IT-АКАДЕМИЯ ТУСУР

Языки описания аппаратуры

Проектирование цифровой техники с применением ПЛИС и аппаратного языка разработки System Verilog

Николай Геннадьевич Зайцев
Кандидат технических наук, преподаватель кафедры КИПР ТУСУР, начальник сектора цифровой электроники ООО «ЛЭМЗ-Т»

Разнообразие языков описания аппаратуры

IT-АКАДЕМИЯ ТУСУР

Языки описания аппаратуры (*Hardware Description Language* – HDL) являются формальной записью, которая может быть использована на всех этапах разработки цифровых электронных систем.

Это возможно вследствие того, что язык легко воспринимается как машиной, так и человеком, он может использоваться на этапах проектирования, верификации, синтеза и тестирования аппаратуры так же, как и для передачи данных о проекте, модификации и сопровождении.

Потребность в данных языках возникла в 1990-е гг., когда разработчики обнаружили, что их производительность труда резко возрастет, если они будут работать на более высоком уровне абстракции, определяя только логическую функцию и предоставляя создание оптимизированных логических элементов системе автоматического проектирования (САПР).

Наиболее универсальные и распространенные языки описания аппаратуры

IT-АКАДЕМИЯ ТУСУР

AHDL. Язык описания аппаратуры AHDL разработан фирмой Altera и предназначен для описания комбинационных и последовательных логических устройств, цифровых автоматов и таблиц истинности. Язык AHDL применим только для разработки кода под ПЛИС IntelFPGA (Altera).

VHDL. Аббревиатура VHDL расшифровывается как *VHSIC Hardware Description Language*. VHSIC, в свою очередь, происходит от сокращения Very High Speed Integrated Circuits – названия программы Министерства обороны США. Разработка VHDL была начата в 1981 г. Министерством обороны для описания структуры и функциональности электронных схем. За основу для разработки был взят язык программирования ADA.

Наиболее универсальные и распространенные языки описания аппаратуры

IT-АКАДЕМИЯ ТУСУР

Verilog (SystemVerilog). Verilog был разработан фирмой Gateway Design Automation в 1984 г. После поглощения Caddence язык стал получать все более широкое распространение среди разработчиков и стал не менее популярен, чем VHDL.

В отличие от VHDL, структура и синтаксис которого напоминают такие «сложные» языки, как ADA или ALGOL, синтаксис Verilog напоминает C++. Verilog позволяет достаточно эффективно выполнить описание и провести моделирование (*simulate*) и синтез цифровых схем благодаря применению встроенных примитивов (*built-in primitives*), примитивов пользователя (*user- defined primitives*), средств временного контроля (*timing checks*), моделирования задержки распространения от входа до выхода (*pin-to-pin delay simulation*), возможности задания внешних тестовых сигналов (*external stimulus*).

Модули и их функциональность в SystemVerilog

IT-АКАДЕМИЯ ТУСУР

Модули являются основными составляющими любого проекта на Verilog. Модуль представляет собой описание цифрового блока с определенным набором входов и выходов. На нижнем уровне модуль может описывать функционирование базовых элементов, таких как вентили, на верхнем – систему в целом.

В одном файле может быть объявлено несколько модулей, но обязательно должен присутствовать один модуль верхнего уровня с именем, совпадающим с именем файла.

```
//------------------------------------------------------------
module sillyfunction
(
    input logic a,
    input logic b,
    input logic c,

    output logic y
);
    assign y = ~a & ~b & ~c | a & ~b & ~c | a & ~b & c;
endmodule
//------------------------------------------------------------
```

Модули и их функциональность в SystemVerilog

IT-АКАДЕМИЯ ТУСУР

Оператор **assign** описывает комбинационную логику. *Тильда* (~) обозначает операцию НЕ, *амперсанд* (&) – И, а вертикальная черта (|) – ИЛИ.

Тип **logic**, введенный в SystemVerilog, заменяет тип **reg**, который часто вызывал сложности при использовании в классическом Verilog. Тип **logic** рекомендуется применять везде, кроме описания сигналов с несколькими источниками (такие сигналы называются цепями и используют net, wire).

Logic и **reg** в SystemVerilog – это один и тот же тип данных. Однако у них есть некоторые различия:

Logic более гибкий. Его можно использовать в любой части кода System Verilog, в то время как **reg** можно применять только в процедурных блоках. **Logic** может быть явно приведен к классу net. В то время как **reg** может быть только переменной.

Моделирование и синтез

IT-АКАДЕМИЯ ТУСУР

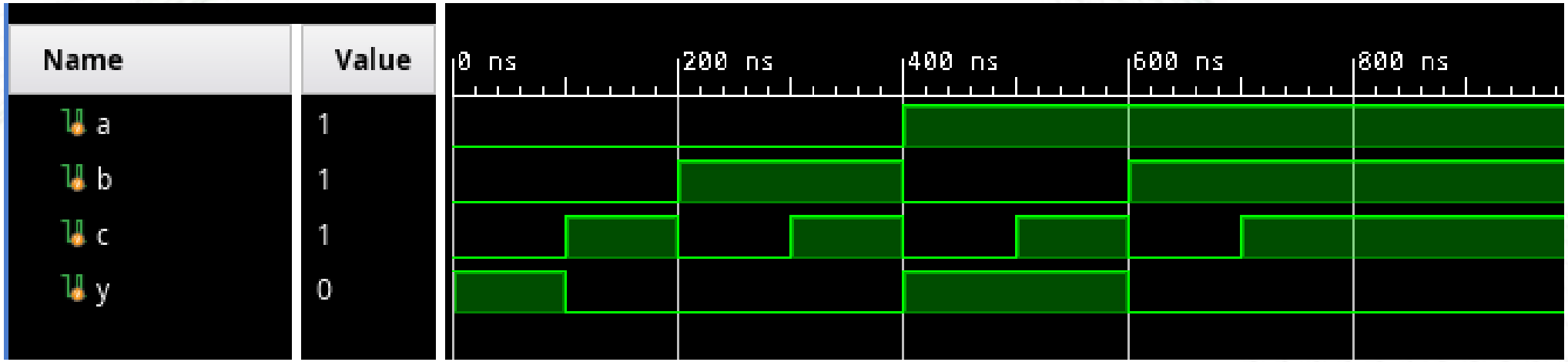
Код логического модуля

```
//*************************************************************
module sillyfunction
(
    input logic a,
    input logic b,
    input logic c,

    output logic y
);
    assign y = ~a & ~b & ~c | a & ~b & ~c | a & ~b & c;
endmodule
//*************************************************************
```

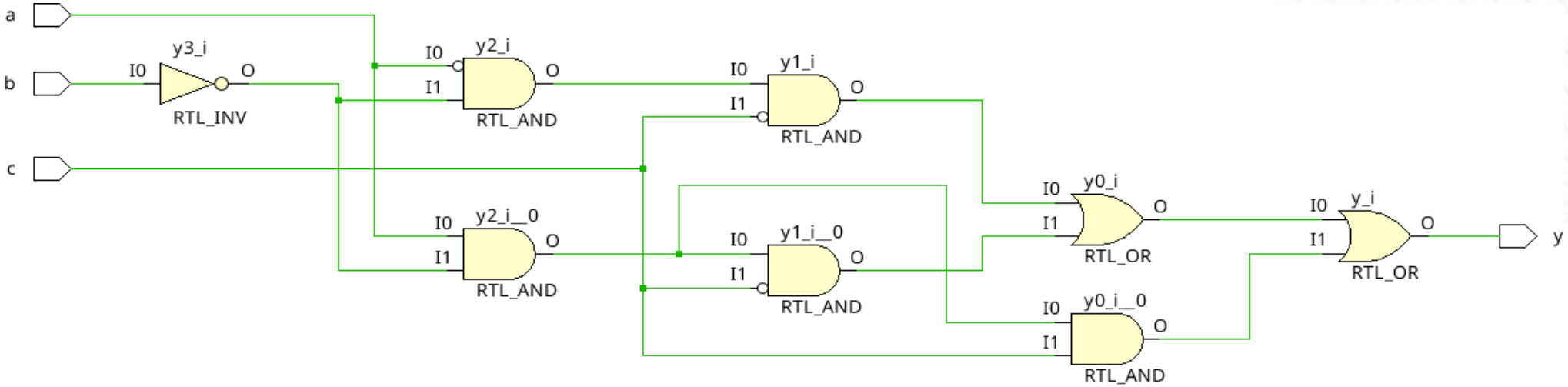
Моделирование и синтез

IT-АКАДЕМИЯ ТУСУР

Моделирование (симуляция) кода логического модуля



Синтез кода логического модуля



Моделирование и синтез

IT-АКАДЕМИЯ ТУСУР

Основные правила разработки на HDL:

Описание схем на HDL напоминает программный код. Однако необходимо помнить, что этот код предназначен для описания аппаратуры.

SystemVerilog – сложный язык со множеством операторов. Не все из них могут быть синтезированы в аппаратуре. Например, оператор вывода результатов на экран во время симуляции не превращается в цифровую схему.

Синтезируемые модули описывают цифровую схему. Среда тестирования содержит код, который подает воздействия на входы модуля и проверяет правильность значений его выходов, а также выводит несоответствия между ожидаемыми и действительными значениями.

Моделирование и синтез

IT-АКАДЕМИЯ ТУСУР

Основные правила разработки на HDL:

Код среды тестирования предназначен только для симуляции и не может быть синтезирован.

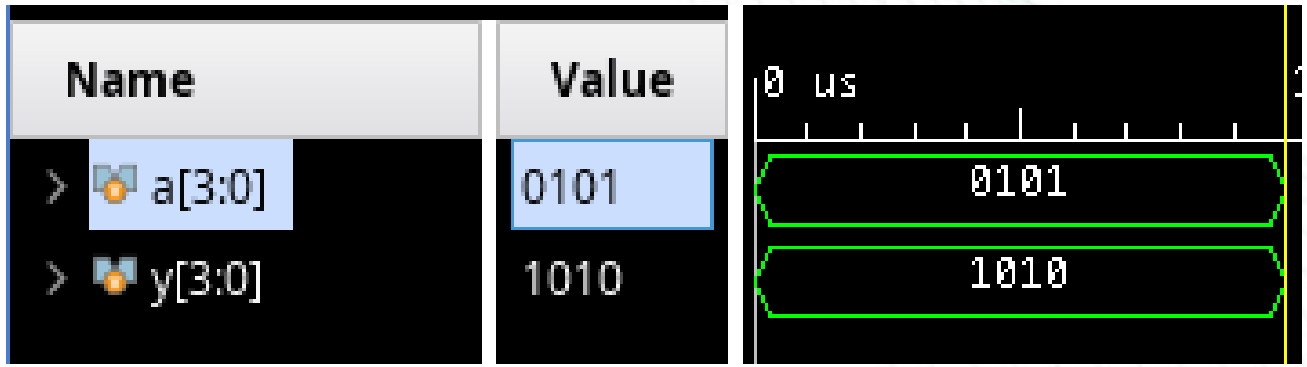
Одна из главных ошибок начинающих разработчиков заключается в том, что они думают о коде на HDL как о компьютерной программе, а не как о подспорье для описания цифровой аппаратуры. Если нет представления, хотя бы примерного, во что должен синтезироваться ваш код на HDL, то, скорее всего, результат будет неправильным и цифровая схема может получиться гораздо больше, чем нужно, или может оказаться, что код симулируется правильно, но не может быть реализован в аппаратуре.

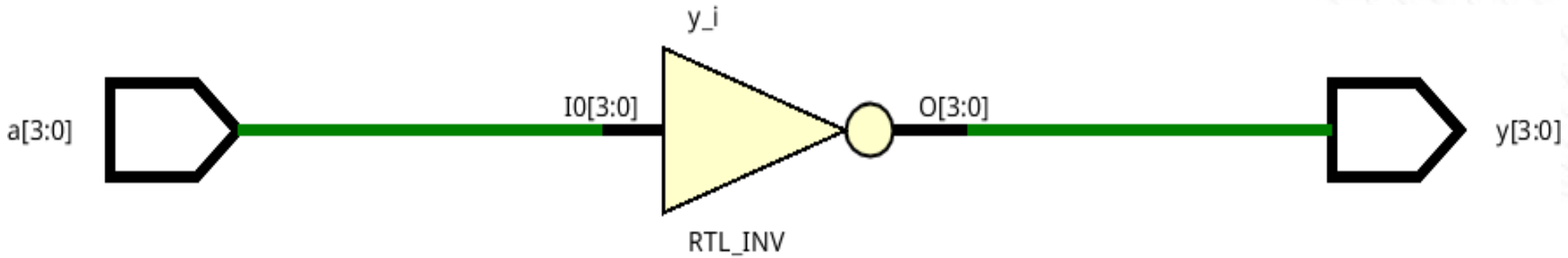
Для реализации правильного кода необходимо думать над разработкой в понятиях комбинационной логики, регистров и конечных автоматов.

Комбинационная логика

Битовые операторы манипулируют однобитовыми сигналами или многоразрядными шинами.

```
//*******************************
module inv
(
    input  logic [3:0] a,
    output logic [3:0] y
);
    assign y = ~a;
endmodule
//*******************************
```



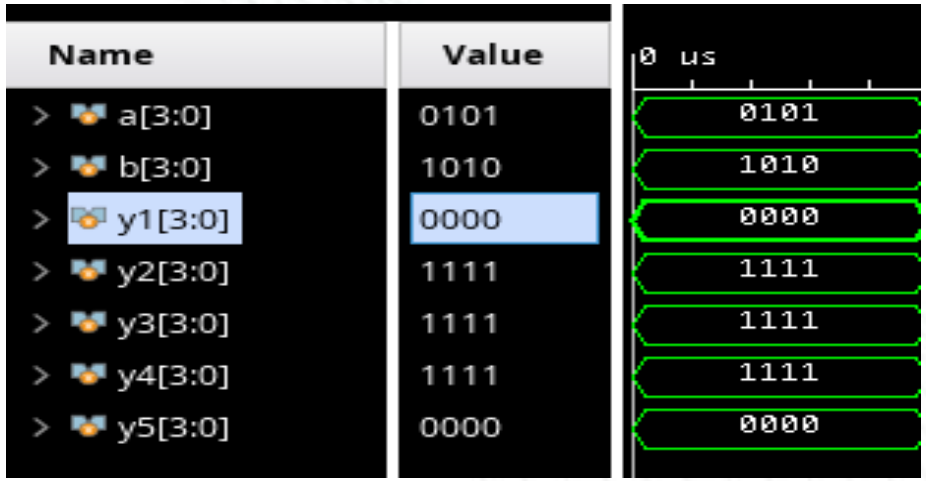


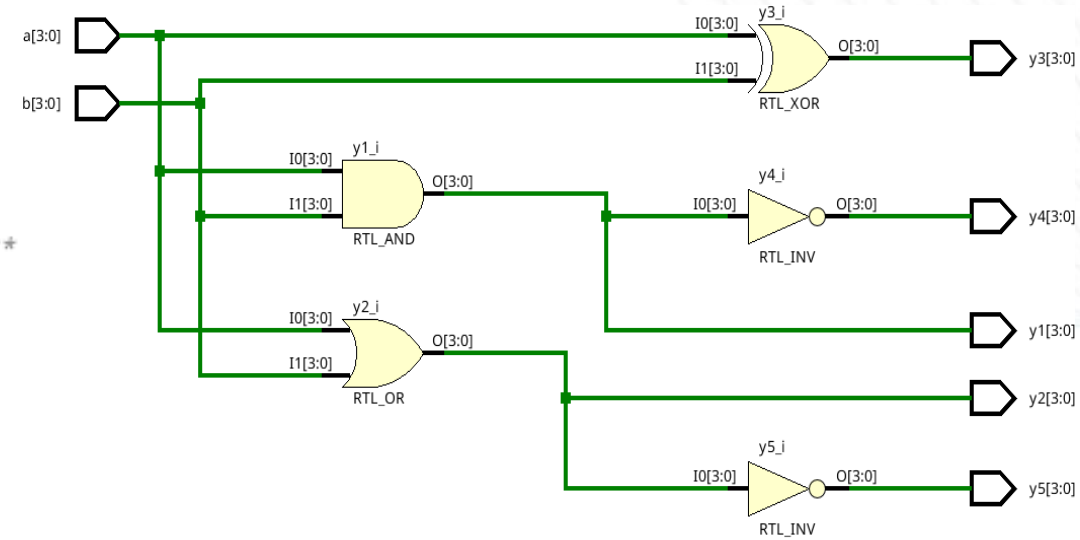
Комбинационная логика

IT-АКАДЕМИЯ ТУСУР

Битовые операторы основных логических функций

```
//*************************************************************
module gates
(
    input  logic [3:0] a,
    input  logic [3:0] b,

    output logic [3:0] y1,
    output logic [3:0] y2,
    output logic [3:0] y3,
    output logic [3:0] y4,
    output logic [3:0] y5
);
    /*five different two-input logic gates acting on 4-bit busses */
    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
endmodule
//*************************************************************
```





Комбинационная логика

IT-АКАДЕМИЯ ТУСУР

Комментарии и пробелы:

Комментарии в языке SystemVerilog схожи с комментариями языков C или Java.

Комментарии, начинающиеся с «/*», могут занимать несколько строк, до следующего знака «*/».

Комментарии, начинающиеся с «//»,продолжаются до конца строки.

```
//*************************************************************
module gates
(
    input  logic [3:0] a,
    input  logic [3:0] b,

    output logic [3:0] y1,
    output logic [3:0] y2,
    output logic [3:0] y3,
    output logic [3:0] y4,
    output logic [3:0] y5
);
    /*five different two-input logic gates acting on 4-bit busses */
    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
endmodule
//*************************************************************
```

Комбинационная логика

IT-АКАДЕМИЯ ТУСУР

Комментарии и пробелы:

SystemVerilog чувствителен к регистру символов (прописным и строчным буквам). y1и Y1в SystemVerilog – это разные сигналы, однако использование множества сигналов, отличающихся только регистром символов, вносит путаницу.

Пробелы, табуляция и разрывы строк помогают сделать читаемым разработанный код. Необходимо быть последовательным при разработке проекта.

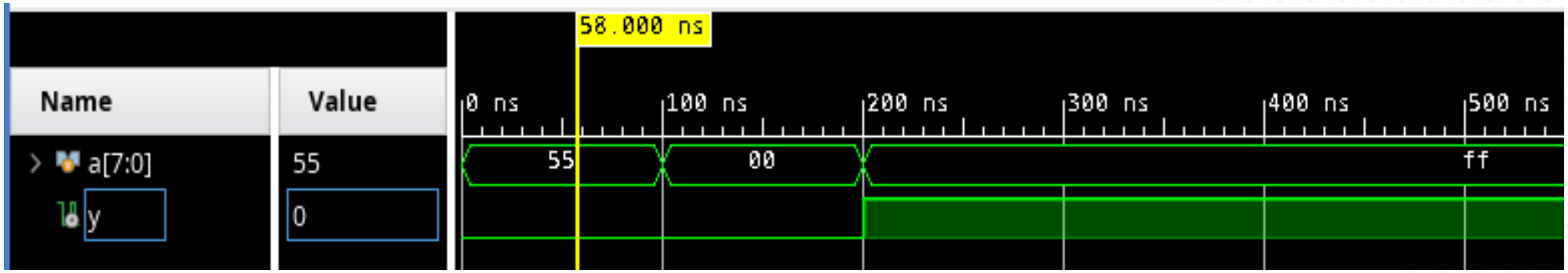
```
//////////////////////////////////////////////////////////
module gates
(
    input  logic [3:0] a,
    input  logic [3:0] b,

    output logic [3:0] y1,
    output logic [3:0] y2,
    output logic [3:0] y3,
    output logic [3:0] y4,
    output logic [3:0] y5
);
    /*five different two-input logic gates acting on 4-bit busses */
    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
endmodule
//////////////////////////////////////////////////////////
```

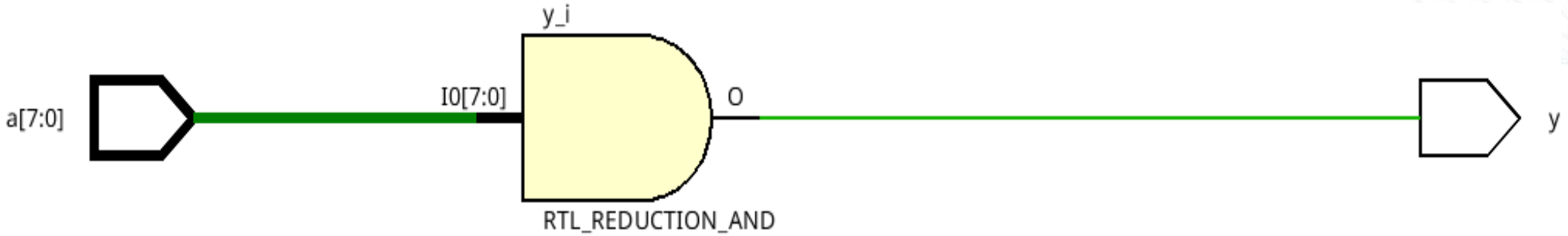
Комбинационная логика

IT-АКАДЕМИЯ ТУСУР

Операторы сокращения

```
//*************************************************************
module and8
(
    input  logic [7:0] a,
    output logic y
);
    assign y = &a;
    // &a is much easier to write than
    // assign y = a[7] & a[6] & a[5] & a[4] & a[3] & a[2] & a[1] & a[0];
endmodule
//*************************************************************
```



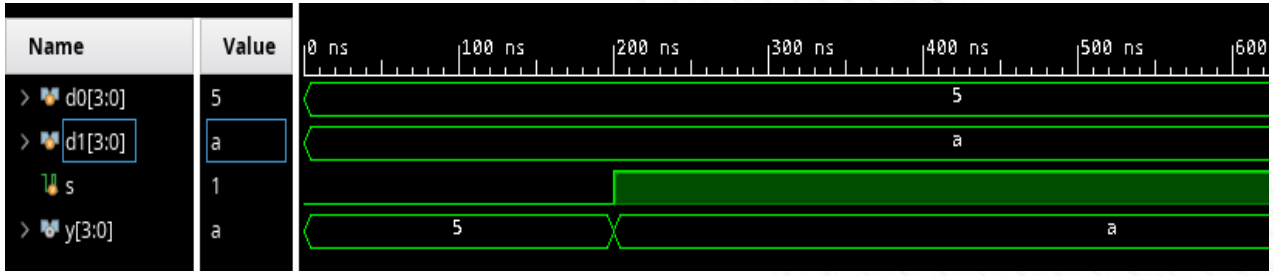


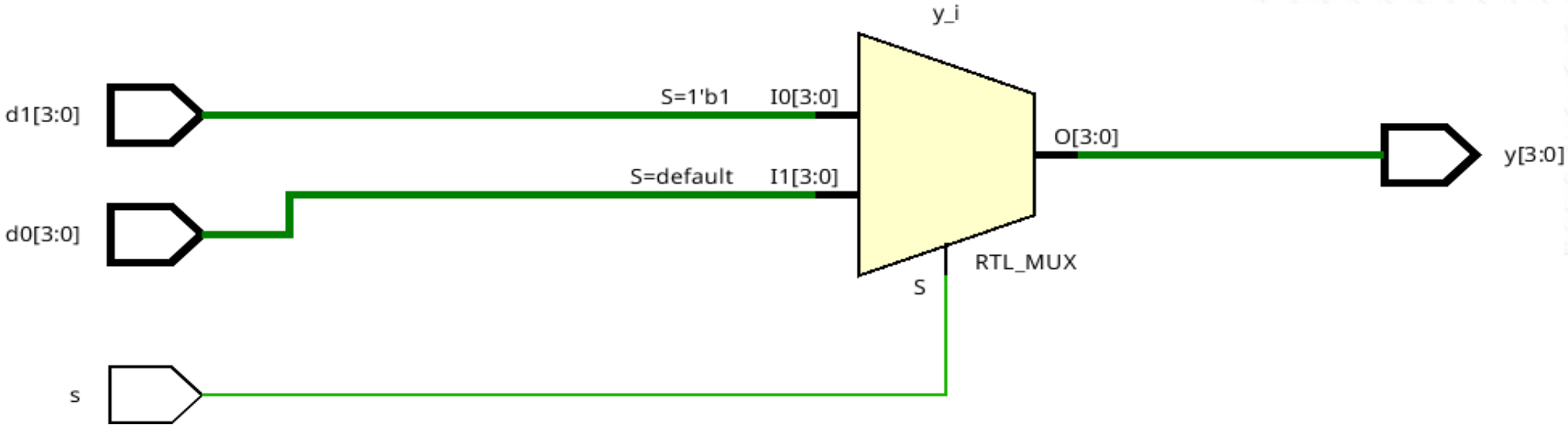
Комбинационная логика

IT-АКАДЕМИЯ ТУСУР

Условное присваивание

```
//**********************************
module mux2
(
    input  logic [3:0] d0,
    input  logic [3:0] d1,

    input  logic s,

    output logic [3:0] y
);
    assign y = s ? d1 : d0;
endmodule
//**********************************
```



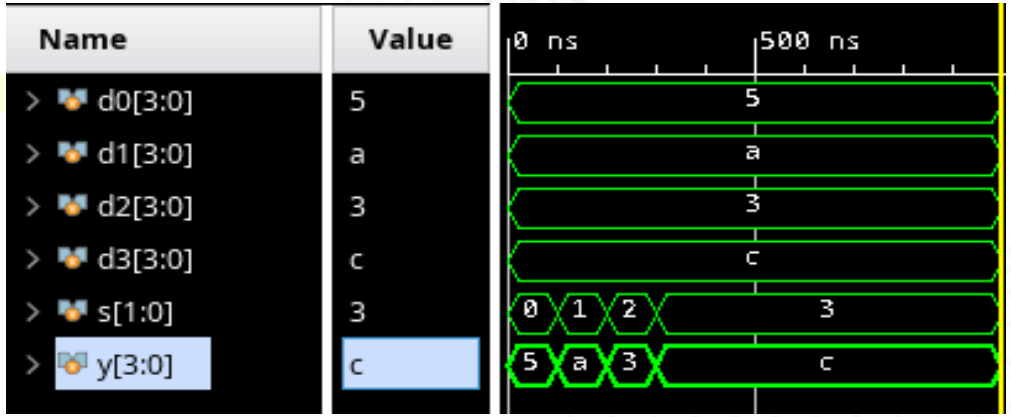


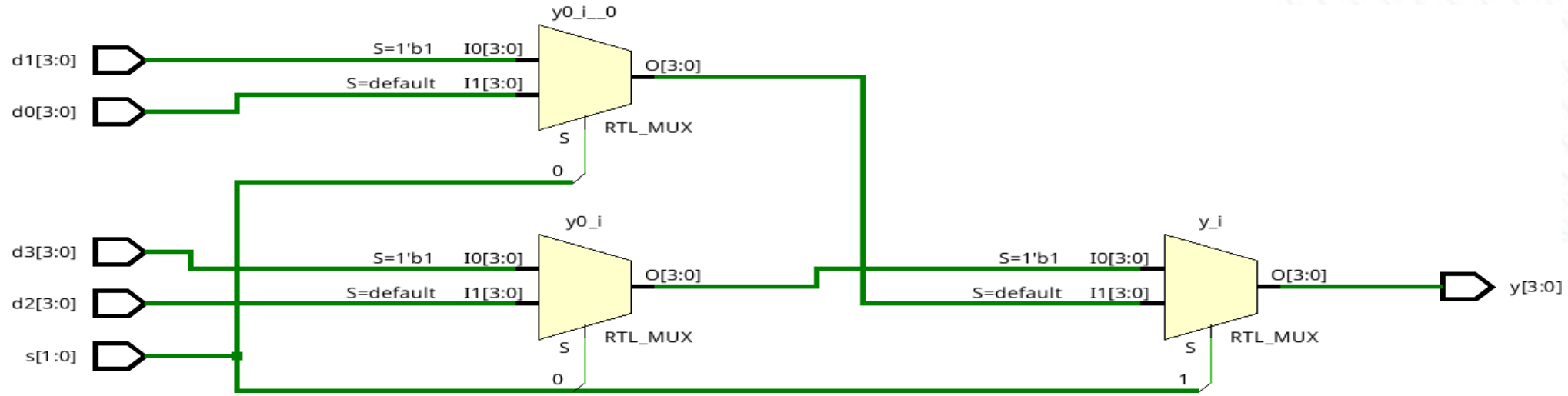
Комбинационная логика

IT-АКАДЕМИЯ ТУСУР

Условное присваивание

```
//*************************************************************
module mux4
(
    input  logic [3:0] d0,
    input  logic [3:0] d1,
    input  logic [3:0] d2,
    input  logic [3:0] d3,

    input  logic [1:0] s,

    output logic [3:0] y
);
    assign y = s[1] ? (s[0] ? d3 : d2) : (s[0] ? d1 : d0);
endmodule
//*************************************************************
```





Комбинационная логика

IT-АКАДЕМИЯ ТУСУР

Внутренние переменные

$$S = A \oplus B \oplus Cin$$
$$Cout = AB + ACin \oplus BCin$$

~

$$P = A \oplus B$$
$$G = AB$$
$$S = P \oplus C\ in$$
$$Cout = G + PCin$$

Операции присваивания в HDL (assign в языке SystemVerilog) происходят параллельно. Это отличается от традиционных языков программирования, таких как C или Java, в которых операторы оцениваются в том порядке, в котором они записаны. В традиционных языках важно, что выражение $S = P \oplus C\ in$ следует за выражением $P = A \oplus B$, поскольку операторы выполняются последовательно.

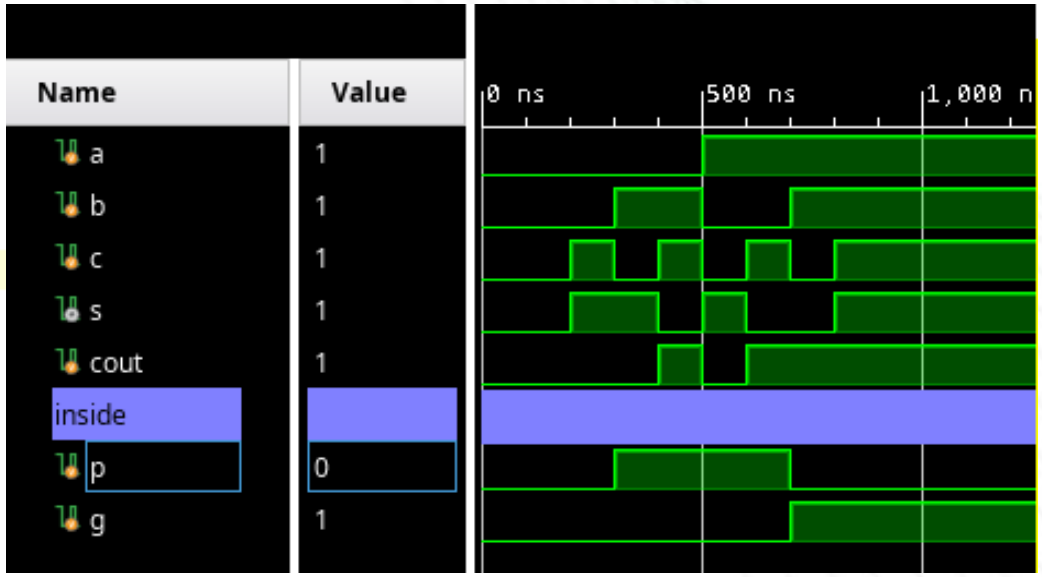
В HDL порядок записи не имеет значения. Подобно аппаратным средствам, операторы присваивания HDL выполняются в момент, когда входы и сигналы с правой стороны выражения меняют свое значение независимо от порядка, в котором операторы присваивания появляются в модуле.

Комбинационная логика

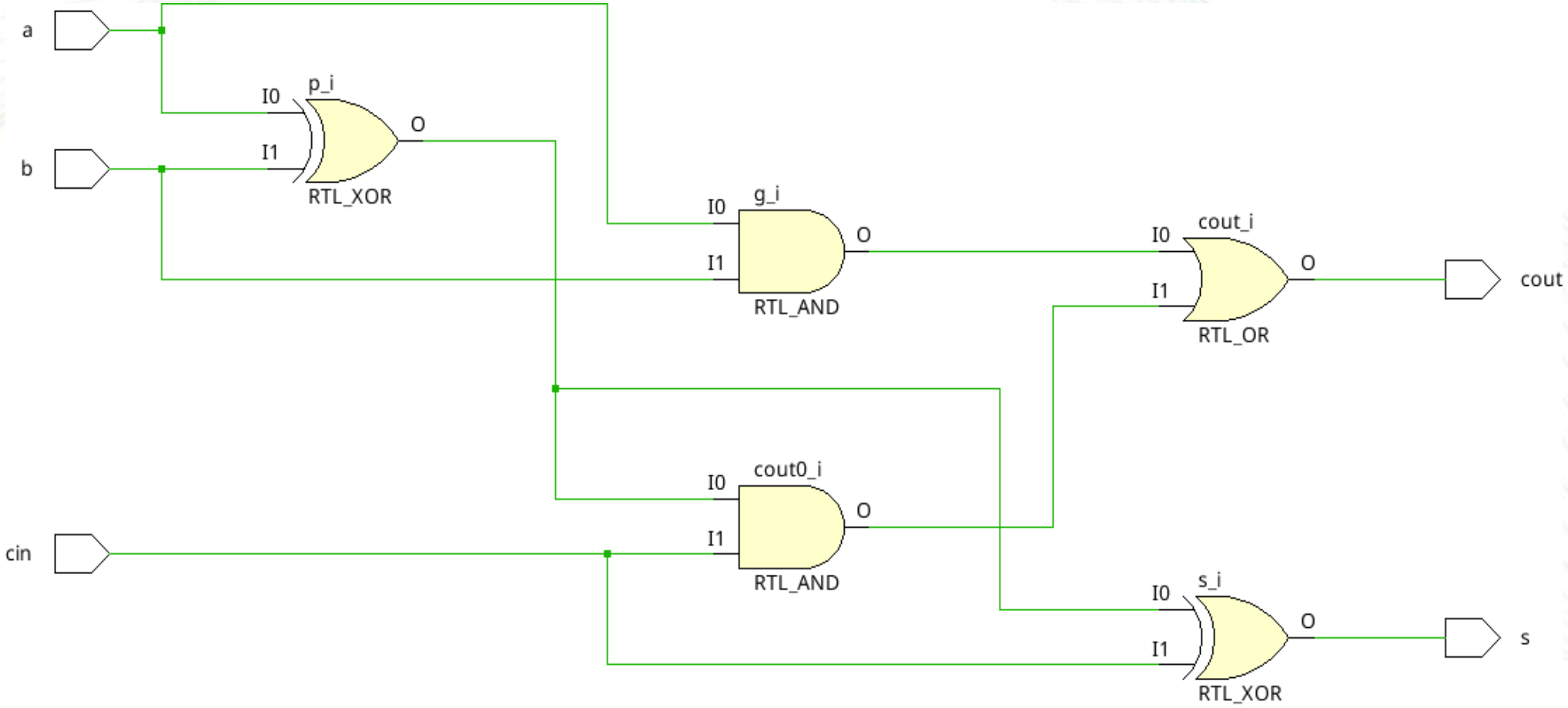
IT-АКАДЕМИЯ ТУСУР

Внутренние переменные

```
//*************************************************************
module fulladder
(
    input  logic a,
    input  logic b,
    input  logic cin,

    output logic s,
    output logic cout
);
    logic p, g;

    assign p = a ^ b;
    assign g = a & b;
    assign s = p ^ cin;
    assign cout = g |(p & cin);
endmodule
//*************************************************************
```



Комбинационная логика



Комбинационная логика

IT-АКАДЕМИЯ ТУСУР

Приоритет

Операция		Значение
Высший	~	Побитовое отрицание (НЕ)
	*, /, %	Умножение, деление, остаток
	+, –	Сложение, вычитание
	<<, >>	Сдвиг влево/вправо
	<<<, >>>	Арифметический сдвиг влево/вправо
	<, <=, >, >=	Сравнение на больше-меньшее
Низший	==, !=	Сравнение на равенство
	&, ~&	И, И-НЕ
	^, ~^	Исключающее ИЛИ, исключающее ИЛИ-НЕ
	|, ~|	ИЛИ, ИЛИ-НЕ
	?:	Условный оператор

Комбинационная логика

IT-АКАДЕМИЯ ТУСУР

Числа:

Целочисленные константы в Verilog задаются в десятичной, шестнадцатеричной, восьмеричной и двоичной системах счисления.

В случае если система счисления не задана, используется десятичная система счисления, при этом число трактуется как целое число со знаком.

Запись	Число бит	Основание	Значение	Представление
3'b101	3	2	5	101
`b11	?	2	3	0000011
8'b11	8	2	3	00000011
8'b1010_1011	8	2	171	10101011
3'd6	3	10	6	110
6'o42	6	8	34	100010
8'hAB	8	16	171	10101011
42	?	10	42	000101010

Исключение: 0 и 1 служат конструкциями SystemVerilog для заполнения шины нулями или единицами соответственно.

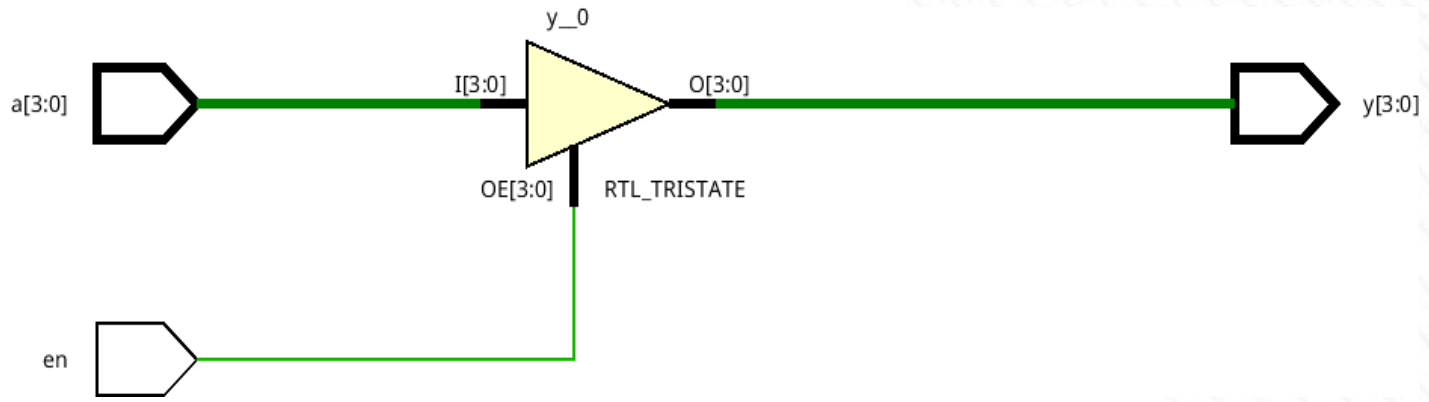
Комбинационная логика

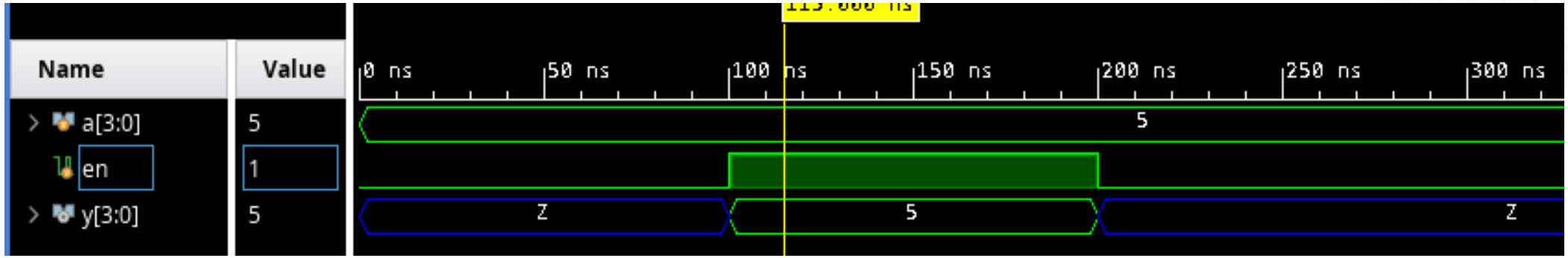
IT-АКАДЕМИЯ ТУСУР

Z-состояния и X-состояния

Z-состояние используется для указания на плавающее значение.
X-состояние используется для указания недействительного логического уровня.
Если на шину одновременно попадают 0 и 1 с двух активных тристабильных буферов (или других элементов), то в результате получаем x, что указывает на конфликт.

```
//*********************************
module tristate
(
    input logic [3:0] a,
    input wire en,

    output tri [3:0] y
);
    assign y = en ? a : 4'bz;
endmodule
//*********************************
```





Комбинационная логика

IT-АКАДЕМИЯ ТУСУР

Манипуляция битами используется при необходимости работать с фрагментом шины или сцеплять (объединять) сигналы для формирования шины.

Оператор {}используется для объединения шин. {3{d[0]}}указывает на три копии d[0].

Необходимо сформировать шину из 9 бит «c2 c1 d0 d0 d0 c0 1 0 1»:

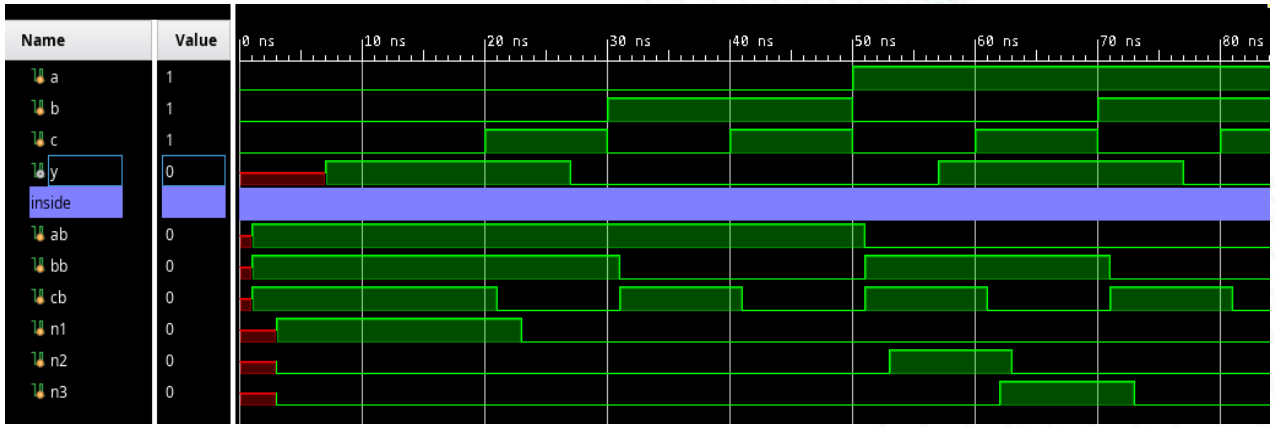
assign y = {c[2:1], {3{d[0]}}, c[0], 3'b101};

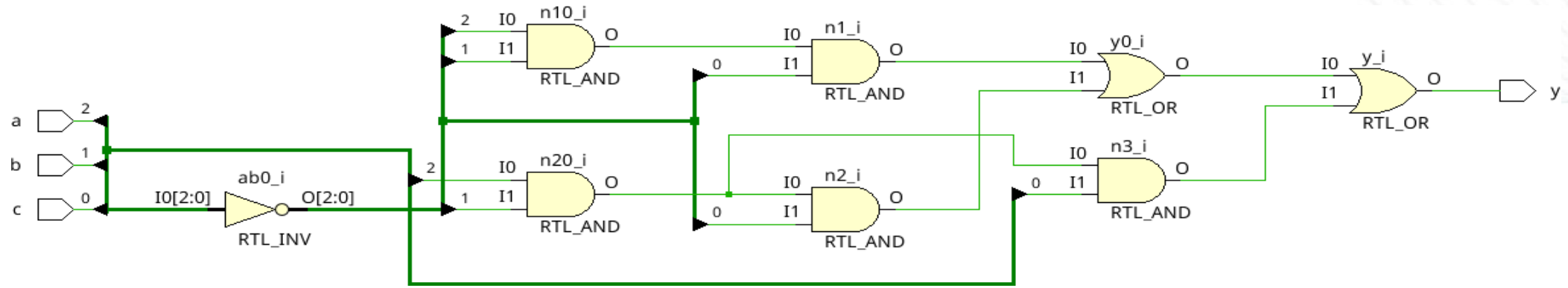
Комбинационная логика

IT-АКАДЕМИЯ ТУСУР

Задержки применимы только при симуляции!

```
//*****************************************
//'timescale 1ns/1ps
module example
(
    input  logic a,
    input  logic b,
    input  logic c,
    output logic y
);
    logic ab, bb, cb, n1, n2, n3;
    assign #1 {ab, bb, cb} = ~{a, b, c};
    assign #2 n1 = ab & bb & cb;
    assign #2 n2 = a & bb & cb;
    assign #2 n3 = a & bb & c;
    assign #4 y = n1 |n2 | n3;
endmodule
//*****************************************
```

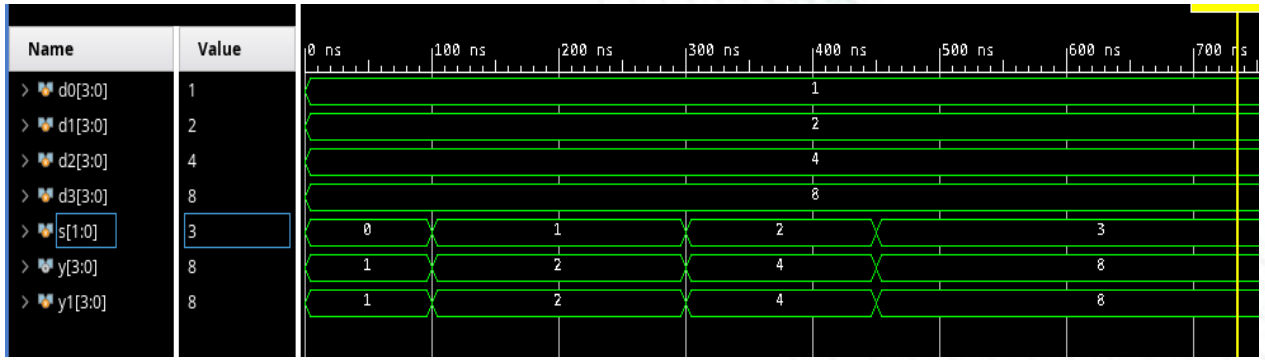


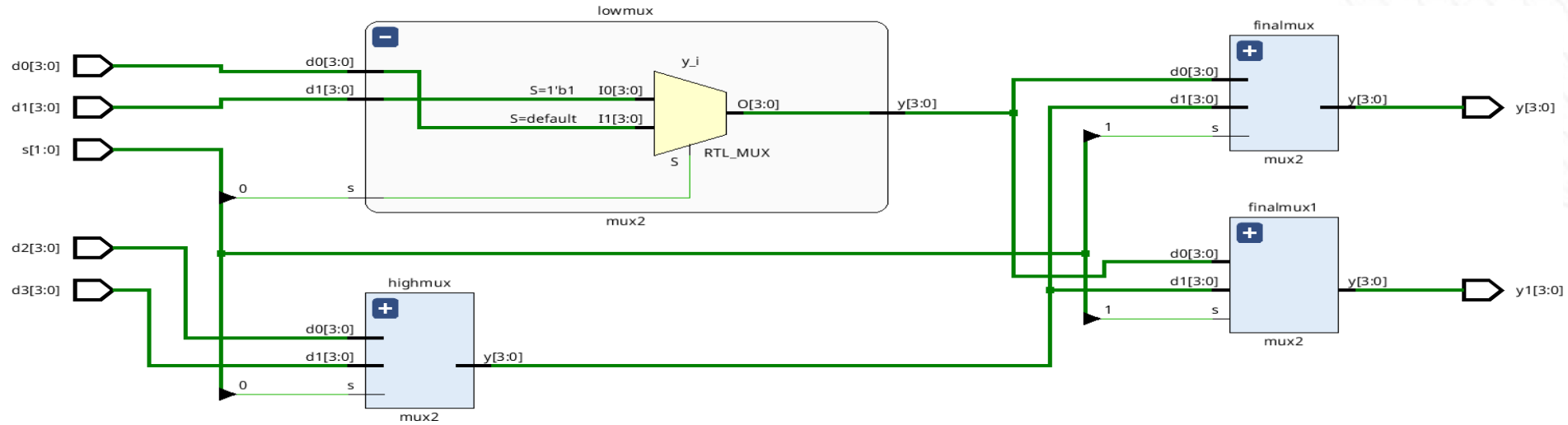


Структурное моделирование

IT-АКАДЕМИЯ ТУСУР

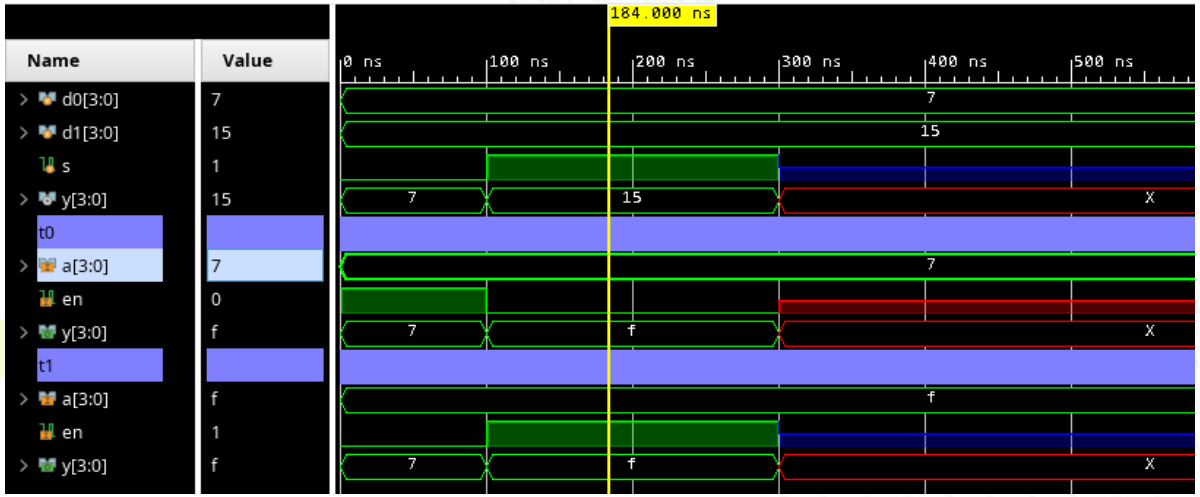
```
//*************************************************************
module mux4_1
(
    input logic [3:0] d0,
    input logic [3:0] d1,
    input logic [3:0] d2,
    input logic [3:0] d3,
    input  logic [1:0] s,
    output logic [3:0] y,
    output logic [3:0] y1
);
    logic [3:0] low, high;
    mux2 lowmux(d0, d1, s[0], low);
    mux2 highmux(d2, d3, s[0], high);

    mux2 finalmux(low, high, s[1], y);

    mux2 finalmux1
    (
        .d0(low),    //input
        .d1(high),   //input
        .s(s[1]),    //input
        .y(y1)       //output
    );
endmodule
//*************************************************************
```

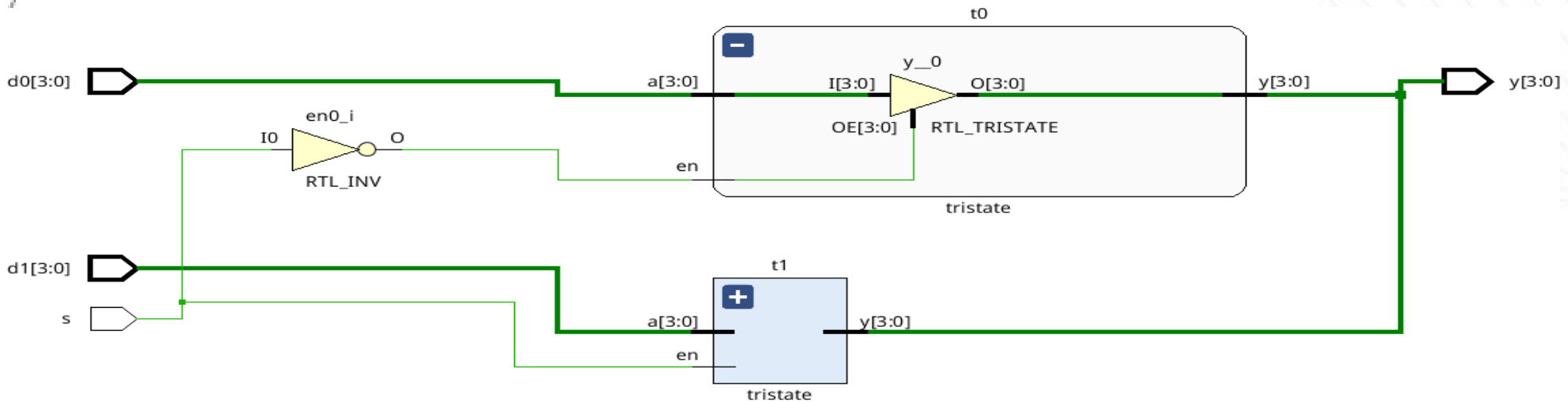




Структурное моделирование

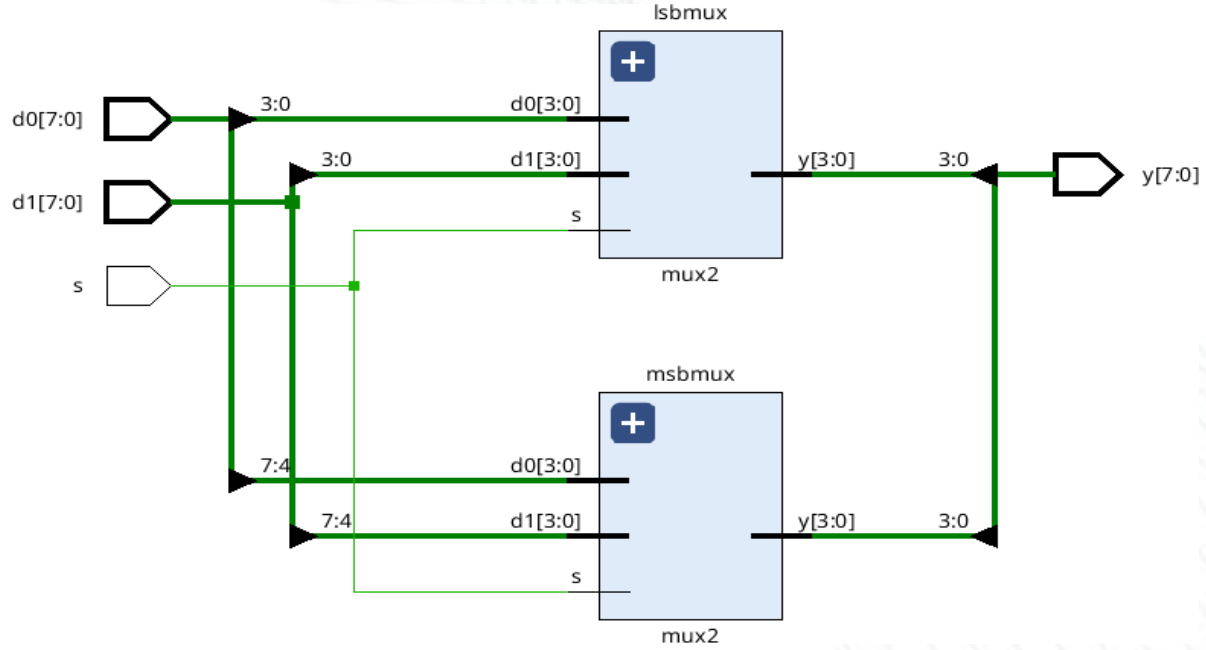
```
//************************************
module mux2_1
(
    input logic [3:0] d0,
    input logic [3:0] d1,
    input logic s,
    output tri [3:0] y
);
    tristate t0(d0, ~s, y);
    tristate t1(d1,  s, y);
endmodule
//************************************
```

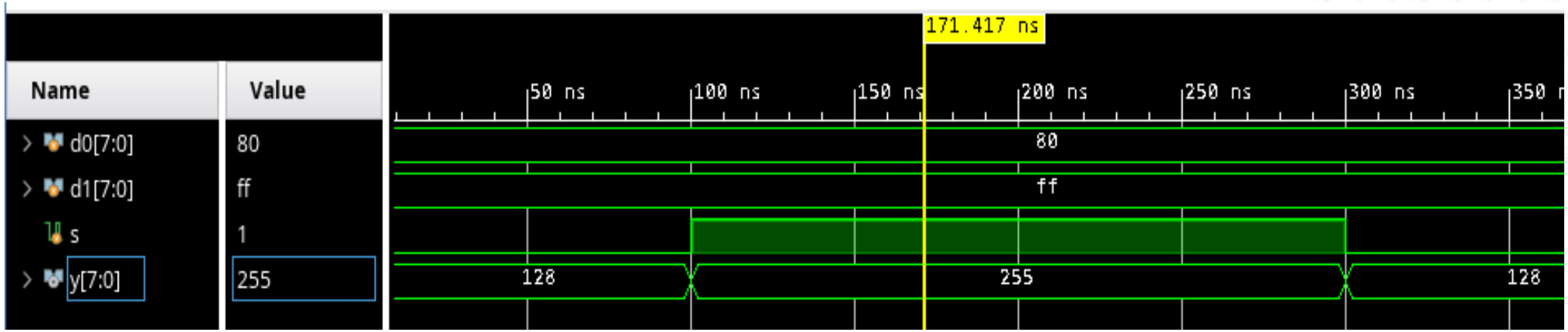




Структурное моделирование

```
//************************************************
module mux2_8
(
    input logic [7:0] d0,
    input logic [7:0] d1,
    input logic s,
    output logic [7:0] y
);
    mux2 lsbmux(d0[3:0], d1[3:0], s, y[3:0]);

    mux2 msbmux
    (
        .d0(d0[7:4]), //input d0
        .d1(d1[7:4]), //input d1
        .s(s),        //input s
        .y(y[7:4])    //output y
    );
endmodule
//************************************************
```





IT-АКАДЕМИЯ ТУСУР

Проектирование цифровой техники с применением ПЛИС и аппаратного языка разработки System Verilog