Co-funded by the
Tempus Programme
of the European Union

DesIRE

Г.В. Табунщик, Т.І. Каплієнко,
О.О. Каплієнко, Д. Ван Мероде

ВЕРИФІКАЦІЯ ТА ВАЛІДАЦІЯ ЦИФРОВИХ СИСТЕМ УПРАВЛІННЯ

НАВЧАЛЬНИЙ ПОСІБНИК

Запоріжжя, 2017

УДК 004.41 (075.8)
ББК 32.972-02 я73
Т-12

Рекомендовано як навчальний посібник для студентів спеціальностей «Інженерія програмного забезпечення», «Комп'ютерні науки та інформаційні технології» та «Електроенергетика, електротехніка та електромеханіка» на засіданні Вченої Ради Запорізького національного технічного університету від 27.03.2017 р., протокол №8.

Р е ц е н з е н т и :

С. І. Гоменюк - *доктор технічних наук, професор, декан математичного факультету Запорізького національного університету*
В. О.Філатов *- доктор технічних наук, професор, завідувач кафедри штучного інтелекту Харківського національного університету радіоелектроніки*
О.Ф. Тарасов *- доктор технічних наук, професор, завідувач кафедри комп'ютерних інформаційних технологій Донбаської державної машинобудівної академії*

Табунщик Г. В.

Т-12 Верифікація та валідація цифрових систем управління/ Г. В. Табунщик, Т.І. Каплієнко, О.О. Каплієнко, Д. Ван Мероде − Запоріжжя: Дике Поле, 2017. – 150 с.
ISBN 978-617-7433-22-3

Навчальний посібник містить підходи та засоби аналізу, верифікації та валідації цифрових систем керування.

Видання призначене для студентів спеціальностей комп'ютерні науки та програмна інженерія вищих навчальних закладів, а також може бути корисним для студентів спеціальностей електричні апарати та радіотехніка та телекомунікації. Практичні завдання можуть бути використанні аспірантами, науковими та педагогічними працівниками, фахівцями-практиками.

Навчальний посібник видано за підтримки проекту Tempus 544091-TEMPUS-1-2013-1-BE-TEMPUS-JPCR «Розробка курсів з вбудованих систем з використанням інноваційних віртуальних підходів для інтеграції науки, освіти та промисловості в Україні, Грузії, Вірменії» [DESIRE].

Проект фінансується при підтримці Європейської комісії. Зміст матеріалу відображає думку авторів та Європейська комісія не несе відповідальності за використання інформації що міститься в навчальному посібнику.

УДК 004.41 (075.8)
ББК 32.972-02 я73

ISBN 978-617-7433-22-3

ЗМІСТ

ВСТУП

Світ поступово переходить у нову фазу розвитку технологій – Інтернет речей (*IoT*). Промисловий інтернет речей це мережа фізичних об'єктів що взаємодіють між собой для досягнення кінцевої мети.

Концептуально до тематики інтернету речей входять наступні розділи [3]:

- сенсори та інтернет речей: енергозбереження сенсорів, безпечність вимірів, взаємодію з сенсорами;
- мережева взаємодія: *IP* прот*о*кол, дротові і бездротові мережі, *Bluetooth Low Energy, ZigBee, WiFi, LTE*;
- стандартизація: технологічні стандарти та аспекти регулювання. Серед технологічних стандартів особливе місце займають питання агрегації даних;
- аналітична обробка даних: дескриптивна аналітика, предикативна аналітика та рекомендаційні системи. *IoT* особливу увагу приділяє системам реального часу та обробці подій ;
- засоби взаємодії містять як міжмашину взаємодію (*M2M*) так і чоловіко-машину взаємодію.

Ця галузь характеризується постійним розвитком та вдосконаленням багатьох методологій, що містять сукупність моделей, методів та програмного забезпечення. Важливим моментом при розробці систем керування з використанням мінікомпьютених систем є їх обмеженість в системних ресурсах, що призводить до посилення вимог до кінцевих продуктів на їх основі. Що призводить до підвищення вимог до етапа верифікації.

Тому дуже важливо навчити студенів вирішувати завдання верифікації та валідації для сучасних цифрових систем керування, що використовуються для застовувань в галузі *IoT*.

У цьому навчальному посібнику автори постарались найбільш органічно викласти матеріал, щоб підвищити рівень його засвоєння студентами напрямів «Програмна інженерія», «Комп'ютерні науки» та «Електромеханіка». Викладений матеріал був апробований під час читання курсів «Інженерія якості «що вивчається студентами напряму «Програмна інженерія», «Верифікація та валідація цифрових систем керування», що вивчається студентами напряму підготовки «Комп'ютерні науки та інформаційні технології» та «Автоматизовані електромеханічні комплекси та системи», що вивчається студентами спеціальності «Електроенергетика, електротехніка та електромеханіка».

Метою книги є надання систематичного огляду до процесів розроблення вбудованих систем, з*о*крема верифікації і валідації інформаційних систем. Метою міжнародного проекту Tempus *544091-TEMPUS-1-2013-1-BE-TEMPUS-JPCR [DesIRE]* «Розробка курсів з вбудованих систем з використанням інноваційних віртуальних підходів для інтеграції науки, освіти та промисловості в Україні, Грузії, Вірменії» було втілити практично-орієнтований підхід до навчання, тому значна частина навчального посібника присвячена практичним аспектам. Структурно підручник містить дві основні частини. Перша частина присвячена теоретичним основам верифікації інформаційних систем. Друга частина має прикладний характер, та містить детальний опис роботи з *IDE Vivado* з використанням плати *Basys3 [2, 7,9]*.

Матеріал рукопису побудований на основі матеріалів лекцій, практичних та лабораторних робіт, що викладаються в Запорізькому національному технічному університеті. З*о*крема, авторами використано матеріали, отримані при виконанні проекту Tempus 544091-TEMPUS-1-2013-1-BE-TEMPUS-JPCR [DesIRE] «Розробка курсів з вбудованих систем з використанням інноваційних віртуальних підходів для інтеграції науки, освіти та промисловості в Україні, Грузії, Вірменії» та держбюджетної теми

№ 0117U000615 «Інформаційна система діагностування мінікомпьютерних систем в багат*ок*омпонентному середовищі».

Кожний розділ закінчується контрольними запитаннями, що можуть бути використані як для самоперевірки, так і при підготовці лабораторних, практичних та контрольних завдань.

ЧАСТИНА 1. ТЕОРЕТИЧНІ ОСНОВИ ВЕРИФІКАЦІЇ ЦИФРОВИХ СИСТЕМ

1 ОСНОВИ ВЕРИФІКАЦІЇ ВБУДОВАНИХ СИСТЕМ

Верифікація системи в найзагальнішому сенсі – це перевірка відповідності між вимогами до системи і властивостями працюючої системи [10,11].

1.1 Методи верифікації вбудованих систем

Розрізняють такі методи верифікації як експертиза, статичний аналіз, формальні методи, динамічні методи, синтетичні методи (рис. 1.1).

Рисунок 1.1 – Методи верифікації

Зазвичай серед експертиз виділяють такі: організаційні експертизи (management review), технічні експертизи (technical review), наскрізний контроль (walkthrough), інспекції (inspection) та аудити (audit). З середини 1990-х активно розвиваються методи оцінки архітектури програмного забезпечення (ПЗ) на основі сценаріїв (scenario based software architecture evaluation), які

зазвичай не співвідносяться з «традиційними» експертизами. Від інших методів верифікації експертизу відрізняє можливість виконувати її, використовуючи тільки самі артефакти життєвого циклу, а не їх моделі (як у формальних методах) або результати роботи (як в динамічних). Експертиза застосовується до будь-яких характеристик ПЗ і до будь-яких артефактів життєвого циклу на всіх етапах життєвого циклу, хоча для різних цілей можуть використовуватися різні її види. Вона дозволяє виявляти широ*к*ий спектр помило*к*, причому робити це на етапі проектування відповідного артефакту, тим самим мінімізуючи час існування дефекту і його наслідки для якості похідних артефактів. У той же час експертиза не може бути автоматизована і вимагає активної участі людей.

Ефективність експертизи істотно залежить від досвіду та мотивації її учасників, організації процесу, а також від забезпечення коректної взаємодії між різними учасниками. Це накладає додаткові обмеження на розподіл ресурсів у проекті і може призводити до конфліктів між розробниками, якщо керівництво проекту звертає мало уваги на комунікативні аспекти проведення експертиз.

Кожний вид експертизи має свої властивості:

- технічна експертиза (*technical review*) – це систематичний аналіз артефактів проекту кваліфікованими фахівцями для оцінки їх внутрішньої узгодженості, точності, повноти, відповідності стандартам і прийнятим в організації процесів, а також відповідності один одному і загальним завданням проекту;

- наскрізний контроль (*walkthrough*) - метод експертизи, в рамках якого один з членів команди перевірки надає її учасникам послідовно всі характеристики артефакту, що перевіряється, і вони аналізують цей артефакт, ставлячи питання, вносячи зауваження, відзначаючи можливі помилки, порушення стандартів і інші дефекти;

- інспекція (*software inspection*) – це послідовне вивчення характеристик артефакту, зазвичай слідує деякому плану, з метою виявлення в ньому помило*к* і дефектів;

- аудит (*audit*) - аналіз артефактів і процесів життєвого циклу, що виконується людьми, які не входять до команду

проекту, для оцінки відповідності цих артефактів і процесів завданням проекту, укладеним контрактом, загальним стандартам, один одному і ін.

Статичний аналіз використовується для перевірки формалізованих правил коректної побудови артефактів ІС і пошуку дефектів за визначеними шаблонами.

Такий аналіз добре автоматизується і може бути практично повністю по*ок*ладений на програмний інструментарій, хоча іноді необхідно вручну визначити, наприклад, прийняті в проекті стандарти кодування. Однак він застосовується лише до коду або до певних форматів уявлення проектних артефактів ІС, і здатний виявляти тільки обмежену кількість типів помил*ок*. Проблемою цих методів є або велика надмірність, при використанні методів строго аналізу, або вірогідність пропустити помилки, при використані методів що ще генерують повідомлення про помилку. Інструменти автоматичної верифікації на основі статичного аналізу застосовуються досить шир*ок*о, оскільки не вимагають спеціальної підготовки і досить зручні у використанні. Більшість технік статичної перевірки коректності програм, які довели свою ефективність на практиці, рано чи пізно стають частиною компіляторів або навіть перетворюються в семантичні правила мов програмування.

Формальні методи верифікації ІС для аналізу властивостей формальних моделей вимог, поведінки ІС і її оточення. Аналіз формальних моделей виконується за допомогою специфічних технік, таких як дедуктивний аналіз (*theorem proving*), перевірка моделей (*model checking*) або абстрактна інтерпретація (*abstract interpretation*). Формальні методи застосовні тільки до тих властивостей, які виражені формально в рамках визначеної математичної моделі, а також до тих артефактів, для яких можна побудувати адекватну формальну модель. Відповідно, для використання таких методів в проекті необхідно затратити значні зусилля на побудову формальних моделей. До того ж, побудувати такі моделі і провести їх аналіз можуть тільки відповідні фахівці. Побудову формальних моделей достатньо складно автоматизувати, для цього завжди необхідна людина. Аналіз їх властивостей значною мірою може бути автоматизований, і зараз

вже є інструменти, здатні аналізувати формальні моделі промислового рівня складності, однак щоб ефективно користуватися ними часто теж потрібно дуже специфічний набір навич*ок* і знань (у специфічних розділах математичної логіки і алгебри). Тим не менш, у ряді областей, де наслідки помилки в системі можуть виявитися надзвичайно дорогими, формальні методи верифікації активно використовуються. Вони здатні виявляти складні помилки, що практично не виявляються за допомогою експертиз або тестування. Крім того, формалізація вимог і проектних рішень можлива тільки при їх глиб*ок*ому розумінні, і тому змушує провести ретельний аналіз цих артефактів, для чого часто необхідна спільна робота фахівців по формальним методам та експертів в предметній області.

На практиці, формальні методи верифікації набагато частіше застосовуються до апаратного забезпечення ніж до програмного. Це обумовлено більш вис*ок*ою вартістю помил*ок*, більшою однорідністю його структури, більш шир*ок*им багаторазовим використанням проектної інформації, а також більшою звичністю строгих обмежень і точних описів для інженерів.

Логіко-алгебраїчні моделі (*property-based models*), вони ж - логічні або алгебраїчні обчислення. При моделюванні ПЗ ІС модель такого типу описує деякий набір властивостей системи, що змінюється з часом, але не дає точного уявлення про те, за рахун*ок* чого змінюються ці властивості.

Здійснимі моделі (або операційні, *executable models*) характеризуються тим, що їх можна визначеним способом виконати, щоб простежити зміну властивостей модельованого ПЗ. Кожна модель, що виконується, є, по суті, програмою для деякої досить строго визначеної віртуальної машини.

Всі види здійсненних моделей можна вважати розширенням і узагальненням моделей на базі кінцевих автоматів.

Моделі проміжного типу мають риси як логіко-алгебраїчних, так і здійсненних моделей. Частина наведених вище прикладів може по ряду властивостей бути віднесена до обох цих класів.

Методи та інструменти перевірки моделей. Перевірка моделей (*model checking*) використовується для перевірки виконання набору властивостей, записаних у вигляді тверджень визначеного логіко-алгебраіческого обчислення за здійсненною моделлю що моделює певні проектні рішення. Найчастіше для перевірки опису властивостей використовується деяка тимчасова логіка або -числення, а в якості моделі, властивості якої перевіряються, виступає кінцевий автомат, стани якого відповідають наборам значень елементарних формул у властивостях, що перевіряється, зазвичай він називається моделлю Кріпке. Перевірку моделі виконує спеціалізований інструмент, який або підтверджує, що модель дійсно володіє заданими властивостями, або видає сценарій її роботи, в кінці якого ці властивості порушуються, або не може прийти до певного вердикту, оскільки аналіз моделі вимагає занадто великих ресурсів. Властивості, що перевіряються, зазвичай поділяють на властивості безпеки (*safety properties*), що означають, що щось небажане при будь-якому варіанті роботи системи ніколи не трапляється, і властивості живучості (*liveness properties*), що означають, навпаки, що щось бажане рано чи пізно відбудеться. Іноді додатково виділяють властивості стабільності (або збереження, *persistence properties*) - при будь-якому сценарії роботи системи задане твердження в деякий момент стає істинним і з тих пір залишається виконаним, і властивості відповідності (*fairness properties*) - деяке твердження при будь-якому сценарії роботи буде виконано в нескінченній множині моментів часу. Ті чи інші властивості відповідності, часто є вихідними припущеннями, при виконанні яких потрібно перевірити властивості безпеки або живучості.

Методи та інструменти перевірки узгодженості. При перевірці узгодженості аналізується відповідність між двома здійснимими моделями, одна з яких моделює артефакт, що перевіряється, зазвичай проект або реальну роботу системи (її компонента), а друга - перевіряються властивості. Перевіряються властивості в цьому випадку - це вимоги до поведінки системи або її компонента, представлені у вигляді узагальненого автомата (системи переходів, мережі Петрі та ін.), Всі сценарії роботи якого

оголошуються правильними. В цьому випадку зазвичай перевіряється, що всі можливі сценарії поведінки реалізації можливі також і в специфікації. Іноді встановлюється їх еквівалентність, тобто додатково перевіряється, що всі сценарії поведінки специфікації є і у реалізації. Більшість методів та інструментів перевірки узгодженості використовують для цього тестування, і тому ставляться до синтетичних методам верифікації – до тестування на основі моделей.

Динамічні методи верифікації, в рамках яких аналіз і оцінка властивостей програмної системи виконуються за результатами її реальної роботи або роботи деяких її моделей і прототипів. Прикладами такого роду методів є звичайне тестування або імітаційне тестування, моніторинг. Для застосування динамічних методів необхідно мати працюючу систему або хоча б деякі її компоненти, або ж їхні прототипи, тому зазвичай вони не використовуються на перших стадіях розробки. З допомогою цих методів можна контролювати характеристики роботи ІС в її реальному оточенні, коли інші підходи використати неможливо. Методи динамічної верифікації дозволяють виявляти тільки помилки, що проявляються при його роботі. Застосування динамічних методів верифікації зазвичай вимагає додаткової кваліфікації – створення тестів, розробка тестової системи, що дозволяє їх виконувати або системи моніторингу, що дозволяє контролювати певні характеристики поведінки ІС. Але системи тестування або моніторингу можуть бути зроблені один раз і використовуватися багаторазово для широ*к*их класів ІС, лише самі тести необхідно готувати заново для кожної тестуємої ІС. У той же час підготовка тестів на ранніх етапах створення ІС дозволяє виявити безліч дефектів в описі вимог і проектних д*ок*ументах - фактично, розробники тестів змушені в ході своєї діяльності виконувати експертизу артефактів, які є основою для тестів. Створення тестових наборів, які дозволять отримати адекватну оцінку якості складної системи, є досить трудомістким завданням. Оскільки тестування це досить трудомісткий процес, тому зазвичай використовуються не надто надійні, але досить дешеві техніки, такі як (нестроге) вірогідне тестування, при якому тестові дані генеруються випадковим чином, або ж тестування на основі

найпростіших сценаріїв використання, що перевіряють лише найбільш прості ситуації.

В *ок*ремі області виділилися динамічні методи, що використовують елементи формальних, тестування на основі моделей (*model-based testing, model driven testing*) і моніторинг формальних властивостей (*runtime verification, passive testing*). Ряд інструментів побудови тестів істотно використовує як формалізацію деяких властивостей ПЗ, так і статичний аналіз коду. Загальна ідея таких методів міститься в наступному - поєднувати переваги основних підходів до верифікації для балансування їх недоліків.

Тестування на основі моделей (*model based testing*) використовує для побудови тестів формальні моделі вимог до ІС. Як критерії повноти тестування, так інші критерії будуються на основі інформації, що міститься в цих моделях. Отримані в результаті тести зазвичай слабко пов'язані зі специфічними особливостями коду тестованої системи, але містять репрезентативну вибірку ситуацій з точки зору вихідної моделі.

В даний час методи тестування на основі моделей використовують такі типи моделей і технік як методи перевірки узгодженості автоматів і систем переходів. Такі методи відносяться до одного з трьох типів, залежно від моделей, що використовуються.

1.2 Моделі відмов апаратного забезпечення

З розвитком технологій складність та потужність апаратних засобів збільшується [6-7]. Модель відмов апаратного забезпечення повинна визначити цільову функцію для тестування та аналізу відмов. Крім того, ефективність моделі відносно фактичних відмов визначаються експериментально. В більшості випадків несправності цифрових систем можна розділити на три групи: помилки проектної документації, помилки виробництва та експлуатаційні відмови. Помилки проектування виникають завдяки людському фактору або помилкам програмного забезпечення САПР (симулятори, генератори та генераторів

шарів) і відбуваються в процесі проектування. Ці відмови не пов'язані безпосередньо з процесом тестування [6,7]. Відмови, що обумовлені недосконалістю процесів виробництва, призводить до дефектів самого обладнання (пропущені при металізації контактні вікна або ділянки *ок*сиду, паразитичні транзистори, пробій *ок*сиду (в МОП-структурах) і т.п.). Фізичні відмови також називаються дефект-орієнтованими відмовами (*defect-oriented faults*). Експлуатаційні або логічні відмови відбуваються через відхилення в умовах експлуатації вбудованих систем. Прикладом таких відхилень виступають електромагнітні перешкоди, помилки оператора, вібрації.

Несправність є моделлю, яка є ефектом фізичного дефекту на логічному або функціональному рівні. Зазначимо, що кілька різних дефектів можуть представлятися однієї і тієї ж несправністю (має місце відношення "багато до одного"). З іншого б*ок*у, одній фізичній дефекту іноді може відповідати кілька несправностей (відношення "один до багатьох").

Апаратні несправності класифікуються як константні відмови, несправності типу замикання, дефекти обриву ланцюга, збоїв пам'яті та інші.

Крім того їх можна класифікувати як структурні та функціональні. Ті несправності, що визначаються на структурній моделі системи, називаються структурними несправностями. Їх ефект, як правило, зводиться до зміни з'єднань компонент. Функціональні несправності визначаються на функціональної моделі системи. Наприклад, ефект функціональної несправності може проявлятися у зміні функції, реалізованої компоненти системи або оператором мови опису апаратури.

Типовими несправностями сполучень компонентів системи є обрив (open) і замикання (short). Обрив відповідає порушенню з'єднання компонентів схеми. Причиною порушення з'єднання може бути нестача або відсутність провідного матеріалу, наприклад, в металевому провіднику. З іншого б*ок*у, відсутність з'єднання може виникнути внаслід*ок* наявності зайвих частин*ок* діелектрика, наприклад, між провідними шарами. Замикання утворюється в результаті з'єднання ліній схеми, які в справній системі ізольовані один від одного. Воно може бути викликано

наявністю зайвих провідних частин*ок* між провідниками, пробоєм *ок*сиду в МОП-структурах, який утворює з'єднання з деяким невеликим, але необов'язково нульовим опором і т.п.

Константні відмови (Stuck-at fault). Для визначення константних несправностей використовується структурна модель у вигляді логічної схеми. Вважається, що одиночна константна несправність (single stack-at fault) діє тільки на з'єднання між вентилями (при цьому логічні елементи функціонують правильно). Кожна лінія схеми може мати два типи цих несправностей: константа 0 і константа 1 (sa-0, sa-1). Отже, константна несправність фіксує на даній лінії постійне значення сигналу 0 або 1 (sa-0, sa-1), незалежно від значення подається на неї сигналу. Часто такі несправності моделюють замикання лінії схеми на землю (sa-0) або джерело живлення (sa-1).

Ця модель може використовуватися для генерації тестів незалежно від її адекватності реальних фізичних дефектів. Відзначимо, що для деяких технологій (наприклад, ТТЛ) адекватність цієї моделі досить вис*ок*а, для інших (КМОП) нижче. Але в цілому, дана модель надзвичайно корисна в силу своєї виняткової простоти і задовільної адекватності, і тому використовується в якості базової для багатьох методів моделювання несправностей і генерації тестів.

Несправності типу замикання мають місце в тому випадку, коли відбувається з'єднання двох або більше ліній схеми і утворюється "дротова логіка" (wired logic) в місці виниклої електричного зв'язку. Кратні замикання (з'єднання більше двох ліній) виникають звичайно на зовнішніх входах ІС. В даний час число дефектів, провідних до замикань, збільшується внаслід*ок* зменшення розмірів схем і збільшення щільності вентилів у кристалі. Очевидно, що число простих замикань (між двома лініями) в схемі, що має m ліній одно C_m^2. Однак, звичайно, не всі лінії схеми можуть замкнутися між собою. Тому реальна кількість можливих замикань істотно менше і залежить від підкладки (фізичної сусідства провідників).

Поведінка логічної схеми при замиканні залежить від технології виготовлення цієї схеми.

Слід зазначити, що дефекти замикання можуть викликати функціональні зміни в логічній схемі, які не можна уявити традиційними моделями - несправностями.

Деякі фізичні дефекти в КМОП технології не можуть бути представлені константними несправностями. Основна причина полягає в тому, що МОП комбінаційні схеми не завжди залишаються комбінаційними при деяких фізичних дефектах. Найбільш поширеними є такі види відмов у МОН технології: 1) обрив і замикання транзисторів; 2) обриви між ст*ок*ом, вит*ок*ом і затвором; 3) короткі замикання: витік - стік, затвор - стік, затвор - витік. Дефекти третьої групи зазвичай обумовлені пробоєм *ок*сиду.

У сучасних цифрових системах можливі ситуації, коли схема структурно і логічно коректна, але час поширення сигналів по деяким її шляхах перевищує допустиме для правильного функціонування значення. У таких випадках говорять про наявність несправності типу "затримка" (поширення сигналів). Такі несправності не можуть бути виявлені на низькій частоті роботи схеми. Метою тестування несправностей "затримка" є визначення правильного функціонування схеми на вис*ок*их тактових робочих частотах. При цьому виявляється, чи містить схема шляху поширення сигналів, які є занадто повільними або швидкими при зміні вхідних наборів. Для цих цілей використовуються дві основні моделі: затримка вентиля та затримка шляху.

Перша модель передбачає, що затримка обумовлена несправним логічним елементом. Слід зазначити, що час перемикання елемента, може залежати від напрямку зміни сигналу - його підйому або спаду. Це є недоліком даної моделі, оскільки не дозволяє у затримці одного елемента врахувати затримки інших елементів шляху. Очевидно, тут також повністю ігноруються затримки з'єднань між елементами.

Друга модель бере до уваги загальну затримку поширення сигналу від зовнішнього входу до зовнішнього виходу схеми. Хоча дана модель вимагає розгляду занадто багатьох можливих шляхів у схемі, вона більш реалістична, особливо для сучасних технологій, де затримки поширення сигналів мають місце

насамперед за рахун*ок* ліній з'єднань. Як правило, тестування затрим*ок* проводиться шляхом подачі на схему пари вхідних наборів на бажаної швидкості та спостереженні для кожного зміненого виходу швидкості його перемикання.

Часові несправності. При даних несправностях відбувається тимчасово поява неправильних сигналів в схемі. Вони зустрічаються в різних цифрових елементах, але найчастіше в мікросхемах пам'яті і мікропроцесорів. Серед цих несправностей розрізняють "короткочасні" (transient) "відмови" intermittent. Короткочасні несправності відбуваються, коли сигнали змінюють своє значення внаслід*ок*, наприклад, шумів. Такі несправності важко виявити та виправити. Тут важливо мінімізувати шуми і підвищити перешкодозахищеність схеми. Дані несправності можуть бути викликані, наприклад, флуктуаціями напруги, метастабільна тригерів або космічним випромінюванням.

Аварії є однією з причин відмов при експлуатації комп'ютерних систем. Серед них можна виділити несправності залежні від коду, які зустрічаються в мікросхемах пам'яті і мікропроцесорах.

Несправності рівня кристалу. В даний час нові технології дозволяють проектувати складні цифрові системи (ЦС) на одному кристалі (*System-on-ChIP - SOC*). При цьому проектування ЦС виконується за допомогою досить складних мов вис*ок*ого рівня опису апаратури (*HDL*), таких як *VHDL*, Verilog і SpecC. Тому актуальною є проблема верифікації та тестування складних ЦС, описаних за допомогою цих мов.

На логічному рівні моделювання, де ЦС представляється у вигляді логічної схеми, основною моделлю фізичних дефектів є константні несправності, які еквівалентні постійним сигналам 0 або 1 на лініях схеми. На відміну від логічного рівня моделей несправностей, де зазвичай можна встановити відповідність між фізичним дефектом провідників у кремнії і з'єднаннями в логічній схемі, на поведінковому рівні, як правило, важко встановити відповідність між описом ЦС на *HDL* і структурним описом. Один оператор *HDL* може відповідати сотням логічних вентилів, з'єднаних між собою. Тому необхідно розглядати функціональні моделі несправностей безпосередньо на мовних конструкціях

HDL. При цьому якість (або адекватність) функціональних моделей несправностей, як правило, перевіряється за допомогою логічного моделювання ЦС, яке визначає повноту тесту щодо одиночних константних несправностей схеми, реалізує ЦС. Тому даний підхід орієнтований швидше на досягнення вис*ок*ої повноти тесту для константних несправностей, а не виявлення помил*ок* в мовних конструкціях *HDL*. Більш того, при цьому ефективність тестової послідовності не може бути визначена безпосередньо на функціональному рівні. Тому в даний час для верифікації та тестування ЦС, описаних на *HDL*, застосовуються методи, запозичені з тестування програмного забезпечення.

1.3 Функціональне тестування апаратного забезпечення

Функціональне тестування з'явилося найпершим. Постійне збільшення складності виробів роблять процес підготовки функціонального тесту нескінченно довгим. Діагностування несправностей, виявлених у процесі функціонального тестування, може бути досить складним, що вимагає залучення кваліфікованих фахівців. Тому перед тестом системи в цілому часто здійснюється тестування на рівні *ок*ремих плат. Тестування плат може бути здійснено і на функціональному рівні, але поділ робить діагностування несправностей і підготовку тестів більш гнучкими. Швидко зростаюча складність інтегральних мікросхем викликає схожі проблеми із функціональним тестуванням на рівні плат, так само як і в системному тестуванні - довгий час підготовки тестів, неточне тестове п*ок*риття, слабка діагностика.

Наступний шир*ок*о поширений метод тестування - це внутрішнє-схемне тестування (*In-Circuit Test, ICT*). Цей метод дозволяє знаходити дефекти і помилки монтажу шляхом забезпечення прямого електричного доступу до компонентів на платі через адаптер. Внутрісхемне тестування ідеально підходило для *DIP*-компонентів і технології штирьового монтажу. Але у зв'язку з появою багатошарових друкованих плат і більш складних корпусів мікросхемтестовий доступ став сильно обмежений.

Технологія внутрисхемного тестування не може розвиватися також швидко, як мініатюризація розмірів компонентів і виробів.

Електронна індустрія передбачала ці проблеми заздалегідь, тому був розроблений метод периферійного сканування, закріплений стандартом *IEEE 1149.1*, який описує порт тестового доступу (*TAP - Test Access Port*) і архітектуру периферійного сканування. Метою створення даного стандарту було подолання недоліків інших методів тестування [4-5].

Дивлячись на еволюцію тестових методів можна зробити наступні спостереження: розробка тестопридатності виробів (*Design-For-Test, DFT*) стає все більш і більш необхідним доповненням функціонального тестування, дозволяючи зробити контроль більш повним і інформативним.

Спочатку, тестування було похідною процесу налагодження нової розробки і пошуку дефектів монтажу. Через зростання складності схем пристроїв керованість цими процесами могла бути підвищена тільки при роздільному їх проведенні. Виявлення і виправлення виробничих дефектів на стадії налагодження дослідних зразків стало необхідністю.

З ростом складності продукції багато виробників почали застосовувати багатоступеневу стратегію тестування, метою застосування якої є якомога більш раннє виявлення і виправлення помил*ок* виробничого процесу.

Перша версія стандарту *Boundary-Scan*, з'явилася на початку 1990 р*ок*у, і отримала ім'я, яке зберіглося і сьогодні - IEEE 1149.1. Стандарт цієї технології також називається *Test Access Port and Boundary-Scan Architecture* (порт тестового доступу та архітектура граничного сканування). Проект був розроблений міжнародною групою експертів, яка носила назву *JTAG* (*Joint Test Action Group* - об'єднана група розробки методів тестування).

Сама архітектура цифрового стандарту Boundary-Scan не відрізняється особливою складністю, на відміну від своїх можливостей. Відповідно до стандарту *IEEE 1149.1*, так звана *Boundary-Scan-IC*, повинна бути оснащена чотирма обов'язковими елементами:

- *TAP*-портом, який складають чотири обов'язкових сигналів, і п'ятий за рішення розробників безпосередньо самої

плати (*TCK* - контакт синхронізації роботи механізму Boundary-Scan; *TMS* - контакт вибору тестового режиму; *TDI* - контакт введення тестових даних);

- TDO - контакт виведення тестових даних (знаходиться в постійно в третьому стані, *о*крім режиму зсуву); RST - контакт асинхронного скидання стану TAP-контролера (може зовсім не бути присутньою)). TAP-контролер виступає одним з важливих елементів управління всією роботою технології Boundary-Scan;
- *IR* (*Instruction Register* від анг. регістр команд) - перша група регістрів, в якій обов'язковим за стандартом повинен бути присутнім хоча б один Регістр Команд (РК);
- *DR (Data Registers* від анг. - Регістри Даних) - друга група регістрів, відповідно до стандарту зобов'язана в себе містити як мінімум два регістри: Регістр Обходу (РО, також його іноді називають Шунт-Регістр), Регістр Boundary-Scan.

Такий мінімальний комплект елементів вимагає стандарт IEEE 1149.1. Інші регістри, які можуть доповнити групу, як IR, так DR на розсуд розробників плат, також допускаються створеним стандартом.

Для того, щоб отримати хороше тестове по*к*риття, немає необхідності в тому, щоб всі компоненти на платі мали *JTAG* - інтерфейс. Наприклад, багато бло*к*ів, складається з не сканованих компонентів (кластера), можуть тестуватися, незважаючи на відсутність прямого доступу для периферійного сканування. У дійсності, існують практичні приклади, коли здійснюється контроль і детальне тестування абсолютно всієї плати (включаючи пам'ять) за допомогою одного або двох компонентів, що підтримують периферійне сканування.

На рис. 1.2 зображена архітектура *Boundary-Scan*. На TAP-контролер подаються 2 (3) сигналу, за допомогою яких контролер встановлює відповідний режим роботи схеми. Сам TAP-контролер є автомат з кінцевим кількістю вершин.

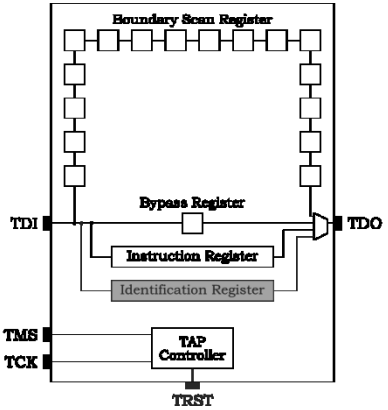


Рисунок 1.2 Архітектура *JTAG*

Сьогодні на світовому ринку в цій області лідирують чотири представники США і Європи, які поставляють програмно-апаратні комплекси (ASSET InterTech Inc. і CORELIS Inc - США; GOEPEL Electronic - Німеччина; *JTAG* Technologies - Нідерланди). Такі розробки називаються BS-тестери.

Використання *JTAG* і технології граничного сканування в мікросхемі, на платі або в пристрої додає вартість і збільшує час розроблення проекту. Але, все ж ці витрати легко *ок*упаються при проведенні тестування, яке забезпечується на кожній стадії циклу життя виробу. Крім безпосередньо граничного тестування, проектувальники використовують технологію *JTAG* для того, щоб виробляти самотестування (*BIST*) (у тих компонентах, де воно реалізовано) і завантажувати внутрішні значення в регістри пристрою або програмувати мікросхеми ПЗУ. Тести, які були розроблені та використані на етапі проектування, можуть бути передані виробництву, для того щоб забезпечити додаткове зниження вартості і часу на перевірку виробів при вихідному контролі.

1.4 Контрольні запитання

1. Які існують методи верифікації?
2. Від чого залежить ефективність експертизи?
3. Які існують формальні методи верифікації?
4. Які існують динамічні методи верифікації?
5. Які існують групи несправності цифрових систем?
6. Що може бути причиною типових несправностей компонентів системи?
7. Для виявлення яких вимог використовують структурну модель?
8. Наведіть приклади апаратних несправностей.
9. Коли вийшла перша версія стандарту Boundary-Scan?
10. До яких методів тестування відноситься тестування на основі моделей?

ЧАСТИНА 2. ПРАКТИЧНІ РОБОТИ З IDE VIVADO

2 ПРОЕКТУВАННЯ В IDE VIVADO

Мета цієї роботи – по*к*азати повний процес створення простої *HDL* схеми в *IDE Vivado* з використанням плати *Basys3*.

По завершенні цієї роботи, ви навчитеся:

– створювати проект в IDE Vivado, використовуючи в ньому моделі *HDL* і вказуючи конкретний *FPGA* пристрій, розташований на платі *Basys3*;

– використовувати вбудовані Xilink Design Constraint (XDC) файли для вказівки розташування контактів;

– симулювати роботи схем в симуляторі Vivado;

– синтезувати і реалізовувати схеми;

– генерувати бітові пото*к*и;

– конфігурувати *FPGA* пристрій, використовуючи згенеровані бітові пото*к*и і перевіряти його функціонування.

2.1 Опис схеми

Схема містить декілька входів, безпосередньо з'єднаних відповідним вихідними *LED*-пристроями. Інші входи логічно обробляються до того, як результати виводяться на інших світлодіодах, як по*к*азано на рисунку 2.1 [2,8,9].

2.2 Поряд*ок* виконання роботи

Крок 1. Створення проекту в IDE Vivado

1. Запустіть *IDE Vivado* і створіть проект, використовуючи панель *XC7A35TCPG236*-1 (*Basys3*), а також *VHDL*, *HDL*. Використовуйте файли lab1.vhd і lab1.xdc з директорії 2015_1_artix7_sources \ lab1.

Відкрийте "*IDE Vivado: Пуск> Всі програми>Xilinx Design Tools> Vivado 2015.1>Vivado 2015.1".*

Натисніть «Створити новий проект». Ви побачите діалогове вікно «Створити новий проект». Натисніть «Далі».

Натисніть кнопку «*Browse*» поруч з полем «Project location», вкажіть папку *C:\Digilent\Basys3Workshop/ 2015_1_artix7_labs,* і натисніть «Select».

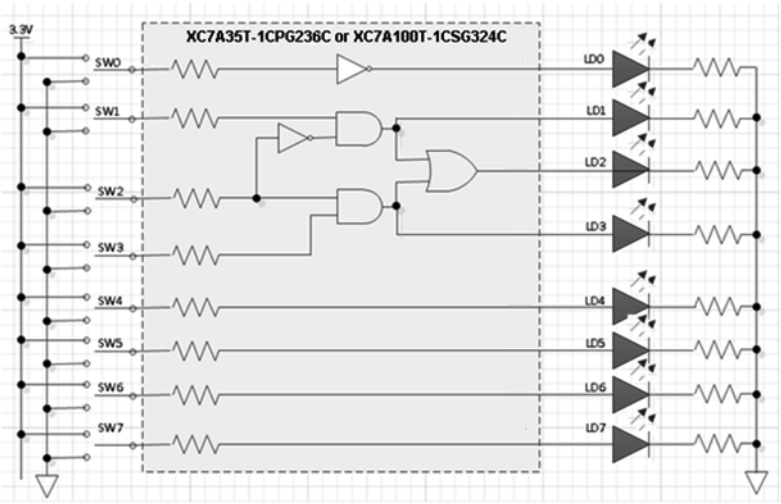


Рисунок 2.1. Повна схема

Введіть «lab1» вполе «*Project name*». Переконайтеся, що встановлено прапорець на пункті «*Create Project Subdirectory*». Натисніть «*Next*» (рис. 2.2).

Виберіть тип проекту «*RTLProject*». Переконайтеся, що знято прапорець з пункту «Do not specify sources at this time». Натисніть «Next».

Використовуючи меню, що випадає, виберіть «*VHDL*» в полі «Target Language» і «Mixed» в полі «Simulator Language» у формі «Add Sources» (рис. 2.3).

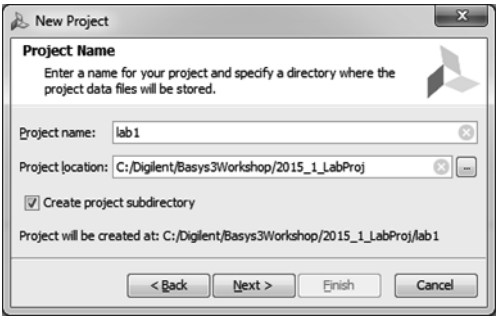


Рисунок 2.2 - Створення нового проекту в IDE Vivado

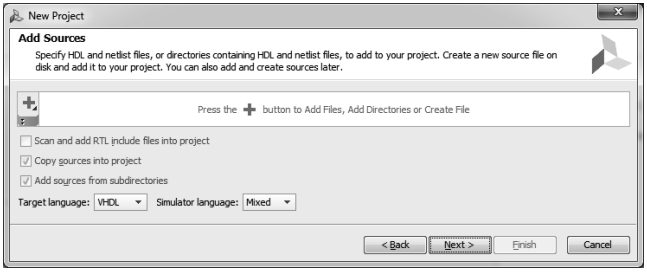


Рисунок 2.3 - Вибір цільової мови і мови симуляції

Натисніть кнопку «+», потім виберіть «AddFiles ...», перейдіть в папку C:\Digilent*Basys3*Workshop\ 2015_1_artix7_sources\lab1, виберіть файл lab1.vhd, і натисніть «*OK*». Переконайтеся, що прапорець пункту «Copysourcesintoproject» встановлено. Натисніть «Next» щоб перейти до форми «AddExisting*IP*».

Оскільки ми ще не створили жодного *IP*, натисніть «Next», щоб перейти до форми «AddConstraints».

Натисніть на кнопку «+», виберіть «AddFiles ...», перейдіть в папку C:\Digilent\Basys3Workshop\ 2015_1_artix7_sources\lab1, виберіть файл lab1_Basys3.xdc і натисніть «*OK*». Переконайтеся, що прапорець «Copysourcesintoproject» встановлено. Натисніть «Next».

Цей *XDC* файл вказує відповідність фізичних входів/виходів *FPGA* пристрою та контактів перемикачів та світлодіодів на панелі. Ця інформація може бути отримана як з схеми панелі, так і з керівництва користувача.

У формі «Default Part», використовуються опції «Parts» і різноманітні пункти меню, що випадає в секції «*Filter*». Виберіть «XC7A35TCPG236-1». Натисніть «*Next*» (рис. 2.4).

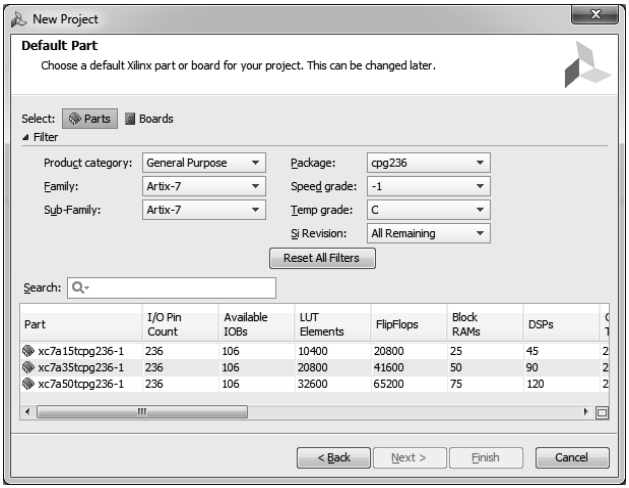


Рисунок 2.4 - Конфігурація панелі *Basys3*

Ви можете вибрати опцію «Boards Specify», виберіть Artix-7 під «*Libraryfilter*» і вкажіть відповідну панель. Майте на увазі, що «*Basys3*» немає в списку, оскільки її немає в базі інструментів.

Натисніть «*Finish*» щоб створити новий проект Vivado. Використовуючи «Провідник», зайдіть в C:\Digilent*Basys3*Workshop\ 2015_1_artix7_labs\lab1. Ви побачите, що папки lab1.cache і lab1.srcs і файл проекту lab1.xpr (Vivado) були створені. Папка «lab1.cache» існує до тих пір, п*оки* не буде створена база даних проекту. Дві папки, «constrs_1» і «sources_1», були створені в папці «lab1.srcs»; в цих папках знаходяться скопійовані файли «lab1.xdc» і «lab1.vhd» (рис. 2.5).

Рисунок 2.5 - Згенерована структура пап*ок*

2. Відкрийте файл «*lab1.vhd*» і ознайомтеся з вмістом

У вкладці «*Sources*», двічі натисніть на «lab1.vhd», щоб відкрити файл в текстовому редакторі (рис. 2.6).

У *VHD* коді, рядки 1-3 закоментовані і описують назву і призначення модуля. Рядки 5-6 підключають стандартні IEEE бібліотеки і пакет STD_LOGIC_1164.

Рядки 8-11 оголошують сутність lab1 і визначають сигнали портів інтерфейсу. Використовуються два вектора типу std_logic_vector (по 8 сигналів у кожного), а також входи swt (перемикачі) та виходи led.

Рядок 13 описує поведінкову архітектуру суті lab1. У рядку 15 оголошується внутрішні сигнали, intLed, для оголошення стану LED.

Рядки 19-24 описують функціонал, в той час як ряд*ок* 26 привласнює вирахувані значення вихідним LED сигналам.

Рядок 28 закінчує архітектуру.

3.Відкрийте вихідний файл lab1_*Basys3*.xdc і перейдіть до вмісту.

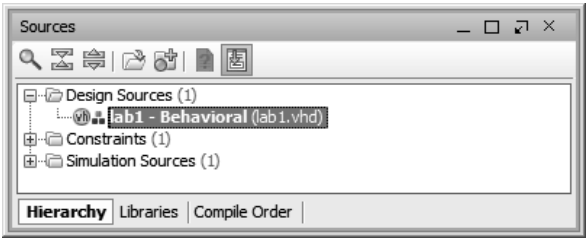


Рисунок 2.6 - Вибір вихідного файлу

У вкладці «*Sources*» відкрийте папку «*Constraints*» і двічі натисніть на файлі lab1.xdc щоб відкрити його в текстовому редакторі (рис. 2.7).

Рядки 5-20 визначають розташування контактів для вхідних перемикачів [7: 0], в той час як рядки 25-40 визначають розташування контактів для вихідних LED [7: 0].

4.Проведення RTL (register-transfer level) аналізу вхідних файлів.

Розгорніть пункт «*Open Elaborated Design*» під завданнями «*RTL Analysis*» на панелі «*Flow Navigation*» і натисніть на «Schematics». Пропустіть інформаційне вікно та натисніть «*OK*».

Буде розроблена модель (схема) і по*к*азане логічне представлення схеми (рис. 2.8). Якщо ваше зображення відрізняється від нижчезазначеного, натисніть на «Elaboration Settings» і виберіть модель «*Blackbox*». Візьміть до уваги попередження про те, що розроблена схема застаріла. Натисніть кнопку «*Reload*».

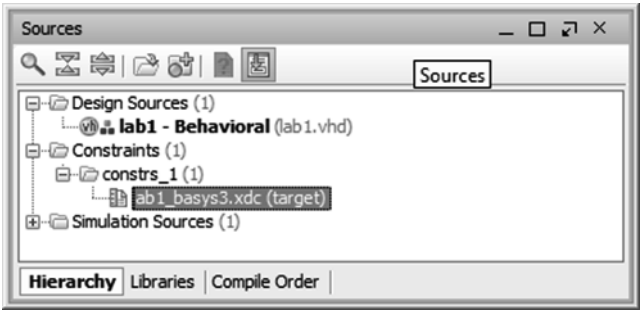


Рисунок 2.7 - Відкриття файлу lab1_*Basys3*.xdc

Зауважте, що деякі вхідні перемикачі перетворюються перед тим як стати вихідними для *LED* діодів, а інші йдуть безпосередньо до *LED* діодів як зазначено в файлі.

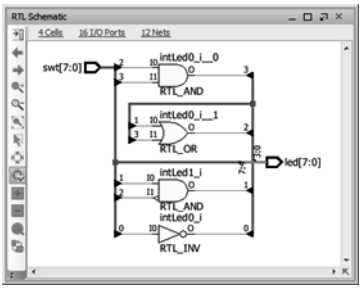


Рисунок2.8 - Логічне представлення схеми

Крок 2.

1. Додавання файлу тестування lab1_tb.vhd

Натисніть на «*Add Sources*» під пунктом «*Project Manager*» на панелі «*Flow Navigation*» (рис. 2.9).

Виберіть пункт «*Add or Create Simulation Sources*» і натисніть «*Next*» (рис. 2.10).

Рисунок 2.9 - Додавання файлу

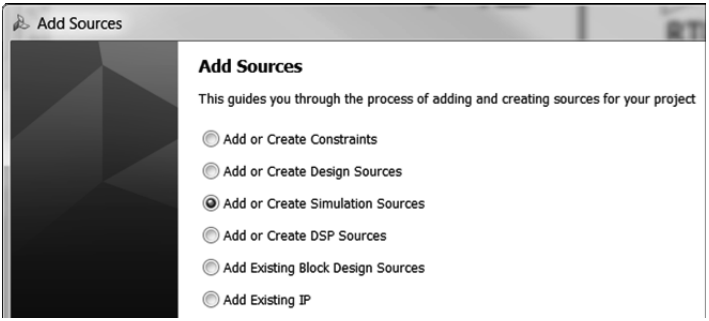


Рисунок 2.10 - Вибір налаштувань додавання вхідних файлів

У формі «*Add Sources Files*», натисніть на кнопку «+», а потім виберіть «*Add Files ...*». Зайдіть в папку C:\Digilent*Basys3*\ Workshot\2015_1_artix7_sources\lab1, виберіть файл lab1_tb.vhd і натисніть «*OK*».

Переконайтеся, що встановлено прапорець на пункті «Copy sources into project». Натисніть «*Finish*». Зайдіть у вкладку «Sources» і розкрийте групу «*Simulation Sources*». Файл lab1_tb.vhd був доданий в групу «*Sources*», і файл lab1.vhd був автоматично доданий в її ієрархію як об'єкт типу «пристрій в тестуванні» (рис. 2.11).

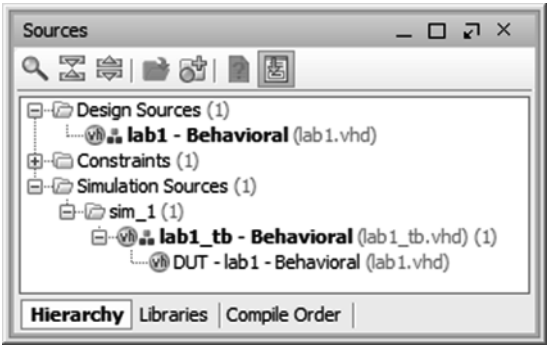


Рисунок 2.11 - Ієрархія файлів симуляції

Використовуючи «Провідник», переконайтеся що папка «sim_1» була створена на одному рівні з «constrs_1» і «sources_1» в папці «lab1.srcs», а також що копія «lab1_tb.vhd» була поміщена в папку «lab1.srcs> sim_1> imports> lab1 ». Двічі натисніть на «lab1_tb» у вкладці «Sources» щоб переглянути вміст файлу (рис. 2.12).

Даний файл запускає бібліотеку *IEEE* і необхідні пакети. Пакет *IEEE.STD_LOGIC_ARITH* необхідний для використання функції «*conv_std_logic_vector*».

Об'єкт є порожнім, тобто немає ні вхідних, ні вихідних сигналів для процесу тестування.

Опис архітектури:

– тестовий компонент *DUT* (*device under testing*);

– кілька сигналів;

– функція, що описує такий же функціонал модуля, який очікується для обчислення значень.

Тіло архітектури:

– ініціалізує об'єкт типу *DUT* (*device under testing*);

– починає процес симуляції і порівнює отримані значення з очікуваними;

– виводить повідомлення в консолі симуляції під час виконання.

2. Симуляція схеми протягом 200 нс з використанням симулятора Vivado.

```
----------------------------------------------------------------------------------
-- Entity Name: lab1_tb
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity lab1_tb is
end lab1_tb;

architecture Behavioral of lab1_tb is
-- component declaration for the Unit Under Test (UUT)
   COMPONENT lab1 is
      Port (swt : in STD_LOGIC_VECTOR (7 downto 0);
            led : out STD_LOGIC_VECTOR (7 downto 0));
   end COMPONENT;

   signal switches, leds, e_leds: std_logic_vector(7 downto 0);

   function expected_led (swt : std_logic_vector(7 downto 0))
                  return std_logic_vector is
      variable exp_led: STD_LOGIC_VECTOR (7 downto 0);
   begin
      exp_led(0) := not swt(0);
--       exp_led(0) := swt(0);  -- replace the previous line with this one, to fail the test
      exp_led(1) := swt(1) and not swt(2);
      exp_led(3) := swt(2) and swt(3);
      exp_led(2) := exp_led(1) or exp_led(3);
      exp_led(7 downto 4) := swt(7 downto 4);
      return exp_led;
   end expected_led;
```

Рисунок 2.12 - Вміст файлу «lab1_tb»

Виберіть пункт *«Simulation Settings»* в меню *«Project Manager»* на панелі *«Flow Navigation».*

З'явиться форма «Project Settings» з інформацією про налаштування симуляції. Приберіть прапорець з пункту «*Generate ScrIPts Only*», якщо він встановлений.

У вкладці «*Simulation*» встановіть значення «*Simulation Run Time*» на 200ns і натисніть «*OK*» (рис. 2.13).

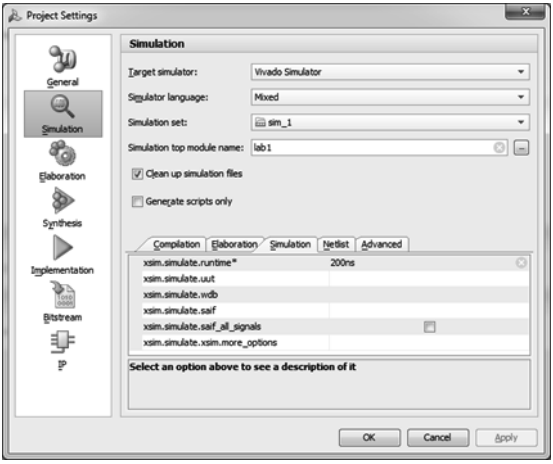


Рисунок 2.13 - Налаштування часу симуляції

Натисніть на «*Run Simulation> Run Behavioral Simulation*» в меню «*Project Manager*» на панелі «*Flow Navigation*». Тестовий і вхідний файли скомпілюються і запуститься симулятор *Vivado* (якщо не виникнуть помилки). Ви побачите результат симуляції приблизно такий, як на рисунку 2.14.

Ви побачите чотири головних вида:

- «*Scopes*», де по*к*азані як тестові, так і глобальні об'єкти,
  «*Objects*», де по*к*азані сигнали верхнього рівня,
- хвилеподібний графік,
  «*Tcl Console*», де відображені процеси симуляції по ходу виконання.

Врахуйте, що папка «lab1.sim» створена в папці «lab1», поряд з кількома папками рівнем нижче (рис. 2.15).

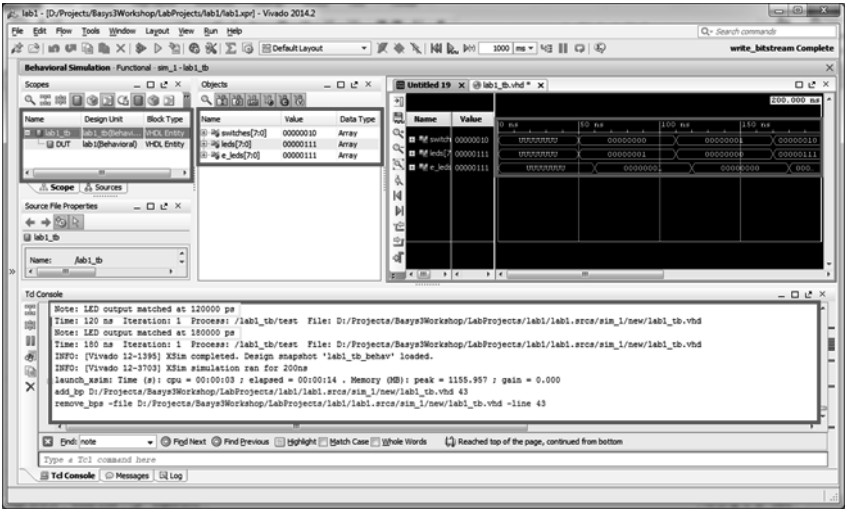


Рисунок 2.14 - Результат симуляції

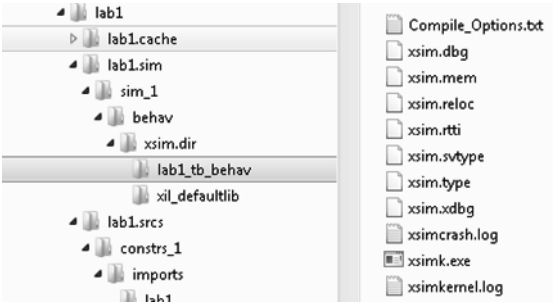


Рисунок 2.15 - Структура пап*ок* після запуску симуляції

Ви побачите кілька кноп*ок* навпроти вікна з графіком (рис. 2.16), які необхідні для деяких дій, які вказані в таблиці нижче.

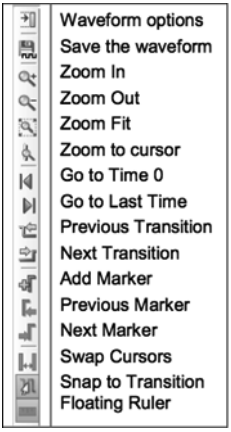


Рисунок 2.16 - Різні кнопки для взаємодії з графіком

Натисніть на кнопку «*Zoom Fit*» щоб побачити весь графік.

Врахуйте, що результат зміниться, якщо зміняться вхідні дані.

Ви можете пересувати графік симуляції використовуючи інструмент «*Float*» у верхньому правому куті графіка (рис. 2.17). Це дозволить вам зробити вікно ширше щоб побачити графік. Щоб повернути графік на своє місце, натисніть на кнопку «*Dock Window*» (рис. 2.18).

Рисунок 2.17 - Кнопка «Float»

Рисунок 2.18 - Кнопка «Dock Window»

3. Зміна формату відображення

Виберіть «switches [7: 0]» на вікні з графіком, натисніть правою кнопкою миші, виберіть «Radix» і виберіть «Hexadecimal». Залиште пункти «leds [7: 0]» і «e_led [7: 0]» як є, щоб побачити кожен вихідний біт.

Додайте більше сигналів для спостереження низькорівневих сигналів і продовжите час симуляції до 500 нс.

Розкрийте об'єкт «lab1_tb», у вікні «Scopes» виберіть об'єкт «DUT». У вікні «Objects» будуть по*ка*зані сигнали «swt [7: 0]» і «led [7: 0]» (рис. 2.19).

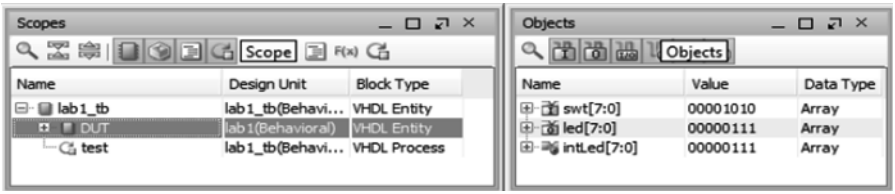


Рисунок 2.19 - Вибір низькорівневих сигналів

Виберіть «swt [7: 0]» і «led [7: 0]» і перетягніть їх у вікно з графіком, щоб простежити їх низькорівневі сигнали.

На панелі інструментів симулятора, введіть «500» в поле «Run time», натисніть на меню, що випадає, з одиницями вимірювання часу і виберіть «ns» і натисніть кнопку  . Симуляція триватиме на 500 нс.

Натисніть на кнопку «Zoom Fit» щоб ознайомитись з результатами (рис. 2.20).

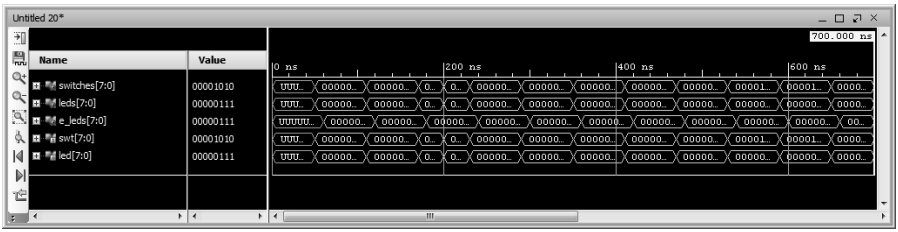


Рисунок 2.20 - Запуск симуляції на 500 нс

Відкрийте «Tcl Console» і ви побачите результат симуляції (рис. 2.21).

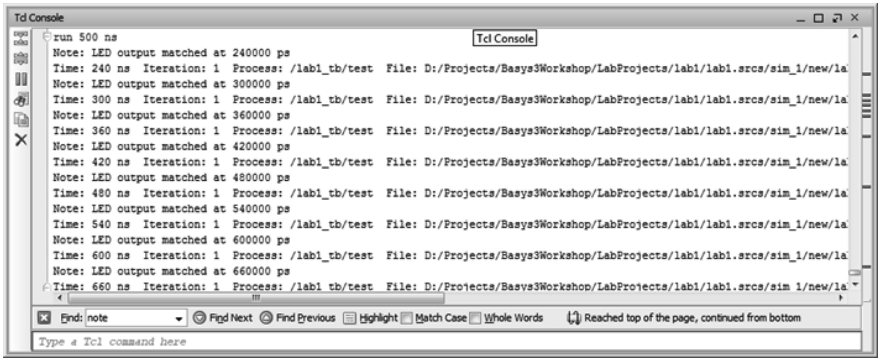


Рисунок 2.21 - «Tcl Console» після запуску симуляції на додаткові 500 нс

Закрийте симулятор, натиснувши на «File> Close Simulation». Натисніть «*OK*», потім «Discard», щоб закрити симулятор без збереження графіка.

Крок 3.

Збірка схеми за допомогою збирача *Vivado* і аналіз результатів

Натисніть на кнопку «*Run Synthesis*» в меню «*Synthesis tasks*» на панелі «*Flow Navigation*».

Виберіть опцію «Open Synthesized Design» і натисніть «*OK*», оскільки ми хочемо подивитися на результат збірки перед тим, як приступити до етапу реалізації. Натисніть «Yes», щоб закрити схему, якщо з'явиться будь-яке діалогове вікно.

Перейдіть у вкладку *«Project Summary*». Якщо ви не можете знайти її, перейдіть в «*Layout> Default Layout*» (рис. 2.22). Виберіть вкладку «Table» на вкладці «*Project Summary*». Зауважте, що в ній знаходяться три попередніх ресурсу LUT і 16 джерел введення-виведення (рис. 2.23).

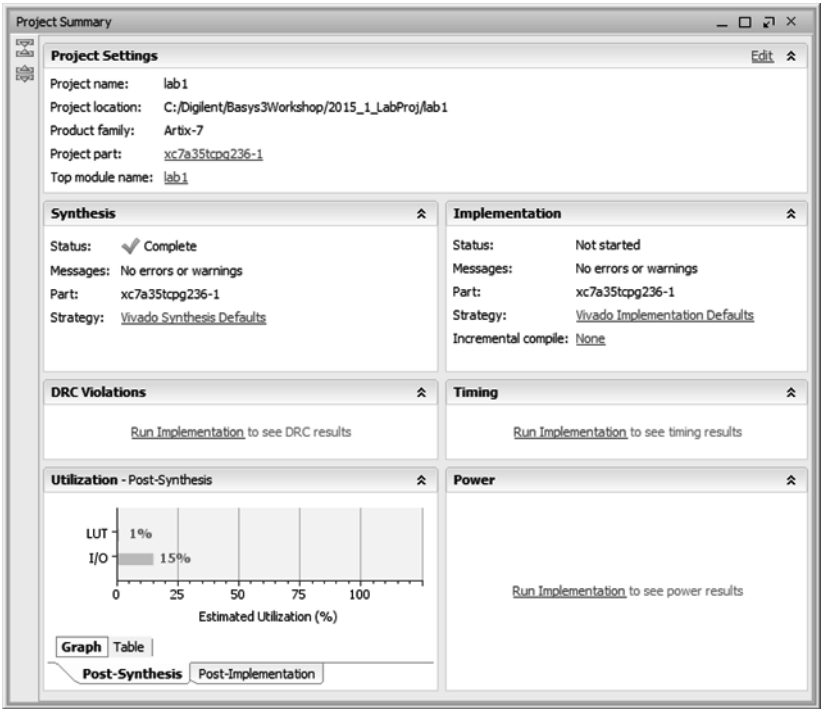


Рисунок 2.22 - Вікно «Project Summary»

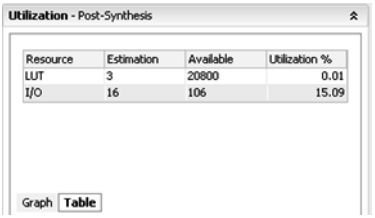


Рисунок 2.23 - Ініціалізація джерел для *Basys3*

На панелі «Flow Navigation», в меню «Synthesis» натисніть на «Schematic» щоб переглянути зібрану схему в схематичному вигляді (рис. 2.24).

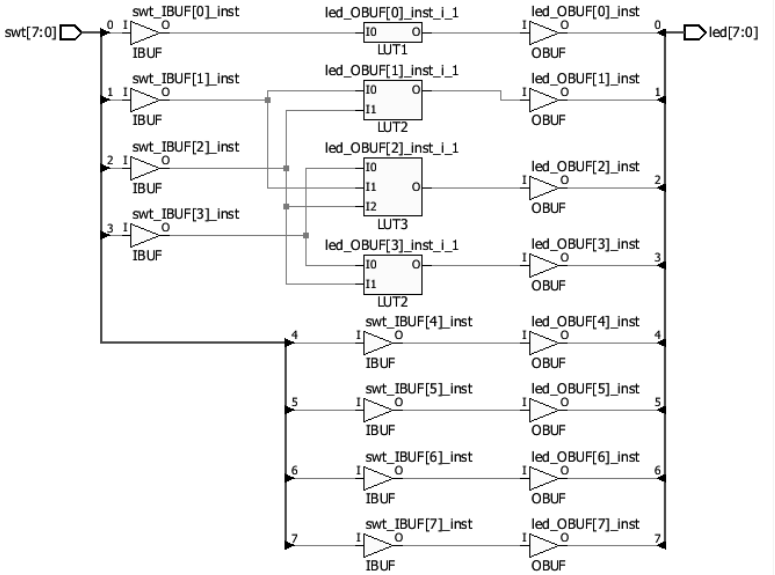


Рисунок 2.24 - Схематичний вид зібраної схеми

Зауважте, що *IBUF* і *OBUF* автоматично добавляються до схеми, оскільки входи та виходи буферизовані. Логічні шлюзи реалізовані в *LUT*-ах (перший вхід позначений як *LUT1* і т.д.). Чотири шлюзи в результаті RTL аналізу відповідаютьчотирьом LUT-ам у результаті збірки. Використовуючи «Провідник», переконайтеся, що папка «lab1.runs» створена в папці «lab1». У ній повинна бути створена папка «synth_1», в якій знаходяться файли, що відносяться до збірки (рис. 2.25).

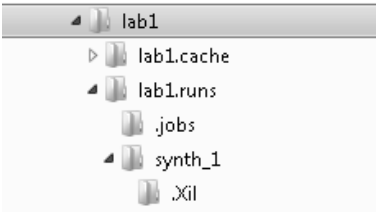


Рисунок 2.25 - Структура пап*ок*після складання схеми

Крок 4.

Реалізація схеми з настройками за замовчуванням і використання результатів аналізу

.Натисніть на кнопку «*Run Implementation*» в меню «Implementation tasks» на панелі «*Flow Navigator*». Процес реалізації буде запущений за зібраною схемою. Коли процес завершиться, з'явиться відповідне діалогове вікно з трьома опціями.

Натисніть на «*Open implemented design*» і натисніть «*OK*», щоб переглянути реалізовану схему на пристрої.

.Якщо необхідно, натисніть «*Yes*», щоб закрити зібрану схему. Далі відкриється реалізована схема.

На панелі «*Netlist*» виберіть одну з мереж (наприклад, led_OBUF [1], рис. 2.26). Зауважте, що вона відкриється в частині X0Y0 у вікні перегляду пристрою (рис. 2.27).

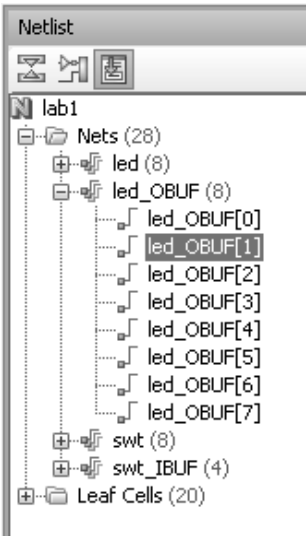


Рисунок 2.26 - Вибір мережі

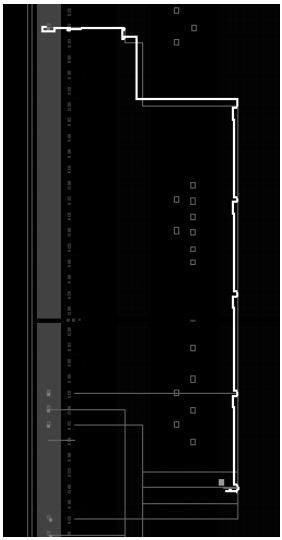


Рисунок 2.27 - Перегляд реалізованої схеми для *Basys3*

Натисніть на кнопку «*Routing Resources*»  , щоб з'явилися ресурси маршрутизації.

Закрийте вікно з реалізованою схемою і перейдіть у вкладку «*Project Summary*» (рис. 2.28), потім у вкладку «*Post-Implementation»*. Зауважте, що актуальне джерело - це три LUT-а і 16 входів введення-виведення. Також, там вказується, що часові рамки не були встановлені.

Використовуючи «Провідник», переконайтеся, що папка «impl_1» була створена на одному рівні з «synth_1» в папці «lab1.runs». Ця папка містить файли, включно з файлами звітності про процес і результати реалізації схеми.

У *Vivado*, перейдіть у вкладку «Reports» (рис. 2.29) і двічі натисніть на «*Utilization Report*» в секції «*Place Design*». Ви побачите звіт на допоміжній панелі, який показує результати

утилізації джерела. Зауважте, що оскільки наша схема комбінаторна, ніякі реєстратори не були використані.

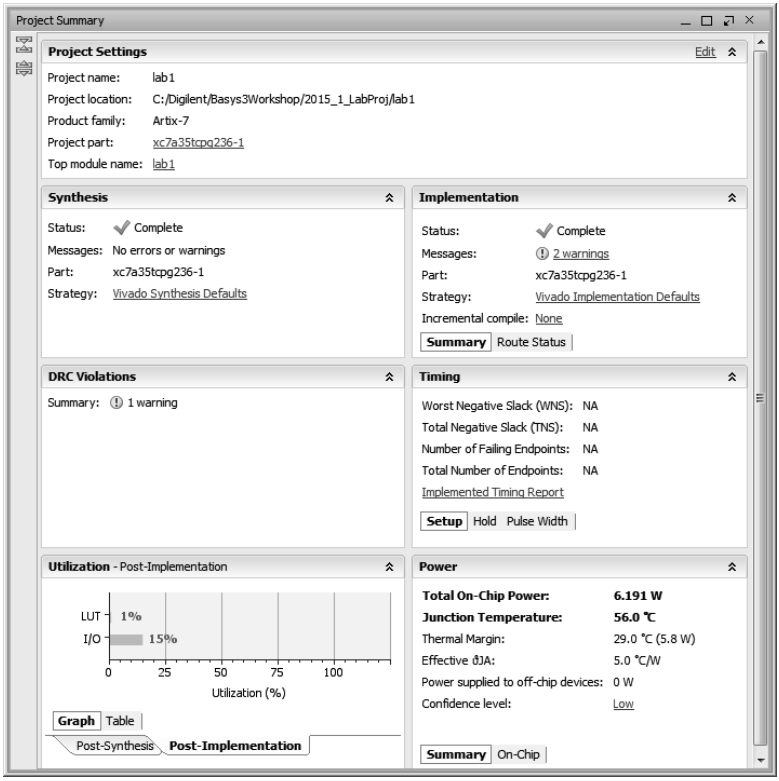


Рисунок 2.28 - Результати реалізації для *Basys3*

Крок 5.

Для запуску часової симуляції натисніть на «Run Simulation> *Run Post-Implementation Timing Simulation*» в меню «*Simulation tasks*» на панелі «*Flow Navigation*». Пропустіть інформаційне вікно, натиснувши «*Y*es».

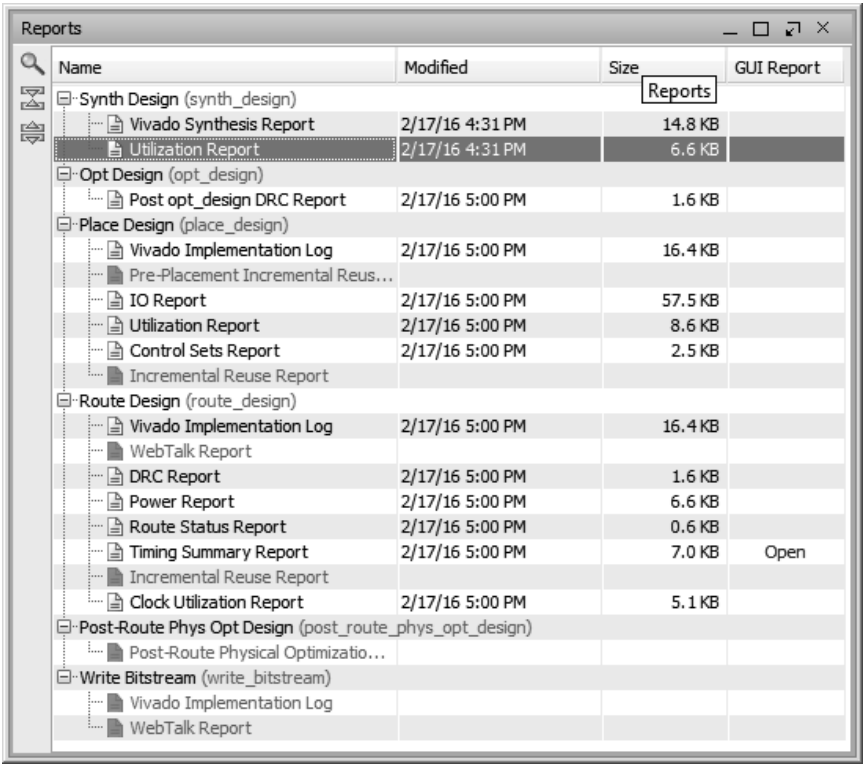


Рисунок2.29 - Доступні звіти

Запуститься симулятор *Vivado*, який буде використовувати реалізовану схему і файл «lab1_tb» як вис*ок*орівневий модуль. Використовуючи «Провідник», переконайтеся, що папка «timing» була створена в «lab1.sim> sim_1> impl». Вона містить файли для запуску часової симуляції.

Натисніть на кнопку «*Zoom Fit*» щоб побачити графік від 0 до 200 нс.

Натисніть правою кнопкою миші на пункт «50 ns» і виберіть «*Markers> Add Marker*».

Аналогічним чином повторіть операцію з пунктами «55 ns» і «60 ns», де змінюється «e_led» (рис. 2.30).

Рисунок 2.30 - Результат часової симуляції

Закрийте симулятор, натиснувши на «File> Close Simulation» без збереження змін.

Крок 6

Підключіть плату і увімкніть живлення. Згенеруйте потік бітів, запустіть апаратну сесію і запрограмуйте *FPGA* .

Переконайтеся, що *Micro-USB* кабель підключений до *JTAG* гнізда.

Переконайтеся, що живлення так само включено (рис. 2.31).

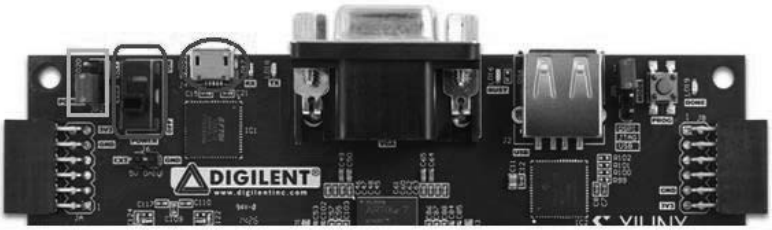


Рисунок 2.31 - Вхід для підключення плати *Basys3*

Увімкніть плату.Натисніть на «Generate Bitsream» в меню «Program and Debug tasks» на панелі «Flow Navigation». Запускається процес генерування бітового пото*ку* для реалізованої

схеми. Коли він завершиться, з'явиться діалогове вікно з двома опціями (рис. 2.32).

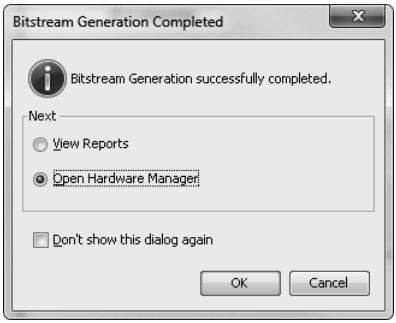


Рисунок 2.32 - Генерування бітового пот*ок*у

Цей процес створює файл «lab1.bit» в папці «impl_1».

Виберіть опцію «*Open Hardware Manager*» і натисніть «*OK*».Відкриється відповідне вікно, що показує статус «Unconnected».

Натисніть на посилання «Open target/Open New Target» (рис. 2.33). Також, ви можете використовувати «Recent Targets» або натиснути на «*Auto Connect*». У цьому випадку, перейдіть відразу до пункту 6-1-10.

Натисніть «Next», щоб відкрити форму «*Hardware Server Settings*».

Вибравши пристрій, натисніть «*Next*». Кабель *JTAG* , який використовує *Xilinx*_tcf повинен визначитися і ідентифікуватися як цільовий пристрій (рис. 2.34). Також, будуть по*к*азані всі пристрої.

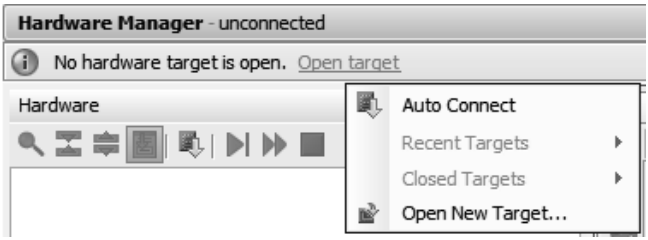


Рисунок2.33 - Відкриття нового пристрою

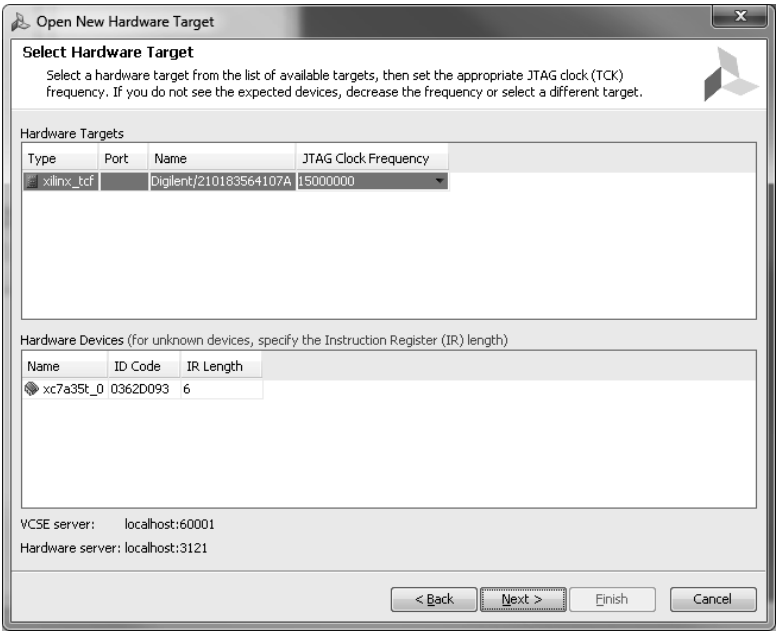


Рисунок 2.34 - Ідентифікація нового цільового пристрою

Натисніть «*Next*», а потім «*Finish*». Статус апаратної сесії зміниться на назву сервера і пристрій буде підсвічено (рис. 2.35). Також зауважте, що статус показує, що пристрій не запрограмований.

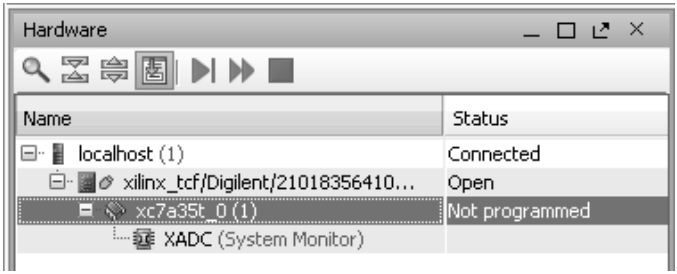


Рисунок 2.35 - Відкрита апаратна сесія

Виберіть пристрій і переконайтеся, що файл «lab1.bit» обранийв пункті «*Programmig file*» на головній вкладці.

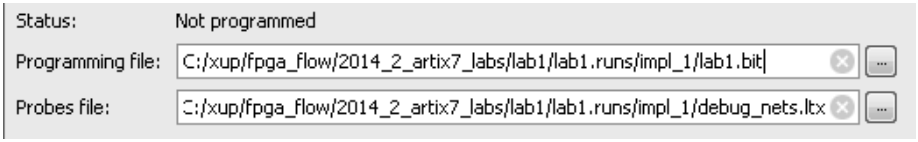


Рисунок 2.36 - Вибір файлу для пункту «Programming file»

Натисніть на посилання «*Program device> XC7A35T_0*» на зеленому інформаційному рядку (рис. 2.37).

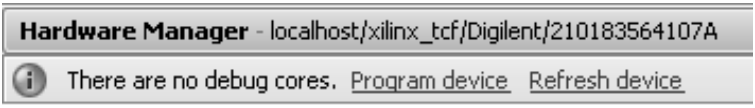


Рисунок 2.37 - Перехід до програмування *FPGA*

Натисніть «*Program*», щоб запрограмувати *FPGA* . Ви побачите напис «*DONE*», як тільки пристрій буде успішно запрограмовано.

Перевірте функціонування пристрою перемиканням перемикачів і переглядом виведення на *LED*.

Якщо результат вас влаштовує, вимкніть плату.

Завершіть апаратну сесію, натиснувши на «*File>Close Hardware Manager*». Натисніть «*OK*» для завершення сесії.

Закрийте програму *Vivado* натиснувши на «File> Exit».

ПЗ *Vivado* може використовуватися для повного проектування схем. Проект був створений за допомогою наданих вхідних файлів (*HDL* модель). Була виконана поведінкова симуляція для того, щоб перевірити функціонування моделі. Модель була зібрана, реалізована, а також був згенерований потік бітів. Була запущена часова симуляція реалізованої схеми. Функціональність пристрою була перевірена за допомогою згенерованого бітового пот*ок*у.

2.3 Контрольні запитання

1. Як створити новий проект у IDE Vivado?
2. Для чого використовується файл з розширенням .xdc?
3. Які елементи є обов`язковими у *VHD* коді?
4. Яким чином можна налаштувати симуляцію у *IDE Vivado*?
5. Яку інформацію містить вікно «Project Summary»?
6. Як переглянути зібрану схему в схематичному вигляді?
7. Як виконати часову симуляцію?
8. Як запрограмувати *FPGA* за допомогою IDE Vivado?
9. Як виконати поведінкову симуляцію?
10. Яке розширення має файл для програмування плати *Basys3*?

3 ЗБІРКА RTL-СХЕМ

Ця робота описує процес і результат зміни налаштувань синтезу. Ви навчитеся аналізувати схему і згенеровані звіти.

По завершенню цієї роботи, ви будете вміти:

1. Використовувати XDC файли для установки часових обмежень схеми.
2. Розробляти схеми і аналізувати результати.
3. Збирати схему з початковими часовими обмеженнями.
4. Аналізувати результати збирання.
5. Змінювати налаштування збирання.
6. Аналізувати процес складання поетапно.

3.1 Основні теоретичні відомості

Схема складається з *УАПП* приймача, що приймає вхідні дані з клавіатури і відображає двійковий еквівалент набраного символу на 8 світлодіодах. При натисканні кнопки, нижній та верхній напівбайти міняються місцями. Бл*ок*-схема п*ок*азана на рис. 3.1.

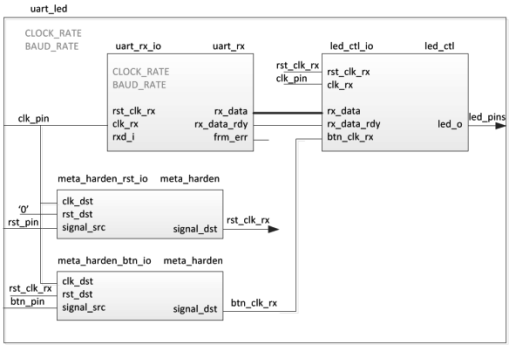


Рисунок 3.1 - Завершена схема

Реалізація та статичний часовий аналіз

Критичний шлях - це шлях, який:

– починається на тактованому елементі;

– проходить через будь-яку кількість комбінаторних елементів і мереж, які їх з'єднують;

– закінчуєтьсяна тактованому елементі.

До тактованих елементів відносяться тригери, блоки оперативної пам'яті, цифрові сигнальні процесори та ін.

До комбінаторних елементів відносяться таблиці пошуку, мультиплексори, ланцюги та ін.

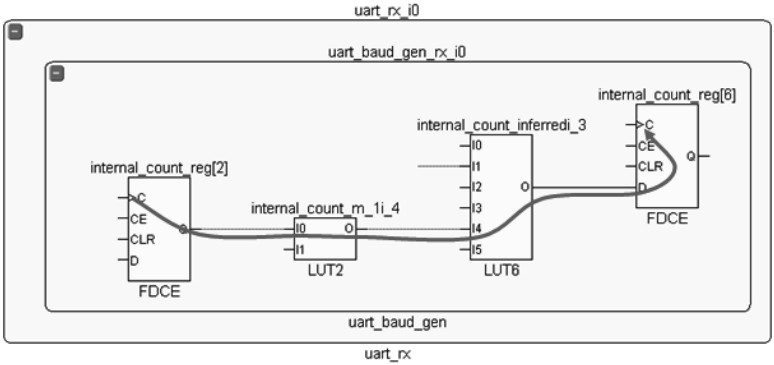


Рисунок 3.2 - Критичний шлях

Перевірка встановлених налаштувань використовується для тестування, що змініна входітактованого елементу вистачає часу перейти до інших тактованих елементівдо наступного такту.

Перевірка затримки використовуться для верифікації, що зміни в тактованому елементі, які були викликані подієютактування, не досягають кінцевого елемента раніше, ніж сама подіятактування не досягне кінцевого елемента.

Найменша затримка використовується для джерела тактування і затримки шляху даних, а найбільша затримка використовується як затримка кінцевого тактування (рис. 3.4).

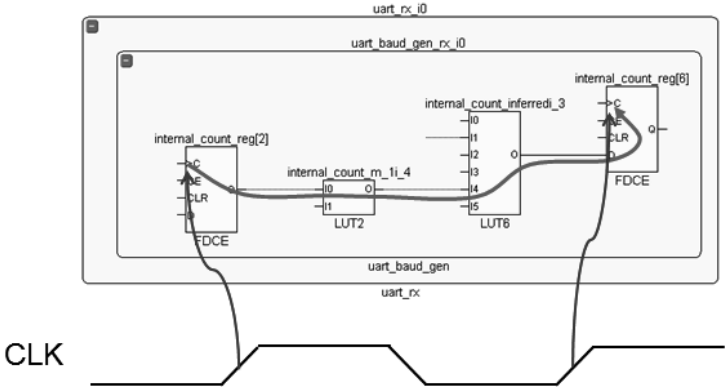


Рисунок 3.3 - Перевірка всіх статичних часових шляхів від початку ланцюга і до його кінця

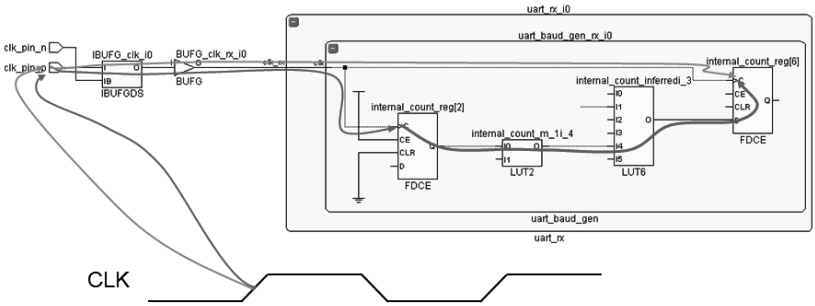


Рисунок 3.4 - Перевірка затримки

3.3 Порядок виконання роботи

Крок 1. Створення проекту Vivado

1. Запустіть *Vivado* і створіть проект, який вказує на пристрій *XC7A35TCPG236*-1 (*Basys3*), використовуючи також файли *VHDL HDL*. Використовуйте надані вхідні файли *VHDL* uart_led_pins_ <Board>.xdc і uart_led_timing.xdc з папки *<2015_1_artix7_sources> \ lab2.*

Відкрийте *Vivado*, вибравши "*Пуск> Всі програми>XilinxDesignTools>Vivado 2015.1>Vivado 2015.1*".

Натисніть "Створити новий проект" для запуску майстра створення проектів. З'явиться діалогове вікно "*CreateNewVivado Projec*t". Натисніть "*Next*".

Натисніть кнопку "Browse" в поле розташування проекту на формі створення нового проекту, перейдіть до папки *<2015_1_artix7_labs>*, і натисніть кнопку "*Select*".

Введіть lab2 в поле "*Project Name*". Переконайтеся, що пункт "Create Project Subdirectory" відзначений галочкою. Натисніть "Next".

Виберіть опцію *RTL* Project як тип проекту, і натисніть кнопку "Next".

У випадаючому меню виберіть *VHDL* як мову перекладу і Mixed в якості мови для симуляції в формі "AddSources".

Натисніть на кнопку +, потім виберіть AddFiles, перейдіть до папки <2015_1_artix7_sources> \ lab2, виберіть всі файли *VHDL* (led_ctl.vhd, meta_harden.vhd, uart_baud_gen.vhd, uart_led.vhd, uart_rx.vhd і uart_rx_ctl.vhd), натисніть кнопку *OK*, а потім натисніть кнопку "Next", щоб перейти до форми "AddExisting *IP*".

Так як у нас немає ніякого *IP* для додавання, натисніть кнопку "Next", щоб перейти до форми "*AddConstraints*".

Натисніть на кнопку +, потім виберіть AddFiles, перейдіть в папку <2015_1_artix7_sources> \ lab2, виберіть uart_led_timing.xdc і натисніть кнопку *OK*.

Натисніть "*Next*". Цей *XDC* файл встановлює основні часові обмеження (період, затримки вхідного та вихідного сигналів) схеми.

У формі "*DefaultPart*", за допомогою опції Parts і різноманітних меню в секції "*Filters*", виберіть XC7A35TCPG236-1 (для *Basys3*). Натисніть "Next".

Натисніть кнопку "*Finish*", щоб створити проект *Vivado*.

2. Аналіз ієрархії вхідних файлів проекту.

На панелі "*Sources*" розгорніть пункт uart_led і зверніть увагу на ієрархію модулів нижнього рівня (рис. 3.5).

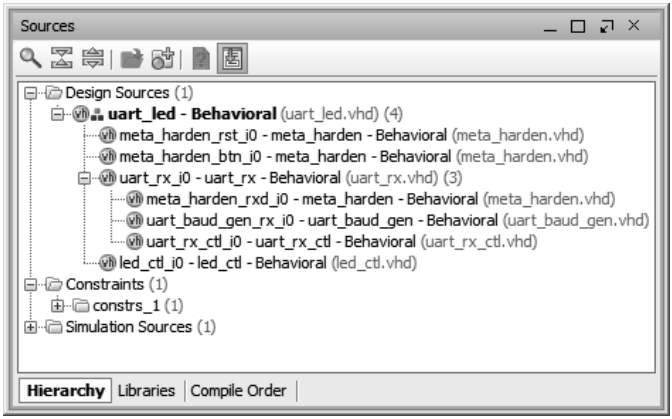


Рисунок 3.5 - Відкриття вхідного файлу

Двічі клацніть на елемент uart_led, щоб переглянути його вміст. Зверніть увагу, що в коді *VHDL*, константи BAUD_RATE і CLOCK_RATE визначені як 115200 і 100 МГц відповідно, як показано на рисунку 1. Також зверніть увагу, що створюються компоненти нижчого рівня. Компоненти meta_harden використовуються для синхронізації асинхронного скидання і входів кноп*ОК*.

.Розгорніть елемент uart_rx_i0, щоб побачити його ієрархію. Цей модуль використовує baud-генератор та автомат кінцевих станів. Rxd_pin оцифровується на швидкості передачі даних x16.

3.Відкрийте вхідний файл uart_led_timing.xdc і проаналізуйте вміст

На панелі "Sources" розгорніть папку "Constraints" і двічі клацніть по пункту uart_led_timing.xdc, щоб відкрити файл в текстовому режимі (рис. 3.6).

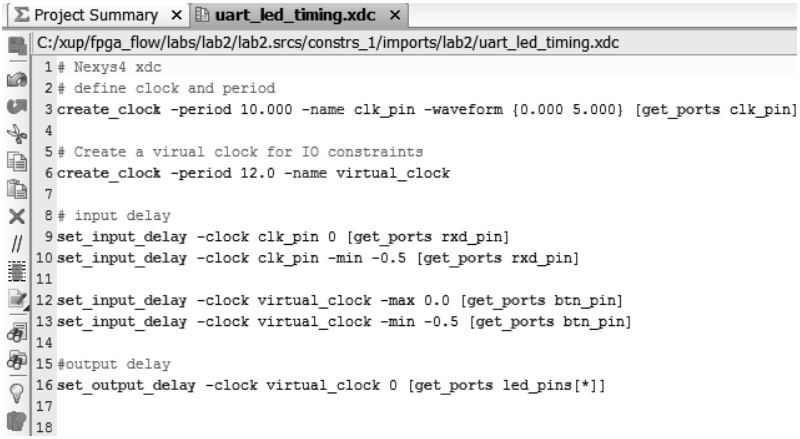


Рисунок 3.6 - Часові обмеження

Рядок3 створює обмеження в 10 нс з робочим циклом 50%. Рядок6 створює віртуальний таймер в 12 нс. Цей таймер можна розглядати як пристрій передачі даних, який генерує результат відповідно до його налаштування. Rxd_pin обмежений щодо таймеру схеми (рядки 9 і 10), в той час як btn_pin обмежений відповідно таймеру передавального пристрою (рядки 12, 13). led_pins обмежені щодо таймера передавального пристрою, тому що приймальний пристрій може його використовувати.

Крок 2. Створення схеми

Створити і провести RTL-аналіз за допомогою оригінального файлу.

Розгорніть пункт "OpenElaboratedDesign" під "RTL AnalysisTasks" на панелі FlowNavigator і натисніть на "Schematic".

Модель (схема) буде створена разом з логічним її поданням. Якщо ваше логічне уявлення відрізняється від наведеного (рис. 3.7), натисніть на "ElaborationSettings", в меню "RTL Analysis" та виберіть "Blackboxmodel". Зверніть увагу на попередження про те, що створена схема є застарілою. Натисніть на кнопку "Reload".

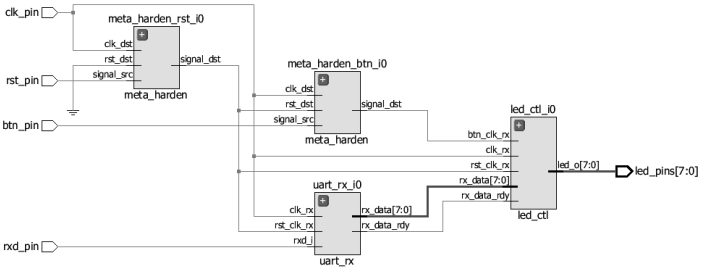


Рисунок3.7 - Логічне представлення схеми

Ви побачите чотири компонента на верхньому рівні, 2 примірника meta_harden, один екземпляр uart_rx, і один примірник led_ctl.

Для того, щоб побачити, де генерується uart_rx_i0, клацніть правою кнопкою миші на uart_rx_i0 і виберіть "GoToSource", де ви побачите, що на рядку 84 у вхідному коді відбувається ініціалізація даного об'єкта.

Двічі клацніть по uart_rx_i0 на схематичному зображенні, щоб побачити основні компоненти (рис. 3.8).

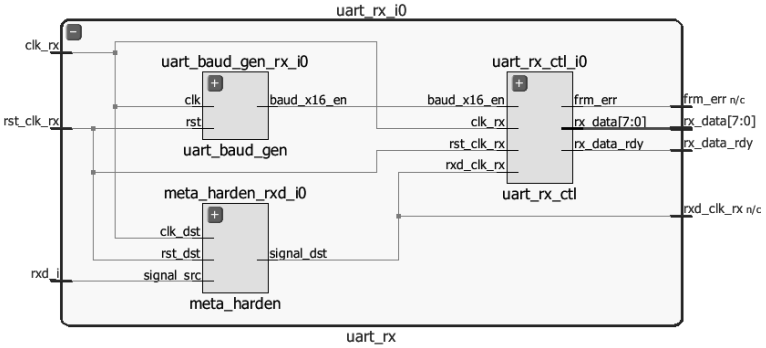


Рисунок 3.8 - Компоненти низького рівня модуля uart_rx_i0

Натисніть на "ReportNoise" в підменю "OpenElaboratedDesign" меню "RTL Analysis" на панелі "FlowNavigator"

Натисніть кнопку *OK*, щоб створити звіт з ім'ям ssn_1.

Відкрийте звіт ssn_1 і подивіться на UnplacedPorts, Summary та I/OBankDetails виділені червоним кольором, оскільки контакти ще не були прикріплені. Зверніть увагу, що показані тільки вихідні контакти, так як шумовий аналіз робиться на вихідних контактах (рис. 3.9).

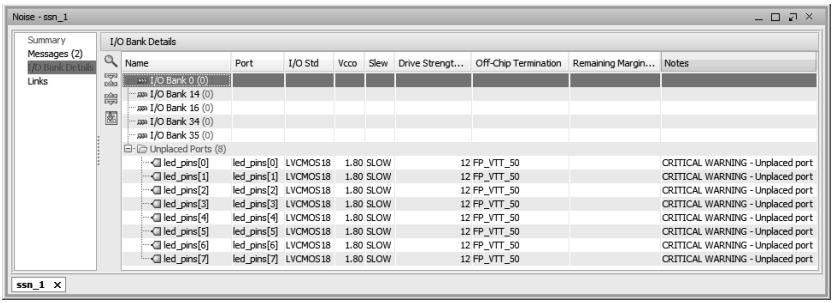


Рисунок 3.9 - Шумовий аналіз

Натисніть на "AddSources" на панелі "Project Navigator", виберіть "AddorCreateConstraints" і натисніть "Next".

Натисніть на AddFiles, перейдіть до папки <2015_1_artix7_sources> \ lab2, виберіть uart_led_pins_*Basys3*.xdc (в залежності від цільової плати), натисніть кнопку *OK*, а потім натисніть кнопку Finish, щоб додати обмеження місць розташування контактів. Зверніть увагу на те, що вхідні файли змінені і IDE помітила це, показуючи попередження, що схему потрібно перезавантажити.

Натисніть на кнопку "Reload". Нові обмеження будуть оброблені та застосовані. Натисніть на "ReportNoise" і натисніть кнопку *OK*, щоб створити звіт з ім'ям ssn_1. Зауважимо, що в цей раз він не показує будь-яких помил*ОК*.

Крок 3. Збірка схеми.

1. Збірка схеми за допомогою інструменту збірки Vivado і аналіз результатів.

Натисніть на "RunSynthesis" в підменю "Synthesis" на панелі FlowNavigator. Процес складання обробить файл uart_led.vhd і всі

файли, підлеглі йому. Коли процес завершиться, з'явиться діалогове вікно "SynthesisCompleted" з трьома опціями.

Виберіть опцію "OpenSynthesizedDesign" і натисніть кнопку *OK*, тому що ми хочемо подивитись на результат збірки. Натисніть кнопку "Yes", щоб закрити створену схему, якщо з'явиться діалогове вікно.

Виберіть вкладку "Project Summary". Якщо ви не бачите вкладку "Project Summary", то виберіть Layout>DefaultLayout, або натисніть на значок"Project Summary".

Натисніть на вкладку "Table" на вкладці "Project Summary" і заповніть наступну інформацію.

Натисніть на "Schematic" в "OpenSynthesizedDesign" в меню "Synthesis" на панелі FlowNavigator для перегляду зібраної схеми в схематичному вигляді (рис. 3.10).

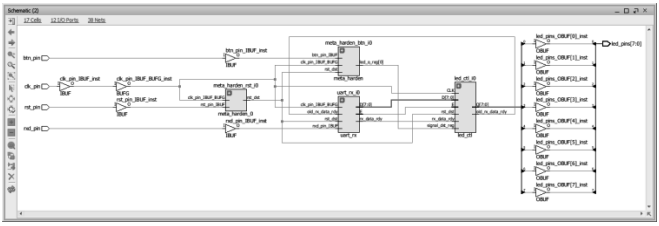


Рисунок 3.10 - Схематичне відображення зібраної схеми

Зверніть увагу на те, що IBUF і OBUF автоматично створюються (додаються) в схему, оскільки вхід та вихід буферизовані. Також, створюються чотири модулі нижнього рівня.

Двічі клацніть на uart_rx_i0 на схематичному вигляді, щоб побачити основні елементи.

Виберіть uart_baud_gen_rx_i0, клацніть правою кнопкою миші і виберіть "GoToSource". Зверніть увагу, що рядокз створюванням виділений. Також зверніть увагу, що параметри CLOCK_RATE і BAUD_RATE передаються в модуль при виклику.

Двічі клацніть на meta_harden_rxd_io, щоб побачити, як схема синхронізації реалізується з використанням двох FF. Ця

синхронізація необхідна, щоб зменшити ймовірність метастабільности.

Натисніть на (<-) в схематичному вигляді, щоб повернутися на батьківський блок.

2.Аналіз звіту про синхронізацію.

Натисніть на "ReportTimingSummary" в меню "SynthesizedDesign" на панелі "FlowNavigator".

Натисніть кнопку *OK*, щоб створити звіт Timing_1 (рис. 3.11).

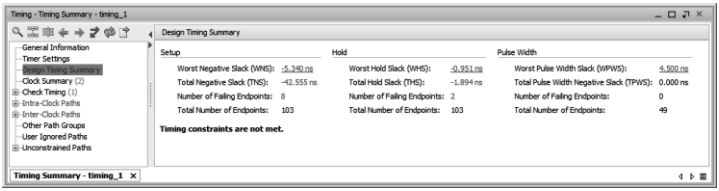


Рисунок 3.11 - Звіт про синхронізацію

Зверніть увагу на те, що пункти "DesignTimingSummary" і "Inter-Clock Paths" в лівій панелі виділяються червоним кольором з повідомленням про порушення синхронізації. У правій частині вікна, інформація згрупована в колонках "Setup", "Hold" і "Width". У колонці "Setup" *WNS* показує , що натискання на нього може дати нам уявлення про те, як сформувався критичний шлях. TNS виділено червоним кольором із зазначенням загальної кількості порушень у схемі, а також загальна кількість критичних шляхів.

Натисніть на посилання *WNS* і ви побачите 8 критичних шляхів (рис. 3.12).

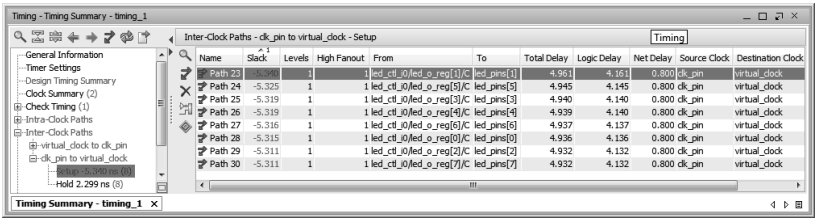


Рисунок3.12 - 8 невдалих шляхів

Двічі клацніть на "Path 23", щоб побачити, як він побудований (рис. 3.13).

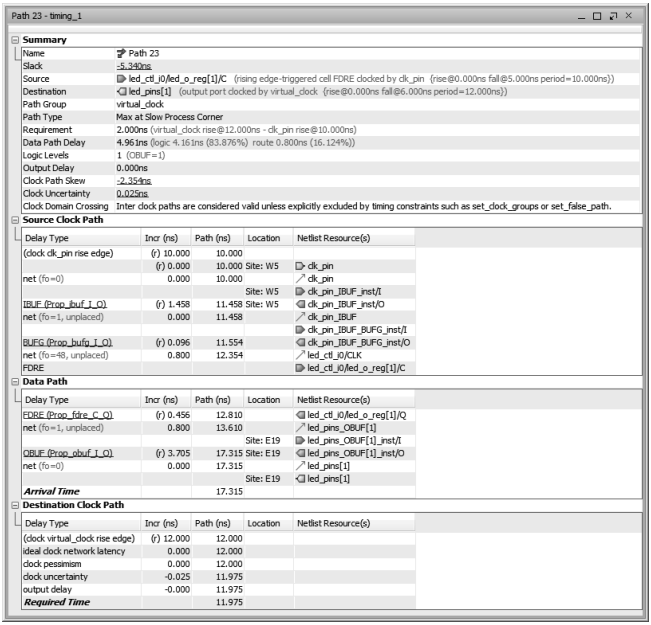


Рисунок 3.13 - Найгірший критичний шлях для *Basys3*

3.Генерування звітів про використання ресурсів та потужність.

Натисніть "*ReportUtilization*" в меню "SynthesizedDesign", і натисніть кнопку *OK*, щоб створити звіт про використання ресурсів (рис. 3.14). Виберіть пункт "Summary" в лівій панелі вкладки "Utilizaion".

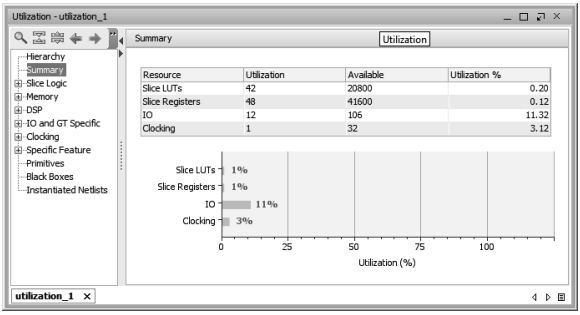


Рисунок 3.14 - Звіт про використання ресурсів *Basys3*

Виберіть "Slice LUT" в лівій панелі і побачите її використання ресурсів екземплярами нижнього рівня (рис. 3.15). Ви можете переглянути подробиці примірників в правій панелі.

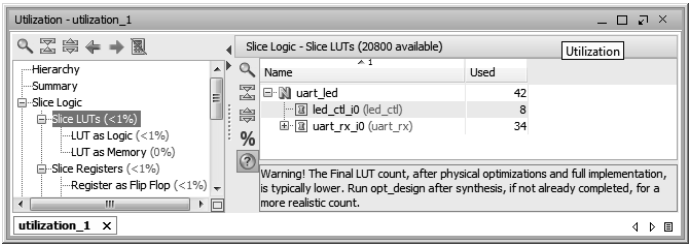


Рисунок 3.15 – Використання ресурсівнизькорівневих модулів *Basys3*

Натисніть "*ReportPower*" в меню "*SynthesizedDesign*", і натисніть кнопку *OK*, щоб згенерувати звіт про енергоспоживання зі значеннями за замовчуванням (рис. 3.16).

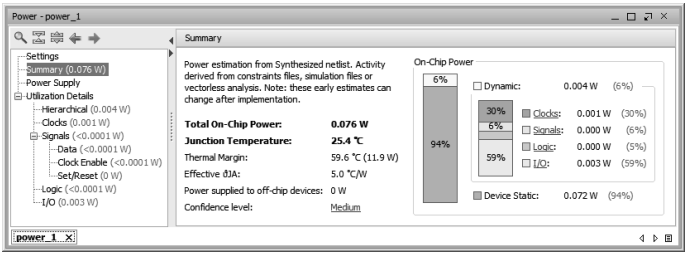


Рисунок 3.16 - Очікуване енергоспоживання *Basys3*

4. Створення контрольної точки для аналізу результатів без проходження фактичного процесу складання.

Натисніть на "*File>WriteCheckpoint*", для збереження схему, щоб можна було відкрити її пізніше для подальшого аналізу.

З'явиться діалогове вікно, яке відображає ім'я файлу за замовчуванням в цій папці проекту (рис. 3.17).

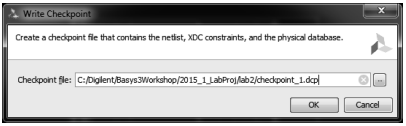


Рисунок 3.17 - Створення контрольної точки

Натисніть *OK*.

5. Зміна параметрів збірки для згладжування схеми. Повторна збірка схеми і аналіз результатів.

Натисніть на "Project Settings" в меню "Project Manager" і виберіть "Synthesis".

Натисніть на меню "flatten_hierarchy" і виберіть "full" (рис. 3.18). Натисніть *OK*.

З'явиться "CreateNewRun" діалогове вікно з питанням про те, чи хочете ви створити новий прогін схеми, оскільки настройки були змінені (рис. 3.19). Натисніть кнопку "Yes". Змініть назву з synth_2 на synth_flatten і натисніть кнопку *OK*.

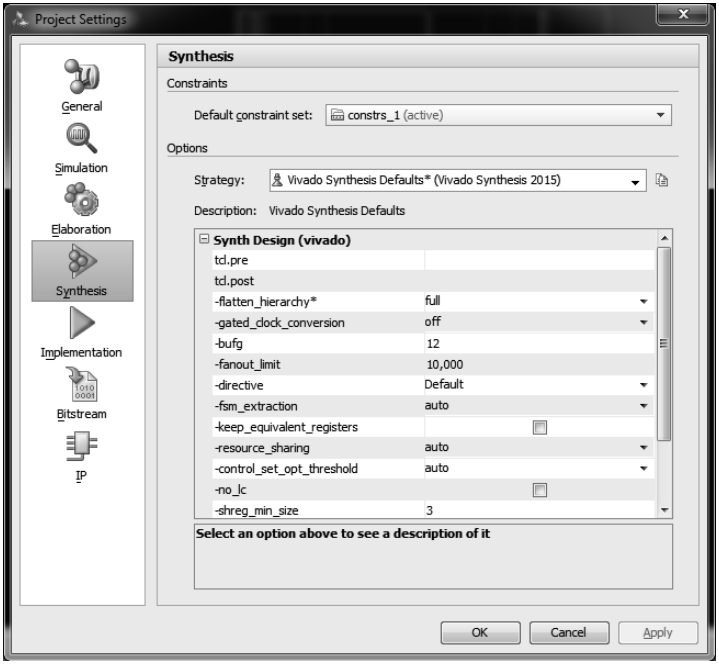


Рисунок 3.18 - Згладжування схеми

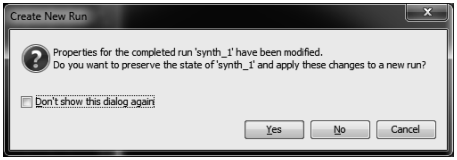


Рисунок 3.19 - У діалоговому вікні "CreateNewRun"

Натисніть "*RunSynthesis*", щоб зібрати схему. Діалогове вікно *ReloadDesign* може знову з'явитися. Натисніть кнопку "*Cancel*".

Натисніть кнопку *OK*, щоб відкрити зібраний проект, коли процес складання буде завершено.

Натисніть на "Schematics" в меню "OpenSynthesizedDesign" на панелі FlowNavigator для перегляду зібраної схеми в

схематичному вигляді (рис. 3.20). Зверніть увагу на те, що конструкція повністю згладжується.

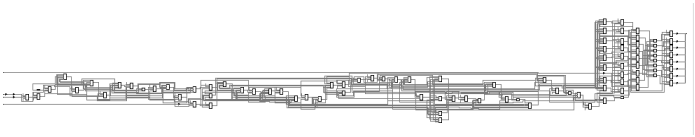


Рисунок 3.20 - Згладжена схема

Натисніть на "*ReportUtilizatio*n" і зауважте, що ієрархічне використання ресурсів більше недоступно. Також зверніть увагу, що значення "SliceRegisters" - 48.

6. Створення контрольної точки для того, щоб проаналізувати результати без проходження фактичного процесу складання.

Натисніть на "*File>WriteCheckpoint*" для збереження обробленої схеми, щоб відкрити її пізніше для подальшого аналізу.

З'явиться діалогове вікно, яке відображає ім'я файлу за замовчуванням (checkpoint_2.dcp) в цій папці проекту.

Натисніть *OK*. Закрийте проект.

Крок 4. Прочитайте контрольну точку

Відкрийте збережену раніше контрольну точку (checkpoint_1) для того, щоб проаналізувати результати без проходження самого процесу складання.

Виберіть "*File>OpenCheckpoint*" на екрані "*GettingStarted*".

Перейдіть в папку *<2015_1_artix7_labs> \ lab2* і виберіть *checkpoint_1*. Натисніть кнопку *OK*.

Якщо схема не відкривається за замовчуванням, на вкладці "*Netlist*", виберіть об'єкт верхнього рівня, uart_led, клацніть правою кнопкою миші і виберіть "Schematic". Ви побачите ієрархічні модулі. Ви можете двічі клацнути на будь-якому з модулів першого рівня і побачите основні модулі. Ви також можете вибрати будь-який модуль нижчого рівня на вкладці "Netlist", клацніть правою кнопкою миші і виберіть "Schematic", щоб побачити відповідний модульний рівень схеми.

На вкладці "*Netlist*", виберіть модуль верхнього рівня, uart_led, натисніть на нього правою кнопкою миші і виберіть "*ShowHierarchy*". Ви побачите, як модулі ієрархічно пов'язані між собою.

Натисніть на *"Tools>Timing>ReportTimingSummary*" і натисніть на кнопку *OK*, щоб побачити звіт, який ви бачили раніше.

Виберіть "*Tools>Report>ReportUtilization*" натисніть кнопку *OK*, щоб побачити звіт про використання ресурсів, який ви бачили раніше

Виберіть *"File>OpenCheckpoint*", перейдіть в *<2015_1_artix7_labs> \ lab2 і виберіть checkpoint_2.*

Натисніть "*No*", щоб залишити *Checkpoint_1* відкритим. Це відкриє друге вікно *Vivado GUI*.

Якщо схема не працює за замовчуванням, у вкладці "Netlist", виберіть модуль верхнього рівня, uart_led, клацніть правою кнопкою миші і виберіть "Schematics". Ви побачите згладжену схему.

Ви можете генерувати необхідні звіти у цій контрольній точці за вашим бажанням.

Закрийте програму *Vivado*, натиснувши "*File>Exit*" і натисніть кнопку *OK*.

У цій роботі ви застосували часові обмеження і зібрали схему. Ви дивилися різні звіти про збірку схеми. Ви створили контрольні точки і відкривали їх, щоб виконати аналіз, який ви робили під час процесу реалізації. Також ви побачили результат зміни параметрів збірки.

3.5 Контрольні запитання

1. Що таке критичний шлях?
2. Де починається та закінчується критичний шлях?
3. Як виконати перевірку всіх статичних часових шляхів?
3. Як виконати перевірку затримки?
4. Як створити і провести RTL-аналіз?
5. Яку інформацію надає звіт про синхронізацію?

6. Яку інформацію надає звіт про використання ресурсів *Basys3*?

7. Як переглянути інформацію про очікуване енергоспоживання *Basys3*?

8. Як створити контрольну точку?

9. Як зберегти та прочитати контрольну точку?

10. Як виконати згладжування схеми?

4 РЕАЛІЗАЦІЯ СХЕМИ

Ця робота є продовженням попередньої. Ви будете виконувати статичний часовий аналіз. Ви будете реалізовувати проект з настройками за замовчуванням і генерувати потік бітів. Тоді ви відкриєте апаратний засіб і запрограмуєте *FPGA* . Ви будете використовувати бортовий *UART* плати *Basys3*, щоб перевірити своюсхему.

Після виконання цієї роботи, ви зможете:

— реалізувати схему;

— генерувати різні звіти та аналізувати результати;

— виконувати статичний часовий аналіз;

— формувати потік бітів і перевірити функціональні можливості в апаратних засобах.

4.1 Порядоквиконання роботи

Крок 1. Створення проекту Vivado

1. Запустіть Vivado і відкрийте проект lab2. Збережіть проект як lab3 в каталозі <2015_1_artix7_labs> переконавшись, що функція створення підкаталогу обрана. Встановіть налаштування flatten_hierarchy harebuilt. Створіть новий запуск синтезу з іменем synth_2.

Посилання на *<2015_1_artix7_labs>* є заповнювачем для директорії *C:\Digilent\Basys3Workshop\ 2015_1_artix7_labs і <2015_1_artix7_sources>* є заповнювачем для директорії C:\Digilent*Basys3*Workshop\ 2015_1_artix7_sources. Посилання на <board> означає *Basys3*.

Запустіть Vivado при необхідності і відкрийте або проект lab2 (lab2.xpr), який ви створили в попередній лабораторній роботі або проект lab2 в каталозі labsolution використовуючи посилання Openproject на сторінці GettingStartedpage.

Виберіть "*File>Save Project As* ...", щоб відкрити "*Save Project As dialog box"*. Введіть lab3 в якості назви проекту. Переконайтеся, що опція "*Create Project Subdirectory"* увімкнена,

шлях до директорії проекту *<2015_1_artix7_labs>* і натисніть кнопку *OK*.

Натисніть на Synthesis Settingsна панелі ‘*Flow Navigator”.*

Переконайтеся, що flatten_hierarchy встановлений в режим rebuilt, що дозволяє ієрархії схеми бути збереженою для синтезу, а потім відновлений, що є більш корисним для аналізу проекту, так як багато логічних посилань будуть збережені (рис. 4.1).

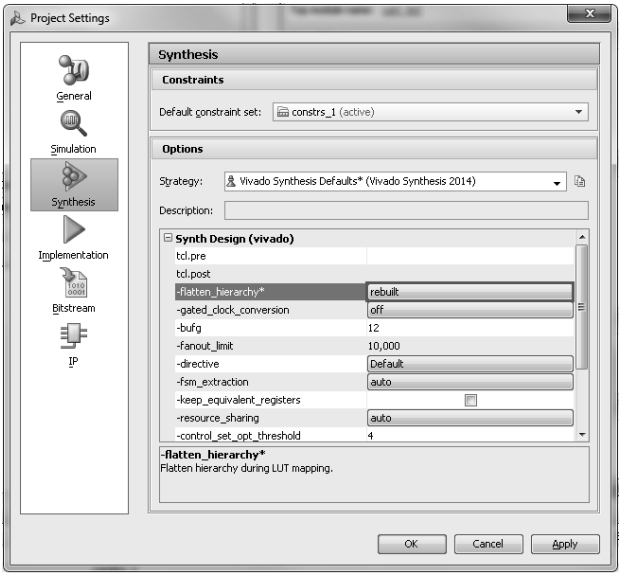


Рисунок 4.1 - Встановлені hierarchy на rebuilt

Натисніть *OK*.

Вікно CreateNewRun з’явиться, питаючи вас, чи потрібно створювати новий запуск. Натисніть кнопку *Yes*, а потім *OK*, щоб створити новий запуск з ім’ям **synth_2**.

2. *Синтезувати схему. Згенерувати часовий перелік і проаналізувати схему.*

Натисніть на RunSysnthesis під завданням *Synthesis* на панелі *FlowNavigator*.

Процес синтезу запуститься на uart_led.vhd і всіх його ієрархічних файлах. Коли процес буде завершений, буде відображено діалогове вікно SynthesisCompleted з трьома варіантами.

Виберіть опцію *OpenSynthesizedDesign*і натисніть кнопку *OK*, щоб побачити результат синтезу.

Натисніть на *ReportTimingSummary* під *SynthesizedDesign* завданнями на панелі *FlowNavigator*.

Залиште всі налаштування без змін та натисніть ***OK***, щоб згенерувати стандартний часовий звіт, *timing_1* (рис. 4.2).

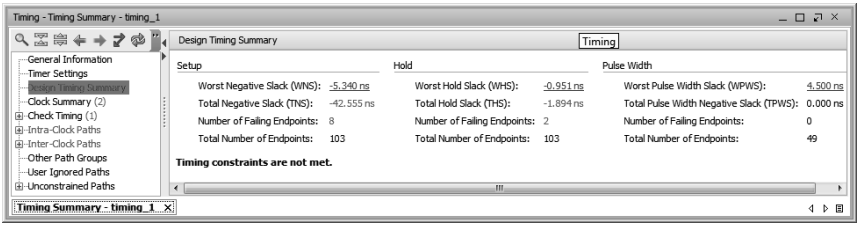


Рисунок 4.2 - Часовий звіт для *Basys3*

Натисніть на посилання біля *WorstNegativeSlack* (*WNS*) та побачите 8 помилкових шляхів.

Зробіть подвійне натискання на Path 23, щоб побачити детальний огляд шляху. Звіт про цей шлях показує чотири розділи: «*Summary*», «*Source Clock Path», «Data Path», ma«Destination Clock Path».*

Оберіть «Path 23» на панелі«timingsummary», або «*Pathsummaryview*», натисніть на ньому правою кнопкою та оберіть «*Schematic*». Буде показаний схематичний розбір для результуючих даних (рис. 4.3).

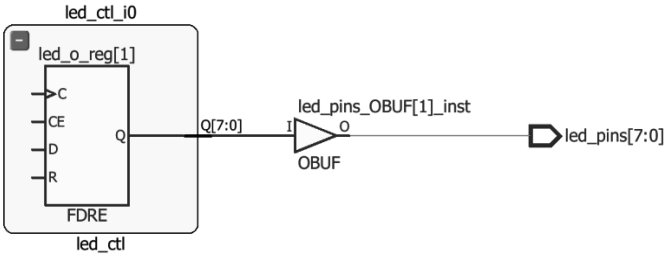


Рисунок 4.3 - Шлях результуючих даних

Для того, щоб побачити, як SourceClockPath зроблена у схематичній формі, зробіть подвійний клік на лівому кінці значку *CFDRE* на схемі.

Це п*ОК*аже зв'яз*ок*між *BUFGma* С *портомFDRE*.

Аналогічним чином, подвійний клік на лівому кінці BUFG дозволить побачити шлях між *IBUF* та *BUFG* (рис. 4.4).

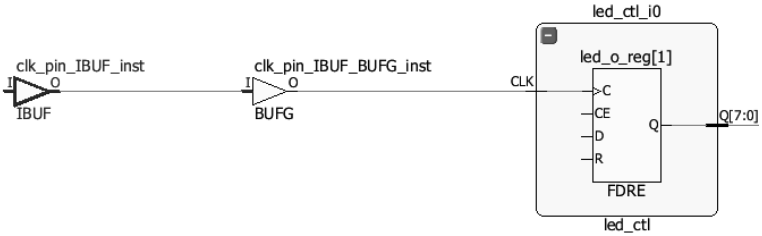


Рисунок 4.4 - Джерело для синхронізації порту *FDRE*

В решті решт, зробіть подвійний клік на значку входу *IBUF*, щоб побачити шлях між входом та *IBUF* (рис. 4.5).

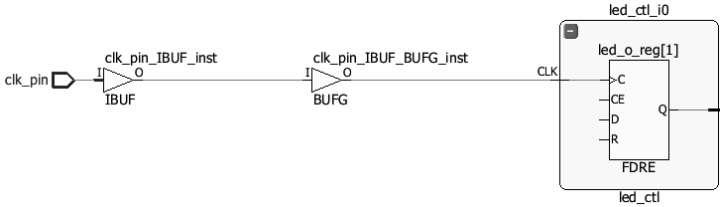


Рисунок 4.5 - Схематичний вигляд шляху джерела тактування

Це відноситься до «*SourceClockPath*»у часовому звіті (рис. 4.6).

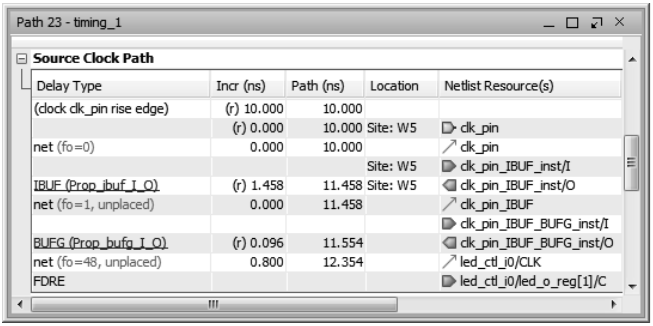


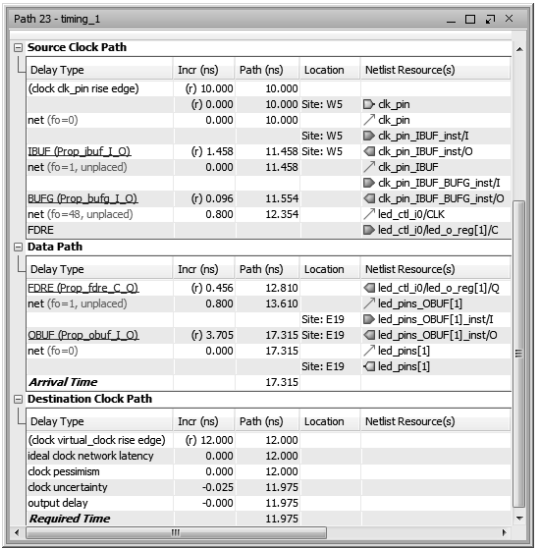
Рисунок 4.6 - Джерело тактування для *Basys3*

Оскільки віртуальнетактування повільніше (12 ns), ніж період clk_pin(10 ns), затримка передачі даних включає період тактування джерела clk_pin (рис. 4.7).

Змініть конструкції обмеження для обмеження віртуального тактового періоду до 10 нс. Проведіть повторний синтез схеми і аналіз результатів.

Натисніть *EditTimingConstraint*s під *SynthesizedDesign*.

З’явиться інтерфейс часового обмеження, показуючи, що схема має два createclock, 4 вхідних та одне вихідне обмеження. Він також покаже обмеження в текстовій формі в секції AllConstraints (рис. 4.8).



Path 23 - timing_1

Source Clock Path

Delay Type	Incr (ns)	Path (ns)	Location	Netlist Resource(s)
(clock clk_pin rise edge)	(r) 10.000	10.000		
	(r) 0.000	10.000	Site: W5	clk_pin
net (fo=0)	0.000	10.000		clk_pin
			Site: W5	clk_pin_IBUF_inst/I
IBUF (Prop_ibuf_I_O)	(r) 1.458	11.458	Site: W5	clk_pin_IBUF_inst/O
net (fo=1, unplaced)	0.000	11.458		clk_pin_IBUF
				clk_pin_IBUF_BUFG_inst/I
BUFG (Prop_bufg_I_O)	(r) 0.096	11.554		clk_pin_IBUF_BUFG_inst/O
net (fo=48, unplaced)	0.800	12.354		led_ctl_i0/CLK
FDRE				led_ctl_i0/led_o_reg[1]/C

Data Path

Delay Type	Incr (ns)	Path (ns)	Location	Netlist Resource(s)
FDRE (Prop_fdre_C_Q)	(r) 0.456	12.810		led_ctl_i0/led_o_reg[1]/Q
net (fo=1, unplaced)	0.800	13.610		led_pins_OBUF[1]
			Site: E19	led_pins_OBUF[1]_inst/I
OBUF (Prop_obuf_I_O)	(r) 3.705	17.315	Site: E19	led_pins_OBUF[1]_inst/O
net (fo=0)	0.000	17.315		led_pins[1]
			Site: E19	led_pins[1]
Arrival Time		17.315		

Destination Clock Path

Delay Type	Incr (ns)	Path (ns)	Location	Netlist Resource(s)
(clock virtual_clock rise edge)	(r) 12.000	12.000		
ideal clock network latency	0.000	12.000		
clock pessimism	0.000	12.000		
clock uncertainty	-0.025	11.975		
output delay	-0.000	11.975		
Required Time		11.975		

Рисунок 4.7 - Найгірше помилкове джерело для *Basys3*

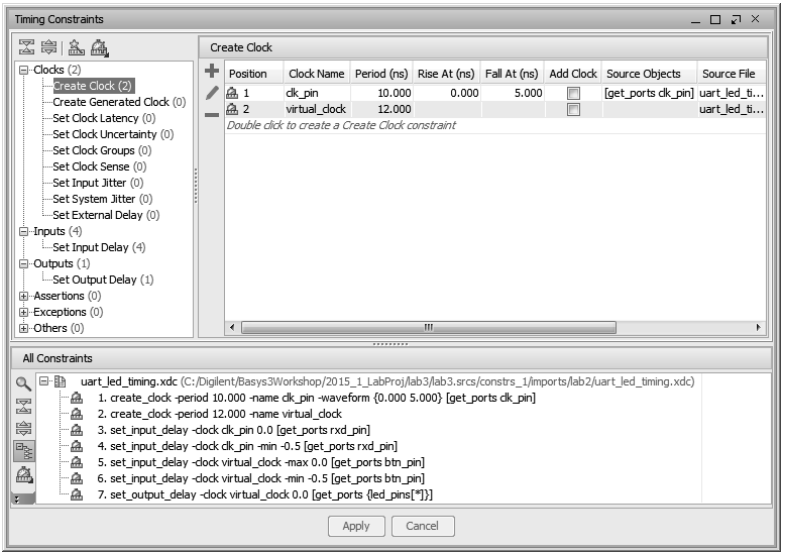


Рисунок 4.8 - Часові обмеження, що показують 12 нс визначеного Virtual Clock періоду

Натисніть в клітці **Period***virtual_clock*та змініть період з 12 на 10 нс.

Натисніть *Apply***.**

Зверніть увагу, що оскільки змінилосячасове обмеження, виводиться попередження в консолі, що потрібно заново запустити звіт.

Натисніть *Rerun***.**

Зверніть увагу на те, що часові порушення зникли. Тим не менш, є ще 2 помилкові шляхи для *Hold* (рис. 4.9).

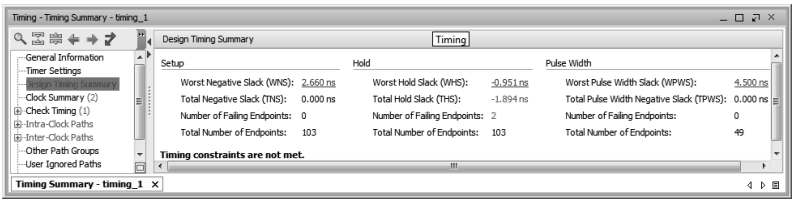


Рисунок 4.9 - Встановлення часових параметрів для *Basys3*

Натисніть на посилання *WHS* для того, щоб переглянути шляхи.

Зробіть подвійне натискання на першому шляху для перегляду часових композицій. Зверніть увагу на те, що затримка шляху clock не включає в себе весь період тактового імпульсу.

Виберіть File > Save Constraints (Файл>Зберегти обмеження)…Натисніть *OK*.

Зверніть увагу на те, що статус SynthesisOut-of-Date відображається у верхньому правому куті.

Крок 2. Реалізація схеми.

1**.** Виконайте реалізацію після збереження запуску синтезу. Виконайте часовий аналіз.

У вкладці *DesignRuns* натисніть правою кнопкою миші на synth_2 та оберіть ResetRuns. Переконайтесь, що створені файли видалені. Натисніть Reset.

Натисніть накнопку *CloseDesignв* рядку стану. Якщо буде запропоновано, не зберігайте нічого.

Натисніть на *"Run Implementation"* на панелі "*Flow Navigator"*.

Натисніть *OK*, якщо буде запропоновано, для того, щоб запустити синтез першим перед запуском самого процесу реалізації. Коли реалізація закінчиться, ви побачите діалогове вікно з трьома можливими варіантами.

Оберіть "*Open Implemented Design"* та натисніть *OK*.

2. Перегляд обсягу FPGA ресурсів, що споживаютьсясхемою за допомогою Звіту про використання (ReportUtilization).

В панелі *FlowNavigator* оберіть "OpenImplementedDesign>ReportUtilization". Далі ви побачите діалогове вікно *ReportUtilization*. Натисніть *OK*.

Звіт про використання відображається в нижній частині Vivado IDE. Ви можете вибрати будь-який з ресурсів зліва для перегляду їх відповідного використання.

Оберіть *SliceLUTs* для перегляду, скільки ресурсів використовують модулі (рис. 4.10).

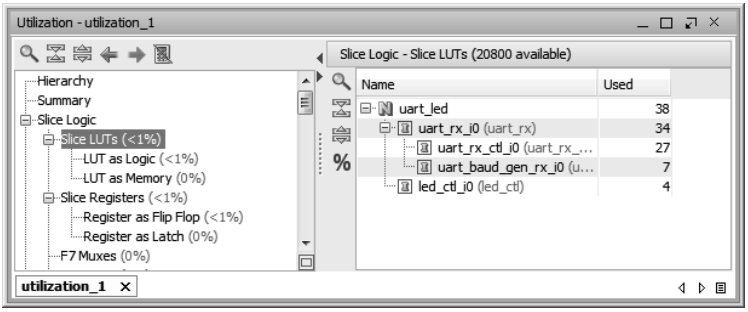


Рисунок 4.10 - Використання ресурсів для *Basys3*

3. Створення зведеного звіту про витрати часу.

В панелі «*Flow Navigator nid "Implementation > Implemented Design*», натисніть ***Report Timing Summary***.

Буде показане діалогове вікно «*Report Timing Summary»*.

Не чіпайте ніякі налаштування та натисніть ***OK***, щоб створити звіт (рис. 4.11).

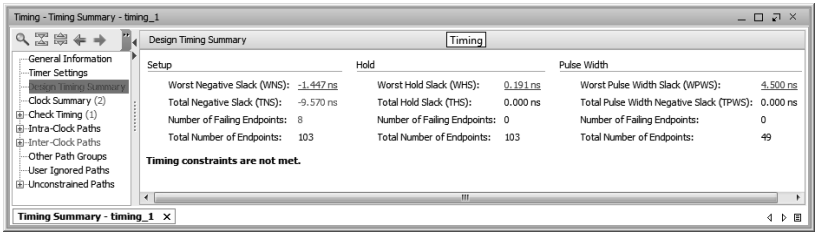


Рисунок 4.11 - Звіт про витрати часу, який показує порушення часових обмежень для *Basys3*

Натисніть на посилання WHSдля перегляду детального звіту, щоб визначити некоректні записи шляху.

Двічі натисніть на першому неправильному шляху для перегляду, чому саме він некоректний (рис. 4.12).

У порівнянні з затримками зі звіту про синтез, мережеві затримки є фактичними затримками (а не за прогнозованими цифрами). Затримка шляху даних більше, ніж затримка кінцевого тактового шляху дає негативну слабину (порушення).

На даний момент ми можемо ігнорувати це порушення, оскількизміна LEDдисплею за декілька наносекунд неможе спостерігатися людським оком. Ми також можемо змінити затримку виходу на -2 нс і задовольнити часові параметри.

Оберіть "*Implemented Design>Edit Timing Constraints"* панель *Flow Navigator.*

Оберіть вхід на лівій панелі "*Set Output Delay"* та змініть значення "*Delay Value"* на -2.000 нс.

Натисніть *Apply*.

Натисніть на посилання *Rerun* для повторного запуску часового звіту.

Зверніть увагу, що порушення часу в шляху *Intra-clock* пропали.

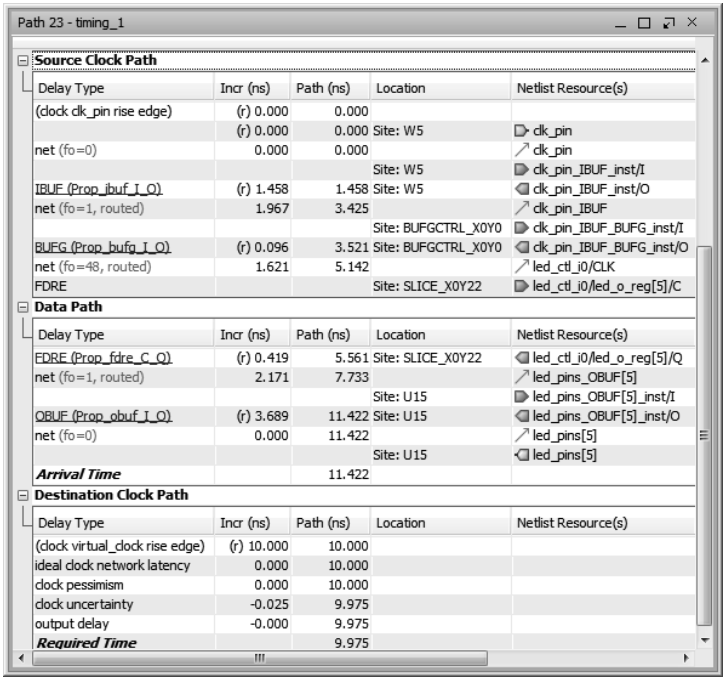


Рисунок 4.12 - Затримка першого некоректного шляху для *Basys3*

Розгорніть папку *Intra-Clock* зліва, розгорніть *clk_pin*, і виберіть групу Налаштувань (Setup), щоб побачити спис*ок*10 найгірших затрим*ок* випадку з правого боку.

Двічі натисніть на будь-якому шляху для перегляду, як саме він зроблений. Також натисніть правою кнопкою на ньому і виберіть **Schematic** (рис. 4.13).

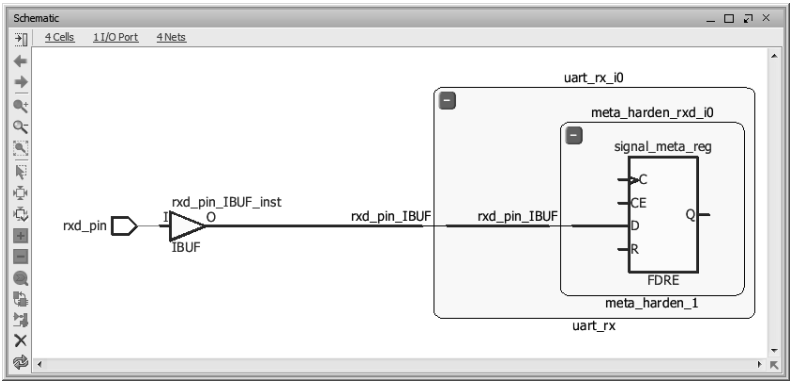


Рисунок 4.13 - Схематичний вид затримки найгіршого шляху

Натисніть на вкладці *Device* та ви побачите виділений шлях. Ви повинні приблизити, щоб побачити цей шлях (рис. 4.14).

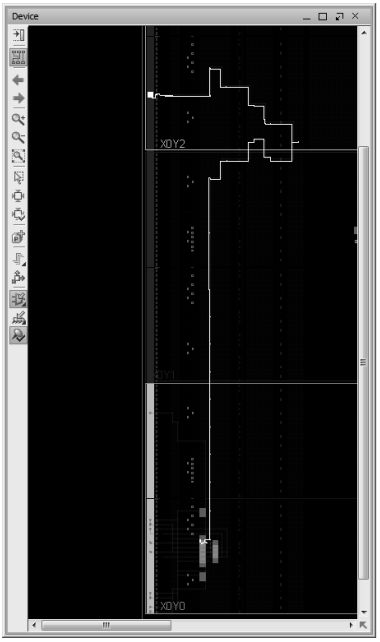


Рисунок 4.14 - Шлях, відображений у перегляді *Device*

Оберіть "*Implemented Design > Report Clock Networks”*.

Натисніть *OK*.

Звіт Clock Networks буде відображатися на панелі Console, що п*ОК*аже два часових мережевих входи.

Оберіть *clk_pin* вхід та відзначте обрані мережі у перегляді *Device* (рис. 4.15).

Часові мережі поширені у багатьох часових поясах.

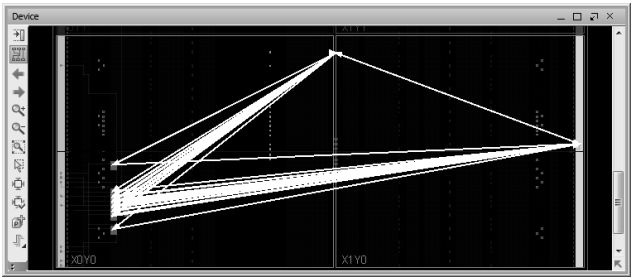


Рисунок 4.15 - Часові мережі для *Basys3*

Крок 3. Створення потоку бітів

Створення потоку бітів.

У панелі *FlowNavigator* під *ProgramandDebug* натисніть Генерувати Потік Бітів (*GenerateBitstream*) (рис. 4.16).

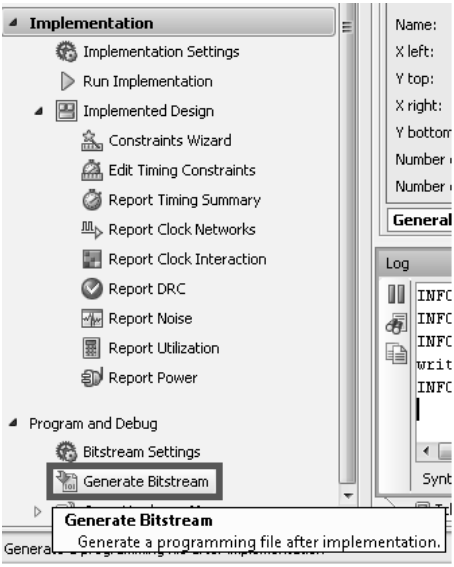


Рисунок 4.16 - Створення потоку бітів

Натисніть кнопку *Save* (Зберегти), щоб зберегти обмеження, так як часові обмеження були змінені, а потім натисніть кнопку *Yes* (Так), щоб скинути історію запусків і повторно запустити всі процеси.

Команда *write_bitstream*буде запущена (ви можете перевірити це у консолі).

Натисніть *Cancel* (Відміна), коли створення потоку бітів завершено.

Крок 4. Перевірка функціональності

1. Підключіть плату і увімкніть її. Відкрийте апаратний сеанс, і запрограмуйте *FPGA* .

Переконайтеся, що кабель мікро-USB підключений до роз'єму *JTAG* PROG (поруч з роз'ємом живлення). Переконайтеся, що перемичка на платі встановлена, щоб вибрати потужність USB (JP2 для *Basys3*).

Оберіть *Open Hardware Manager*та натисніть *OK*.

Вікно HardwareManager відкриється, вказуючи "непідключений" статус (unconnected).

Натисніть на посилання *Open a new hardware target*.

Ви також можете натиснути на посилання *Openrecenttarget*, якщо плата була раніше обрана (рис 4.17).

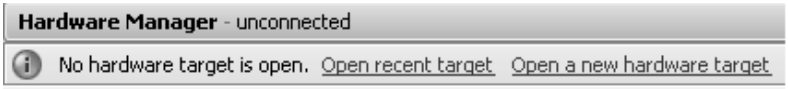


Рисунок 4.17 - Відкриття нового апаратного напрямку

Натисніть *Next* для перегляду форми "*Hardware Server Settings"*.

Натисніть *Next* з обраним пунктом "*Target Hardware"*.

JTAG -кабель, який використовує digilent_plugin повинен бути виявленим і ідентифікованим як апаратний напрям*ОК*. Він також п*ОК*аже апаратні пристрої, виявлені в ланцюзі.

Натисніть *Next* та *Finish*.

Статус "*Hardware Session"* змінюється від «непідключеного» на ім'я серверу та пристрій підсвітиться. Також зверніть увагу, що статус вказує на те, що він не запрограмований.

Виберіть пристрій у властивостях обладнання пристроїв (*HardwareDeviceProperties*), і переконайтеся, що uart_led.bit обраний в якості файлу програмування на вкладці Загальні (General).

2.Запустіть програму-емулятор терміналу, таку як *TeraTerm* або *HyperTerminal*. Виберіть відповідний *COM* порт (ви можете знайти правильний номер *COM* за допомогою панелі управління). Встановіть *COM*-порт для зв'язку на швидкість передачі 115200. Запрограмуйте *FPGA* і перевірте працездатність.

Запустіть програму-емулятор терміналу, такі як TeraTerm або HyperTerminal.

Виберіть відповідний *COM* порт (ви можете знайти правильний номер *COM* за допомогою панелі управління).

Встановіть *COM*-порт для зв'язку на швидкість передачі 115200.

Натисніть правою кнопкою миші на вході *FPGA* у вікні Hardware та оберіть *ProgramDevice*…

Натисніть на кнопку Програмувати (*Program*).

Буде завантажений біт-файл програмування, іколи *FPGA* буде запрограмоване, підсвітиться «*DONE*».

Введіть кілька символів у вікні емулятора термінала і побачите відповідний бітову послідовність еквіваленту ASCII-коду, що відображається на світлодіодах.

Натисніть і утримуйте BTNU і побачите, як верхні чотири біта міняються місцями з нижніми чотирма бітами на світлодіодах (LEDs).

Після завершення перевірки закрийте програму емуляції терміналу і вимкніть живлення плати.

Оберіть "*File > Close Hardware Manager"*. Натисніть *OK*.

Закрийте програму *Vivado* натиснувши "*File>Exit"* та кнопку *OK*.

У цій роботі ви дізналися про різні види звітів, доступних для дизайнерів в *Vivado IDE*. Ви мали можливість вивчити основні інструменти аналізу проекту, включаючи Schematicviewer, властивості затримки шляху, засоби перегляду звітів і перегляду пристроїв. Ви також дізналися про основні параметри звіту щодо часу. Ви перевірили функціональні можливості в апаратних засобах, набравши символи на хост-машині і побачили, як *LED* шаблон змінюється.

4.2 Контрольні запитання

1. Як отримати часовий звіт для *Basys3*?
2. Як отримати схематичний розбір для результуючих даних?
3. Як отримати схематичний вигляд шляху джерела тактування?
4. Як встановити часові обмеження?
5. Як виконати реалізацію після збереження запуску синтезу?
6. Як створити зведений звіт про витрати часу?
7. Як створити пото*к*бітів?
8. Як запрограмувати *FPGA* ?
9. Який функціонал містить програма-емулятор терміналу?
10. Що таке Schematic viewer?

5 ВИКОРИСТАННЯ *IP* КАТАЛОГУ ТА *IP* ІНТЕГРАТОРА

У цій роботі ви навчитеся використовувати *IP* каталог для генерування хронометричного ресурсу. Ви навчитеся форматувати згенерований хронометр в наданий генератор сигналів. Також, ви навчитеся використовувати *IP* інтегратор, щоб генерувати ядро *FIFO* і потім використовувати його в *HDL* схемах.

5.1 Основні теоретичні відомості

У цій роботі ви будете використовувати програмований генератор сигналів.

У даній схемі він представлений як незалежний пристрій, який управляється за допомогою *ПК* (або іншого термінального пристрою) з використанням протоколу *RS-232*. Описана в цій роботі схема реалізує з'єднання для *RS-232*, генератор сигналів і підключення до зовнішнього DAC, а також простий парсер для реалізації невеликої кількості "команд" для управління генератором сигналів.

Генератор сигналів створює довідкову таблицю, що складається з 1024 зразків, які займають по 16 біт і *ОЗП* кожен. Також, генератор сигналів створює 3 змінні:

- **nsamp**: Число зразків, які використовуються для виведення генератора. Повинно бути між 1 і 1024.

- **prescale**: Переддільник для зразка тактового об'єкта. Мінімум 32.

speed: Швидкість (або частота) для виведення зразків (в од.як у тактового переддільника).

Генератору сигналів може бути дано вказівку направити необхідну кількість зразків один раз, циклом від 0 до nsamp-1 раз, а потім зупиняється, або безперервно, де безперервно проходить циклом все nsamp зразки. Якщо ця функція включена, незалежно від режиму, генератор сигналів пошле один зразок на DAC всі тактові цикли clk_tx. Вміст оперативної пам'яті, а також трьох змінних, можуть бути змінені за допомогою команд, що посилають ся по RS-232 з'єднанню, яке може змінювати режим генератора

сигналів. Генератор сигналів буде генерувати відповіді для всіх команд.

Є три тактових домени для цієї схеми: clk_rx, clk_tx і clk_samp. Модуль тактового генератора створює всі необхідні ресурси, які потрібні для створення цих трьох хронометрів. Всі три хронометра є похідними від одного входу синхронізації, що приходять на clk_pin з частотою 100 МГц. Блок-схема показана на рис. 5.1.

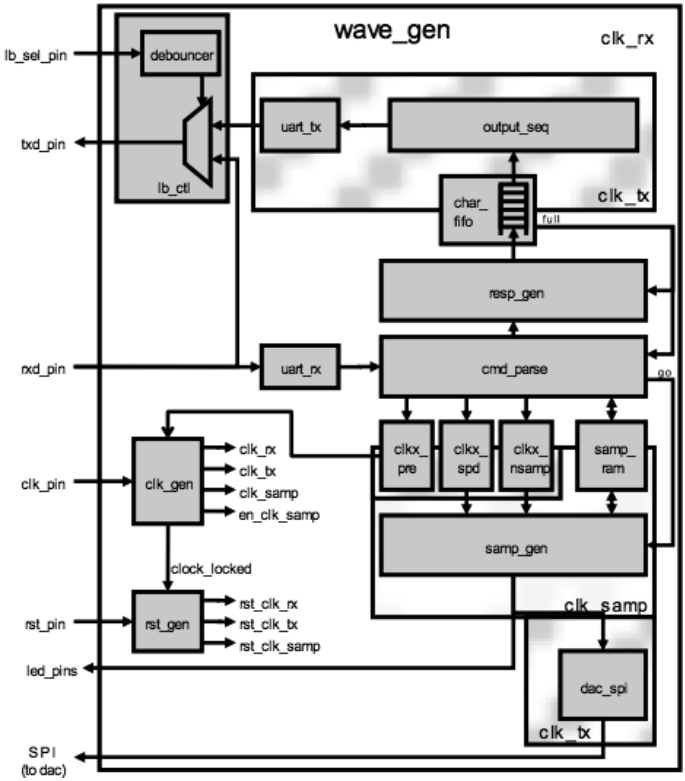


Рисунок 5.1 - Бл*ок*схема

5.2 Порядоквикнання роботи

Крок 1. Створення проекту в Vivado IDE

Запустити *Vivado* і створити проект для пристрою XC7A35TCPG236-1 (*Basys3*) з використанням *VHDL*. Використовуйте надані вхідні файли *VHDL*, вказану конкретну *IP*-адресу і XDC файли з папки <2015_1_artix7_sources> \ lab4 \

Відкрийте *Vivado*, вибравши Пуск> Всі програми>XilinxDesignTools>Vivado 2015,1> 2015,1 Vivado

Натисніть "Createnew Project" для запуску майстра. З'явиться діалогове вікно *"CreateNewVivado Project*". Натисніть "Next".

Натисніть кнопку "Browse" в поле "Project location" наформі створення нового проекту, перейдіть в папку <2015_1_artix7_labs>, і натисніть кнопку "Select".

Введіть *lab4* вполе "*Project Name*". Переконайтеся, що прапорець "Create Project Subdirectory" встановлено. Натисніть "Next".

Виберіть опцію *RTL* Project наформі "Project type", і натисніть кнопку "Next".

За допомогою меню, що випадає, виберіть *VHDL* як "TargetingLanguage" і Mixed як "Simulator Language" в формі "AddSources".

Натисніть на кнопку +, потім виберіть "AddFiles ...", перейдіть до каталогу<2015_1_artix7_sources> \ lab4, виберіть всі файли *VHDL*, натисніть кнопку *OK*, а потім натисніть кнопку "Next".

У формі "Add *IP*", натисніть на кнопку +, потім виберіть "AddDirectories ...," перейдіть до каталогу<2015_1_artix7_sources> \ lab4 \ *IP* \ <плата>. Виділіть папку і натисніть кнопку "Select".

Натисніть кнопку "Next", щоб перейти до форми додавання обмежень.

Натисніть на кнопку +, потім виберіть "AddFiles ...". Перейдіть до каталогу <2015_1_artix7_sources> \ lab4, виберіть wave_gen_timing.xdc і відповідний wave_gen_pins_ <плата>.xdc і натисніть кнопку "Open".

У формі "DefaultPart", за допомогою опції Parts і різних випадаючих меню секції "Filters", виберіть XC7A35TCPG236-1 для *Basys3*. Натисніть "Next".

Натисніть кнопку "Finish", щоб створити проект Vivado.

На панелі "Sources" розгорніть пункт "DesignSources" і при необхідності wave_gen, і двічі клацніть на елементі clk_gen_i0. Перегляньте файл і зверніть увагу, що на рядку 85 оголошується тактова частота ядра і на рядку 103 створюється екземпляр тактового ядра.

Крок 2. Створення та ініціалізація генератора тактових імпульсів

1. Запустіть майстер тактування з *IP* каталогу *Vivado* і згенеруйте тактовее ядро з вхідною частотою 100,00 МГц і двома вихідними хронометрами по 100.000 МГц кожен.

Натисніть на *IP*-каталог на панелі FlowNavigator. *IP* каталог відкриється на допоміжній панелі.

Двічі клацніть на елементі "ClockingWizard". *IP* каталог може виглядати, як показано на рис. 5.2.

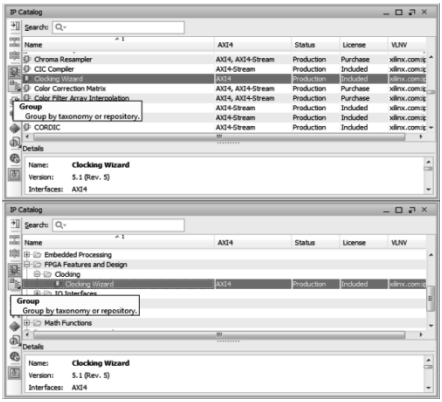


Рисунок 5.2 - Тактовий майстер

Змініть ім'я ядра на clk_core. Переконайтеся, що прапорець "Show disabled ports" знято. Переконайтеся, що основна частота

вхідного тактового сигналу *становить* 100,000 МГц і використовуваним примітивом є *MMCM* (рис. 5.3).

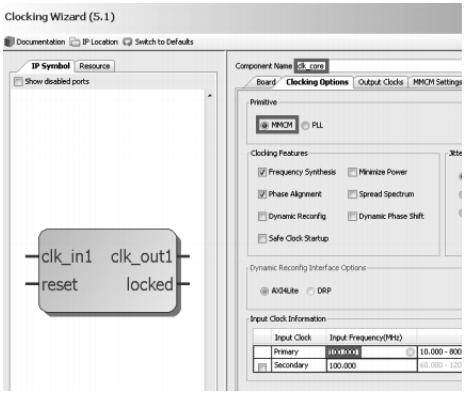
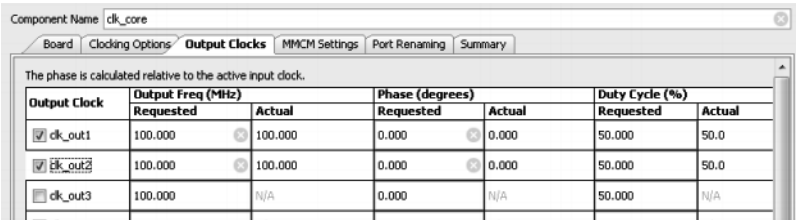


Рисунок 5.3 - Тактовий майстер

Виберіть вкладку "*OutputClocks*". Натисніть на прапорець, щоб включити другий тактовий вихід (рис. 5.4).



Output Clock	Output Freq (MHz)		Phase (degrees)		Duty Cycle (%)	
	Requested	Actual	Requested	Actual	Requested	Actual
clk_out1	100.000	100.000	0.000	0.000	50.000	50.0
clk_out2	100.000	100.000	0.000	0.000	50.000	50.0
clk_out3	100.000	N/A	0.000	N/A	50.000	N/A

Рисунок 5.4 - Налаштування тактових виходів

Натисніть на вкладку Summary і перегляньте інформацію (рис. 5.5).

Натисніть кнопку *OK*, щоб побачити форму "Generateoutputproducts" (рис. 5.6).

Натисніть на Generate для створення вхідних продуктів, включаючи шаблон для створення екземпляра.

2. Створення примірника згенерованого тактового ядра.

Виберіть вкладку "*IP Sources*" в панелі "*Sources*".

Розгорніть вкладку "*IP* (2)". Зверніть увагу на два записи *IP*. *IP* char_*FIFO* є ядром, яке було додано при створенні проекту. Друге ядро clk_core – те, яке ви згенерували.

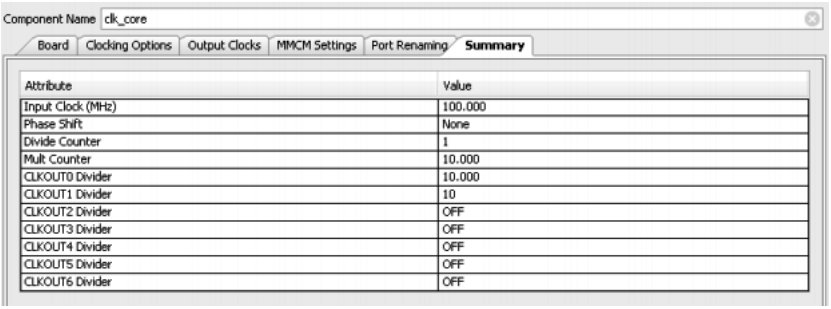


Рисунок 5.5 - Вкладка Summary

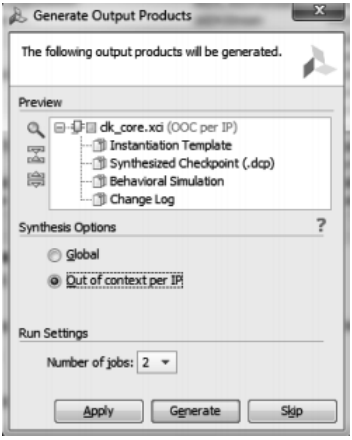


Рисунок 5.6 - Форма "Generate output products"

Розгорніть вкладку "*clk_core>InstantiationTemplate*" і двічі клацніть на clk_core.vho, щоб побачити шаблон для створення екземпляра.

Скопіюйте рядки з 66 по 88 і вставте їх в область декларації архітектури clk_gen.

Скопіюйте рядки з 89 по 104 і вставте їх в тіло архітектори clk_gen.

Змініть ім'я екземпляра і імена мереж, як показано на рисунку нижче, щоб відповідати іменам існуючих сигналів в схемі (рис. 5.7).

```
-- Instantiate clk_core - generated by the Clocking Wizard
-- The following code must appear in the VHDL architecture
-- body. Substitute your own instance name and net names.
------------ Begin Cut here for INSTANTIATION Template ----- INST_TAG
clk_core_i0 : clk_core
  port map (

  -- Clock in ports
  clk_in1 => clk_pin,
 -- Clock out ports
  clk_out1 => clk_rx_int,
  clk_out2 => clk_tx_int,
 -- Status and control signals
  reset => rst_i,
  locked => clock_locked
 );
-- INST_TAG_END ----- End INSTANTIATION Template ----------
```

Рисунок 5.7 - Зміна імені примірника і мережевих з'єднань

Виберіть "File>SaveFile", щоб зберегти clk_gen.vhd

Виберіть вкладку "Hierarchy" і розкрийте вкладку "wave_gen>clk_gen_i0" та переконайтеся, що clk_core.xci знаходиться там. *IP* має жовтий квадратний значокпоруч з ним (рис. 5.8).

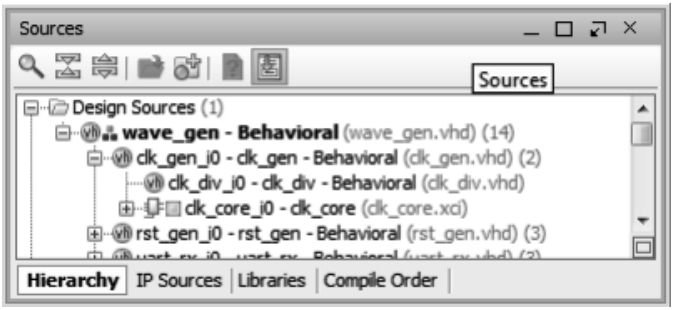


Рисунок 5.8 - clk_core

Крок 3. Реалізація схеми

1. Реалізація проекту.

Натисніть на "*RunImplementation*" на панелі FlowNavigator.

Натисніть кнопку *OK* і запустіть синтез перед запуском процесу реалізації. Коли реалізація завершена, з'явиться діалогове вікно з трьома варіантами.

Виберіть "*OpenImplementedDesign*" і натисніть кнопку *OK*.

2. Перегляд обсягу ресурсів FPGA , споживаних схемою з використанням "UtilizationReport"

На панелі FlowNavigator, виберіть "*ImplementedDesign>UtilizationReport*". Звіт відкриється в діалоговому вікні "Utilization".

Натисніть кнопку *OK*.

Переконайтеся в тому, що схема використовує тактові ресурси (рис. 5.9).

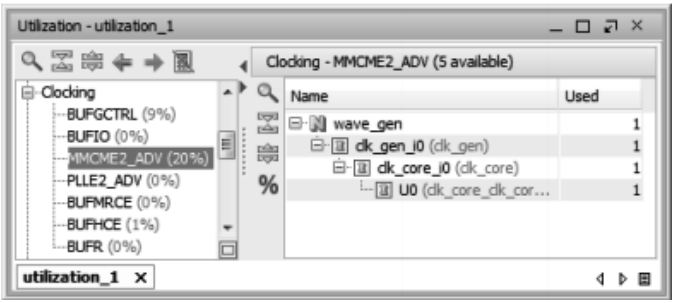


Рисунок 5.9 - Тактові ресурси для *Basys3*

Крок 4. Генерування бітового потоку і перевірка працездатності

1. Сформувати потік бітів.

На панелі "*FlowNavigator*", в пункті "*ProgramsandDebug*", натисніть кнопку "*GenerateBitstream*".

Буде виконана команда "write_bitstream" (ви можете перевірити це, подивившись в *Tcl* консоль).

Натисніть кнопку "*Cancel*", коли генерування бітового потоку завершиться.

2. Підключіть плату і подайте живлення. Створіть апаратну сесію і запрограмуйте FPGA .

Переконайтеся, що кабель Micro-USB підключений в роз'єм *JTAG PROG* (поруч з роз'ємом живлення). Переконайтеся, що перемикач на платі встановлений в те положення, щоб використовувати живлення від USB (JP2 для *Basys3*).

Виберіть "Open Hardware Manager", інатиснітькнопку *OK*. Відкриється вікно "Hardware Manager", що показує статус "unconnected".

Натисніть на "*Open a new hardware target*". Ви також можете натиснути на посилання "*Latest Targets*", якщо плата вже підключалася раніше, або натисніть на AutoConnect. У цьому випадку перейдіть до кроку 4

Натисніть кнопку "*Next*", щоб побачити форму для настройки серверного обладнання.

Натисніть кнопку "*Next*", з обраним пунктом "*TargetHardware*". *JTAG* кабель, який використовує плагін xilinx_tcf, повинен бути виявлений і ідентифікований у якості апаратного забезпечення цільової плати. Він також покаже виявлені апаратні пристрої.

Натисніть кнопку "*Next*" і "*Finish*".

Статус менеджера обладнання змінюється с "Unconnected" на ім'я сервера і виділеного пристрою. Також зверніть увагу, що статус вказує на те, що пристрій не запрограмований.

Виберіть пристрій і переконайтеся, що wave_gen.bit, обраний в якості файлу програмування, знаходиться у вкладці "General".

3. Запустіть програму-емулятор терміналу, TeraTerm або HyperTerminal. Виберіть відповідний *COM* порт (ви можете знайти правильний номер *COM* порту за допомогою панелі управління). Встановіть швидкість для *COM*-порта115200 бод. Запрограмуйте *FPGA* і перевірте працездатність.

Запустіть програму-емулятор терміналу, такі як TeraTerm або HyperTerminal.

Виберіть відповідний *COM* порт (ви можете знайти правильний номер *COM* за допомогою панелі управління).

Встановіть швидкість для *COM*-порта 115200 бод.

Клацніть правою кнопкою миші на елементі *FPGA* в вікні "*Hardware*" і виберіть "*Program Device ...*"

Натисніть на кнопку *Program*. Біт-файл для програмування буде завантажений і після завершення процесу програмування засвітиться діод *DONE*.

Поставте "*Switch* 0" в положення *ON* і введіть кілька символів у вікні терміналу – і ви побачите, що ці символи відображаються на екрані. Налаштування "*Switch* 0" в положення ON працює якloopback.

Встановіть "*Switch* 0" назад в положення *OFF* (вниз), так що схема більше не знаходиться в режимі loopback.

Виберіть "*File>Send File ..."* увікні TeraTerm.

Перейдіть до каталогу "*File>Send File ..." <2015_1_artix7_sources> \ lab4*", виберіть testpattern.txt і натисніть кнопку "*Open*". Вміст файлу буде відправлено на схему. Вміст файлу виглядає наступним чином:

```
*PFFFF          < --  specifies the pre-scaling
*S0fff          < -- specifies the speed value
*N000f          < -- specifies the number of samples to play
*W00000000       < -- write first sample of value 0 at location 0000
*W00011111       < -- write second sample of value 0x1111 at location 0001
*W00022222
*W00033333
*W00044444
*W00055555
*W00066666
*W00077777
*W00088888
*W00099999
*W000AAAAA
*W000BBBBB
*W000CCCCC
*W000DDDDD
*W000EEEEE
*W000FFFFF
```

Схема розуміє різні команди, перераховані на рисунку нижче. Всі значення в шістнадцятковому форматі. Всі значення і адреси в шістнадцятковому форматі (рис. 5.10).

Далі, введіть *G у вікні терміналу і спостерігайте, як *LED*-послідовність повільно змінюється, як зазначеноу файлі вище.

Ви можете ввести *s, щоб побачити значення зразку, *p, щоб побачити значення масштабу, а *n, щоб дізнатися, скільки зразків відтворюється.

Cmd	Input	Response	Description
*W	aaaavvvv	-OK or -ERR	03ff ≥aaaa≥0000. Value "vvvv" is written into RAM at location "aaaa" and "-OK" is return.
*R	aaaa	-hhhh dddd or -ERR	03ff ≥aaaa≥0000. If in range, then the value at "aaaa" is returned in hex and decimal.
*N	vvvv	-OK or –ERR	0400 ≥vvvv≥0001. Specifies the number of samples before recycling.
*P	vvvv	-OK or –ERR	ffff ≥vvvv≥0020. Specifies prescaling value to divide *clk_tx* by to produce *clk_samp*.
*S	vvvv	-OK or –ERR	ffff ≥vvvv≥0001. Specifies "speed" value to divide *clk_samp* by to produce the rate of read from RAM.
*n/*p/*s		-hhhh dddd	Returns current value of nsamp, prescale, and speed.
*G		-OK	Triggers a single pass through nsamp memory locations.
*C		-OK	Starts continuous triggering.
*H		-OK	Halts continuous loop at end of current cycle.

Рисунок 5.10 - Команди

Можна також ввести *H, щоб зупинити відтворення (рис. 5.11).

Рисунок 5.11 – Термінал

Після завершення перевірки закрийте програму емуляції терміналу і вимкніть живлення плати.

Виберіть "*File>Close Hardware Manager*". Натисніть кнопку *OK*.

Крок 5. Створення та генерування IPI блоку

1. Збережіть проект як lab4_*IP*i. Видаліть char_*FIFO IP* зі схеми.

Виберіть "*File>Save Project As ...*" і збережіть його як lab4_*IP*i в каталозі <2015_1_artix7_labs>, переконавшись, що опція "*Create Project Subdirectory*" відзначена.

Виберіть вкладку "*IP* Source" в панелі "Sources".

Клацніть правою кнопкою миші на char_*FIFO* і виберіть "Remove File from Project ..."

Натисніть на прапорці "*Also delete the project local file / directory from disk*", і натисніть кнопку *OK* (рис. 5.12).

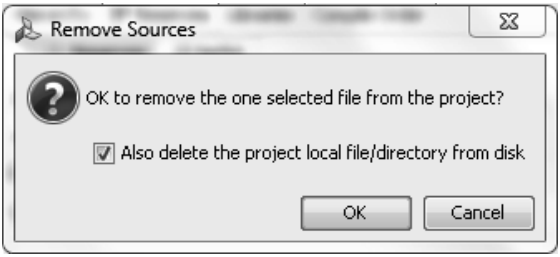


Рисунок 5.12 - Видалення існуючого *IP* з проекту

Виберіть вкладку "*Hierarchy*" на панелі "*Sources*" і зауважте, що char_*FIFO* має знак "?", який показує , що відсутній вхідний файл (рис. 5.13).

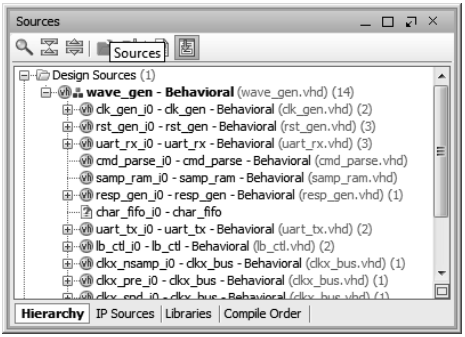


Рисунок 5.13 - Видалений вхідний файл

Двічі клацніть на wave_gen.vhd, щоб відкрити його у вікні редактора.

Видаліть створення екземпляра char*_FIFO* з файлу на рядку 554.

Видаліть декларацію char*_FIFO* з файлу на рядку 320.

Виберіть "File>SaveFile".

2. Створіть блок-схему з ім'ям char*_FIFO* і додайте екземпляр *FIFO* Generator *IP*.

Натиснітьна "Create Block Design" напанелі FlowNavigator.

Введіть char*_FIFO* якім'яблок-схеми (рис. 5.14).

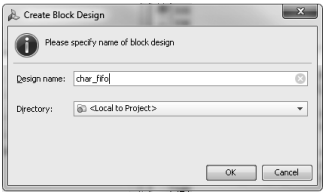


Рисунок 5.14 - Назва нової блок-схеми

Натисніть кнопку *OK*. Відкриється робоча область *IP*Integrator; в інформаційному полі потрібно додати *IP*.

У Tcl консолі, введіть наступну команду:

Set_Param bd.skIPSupportedIPCheck true

Це дозволить генератору *FIFO* з'явитися в *IP* каталозі "*IP* Integrator". Це необхідно, оскільки за замовчуванням *IP*-каталог в

IP блок - схемі буде включати до себе тільки системні *IP*-адреси процесора.

Клацніть правою кнопкою миші на області "*IP* Integrator" і виберіть "*Add IP*". Відкриється "*IP* Integrator" *IP* Каталог, що відображає спис*окIP*, доступних в *IP* Integrator.

Введіть *FIFO* в поле пошуку у верхній частині "*IP* Integrator Catalog", щоб побачити доступні *IP*-адреси для *FIFO* (рис. 5.15).

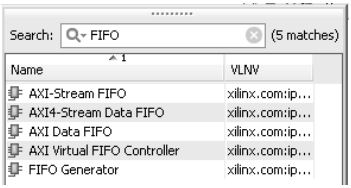


Рисунок 5.15 - Пошук *IP* в *IP* Каталозі

Двічі клацніть *FIFO*Generator. *FIFO* додається в область проекту "*IP Integrator*" (рис. 5.16).

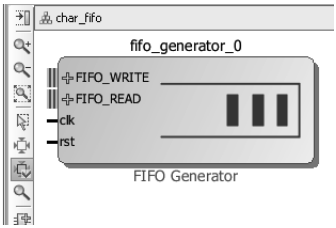


Рисунок 5.16 –Створений *FIFO* генератор

3. Налаштування екземпляра FIFOGeneratorIP.

Двічі клацніть по *FIFOGeneratorIP*. *FIFOGenerator* відкриється в діалоговому вікні "Re-customize*IP*".

Переконайтеся, що як варіант за замовчуванням вибрано "Native" для даного типу інтерфейсу.

Виберіть "*Independent Clocks Block RAM*" зісписку "*FIFO* Implementation" (рис. 5.17).

Виберіть вкладку "NativePorts". На вкладці "NativePorts" можна налаштувати режим читання, вбудовані опції *FIFO*, параметри порту даних, а також варіанти його реалізації.

Виберіть "First Word FallThrough" якрежимчитання.

Встановіть розмір записи в 8 біт.

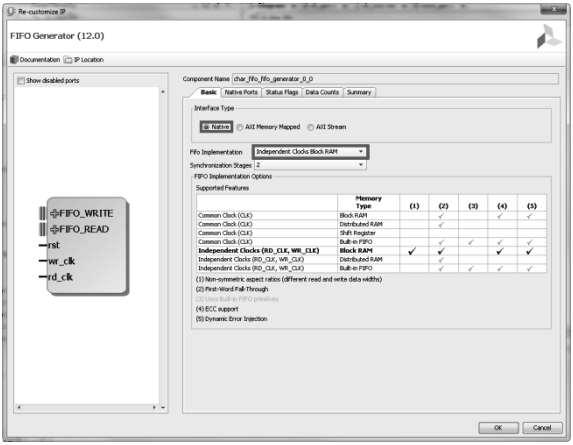


Рисунок 5.17 - Настройка BRAM для роздільного читання і запису тактів

Натисніть на поле "*ReadWidth*", щоб змінити його автоматично відповідно до ширини запису.

Залиште всі інші настройки за замовчуванням (рис. 5.18).

Подивіться на настройки у вкладках "*StatusFlags*" і "*DataCounts*". Ці вкладки налаштовують інші параметри для генератора *FIFO*. Для цієї схеми, залиште всі настройки за замовчуванням.

Виберіть вкладку "*Summary*". На цій вкладці відображається зведення всіх обраних опцій конфігурації, а також лістинг ресурсів, використовуваних для цієї конфігурації (рис. 5.19).

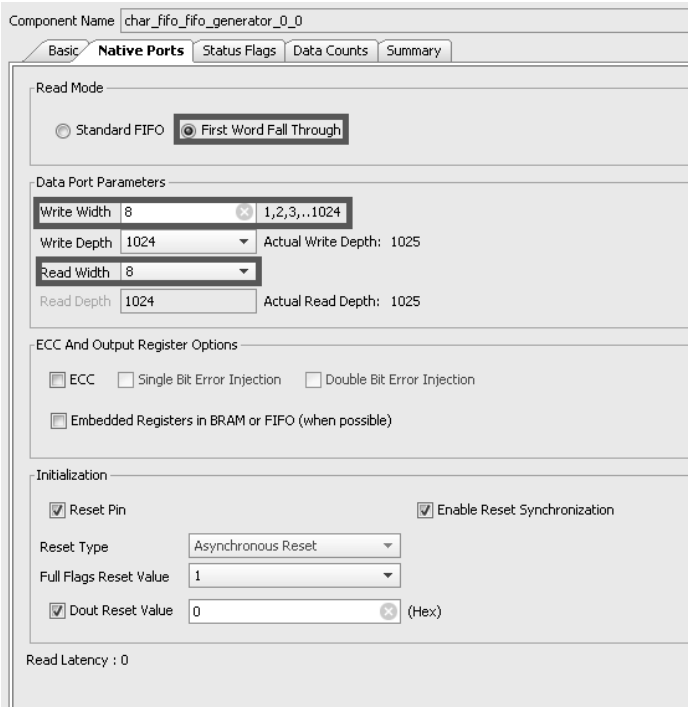


Рисунок 5.18 - Налаштування ширини порту і режиму читання

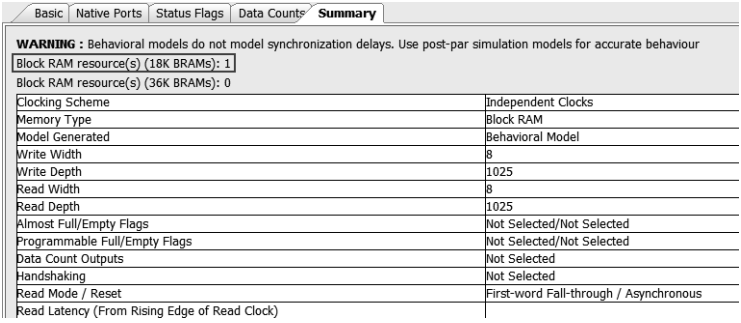


Рисунок 5.19 - Вкладка "Summary"

Переконайтеся в тому, що інформація вірна. Для цієї конфігурації ви використовуєте один блок оперативної пам'яті 18K. Натисніть кнопку *ОК*.

4. Зробіть порти зовнішніми і назвіть їх, як показано нижче (рис. 5.20).

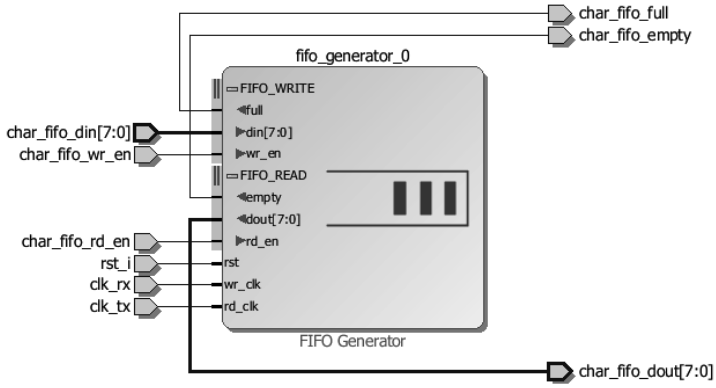


Рисунок 5.20. Генератор *FIFO IP,* повністю згенерований і підключений

Розгорніть інтерфейси "*FIFO*_WRITE" і "*FIFO*_READ".

Виберіть wr_clk, а потім натисніть і утримуйте Ctrl і виберіть порти *FIFO*rd_clk (рис. 5.21).

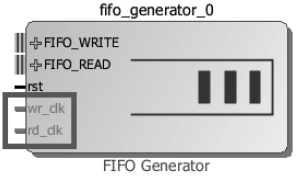


Рисунок 5.21. Вибір кількох портів

Клацніть правою кнопкою миші на виділені порти і виберіть *MakeExternal*. Два зовнішніх з'єднання будуть створені для обраних портів *FIFO*. Зверніть увагу на те, що зовнішні з'єднання мають те ж ім'я, що і порт модулю *IP*, до якого вони підключені.

Ви можете перейменувати ці з'єднання, вибравши їх і змінивши ім'я у вікні "ExternalPortProperties".

Виберіть зовнішній порт з ім'ям wr_clk.У вікні "*ExternalPortProperties*" в поле "*Name*" на вкладці "*General*" введіть ім'я clk_rx і натисніть Enter. Аналогічним чином, виберіть зовнішній порт з ім'ям rd_clk і змініть його назву на clk_tx (рис. 5.22).

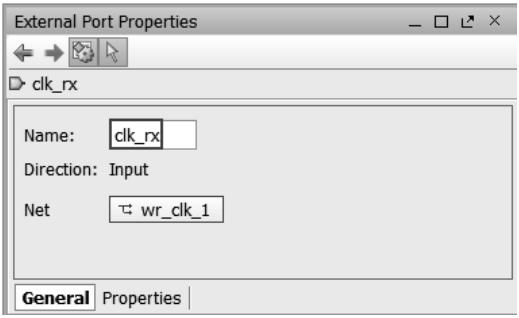


Рисунок 5.22 - Зміна імені зовнішнього порту

Вам потрібно буде відкрити пункти *FIFO*_WRITE і *FIFO*_READ, щоб побачити імена сигналів, натиснувши на символи "+" поруч з назвами шин. Утримуючи Ctrl, клацніть на всі інші вхідні та вихідні порти *FIFO* і зробіть їх зовнішніми.

Перейменуйте їх, як показано нижче:

- din = char_fifo_din
- dout = char_fifo_dout
- empty = char_fifo_empty
- full = char_fifo_full
- rd_en = char_fifo_rd_en
- wr_en = char_fifo_wr_en
- rst = rst_i

Коли ви закінчите, ваша підсхема повинна виглядати як на рис. 5.23.

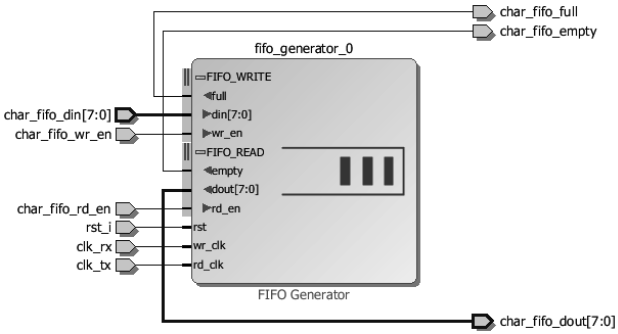


Рисунок 5.23 - Перейменовані зовнішні порти

Натисніть піктограму "*Refresh*" на вертикальній панелі інструментів, щоб побачити діаграму, наведену вище.

Виберіть "*Tools>ValidateDesign*". Ви побачите повідомлення, що перевірка пройшла успішно.

5. Генерація вихідного продукту.

На вкладці "*IP*Sources" у вікні "Sources" виберіть char*_FIFO* в пункті "BlockDesigns".

Клацніть правою кнопкою миші і виберіть "GenerateOutputProducts" (рис. 5.24).

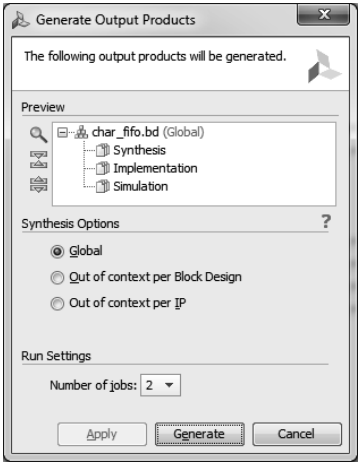


Рисунок 5.24 - Формування вихідних продуктів

Натисніть кнопку "*Generate*". Ви повинні побачити різні вихідні продукти *IP*, які відображаються на вкладці "*IP Sources*" у вікні "*Sources*" (рис. 5.25).

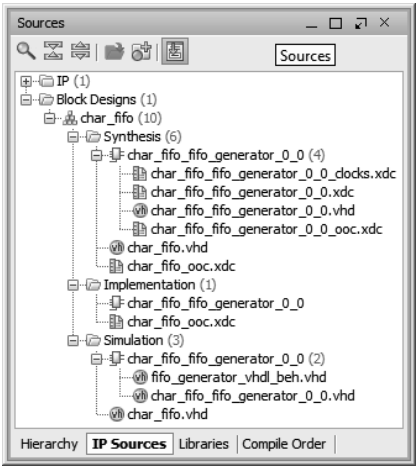


Рисунок 5.25 - Згенеровані продукти

6. Створенняchar_FIFO IP в проекті.

На вкладці "*IP Sources*" у вікні "Sources" виберіть модуль char*_FIFO*.

Клацніть правою кнопкою миші і виберіть "*ViewInstantiationTemplate"* (рис. 5.26).

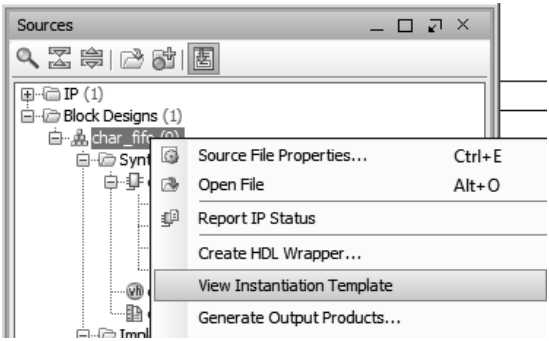


Рисунок 5.26 - Генерація шаблону для створення екземпляра

Шаблон char_*FIFO*_wrapper.v відкриється в текстовому редакторі в VivadoIDE (рис. 5.27).

Скопіюйте рядки з 29 (component ...) по 41 (endcomponent ...), і вставте їх на рядок321 в файлі wave_gen.vhd. (Зона декларацій архітектури).

Скопіюйте рядки з 43 (char_*FIFO*_i: ...) по 54 ();), і вставте їх на рядок555 в файлі wave_gen.vhd. (Архітектура)

Збережіть файл *VHDL*.

7. *Сформуйте потік бітів і перевірте функціональність на пристрої.*

Натисніть на "RunImplementation" напанелі FlowNavigator.Якщо відобразиться діалогове вікно, натисніть кнопку "Yes", щоб зберегти проект, перш ніж продовжити.

Натисніть кнопку *OK*, щоб повторно запустити процес синтезу, а потім натисніть кнопку "Save", щоб зберегти схему.

Після завершення створіть звіт про використання ресурсів і переконайтеся, що використовується тільки один *FIFO*.

На панелі "*Flow Navigator*", в меню "*Programand Debug*", натиснітькнопку "*Generate Bitstream*".

Відкрийте диспетчер апаратного забезпечення і перевірте працездатність схеми на пристрої.

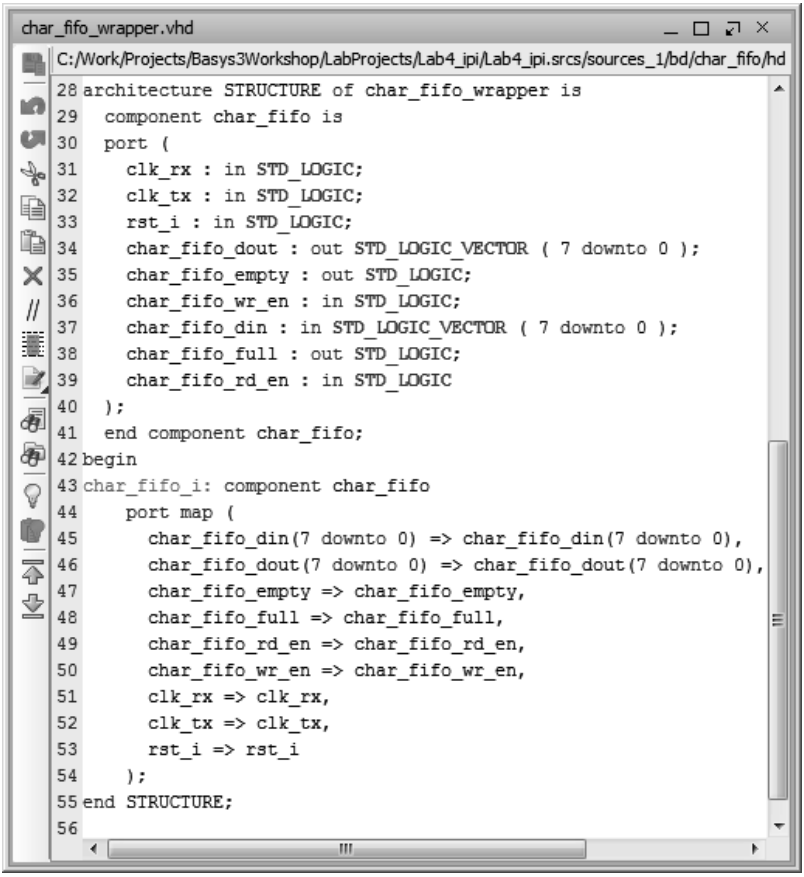


Рисунок 5.27 - Частина шаблону примірника

Крок 6. Перевірка функціональних можливостей на зовнішніх пристроях (за бажанням)

Підключіть "PmodTPH" в JB роз'єм плати BASYS 3, контакти 1 ... 6.

Підключіть "PmodDA1" до "PmodTPH".

Встановіть waveform, завантаживши її за посиланням: *http://digilentinc.com/Products/Detail.cfm?NavPath=2,66,849&Prod=WAVEFORMS*

Підключіть "AnalogDiscovery" до USB-порту комп'ютера.

Підключіть цифрові щупи 0 ... 3 до контактів 1 ... 4, роз'єму J2.

Підключіть диференціальні щупи області видимості каналу 1 до J2: щуп "1+" до контакту 1, щуп "1" до контакту 5 (GND).

Запустіть програму-емулятор терміналу, таку як TeraTerm або HyperTerminal. Виберіть відповідний *COM* порт (ви можете знайти правильний номер *COM* за допомогою панелі управління). Швидкість – 115200 бод.

Клацніть правою кнопкою миші на елементі *FPGA* в вікні "*Hardware*" і виберіть "*Program Device ...*". Натисніть на кнопку "*Program*".

Виберіть "*File>SendFile ...*" увікні TeraTerm.

Перейдіть до каталогу <2015_1_artix7_sources> \ lab4, виберіть файл testpattern.txt і натисніть кнопку "Open". Вміст файлу буде відправлено на схему.

Зверніть увагу, що світлодіоди блимають з вис*ОК*ою швидкістю.

Запустіть вейв-форму. Відкрийте "*LogicAnalyzer*" і відкрийте інтерпретатор *SPI* шини. Налаштуйте його як на рис. 5.28.

Налаштуйте "*LogicAnalyzer*", як на рисунку нижче, і натисніть кнопку "*Run*" або "*Single*". Зверніть увагу на повідомлення *SPI*, відправлені на *PmodDA1* (рис. 5.29).

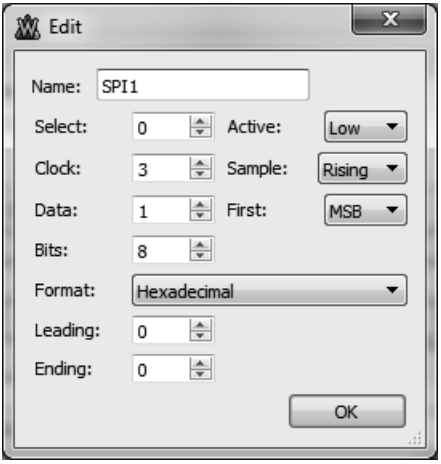


Рисунок 5.28 - Налаштування Waveforms шини SPI

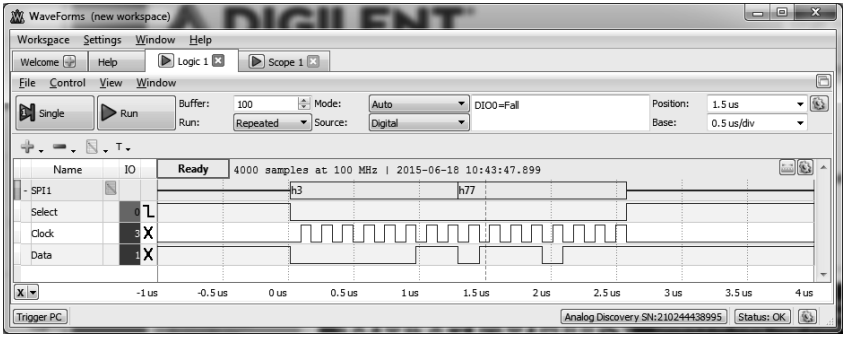


Рисунок 5.29 - WaveformsSPI шина в логічному аналізаторі

Відкрийте "Scope" в Waveformsi налаштуйте його як на рисунку нижче. Натисніть кнопку "Run" або "Single" і спостерігайте за аналоговим сигналом, сформованим PmodDA1 (рис. 5.30).

Змініть швидкість і/або перед дільник, використовуючи команди *S ... і *P .... Зверніть увагу на поведінку проекту.

Коли закінчите, закрийте програму Vivado, вибравши "File>Exit" і натисніть кнопку *OK*.

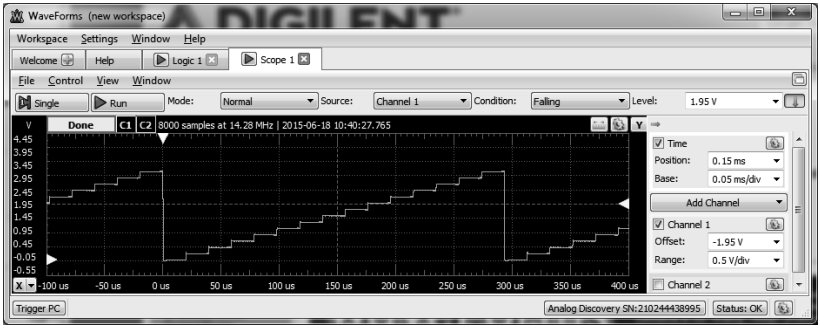


Рисунок 5.30 - Waveforms

У цій роботі ви дізналися, як додати існуючий *IP* при створенні проекту. Ви також дізналися, як використовувати *IP*-каталог і створити тактове ядро. Потім ви інстанціювали ядро в схему, реалізували схему, та перевірили схему на апаратному рівні. Ви також використовували можливості інструменту *IP* Integrator для генерації *FIFO*, а потім використовували його в *HDL* схемі.

5.3 Контрольні запитання

1. З якою метою використовують програмований генератор сигналів?
2. З якою метою використовують *IP* інтегратор?
3. Як створити та ініціалізувати генератор тактових імпульсів?
4. Як виконати зміну імені примірника і мережевих з'єднань?
5. Як згенерувати бітовий пото*к* і перевірку працездатності?
6. Наведіть приклади схем, які розуміє схема *Basys3*.
7. Як створити та згенерувати *IP* I блок?
8. Як налаштувати ширину порту і режиму читання?
9. Як згенерувати шаблон для створення екземпляра?
10. Як перевірити функціональні можливості на зовнішніх пристроях?

6 ОБМЕЖЕННЯ XILINX СХЕМ

У цій роботі ви будете використовувати схему uart_led, яка була розглянута в роботі 5. Ви створите проект з типом "I/O Planning", задасте розташування контактів, а також експортуєте його в rtl. Після цього створите часові обмеження і виконайте часової аналіз.

6.1 Основні теоретичні відомості

Ця лабораторна робота розбита на Кроки, що складаються з короткого опису, які необхідно виконати для завершення лабораторної роботи.

Схема складається з uart-приймача, що приймає вхідні дані с клавіатури і відображає двійковий еквівалент набраних символів на 8 світлодіодах. При натисканні кнопки, нижня і верхня напівбайти міняються місцями. Блок-схема показана на рис. 6.1.

6.2 Порядоквиконання роботи

Крок 1. Створення проекту з I/O Planning

Запустіть Vivado і створіть проект I/OPlanning для пристрою XC7A35TCPG236-1 (Basys3).

Відкрийте *Vivado*, вибравши *“Пуск> Всі програми>XilinxDesignTools>Vivado 2015.1> 2015.1 Vivado”*

Натисніть “*CreateNewProject*” для запуску майстра. З'явиться діалогове вікно “*CreateaNewVivadoProject*”. Натисніть кнопку “*Next*”.

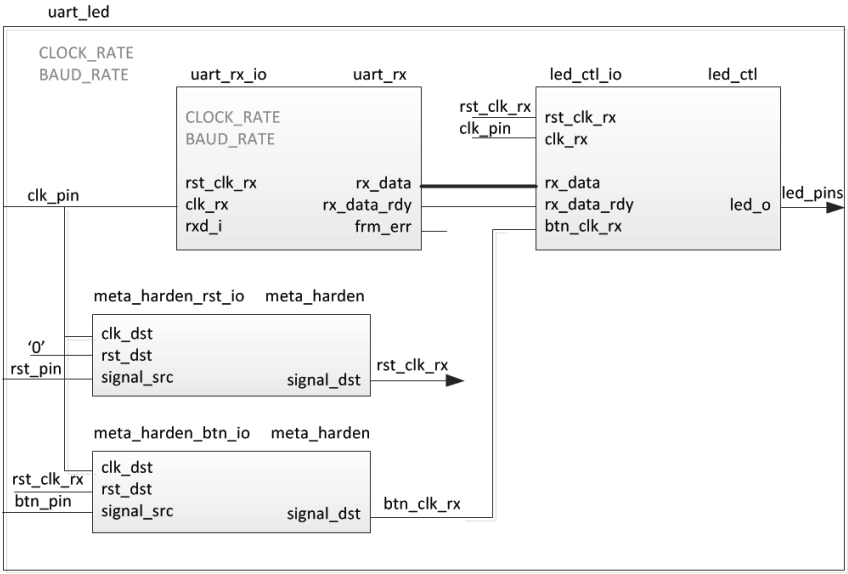


Рисунок 6.1 - Блок схема

Натисніть кнопку *Browse* в полі *Projectlocation* у формі *NewProject*, перейдіть до *<2015_1_artix7_labs>*, і натисніть кнопку Select.

Введіть lab5 вполе "*Projectname*". Переконайтеся, що прапорець "*CreateProjectSubdirectory*" відмічений. Натисніть кнопку Next.

Виберіть опцію "I/OPlanningProject" уформі "ProjectType", і натисніть кнопку Next.

Виберіть "*Donotimport I/O portsatthistime*", і натисніть кнопку Next.

У формі "*DefaultPart*", за допомогою опції Parts і різних випадаючих полей секції Filter, виберіть *XC7A35TCPG236-1* для *Basys3*. Натисніть кнопку *Next*.

Натисніть кнопку *Finish*, щоб створити проект *Vivado*.

З'явиться вкладка "*Deviceview*" і "*PackagePins*" (рис. 6.2).

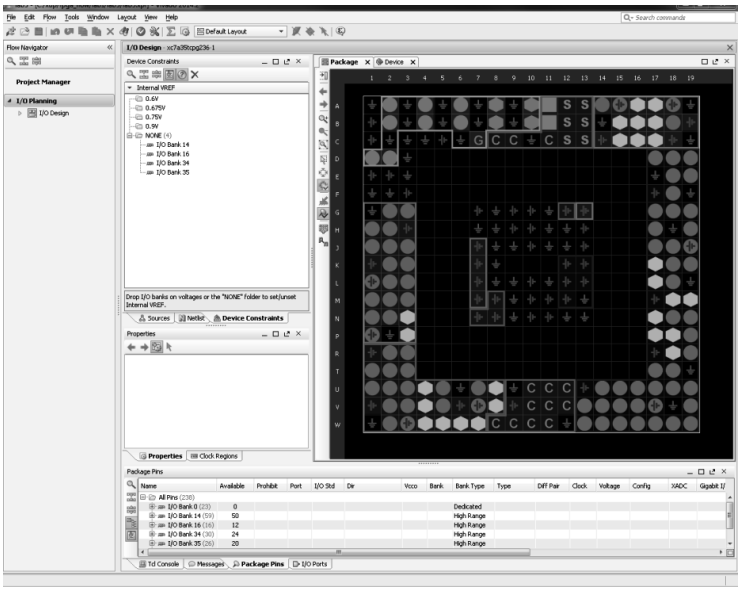


Рисунок 6.2 - I/O Planning

Крок 2. Назначеня різних контактів та додавання вхідних файлів

1. Для *Basys3*, призначити вхідні контакти clk_pin, btn_pin, rxd_pin і rst_pin до W5, T18, B18 і U18, використовуючи вид *Device iPackagePins.*

У нижній панелі виберіть вкладку “I/*OPorts*”. Натисніть на іконку “CreateI/OPorts”, на лівій стороні вкладки “I/OPorts”.

Встановіть ім'я порту clk_pin і стандарт *I/OLVCMOS33*, як в наведеному нижче рис. 6.3. Натисніть кнопку *OK*.

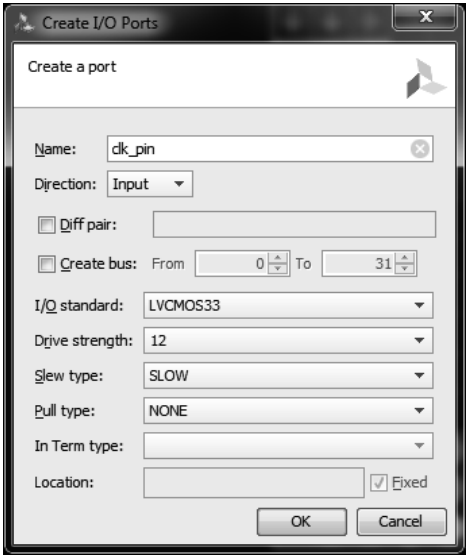


Рисунок 6.3 Встановлення параметрів

Для *Basys3*, наведіть курсор миші на W5 в вікні перегляду пристрою (рис. 6.4).

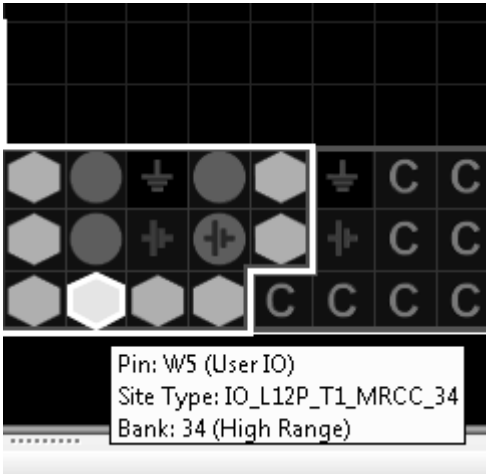


Рисунок 6.4 – Місце знаходження контакту W5

Коли ви його знайдете, натисніть на нього. Контакт буде підсвічено і він буде відображуватися на вкладці *PackagePins*.

На панелі PackagePins, клацніть по колонці "*Ports*" у рядку W5 (*Basys3*), і виберіть clk_pin.

Аналогічним чином додайте вхідний порт btn_pin на T18 (*Basys3*) зі стандартом *LVCMOS33*.

Виберіть "*Edit>Find*" або *Ctrl*+F, щоб відкрити форму пошуку. Виберіть "PackagePins" в меню Find , введіть *B18 (для *Basys3*) в поле пошуку, і натисніть на кнопку *OK* (рис. 6.5).

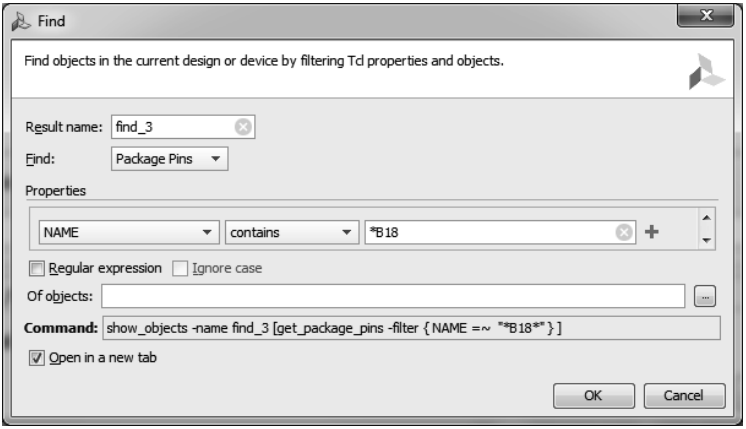


Рисунок 6.5 - Пошук Package Pin

Натисніть на контакт на вкладці з результатами. Зверніть увагу на те, що запис буде виділено у вікні Device.

Створить вхідні порти rxd_pin і rst_pin зі стандартом LVCMOS33.

На панелі "I/O Ports", клацніть в стовпці Site сигналу rxd_pin і виберіть B18 (для *Basys3*). Точно так же призначте вхід rst_pin на U18 (для *Basys3*).

2. Для *Basys3*, призначте вихідні контакти led_pins[7] …led_pins[0] на місця V14, U14, U15, W18, V19, U19, E19 та U16. Створіть їх у якості вектора та призначте за допомогою Tcl-команди set_property. Усі контактиповинні бути LVCMOS33.

На вкладці *I/O Ports*, натисніть на кнопку створення портів введення/виводу на лівій вертикальній стрічці (рис. 6.6).

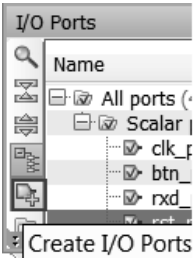


Рисунок 6.6 - Створення кнопки I/OPorts

Введіть led_pins в поле *Name*, виберіть напрямок Outputdirection, встановіть прапорець *Create_bus*, встановіть *MSB* до 7, і виберіть стандарт *LVCMOS33I*/O, натисніть кнопку *OK* (рис. 6.7).

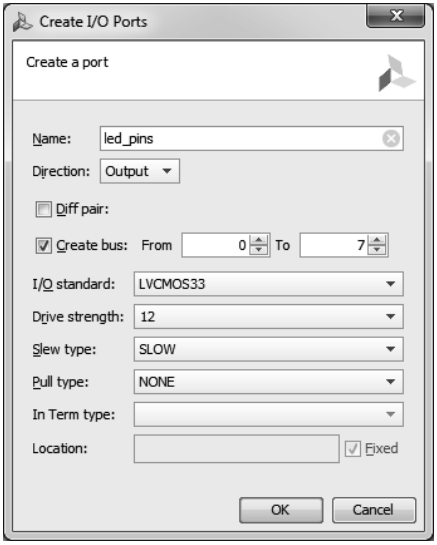


Рисунок 6.7 - Створення I/O портів для виводу led_pins

Будуть створені і відображені записи led_pinsна вкладці *I/OPorts*. Зверніть увагу на те, що стандарт *I/O* і напрямки вже встановлені, залишилось тільки призначити розташування контактів.

Введіть наступні команди в консолі для призначення місця розташування контактів.

Forthe *Basys3*:

```
set_property PACKAGE_PIN V14 [get_ports led_pins[7]]
set_property PACKAGE_PIN U14 [get_ports led_pins[6]]
set_property PACKAGE_PIN U15 [get_ports led_pins[5]]
set_property PACKAGE_PIN W18 [get_ports led_pins[4]]
set_property PACKAGE_PIN V19 [get_ports led_pins[3]]
set_property PACKAGE_PIN U19 [get_ports led_pins[2]]
set_property PACKAGE_PIN E19 [get_ports led_pins[1]]
set_property PACKAGE_PIN U16 [get_ports led_pins[0]]
```

ВиберітьFile>Save Constraints.

На екрані з'явиться форма Save Constraints.

Введіть uart_led вполе Filename і натисніть кнопку *OK*. (Якщо це неможливо, залиште ім'я за замовчуванням) (рис. 6.8)

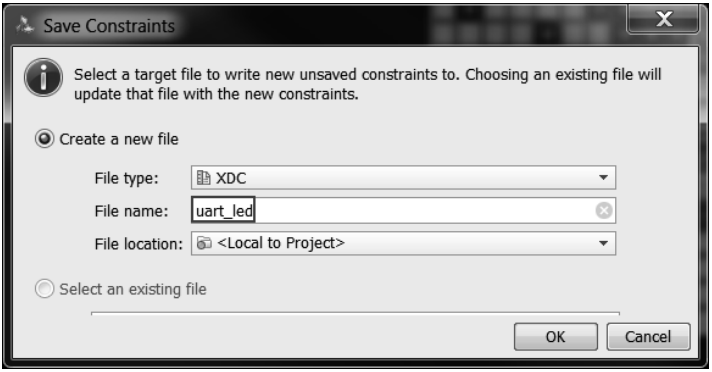


Рисунок 6.8 - Збереженнямайстраобмеження

Файл uart_led.xdc буде створений і доданий до вкладки Sources (рис. 6.9).

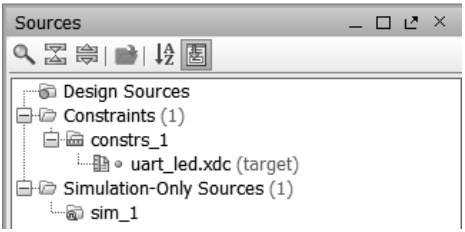


Рисунок 6.9 - Файл uart_led.xdc доданий до вхідного дерева

Розверніть *“I/O Planning> I/O Design”* впанелі FlowNavigator

Натисніть на Report DRC та натисніть *OK*. Зверніть увагу, що виконується перевірка правил проектування і повідомляється одне попередження. Не звертайте уваги на попередження.

Натисніть на ReportNoise та натисніть *OK*. Зверніть увагу, що аналіз шуму зроблений тільки на вихідних контактах (led_pins) та відображаються результати.

Натисніть на *MigratetoRTL*. Форма Migrateto RTL буде відображатися з полем файлаTop RTL, яке показує *C:/Digilent/Basys3Workshop/2015_1_artix7_labs/lab5/io_1.vhd.*

Установіть *Netlist* формат на *VHDL*. Змініть io_1.vhd на uart_led.vhd, і натисніть кнопку *OK* (рис. 6.10).

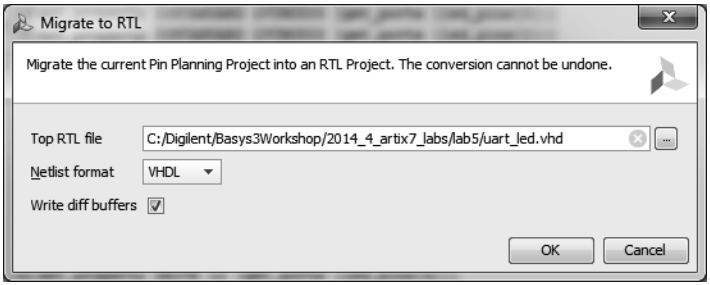


Рисунок 6.10 - Присвоєння імені файлу верхнього рівня (зміна на vhd)

Виберіть вкладку **Hierarchy** і зверніть увагу, що файл uart_led.vhd був доданий до проекту з ім'ям модуля верхнього рівня «ios». Якщо ви зробите подвійний клік на записі, ви побачите ім'я модуля зі списком портів (рис. 6.11).

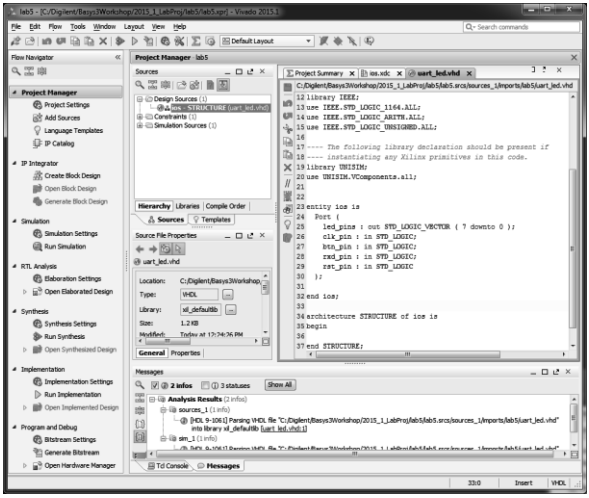


Рисунок 6.11 - Зміст модуля верхнього рівня та ієрархія схеми після переходу на RTL

3. Додайте надані вхідні файли (з <2015_1_artix7_sources> \ lab5) до проекту. Скопіюйте вміст файлу uart_led.txt (розташованого в <2015_1_artix7_sources> \ lab5) у вхідний файл верхнього рівня.

Натисніть на *AddSources* в панелі *FlowNavigator*.

Вформі AddSources виберіть *Add orCreateDesignSources* танатисніть Next.

Натиснітькнопку + тавиберітьAddFiles…,

Перейдіть в *<2015_1_artix7_sources>\lab5* та виберіть всі .vhd файли (led_ctrl, uart_rx, meta_harden, uart_baud_gen, uart_rx_ctl) і натисніть кнопку *OK*.

Натисніть *Finish*.

За допомогою провідника Windows перейдіть до папки <2015_1_artix7_sources> \ lab5 та відкрийте uart_led.txt за

допомогою будь-якого текстового редактора. Скопіюйте його вміст і вставте його в uart_led.vhd (навколо рядка 34) в проекті Vivado (замінюючи рядок"begin" в порожній архітектурі).

Виберіть *File>SaveFile*.

Відкрийте файл *XDc* і закоментуйте наступні рядки (якщо вони є), використовуючи «#» для кожного рядка (або виділивши рядки і натиснувши на кнопку блокового коментаря «//»): (якщо немає, то вони автоматично видаляються)

```
set_property direction IN [get_ports {clk_pin}]
set_property direction IN [get_ports {btn_pin}]
set_property direction IN [get_ports {rxd_pin}]
set_property direction IN [get_ports {rst_pin}]
set_property direction OUT [get_ports {led_pins[0]}]
set_property direction OUT [get_ports {led_pins[1]}]
set_property direction OUT [get_ports {led_pins[2]}]
set_property direction OUT [get_ports {led_pins[3]}]
set_property direction OUT [get_ports {led_pins[4]}]
set_property direction OUT [get_ports {led_pins[5]}]
set_property direction OUT [get_ports {led_pins[6]}]
set_property direction OUT [get_ports {led_pins[7]}]
```

Виберіть File>Save File.

Крок 3. Генерація та введення часових обмежень

Згенеруйте схему. Використати майстер обмежень для завдання тактової частоти, а потім використати редактор часових обмежень для додавання вхідних і вихідних затрим*ок*в файл XDC.

Натисніть на Run Sysnthesis в панелі Flow Navigator.

Коли генерація завершиться, буде відображена форма з трьома варіантами.

Виберіть OpenSynthesizedDesign та натисніть *OK*.

В панелі *FlowNavigator* (під *Synthesized* Design) натисніть на кнопку *ConstraintsWizard*. Це відкриє майстра обмежень.

Прочитайте інформацію у вікні Identify and Recommend Missing Timing Constraints для того, щоб зрозуміти, що робить майстер та натисніть Next.

Встановіть частоту об'єкта "clk_pin" на 100 МГц, зверніть увагу, що Period, RiseAt та FallAt автоматично заповнюються. Також зверніть увагу на Tcl-команду, яку можна переглянути в

нижній частині майстра. Натисніть Next для продовження (рис. 6.12).

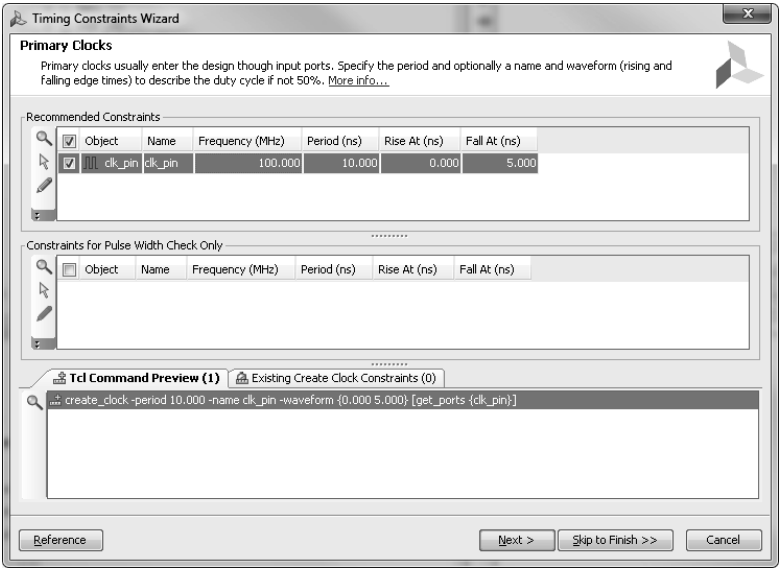


Рисунок 6.12 - Майстер часових обмежень

Майстер інформує вас, що тут немає пропущених Generated Clocks, тому натисніть *Next* для продовження.

Немає пропущених "*Forwarded Clocks"*, тому натисніть *Next* для продовження.

Немає пропущених "*External Feedback Delays*", тому натиснітьNext для продовження.

Майстер ідентифікує вхідні затримки, необхідні для контактів: btn_pin, rst_pin і rxd_pin. Виконайте наступні дії:

1. Натисніть *Ctrl* та виберіть три рядки.

2. Введіть tco_min-0.5 ns та все інше 0 ns. Натисніть *Apply*.

3. Зверніть увагу на місце нижче вкладки *TclCommandPreview*, були згенеровані 6 Tcl-команд.

4. Натисніть *Next* (рис. 6.13).

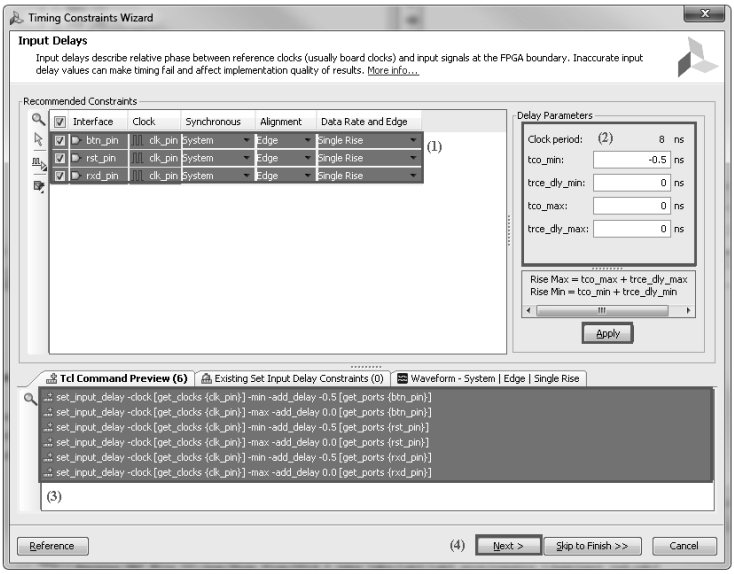


Рисунок 6.13 - Завдання вхідних затрим*ок*для btn_pin, rst_pin і rxt_pin

Для вихідних затримок визначте всі значення (tsu, trce_dly_max, thd, trce_dly_min) як 0 ns. Натисніть *Apply* та далі натисніть *Next*.

Далі ви побачите, що немає ідентифікованих CombinatorialDelays, тому натисніть *Next*.

Натисніть Next для наступних екранів аж до того моменту, коли побачите сторінку "*Constraints Summary"*. Прочитайте опис до кожної зі сторінок.

Перевірте *OnFinish – ViewTimingConstraints* і натисніть кнопку Finish, щоб закрити майстра. Опція відкриє редактор *TimingConstraints*, щоб показати вам згенероване обмеження часу (рис. 6.14, 6.15).

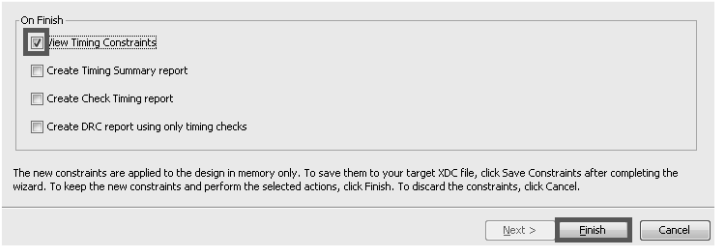


Рисунок 6.14- Вибір перегляду обмежень часу

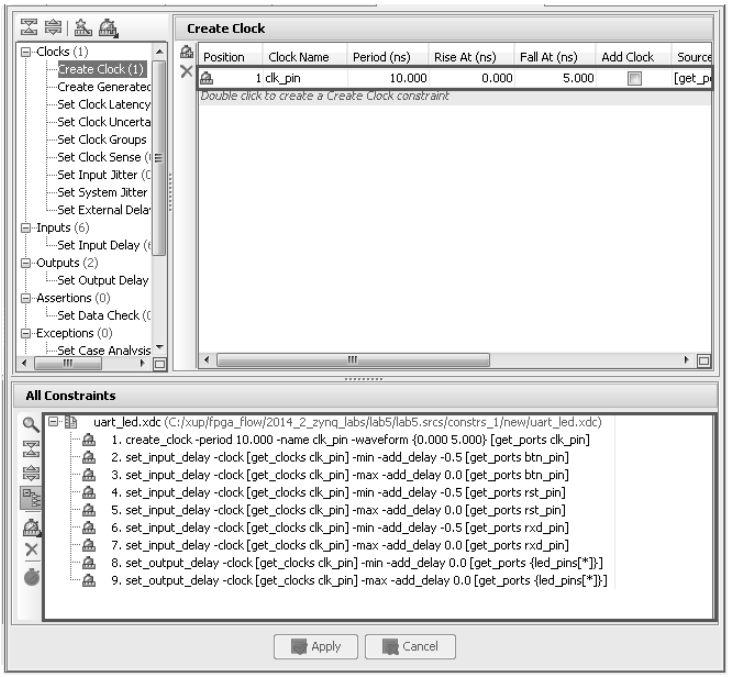


Рисунок 6.15 - Обмеження, додані після використання майстра обмежень

Зверніть увагу, що майстер сформував clk_pin обмеження на період 10 ns (або 100 МГц). Зауважте, що у вікні *AllConstraints* будуть створені 9 обмеженнь.

Немає необхідності натиснути кнопку Apply, оскільки обмеження вже застосовувалися в майстрі обмежень.

Виберіть "*File>Save Constraints*" для збереження змін до цільового XDC. Натисніть *OK* у попередженні, яке з'явиться на екрані.

Відкрийте .xdc файл (якщо *він вже відкритий, натисніть Reload в жовтому рядку стану) і зве*рніть увагу на додаткові обмеження, що були додані до останнього рядка файлу.

2. Генерація прогнозуємого звіту про час, що показує Setup та HoldPath у схемі.

У *FlowNavigator*, виберіть "*Synthesized Design> Report Timing Summary*".

На вкладці *Options* виберіть min_max із списку "*Path delay type*" (рис. 6.16).

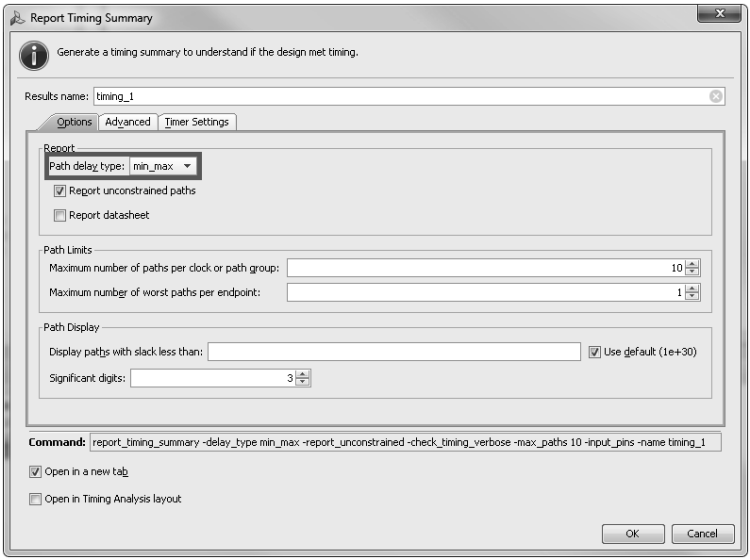


Рисунок 6.16 - Виконання аналізу використаного часу

Натисніть *OK* для запуску аналізу.

Результати відкриються на нижній частині *Vivado IDE* (рис. 6.17).

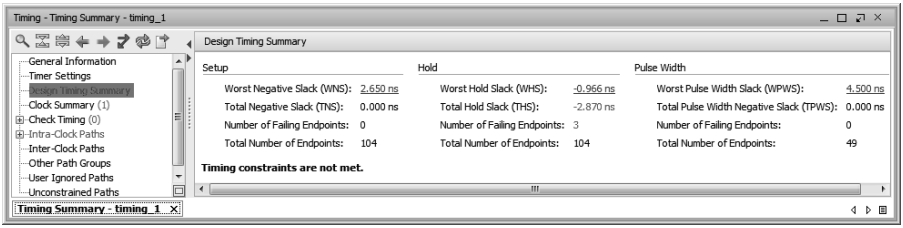


Рисунок 6.17 - Підсум*ок*використаного часу для *Basys3*Workshop

Звіт *DesignTimingSummary* містить коротку інформацію про найгіршу *Setup* та *Holdslack*, а також Numberoffailingendpoints, щоб показати чи вклалась схема у час чи ні.

Зверніть увагу, що існують три помилки синхронізації в рамках перевірки утримання.

Натисніть на посилання рядом з *WorstHoldSlack* (*WHS*), щоб побачити списокпомилкових шляхів (рис. 6.18).

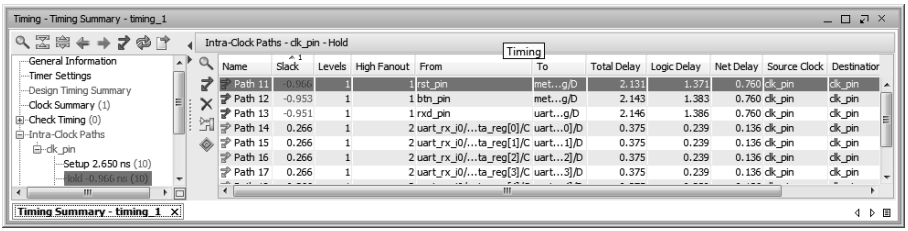


Рисунок 6.18 - Спис*ок*шляхів, який показує порушення для *Basys3*

Зробіть подвійний клік на шляху 11 для перегляду подробиць фактичного шляху (рис. 6.19).

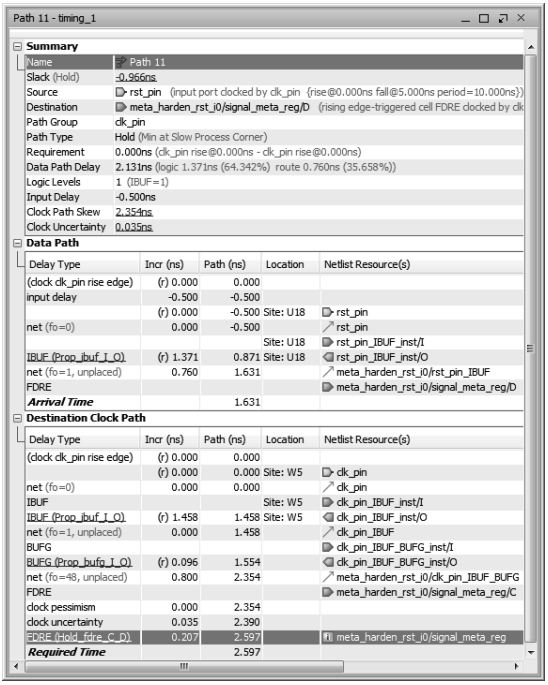


Рисунок 6.19 - Помилковий шлях утримання для *Basys3*

Оберіть Шлях 11, натисніть правою кнопкою та оберіть Schematic (рис. 6.20).

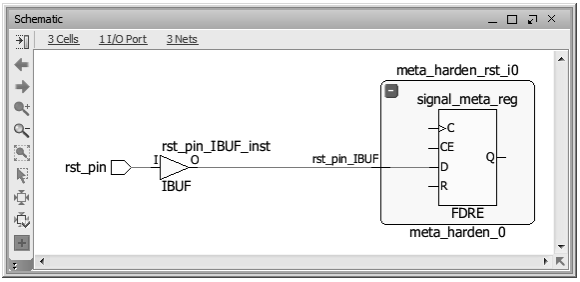


Рисунок 6.20 - Схематичний перегляд помилкового шляху

Три помилкові шляхи – це btn_pin, rxd_pin і rst_pin.

Крок 4. Реалізація та аналіз підсумку затраченого часу

1. *Реалізація схеми*.

Натисніть на *RunImplementation* в панелі *FlowNavigator*.

Натисніть *Yes*, щобспочатку запустити генерацію перед запуском процесу реалізації.

Коли реалізація закінчена, ви побачите діалогове вікно з трьома варіантами вибору.

Виберітьопцію "*Open Implemented Design*" танатисніть *OK*.

Натисніть *Yes*, якщо вам буде запропоновано закрити згенеровану схему.

2. Генерація загального звіту про затрачений час.

В панелі FlowNavigator під Implementation>Implemented Design, натисніть Report Timing Summary.

Натисніть *OK* для генерації звіту, використовуючи налаштування за замовчуванням.

Вікно "*Design Timing Summary"* відкриється внизу на вкладці Timing.

Зауважте, що помилкові часові шляхи помічені червоним кольором (рис. 6.21).

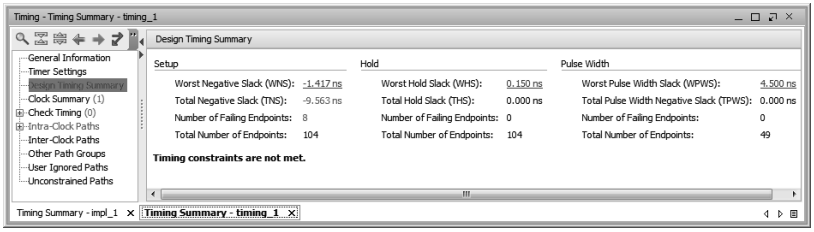


Рисунок 6.21 - Помилкові Setup-шляхи для *Basys3*

Натисніть на *WNS* для перегляду помилкових шляхів.

Натисніть двічі на першому зверху помилковому шляху для перегляду детального аналізу.

Затримки на вихідному шляху можна знизитирозміщенням регістра в *IOB*.

Застосуйте обмеження, ввівши таку команду в консолі Tcl:

set_property IOB TRUE [get_ports led_pins[*]]

Оберіть “*File > Save Constraints*”. Натисніть кнопку *OK* у попередженні.

Натисніть на RunImplementation.

Натисніть Yes для рестарту генерацііі запуску реалізації.

Відкрийте реалізовану схему та подивіться, що кількість помилкових шляхів у вкладці *DesignRuns* дорівнює 0.

4. Натисніть *ReportTimingSummary* та подивіться, що тут також немає помилкових шляхів.

Крок 5. Генерація бітового потоку і перевірка працездатності (необов'язково)

1. *Генерація бітового потоку.*

В панелі *FlowNavigator*, під *ProgramandDebug*, натисніть GenerateBitstream.

Команда write_bitstream буде виконана (ви можете перевірити це, подивившись в консолі *Tcl*).

Натисніть *Cancel*, коли генерація завершена.

2. Підключіть плату і увімкніть живлення. Відкрийте апаратний сеанс, і запрограмуйте *FPGA* .

Переконайтеся, що кабель Micro-USB підключається до роз'єму *JTAG PROG* (поруч з роз'ємом живлення). Переконайтеся, що перемикач на платі встановлений, щоб вибрати живлення по *USB* (*JP2* на *Basys3*).

Включіть живлення.

Виберіть опцію *OpenHardwareManager*.

Відкриється вікно *“Hardware Manager”*, вказуючи статус "непідключений".

Натисніть на посилання Open a new *hardware* target.

Ви також можете натиснути на посилання Open recent target, якщо плата вже була запланована раніше. В такому випадку переходьте до пункту 5-2-8.

Натисніть *Next* для перегляду форми VivadoHardwareServerSettings.

Натисніть *Next* зобраним “*Target Hardware”.*

JTAG -кабель, який використовує xilinx_tcf, повинний бути виявлений і ідентифікований. Він також показує апаратні пристрої, виявлені в ланцюзі.

Натисніть *Next* та *Finish*.

Статус *Hardware Manager* змінюється від *Unconnected* на ім'я сервера і пристрійпідсвітиться. Також зверніть увагу, що статус вказує на те, що він не запрограмований.

Виберіть пристрій і переконайтеся, що uart_led.bit обраний в якості файлу програмування на вкладці *General*.

3. Запустіть програму-емулятор терміналу, таку як TeraTerm або HyperTerminal. Виберіть відповідний *COM* порт (ви можете знайти правильний номер *COM* за допомогою панелі управління). Встановіть *COM*-порт для зв'язку на швидкості передачі 115200. Запрограмуйте *FPGA* і перевірте працездатність.

Запустіть програму-емулятор терміналу, таку як TeraTerm або HyperTerminal.

Виберіть відповідний *COM* порт (ви можете знайти правильний номер *COM* за допомогою панелі управління).

Налаштуйте *COM*-порт на швидкість передачі 115200.

Клацніть правою кнопкою миші на елементі *FPGA* в вікні Hardware і виберіть "*Programming Device*…"

Натисніть на кнопку *Program*.

Запрограмований біт-файл можна завантажити.Загориться світлодіод «*DONE*», що вказує, що *FPGA* запрограмований.

Перевірте функціональність, як ви робили в попередній лабораторній роботі, набравши кілька символів в терміналі, і спостерігаючи, як з'являються відповідні значення насвітлодіодах.

Після завершення перевірки закрийте програму емуляції терміналу і відключіть плату.

Виберіть "*File>Close Hardware Manager"*. Натисніть *OK*, щоб закрити.

Колиз акінчите, закрийте программу *Vivado*, вибравши "*File>Exit"* і натисніть кнопку *OK*.

У цій роботі ви дізналися, як створити проект I/OPlanningi призначити контакти через вид *Device*, вкладку *PackagePins* та Tcl-команди. Потім виекспортували в rtl проект, в якому розмістили надані вхідні файли. Далі ви створили часові обмеження і виконали пост-генераціюта аналіз синхронізації після реалізації.

6.3 Контрольні запитання

1. Як створити проект з I/O Planning?
2. Як додати вхідний порт?
3. Як призначити вихідні контакти?
4. Як призначити місця розташування контактів?
5. Як налаштувати Майстера часових обмежень?
6. Як переглянути підсум*ок* використаного часу для *Basys3*Workshop?
7. Як переглянути спис*ок*шляхів, який показує порушення для *Basys3*?
8. Як можна знизити затримки на вихідному шляху?
9. Як застосувати обмеження, використовуючи консоль Tcl?
10. Як виконати пост-генерацію та аналіз синхронізації після реалізації?

7 НАЛАГОДЖЕННЯ АПАРАТНИХ ЗАСОБІВ

У цій роботі ви будете використовувати uart_led схему, з якою працювали в попередніх роботах. Ви будете використовувати функцію *MarkDebug*, а також доступне ядро *IntegratedLogicAnalyzer (ILA*) (в *IP*-каталозі) для налагодження апаратного забезпечення.

7.1 Основні теоретичні відомості

Ця робота розбита на Кроки, які складаються із загального короткого огляду тверджень, що містять інформацію про д*ок*ладні інструкції, яким потрібно слідувати.

Конструкція складається з uart-приймача, що приймає вхідні дані з клавіатури і відображає двійковий еквівалент набраного символу на 8 світлодіодах. При натисканні кнопки, нижній і верхній напівбайти міняються місцями. Бл*ок*-схема показана на рис. 7.1.

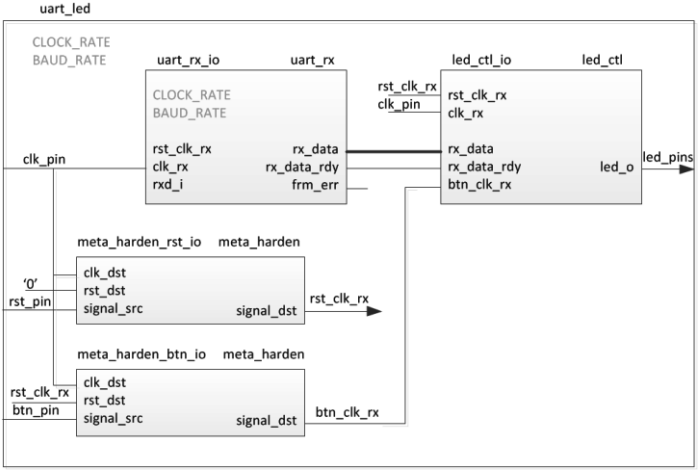


Рисунок 7.1 - Закінчена схема

7.2 Порядоквиконання роботи

Крок 1. Створити проект Vivado, використовуючи IDE

Запустіть Vivado і створіть проект направлений на пристрій *XC7A35TCPG236*-1 (*Basys3*) і використовуючи *VHDL HDL*. Скористайтесь *VHDL* і вхідними файлами XilinxDesignConstraints з каталогу <2015_1_artix7_sources> \ Lab6.

Посилання на <2015_1_artix7_labs> є заповнювачем для *"C:\Digilent\Basys3Workshop\2015_1_artix7_labs*" каталогу і *<2015_1_artix7_sources>* є заповнювачем для *"C:\Digilent\Basys3Workshop\2015_1_artix7_sources*" каталогу.

Посилання на <board> означає *Basys3*.

Відкрийте *Vivado*,обрав "*Start > All Programs > Xilinx Design Tools >Vivado 2015.1 >Vivado 2015.1*"

Натисніть "*Create New Project"* для створення майстра. Ви побачити діалогове вікно *"Create A NewVivado Project"*.

Натисніть на кнопку *Browse* у полі *Projectlocation* форми *NewProject*, перейдіть в *<2015_1_artix7_labs>* та натисніть Select.

Введіть lab6 в полі Project name. Переконайтесь, що прапорець *CreateProjectSubdirectory* встановлено. Натисніть *Next*.

Виберіть опцію *RTLProject* в формі *ProjectType* і натисніть Next.

Використовуючи меню, що розкриваються, виберіть *VHDL* як *TargetLanguage* і *Mixed* як Simulatorlanguage в формі *AddSources*.

Натисніть на кнопку +, далі оберіть *AddFiles*…, перейдіть до директорії *<2015_1_artix7_sources>\lab6*, виберіть всі файли VHDL (led_ctl.vhd, meta_harden.vhd, uart_baud_gen.vhd, uart_led.vhd, uart_rx.vhd, anduart_rx_ctl.vhd), натисніть *OK*, і потім Next.

Натисніть Next для перегляду форми *AddConstraints*.

Натисніть на кнопку +, далі оберіть *AddFiles*…, перейдіть до директорії *<2015_1_artix7_sources>\lab6*, виберіть файлі uart_led_timing.xdc і uart_led_pins_<board>.xdc. Натисніть *OK*.

Натисніть *Next*.

В формі *DefaultPart*, використовуючи опцію Parts та різноманітні поля, що розкриваються з секції Filter, виберіть XC7A35TCPG236-1 для *Basys3*. Натисніть *Next*.

Натисніть Finish для створення проекту *Vivado*.

Крок 2. Додавання ядра ILA

Натисніть *IPCatalog* під завданнями *ProjectManager* у панелі FlowNavigator. Каталог буде відображений у панелі *Auxiliary*.

Натисніть двічі на пункті *ILA*. Каталог *IP* може мати будь-який вигляд з двох версій нижче. Групуйте/розгрупуйте кнопкою Group, щоб змінити зовнішній вигляд. Розгорніть папки "*Debug&Verification>Debug»* при необхідності (рис. 7.2).

У цій вправі ви будете підключати *ILA* ядро/компонент до 8-бітного світлодіодного порту.

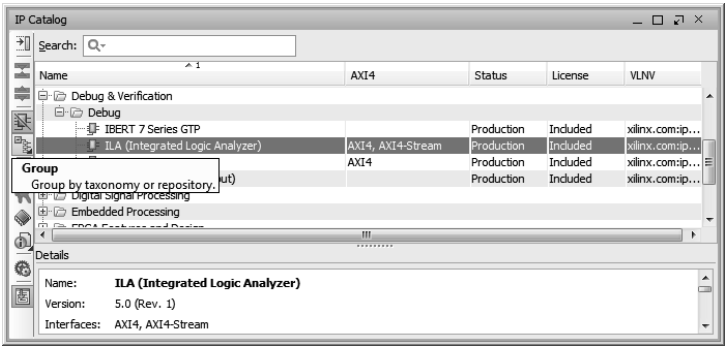


Рисунок 7.2 - *ILA* в *IP* каталозі

Змініть ім'я компоненту на ila_led. Змініть *Number* of *Probes* на 2 (рис. 7.3).

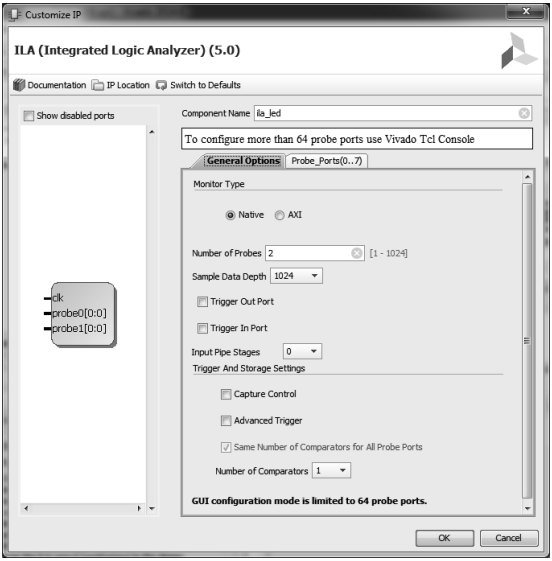


Рисунок 7.3 - Установка імені компоненту та кількості зондів

Виберіть вкладку *ProbePorts* і змініть ширину порту PROBE1 до 8; залиште ширину *PROBE0* 1 (рис. 7.4).

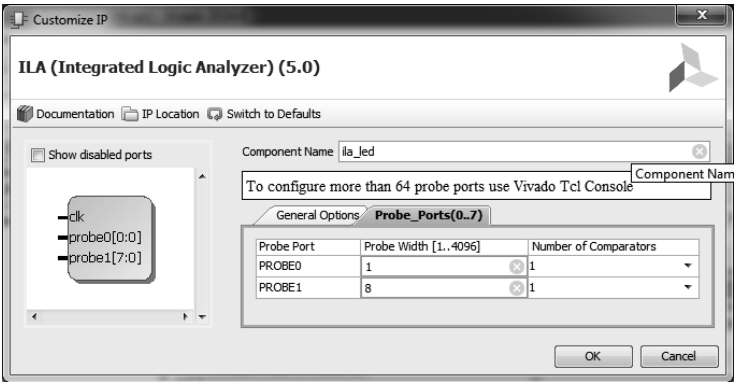


Рисунок 7.4 - Установка ширини зондів

Натисніть *OK*.

З’явиться діалогове вікно “*Generate Output Products”* (рис. 7.5).

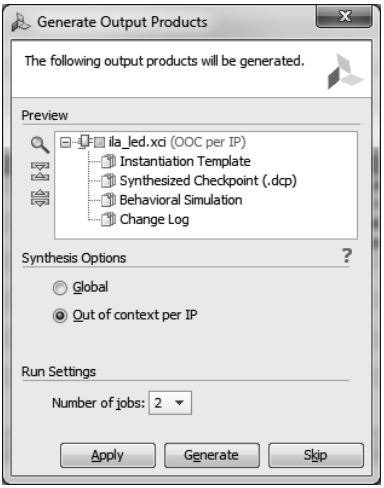


Рисунок 7.5 - Generate Output Products

Натисніть на кнопку *Generate* для генерації ядра, включаючи шаблон установки. Зверніть увагу, що ядро додане до перегляду DesignSources. Натисніть *OK* у вікні інформації (рис. 7.6).

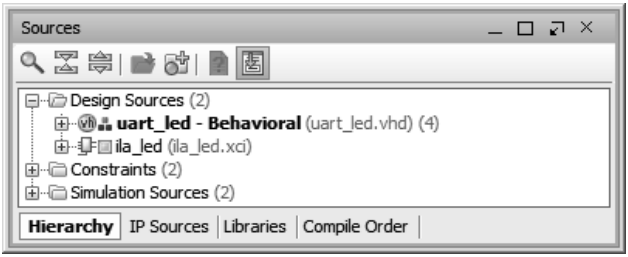


Рисунок 7.6 - Нове згенероване ядро ila, додане до вхідної схеми

Виберіть вкладку *IP*Sources, розширте *IP*(1) >ila_led>InstantiationTemplate та двічі натисніть на ila_led.vho для перегляду шаблону установки.

Зробіть опис ila_led в схемі шляхом копіювання рядків ~ 49 - 61 і додання до ~ рядку 49 (відразу після "architecture..” - область декларації архітектури) у файлі uart_led.vhd.

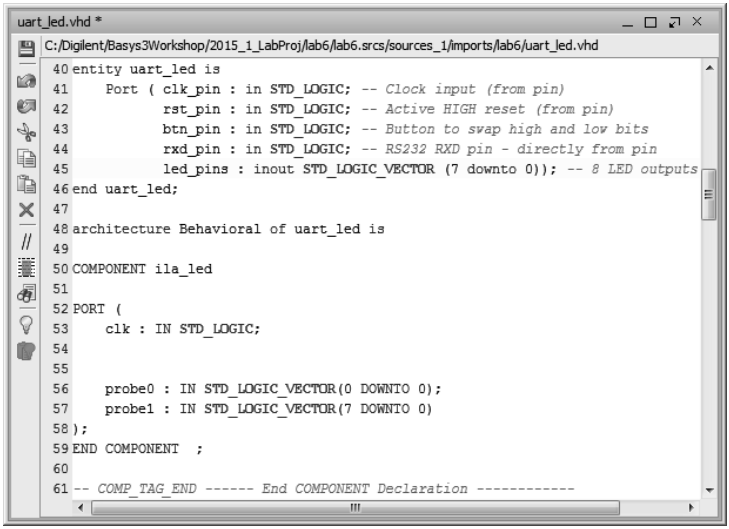
У файлі uart_led.vhd, entity/port змініть напрям*ок* порту led_pins з out на inout.

Створіть ila_led в схемі шляхом копіювання рядків ~ 66 - 79 з ila_led.vho і вставки в uart_led.vhd, з рядку 106 (відразу після рядку з "begin” –тіло архітектури) у файлі uart_led.vhd (рис. 7.7).

Змініть your_instance_name на ila_led_i0.

Змініть наступні імена портів в коді *VHDL* (*PORTMAP*) для з’єднання ila з існуючими сигналами в схемі:

clk =>clk_pin,
probe1 =>led_pins,
probe0(0) =>rx_data_rdy



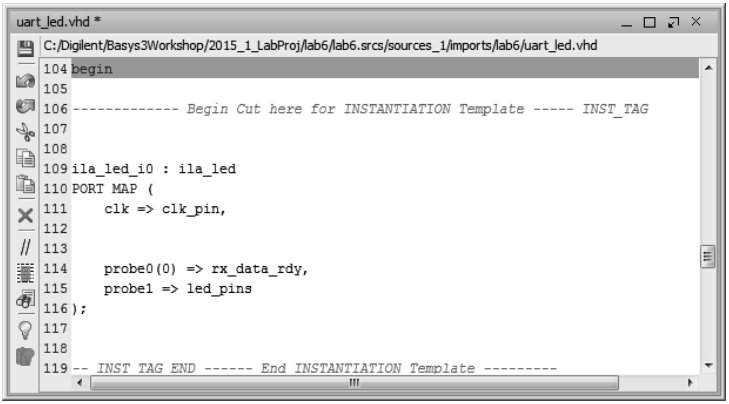


Рисунок 7.7 - Підготовлення, оголошення та створення ядру ILA в uart_led.vhd

Виберіть "*File>SaveFile*".

Зверніть увагу, що ядро ILA знаходиться в ієрархії схеми (рис. 7.8).

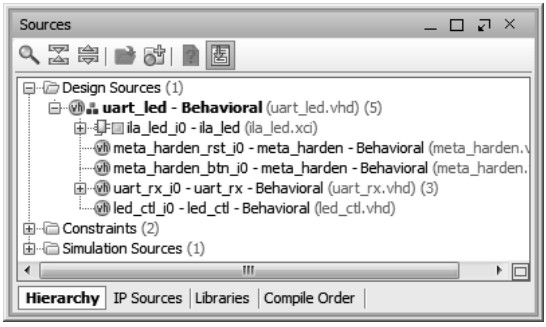


Рисунок 7.8 - Ядро ILA, додане до дизайну

Крок 3. Синтез схеми та позначки для відлагодження

Синтезуйте схему. Відкрити синтезовану схему. Додати MarkDebug до шини rx_data між об'єктами uart_rx_i0 та led_ctl_i0.

Натисніть на *RunSynthesis* під завданнями *Synthesis* в панелі FlowNavigator.

Процес синтезу почнеться в файлі uart_led.vhd та всіх ієрархічно залежних файлах. Коли процес завершиться, ви побачите діалогове вікно SynthesisCompleted з трьома варіантами вибору.

Виберіть опцію *OpenSynthesizedDesign* та натисніть *OK*.

Натисніть на Schematic під завданням SynthesizedDesign в панелі FlowNavigator для перегляду синтезованоїсхеми.

Виберіть шину rx_data між об'єктами uart_rx_i0 та led_ctl_i0, натисніть правою кнопкою та виберіть *MarkDebug* (рис. 7.9).

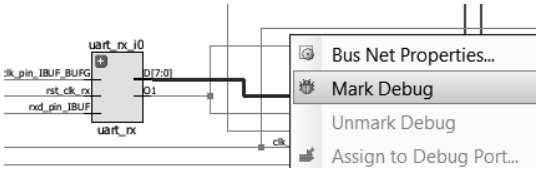


Рисунок 7.9 - Вибір шини для відлагодження

З'явиться попередження про те, що синтез застарів. Натисніть кнопку *OK* (рис. 7.10).

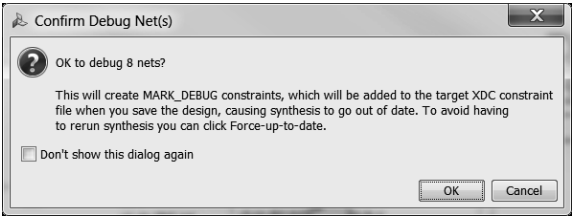


Рисунок 7.10 - Підтвердження діалогового вікна

Виберіть вкладку *Netlist* та зауважте, що мережі, які відмічені для відлагодження, мають відповідний знак біля них (рис. 7.11).

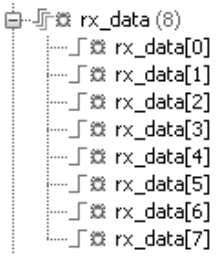


Рисунок 7.11 - Шини зі знаками відлагодження

Виберіть макет Debug або *“Layout>Debug”*.

Зверніть увагу на те, що вкладка Debug видна в панелі Console, п*ок*азуючи групи назначених та неназначених мереж для відлагодження (рис. 7.12).

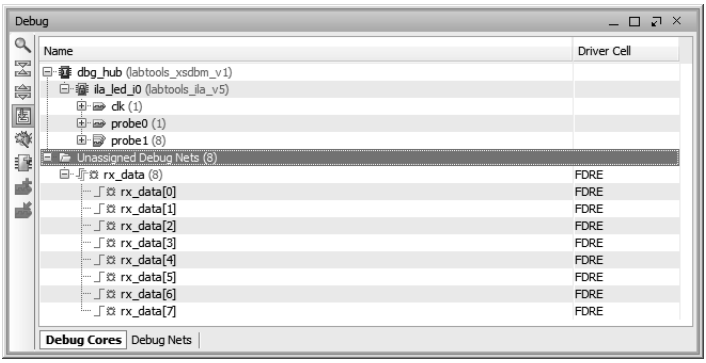


Рисунок 7.12 - Вкладка відлагодження, яка п*ок*азуєназначені та неназначені мережі

Натисніть кнопку у лівій вертикальній панелі або правою кнопкою миші на *UnassignedDebugNets* та виберіть опцію **SetupDebug**…

В майстрі *SetUpDebug* натисніть *Next*.

Зажміть *Ctrl* та виділіть всі мережі. Натисніть на бічній панелі кнопку *SelectClockDomain*, щоб призначити сигналам тактову область (рис.7.13).

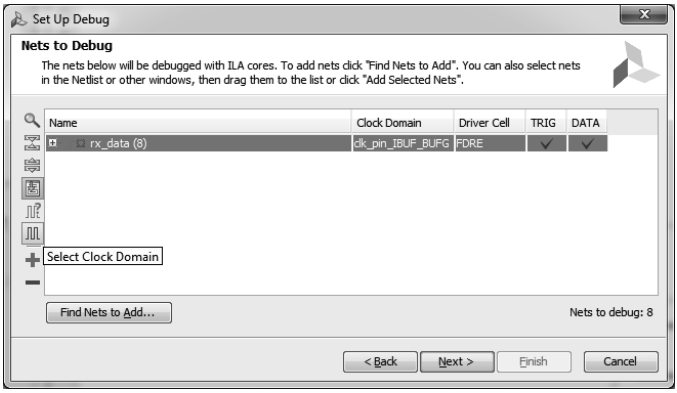


Рисунок 7.13 - Призначення тактової області сигналам

У вікні *SelectClockDomain* виберіть *ALL_CLOCK* з випадаючого меню ALL_CLOCK і зніміть прапорець Display uniquenets (рис. 7.14). Зверніть увагу на спис*ок* знайдених тактових сигналів. Виберіть clk_pin_IBUF_BUFG як тактову область. Натисніть кнопку *OK*, щоб закрити вікно.

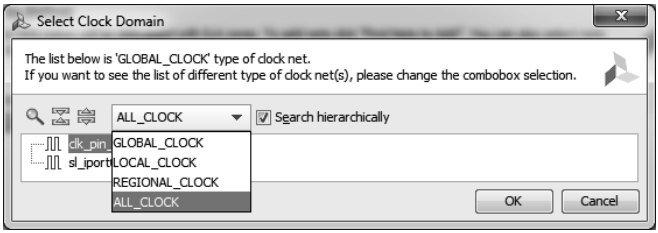


Рисунок 7.14 - Вікно *SelectClockDomain*

Мережі тепер наведено з clk_pin_*IBUF*_BUFG, обраним в якості тактової області. Оскільки деякі з мереж вже призначені, ми можемо видалити їх з цієї установки. Виберіть і видаліть мережі

led_pins (8) і rx_data_rdy. Натисніть *RemoveNets* на бічній панелі, щоб видалити раніше згадані мережі зі списку (рис. 7.15).

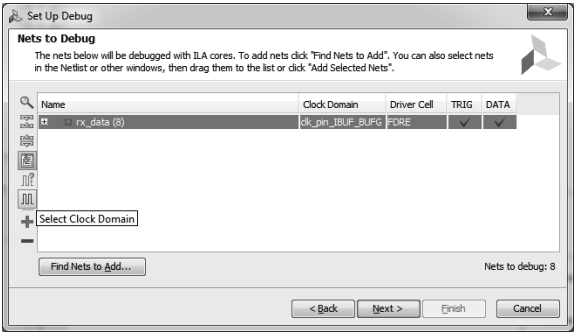


Рисунок 7.15 - Мережі, що залишились після видалення вже підключених мереж в майстрі *Set Up Debug*

Натисніть *Next* і знову *Next* (залишивши все за замовчуванням), і *Finish*.

В схемі SynthesizedDesignSchematic, натисніть на мережу з вихідного б*оку* *IBUF_BUFG* для вставки значку з іменем clk_pin (рис. 7.16).

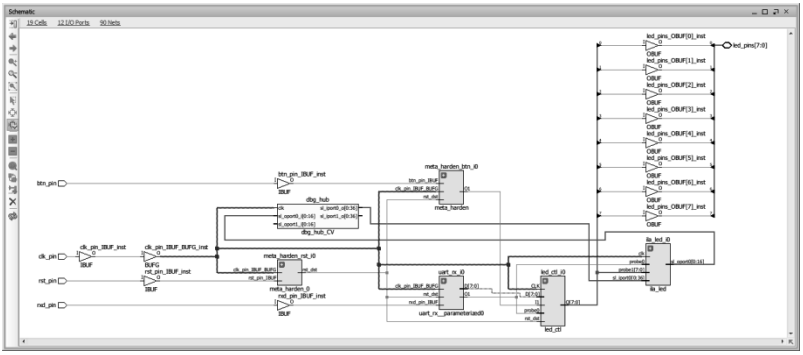


Рисунок 7.16 - Виявлення clk_pin_IBUF_BUFG в схемі

Натисніть правою кнопкою на uart_led_pins.xdc в панелі вхідних файлів та виберіть Set as Target Constrain File. Це збереже зміни до файлу.

Виберіть "*File>Save Constraints"* і натисніть *OK*. Натисніть *YES* для збереження змін синтезованої схеми. Натисніть *OK*.

Відкрийте uart_led_pins.xdc і зауважте, що мережі, що налагоджуються, були додані до нижньої частини файлу.

Крок 4. Створення та генерація потоку бітів

Згенеруйте потік бітів.

Натисніть на *GenerateBitstream* для запускуреалізаціїта процесу генерації бітів.

Натисніть *Yes* для запуску процесу реалізації.

Коли процес успішно завершиться, з'явиться вікно з чотирма варіантами. Виберіть опцію OpenHardwareManager та натисніть *OK*.

Крок 5. Налагодження в апаратних засобах

1**.** Підключіть плату і увімкніть живлення. Відкрийте апаратний сеанс і запрограмуйте *FPGA* .

Переконайтеся, що кабель *Micro-USB* підключений до роз'єму *JTAG* PROG (поруч з роз'ємом живлення). Переконайтеся, що перемичка встановлена, щоб вибрати живлення через *USB* (*JP2* на *Basys3*).

Включіть живлення.

Виберіть опцію "*Open Hardware Manager"* і натисніть *OK*.

Відкриється вікно *"Hardware Manager"*, яке вказує "непідключений" статус.

Натисніть на посилання *Open* a new hardware target.

Ви також можете натиснути на посилання *"Open recent target"*, якщо плата вже була вибрана до цього.

Натисніть *Next* для перегляду форми *"Vivado Hardware Server Settings"*.

Натисніть *Next3* обраним *"Target Hardware"*.

JTAG -кабель, який використовує xilinx_tcf, повинний бути виявлений і ідентифікований в якості апаратної мети. Він також покаже апаратні пристрої, виявлені в ланцюзі.

Натисніть *Next* та *Finish*.

Статус *"Hardware Session"* змінюється від *Unconnected* на ім'я серверу і пристрій засвітиться. Також зверніть увагу, що статус вказує на те, що він не запрограмований.

Виберіть пристрій і переконайтеся, що uart_led.bit обраний в якості файлу програмування на вкладці *General*.

2. Запустіть програму-емулятор терміналу, такі як TeraTerm або HyperTerminal. Виберіть відповідний *COM* порт (ви можете знайти правильний номер *COM* за допомогою панелі управління). Встановіть *COM*-порт на швидкість передачі 115200. Запрограмуйте *FPGA* і перевірте працездатність.

Запустіть програму-емулятор терміналу, такі як TeraTerm або HyperTerminal.

Виберіть відповідний *COM* порт (ви можете знайти правильний номер *COM* за допомогою панелі управління).

Встановіть *COM*-порт на швидкість передачі 115200.

Натисніть правою кнопкою на *FPGA* , виберіть ProgramDevice… та натисніть Program.

Завантажиться біт-файл програмування і засвітиться діод DONE, вказуючи, що *FPGA* запрограмований.

У вікні *"Hardware Manager"* відкриті дві вкладки: hw_ila_1 – ядра ILA та hw_ila_2 – методу *MARK DEBUG* (рис. 7.17).

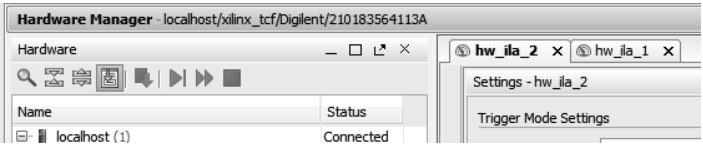


Рисунок 7.17 - Вкладки hw_ila

Також відкривається вікно стану апаратного сеансу, по*к*азуючи, що *FPGA* програмується з двома ila ядрами в неактивних станах (рис. 7.18).

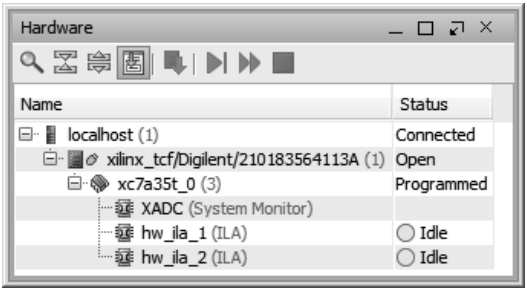


Рисунок 7.18 - Статус апаратного сеансу для *Basys3*

3. Установіть в умовах тригера запуск під час запису в порт led (rx_data_rdy = 1), а також позицію тригера на 512. Активуйте тригер.

У вкладці hw_ila_1, вікна Trigger Setup, натисніть на кнопку +, виберіть rx_data_rdy і натисніть *OK* (рис. 7.19).

У вікні "*Basic Setup Trigger"* натисніть кнопку, що розкривається, установіть значення порівняння (== [B] X) і змініть значення з x на 1. Натисніть кнопку *OK* (рис. 7.20).

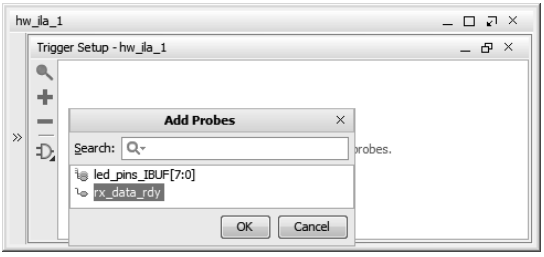


Рисунок 7.19 - Додання сигналу для налаштування тригеру

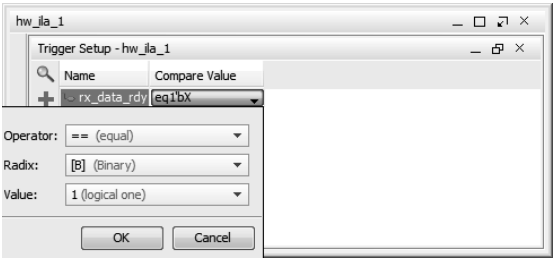


Рисунок 7.20 - Встановлення умов тригеру

Встановіть значення hw_ila_1 на 512 (рис. 7.21).

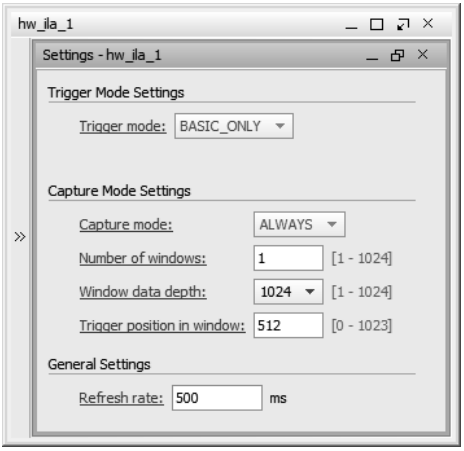


Рисунок 7.21 - Установка позиції тригеру

Аналогічним чином, установіть позицію hw_ila_2 на 512.

Виберіть hw_ila_1 у вікні Hardware, а потім натисніть на кнопку *RunTrigger* (). Зауважимо, що ядро hw_*ila*_1 активоване і показує статус *WaitingforTrigger* (рис. 7.22).

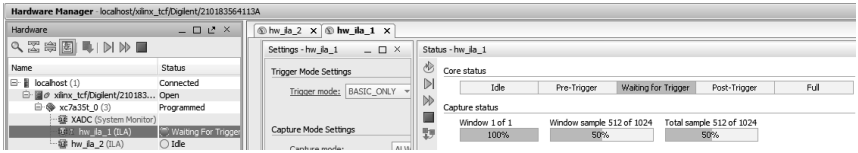


Рисунок 7.22 - Апаратний аналізатор, запущений в режимі зйомки

У вікні емулятора термінала введіть символ і спостерігайте, як статус hw_ila_1 змінюється від захоплення на холостий, коли rx_data_rdy стає 1.

Виберіть вікно hw_ila_data_1.wcfg і подивіться на графік. Натисніть значок, щоб побачити весь запис. Зверніть увагу на те, що rx_data_rdy йде від 0 до 1 на 512-му зразку і біти led_ctl_i0 [7:0] також змінюються від 0 до бітової комбінації, яка відповідає надрукованому символу (рис. 7.23).

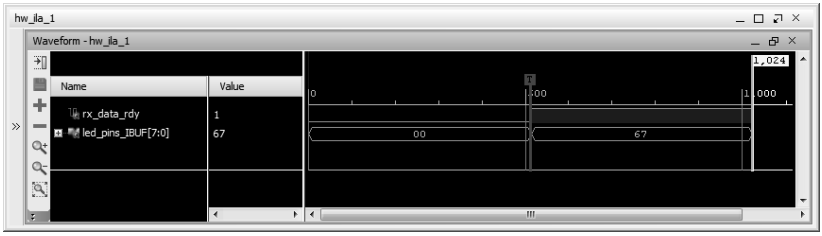


Рисунок 7.23 - Збільшений вигляд сигналу

Додайте зонди hw_ila_2 до вікна тригеру в hw_ila_2 і змініть в умовах тригеру основу для rx_data [7:0]'s з шістнадцяткової системи в двійкову (рис. 7.24). Змініть XXXX_XXXX до 0101_0101 (для ASCII еквівалент літери U).

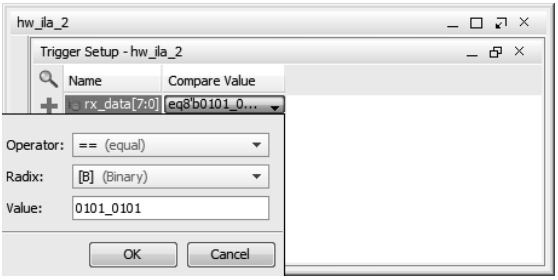


Рисунок 7.24 - Налаштування умов тригеру для конкретного вхідного шаблону

У вікні *Hardware*, клацніть правою кнопкою миші на hw_ila_2 і виберіть *RunTrigger* (або натисніть кнопку Run Trigger for this ilacore на вкладці hw_ila_2), і зверніть увагу, що статус hw_ila_2 змінюється з холостого до Waiting for Trigger. Також зверніть увагу, що статус hw_ila_1 не змінюється зхолостого, оскільки він не активований.

Перейдіть у вікно емулятора термінала і введіть *U (Shift + U)* для запуску тригера ядра.

Виберіть відповідну форму сигналу, натисніть *ZoomFit*, і переконайтеся, що він показує 55 після запуску тригеру (рис. 7.25).

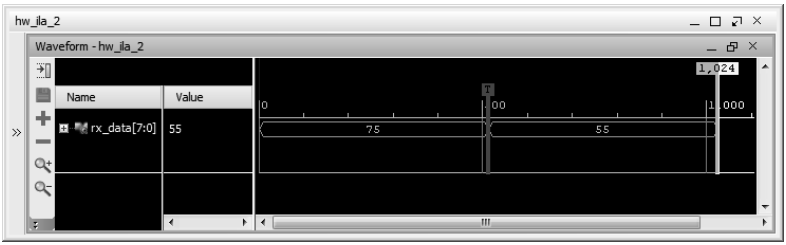


Рисунок 7.25 - Спрацювало друге ядро ila

Після завершення перевірки закрийте емулятор терміналу і відключіть живлення плати. Виберіть "*File>Close Hardware Manager"* і натисніть кнопку *OK*. Закрийте програму *Vivado*: *File>Exit* і натисніть кнопку *OK*.

Ви використовували ядро *ILA* з каталогу *IP* і функції *MarkDebug* ресурсу *Vivado* для налагодження апаратного дизайну.

7.3 Контрольні запитання

1. Для чого використовується функція *MarkDebug*?
2. Як додати у проект ядро *ILA*?
3. Як установити ширину зондів?
4. Як згенерувати ядро?
5. Опишіть процес підготовки, оголошення та створення ядру *ILA*.
6. Яку інформацію містить вкладка відлагодження?
7. Як призначити сигналам тактову область?
8. Як додати сигнал для налаштування тригеру?
9. Як налаштувати умови тригеру для конкретного вхідного шаблону?
10. Що показує апаратний аналізатор, запущений в режимі зйомки?

ЛІТЕРАТУРА

1. Bragina T. Testing methods library for applications with web-based interfaces / T. Bragina, G. Tabunshchyk, D. Moroka // Central European Researchers Journal. – 2015. Vol. 2. -pp.90-94

2. Digilent *Basys3* Board https://www.xilinx.com/support/university/boards-portfolio/xup-boards/Digilent*Basys3*Board.html#overview

3. Inside the Internet of Things (IoT). A primer on the technologies building the IoT. / J. Holdowsky, M. Mahto, M. E. Raynor, M. Cotteleer Режим доступу: http://dupress.deloitte.com/dup-us-en/focus/internet-of-things/iot-primer-iot-technologies-applications.html?icid=interactive:not:aug15#iot_foundations

4. "*JTAG* Boundary - Тестування сканування в MAXV пристроїв" Режим доступу: http://www.altera.com/literature/hb/max-v/mv51008.pdf

5. *JTAG* Boundary - Тестування сканування для CycloneIV пристроїв http://www.altera.com/literature/hb/cyclone-iv/cyiv-51010.pdf

6. Modelling, diagnostics and testing of digital systems [Available electronically]/ INTUIT. Режим доступу:: http://www.intuit.ru/studies/courses/3440/682/info [RU]

7. Yu. Skobtsov . Logical modelling and testing of digital systems /Yu. Skobtsov,. V. Skobtsov.- Donetsk:*IP*MM NASU, DonNTU, 2005.-436c [RU]

8. Vivado Design Suite - HLx Editions Режим доступу: https://www.xilinx.com/products/design-tools/vivado.html

9. Xilinx University Program Режим доступу: http://www.xilinx.com/support/university.html

10. Віддалений та віртуальний інструментарій в інжинірингу: монографія / А. Пархоменко, Г. Табунщик, М.

Поляков, О.Гладкова, Т. Брагіна, Т. Ларіонова. – Запоріжжя: Дике поле, 2015. – 250

11. Табунщик Г.В. Інженерія якості програмного забезпечення: навчальний посібник. – 2-ге вид. – Запоріжжя: ЗНТУ, Дике Поле. Табунщик Г.В, Кудерметов Р.К., Каплієнко Т.І., 2016. – 176 с

12. А.С. № 65139 Компьютерна програма «Бібліотека методів діагностики засобів бездротової передачі даних» /Табунщик Г.В. Каплієнко Т.І. ; опубл. 04.05.2016

АВТОРИ

DirkVAN MERODE Research Engineer EmSys Thomas More | Technology & Design Campus De Nayer.

In 2002 Dirk Van Merode finished his engineering studies in Electronics to become a Master in Science, in 2007moved to Lessius University College, currently renamed Thomas More University College, to take up a teaching assignment and to do research. Shepres of interests are research in space applications and satellite development.His field of expertise is in digital systems design, printed circuit board design and production, and audio-video production. Research topics are mainly in European projects, both on curriculum development and student and staff mobility with countries outside the EU. He is project coordinator of the DESIRE Tempus project, Development of Embedded System Courses with implementation of Innovative Virtual approaches for integration of Research, Education and Production in UA, GE, AM " – 544091-TEMPUS-1-2013-1-BE-TEMPUS-JPCR and partner in the APPLE Erasmus+ project, Applied curricula in space exploration and intelligent robotic systems. For the department electronics – ICT he is the international coordinator.

ТАБУНЩИК Галина Володимирівна – к.т.н., доцент, професор Запорізького національного технічного університету. Закінчила Запорізький державний технічний унверситет за спеціальність програмне забезпечення автоматизованих систем, захистила дисертацію на звання кандидат технічних наук зі спецальності 05.13.03 – системи і процеси керування. Автор понад 100 наукових праць. Наукові інтереси – інженерія програмного забезпечення, верифікація інформаційних систем, програмування вбудованих систем, керування ризиками

galina.tabunshchik@gmail.com

GalynaTABUNSHCHYK– PhD, Prof of Software Tool Department of Zaporizhzhya National Technical Univerity. Graduated from Zaporizhzhya National Technical University with speciality Software Engineering, in 2004 finished PhD work in control systems and process. Have more than 100 scientific works. Scientific interests – Software Engineering, System Verification, Embeded Systems, Risk Management

КАПЛІЄНКО Тетяна Ігорівна -кандидат технічних наук, доцент кафедри програмних засобів Запорізького національного технічного університету. Закінчила магістратуру за спеціальністю «Програмне забезпечення автоматизованих систем», аспірантуру за спеціальністю «Інформаційні технології». Захищено дисертацію у 2015 році. Автор 33 наукових публікацій, 3 авторських свідоцтв і одного патенту. Основні напрямки наукових досліджень: аналіз та верифікація якості програмного забезпечення; керування програмними проектами; керування ризиками.
bragina.zntu@gmail.com

Tetiana KAPLIIENKO, Ph.D., an associate professor of Software Tools Department at Zaporizhzhya National Technical University. She has a master's degree in "Software Engineering". She finished PhD study in "Information technology", and defended thesis in 2015. She is the author of 33 scientific publications, 3 Certificates for invention and 1 patent. Main research fields: the analysis and verification for the software quality; software design managements; risks management.

КАПЛІЄНКО Олександр Олегович – старший викладач кафедри електричних та електронних апаратів Запорізького національного технічного університету. Закінчив магістратуру за спеціальністю «Електричні машини та апарати», аспірантуру за спеціальністю «Електротехнічні комплекси та системи».Наукові інтереси – електроенергетика, електротехніка та електромеханіка; електрична інженерія;електромеханічне обладнання енергоємних виробництв;електричні та електронні апарати.
alexandr.kaplienko@gmail.com

Oleksandr KAPLIIENKO, assistant professor of Electric and Electronic Apparatus Department at Zaporizhzhya National Technical University. He has a master's degree in "Electric machines and apparatus". He finished PhD study in "Electrotechnical complexes and systems". He is the author of 21 scientific publications. Main research fields: power engineering, electrotechnology and electromecanics; electrical engineering; electromechanical equ*IP*ment energy-intensive industries; electrical and electronic devices.

Galyna Tabunshchyk
Tetiana Kapliienko
Oleksandr Kapliienko
Dirk Van Merode

Verification and Validation of Digital Control Systems

Students Textbook
(in Ukrainian)

Co-funded by the
Tempus Programme
of the European Union

CONTENTS

Навчальне видання

Табунщик Галина Володимирівна
Каплієнко Тетяна Ігорівна
Каплієнко Олександр Олегович
Дірк Ван Мероде

Верифікація та валідація цифрових систем управління

Навчальний посібник

Комп'ютерний набір	Г.В. Табунщик
Дизайн обкладинки	М.О. Андреєв
Технічний редактор	Л. А. Рябоконь
Коректор	Н. В. Чечеко

Формат 60x84/16.
Папір офсетний. Гарнітура *Times*. Друк офсетний. 300 прим.
Підписано до друку 20.07.2017. Ум. друк. арк. 8,71.

Видавництво «Дике Поле»
Україна, 69063, м. Запоріжжя, вул. Троїцька, 31-А.
Тел.: (061) 213-75-95; 213-75-05.
Свідоцтво суб'єкта видавничої справи ЗЗ № 004
від 23.08.2001 р.