Продолжение. Начало в № 3 `2008

Краткий курс HDL. Часть 5. Написание кода, независимого от аппаратной платформы

Иосиф КАРШЕНБОЙМ
iosifk@narod.ru

Содержание этой главы — основной стиль и методика создания текстов описания проектов на языках HDL HDL в соответствии с документом фирмы Actel [1, раздел "Technology Independent Coding Styles"]. Приведенные здесь примеры можно рассматривать как справочник по наиболее часто используемым компонентам. Аналогичные описания можно найти и в [2, 3].

Приведенный здесь стиль описания является примером эффективного и стандартного HDL-кода и позволяет создать описание компонентов, независимых от примененных микросхем.

Вентили (Gate)

Ключевые слова:
and, nand, or, nor, xor, xnor, buf, not

Вентили, поддерживаемые языком Verilog, определяются следующим образом: на первом месте помещается ключевое слово, потом пробел, имя и названия входов. Имя может быть опциональным. Вентили **and** и **or** имеют один выход и два или более входов. Например, для **and** это будет выглядеть следующим образом (пример 1).

```
and <name><list of arguments>
and myand(out, in1, in2, in3);     // вентиль and с тремя входами,
                                   // имя вентиля — myand
and (out, in1, in2);               // вентиль and без имени.
```

Пример 1. Синтаксис и примеры для вентилей and и or

По соглашению о порядке расположения портов для вентилей, определенном в библиотеке примитивов и в описании на язык Verilog, в начало списка аргументов помещается сигнал выхода, а за ним следуют входы.

Вентили **buf** и **not** имеют только один вход и один или несколько выходов. Для них порядок расположения сигналов — обратный, то есть сначала следуют выходы, а последним сигналом записывается вход (пример 2).

```
buf mybuf(out1, out2, out3, in);
not (out, in);
```

Пример 2. Синтаксис и примеры для вентилей buf и not

Устройства с памятью

Устройство с памятью — это или триггер, срабатывающий по фронту, или защелка, управляемая потенциалом. Это однобитовые устройства памяти.

Триггеры (регистры)

Кроме отдельных триггеров, в проектах обычно используются массивы триггеров. В этом случае массив из триггеров будет называться **регистром**. Эти триггеры и регистры в описании проектов на VHDL участвуют в операторах "wait" и "if" в пределах процесса — "process". Для тактирования триггеров или регистров в описании используют или положительный, или отрицательный фронт синхрочастоты, либо вызов функции. Есть два типа выражений, которые можно использовать: **'event** — признак события или вызов функции. Например:

- (clk'event **and** clk = '1') — положительный фронт;
- (clk'event **and** clk = '0') — отрицательный фронт;
- rising_edge (clock) — вызов функции стробирования по положительному фронту;
- falling_edge (clock) — вызов функции стробирования по отрицательному фронту.

В примерах в этом справочнике используется только положительный фронт, но отрицательный фронт синхросигнала тоже можно применять. Использование выражений **'event** предпочтительнее, потому что некоторые инструментальные средства синтеза VHDL, вероятно, не могут распознать, какую именно функцию им надо вызывать. Но, вместе с тем, использование вызова функции еще предпочтительнее для моделирования, потому что вызов функции обнаруживает только перепад на фронте импульса (от 0 до 1 или от 1 до 0), но не переход от X в 1 или 0 в X, что, возможно, не будет являться достоверным переходом для данного симулятора. Это особенно существенно при использовании таких типов данных, как std_logic, которые имеют девять возможных значений (U, X, 0, 1, Z, W, L, H).

В этом разделе будут представлены триггеры, работающие по переднему фронту сигнала.

D-триггер,
работающий по переднему фронту,
без асинхронного сброса и установки

Примеры 3 и 4 описывают D-триггер (рис. 1) без асинхронного или синхронного сброса и без предварительной установки. Этот триггер представляет собой основной компонент, который находится в каждой ячейке FPGA. На рис. 1 и последующих рисунках, изображающих триггеры, прямоугольник голубого цвета представляет собой схематическое изображение триггерного примитива, находящегося в кристалле. В файлах примеров названия портов даны произвольно, например "data", "clk" и т. д. Но в отечественной практике большее распространение получили «ГОСТовские» названия "D", "C" и т. д. Поэтому на графические изображения, для того, чтобы примеры читались легче, на-

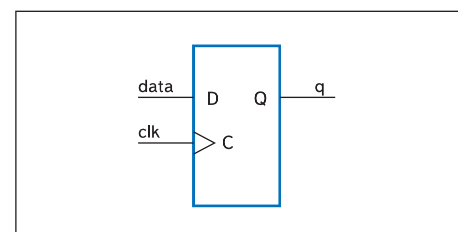

Рис. 1. D-триггер

несены и эти названия. Но необходимо подчеркнуть, что это не названия портов для приводимых примеров, а просто дополнительная информация, облегчающая понимание схемы.

```
VHDL

library IEEE;
use IEEE.std_logic_1164.all;

entity dff is
port (data, clk  : in std_logic;
      q          : out std_logic);
end dff;

architecture behav of dff is
begin
process (clk) begin
    if (clk'event and clk = '1') then
        q <= data;
    end if;
end process;
end behav;
```

Пример 3. VHDL-код, описывающий D-триггер

```
Verilog

module dff (data, clk, q);
    input data, clk;
    output q;
    reg q;
always @(posedge clk)
q = data;
endmodule
```

Пример 4. Verilog-код, описывающий D-триггер

D-триггер, работающий по переднему фронту, с асинхронным сбросом

В примерах 5 и 6 показано, как производится описание D-триггера с асинхронным сбросом (рис. 2).

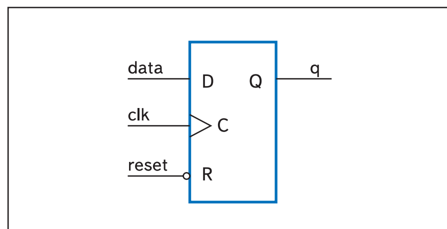

Рис. 2. D-триггер с асинхронным сбросом

```
VHDL

library IEEE;
use IEEE.std_logic_1164.all;

entity dff_async_rst is
port (data, clk, reset : in std_logic;
      q                : out std_logic);
end dff_async_rst;

architecture behav of dff_async_rst is
begin
process (clk, reset) begin
    if (reset = '0') then
        q <= '0';
    elsif (clk'event and clk = '1') then
        q <= data;
    end if;
end process;
end behav;
```

Пример 5. VHDL-код, описывающий D-триггер с асинхронным сбросом

```
Verilog

module dff_async_rst (data, clk, reset, q);
    input data, clk, reset;
    output q;
    reg q;
always @(posedge clk or negedge reset)
    if (~reset)
        q = 1'b0;
    else
        q = data;
endmodule
```

Пример 6. Verilog-код, описывающий D-триггер с асинхронным сбросом

D-триггер, работающий по переднему фронту, с асинхронной установкой

В примерах 7 и 8 показано, как производится описание D-триггера с асинхронной установкой (рис. 3).

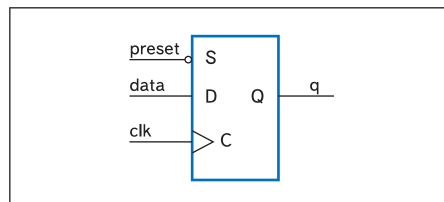

Рис. 3. D-триггер с асинхронной установкой

```
VHDL

library IEEE;
use IEEE.std_logic_1164.all;

entity dff_async_pre is
port (data, clk, preset : in std_logic;
      q                 : out std_logic);
end dff_async_pre;

architecture behav of dff_async_pre is
begin
process (clk, preset) begin
    if (preset = '0') then
        q <= '1';
    elsif (clk'event and clk = '1') then
        q <= data;
    end if;
end process;
end behav;
```

Пример 7. VHDL-код, описывающий D-триггер с асинхронной установкой

```
Verilog

module dff_async_pre (data, clk, preset, q);
    input data, clk, preset;
    output q;
    reg q;
always @(posedge clk or negedge preset)
    if (~preset)
        q = 1'b1;
    else
        q = data;
endmodule
```

Пример 8. Verilog-код, описывающий D-триггер с асинхронной установкой

D-триггер, работающий по переднему фронту, с асинхронными входами сброса и установки

В примерах 9 и 10 показано, как производится описание D-триггера с асинхронными входами сброса и установки (рис. 4).

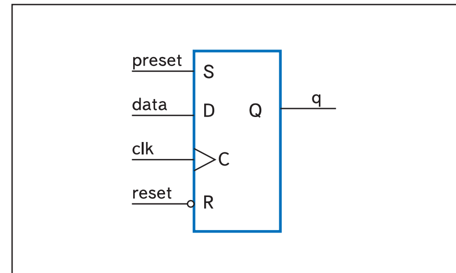

Рис. 4. D-триггер с асинхронными входами сброса и установки

```
VHDL

library IEEE;
use IEEE.std_logic_1164.all;

entity dff_async is
port (data, clk, reset, preset : in std_logic;
      q                        : out std_logic);
end dff_async;

architecture behav of dff_async is
begin
process (clk, reset, preset) begin
    if (reset = '0') then
        q <= '0';
    elsif (preset = '1') then
        q <= '1';
    elsif (clk'event and clk = '1') then
        q <= data;
    end if;
end process;
end behav;
```

Пример 9. VHDL-код, описывающий D-триггер с асинхронными входами сброса и установки

```
Verilog

module dff_async (reset, preset, data, q, clk);
    input clk;
    input reset, preset, data;
    output q;
    reg q;

always @ (posedge clk or negedge reset or posedge preset)
    if (~reset)
        q = 1'b0;
    else if (preset)
        q = 1'b1;
    else q = data;
endmodule
```

Пример 10. Verilog-код, описывающий D-триггер с асинхронными входами сброса и установки

D-триггер, работающий по переднему фронту, с синхронным входом сброса

В примерах 11 и 12 показано, как производится описание D-триггера с синхронным входом сброса (рис. 5).

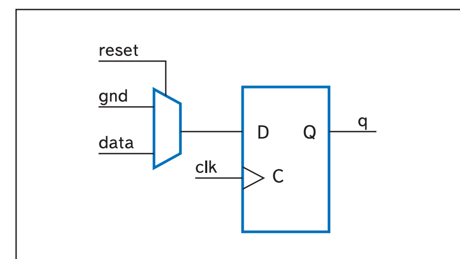

Рис. 5. D-триггер с синхронным входом сброса

```
VHDL

library IEEE;
use IEEE.std_logic_1164.all;

entity dff_sync_rst is
port (data, clk, reset : in std_logic;
      q                : out std_logic);
end dff_sync_rst;
architecture behav of dff_sync_rst is
begin
process (clk) begin
     if (clk'event and clk = '1') then
        if (reset = '0') then
           q <= '0';
        else q <= data;
        end if;
     end if;
end process;
end behav;
```

Пример 11. VHDL-код, описывающий D-триггер с синхронным входом сброса

```
Verilog

module dff_sync_rst (data, clk, reset, q);
    output q;
    reg q;
always @ (posedge clk)
    if (~reset)
       q = 1'b0;
else q = data;
endmodule
```

Пример 12. Verilog-код, описывающий D-триггер с синхронным входом сброса

D-триггер, работающий по переднему фронту, с синхронным входом установки

В примерах 13 и 14 показано, как производится описание D-триггера с синхронным входом установки (рис. 6).

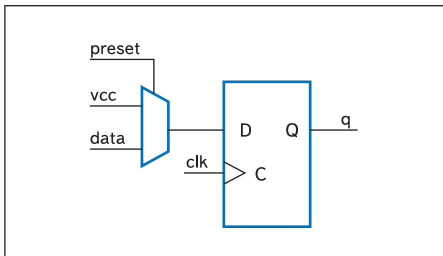

Рис. 6. D-триггер с синхронным входом установки

```
VHDL

library IEEE;
use IEEE.std_logic_1164.all;

entity dff_sync_pre is
port (data, clk, preset : in std_logic;
      q                 : out std_logic);
end dff_sync_pre;

architecture behav of dff_sync_pre is
begin
process (clk) begin
     if (clk'event and clk = '1') then
        if (preset = '0') then
           q <= '1';
        else q <= data;
        end if;
     end if;
end process;
end behav;
```

Пример 13. VHDL-код, описывающий D-триггер с синхронным входом установки

```
Verilog

module dff_sync_pre (data, clk, preset, q);
    input data, clk, preset;
    output q;
    reg q;
always @ (posedge clk)
    if (~preset)
       q = 1'b1;
    else q = data;
endmodule
```

Пример 14. Verilog-код, описывающий D-триггер с синхронным входом установки

D-триггер, работающий по переднему фронту, с асинхронным входом сброса и разрешением записи

В примерах 15 и 16 показано, как производится описание D-триггера с асинхронным входом сброса и разрешением записи (рис. 7).

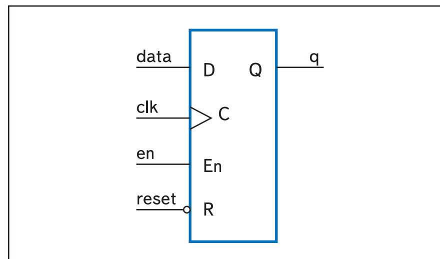

Рис. 7. D-триггер с асинхронным входом сброса и разрешением записи

```
VHDL

library IEEE;
use IEEE.std_logic_1164.all;

entity dff_ck_en is
port (data, clk, reset, en : in std_logic;
      q                    : out std_logic);
end dff_ck_en;
architecture behav of dff_ck_en is
begin
process (clk, reset) begin
     if (reset = '0') then
         q <= '0';
     elsif (clk'event and clk = '1') then
     if (en = '1') then
         q <= data;
     end if;
     end if;
end process;
end behav;
```

Пример 15. VHDL-код, описывающий D-триггер с асинхронным входом сброса и разрешением записи

```
Verilog

module dff_ck_en (data, clk, reset, en, q);
    input data, clk, reset, en;
    output q;

always @ (posedge clk or negedge reset)
    if (~reset)
       q = 1'b0;
    else if (en)
       q = data;
endmodule
```

Пример 16. Verilog-код, описывающий D-триггер с асинхронным входом сброса и разрешением записи

D-триггер-защелка (D-Latches)

D-триггер-защелка с входом данных и входом разрешения записи

В примерах 17 и 18 показано, как производится описание D-триггера-защелки с входом данных и входом разрешения записи (рис. 8).

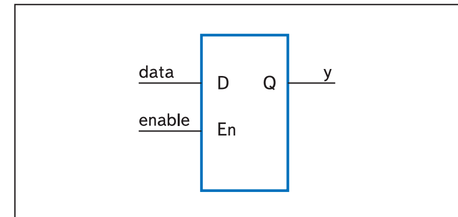

Рис. 8. D-триггер-защелка с входом данных и входом разрешения записи

```
VHDL

library IEEE;
use IEEE.std_logic_1164.all;

entity d_latch is
port (enable, data : in std_logic;
      y            : out std_logic);
end d_latch;

architecture behave of d_latch is
begin
process (enable, data)
begin
     if (enable = '1') then
        y <= data;
     end if;
end process;
end behave;
```

Пример 17. VHDL-код, описывающий D-триггер-защелку с входом данных и входом разрешения записи

```
Verilog

module d_latch (enable, data, y);
    input enable, data;
    output y;
    reg y;
always @(enable or data)
    if (enable)
       y = data;
endmodule
```

Пример 18. Verilog-код, описывающий D-триггер-защелку с входом данных и входом разрешения записи

D-триггер-защелка с входом данных и асинхронным входом разрешения данных

В примерах 19 и 20 показано, как производится описание D-триггера-защелки с асинхронным входом разрешения данных (рис. 9).

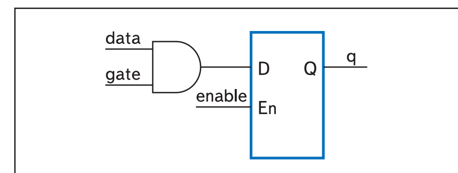

Рис. 9. D-триггер-защелка с входом данных и асинхронным входом разрешения данных

```
VHDL

library IEEE;
use IEEE.std_logic_1164.all;

entity d_latch_e is
```

```
port (enable, gate, data  : in std_logic;
        q                 : out std_logic);
end d_latch_e;

architecture behave of d_latch_e is
begin
process (enable, gate, data) begin
        if (enable = '1') then
            q <= data and gate;
        end if;
end process;
end behave;
```

Пример 19. VHDL-код, описывающий D-триггер-защелку с входом данных и асинхронным входом разрешения данных

Verilog

```
module d_latch_e(enable, gate, data, q);
        input enable, gate, data;
        output q;
        reg q;
always @ (enable or data or gate)
        if (enable)
            q = (data & gate);
endmodule
```

Пример 20. Verilog-код, описывающий D-триггер-защелку с входом данных и асинхронным входом разрешения данных

D-триггер-защелка с входом разрешения записи и асинхронным входом разрешения

В примерах 21 и 22 показано, как производится описание D-триггера-защелки с асинхронным входом разрешения (рис. 10).

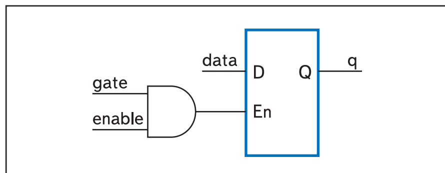

Рис. 10. D-триггер-защелка с входом разрешения записи и асинхронным входом разрешения

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;

entity d_latch_en is
port (enable, gate, d : in std_logic;
        q             : out std_logic);
end d_latch_en;
architecture behave of d_latch_en is
begin
process (enable, gate, d) begin
        if ((enable and gate) = '1') then
            q <= d;
        end if;
end process;
end behave;
```

Пример 21. VHDL-код, описывающий D-триггер-защелку с входом разрешения записи и асинхронным входом разрешения

Verilog

```
module d_latch_en(enable, gate, d, q);
        nput enable, gate, d;
        output q;
        reg q;
always @ (enable or d or gate)
        if (enable & gate)
            q = d;
endmodule
```

Пример 22. Verilog-код, описывающий D-триггер-защелку с входом разрешения записи и асинхронным входом разрешения

D-триггер-защелка с асинхронным входом сброса

В примерах 23 и 24 показано, как производится описание D-триггера-защелки с асинхронным входом сброса (рис. 11).

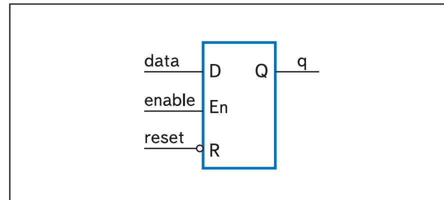

Рис. 11. D-триггер-защелка с асинхронным входом сброса

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;

entity d_latch_rst is
port (enable, data, reset : in std_logic;
        q                 : out std_logic);
end d_latch_rst;
architecture behav of d_latch_rst is
begin
process (enable, data, reset) begin
        if (reset = '0') then
            q <= '0';
        elsif (enable = '1') then
            q <= data;
        end if;
end process;
end behav;
```

Пример 23. VHDL-код, описывающий D-триггер-защелку с асинхронным входом сброса

Verilog

```
module d_latch_rst (reset, enable, data, q);
        input reset, enable, data;
        output q;
        reg q;
always @ (reset or enable or data)
        if (~reset)
            q = 1'b0;
        else if (enable)
            q = data;
endmodule
```

Пример 24. Verilog-код, описывающий D-триггер-защелку с асинхронным входом сброса

Приоритетный шифратор, использующий функцию If-Then-Else

Утверждения условного оператора **if-then-else** используются для того, чтобы по результату проверки условия выполнить последовательность утверждений. Каждое из условий утверждения условного оператора проверяется до тех пор, пока не будет найдено условие-«истина». Утверждения, связанные с таким условием, будут выполняться, а остальная часть утверждения будет игнорироваться. Утверждения условного оператора **if-then-else** должны использоваться в том случае, когда необходимо иметь приоритет при обработке сигналов. В примерах 25 и 26 сигнал **c** будет проверяться первым, и, следовательно, он будет иметь высший приоритет при обработке (рис. 12).

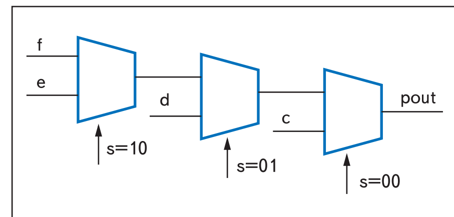

Рис. 12. Приоритетный шифратор, использующий функцию If-Then-Else

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;

entity my_if is
port (c, d, e, f   : in std_logic;
        s          : in std_logic_vector(1 downto 0);
        pout       : out std_logic);
end my_if;

architecture my_arc of my_if is
begin
myif_pro: process (s, c, d, e, f) begin
        if s = «00» then
            pout <= c;
        elsif s = «01» then
            pout <= d;
        elsif s = «10» then
            pout <= e;
        else pout <= f;
        end if;
end process myif_pro;
end my_arc;
```

Пример 25. VHDL-код, описывающий приоритетный шифратор, использующий функцию If-Then-Else

Verilog

```
module IF_MUX (c, d, e, f, s, pout);
        input c, d, e, f;
        input [1:0]s;
        output pout;
        reg pout;
always @(c or d or e or f or s) begin
        if (s == 2'b00)
            pout = c;
        else if (s ==2'b01)
            pout = d;
        else if (s ==2'b10)
            pout = e;
        else pout = f;
        end
endmodule
```

Пример 26. Verilog-код, описывающий приоритетный шифратор, использующий функцию If-Then-Else

Мультиплексоры, использующие функцию Case

Оператор **case** подразумевает параллельное декодирование. Этот оператор используется для того, чтобы выбрать одно из нескольких альтернативных утверждений, основанных на значении условия. Условие будет проверяться в каждом из вариантов выбора в операторе **case** до тех пор, пока не будет найдено соответствие. Когда соответствие будет найдено, тогда будут выполнены утверждения, связанные с этим выбором. Оператор **case** должен включить все возможные значения для заданного условия или иметь выбор значения по умолчанию, которое выполняется, если ни одно из соответствий не выбрано. В следующих примерах показано выполнение мультиплексора, использующее опе-

ратор **case**. Инструментальные средства синтеза VHDL автоматически производят параллельное декодирование без приоритетов в операторах **case**.

Однако некоторые инструментальные средства Verilog могут выполнить схему с приоритетом, и, возможно, необходимо будет добавить директиву к используемому оператору **case**, чтобы гарантировать, что нет выполнения логики с приоритетом. Более подробную информацию об этом можно получить в описаниях на инструмент синтеза.

Мультиплексор 4:1

В примерах 27 и 28 приведено описание мультиплексора 4:1, использующее функцию **Case** (рис. 13).

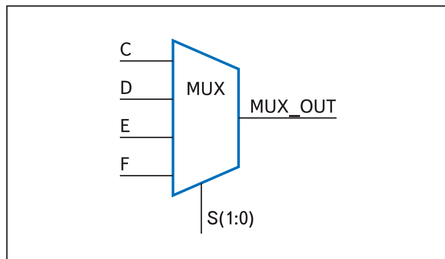

Рис. 13. Мультиплексор 4:1, использующий Case

VHDL

```
--4:1 Multiplexor
library IEEE;
use IEEE.std_logic_1164.all;

entity mux is
port (C, D, E, F  : in std_logic;
      S           : in std_logic_vector(1 downto 0);
      mux_out     : out std_logic);
end mux;

architecture my_mux of mux is
begin
mux1: process (S, C, D, E, F) begin
            case s is
                        when «00» => muxout <= C;
                        when «01» => muxout <= D;
                        when «10» => muxout <= E;
                        when others => muxout <= F;
            end case;
end process mux1;
end my_mux;
```

Пример 27. VHDL-код, описывающий мультиплексор 4:1, использующий Case

Verilog

```
//4:1 Multiplexor
module MUX (C, D, E, F, S, MUX_OUT);
      input C, D, E, F;
      input [1:0] S;
      output MUX_OUT;
      reg MUX_OUT;
always @(C or D or E or F or S)
begin
      case (S)
        2'b00 : MUX_OUT = C;
        2'b01 : MUX_OUT = D;
        2'b10 : MUX_OUT = E;
        default : MUX_OUT = F;
      endcase
end
endmodule
```

Пример 28. Verilog-код, описывающий мультиплексор 4:1, использующий Case

Мультиплексор, использующий функцию Case X

Пример 29, написанный на языке Verilog, показывает мультиплексор, в котором используется утверждение **Case X**. Но многие инструментальные средства синтеза VHDL и Verilog обычно не поддерживают уровень сигнала X.

Verilog

```
//8 bit 4:1 multiplexor with don't care X,
// 3:1 equivalent mux
module mux4 (a, b, c, sel, q);
      input [7:0] a, b, c;
      input [1:0] sel;
      output [7:0] q;
      reg [7:0] q;
always @ (sel or a or b or c)
 casex (sel)
        2'b00: q = a;
        2'b01: q = b;
        2'b1x: q = c;
        default: q = c;
 endcase
endmodule
```

Пример 29. Verilog-код, описывающий мультиплексор, в котором используется Case X

Декодер (демультиплексор)

Декодеры применяются для того, чтобы декодировать данные. В примерах 30 и 31 показано использование декодера 3–8, имеющего вход разрешения **En**.

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;

entity decode is
port ( Ain   : in std_logic_vector (2 downto 0);
       En    : in std_logic;
       Yout  : out std_logic_vector (7 downto 0));
end decode;

architecture decode_arch of decode is
begin
      process (Ain)
      begin
        if (En='0') then
          Yout <= (others => '0');
        else
          case Ain is
            when «000» => Yout <= «00000001»;
            when «001» => Yout <= «00000010»;
            when «010» => Yout <= «00000100»;
            when «011» => Yout <= «00001000»;
            when «100» => Yout <= «00010000»;
            when «101» => Yout <= «00100000»;
            when «110» => Yout <= «01000000»;
            when «111» => Yout <= «10000000»;
            when others => Yout <= «00000000»;
          end case;
        end if;
      end process;
end decode_arch;
```

Пример 30. VHDL-код, описывающий декодер 3–8 с входом разрешения En

Verilog

```
module decode (Ain, En, Yout);
      input En;
      input [2:0] Ain;
      output [7:0] Yout;
      reg [7:0] Yout;
always @ (En or Ain)
      begin
      if (!En)
        Yout = 8'b0;
      else
        case (Ain)
          3'b000 : Yout = 8'b00000001;
          3'b001 : Yout = 8'b00000010;
          3'b010 : Yout = 8'b00000100;
          3'b011 : Yout = 8'b00001000;
          3'b100 : Yout = 8'b00010000;
          3'b101 : Yout = 8'b00100000;
          3'b110 : Yout = 8'b01000000;
          3'b111 : Yout = 8'b10000000;
          default : Yout = 8'b00000000;
        endcase
      end
endmodule
```

Пример 31. Verilog-код, описывающий декодер 3–8 с входом разрешения En

Счетчики

Счетчики — это одни из наиболее применяемых в разработке цифровой аппаратуры узлов. Они считают число изменений сигналов на входе. Эти изменения происходят или в произвольные моменты времени, или в равные промежутки времени. Далее приведены примеры описания счетчиков с различными режимами работы. Компилятор, обрабатывая весь проект, произведет оптимизацию счетчика в соответствии с заданными ему критериями. Но необходимо помнить, что в том случае, если ваш счетчик находится в высокочастотной части проекта, где скорости обработки данных велики, будет лучше воспользоваться библиотекой функциональных узлов, предоставляемой фирмой — изготовителем кристаллов. Можно воспользоваться и визардом, входящим в состав программного инструмента, предоставляемого фирмой-изготовителем. Такой компонент будет оптимизирован и проверен под конкретную технологию и конкретные кристаллы. После того как сгенерированный компонент будет выполнен в виде файла, его можно скопировать в проект или подключить как дополнительный файл проекта. Следующие примеры показывают различные типы счетчиков.

Счетчик 8-bit Up Counter с входом разрешения счета и асинхронным сбросом

В примерах 32 и 33 показан счетчик **8-bit Up Counter** с входом разрешения счета и асинхронным сбросом.

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;

entity counter8 is
   port (clk, en, rst : in std_logic;
         count        : out std_logic_vector (7 downto 0));
end counter8;

architecture behav of counter8 is
signal cnt: std_logic_vector (7 downto 0);
begin
process (clk, en, cnt, rst)
      begin
        if (rst = '0') then
          cnt <= (others => '0');
      elsif (clk'event and clk = '1') then
        if (en = '1') then
          cnt <= cnt + '1';
      end if;
```

```
end process;
    count <= cnt;
end behav;
```

Пример 32. VHDL-код, описывающий счетчик 8-bit Up Counter с входом разрешения счета и асинхронным сбросом

Verilog

```
module count_en (en, clock, reset, out);
    parameter Width = 8;
    input clock, reset, en;
    output [Width-1:0] out;
    reg [Width-1:0] out;
always @(posedge clock or negedge reset)
    if (!reset)
        out = 8'b0;
    else if (en)
        out = out + 1;
endmodule
```

Пример 33. Verilog-код, описывающий счетчик 8-bit Up Counter с входом разрешения счета и асинхронным сбросом

Счетчик 8-bit Up Counter с входом параллельной загрузки и входом асинхронного сброса

В примерах 34 и 35 показан счетчик **8-bit Up Counter** с входом параллельной загрузки и с входом асинхронного сброса.

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;

entity counter is
  port (clk, reset, load : in std_logic;
        data             : in std_logic_vector (7 downto 0);
        count            : out std_logic_vector (7 downto 0));
end counter;

architecture behave of counter is
    signal count_i : std_logic_vector (7 downto 0);
begin
  process (clk, reset)
  begin
    if (reset = '0') then
          count_i <= (others => '0');
      elsif (clk'event and clk = '1') then
        if load = '1' then
          count_i <= data;
        else
          count_i <= count_i + '1';
        end if;
      end if;
    end process;
  count <= count_i;
end behave;
```

Пример 34. VHDL-код, описывающий счетчик 8-bit Up Counter с входом параллельной загрузки и входом асинхронного сброса

Verilog

```
module count_load (out, data, load, clk, reset);
    parameter Width = 8;
    input load, clk, reset;
    input [Width-1:0] data;
    output [Width-1:0] out;
    reg [Width-1:0] out;
always @(posedge clk or negedge reset)
    if (!reset)
        out = 8'b0;
    else if (load)
        out = data;
    else
        out = out + 1;
endmodule
```

Пример 35. Verilog-код, описывающий счетчик 8-bit Up Counter с входом параллельной загрузки и входом асинхронного сброса

Счетчик 8-bit Up Counter с входом параллельной загрузки, разрешения счета с входом асинхронного сброса и выходом окончания счета при достижении состояния «все единицы»

В примере 36 показан счетчик **8-bit Up Counter** с входом параллельной загрузки, разрешения счета с входом асинхронного сброса и выходом окончания счета при достижении состояния «все единицы».

Verilog

```
module count_load (out, cout, data, load, clk, en, reset);
    parameter Width = 8;
    input load, clk, en, reset;
    input [Width-1:0] data;
    output cout; // carry out
    output [Width-1:0] out;
    reg [Width-1:0] out;

always @(posedge clk or negedge reset)
    if (!reset)
      out = 8'b0;
    else if (load)
      out = data;
    else if (en)
      out = out + 1;
// cout=1 when all out bits equal 1
assign cout = &out;
endmodule
```

Пример 36. Verilog-код, описывающий счетчик 8-bit Up Counter с входом параллельной загрузки, разрешения счета с входом асинхронного сброса и выходом окончания счета при достижении состояния «все единицы»

Счетчик n-bit Up Counter с входом параллельной загрузки, разрешения счета с входом асинхронного сброса

В примерах 37 и 38 показан счетчик **n-bit Up Counter** с входом параллельной загрузки, разрешения счета с входом асинхронного сброса.

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;

entity counter is
generic (width : integer := n);
port (data              : in std_logic_vector (width-1 downto 0);
      load, en, clk, rst : in std_logic;
      q                 : out std_logic_vector (width-1 downto 0));
end counter;

architecture behave of counter is
signal count : std_logic_vector (width-1 downto 0);
begin
process (clk, rst)
    begin
      if rst = '1' then
          count <= (others => '0');
      elsif (clk'event and clk = '1') then
        if load = '1' then
          count <= data;
        elsif en = '1' then
          count <= count + '1';
        end if;
      end if;
end process;
q <= count;
end behave;
```

Пример 37. VHDL-код, описывающий счетчик n-bit Up Counter с входом параллельной загрузки, разрешения счета и входом асинхронного сброса

Verilog

```
module count_load (out, data, load, clk, reset);
    parameter Width = 8;
    input load, clk, reset;
    input [Width-1:0] data;
    output [Width-1:0] out;

reg [Width-1:0] out;

always @(posedge clk or negedge reset)
    if (!reset)
      out = 8'b0;
    else if (load)
      out = data;
    else
      out = out + 1;
endmodule
```

Пример 38. Verilog-код, описывающий счетчик n-bit Up Counter с входом параллельной загрузки, разрешения счета и входом асинхронного сброса

Арифметические операторы

В примерах 39 и 40 приводятся арифметические операторы: сложение, вычитание, умножение и деление. Программные инструменты, применяемые для синтеза, позволяют оптимизировать использование данных операторов для применяемой элементной базы.

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity arithmetic is
  port (A, B    : in std_logic_vector (3 downto 0);
        Q1      : out std_logic_vector (4 downto 0);
        Q2, Q3  : out std_logic_vector (3 downto 0);
        Q4      : out std_logic_vector (7 downto 0));
end arithmetic;

architecture behav of arithmetic is
begin
process (A, B)
begin
    Q1 <= ('0' & A) + ('0' & B); --addition
    Q2 <= A — B; --subtraction
    Q3 <= A / B; --division
    Q4 <= A * B; --multiplication
end process;
end behav;
```

Пример 39. VHDL-код, описывающий арифметические операторы

Если операнды умножения и деления представляют собой числа, равные степени двух, то в проекте такое умножение и деление может быть заменено соответствующими сдвигами на регистрах. Сдвиговые регистры обеспечивают выполнение проекта с большим выигрышем по скорости выполнения вычислительных операций, и они занимают очень мало ресурсов. Например, операция:

```
Q <= C/16 + C*4;
```

может быть представлена как:

```
Q <= shr (C, «100») + shl (C, «10»);
```

или на VHDL вот так:

```
Q <= «0000» & C (8 downto 4) + C (6 downto 0) & «00»;
```

Функции “shr” и “shl” находятся в библиотеке IEEE.std_logic_arith.all.

Рассмотрим тот же случай:

```
Q = C/16 + C*4;
```

в Verilog эта операция будет выглядеть следующим образом:

```
Q = {4b'0000 C[8:4]} + {C[6:0], 2b'00};
```

```
Verilog

module arithmetic (A, B, Q1, Q2, Q3, Q4);
      input [3:0] A, B;
      output [4:0] Q1;
      output [3:0] Q2, Q3;
      output [7:0] Q4;
      reg [4:0] Q1;
      reg [3:0] Q2, Q3;
      reg [7:0] Q4;
always @ (A or B)
      begin
         Q1 = A + B; //addition
         Q2 = A — B; //subtraction
         Q3 = A / 2; //division
         Q4 = A * B; //multiplication
      end
endmodule
```

Пример 40. Verilog-код, описывающий арифметические операторы

Операторы соотношения (Relational Operators)

Операторы соотношения выполняют действия над операндами, выдавая значение **true** или **false** в зависимости от того, равны или не равны, больше или меньше операнды друг относительно друга или нет. Примеры 41 и 42 показывают применение операторов соотношения. Программные инструменты, применяемые для синтеза, позволяют оптимизировать использование данных операторов для применяемой элементной базы.

```
VHDL

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity relational is
port (A, B                  : in std_logic_vector(3 downto 0);
        Q1, Q2, Q3, Q4 : out std_logic);
end relational;
architecture behav of relational is
begin
process (A, B)
begin
      -- Q1 <= A > B; -- greater than
      -- Q2 <= A < B; -- less than
      -- Q3 <= A >= B; -- greater than equal to

      if (A <= B) then -- less than equal to
         Q4 <= '1';
      else
         Q4 <= '0';
      end if;
end process;
end behav;
```

Пример 41. VHDL-код, описывающий операторы соотношения

```
Verilog

module relational (A, B, Q1, Q2, Q3, Q4);
      input [3:0] A, B;
      output Q1, Q2, Q3, Q4;
      reg Q1, Q2, Q3, Q4;

always @ (A or B)
begin
      // Q1 = A > B; //greater than
      // Q2 = A < B; //less than
      // Q3 = A >= B; //greater than equal to
      if (A <= B) //less than equal to
         Q4 = 1;
      else
         Q4 = 0;
      end
endmodule
```

Пример 42. Verilog-код, описывающий операторы соотношения

Операторы равенства и неравенства (Equality Operator)

Операторы равенства и операторы неравенства выполняют действия над операндами, выдавая значение **true** или **false** в зависимости от того, равны ли операнды друг другу или нет. Примеры 43 и 44 показывают применение операторов равенства.

```
VHDL

library IEEE;
use IEEE.std_logic_1164.all;

entity equality is
port (
        A   : in STD_LOGIC_VECTOR (3 downto 0);
        B   : in STD_LOGIC_VECTOR (3 downto 0);
        Q1 : out STD_LOGIC;
        Q2 : out STD_LOGIC
        );
end equality;

architecture equality_arch of equality is
begin
   process (A, B)

   begin
         Q1 <= A = B; -- equality
      if (A /= B) then -- inequality
         Q2 <= '1';
      else
         Q2 <= '0';
      end if;
   end process;
end equality_arch;

Или можно сделать вот так:

library IEEE;
use IEEE.std_logic_1164.all;

entity equality is
port (
        A   : in STD_LOGIC_VECTOR (3 downto 0);
        B   : in STD_LOGIC_VECTOR (3 downto 0);
        Q1 : out STD_LOGIC;
        Q2 : out STD_LOGIC
        );
end equality;

architecture equality_arch of equality is
begin
      Q1 <= '1' when A = B else '0'; -- equality
      Q2 <= '1' when A /= B else '0'; -- inequality
end equality_arch;
```

Пример 43. VHDL-код, описывающий операторы равенства и неравенства

```
Verilog

module equality (A, B, Q1, Q2);
      input [3:0] A;
      input [3:0] B;
      output Q1;
      output Q2;
      reg Q1, Q2;
always @ (A or B)
      begin
         Q1 = A == B; //equality
      if (A != B) //inequality
         Q2 = 1;
      else
         Q2 = 0;
      end
endmodule
```

Пример 44. Verilog-код, описывающий операторы равенства и неравенства

Операторы сдвига (Shift Operators)

Операторы сдвига сдвигают данные влево или вправо на то число битов, которое указано в операторе. В примерах 45 и 46 показано применение этого оператора.

```
VHDL

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity shift is
port (data     : in std_logic_vector(3 downto 0);
        q1, q2  : out std_logic_vector(3 downto 0));
end shift;

architecture rtl of shift is
begin
   process (data)
   begin
      q1 <= shl (data, «10»); -- logical shift left
      q2 <= shr (data, «10»); --logical shift right
   end process;
end rtl;

Или можно сделать вот так:

library IEEE;
use IEEE.std_logic_1164.all;

entity shift is
port (data     : in std_logic_vector(3 downto 0);
        q1, q2  : out std_logic_vector(3 downto 0));
end shift;

architecture rtl of shift is
begin
   process (data)
   begin
      q1 <= data(1 downto 0) & «10»; -- logical shift left
      q2 <= «10» & data(3 downto 2); --logical shift right
   end process;
end rtl;
```

Пример 45. VHDL-код, описывающий операторы сдвига

```
Verilog

module shift (data, q1, q2);
      input [3:0] data;
      output [3:0] q1, q2;
      parameter B = 2;
      reg [3:0] q1, q2;

always @ (data)
begin
      q1 = data << B; // logical shift left
      q2 = data >> B; //logical shift right
end
endmodule
```

Пример 46. Verilog-код, описывающий операторы сдвига

Автоматы состояний

Конечный автомат (finite state machine, FSM) — это тип последовательной схемы, ко-

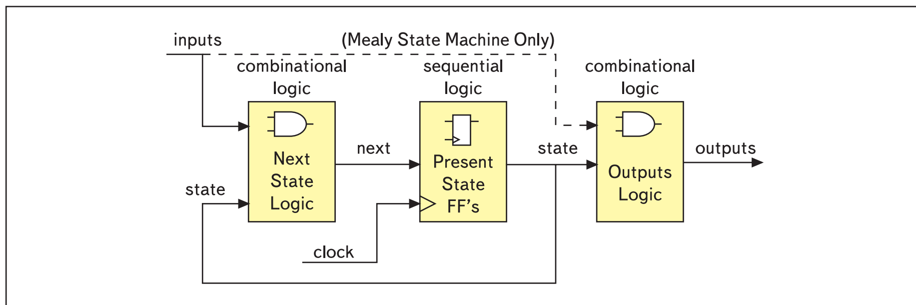

Рис. 14. FSM Мура и Мили

торая спроектирована так, что она имеет набор определенных конечных состояний, и автомат проходит через эти состояния предопределенным последовательным способом. Есть два типа FSM — Мили (Mealy) и Мура (Moore). FSM Мили имеет выходы, которые являются функцией текущего состояния и входов. FSM Мура имеет выходы, которые являются функцией только текущего состояния. FSM состоит из трех частей:

1. Регистр текущего состояния для последовательных переходов — это регистр, имеющий разрядность в n-bit и состоящий из триггеров (объявляются как вектор), синхронизирующихся одним для всех триггеров сигналом синхрочастоты. Вектор состояния, имеющий разрядность в n-bit, имеет 2n возможных значений. Зачастую не все из 2^n состояний необходимы для работы автомата, поэтому он должен быть спроектирован таким образом, чтобы в течение нормальной работы автомат не попал в неиспользованные состояния. Альтернативно, FSM с m-состояниями потребует регистр с разрядностью, по крайней мере, log2 (m).
2. Комбинационная логика для следующего состояния. В каждый момент времени FSM может находиться только в одном состоянии, и каждый активный фронт синхрочастоты заставляет его изменяться от его текущего до следующего состояния так, как определено логической частью схемы. Следующее состояние определяется как функция входов FSM и его текущего состояния.
3. Комбинационная логика для выходов. Для FSM Мура выходы — это функция текущего состояния. Для FSM Мили выходы — это функция текущего состояния и первичных входов FSM. Кроме того, в FSM Мура есть возможность получить сигналы выходов не из текущего состояния, а из следующего состояния, для того чтобы уменьшить время распространения сигнала от фронта синхронизации до выхода. FSM Мура и Мили показаны на рис. 14.

Сигнал сброс используется для того, чтобы гарантировать отказоустойчивое поведение. Он гарантирует, что FSM всегда инициализируется в известное достоверное состояние перед первым активным фронтом синхрочастоты, после чего и начинается нормальная работа автомата. Если сигнал сброса не использовать, то невозможно предсказать начальное значение триггеров регистра состояний в течение включения питания FPGA. Они могут произвольно встать в одно из незакодированных состояний и находиться там неопределенно долго. Сброс должен быть описан как одно из состояний в описании FSM.

Асинхронный сброс будет предпочтительнее синхронного сброса, потому что асинхронный сброс не требует декодирующей логики и не должен быть описан в неиспользуемых состояниях автомата.

Автомат состояний может быть выполнен в двух вариантах. В одном из вариантов схема содержит регистр состояний, имеющий разрядность log2 (m) и выходную декодирующую логику. В другом варианте разрядность регистра состояний выбирается равной числу состояний, при этом выходная логика не используется. Такое выполнение называется "one hot". Поскольку в структуре FPGA имеется достаточно много триггеров и регистров, то при выполнении программным инструментом синтеза конечного автомата по способу "one hot" можно получить оптимальные результаты по использованию площади кристалла (area) и производительности.

Автомат Мили

В примерах 47 и 48 представлен автомат Мили для диаграммы состояний, показанной на рис. 15.

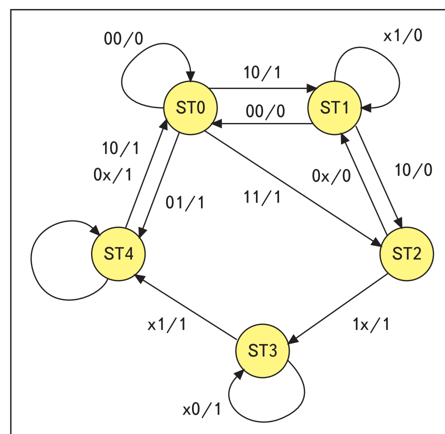

Рис. 15. Диаграмма переходов автомата Мили

VHDL

```
-- Example of a 5-state Mealy FSM
library ieee;
use ieee.std_logic_1164.all;

entity mealy is
  port (clock, reset  : in std_logic;
        data_out      : out std_logic;
        data_in       : in std_logic_vector (1 downto 0));
  end mealy;
architecture behave of mealy is
  type state_values is (st0, st1, st2, st3, st4);
  signal pres_state, next_state: state_values;
begin
-- FSM register
statereg: process (clock, reset)
begin
  if (reset = '0') then
    pres_state <= st0;
  elsif (clock'event and clock ='1') then
    pres_state <= next_state;
  end if;
end process statereg;

-- FSM combinational block
fsm: process (pres_state, data_in)
begin
case pres_state is
when st0 =>
  case data_in is
    when «00» => next_state <= st0;
    when «01» => next_state <= st4;
    when «10» => next_state <= st1;
    when «11» => next_state <= st2;
    when others => next_state <= (others <= 'x');
  end case;
when st1 =>
  case data_in is
    when «00» => next_state <= st0;
    when «10» => next_state <= st2;
    when others => next_state <= st1;
  end case;
when st2 =>
  case data_in is
    when «00» => next_state <= st1;
    when «01» => next_state <= st1;
    when «10» => next_state <= st3;
    when «11» => next_state <= st3;
    when others => next_state <= (others <= 'x');
  end case;
when st3 =>
  case data_in is
    when «01» => next_state <= st4;
    when «11» => next_state <= st4;
    when others => next_state <= st3;
  end case;
when st4 =>
  case data_in is
    when «11» => next_state <= st4;
    when others => next_state <= st0;
  end case;
when others => next_state <= st0;

end case;
end process fsm;

-- Mealy output definition using pres_state w/ data_in
outputs: process (pres_state, data_in)
begin
  case pres_state is
    when st0 =>
      case data_in is
        when «00» => data_out <= '0';
        when others => data_out <= '1';
      end case;
    when st1 => data_out <= '0';
    when st2 =>
      case data_in is
        when «00» => data_out <= '0';
        when «01» => data_out <= '0';
        when others => data_out <= '1';
      end case;
    when st3 => data_out <= '1';
    when st4 =>
      case data_in is
        when «10» => data_out <= '1';
        when «11» => data_out <= '1';
        when others => data_out <= '0';
      end case;
    when others => data_out <= '0';
  end case;
end process outputs;
end behave;
```

Пример 47. VHDL-код, описывающий автомат Мили

```
Verilog

// Example of a 5-state Mealy FSM
module mealy (data_in, data_out, reset, clock);
   output data_out;
   input [1:0] data_in;
   input reset, clock;
   reg data_out;
   reg [2:0] pres_state, next_state;
   parameter st0=3'd0, st1=3'd1, st2=3'd2, st3=3'd3, st4=3'd4;

// FSM register
always @ (posedge clock or negedge reset)
   begin: statereg
   if (!reset)// asynchronous reset
      pres_state = st0;
   else
      pres_state = next_state;
   end // statereg

// FSM combinational block
always @(pres_state or data_in)
   begin: fsm
      case (pres_state)
         st0: case (data_in)
            2'b00: next_state=st0;
            2'b01: next_state=st4;
            2'b10: next_state=st1;
            2'b11: next_state=st2;
      endcase
         st1: case (data_in)
            2'b00: next_state=st0;
            2'b10: next_state=st2;
         default: next_state=st1;
      endcase
         st2: case (data_in)
            2'b0x: next_state=st1;
            2'b1x: next_state=st3;
      endcase
         st3: case (data_in)
            2'bx1: next_state=st4;
         default: next_state=st3;
      endcase
         st4: case (data_in)
            2'b11: next_state=st4;
         default: next_state=st0;
      endcase
   default: next_state=st0;
   endcase
end // fsm

// Mealy output definition using pres_state w/ data_in
always @(data_in or pres_state)
   begin: outputs
      case (pres_state)
         st0: case (data_in)
         2'b00: data_out=1'b0;
         default: data_out=1'b1;
      endcase
         st1: data_out=1'b0;
         st2: case (data_in)
         2'b0x: data_out=1'b0;
         default: data_out=1'b1;
      endcase
         st3: data_out=1'b1;
         st4: case (data_in)
         2'b1x: data_out=1'b1;
   default: data_out=1'b0;
      endcase
      default: data_out=1'b0;
      endcase
end // outputs

endmodule
```

Пример 48. Verilog-код, описывающий автомат Мили

Автомат Мура

В примерах 49 и 50 показан статический автомат Мура.

```
VHDL

-- Example of a 5-state Moore FSM
library ieee;
use ieee.std_logic_1164.all;

entity moore is
   port (clock, reset  : in std_logic;
         data_out      : out std_logic;
         data_in       : in std_logic_vector (1 downto 0));
end moore;

architecture behave of moore is
            type state_values is (st0, st1, st2, st3, st4);
            signal pres_state, next_state: state_values;
begin
-- FSM register
statereg: process (clock, reset)
begin
   if (reset = '0') then
      pres_state <= st0;
   elsif (clock ='1' and clock'event) then
      pres_state <= next_state;
   end if;
end process statereg;

-- FSM combinational block
fsm: process (pres_state, data_in)
begin
   case pres_state is
      when st0 =>
         case data_in is
            when «00» => next_state <= st0;
            when «01» => next_state <= st4;
            when «10» => next_state <= st1;
            when «11» => next_state <= st2;
            when others => next_state <= (others <= 'x');
         end case;
      when st1 =>
         case data_in is
            when «00» => next_state <= st0;
            when «10» => next_state <= st2;
            when others => next_state <= st1;
         end case;
      when st2 =>
         case data_in is
            when «00» => next_state <= st1;
            when «01» => next_state <= st1;
            when «10» => next_state <= st3;
            when «11» => next_state <= st3;
            when others => next_state <= (others <= 'x');
         end case;
      when st3 =>
         case data_in is
            when «01» => next_state <= st4;
            when «11» => next_state <= st4;
            when others => next_state <= st3;
         end case;
      when st4 =>
         case data_in is
            when «11» => next_state <= st4;
            when others => next_state <= st0;
         end case;
      when others => next_state <= st0;
   end case;
end process fsm;

-- Moore output definition using pres_state only
outputs: process (pres_state)
begin
   case pres_state is
      when st0 => data_out <= '1';
      when st1 => data_out <= '0';
      when st2 => data_out <= '1';
      when st3 => data_out <= '0';
      when st4 => data_out <= '1';
      when others => data_out <= '0';
   end case;
end process outputs;

end behave;
```

Пример 49. VHDL-код, описывающий автомат Мура

```
Verilog

// Example of a 5-state Moore FSM
module moore (data_in, data_out, reset, clock);
   output data_out;
   input [1:0] data_in;
   input reset, clock;
   reg data_out;
   reg [2:0] pres_state, next_state;
   parameter st0=3'd0, st1=3'd1, st2=3'd2, st3=3'd3, st4=3'd4;

//FSM register
always @(posedge clock or negedge reset)
   begin: statereg
      if (!reset)
         pres_state = st0;
      else
         pres_state = next_state;
   end // statereg

// FSM combinational block
always @(pres_state or data_in)
   begin: fsm
      case (pres_state)
         st0: case (data_in)
            2'b00: next_state=st0;
            2'b01: next_state=st4;
            2'b10: next_state=st1;
            2'b11: next_state=st2;
      endcase
         st1: case (data_in)
            2'b00: next_state=st0;
            2'b10: next_state=st2;
            default: next_state=st1;
      endcase
         st2: case (data_in)
            2'b0x: next_state=st1;
            2'b1x: next_state=st3;
      endcase
         st3: case (data_in)
            2'bx1: next_state=st4;
            default: next_state=st3;
      endcase
         st4: case (data_in)
            2'b11: next_state=st4;
            default: next_state=st0;
      endcase
      default: next_state=st0;
      endcase
   end // fsm

// Moore output definition using pres_state only
always @(pres_state)
   begin: outputs
      case (pres_state)
         st0: data_out=1'b1;
         st1: data_out=1'b0;
         st2: data_out=1'b1;
         st3: data_out=1'b0;
         st4: data_out=1'b1;
      default: data_out=1'b0;
      endcase
   end // outputs
endmodule // Moore
```

Пример 50. Verilog-код, описывающий автомат Мура

Буферы входов/выходов (Input/Output Buffers)

В проектах можно использовать буферы входа/выхода. Приведенные далее примеры показывают, как устанавливать в проект буферы. Их устанавливают в проект верхнего уровня.

Трехстабильные буферы

Буфер с третьим состоянием (рис. 16) может работать как выход с возможностью перехода в состояние с высоким импедансом. В примерах 51 и 52 показано, как устанавливать в проект буфер с третьим состоянием.

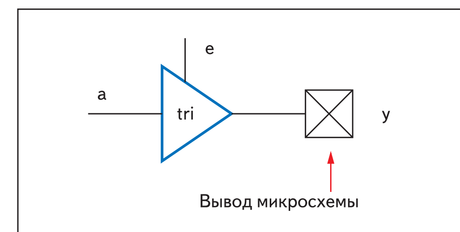

Рис. 16. Буфер с трехстабильным выходом

```
VHDL

library IEEE;
use IEEE.std_logic_1164.all;
entity tristate is
port (e, a  : in std_logic;
      y     : out std_logic);
end tristate;

architecture tri of tristate is
begin
   process (e, a)
      begin
         if e = '1' then
            y <= a;
         else
```

```
            y <= 'Z';
        end if;
    end process;
end tri;
```

Или можно сделать вот так:

```
library IEEE;
use IEEE.std_logic_1164.all;

entity tristate is
port (e, a : in std_logic;
        y    : out std_logic);
end tristate;

architecture tri of tristate is
begin
    Y <= a when (e = '1') else 'Z';
end tri;
```

Пример 51. VHDL-код, описывающий буфер с трехстабильным выходом

Verilog

```
module TRISTATE (e, a, y);
    input a, e;
    output y;
    reg y;
always @ (e or a) begin
    if (e)
        y = a;
    else
        y = 1'bz;
    end
endmodule
```

Или можно сделать вот так:

```
module TRISTATE (e, a, y);
    input a, e;
    output y;

    assign y = e ? a : 1'bZ;

endmodule
```

Пример 52. Verilog-код, описывающий буфер с трехстабильным выходом

Установка компонента в проекте показана в примерах 53 и 54.

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;

entity tristate is
port (e, a : in std_logic;
        y    : out std_logic);
end tristate;

architecture tri of tristate is
component TRIBUFF
port (D, E   : in std_logic;
        PAD  : out std_logic);
end component;

begin
U1: TRIBUFF port map (D => a,
        E => e,
        PAD => y);
end tri;
```

Пример 53. VHDL-код, описывающий установку в проекте буфера с трехстабильным выходом

Verilog

```
module TRISTATE (e, a, y);
        input a, e;
        output y;
        TRIBUFF U1 (.D(a), .E(e), .PAD(y));
endmodule
```

Пример 54. Verilog-код, описывающий установку в проекте буфера с трехстабильным выходом

Двунаправленные буферы

Двунаправленный буфер (рис. 17) может работать как вход или как выход с возможностью перехода в состояние с высоким импедансом. В примерах 55 и 56 показано, как устанавливать в проект двунаправленные буферы.

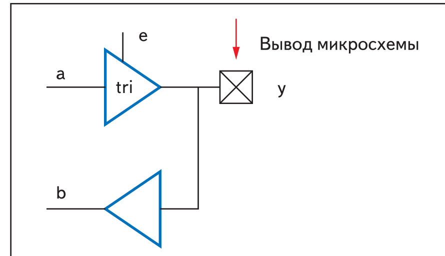

Рис. 17. Двунаправленный буфер

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;

entity bidir is
port (y    : inout std_logic;
        e, a : in std_logic;
        b    : out std_logic);
end bidir;

architecture bi of bidir is
begin
    process (e, a)
        begin
            case e is
                when '1' => y <= a;
                when '0' => y <= 'Z';
                when others => y <= 'X';
            end case;
    end process;
b <= y;
end bi;
```

Пример 55. VHDL-код, описывающий двунаправленный буфер

Verilog

```
module bidir (e, y, a, b);
    input a, e;
    inout y;
    output b;
    reg y_int;
    wire y, b;
always @ (a or e)
    begin
        if (e == 1'b1)
            y_int <= a;
        else
            y_int <= 1'bz;
        end
assign y = y_int;
assign b = y;
endmodule
```

Пример 56. Verilog-код, описывающий двунаправленный буфер

Установка компонента в проекте показана в примерах 57 и 58.

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;

entity bidir is
port (y    : inout std_logic;
        e, a : in std_logic;
        b    : out std_logic);
end bidir;
architecture bi of bidir is
component BIBUF
    port (D, E   : in std_logic;
            Y      : out std_logic;
            PAD  : inout std_logic);
end component;

begin
U1: BIBUF port map
        (D => a,
        E => e,
        Y => b,
        PAD => y);
end bi;
```

Пример 57. VHDL-код, описывающий установку в проекте двунаправленного буфера

Verilog

```
module bidir (e, y, a, b);
        input a, e;
        inout y;
        output b;
        BIBUF U1 ( .PAD(y), .D(a), .E(e), .Y(b) );
endmodule
```

Пример 58. Verilog-код, описывающий установку в проекте двунаправленного буфера

Параметры (Generics and Parameters)

Параметры **generic** и **parameter** используются для того, чтобы определить компонент с переменными исполнениями. Это позволяет выполнять параметризированные компоненты, у которых разрядность и определенный наборов признаков можно задавать при установке этих компонентов в проект. В примерах 59 и 61 показано, как использовать параметры **generic** и **parameter**, описывая сумматор. А также то, как этот сумматор можно установить в проект, используя переменные значения параметров для разрядности.

VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity adder is
generic (WIDTH : integer := 8);
port (A, B    : in UNSIGNED(WIDTH-1 downto 0);
        CIN    : in std_logic;
        COUT : out std_logic;
        Y       : out UNSIGNED(WIDTH-1 downto 0));
end adder;

architecture rtl of adder is
begin
    process (A,B,CIN)
        variable TEMP_A,TEMP_B,TEMP_Y:UNSIGNED(A'length
downto 0);
            begin
                TEMP_A := '0' & A;
                TEMP_B := '0' & B;
                TEMP_Y := TEMP_A + TEMP_B + CIN;
                Y <= TEMP_Y (A'length-1 downto 0);
                COUT <= TEMP_Y (A'length);
    end process;
end rtl;
```

Пример 59. VHDL-код, описывающий сумматор с использованием параметра generic

Параметр “Width” определяет разрядность сумматора. Вот как выглядит установка 16-разрядного сумматора в проекте (пример 60).

```
U1: adder generic map (16) port map (A_A, B_A, CIN_A, COUT_A, Y_A);
```

Пример 60. VHDL-код, описывающий установку 16-разрядного сумматора в проекте

```
Verilog

module adder (cout, sum, a, b, cin);
    parameter Size = 8;
    output cout;
    output [Size-1:0] sum;
    input cin;
    input [Size-1:0] a, b;

assign {cout, sum} = a + b + cin;

endmodule
```

Пример 61. Verilog-код, описывающий сумматор с использованием параметра parameter

Параметр “Size” определяет разрядность сумматора. Вот как выглядит установка 16-разрядного сумматора в проекте (пример 62).

```
adder #(16) adder16(cout_A, sun_A, a_A, b_A, cin_A);
```

Пример 62. Verilog-код, описывающий установку 16-разрядного сумматора в проекте

В следующем разделе мы обсудим вопросы по написанию кода, зависимого от аппаратной платформы.

Литература

1. Actel HDL CodingStyle Guide. http://www.actel.com
2. Predicting the output of Finite State Machines. Application Note. Mentor Graphics, 1986–2006.
3. Golson S. State Machine Design Techniques for Verilog and VHDL. Trilobyte Systems // Synopsys Journal of High-Level Design. September 1994.