

# Лабораторна робота №0

## Базові логічні елементи

<b>0.1 Практичне застосування досліджуваних схем</b>	<b>1</b>
<b>0.2 Теоретична частина</b>	<b>1</b>
0.2.1 Логічні рівні цифрового сигналу	1
0.2.2 Параметри імпульсного сигналу	5
0.2.3 Логічні елементи	6
0.2.4 Компаратор на логічних елементах	7
0.2.5 Мікросхеми програмованої логіки типу FPGA	8
0.2.6 Опис модуля у мові Verilog	10
0.2.7 Типи wire і reg у мові Verilog	11
0.2.8 Оператор assign у мові Verilog	14
<b>0.3 Практична частина</b>	<b>15</b>
0.3.1 Завантаження та встановлення Quartus Prime Lite	15
0.3.2 Налагоджувальні плати для виконання лабораторних робіт	16
0.3.3 Створення проекту в Quartus Prime Lite	16
0.3.4 Імпорт налаштувань портів вводу-виводу в Quartus Prime Lite	20
0.3.5 Створення схем у схемному редакторі	22
0.3.6 Компіляція проекту в Quartus Prime Lite	27
0.3.7 Конфігурування FPGA за результатами компіляції в Quartus Prime Lite	27
0.3.8 Опис проекту з використанням мови Verilog	29
0.3.9 Перегляд результатів компіляції в Netlist Viewers	31
0.3.10 Завдання на лабораторну роботу	33
<b>0.4 Контрольні запитання</b>	<b>34</b>
<b>0.5 Перелік посилань</b>	<b>35</b>

## 0.1 Практичне застосування досліджуваних схем

В початковій лабораторній роботі ми дослідимо базові логічні елементи І (AND), АБО (OR), НІ (NOT). Ці логічні елементи особливі тим, що на їх основі можна реалізувати будь яку логічну функцію. Тому зазначені логічні елементи широко застосовуються в сучасних мікросхемах. Також розглянемо логічні елементи ВИКЛЮЧАЮЧЕ-АБО (XOR), І-НІ (NAND), АБО-НІ (NOR), ВИКЛЮЧАЮЧЕ-АБО-НІ (XNOR), які можна побудувати використовуючи базові логічні елементи.

У якості практичного завдання створимо перший проект на базі програмованої логіки типу FPGA, в якому реалізуємо найпростіші логічні елементи у схемному редакторі та за допомогою мови Verilog. Наостанок створимо проект компаратора для порівняння двох 4-розрядних чисел.

## 0.2 Теоретична частина

### 0.2.1 Логічні рівні цифрового сигналу

Цифрові мікросхеми оперують числами в двійковій формі. Відомо, що один біт даних може приймати два стійких стани: **0** і **1**. Існує ще третій, високоімпедансний стан, але його зараз розглядати не будемо.

В цифровій схемотехніці логічний 0 та логічну 1 представляють діапазони напруг. Напруга з діапазону **від 0 до  $V_L$**  представляє логічний 0, а напруга з діапазону **від  $V_H$  до  $V_{dd}$**  (напруга живлення) представляє логічну 1. Для різних типів мікросхем і різних напруг живлення  $V_L$  та  $V_H$  приймають різні значення.

Напруга з діапазону від  $V_L$  до  $V_H$  не представляє ні 0 ні 1 і не повинна подаватися на входи цифрових мікросхем (якщо вони не обладнані тригерами Шмітта), інакше можлива некоректна робота мікросхеми. Тому напруги від  $V_L$  до  $V_H$  заборонені на провідниках, що передають цифрові сигнали.

Графічне представлення діапазонів напруг для логічних рівнів наведено на рис.0.1

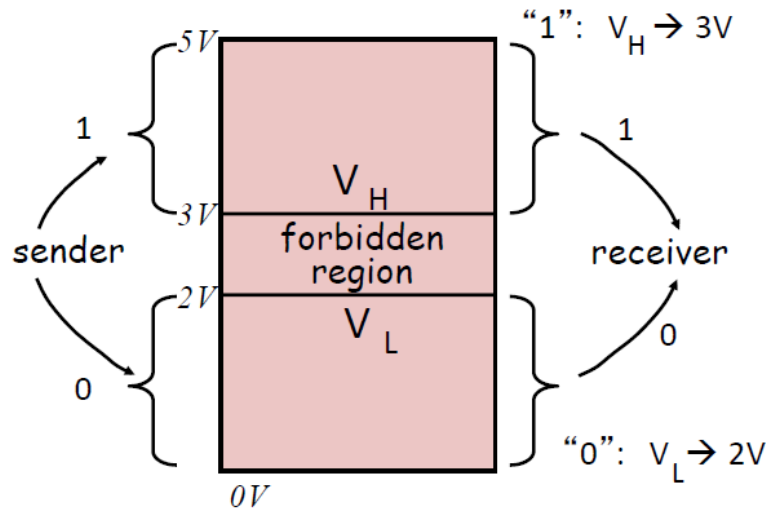


Рис.0.1 - Графічне представлення діапазонів напруг логічних рівнів для напруги живлення  $V_{dd} = 5V$

Можлива наступна ситуація. Передавач виставляє на виході коректну напругу логічного рівня, скажімо 3.1В, що представляє логічну 1. На цю напругу накладається завада величиною 0.2В від електромагнітних хвиль, або від сусідніх провідників через паразитні ємності друкованої плати. В результаті напруга логічного сигналу зменшується з 3.1В до 2.9В, попадає в діапазон заборонених напруг і може бути некоректно зчитана приймачем. Подібна ситуація проілюстрована на рис.0.2

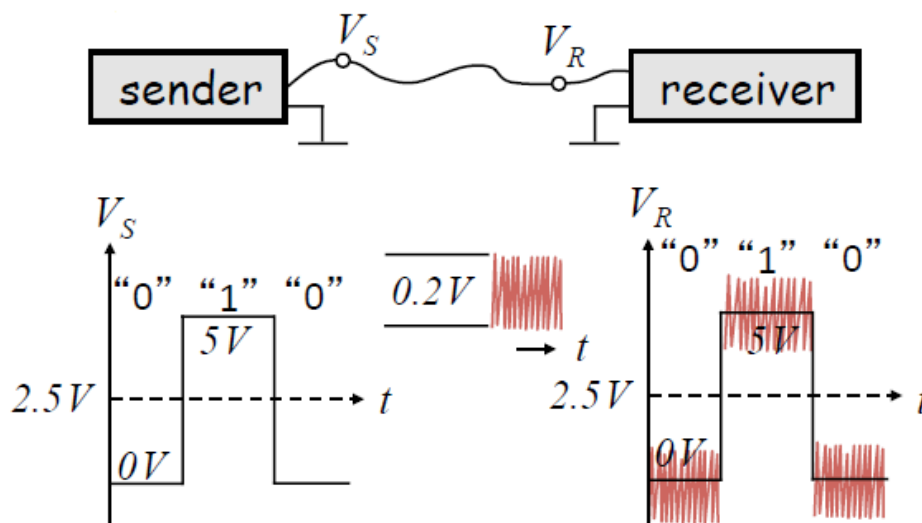


Рис.0.2 - Напруги логічних рівнів на які наклалися сигнали завад

Для боротьби з завадами діапазон напруг логічних рівнів приймача роблять більшим ніж у передавача - рис.0.3.

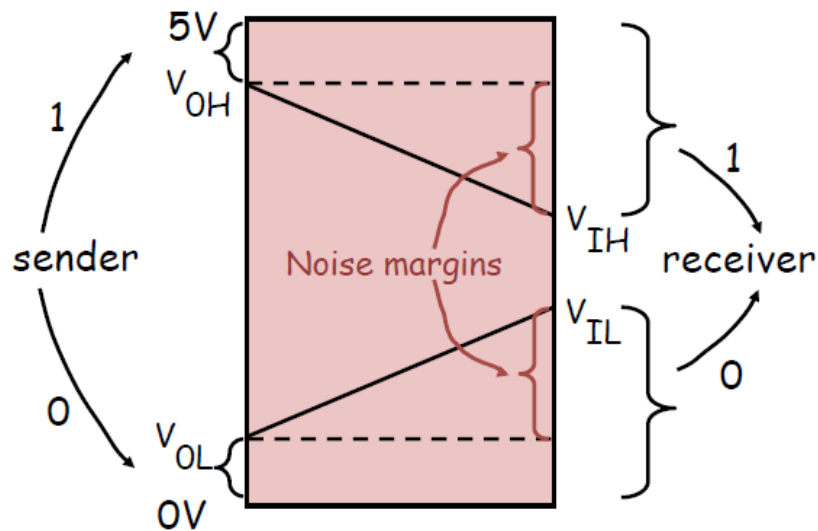


Рис.0.3 - Діапазони напруг логічних рівнів передавача (sender) і приймача (receiver)

В такому випадку діапазони напруг логічних рівнів передавача лежать в межах  $0 \dots V_{OL}$  для лог. 0 та  $V_{OH} \dots V_{dd}$  для лог.1. Для приймача лог. 0 представляється діапазоном напруг  $0 \dots V_{IL}$ , а лог. 1 представляється діапазоном напруг  $V_{IH} \dots V_{dd}$ .

Рис 0.4 ілюструє коректну передачу даних при наявності завад між передавачем і приймачем з різними діапазонами логічних рівнів.

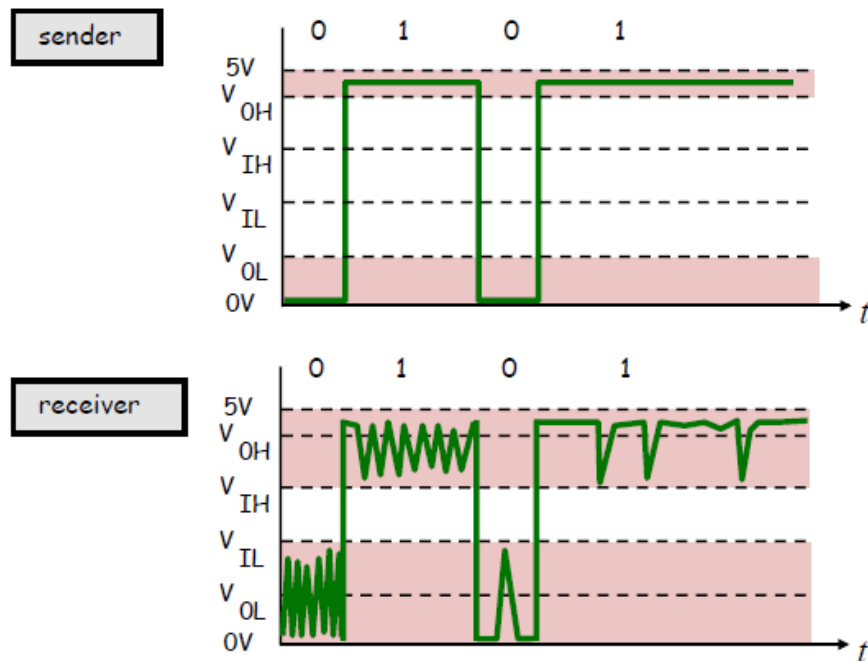


Рис.0.4 - Передача даних при наявності завад між передавачем і приймачем з різними діапазонами логічних рівнів

Деякі широко розповсюджені логічні рівні:

- TTL ( $V_{DD}=5V$ )
- CMOS ( $V_{DD}=5V$ )
- LVTTL ( $V_{DD}=1.8V, 2.7V, 3.3V$ )
- LVCMOS ( $V_{DD}=1.8V, 2.7V, 3.3V$ )
- HSTL (High-speed transceiver logic)
- SSTL (Stub Series Terminated Logic). Використовується в DDR SDRAM
- LVDS (differential)
- RS-232 levels
- RS-485 levels (differential)
- PCI logic levels

Детальніше зі специфікаціями логічних рівнів можна ознайомитись за посиланням [\[0.1\]](#)

Багато сучасних мікросхем, зокрема широко розповсюджені мікроконтролери STM32, використовують логічні рівні LVCMOS.

Logic Family	$V_{DD}$	$V_{IL}$	$V_{IH}$	$V_{OL}$	$V_{OH}$
TTL	5 (4.75 - 5.25)	0.8	2.0	0.4	2.4
CMOS	5 (4.5 - 6)	1.35	3.15	0.33	3.84
LVTTL	3.3 (3 - 3.6)	0.8	2.0	0.4	2.4
LVCMOS	3.3 (3 - 3.6)	0.9	1.8	0.36	2.7

Рис.0.5 - Значення логічних рівнів TTL, CMOS, LVTTL, LVCMOS

Логічні рівні якими оперує мікросхема завжди можна знайти в її документації.

## 0.2.2 Параметри імпульсного сигналу

Вигляд імпульсного сигналу наведено на рис.0.6.

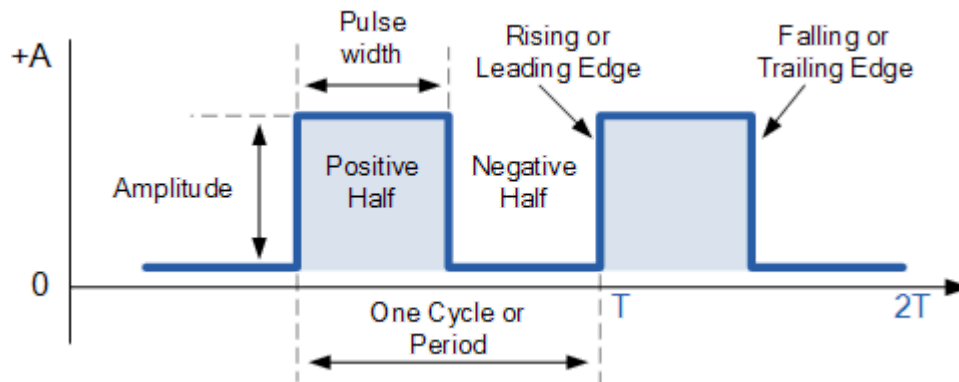


Рис.0.6 - Вигляд імпульсного сигналу

Максимальне значення напруги імпульсного сигналу називають амплітудою імпульсного сигналу. Зазвичай амплітуда приймає значення напруги логічної 1.

Інтервал часу між двома сусідніми імпульсами називається **періодом** імпульсів. Період часто позначається літерою **T** і вимірюють в одиницях часу.

Величина обернена до періоду називається частотою і показує скільки імпульсів відбувається за секунду. **Частота** вимірюється в Герцах [Гц]. Частоту часто позначають літерою **F**. Очевидно, що  $F = 1/T$

Інтервал часу протягом якого сигнал приймає ненульове значення називають **тривалістю імпульсу**. Тривалість імпульсу часто позначають літерою  $\tau$ .

Відношення тривалості імпульсу до періоду ( $\tau/T$ ) називають **коефіцієнтом заповнення** імпульсного сигналу. Ми повернемося до цього визначення під час вивчення широтно-імпульсної модуляції.

Зміну сигналу з лог. 0 на лог. 1 називають **переднім фронтом** сигналу.

Зміну сигналу з лог. 1 на лог. 0 називають **заднім фронтом** сигналу.

Зазвичай фронти відбуваються не миттєво, а протягом певного часу, тобто передній і задній фронт мають певну тривалість.

Передній і задній фронти сигналу проілюстровані на рис.0.7.

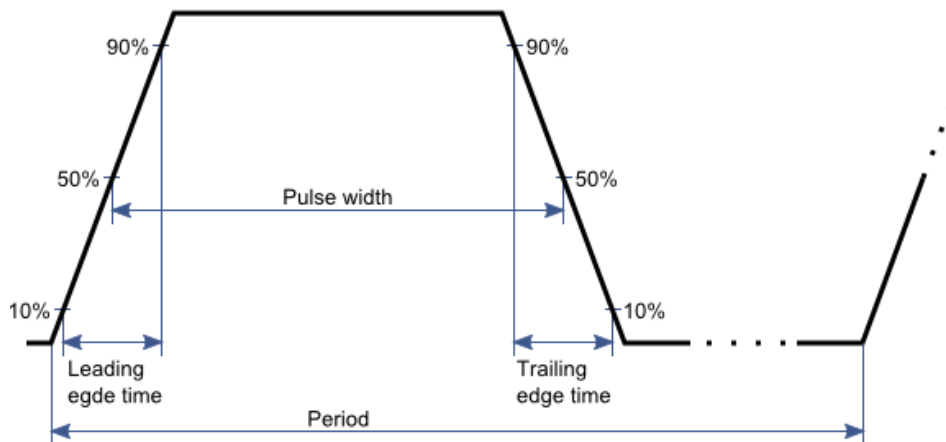


Рис.0.7 - Передній і задній фронти імпульсу

### 0.2.3 Логічні елементи

**Комбінаційною схемою** (комбінаційною логікою, combinational logic) називається цифрова схема, сигнали на виходах якої залежать виключно від сигналів на входах. Операції, які виконуються такою схемою повністю описуються таблицею істинності, в якій кожній комбінації сигналів на входах поставлена у відповідність комбінація сигналів на виходах. Комбінаційні схеми не містять пам'яті.

Найпростішими комбінаційними схемами є логічні елементи (вентилі), що реалізують логічні операції І (AND), АБО (OR), НІ (NOT). Комбінаційну схему будь якого рівня складності можна побудувати з використанням вентилів цих трьох типів.

Умовні графічні позначення і таблиці істинності логічних елементів І (AND), АБО (OR), НІ (NOT) зображені на рис.0.8.

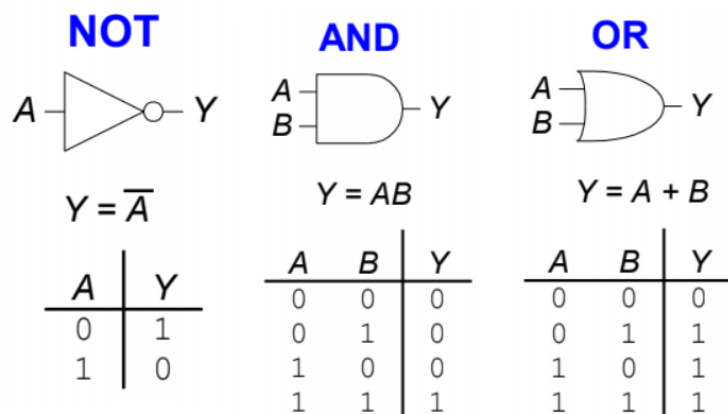


Рис.0.8 - Умовні графічні позначення вентилів І (AND), АБО (OR), НІ (NOT)

Окрім базових логічних елементів часто використовують ВИКЛЮЧАЮЧЕ-АБО (XOR), І-НІ (NAND), АБО-НІ (NOR), ВИКЛЮЧАЮЧЕ-АБО-НІ (XNOR). Умовні графічні позначення і таблиці істинності зазначених елементів зображені на рис.0.9.

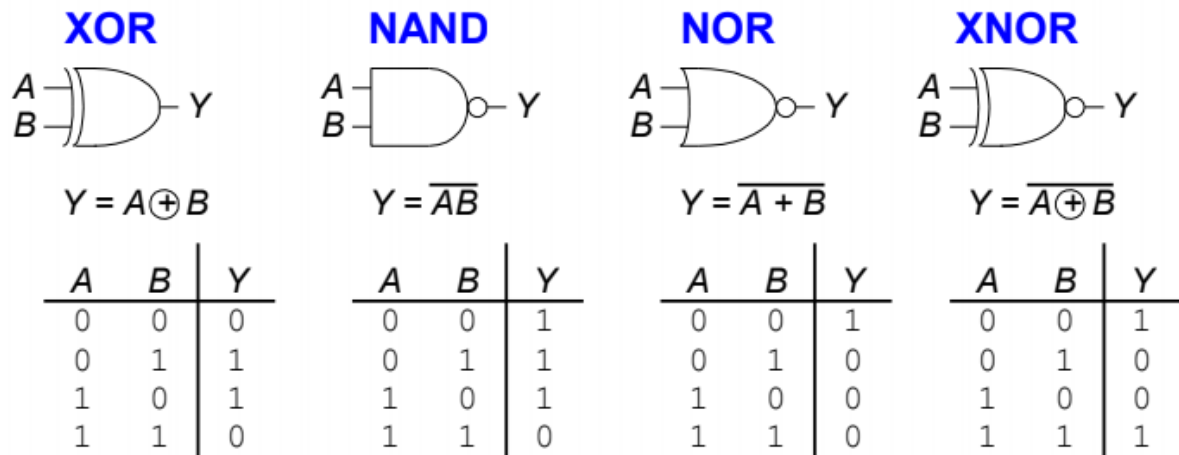


Рис.0.9 - Умовні графічні позначення вентилів XOR, NAND, NOR, XNOR

#### 0.2.4 Компаратор на логічних елементах

У якості прикладу комбінаційної схеми на лог. елементах розглянемо схему компаратора для порівняння двох 4-розрядних двійкових чисел. Схема наведена на рис.0.10.

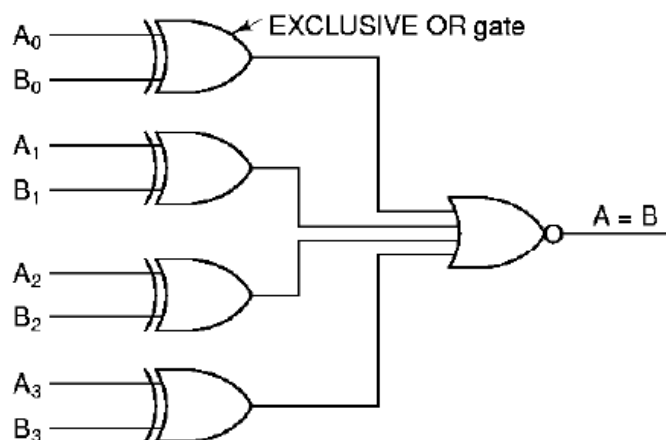


Рис.0.10 - Схема компаратора двох 4-розрядних двійкових чисел

Схема компаратора використовує елемент XOR для порівняння бітів у відповідних розрядах чисел (порівняння  $A_0$  і  $B_0$ ,  $A_1$  і  $B_1$  і т.д.). Відповідно до таблиці



істинності (рис.0.9), на виході XOR буде 0, якщо його входи приймають однакові значення (00, 11). Якщо входи приймають різні значення (01 і 10), на виході XOR буде 1. У випадку коли біти числа А дорівнюють бітам числа В, на виходах всіх елементів XOR будуть нулі і елемент NOR поверне 1. Якщо ж хоча б для одного розряду біти числа А і В відрізнятимуться (наприклад,  $A_0 \neq B_0$ ), відповідний елемент XOR видасть 1 і на виході NOR буде 0.

### 0.2.5 Мікросхеми програмованої логіки типу FPGA

В лабораторних роботах будемо реалізувати цифрові схеми на мікросхемах програмованої логіки типу FPGA (Field-Programmable Gate Array).

FPGA мікросхеми складаються з десятків тисяч (а іноді сотень тисяч, чи мільйонів) логічних комірок, що з'єднані між собою конфігурованою матрицею з'єднань, в якій вихід будь-якої комірки можна з'єднати з входом будь-якої іншої комірки (ця матриця з'єднань реалізована на мультиплексорах). Структура FPGA мікросхеми наведена на рис.0.11.

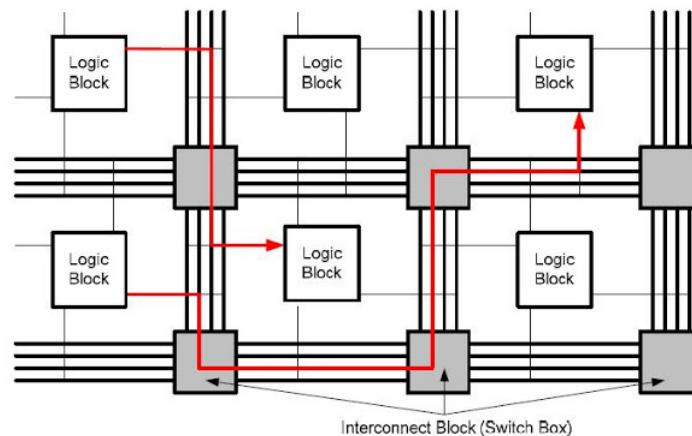


Рис.0.11 - Структура програмованої логіки типу FPGA

Кожна логічна комірка складається з апаратно реалізованого синхронного по фронту D-тригера з входом дозволу запису і таблиці істинності на базі статичної пам'яті, що здатна реалізувати комбінаційну функцію на кілька входів. Вихід таблиці істинності можна підключити на вхід D-тригера. Але можна використовувати таблицку істинності і тригер окремо. Спрощена структура лог. ком. FPGA наведена на рис.0.12.

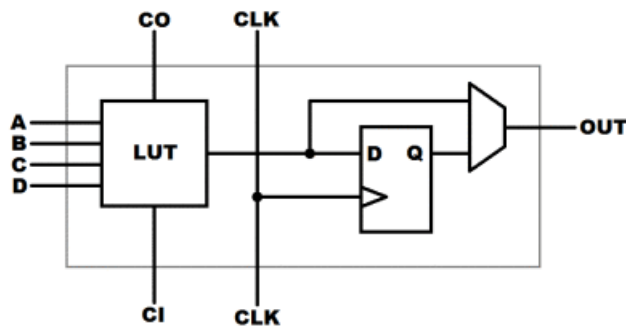


Рис.0.12 - Спрощена структура логічної комірки програмованої логіки типу FPGA

Вмістом таблиць істинності, станом D-тригерів та матриці з'єднань можна керувати за допомогою статичної конфігураційної пам'яті FPGA.

Після вимкнення напруги живлення вміст статичної конфігураційної пам'яті FPGA не зберігається, тож FPGA необхідно кожного разу заново конфігурувати після подачі напруги живлення. Вміст конфігураційної пам'яті зберігається в зовнішній FLASH пам'яті, а конфігуруванням FPGA займається спеціалізована мікросхема конфігуратор (configuration device).

Існує кілька виробників FPGA мікросхем: Intel FPGA (стара назва - Altera), Xilinx, Lattice. В лабораторних роботах ми будемо використовувати мікросхеми компанії Intel FPGA.

Розробка проектів для платформи FPGA виглядає наступним чином. Інженер описує необхідний функціонал цифрової мікросхеми або у вигляді схеми у схемному редакторі (такий підхід використовують переважно початківці під час навчання), або на мові опису апаратури (Verilog, VHDL). Далі система автоматизованого проектування (САПР) створює з цього опису цифрову схему, що складається з ресурсів FPGA (D-тригерів і таблиць істинності). Такий процес називається **синтезом** цифрової схеми. В більшості випадків створена цифрова схема має таку ж саму логіку роботи, як і опис на Verilog/VHDL. САПР визначає які з логічних комірок FPGA будуть використані, який вміст повинен бути у цих комірок і як їх з'єднати між собою. Зрештою генерується конфігураційний файл, який можна записати в конфігураційну пам'ять FPGA.

САПР для FPGA компанії Intel називають Quartus Prime.

## 0.2.6 Опис модуля у мові Verilog

Для побудови цифрових схем в лабораторних роботах будемо використовувати схемний редактор Quartus Prime і мову Verilog. Перші схеми будуть створені у схемному редакторі, а із заглибленням в предметну область будемо все більше переходити на Verilog.

Одразу варто зазначити, що в лабораторних роботах будуть описані лише основи мови Verilog, оскільки це досить складна мова і для її повноцінного вивчення необхідно багато часу. Більш детально мова Verilog описана у відео курсі [\[0.2\]](#), а у вашому навчальному плані існує окремий предмет з вивчення Verilog.

У мові Verilog цифрові схеми групують в модулі (**module**). Оскільки більшість операторів Verilog не можна використовувати поза модулем, кожен проект містить мінімум один модуль.

**Лістинг 0.1** - Приклад опису модуля:

```
module (i_clk, i_data, o_data, o_addr, io_ext_bus);

input i_clk;
input [7:0] i_data;

output [7:0] o_data;
output reg [3:0] o_addr;

inout [15:0] io_ext_bus;

// Тут розміщено вихідний код модуля

/* Коментарі реалізовано як у мовах C і C++ */

endmodule
```

Модулі мають вхідні порти (**input**) та вихідні порти (**output**). Також порти модуля можуть бути двонаправленими (**inout**). Двонаправлені порти використовуються рідше ніж вхідні, чи вихідні порти.

Для кращого сприйняття коду імена вхідних портів пишуть з префіксом **i\_**, імена вихідних портів пишуть з префіксом **o\_**, а імена двонаправлених портів пишуть з префіксом **io\_**.

Входи і виходи модуля можуть бути як однорозрядними так і багато розрядними. Розрядність порта модуля вказують в квадратних скобках після імені модуля. Наприклад: `input [7:0] i_data`; Лівий індекс (в даному випадку 7) позначає старший біт порта. Правий індекс (в даному випадку 0) позначає молодший біт порта.

Входи і виходи по замовчуванню мають тип `wire` але за необхідності тип вихідного порту можна визначити як `reg` додавши ключове слово `reg`. Типи `wire` і `reg` детальніше розглянуто в наступному розділі. Входи і виходи модуля можна використовувати як сигнали відповідних типів (`wire` для входів, `wire` і `reg` для виходів).

За необхідності у модуля можуть бути відсутні входи і виходи. В такому випадку одразу після імені модуля ставиться крапка з комою.

Створення екземплярів модулів розглянемо в наступних лабораторних роботах.

### 0.2.7 Типи `wire` і `reg` у мові Verilog

Два найбільш розповсюджених типи даних у мові Verilog це `wire` і `reg`.

Тип `wire` використовують для створення провідників, що сполучають між собою входи і виходи цифрових схем описаних на Verilog. Провідник типу `wire` не є змінною і не зберігає значення, а лише транслює сигнал від драйвера провідника. Отже у кожного провідника типу `wire` повинен бути драйвер, що визначатиме логічний рівень провідника (якщо драйвер видає 1, провідник приймає значення 1, якщо драйвер видає 0, провідник приймає значення 0). Драйвером провідника може виступати вихідний порт екземпляру модуля або комбінаційна схема створена за допомогою оператора `assign`. Приклади створення провідників типу `wire` показані в лістингу 0.2.

Можна створити як однорозрядний провідник типу `wire`, так і шину провідників. Правила визначення шин такі самі, як і для портів модуля.

**Лістинг 0.2** - Створення провідників типу `wire` у мові Verilog

```
wire a;  
wire [3:0] b;  
wire [6:1] c;  
wire [1:0] d = b[2:1];  
wire e = a & b[1];
```

В лістингу вище створюється однорозрядний провідник **a** типу **wire**. Також створюється 4-розрядна шина провідників **b**. Молодший розряд шини **b** має номер (індекс) 0, а старший розряд шини **b** має індекс 3. Також створюється 6-розрядна шина провідників **c**. Молодший розряд шини **c** має номер (індекс) 1, а старший розряд шини **c** має індекс 6. Краще уникати такої нетипової індексації розрядів шини і обирати 0 у якості номера молодшого розряду (як це зроблено для шини **b**).

За необхідності можна вибрати окремий розряд шини за допомогою оператора квадратних скобок. Наприклад **b[2]** вибирає провідник шини **b** з індексом 2 (другий розряд). Цим розрядом можна оперувати як окремим провідником. За необхідності біт, що вибирається з шини можна визначати не константним числом, а змінною. Наприклад, **b[addr]**. Такий підхід використовують при описі мультиплексорів.

Можна вибрати кілька розрядів за допомогою операції двокрапки. Наприклад, **b[2:1]** вибирає двохрозрядну шину (провідники з номерами 2 та 1 шини **b**).

Драйвер для провідника, або шини провідників можна визначити одразу під час їх створення. Для цього необхідно після імені провідника/шини поставити оператор **=** та визначити вираз, що описує цифрову схему драйвера (це може бути інший провідник чи шина, логічний вираз складений з логічних операторів Verilog, можливе використання арифметичних операцій, функцій Verilog). Таким чином можна описувати найпростішу комбінаційну логіку.

У лістингу 0.2 створюється двохрозрядна шина провідників **d** і у якості драйверу для неї визначаються два розряди шини **b**. Тепер зміна розряду **b[1]** приводить до такої ж зміни розряду **d[0]**. Аналогічно, зміна розряду **b[2]** приводить до такої ж зміни розряду **d[1]**.

Також в лістингу 0.2 створюється однорозрядний провідник **e** і у якості драйверу для нього визначається логічна функція **I** (оператор **&** мови Verilog) над значенням провідника **a** та розряду **b[1]**.

На відміну від типу **wire** тип **reg** має властивості змінної і зберігає записане в нього значення до наступного запису.

Під час моделювання в симуляторі тип **reg** можна використовувати точно так же, як використовують звичайну змінну у мовах програмування (можливість задавати довільну розрядність змінної тут є лише перевагою).

Під час опису цифрових схем змінні типу **reg** можна використовувати для опису як комбінаційної логіки, так і елементів пам'яті (тригерів синхронних по рівню і фронту).

В змінну типу **reg** можна записувати значення лише всередині процедурних блоків (**always** та **initial** блоки).

Приклади створення змінних типу **reg** показані в лістингу 0.3.

**Лістинг 0.3** - Створення провідників типу **reg** у мові Verilog

```
reg a;  
reg [3:0] b;  
reg [1:0] c = 2'b00;
```

Змінні типу **reg** можуть бути як одно розрядними, так і багато розрядними. Правила створення змінних з кількох розрядів такі ж самі, як і для шин провідників типу **wire**. Після створення змінної вона по замовчуванню має невизначене значення **x**. Під час створення змінної можна ініціалізувати її певним значенням, наприклад:

```
reg [1:0] c = 2'b00;
```

Зверніть увагу на визначення константи у мові Verilog: **2'b00**; Спершу йде число, що визначає розрядність константи, далі літера специфікатор, що визначає систему числення константи (d - decimal, b - binary, o - octal, h - hexadecimal). Після літери-специфікатора пишуть значення константи в обраній системі числення.

Під час опису цифрових схем з використанням типу **reg** для реальних мікросхем, що виготовляються з кремнію, початкова ініціалізація змінних типу **reg** ігнорується синтезатором цифрової схеми. Однак у випадку використання мікросхем програмованої логіки на зразок FPGA, початкова ініціалізація змінної типу **reg** дозволяє задати стан після конфігурації FPGA для тригерів чи таблиць істинності у які синтезується змінна типу **reg**. Таким чином можна відмовитися від провідників глобального зкидання, що полегшить трасування з'єднань між компонентами всередині FPGA. Однак варто пам'ятати, що цей трюк працює лише для FPGA і призводить до створення не кросплатформеного коду, який доведеться переписувати при необхідності використання для виробництва спеціалізованих мікросхем типу ASIC.

## 0.2.8 Оператор assign у мові Verilog

Оператор неперервного присвоювання **assign** у мові Verilog використовують для опису комбінаційної логіки і створення драйверів провідників типу **wire**.

Приклади використання оператора **assign** показані в лістингу 0.4.

**Лістинг 0.4** - Використання оператора **assign** у мові Verilog

```
input [3:0] a, b;
output [3:0] y0, y1, y2, y3, y4, y5, y6, sum;

assign y0 = ~a;           // NOT
assign y1 = a & b;        // AND
assign y2 = a | b;        // OR
assign y3 = a ^ b;        // XOR
assign y4 = ~(a & b);     // NAND
assign y5 = ~(a | b);     // NOR
assign y6 = ~(a ^ b);     // XNOR
assign sum = a + b;       // adder
```

Запам'ятайте, що у випадку застосування оператора **assign**, у лівій стороні оператора = повинні бути лише провідники типу **wire**.

Як тільки змінюється хоч один із аргументів виразу з правої сторони оператора =, одразу ж розраховується новий результат виразу і значення логічних рівнів результату виставляються на провідники шини, визначеної зліва від оператора =. Назвемо цю шину зліва від оператора = шиною-приймачем. При цьому молодший біт результату виразу є драйвером молодшого біту шини-приймача, а старший біт результату виразу є драйвером старшого біту шини-приймача.

В лістингу 0.4 використовуються побітові операції логічного NOT (~), AND (&), OR (|), XOR (^). Аргументами побітової операції є дві шини, однакової розрядності. Це можуть бути шини як типу **wire**, так і типу **reg**. Результат побітової операції має розрядність операндів. Побітова операція виконується над бітами кожного розряду обох операндів і формує значення відповідного розряду результату. Наприклад, побітова операція AND над двома 4-розрядними операндами зображена на рис.0.13.

Виключенням з написаного є операція побітової інверсії ~, яка просто інвертує значення бітів єдиного аргументу.

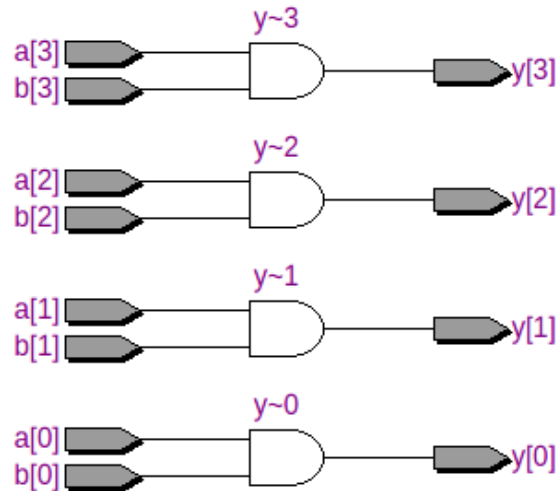


Рис.0.13 - Побітова операція AND над двома 4-розрядними операндами

Побітові операції можна застосовувати не лише до шин, а і до однорозрядних сигналів типу **wire** і **reg**. В такому випадку вони будуть працювати, як звичайні логічні операції.

### 0.3 Практична частина

#### 0.3.1 Завантаження та встановлення Quartus Prime Lite

Для синтезу Verilog файлів і реалізації проектів на платформі Intel FPGA будемо використовувати доступний безкоштовно САПР Quartus Prime Lite. Існують версії програми для ОС Windows і Linux.

Завантажте та встановіть найновішу версію Quartus Prime Lite за посиланням [\[0.3\]](#). Станом на осінь 2018 року найновішою є версія 18.0.

Разом з Quartus Prime Lite, з того ж ресурсу завантажте і встановіть симулятор ModelSim-Intel FPGA Starter Edition. В наступних лабораторних роботах будемо використовувати цю програму для моделювання цифрових схем описаних на Verilog.

Достатньо встановлювати підтримку лише FPGA мікросхем сімейств Cyclone V та MAX.



### 0.3.2 Налагоджувальні плати для виконання лабораторних робіт

Лабораторні роботи з незначними модифікаціями можуть бути виконані на налагоджувальних платах DE0-CV, DE1-SoC, DE10-Lite, DE10-nano. Документацію на налагоджувальні плати і налаштування портів вводу-виводу можна завантажити з репозиторію [0.4].

Вихідні коди прикладів до лабораторних робіт [0.5] виконані для налагоджувальної плати DE10-Lite. Якщо для запуску прикладу на інших налагоджувальних платах приклад необхідно модифікувати, буде надано пояснення, як це зробити.

### 0.3.3 Створення проекту в Quartus Prime Lite

Для створення нового проекту запустіть Quartus Prime Lite та оберіть пункт меню File -> New Project Wizard. Відкриється початкове вікно діалогу створення проекту, під назвою **Introduction**. Пропустіть це вікно натиснувши кнопку Next. У наступному вікні під назвою **Directory, Name, Top-Level Entity** оберіть ім'я проекту (наприклад, Lab0), шлях до директорії проекту та ім'я модуля верхнього рівня ієрархії (рис.0.14).

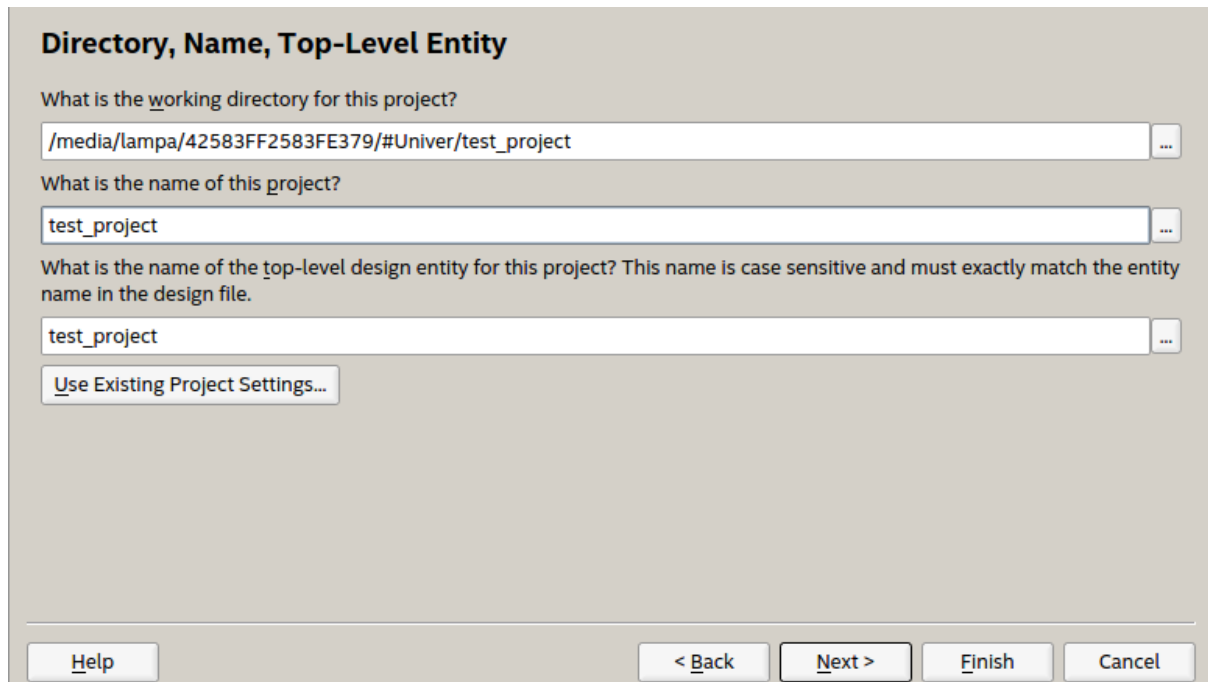


Рис.0.14 - Вікно визначення імені проекту

Простими словами, Top-Level Entity - це ім'я схемного файлу (з розширенням \*.bdf), або ім'я модуля на мові Verilog, в якому описана схема проекту.

У наступному вікні діалогу створення проекту, що називається **Project Type**, оберіть Empty Project і натисніть Next.

У вікні **Add Files** можна додати необхідні файли до проекту, що створюється. Це можуть бути Verilog файли (з розширенням \*.v), файли зі схемами (з розширенням \*.bdf), файли з обмеженнями на синтез, що визначають тактові частоти проекту і затримки розповсюдження сигналів (з розширенням \*.sdc). Вигляд вікна додавання файлів зображено на рис.0.15. В першому проекті у нас поки немає файлів (ми ці файли створимо і додамо до проекту пізніше), тому натискаємо кнопку Next.

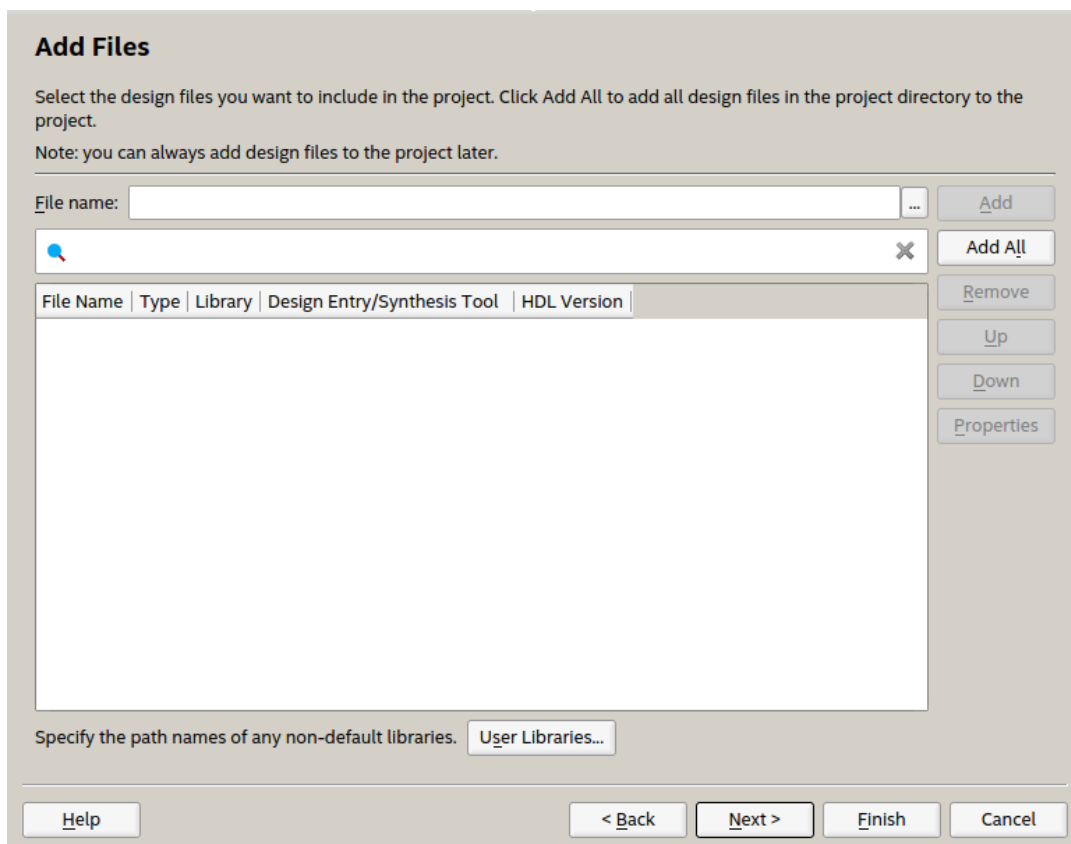


Рис.0.15 - Вікно додавання файлів до проекту

У наступному вікні, що називається **Family, Device & Board Settings**, на вкладці Device можна обрати FPGA мікросхему для якої створюється проект (рис.0.16). Для спрощення пошуку в полі Name Filter можна ввести ім'я обраної FPGA мікросхеми. Альтернативно, замість вибору FPGA мікросхеми, на вкладці Board можна обрати налагоджувальну плату для якої створюється проект (рис.0.17).

### Family, Device & Board Settings

Device

Board

Select the family and device you want to target for compilation.  
You can install additional device support with the Install Devices command on the Tools menu.

To determine the version of the Quartus Prime software in which your target device is supported, refer to the [Device Support List](#) webpage.

**Device family**  
Family: MAX 10 (DA/DF/DC/SA/SC)  
Device: All

**Show in 'Available devices' list**  
Package: Any  
Pin count: Any  
Core speed grade: Any  
Name filter:  
☒ Show advanced devices

**Target device**  
☐ Auto device selected by the Fitter  
☒ Specific device selected in 'Available devices' list  
☐ Other: n/a

Available devices:

Name	Core Voltage	LEs	Total I/Os	GPIOs	Memory Bits	Embedded multiplier 9-bit e
10M50DAF256I7G	1.2V	49760	178	178	1677312	288
10M50DAF484C6GES	1.2V	49760	360	360	1677312	288
10M50DAF484C7G	1.2V	49760	360	360	1677312	288
10M50DAF484C8G	1.2V	49760	360	360	1677312	288
10M50DAF484C8GES	1.2V	49760	360	360	1677312	288
10M50DAF484I7G	1.2V	49760	360	360	1677312	288

Help

< Back

Next >

Finish

Cancel

Рис.0.16 - Вікно вибору FPGA мікросхеми для якої створюється проект

### Family, Device & Board Settings

Device

Board

Select the board/development kit you want to target for compilation.

Family: MAX 10
Development Kit: Any

Available boards:

	Name	Version	Family	Device
	Arrow MAX 10 DECA	0.9	MAX 10	10M50DAF484C
	BeMicro MAX 10 FPGA Evaluation Kit	1.0	MAX 10	10M08DAF484C
	MAX 10 DE10 - Lite	1.0	MAX 10	10M50DAF484C
	MAX 10 FPGA 10M08 Evaluation Kit	1.0	MAX 10	10M08SAE144C
	MAX 10 FPGA Development Kit	1.0	MAX 10	10M50DAF256C
	MAX 10 NEEK	1.0	MAX 10	10M50DAF484I7
	Odyssey MAX 10 FPGA Kit	1.0	MAX 10	10M08SAU169C

☒ Create top-level design file.

Can't find your board? Check the [Design Store](#) for additions and search for baseline under Design Examples.

Help

< Back

Next >

Finish

Cancel

Рис.0.17 - Вікно вибору налагоджувальної плати для якої створюється проект  
(альтернатива вибору вибору FPGA мікросхеми)

Для кожної налагоджувальної плати з якою ви будете працювати найменування FPGA мікросхеми, що встановлена на платі, можна переглянути в описі плати (User Manual) за посиланням [0.4], або прочитати на корпусі мікросхеми безпосередньо на платі. Наприклад, налагоджувальна плата DE10-Lite використовує FPGA мікросхему 10M50DAF484C7G серії MAX10.

У наступному вікні діалогу створення нового проекту, під назвою **EDA Tool Settings**, залишаємо налаштування за замовчуванням і натискаємо Next.

У фінальному вікні **Summary** натискаємо Finish.

Вітаю, ми створили проект! Для внесення змін до налаштувань проекту необхідно обрати пункт меню Assignments -> Settings. Відкриється вікно налаштувань (рис.0.18). Зокрема на вкладці General вікна налаштувань можна обрати нове ім'я модуля верхнього рівня ієрархії (Top-Level Entity). А на вкладці Files можна додати нові файли до проекту. Для зміни FPGA мікросхеми в проекті необхідно обрати пункт меню Assignments -> Device.

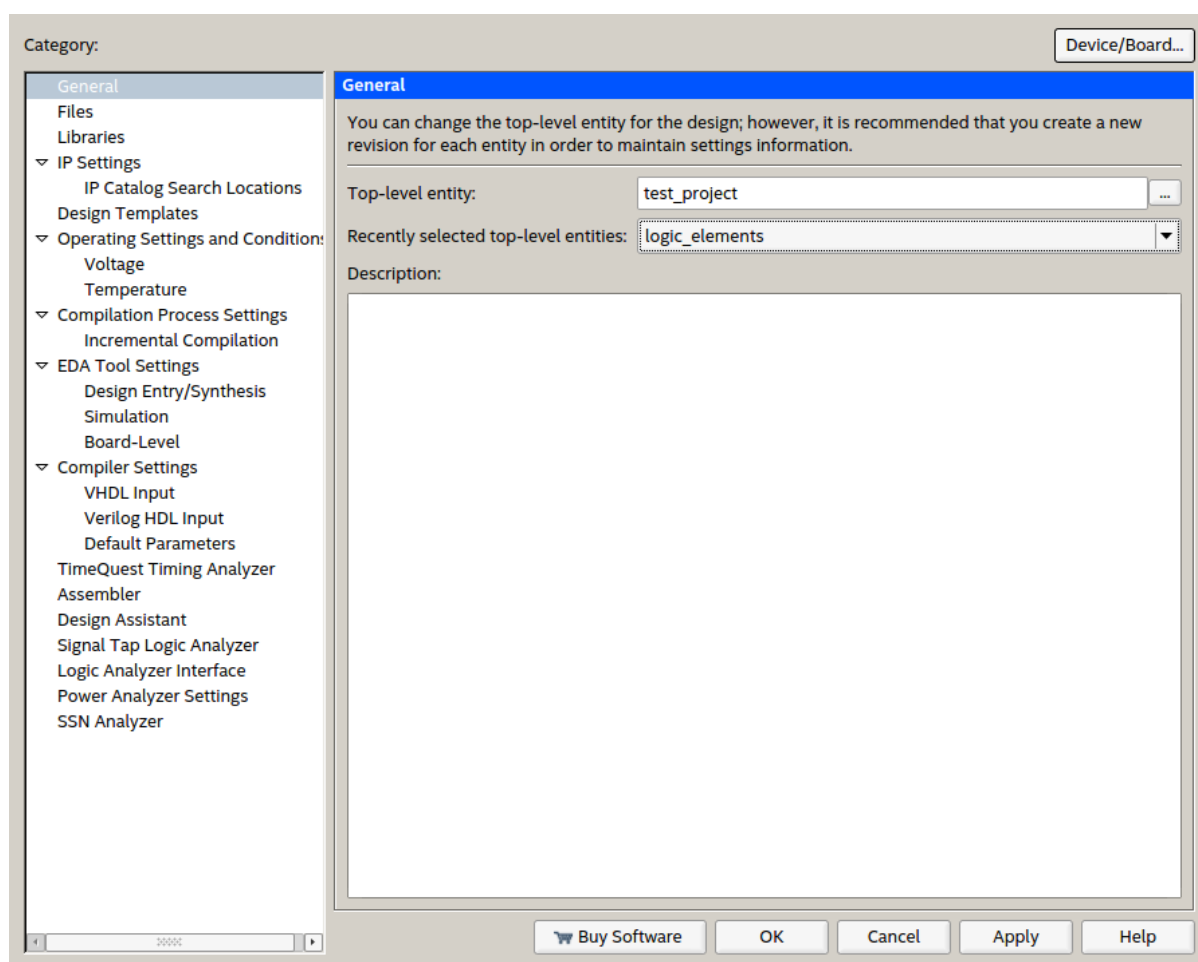


Рис.0.18 - Вікно налаштувань проекту

### 0.3.4 Імпорт налаштувань портів вводу-виводу в Quartus Prime Lite

В проектах для мікросхем FPGA необхідно задати налаштування портів вводу-виводу. Наприклад, зконфігурувати на вихід порти (контакти) FPGA мікросхеми, що підключені до світлодіодів (рис.0.19). А порти підключені до генератора тактової частоти (рис.0.20) і кнопок зконфігурувати на вхід.

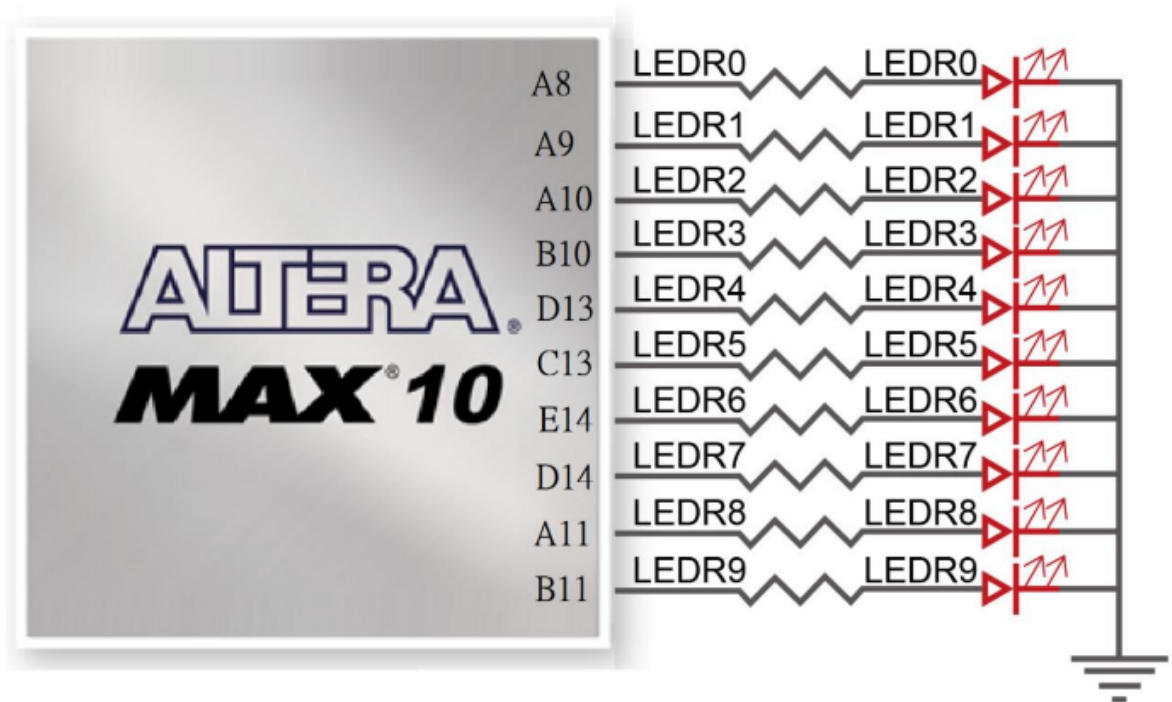


Рис.0.19 - Приклад підключення портів FPGA мікросхеми, що керують світлодіодами і які необхідно зконфігурувати на вихід

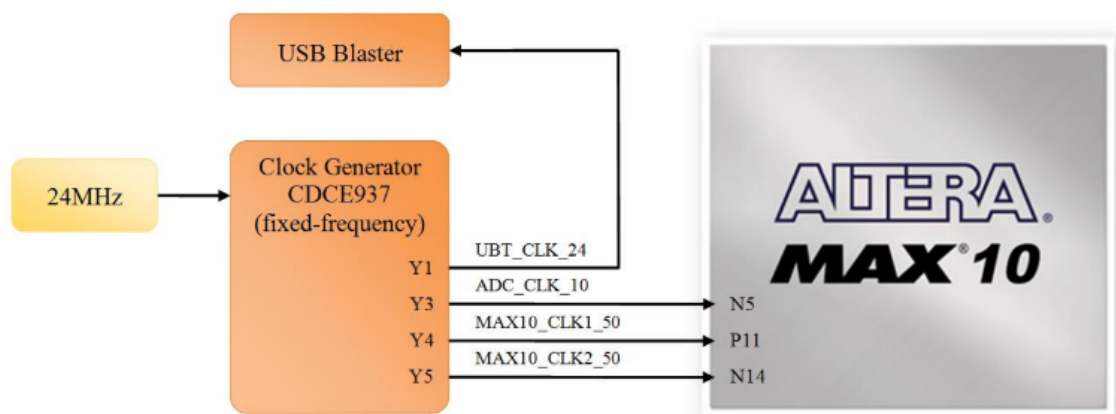


Рис.0.20 - Приклад підключення портів FPGA мікросхеми на які подаються імпульси тактової частоти і які необхідно зконфігурувати на вхід

Для налаштування портів вводу-виводу FPGA мікросхеми необхідно обрати пункт меню Assignments -> Pin Planner. Одразу після створення проекту жодних налаштувань портів вводу-виводу не буде. Ці налаштування необхідно додати вручну.

Можливі два підходи. **Підхід 1:** використовувати довільні імена входів і виходів модуля у Verilog файлі, чи у схемному редакторі, а потім в налаштуваннях поставити у відповідність кожному входу і виходу певний контакт мікросхеми FPGA і тип логічних рівнів. Такий підхід універсальніший, але потребує більше додаткової роботи. **Підхід 2:** імпортувати налаштування контактів для обраної FPGA з файлу. Файли з налаштуваннями портів вводу-виводу для наявних FPGA плат мають розширення \*.csv і доступні за посиланням [\[0.4\]](#) всередині кожного каталогу з іменем налагоджувальної плати. Всередині таких файлів кожному контакту мікросхеми поставлене у відповідність певне ім'я, напрямок (вхід/вихід) та тип логічних рівнів. Наприклад, контакти FPGA мікросхеми, що підключені до світлодіодів мають назву LED[0], LED[1] і т.д., зконфігуровані на вихід і мають тип логічних рівнів 3.3V - LVTTTL. Після імпорту налаштувань з файлу необхідно використовувати в проекті задані в налаштуваннях імена портів вводу-виводу.

Отже імпортуємо із файлу налаштування портів вводу-виводу FPGA мікросхеми для налагоджувальної плати DE10-Lite. Для цього оберемо пункт меню Assignments -> Import Assignments. Відкриється вікно імпорту налаштувань з файлу (рис.0.21). Обираємо шлях до файлу DE10\_LITE\_Default.csv і натискаємо ОК. Налаштування імпортовано.

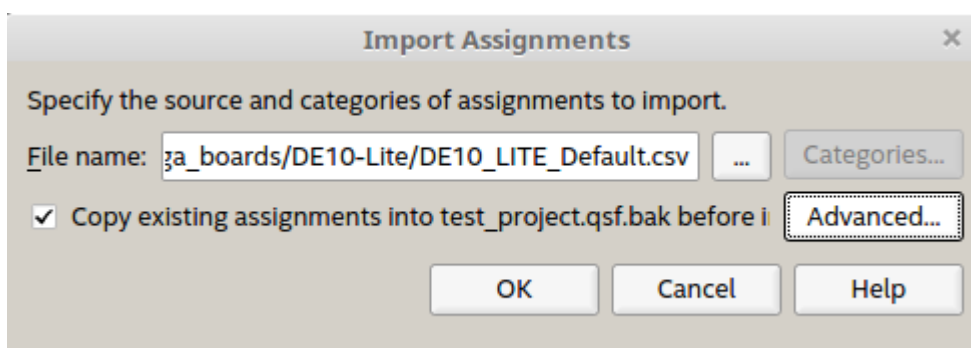


Рис.0.21 - Вікно імпорту налаштувань з файлу

Для перегляду налаштувань імпортованих з файлу обираємо пункт меню Assignments -> Pin Planner. Бачимо таблицю з іменами портів вводу-виводу, де кожному

порту поставлений у відповідність напрямок передачі (вхід/вихід), контакт FPGA мікросхеми і тип логічних рівнів (рис.0.22).

Стовбець Node Name містить ім'я порту вводу-виводу, яке використовується у якості імені вхідного або вихідного порту Verilog модуля верхнього рівня ієрархії (Top-Level Entity), або у якості імені вхідного чи вихідного порту у схемному редакторі.

Стовбець Location містить ім'я контакту FPGA мікросхеми, який поставлений у відповідність порту з певним іменем.

Стовбець Direction дозволяє обирати напрямок порту. Після завершення компіляції проекту комірки цього стовбчика встановлюються у значення Input, Output, Inout (bidir), відповідно до обраного напрямку портів вводу-виводу у Verilog файлі, чи у схемному редакторі.

Стовбець I/O Standard дозволяє обирати тип логічних рівнів певного контакту FPGA мікросхеми (LVTTL, LVCMOS і т.д.).

Аналогічні налаштування ви можете створити і самостійно (за необхідності).

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength
KEY[0]	Unknown	PIN_B8	7	B7_NO		3.3 V S...Trigger		8mA (default)
KEY[1]	Unknown	PIN_A7	7	B7_NO		3.3 V S...Trigger		8mA (default)
LEDR[0]	Unknown	PIN_A8	7	B7_NO		3.3-V LVTTL		8mA (default)
LEDR[1]	Unknown	PIN_A9	7	B7_NO		3.3-V LVTTL		8mA (default)
LEDR[2]	Unknown	PIN_A10	7	B7_NO		3.3-V LVTTL		8mA (default)
LEDR[3]	Unknown	PIN_B10	7	B7_NO		3.3-V LVTTL		8mA (default)
LEDR[4]	Unknown	PIN_D13	7	B7_NO		3.3-V LVTTL		8mA (default)
LEDR[5]	Unknown	PIN_C13	7	B7_NO		3.3-V LVTTL		8mA (default)
LEDR[6]	Unknown	PIN_E14	7	B7_NO		3.3-V LVTTL		8mA (default)
LEDR[7]	Unknown	PIN_D14	7	B7_NO		3.3-V LVTTL		8mA (default)
LEDR[8]	Unknown	PIN_A11	7	B7_NO		3.3-V LVTTL		8mA (default)
LEDR[9]	Unknown	PIN_B11	7	B7_NO		3.3-V LVTTL		8mA (default)
SW[0]	Unknown	PIN_C10	7	B7_NO		3.3-V LVTTL		8mA (default)
SW[1]	Unknown	PIN_C11	7	B7_NO		3.3-V LVTTL		8mA (default)
SW[2]	Unknown	PIN_D12	7	B7_NO		3.3-V LVTTL		8mA (default)
SW[3]	Unknown	PIN_C12	7	B7_NO		3.3-V LVTTL		8mA (default)
SW[4]	Unknown	PIN_A12	7	B7_NO		3.3-V LVTTL		8mA (default)

Рис.0.22 - Імпортовані налаштування портів вводу-виводу FPGA мікросхеми

### 0.3.5 Створення схем у схемному редакторі

Для створення цифрової схеми у схемному редакторі Quartus Prime виконайте пункт меню File -> New. У вікні створення нового документу, що відкриється, оберіть Block Diagram/Schematic File (рис.0.23). З'явиться вікно для створення схем. Схемний редактор за потреби можна налаштувати обравши пункт меню Tools -> Options (розділ Block/Symbol Editor).

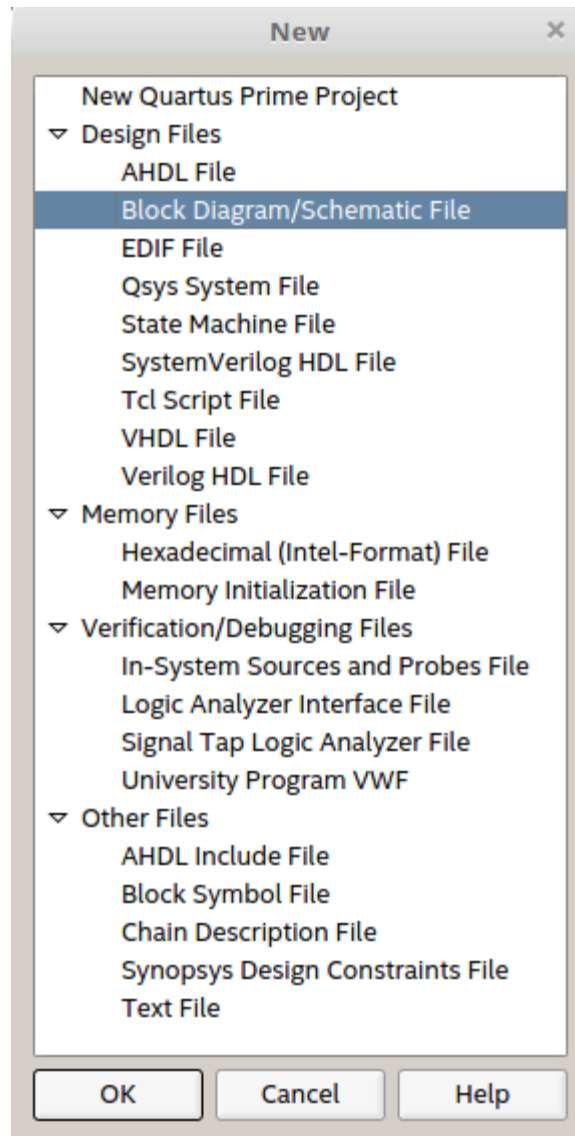



Рис.0.23 - Вікно створення нового документу

Для додавання на схему нового логічного елементу, чи порту вводу-виводу натисніть на кнопку , або просто двічі клікніть лівою клавішею миші по вільному місцю схемного редактора. Вигляд вікна додавання нових елементів на схему зображений на рис.0.24. Можна або вручну обрати необхідний компонент із деревовидного меню, або скористатися пошуком ввівши назву елементу в поле Name (наприклад, Input для вхідного порту, nand2 для елементу І-НІ з двома входами і т.д.).

Додайте на схему елементи НІ (not), І (and2), І-НІ (nand2), АБО (or2), АБО-НІ (nor2), ВИКЛЮЧАЮЧЕ-АБО (xor), вхідний порт (input), вхідний порт (output), лог. 0 (gnd), лог. 1 (vcc). З'єднайте між собою елементи як показано на рис.0.25.



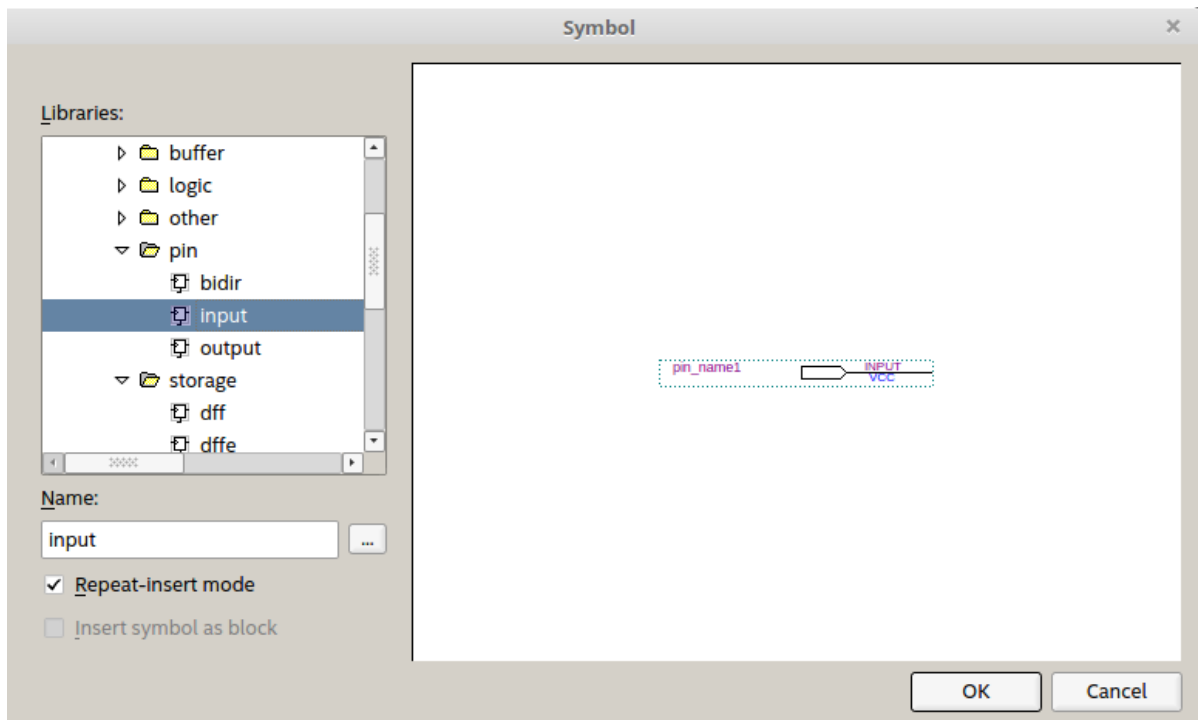


Рис.0.24 - Вікно додавання нових елементів на схему

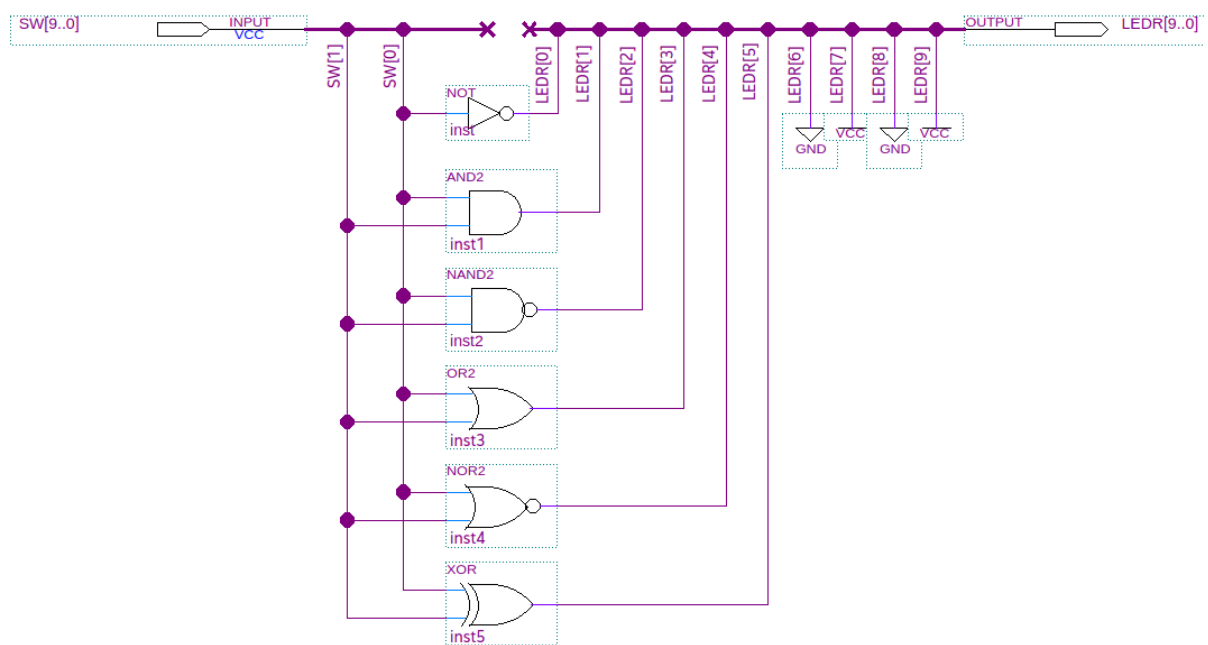




Рис.0.25 - Схема для дослідження базових логічних елементів

Переконайтеся, щоб у всіх елементів на схемі були різні (унікальні) імена (inst, inst1, inst2 і т.д.). Ім'я елементу визначається в його властивостях (поле Instance Name).

Для з'єднання між собою елементів на схемі використовують провідники і шини. Провідники представляють собою 1 розряд і позначаються на схемі тонкою лінією. Шина складається з кількох розрядів і позначається на схемі товстою лінією. Для малювання провідника натискаємо кнопку . Для малювання шини натискаємо кнопку .

До однорозрядного порту (вхідного або вихідного) необхідно підключати провідник. До багаторозрядного порта (вхідного або вихідного) необхідно підключати шину такої ж розрядності, як і розрядність порта. При підключенні шини до порта по замовчуванню шина матиме таке ж ім'я, як і ім'я порта. За бажанням ви можете змінити ім'я шини у властивостях шини, в полі Name.

Для того, щоб визначити ім'я вхідного або вихідного порта необхідно зайти у властивості цього порта і у полі Pin Name вказати ім'я порта. Якщо присутні налаштування портів FPGA мікросхеми, обране ім'я порта повинно бути у списку визначених імен портів (рис.0.22). Якщо налаштувань портів поки немає і ви створили порт в схемному редакторі та задали йому певне ім'я, необхідно вручну додати налаштування для цього порта, як описано в розділі 0.3.4.

В даному прикладі у якості входів будемо використовувати контакти FPGA до яких підключені перемикачі SW, що наявні на платі DE10-Lite (рис.0.26). У якості виходів будемо використовувати контакти FPGA до яких підключені червоні світлодіоди LEDR (рис.0.19).



Рис.0.26 - Перемикачі SW на платі DE10-Lite

Оскільки маємо 10 перемикачів SW і 10 світлодіодів LEDR, порти SW та LEDR мають 10 розрядів. Розрядність порту вказується в квадратних скобках після імені порту в форматі [MSB..LSB], де MSB номер старшого розряду, LSB номер молодшого розряду. Наприклад, ім'я 10-розрядного вхідного порту в нашому прикладі буде SW[9..0]. А для 10-розрядного вихідного порту використовуємо ім'я LEDR[9..0]

Зверніть увагу, що використані багаторозрядні порти SW[9..0] та LEDR[9..0] є у списку налаштувань контактів FPGA мікросхеми (розділ 0.3.4, рис.0.22) і кожному розряду для кожного порту поставлений у відповідність певний контакт FPGA мікросхеми з заданими налаштуваннями. Для того, щоб виділити певний розряд вхідного чи вихідного порту використовуємо індексацію за допомогою квадратних скобок, наприклад SW[0], або LEDR[1].

Отже для плати DE10-Lite в даному прикладі використовуємо порти SW[9..0] та LEDR[9..0]. Для інших плат імена та розрядності портів можуть дещо відрізнятися. Однак це не критично для роботи прикладів. Наприклад, на платі DE10-nano порт SW має 4 розряди, а не 10 розрядів, а порт для підключення світлодіодів називається не LEDR, а LED і має 8, а не 10 розрядів. Тобто для інших налагоджувальних плат необхідно внести мінімальні зміни в імена портів на схемі.


Імена портів для кожної налагоджувальної плати можна переглянути у вікні Assignments -> Pin Planner після імпорту налаштувань контактів FPGA мікросхеми із файлу для обраної плати (як зробити імпорт вказано в розділі 0.3.4). Також імена портів з файлу налаштувань можна подивитися в описі плати (User Manual) [\[0.4\]](#).

Для того, щоб підключити провідник до певного розряду шини, необхідно з'єднати провідник з шиною і у властивостях провідника задати ім'я розряду шини до якого відбувається підключення (SW[0], SW[1], LEDR[0], тощо). Ім'я провідника визначається в полі Name властивостей провідника. На рис.0.25 показано підключення провідників до розрядів шин SW та LEDR.

Після створення схеми, збережіть її з іменем logic\_elements.bdf (для збереження файл оберіть пункт головного меню File -> Save).

Необхідно обов'язково задати значення Top Level Entity рівне імені схемного файлу, який ми будемо компілювати (в нашому випадку це logic\_elements). Раніше ми вже писали, що параметр Top Level Entity визначається на вкладці General налаштувань проекту (рис.0.18).

### 0.3.6 Компіляція проекту в Quartus Prime Lite

Для компіляції схеми і створення конфігураційного файлу для FPGA натисніть кнопку , або оберіть пункт головного меню Processing -> Start Compilation.

Після завершення компіляції з'явиться вікно Compilation Report (рис.0.27), в якому ви можете переглянути скільки ресурсів мікросхеми (таблиць істинності logic elements, D-тригерів registers, тощо) зайняв проект. Або яка у проекту максимальна тактова частота роботи (вкладка TimeQuest Timing Analyzer для синхронних схем).

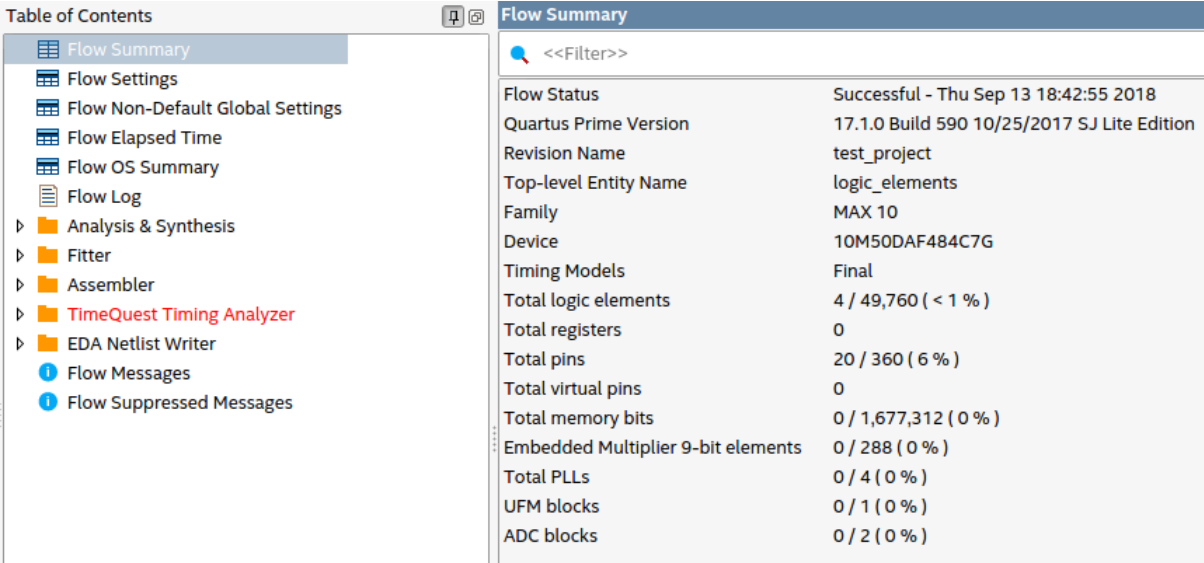


Table of Contents	
Flow Summary	
Flow Settings	
Flow Non-Default Global Settings	
Flow Elapsed Time	
Flow OS Summary	
Flow Log	
Analysis & Synthesis	
Fitter	
Assembler	
TimeQuest Timing Analyzer	
EDA Netlist Writer	
Flow Messages	
Flow Suppressed Messages	

Flow Summary	
<<Filter>>	
Flow Status	Successful - Thu Sep 13 18:42:55 2018
Quartus Prime Version	17.1.0 Build 590 10/25/2017 SJ Lite Edition
Revision Name	test_project
Top-level Entity Name	logic_elements
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	4 / 49,760 ( < 1 % )
Total registers	0
Total pins	20 / 360 ( 6 % )
Total virtual pins	0
Total memory bits	0 / 1,677,312 ( 0 % )
Embedded Multiplier 9-bit elements	0 / 288 ( 0 % )
Total PLLs	0 / 4 ( 0 % )
UFM blocks	0 / 1 ( 0 % )
ADC blocks	0 / 2 ( 0 % )

Рис.0.27 - Вікно з результатами компіляції проекту

### 0.3.7 Конфігурування FPGA за результатами компіляції в Quartus Prime Lite

Для запису в FPGA мікросхему конфігураційного файлу, який було створено на етапі компіляції проекту, використовують програматор USB-Blaster.

Переконайтеся, що у вас встановлені драйвери на USB-Blaster. Інструкцію по встановленню драйверу на ОС Windows можна переглянути за посиланням [\[0.6\]](#). Інструкцію по встановленню драйверу на ОС Linux можна переглянути за посиланням [\[0.7\]](#).

Запустіть програматор виконавши пункт меню Tools -> Programmer. Відкриється вікно програматора (рис.0.28). Переконайтеся, що кнопка Start активна, а біля кнопки Hardware Setup є напис USB-Blaster [USB-0].

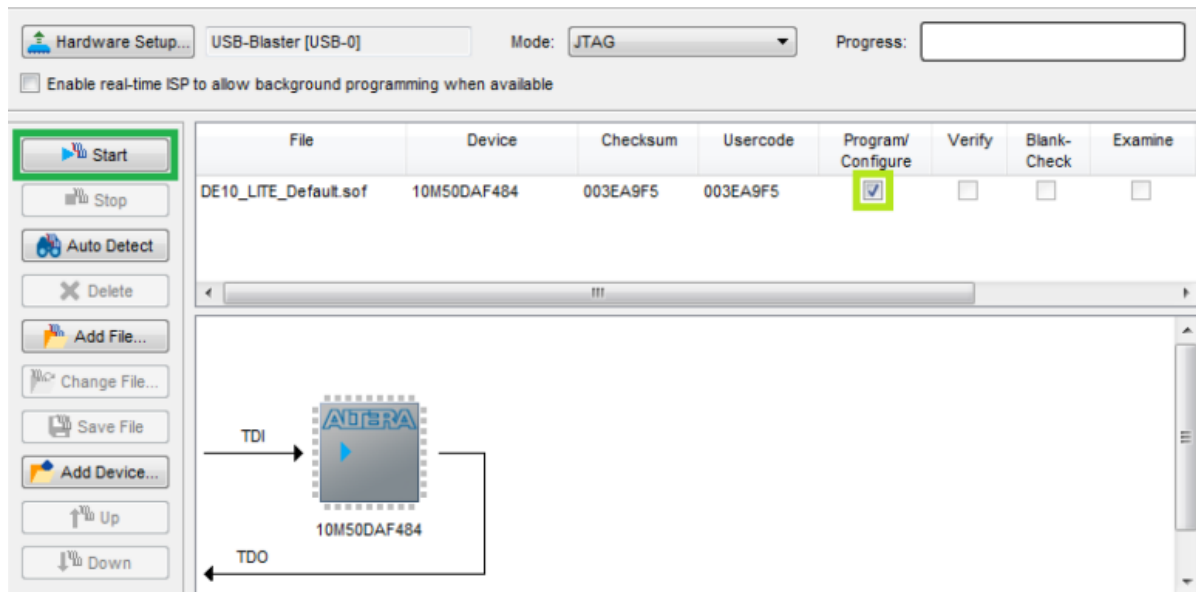


Рис.0.28 - Вікно програматора USB-Blaster

Якщо ж біля кнопки Hardware Setup написано No Hardware, натисніть кнопку Hardware Setup. Відкриється вікно вибору програматора (рис.0.29). У вікні вибору програматора оберіть USB-Blaster і натисніть Close. Якщо ж у вікні вибору програматора відсутній USB-Blaster, це означає, що у вас не встановлені драйвери. Встановіть драйвери, перезапустіть Quartus Prime і знову відкрийте вікно програматора.

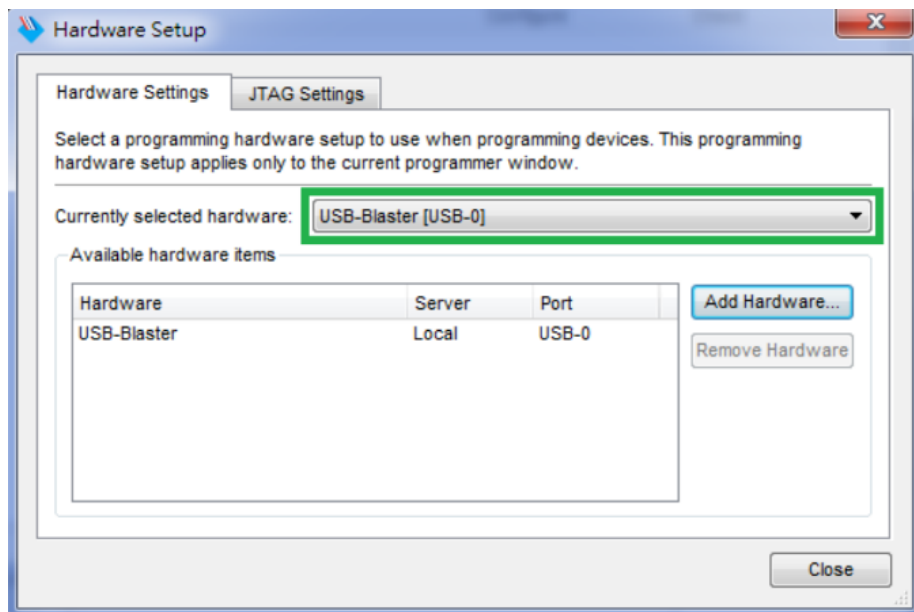


Рис.0.29 - Вікно вибору програматора

Для завантаження в FPGA конфігураційного файлу натисніть кнопку Start.

Зверніть увагу, що \*.sof файл з конфігурацією FPGA з'явиться в стовбці File лише після компіляції проекту. За бажанням можна видалити цей файл і додати інший \*.sof файл натиснувши кнопку Add File (зазвичай \*.sof файл зберігається в папці output\_files). Після додавання нового файлу, необхідно поставити галочку Program/Configure і можна знову конфігурувати FPGA.

Таким чином можна сконфігурувати FPGA лише однократно, до вимкнення напруги живлення. Після повторної подачі напруги живлення конфігурація FPGA буде завантажена з FLASH пам'яті. До речі, в усіх сімействах Intel FPGA, окрім MAX10, конфігураційна FLASH пам'ять зовнішня. В MAX10 конфігураційна FLASH пам'ять знаходиться всередині мікросхеми.

За необхідності можна записати дані в конфігураційну FLASH пам'ять, щоб після подачі напруги живлення, мікросхема FPGA автоматично конфігурувалася новим файлом. Однак цього робити не бажано, оскільки в конфігураційній пам'яті за замовчуванням зберігається стандартна прошивка, що тестує периферію налагоджувальної плати, тож одразу можна помітити компонент, який вийшов з ладу (наприклад, не працює VGA, або не світиться світлодіод). Якщо все ж є необхідність записати новий конфігураційний файл в конфігураційну FLASH пам'ять, видаліть старі \*.sof файл у вікні програматора і додайте туди \*.rof файл, що зберігається в папці output\_files. Після натиснення кнопки Start конфігураційний файл буде записано в конфігураційну FLASH пам'ять.

### **0.3.8 Опис проекту з використанням мови Verilog**

В одному з попередніх розділів ми створили цифрову схему у схемному редакторі Quartus Prime.

У цьому розділі ми опишемо таку ж саму цифрову схему, але на мові Verilog.

Для опису цифрової схеми на мові Verilog бажано створити новий проект в Quartus Prime Lite. Однак можна повторно використати проект раніше створений у розділі 0.3.3. У випадку застосування раніше створеного проекту, необхідно видалити з файлів проекту старі файли створені у схемному редакторі. Це можна зробити на вкладці Files налаштувань проекту. Вікно налаштувань проекту можна відкрити обравши пункт меню Assignments -> Settings.

За необхідності ви можете створити Verilog файл у текстовому редакторі і додати його до файлів проекту на вкладці Files налаштувань проекту. На цій же вкладці ви можете додати до файлів проекту інші існуючі Verilog файли.

Інший шлях - створити Verilog файл в Quartus Prime. Для цього оберіть пункт головного меню File -> New і у вікні, що з'явиться (рис.0.23) оберіть Verilog HDL File.

Вставте у створений файл опис цифрової схеми з рис.0.25 на мові Verilog (лістинг 0.5)

**Лістинг 0.5** - Опис цифрової схеми з рис.0.25 на мові Verilog

```
module logic_elements(SW, LEDR);

    input  [9:0]  SW;
    output [9:0]  LEDR;

    wire    [9:0]  y;
    wire    [9:0]  x;


    assign LEDR = y;
    assign x = SW;

    assign y[0] = ~x[0];
    assign y[1] = x[0] & x[1];
    assign y[2] = ~(x[0] & x[1]);
    assign y[3] = x[0] | x[1];
    assign y[4] = ~(x[0] | x[1]);
    assign y[5] = x[0] ^ x[1];
    assign y[9:6] = 4'b1010;

endmodule
```

Збережіть створений файл під іменем logic\_elements.v. Файли створені в Quartus Prime автоматично додаються до файлів проекту.

Переконайтеся, що ім'я модуля верхнього рівня ієрархії (Тобто модуля, що містить всі інші модулі і логічні елементи. В нашому випадку це модуль logic\_elements) дорівнює імені виставленому в полі Top-Level Entity (рис.0.18).

Для компіляції схеми і створення конфігураційного файлу для FPGA натисніть кнопку  , або оберіть пункт головного меню Processing -> Start Compilation.

Завантажте створений конфігураційний файл в FPGA та перевірте правильність роботи схеми.

Зверніть увагу, що Verilog код наведений в лістингу 0.5 призначений для налагоджувальної плати DE10-Lite. Для використання цього Verilog коду на інших налагоджувальних платах необхідно внести незначні модифікації в імена портів вводу-виводу (більш детальна інформація по іменуванню портів вводу виводу на інших налагоджувальних платах наведена в кінці розділу 0.3.5).

### 0.3.9 Перегляд результатів компіляції в Netlist Viewers

За необхідності ви можете переглянути цифрову схему створену в результаті операції синтезу вашого Verilog коду.

Для перегляду цифрової схеми на базі типових логічних елементів виконайте пункт меню Tools -> Netlist Viewers -> RTL Viewer. Наприклад, для Verilog коду з лістингу 0.5 в RTL Viewer буде відображена схема на рис.0.30.

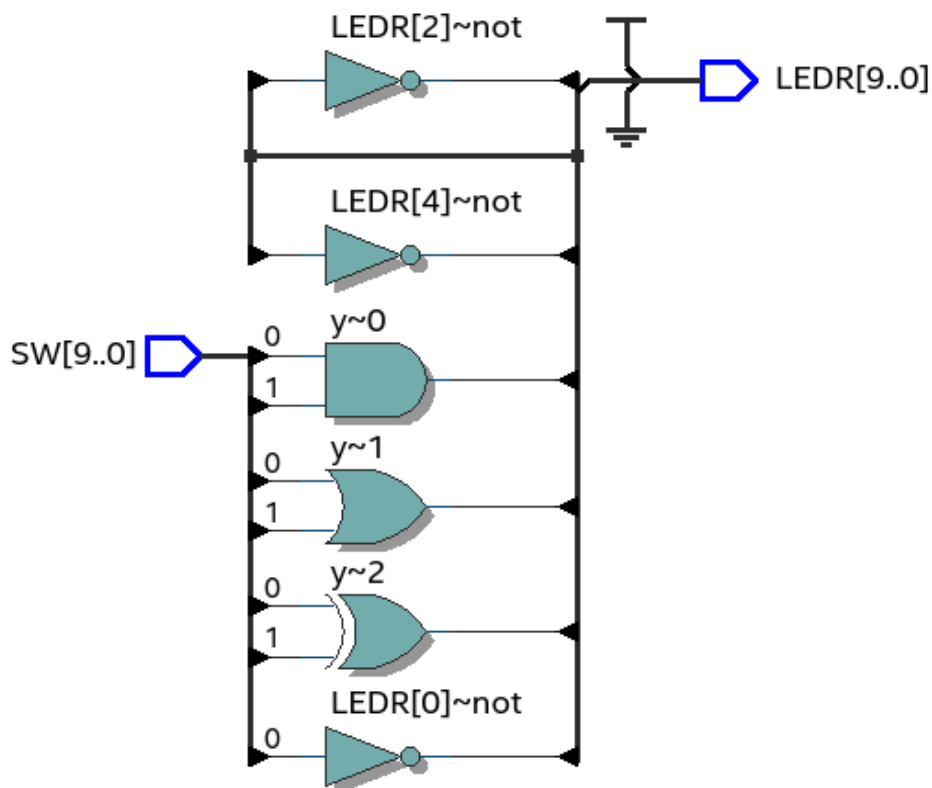


Рис.0.30 - Результат синтезу Verilog коду з лістингу 0.5 в RTL Viewer

Після отримання схеми на базі стандартних логічних елементів з Verilog коду внаслідок операції синтезу, виконується відображення отриманої схеми на ресурси



FPGA (таблиці істинності та D-тригери). Далі відбувається планування з'єднань між обраними ресурсами FPGA за допомогою комунікаційної матриці. Згадані дві операції називаються Place & Route (або Mapping) і виконуються модулем, що називається Fitter.

Для того, щоб переглянути схему отриману після Place & Route виконайте пункт меню Tools -> Netlist Viewers -> Technology Map Viewer (Post Fitting). Наприклад, для Verilog коду з лістингу 0.5 в Technology Map Viewer (Post Fitting) буде відображена схема на рис.0.31.

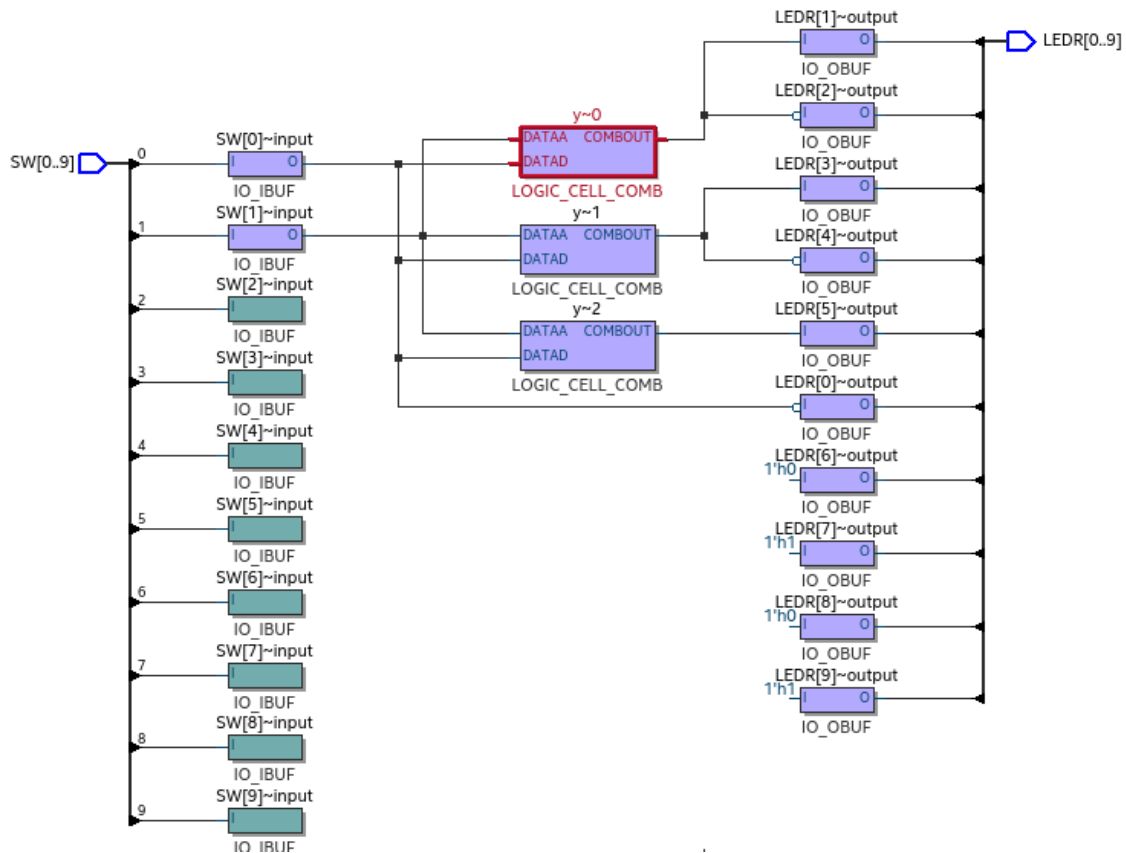


Рис.0.31 - Результат синтезу Verilog коду з лістингу 0.5 в Technology Map Viewer (Post Fitting)

Можна зайти у властивості кожної таблички істинності (LOGIC\_CELL\_COMB) і переглянути логічну функцію та значення таблички істинності.

Зверніть увагу, що на рис. 0.31 інверсія відбувається в комірках вихідних портів FPGA мікросхеми.

Можна переглянути розміщення використаних ресурсів (таблиць істинності та D-тригерів) на кристалі FPGA виконавши пункт меню Tools -> Chip Planner (рис.0.32).

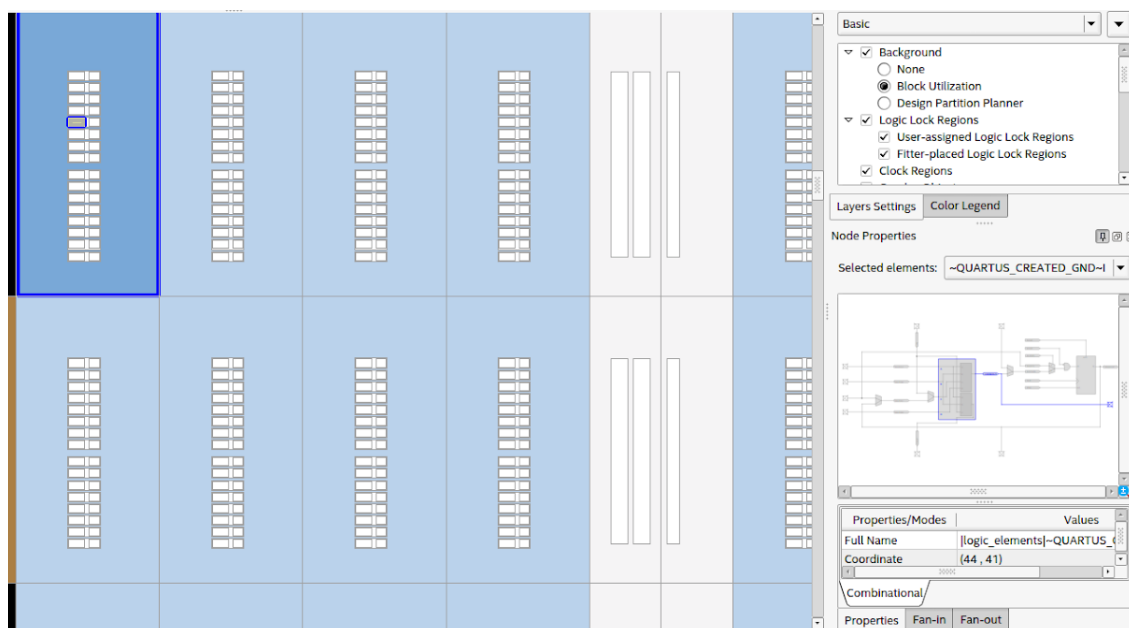


Рис.0.32 - Вигляд кристалу з використаними ресурсами FPGA в Chip Planner

### 0.3.10 Завдання на лабораторну роботу

1. Завантажте і встановіть Quartus Prime Lite;
2. Встановіть драйвера для програматора USB-Blaster;
3. Створіть проект в Quartus Prime Lite для обраної налагоджувальної плати;
4. Імпортуйте налаштування портів вводу-виводу для FPGA мікросхеми обраної налагоджувальної плати;
5. Створіть у схемному редакторі Quartus Prime Lite схему з рис.0.25. Синтезуйте створену схему. Запишіть створений файл конфігурації в FPGA та переконайтеся в коректній роботі створеної схеми;
6. Створіть Verilog файл, що містить Verilog код з лістингу 0.5. Синтезуйте створений опис схеми на мові Verilog. Запишіть створений файл конфігурації в FPGA та переконайтеся в коректній роботі створеної схеми;
7. Створіть схему компаратора двох 4-розрядних чисел у схемному редакторі Quartus Prime Lite (рис.0.10). Один аргумент компаратора представлений розрядами [3:0] перемикачів SW. Інший аргумент компаратора представлений розрядами [7:4] перемикачів SW. Результат компаратора виведіть на світлодіод LEDR[0]. У випадку рівності чисел на вході компаратора світлодіод повинен

засвічуватись. Якщо числа не рівні світлодіод не повинен світитися. Для налагоджувальних плат з меншою кількістю розрядів перемикачів SW можна використовувати меншу розрядність вхідних чисел та застосовувати кнопки KEY. Синтезуйте створену схему. Запишіть створений файл конфігурації в FPGA та переконайтеся в коректній роботі створеної схеми;

8. Опишіть схему із попереднього завдання на мові Verilog. Синтезуйте створену схему. Запишіть створений файл конфігурації в FPGA та переконайтеся в коректній роботі створеної схеми. Перегляньте результат синтезу в Netlist Viewers;

#### **0.4 Контрольні запитання**

1. Поясніть один або кілька етапів роботи з САПР Quartus Prime, розглянутий в лабораторній роботі (створення проекту, імпорт налаштувань портів вводу-виводу FPGA мікросхеми, створення схеми у схемному редакторі, синтез цифрової схеми, запис конфігураційного файлу в FPGA, перегляд результатів синтезу в Netlist Viewers);
2. Поясніть, що таке логічні рівні цифрового сигналу;
3. Поясніть чому логічні рівні передавача і приймача цифрового сигналу відрізняються;
4. Поясніть, як використання логічних рівнів збільшує завадозахищеність передачі даних;
5. Поясніть чому дорівнюють значення  $V_{IL}$ ,  $V_{IH}$ ,  $V_{OL}$ ,  $V_{OH}$  для логічних рівнів LVCMOS;
6. Дайте визначення параметрам імпульсного сигналу (період, частота, тривалість імпульсу, коефіцієнт заповнення, передній фронт, задній фронт);
7. Напишіть таблицю істинності і логічну функцію та намалюйте умовне графічне позначення одного з логічних елементів ВИКЛЮЧАЮЧЕ-АБО (XOR), І-НІ (NAND), АБО-НІ (NOR), ВИКЛЮЧАЮЧЕ-АБО-НІ (XNOR);
8. Намалюйте схему компаратора для перевірки двох 5-розрядних чисел на рівність. Поясніть принцип роботи такого компаратора;
9. Поясніть в загальному вигляді конструкцію FPGA мікросхем;

10. Поясніть, як описати модуль з багаторозрядними вхідними і вихідними портами на мові Verilog;
11. Поясніть, які типи даних можна використовувати для опису вхідних і вихідних портів модуля на мові Verilog;
12. Поясніть властивості і область використання типу wire в мові Verilog;
13. Поясніть властивості і область використання типу reg в мові Verilog;
14. Поясніть як створювати багаторозрядні шини типів wire та reg у мові Verilog;
15. Поясніть синтаксис запису констант у мові Verilog;
16. Поясніть логіку роботи оператора assign у мові Verilog;

### **0.5 Перелік посилань**

[0.1] “IO Interface Standards. Application Note AN-230” 2004. [Online]. Available: <https://www.idt.com/document/apn/230-io-interface-standards>

[0.2] Відео лекції із цифрового дизайну на мові Verilog, 2015 [Online]. Режим доступу:

<https://www.youtube.com/playlist?list=PL4WQQHlheqfyIZSzsJXqs6kBQhhsTRBC7>

[0.3] САПР Quartus Prime Lite, 2018 [Online]. Режим доступу:

<http://fpgasoftware.intel.com/?edition=lite>

[0.4] Опис налагоджувальних плат для виконання лабораторних робіт, 2018 [Online]. Режим доступу: [https://github.com/KorotkiyEugene/intel\\_fpga\\_boards](https://github.com/KorotkiyEugene/intel_fpga_boards)

[0.5] Вихідні коди прикладів до лабораторних робіт, 2018 [Online]. Режим доступу: [https://github.com/KorotkiyEugene/digital\\_lab](https://github.com/KorotkiyEugene/digital_lab)

[0.6] Інструкція зі встановлення драйверу програматора USB Blaster для ОС Windows, 2018 [Online]. Режим доступу:

[http://www.terasic.com.tw/wiki/Altera\\_USB\\_Blaster\\_Driver\\_Installation\\_Instructions](http://www.terasic.com.tw/wiki/Altera_USB_Blaster_Driver_Installation_Instructions)

[0.7] Інструкція зі встановлення драйверу програматора USB Blaster для ОС Linux, 2018 [Online]. Режим доступу:

<https://rocketboards.org/foswiki/Documentation/UsingUSBBlasterUnderLinux>