

Пакеты проектирования сверхбольших интегральных схем

Лекция 3

Debug

Рассказывает:

Подымов Владислав Васильевич

e-mail:

valdus@yandex.ru

Осень 2016

Как вы могли убедиться...

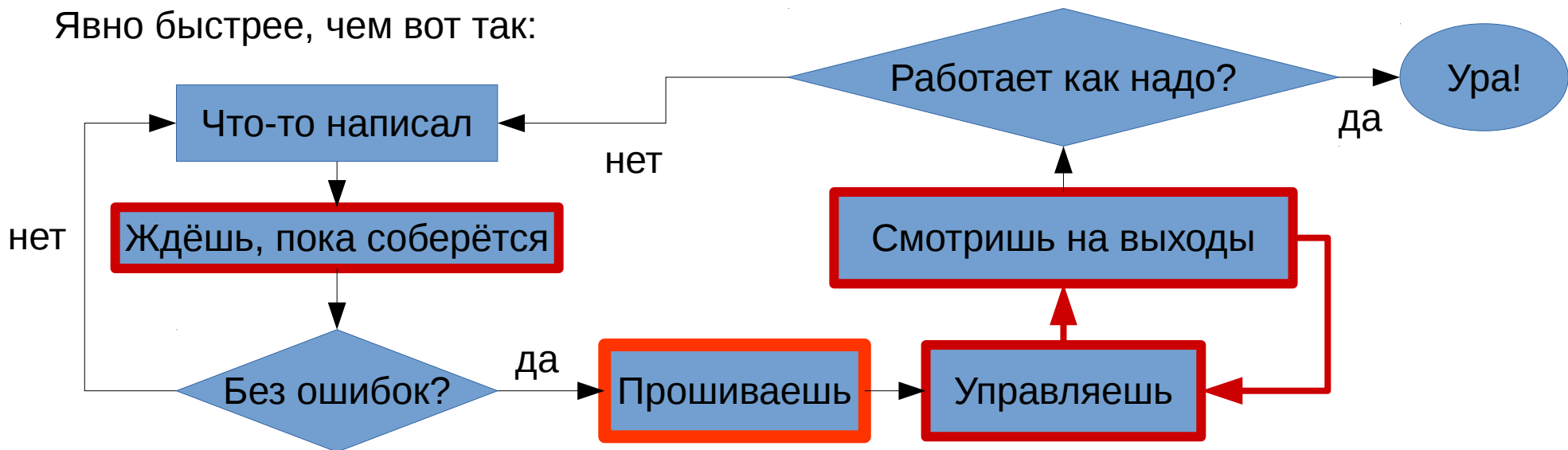
Даже если вы сделали так, что схема успешно компилируется и размещается, она скорее всего не работает как надо с первого раза

Ошибок никак не избежать

Чтобы легче работалось, их нужно уметь **быстро** отлавливать и исправлять

А насколько быстро?

Явно быстрее, чем вот так:



А если нет возможности залить всё на плату, то эта схема вообще не работает

Debug программ

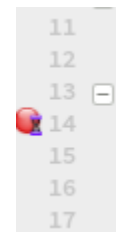
Локализация ошибки,
вычитывание кода

```
for(int i = 0; i < 5; ++j)  
    sum = sum + i;
```

Ну конечно, тут же должно быть
“++i”, а не “++j”!

Отладчик

- Поставил breakpoint'ы



```
11  
12  
13  
14  
15  
16  
17  
  
char c_get;  
if(!is.get(c_get)) return false;  
if(c_get != c) {  
    is.unget(c_get);  
    return false;  
}  
return true;
```

- Запустил отладчик
- Посмотрел **промежуточные** значения

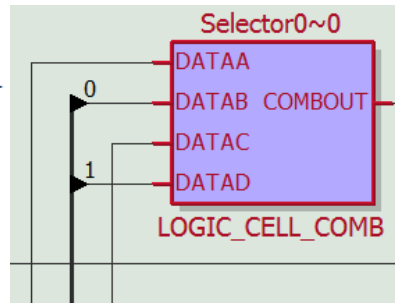
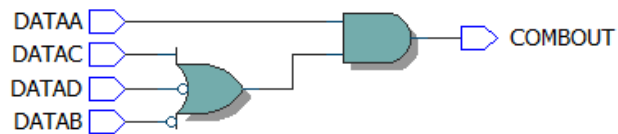
Debug схем

Локализация ошибки,
вычитывание кода

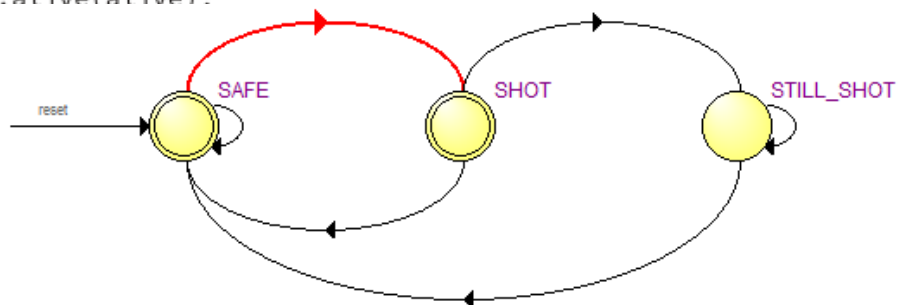
```
killer k(  
  .kill(kill),  
  .door(active_door),  
  .who(SW[3:2]),  
  .kill1(OT1[2]),  
  .door1(OT1[1:0]),  
  .kill2(OT2[2]),  
  .door2(OT2[1:0])  
);
```

← Не помогает

Тогда можно
посмотреть на схему



Или, например,
на диаграмму Мура



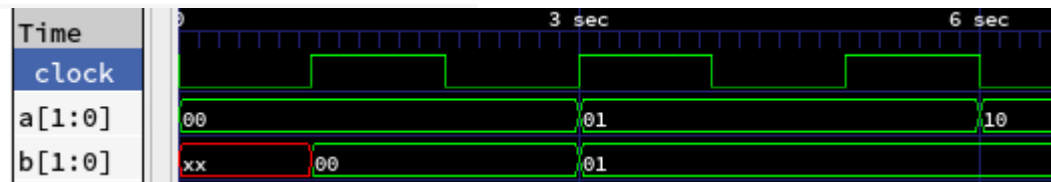
Отладчик

```
module testbench;  
  reg [1:0] a;  
  reg clock;  
  wire [1:0] b;  
  top t(.a(a), .b(b), .clock(clock));  
  always  
    #1 clock = ~clock;  
  initial  
  begin  
    clock = 0;  
    a = 0;  
    #3  
    a = 1;  
    #3  
    a = 2;  
    #3  
    a = 0;  
    #3  
    $finish;  
  end  
  initial  
  begin  
    $dumpfile("out.vcd");  
    $dumpvars(0,test);  
  end  
  initial  
    $monitor($stime,, a,, b,, clock);  
endmodule
```

Написать модуль
тестирования
(testbench)

Запустить симулятор

Посмотреть на
нужные сигналы



Как смотреть схему



Что такое testbench

testbench – это модуль на языке verilog

У этого модуля нет аргументов; в него вставляется тестируемый модуль, и все входы-выходы этого модуля выводятся в **reg** и **wire**

Verilog содержит ряд инструкций, которые **могут игнорироваться** компилятором, но содержат информацию, полезную для отладки и симуляции

Среди таких инструкций:

- отладочный вывод
- вывод значений переменных вдоль конкретной трассы выполнения схемы
- вывод значения текущего времени от начала работы схемы

А вывод куда?

- В консоль при работе средства симуляции
- В файл специального формата (.vcd), из которого можно получить
наглядное представление того, как изменяются сигналы во времени

А сигналы какие?

Какие хотим, вплоть до всех переменных всех модулей проекта

Как писать testbench

Типовой набор команд и конструкций, который можно использовать в testbench'е:

Инициализация

В реальной схеме **нет** единого момента времени, с которого начинается работа

У симулятора есть такой момент: 0 секунд

Можно задавать значения сигналов в начальный момент времени:

```
reg [2:0] a = 3'b001;
```

```
reg b;
```

```
reg clock;
```

```
initial
```

```
begin
```

```
  b = 1;
```

```
  clock = 0;
```

```
end
```

Обязательно "reg"!



Функциональный блок, запускающийся в момент времени 0



Блокирующее присваивание, ...



Как писать testbench

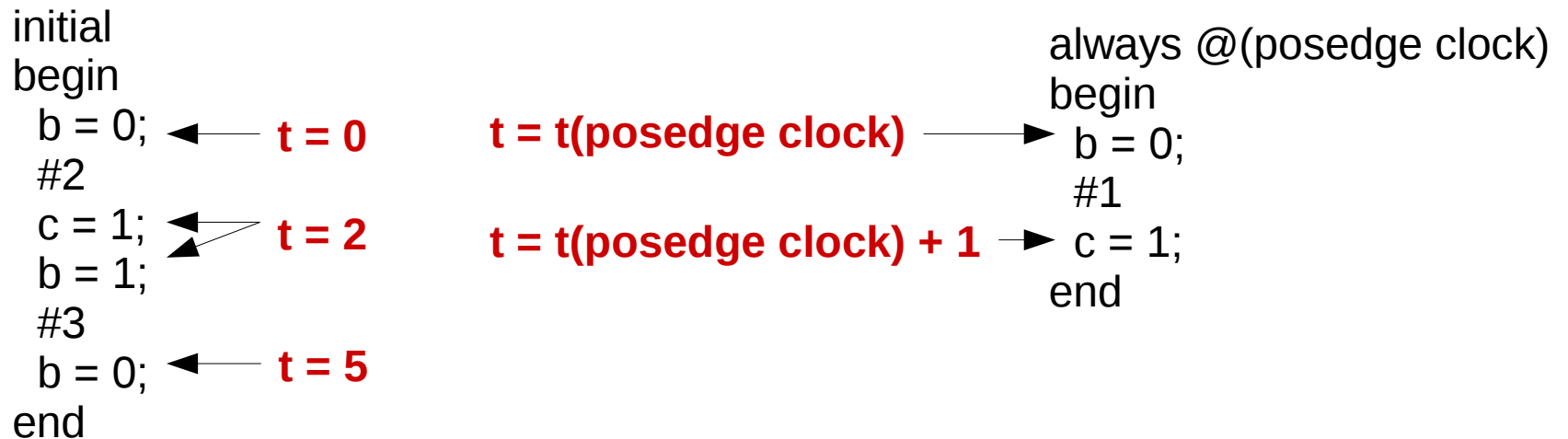
Типовой набор команд и конструкций, который можно использовать в testbench'е:

Контроль временных интервалов

Симулятор предполагает, что операции, записанные в функциональных блоках

(always, initial)

происходят **одновременно**; в симуляторе можно явно разнести эти операции во времени:



“#i” означает “всё, что дальше, происходит через i единиц времени”

Как писать testbench

Типовой набор команд и конструкций, который можно использовать в testbench'е:

“Хитрый” always-блок

```
always
begin
  b = 0;      t = 0      t = 5      t = 10
  #1
  c = 1;      t = 1      t = 6      t = 11      ...
  #1
  b = 1;      t = 2      t = 7      t = 12
  #1
  c = 0;      t = 3      t = 8      t = 13
  #2
end
```

Такой always-блок выполняется **всегда**

В нём явно должен быть указан
хотя бы один ненулевой оператор продвижения времени

Как писать testbench

Типовой набор команд и конструкций, который можно использовать в testbench'е:

“Хитрый” always-блок

В частности, вот так можно моделировать тактовый сигнал:

```
reg clock = 0;  
always  
#1  
clock = ~clock;
```

Начинаем со значения clock = 0



Каждую единицу времени
меняем значение clock на противоположное



Как писать testbench

Типовой набор команд и конструкций, который можно использовать в testbench'e:

Отладочная печать

`$display("format", args...)` ← **Полный аналог printf в C/C++**

`$monitor(var1, var2, ...)` ← **Когда изменяется хотя бы одно из значений переменных `vari`, делать `$display` строки значений переменных**

`$monitor(var1,, var2,,, var3)` ← **Если между запятыми ничего нет, то в этом месте печатается пробел**

А куда это всё выводится?

Симулятор – это обычная программа с обычным потоком вывода

В этот поток всё и выводится

Как писать testbench

Типовой набор команд и конструкций, который можно использовать в testbench'е:

Текущее время

\$time, \$stime, \$realtime

Это переменные, в которых хранится значение текущего времени:

- в формате int_64
- в формате uint_32
- в формате float

Эти переменные можно использовать, например, так:

```
$monitor($stime,, a,, b)
```

Тогда вывод производится *как минимум* в каждый новый момент времени

Как писать testbench

Типовой набор команд и конструкций, который можно использовать в testbench'е:

Конец симуляции

`$finish`

Это команда, завершающая симуляцию

Например:

```
...  
initial  
  #100  
  $finish;  
...
```

Завершить симуляцию
через 100 единиц времени

```
...  
initial  
begin  
  #1 b = 0;  
  #1 c = 1;  
  #3 $finish;  
end  
...
```

t = 1: сделать b = 0
t = 2: сделать c = 1
t = 5: завершить симуляцию

Как писать testbench

Типовой набор команд и конструкций, который можно использовать в testbench'е:

Генерация осциллограмм

(так можно перевести “*waveform*”)

`$dumpfile(“file”)`

С выполнения этой команды устанавливается имя файла file:
в него по другим командам будут записываться осциллограммы

`$dumpvars(level, objlist)`

С выполнения этой команды в установленный файл начинает записываться информация об изменении переменных, достаточная для создания осциллограммы
objlist – список имён переменных и модулей

- level = 0: для каждого модуля отслеживаем все его переменные и переменные используемых в нём экземпляров
- level = 1: не отслеживаем переменные экземпляров

`$dumpoff` – выключить запись изменения значений переменных

`$dumpon` – включить запись обратно после `$dumpoff`

Как работать с симулятором



- *Quartus, наверху кнопка “RTL simulation”*
- *Icarus Verilog: iverilog + vvp + gtkwave*