

ИСПОЛЬЗОВАНИЕ БЛОКА **ALWAYS** (всегда)

Лекционное занятие



Блок **Always**

Каждый раз, когда один из сигналов в списке чувствительности изменяет состояние, все инструкции в блоке **always** выполняются последовательно.

```
always @(<sensitivity_list>) begin  
// Выполнение какого-нибудь кода  
end
```

Списки чувствительности

После выполнения последней строки кода в блоке, программа возвращается к началу блока `always` .

Однако, реальная схема, будет оставаться в устойчивом состоянии до тех пор, пока один из входных сигналов не изменит состояние.

Для эмуляции такого поведения, используют **список чувствительности** в блоке `always`.

Код в блоке `always` будет выполняться только после того, как один из сигналов в **списке чувствительности** изменит состояние.

Списки чувствительности

Выходные данные триггера D-типа меняют состояние только при положительном фронте синхронизации.

Таким образом, включаем тактовый сигнал в список чувствительности, чтобы блок `always` выполнялся только по положительному фронту.

```
always @(posedge clock) begin
    q <= d;
end
```

Списки чувствительности

```
always @(posedge clock) begin  
    q <= d;  
end
```

Использование **posedge** определяет, когда происходит переход от "0" к "1". Все остальные изменения состояния просто игнорируются.

Единственная строка кода в блоке **always** будет выполняться, когда этот макрос будет оценен как true. Эта строка кода присваивает значение D выходному сигналу (Q).

Также есть макрос **negedge**, который обладает противоположной функциональностью. Когда его используют, блок **always** будет выполняться всякий раз, когда часы изменяются с "1" на "0".

Несколько сигналов в списке чувствительности

```
always @(posedge clock, posedge reset) begin
    if (reset) begin
        q <= 1'b0;
    end
    else begin
        q <= d;
    end
end
```

Возникают случаи, когда необходимо включить более одного сигнала в список чувствительности, например моделирование поведения триггеров с асинхронными сбросами.

В этом случае нужна триггерная модель для выполнения действия всякий раз, когда сброс или тактовые сигналы меняют состояние.

Для этого достаточно перечислить оба сигнала внутри списка чувствительности.

Несколько сигналов в списке чувствительности

```
always @(posedge clock, posedge reset) begin
    if (reset) begin
        q <= 1'b0;
    end
    else begin
        q <= d;
    end
end
```

Сброс активен, когда он равен единице.

Затем используется конструкция **if**, чтобы определить, срабатывает ли блок **always** сигналом сброса или тактовым сигналом.

Блок `always_comb`

Блок `always_comb` применяется при моделировании комбинационной логической схемы.

Используя эту конструкцию, происходит приказ компилятору и инструментам синтеза о принятие более строгих правил.

Цель этих правил - помочь убедиться в правильности смоделированной логической схемы.

```
always_comb begin
// Код какой-то функции
end
```


Блок `always_comb`

При использовании блока `always_comb` компилятор автоматически генерирует список чувствительности, когда используют эту конструкцию.

Блок SystemVerilog `always_comb` примерно равен блоку `always @(*)` в verilog.

В этой конструкции символ `*` указывает инструментам verilog автоматически решать, какие сигналы включать в список чувствительности.

```
always_comb begin
    case (addr)
        0 : begin
            // Эта ветвь выполняется, когда addr = 0
            mux_out = a;
        end
        1 : begin
            // Эта ветвь выполняется, когда addr = 1
            mux_out = b;
        end
        2 : begin
            // Эта ветвь выполняется, когда addr = 2
            mux_out = c;
        end
        3 : begin
            // Эта ветвь выполняется, когда addr = 3
            mux_out = d;
        end
    endcase
end
```

Пример мультиплексора

Для проектирования базового мультиплексора 4 к 1 можно использовать оператор `case` .

Оператор `case` - это конструкция, которая может находиться только внутри процедурных блоков, таких как блок `always_comb`.

В оператор `case` можно включить столько различных ветвей, сколько потребуется.

Блок `always_latch`

Рекомендуют избегать использования защелок при проектировании.

Однако может возникнуть случай, когда это неизбежно, и необходимо реализовать защелку в проекте.

Когда нужно создать защелку используется блок `always_latch` .

```
always_latch begin
    if (enable) begin
        // Выполнение какого-нибудь кода
    end
end
```

Поле `<enable>` определяет, какой сигнал в конструкции будет управлять защелкой.

Как и в случае с блоком `always_comb` не нужно писать список чувствительности, когда используют блок `always_latch` . Компилятор автоматически сгенерирует список чувствительности.

Блок `always_ff`

Использование блока `always_ff` рекомендуют для реализации последовательной логической схемы.

В этом коде необходимо включить список чувствительности,

```
always_ff @(<sensitivity_list>) begin  
// Выполнение какого-нибудь кода  
end
```

В данном коде используют практически один и тот же синтаксис как для блока `always_ff`, так и для блока `always`.

Различия блоков `always` и `always_ff`

`always`

- Использование блокирующих и неблокирующих назначений сигнала.
- Реализует комбинационную и последовательностную логику.
- Присваивание переменной данных из нескольких блоков `always`.

`always_ff`

- Использование только неблокирующих назначений сигнала;
- Разработан специально для моделирования последовательных логических схем;
- Присваивание данных переменной только из одного блока `always_ff`.