Мови опису апаратури та основи SystemVerilog

Мета лекції

Опанувати призначення мов опису апаратури та базовий синтаксис SystemVerilog, необхідний для читання, моделювання й подальшого проєктування цифрових модулів.

1. Мова опису апаратури та процес проєктування

Місце теми в курсі

Попередні теми описували цифрові вузли за допомогою таблиць істинності, рівнянь і структурних схем. SystemVerilog дозволяє подати ці самі об’єкти як вихідний код.

Текстовий опис зручний для повторного використання, параметризації, автоматичної перевірки та синтезу на програмованій вентильній матриці або в спеціалізованій мікросхемі.

- Булева алгебра задає функцію
- Схема показує з’єднання елементів
- SystemVerilog описує ту саму апаратуру текстом
- Засоби проєктування виконують симуляцію та синтез

Результати навчання

Мета лекції полягає не у вивченні всієї мови, а у формуванні правильної базової моделі. Студент повинен розуміти, що код описує апаратну структуру, яка працює паралельно.

Практичний результат: уміння прочитати модуль, визначити його порти й розрядність, пояснити оператори та передбачити, яку логічну схему може утворити синтезатор.

- Розшифровувати HDL, RTL, EDA, DUT і TB
- Створювати module з портами input та output
- Описувати вектори, числа і параметри
- Розрізняти логічні, побітові та редукційні оператори
- Застосовувати assign, always_comb і case

Що описує HDL

HDL описує цифровий пристрій на кількох рівнях. Інтерфейс визначає входи та виходи. Внутрішній опис задає логіку, регістри й зв’язки між підмодулями.

Окремий тестовий опис формує вхідні сигнали та перевіряє виходи. Такий опис не обов’язково синтезується в апаратуру, але використовується симулятором.

- Інтерфейс цифрового вузла
- Комбінаційні залежності між сигналами
- Регістри та оновлення за Clock
- Ієрархію модулів і з’єднання
- Тестові стимули для симуляції

HDL і програмна мова

У звичайній програмі послідовність операторів часто визначає порядок обчислень. У HDL багато конструкцій описують фізично паралельні блоки.

Наприклад, два оператори assign безперервно обчислюють два виходи одночасно. Переміщення одного assign вище або нижче не створює часової послідовності між ними.

- Програма виконує інструкції в заданому порядку
- Апаратні блоки працюють одночасно
- Два assign описують дві паралельні логічні залежності
- Порядок незалежних безперервних присвоєнь не задає порядок роботи

Ознака	Програмна мова	HDL
Модель виконання	Послідовність інструкцій	Паралельна апаратна структура
Основний об’єкт	Алгоритм	Сигнали, блоки та зв’язки
Результат	Машинний код для процесора	Модель або синтезована схема

Рівні опису апаратури

Поведінковий опис (behavioral description) концентрується на функції. Рівень передавання між регістрами (Register Transfer Level, RTL) явно розділяє регістри й комбінаційні шляхи.

Структурний опис (structural description) нагадує текстову принципову схему: створюються екземпляри модулів і задаються з’єднання. Після синтезу засіб може сформувати вентильне подання.

- Поведінковий рівень описує потрібний результат
- RTL описує регістри та комбінаційну логіку між ними
- Структурний рівень з’єднує готові модулі та примітиви
- Вентильний рівень містить логічні елементи й з’єднання

Рівень	Що описує	Приклад
Поведінковий	Функцію без детальної структури	if, case, вирази
RTL	Регістри та логіку між ними	always_ff, always_comb
Структурний	Екземпляри й з’єднання	module instances
Вентильний	Логічні елементи	and, or, xor

Verilog і SystemVerilog

Verilog є основою синтаксису. SystemVerilog додає точніші типи даних, спеціалізовані процедурні блоки, інтерфейси, перевірки та об’єктні засоби для тестового оточення.

У навчальному курсі використовується SystemVerilog, тому вихідні файли зазвичай мають розширення .sv. Базові модулі залишаються зрозумілими для тих, хто бачив класичний Verilog.

- Verilog створений для опису цифрової апаратури
- SystemVerilog зберігає сумісність із більшістю Verilog-коду
- logic замінює багато застосувань reg
- always_comb, always_ff і always_latch уточнюють призначення блоків
- Мова містить розширені засоби testbench

Шлях від коду до схеми

Засоби автоматизації електронного проєктування (Electronic Design Automation, EDA) обробляють текстовий опис кількома етапами. Компілятор спочатку знаходить синтаксичні й типові помилки.

Симулятор виконує модель і формує часові діаграми. Синтезатор будує апаратну мережу. Для FPGA після синтезу виконуються розміщення (place) та трасування (route).

- Редагування вихідного файлу .sv
- Компіляція та перевірка синтаксису
- Симуляція з тестовим оточенням
- Синтез у логічну структуру
- Розміщення й трасування для FPGA

Файли проєкту

Розширення .sv повідомляє інструменту, що файл потрібно обробляти як SystemVerilog. Назва файла не зобов’язана збігатися з назвою модуля, але така домовленість спрощує навігацію.

У невеликому проєкті кожний основний модуль зазвичай зберігають окремо. Testbench також виносять у власний файл, щоб не змішувати синтезований опис із симуляційними конструкціями.

- .sv містить код SystemVerilog
- Один файл може містити кілька модулів
- Назва файла часто відповідає головному module
- RTL-код відокремлюється від testbench
- Порядок компіляції має враховувати залежності

2. Модулі, порти й типи сигналів

Поняття модуля

Модуль (module) можна порівняти з блоком на структурній схемі. Він має визначений інтерфейс і приховану внутрішню реалізацію.

Один опис модуля є шаблоном. Кожне створення екземпляра (instance) утворює окрему апаратну копію з власними сигналами.

- module задає ім’я цифрового блока
- Список портів визначає зовнішній інтерфейс
- Внутрішня частина містить сигнали й логіку
- endmodule завершує опис
- Модуль можна створювати багато разів як екземпляр

Каркас модуля

Найпростіший модуль має заголовок, список портів, внутрішню логіку та ключове слово endmodule. Кожна декларація або оператор завершується крапкою з комою.

Коментар після endmodule із назвою модуля не впливає на модель, але допомагає читати великі файли.

```
module and2 (
   input  logic a,
   input  logic b,
   output logic y
);
   assign y = a & b;
endmodule
```

- module оголошує блок
- Порти перелічуються в дужках
- Крапка з комою завершує оголошення
- Внутрішні оператори розміщуються між module та endmodule

Назва модуля і список портів

Ім’я модуля та сигналів має бути змістовним. Ідентифікатори data_out і Data_Out є різними, оскільки SystemVerilog чутливий до регістру літер.

Сучасний стиль ANSI дозволяє одразу в списку портів вказати input, output, розрядність і тип. Це зменшує кількість повторних декларацій.

- Ідентифікатор починається літерою або символом підкреслення
- Мова розрізняє великі й малі літери
- Порти розділяються комами
- Напрямок і тип доцільно задавати безпосередньо в заголовку

Напрямки портів

Напрямок порту визначається відносно конкретного модуля. Сигнал, який є output одного модуля, може бути input іншого.

Двонапрямлений порт inout застосовують, наприклад, для зовнішньої шини з трьома станами. Для внутрішніх FPGA-з’єднань зазвичай використовують окремі сигнали входу, виходу та дозволу.

- input читається всередині модуля
- output формується модулем
- inout може передавати дані в обох напрямках
- Для звичайної внутрішньої логіки inout майже не потрібний

Напрямок	Значення
input	Сигнал надходить у модуль
output	Модуль формує сигнал
inout	Двонапрямлена лінія з керуванням станом Z

Тип logic

Тип logic зберігає логічний нуль, одиницю, невідоме значення X або високоімпедансний стан Z. Це дозволяє симулятору показувати невизначені та від'єднані сигнали.

У SystemVerilog logic використовують для портів і внутрішніх змінних. Якщо сигнал має кілька апаратних джерел, потрібно перевірити правила драйверів і, за потреби, застосувати мережевий тип wire.

- logic є чотиристанним типом
- Може бути скаляром або вектором
- Замінює більшість навчальних застосувань reg
- Один сигнал не повинен мати кілька незалежних драйверів

wire і logic

wire моделює фізичне з'єднання, значення якого визначають драйвери. logic подає змінну, якій присвоюють значення в процедурному блоці або формують одним джерелом.

Для початкових робіт зручно оголошувати більшість портів як logic, але потрібно пам'ятати правило єдиного драйвера. Конфлікт кількох присвоєнь є помилкою структури.

- wire є мережевим типом
- logic є типом змінної
- assign природно формує безперервний драйвер
- always_comb присвоює процедурній змінній
- Один output logic не слід формувати з кількох блоків

Властивість	wire	logic
Категорія	Мережа	Чотиристанна змінна
Типове джерело	Безперервний або портовий драйвер	always_comb, always_ff або один драйвер
Кілька драйверів	Можливі за правилами розв'язання	Не допускаються для звичайного RTL

Скаляр і вектор

Скалярний сигнал має один розряд. Упакований вектор (packed vector) задають діапазоном перед іменем. logic [7:0] data містить біти від 7 до 0.

Окремий біт вибирається індексом, а частина вектора — діапазоном. Вираз data[7:4] містить чотири старші біти восьмирозрядного слова.

```
logic       ready;
logic [7:0] data;
logic [3:0] high_nibble;

assign high_nibble = data[7:4];
```

- logic а описує один біт
- logic [7:0] data описує 8 бітів
- data[0] вибирає окремий біт
- data[7:4] вибирає частину вектора
- Ширина виразу впливає на результат

Напрямок індексів

Діапазон [7:0] означає індекси 7,6,...,0. Зазвичай біт 7 трактують як старший біт (Most Significant Bit, MSB), а біт 0 — як молодший біт (Least Significant Bit, LSB).

Висхідний діапазон [0:7] також містить вісім бітів, але змінює порядок індексів. Змішування напрямків без явного контролю може переставити біти.

- [7:0] має спадний діапазон
- [0:7] має висхідний діапазон
- Лівий індекс є найлівішим у записі вектора
- Домовленість [MSB:LSB] спрощує читання двійкових чисел

3. Числа, параметри та оператори

Числові літерали

Формат числа має вигляд ширина'основа_значення. Літера b задає двійкову основу, o — вісімкову, d — десяткову, h — шістнадцяткову.

Явна ширина запобігає неочікуваному обрізанню або розширенню. 8'hA6 і 8'b1010_0110 задають однакові вісім бітів.

```
logic [7:0] a = 8'b1010_0110;
logic [7:0] b = 8'hA6;
logic [11:0] n = 12'd25;
```

- 8'b1010_0110 — 8 бітів у двійковій системі
- 8'hA6 — те саме значення у шістнадцятковій системі
- 12'd25 — десяткове 25 у 12 бітах
- 4'o7 — вісімкове 7 у 4 бітах
- Підкреслення покращує читання і не змінює значення

Літерал	Ширина	Основа	Значення
8'b1010_0110	8	Двійкова	166
8'hA6	8	Шістнадцяткова	166
12'd25	12	Десяткова	25
4'o7	4	Вісімкова	7

Значення X і Z

X не є окремим фізичним рівнем цифрового входу. Це значення симулятора, яке означає, що нуль або одиницю визначити неможливо.

Z моделює високоімпедансний стан. Зовнішній двонапрямлений контакт може переходити в Z, щоб інший пристрій отримав право керувати лінією.

```
assign bus = enable ? data_out : 8'bz;
```

- X виникає через невизначену ініціалізацію або конфлікт
- Z дозволяє від’єднати драйвер від спільної шини
- X і Z корисні для діагностики симуляції
- У синтезованій внутрішній логіці FPGA стан Z часто перетворюється на мультиплексор

Знакові й беззнакові дані

Вектор 8'b1111_1111 без знакової ознаки дорівнює 255. Той самий бітовий шаблон у logic signed [7:0] дорівнює −1 у додатковому коді.

Тип signed впливає на порівняння, розширення розрядності та оператор >>>. Для передбачуваного результату операнди мають узгоджені ширини й знаковість.

- logic [7:0] за замовчуванням беззнаковий
- logic signed [7:0] подає знакове число
- Однакові біти можуть мати різне числове тлумачення
- Змішані signed і unsigned вирази потребують обережності

Параметри module

Параметр (parameter) створює конфігурований модуль. Наприклад, WIDTH=8 задає типову ширину, але під час створення екземпляра можна вибрати інше значення.

Локальний параметр (localparam) застосовують для внутрішніх констант, які не повинні змінюватися зовні. Це захищає службові значення модуля.

```
module pass #(
    parameter int WIDTH = 8
) (
    input  logic [WIDTH-1:0] a,
    output logic [WIDTH-1:0] y
);
```

```
    assign y = a;
endmodule
```

- parameter задає значення під час створення екземпляра
- localparam не призначений для зовнішньої зміни
- Параметр WIDTH часто керує розрядністю портів
- Один опис може реалізувати модулі різної ширини

Класи операторів

SystemVerilog має оператори, подібні до програмних мов, але результат стає апаратною логікою. Наприклад, додавання може створити суматор, а умовний оператор — мультиплексор.

Однаковий символ може працювати з векторами різної ширини. Перед використанням потрібно визначити ширину результату, знаковість і поведінку X.

- Арифметичні: +, -, *, /, %
- Порівняння: ==, !=, <, <=, >, >=
- Логічні: !, &&, ||
- Побітові: ~, &, |, ^
- Зсуви: <<, >>, <<<, >>>
- Об’єднання та умовний вибір

Логічні та побітові оператори

Логічні оператори трактують багаторозрядний операнд як умову: нуль означає false, ненульове значення — true. Результат має один біт.

Побітові оператори зіставляють біти за позиціями. Для a=4'b1010 і b=4'b1100 вираз a & b дорівнює 4'b1000. Заперечення $\bar{A}$ у формулах для окремого біта часто записується в SystemVerilog як ~a.

```
a = 4'b1010;
b = 4'b1100;

a & b   // 4'b1000
a && b  // 1'b1
~a      // 4'b0101
```

- a && b формує однобітний логічний результат
- a & b виконує побітове І
- !a є логічним запереченням
- ~a інвертує кожний біт вектора
- Для однобітних сигналів результати часто збігаються, але зміст різний

Оператор	Тип	Результат для a=1010, b=1100
a & b	Побітовий	1000
a | b	Побітовий	1110
a && b	Логічний	1

Оператор	Тип	Результат для a=1010, b=1100
!a	Логічний	0
~a	Побітовий	0101

Редукційні оператори

Редукційний оператор (reduction operator) стоїть перед одним вектором. Він послідовно застосовує операцію до всіх бітів і формує один біт.

Для data=4'b1111 вираз &data дорівнює 1. Для data=4'b1010 вираз |data дорівнює 1, а ^data дорівнює 0, оскільки кількість одиниць парна.

```
logic [3:0] data = 4'b1010;

&data  // 0
|data  // 1
^data  // 0
```

- &data дорівнює 1, якщо всі біти data дорівнюють 1
- |data дорівнює 1, якщо хоча б один біт дорівнює 1
- ^data обчислює парність
- Редукція має один операнд
- Побітова операція зазвичай має два операнди

Вираз	data=1010	Тлумачення
&data	0	Усі біти дорівнюють 1
|data	1	Є хоча б одна 1
^data	0	Парність кількості одиниць

Оператори зсуву

Логічні зсуви переміщують біти й заповнюють звільнені позиції нулями. Для фіксованої розрядності біти, що вийшли за межі, втрачаються.

Арифметичний правий зсув >>> для знакового операнда копіює знаковий біт. Це відповідає діленню від'ємного числа на степінь двійки з урахуванням знака.

```
logic [7:0] u = 8'b1001_0000;
logic signed [7:0] q = 8'b1001_0000;

u >> 2   // 0010_0100
q >>> 2  // 1110_0100
```

- << — логічний зсув ліворуч
- >> — логічний зсув праворуч

- <<< — арифметичний зсув ліворуч
- >>> — арифметичний зсув праворуч
- Поведінка >>> залежить від signed

Конкатенація і повторення

Конкатенація (concatenation) створює вектор з кількох частин. Ліва частина стає старшими бітами результату.

Якщо hi=4'b1010 і lo=4'b0011, то {hi,lo}=8'b1010_0011. Реплікація {4{1'b1}} формує 4'b1111.

```
logic [3:0] hi = 4'b1010;
logic [3:0] lo = 4'b0011;
logic [7:0] word;

assign word = {hi, lo};
assign mask = {8{enable}};
```

- {a,b} виконує конкатенацію
- {4{bit}} повторює bit чотири рази
- Порядок елементів визначає порядок бітів
- Загальна ширина дорівнює сумі ширин частин

Умовний оператор

Умовний оператор ?: формує результат залежно від умови. Для однобітного sel вираз sel ? d1 : d0 реалізує функцію мультиплексора 2:1.

Ширини обох гілок мають бути узгоджені з виходом. Надмірно вкладені умовні оператори важко перевіряти, тому для багатьох варіантів доцільніший case.

```
assign y = sel ? d1 : d0;
```

- Формат: condition ? value_true : value_false
- Умова вибирає один із двох виразів
- Гілки повинні мати узгоджену ширину
- Вкладені оператори знижують читабельність

4. Комбінаційний опис

Безперервне присвоєння assign

Безперервне присвоєння (continuous assignment) постійно обчислює праву частину. Якщо будь-який вхід змінюється, симулятор переобчислює вихід.

Кілька assign у модулі не виконуються послідовно. Кожний описує окрему апаратну залежність. Ліва частина не повинна одночасно формуватися іншим блоком.

```
assign and_y = a & b;
assign or_y  = a | b;
assign inv_y = ~a;
```

- assign описує комбінаційну залежність
- Ліва частина має одного драйвера

- Оператори assign працюють паралельно
- Затримку поширення визначає реалізована схема

Логічні елементи в SystemVerilog

Для однобітних входів побітові оператори відповідають логічним елементам. Формула $Y=\overline{A}B \vee AC$ у SystemVerilog записується як assign y = (~a & b) | (a & c).

Дужки роблять групування явним. Синтезатор спрощує вираз і будує еквівалентну комбінаційну мережу.

```
module logic_fn (
   input  logic a, b, c,
   output logic y
);
   assign y = (~a & b) | (a & c);
endmodule
```

- І: y=a&b
- АБО: y=a|b
- Виключне АБО: y=a^b
- НЕ: y=~a
- Складений вираз зберігає пріоритет або використовує дужки

Мультиплексор 2 до 1

Мультиплексор 2:1 має два входи даних, один вхід вибору та один вихід. Таблиця функції містить лише два режими.

Умовний оператор прямо відображає цю функцію. Якщо d0 і d1 є векторами, один і той самий вираз формує багаторозрядний мультиплексор.

```
module mux2 #(parameter int WIDTH=8) (
   input  logic [WIDTH-1:0] d0, d1,
   input  logic             sel,
   output logic [WIDTH-1:0] y
);
   assign y = sel ? d1 : d0;
endmodule
```

- sel=0 передає d0
- sel=1 передає d1
- Вихід є комбінаційним
- Опис параметризується за шириною WIDTH

Процедурний блок always_comb

always_comb є процедурним блоком SystemVerilog для комбінаційного опису. Симулятор повторно виконує блок, коли змінюється будь-який прочитаний сигнал.

На відміну від незалежних assign, оператори всередині одного begin...end виконуються в записаному порядку під час одного обчислення блока. Результат усе одно має описувати комбінаційну схему без пам’яті.

```
always_comb begin
    y = a & b;
end
```

- Блок призначений для комбінаційної логіки
- Список чутливості формується автоматично
- Виходи мають отримувати значення на всіх шляхах
- Кілька операторів усередині виконуються процедурно

Блокувальне присвоєння

Блокувальне присвоєння (blocking assignment) позначається символом =. Воно завершується перед переходом до наступного оператора процедурного блока.

Для комбінаційної логіки це дозволяє послідовно обчислювати проміжні значення. Неблокувальне присвоєння <= буде основним для always_ff і регістрів у лекції про послідовнісну логіку SystemVerilog.

```
always_comb begin
    temp = a & b;
    y = temp | c;
end
```

- Оператор = оновлює змінну одразу в межах блока
- Наступний рядок бачить нове значення
- Порядок рядків усередині блока має значення
- Неблокувальне <= переважно застосовують у тактових блоках

Мультиплексор у always_comb

Мультиплексор можна описати умовним оператором або конструкцією if. Обидва описи синтезуються в еквівалентну схему, якщо всі випадки визначені.

У гілці if формується d1, у else — d0. Відсутність else залишила б у без нового значення для sel=0 і могла б створити засувку.

```
always_comb begin
    if (sel)
        y = d1;
    else
        y = d0;
end
```

- if перевіряє sel
- Обидві гілки присвоюють y
- Вихід визначений для sel=0 і sel=1
- Синтезована функція збігається з assign

Повнота присвоєнь

Комбінаційна схема не зберігає попередній вихід. Тому в always_comb кожна змінна зліва повинна отримувати значення за всіх умов.

Поширений шаблон задає безпечне значення на початку блока, а потім змінює його в окремих гілках. Це робить повноту видимою й запобігає ненавмисній засувці.

```
always_comb begin
    y = '0;
    if (enable)
        y = data;
end
```

- Кожна гілка if має визначати вихід
- case повинен охоплювати всі значення або мати default
- Значення за замовчуванням задається на початку блока
- Неповний опис може створити latch

Конструкція case

Конструкція case порівнює керувальний вираз із переліком варіантів. Вона природно описує мультиплексори, декодери та логіку керування.

Гілка default визначає поведінку для неохоплених кодів, а також допомагає уникнути пам'яті. Для базових синтезованих описів слід починати зі звичайного case.

```
always_comb begin
    case (sel)
        2'b00: y = d0;
        2'b01: y = d1;
        2'b10: y = d2;
        default: y = d3;
    endcase
end
```

- Вираз case обчислюється один раз
- Кожний пункт задає конкретне значення
- default обробляє невказані випадки
- begin...end групує кілька операторів у пункті

Дешифратор 2 до 4

Дешифратор 2:4 перетворює двобітний код в один активний вихід із чотирьох. На початку у отримує 4'b0000.

case встановлює один біт відповідно до a. Для a=2'b10 вихід дорівнює 4'b0100. Такий опис легко перевірити за таблицею істинності.

```
always_comb begin
    y = 4'b0000;
    case (a)
        2'b00: y[0] = 1'b1;
        2'b01: y[1] = 1'b1;
        2'b10: y[2] = 1'b1;
        2'b11: y[3] = 1'b1;
    endcase
end
```

- Вхід a має два біти
- Вихід y має чотири біти
- У кожному режимі активна одна позиція
- Опис охоплює всі чотири коди

5. Ієрархія модулів

Створення екземпляра module

Щоб використати модуль усередині іншого, створюють його екземпляр. Назва модуля вказує тип блока, а ім'я u_mux визначає конкретну копію.

З'єднання портів передає сигнали між рівнями ієрархії. Два екземпляри одного mux2 утворюють два незалежні мультиплексори.

```
mux2 #(.WIDTH(8)) u_mux (
    .d0  (a),
    .d1  (b),
    .sel (select),
    .y   (result)
);
```

- Спочатку записується тип модуля
- Потім задається унікальне ім'я екземпляра
- Порти підключаються до локальних сигналів
- Кожний екземпляр створює окрему апаратну копію

Іменоване і позиційне підключення

Позиційний запис коротший, але кожний сигнал повинен стояти на тому самому місці, що й порт у декларації модуля. Помилка порядку може залишитися синтаксично правильною.

Іменоване підключення явно зв'язує назву порту з локальним сигналом. У навчальних і командних проєктах цей стиль є кращим за замовчуванням.

```
// Іменоване
and2 u1 (.a(x), .b(z), .y(f));

// Позиційне
and2 u2 (x, z, f);
```

- Позиційне підключення залежить від порядку портів
- Іменоване має формат .port(signal)
- Іменований стиль дозволяє читати призначення кожного зв'язку
- Зміна порядку портів не ламає іменоване підключення

Ієрархія цифрової системи

Верхній модуль (top module) відповідає зовнішнім контактам або основному рівню моделі. Усередині він створює екземпляри менших функціональних блоків.

Декомпозиція зменшує складність: кожний модуль можна перевірити окремо, а потім перевірити їхню взаємодію на верхньому рівні.

- Top module є коренем проєкту
- Підмодулі реалізують окремі функції
- Сигнали з'єднують виходи та входи
- Ієрархія спрощує повторне використання й перевірку

6. Симуляція, testbench і синтез

Симуляція і синтез

Симуляція (simulation) перевіряє функціональну поведінку моделі. Вона дозволяє бачити X, Z, затримки тестових стимулів і значення внутрішніх сигналів.

Синтез (synthesis) перетворює підтримувану підмножину SystemVerilog на логічні ресурси. Конструкції тестового оточення можуть працювати в симуляторі, але не мати апаратного аналога.

- Симуляція показує значення сигналів і часові діаграми
- Синтез оцінює, яку логіку можна реалізувати
- Не кожна конструкція симуляції є синтезованою
- Коректна симуляція не гарантує правильних часових параметрів реалізації

Ознака	Симуляція	Синтез
Мета	Перевірити поведінку	Побудувати апаратну структуру
Результат	Waveform і повідомлення	Нетліст та оцінка ресурсів
Підтримувані конструкції	Широкий набір, включно з testbench	Лише синтезована підмножина

initial і затримка решітка

Блок initial виконується один раз на початку симуляції. У testbench він задає початкові значення, потім змінює входи через визначені інтервали.

Конструкція #10 є симуляційною затримкою. Вона не замінює тактовий сигнал і не повинна використовуватися як спосіб опису затримки реальної комбінаційної логіки.

```
initial begin
   a = 1'b0;
   b = 1'b0;
   #10 a = 1'b1;
   #10 b = 1'b1;
end
```

- initial починає виконання на початку симуляції
- #10 затримує наступну дію на десять одиниць модельного часу
- Так формують тестові стимули
- У синтезованому RTL часову поведінку задає Clock, а не #delay

Тестове оточення

Тестове оточення (testbench, TB) є верхнім модулем симуляції без зовнішніх портів. Усередині воно створює проєктований пристрій (Design Under Test, DUT).

Початкові блоки формують стимули. Результати можна переглядати на часовій діаграмі (waveform) або порівнювати з очікуваними значеннями через автоматичні перевірки.

- Testbench (TB) не має зовнішніх портів
- DUT означає Design Under Test
- Входи DUT формує testbench
- Виходи спостерігаються у waveform або перевіряються автоматично
- Testbench не входить у синтезовану схему

Базовий цикл перевірки

Компілятор часто показує багато повідомлень після однієї ранньої помилки. Доцільно спочатку виправити перше зрозуміле повідомлення й повторити компіляцію.

Після успішної компіляції потрібно перевірити всі суттєві комбінації входів. Синтез починають після того, як функціональна модель поводиться правильно.

- Зберегти RTL і testbench
- Запустити компіляцію
- Виправити першу синтаксичну помилку
- Запустити симуляцію
- Перевірити waveform та очікувані значення
- Лише після цього переходити до синтезу

7. Стиль і перевірка

Стиль коду SystemVerilog

Форматування не змінює апаратну функцію, але суттєво впливає на перевірку. Короткі змістовні модулі легше читати, симулювати та повторно використовувати.

Назви data_valid, write_enable і next_state передають роль сигналу краще, ніж x1 або temp2. Коментар потрібний для неочевидного рішення, обмеження або одиниці вимірювання.

- Один рівень вкладеності має однаковий відступ
- Порти й підключення вирівнюються для читання
- Назви описують функцію сигналу
- Параметри й константи мають послідовний стиль
- Коментар пояснює причину, а не повторює очевидний код

Типові помилки

Синтаксичні помилки зазвичай знаходить компілятор. Семантичні помилки складніші: код компілюється, але описує іншу схему.

Для діагностики потрібно читати попередження про ширину, latch і драйвери. Після кожної зміни корисно перевірити очікувану таблицю істинності та форму сигналів.

- Пропущена крапка з комою або endmodule
- Різний регістр літер у назвах
- Невідповідність ширини порту й сигналу
- Кілька драйверів одного logic
- Неповний if або case створює latch

- Логічний оператор використано замість побітового

Підсумок базових конструкцій

Базовою одиницею SystemVerilog є module. Порти формують зовнішній контракт, типи й діапазони визначають подання сигналів, а оператори задають логічні залежності.

assign підходить для компактних рівнянь, always_comb — для розгалуженої комбінаційної логіки. Ієрархія складається з екземплярів. Testbench і симуляція перевіряють функцію до синтезу.

- module і порти визначають інтерфейс
- logic і вектори описують сигнали
- Явні літерали задають ширину та основу
- assign і always_comb описують комбінаційну логіку
- Екземпляри утворюють ієрархію
- Симуляція передує синтезу

8. Покроковий приклад

Потрібно описати однобітний мультиплексор 2:1. Входи d0 і d1 містять дані, sel вибирає джерело, y є виходом. За sel=0 вихід дорівнює d0, за sel=1 — d1.

```
module mux2 (
   input  logic d0, d1,
   input  logic sel,
   output logic y
);
   assign y = sel ? d1 : d0;
endmodule
```

- Рядок module mux2 задає назву блока.
- input оголошує три вхідні сигнали.
- output оголошує вихід.
- Умовний оператор вибирає d1 за sel=1 та d0 за sel=0.
- endmodule завершує опис.

9. Терміни та скорочення

Скорочення	Повна назва	Зміст
HDL	Hardware Description Language	Мова опису апаратури
RTL	Register Transfer Level	Рівень передавання між регістрами
EDA	Electronic Design Automation	Засоби автоматизації проєктування
DUT	Design Under Test	Пристрій, який перевіряється
TB	Testbench	Тестове оточення
MSB	Most Significant Bit	Старший біт

Скорочення	Повна назва	Зміст
LSB	Least Significant Bit	Молодший біт
FPGA	Field-Programmable Gate Array	Програмована вентильна матриця

10. Питання для самоперевірки

- Чим HDL відрізняється від звичайної програмної мови?
- З яких частин складається module?
- Чим input відрізняється від output та inout?
- Що означає logic [7:0]?
- Як записати 166 у восьми бітах у двійковій і шістнадцятковій формах?
- Чим оператор & відрізняється від &&?
- Для чого потрібні редукційні оператори?
- Коли використовують assign, а коли always_comb?
- Чому неповний if може створити latch?
- Чим симуляція відрізняється від синтезу?

11. Практична основа для моделювання комбінаційної функції

Лабораторна робота №5 поєднує булеву алгебру з базовими конструкціями SystemVerilog. Для її виконання потрібно не тільки записати модуль, а й створити тестове оточення, перебрати всі набори входів, отримати консольний результат і прочитати часову діаграму.

Стандартний приклад

Нехай задано F(A,B,C)=Σm(1,3,6,7). Це означає, що F=1 для наборів 001, 011, 110 і 111. Номер мінтерма утворюється з двійкового слова ABC, де A є старшим бітом, а C — молодшим.

m	A	B	C	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

На карті Карно група m(1,3) зберігає A=0 і C=1, тому дає $\bar{A}C$. Група m(6,7) зберігає A=1 і B=1, тому дає AB. Мінімізований вираз: F=$\bar{A}C$+AB.

Перенесення булевого виразу в код

Булева операція	Позначення у формулі	SystemVerilog
НЕ	$\bar{A}$	~A
І	AB або A·B	A & B
АБО	A+B	A | B

Для стандартного прикладу $\bar{A}C$ переходить у (~A & C), а AB — у (A & B). Повний кодовий вираз дорівнює (~A & C) | (A & B). Дужки роблять межі логічних добутків очевидними.

Файл design.sv

```
module logic_function (
    input  logic A, B, C,
    output logic F
);
    assign F = (~A & C) | (A & B);
endmodule
```

module та endmodule задають межі цифрового блока. input logic оголошує входи, output logic — вихід. assign описує безперервну комбінаційну залежність. Вихід змінюється після кожної зміни будь-якого операнда у правій частині.

Повний testbench

```
`timescale 1ns/1ps
```

```
module testbench;
    logic A, B, C;
    logic F;
    integer i;

    logic_function dut (
        .A(A), .B(B), .C(C), .F(F)
    );

    initial begin
        $dumpfile("dump.vcd");
        $dumpvars(0, testbench);
        $display("A B C | F");
        for (i = 0; i < 8; i = i + 1) begin
            {A, B, C} = i[2:0];
            #1;
            $display("%0b %0b %0b | %0b", A, B, C, F);
        end
        $finish;
    end
endmodule
```

Testbench не має зовнішніх портів. Екземпляр dut підключає локальні сигнали до портів logic_function. Цикл for генерує всі вісім комбінацій у двійковому порядку. Затримка #1 утримує кожний набір одну наносекунду та відокремлює його на часовій діаграмі.

Системні задачі та формат виведення

Запис	Значення
`timescale 1ns/1ps	одиниця часу 1 нс, точність 1 пс
$dumpfile("dump.vcd")	назва файла часових даних
$dumpvars(0, testbench)	запис сигналів тестового оточення
$display	одноразове консольне виведення
%0b	двійкове значення без зайвих початкових нулів
$finish	завершення симуляції

Очікуваний результат

```
A B C | F
0 0 0 | 0
0 0 1 | 1
0 1 0 | 0
0 1 1 | 1
1 0 0 | 0
1 0 1 | 0
1 1 0 | 1
1 1 1 | 1
```

Кожний рядок зіставляється з мінтермом. Наприклад, ABC=011 має номер 3, тому m3 належить списку і F=1. Набір ABC=101 має номер 5, мінтерм m5 відсутній, тому F=0.

Часова діаграма

Час, нс	ABC	A	B	C	F
0–1	000	0	0	0	0
1–2	001	0	0	1	1

2–3	010	0	1	0	0
3–4	011	0	1	1	1
4–5	100	1	0	0	0
5–6	101	1	0	1	0
6–7	110	1	1	0	1
7–8	111	1	1	1	1

A змінюється раз після четвертого набору. B змінюється через кожні два набори. C змінюється на кожному наборі. Послідовність F дорівнює 0, 1, 0, 1, 0, 0, 1, 1. Високі ділянки F мають бути саме на інтервалах 1–2, 3–4, 6–7 і 7–8 нс.

Вертикальний фронт лише позначає момент зміни. Значення читається на горизонтальній ділянці між двома фронтами. У комбінаційній схемі F визначається поточними A, B і C, а не попереднім станом.

EDA Playground і EPWave

- У Testbench + Design обирається SystemVerilog/Verilog.
- У Tools & Simulators, у групі Free Simulators, обирається Icarus Verilog 12.0.
- Прапорець Open EPWave after run вмикається до запуску.
- Код TB розміщується у testbench.sv, код модуля — у design.sv.
- Після Run консольний результат перевіряється у Log.
- У EPWave натискається Get Signals, обирається область testbench, сигнали A, B, C, F і команда Append Selected.

Еквівалентний опис через always_comb

```
always_comb begin
    case ({A, B, C})
        3'b001, 3'b011,
        3'b110, 3'b111: F = 1'b1;
        default:         F = 1'b0;
    endcase
end
```

case перевіряє трьохбітний код {A,B,C}. Чотири перелічені набори дають F=1, усі інші охоплює default. Оскільки F отримує значення в кожній можливій ситуації, опис залишається комбінаційним і не створює latch.

Самоперевірка готовності

- Умію перетворити номер мінтерма на двійковий набір ABC.
- Умію перенести надриску, І та АБО в оператори ~, & та |.
- Розрізняю design.sv і testbench.sv.
- Розумію призначення DUT, initial, for, #1, $display, $dumpvars і $finish.
- Умію зіставити рядок таблиці істинності з інтервалом часової діаграми.
- Можу довести еквівалентність assign та always_comb за всіма вісьмома наборами.