# Verilog: базовый синтаксис

**Преподаватели: Эпштейн Леонид Борисович**

# Языки описания аппаратуры

**Языки описания аппаратуры** (Hardware Description Languages) – это тип компьютерных языков для формального описания электрических цепей, в особенности цифровой логики. Они описывают структуру и функционально поведение схемы

# Что такое HDL

Представь себе механизм, имеющий какую-либо связь с внешним миром. Как только снаружи приходит сигнал (нажали кнопку на пульте управления), внутри механизма что-то срабатывает, и им совершается действие (например, загорается лампочка). Для удобного описания всяких механизмов и причинно-следственных связей и были придуманы HDL-языки.

Совсем необязательно HDL описывает микросхему или ПЛИС.

Главная сложность, с которой сталкивается программист, начинающий писать на HDL, – ему необходимо осознать, что весь код будет в итоге исполняться одновременно. Как шестеренки в часах вращаются синхронно по событию маятника, так и операции внутри процессора выполняются сразу же, параллельно, успевая до наступления следующего такта!

В этом и есть отличие алгоритмических языков типа C или Pascal от HDL-подобных.

Если первые «указывают» некоторому абстрактному роботу-исполнителю последовательность действий, то вторые описывают внутренности самого «исполнителя», поведение которого будет зависеть от этих самых внутренностей.

# Базовый синтаксис: Комментарии, имена, константы, числа

**//Это комментарий**
**/***
**описание модуля**
***/**
В Verilog'е имя (идентификатор) - последовательность букв и цифр, знаков «$» и «_», причем начинаться оно обязано не с цифры
Verilog – чувствителен к регистру.

Константы в Verilog'е имеют особую форму записи. Сначала идет разрядность числа, потом кавычка ('), за ним - основание системы счисления (b, o, d, h) и сами цифры.
Примеры:
**«7'h7F» //семибитное число 127, записанное в шестнадцатеричной (h - hex) форме**
**«7'b1111_1111» //то же самое число, записанное в двоичной форме. Знак «_» игнорируется.**
**«10'b1111_1111» // число 127, занимающее не 7 бит, а 10. То есть, равно 000_1111_1111.**
**«18» // число, записанное в стандартной форме, будет приведено к десятичному Integer**
**«0.5» // будет приведено к типу float.**

# Базовый синтаксис: регистры (reg)

То, что обычно называется переменной, в Verilog'е называется регистром. Например:
reg [7:0] character;
Так мы объявили регистр шириной 8 бит (от нуля до семи) с именем character. Как и в переменную, в регистр можно класть значение и читать его оттуда:

```
reg [7:0] var1;
reg [15:0] var2 = 16'b1001_0110_1011_1101;
...skip...
var1 [7:0] = var2[15:8];
```

Здесь мы кладем в регистр var1 старшую половину регистра var2, который в два раза «шире». В итоге, там будет лежать число 8'b1001_0110, то есть 0x96h. Примечание: строго говоря, в Verilog'е тоже есть нормальные человеческие переменные, причем регистр - это переменная типа reg. Также бывают типы integer, time, real и другие. Но в ближайшее время это тебе не понадобится, поэтому считаем, что переменная в Verilog'е и есть регистр.
Кстати, массивы здесь тоже есть!

```
reg [2:6] Array [0:5]; //6 пятибитных векторов.
```

# Базовый синтаксис: сигналы (wire)

Так называемое «соединение-цепь».

Предположим, есть у нас некая переменная-регистр, к отдельным битам которой часто приходится обращаться по ходу действия. Введем сигнал, привязанный к одному биту переменной:

```
reg [7:0] device_config;
wire port_0_direction = device_config[0];
wire port_1_direction = device_config[1];
...Пропущено...
if (!port_0_direction)    device_data[0] <= par_port_0[7:0];
```

Связь port_0_direction теперь тождественно равна младшему биту регистра device_config[0]. Стало удобнее: не надо запоминать, что там и как, в большом регистре device_config, а главное – мы можем в нужный момент программной логикой перебросить этот сигнал на другой регистр:

```
port_0_direction = device_config[0] & device_config[2];
```

Теперь port_0_direction будет равен 1, только если единице равны 0 и 2 биты регистра device_config.

# Базовый синтаксис:
# Процессы always & initial

Процесс - это такой кусок кода, внутри которого все операции выполняются последовательно. Процесс может быть непрерывным, срабатывающим на какое-либо событие или вообще исполняемым ровно один раз для инициализации.

```
reg[7:0] counter;
always //always обязан содержать хотя бы
// один оператор ожидания, что мы и видим
@ (posedge Sysclock) //Событие без ";"
// является условием запуска следующей за ним опер. группы
begin
counter = counter + 1'b1;
end
```

Ключевое слово, указывающее на процесс - always. Затем идет так называемый оператор ожидания события «@» и список чувствительности в скобках (смотри врезку). А между словами begin и end - само тело процесса.

Как только значение переменной Sysclock сменится с 0 на 1, регистр counter увеличится на единичку. Всякие «нормальные» последовательные операторы ("=","<=","if","case" etc.) разрешено использовать только внутри процессов. Полный список операторов в языке Verilog указан во врезке.

Если вместо always стоит initial, то процесс выполнится однократно.

# Базовый синтаксис:

# Модули

Модули (module) в Verilog'e - это что-то типа черных ящиков или блоков обработки. Больше всего они похожи на функции в алгоритмических языках программирования. Как и функции, модули имеют входные и выходные параметры.

```
module Not (inputwire1, outwire1);
input inputwire1;
//Не указываю разрядность, значит однобитовые параметры
output outwire1;
reg outwire1;
always @(inputwire1)
outwire1<=!(inputwire1);
endmodule;
```

# Базовый синтаксис: Операции

- Сложение (+),
- Вычитание (-),
- Умножение (*), рекомендуется писать свою реализацию умножения
- Целочисленное деление (/),
- Модуль (%),
- Поразрядные И, ИЛИ, НЕ, XOR (&,|,~,^)
- Логические И, ИЛИ, НЕ (&&,||,!), дают однобитовый результат
- Операции отношения (==,!=,>,<,>=,<=)
- Сдвиги (>>,<<)

А также есть интересная операция «склеивания», то есть конкатенации:

```
reg[7:0] Lights=8'b0000_0001;
...skip...
Lights[7:0] = { Lights[0] , Lights[6:1] };
```

То, что справа – имеет ширину 8 бит и склеено из нулевого и оставшихся семи бит переменной Light