

Языки описания схем

mk.cs.msu.ru → Лекционные курсы → Языки описания схем

Блок 13

Verilog:
Схемный «Hello, World!»

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

```
Файл main.v:  
module main(x, y, z);  
    input [1:0] x, y;  
    output [2:0] z;  
    assign z = x + y;  
endmodule
```

```
Файл test.v:  
module test();  
    reg [1:0] x, y;  
    wire [2:0] z;  
    main _testee(.x(x), .y(y), .z(z));  
    initial begin  
        $monitor(x,,y,,z,,$stime);  
        #1 x = 1; y = 2;  
        #2 x = 2;  
        #2 $finish;  
    end  
endmodule
```

```
Запуск в консоли Linux:  
>iverilog main.v test.v  
>./a.out  
x x x      0  
1 2 3      1  
2 2 4      3  
>
```

Типы, точки

```
module main(x, y, z);  
    input wire [1:0] x, y;  
    output wire [2:0] z;  
    assign z = x + y;  
endmodule
```

Типы данных \mathcal{V} делятся на две категории: типы соединений и типы переменных

Соединения и переменные будем называть точками схемы (~ переменные в C/C++)

Категории типов различаются

- ▶ идейно: можно понимать соединения как элементы схемы без памяти, а переменные — как потенциально имеющие память
- ▶ технически: категориями задаются синтаксические конструкции, в которых можно использовать точки соответствующих типов

Объявление всех точек схемы устроено примерно как в C/C++:

<тип> <имена через запятую>;

Проводá, шины

```
module main(x, y, z);  
    input wire [1:0] x, y;  
    output wire [2:0] z;  
    assign z = x + y;  
endmodule
```

`wire` — это тип **провода**, понимающегося как «пассивное» соединение несколько мест в схеме с заданием значения в нём каким-либо из этих мест

`wire [<msb>:<lsb>]` — это **шина** проводов, в которой старший разряд имеет номер *msb*, и младший — *lsb*

Рекомендуется (пока не разберётесь углублённо в документации) всегда писать *lsb* равный 0 и положительный *msb*

Порты

```
module main(x, y, z);  
    input wire [1:0] x, y;  
    output wire [2:0] z;  
    assign z = x + y;  
endmodule
```

Ключевые слова `input` и `output` дописываются слева к объявлениям точек, чтобы назначить эти точки соответственно **входами** (**входными портами**) и **выходами** (**выходными портами**) схемы

Таким образом, модуль `main` имеет три порта: входные шины `x` и `y` ширины 2 и выходную шину `z` ширины 3

Порты

```
module main(x, y, z);  
    input wire [1:0] x, y;  
    output wire [2:0] z;  
    assign z = x + y;  
endmodule
```

Дообъявление порта — это ключевое слово `input` или `output` (или другое из соответствующего списка) вместе с объявлением точки или в допустимых сочетаниях вне объявления (см. документацию)

Объявление типа и вида порта может быть сделано и у имени модуля (как дообъявление, но через «,» вместо «;»):

```
module main(input wire [1:0] x, y, output wire [2:0] z);  
    assign z = x + y;  
endmodule
```

assign

```
module main(x, y, z);  
  input wire [1:0] x, y;  
  output wire [2:0] z;  
  assign z = x + y;  
endmodule
```

`assign <соединение> = <выражение>;`

Это **непрерывное присваивание** значения *выражения* в *соединение*
Коротко об аппаратной семантике:

- ▶ Если использовать только «хорошие» точки и операции, то *выражением* задаётся комбинационная схема Σ , выход которой сгруппирован в шину
- ▶ *Соединение* с поправкой на подходящее расширение или «обрезание» разрядов объявляется выходом Σ

То есть непрерывное присваивание является поддерживаемым, если поддерживаются все операции в *выражении*

assign

```
module main(x, y, z);  
  input wire [1:0] x, y;  
  output wire [2:0] z;  
  assign z = x + y;  
endmodule
```

`assign <соединение> = <выражение>;`

Коротко о программной семантике: каждый раз, когда изменяется значение *выражения*, немедленно (в том же регионе) изменяется значение *соединения*

Программная семантика операции +: сложение чисел в двоичных записях

Аппаратная семантика операции +: схема **сумматора**

assign

```
module main(x, y, z);  
    input wire [1:0] x, y;  
    output wire [2:0] z;  
    assign z = x + y;  
endmodule
```

Таким образом,

- ▶ в аппаратном смысле модуль выше представляет собой схему сумматора, и
- ▶ программный смысл согласуется с аппаратным: при каждом изменении суммы чисел на входах x и y в выход z немедленно направляется эта сумма

Тестирующие модули

```
module test();
  reg [1:0] x, y;
  wire [2:0] z;
  main _testee(.x(x), .y(y), .z(z));
  initial begin
    $monitor(x,y,z,$stime);
    #1 x = 1; y = 2;
    #2 x = 2;
    #2 $finish;
  end
endmodule
```

Разработанный модуль схемы принято тестировать, и один из способов это сделать — разработать **тестирующий модуль**

В модуле схемы следует использовать только поддерживаемые и игнорируемые конструкции

В тестирующем модуле можно использовать все возможности языка

Тестирующие модули

```
module test();
  reg [1:0] x, y;
  wire [2:0] z;
  main _testee(.x(x), .y(y), .z(z));
  initial begin
    $monitor(x,y,z,$stime);
    #1 x = 1; y = 2;
    #2 x = 2;
    #2 $finish;
  end
endmodule
```

В тестирующем модуле обычно не нужны порты, и в нём содержатся

- ▶ экземпляр тестируемой схемы с подходящей управляющей обёрткой и
- ▶ сценарий выполнения тестируемой схемы

Экземпляры модулей, назначения портов

```
reg [1:0] x, y;  
wire [2:0] z;  
main _testee(.x(x), .y(y), .z(z));
```

Вставка экземпляра модуля в тело другого модуля устроена так:

<имя модуля> <имя экземпляра>(<назначение портов через «,»>)
(~ объявление объекта в C/C++)

Рекомендованный способ назначения порта устроен так:¹

.<имя порта>(<выражение>)

При таком назначении

- ▶ во входной порт непрерывно присваивается значение выражения, и
- ▶ значение выходного порта непрерывно присваивается в выражение
 - ▶ (и тогда выражение должно быть таким, в которое можно присвоить, как lvalue в C/C++)

¹ Можно назначать и по-другому, но «бывалые» иногда отмечают, что рекомендованный способ даёт существенно меньше ошибок по невнимательности

Экземпляры модулей, назначения портов

```
reg [1:0] x, y;  
wire [2:0] z;  
main _testee(.x(x), .y(y), .z(z));
```

То есть в фрагменте кода выше используется экземпляр модуля `main`, имеющий имя `_testee`, в котором

- ▶ переменные `x` и `y` внешнего модуля непрерывно присваиваются в одноимённые входные порты экземпляра и
- ▶ значение выходного порта `z` экземпляра непрерывно присваивается в одноимённое соединение модуля

Сценарии выполнения

```
initial begin
    $monitor(x,,y,,z,,$stime);
    #1 x = 1; y = 2;
    #2 x = 2;
    #2 $finish;
end
```

После вставки экземпляра следует (*всеми правдами и неправдами*) придумать сценарий выполнения этого экземпляра и реализовать этот сценарий в значениях, посылающихся во входные порты экземпляра

Кроме того, по итогам проигрывания этого сценария было бы неплохо получить «осязаемый» итог, отладочную информацию

В примере выше всё это реализовано в одной начальной процедуре

begin-end, \$monitor

```
initial begin
    $monitor(x,y,z,$stime);
    #1 x = 1; y = 2;
    #2 x = 2;
    #2 $finish;
end
```

Ключевыми словами `begin` и `end` обрамляется **составная команда**, содержащая в себе последовательность команд через «;» (как в **Pascal**)

Команда `$monitor` аналогична команде `$display`, но значения выводятся не один раз немедленно, а каждый раз при изменении значений аргументов

Кроме того, `$monitor` допускает аргументы без формата (как выше):

- ▶ значения аргументов выводятся в некотором формате по умолчанию, и
- ▶ пустота между запятыми в аргументах означает пробел в выводе

\$time, \$stime

```
initial begin
    $monitor(x,,y,,z,,$stime);
    #1 x = 1; y = 2;
    #2 x = 2;
    #2 $finish;
end
```

```
$time
$stime
```

Это неподдерживаемые выражения, которые возвращают значение текущего модельного времени в разном числе разрядов (\$time шире, \$stime уже)

Блокирующее присваивание

```
initial begin
    $monitor(x,y,z,$stime);
    #1 x = 1; y = 2;
    #2 x = 2;
    #2 $finish;
end
```

<точка> = <выражение>

Это одна из команд \mathcal{V} , **блокирующее присваивание**

Синтаксически отличается от непрерывного присваивания тем, что

- ▶ нет ключевого слова `assign` и
- ▶ Непрерывное присваивание — это «внешняя» конструкция (уровня подсхем), а блокирующее — «внутренняя» (записывается внутри «внешних» конструкций)

Задержка

```
initial begin
    $monitor(x,y,z,$stime);
    #1 x = 1; y = 2;
    #2 x = 2;
    #2 $finish;
end
```

#<целое число>

Это **задержка**: игнорируемая запись, которую можно вставить, в частности, перед любой командой

Общий приблизительный смысл задержки — это указание в заданном месте «подождать» с указанием длительности ожидания

Задержка, записанная перед блокирующим присваиванием, означает приостановку выполнения процедуры с продолжением спустя указанное число единиц времени (в регионе $t+N$ относительно текущего региона t для задержки $\#N$)

\$finish

```
initial begin
    $monitor(x,,y,,z,,$stime);
    #1 x = 1; y = 2;
    #2 x = 2;
    #2 $finish;
end
```

\$finish

Это команда принудительного завершения симуляции (~ `exit()` в C/C++)

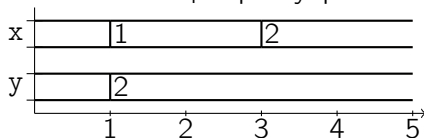
Сценарии выполнения и диаграммы сигналов

```
initial begin
    $monitor(x,,y,,z,,$stime);
    #1 x = 1; y = 2;
    #2 x = 2;
    #2 $finish;
end
```

Таким образом, выполнение начальной процедуры выше устроено так:

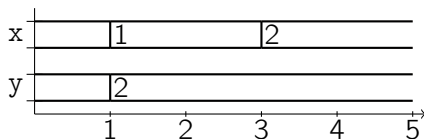
- ▶ В начале симуляции (в регионе 0) начать отслеживать значения точек x , y , z и модельного время, выводя их при каждом изменении
- ▶ Спустя единицу времени (в регионе 1) присвоить 1 в x и 2 в y
- ▶ Спустя две единицы времени (в регионе 3) присвоить 2 в x
- ▶ Спустя две единицы времени (в регионе 5) завершить симуляцию

То есть здесь реализован такой сценарий управления схемой:



Заключение

```
module main(x, y, z);  
    input [1:0] x, y;  
    output [2:0] z;  
    assign z = x + y;  
endmodule
```



```
>iverilog main.v test.v  
>./a.out  
x x x          0  
1 2 3          1  
2 2 4          3  
>
```