

# Языки описания схем

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

## Блок 22

Verilog:

Основные процедурные команды

Использование задержек

Лектор:

**Подымов Владислав Васильевич**

E-mail:

**valdus@yandex.ru**

ВМК МГУ, 2024/2025, осенний семестр

# Напоминание нескольких команд

Составная команда:

```
begin <последовательность команд> end
```

*Выполнение:* команды последовательно выполняются, каждая следующая — немедленно после завершения предыдущей

Завершение симуляции:

```
$finish;
```

*Выполнение:* симуляция завершается

Блокирующее присваивание:

```
<переменная> = <выражение>;
```

*Выполнение:* значение *выражения* вычисляется и немедленно присваивается в *переменную*, и выполнение процедуры продолжается **после** присваивания

# Неблокирующее присваивание

```
<переменная> <= <выражение>;
```

**Выполнение:** вычисляется значение *val* выражения, и в текущем регионе планируется отложенное присваивание *val* в переменную

## Пример:

```
reg x, y;  
initial begin x = 0; y = 1; x = y; y = x; end
```

После выполнения всех присваиваний значения *x* и *y* — это 1 и 1

```
reg x, y;  
initial begin x = 0; y = 1; x <= y; y <= x; end
```

После выполнения всех присваиваний значения *x* и *y* — это 1 и 0

## Использование задержек

```
#<число> <команда>
```

Выполнение *команды* планируется в группу неактивных действий региона  $\tau + \text{число}$  (вместо группы активных текущего региона)

```
<переменная> = #<число> <выражение>;
```

- ▶ Немедленно вычисляется значение *val* *выражения*
- ▶ Планируется неактивное присваивание *val* в *переменную* в  $\tau + \text{число}$
- ▶ Процедура продолжает выполняться только сразу после присваивания

```
<переменная> <= #<число> <выражение>;
```

Вместо текущего региона отложенное присваивание планируется в регион  $\tau + \text{число}$

```
assign #<число> <переменная> = <выражение>;
```

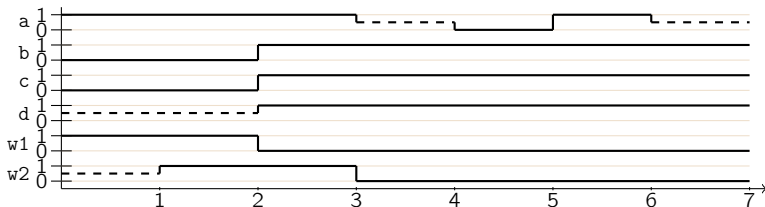
(Это не совсем команда, но здесь тоже можно писать задержку)

При изменении значения *выражения* присваивание планируется не в текущий регион, а в регион  $\tau + \text{число}$

# Использование задержек: бессмысленный пример

```
reg a, b, c, d;  
wire w1, w2;  
assign w1 = !b;  
assign #1 w2 = !b;  
initial begin a = 1; b = 0; end  
initial begin  
    c = 0; #1 a <= #3 0; b = #1 1; c = a; d = b;  
    #1 a = 1'bx; #3 a = 1'bx;  
end  
initial begin  
    #5 a = 1; #2 $finish;  
end
```

Этот код отвечает такому сценарию выполнения:



## Вывод в консоль и пустая команда

```
$display(...);
```

*Выполнение:* немедленно производится вывод в консоль согласно формату и аргументам

```
$monitor(...);
```

*Выполнение:* при каждом изменении значений, указанных в аргументах, планируется отслеживающее действие вывода в консоль согласно формату и аргументам (один раз для региона)

```
$strobe(...);
```

Полностью аналогично `$display`, только планируется не активное действие, а отслеживающее

**Пустая команда:**

```
;
```

*Выполнение:* ничего не происходит

# Команда ветвления

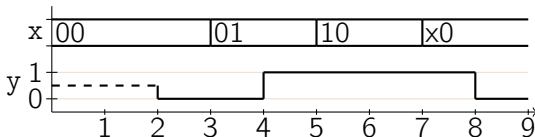
```
if(<выражение>) <команда> else <команда>
```

Выполнение:

- ▶ Вычисляется значение *val* выражения
- ▶ Если *val* — это (00...0) или содержит разряды  $\mathcal{X}$  или  $\mathcal{Z}$ , то выполняется правая команда
- ▶ Иначе выполняется левая команда

## Пример

```
reg [1:0] x;  
reg y;  
always #2 if(x) y = 1; else y = 0;  
initial begin  
    x = 0; #3 x = 2'b1; #2 x = 2'b10;  
    #2 x = 2'bx0; #2 $finish;  
end
```



# Команда точного выбора

```
case(<выражение>
  <последовательность случаев>
  default : <команда>
endcase
```

<случай> ::= <выражение> : <команда>

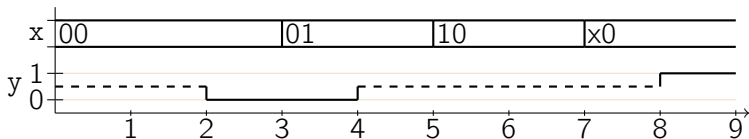
## Выполнение:

- ▶ Вычисляется значение *val* выражения в скобках
- ▶ Последовательно по порядку до **успеха** обрабатываются случаи:
  - ▶ Вычисляется значение *sva* выражения случая
  - ▶ Если *sva* поразрядно совпадает с *val*, то **успех**, и выполняется команда случая
- ▶ Если все случаи обработаны неуспешно, то выполняется команда после слова «default»



## Команда точного выбора: пример

```
reg [1:0] x; reg y;
always #2
  case(x)
    2'b00: y = 0;
    2'bx0: y = 1;
    default: y = 1'bx;
  endcase
initial begin
  x = 0;
  #3 x = 2'b1;
  #2 x = 2'b10;
  #2 x = 2'bx0;
  #2 $finish;
end
```



## Команда выбора по шаблону

```
caseх(<выражение>
  <последовательность случаев>
  default : <команда>
endcase
```

*Выполнение* отличается от выполнения case только критерием успеха

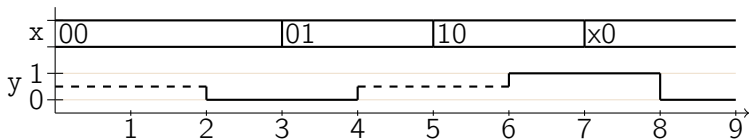
**Успех** ⇔

для каждой пары соответствующих разрядов значений *val*, *cval*  
выражения в скобках и выражения случая верно следующее:

**если значения обоих разрядов булевы**, то эти значения совпадают

## Команда выбора по шаблону: пример

```
reg [1:0] x; reg y;
always #2
  casex(x)
    2'b00: y = 0;
    2'bx0: y = 1;
    default: y = 1'bx;
  endcase
initial begin
  x = 0;
  #3 x = 2'b1;
  #2 x = 2'b10;
  #2 x = 2'bx0;
  #2 $finish;
end
```



## Ещё немного о ветвлении и выборе

```
if(<выражение>) <команда>
```

Этот код эквивалентен коду

```
if(<выражение>) <команда> else ;
```

(как в C/C++)

```
casex(<выражение>)  
  <послед. случаев>  
endcase
```

```
casex(<выражение>)  
  <послед. случаев>  
endcase
```

Здесь подразумевается добавка

```
default: ;
```

(как в C/C++)

Выражения случаев не обязаны быть константами

Слева от «:» в случае можно написать несколько выражений через «,», подразумевается несколько случаев с одинаковой командой