V.III Основы языка Verilog HDL

Verilog был разработан фирмой Gateway Design Automation в 1984 г

Стандарт Verilog LRM (Language Reference Manual), IEEE1364-1995 принят в 1995 году

Verilog и VHDL

•VHDL обладает большей универсальностью и может быть использован не только для описания моделей цифровых электронных схем, но и для других моделей.

•Из-за своих расширенных возможностей VHDL проигрывает в эффективности и простоте, то есть на описание одной и той же конструкции в Verilog потребуется в 3–4 раза меньше символов (ASCII), чем в VHDL.

•Как и VHDL, Verilog изначально предназначался для моделирования цифровых систем и как средство описания синтезируемых проектов стал использоваться с 1987 г. В настоящее время ведущие пакеты синтеза систем на ПЛИС, такие как продукты фирм Synopsis, Caddence, Mentor Graphics, многих производителей ПЛИС, поддерживают синтез с описания на языке Verilog.

•Создавать свои типы данных в Verilog нельзя. Основной тип данных для синтезируемых описаний: целое — integer (32-битовое со знаком).

•В Verilog могут быть использованы специфические объекты (UDP, Specify-блоки), не имеющие аналогов в VHDL, а также многочисленные функции PLI (Program Language Interface).

Основные понятия Verilog

- Описание проекта основано на описании модулей (module) и их встраивании в модули верхнего уровня при помощи переменных различных типов.
- В Verilog определены стандартные типы:
  - данных (числа, строки):17 (десятичная 17), 'h17 (десятичная 23, шестнадцатеричная 17), 5'b10011 (десятичная 19,двоичная 10011), 12'h01F (десятичная 31, шестнадцатеричная 01F)
  - переменных: reg (регистр), wire (провод), integer (целое), real (вещественное), event (событие) и time (время).
- Каждый разряд в переменной типа reg или wire может принимать одно из четырех значений: 1, 0, x или z.
- Операции с переменными и константами имеют С-подобный синтаксис (+ | - | . | * | / | > | < | >= | <= | !&& | || | == | != | ?:|{}|%|===|!==|~|&| | |<<|>>|)
- Над операциями определены стандартные события (posedge, negedge, и др.)
- Логические операции могут быть битовыми или операторами свертки.

```
EX:     D1=4'b1111; D2=4'b1010;
        A=D1~^D2; // bitwise операция — два операнда
        a=~^D2; // reduction операция — один операнд
        A=1010; a=1;
```

Пример реализации d-триггера

```
module ff(clk, rst, A_IN, B_OUT);

    input clk,rst,A_IN;
    output B_OUT;
    reg arb_onebit = 1'b0;

    always @(posedge clk or posedge rst)
    begin
            if (rst)
                    arb_onebit <= 1'b1;
            else
                    arb_onebit <= A_IN;
            end
    end
    B_OUT <= arb_onebit;
endmodule
```

Числа в Verilog

Целые числа (Integers)

Целые числа могут быть двоичными (binary), обозначаются b или B, десятичными (decimal, d или D), шестнадцатеричными (hexidecimal, h, H) или восьмеричными (octal, o или O).

Неопределенное и высокоимпедансное состояния (X and Z values)

Символ x используется для задания неопределенного состояния, символ z показывает третье (высокоимпедансное). При использовании в качестве цифры в числах вместо символа z можно использовать «?». Это рекомендуется делать в операторах выбора (case expressions) для улучшения читаемости кода. Ниже приведены примеры использования символов x и z в числах.

```
8'b1010_0010 // 8-битное число в двоичной системе
8'hA2 // 8-битное число в шестнадцатеричной системе
4'b10x0
4'b101z
12'dz
12'd? //zzz...zzz
8'h4x
```

Цепи в Verilog (Nets)

Для обозначения цепей используются следующие ключевые слова: wire, supply0, supply1. Величина по умолчанию (default value) — z. Разрядность по умолчанию (default size) — 1 бит.

По своему назначению цепь (Net) в Verilog сходна с сигналом в VHDL. Цепи обеспечивают непрерывное модифицирование сигналов на выходах цифровой схемы относительно изменения сигналов на ее входах.

Если драйвер (источник) сигнала цепи имеет некоторое значение, то и цепь принимает то же значение. Если драйверы цепи принимают различные значения, цепь принимает значение наиболее «сильного» сигнала (strongest), если же «сила» каждого сигнала равнозначна, то цепь принимает неопределенное состояние (x).

Цепи не удерживают своего состояния (за исключением типа trireg) их состояние дожно непрерывно удерживаться (continious assignment) выходом с логического вентиля или комбинационной схемы.

Если к цепи не подключен источник (driver) то цепь переходит в состояние высокого импеданса. Исключение составляет цепь типа trireg которая удерживает предыдущее состояние (но это не регистр!).

Для обозначения цепи наиболее часто применяется ключевое слово wire, ключевые слова supply0 и supply1 используются для моделирования источников питания (power supplies) в схеме.

```
wire [31:0] arb_request;
wire signed [8:0] arb_signed;
wire [7:0] mem_array [63:0];
wire [7:0] array3 [0:15][0:255][0:15];
```

Регистры (Registers)

Для обозначения регистров применяется ключевое слово reg. Величина по умолчанию — x. Разрядность по умолчанию — 1 бит.

Основное различие между цепями (nets) и регистрами (registers) состоит в том, что значение регистра должно быть назначено явно. Эта величина сохраняется до тех пор, пока не сделано новое назначение. Рассмотрим использование этого свойства на примере триггера с разрешением (E-type flip flop):

```
module E_ff(q, data, enable, reset, clock);
        output q;
        input data, enable, reset, clock;
        reg q;

        always @(posedge clock)
                if (reset == 0)
                        q = 1'b0;
                else if (enable==1)
                        q = data;
endmodule
```

Регистр q хранит записанную в него величину до тех пор, пока не произойдет новое назначение сигнала.

Операторы языка Verilog

Concatenation	{}	Bitwise AND	&
Replication	{{}}	Bitwise Inclusive OR	|
Arithmetic	+, -, *,**	Bitwise Exclusive OR	^
Addition	+	Bitwise Equivalence	~^, ^~
Subtraction	-	Reduction AND	&
Multiplication	*	Reduction NAND	~&
Power	**	Reduction OR	|
Relational	>, <, >=, <=	Reduction NOR	~|
Logical Negation	!	Reduction XOR	^
Logical AND	&&	Reduction XNOR	~^, ^~
Logical OR	||	Left Shift	<<
Logical Equality	==	Right Shift Signed	>>>
Logical Inequality	!=	Left Shift Signed	<<<
Case Equality	===	Right Shift	>>
Case Inequality	!==	Conditional	?:
Bitwise Negation	~	Event OR	or, ','

Операторы языка Verilog

a b	a==b	a===b	a!=b	a!==b	a&b	a&&b	a|b	a||b	a^b
0 0	1	1	0	0	0	0	0	0	0
0 1	0	0	1	1	0	0	1	1	1
0 x	x	0	x	1	0	0	x	x	x
0 z	x	0	x	1	0	0	x	x	x
1 0	0	0	1	1	0	0	1	1	1
1 1	1	1	0	0	1	1	1	1	0
1 x	x	0	x	1	x	x	1	1	x
1 z	x	0	x	1	x	x	1	1	x
x 0	x	0	x	1	0	0	x	x	x
x 1	x	0	x	1	x	x	1	1	x
x x	x	1	x	0	x	x	x	x	x
x z	x	0	x	1	x	x	x	x	x
z 0	x	0	x	1	0	0	x	x	x
z 1	x	0	x	1	x	x	1	1	x
z x	x	0	x	1	x	x	x	x	x
z z	x	1	x	0	x	x	x	x	x

Блоки и модули языка Verilog

Основной структурной единицей Verilog описания является module. Модуль соответствует entity в VHDL. Модуль описывается ключевыми словами module — endmodule. В файле может быть описано несколько модулей. Другие модули могут подключаться к цепям модуля, образуя иерархическую структуру.

Блоки объединяют процедурные описания (аналог последовательных операторов VHDL). Блоки Verilog бывают двух типов: always (синтезируемый, аналог process в VHDL) и initial (не синтезируемый).

```
module mux4 (sel, a, b, c, d,
outmux);
input [1:0] sel;
input [1:0] a, b, c, d;
output [1:0] outmux;
reg [1:0] outmux;
initial
begin
        outmux=2’b00;
end
```

```
always @(sel or a or b or c or d)
begin
        case (sel)
                2'b00: outmux = a;
                2'b01: outmux = b;
                2'b10: outmux = c;
                default: outmux = d;
        Endcase
end
endmodule
```

Непрерывное и процедурное присваивание

Применяться операторы могут как к цепям (wire), так и к сигналам (reg) и переменным. При этом используются различные типы присвоения. Для цепей, которые являются моделью физического соединения (провода), требуется подключение непрерывного воздействия, которое моделируется непрерывным (continuous) присвоением. Значения же регистров и переменных могут изменяться в результате процедурных действий и сохраняться между воздействиями (так же, как и переменные процедурного языка программирования). Для моделирования этого используется процедурное (procedural) присвоение.

```
assign parity=^data; //непрерывное присвоение
```

Непрерывное присвоение употребляется вне процедурных блоков initial или begin и используется либо в описании цепи, либо с ключевым словом assign (существуют также процедурные непрерывные присвоения в блоках initial или always с ключевыми словами assign и deassign). Слева от оператора непрерывного присвоения (=) должен находиться объект типа цепь. При изменении значения какого-либо из объектов, входящих в выражение справа от =, данное выражение будет вычислено, и новое значение будет присвоено.

Процедурное присваивание

Процедурные присвоения бывают двух типов: blocking (=) и nonblocking (<=).

Пример1:
```
always @ (posedge CLK) a=b;
always @ (posedge CLK) b=a;
```

Пример2:
```
always @ (posedge CLK) a<=b;
always @ (posedge CLK) b<=a;
```

Для пользователей VHDL можно провести параллель между variable assignment (:= VHDL) и blocking assignment (= Verilog), signal assignment (<= VHDL) и nonblocking assignment (<= Verilog) соответственно. Но следует учесть, что в процедурных конструкциях Verilog различий между регистром и переменной не делается.

Построение иерархии описаний

```
module mux5 (sel, a, b, c, d, e, f, g, h, outmux);

        input [2:0] sel;
        input [1:0] a, b, c, d, e, f, g, h;
        output [1:0] outmux;

        wire [1:0] outmux0,outmux1;

        //мультиплексор
        wire [1:0] outmux=(sel[2])?outmux1:outmux0;

        // Variant 2
        //assign outmux=(sel[2])?outmux1:outmux0;

        //подключение по расположению
        mux4 mux_1(sel[1:0], a, b, c, d, outmux0);

        //подключение по имени
        mux4 mux_2(.sel(sel[1:0]), .a(e), .b(f), .c(g), .d(h), .outmux(outmux1));

endmodule
```

Список чувствительности

```
module binaryToESeg_Behavioral
(output reg eSeg,
input A, B, C, D);
always@(A, B, C, D) begin
        eSeg = 1;
        if(~A & D)
                eSeg = 0;
        if(~A & B & ~C)
                eSeg = 0;
        if(~B & ~C & D)
                eSeg = 0;
        end
endmodule
```

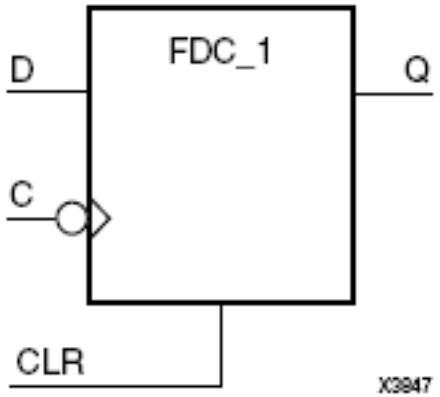
```
module binaryToESeg_Behavioral
(output reg eSeg,
input A, B, C, D);
always@(*) begin
        eSeg = 1;
        if(~A & D)
                eSeg = 0;
        if(~A & B & ~C)
                eSeg = 0;
        if(~B & ~C & D)
                eSeg = 0;
        end
endmodule
```

Настроечные параметры

```
module xorx (xout, xin1, xin2);

        parameter width = 4,
        delay = 10;

        output [1:width] xout;
        input [1:width] xin1,xin2;
        assign #(delay)
        xout = xin1 ^ xin2;
endmodule
```

Примеры описания устройств на языках VHDL и Verilog

Динамический D-триггер (flip-flop with positive edge clock).
VHDL описание

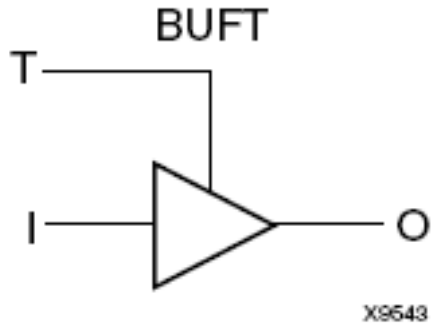



```
library ieee;
use ieee.std_logic_1164.all;

entity registers_2 is
port(C, D, CLR : in std_logic;
Q : out std_logic);
end registers_2;

architecture archi of registers_2 is
begin
process (C, CLR)
begin
        if (CLR = '1')then
                Q <= '0';
        elsif (C'event and C='0')then
                Q <= D;
        end if;
end process;
end archi;
```

Динамический D-триггер (flip-flop with positive edge clock). VHDL описание

```
module v_registers_2 (C, D, CLR, Q);

input C, D, CLR;
output Q;
reg Q;

always @(negedge C or posedge CLR)
begin
        if (CLR)
                Q <= 1'b0;
        else
                Q <= D;
        end
endmodule
```

Буфер с третьим состоянием (buft). VHDL описание



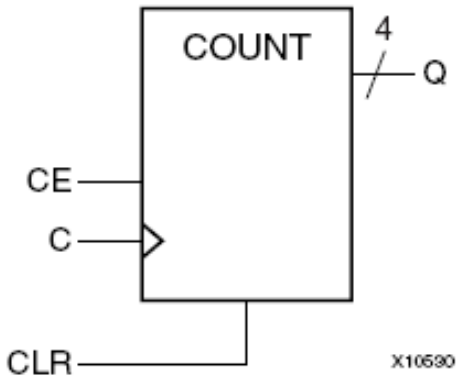

```
entity three_st_1 is
port(T : in std_logic;
I : in std_logic;
O : out std_logic);
end three_st_1;

architecture archi of three_st_1 is
begin
        process (I, T)
        begin
                if (T='0') then
                        O <= I;
                else
                        O <= 'Z';
                end if;
        end process;
        -- Variant 2
        -- O <= I when (T='0') else 'Z';
end archi;
```

Буфер с третьим состоянием (buft). Verilog описание

```
module v_three_st_1 (T, I, O);

input T, I;
output O;
reg O;

always @(T or I)
begin
        if (~T)
                O = I;
        else
                O = 1'bZ;
end

//Variant 2
assign O = (~T) ? I: 1'bZ;

endmodule
```

Счетчик с разрешением счета (Counter). VHDL описание

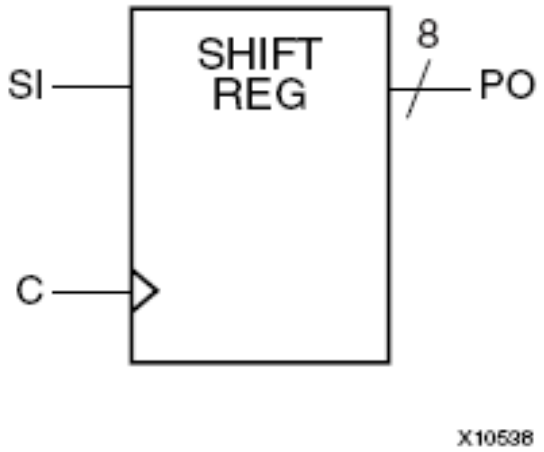
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity counters_5 is
port(C, CLR, CE : in std_logic;
Q : out std_logic_vector(3 downto 0));
end counters_5;

architecture archi of counters_5 is
signal tmp: std_logic_vector(3 downto 0);
begin
        process (C, CLR)
        begin
                if (CLR='1') then
                        tmp <= "0000";
                elsif (C'event and C='1') then
                        if (CE='1') then
                                tmp <= tmp + 1;
                        end if;
                end if;
        end process;
        Q <= tmp;
end archi;
```



Счетчик с разрешением счета (Counter). Verilog описание

```
module v_counters_5 (C, CLR, CE, Q);

input C, CLR, CE;
output [3:0] Q;
reg [3:0] tmp;

always @(posedge C or posedge CLR)
begin
        if (CLR)
                tmp <= 4'b0000;
        else if (CE)
                tmp <= tmp + 1'b1;
        end
        assign Q = tmp;
endmodule
```

Сдвиговый регистр с последовательной загрузкой (Shift register with serial in and parallel out).
VHDL описание



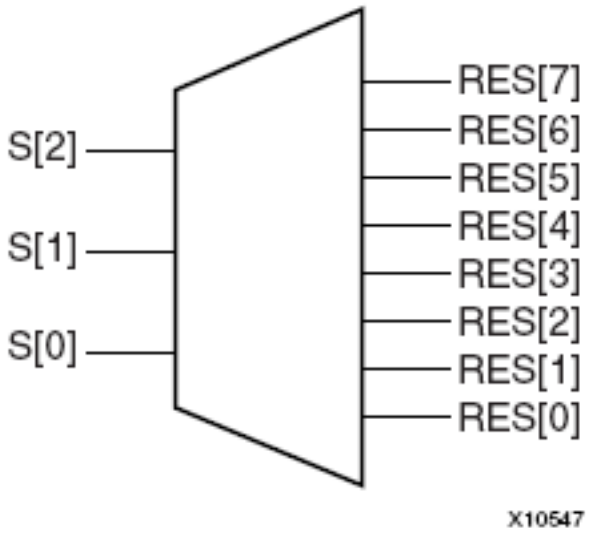

```
library ieee;
use ieee.std_logic_1164.all;

entity shift_registers_5 is
port(C, SI : in std_logic;
PO : out std_logic_vector(7 downto 0));
end shift_registers_5;

architecture archi of shift_registers_5 is
signal tmp: std_logic_vector(7 downto 0);
begin
        process (C)
        begin
                if (C'event and C='1') then
                        tmp <= tmp(6 downto 0)& SI;
                end if;
        end process;
        PO <= tmp;
end archi;
```

Сдвиговый регистр с последовательной загрузкой (Shift register with serial in and parallel out). Verilog описание

```
module v_shift_registers_5 (C, SI, PO);
input C,SI;
output [7:0] PO;
reg [7:0] tmp;

always @(posedge C)
        tmp <= {tmp[6:0], SI};

assign PO = tmp;

endmodule
```

Дешифратор (Decoder).
VHDL описание

```
library ieee;
use ieee.std_logic_1164.all;

entity decoders_1 is
port (sel: in std_logic_vector (2 downto 0);
res: out std_logic_vector (7 downto 0));
end decoders_1;

architecture archi of decoders_1 is
begin
        res <= "00000001" when sel = "000" else
               "00000010" when sel = "001" else
               "00000100" when sel = "010" else
               "00001000" when sel = "011" else
               "00010000" when sel = "100" else
               "00100000" when sel = "101" else
               "01000000" when sel = "110" else
               "10000000";
end archi;
```



Дешифратор (Decoder).
VHDL описание

```
module v_decoders_1 (sel, res);

input [2:0] sel;
output [7:0] res;
reg [7:0] res;

always @(sel or res)
begin
        case (sel)
                3'b000 : res = 8'b00000001;
                3'b001 : res = 8'b00000010;
                3'b010 : res = 8'b00000100;
                3'b011 : res = 8'b00001000;
                3'b100 : res = 8'b00010000;
                3'b101 : res = 8'b00100000;
                3'b110 : res = 8'b01000000;
                default : res = 8'b10000000;
        endcase
end
endmodule
```