючи програмне середовище **Intel® FPGA Monitor Program**, конфігурується розроблена схема на пристрої FPGA та завантажується на виконання програма для процесора Nios II.

4. Послідовність виконання роботи

1. Для початку створіть новий проект. Розмістіть його у директорії ***platformdesigner_tutorial*** та назвіть ***lights*** (рис. 3).

Рис. 3. Створення нового проекту

Це має бути пустий проект (***empty project***), не додавайте ніяких файлів, виберіть свою мікросхему (**5CSEMA5F31C6** для DE1-SoC) Налаштування **EDA Tool settings** залиште за замовчанням.

2. Відкрийте **Platform Designer Tool**. Для цього оберіть меню ***Tools*** → ***Platform Designer***. Відкриється вікно як на рис. 4.

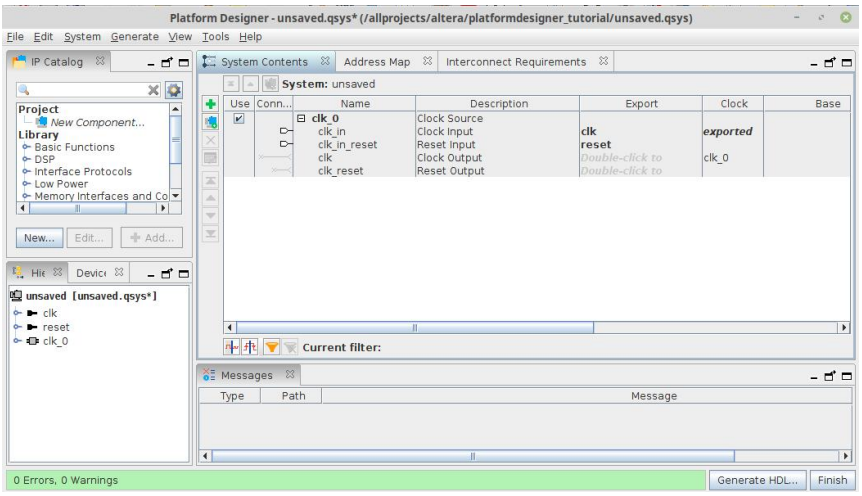

Рис. 4. Вікно Platform Designer Tool

Як ви бачите, система вже містить один компонент ***clk_0***. Двічі натисніть на нього, щоб переглянути його параметри (рис. 5). Частота тактового сигналу зараз встановлена 50 МГц, це можна змінити, або вказати, що частота заздалегідь не відома, але у даній роботі потрібно залишити її саме такою.

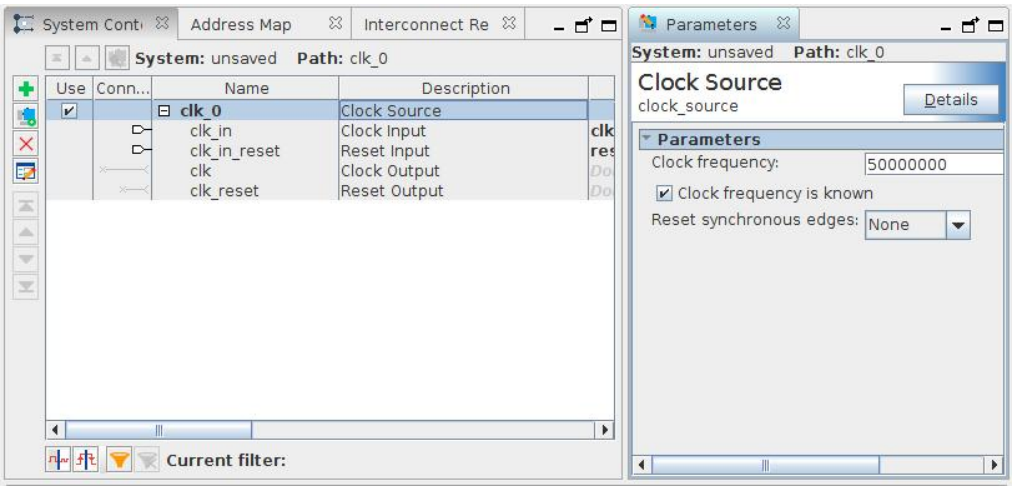

Рис. 5**.** Параметри clk_0

3. Додайте до системи процесорне ядро. Для цього, у вкладці ***IP Catalog*** (вкладка знаходиться зліва у вікні ***Platform Designer Tool***) оберіть ***Library → Processors and Peripherals → Embedded Processors → Nios II (Classic) Processor*** та натисніть кнопку ***Add.*** Відкриється вікно зображене на рис. 6.

У консольному вікні (знизу) може з'явитися сповіщення про деякі помилки! Не хвилюйтеся, ви їх пізніше виправите.

4. У вікні налаштування процесорного ядра оберіть ***Nios II/e*** — економічну версію процесора (рис. 7) і натисніть ***Finish***. Усі інші налаштування зробите потім.

На рис. 8 показано, як має виглядати результат.

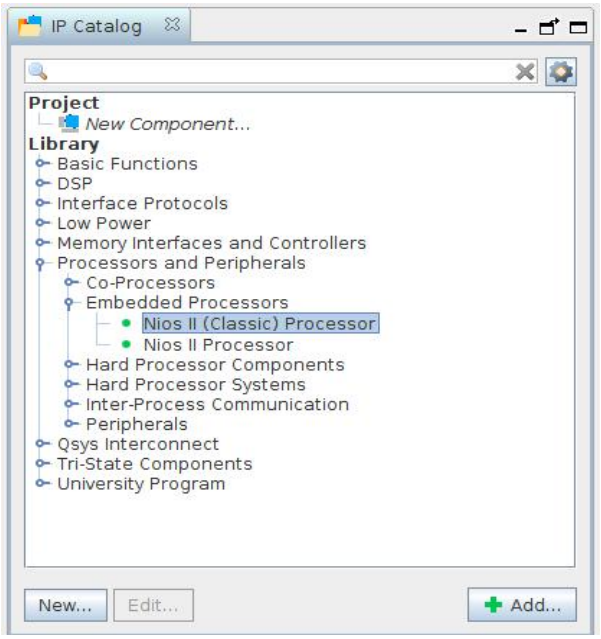

Рис. 6. Додавання ядра Nios II

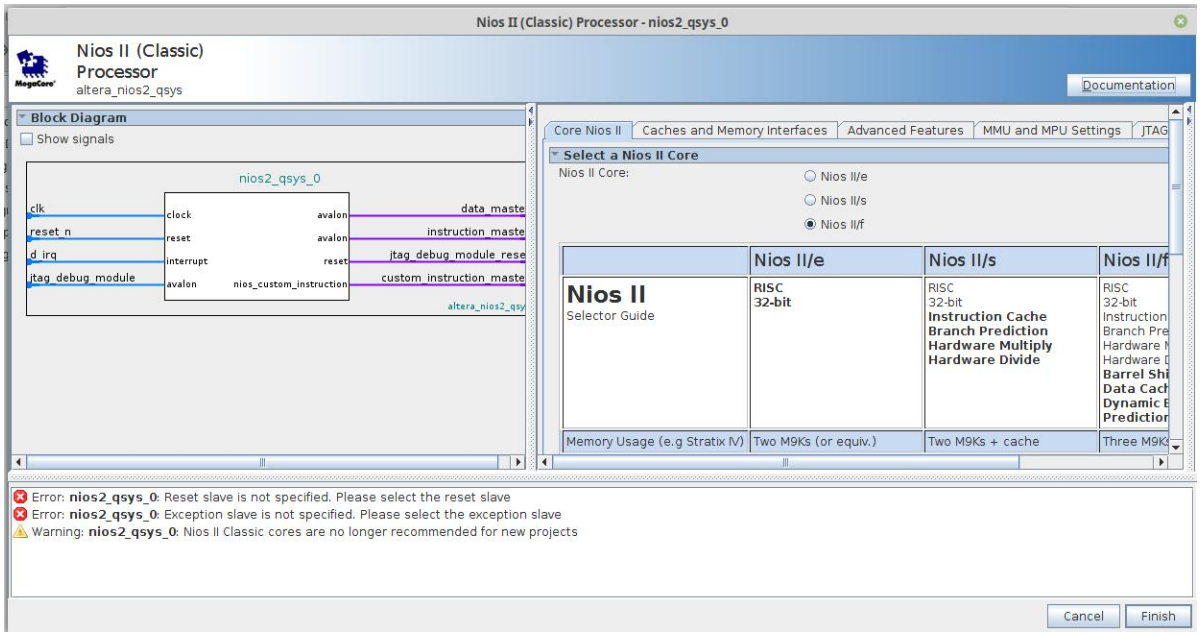

Рис. 7. Вікно налаштування Nios II

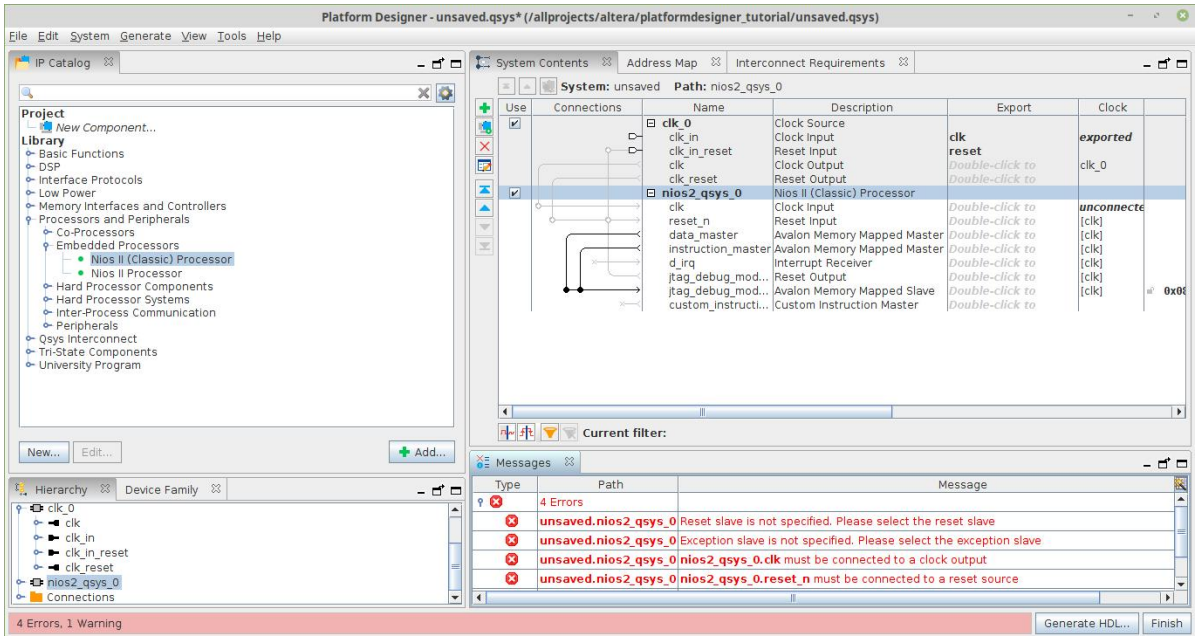

Рис. 8. Вигляд системи після додавання Nios II/e

4. Додайте пам’ять. Для цього оберіть ***IP Catalog → Library → Basic Functions → On Chip Memory → On-Chip Memory (RAM or ROM) Intel FPGA IP*** (рис. 9).

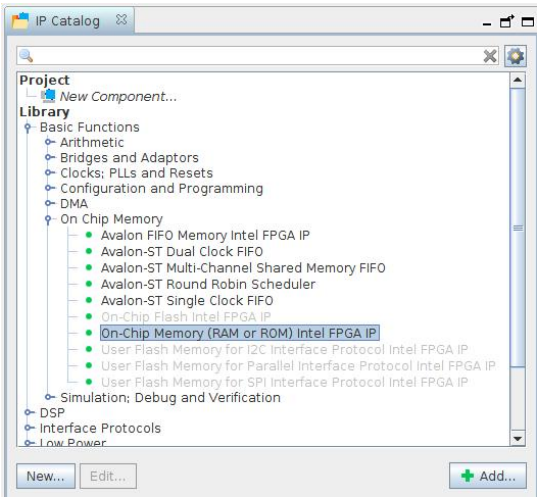

Рис. 9. Додавання пам’яті

Зробіть наступні налаштування ***Slave S1 data width = 32*** та ***Total memory size = 4096 bytes*** (рис. 10). Потім натисніть ***Add***.

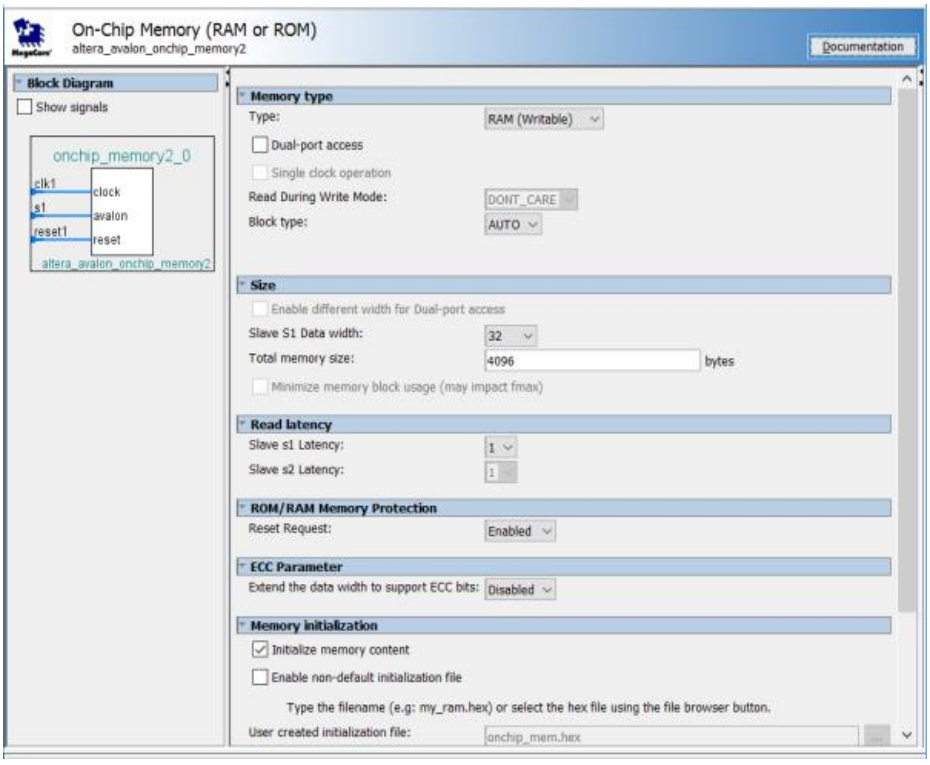

Рис. 10. Налаштування пам’яті

6. Додайте два 8-бітні порти введення/виведення. Для цього оберіть ***IP Catalog → Library → Processors and Peripherals→ Peripherals → PIO (Parallel I/O) Intel FPGA IP*** та натисніть ***Add***. Налаштуйте їх так, як зображено на рис.11 та рис.12.

Змініть назви портів введення/виведення на “***switches***” (порт введення) та “***LEDs***” (порт виведення). Для цього потрібно натиснути правою кнопкою миші на назві порту, та обрати команду ***Rename*** з контекстного меню. Потім - ввести нову назву.

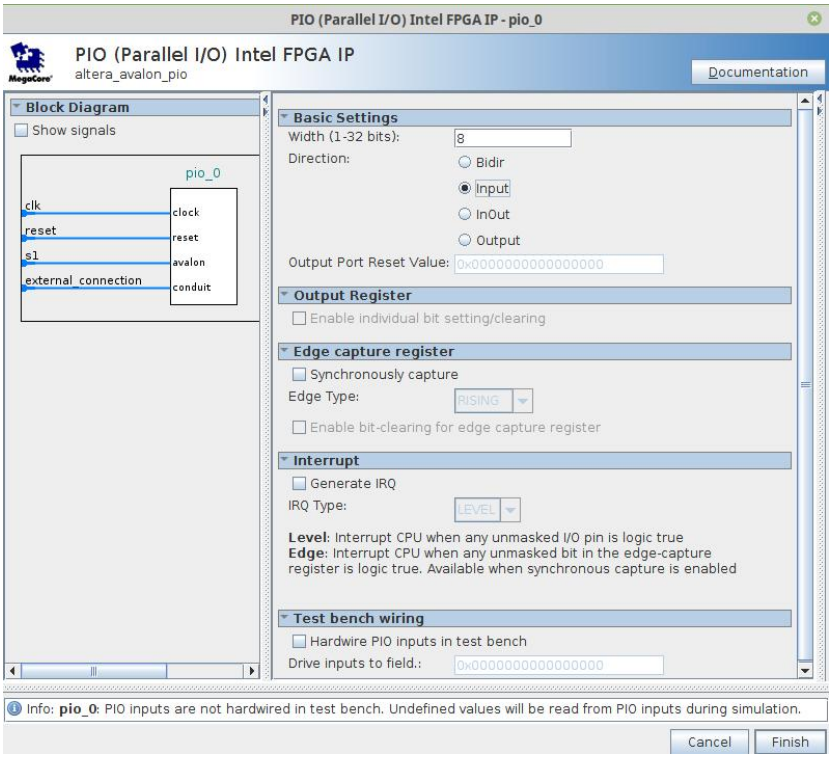

Рис. 11. Налаштування порту введення

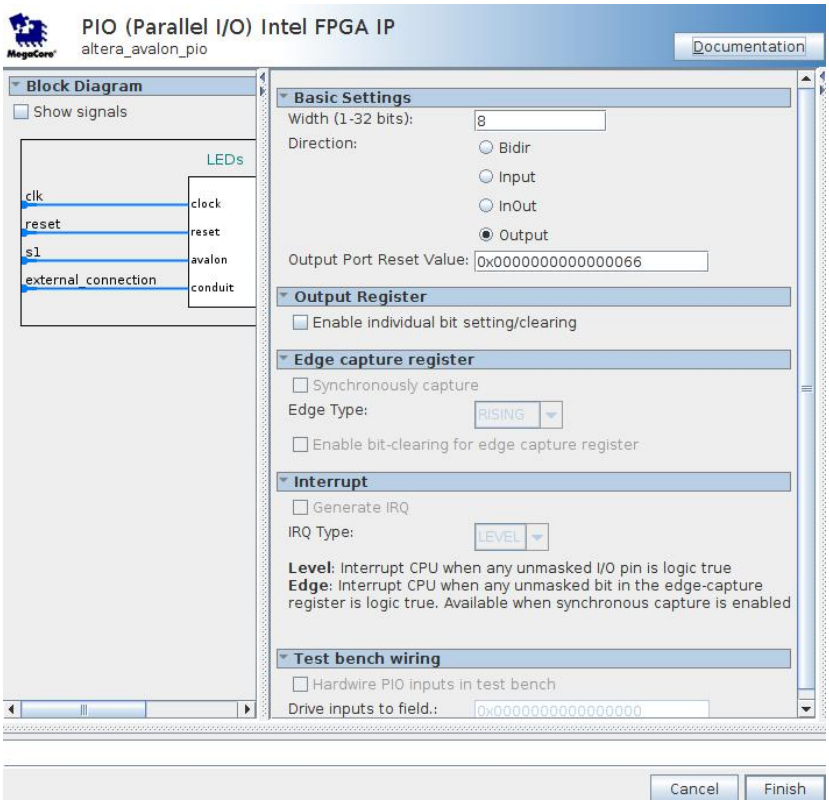

Рис. 12. Налаштування порту виведення

7. Додайте JTAG UART, для цього оберіть ***IP Catalog → Library → Interface Protocols → Serial → JTAG UART Intel FPGA IP*** та натисніть ***Add***. Відкриється вікно у якому все залиште за замовчанням.

8. У вікні ***System Contents*** відобразяться додані компоненти, наявні та можливі зв’язки між ними (рис. 13).

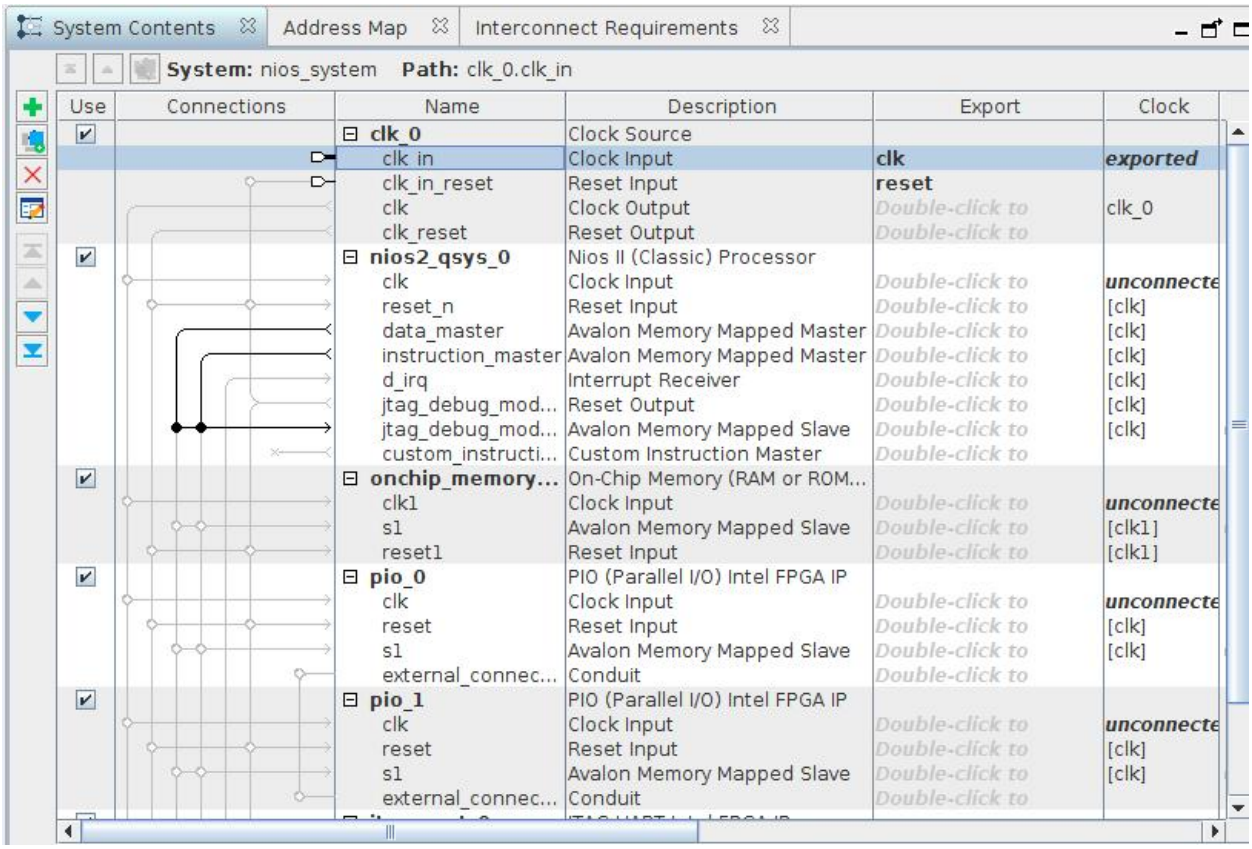

Рис. 13**.** Вигляд вікна System Contents

Натискаючи на кружечки у стовпці ***Connections*** можна додавати та прибирати зв’язки між компонентами системи. Налаштуйте зв’язки так, як вказано на рис. 14.

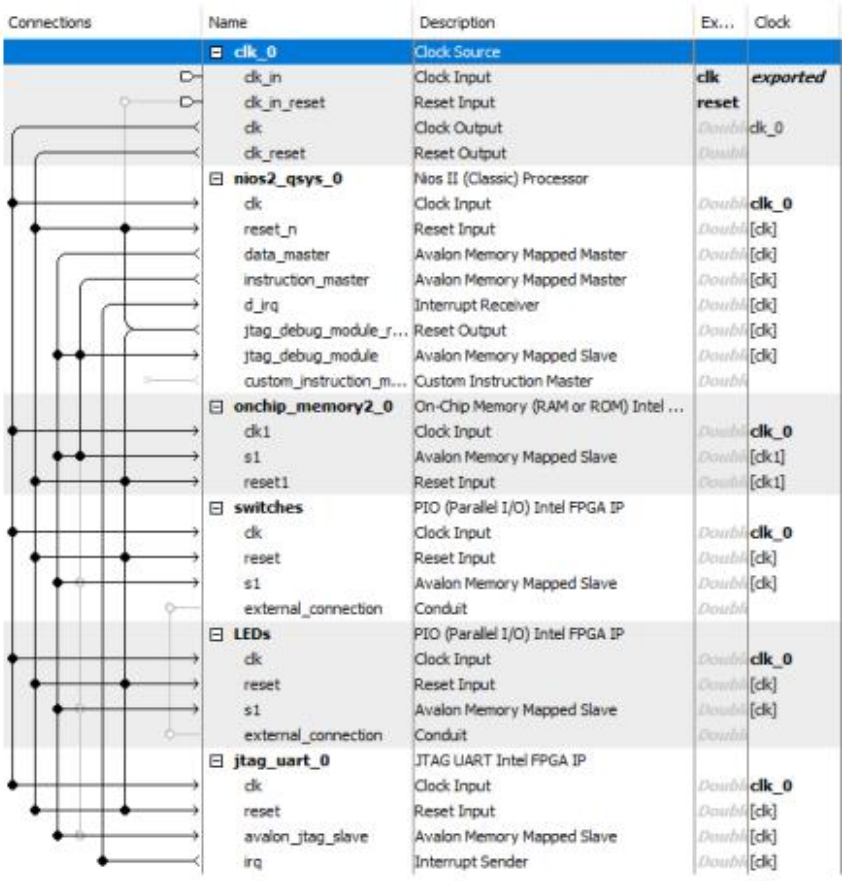

Рис. 14. Налаштування зв’язків між компонентами

9. Під’єднайте лінію запиту на переривання (IRQ = Interrupt Request), якщо вона не під’єдналась автоматично до ядра процесора. Призначте перериванню номер 5. Для цього оберіть квадратик у стовпці ***IRQ*** у рядку ***jtag_uart_0 → IRQ*** та введіть “5”.

10. Далі необхідно встановити базові адреси для пристроїв, під’єднаних до шини *Avalon*. Їх можна призначити вручну, або автоматично. Блоку пам’яті призначте адресу вручну. Для цього двічі натисніть на значенні у стовпці ***Base*** рядка ***onchip_memory → s1*** (рис. 15).

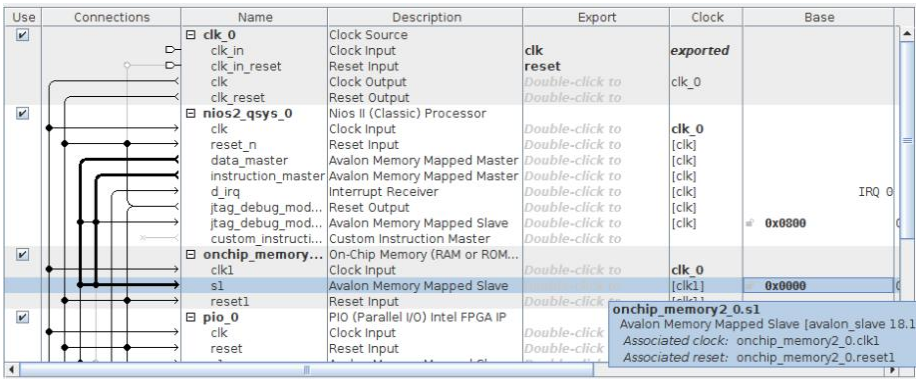

Рис. 15. Призначення адреси блоку пам’яті

Встановіть значення 0x0000. Значення має бути саме таким — це важливо! Потім заблокуйте цю адресу, натиснувши на відповідний символ поруч.

11. Інші адреси призначте автоматично. Оберіть у меню команду ***System → Assign Base Addresses***. Наразі, базові адреси компонентів визначені коректно.

12. На наступному кроці необхідно виконати налаштування у пам'яті адреси розташування векторів переривань (***Exception memory***) та початку виконання програми (***Reset Vector memory***) для ядра Nios II. Для цього натисніть правою кнопкою миші на ***nios2_qsys_0*** та оберіть з контекстного меню команду ***Edit***. Для параметрів ***Reset vector memory*** та ***Exception vector memory*** оберіть ***onchip_memory2_0.s1*** (рис. 16).

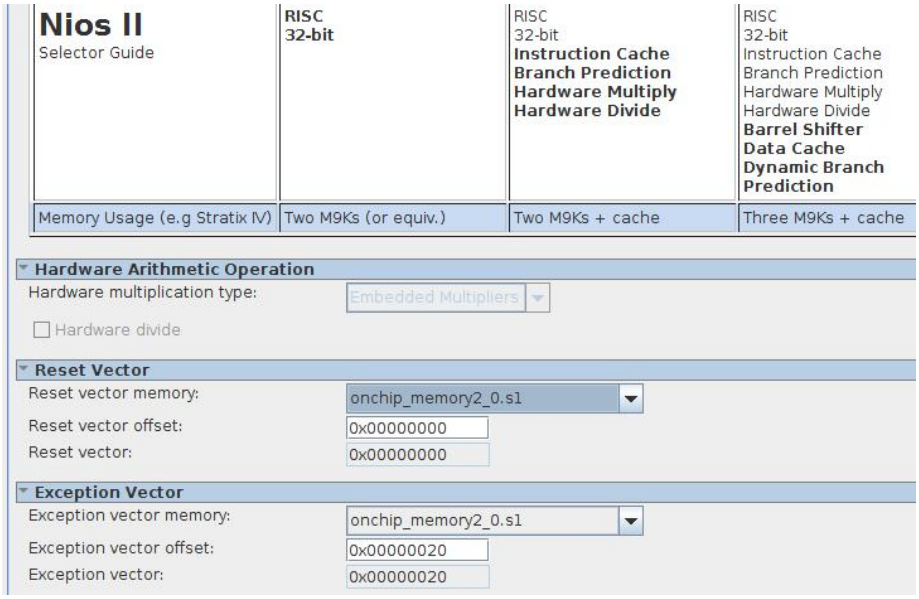

Рис. 16. Налаштування векторів переривань та початку виконання програми процесора Nios II

Всі інші параметри залиште за замовченням. Натисніть ***Finish***. Зверніть увагу - всі помилки (Errors) зникли!

13. Далі необхідно налаштувати передачу сигналів портів ***switches*** та ***LEDs*** за межі системи, яка налаштовується у Platform Designer Tool. Для цього, двічі натисніть на напис ***Double-click to export*** у стовпці ***Export*** у рядку ***switches → external_connection*** (див. рис. 17). Впишіть ім'я ***switches***. Так само для порту ***LEDs*** впишіть ім'я ***leds***. Результат зображено на рис. 18.

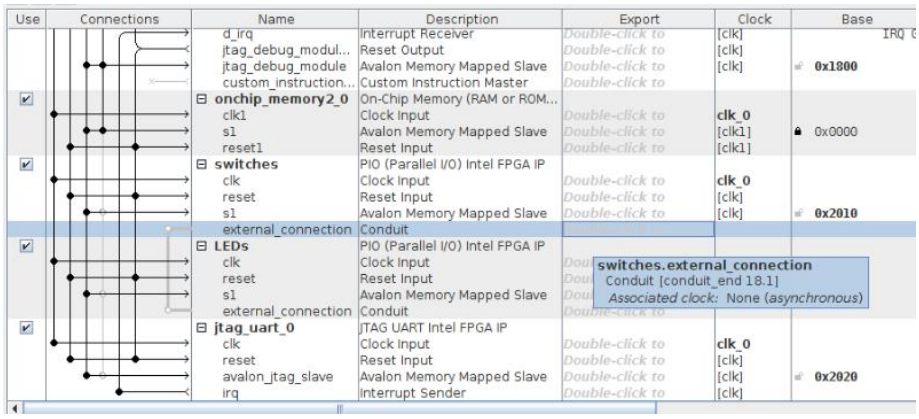

Рис. 17. Натисніть на напис Double-click to export

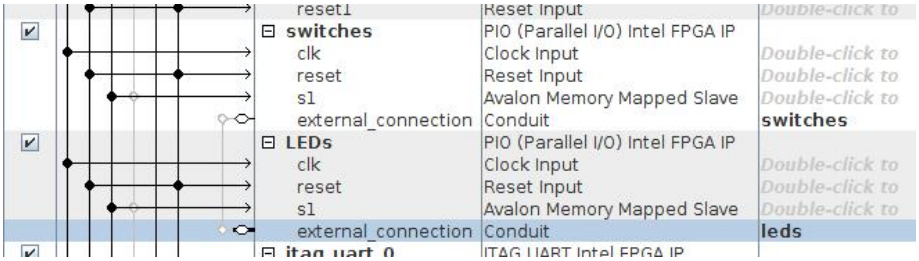

Рис. 18. Налаштування портів

14. Збережіть систему під назвою ***nios_system***.

15. Далі оберіть команду ***Generate → Generate HDL***. Для ***Create simulation model*** оберіть ***None***. У цій роботі не буде виконуватися симуляція пристрою. Результати налаштувань відображені на рис. 19. Натисніть *Generate* та дочекайтеся сповіщення ***Generate completed***.

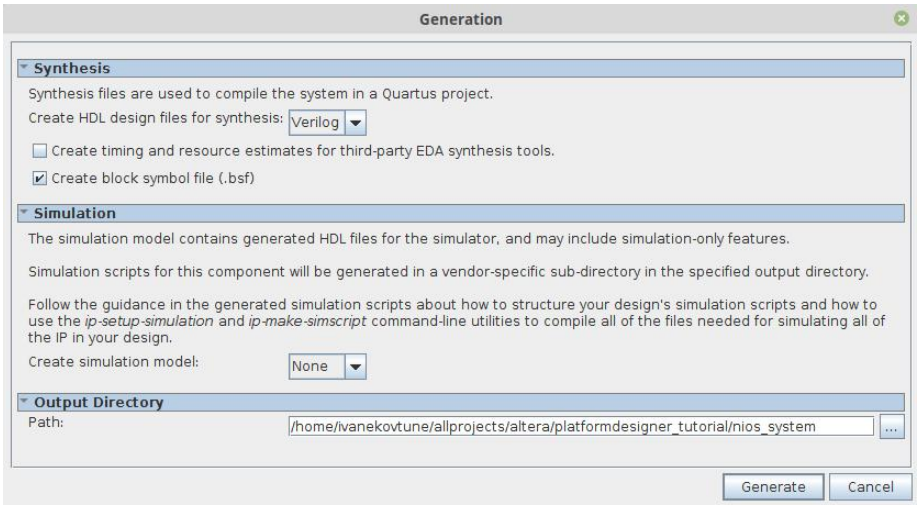

Рис. 19. Налаштування у вікні Generation

16. Поверніться до головного вікна середовища ***Quartus Prime***. Наразі необхідно долучити до проекту створену ***nios_system***. Для цього у вікні ***Project***

Navigator → Files (зліва) натисніть правою кнопкою миші на піктограмі ***Files*** та оберіть команду ***Add/Remove Files in Project*** з контекстного меню. Відкриється вікно додавання файлів. Додайте файл ***platformdesigner_tutorial/nios_system/synthesis/nios_system.v*** (рис. 20). Так само додайте файл ***nios_system.qip***, що знаходиться у папці разом з ***nios_system.v***.

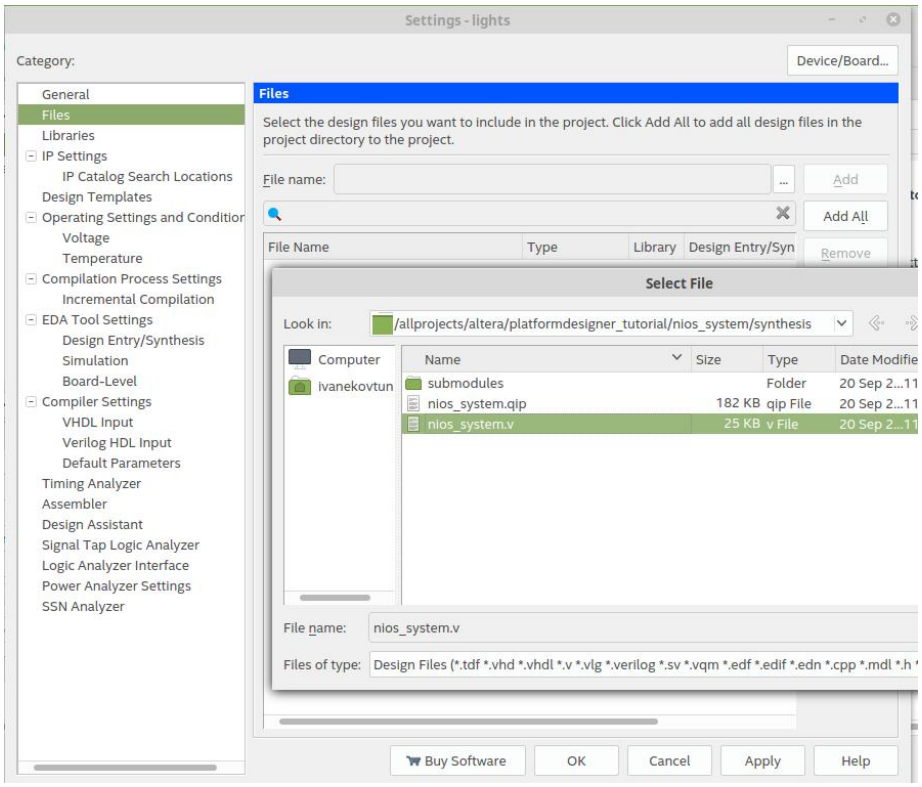

Рис. 20. Додання файлу nios_system.v

17. Створіть файл верхнього рівня ієрархії ***lights.v***. У ньому буде підключено щойно згенеровану та додану до проекту систему з ядром Nios. Додайте у файл ***lights.v*** наступний текст:

```
module lights (CLOCK_50, SW, KEY, LEDR);
    input CLOCK_50;
    input [7:0] SW;
    input [0:0] KEY;
    output [7:0] LEDR;
// Instantiate the Nios II system module generated
// by the Platform Designer tool:
```

```
    nios_system NiosII (
        .clk_clk(CLOCK_50),
        .reset_reset_n(KEY),
        .switches_export(SW),
        .leds_export(LEDR));

endmodule
```

18. Виконайте призначення контактів мікросхеми (pin assignment) для плати DE1-SoC. Для цього оберіть меню ***Assignments → Import Assignments*** та вкажіть файл ***DE1_SoC.qsf***.

19. Запустіть компіляцію проекту.

20. Завантажте скомпільований проект до плати DE1-SoC за допомогою ***Programmer Tool***. (Підказка: якщо ви раніше не працювали саме з DE1-SoC, то вам потрібно буде натиснути ***Add Device*** та обрати ***Soc Series V → SOCVHPS*** перш ніж завантажувати).

21. Наступний етап – розробка програмного забезпечення для процесорного ядра. Створіть файл ***lights.s*** у директорії ***platformdesigner_tutorial/app_software*** (директорію теж необхідно створити). Помістіть у файл наступний текст:

```
.equ   switches,    0x00002010
.equ   leds,        0x00002000
.global     _start
_start:moviar2,     switches
            moviar3,    leds
LOOP:       ldbio  r4,    0(r2)
            stbio  r4,    0(r3)
            br          LOOP
.end
```

Це програма мовою Асемблер для Nios II. Замість чисел 0x00002010 та 0x00002000 укажіть базові адреси відповідних портів вашого пристрою (якщо адреси відрізняються).

22. Запустіть ***Intel FPGA Monitor Program***. (Підказка: ця програма не є частиною Quartus Prime, встановлюється і запускається окремо від нього).

23. Коли відкриється головне вікно програми, оберіть команду ***File → New Project***. На першій вкладниці вкажіть наступні значення:

Project Directory = platformdesigner_tutorial/app_software

Project Name = lights_example

Architecture = Nios II

Натисніть ***Next.***

24. На вкладинці ***Specify the system*** вкажіть наступні параметри (рис. 21):

Select a system = <Custom system>

System description file =
platformdesigner_tutorial/nios_system.sopcinfo

FPGA programming (SOF) file =
platformdesigner_tutorial/output_files/lights.sof

Preloader = Not required

Натисніть ***Next.***

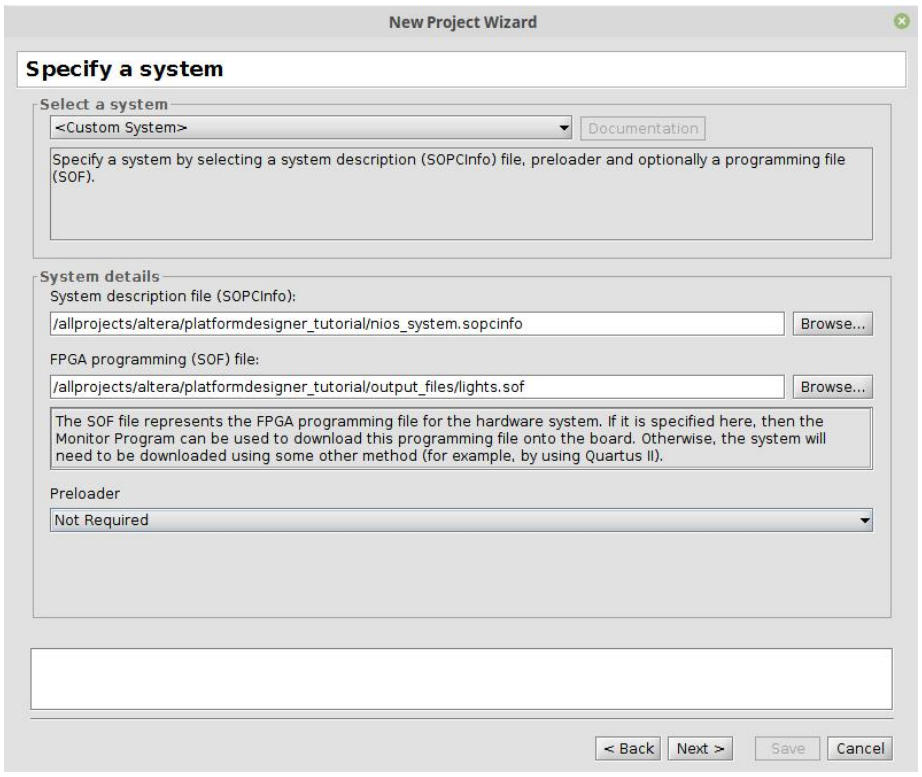

Рис. 21. Закладка Specify the system

25. На вкладинці ***Specify a program type*** укажіть значення ***Program Type = Assembly Program***. Натисніть ***Next.***

26. На вкладинці ***Specify program details*** натисніть ***Add,*** та оберіть файл програми ***lights.s***. Вкажіть ***Start symbol = _start***. Натисніть ***Next.***

27. На вкладинці ***Specify system parameters*** укажіть наступні значення

Host connection = DE-SoC [3-2]

Processor = nios2_qsys_0

Terminal device = jtag_uart_0

(Підказка: якщо немає параметра для ***Host connection***, то підключіть плату і натисніть ***Refresh***).

28. На останній вкладинці (рис. 22) натисніть ***Finish***, або ***Save***.

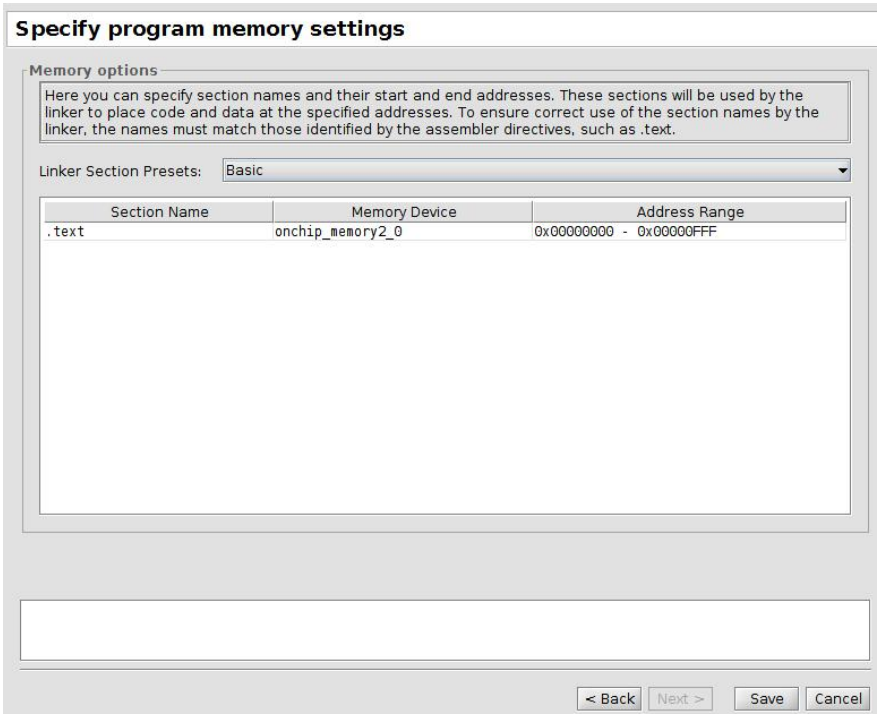

Рис. 22. Закладка Specify program memory settings

На запит чи завантажувати систему на плату (рис. 23) відповідайте ***Yes***.

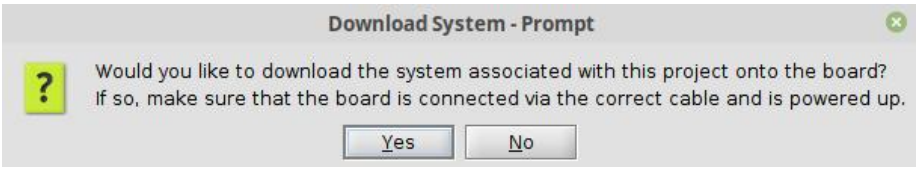

Рис. 23. Запит, на який треба відповісти Yes

29. Після сповіщення про успішне завантаження, у вікні ***Intel FPGA Monitor Program — lights_example*** (рис. 24) оберіть меню ***Actions → Compile & Load.***

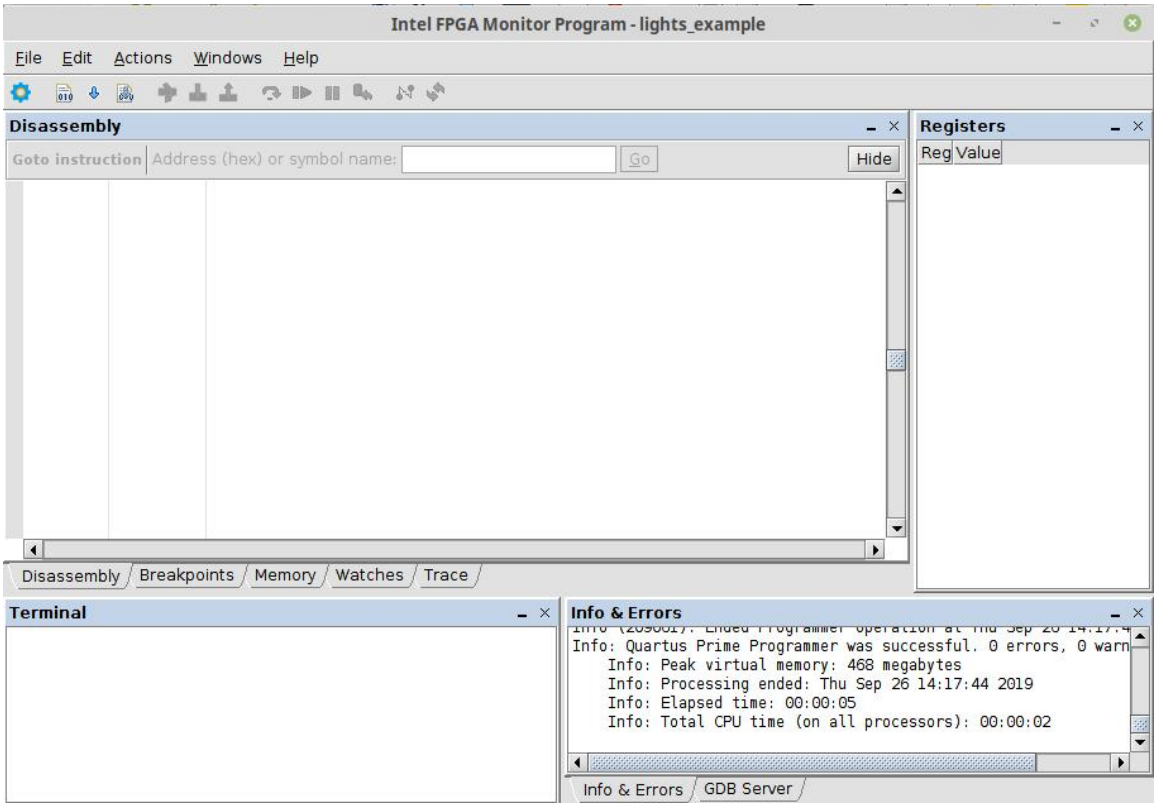

Рис. 24. Вікно Intel FPGA Monitor Program

Звертаємо увагу! Якщо ви отримали помилку та у вікні ***Info & Errors*** (рис. 25) з'явився напис (це стосується Linux):

make: *** No rule to make target 'compile'. Stop.

Рис. 25. Вікно Info & Errors

то виправити помилку можливо, якщо у директорії ***platformdesigner_tutorial/app_software*** відкрити файл ***makefile,*** та у строчці, наступній від # ***Targets***, слово ***COMPILE*** (великими літерами) замініте на ***compile*** (маленькими літерами).

30. Після завантаження програми, відкриється вікно дебагера. Воно показано на рис. 26.

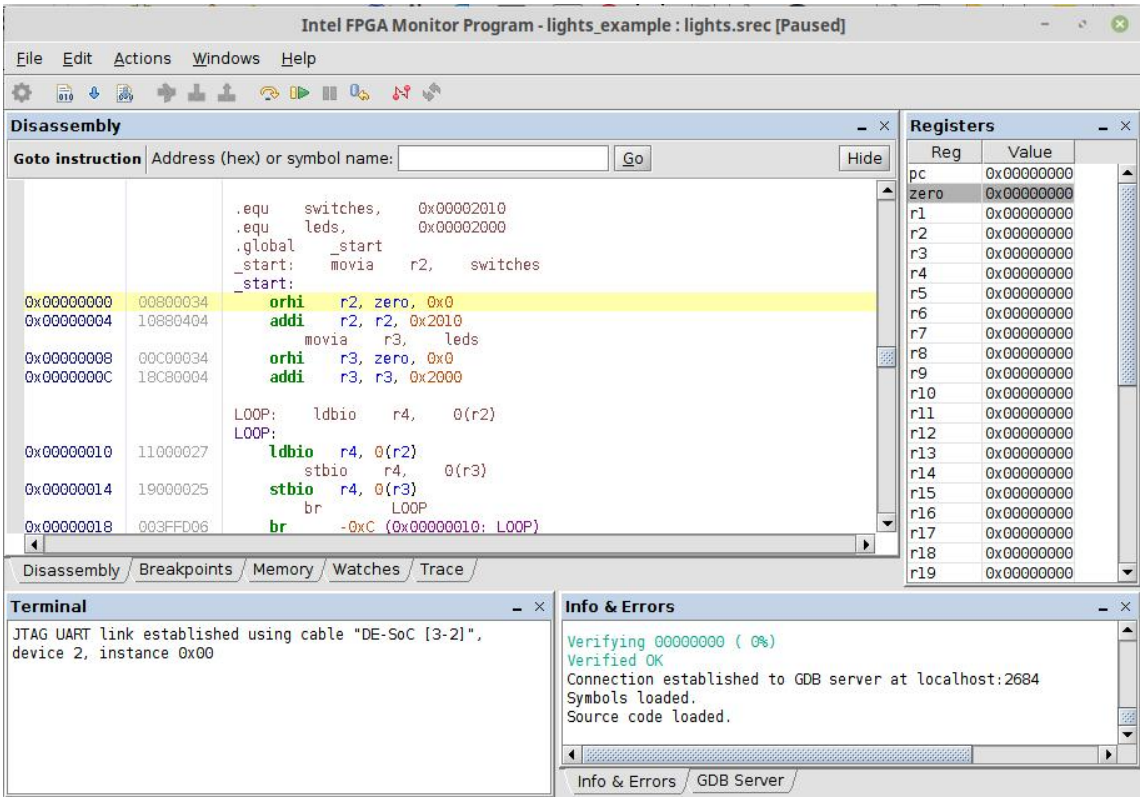

Рис. 26**.** Вікно дебагера

31. Запустіть програму на виконання у безперервному (зелений трикутник), або покроковому режимі (жовта стрілочка) та спостерігайте її виконання.

Переконайтеся, що червоні світлодіоди змінюють свій стан відповідно до стану перемикачів при виконанні інструкції
stbio R4, 0(R3).

5. Самостійна робота

Завдання 1. Для синтезованого процесорного ядра розробіть програму, яка використовує більш складну залежність між комбінацією перемикачів та послідовністю включення світлодіодів (наприклад, «вогник що біжить» та ін.). У якості шаблону використовуйте наступний код мовою С:

```
#define switches (volatile char *) 0x0002010
#define leds (char *) 0x0002000
void main()
{       while (1)
        *leds = *switches;
}
```

Завдання 2. Створіть новий проект та перевірте роботу програми, що обчислює результат накопичення добутків двох векторів [2].

Dot product = ΣA(i) x B(i)

Код програми наведено нижче. Поясніть, як працює програма та перевірте її виконання у покроковому режимі.

```
.include "nios_macros.s"
.global _start
_start:
    movia r2, AVECTOR    /* Register r2 is a pointer to vector A */
    movia r3, BVECTOR   /* Register r3 is a pointer to vector B */
    movia r4, N
    ldw r4, 0(r4) /*Register r4 is used as the counter for loop iterations*/
    add r5, r0, r0  /* Register r5 is used to accumulate the product */
LOOP:    ldw r6, 0(r2)       /* Load the next element of vector A */
            ldw r7, 0(r3)       /* Load the next element of vector B */
     mul r8, r6, r7      /* Compute the product of next pair of elements */
     add r5, r5, r8      /* Add to the sum */
     addi r2, r2, 4      /* Increment the pointer to vector A */
     addi r3, r3, 4      /* Increment the pointer to vector B */
     subi r4, r4, 1      /* Decrement the counter */
     bgt r4, r0, LOOP       /* Loop again if not finished */
     stw r5, DOT_PRODUCT(r0)       /* Store the result in memory */
STOP: br STOP
N:
.word 6                 /* Specify the number of elements */
AVECTOR:
.word 5, 3, -6, 19, 8, 12        /* Specify the elements of vector A */
BVECTOR:
.word 2, 14, -3, 2, -5, 36       /* Specify the elements of vector B */
DOT_PRODUCT:
.skip 4
```

Важливе зауваження!!!! *Операція перемноження (mul) підтримується процесорним ядром NiosII тільки версій* ***standard*** *або* ***fast****.*

6. Контрольні запитання.

1. В чому полягають особливості процесорного ядра NiosII?
2. Яку функцію виконує модуль JTAG DEBUG?
3. Чим відрізняються ***Exception memory*** від ***Reset Vector memory***?
4. Як визначаються сигнали, що передаються за межі процесорної системи?

2. Лабораторна робота № 2.
Створення власних компонентів у Platform Designer Tool

1. Вступ

Platform Designer component — це електронний пристрій, який доступний як бібліотечний компонент для використання у Platform Designer Tool. Вони, зазвичай, складаються з двох частин: внутрішньої - яка містить опис принципу дії, та зовнішньої — яка містить опис інтерфейсу до шини Avalon.

Основні типи інтерфейсів шини Avalon [3]:

1) ***Avalon Clock Interface*** – розповсюдження тактових сигналів;

2) ***Avalon Reset Interface*** – розповсюдження сигналу скидання;

3) ***Avalon Memory-Mapped Interface (Avalon MM)*** – інтерфейс, що забезпечує виконання операцій зчитування-запису за адресами пристроїв. Такий інтерфейс є типовим для архітектури **master -slave**.

4) ***Avalon Streaming Interface (Avalon-ST)*** – інтерфейс, що використовує односпрямований потік даних (передавач – приймач);

5) ***Avalon Conduit Interface*** – об'єднує усі користувацькі інтерфейси, які не належать до інших типів. Можуть використовуватися для сполучень за межи системи, що розробляється у Platform Designer (з зовнішніми компонентами).

Будь-який компонент може підтримувати одразу кілька інтерфейсів різних типів, але обов'язково він повинен мати ***Clock*** і ***Reset*** інтерфейси.

У якості прикладу в даній роботі розглянемо створення дешифратора для семисегментного індикатора. Окрім обов'язкових сигналів (пов'язаних з формуванням символів на індикаторі), дешифратор буде мати ***Memory-Mapped*** та ***Conduit*** інтерфейси.

Нижче наведено код мовою Verilog, що описує 32-бітний регістр (**reg32**) для запису значення, яке має відображатися на індикаторі та модуль *Memory-Mapped* інтерфейсу (**reg32_avalon_interface**) для сполучення регістру з шиною Avalon. Тобто, наведений код описує 2 модуля: власне регістр та блок інтерфейсу регістра до шини Avalon. Особливістю шини Avalon є можливість вести запис до регістру кожного байту окремо, для цього використовується 4-бітний сигнал ***byteenable***.

```
module reg32 (clock, resetn, D, byteenable, Q);
        input clock, resetn;
        input [3:0] byteenable;
        input [31:0] D;
        output reg [31:0] Q;
        always @(posedge clock)
                if (!resetn)
                        Q <= 32'b0;
                else
        begin
                // Enable writing to each byte separately
                if (byteenable[0]) Q[7:0] <= D[7:0];
                if (byteenable[1]) Q[15:8] <= D[15:8];
                if (byteenable[2]) Q[23:16] <= D[23:16];
                if (byteenable[3]) Q[31:24] <= D[31:24];
        end
endmodule

module reg32_avalon_interface (clock, resetn, writedata,
                readdata, write, read,
                byteenable, chipselect, Q_export);
        // signals for connecting to the Avalon fabric
        input clock, resetn, read, write, chipselect;
        input [3:0] byteenable;
        input [31:0] writedata;
        output [31:0] readdata;
        // signal for exporting register contents
        // outside of the embedded system
```

```
    output [31:0] Q_export;
    wire [3:0] local_byteenable;
    wire [31:0] to_reg, from_reg;
    assign to_reg = writedata;
    assign local_byteenable = (chipselect & write) ?
                              byteeenable : 4'd0;
    reg32 U1 ( .clock(clock), .resetn(resetn), .D(to_reg),
          .byteenable(local_byteenable), .Q(from_reg) );
    assign readdata = from_reg;
    assign Q_export = from_reg;
endmodule
```

*Для детального розуміння процедури передачі даних рекомендуємо познайомитись з описом **Avalon Interconnect Specifications**.*

2. Порядок виконання роботи

1. Створіть новий проект у Quartus. Назвіть його ***component_tutorial***. Виконайте всі потрібні налаштування.

2. Додайте до проекту файли ***reg32.v*** та ***reg32_avalon_interface.v***, з кодом, наведеним вище.

3. Відкрийте ***Platform Designer Tool***.

4. Додайте компоненти ***Nios II (Classic) Processor*** та ***On-Chip Memory (RAM or ROM) Intel FPGA IP***. Сполучіть їх як зображено на рис. 27а та налаштуйте як у попередній лабораторній роботі. Виконайте автоматичне призначення адреси.

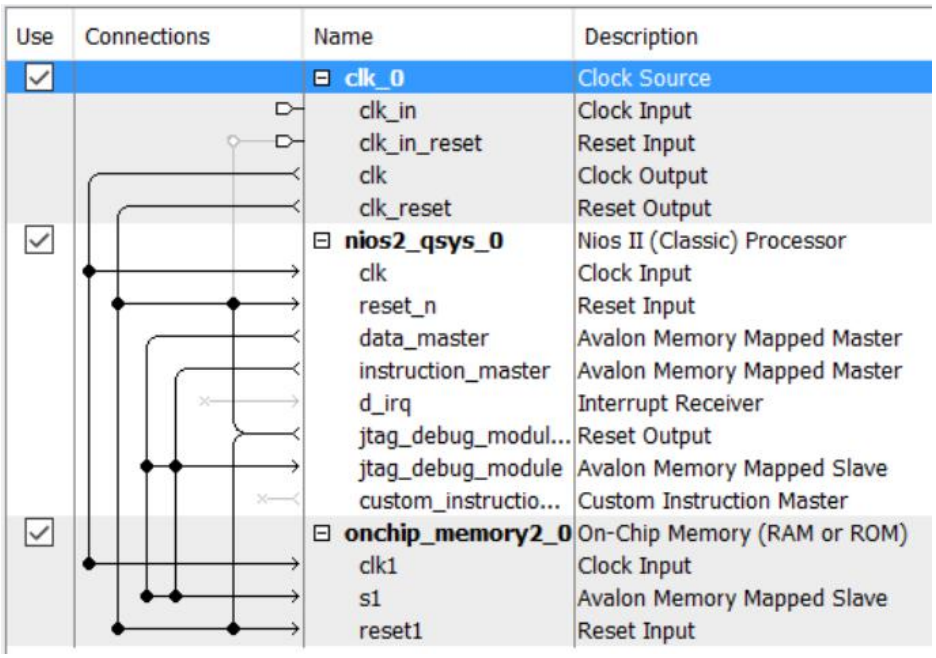

Рис. 27а. Сполучення компонентів

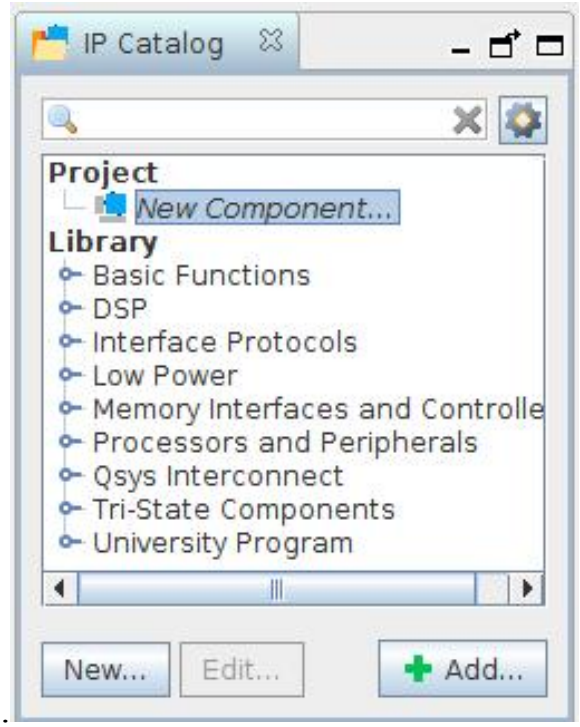

Рис. 27б. Вкладка IP Catalog

5. Наразі необхідно створити власний компонент, який буде додано до процесорного ядра Nios II. Натисніть ***New Component*** на вкладці *IP Catalog* (рис. 27б). Відкриється вікно ***Component Editor***, заповніть його у відповідності до рис. 28.

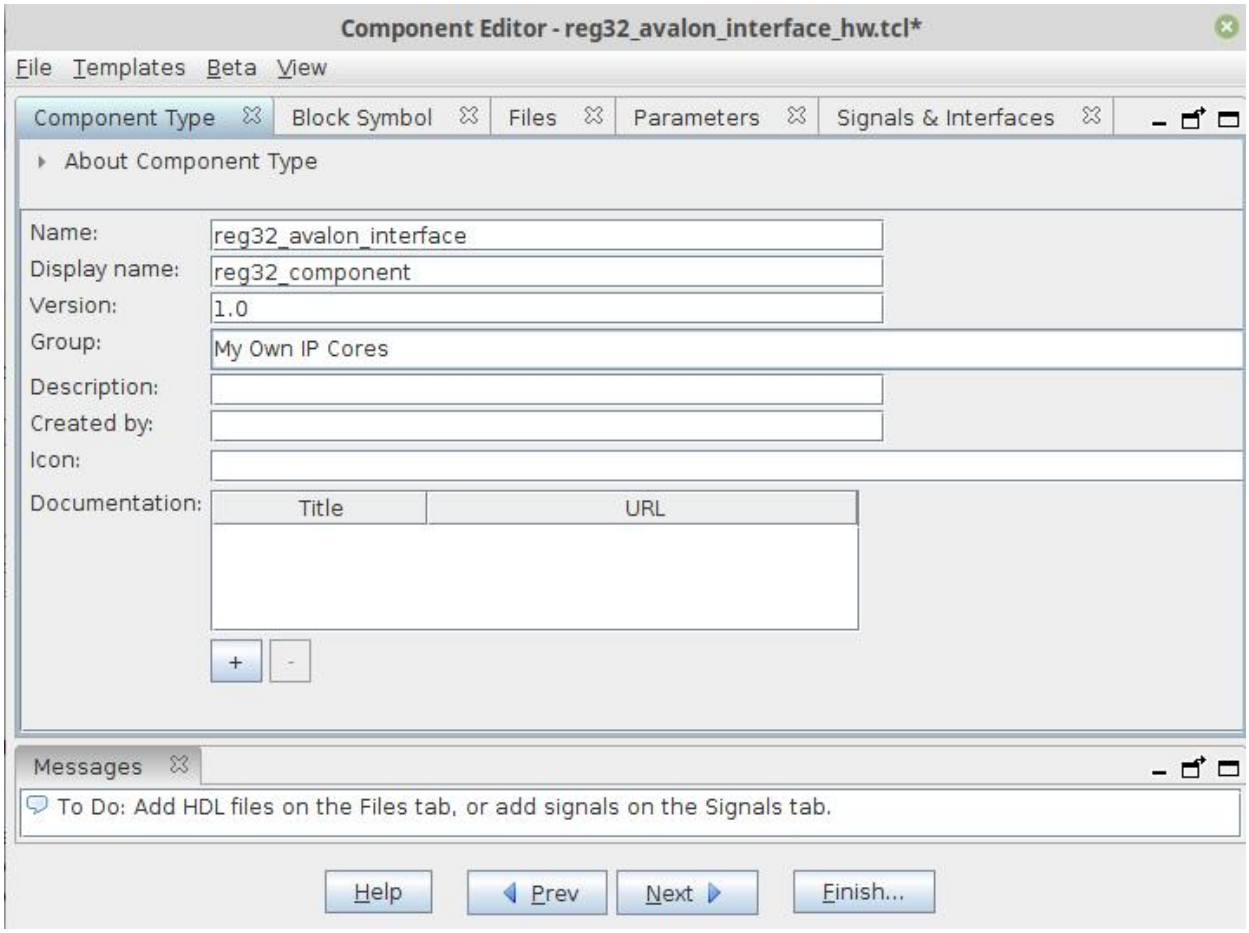

Рис. 28**.** Вікно Component Editor

6. Наступний крок – додати файли з описом компонента. Перейдіть до вкладки ***Files***. Додайте ***reg32_avalon_interface.v***. Натисніть кнопку ***Analyze Synthesis Files***. Цей аналіз потрібен для визначення відсутності помилок у обраному файлі.

7. Також, додайте файл ***reg32.v*** та проаналізуйте його. Якщо аналіз виявить помилки, виправте їх і проведіть аналіз повторно.

8. Натисніть кнопку ***Copy from Synthesis Files*** в розділі ***Verilog Simulation Files.*** Це вказівка компілятору обирати дані для моделювання з синтезованого файлу.

9. Далі необхідно узгодити інтерфейс власного компонента з інтерфейсом шини Avalon. Перейдіть на вкладку ***Signals & Interfaces***. Зліва, натисніть на полі ***<<add interface>>*** і оберіть з контекстного меню ***Clock Input***. Перетягніть сигнал ***clock[1]*** до новоствореного інтерфейсу та змініть тип сигналу (у правому вікні) на ***clk***. Результат зображено на рис. 29.

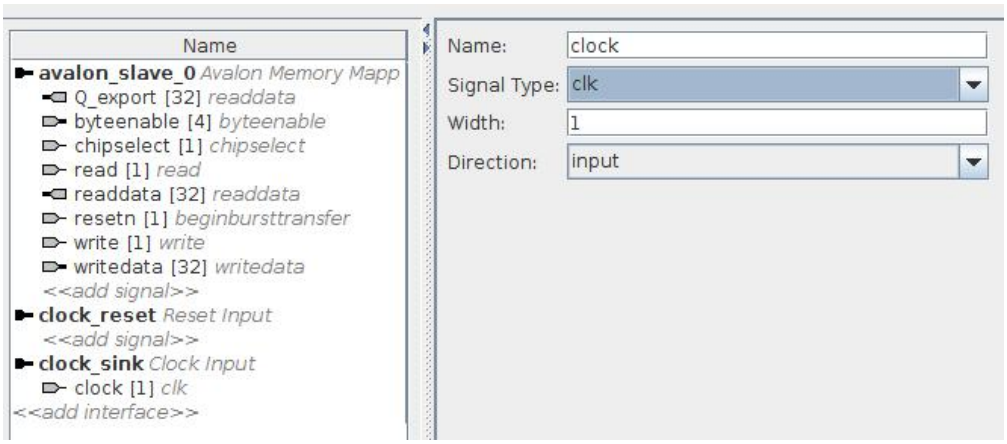

Рис. 29. Налаштування сигналу clock[1]

10. Виконайте відповідні налаштування для інших сигналів. Результат повинен відповідати рис. 30.

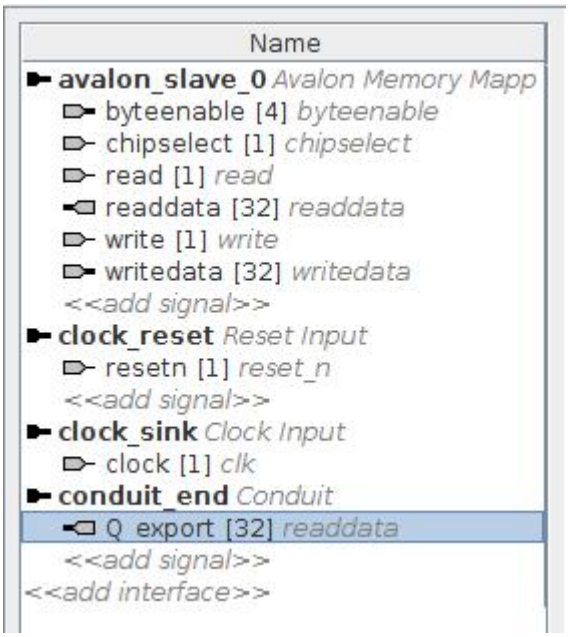

Рис. 30. Налаштування сигналів

11. Тепер перейдіть до *avalon_slave_0* (виділіть шину) та у правому вікні зробіть наступні налаштунки:

Associated Clock* = *clock_sink

Associated Reset* = *clock_reset

Read Wait* = *0

Інші налаштування залиште за замовчанням.

12. Виконайте наступні налаштування для шини ***clock_reset***:

Associated Clock* = *clock_sink

Натисніть ***Finish***.

13. Збережіть компонент, коли виникне відповідний запит.

14. Ви повернулися до головного вікна ***Platform Designer***. Додайте щойно створений компонент. При додаванні відкриється вікно, зображене на рис. 31. Натисніть ***Finish***.

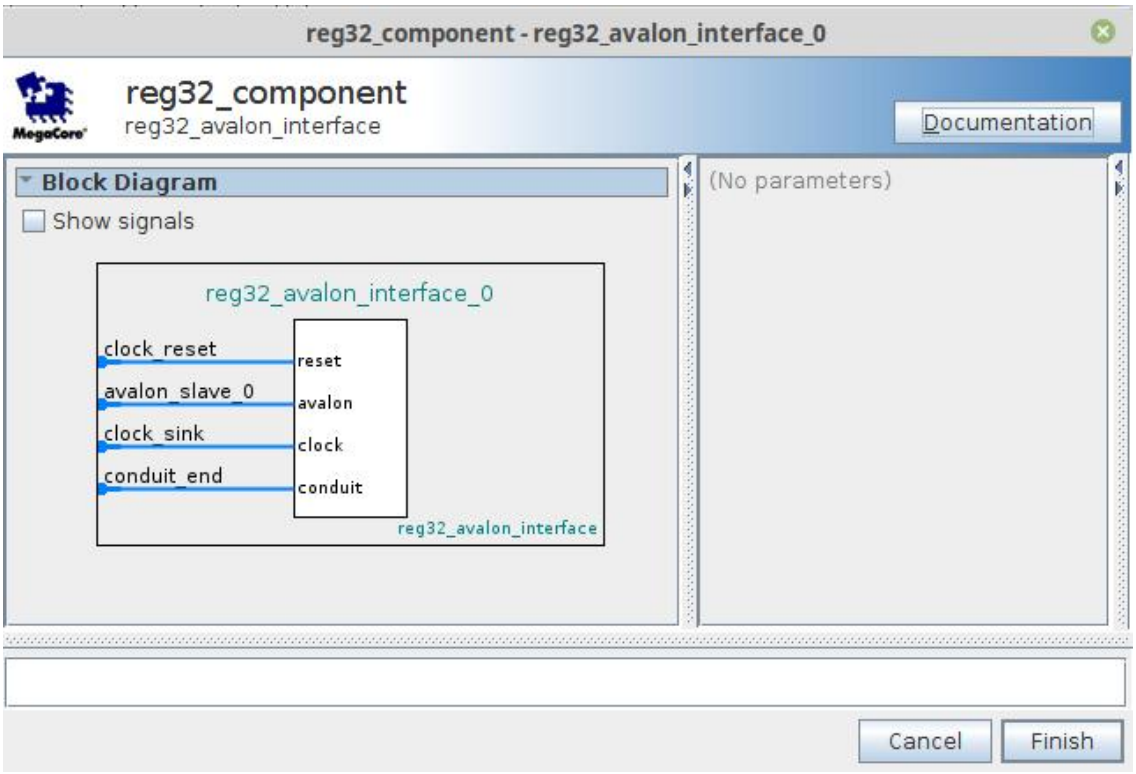

Рис. 31. Додання до проекту reg32_component

17. Підключіть власний компонент як зображено на рис. 32. Встановіть базову адресу ***reg32_avalon_interface_0*** рівною 0, іншим компонентам призначте її автоматично. Для ***reg32_avalon_interface_0*** → ***Conduit_end*** у графу ***export*** впишіть ***to_hex***.

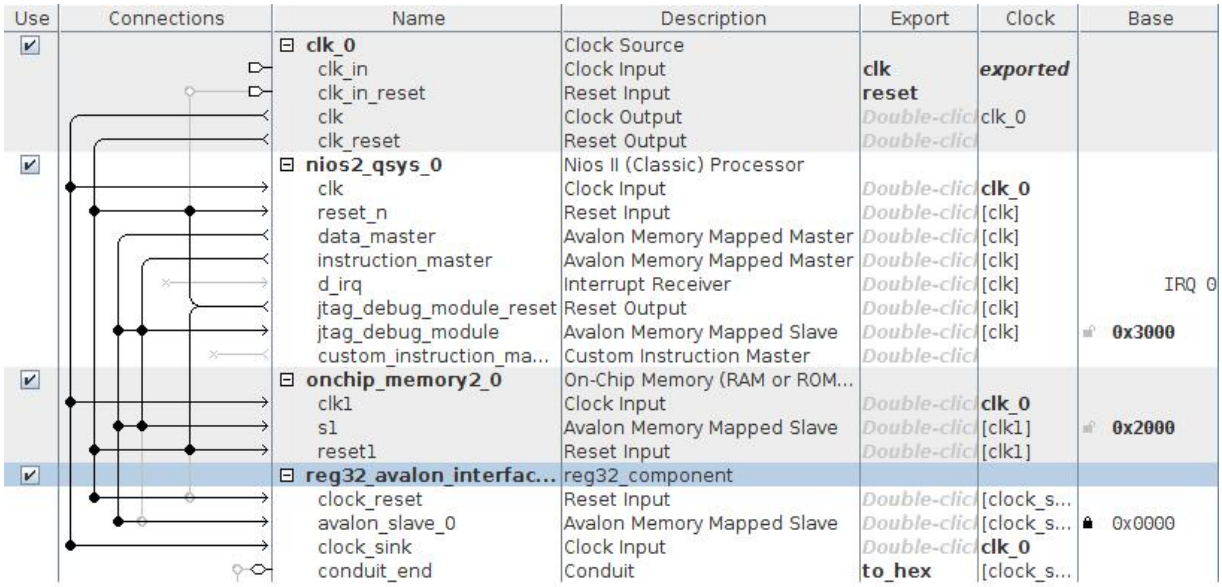

Рис. 32. Сполучення компонентів

18. Збережіть створену систему під назвою ***embedded_system***. Оберіть меню ***Generate → Show Instantiation Template***. Ви побачите вікно, показане на рис. 33.

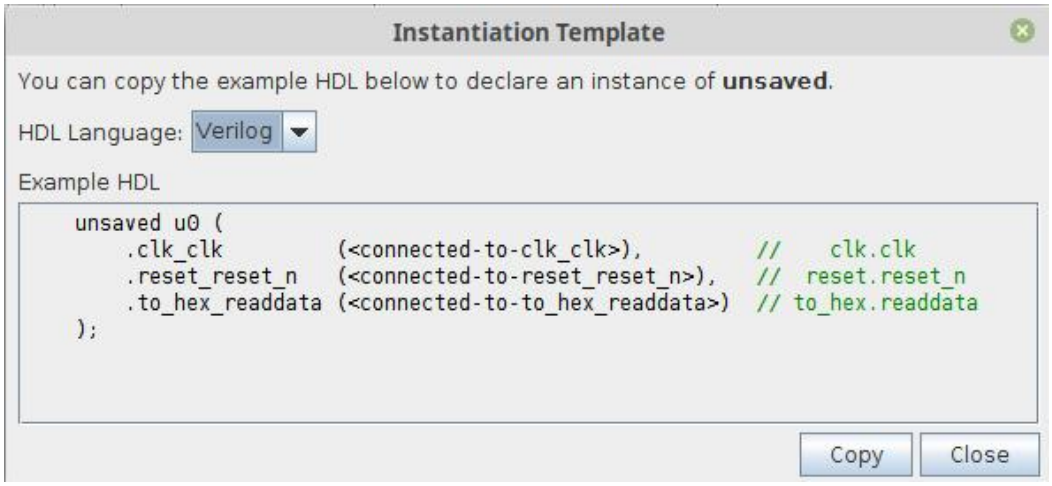

Рис. 33. Шаблон для використання системи

19. Згенеруйте систему і закрийте ***Platform Designer***.

20. У Quartus Prime додайте до проекту файл ***embedded_system/synthesis/embedded_system.qip*** (згенерована система).

21. Далі створіть 2 файли: **component_tutorial.v** та **hex7seg.v**. Помістіть до них наступний код та додайте їх до проекту:

```
module component_tutorial (CLOCK_50, KEY,
                                   HEX0, HEX1, HEX2, HEX3);
    input CLOCK_50;
    input [0:0] KEY;
    output [0:6] HEX0, HEX1, HEX2, HEX3;
    wire [15:0] to_HEX;

    embedded_system U0 (
        .clk_clk(CLOCK_50),
        .reset_reset_n(KEY[0]),
        .to_hex_readdata(to_HEX)
        );

    hex7seg h0(to_HEX[3:0], HEX0);
    hex7seg h1(to_HEX[7:4], HEX1);
    hex7seg h2(to_HEX[11:8], HEX2);
endmodule


module hex7seg (hex, display);
    input [3:0] hex;
    output [0:6] display;
    reg [0:6] display;
/*     - 0 -
 * 5 |       | 1
 *     - 6 -
 * 4 |       | 2
 *     - 3 -
```

```
*/
        always @ (hex)
                case (hex)
                        4'h0: display = 7'b0000001;
                        4'h1: display = 7'b1001111;
                        4'h2: display = 7'b0010010;
                        4'h3: display = 7'b0000110;
                        4'h4: display = 7'b1001100;
                        4'h5: display = 7'b0100100;
                        4'h6: display = 7'b0100000;
                        4'h7: display = 7'b0001111;
                        4'h8: display = 7'b0000000;
                        4'h9: display = 7'b0001100;
                        4'hA: display = 7'b0001000;
                        4'hb: display = 7'b1100000;
                        4'hC: display = 7'b0110001;
                        4'hd: display = 7'b1000010;
                        4'hE: display = 7'b0110000;
                        4'hF: display = 7'b0111000;
                endcase
endmodule
```

22. Виконайте призначення контактів (з файлу DE1-SoC.qsf). Запустіть компіляцію проекту.

23. Відкрийте ***Intel FPGA Monitor Program***. Створіть новий проект. Для 2-х перших вкладок вкажіть дані як у попередній лабораторній роботі, а на вкладці ***Specify a program type*** вкажіть ***No Program***.

24. Після створення проекту, оберіть меню ***Actions → Connect to System***.

25. Перейдіть до вкладки ***Memory Tab*** та натисніть ***Refresh***. Оберіть адресу ***0x0000***. Впишіть у цю комірку значення ***0x1234579d***. Спостерігайте результат на платі.

3. Самостійна робота

Завдання 1. Доопрацюйте проект таким чином, щоб були задіяні всі шість семи сегментних індикатори, які розташовані на платі DE1-SoC. Продемонструйте роботу пристрою.

Завдання 2. Доопрацюйте проект таким чином, щоби була можливість програмного керування відображенням значення на семи сегментних індикаторах. Значення повинно задавати за допомогою перемикачів на платі DE1-SoC. Продемонструйте роботу пристрою.

4. Контрольні запитання.

1. Які основні типи інтерфейсів шини Avalon?
2. Які головні сигнали використовуються в шині Avalon-MM?
3. Яке середовище використовується для налаштування власного компонента?
4. Як відбувається поєднання компонентів у середовищі ***Platform Designer***?

3. Лабораторна робота № 3.
Ядро ARM. Основи роботи у середовищі Intel FPGA Monitor Program

1. Вступ

Hardware Processing System — це вбудоване у ПЛІС ARM-ядро з власною периферією. Використовується тому, що працює воно набагато швидше, ніж ядра, синтезовані на FPGA.

Застосуйте Intel FPGA Monitor Program для налаштування програми, написаної для HPS. У даному описі розглянуто роботу Intel FPGA Monitor Program із платою Terasic DE1-SoC [4].

2. Порядок виконання роботи

Завантажте з офіційного сайту Intel FPGA доповнення ***University Program*** та встановіть його. Серед його компонентів знаходиться ***Intel FPGA Monitor Program***. Інструкція до встановлення та запуску цієї програми знаходиться всередині завантаженого архіву [5].

1. Запустіть ***Intel FPGA Monitor Program***.
2. Оберіть меню ***File → New Project***. Збережіть проект у директорії ***Monitor_Tutorial/tutorial1***. Оберіть архітектуру ***ARM Cortex-A9*** (рис. 34). Натисніть ***Next***.

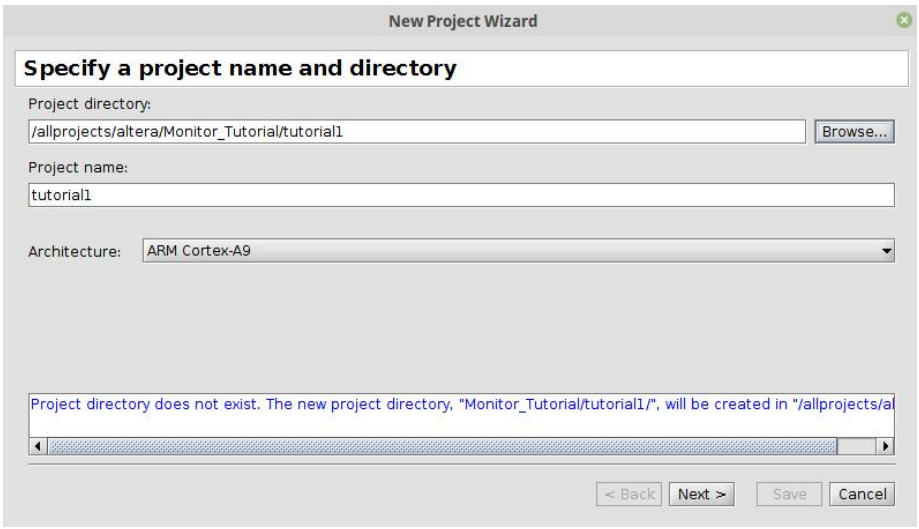

Рис. 34**.** Вкладка Specify a project name and directory

3. На вкладці ***Specify a system*** у минулих роботах вказували власноруч створену систему. Цього разу укажіть ***DE1-SoC Computer.*** Також доступні схожі комп’ютери для інших плат серії DE, <Custom System> —

уже знайома вам функція створення власної системи, та ARM Cortex A9 System — для використання буде доступною лише *HPS* частина чипу.

4. Необхідні ***SOPCInfo-*** та ***SOF***-файли підтягнуться автоматично. Також автоматично обереться ***Preloader = DE1-SoC***. У минулих роботах вказували ***Preloader = not Required*** (це було для *Nios II)*, але для систем, що містять *HPS*, він є обов'язковим.

Результат можна побачити на рис. 35.

5. Натисніть ***Documentation*** справа від обраного ***DE1-SoC Computer***. Завантажиться документ ***DE1-SoC_Computer_ARM.pdf***, у якому описана система, з якою будемо зараз працювати.

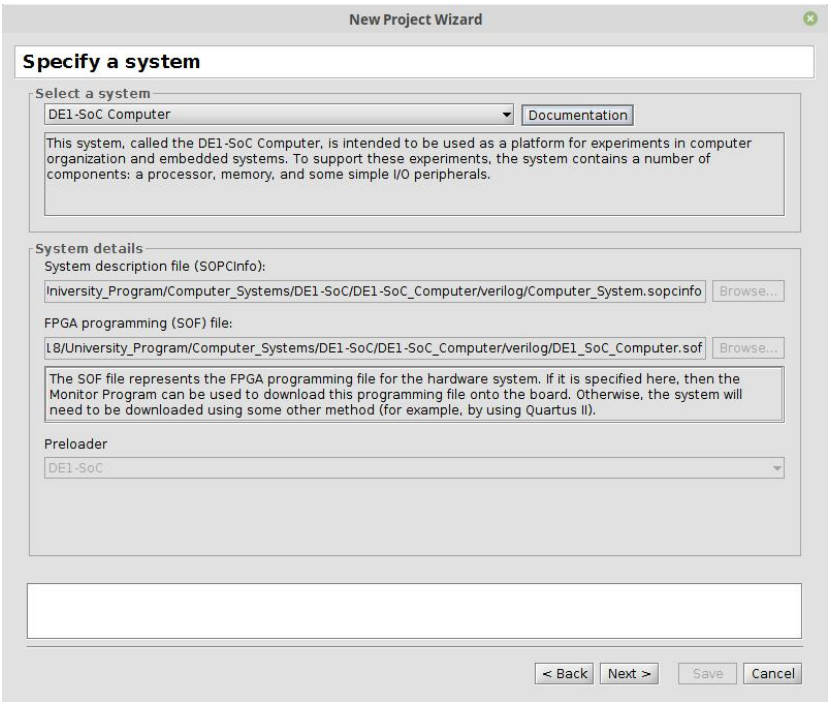

Рис. 35. Вкладка Specify a system

6. Ознайомтеся з системою. Погляньте на зображення у документі ***figure 1*** — блок-схему системи (рис. 36). Ознайомтесь з описом ***DE1-SoC Комп'ютера***.

Після ознайомлення з системою закрийте опис та перейдіть до виконання наступних кроків.

7. Ви зупинилися на вкладці ***Specify a system***. Натисніть ***Next***.

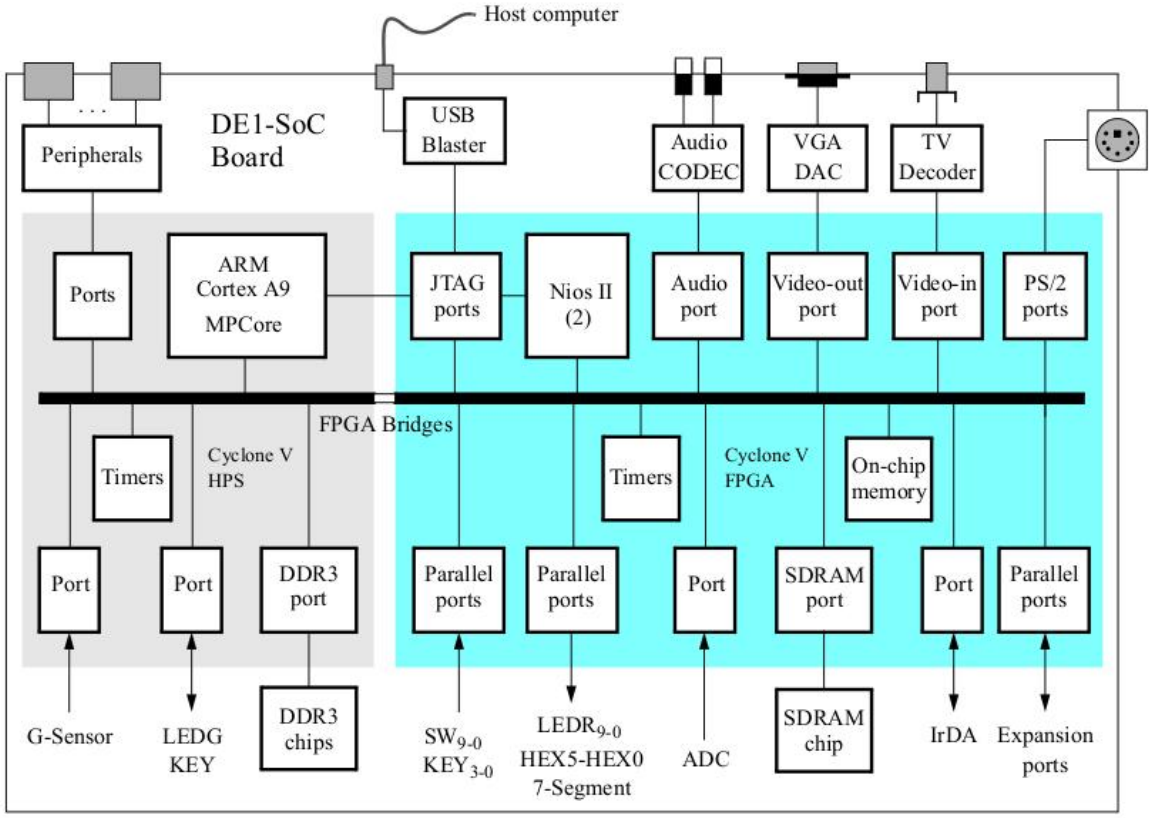

Рис. 36. Блок-схема DE1-SoC Computer

8. На вкладці ***Specify a program type*** можливо обрати одну з наступних опцій:

Assembly Program — програма написана мовою Асемблер;

C Program — програма написана мовою С;

AXF, ELF or SREC File — використання вже скомпільованої програми;

No Program — під'єднання до системи без завантаження програми, використовується для безпосередньої роботи з периферією.

9. Цього разу оберіть ***Assembly Program***. Також поставте галочку у графі ***Include a sample program with the project***, вона доступна тільки для ***DE-X Computer*** та не доступна для власноруч розроблених систем. Оберіть приклад програми під назвою ***simple_program***. Вона переносить стан перемикачів на світлодіоди. рис. 37 показує, як має виглядати вікно. Натисніть ***Next***.

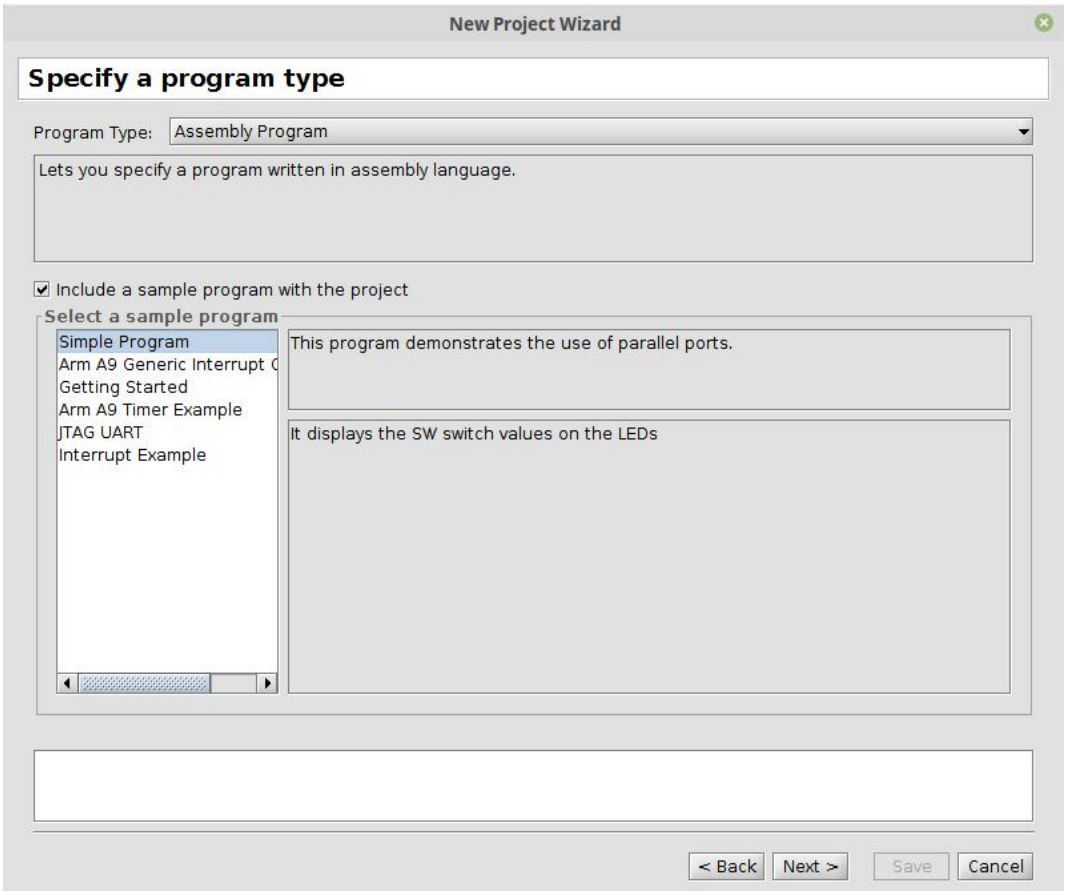

Рис. 37. Вкладка Specify a program type

10. На вкладці ***Specify program details*** показані вихідні файли, які будуть використані у проекті. Вони автоматично будуть додані у директорію вашого проекту. Також, можливо додати власні файли, для цього треба натиснути кнопку ***Add*** та обрати ці файли. Зараз цього робити не потрібно.

Крім того, на цій вкладці вказано, що перша команда програми починається з мітки ***_start*** (рис. 38). Натисніть ***Next.***

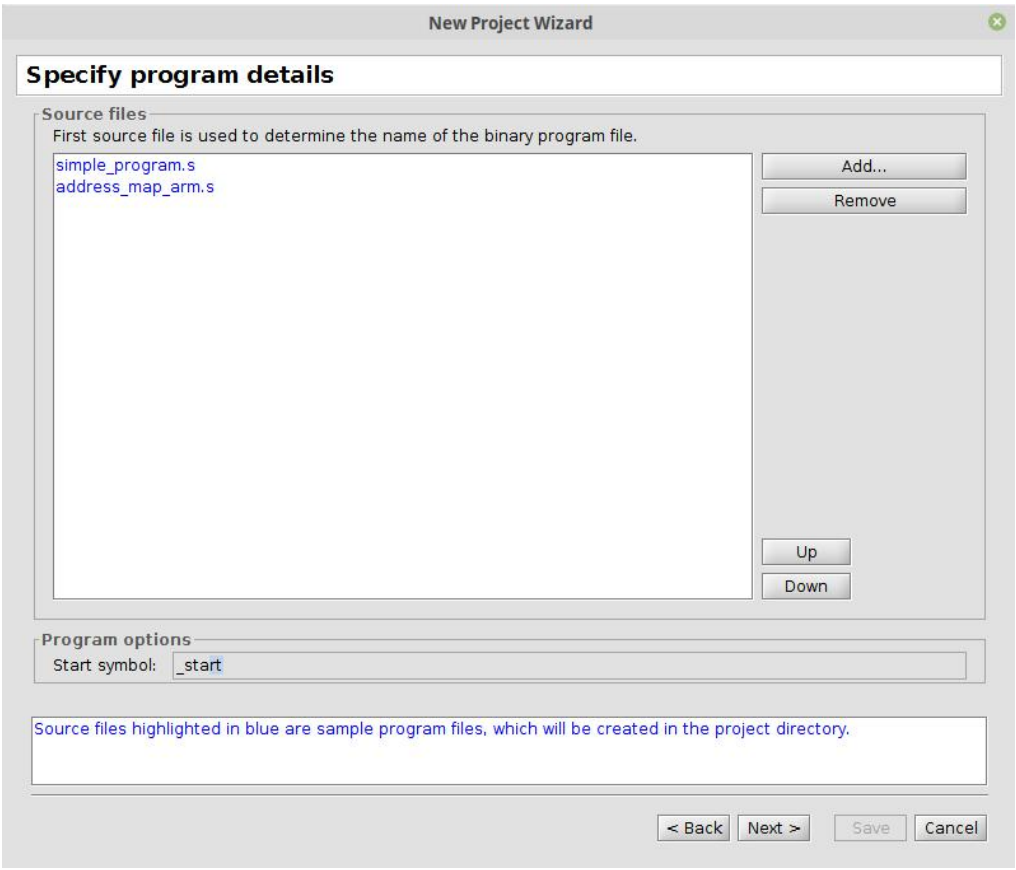

Рис. 38. Вкладка Specify program details

11. Вкладку ***Specify System parameters*** налаштуйте як на рис. 39.

Host Connection містить доступні для під'єднання з вашого комп'ютера плати. Оберіть вашу плату, вона має бути єдиною у списку. Якщо список порожній, підключіть DE1-SoC до комп'ютера та натисніть ***Refresh***.

Processor — список процесорів, що доступні поточному проекту. Оберіть ***ARM_A9_HPS_arm_a9_0.***

Terminal Device – вибір пристрою, на який будуть передаватися сповіщення процесором. Оберіть ***JTAG_UART_for_ARM_0***.

Натисніть ***Next***.

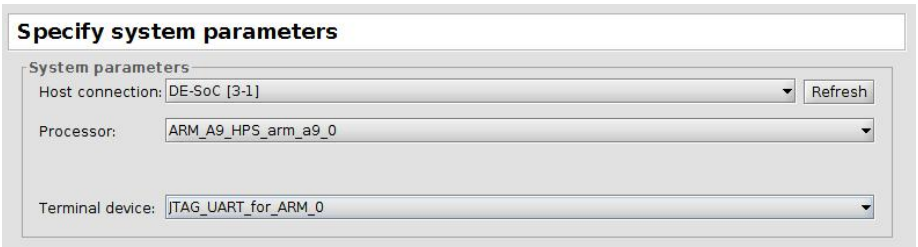

Рис. 39. Вкладка Specify System parameters

12. На вкладці ***Specify program memory settings*** (рис. 40) має за замовчанням бути обрано ***Basic*** у графі ***Linker Section Presets***. Ця вкладка показує розподілення пам'яті. Доступні варіанти:

Basic mode розміщує ***.text*** — область програми — починаючи з адреси 0x0000.

Interrupt / Exceptions mode розміщує ***.text*** починаючи з адреси 0x0040, а адресний простір 0x0000 — 0x003F зайнято ***.vectors*** — таблицею векторів переривання.

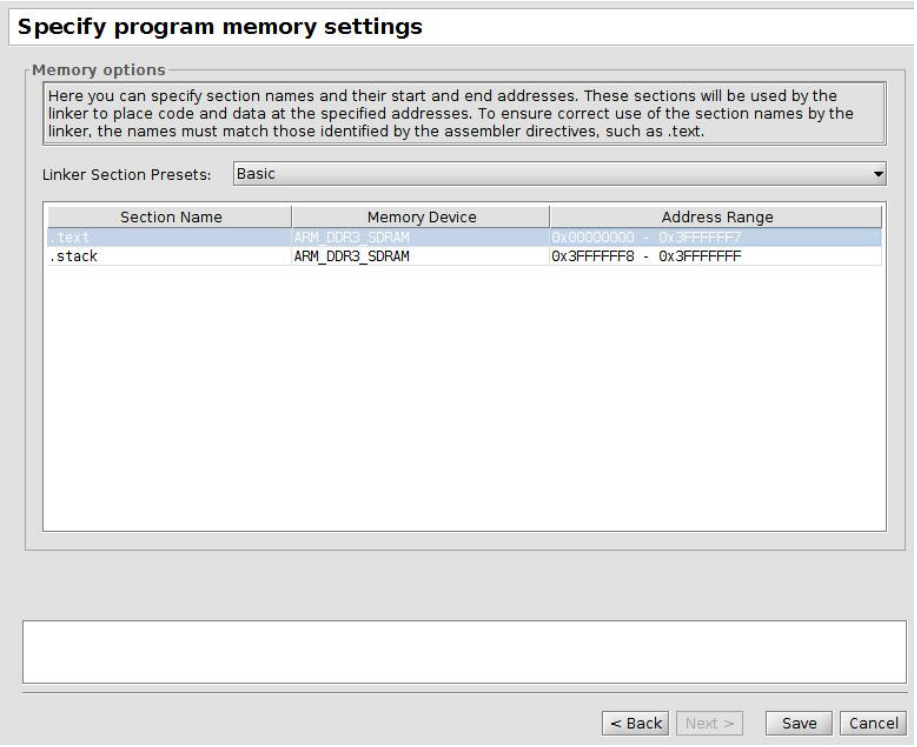

Рис. 40. Вкладка Specify program memory settings

Загалом, ви можете змінити розподіл пам'яті за власним розсудом. Можна додавати власні розділи та змінювати розташування вже існуючих. Зверніть увагу - розділи можуть знаходитись на різних пристроях, як в середині мікросхеми, так і за її межами.

13. Натисніть ***Save / Finish*** для завершення створення нового проекту. З'явиться запитання, чи хочете ви завантажити вашу hardware-систему (тобто ***DE1-SoC Computer***) до ПЛІС зараз (рис. 41). Погодьтесь.

Ви можете завантажити систему у будь-який момент за допомогою меню ***Actions → Download System***.

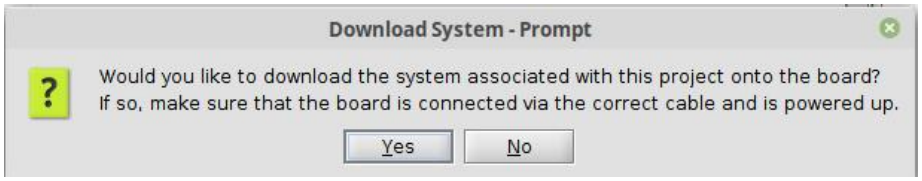

Рис. 41. Завантаження системи

14. Після успішного завантаження системи відкриється вікно ***Intel FPGA Monitor Program – tutorial1***, як зображено на рис. 42.

Якщо ви хочете ще раз переглянути документацію на DE1-SoC Computer, натисніть ***View/edit the system settings*** (кнопка з блакитною піктограмою) , оберіть вкладку ***System Settings*** та натисніть кнопку ***Documentation***.

15. Відкрийте директорію ***Monitor_Tutorial/tutorial1*** (директорію проекту) та дослідіть її вміст. Вона містить 4 файли:

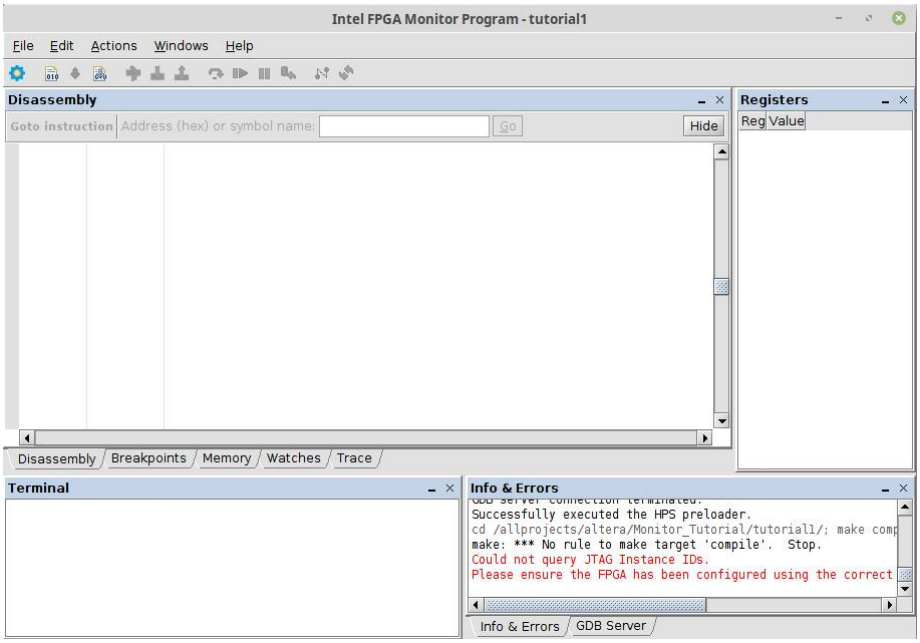

Рис. 42. Вікно Intel FPGA Monitor Program – tutorial1

address_map_arm.s — файл мовою Асемблер, що містить опис простору пам’яті: за якими адресами знаходяться динамічна пам’ять, регістри доступу до світлодіодів, таймерів, кнопок, тощо. Не містить команд для виконання, підключається до проекту як файл-заголовок;

simple_program.s — написаний мовою Асемблер вхідний файл;

makefile — це скрипт для компіляції та компонування проекту, на вході отримує header- та source-файли, на виході генерується бінарний файл, який можна завантажити до пристрою (для крос-компіляції, як у даному випадку),

або виконати (у випадку, коли програма призначена для виконання на вашому ж ПК);

tutorial1.amp — це файл з описом проекту, що містить налаштування, які були внесені під час створення проекту.

16. Поверніться до Monitor Program. Скомпілюйте проект. Для цього оберіть меню ***Actions → Compile***. Спостерігайте за повідомленнями у розділі ***Info & Errors***.

Спробуйте виправити усі помилки самостійно.

Якщо у вас виникли такі помилки:

- make: *** No rule to make target 'clean'. Stop.

- make: *** No rule to make target 'compile'. Stop.

тоді відкрийте ***makefile,*** знайдіть наступні строки:

COMPILE: simple_program.srec

CLEAN:

та замініть слова ***COMPILE*** та ***CLEAN*** великими літерами на ***compile*** та ***clean*** маленькими літерами.

17. Завантажте програму на DE1-SoC, для цього оберіть меню ***Actions → Load***. Також ви можете використовувати функцію ***Compile & Load*** для компіляції та завантаження одразу.

Відкриється вікно, зображене на рис. 43.

У вікні справа відображається перелік регістрів та їх поточні значення, зліва — вікно дизасемблера, яке містить машинні коди та строки програми, відповідно. Жовтим кольором виділена строчка коду, на якій зупинився процесор.

18. Для запуску та налагодження програми існують наступні можливості у меню *Actions*:

Single Step

Step Over Subroutine

Continue

Stop

Restart

Reset System

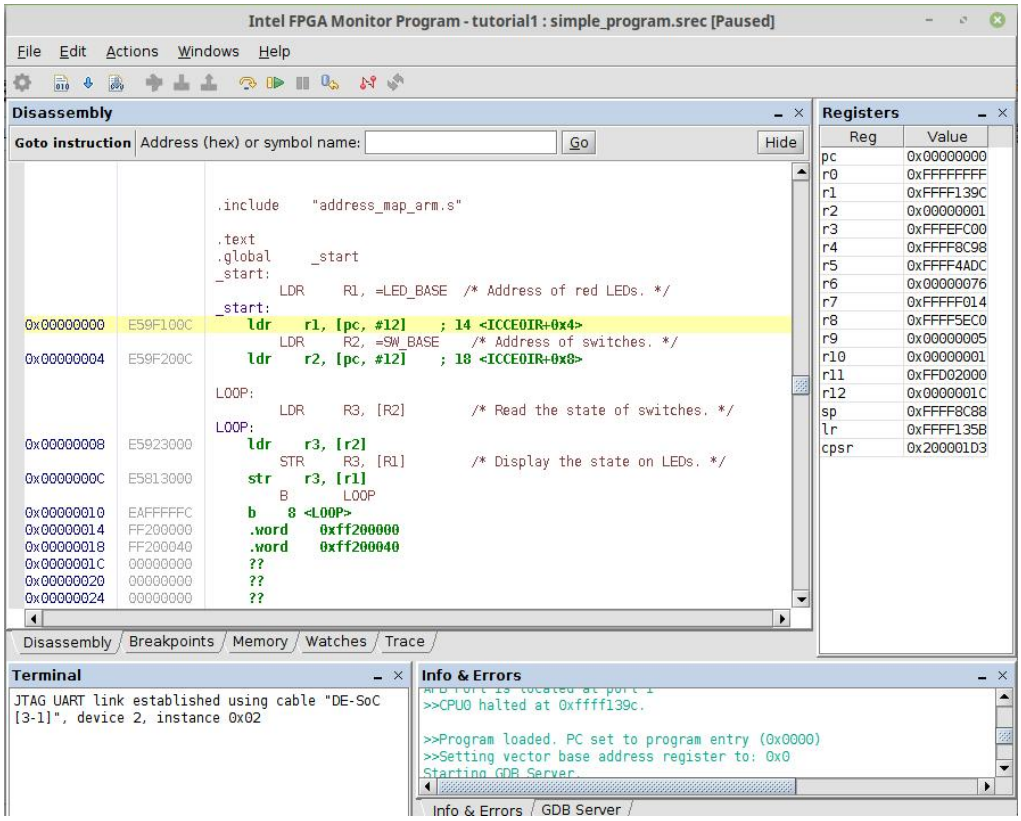

Рис. 43**.** Вікно Intel FPGA Monitor Program після завантаження програми

Також ці команди винесені на панель інструментів. Призначення цих команд є очевидним.

19. Дослідіть роботу програми, виконуючи її покроково та запустивши її командою ***Continue***. Поясніть роботу програми.

3. Самостійна робота

Завдання 1. Згенеруйте та виконайте перевірку проекту для прикладу **Getting Started**. Поясніть, як працює програма.

Завдання 2. Згенеруйте та виконайте перевірку проекту для прикладу **Interrupt Example**. Поясніть, як працює програма.

Завдання 3. Згенеруйте та виконайте перевірку проекту для прикладу **ARM A9 Timer Example**. Поясніть, як працює програма.

4. Контрольні запитання.

1. Що таке ***Hardware Processing System***? Чим вона відрізняється від процессорного ядра NiosII?
2. Які компоненти входять до складу ***DE1-SoC Computer***?
3. Які типи програмного забезпечення підтримуються процессорною системою?
4. Які варіанти розподілу пам’яті доступні процесорній системі?

4. Лабораторна робота № 4. Налаштування ядра ARM у середовищі Intel FPGA Monitor Program

1. Вступ

Мета цієї лабораторної роботи – навчитися самостійно налаштовувати процесорне ядро ARM. Познайомитись з архітектурою ядра можна за наступним посиланням: https://ftp.intel.com/Public/Pub/fpgaup/pub/Teaching_Materials/current/Tutorials/ARM_A9_intro_intelfpga.pdf .

Крім того, до процесорного ядра будуть додані компоненти, що синтезуються на логічних комірках FPGA, та поєднуються з ARM за допомогою спеціальних каналів зв'язку (LWH2F – Lightweight HPS-to-FPGA bridge).

На завершення, для синтезованої системи на кристалі, буде розроблено та перевірено відповідне програмне забезпечення.

2. Порядок виконання роботи

1. Створіть у середовищі Quartus Prime новий проект. Для цього:
- встановіть робочу директорію проекту;
- вкажіть назву проекту – ***comp_arm;***
- оберіть тип мікросхеми - **5CSEMA5F31C6.**
2. Додайте до проекту файли ***comp_arm.v*** та ***hex7seg.v***. Визначте файл ***comp_arm.v*** файлом верхнього рівня ієрархії.
3. Відкрийте середовище QSys.
4. Перед початком створення нової системи видаліть компонент ***Clock Source***.
5. У вікні **IP Catalog** оберіть з папки **University Program → Clock** модуль ***System and SDRAM Clocks for DE-series Boards*** та додайте його до проекту. Цей модуль дозволяє відразу зробити необхідні підключення блоку PLL, що використовується для формування тактових сигналів на платі DE1-SoC.
6. Зробіть для нього налаштування у відповідності до рис. 44

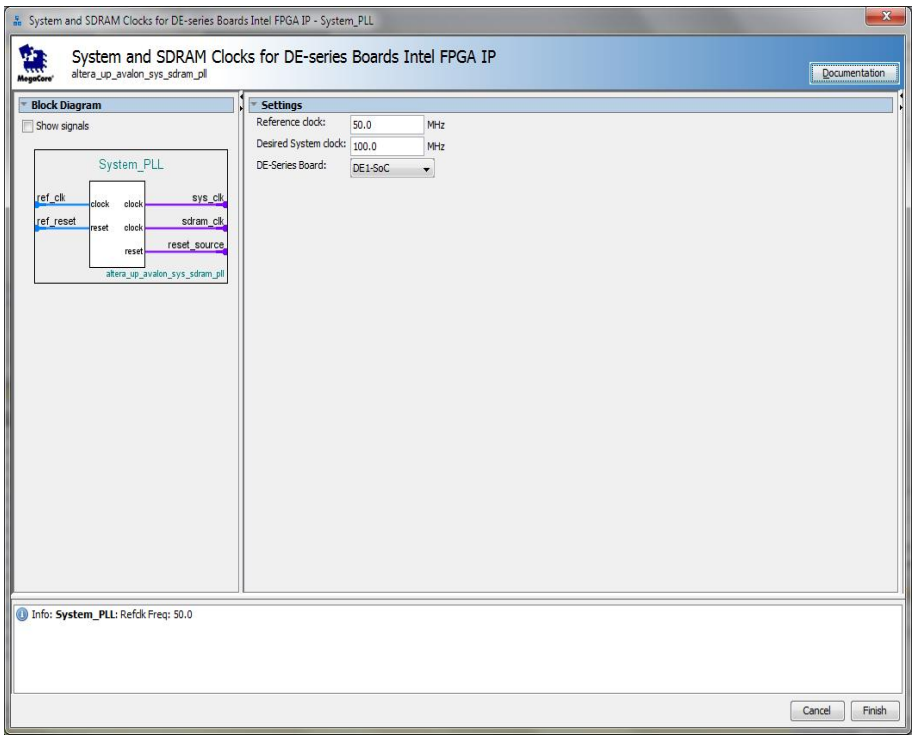

Рис. 44. Налаштування блоку формування тактових сигналів

7. Додайте до системи процесорне ядро. Для цього, у вкладці ***IP Catalog*** (вкладка знаходиться зліва у вікні ***Platform Designer Tool***) оберіть ***Library → Processors and Peripherals → Hard Processor Systems → ArriaV/CycloneV Hard Processor System*** та натисніть кнопку ***Add.*** Відкриється вікно зображене на рис. 45.

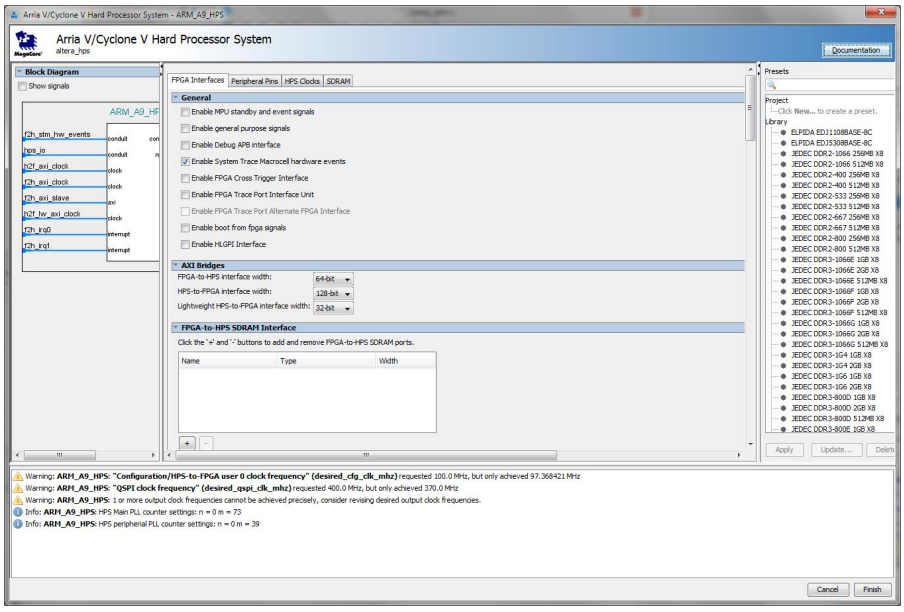

Рис. 45. Налаштування процесорного ядра ARM

8. На сторінці ***FPGA Interfaces*** виконайте налаштування у відповідності до рис. 45 – 47.

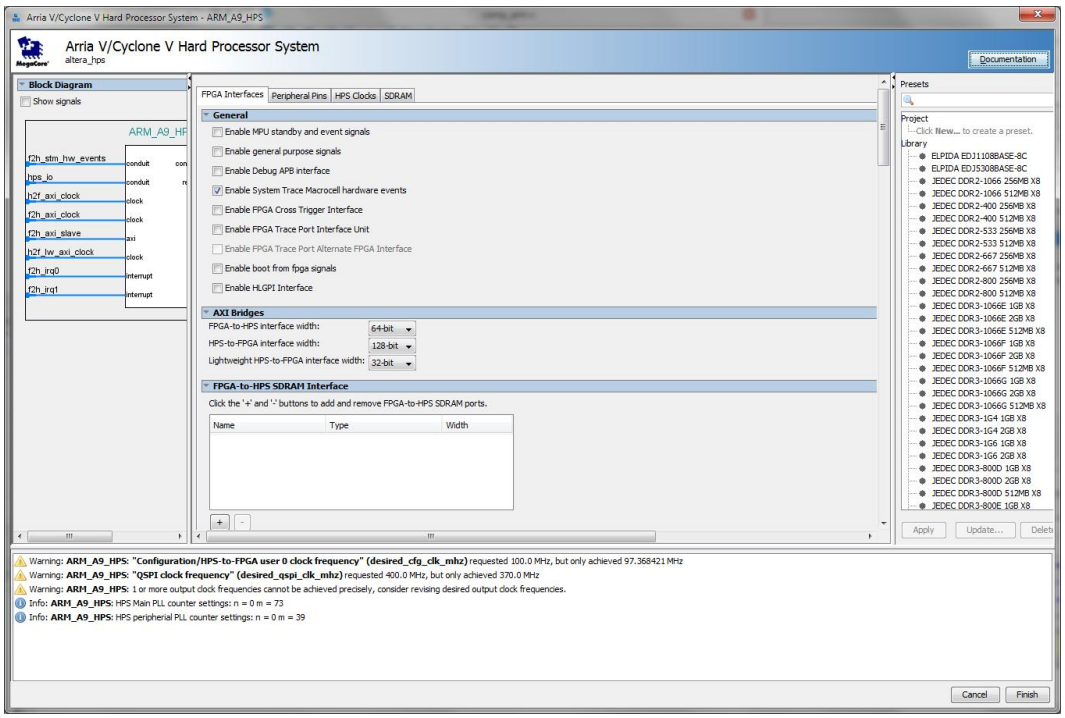

Рис. 46. Налаштування процесорного ядра ARM

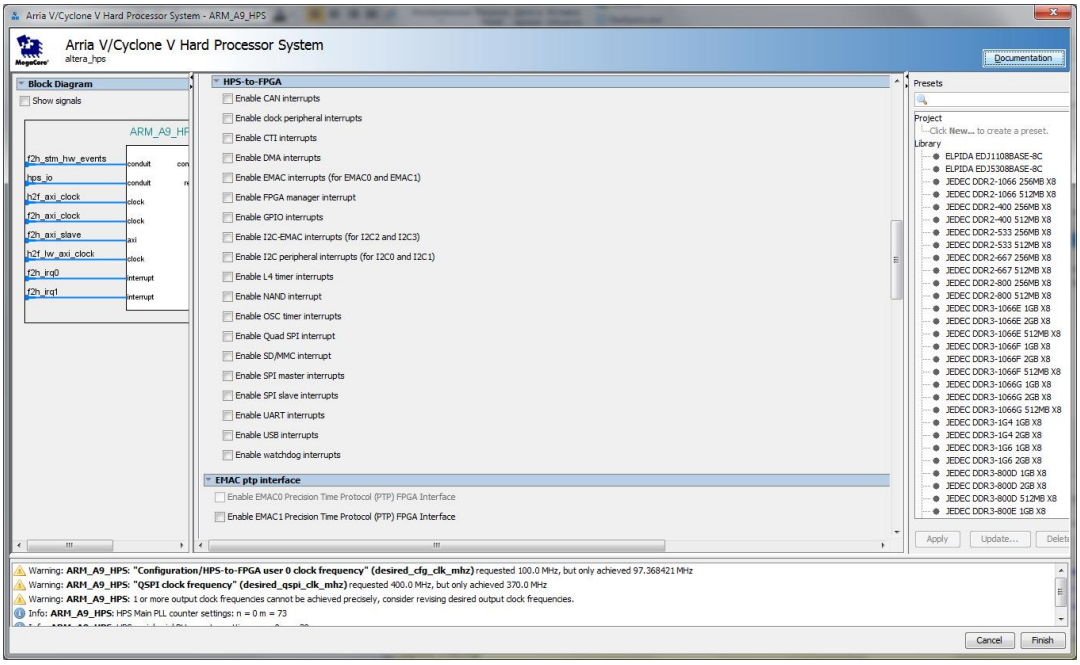

Рис. 47. Налаштування процесорного ядра ARM

9. Перейдіть до сторінки ***Peripheral Pins*** та виконайте налаштування периферійних компонентів процесорного ядра у відповідності до рис. 48 – 50.

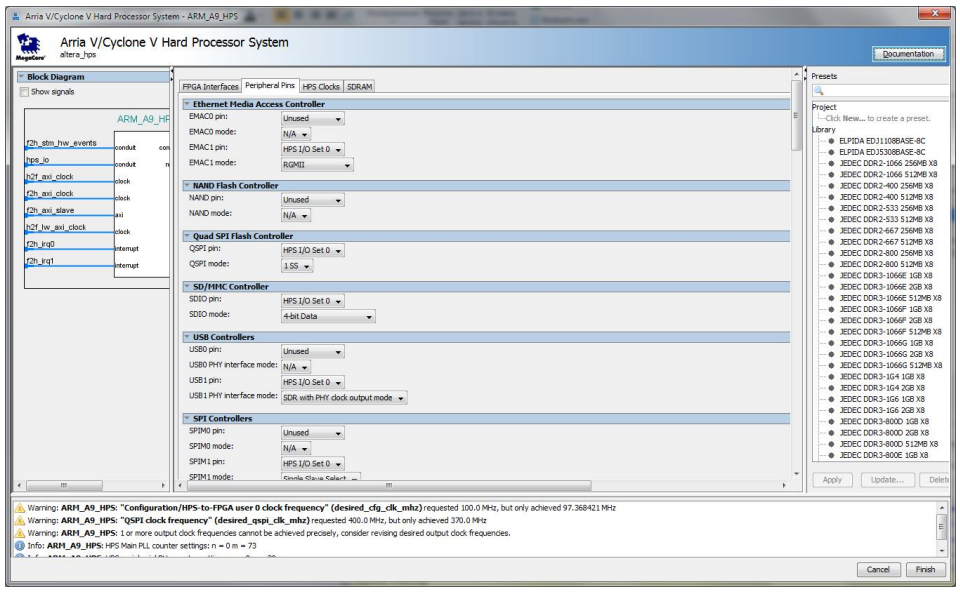

Рис. 48. Налаштування периферійних пристроїв

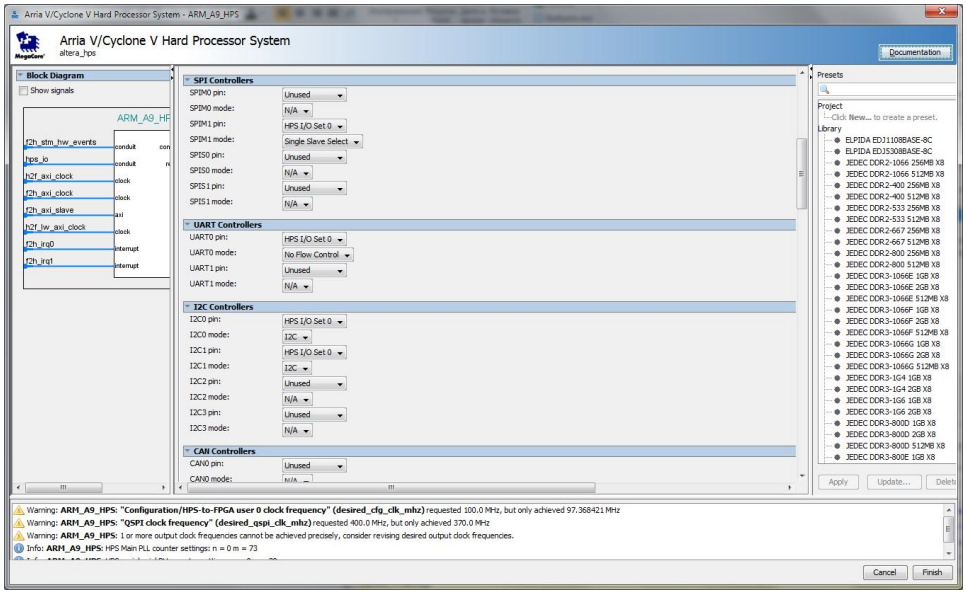

Рис. 49. Налаштування периферійних пристроїв

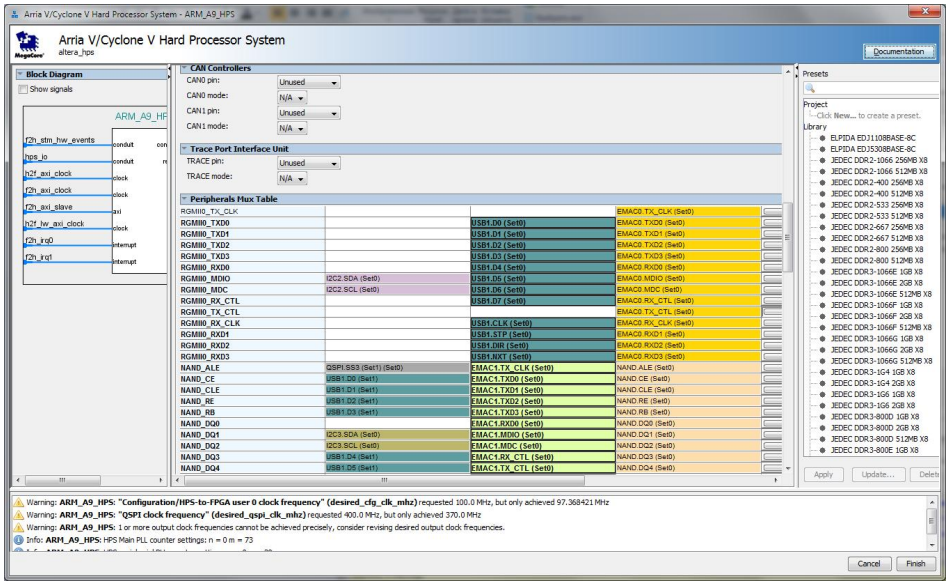

Рис. 50. Налаштування периферійних пристроїв

Крім того вкажіть, що процесором будуть використовуватись контакти загального призначення ***GPIO09, GPIO35, GPIO40, GPIO41, GPIO48, GPIO53, GPIO54, GPIO61***. За їх допомогою відбувається взаємодія з зовнішніми периферійними пристроями.

10. Перейдіть до сторінки ***HPS Clocks*** та виконайте налаштування синхронізуючих сигналів процесорного ядра у відповідності до рис. 51 – 53.

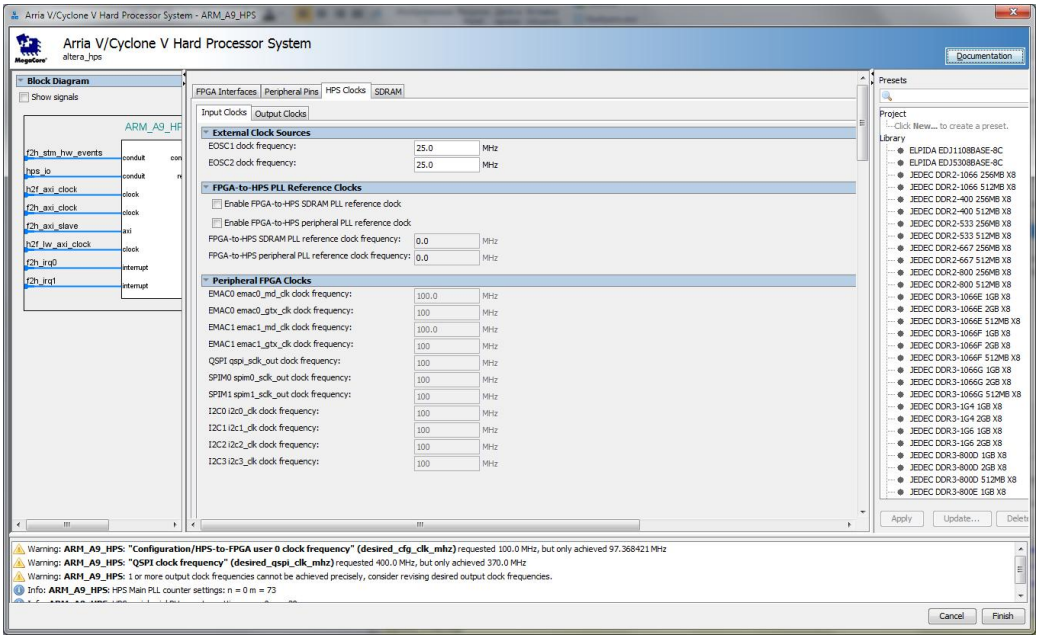

Рис. 51. Налаштування синхронізуючих сигналів

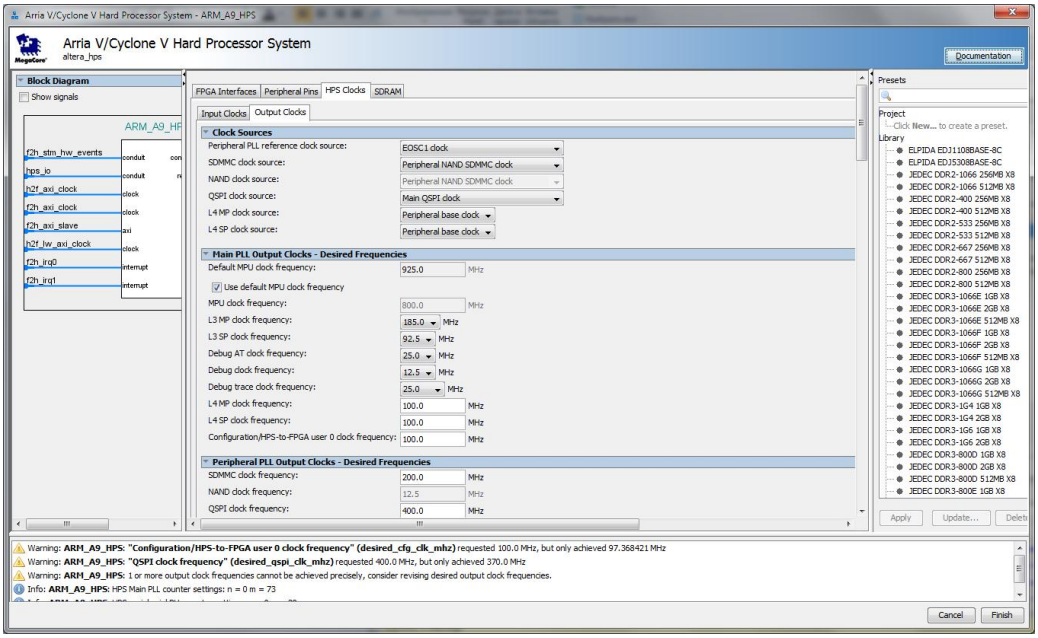

Рис. 52. Налаштування синхронізуючих сигналів

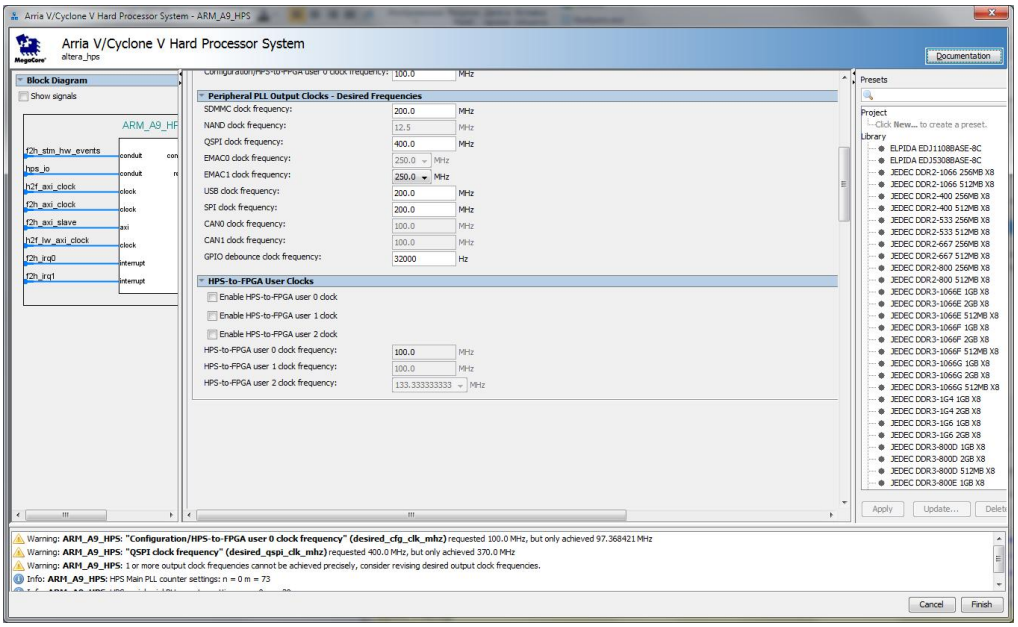

Рис. 53. Налаштування синхронізуючих сигналів

11. Перейдіть до сторінки ***SDRAM*** та виконайте налаштування інтерфейсу зовнішньої оперативної пам’яті у відповідності до рис. 54– 58.

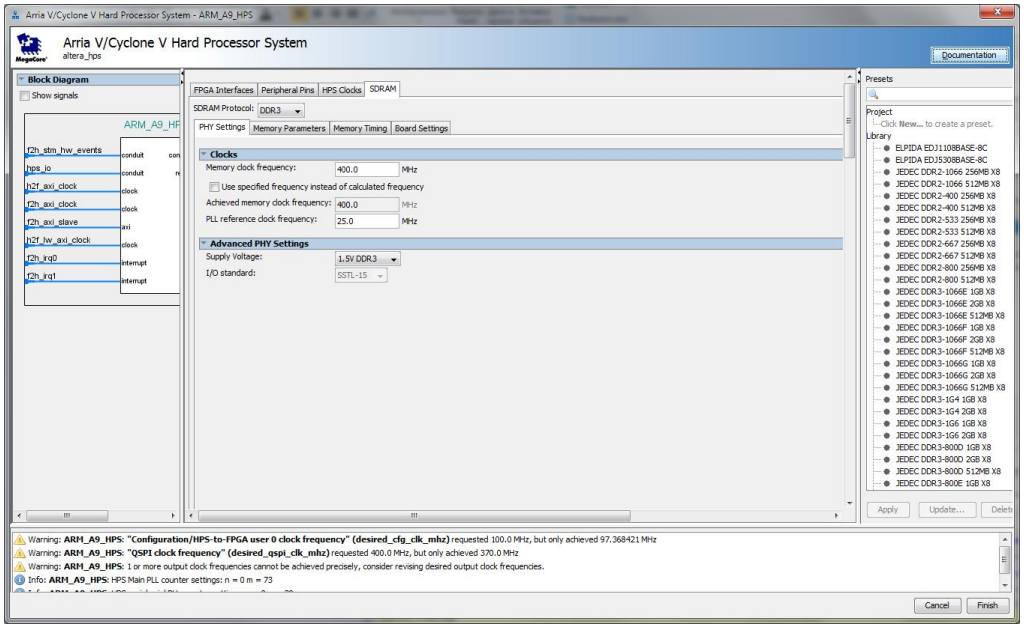

Рис. 54. Налаштування інтерфейсу SDRAM

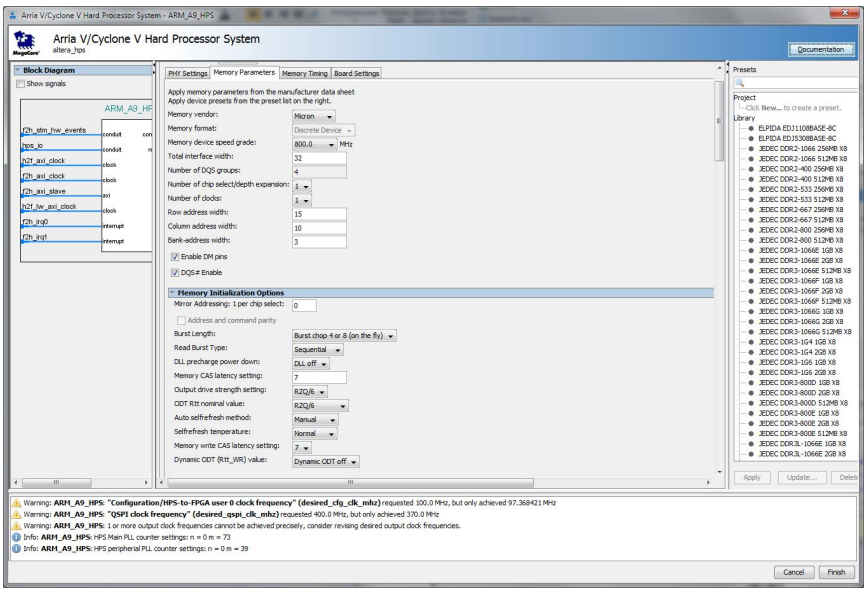

Рис. 55. Налаштування інтерфейсу SDRAM

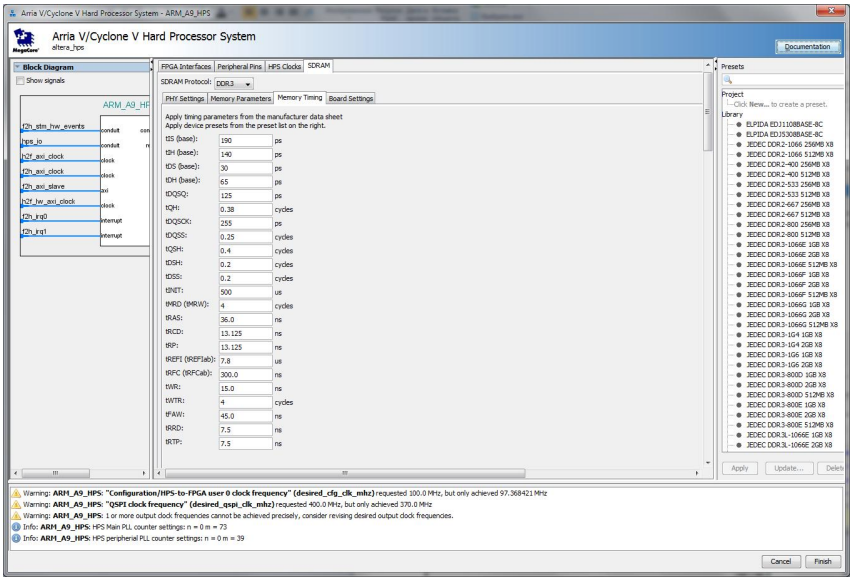

Рис. 56. Налаштування інтерфейсу SDRAM

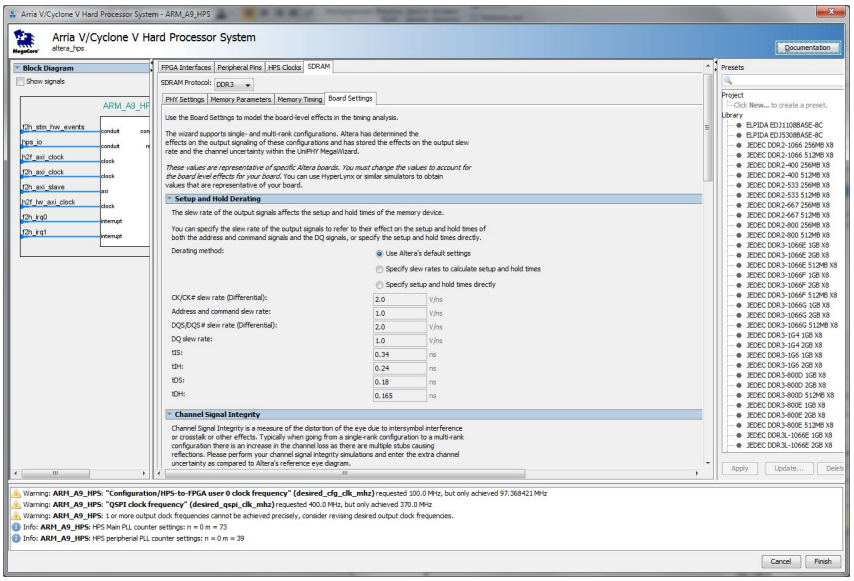

Рис. 57. Налаштування інтерфейсу SDRAM

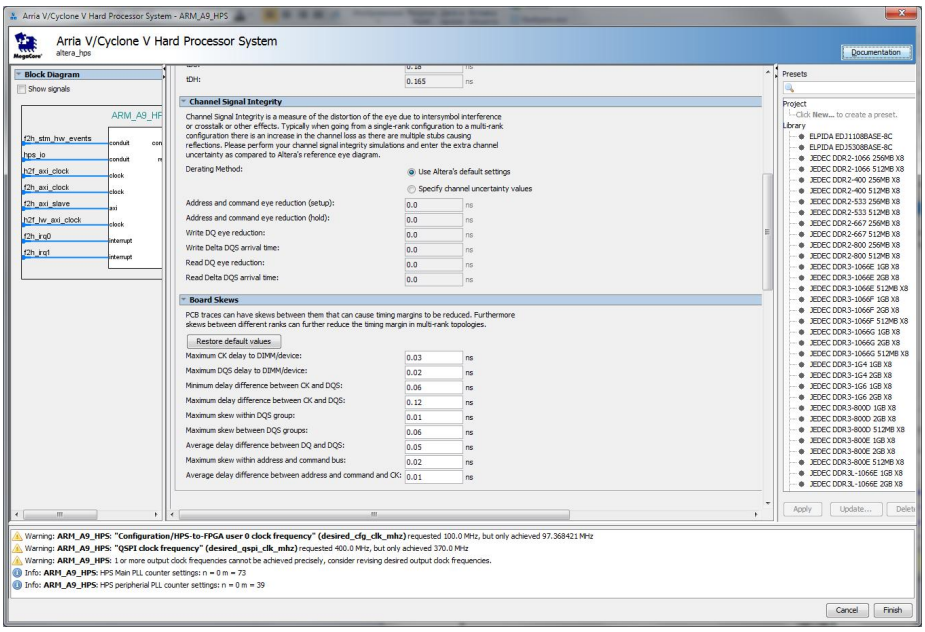

Рис. 58. Налаштування інтерфейсу SDRAM

12. Натисніть ***Finish***.
13. Додайте до системи модуль ***JTAG to Avalon Master Bridge.*** Він знаходиться у папці ***Basic Functions -> Bridges and Adaptors -> Memory Mapped.*** Залиште його налаштувння за замовченням (рис. 59).

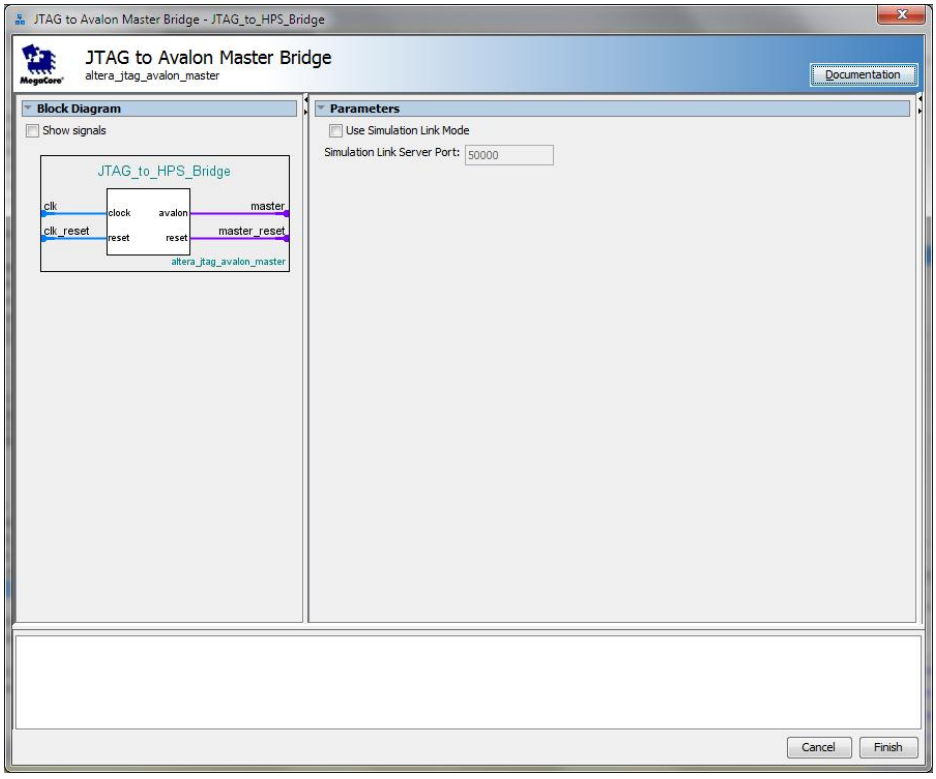

Рис. 59. Налаштування JTAG to Avalon Master Bridge

14. Додайте до системи модуль ***On-Chip Memory (RAM or ROM)*** та налаштуйте його у відповідності до рис. 60.

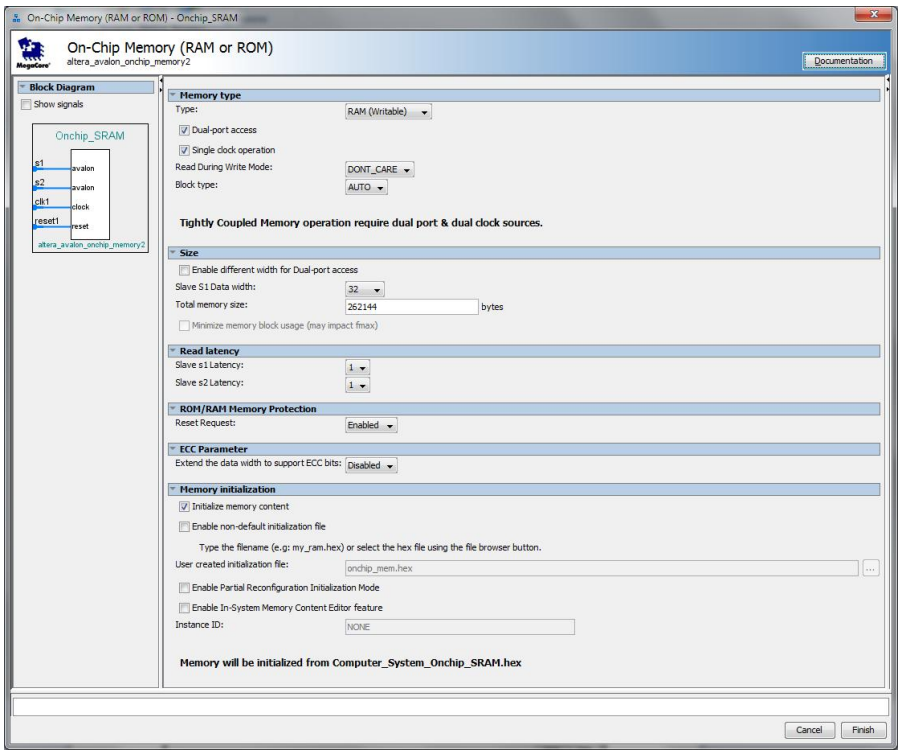

Рис. 60. Налаштування On-Chip Memory (RAM or ROM)

15. Додайте до системи два модулі ***PIO (Parallel I/O)*** та налаштуйте їх у відповідності до рис. 61 та 62.

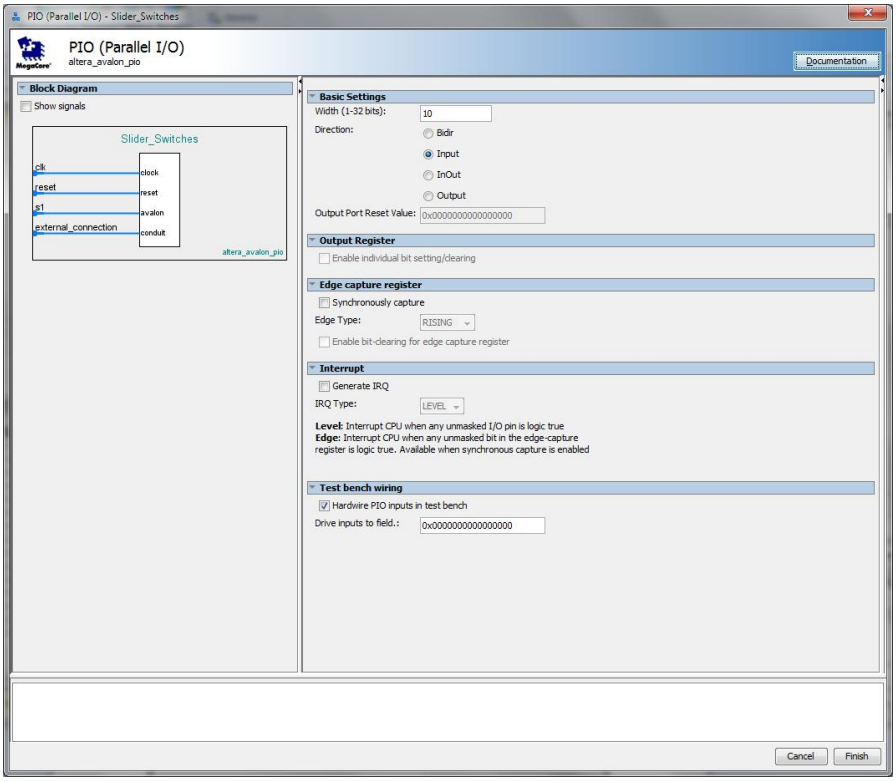

Рис. 61. Налаштування порту Slider_Switches

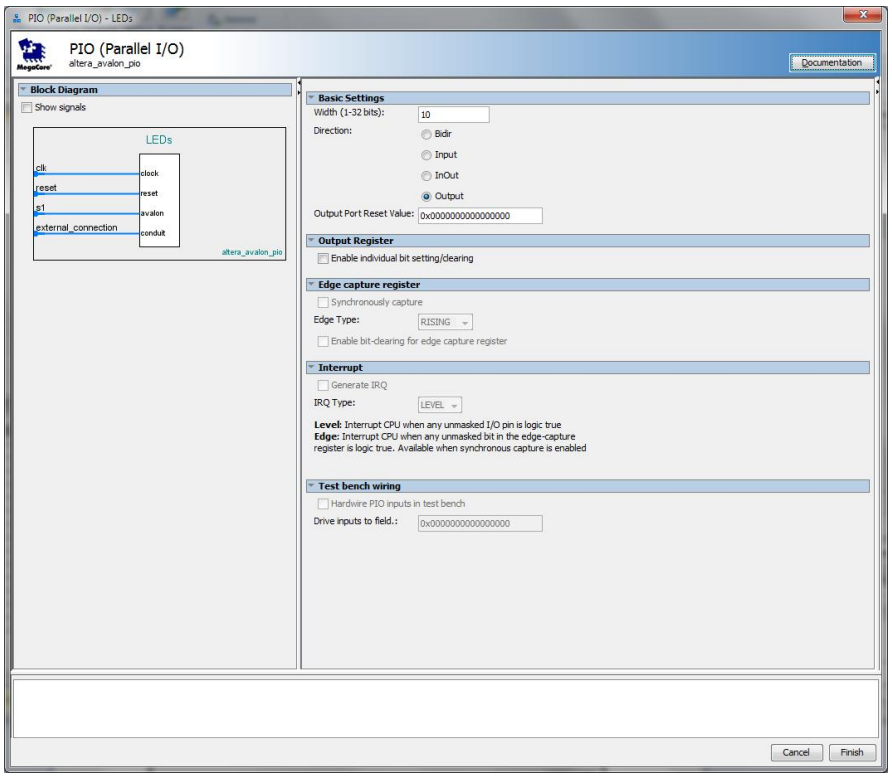

Рис. 62. Налаштування порту LEDs

16. Додайте до системи модуль ***System ID Peripheral*** та залиште його налаштування за замовченням (рис. 63). Він знаходиться у папці ***Basic Functions -> Simulation; Debug and Verification -> Debug and Performance.***

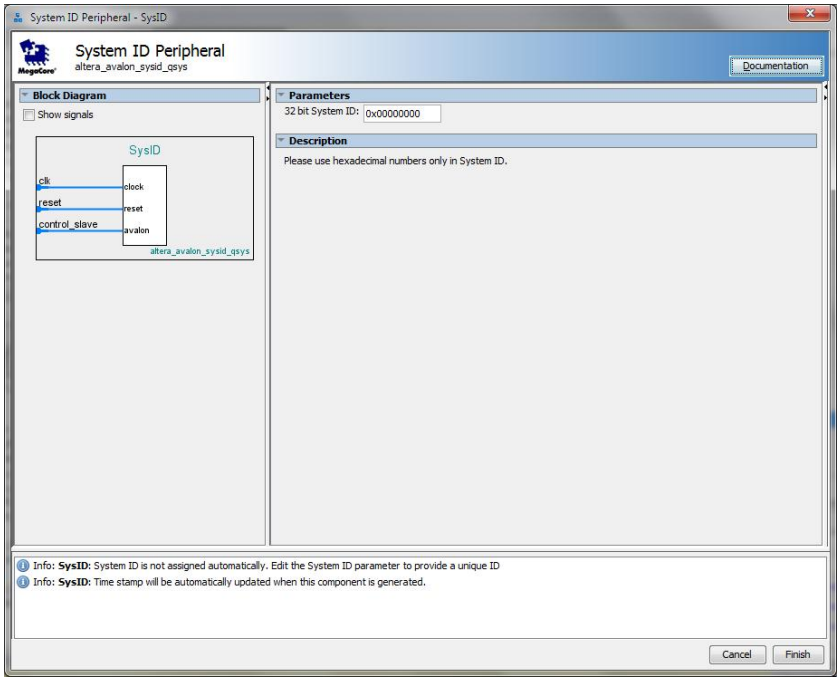

Рис. 63. Налаштування System ID Peripheral

17. Зробіть з’єднання компонентів системи у відповідності до рис. 64. Змініть назви компонентів системи. Визначте зовнішні сигнали (стовпчик ***Export***). Зверніть увагу на базові адреси компонентів.

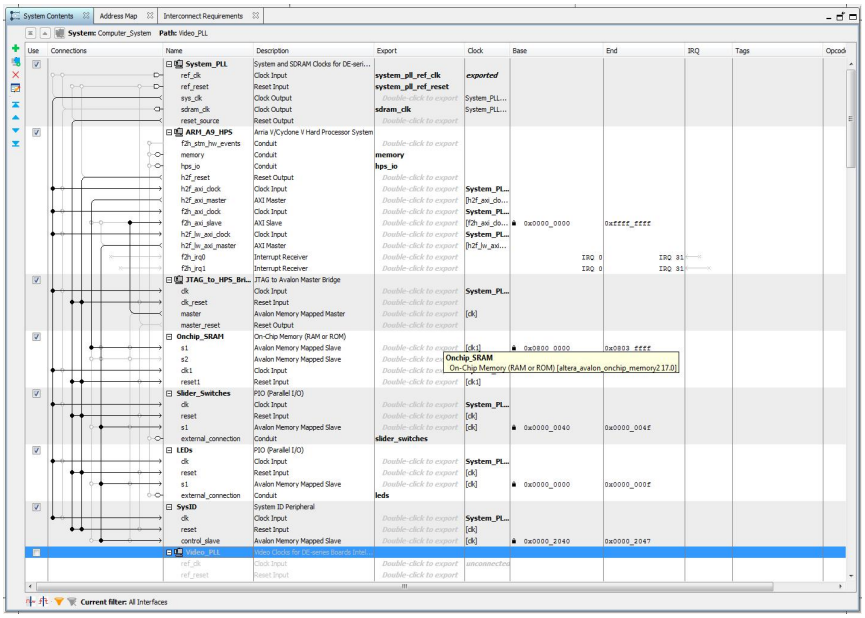

Рис. 64. З’єднання компонентів системи

18. Збережіть систему під назвою ***Computer_System.***

Тепер пояснимо, як FPGA взаємодіє з HPS частиною. Між ними є 3 мости:

F2H - FPGA-to-HPS bridge;

H2F - HPS-to-FPGA bridge;

LWH2F - Lightweight HPS-to-FPGA bridge.

F2H дозволяє FPGA звертатись до складових HPS (наприклад USB, або CAN контролери), при цьому FPGA буде master, а HPS — slave. Його слово може мати 32, 64, або 128 біт.

LWH2F використовується для доступу ARM-ядром до IP-cores, що генеруються у FPGA. Слово має розрядність 32 біти. НЕ МОЖНА використовувати для доступу до пам’яті.

H2F використовується для доступу ARM-ядром до пам’яті через FPGA. Його слово може мати 32, 64, або 128 біт.

19. Далі оберіть команду ***Generate → Generate HDL***. Для ***Create simulation model*** оберіть ***None***. Натисніть *Generate* та дочекайтеся сповіщення ***Generate completed***.

20. Поверніться до головного вікна середовища ***Quartus Prime***. Наразі необхідно долучити до проекту створену ***Computer_System***. Для цього у вікні ***Project Navigator → Files*** (зліва) натисніть правою кнопкою миші на піктограмі ***Files*** та оберіть команду ***Add/Remove Files in Project*** з контекстного меню. Відкриється вікно додавання файлів. Додайте файли ***директорія проекту/ Computer_System / synthesis/ Computer_System.v*** та ***Computer_System.qip***.
21. У файлі верхнього рівня ієрархії ***comp_arm.v*** підключіть сигнали до згенерованої системи:

```
Computer_System The_System (
.system_pll_ref_clk_clk                 (CLOCK_50),
.system_pll_ref_reset_reset             (1'b0),
.slider_switches_export                 (SW),
.leds_export                            (LEDR),
.sdram_clk_clk                          (DRAM_CLK),
.memory_mem_a                           (HPS_DDR3_ADDR),
.memory_mem_ba                          (HPS_DDR3_BA),
.memory_mem_ck                          (HPS_DDR3_CK_P),
.memory_mem_ck_n                        (HPS_DDR3_CK_N),
.memory_mem_cke                         (HPS_DDR3_CKE),
.memory_mem_cs_n                        (HPS_DDR3_CS_N),
.memory_mem_ras_n                       (HPS_DDR3_RAS_N),
.memory_mem_cas_n                       (HPS_DDR3_CAS_N),
.memory_mem_we_n                        (HPS_DDR3_WE_N),
.memory_mem_reset_n                     (HPS_DDR3_RESET_N),
.memory_mem_dq                          (HPS_DDR3_DQ),
.memory_mem_dqs                         (HPS_DDR3_DQS_P),
.memory_mem_dqs_n                       (HPS_DDR3_DQS_N),
.memory_mem_odt                         (HPS_DDR3_ODT),
.memory_mem_dm                          (HPS_DDR3_DM),
.memory_oct_rzqin                       (HPS_DDR3_RZQ),
.hps_io_hps_io_gpio_inst_GPIO35         (HPS_ENET_INT_N),
.hps_io_hps_io_emac1_inst_TX_CLK        (HPS_ENET_GTX_CLK),
.hps_io_hps_io_emac1_inst_TXD0          (HPS_ENET_TX_DATA[0]),
.hps_io_hps_io_emac1_inst_TXD1          (HPS_ENET_TX_DATA[1]),
.hps_io_hps_io_emac1_inst_TXD2          (HPS_ENET_TX_DATA[2]),
.hps_io_hps_io_emac1_inst_TXD3          (HPS_ENET_TX_DATA[3]),
.hps_io_hps_io_emac1_inst_RXD0          (HPS_ENET_RX_DATA[0]),
.hps_io_hps_io_emac1_inst_MDIO          (HPS_ENET_MDIO),
```

```
.hps_io_hps_io_emac1_inst_MDC      (HPS_ENET_MDC),
.hps_io_hps_io_emac1_inst_RX_CTL       (HPS_ENET_RX_DV),
.hps_io_hps_io_emac1_inst_TX_CTL       (HPS_ENET_TX_EN),
.hps_io_hps_io_emac1_inst_RX_CLK       (HPS_ENET_RX_CLK),
.hps_io_hps_io_emac1_inst_RXD1   (HPS_ENET_RX_DATA[1]),
.hps_io_hps_io_emac1_inst_RXD2   (HPS_ENET_RX_DATA[2]),
.hps_io_hps_io_emac1_inst_RXD3   (HPS_ENET_RX_DATA[3]),
.hps_io_hps_io_qspi_inst_IO0       (HPS_FLASH_DATA[0]),
.hps_io_hps_io_qspi_inst_IO1       (HPS_FLASH_DATA[1]),
.hps_io_hps_io_qspi_inst_IO2       (HPS_FLASH_DATA[2]),
.hps_io_hps_io_qspi_inst_IO3       (HPS_FLASH_DATA[3]),
.hps_io_hps_io_qspi_inst_SS0       (HPS_FLASH_NCSO),
.hps_io_hps_io_qspi_inst_CLK           (HPS_FLASH_DCLK),
.hps_io_hps_io_gpio_inst_GPIO61   (HPS_GSENSOR_INT),
.hps_io_hps_io_gpio_inst_GPIO40   (HPS_GPIO[0]),
.hps_io_hps_io_gpio_inst_GPIO41   (HPS_GPIO[1]),
.hps_io_hps_io_gpio_inst_GPIO48   (HPS_I2C_CONTROL),
.hps_io_hps_io_i2c0_inst_SDA           (HPS_I2C1_SDAT),
.hps_io_hps_io_i2c0_inst_SCL       (HPS_I2C1_SCLK),
.hps_io_hps_io_i2c1_inst_SDA           (HPS_I2C2_SDAT),
.hps_io_hps_io_i2c1_inst_SCL       (HPS_I2C2_SCLK),
.hps_io_hps_io_gpio_inst_GPIO54   (HPS_KEY),
.hps_io_hps_io_gpio_inst_GPIO53   (HPS_LED),
.hps_io_hps_io_sdio_inst_CMD           (HPS_SD_CMD),
.hps_io_hps_io_sdio_inst_D0        (HPS_SD_DATA[0]),
.hps_io_hps_io_sdio_inst_D1        (HPS_SD_DATA[1]),
.hps_io_hps_io_sdio_inst_CLK           (HPS_SD_CLK),
.hps_io_hps_io_sdio_inst_D2        (HPS_SD_DATA[2]),
.hps_io_hps_io_sdio_inst_D3        (HPS_SD_DATA[3]),
.hps_io_hps_io_spim1_inst_CLK     (HPS_SPIM_CLK),
.hps_io_hps_io_spim1_inst_MOSI    (HPS_SPIM_MOSI),
.hps_io_hps_io_spim1_inst_MISO    (HPS_SPIM_MISO),
.hps_io_hps_io_spim1_inst_SS0          (HPS_SPIM_SS),
.hps_io_hps_io_uart0_inst_RX       (HPS_UART_RX),
.hps_io_hps_io_uart0_inst_TX       (HPS_UART_TX),
.hps_io_hps_io_gpio_inst_GPIO09   (HPS_CONV_USB_N),
.hps_io_hps_io_usb1_inst_D0        (HPS_USB_DATA[0]),
.hps_io_hps_io_usb1_inst_D1        (HPS_USB_DATA[1]),
.hps_io_hps_io_usb1_inst_D2        (HPS_USB_DATA[2]),
```

```
.hps_io_hps_io_usb1_inst_D3        (HPS_USB_DATA[3]),
.hps_io_hps_io_usb1_inst_D4        (HPS_USB_DATA[4]),
.hps_io_hps_io_usb1_inst_D5        (HPS_USB_DATA[5]),
.hps_io_hps_io_usb1_inst_D6        (HPS_USB_DATA[6]),
.hps_io_hps_io_usb1_inst_D7        (HPS_USB_DATA[7]),
.hps_io_hps_io_usb1_inst_CLK           (HPS_USB_CLKOUT),
.hps_io_hps_io_usb1_inst_STP           (HPS_USB_STP),
.hps_io_hps_io_usb1_inst_DIR           (HPS_USB_DIR),
.hps_io_hps_io_usb1_inst_NXT           (HPS_USB_NXT)
);
```

22. Збережіть файл.
23. Імпортуйте призначення контактів вводу/виводу з файлу ***comp_arm.qsf.***
24. Виконайте компіляцію проекту.

5. Розробка програмного забезпечення.

1. Запустіть ***Intel FPGA Monitor Program***.
2. Оберіть меню ***File → New Project***. Збережіть проект під власною назвою у директорії ***app_software***. Оберіть архітектуру ***ARM Cortex-A9***. Натисніть ***Next***.

3. На вкладинці ***Specify the system*** вкажіть наступні параметри:

Select a system = <Custom system>

System description file =
директорія проекту/Computer_System.sopcinfo

FPGA programming (SOF) file =
директорія проекту /output_files/comp_arm.sof

Preloader = DE1-SoC

4. Натисніть ***Next.***
5. На вкладинці ***Specify a program type*** укажіть значення ***Program Type = Assembly Program***. Натисніть ***Next.***
6. На вкладинці ***Specify program details*** натисніть ***Add,*** та оберіть файли програми ***address_map_arm.s*** та ***simple_program.s*** .. Вкажіть ***Start symbol = _start***. Натисніть ***Next.***

   Цей приклад використовувався у лабораторній роботі № 3 – у ньому виконувалося копіювання стану перемикачів до регістру світлодіодів. Додатково, у цьому прикладі додаються семисегментні індикатори, які будуть відображати число, еквівалентне поточному стану світлодіодів.

7. На вкладинці ***Specify system parameters*** укажіть наступні значення

Host connection = DE-SoC [3-2]

Processor = ARM_A9_HPS_arm_a9_0

Terminal device = Semihosting

8. На останній вкладинці натисніть ***Finish***, або ***Save***. На запит чи завантажувати систему на плату відповідайте ***Yes.***
9. Після сповіщення про успішне завантаження, у вікні ***Intel FPGA Monitor Program —*** оберіть меню ***Actions → Compile & Load.***
10. Запустіть програму на виконання у безперервному (зелений трикутник), або покроковому режимі (жовта стрілочка) та спостерігайте її виконання. Переконайтеся, що червоні світлодіоди змінюють свій стан відповідно до стану перемикачів при виконанні інструкції
    stbio R4, 0(R3)**,** а на семи сегментних індикаторах відображається вірне значення.
11. Зупиніть виконання програми.
12. Другим проектом у ***Monitor_Program*** буде програма мовою С, основана на прикладі «робота з Semihosting».
13. Створіть новий проект в окремій директорії. Повторіть налаштунки попереднього проекту – за виключенням того, що проект ґрунтується на скомпільованій програмі AXF, написаної мовою С.
14. Додайте до проекту файл ***semihosting_example.axf.*** Впевніться, що всі додаткові файли (надаються до лабораторної роботи) теж знаходяться у папці проекту.
15. Завантажте систему, скомпілюйте програму та запустіть її на виконання. ***Semihosting*** - це спеціальний механізм, який забезпечує програмам, що виконуються на ARM-процесорі, доступ до сервісу дебаґера. Стандартні бібліотеки функцій модифіковані для роботи з ***Semihosting***. Наприклад, ***printf()*** та ***scanf()***, які зазвичай працюють з терміналом комп'ютера, на якому запущена програма, що їх викликає, будуть працювати з терміналом програми-дебаґера.
16. Якщо програма працює правильно, то можно буде запустити діалог (надати відповіді на запитання у консольному вікні) які надходять з процесора (рис. 65).

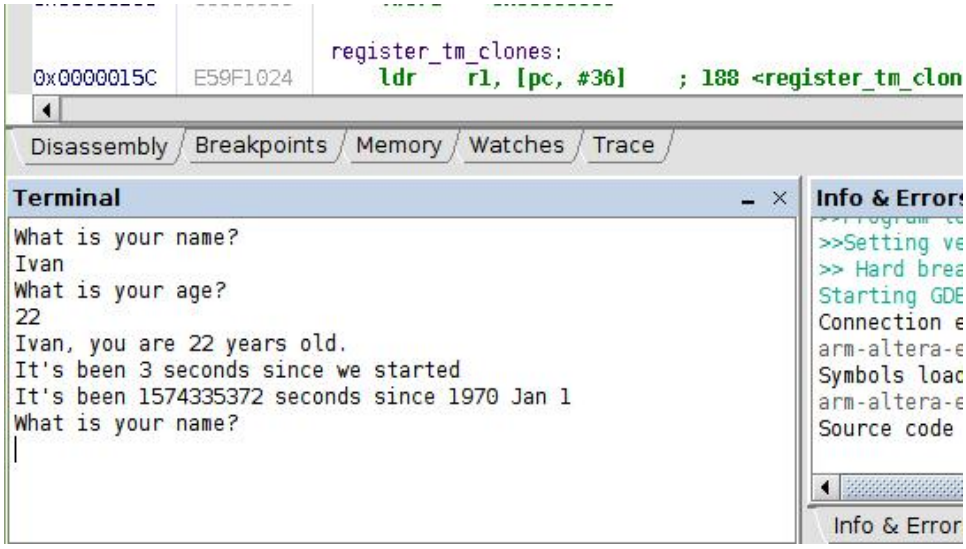

Рис. 65- Робота програми ***Semihosting***

3. Самостійна робота

Внесіть модифікацію до програми ***Semihosting***: нехай відповідь виводиться на світлодіоди. Наприклад, додайте коди:

volatile int * led_ptr = (int *)LEDR_BASE;

****led_ptr = age;***

у потрібних місцях. Коди можуть знаходитись не разом.

Якщо усе зроблено правильно, то після відповіді на запитання "*What is your age?*" вказане число буде виведене на LEDR[7:0] та HEX0-1.

4. Контрольні запитання.

1. За допомогою якої шини поєднуються до процесорного ядра ARM компоненти, синтезовані у FPGA?
2. Як відбувається налаштування периферійних компонентів процесорного ядра ARM?
3. Як відбувається налаштування синхронізуючих сигналів процесорного ядра ARM?
4. Яке призначення модулю ***JTAG to Avalon Master Bridge***?

5. Лабораторна робота № 5. Обробка аудіосигналів

1. Вступ.

Метою виконання цієї роботи є набуття навичок обробки аудіосигналів системою на кристалі, реалізованою на платі DE1-SoC.

2. Порядок виконання роботи.

1. Основою для створення власного проекту обираємо систему на кристалі, що використовується у Intel FPGA Monitor Program. Для цього — зробіть копію проекту, який розташовано у папці ***Директорія_розташування_Quartus***\University_Program\Computer_Systems\DE1-SoC\DE1-SoC_Computer\verilog до власної робочої директорії.

2. Відкрийте проект **DE1_SoC_Computer.**

3. Запустіть утіліту **Qsys (**або **Platform Designer).** Відкрийте файл опису системи **Computer_System.qsys** (рис. 66).

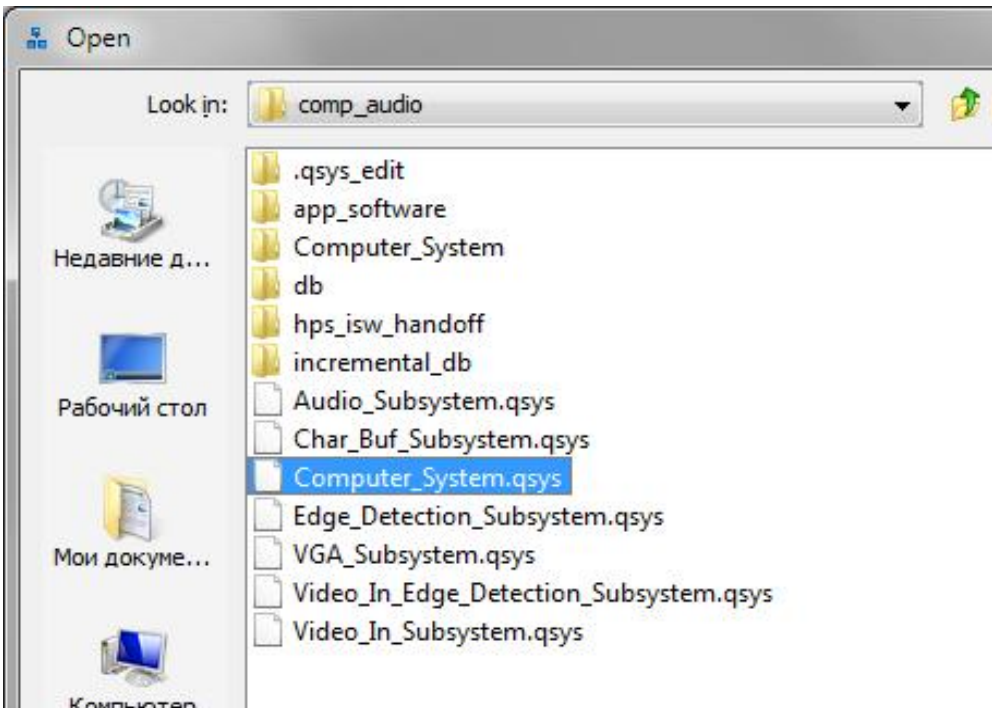

Рис. 66. Вибір файлу опису системи на кристалі

4. В обраній системі залишаємо лише ті компоненти, які використовуються при обробці аудіосигналу (рис. 67 та рис. 68).

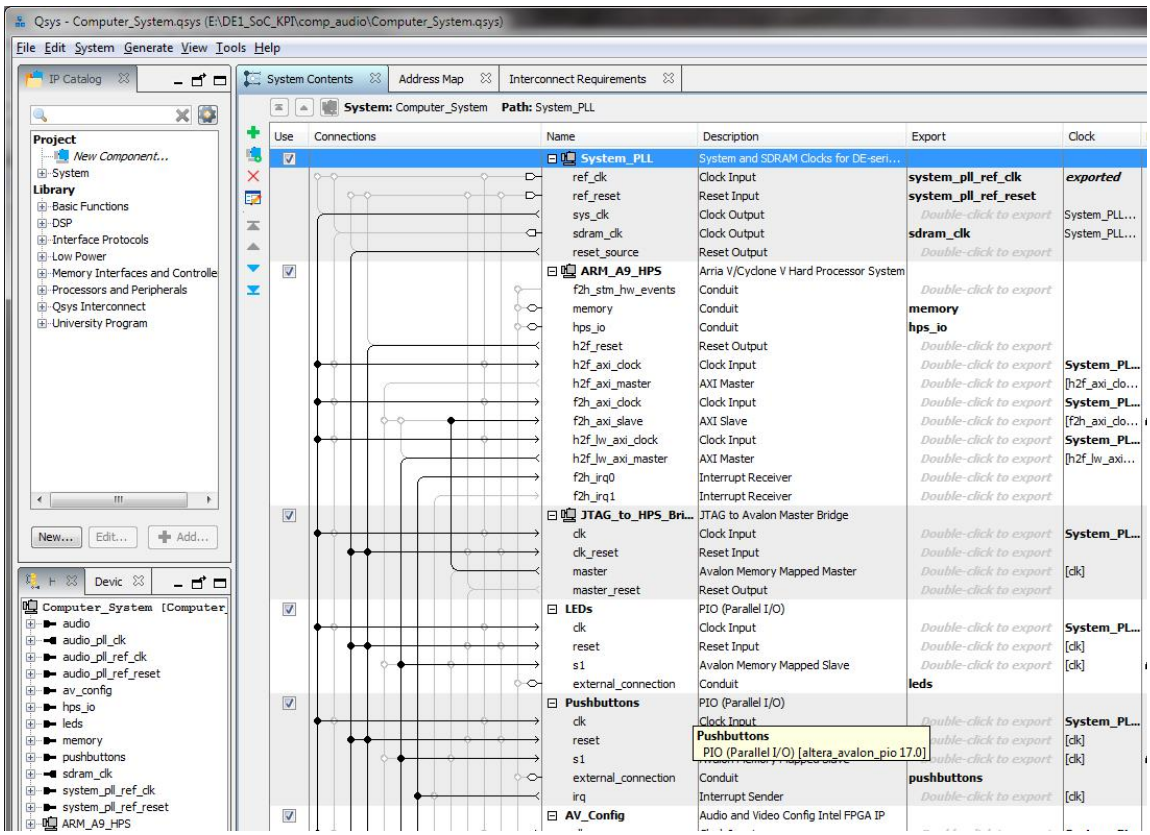

Рис. 67. Налаштування системи для обробки аудіосигналу А

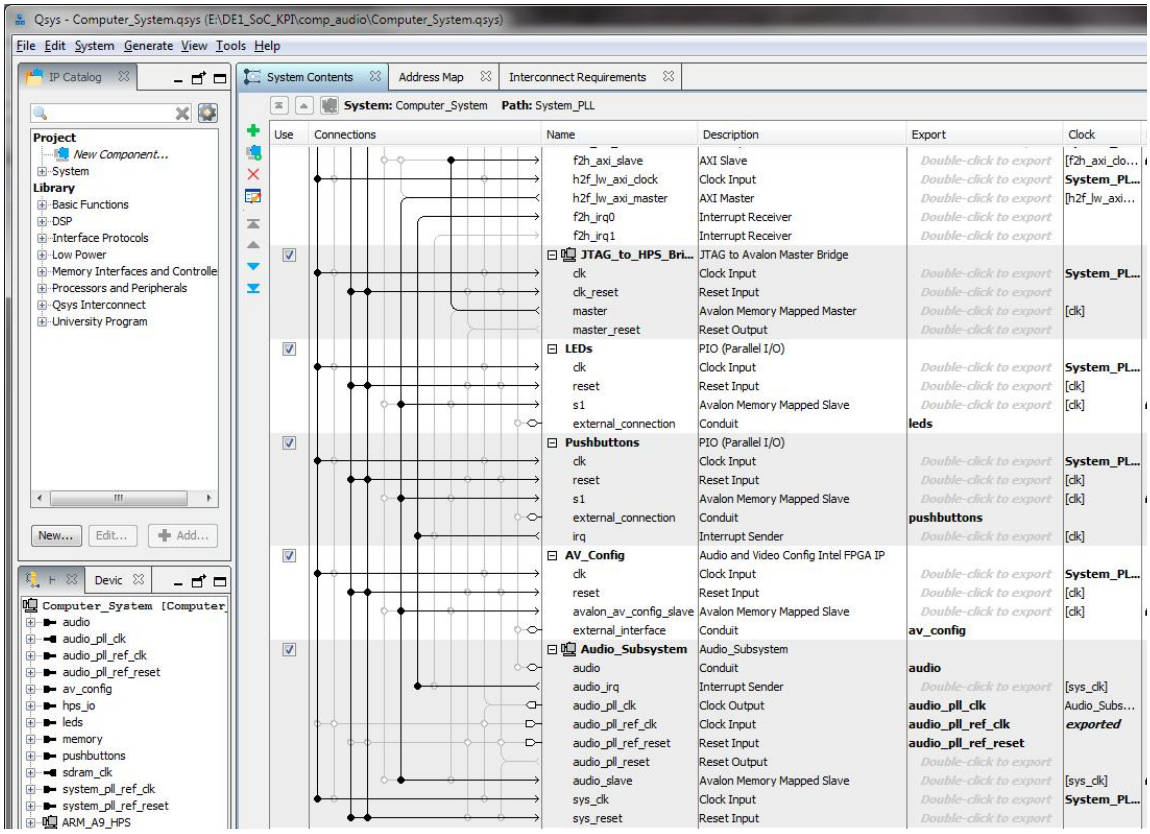

Рис. 68. Налаштування системи для обробки аудіосигналу Б

5. Важливу функцію при обробці аудіосигналу відіграють два компоненти - **AV_Config** та **Audio_Subsystem**. Розглянемо їх більш уважно.

6. Оберіть компонент **AV_Config,** натисніть праву кнопку миші та оберіть команду **Edit.**

7.Відкриється вікно налаштувань, яке показано на рис. 69.

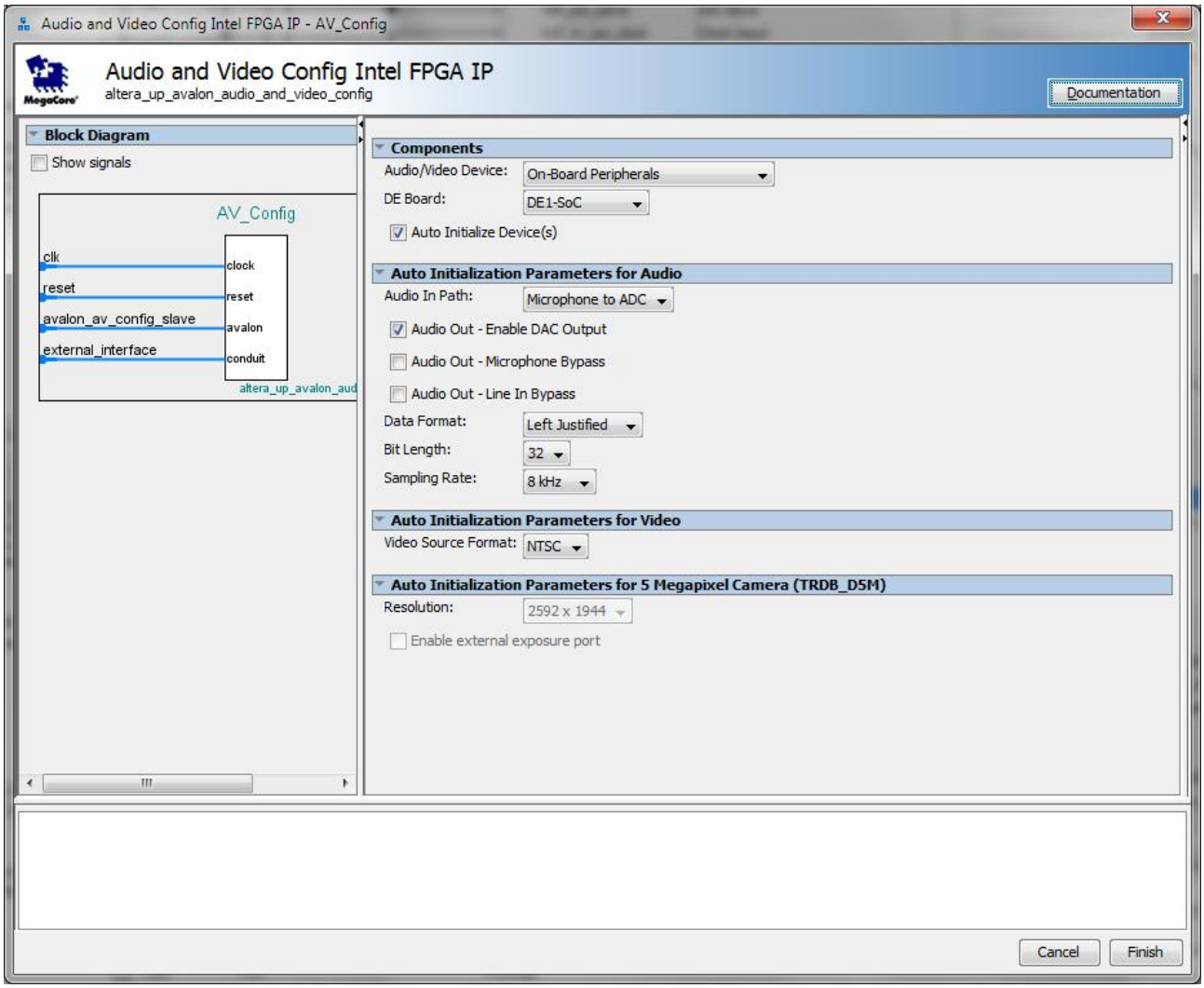

Рис. 69. Вікно налаштувань модуля AV_Config

8. Цей модуль використовується для налаштування зовнішнього аудіокодека WM8731, що розташований на платі DE1-SoC. За налаштуваннями видно, що вхід АЦП аудіокодеку під'єднано до мікрофону, а вихід ЦАП - до виходу Audio OUT. Вхідні дані містять 32 біти. Частота дискретизації становить 8 Кгц. Встановлено режим автоматичної ініціалізації кодеку при подачі живлення. Перевірьте ці налаштування у власному проекті.

9. Оберіть компонент **Audio_Subsystem** та відкрийте його для редагування на рис. 70.

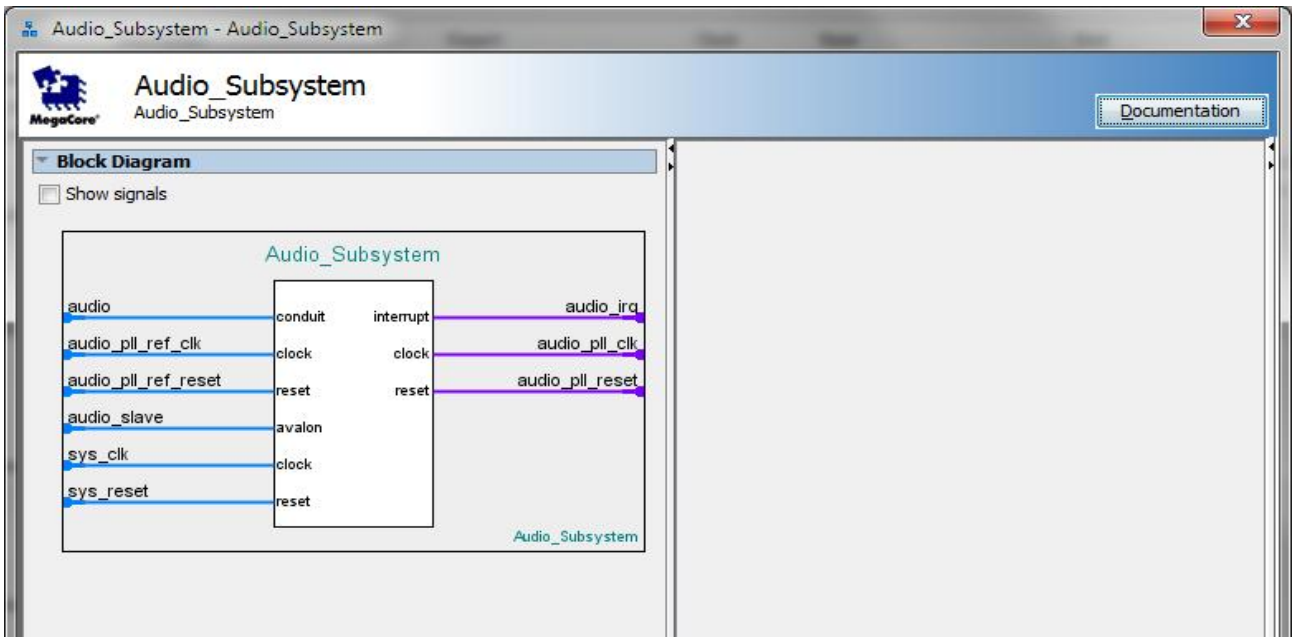

Рис. 70. Вікно налаштувань компоненту Audio_Subsystem

10. Будь які налаштування для цього компоненту відсутні. Цей модуль використовується для обміну даними між аудіокодеком та системною шиною Avalon.

11. Збережіть налаштування системи. Виконайте команду **Generate -> Generate HDL.**

12. Поверніться до проекту у САПР Quartus. У файлі верхнього рівня ієрархії закоментуйте сигнали, які не використовуються у модулі **Computer_System.**

13. Збережіть проект та запустіть його на компіляцію.

14. Після успішної компіляції проекту переходимо до налаштування програмного забезпечення. Для цього необхідно створити новий проект у середовищі **Monitor Program**.

15. Перед початком, завантажте до робочої директорії проекту папку **app_software** з файлами програмного забезпечення (додаток до виконання лабораторної роботи).

16. Вкажіть робочу директорію проекту (**app_software),** назву проекту - **audio,** та архітектуру - **ARM Cortex-A9.**

17. Оберіть **Custom System**, відповідні файли **sopsinfo** та **sof.** Preloader -> **DE1-SoC**.

18. Оберіть тип програми - **AXF, ELF or SREC File.**

19. В якості вхідного файлу оберіть **audio.axf.**

20. Наступні налаштування оберіть за замовченням.

21. Після завантаження системи виконайте завантаження програми до процесорного ядра ARM.

22. Для перевірки програми необхідно підключити мікрофон до воду MIC_IN та навушники до виходу LINE_OUT плати DE1-SoC. Програма працює наступним чином - після натискання кнопки KEY_0 (загориться світлодіод LEDR_0) система переходить у режим запису з мікрофону. Запис відбувається протягом 10 сек (до згасання світлодіоду LEDR_0). Після цього, натиснувши кнопку KEY_1, можна перейти до прослуховування запису.

23. Перевірте роботу програми.

3. **Самостійна робота**

1.Доопрацюйте систему таким чином, щоби вхідний аудіопотік подавався на вхід LINE_IN (відповідно, необхідно змінити частоту дискретизації аудіокодеку).

2.Доопрацюйте програмне забезпечення таким чином, щоби аудіопотік передавався на вихід системи відразу (без запису).

3.Спробуйте виконати обробку аудіопотоку за допомогою програмно реалізованого фільтру низьких частот.

4. Контрольні запитання.

1. Яке призначення модулю AV_Config?

2. Яку функцію виконує модуль Audio_Subsystem?

3. Які стандартні значення частоти дискретизації використовуються в аудіокодеку?

4. Що означає назва "аудіокодек"?

6. Лабораторна робота № 6. Обробка відеосигналу

1. Вступ

Метою виконання цієї роботи є набуття навичок обробки відеосигналів системою на кристалі, реалізованою на платі DE1-SoC.

2. Порядок виконання роботи.

Частина 1. Обробка відеосигналу з використанням HPS.

1. Основою для створення власного проекту обираємо систему на кристалі, що використовується у Intel FPGA Monitor Program. Для цього — зробіть копію проекту, який розташовано у папці ***Директорія_розташування_Quartus***\University_Program\Computer_Systems\DE1-SoC\DE1-SoC_Computer\verilog до власної робочої директорії.

2. Відкрийте проект **DE1_SoC_Computer.**

3. Запустіть утіліту **Qsys (**або **Platform Designer).** Відкрийте файл опису системи **Computer_System.qsys** (рис. 71).

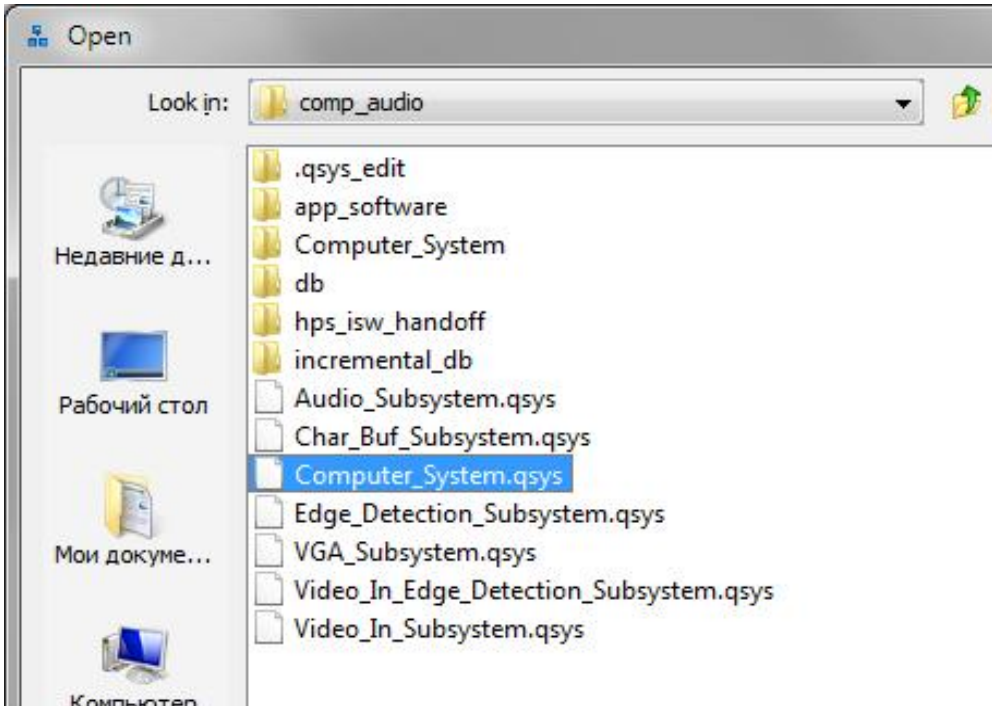

Рис. 71. Вибір файлу опису системи на кристалі

4. В обраній системі залишаємо лише ті компоненти, які використовуються при обробці відеосигналу (рис. 72 - 76).

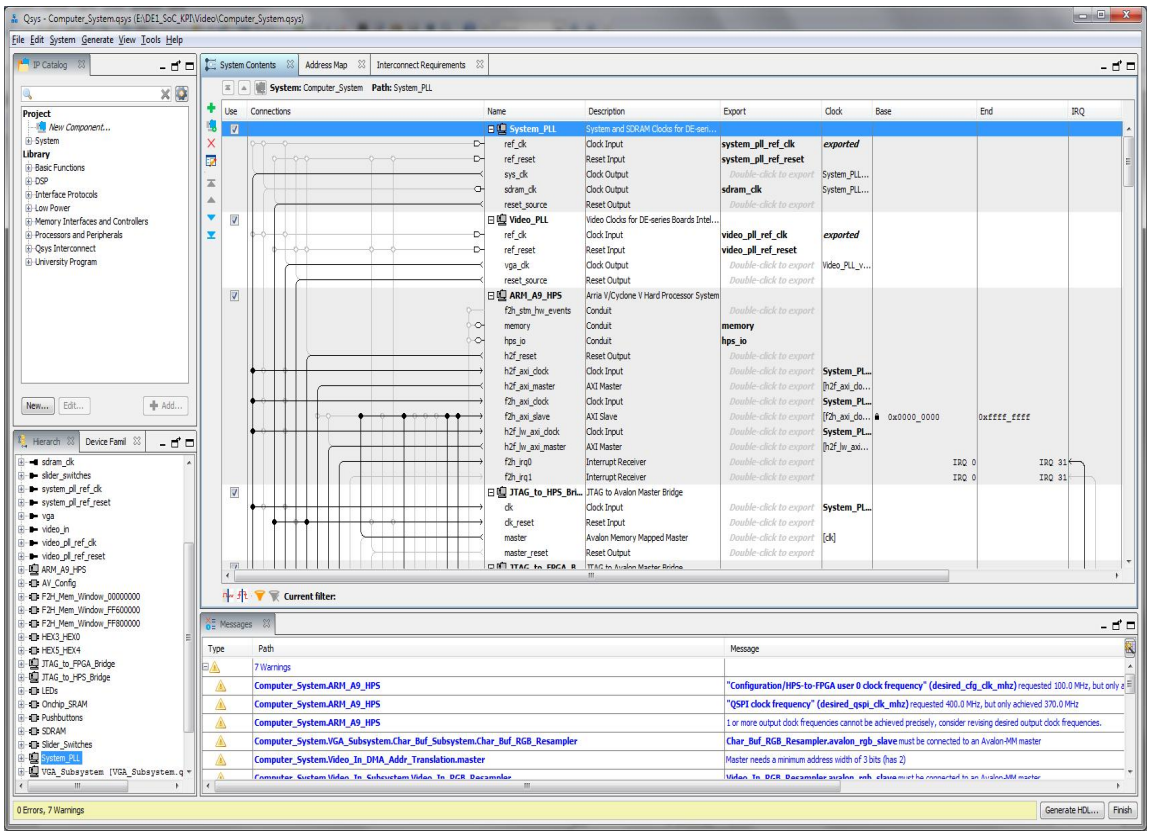

Рис. 72. Налаштування системи для обробки відеосигналу

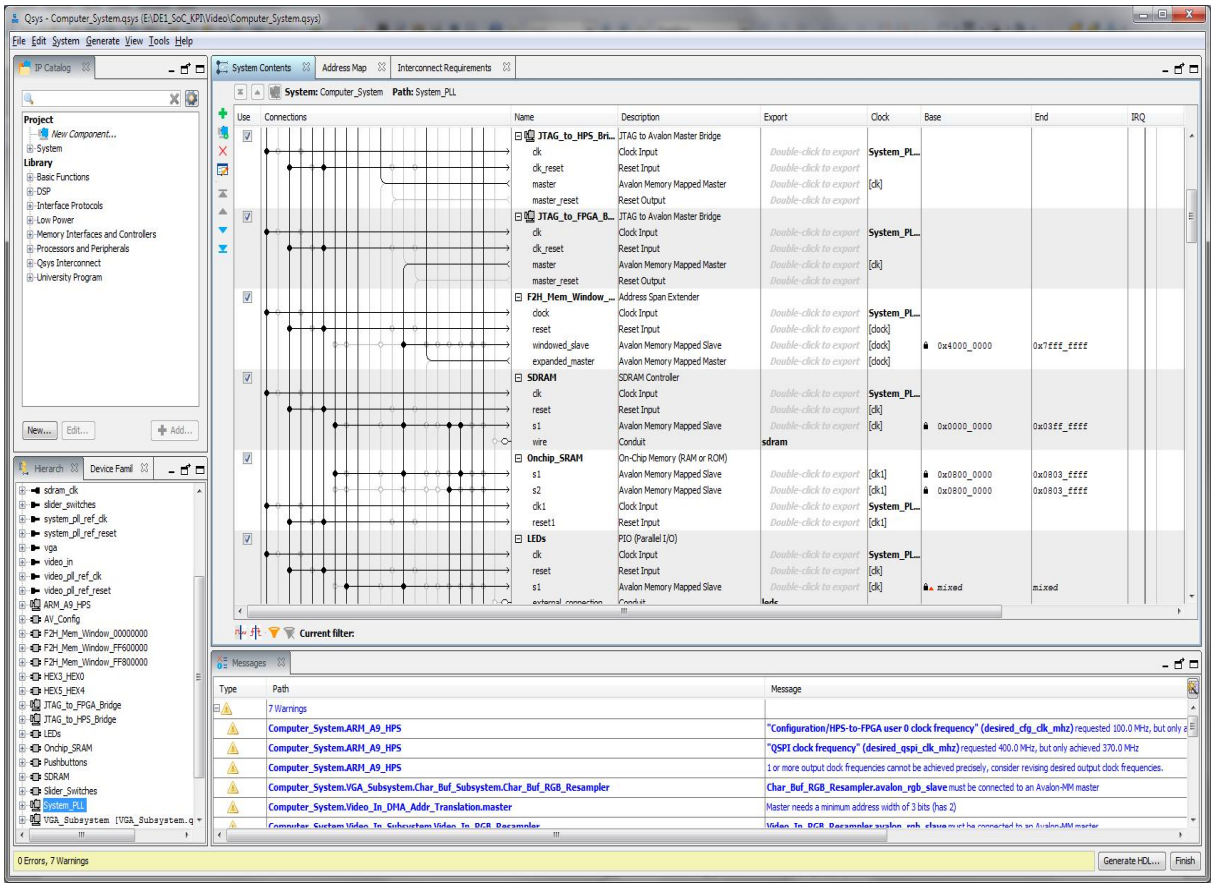

Рис. 73. Налаштування системи для обробки відеосигналу

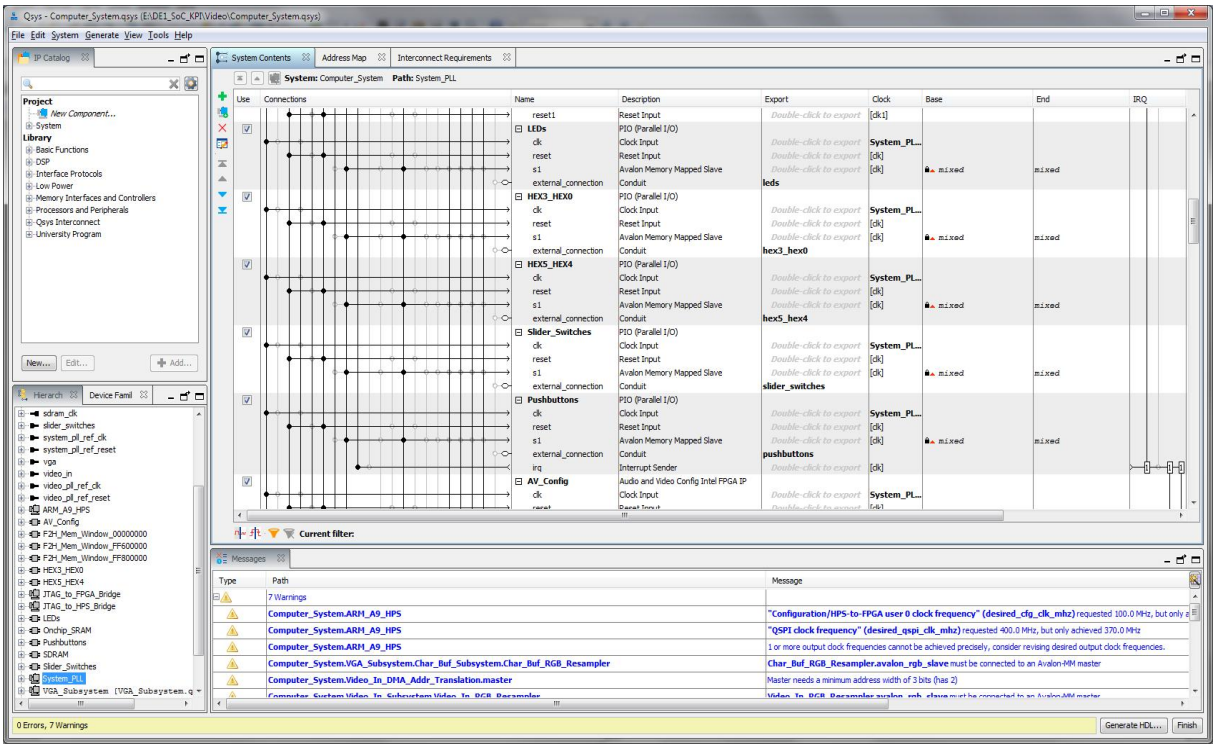

Рис. 74. Налаштування системи для обробки відеосигналу

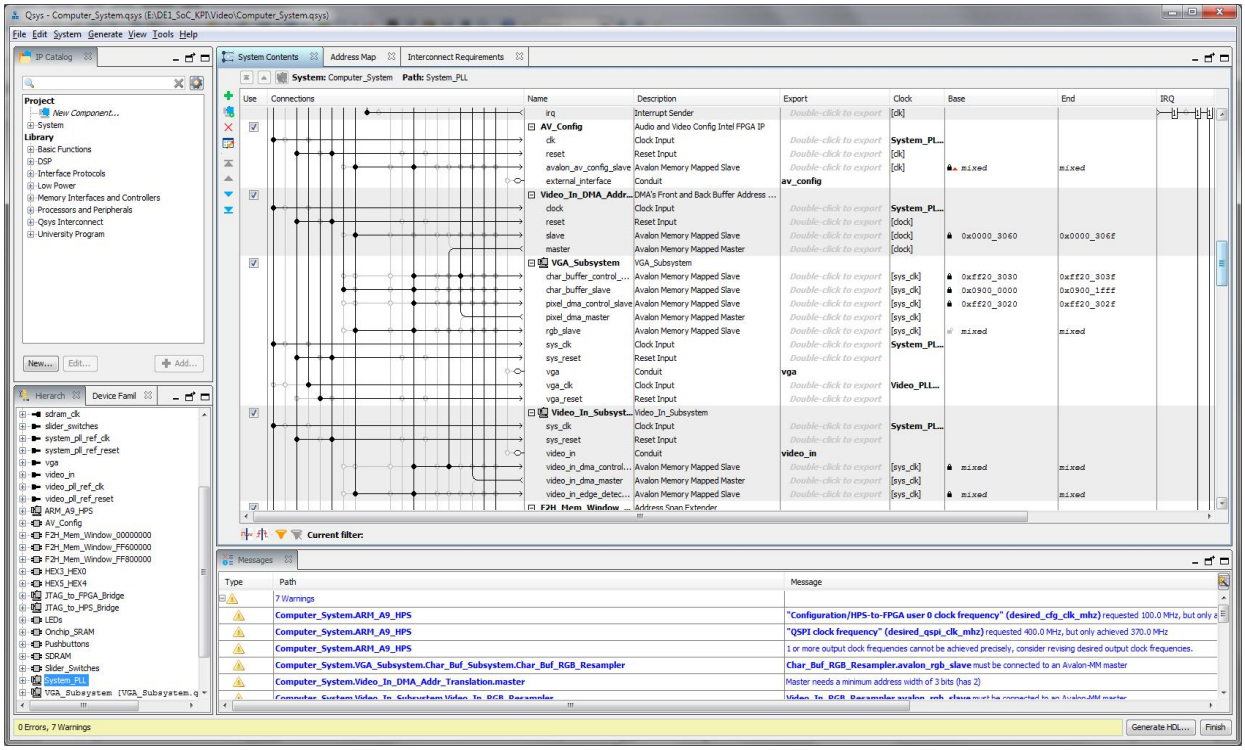

Рис. 75. Налаштування системи для обробки відеосигналу.

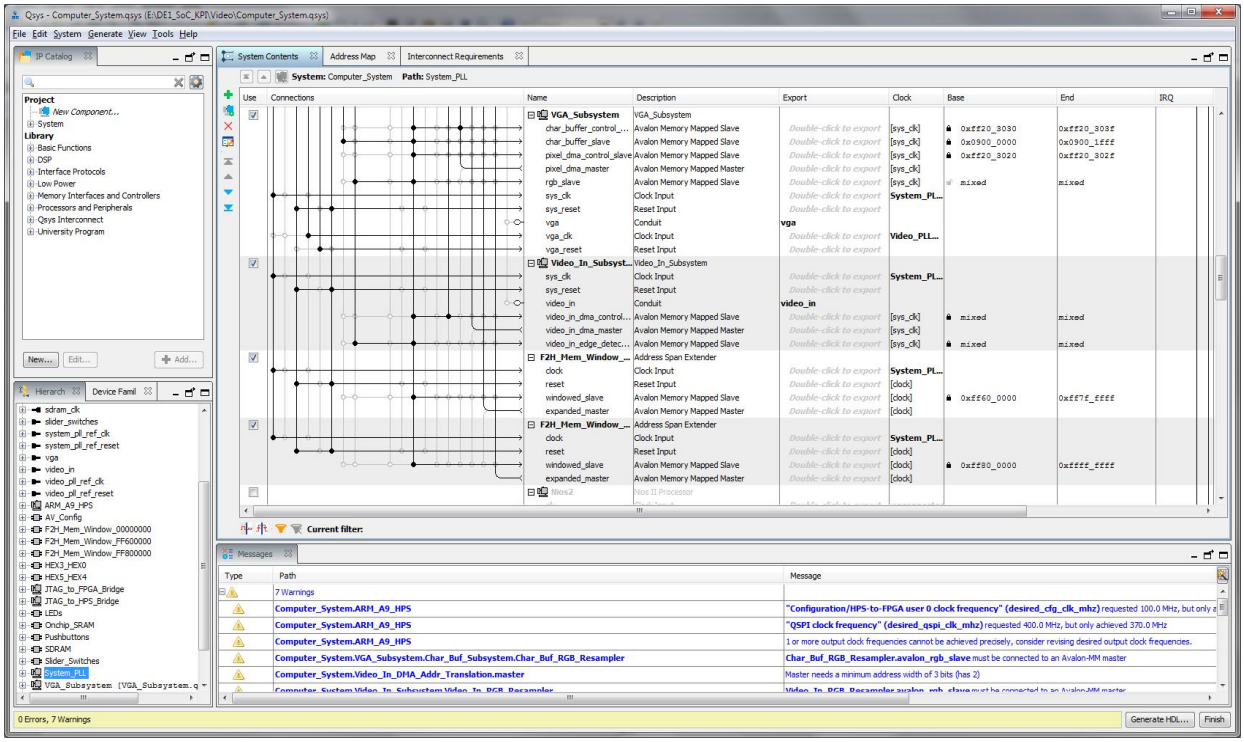

Рис. 76. Налаштування системи для обробки відеосигналу

5. Важливу функцію при обробці відеосигналу відіграють чотири компоненти - **AV_Config, VGA_Subsystem, Video_In_Subsystem, Video_In_DMA_Addr_Translation**. Розглянемо їх більш уважно.

6. Оберіть компонент **AV_Config,** натисніть праву кнопку миші та оберіть команду **Edit.**

7. Відкриється вікно налаштувань, яке показано на рис. 77.

8. Цей модуль використовується для налаштування зовнішнього відеокодека ADV7180, що розташований на платі **DE1-SoC**. Налаштуйте його на обробку відеосигналу формату PAL.

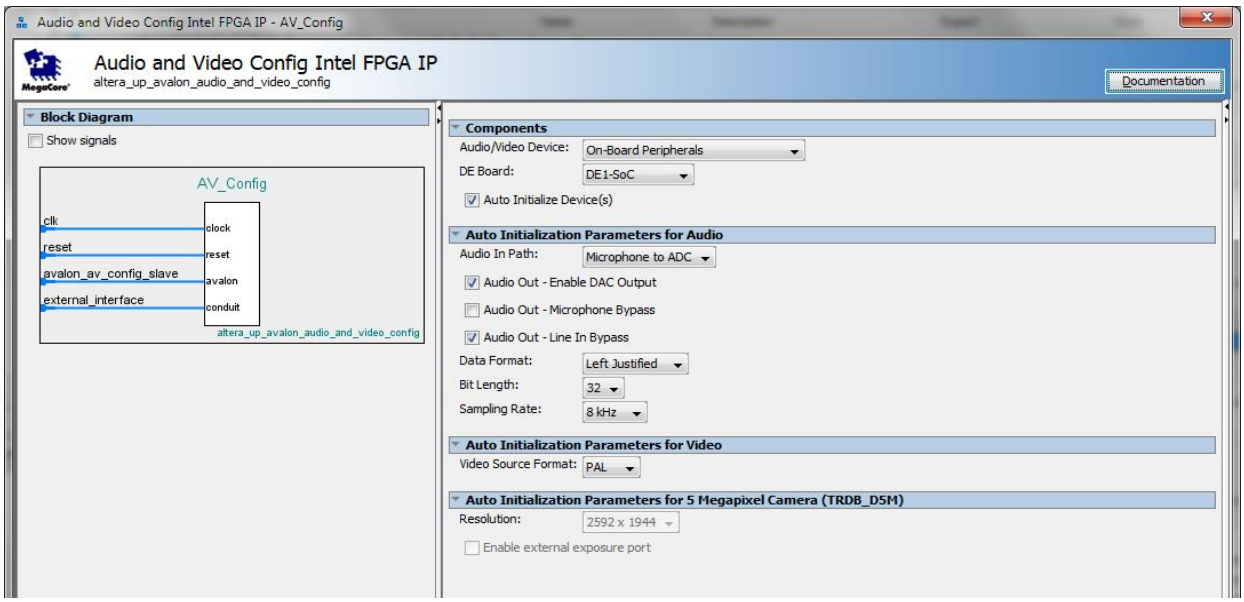

Рис. 77. Вікно налаштувань модуля AV_Config

9. Оберіть компонент **Video_In_Subsystem** та відкрийте його для редагування (рис. 78).

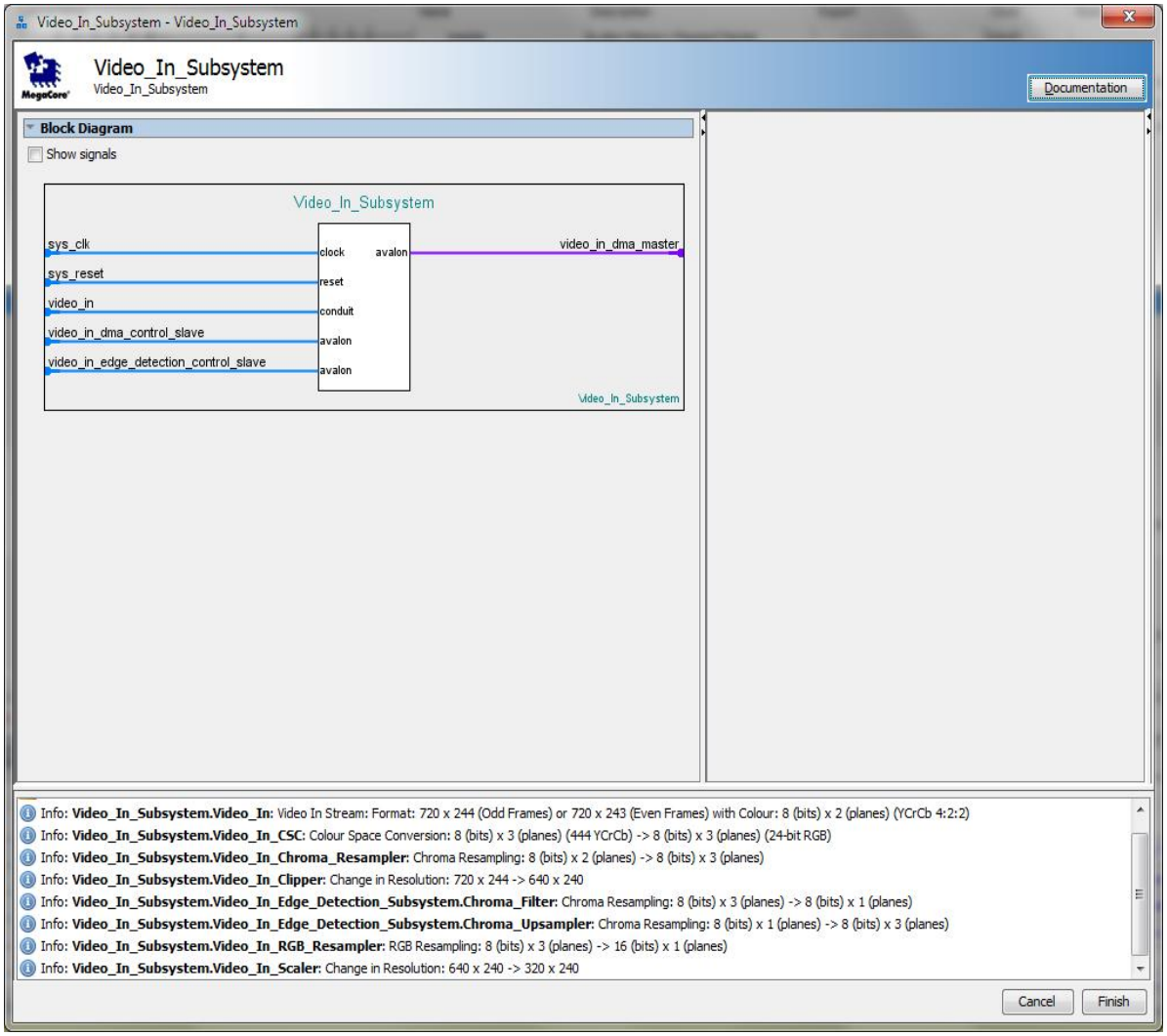

Рис. 78. Вікно налаштувань компоненту Video_In_Subsystem

10. Будь які налаштування для цього компоненту відсутні. Цей модуль використовується для обміну даними між відеокодеком та системною шиною Avalon. В інформаційному вікні можна побачити налаштування окремих складових цього модулю. Таким же чином можна переглянути налаштування всіх складових частин, які формують систему обробки відеосигналу.

11. Збережіть налаштування системи. Виконайте команду **Generate -> Generate HDL.**

12. Поверніться до проекту у САПР Quartus. У файлі верхнього рівня ієрархії закоментуйте сигнали, які не використовуються у модулі **Computer_System.**

13. Збережіть проект та запустіть його на компіляцію.

14. Після успішної компіляції проекту переходимо до налаштування програмного забезпечення. Для цього необхідно створити новий проект у середовищі **Monitor Program**.

15. Перед початком, завантажте до робочої директорії проекту папку **app_software_Lab6** з файлами програмного забезпечення (додаток до виконання лабораторної роботи).

16. Вкажіть робочу директорію проекту (**app_software),** назву проекту — **video,** та архітектуру - **ARM Cortex-A9.**

17. Оберіть **Custom System**, відповідні файли **sopsinfo** та **sof.** Preloader -> **DE1-SoC**.

18. Оберіть тип програми - **AXF, ELF or SREC File.**

19. В якості вхідного файлу оберіть **video.axf.**

20. Наступні налаштування оберіть за замовченням.

21. Після завантаження системи виконайте завантаження програми до процесорного ядра ARM.

22. Для перевірки програми необхідно підключити відеокамеру до воду VIDEO_IN та монітор до виходу VGA плати DE1-SoC. Програма працює наступним чином — після появи зображення, натискання кнопки KEY_0 призводить до появи стоп-кадру. Повторне натискання кнопки KEY_0 відновлює відеопотік. Переключення перемикача SW[0] вмикає обробку зображення з використанням фільтру **edge_detection**.

23. Перевірте роботу програми.

3. **Самостійна робота до частини 1.**

Спробуйте доопрацювати програмне забезпечення реалізованого фільтру для зміни його налаштування.

Частина 2. Обробка відеосигналу безпроцесорною системою.

1. Для виконання цієї частини роботи скористуємось проектом, що надається компанією Terasic. Розархівуйте проект DE1_SoC_TV, що міститься у додатках до лабораторної роботи.

2. Відкрийте цей проект у середовищі Quartus Prime.

3. Уважно перегляньте структуру проекту та з'ясуйте, які компоненти використовуються для обробки відеосигналу. Спробуйте розібратися з їх призначенням.

4. Скомпілюйте проект, та завантажте його до плати DE1-SoC.

5. Підключіть до плати відеокамеру та монітор. Перевірте роботу системи. Якщо все зроблено правильно - Ви повинні побачити на екрані монітору зображення з відеокамери.

4. Самостійна робота до частини 2.

Доопрацюйте проект таким чином, щоб була можливість виводити на монітор чорно-біле та негативне зображення. Для переключення між режимами можна використати перемикачі, які розташовані на платі DE1-SoC.

5. Контрольні запитання.

1. Яке призначення модулю Video_In_DMA_Addr_Translation?

2. Яку функцію виконує модуль Video_In_Subsystem?

3. В чому полягає принцип роботи фільтру edge_detection?

4. Які модулі використовуються в проекті DE1_SoC_TV? Поясніть їх призначення.

7. Лабораторна робота № 7. Завантаження Linux та розробка першого модуля

1. Вступ

Для того щоб завантажити Linux DE1-SoC необхідно завантажити зібраний образ Linux за посиланням (рис. 79):

https://www.intel.com/content/www/us/en/developer/topic-technology/fpga-academic/materials-embedded-systems.html

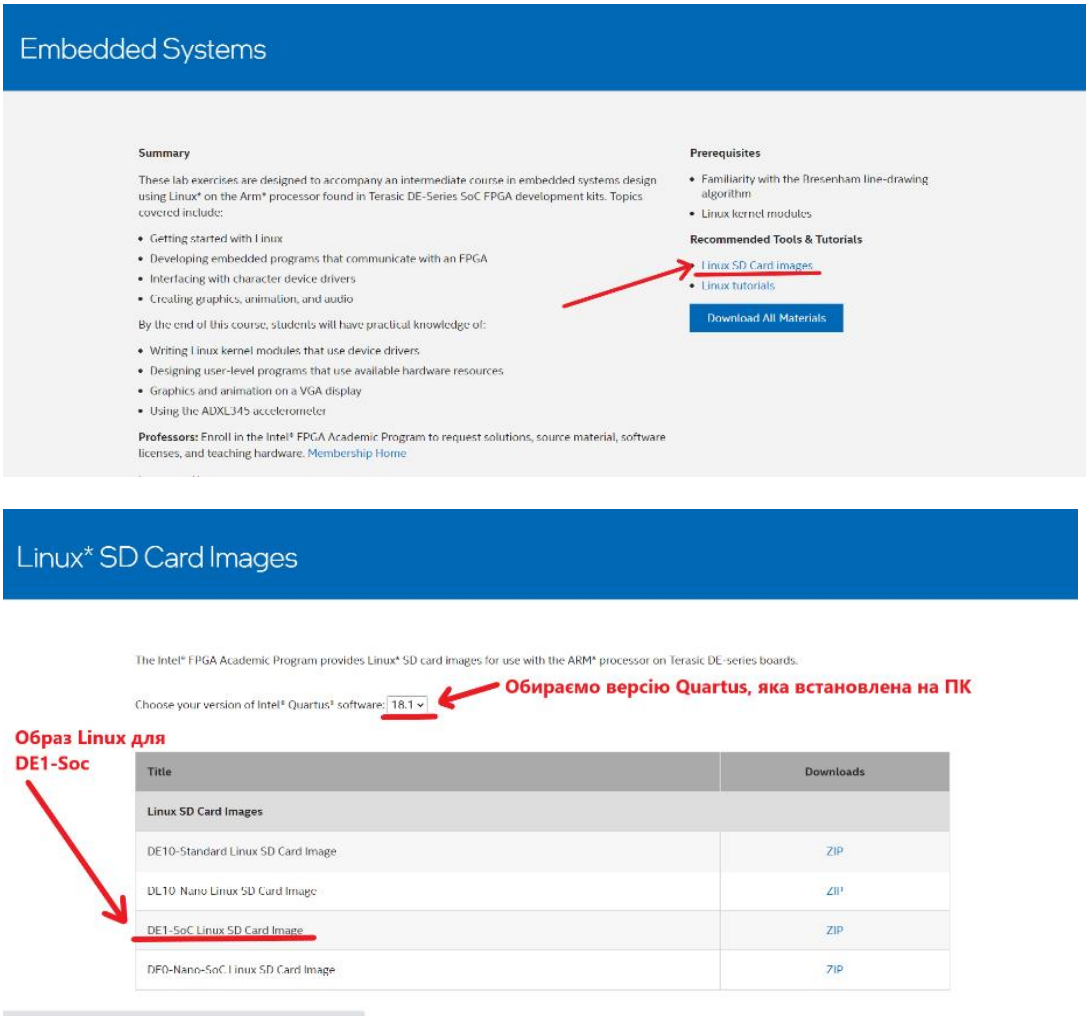

Рис. 79. Посилання, де можна викачати образ Linux

2. Порядок виконання роботи.

1. Далі потрібно розархівувати завантажений DE1-SoC.zip

2. Завантажте та встановіть програму з інтернету Win32 Disk Imager для запису образу Linux (DE1-SoC-UP-Linux.img) на microSD за посиланням: https://win32diskimager.org/#google_vignette.

3. Вставте картку microSD (8 ГБ або більше) у пристрій читання/запису картки microSD на комп'ютері, а потім запустіть програму Win32 Disk Imager.

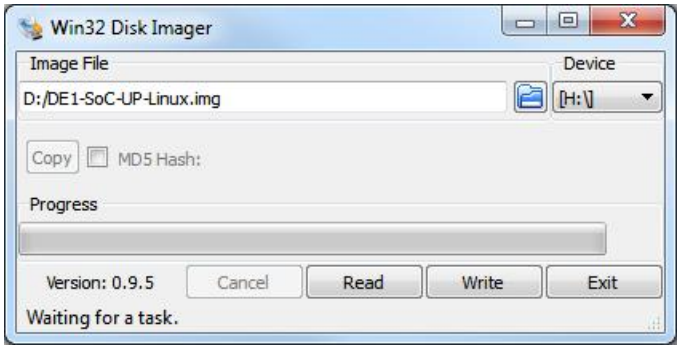

Рис. 80. Вікно Win32 Disk Imager

4. Виберіть літеру диска, відповідну картці microSD у розділі «Device», як показано на рис. 80.

5. Виберіть образ DE1-SoC-UP-Linux.img у розділі «Image File» як показано на рис. 81.

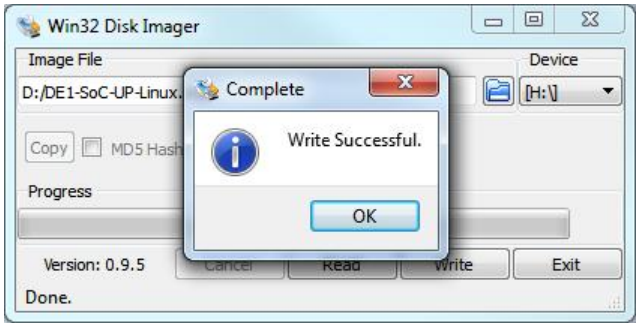

Рис. 81. Запис образу linux до microSD

6. Налаштуйте DE1-SoC для використання Linux. Спочатку переконайтеся, що плата DE-серії вимкнена, а потім вставте карту microSD Linux у microSD слот для карти. Перш ніж увімкнути плату, переконайтеся, що перемикачі MODE SELECT (MSEL), які знаходяться на нижній стороні плати, відповідають параметрам, зображеним на рис.82. Ці параметри налаштовують чіп SoC Cyclone V таким чином, що програмування ПЛІС відбувається за допомогою процесору ARM. Слід зазначити, що внесення цієї зміни до перемикачів MSEL не запобігає програмуванню ПЛІС за допомогою інших методів, наприклад, за допомогою програмування з середовища Intel Quartus Prime.

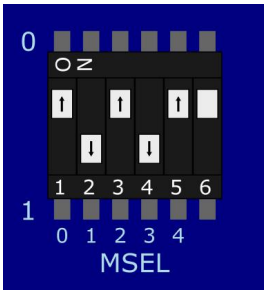

Рис. 82. Налаштування перемикачів MSEL на платі серії DE

7. Підключіть плату DE-серії до хост-комп’ютера за допомогою кабелю USB.

8. Далі необхідно зайти до диспетчеру пристроїв Windows та подивитись COM-порт під’єднаної плати. На рис.83. показано список доступних COM-портів диспетчера пристроїв на одному конкретному комп’ютері. Тут є лише один COM порт (UART-USB), якому присвоєно номер 3 (COM3). Якщо було перераховано більше COM-портів, то UART-USB -порт можна визначити, від'єднавши та знову підключивши кабель, щоб побачити, який порт COM зникає, потім знову з’являється у списку.

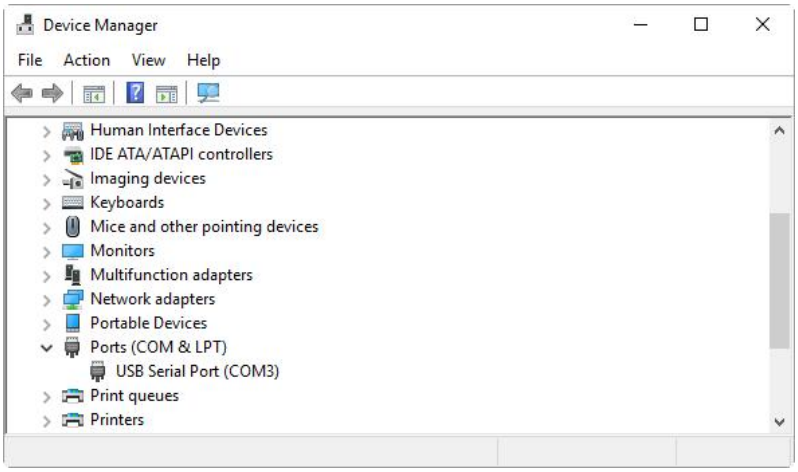

Рис. 83. Визначення COM-порту з'єднання UART-USB у Диспетчері пристроїв

9. Далі необхідно завантажити та встановити Putty за посиланням: https://putty.org.ru/download.html

10. Запустіть програму Putty. Виберіть панель "Serial", встановіть швидкість (швидкість передачі даних) до 115200 біт в секунду та введіть номер COM-порту. Приклад налаштувань показано на рис. 84.

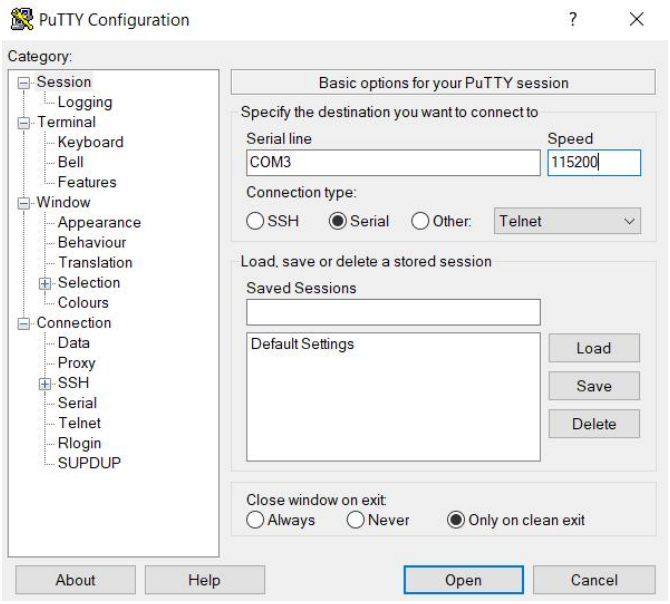

Рис. 84. Налаштування Serial порту в Putty

11. Натисніть Open після цього має відкритися віконце як на рис. 85.

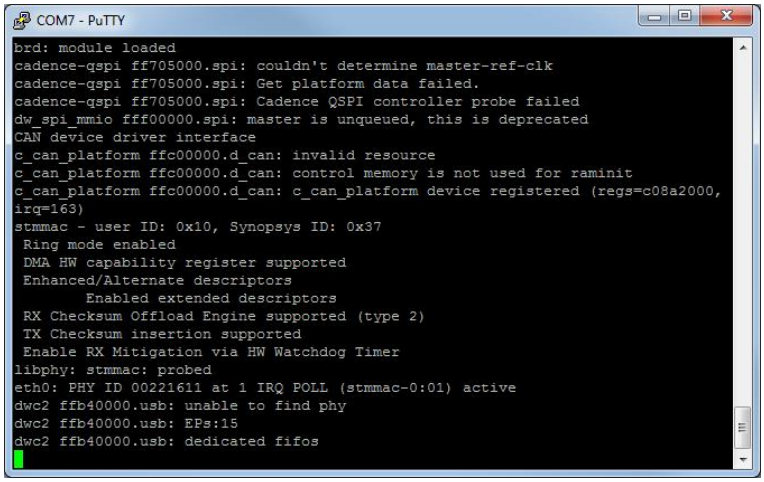

Рис. 85. Термінал Putty, що відображає текст під час завантаження ядра Linux.

12. Наступним кроком необхідно під’єднати плату до комп’ютеру Ethernet кабелем.

Слід зазначити: Плата серії DE налаштована на використання будь-якої з двох мережевих адрес IPv4 через Ethernet: 192.168.0.123 та 192.168.1.123. Для того, щоб хост-комп’ютер підключався до плати серії DE, мережа головного комп’ютера та адреса плати повинна знаходитися в одній підмережі. Це означає, що мережева адреса головного комп’ютера повинна мати форму 192.168.0.xxx або 192.168.1.xxx.

13. Далі необхідно змінити IP-адресу плати DE1-SoC. Для цього необхідно в терміналі через команду ifconfig переглянути та змінити IP-адресу. Наприклад, припустимо, що ваш головний комп’ютер має IP -адресу 192.168.86.33. Тоді ви можете скористатися командою **ifconfig eth0 192.168.86.**123 щоб змінити IP - адресу вашої плати. У цій команді 123 є лише прикладом - це може бути будь - яке число, яке не використовується у підмережі маршрутизатора. Зауважте, що eth0 - це назва порту Ethernet, доступного для Linux. Відповідні налаштування зображено на рис.86.

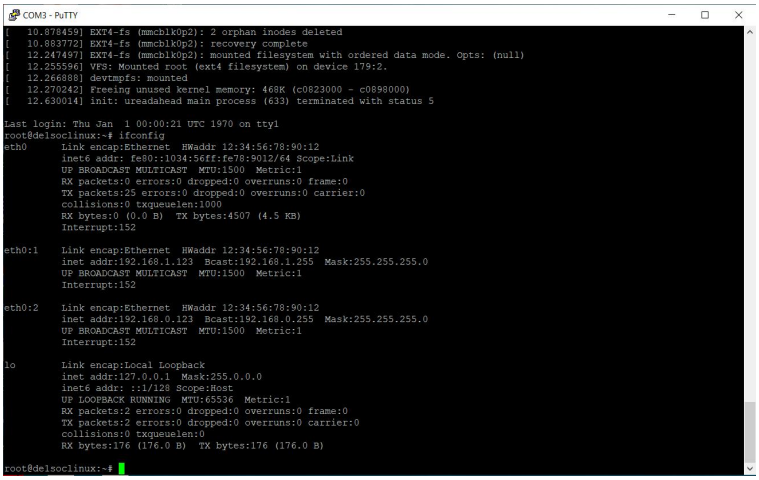

Рис. 86. Відображення інформації про IP-адресу плати

14. Інший варіант – це змінити IP-адресу вашого комп’ютера. Зайдіть в панель керування Windows. На панелі керування відкрийте пункт Центр мережі та спільного доступу. Натисніть Змінити налаштування адаптера, а потім клацніть правою кнопкою миші на адаптері Ethernet і відкрийте діалогове вікно Властивості. Виділіть протокол IP версії 4 (TCP/IPv4) пункт і натисніть Властивості. На вкладці Загальні натисніть Використовувати таку IP -адресу. У полі IP -адреса введіть 192.168.0.xxx (або 192.168.1.xxx), де xxx - це номер за вашим вибором (який ще не використовується у підмережі). У поле Маска підмережі введіть 255.255.255.0. Залиште поле шлюзу за замовчуванням порожнім рис. 87.

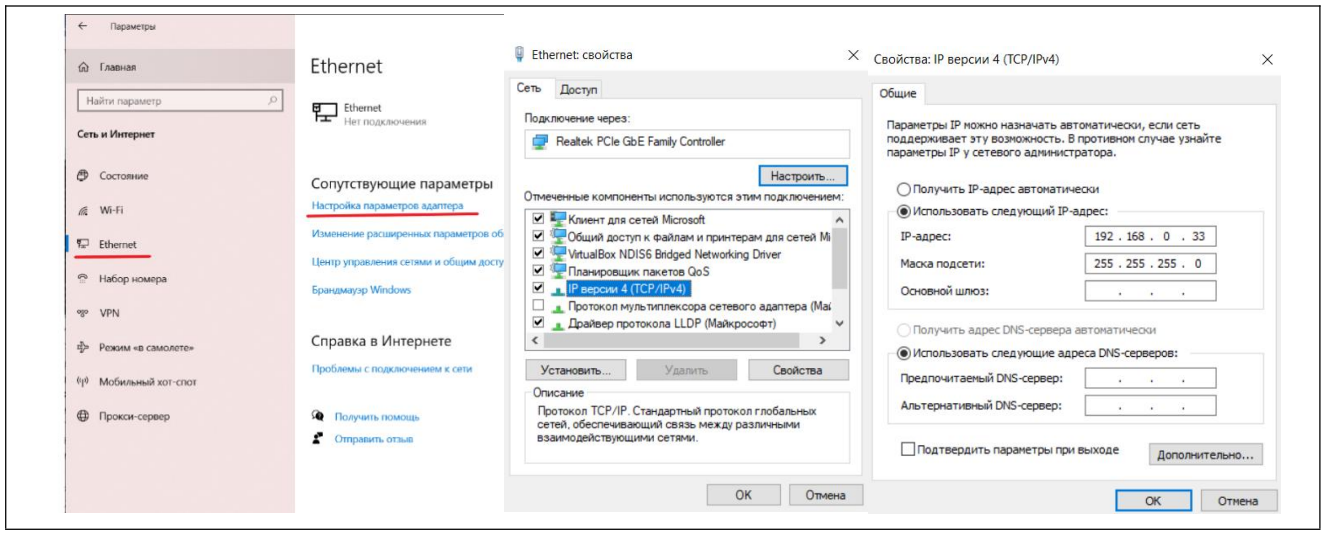

Рис. 87. Налаштування IP-адреси комп’ютера

15. Після того, як ви встановили мережеве з'єднання між вашим хост-комп'ютером та платою серії DE, ви можете скористатися програмою VNC Viewer на вашому головному комп’ютері для підключення до графічного інтерфейсу Linux. VNC Viewer можна завантажити за посиланням: https://www.realvnc.com/en/connect/download/viewer/

16. Відкрийте програму VNC Server та вкажіть адресу сервера (плати). На рис. 88. використовується IP -адреса за замовчуванням 192.168.0.123, а порт 5901 вказано відповідно до вимог.

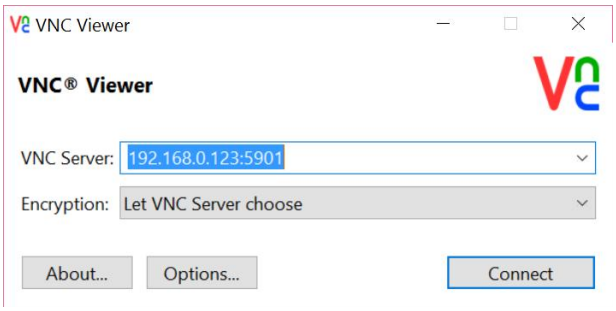

Рис. 88. Діалогове вікно RealVNC

17. Натисніть кнопку «Connect» і відкриється інше діалогове вікно, у якому буде запропоновано ввести пароль, введіть пароль password. В результаті появиться вікно графічного інтерфейсу Linux (рис.89).

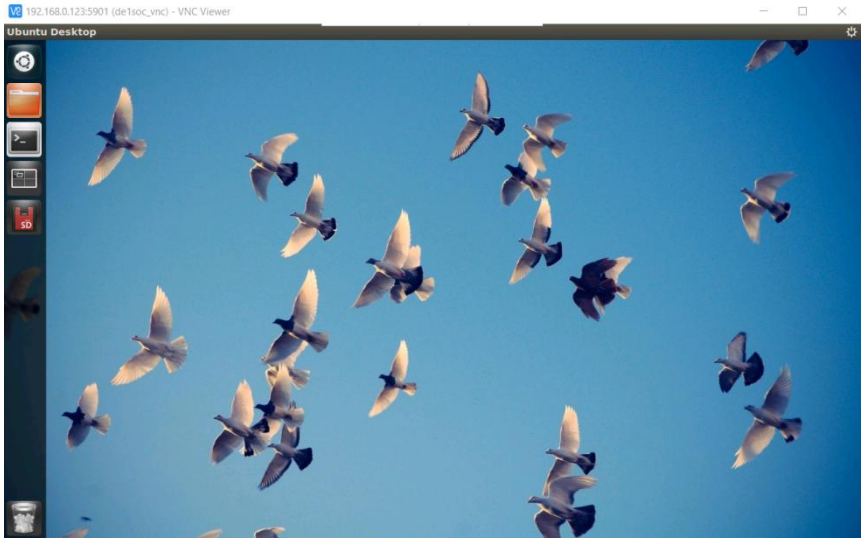

Рис. 89. Основне вікно RealVNC

18. Для передачі файлів з комп'ютера до серверу (плати) буде використовуватись програма FileZilla, яка доступна за посиланням: https://filezilla-project.org/

19. Запустіть FileZilla та в комірці «Хост» введіть IP- адресу плати. Введіть ім'я користувача – root, пароль – password, порт – 22 (рис. 90.).

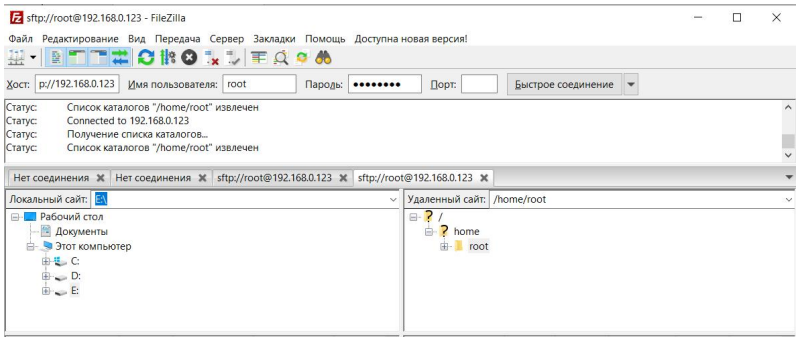

Рис. 90. Діалогове вікно FileZilla.

20. Розархівуйте архів 1_Linux_On_DE_Series_Boards.rar в зручну для вас папку та через програму FileZilla перекиньте вміст архіву (а саме tutorial_files для DE1-SoC) до серверу (плати).

21. В папці tutorial_files є три папки, а саме helloworld, increment_leds, pushbutton_irq_handler. Перейдемо в папку helloworld та запустимо программу helloworld.c. Для того щоб скомпілювати программу набираємо команду **gcc**

22. **helloworld.c -o helloworld**, де відповідно перший аргумент -це файл вихідного коду helloworld.c, другий - -o helloworld, інформує компілятору, щоб той створив виконавчий файл з назвою helloworld. Коли компіляція буде завершена, ми зможемо запустити програму набравши ./helloworld. В результаті має бути виведений рядок “Hello World!” (рис. 91.)

Рис. 91. Виконання програми HelloWorld.c

23. Вийдіть з папки helloworld та перейдіть в папку increment_leds. Там знаходиться програма increment_leds.c.

24. Скомпілюйте та запустіть дану програму, використовуючи компілятор gcc та продемонструйте виконання програми.

25. В папці pushbutton_irq_handler знаходиться модуль ядра Linux під назвою pushbutton_irq_handler.c. Зібрати його ви можете за допомогою Makefile. Для цього в терміналі введіть команду **make**, після цього ви побачите зібраний pushbutton_irq_handler.ko. Запустити його можна командою

26. insmod pushbutton_irq_handler.ko Продемонструйте виконання програми. Для того щоб зупинити модуль введіть команду **rmmod pushbutton_irq_handler**. Продемонстуйте виконання модуля.

3. Самостійна робота

Завдання 1. Розібратись з прикладами та доопрацювати проект таким чином, щоб програма (userspace) включала по одному світлодіоду LEDR одночасно. Спочатку повинен увімкнутись крайній правий світлодіод LEDR0, потім LEDR1, потім LEDR2 тощо. Коли ви дістанетесь до крайнього лівого світлодіода LEDR9, напрямок слід змінити. Ліхтарики повинні ходити з ліва

направо і так циклічно з затримкою. Щоб реалізувати затримку, ви можете використовувати системну функцію Linux, наприклад nanosleep. Документацію можна знайти в Інтернеті. Рекомендовано прочитати документацію Linux_On_DE_Series_Boards, яку можна завантажити за посиланням: https://ftp.intel.com/Public/Pub/fpgaup/pub/Teaching_Materials/current/Tutorials/Linux_On_DE_Series_Boards.pdf

Завдання 2. Напишіть модуль ядра Linux, який рахує кількість натискань та виводить на семисегментні індикатори обчислене значення. Рекомендовано прочитати документацію DE1-SoC_Computer_ARM, де ви зможете знайти інформацію про віртуальні адреси регістрів, які відповідають за периферію і як з ними працювати.

4. Контрольні запитання.

1. Як налаштовується плата DE1-SoC для програмування ПЛІС процесором ARM?

2. Як змінити мережеву адресу плати DE1-SoC?

3. За допомогою якого інтерфейсу відбувається взаємодія процесорного ядра ARM з периферійними пристроями, синтезованими у ПЛІС?

4. За якими адресами у просторі пам'яті розташовані регістр масок переривань та регістр даних зовнішніх кнопок?.

8. Лабораторна робота № 8. Таймер та простий символьний пристрій

1. Вступ

Для виконання даної роботи слід знати, як працювати з апаратним таймером. DE1-SoC має можливість використовувати цілий ряд апаратних таймерів. В даній роботі буде застосовано інтервальний таймер, реалізований в FPGA, який називається FPGA Timer0.

2. Порядок виконання роботи.

Регістри інтерфейсу цього таймеру розташовані за базовою адресою 0xFF202000. Як показано на рис. 92, цей таймер має шість 16-розрядних регістрів для налаштування. Для використання таймера потрібно записати відповідне значення в регістри початкових значень лічильника (їх два, один для старших 16 біт і один для молодших 16 біт повного 32-бітного значення лічильника). Щоб запустити лічильник, необхідно встановити біт START у регістрі керування в 1. Після запуску, таймер дорахує до 0 від початкового значення з регістру початкового значення лічильника. Лічильник автоматично перезавантажить це значення та продовжить рахувати, якщо біт CONT в регістрі керування встановлено у 1. Коли лічильник досягне 0, він встановить біт TO в регістрі стану у 1.

Цей біт можна очистити програмно, записавши до нього 0. Якщо біт ITO в регістрі керування встановлено до 1, тоді таймер генеруватиме переривання ARM* кожного разу, коли встановлюється біт TO. Тактова частота таймера становить 100 МГц. Вектор переривання таймера дорівнює 72. Скористуйтеся прикладом з документу «Using Linux on DE-series Boards», для визначення номеру вектора переривання у середовищі Linux*, та переконайтеся, що переривання викликає саме модуль таймеру.

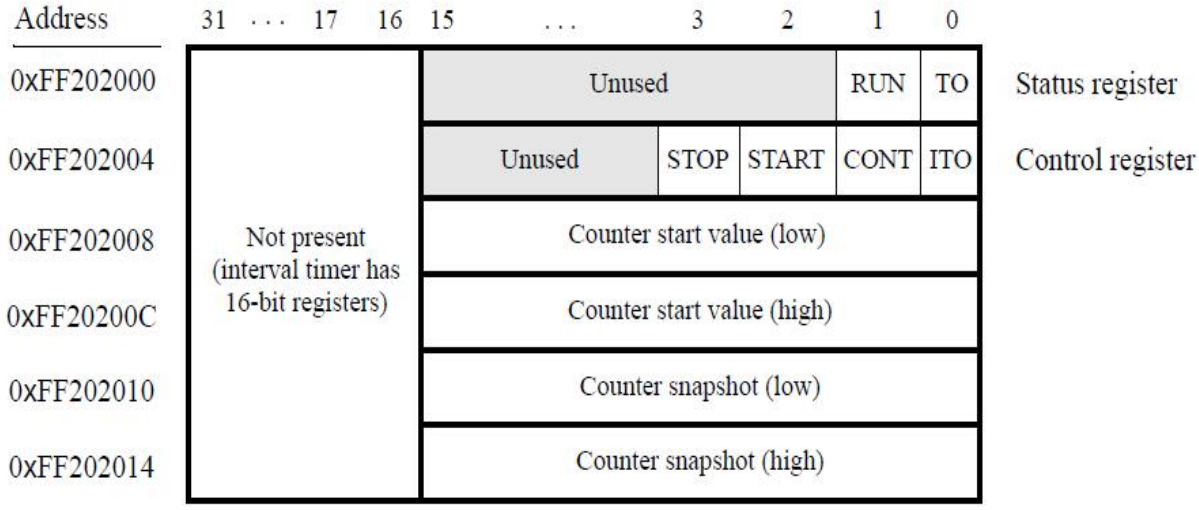

Рис. 92. Регістри керування FPGA Timer0

Реєстрація переривань від таймеру відбувається наступним чином:

```
// Register the interrupt handler.
value = request_irq (INTERVAL_TIMER_IRQi, (irq_handler_t) irq_handler, IRQF_SHARED,
                     "timer irq handler", (void *) (irq_handler));
```

де INTERVAL_TIMER_IRQi – ідентифікатор переривання від таймеру, а саме 72;

- ✓ (irq_handler_t) irq_handler –вказівник фунції, яка є обробником переривання;
- ✓ IRQF_SHARED – флаг, що дозволяє розповсюджувати запити на переривання між кількома пристроями;
- ✓ “timer irq handler” – скорочена назва для пристрою, яку можна побачити в /proc/interrupt списку;
- ✓ (void *) (irq_handler) – ідентифікатор пристрою. Для цього параметру, зазвичай, встановлюють значення NULL, але якщо є необхідність використовувати IRQ shared, то ця комірка має містити вказівник (будь-який, головне, щоб при виклику free_irq(), передавався саме цей вказівник).

ВАЖЛИВО!!! Спершу скидайте у 0 комірку TO в обробнику переривань.

Видаляти вказівник на програму обробки переривання необхідно таким чином:

```
free_irq (INTERVAL_TIMER_IRQi, (void*) irq_handler);
```

Тепер розглянемо реалізацію інтерфейсу символьного пристрою в Linux ядрі на прикладі простого модуля.

Драйвери символьних пристроїв — це модулі ядра, які забезпечують файловий інтерфейс для програм на рівні користувача. Коли драйвер

символьного пристрою вставляється в ядро Linux, особливий тип файлу, пов'язаного з драйвером створюється, зазвичай, у папці файлової системи /dev. Наприклад, якщо драйвер називається chardev, то пов’язаний з ним файл буде /dev/chardev. Програма на рівні користувача може читати або записувати цей файл для зв’язку з драйвером.

Розглянемо драйвер символьного пристрою, який може використовуватися користувачем для друку різного рядку тексту у вікні терміналу.

Приклад коду драйверу символьного пристрою наведено на рис. 93а. Мається на увазі нескладний приклад, щоб проілюструвати, як повинен бути структурований код драйвера. Якщо користувач читає з цього драйвера, він просто відповідає з текстом Hello від chardev. Один із способів читання з драйвера - виконати команду Linux: **cat /dev/chardev**. Цей драйвер підтримує як читання, так і запис. Користувач може змінити рядок символів, написавши йому. Одним із способів запису в драйвер є використання команди Linux:

echo -n «Нове повідомлення" > /dev/chardev.

Драйвер створюється шляхом оголошення ряду змінних і структур даних та виклику кількох функцій взаємодії з ядром Linux. Нижче пояснюються важливі рядки коду з рис. 93а та рис. 93б. Рядки з 1 по 7 включають певні файли заголовків, які необхідні для драйверів символьних пристроїв. Рядки з 10 по 13 містять оголошення прототипів для функцій device_open, device_release, device_read і device_write. Це функції, які будуть викликатися ядром Linux, коли програма користувача виконує операції відкриття, закриття, читання або запису, відповідно, над файлом /dev/chardev. Рядки з 18 по 20 оголошують деякі спеціальні типи змінних. Усі драйвери символьних пристроїв повинні оголошувати ці змінні, що мають типи dev_t, cdev і class. Вони ініціалізуються в рядках з 32 по 56 у функції start_chardev, яка викликає ряд функцій ядра.

Показано перший крок для ініціалізації цих змінних у рядку 36 — отримати номер пристрою шляхом виклику функції ядра alloc_chrdev_region. Потім, структура класу створюється в рядку 40 викликом class_create, а структура cdev ініціалізується в рядку 43 за допомогою виклику функції cdev_alloc. Рядок 44 ініціалізує структуру cdev з вказівником на структуру fops, який визначає імена функцій, що відкривають, закривають, зчитують і записують у драйвер символьного пристрою. Нарешті, рядки 48 і 52 додають пристрій cdev і відповідний клас до ядра. Рядок 53 ініціалізує масив chardev_msg із текстовим повідомленням за замовчуванням. Функція start_chardev виконується, коли є драйвер символьного пристрою ініціалізовано

в ядрі Linux. Коли цей драйвер видалено з ядра, викликається функція stop_chardev, яка показана у рядках з 57 по 63.

Рядки з 65 по 75 містять код для device_open і device_release. Для більшості драйверів символьних пристроїв нічого не потрібно робити в цих функціях, і тому код просто повертає 0.

```
1  #include <linux/module.h>
2  #include <linux/kernel.h>
3  #include <linux/fs.h>
4  #include <linux/cdev.h>
5  #include <linux/device.h>
6  #include <asm/io.h>
7  #include <asm/uaccess.h>
8
9  /* Kernel character device driver /dev/chardev. */
10 static int device_open (struct inode *, struct file *);
11 static int device_release (struct inode *, struct file *);
12 static ssize_t device_read (struct file *, char *, size_t, loff_t *);
13 static ssize_t device_write(struct file *, const char *, size_t, loff_t *);
14
15 #define SUCCESS 0
16 #define DEVICE_NAME "chardev"
17
18 static dev_t dev_no = 0;
19 static struct cdev *chardev_cdev = NULL;
20 static struct class *chardev_class = NULL;
21 #define MAX_SIZE 256    // assume that no message longer than this will be used
22 static char chardev_msg[MAX_SIZE];  // the string that can be read or written
23
24 static struct file_operations fops = {
25     .owner = THIS_MODULE,
26     .open = device_open,
27     .release = device_release,
28     .read = device_read,
29     .write = device_write
30 };
31
32 static int __init start_chardev(void)
33 {
34     int err = 0;
35     /* Get a device number. Get one minor number (0) */
36     if ((err = alloc_chrdev_region (&dev_no, 0, 1, DEVICE_NAME)) < 0) {
37         printk (KERN_ERR "chardev: alloc_chrdev_region() error %d\n", err);
38         return err;
39     }
40     chardev_class = class_create (THIS_MODULE, DEVICE_NAME);
41
42     // Allocate and initialize the char device
43     chardev_cdev = cdev_alloc ();
44     chardev_cdev->ops = &fops;
45     chardev_cdev->owner = THIS_MODULE;
46
47     // Add the character device to the kernel
48     if ((err = cdev_add (chardev_cdev, dev_no, 1)) < 0) {
49         printk (KERN_ERR "chardev: cdev_add() error %d\n", err);
50         return err;
51     }
52     device_create (chardev_class, NULL, dev_no, NULL, DEVICE_NAME );
53     strcpy (chardev_msg, "Hello from chardev\n");
54
55     return 0;
56 }
```

Рис. 93а. Приклад драйвера символьного пристрою (Частина 1)

```
57  static void __exit stop_chardev(void)
58  {
59      device_destroy (chardev_class, dev_no);
60      cdev_del (chardev_cdev);
61      class_destroy (chardev_class);
62      unregister_chrdev_region (dev_no, 1);
63  }
64
65  /* Called when a process opens chardev */
66  static int device_open(struct inode *inode, struct file *file)
67  {
68      return SUCCESS;
69  }
70
71  /* Called when a process closes chardev */
72  static int device_release(struct inode *inode, struct file *file)
73  {
74      return 0;
75  }
76
77  /* Called when a process reads from chardev. Provides character data from
78   * chardev_msg. Returns, and sets *offset to, the number of bytes read. */
79  static ssize_t device_read(struct file *filp, char *buffer, size_t length,
        loff_t *offset)
80  {
81      size_t bytes;
82      bytes = strlen (chardev_msg) - (*offset);    // how many bytes not yet sent?
83      bytes = bytes > length ? length : bytes;     // too much to send at once?
84
85      if (bytes)
86          (void) copy_to_user (buffer, &chardev_msg[*offset], bytes);
87      *offset = bytes;     // keep track of number of bytes sent to the user
88      return bytes;
89  }
90
91  /* Called when a process writes to chardev. Stores the data received into
92   * chardev_msg, and returns the number of bytes stored. */
93  static ssize_t device_write(struct file *filp, const char *buffer, size_t
        length, loff_t *offset)
94  {
95      size_t bytes;
96      bytes = length;
97
98      if (bytes > MAX_SIZE - 1)    // can copy all at once, or not?
99          bytes = MAX_SIZE - 1;
100     (void) copy_from_user (chardev_msg, buffer, bytes);
101     chardev_msg[bytes] = '\0';  // NULL terminate
102     // Note: we do NOT update *offset; we keep the last MAX_SIZE or fewer bytes
103     return bytes;
104 }
105
106 MODULE_LICENSE("GPL");
107 module_init (start_chardev);
108 module_exit (stop_chardev);
```

Рис. 93б. Приклад драйверу символьного пристрою (Частина 2)

Рядки з 77 по 89 на Рис.2 показують функцію device_read. Перший аргумент цієї функції, filp, не використовується в цьому прикладі. Другий аргумент, буфер, використовується для передачі символьних даних від драйверу до процесу на рівні користувача, який читає з /dev/chardev. Аргумент length визначає максимальну кількість байтів, яку можна зберігати в буфері. Рядки з 82 по 83 визначають, скільки даних можна надіслати назад до ядра. Якщо length більше bytes, то може все повідомлення бути відправлено відразу. В іншому випадку, можна надіслати кількість символів довжиною length. Значення bytes обчислюється в рядку 82. Коли користувач відкриває файл /dev/chardev, а потім читає з файлу, *offset буде 0. Таким чином, змінна bytes встановлює значення довжини рядка символів, яка повертається під час зчитування драйвера.

У середовищі Linux ядро не має безпосереднього доступу до адрес пам'яті або змінних у програмі на рівні користувача. Так само, код на рівні користувача не може отримати доступ до адрес пам'яті в самому ядрі. Ядро надає спеціальні функції для передачі даних між собою та програмами на рівні користувача. Одна з таких функцій, copy_to_user, викликається в рядку 86 (рис. 93б). Вона копіює символьні дані з ядра до програми рівню користувача через буфер. Змінній offset присвоюється значення змінної bytes в рядку 87. Змінна offset, яка пов'язане з /dev/chardev зберігається в ядрі, доки файл залишається відкритим. У рядку 88 функція device_read повертає кількість байтів, що зберігаються в buffer. Типова програма на рівні користувача, яка читає з драйвера пристрою (наприклад, cat /dev/chardev) буде читати з файлу, доки device_read не поверне 0, який вказує на кінець файлу. У нашому прикладі це працює наступним чином. Перший раз, коли device_read викликається, вона копіює рядок символів назад у buffer і повертає довжину рядка. Але наступний виклик device_read нічого не копіює в buffer і повертає 0. Цей механізм спрощується за рахунок використання змінної offset.

Рядки з 91 по 104 показують функцію device_write. Перший аргумент цієї функції, filp, не використовується. Другий аргумент, buffer, використовується для завантаження символьних даних з програми користувача до драйверу. Аргумент length вказує кількість даних, які потрібно передати. Аргумент offset не використовується в даному прикладі. Дані, передані від користувача, зберігаються в chardev_msg, перезаписуючи попередні повідомлення. Рядок 98 перевіряє наявність переповнення, щоб у разі потреби кількість даних можна було зменшити. У рядку 100 ядро функції copy_from_user викликається, щоб отримати дані користувача та скопіювати їх у chardev_msg.

Функція device_write повертає кількість байтів, скопійованих у chardev_msg.

Типова програма на рівні користувача (наприклад, echo -n "Нове повідомлення" > /dev/chardev) продовжуватиме викликати device_write, поки драйвер не отримає необхідну кількість байтів. У нашому прикладі, кожний виклик device_write перезаписує дані, що зберігаються в chardev_msg.

Також можна написати програму у просторі користувача, яка буде читати і записувати данні в файл. На рис. 94 представлено приклад такої програми.

```c
#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <errno.h>
#include <fcntl.h>
#include <unistd.h>

#define BYTES 256   // max # of bytes to read from /dev/chardev

volatile sig_atomic_t stop;
void catchSIGINT(int signum){
    stop = 1;
}

/* This code uses the character device driver /dev/chardev. The code reads the
 * default message from the driver and then prints it. After this the code
 * changes the message in a loop by writing to the driver, and prints each new
 * message. The program exits if it receives a kill signal (for example, ^C
 * typed on stdin). */
int main(int argc, char *argv[]){
    int chardev_FD;              // file descriptor
    char chardev_buffer[BYTES];  // buffer for chardev character data
    int ret_val, chars_read;     // number of characters read
    char new_msg[128];           // space for the new message
    int i_msg;

    // catch SIGINT from ^C, instead of having it abruptly close this program
    signal(SIGINT, catchSIGINT);

    // Open the character device driver for read/write
    if ((chardev_FD = open("/dev/chardev", O_RDWR)) == -1) {
        printf("Error opening /dev/chardev: %s\n", strerror(errno));
        return -1;
    }

    i_msg = 0;
    while (!stop) {
        chars_read = 0;
        while ((ret_val = read (chardev_FD, chardev_buffer, BYTES)) != 0)
            chars_read += ret_val;  // read the driver until EOF
        chardev_buffer[chars_read] = '\0';  // NULL terminate
        printf ("%s", chardev_buffer);

        sprintf (new_msg, "New message %d\n", i_msg);
        i_msg++;
        write (chardev_FD, new_msg, strlen(new_msg));

        sleep (1);
    }

    close (chardev_FD);
    return 0;
}
```

Рис. 94. Приклад програми яка взаємодіє з /dev/chardev

3. Самостійна робота

Завдання 1: Використовуючи напрацювання минулої лабораторної роботи, необхідно створити програму у просторі користувача, яка відображає на дисплеї рухоме повідомлення Intel Soc FPGA, яка на семисегментних індикаторах виглядає наступним чином. Літери повинні прокручуватись, як в рекламній стрічці.

Завдання 2: Написати драйвер ядра, який являє собою годинник реального часу. Відображати час на семисегментних дисплеях, якщо є, або у вікні терміналу. Час має відображатися у форматі MM:SS:DD, де MM — хвилини, SS — це секунди, а DD — соті частки секунди. Для відстеження часу слід використовувати апаратний модуль таймера.

Завдання 3: Написати драйвер символьного пристрою, який відображає на семисегментному індикаторі записані в нього цифри.

4. Контрольні запитання.

1. Яке призначення регістрів інтервального таймеру?

2. За яких умов інтервальний таймер генерує запит на переривання процесорного ядра ARM ?

3. Де у середовищі Linux зберігаються драйвери до символьного дісплею?

4. Які функції використовуються у програмному забезпеченні для взаємодії з символьним дисплеєм?

Список використаної літератури

1. Prakash Rashinkar, Peter Paterson, Leena Singh. System-on-a-Chip Verification. Methodology and Techniques. Kluwer Academic Publishers, 2002. – 393 p.
2. Dr. David J Greaves. System on Chip. Design and Modelling. Lecture Notes. University of Cambridge. 2011. – 131 p.
3. https://fpgacademy.org/tutorials.html
4. https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=&No=656#contents
5. https://people.ece.cornell.edu/land/courses/ece5760/

Список рекомендованої літератури

6. В.В. Амосов. Схемотехника и средства проектирования цифровых устройств. СПб.: БХВ-Петербург, 2007. – 560 с.
7. В.М. Рябенький, В.Я. Жуйков, В.Д. Гулий. Цифрова схемотехніка. Навчальний посібник. – Львів; «Новий Світ – 2000», 2009. – 736 с.
8. В. Немудров, Г. Мартин. Системы-на-кристалле. Проектирование и развитие. – М.; Техносфера, 2004. – 216 с.