

# SystemVerilog

Лекционное занятие



# ТИПЫ ДАННЫХ



# Представление данных в SystemVerilog

```
// Двоичное значение 8
4'b1000;

// Шестнадцатеричное значение 8
4'h8;

// Восьмеричное значение 8
4'o10;

// Десятичное значение 8
4'd8
```

Общий синтаксис представления цифровых данных в SystemVerilog.

Состояние	Описание
0	Двоичное значение 0
1	Двоичное значение 1
z	Высокое значение импеданса
x	Неизвестное значение

В Verilog отдельным битам данных можно назначить четыре различных состояния.

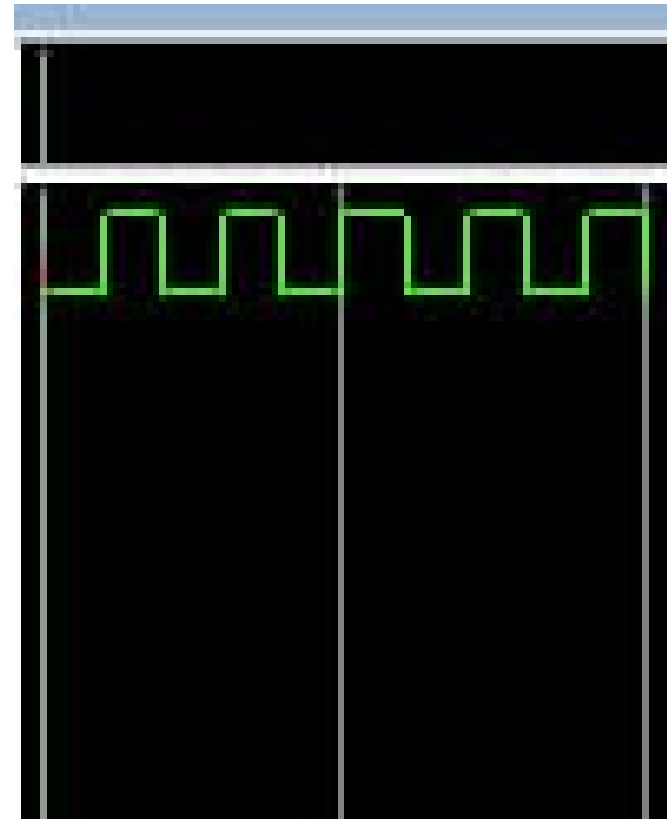
# Существующие типы данных

- **reg** – элемент хранения данных, который способен сохранять присвоенное значение (работает как переменная в языках программирования);
- **wire** – элемент, который можно рассматривать как соединения различных элементов не хранящих в себе значения;
- **logic** – элемент, который выбирается автоматически (**wire** / **reg**) [ не рекомендован к использованию в процессе обучения]

# Типы данных в SystemVerilog

Пример использования reg

```
module moduleName;  
  reg clk;  
  
  initial  
    clk=0;  
  always  
    #10 clk=~clk;  
  
endmodule
```



# Типы данных в SystemVerilog

Пример использования wire

```
3 module and_or_not_sv
4 (
5     input [1:0] key,      //входная 2 разрядная шина
6     output [2:0] led      //выходная 3 разрядная шина
7 );
8
9     wire a = ~ key[0];    //провод с инвертированным сигналом key[0]
10    wire b = ~ key[1];    //провод с инвертированным сигналом key[1]
11
12    assign led[0] = ~ a;   //присваиваем выходному [0] проводу сигнал НЕ a
13    assign led[1] = a | b; //присваиваем выходному [1] проводу сигнал a | b
14    assign led[2] = a & b; //присваиваем выходному [2] проводу сигнал a & b
15
16 endmodule
```

# Типы данных в SystemVerilog

Язык описания аппаратуры  
SystemVerilog

Тип	Описание	Число состояний
wire	Используется для моделирования соединений (тип Verilog)	4
reg	Используется в последовательных блоках (тип Verilog)	4
logic	Замена в SystemVerilog для reg и wire	4
bit	Эквивалент состояния логического типа (тип SystemVerilog)	2
integer	32-разрядный знаковый тип для моделирования целых чисел (тип Verilog)	4
int	Эквивалент состояния целому числу (тип SystemVerilog)	2
byte	8-битный знаковый тип для моделирования целых чисел (тип SystemVerilog)	2
shortint	16-битный знаковый тип для моделирования целых чисел (тип SystemVerilog)	2
longint	64-разрядный знаковый тип для моделирования целых чисел (тип SystemVerilog)	2
real	64-разрядный номер двойной точности (тип SystemVerilog)	2
shortreal	32-разрядное число одинарной точности (тип SystemVerilog)	2
time	64-битное число без знака, представляющее время моделирования (тип Verilog)	4

# ВВЕДЕНИЕ В ОПЕРАТОРЫ





# Арифметические операторы

Предназначены для выполнения базовых математических функций над переменными.

// Возвращает значение a плюс b

$y = a + b;$

// Возвращает значение a минус b

$y = a - b;$

// Возвращает значение a, умноженное на b

$y = a * b;$

// Возвращает значение a, деленное на b

$y = a / b;$

// Возвращает модуль a, деленный на b

$y = a \% b;$

// Возвращает значение a в степени b

$y = a ** b;$

Оператор	Описание
+	Дополнение
-	Вычитание
*	Умножение
/	Деление
%	По модулю
**	Возведение в степень

# Побитовые операторы

```
// Возвращает значение не a
y = ~a;

// Возвращает значения a и b
y = a & b;

// Возвращает значение a или b
y = a | b;

// возвращает значение a или b
y = a ^ b;

// возвращает значение a или b
y = a ~^ b;
```

**Побитовые операторы:** если операнд один, то дают 1 бит, если два, то разрядность результата равна длине большего операнда.

Оператор	Описание
~	Побитовое НЕ (NOT)
&	Побитовое И (AND)
	Побитовое ИЛИ (OR)
^	Побитовое исключающее ИЛИ (XOR)
~^	Побитовое исключающее НЕ ИЛИ (XNOR)

# Побитовые операторы. Примеры

```
2  module pobit_tb #()();
3      logic [3:0] a = 4'b1100;
4      logic [3:0] b = 4'b1010;
5
6      initial begin
7          $display("a = %b", a );
8          $display("b = %b", b );
9          $display();
10         $display("result a | b = %b", a | b );
11         $display("result a & b = %b", a & b );
12         $display("result a ^ b = %b", a ^ b );
13         $display("result ~(a ^ b) = %b", ~(a ^ b) );
14         $display("result ~(a|b) = %b", ~(a | b ));
15         $display();
16         $display("result (a ~| b) = Operator '~&' is not supported by default");
17         $display(); // Операторы редукции (сокращения)
18         $display("resolt  &a = %b", &a);
19         $display("resolt  |a = %b", |a);
20         $display("resolt  ^a = %b", ^a);
21         $display("resolt ~&a = %b", ~&a);
22         $display("resolt ~|a = %b", ~|a);
23         $display("resolt ~^a = %b", ~^a);
24
25     end
26 endmodule
27 //Если операнды имеют неодинаковую длину, более короткий операнд заполняется нулями
28 //в позициях старших бит.
```

```
# vsim work.pobit_tb -do "run -all; quit"
# Start time: 11:16:53 on Oct 24,2024
# Loading sv_std.std
# Loading work.pobit_tb
# run -all
# a = 1100
# b = 1010
#
# result a | b      = 1110
# result a & b      = 1000
# result a ^ b      = 0110
# result ~(a ^ b)   = 1001
# result ~(a|b)     = 0001
#
# result (a ~| b) = Operator '~&' is not supported by default
#
# resultt  &a = 0
# resultt  |a = 1
# resultt  ^a = 0
# resultt ~&a = 1
# resultt ~|a = 0
# resultt ~^a = 1
# quit
```

```
1 vlog -work work .\pobit_tb.sv
2 vsim work.pobit_tb -do "run -all; quit"
```

# Реляционные операторы

```
// Возвращает значение не a
```

```
y = a > b;
```

```
// Возвращает значения a и b
```

```
y = a >= b;
```

```
// Возвращает значение a или b
```

```
y = a < b;
```

```
// Возвращает значение ни a, ни b
```

```
y = a <= b;
```

```
// Возвращает значение ни a, ни b
```

```
y = a == b;
```

```
// возвращает значение a или b
```

```
y = a != b;
```

Использование операторов отношения предназначается для сравнения значений двух разных переменных. Результат этого сравнения возвращает либо значение логическое "1" , либо "0" , представляющее **true** и **false** соответственно.

Оператор	Описание
>	Больше , чем
>=	Больше или равно
<	Меньше , чем
<=	Меньше или равно
==	Равно
!=	Не равнозначно

# Логические операторы

Логические операторы аналогичны побитовым операторами, предназначены для объединения операторов отношения. В результате становится возможным создавать более сложные выражения, которые могут выполнять более одного сравнения.

// Возвращает 1, если a равно b, И c равно d

```
y = (a == b) && (c == d);
```

// Возвращает 1, если a равно b ИЛИ a равно c

```
y = (a == b) || (a == c);
```

// Возвращает 0, если a равно b

```
y = !(a == b);
```

Оператор	Описание
&&	Логическое И
	Логическое ИЛИ
!	Логическое НЕ

# Операторы сдвига

Оператор сдвига требует двух аргументов.

Первый - это название сигнала, который нужно сдвинуть.

Второй аргумент - это количество битов, на которое нужно сдвинуть.

При логическом сдвиге на требуемое количество бит, все пустые позиции заполняются значением 0b.

Арифметический сдвиг сохраняет знак сдвинутого сигнала.

```
// Сдвиг сигнал a влево на 3 бита
```

```
a = a << 3;
```

```
// Сдвиг сигнал b вправо на 8 бит
```

```
b = b >> 8;
```

```
// Сдвиг сигнал a влево на 3 бита
```

```
c = c <<< 3;
```

```
Сдвиг сигнала d вправо на 5 бит
```

```
d = d >>> 5;
```

Оператор	Описание
<<	Логический сдвиг влево
>>	Логический сдвиг вправо
<<<	Арифметический сдвиг влево
>>>	Арифметический сдвиг вправо

# Операторы конкатенации и репликации

В обоих случаях выходные данные этих операторов имеют векторный тип. Однако входные данные для обоих этих операторов могут быть либо однобитовыми, либо векторными типами.

**Оператор конкатенации** объединяет два или более сигналов в вектор. Биты в выходных данных соответствуют порядку, в котором они перечислены внутри скобок.

**Оператор репликации** присваивает одно и то же значение некоторому количеству битов в векторе.

Необходимо указать сигнал или значение, которое нужно реплицировать, так и количество раз, которое будет его реплицировать.



# Операторы конкатенации и репликации

```
// Объединить сигналы a и b в вектор, используя конкатенацию  
c = {a, b};  
// Повторить сигнал "c" 3 раза  
d = { 3{c} };
```

Оператор	Описание
{ }	Оператор конкатенации
{ { } }	Оператор репликации

# Непрерывное **назначение**

В SystemVerilog возможно использовать два разных метода для реализации непрерывного присваивания:

- **явное непрерывное присвоение**;
- **неявное непрерывное присваивание**.

Непрерывное присвоение используется либо с **logic** типом, либо с сетевыми типами, такими как **wire**.

# Явное непрерывное присвоение

Используют ключевое слово `assign`,

```
assign <variable> = <value>;
```

```
assign a = b;
```

Поле `<переменная>` указывает имя сигнала, которому присваиваются данные.

Поле `<значение>` может быть фиксированным значением.

При непрерывном присваивании, значение `<переменной>` изменяется всякий раз, когда один из сигналов в поле `<значение>` меняет состояние.

# Неявное непрерывное присваивание

Неявное непрерывное присваивание помещается в оператор, который объявляет сигнал. Это может позволить сократить объем кода.

Для этого используют символ `=` для присвоения значения сигналу при его объявлении.

```
<type> <variable> = <value>;
```

```
logic a = b;
```

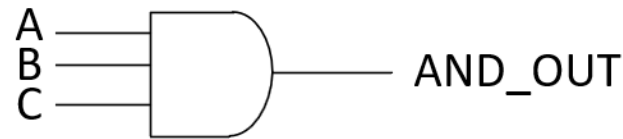
Поля `<переменная>` и `<значение>` выполняют одну и ту же функцию для явного и неявного непрерывного присвоения.

В качестве примера приведен код неявного непрерывного присвоения, который позволяет присвоить значение `b` сигналу `a`.

# Комбинационные логические **схемы**

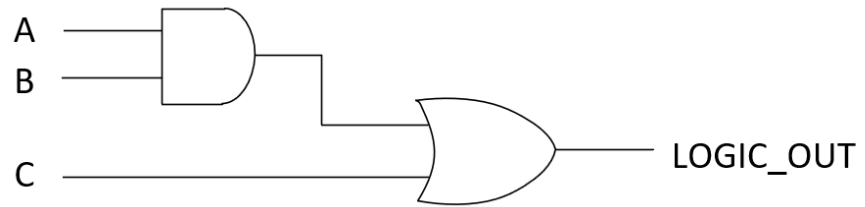
Для моделирования схемы с тремя входами и вентилями используют ключевое слово **assign** для передачи данных на вывод **and\_out** и побитовый оператор **and (&)** .

```
assign and_out = a & b & c;
```



Для построения более сложных комбинационных схем, можно использовать смесь различных побитовых операторов.

```
assign logic_out = (a & b) | c;
```



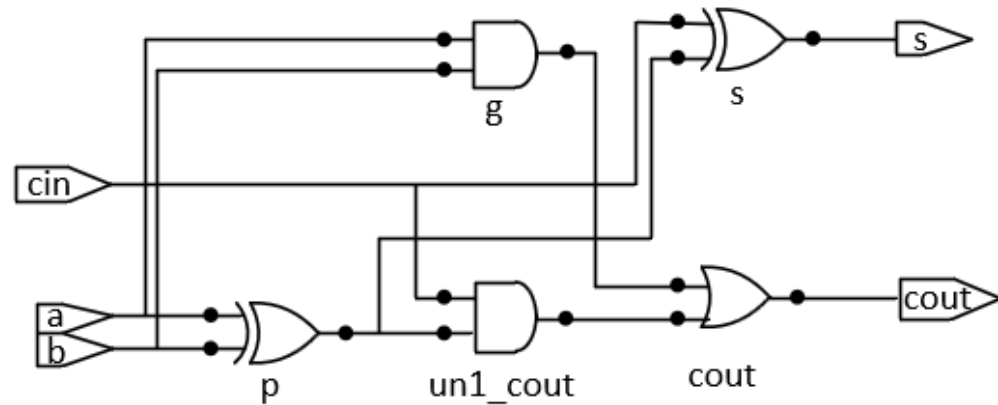
# Внутренние переменные

```

module fulladder(
    input  logic  a, b, cin,
    output logic  s, cout );

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

ПОЛНЫЙ СУММАТОР



Синтезированная схема модуля  
fulladder

# Приоритет

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

Для определения порядка операций, используют приоритет:

$C_{out} = G + P \cdot C_{in}$ , а не  $C_{out} = (G + P) \cdot C_{in}$ .

# Условный оператор

Когда выражение, указанное в поле **<условие>**, вычисляется как true, то выходным данным присваивается значение, указанное в поле **<истина>**.

Если условное выражение вычисляется как false, то выходным данным присваивается значение, заданное полем **<ложь>**.

```
output = <condition> ? <true> : <false>
```

```
// Присвоение a значению c, когда оно больше, чем b
```

```
a = c > b ? c : b;
```



# Условное присваивание

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

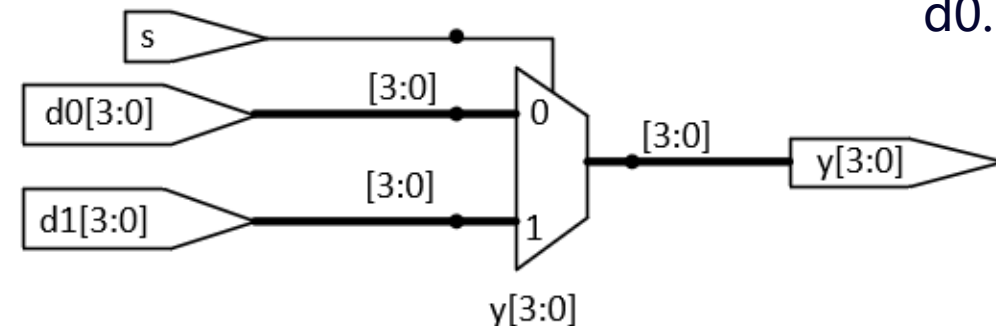
ДВУХВХОДОВОЙ МУЛЬТИПЛЕКСОР

Операторы условного присваивания выбирают определенный выход среди других, исходя из состояния входа, называемого **условие**.

## Содержание модуля

Оператор **?** – выбирает между вторым и третьим выражениями.

Если **s** равно 1, то  $y = d1$ , иначе  $y = d0$ .



Синтезированная схема модуля mux2

# Условное присваивание

```
module mux4(  
    input  logic [3:0]    d0, d1, d2, d3,  
    input  logic [1:0]    s,  
    output logic [3:0]    y);  
    assign y = s[1] ? (s[0] ? d3 : d2) : (s[0] ? d1 : d0);  
endmodule
```

ЧЕТЫРЕХВХОДОВОЙ МУЛЬТИПЛЕКСОР

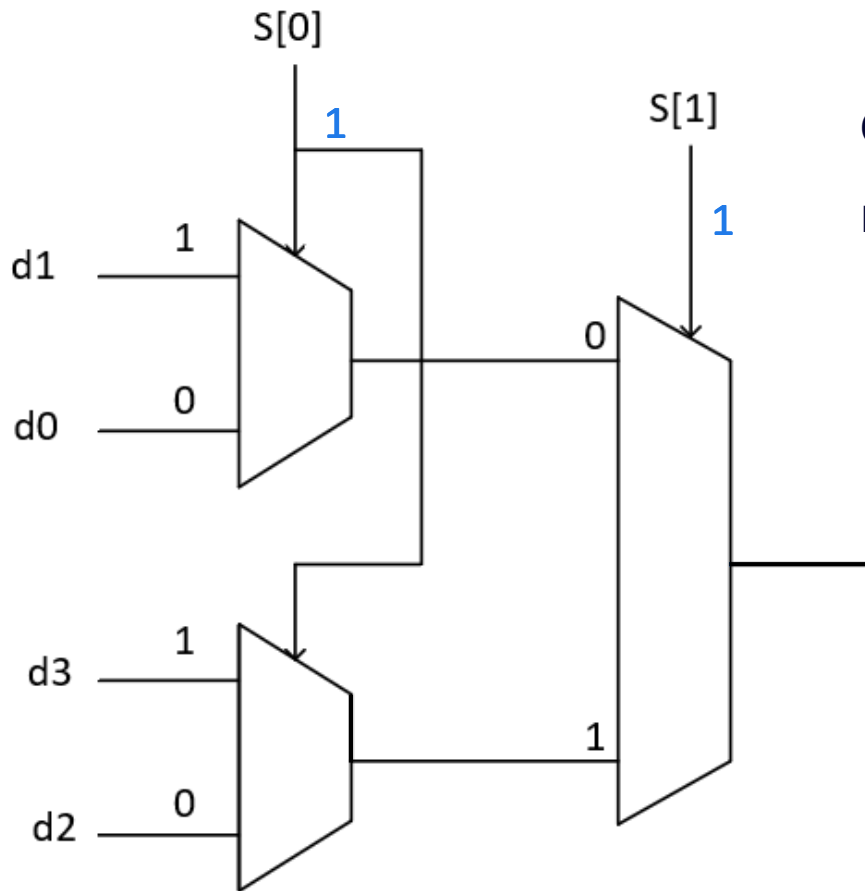
## Содержание модуля

Если  $s[1]$  равняется 1, тогда мультиплексор выбирает  $(s[0] ? d3 : d2)$ .

Выражение выбирает  $d3$  или  $d2$  на основе  $s[0]$  ( $y = d3$ , если  $s[0] = 1$  и  $d2$ , если  $s[0] = 0$ ).

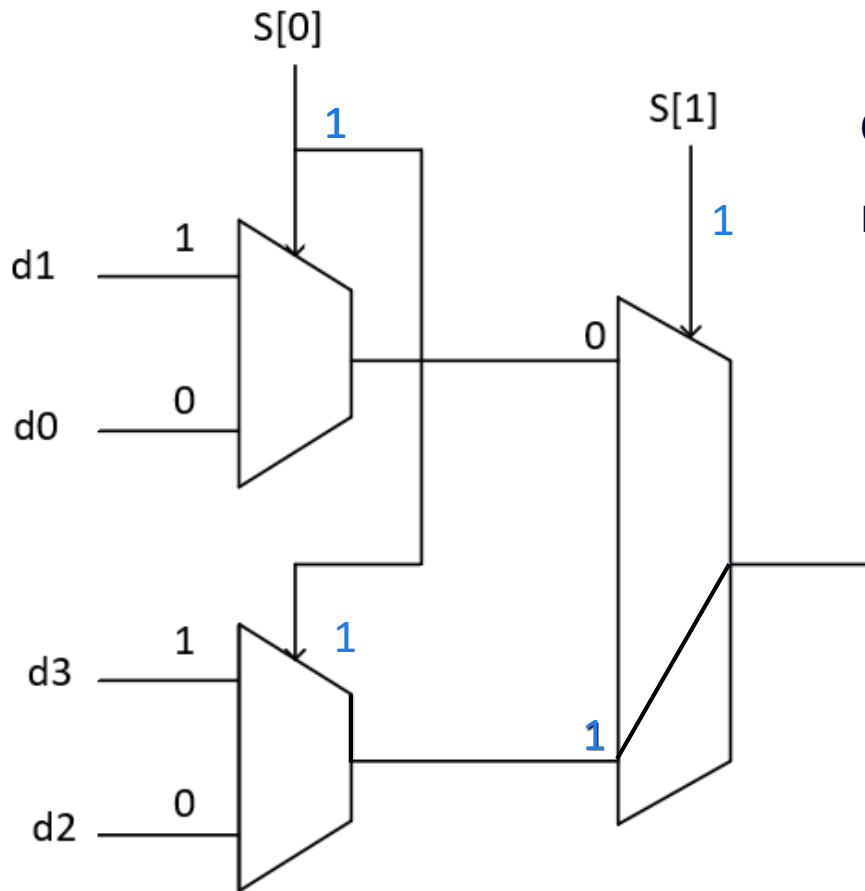
Иначе, если  $s[1] = 0$ , выбирает второе выражение, которое дает или  $d1$ , или  $d0$ .

# Условное присваивание



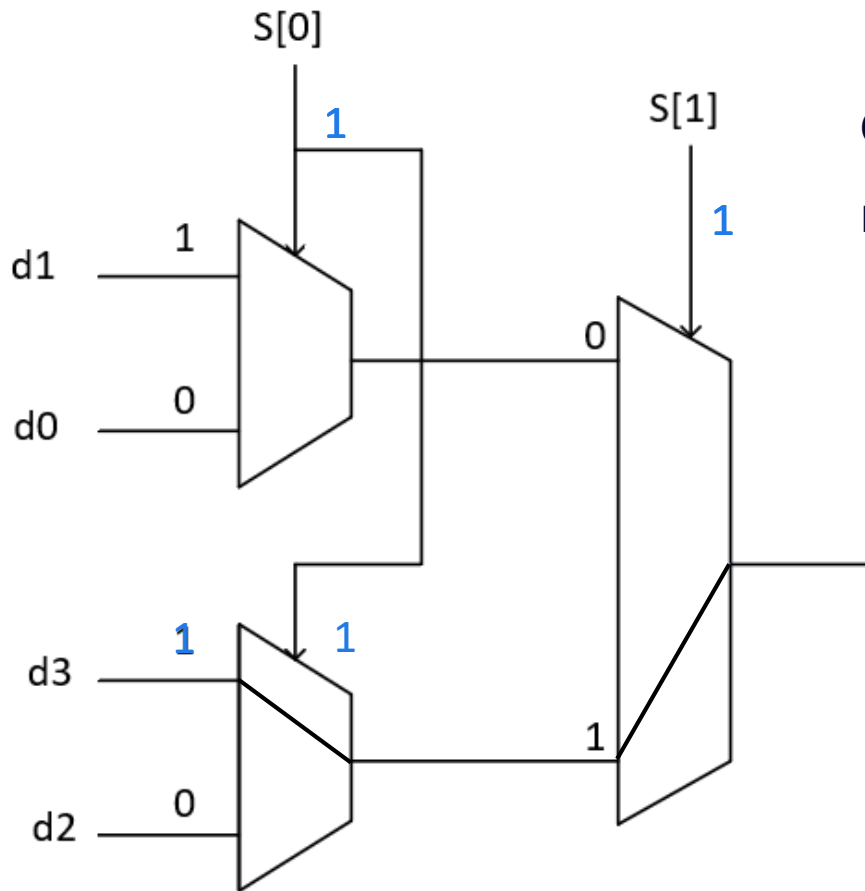
Синтезированная схема модуля  
mux4

# Условное присваивание



Синтезированная схема модуля  
mux4

# Условное присваивание



Синтезированная схема модуля  
mux4

# Комментарии и пробелы

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

Комментарии, начинающиеся с `//`, продолжаются до конца строки.

SystemVerilog чувствителен к регистру символов (прописным и строчным буквам).

`y1` и `Y1` – это разные сигналы. Использование множества сигналов, отличающихся только регистром символов, вносит путаницу.