Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут»

Опис та симуляція моделей цифрових пристроїв на Verilog

Методичні вказівки
до виконання домашньої контрольної роботи
для студентів спеціальностей
«Радіотехніка»,
«Апаратура радіозв'язку, радіомовлення та телебачення»

Затверджено Методичною радою Радіотехнічного факультету НТУУ «КПІ»

Київ
НТУУ «КПІ»
2012

Опис та симуляція моделей цифрових пристроїв на Verilog : Метод. вказівки до викон. домашньої контрольної роботи для студ. спец. «Радіотехніка», «Апаратура радіозв'язку, радіомовлення та телебачення» / Уклад.: В.С. Мосійчук. – К. : НТУУ «КПІ», 2012 – 35 с.

Гриф надано Методичною радою Радіотехнічного факультету НТУУ «КПІ»
(Протокол № 9 від 28.06.2012)

Навчальне видання

Опис та симуляція моделей цифрових пристроїв на Verilog

Методичні вказівки

до виконання розрахунково-графічної роботи
для студентів спеціальностей
«Радіотехніка» ,
«Апаратура радіозв'язку, радіомовлення та телебачення»

Укладач: *Мосійчук Віталій Сергійович, канд. техн. наук*

Відповідальний редактор О.І. Рибін, д-р техн. наук, проф.

Рецензент О.О. Шпилька, канд. техн. наук, доц.

За редакцією укладача

ЗМІСТ

Вступ 4
Мета та завдання РГР 4
Завдання і порядок виконання роботи 4
Програмне забезпечення 5
Частина 1 ПРАВИЛА ТА СИНТАКСИС МОВИ ОПИСУ ЦИФРОВИХ ПРИСТРОЇВ НА VERILOG HDL 6
1.1 Синтаксис Verilog 6
1.2 Опис комбінаційних цифрових пристроїв 8
1.2.1 Комбінаційні схеми з декількома виходами 8
1.2.2 Опис повного суматора 9
1.2.3 Опис мультиплексорів 10
1.2.4 Опис дешифраторів 11
1.3 Опис послідовнісних синхронних цифрових пристроїв 12
1.4 Опис цифрових автоматів Мілі та Мура 15
1.5 Опис ієрархічних проектів цифрових пристроїв 18
Частина 2 СИМУЛЯЦІЯ ТА ТЕСТУВАННЯ МОДУЛІВ ЦИФРОВИХ ПРИСТРОЇВ 20
2.1 Методологія тестування модулів 20
2.2 Конструкції та диретиви Verilog, що використовуються для симуляції 20
2.3 Модулі автоматичного тестування (TestBench) 21
2.3.1 Загальна структурна схема 21
2.3.2 Вивід інформації у консоль під час симуляції 22
2.3.3 Тестовий модуль з автоматичним аналізом стимулів 23
2.3.4 Універсальний тестовий модуль 24
СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ 26
ДОДАТКИ 27
Додаток А 28
Додаток Б 32

Вступ

Методичні вказівки до виконання розрахунково-графічної роботи (РГР) з дисципліни «Схемотехніка електронних апаратів. Частина 2» призначені для студентів радіотехнічного факультету НТУУ «КПІ» денної та заочної форм навчання.

Методичні вказівки містять:

- конспективне викладення теоретичного матеріалу стосовно синтаксису на правил опису цифрових пристроїв на мові VerilogHDL з прикладами (Частина 1);
- теоретичні відомості стосовно методів симуляції та автоматичного тестування модулів цифрових пристроїв (Частина 2);
- додатки з покроковими інструкціями стосовно створення та тестування проектів цифрових пристроїв в ПЗ Quartus II та ModelSIM.

Мета та завдання РГР

Мета розрахунково-графічної роботи – ознайомлення студентів з сучасними методами розробки цифрових пристроїв та систем, засвоєння навичок проектування цифрових пристроїв на програмованих логічних інтегральних схемах (ПЛІС) та цифрової частини спеціалізованих інтегральних схем (ASIC) на основі мов опису апаратних засобів (Hardware Description Languages), зокрема Verilog HDL.

Завдання і порядок виконання роботи

Завдання РГР спільне для всіх студентів, проте окремі приклади слід виконувати за індивідуальними завданнями, що отримані як домашнє завдання на практичних роботах. Зміст РГР обов’язково повинен включати в себе наступні модуль цифрових пристроїв:

- поведінковий та структурний опис комбінаційного ЦП (дешифратор, шифратор, мультиплексор, перетворювач кодів), (дом. завд. 1 та 2);
- опис тригерів та фіксаторів, синхронного лічильника з довільним

модулем лічби (дом. завд. 3);

- опис цифрових автоматів Мілі та Мура (дом. завд. 5);
- опис ієрархічних проектів (багаторозрядний суматор, арифметико-логічний пристрій, інтерфейс прямого доступу до пам'яті.

Розрахунково-графічна робота оформлюється у вигляді альбому схем з описом цифрових пристроїв на VerilogHDL, синтезованою схемою модуля (RTL) та тестовими векторами. Розрахунково графічна робота виконується протягом семестру з захистом в кінці.

Програмне забезпечення

Програмне забезпечення, що рекомендується використовувати для виконання завдання РГР орієнтоване на реалізацію цифрових пристроїв на ПЛІС, зокрема фірми Altera Corp. Quartus II. Дане ПЗ надається у безкоштовне користування у версії Web Edition, його можливо завантажити з офіційного сайту розробника www.altera.com. У ПЗ Quartus наявні можливості для симуляції проекту за допомогою Vector Waveform Analyze. Проте цей засіб не підтримує можливості симуляції, що наявні в Verilog. Тому симуляцію на Verilog рекомендується виконувати в сторонньому ПЗ ModelSIM від Mentor Graphics, що інтегрується у Quartus II під назвою ModelSIM-Altera[1]. Покрокова інструкція створення проекту в Quartus та запуск на симуляцію в ModelSIM приведені у додатках А і Б.

[1] ModelSIM-Altera встановлюється окремо.

Частина 1
ПРАВИЛА ТА СИНТАКСИС МОВИ ОПИСУ ЦИФРОВИХ ПРИСТРОЇВ НА VERILOG HDL

Сучасний рівень проектування цифрових пристроїв передбачає високий рівень абстракції. Це дозволяє перейти до опису моделей цифрових пристроїв на одній з мов опису апаратних засобів (HDL) і покласти рутинну роботу з оптимізації схем на рівні базових логічних елементів на системи автоматизованого проектування (САПР). У цьому разі для розробника достатньо вказати логічну функцію або таблицю перемикань станів цифрового пристрою. Двома найбільш поширеними мовами опису ЦП є Verilog та VHDL.

1.1 Синтаксис Verilog

Базовим блоком моделі цифрового пристрою з інтерфейсом (входами та виходами) в Verilog є модуль *module*. Існує два стилі опису ЦП поведінковий (*behavioral*) та структурний (*structural*). У разі використання поведінкового стилю в модулі описується те як пристрій має функціонувати, а у разі структурного – те як модуль побудований на основі більш простих частин. Далі приведено приклад опису простої комбінаційної схеми виключного АБО на чотири входи (рис. 1.1) двома стилями:

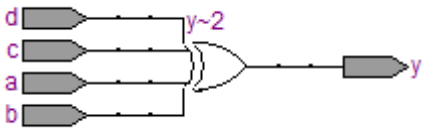


Рис. 1.1 – Виключне АБО (XOR)

Поведінковий:

```
module xor4 (y, a, b, c, d);
input  a, b, c, d;
output y;
assign y = a ^ b ^ c ^ d;
endmodule
```

Структурний:

```
module xor4 (y, a, b, c, d);
input  a, b, c, d;
output y;
xor (y, a, b, c, d);
endmodule
```

Кожен модуль починається та закінчується ключовими словами **module** та **endmodule**. Далі слідує назва модуля та перелік портів вводу та виводу. За допомогою виразу **assign** (*continuous assignment*) визначається логічна

функція комбінаційної схеми. Оператори, що можуть бути використані для визначення логічної функції наведені у табл. 1.1.

Таблиця 1.1 **Приклади операторів арифметичних та логічних функцій**

Пріоритет	Оператор	Лог. функція
↑ вищий	**!, ~**	NOT – Лог. НІ
	***, / , %**	множення, ділення, остача
	+, –	додавання, відмімання
	<<, >>	логічний зсув вліво та вправо
	<<<, >>>	арифметичний зсув вліво та вправо
	<, <=, >, >=	операції порівняння
	==, !=	порівняння на рівність
	&, ~&	AND – лог. І, NAND – лог. І-НЕ
	^, ~^	XOR – викл. АБО, NXOR – викл. АБО-НЕ
нижчий ↓	**|, ~|**	OR – лог. АБО, NOR – лог. АБО-НЕ
	?:	умовний оператор присвоєння

У разі структурного опису виключного АБО на чотири входи був використаний примітивний елемент з бібліотеки САПР

```
xor (y, a, b, c, d);
```

При чому, спочатку вказується вихід, а потім перелічуються всі входи до яких приєднані сигнали модуля. Кількість входів внаслідок технологічних обмежень може бути не більше 9-и.

Також поведінковим стилем комбінаційну схему можливо визначити за допомогою процедурного блоку always . Блок always виконується щоразу як змінюється один з синалів, що приведений у списку чутливості (*sensitivity list*) `@(a or b or c),` тому обов'язковою умовою для комбінаційних схем є включення до списку усіх його входів.

Приклад 1: Описати на Verilog Який з приведених описів модулів на VerilogHDL поведінковим стилем відповідає наступній специфікації: Вихід модуля "y" рівний 1, якщо хоча б два з трьох входів "a", "b" або "c" рівні 0.

```
module logic (y, a, b, c);
 input  a, b, c;
 output y;
 reg y;
always @(a or b or c) begin
   y = 0;
   if (!a & !b)
      y = 1;
   else if (!a & !c)
      y = 1;
   else if (!b & !c)
      y = 1;
  end
endmodule
```

У цьому разі вихід комбінаційного пристрою додатково визначається як **reg** . Це є обов'язковою умовою для всіх змінних яким виконується присвоєння в процедурному блоці always, проте оскільки в комбінаційних схемах не може бути елементів пам'яті то необхідно дотримуватися правила, що вихід обов'язково має має значення за замовчуванням. У нашому випадку **`y = 0.`** Іншим типом змінної є **wire**, що використовується для з'єднання модулів між собою. Ця змінна не передбачає створення елементів пам'яті у разі синтезу схеми.

1.2 Опис комбінаційних цифрових пристроїв

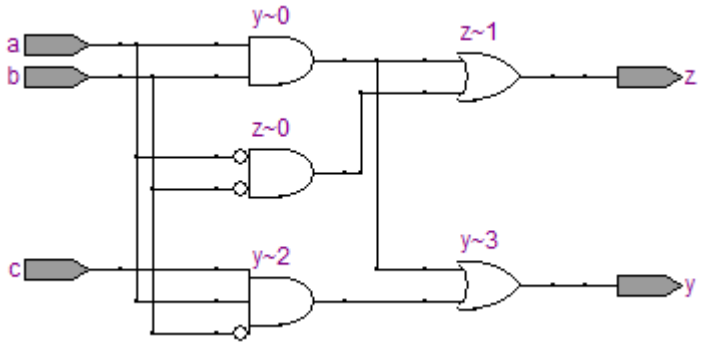
1.2.1 Комбінаційні схеми з декількома виходами

Приклад 1. Описати на Verilog комбінаційну схему, що визначається двома логічними функціями:

$$y = (a \cdot b) + (a \cdot \bar{b} \cdot c)$$
$$z = (a \cdot b) + (\bar{a} \cdot \bar{b})$$

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

Синтезова схема на RTL рівні виглядає наступним чином:



Слід зазначити, що під час синтезу САПР може виконувати мінімізацію функцій автоматично.

1.2.2 Опис повного суматора

Приклад 2. Описати на Verilog комбінаційну схему одно розрядного повного суматора.

На основі таблиці істинності повного суматора нескладно записати лог. функції повного суматора:

cin	a	b	s	cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$s = a \wedge b \wedge cin$$
$$cout = a \cdot b + a \cdot cin + b \cdot cin$$

Лог. функції з формуванням сигналів генерації g та транзиту p переносу:

$$g = a \wedge b$$
$$p = a \cdot b$$
$$s = g \wedge cin$$
$$cout = g + p \cdot cin$$

Тоді опис на Verilog можливо реалізувати двома способами:

```
module fulladder
(s, cout, a, b, cin);
  input a, b, cin;
  output reg s, cout;
  wire p, g;

  assign  p = a ^ b;
  assign  g = a & b;
  assign  s = p ^ cin;
  assign  cout = g | (p & cin);

endmodule
```

```
module fulladder
(s, cout, a, b, cin);
  input a, b, cin;
  output reg s, cout;
  reg p, g;
  always @ (a or b or cin) begin
    p = a ^ b;
    g = a & b;
    s = p ^ cin;
    cout = g | (p & cin);
  end
endmodule
```

1.2.3 Опис мультиплексорів

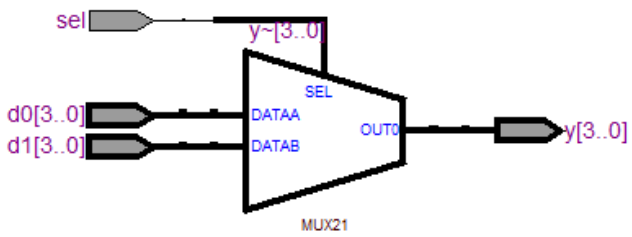
Для опису мультиплексорів або цифрових пристроїв на їх основі зручно використовувати умовне присвоєння (*conditional assignment*). Загальний синтаксис такого присвоєння наступний:

```
assign y = condition ? exp1 : exp2,
```

тобто якщо умова ***condition*** виконується, то **y = *exp1***, інакше **y = *exp2***

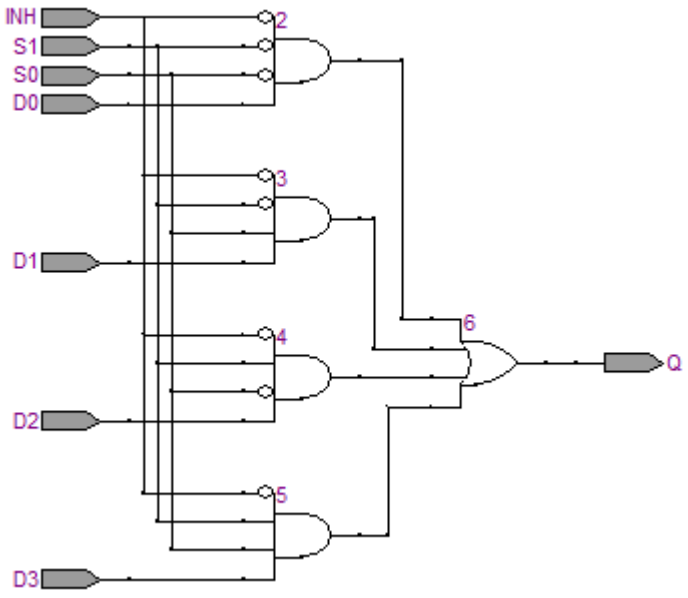
Приклад 3: Описати на Verilog мультиплексор чотирьохрозрядних шин 2:1

```
module mux2 (d0, d1, sel, y);
  input  [3:0] d0, d1;
  input  sel;
  output [3:0] y;
  assign y = sel ? d1 : d0;
endmodule
```



Приклад 4: Описати на Verilog мультиплексор 4:1

```
module mux41
(D0,D1,D2,D3,S0,S1,Q);
 input  D0, D1, D2, D3;
 input  S0, S1;
 output Q ;
 assign Q = S1?(S0?D3:D2)
              :(S0?D1:D0);
endmodule
```

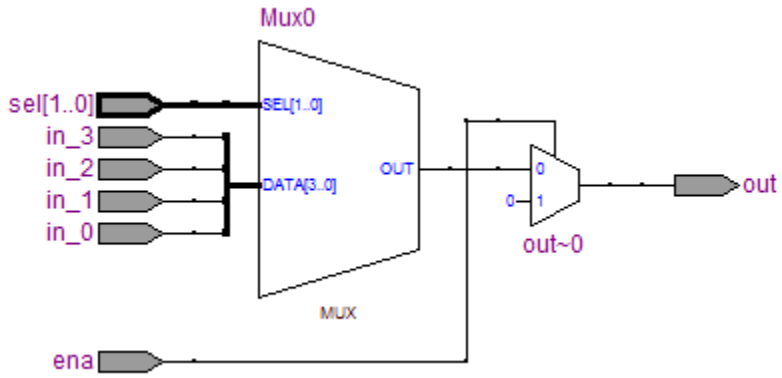


На рисунку приведена синтезована схема мультиплексора на два адресних входи.

Приклад 5: Описати на Verilog мультиплексор 4:1 з сигналом дозволу за допомогою процедурного блоку ***always***

.

```
module mux41 (out, in_3, in_2, in_1, in_0, sel, ena);
 output out;
 input  in_3, in_2, in_1, in_0;
 input  [1:0] sel;
 input  ena;
 reg    out;
 always @ (in_3 or in_2 or in_1 or in_0 or sel or ena)
  if(ena) out = 0;
  else begin
   case (sel)
   0: out = in_0;
   1: out = in_1;
   2: out = in_2;
   3: out = in_3;
   endcase
 end
endmodule
```

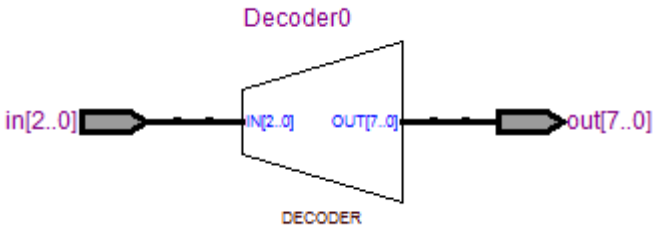


Для опису мультиплексора на 4 інформаційні входи використано конструкцію **case**. У разі її застосування слід описувати всі можливі комбінації варіантів, інакше під час синтезу у комбінаційну схемі буде введений елемент пам'яті (Latch). Іншим варіант – описати значення на виході схеми за замовчуванням (default). З синтезованої схеми видно, що функціонал дозволу роботи пристрою теж реалізовується у вигляді окремого мультиплексора

1.2.4 Опис дешифраторів

Приклад 6: Описати на Verilog дешифратор 3:8. У дешифраторі на виході використовується унітарний код, враховуючи чи це опис можливо виконати наступним чином:

```
module decoder(out,in);
 output reg [7:0]out;
 input      [2:0]in;
always @ (in)
 begin
   out = 8'h00;
   out[in] = 1'b1;
  end
endmodule
```

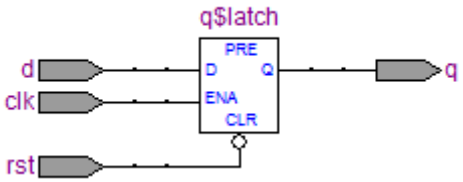


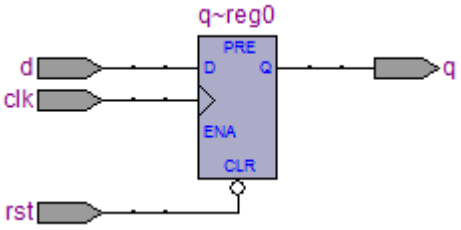
1.3 Опис послідовнісних синхронних цифрових пристроїв

Приклад 7: Описати на Verilog D фіксатор (Latch) та D-тригер, визначити різницю в описі та особливості, яких слід дотримуватись у разі опису тактованих фронтом тригерів

```
module DLatch (clk,rst,d,q);
 input clk, rst;
 input  d;
 output reg q;
always @ (clk or d)
 if (!rst)
   q = 1'b0;
 else if (clk)
   q = d;
endmodule
```

```
module DFF (clk,rst,d,q);
 input clk, rst;
 input  d;
 output reg q;
always @(posedge clk or negedge rst)
 if (!rst)
   q <= 1'b0;
 else if (clk)
   q <= d;
endmodule
```

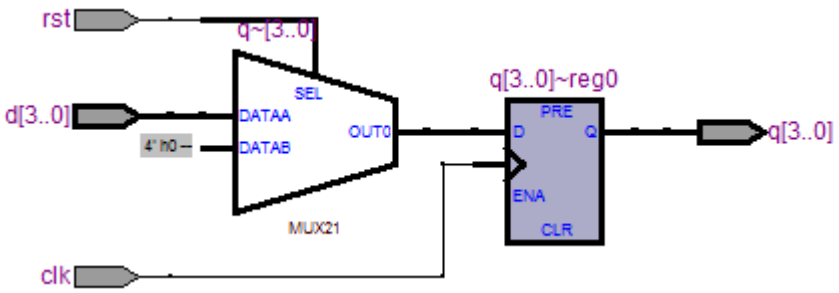




З порівняння описів приведених у таблиці, для того що був виконаний синтез тригера слід у списку чутливості процедурного блоку **always** вказувати тактовий сигнал з ключовим словом яке вказує фронт – наростаючий **posedge** чи спадаючий **negedge**. Окрім того для, того щоб тригер мав можливість асинхронного скидання у список чутливості також слід додати фронт сигналу скидання, що переводить його у активний стан. Інша особливість при описі тригера полягає у використання не блокуючого присвоєння <= , замість блокуючого =. Особливість не блокуючого присвоєння полягає у тому, що всі вирази, що знаходяться в основному блоці виконуються паралельно за фронтом, у той час як блокуючи присвоєння передбачає виконання виразів у послідовності їх запису.

Приклад 8: Описати на Verilog чотирьох розрядний регістр з синхронним скиданням.

```
module registr (clk, rst, d, q);
 input clk, rst;
 input [3:0] d;
 output reg [3:0] q;
always @ (posedge clk)
 if (rst)
   q <= 4'b0000;
 else
   q <= d;
endmodule
```



У разі синхронного скидання, сигнал rst не поміщується у таблицю чутливості. Внаслідок цього реакція та сигнал скидання буде виконана одразу після наступного фронту тактового сигналу. Під час синтезу для реалізації синхронного скидання перед регістром додатково буде ще використаний мкльтиплексор, що керується сигналом скидання.

Приклад 9: Описати на Verilog додавальний лічильник з модулем лічби M=9, стани від 0 до 9. У лічильнику має бути передбачена можливість асинхронного скидання.

```
module upcounter (clk, rst, q);
 input clk, rst;
 output reg [3:0] q;
assign compare = (q == 4'b1001);
always @ (posedge clk or negedge rst)
 if (!rst)
   q <= 4'b0000;
 else if (compare)
   q <= 4'b0000;
 else
   q <= q+1'b1;
endmodule
```

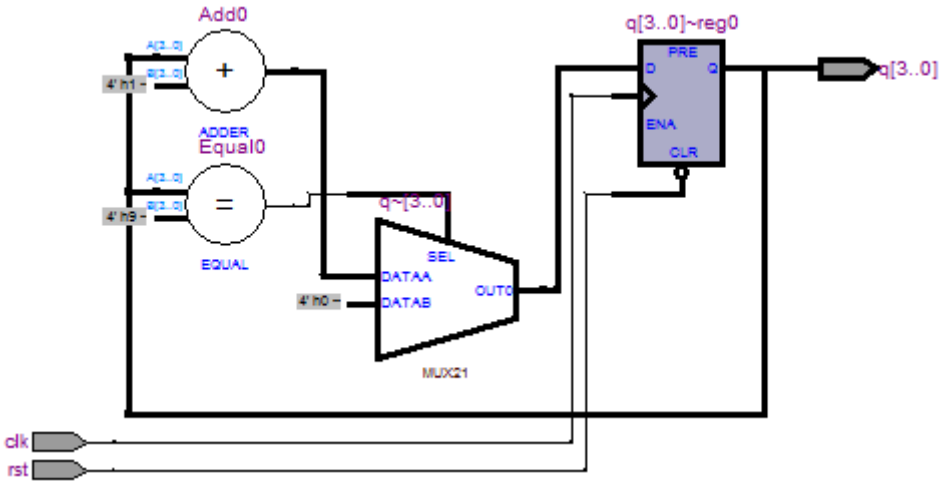
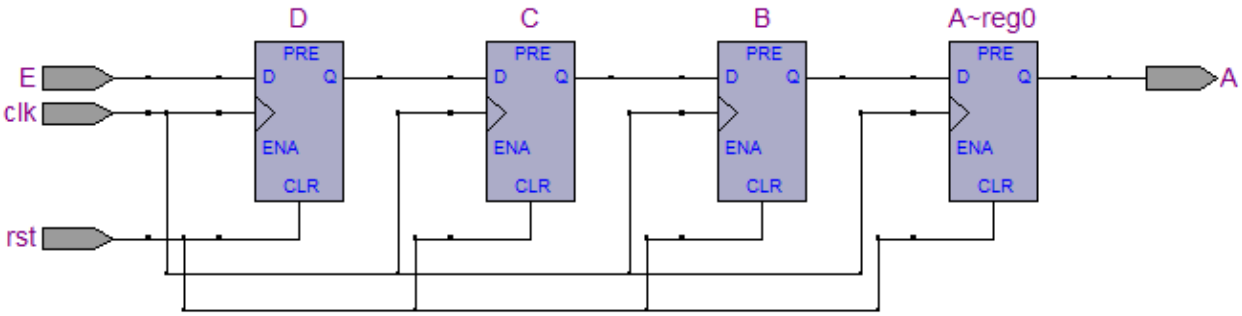


Схема лічильник, як добре видно з рисунку, складається з двох основних частин : регістру з чотирьох тригерів та комбінаційної схеми, яка складається з суматора та компаратора. При чому сигнал з компаратора є керуючим для мультиплексора, який і визначає в який наступний стан встановиться регістр.

Приклад 10: Описати на Verilog зсуваючий регістр з асинхронним активним високим рівнем скиданням .

```
module shifter (A, E, clk, rst);
 output A;
 input  E;
 input  clk, rst;
 reg    A, B, C, D;
always @ (posedge clk or posedge rst)
 if (rst) begin A <= 0; B <= 0; C <= 0; D <= 0; end
 else begin
   A <= B; B <= C; C <= D; D <= E;
 end
endmodule
```



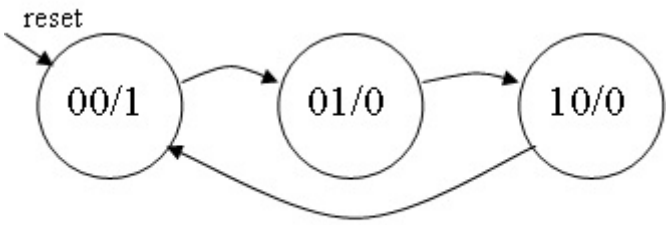
1.4 Опис цифрових автоматів Мілі та Мура

Кінцеві автомати є потужним шляхом системного проектування послідовнісних цифрових пристроїв на основі специфікації. У разі проектування КА доцільним є наступний алгоритм:

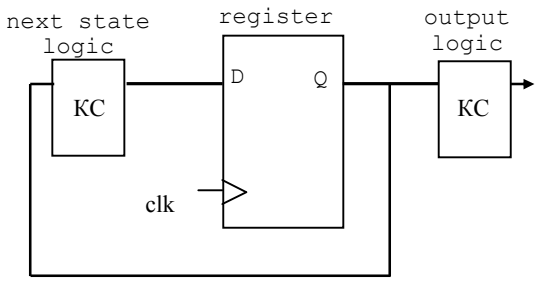
– визначити входи та виходи КА;
– зобразити діаграму перемикань;
– обрати кодування станів (від вибору може залежати складність проекту);
– описати регістр, що має мати достатньо кількість розрядів;
– описати комбінаційні схеми, що визначають наступний стан (*next state*) і вихідний сигнал (*output logic*).

Приклад 11: Описати на Verilog кінцевий автомат, що може забезпечити виділення тактової частоти діленої на 3.

Діаграма перемикань

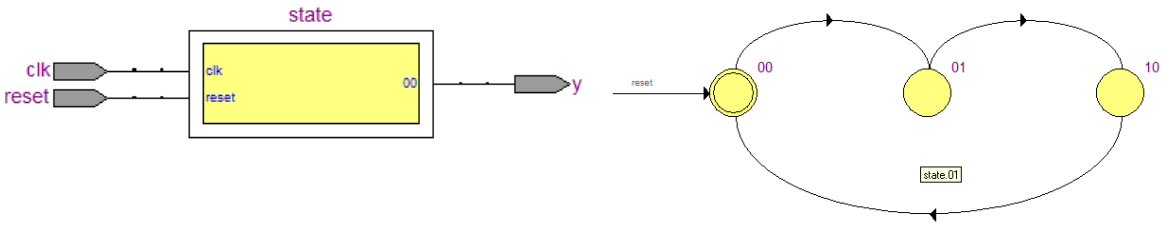


Загальна структурна схема




```
module FSM (clk, reset, y);
 input clk, reset;
 output y;
 reg [1:0] state, nextstate;
// register
always @ (posedge clk or posedge reset)
    if (reset)
      state <= 2'b00;
    else
      state <= nextstate;
// next state logic
always @ (state)
 case (state)
   2'b00:  nextstate = 2'b01;
   2'b01:  nextstate = 2'b10;
   2'b10:  nextstate = 2'b00;
   default: nextstate = 2'b00;
 endcase
// output logic
assign y = (state == 2'b00);
endmodule
```

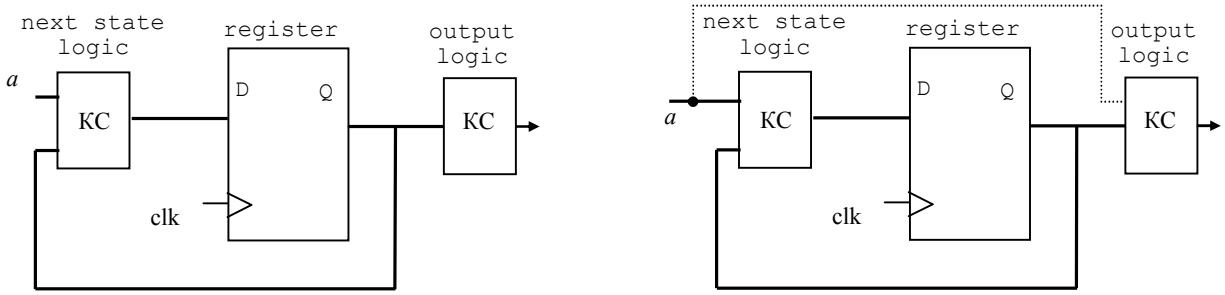
Результат синтезу на RTL рівні відображається у вигляді одного блоку, функціонування якого також подається увигляді діаграми перемикань



Приклад 12: Описати на Verilog кінцевий автомат, що може забезпечити виявлення у потоці кодової послідовності 1 1 0 1. Оцінити різницю між автоматом Мура та Мілі

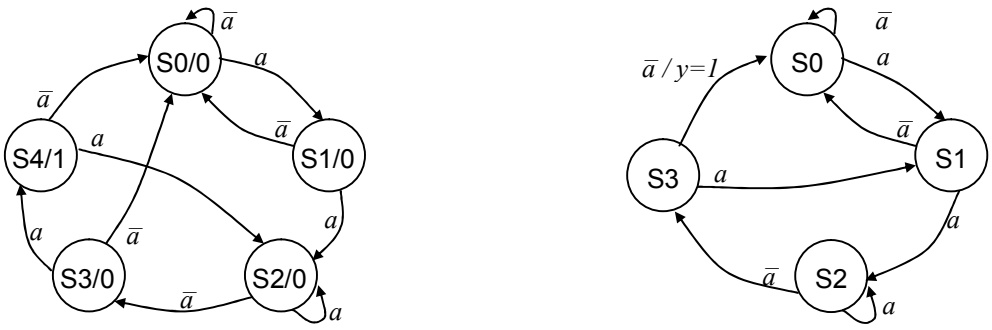
Відмінність між автоматами Мілі та Мура зобразити на структурних схемах

КА Мура КА Мілі



Приведемо діаграми перемикань для двох типів кінцевих автоматів. Як видно з діаграм КА Мура має на один стан більше, що у нашому випадку вимагатиме використання регістра з трьох тригерів, у той час як КА Мілі буде мати регістр з двох тригерів. Проте, у КА Мілі буде складнішою комбінаційна схема, що формує вихідний сигнал.

КА Мура КА Мілі



Опис КА Мура та Мілі на Verilog приведений у таблиці з можливістю порівняння:

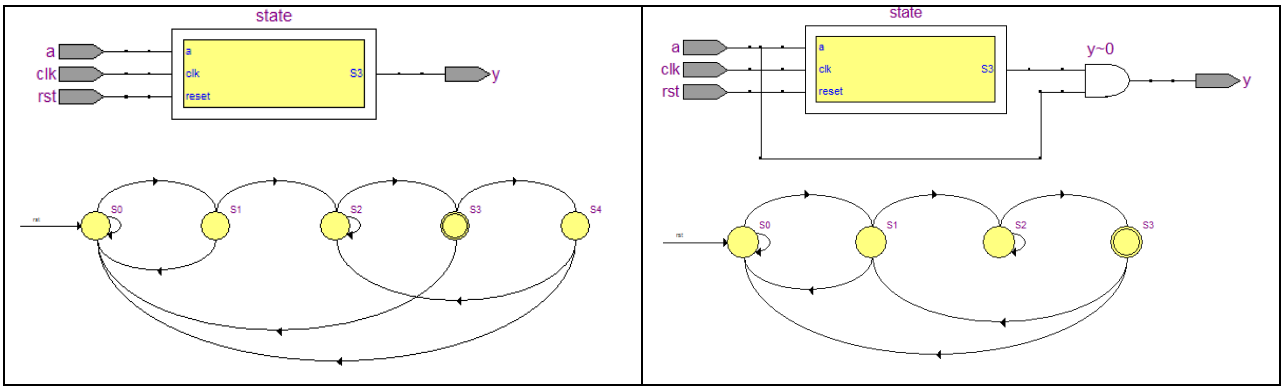
```
module Moore (clk, rst, a, y);
 input clk, rst, a;
 output y;
 reg [2:0] state, nextstate;
 parameter S0 = 3'b000;
 parameter S1 = 3'b001;
 parameter S2 = 3'b010;
 parameter S3 = 3'b011;
 parameter S4 = 3'b100;
// register
always @(posedge clk or posedge rst)
   if (rst)
     state <= S0;
   else
     state <= nextstate;
// next state logic
always @ (state)
 case (state)
  S0: if(a) nextstate = S1;
      else  nextstate = S0;
  S1: if(a) nextstate = S2;
      else  nextstate = S0;
  S2: if(a) nextstate = S2;
      else  nextstate = S3;
  S3: if(a) nextstate = S4;
      else  nextstate = S0;
  S4: if(a) nextstate = S2;
      else  nextstate = S0;
  default:  nextstate = S0;
 endcase
// output logic
assign y = (state == S4);
endmodule
```

```
module Mealy (clk, rst, a, y);
 input clk, rst, a;
 output y;
 reg [1:0] state, nextstate;
 parameter S0 = 2'b00;
 parameter S1 = 2'b01;
 parameter S2 = 2'b10;
 parameter S3 = 2'b11;

// register
always @(posedge clk or posedge rst)
   if (rst)
     state <= S0;
   else
     state <= nextstate;
// next state logic
always @ (state)
 case (state)
  S0: if(a) nextstate = S1;
      else  nextstate = S0;
  S1: if(a) nextstate = S2;
      else  nextstate = S0;
  S2: if(a) nextstate = S2;
      else  nextstate = S3;
  S3: if(a) nextstate = S1;
      else  nextstate = S0;
  default:  nextstate = S0;
 endcase


// output logic
assign y = (a & state == S3);
endmodule
```

Результат синтезу КА Мура та Мілі:



1.5 Опис ієрархічних проектів цифрових пристроїв

У разі побудови складних проектів рекомендується дотримуватись концепції модульності та повторюваності. Модульність передбачає побудову проекту за ієрархічною структурою. Повторюваність – створення універсальних модулів і широке їх використання у всіх частинах проекту. У цьому разі виникає потреба з'єднувати модулі між собою через описані в них інтерфейси. З'єднувати модулі або примітиви між собою можливо двома способами – за позицією та за назвами портів. У першому варіанті запис є коротшим, але у випадку коли портів багато, то легше і надійніше використовувати другий спосіб, зокрема $.formal\_name(actual\_name)$, де *formal_name* – назва порта екземпляру модуля до якого слід приєднатися, *actual_name* – назва змінної у модулі вищого рівня для з'єднання.

Приклад 13: Описати на Verilog мультиплексор 4:1 структурним стилем на основі мультиплексорів 2:1.

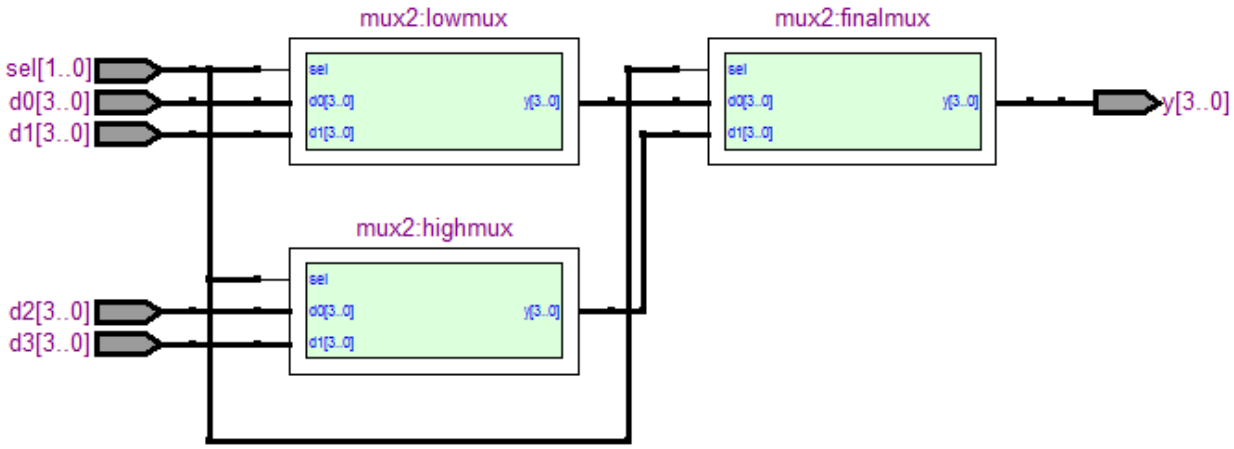
Опис мультиплексора на два ынформаційні входи вже демонструвався у п. 1.2.3, але оскільки у нашому прикладі ми будемо створювати його екземпляри, то приведемо його зному:

```
module mux2 (d0, d1, sel, y);
  input  [3:0] d0, d1;
  input  sel;
  output [3:0] y;
  assign y = sel ? d1 : d0;
endmodule
```

У модулі **mux4** створено три екземпляри модуля **mux2** з назвами **lowmux** , **highmux** та **finalmux**. До портів перших двох екземплярів приєднання виконано за позицією, а в останньому – за назвами портів.

```
module mux4 (d0, d1, d2, d3, sel, y);
  input  [3:0] d0, d1, d2, d3;
  input  [1:0] sel;
  output [3:0] y;
  wire [3:0] low, high;
mux2  lowmux  (d0, d1, sel[0], low);
mux2  highmux (d2, d3, sel[0], high);
mux2  finalmux(.d0(low), .d1(high), .sel(sel[1]), .y(y));
endmodule
```

Результат синтезу мультиплексора 4:1 структурним стилем:



Приклад 14: Описати на Verilog 4-х розрядний двійковий суматор на основі одно розрядного напівсуматора структурним стилем.

```
module add4 (a, b, cin, sum, cout);
  input  [3:0] a, b;
  input  cin;
  output [3:0] sum;
  output cout;
  wire c0, c1, c2;
  fulladder fadd0 (a[0], b[0], cin, sum[0], c0);
  fulladder fadd1 (a[1], b[1], c0, sum[1], c1);
  fulladder fadd2 (.a(a[2]), .b(b[2]), .cin(c1), .sum(sum[2]), .cout(c2));
  fulladder fadd3 (.a(a[3]), .b(b[3]), .cin(c2), .sum(sum[3]), .cout(cout));
endmodule
```

```
module adder (s, cout, a, b);
  input    a, b;
  output  s, cout;
  xor  (s, a, b);
  and  (cout, a, b);
endmodule
```

```
module fulladder (a, b, cin, sum, cout);
  input  a, b, cin;
  output sum, cout;
  wire  w1, w2, w3;
  adder M0 (w1, w2, a, b);
  adder M1 (sum, w3, w1, cin);
  or  M2 (cout, w2, w3);
endmodule
```

Частина 2
СИМУЛЯЦІЯ ТА ТЕСТУВАННЯ МОДУЛІВ ЦИФРОВИХ ПРИСТРОЇВ

2.1 Методологія тестування модулів

З ускладненням цифрових пристроїв вкрай важливою стає перевірка коректності їх функціональності. Мови опису апаратури Verilog та VHDL набули широкого поширення саме через їх зручність як для проектування, так і для тестування проектів для ПЛІС та ASIC.

Тестування та симуляція здійснюються з метою перевірки відповідності опису моделі цифрового пристрою на Verilog його специфікації. Для цього використовуються наступні методи:

1. Симуляція логіки (logic simulation);
2. Формальна верифікація (formal verification).

У першому способі застосовується набір стимулів до схеми і перевіряється реакція на них на виході, на основі чого і приймається рішення стосовно коректності синтезу. У другому випадку застосовується детальне математичне доведення коректності роботи, але у разі складних схем така симуляція стає майже неможливою. Тому, зазвичай, використовують симуляцію логіки.

Слід зазначити, що для того щоб бути впевненим у тому, що логіка повністю перевірена і коректно функціонує, великі та складні схеми необхідно тестувати і перевіряти системно. Безпланова методика тестування призводить до важкого процесу ладнання (debuging) та створює хибне відчуття того, що описана модель цифрового пристрою є коректною, призводить до невиправданого ризику збоїв як результату нетестованих частин логіки. Тому на практиці команда розробників детально описує план тестування, визначає особливості і сам процес тестування.

2.2 Конструкції та диретиви Verilog, що використовуються для симуляції

Стимулятор в Verilog - це частина програмного забезпечення, що

дозволяє виконувати вирази в блоках **initial та always.** В симуляторі можливо вказати послідовність подій і точно визначати стимули сигналів у часі. Конструкції та блоки, що використовуються у Verilog для симуляції, не синтезуються. Приведемо приклад модуля з блоком тестування.

```
`timescale 1ns/100ps;
module testbench ();
     wire y, c_out;
     reg  c_in, a, b;

 SM  SM_inst (.y(y), .c_out(c_out), .a(a), .b(b), .c_in(c_in));
 initial
 begin
    #0  a = 0; b = 0 ; c_in = 0;
    #50 a = 0; b = 0 ; c_in = 1;
    #50 a = 0; b = 1 ; c_in = 0;
    #50 a = 1; b = 1 ; c_in = 1;
    #50 a = 1; b = 0 ; c_in = 0;
    #50 a = 1; b = 0 ; c_in = 1;
    #50 a = 1; b = 1 ; c_in = 0;
    #50 a = 1; b = 1 ; c_in = 1;
 end
endmodule
```

2.3 Модулі автоматичного тестування (TestBench)

2.3.1 Загальна структурна схема

Для тестування на VerilogHDL слід створити окремий модуль, що як правило, називається testbench (дослівно тестовий стенд). У testbench можна задавати вектора стимулів до спроектованої цифрової схеми та аналізувати на її виході сигнали на відповідність логічній функції за якою і реалізовано конкретний цифровий пристрій.

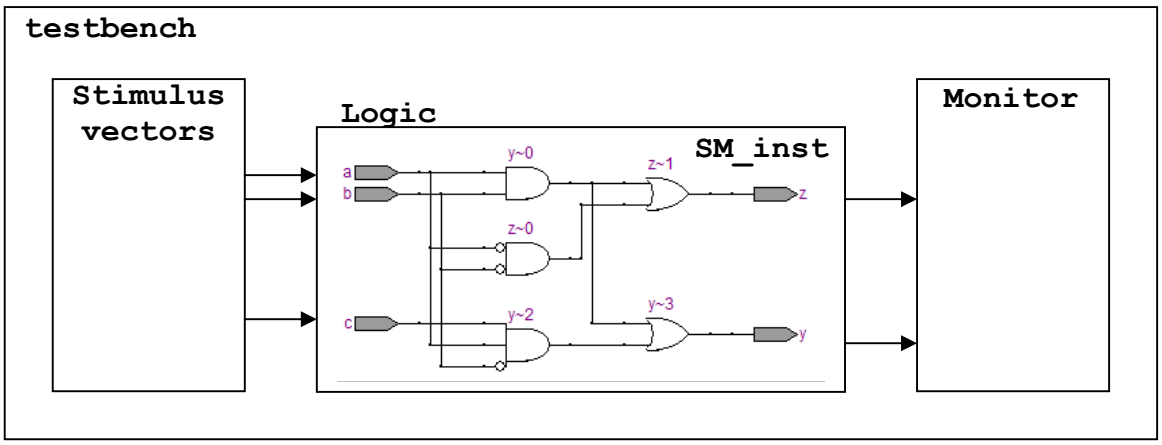


Рис. 2.1 – Загальна структурна схема тестового модуля

У приведеному вище прикладі ініціалізується модуль верхнього рівня з назвою **testbench**. Системна директива **`timescale** дозволяє задати базову одинцю часу з якою далі можна оперувати за допомогою директив #n, де n задає затримку (у нашому випадку в нс). Далі приєднується модуль, що необхідно протестувати, у нашому разі зразок (instance) модуля однорозрядного суматора **SM** Програмне забезпечення Quartus дозволяє автоматично створювати зразки модулів за допомогою команди в меню: File > Create / Update > Create Verilog Instantiated Files for Current File.

2.3.2 Вивід інформації у консоль під час симуляції

Вивід даних у консоль дозволяє полегшувати пошук помилок у описі цифрових пристроїв, а також повідомляти про поточний стан симуляції. Розглянемо детальніше дві найбільш поширені команди, що використовуються для виводу даних у консоль. Нижче наведені найбільш поширені директиви симулятору:

Директива	Опис
$monitor	дозволяє вести спостереження за конкретними змінними або сигнали під час симуляції. У разі зміни значення одного з сигналів буде ініціюватися вивід повідомлення у консоль у заданому форматі. Паралельно у часі може діяти лише одна директива **$monitor.** Є потужним засобом симуляції та лагодження.
$display	використовується для виводу інформації в консоль з додаванням символів переходу на нову стрічку в кінці. У вивід можливо включати змінні, які підставляються у шаблон у місця символів: %b – двійковий формат, %h – шістнадцяткий, %d – десяткового.
$time	дозволяє виводити у консоль поточний час симуляції, часто використовується разом з директивою **$display**
$finish	завершення процедури симуляції
$readmemb	зчитування даних з файлу у бінарному форматі
$readmemh	зчитування даних з файлу у шістнадцятковому форматі

Приклад використання директиви $display:

```
$display($time,"<<count = %d enable >>", cnt_out);
```

Всі символи, що знаходяться між лапками будуть виводитись у консоль з поверненням каретки вкінці. Значення cnt_out замінить %d буде виведено у десятковому форматі, в той час як поточний час моделювання буде виведений попереду повідомлення.

2.3.3 Тестовий модуль з автоматичним аналізом стимулів

Приклад реалізації автоматичного тестування модуля **SM_inst**.

```
`timescale 1ns/100ps;
module testbench ();
     wire y, c_out, y_ext;
     reg  c_in, a, b;

SM  SM_inst (.y(y), .c_out(c_out), .a(a), .b(b), .c_in(c_in));
 initial
 begin
 $monitor ($time,"a = %b, b = %b, c_in =%b, y = %d, c_out = %b",
a , b, c_in, y, c_out);

    #0 a = 0; b = 0 ; c_in = 0;
    #50 if((y!=0) | (c_out!=0))   $display("000 test failet");

    #50 a = 1; b = 0 ; c_in = 0;
    #50 if((y!=1) | (c_out!=0))   $display("001 test failet");

    #50 a = 0; b = 1 ; c_in = 0;
    #50 if((y!=1) | (c_out!=0))   $display("010 test failet");

    #50 a = 1; b = 1 ; c_in = 0;
    #50 if((y!=0) | (c_out!=1))   $display("011 test failet");

    #50 a = 0; b = 0 ; c_in = 1;
    #50 if((y!=1) | (c_out!=0))   $display("100 test failet");

    #50 a = 1; b = 0 ; c_in = 1;
    #50 if((y!=0) | (c_out!=1))   $display("101 test failet");

    #50 a = 0; b = 1 ; c_in = 1;
    #50 if((y!=0) | (c_out!=1))   $display("110 test failet");

    #50 a = 1; b = 1 ; c_in = 1;
    #50 if((y!=1) | (c_out!=0))   $display("111 test failet");
    #50 $finish;
 end
endmodule
```

2.3.4 Універсальний тестовий модуль

У попереднього прикладу для проведення автоматичного аналізу необхідним є громіздке прописування фактично таблички істинності синтезованого модуля. Чим більше комбінацій тестових векторів, тим складнішою стає і автоматичний аналіз. Тому доцільним є винесення усіх тестових векторів у окремий файл, з якого тестовий модуль почергово буде зчитувати тестові вектори і перевіряти коректність реакції на них.

```
module testbench();
wire y,c_out;
reg a,b,c_in;
reg clk, rst;
reg [4:0] testvector [8:0];
reg [3:0] errors, testcomb;
reg y_exp, c_out_exp;

fulladder add (.sum(y),.cout(c_out),.a(a),.b(b),.cin(c_in));

always begin
    clk = 1; #50 clk = 0;  #50;
end

initial begin
     $readmemb("vectors.tv", testvector);
     errors = 0 ; testcomb = 0;
     rst = 1; #10 rst = 0;
end

always @(posedge clk) begin
     #1 {a,b,c_in,y_exp,c_out_exp} = testvector[testcomb];
end

always @(negedge clk)
 begin
   if(~rst) begin
     if((y != y_exp) | (c_out != c_out_exp)) begin
       $display("Error: input =%b outputs = %b (%b expected)",
       {a,b,c_in}, {y, c_out}, {y_exp, c_out_exp});
          errors = errors + 1;
     end
     testcomb = testcomb + 1;
     if(testvector[testcomb] === 5'bx) begin
       $display("Perebrano %d kombinaciy ta znaydeno %d pomylok",
       testcomb, errors);
       $finish;
     end
   end
 end
endmodule
```

На початку модуля формується тактовий сигнал з періодом в 100 нс. використовуючи процедурний блок **always**, далі з файлу **vectors.tv** на початку симуляції зчитуються тестові вектори та значення сигналів, що очікуються на виході, у двійковому коді. Наприклад:

```
000_00
001_10
010_10
011_01
100_10
101_01
110_01
111_11
```

Директива **$readmemb** зчитує значення файлу **"vectors.tv"** в масив **testvector**. Наступний два блоки програми симуляції за наростаючим фронтом тактового сигналу застосовує стимули на входи модуля, що тестується, а по спадаючому фронту перевіряє реакцію на виході з очікуваним значенням. У разі якщо **y** не співпадає з очікуваним **y_exp**, то формується повідомлення у консоль у форматі **"Error: input =%b outputs = %b (%b expected)"**, де замість **%b** будуть підставлятись вхідний стимул, отриманий сигнал на виході та очікуваний. Фігурні дужки **{a,b,c_in}** виконують функцію об'єднання (*concatenation*) окремих сигналів у шину. Опісля досягнення кінця масиву тестових векторів у консоль буде виведений звіт про перебрану кількість комбінацій тестових векторів та кількість помилок.

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. Угрюмов Е. П. Цифровая схемотехника / Е. П. Угрюмов. – СПб. : БХВ-Петербург, 2004. – 528 с. – Библиогр. : ISBN 5-8206-0100-9.

2. Рябенький В. М. VERILOG. Практика проектування цифрових пристоїв на ПЛІС : Навч. посіб. / В.М. Рябенький, О.О. Ушкаренко ; Нац. ун-т кораблебудування ім. адм. Макарова. – Миколаїв : Іліон, 2007. – 324 с.

3. Поляков А. К. Языки VHDL и VERILOG в проектировании цифровой аппаратуры / А.К. Поляков. – М. : СОЛОН-Пресс, 2003. – 320 с. – Библиогр. : ISBN 5-08003-016-6.

4. Точчи Р. Д. Цифровые системы. Теория и практика / Р. Д. Точчи, Н.С.Уидмер ; 8-е узд. ; пер. с англ.. – М. : Издательский дом «Вильямс», 2004. – 1024 с. – Библиогр. : ISBN 5-8459-0586-9.

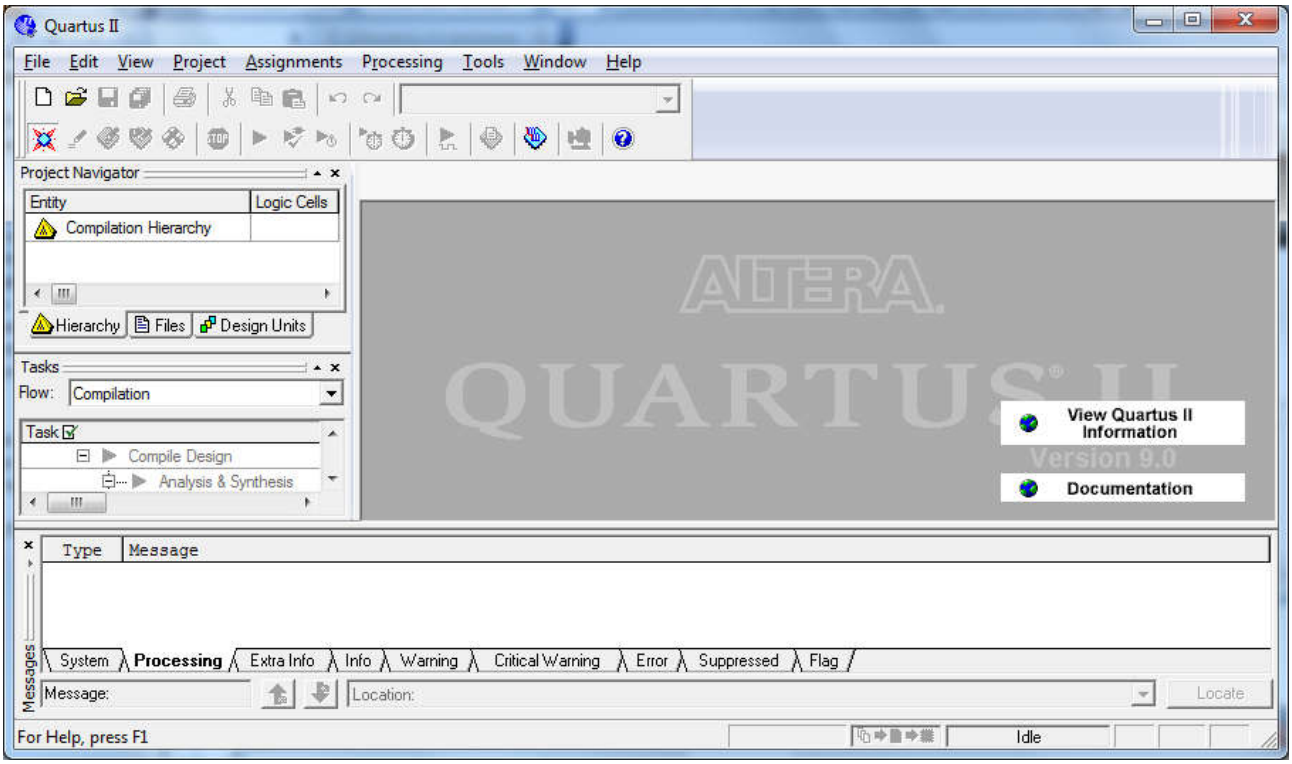
5. Harris D. M. Digital Design and Computer Architecture / D.M. Harris , S. L. Harris. ; Sec. Ed. – Morgan Kaufmann, 2013. – 560 c. – ISBN 978-0-12-394424-5.

6. Ciletti M. D. Advanced Digital Design with the Verilog HDL / M.D. Ciletti. – Prentice Hall. – 982 p. – ISBN 978-0-13-089161-7.

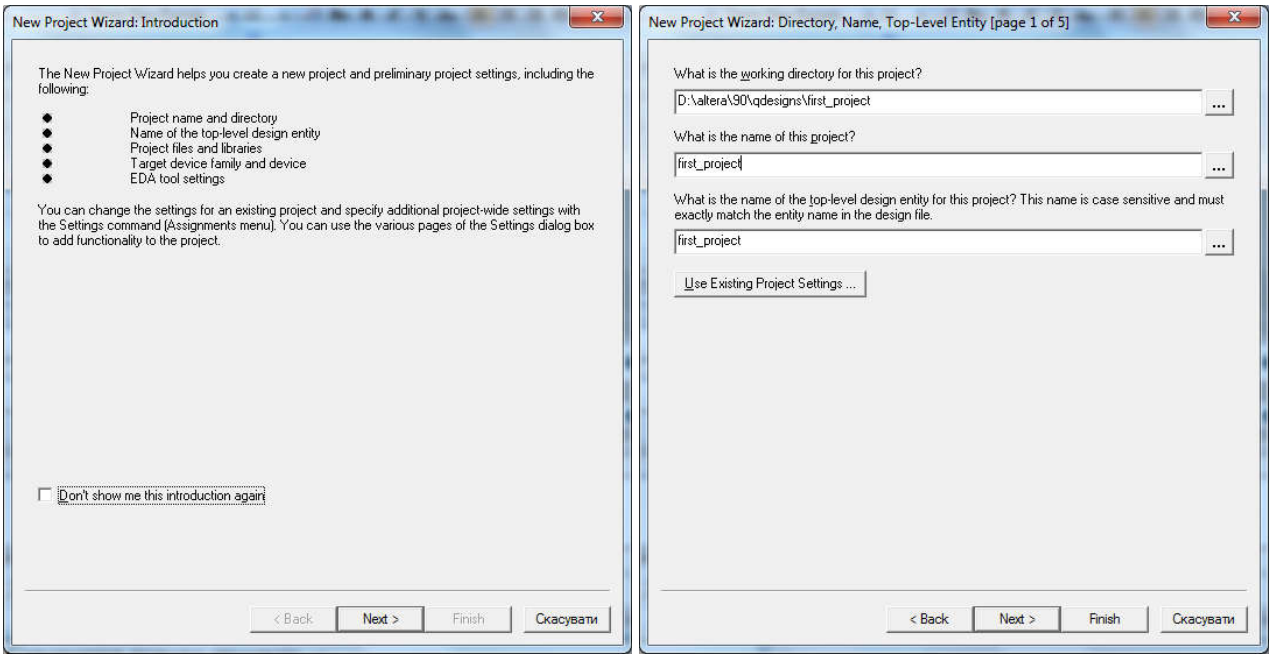
ДОДАТКИ

Додаток А
СТВОРЕННЯ ПРОЕКТУ У САПР Quartus II

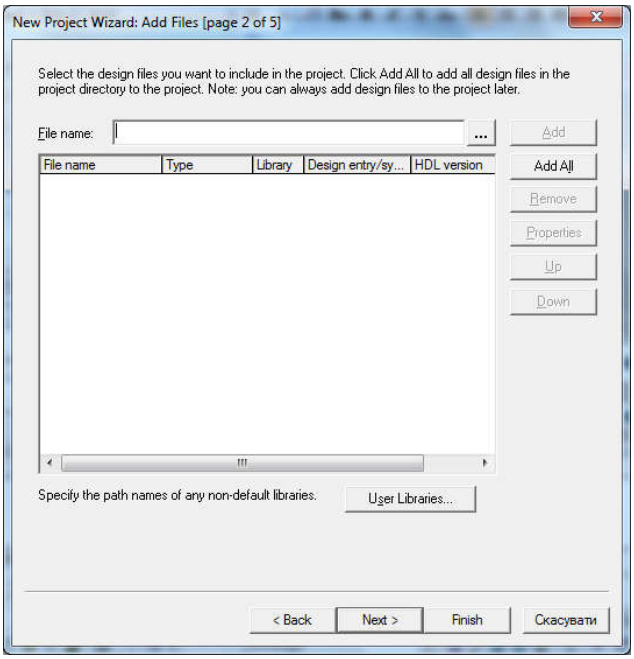
1. Загальний вигляд та інтерфейс Quartus II

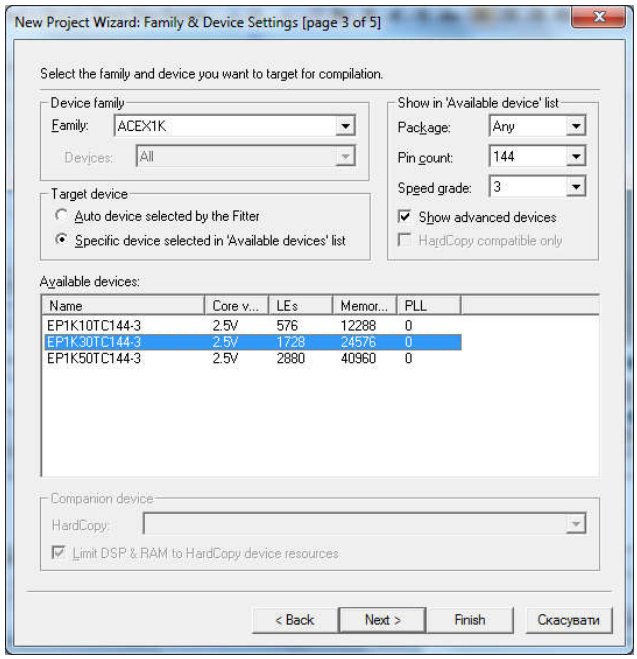


2. Створення нового проекту.
У меню “File” слід обрати “New Project Wizard” та слідувати інструкціям.

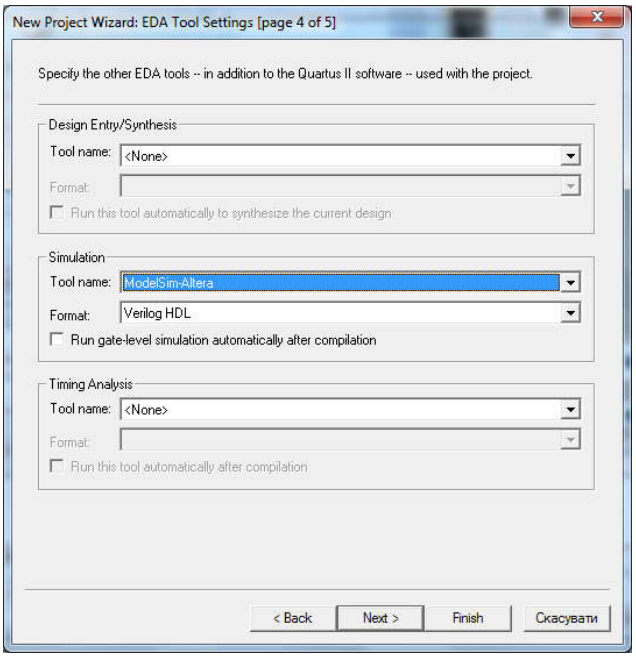


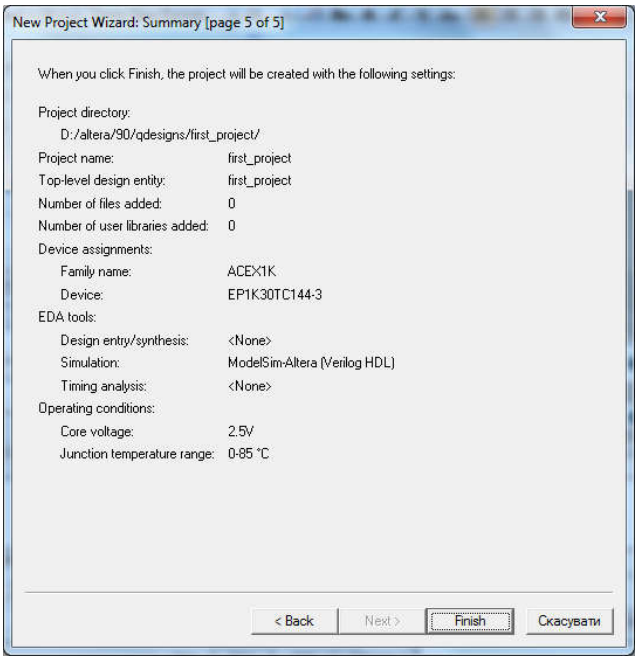
3. На кроці 1 слід вказати директорію для збереження проекту та його назву. На кроці 2 можливо у проект додати файли, якщо такі вже перед цим були створені. На кроці 3 слід обрати сімейство та модель ПЛІС.



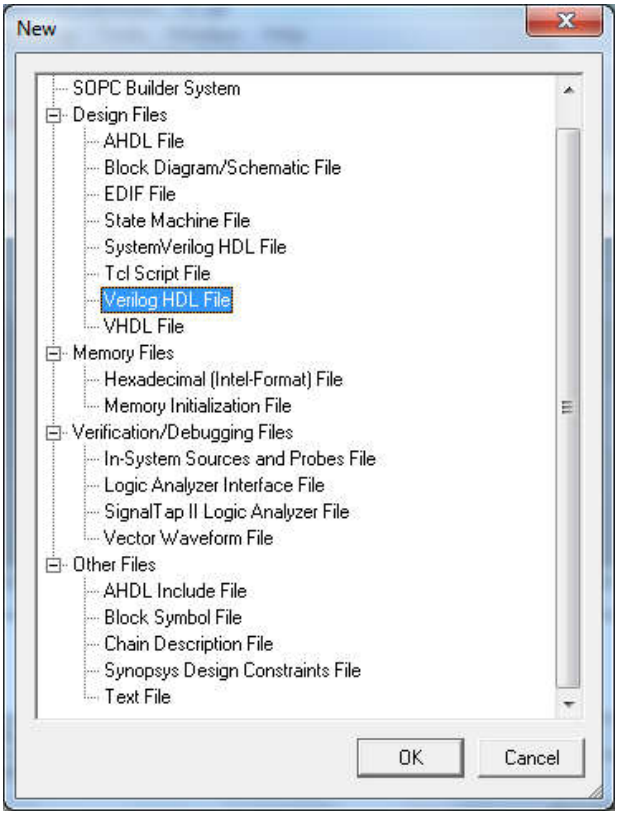


4. На кроці 4 вказуємо засоби третіх розробників у разі потреби. Ми обираємо для симуляції “ModelSIM-Altera”. На кроці 5 можливо перевірити всі попередні налаштування.





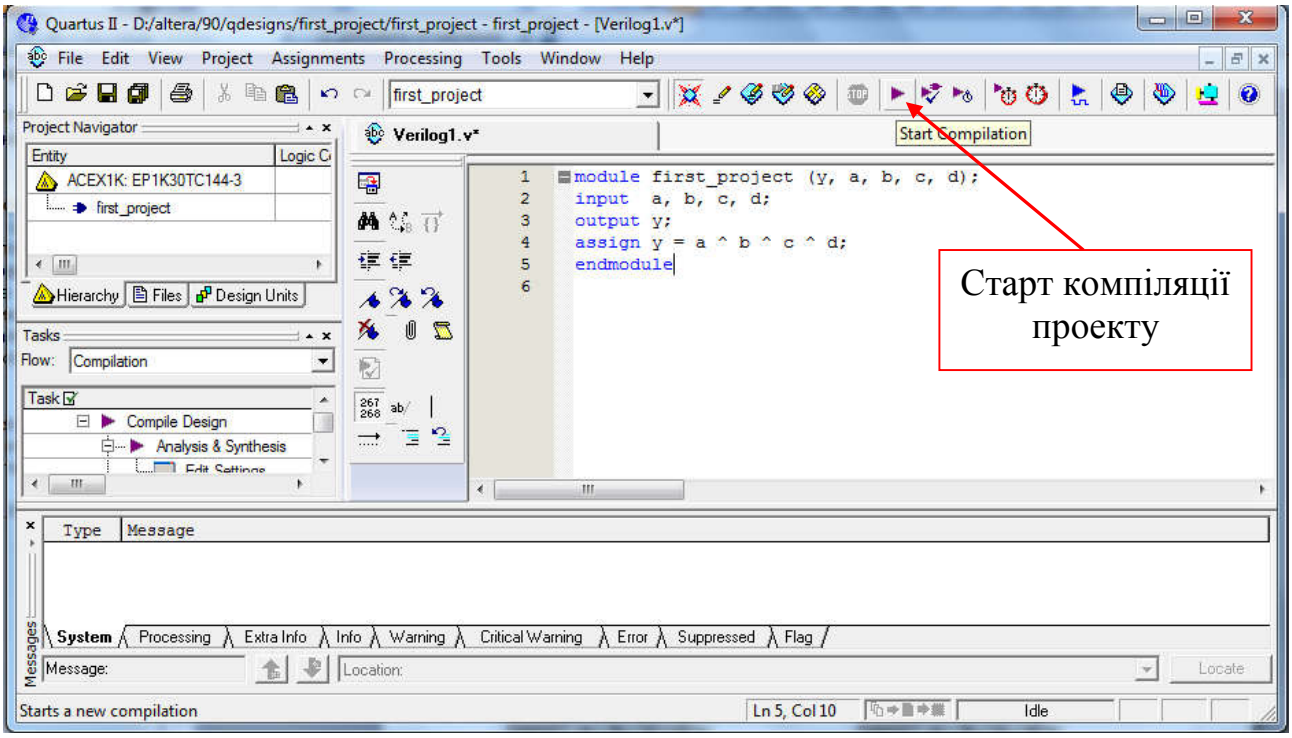
5. Опісля створення проекту додаємо до нього новий файл з описом комбінаційної схеми. В меню ”File -> New-> Verilog HDL File” .



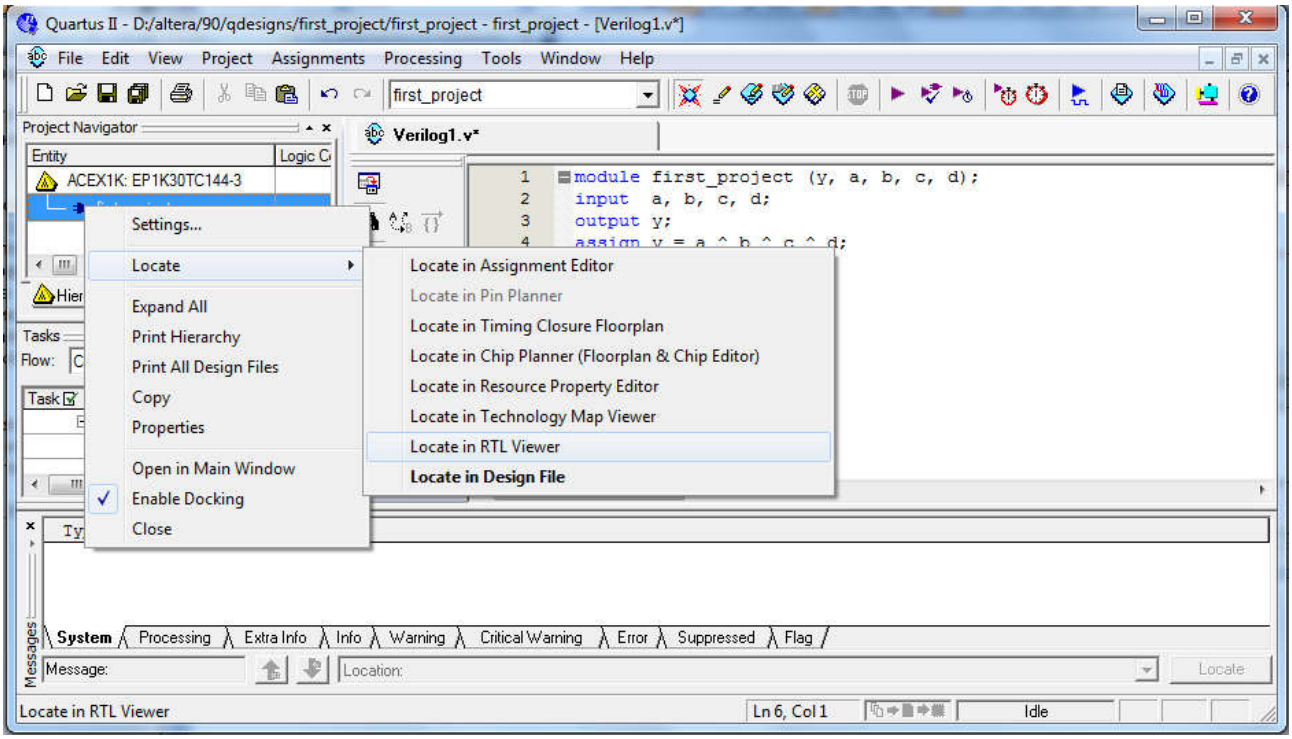
Опис комбінаційної схеми

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

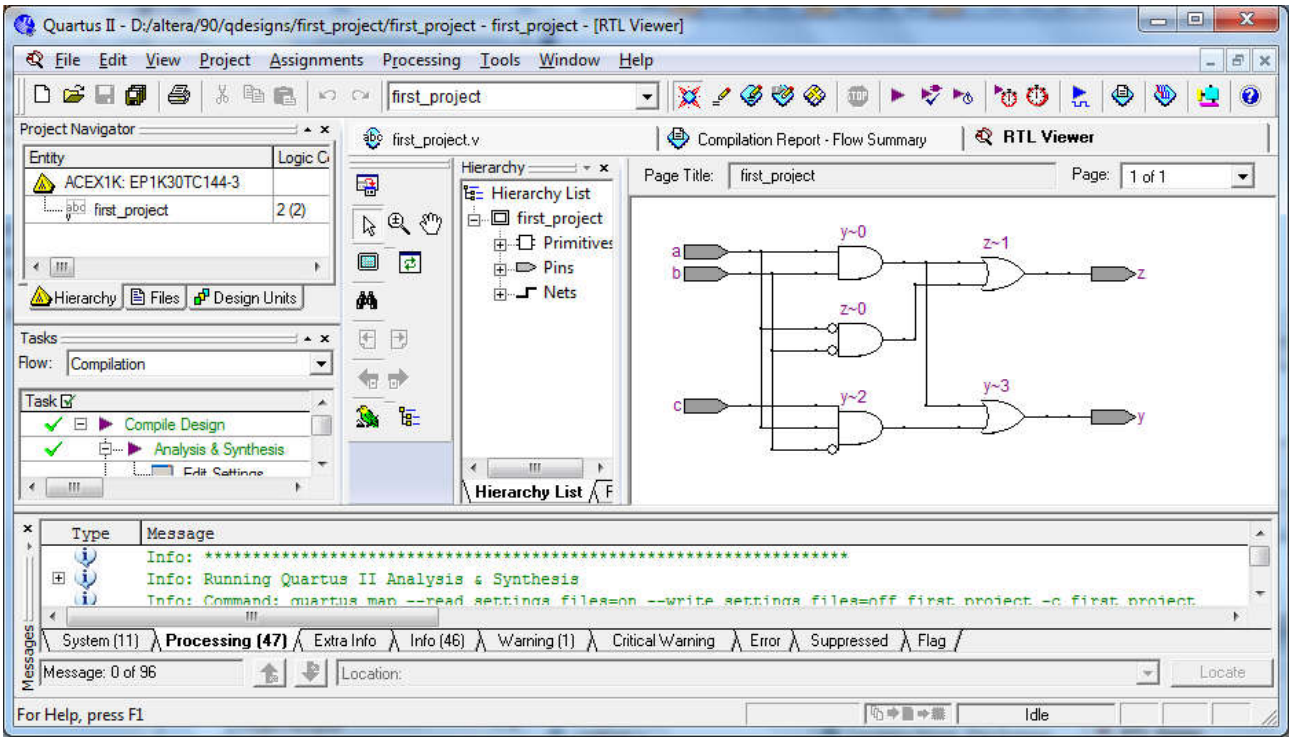
6. Опісля того, як опис введено можливо розпочати процес компіляції та синтезу. Проект та модуль верхнього рівня мають мати однакові назви.



7. У разі успішного компіляції та синтезу проекту можливо переглянути схему синтезованого цифрового пристрою на RTL рівні. Для цього на назві проекту слід правою клавішею миші відкрити контекстне меню, далі ”Locate -> Locate in RTL Viewer ”.



8. Схема синтезованого цифрового пристрою на RTL рівні.



Додаток Б
ТЕСТУВАННЯ МОДУЛІВ ЦИФРОВИХ ПРИСТРОЇВ В ModelSIM

У **Quartus** відсутня можливість проведення тестування за допоогою VerilogHDL, проте наявна можливість інтеграції з спеціалізованим програмним забезпеченням, таким як, ModelSIM. Для його використання необхідно виконати наступні налаштування. У меню (рис. В.1) **`Assigments > EDA Tool Settings > Simulation`** у вікні **`Settings`** (рис. В.2) необхідно вказати ModelSim Altera, мову опису цифрових пристроїв VerilogHDL та файл testbench.

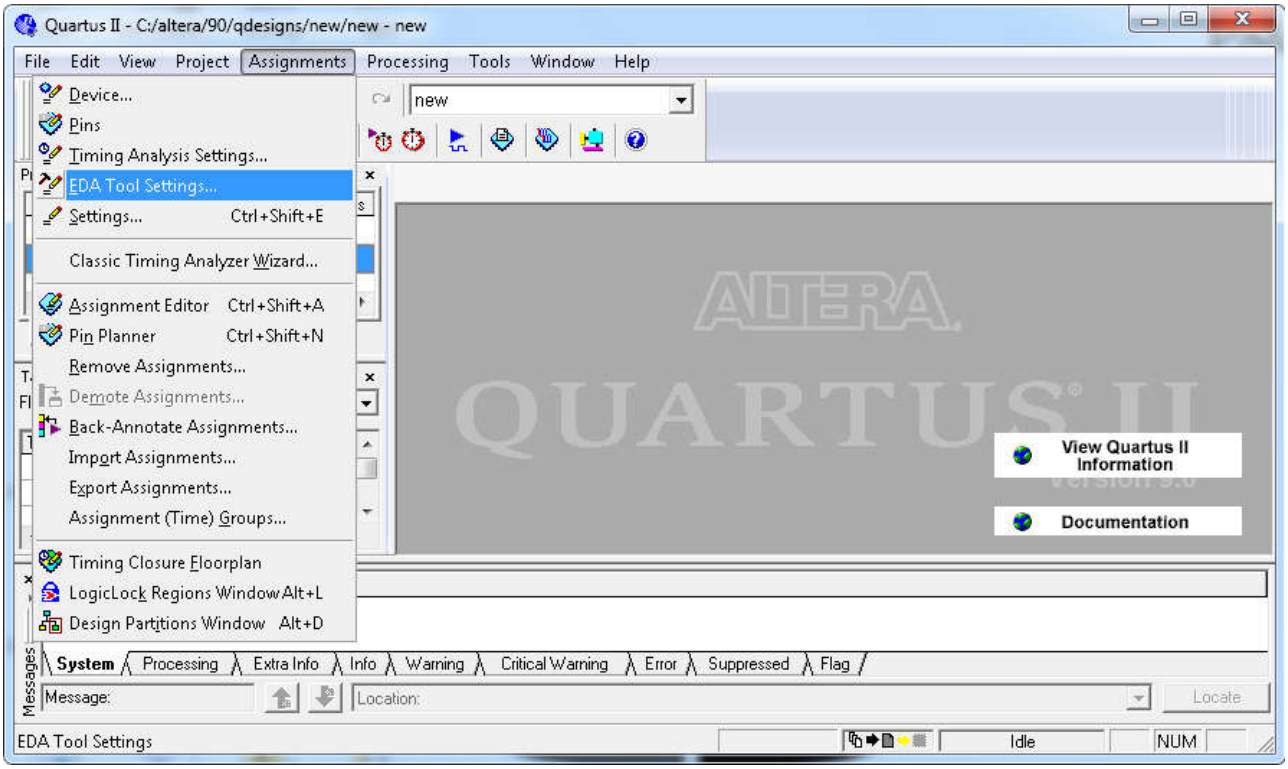


Рис. В.1 – Шлях до вікна з налаштуванням симуляції

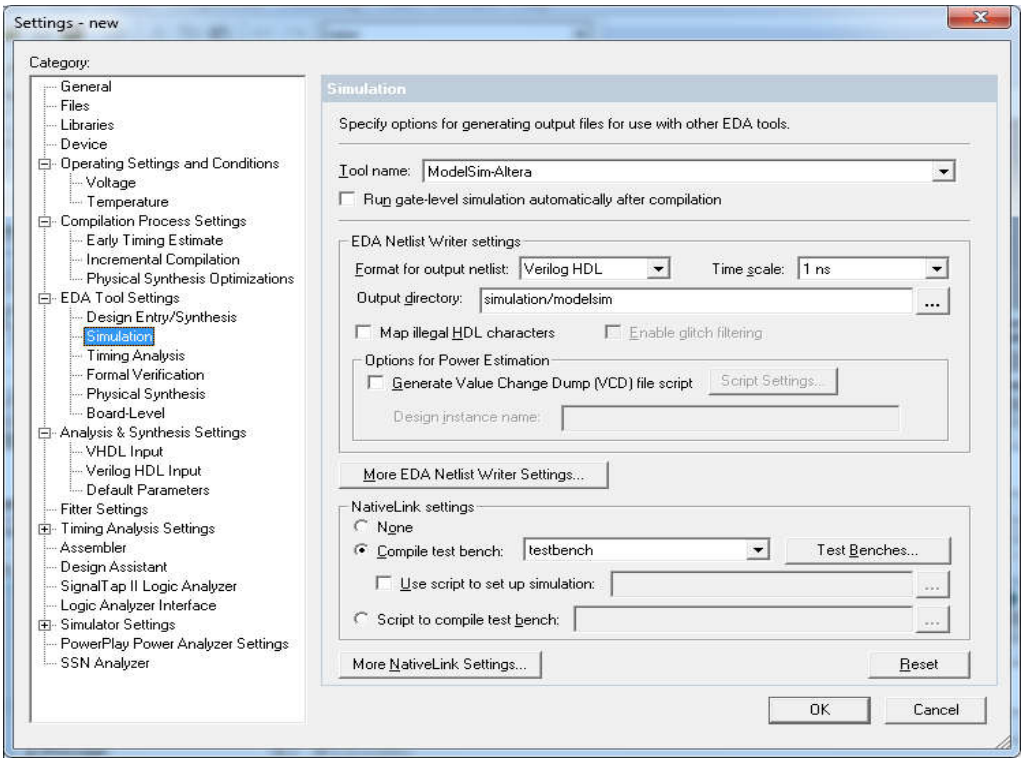


Рис. В.2 – Вибір засобів симуляції та їх налаштування

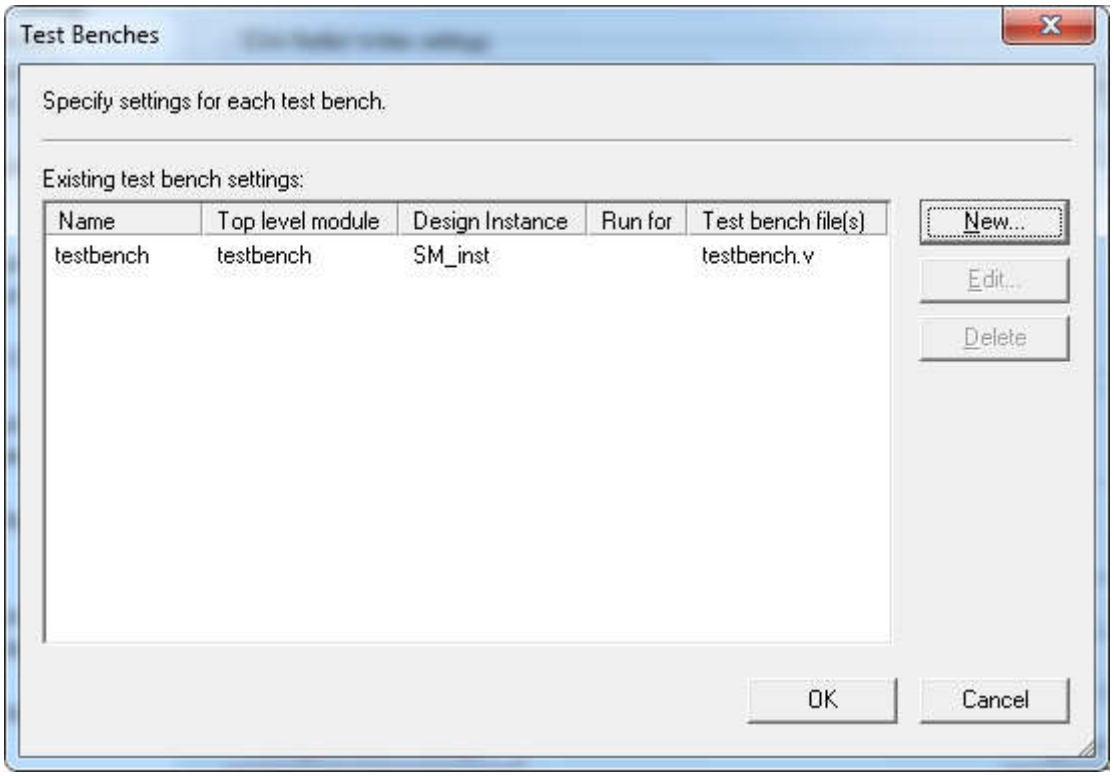


Рис. В.3 – Налаштування testbench

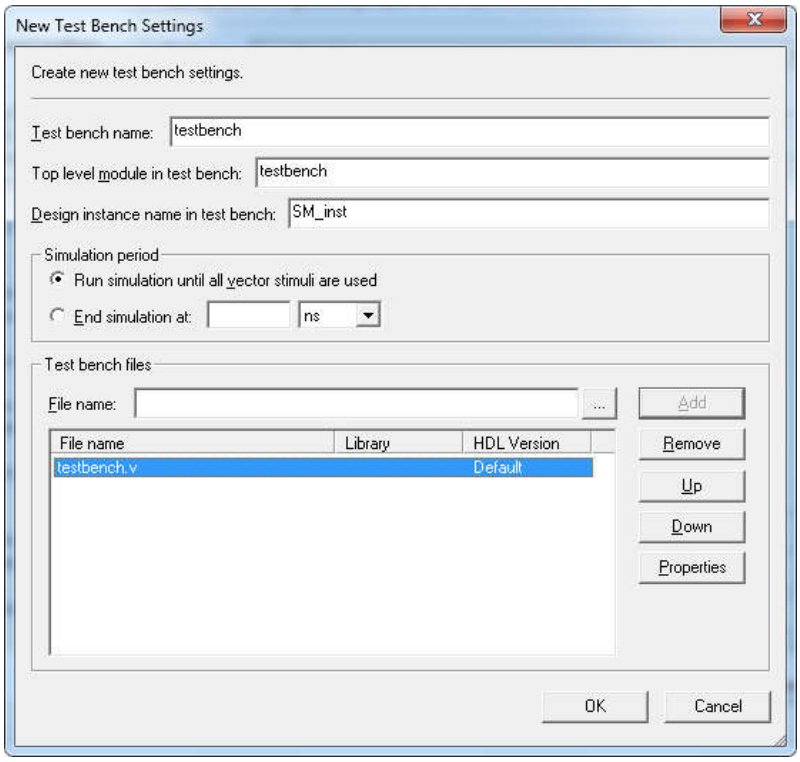


Рис. В.4 – Налаштування testbench

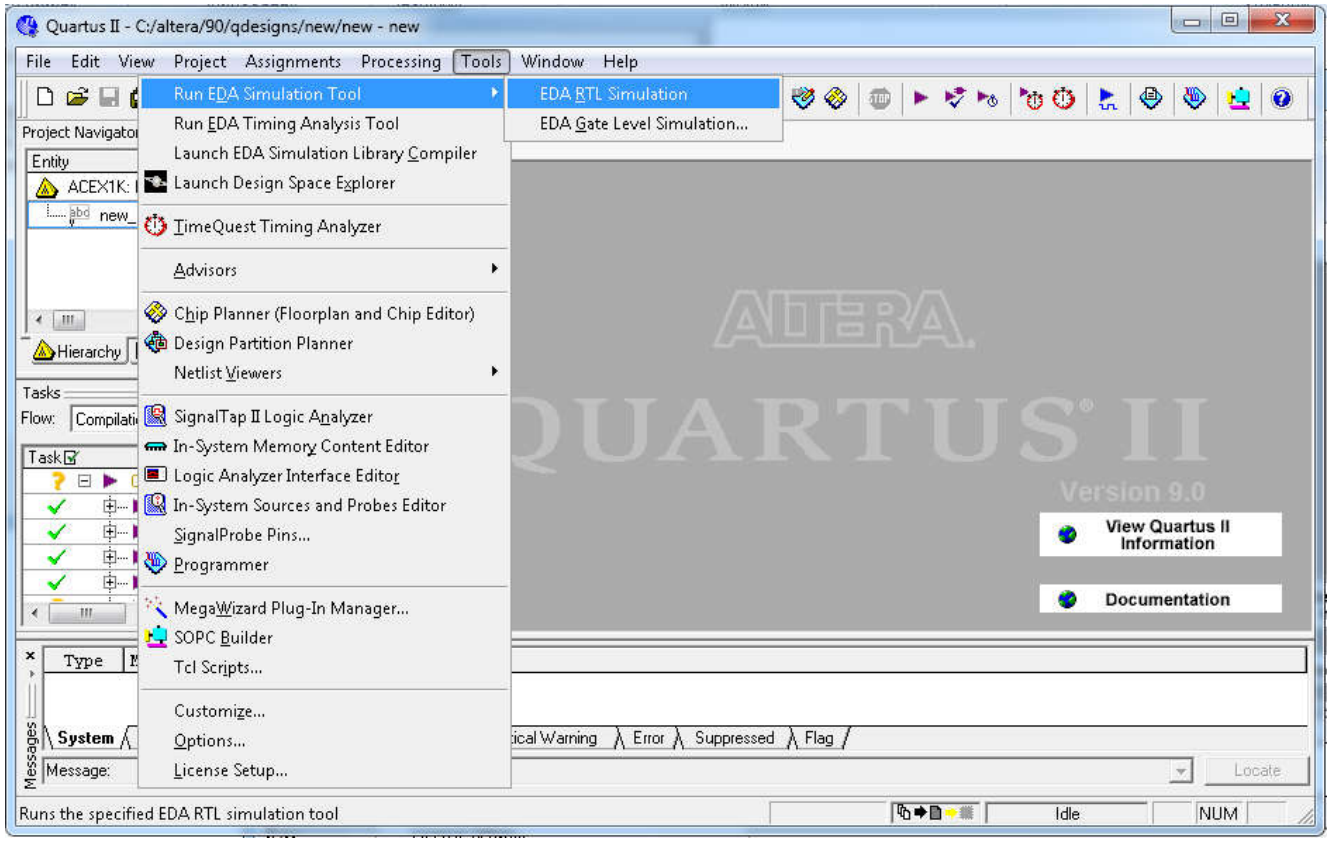


Рис. В.5 – Запуск ModelSIM з меню Quartus

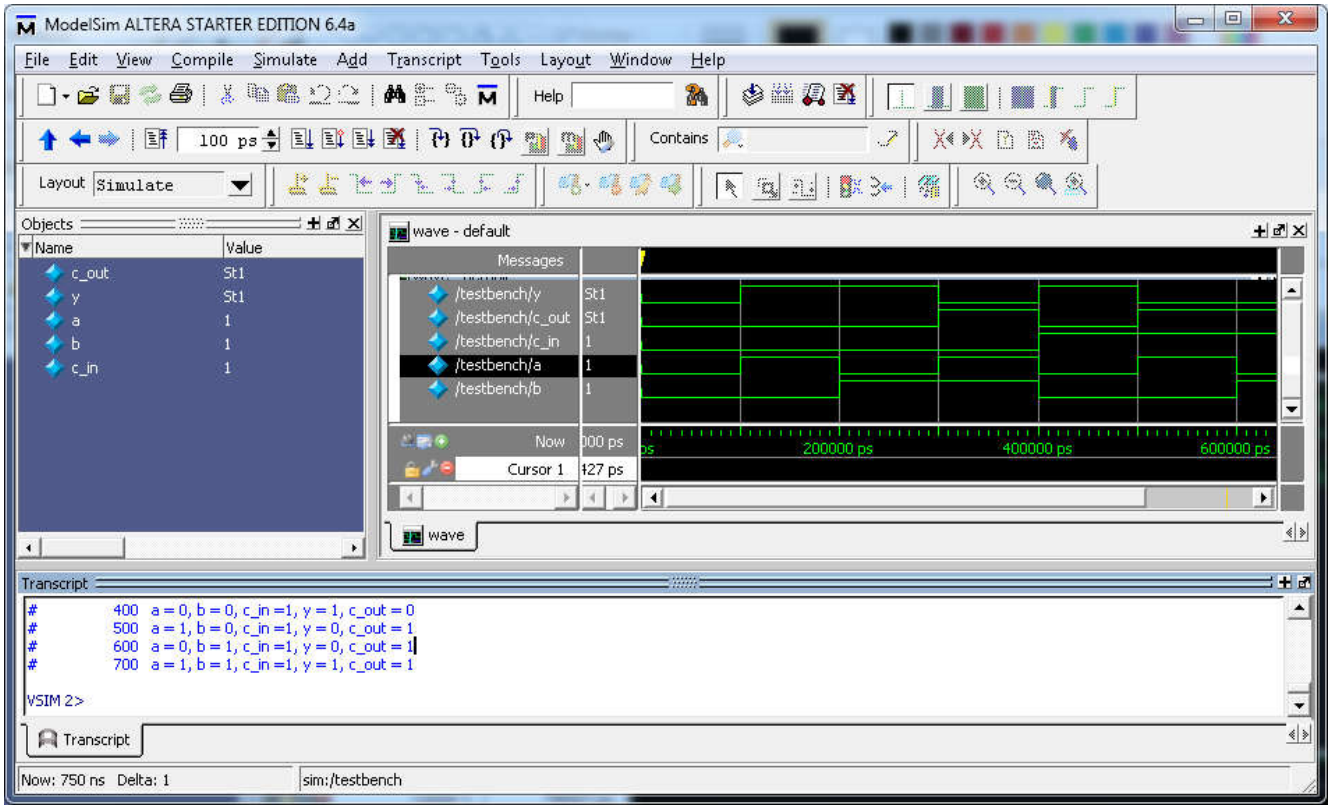


Рис. В.6 – Тестові вектори та консоль ModelSIM згенеровані на основі компіляції тестового модуля (testbench)