Синхронний дворозрядний вісімковий лічильник

Design Specification

Розробити синхронний дворозрядний вісімковий лічильник (Synchronous Two-Digit Octal Counter) з асинхронним скиданням (Reset).

Schematic

На рівні логічних вентилів (Gate-Level) дизайн базується на D-тригерах та комбінаційній логіці керування. Проектування на рівні логічних елементів виконано у середовищі Intel Quartus Prime. Дворозрядний вісімковий лічильник складається з двох 3-х бітних лічильників, кожен з яких рахує від 0 до 7 та має вихід переповнення (o_co). На рис.1 представлена схема на логічних елементах одного 3-х бітного лічильника (counter_3bit).

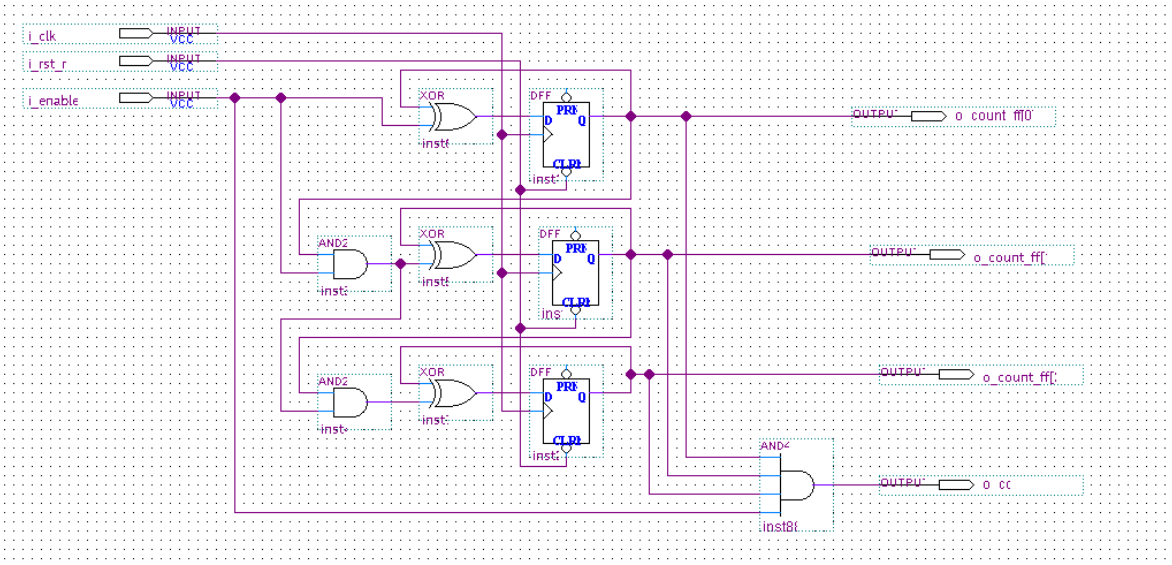

Рисунок 1 – 3-х бітний лічильник

На рис. 2 представлено вейформи симуляції 3-розрядного лічильника, що демонструють повний цикл рахунку від 0 до 7 та переповнення.

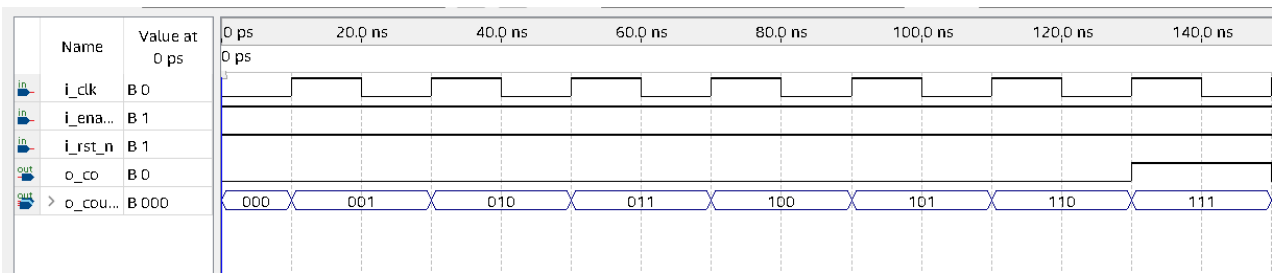

Рисунок 2 - Часові діаграми роботи 3-бітного синхронного лічильника.

На рис. 3 зображено архітектуру верхнього рівня (octal_counter_top), яка складається з двох екземплярів базового модуля counter_3bit.

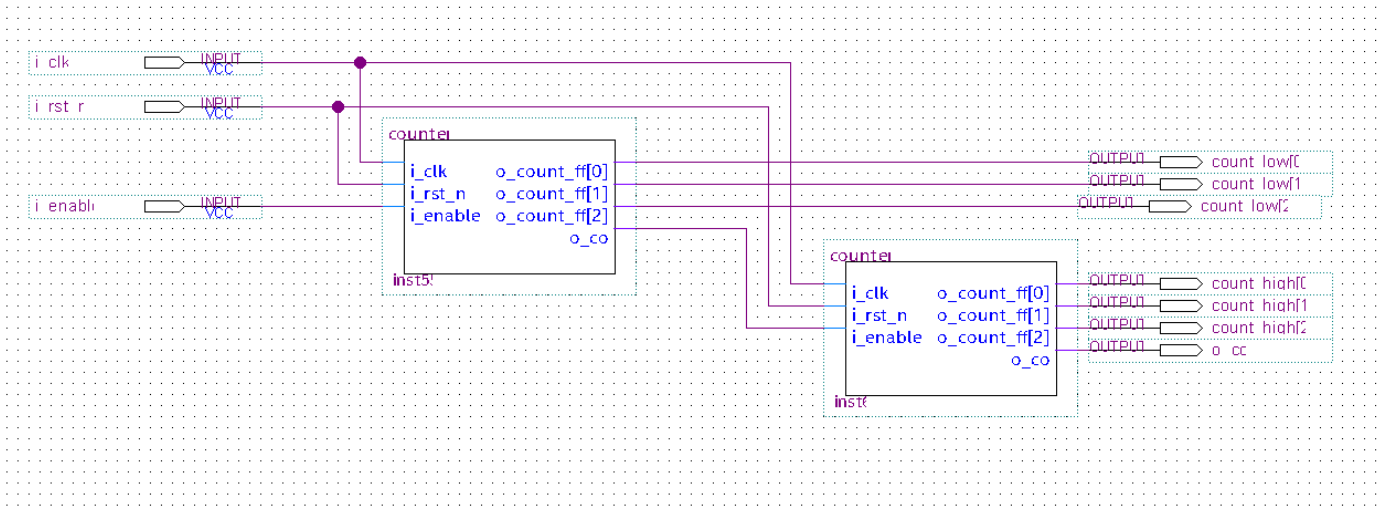

Рисунок 3 - Схема дворозрядного вісімкового лічильника

На рис. 4 наведено результати симуляції роботи повного дворозрядного лічильника.

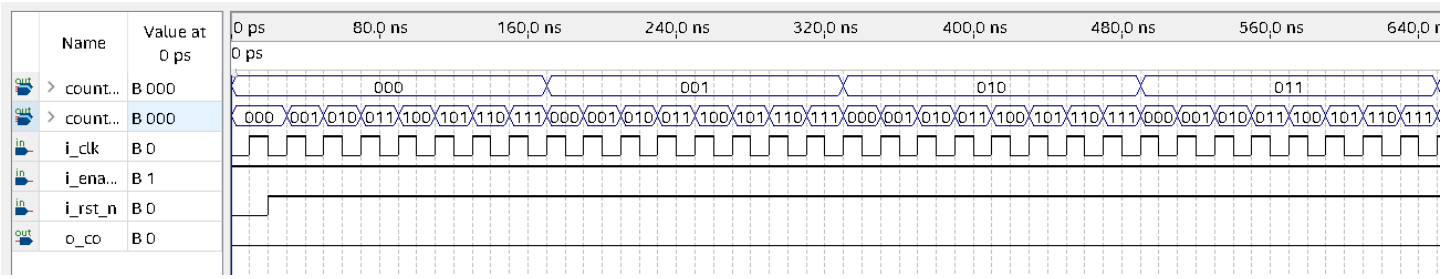

Рисунок 4 - Часові діаграми роботи повного лічильника

RTL

RTL-опис дизайну виконано мовою SystemVerilog. Код повністю відповідає розробленій Gate-Level схемі та складається з двох рівнів ієрархії:

Базовий модуль (counter_3bit)

Послідовнісна логіка: Блок always_ff описує поведінку D-тригерів, які запам'ятовують стан по фронту тактового сигналу (posedge clk) та реагують на асинхронне скидання (negedge rst_n).

Комбінаційна логіка: Блок always_comb реалізує логічні операції.

Верхній рівень (octal_counter_top)

Цей модуль об'єднує два екземпляри 3-бітних лічильників для формування дворозрядного числа.

Інстанціювання: Створено два об'єкти: lower_inst (молодший розряд) та upper_inst (старший розряд).

Логіка каскадування: Реалізовано механізм перенесення розряду. Старший лічильник отримує сигнал дозволу на рахунок лише за умови, що молодший лічильник досяг свого максимального значення (7) і загальний сигнал i_enable активний.

Verification

Для перевірки коректності функціонування RTL-дизайну було розроблено тестове оточення (Testbench) з використанням мови SystemVerilog. Застосовано підхід Constrained Random Verification.

Архітектура тестового оточення:

Generator: Відповідає за створення випадкових вхідних транзакцій. Генерує випадкові значення сигналу i_enable. Реалізує ймовірнісний розподіл для сигналу i_rst_n (98% — активна робота, 2% — випадкове скидання).

Driver: Отримує транзакції від генератора та подає їх на фізичний інтерфейс лічильника.

Scoreboar: Містить еталонну модель (Golden Model), яка програмно відтворює очікувану поведінку лічильника. Порівнює реальний вихід RTL-моделі (o_cnt_low, o_cnt_high) з очікуваним значенням у кожному такті. У разі розбіжності (Mismatch) симуляція фіксує помилку.

Environment: Створює та з'єднує між собою всі компоненти верифікації (Generator, Driver, Scoreboard). Контролює порядок виконання тесту: спочатку ініціалізація (Reset), потім активна генерація транзакцій, і насамкінець — збір фінальної статистики.

Top Testbench: Модуль верхнього рівня, що об'єднує програмну та апаратну частини. Генерує системний тактовий сигнал. Інстанціює модуль лічильника та підключає його до тестового інтерфейсу. Створює об'єкт класу Environment та запускає головний процес симуляції.

Симуляцію та верифікацію проекту виконано у середовищі Cadence Xcelium. На рис. 5 представлено результати успішної симуляції: розподіл тестових циклів та відсутність помилок.

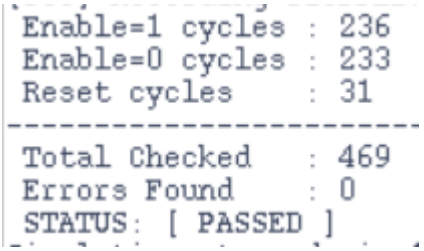

Рисунок 5 - Фрагмент логу консолі симулятора

Possibility of metastability

Якщо зміна вхідного сигналу відбувається занадто близько до фронту тактового сигналу (порушуючи час встановлення Setup Time або утримання Hold Time), тригер може увійти в невизначений стан (не 0 і не 1). Для розробленого лічильника було ідентифіковано два критичних шляхи, вразливих до метастабільності:

1. Асинхронне скидання (i_rst_n)

Якщо сигнал скидання з активним низьким рівнем деактивується (переходить з 0 в 1) занадто близько до фронту тактового сигналу - це може призвести до того, що тригери перейдуть у метастабільний стан.

Для вирішення цієї проблеми реалізовано окремий модуль синхронізатора, представлений на рисунку 6.

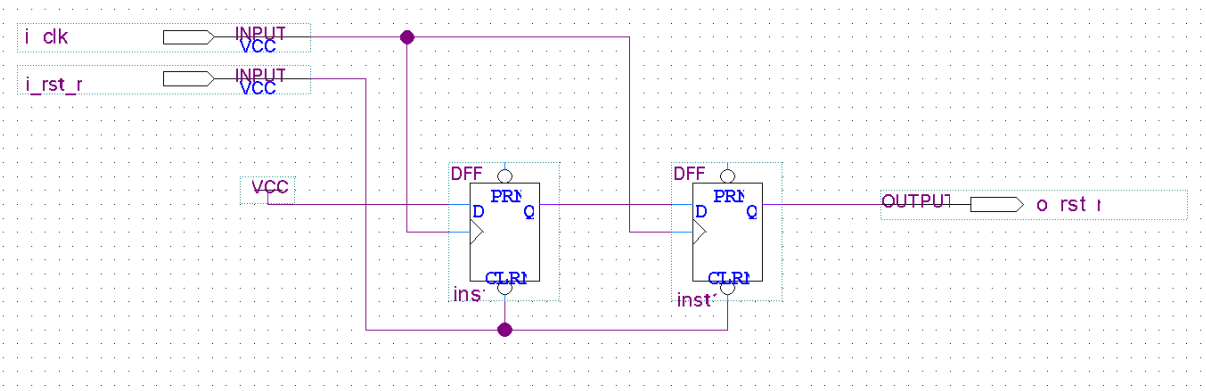

Рисунок 6 – Модуль синхронізації i_rst_n

Скидання відбувається асинхронно (миттєво), а вихід зі скидання — строго синхронно.

2. Асинхронний дозвіл (i_enable)

Сигнал дозволу надходить із зовнішнього середовища. Його зміна в довільний момент часу може спричинити порушення таймінгів.

Для вирішення цієї проблеми реалізовано окремий модуль синхронізатора, представлений на рисунку 7.

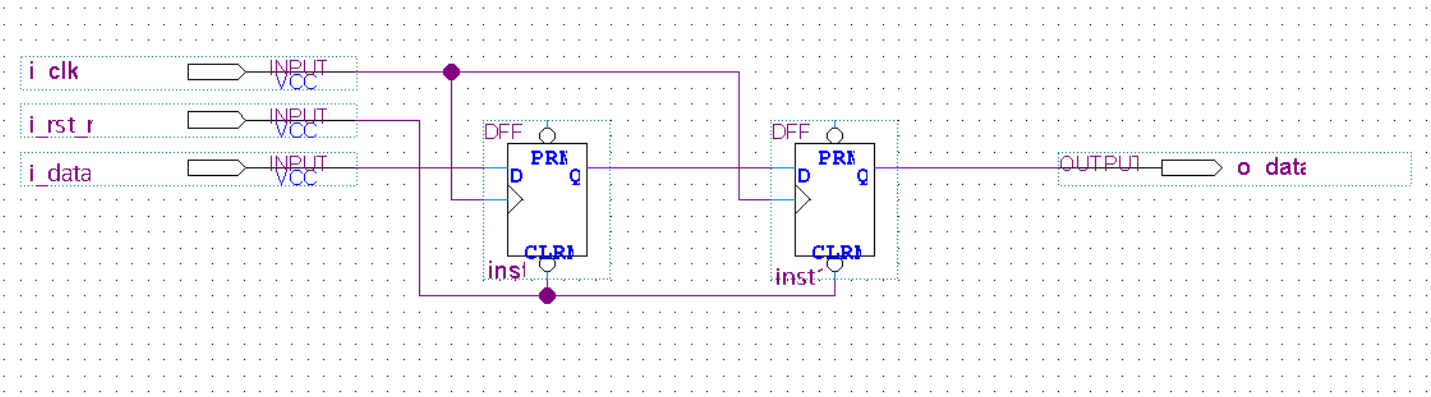

Рисунок 7 – Модуль синхронізації i_enable

Сигнал проходить через ланцюжок із двох D-тригерів. Якщо перший тригер потрапляє у метастабільний стан через порушення таймінгів, протягом повного такту тригер з найвищою ймовірністю перейде у стабільний стан (0 або 1). І тоді другий тригер отримає це значення.

На найвищому рівні ієрархії реалізовано інтеграцію всіх компонентів системи. Як видно з Рис. 8, асинхронні зовнішні сигнали (i_rst_n, i_enable) не подаються напряму на лічильник. Спочатку вони проходять через блоки rst_sync та data_sync.

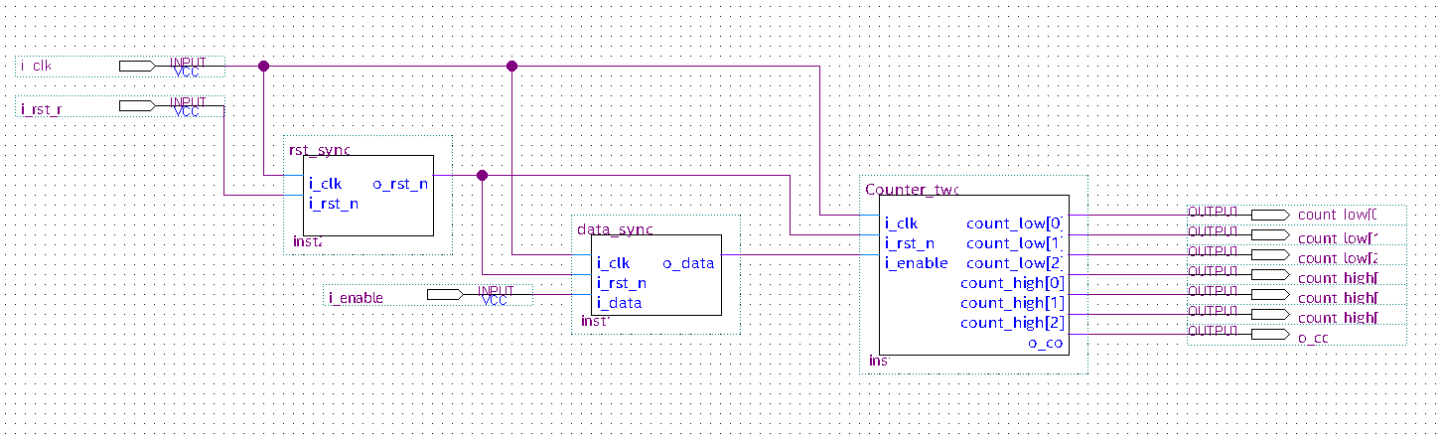

Рисунок 8 - Структурна схема верхнього рівня ієрархії (top.sv)

Впровадження схем захисту від метастабільності неминуче вносить затримку. Загальна затримка від зміни входу до зміни вихідного значення лічильника складає 4 такти (2 такти синхронізації + такти внутрішньої логіки лічильника). На Рис. 9 проілюстровано цей ефект:

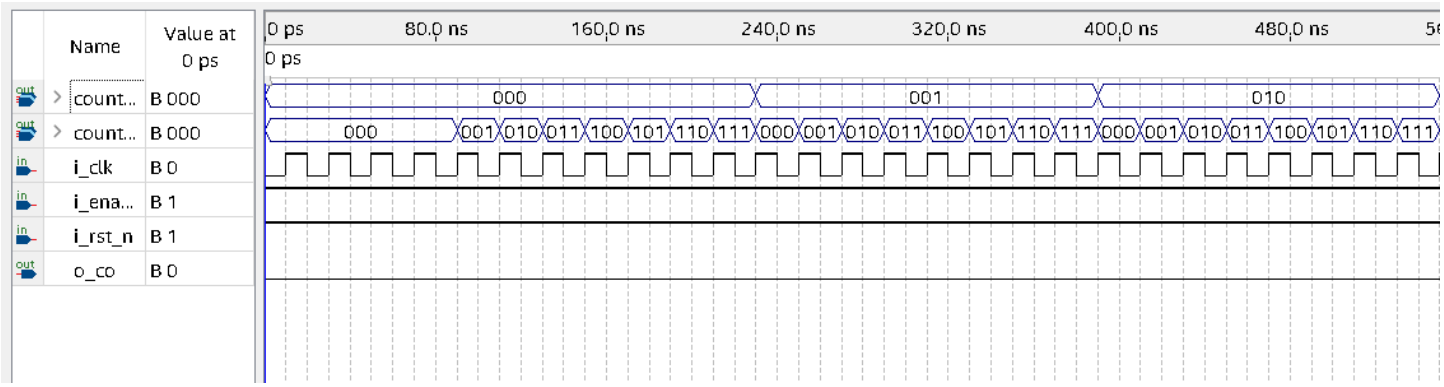

Рисунок 9 - Часова діаграма затримок проходження сигналу

Static Timing Analysis

Для підтвердження можливості роботи пристрою на цільовій частоті 100 МГц було проведено статичний часовий аналіз (STA) у середовищі Intel Quartus Prime.

У файлі часових обмежень (.sdc) було задано параметри тактового сигналу:

create_clock -period 100MHz [get_ports i_clk]

derive_clock_uncertainty

Аналізатор (Timing Analyzer) розрахував затримки на всіх шляхах схеми з урахуванням фізичного розміщення елементів на кристалі FPGA. (рис. 10)

	Clock	Slack	End Point TNS
1	i_clk	7.806	0.000

(a)

	Clock	Slack	End Point TNS
1	i_clk	0.427	0.000

(b)

Рисунок 10 - (а) Setup Summary — перевірка часу встановлення; (б) Hold Summary — перевірка часу утримання.

Setup Summary. Slack: +7.806 нс. Це підтверджує, що сигнал встигає пройти через логічні елементи до наступного фронту тактового сигналу. Великий запас дозволяє потенційно збільшити частоту роботи пристрою.

Hold Summary. Slack: +0.427 нс. Позитивне значення свідчить про відсутність "гонок сигналів" (Race Conditions). Дані залишаються стабільними достатній час після фронту тактового сигналу, що гарантує коректний запис у тригери.

Висновок: Оскільки значення End Point TNS у обох випадках дорівнює 0.000, схема не має критичних порушень таймінгу.

MTBF (Mean Time Between Failures)

Відповідно до книги Д. Гарріс та С. Гарріс «Цифрова схемотехніка та архітектура комп'ютера», середній час між помилками синхронізації визначається за наступною формулою:

$$MTBF = \frac{1}{P(\text{failure})/\text{sec}} = \frac{T_c e^{-\frac{T_c - t_{setup}}{\tau}}}{NT_0},$$

Де

Tc - період тактового сигналу для робочої частоти проекту 100 MHz (10 нс);

t_setup - час встановлення тригера;

τ, T0 - технологічні константи;

N - кількість змін асинхронного вхідного сигналу за одну секунду.

Оскільки у мене немає технологічних констант, для розрахунку використано параметри з книги Д. Гарріса та С. Гарріса

$$MTBF = \frac{10^{-8} * e^{\frac{7.806*10^{-9}}{200*10^{-12}}}}{1 * (150 \times 10^{-12})} = 5.9 * 10^{18}\ c = 1.87 * 10^{11}\ \text{років}$$

Якби N було більше то відповідно MTBF зменшилось би.