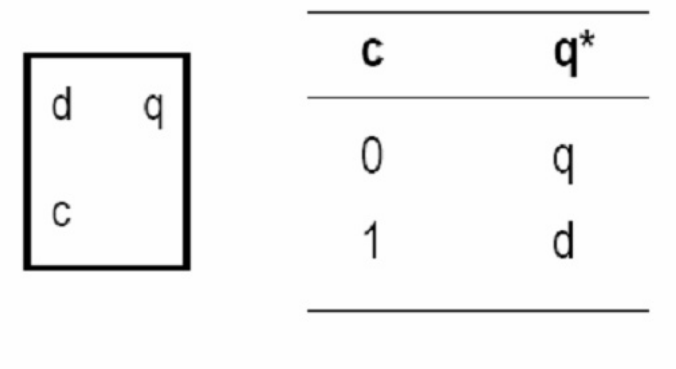
Тема 6. Описание последовательных устройств на VHDL

Описание последовательных элементов на VHDL. Триггеры, регистры, счетчики. Правила кодирования последовательных логических элементов на языке VHDL. Регистровый файл.

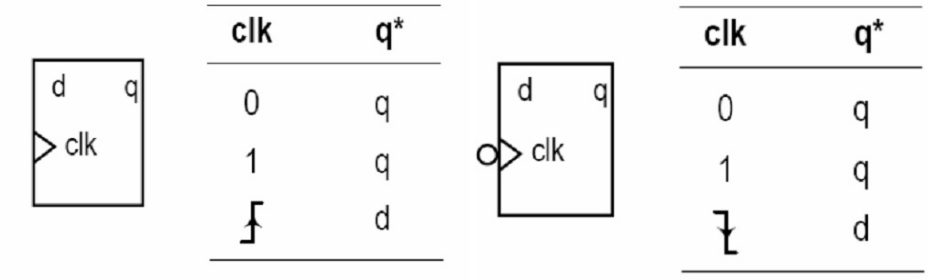
Базовые элементы памяти

Базовыми элементами памяти, т.е. логическими устройствами, которые запоминают состояния сигналов, действующих на их входах в предыдущие моменты времени, являются D триггер-защелка со статической синхронизацией (в английской литература D-Latch) и D триггер с динамической синхронизацией (D flip-flop, DFF). Вспомним их отличие друг от друга: DFF работает по фронту синхросигнала (переднему или заднему), т.е. может изменить значение на своем может изменять свое значение на выходе в любой момент времени, пока синхросигнал равен 1.



c	q*
0	q
1	d

Рис 1. D-latch



clk	q*
0	q
1	q
↓	d

clk	q*
0	q
1	q
↓	d

Рис 2. DFF, работающие по переднему и заднему фронтам

Сказанное выше иллюстрирует рисунок 3.

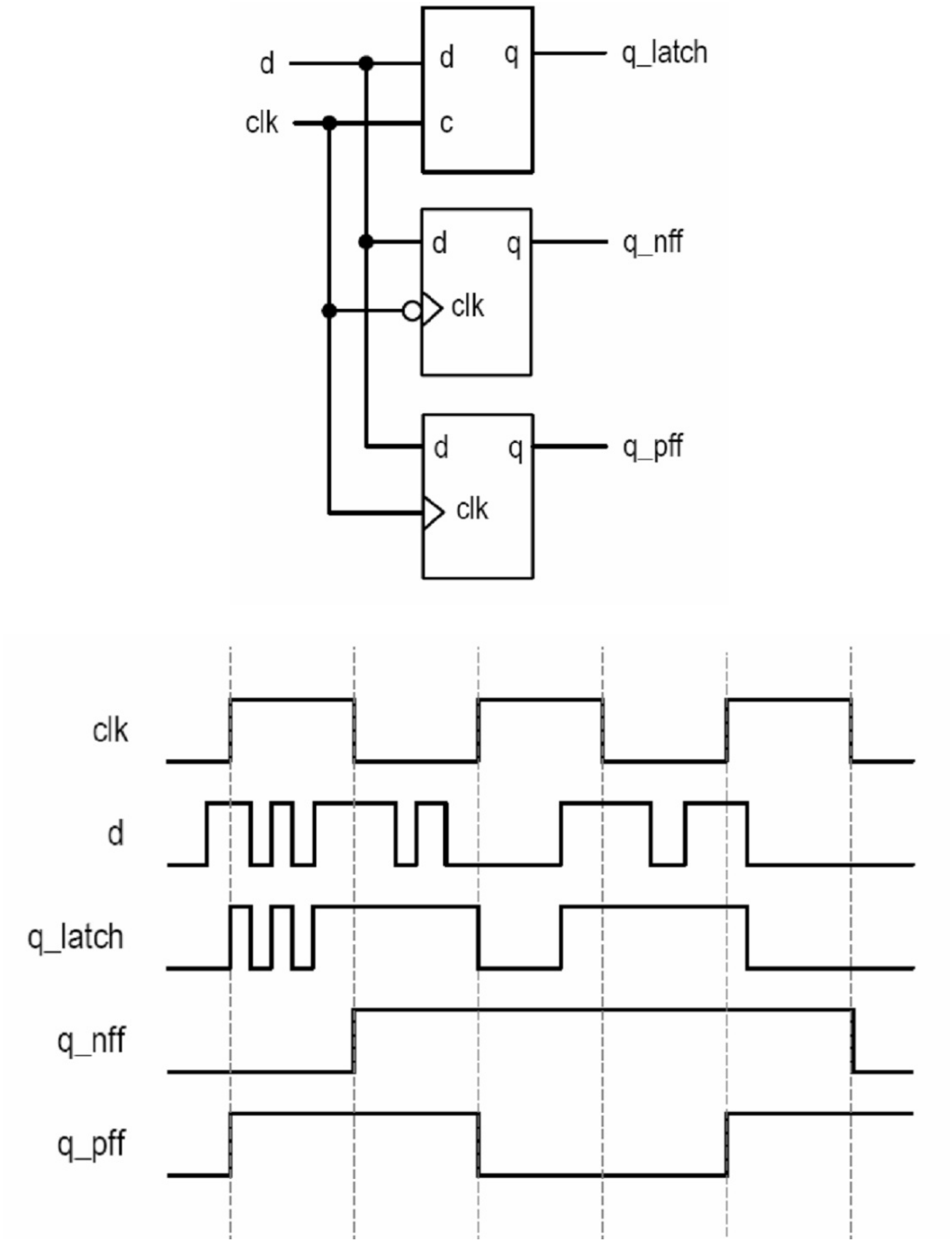


Рис 3. Временные диаграммы переключения D-latch и DFF

Скажем сразу, что использование D-Latch не желательно при построении синхронных схем (то чем мы учимся заниматься), и мы не будем их использовать. Хотя на VHDL можно закодировать и D-Latch и DFF, и мы покажем, как это сделать.

Временная характеристика DFF

Выше было сказано, что DFF переключается в небольшом промежутке времени вокруг фронта синхросигнала. Это необходимо прояснить. Рассмотрим детальную временную диаграмму переключения DFF (Рис 4).

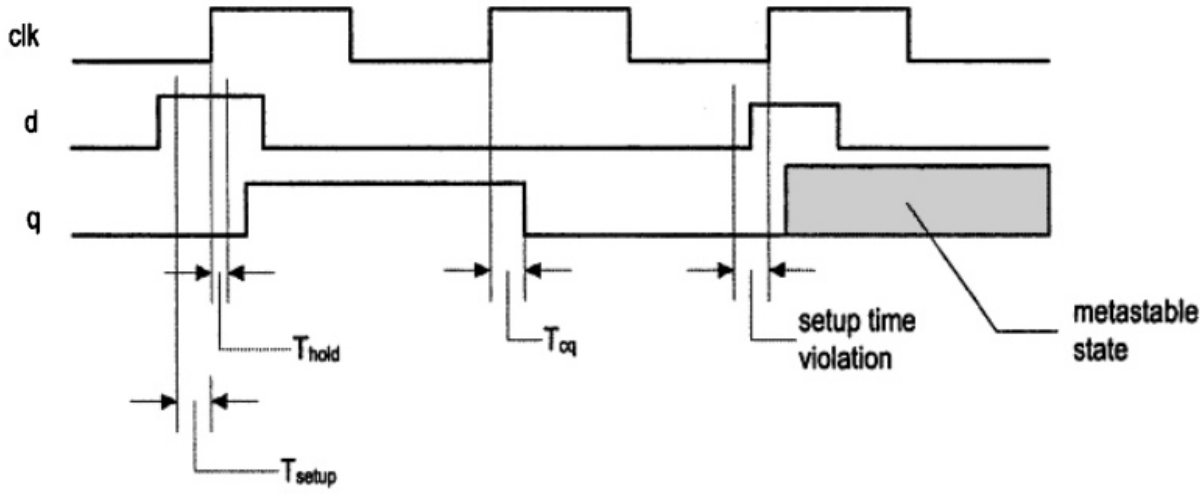


Рис 4. Временная диаграмма работы DFF

Введем следующие временные параметры:

- T_{cq} – выходная задержка триггера. Время после момента прихода фронта синхросигнала, когда на выходе триггера устанавливается стабильное значение логического сигнала.
- T_{setup} – время предустановки. Временной интервал до момента прихода фронта синхросигнала, в течение которого значение на входе триггер q должно быть стабильным
- T_{hold} – время удержания. Временной интервал после прихода фронта, в течение которого значение на входе должно быть стабильным.

T_{cq} грубо соответствует задержке на комбинационной схеме. T_{setup} и T_{hold} являются *ограничениями* схемы (*constraints)*: если входной сигнал триггера будет менять в этом промежутке времени, то триггер может войти в, так называемое метастабильное состояние, в котором выход триггера принимает случайное значение.

Обобщенное представление последовательного элемента

Прежде чем перейти к описанию последовательных устройств на VHDL введем еще одно понятие: обобщенное представление последовательного элемента. В прошлой лекции уже было

сказано, что сложные последовательные элементы состоят из непосредственно элементов памяти – триггеров и некоторой комбинационной схемы, определяющей как раз логику работы элемента. В обобщенном виде это можно представить так (рис 5):

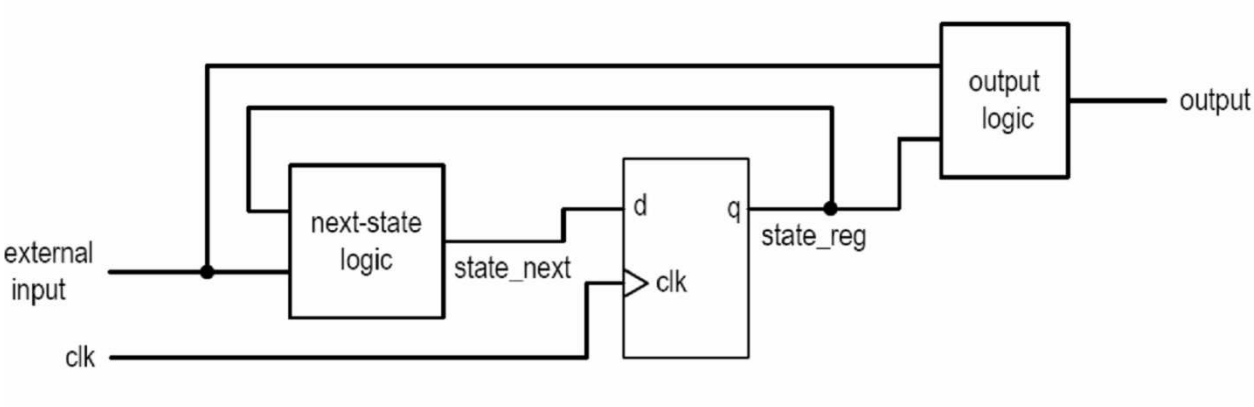


Рис 5. Обобщенное представление последовательного элемента

В соответствие с этим представлением элементом памяти является базовый элемент DFF (рис 2). Он «хранит» в себе значение state_reg. Next_state_logic – это комбинационная схема, которая на основе действующих на данном такте синхросигнала значений state_reg и внешних сигналов external input вычисляет state_next, которое будет записано в триггер на следующем такте (в момент прихода следующего фронта синхросигнала). Комбинационная схема output logic вычисляет значения выходных сигналов элемента в общем случае на основе значений state_reg и внешних сигналов.

Для выполнения временных ограничений DFF необходимо, чтобы период тактового сигнала был больше чем сумма задержки на next_state_logic, T_{cq} и T_{setup}. Этот вопрос будет еще обсуждаться в конце лекции.

Описание последовательных элементов

D-latch

Хотя мы и не будем использовать D-Latch, но как ее описать на VHDL, знать надо:

```
library ieee;
use ieee.std_logic_1164.all;
entity dlatch is
   port (
      c  : in  std_logic;
      d  : in  std_logic;
      q  : out std_logic
   );
end dlatch;
architecture arch of dlatch is
begin
   process(c,d)
   begin
      if (c = '1') then
         q  <= d;
      end if;
   end process;
end arch;
```

Ее интерфейс в entity совпадает с интерфейсом на рис 1. Архитектура элемента состоит всего из одного процесса, в список чувствительности которого состоит из обоих входных сигнала c и d. Далее в теле процесса есть один оператор if, который и реализует логику работы защелки. Отметим, что у него отсутствует ветка else, и в соответствии с семантикой VHDL, в том случае, когда c = ‘0’, сигнал q будет сохранять свое значение. Такое кодирование недопустимо при описании комбинационных элементов, но тут оно как раз уместно.

D FF

```
library ieee;
use ieee.std_logic_1164.all;
entity dff is
   port (
      clk   : in  std_logic;
      d     : in  std_logic;
      q     : out std_logic
   );
end dff;
architecture arch of dff is
begin
   process(clk)
   begin
      if (clk'event and clk = '1') then
         q  <= d;
      end if;
   end process;
end arch;
```

Интерфейс DFF соответствует рис 2. Архитектура элемента состоит из одного процесса, в список чувствительности которого входит только один сигнал clk – синхросигнал. Это правильно и соответствует семантике VHDL, по которой считается, что процесс запускается при изменении любого сигнала из списка чувствительности. Так как у нас триггер переключается только по фронту синхросигнала, то в список чувствительности в данном случае входит только один сигнал. Далее, очевидно, что за 1 такт синхросигнала процесс будет запускаться два раза: один раз на переднем фронте, и один раз на заднем. Выбор фронта, по которому будет идти переключение сигнала, осуществляет оператор if с условием (ck'event and clk='1'). Тут 'event – это *атрибут* сигнала clk, по сути это булевая функция, которая равна TRUE в момент изменения сигнала clk. Clk='1' осуществляет выбор фронта, и в данном случае выбран передний фронт. Выбор заднего фронта будет осуществлен при условии clk = '0'.

```
library ieee;
use ieee.std_logic_1164.all;
entity dff is
   port (
      clk   : in  std_logic;
      d     : in  std_logic;
      q     : out std_logic
   );
end dff;
architecture arch of dff is
begin
   process(clk)
   begin
      if (clk'event and clk = '0') then
         q  <= d;
      end if;
   end process;
end arch;
```

DFF с сигналом асинхронного сброса

У триггера DFF может быть сигнал асинхронного сброса (рис). Асинхронным он называется потому что, имеет более высокий приоритет чем все остальные сигналы, в том числе и тактовый. Это значит, что если reset = ‘1’, то триггер сбросится сразу в тот же момент времени, и не будет ждать прихода фронта синхросигнала.

reset	clk	q*
1	-	0
0	0	q
0	1	q
0	↓	d

Рис. DFF с сигналом асинхронного сброса

Кодируется такой элемент следующим образом:

```
library ieee;
use ieee.std_logic_1164.all;
entity dffr is
    port (
        clk   : in  std_logic;
        reset : in  std_logic;
        d     : in  std_logic;
        q     : out std_logic
    );
end dffr;
architecture arch of dffr is
begin
    process(clk,reset)
    begin
        if reset = '1' then
            q <= '0';
        elsif (clk'event and clk = '1') then
            q  <= d;
        end if;
    end process;
end arch;
```

Тут видно, что процесс имеет расширенный список чувствительности: теперь в него входит сигнал reset. Это означает, что процесс запустится при изменении этого сигнала. Далее, сначала идет проверка reset=’1’, в этом случае q <= ‘0’. Ветка elsif имеет более низкий приоритет, так как если первое условие reset=’1’ выполнилось, то вторая ветка не будет выполняться. Сигнал асинхронного сброса всегда применяется для ввода всех триггеров схемы в начальное состояние, например при включении питания, или при переходе в другой режим работы. И он должен использоваться только для этого. Для сброса триггера во время работы схемы необходимо использовать синхронный сигнал сброса, что мы обсудим позже.

Регистр

Регистр получается, когда несколько триггеров объединяются вместе одним тактовым сигналом. Записываться будет так:

```
entity reg8 is
   port (
      clk   : in  std_logic;
      reset : in  std_logic;
      d     : in  std_logic_vector(7 downto 0);
      q     : out std_logic_vector(7 downto 0)
   );
end reg8;
architecture arch of dffr is
begin
   process(clk,reset)
   begin
      if reset = '1' then
         q <= (others => '0');
      elsif (clk'event and clk = '1') then
         q  <= d;
      end if;
   end process;
end arch;
```

Тут все так, как и для DFF, только тип данных d и q – это std_logic_vector(7 downto 0).

DFF с сигналом разрешения работы enable

reset	clk	en	q*
1	-	-	0
0	0	-	q
0	1	-	q
0	↓	0	q
0	↓	1	d

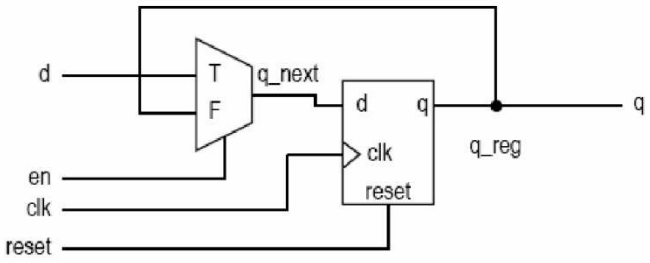


Рис DFF с сигналом разрешения работы

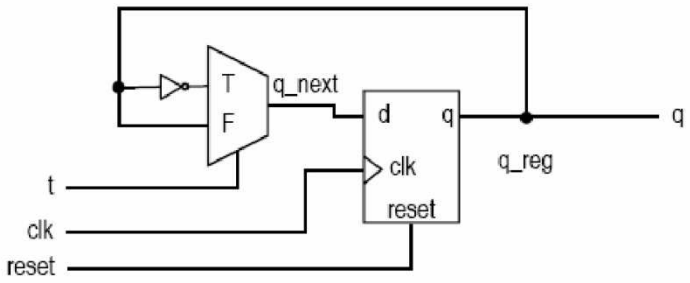
Этот элемент знаком вам по предыдущей лекции. Записываться на VHDL это будет так:

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

Необходимо подчеркнуть, что условие en = ‘1’ должно находится внутри ветки elsif, иначе программа синтезатор неправильно создаст схему.

TFF

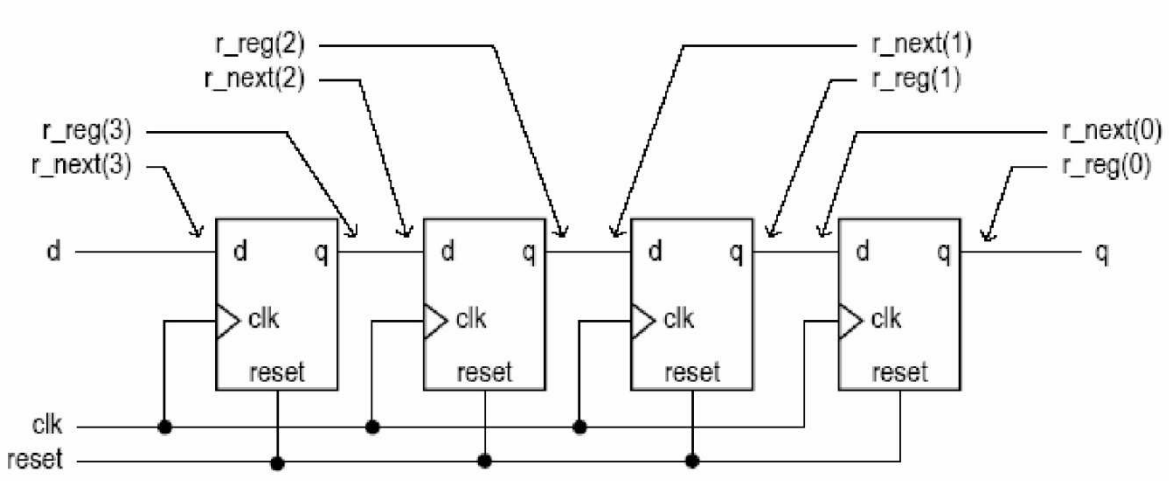
reset	clk	t	q*
1	-	-	0
0	0	-	q
0	1	-	q
0	↓	0	q
0	↓	1	q'



```vhdl
entity tff is
    port (
        clk   : in  std_logic;
        reset : in  std_logic;
        en    : in  std_logic;
        t     : in  std_logic;
        q     : out std_logic
    );
end tff;
architecture arch of tff is
signal q_s  : std_logic;
begin
    process(clk,reset)
    begin
        if reset = '1' then
            q <= '0';
        elsif (clk'event and clk = '1') then
            if en = '1' then
                q_s   <= not q_s;
            end if;
        end if;
    end process;
    q  <= q_s;
end arch;
```

Сдвиговый регистр

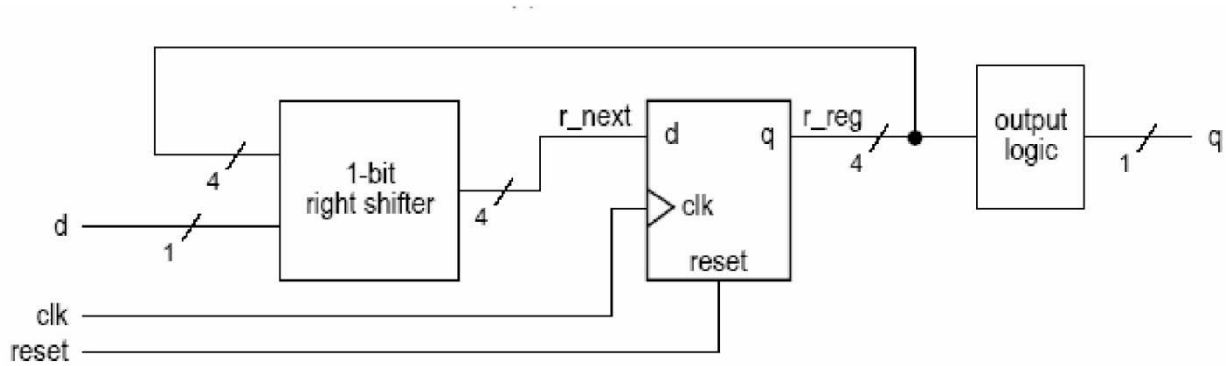
Сдвиговый регистр сдвигает значение регистра на один бит каждый такт. Основное применение – это преобразовывать параллельный код в последовательный и наоборот.



Для описания такой схема на VHDL применяется уже знакомый вам оператор конкатенации.

```
entity shift_right_register is
   port (
      clk, reset   : in  std_logic;
      d      : in  std_logic;
      q      : out std_logic
   );
end shift_right_register;
architecture arch of shift_right_register is
signal shift_reg  : std_logic_vector(3 downto 0);
begin
   process(clk,reset)
   begin
      if reset = '1' then
         shift_reg <= (others => '0');
      elsif (clk'event and clk = '1') then
         shift_reg   <= d & shift_reg(3 downto 1);
      end if;
   end process;
   q  <= shift_reg(0);
end arch;
```

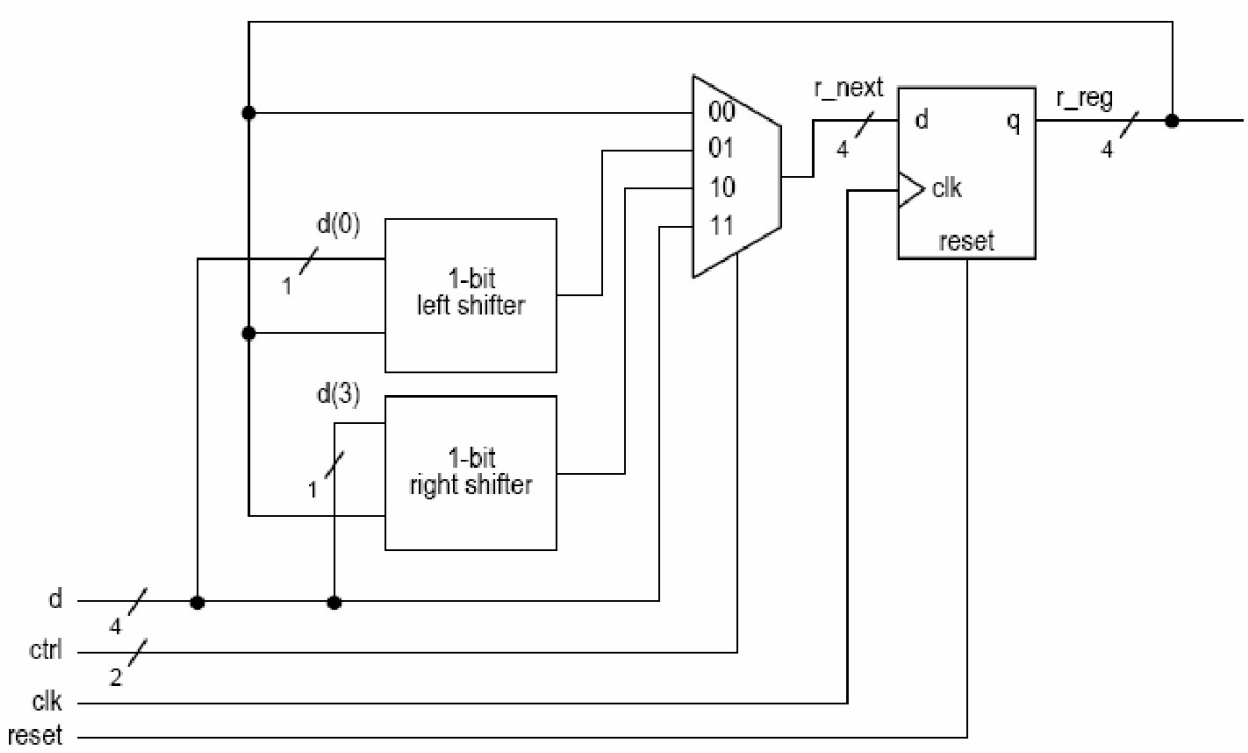
Сдвиговый регистр можно представить и такой картинкой:



Здесь 1-bit right shifter - просто абстрактное обозначение изменения связей в шине данных, и никаких логических элементов это не требует.

Универсальный сдвиговый регистр

Универсальный сдвиговый регистр может быть загружен начальным значением и сдвигать данные в разных направлениях. Определены 4 операции: загрузка, сдвиг влево, сдвиг в право, пауза. Действие выбирается с помощью сигнала ctrl.



На VHDL это описывается так:

```
entity shift_register is
   port (
      clk, reset  : in  std_logic;
      ctrl  : in  std_logic_vector(1 downto 0);
      d     : in  std_logic_vector(3 downto 0);
      q     : out std_logic_vector(3 downto 0)
   );
end shift_register;
architecture arch of shift_register is
signal shift_reg  : std_logic_vector(3 downto 0);
begin
   process(clk,reset)
   begin
      if reset = '1' then
         shift_reg <= (others => '0');
      elsif (clk'event and clk = '1') then
         case ctrl is
            when "11" =>   shift_reg <= d;
            when "01" =>   shift_reg <= shift_reg(2 downto 0) & d(0);    -- shift left
            when "10" =>   shift_reg <= d(3) & shift_reg(3 downto 1);    -- shift right
            when others  =>   shift_reg <= shift_reg;
         end case;
      end if;
   end process;
   q  <= shift_reg;
end arch;
```

Тут используется оператор CASE. Переписать это с помощью оператора IF читателю предлагается самостоятельно.

Непоследовательный счетчик

Иногда требуется, чтобы схема выдавала заранее известную последовательность кодов, определенную таблицей. Например, требуется создать счетчик, последовательно выдающий следующие коды: “000”, “011”, “110”, “101”, “111”. Как сделать такой логический элемент? Очень просто:

- необходимо создать комбинационную схему, работающую по следующей таблице истинности:

input pattern	output pattern
000	011
011	110
110	101
101	111
111	000

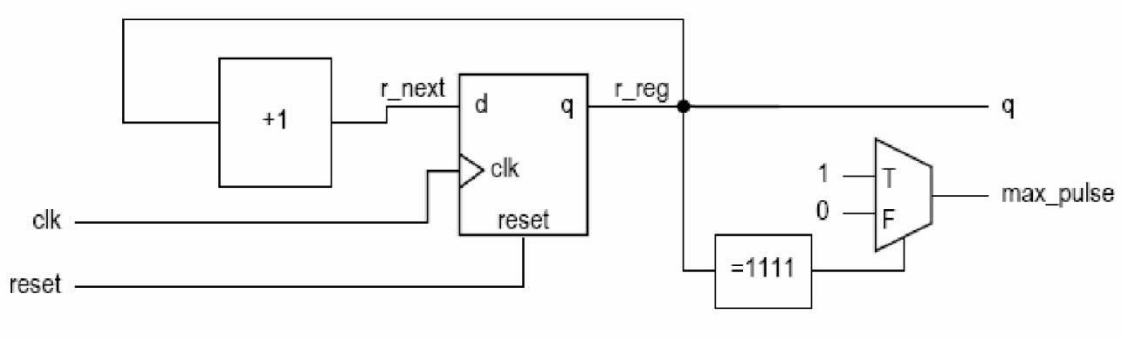
- выходное значение этой схемы сохранять в регистре
- выход регистра соединить со сходом комбинационной схемы

На VHDL это будет записано так:

```
entity arbi_seq_counter is
   port (
      clk, reset  : in  std_logic;
      q     : out std_logic_vector(2 downto 0)
   );
end arbi_seq_counter;
architecture arch of arbi_seq_counter is
signal r_reg   : std_logic_vector(2 downto 0);
begin
   process(clk,reset)
   begin
      if reset = '1' then
         r_reg <= (others => '0');
      elsif (clk'event and clk = '1') then
         case r_reg is
            when "000" =>  r_reg <= "011";
            when "011" =>  r_reg <= "110";
            when "110" =>  r_reg <= "101";
            when "101" =>  r_reg <= "111";
            when "111" =>  r_reg <= "000";
            when others  =>   r_reg <= "000";
         end case;
      end if;
   end process;
   q  <= r_reg;
end arch;
```

Двоичный счетчик

Двоичный счетчик выдает последовательность кодов, совпадающую с последовательностью положительных целых чисел в некотором диапазоне. Тут опять будет регистр и комбинационная схема, реализующая операцию инкремента.



На VHDL это будет закодировано с помощью оператора +. Для его использования необходимо подключить новые пакеты из стандартных библиотек.

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

Код выглядит так:

```
entity bin_count_pulse is
   port (
      clk, reset   : in  std_logic;
      max_pulse    : out std_logic;
      q      : out std_logic_vector(3 downto 0)
   );
end bin_count_pulse;
architecture arch of bin_count_pulse is
signal r_reg    : std_logic_vector(3 downto 0);
begin
   process(clk,reset)
   begin
      if reset = '1' then
         r_reg <= (others => '0');
      elsif (clk'event and clk = '1') then
         r_reg <= r_reg + 1;
      end if;
   end process;
   max_pulse <= '1' when r_reg = "1111" else '0';
   q  <= r_reg;
end arch;
```

Этот счетчик довольно простой: у него всего два входных сигнала, clk и reset. И он просто считает последовательно, а при переполнении разрядной сетки обнуляется. При достижении значения “1111” на выход счетчика подается однотактовый сигнал max_pulse.

Часто требуется более сложно поведение, а именно, необходим сигнал разрешения работы, чтобы счетчик считал только на определенных тактах (а на других просто сохранял свое значение), могут быть сигнал, определяющий направление счета (+ или -), сигнал начальной установки значения для счета, сигнал сброса счетчика в 0. Вся эта логика будет реализовываться путем добавления дополнительных комбинационных схем вокруг регистра и может быть легко описана на VHDL. Например, рассмотрим часто требующийся в реальной практике счетчик с сигналом синхронного сброса, установки начального значения и разрешения работы.

При этом надо заметить, что лучше всего записывать это в одном операторе IF, просто в разных ветках, и именно в таком порядке – приоритете.

```
entity bin_count_f is
   port (
      clk, reset        : in  std_logic;
      syn_clr, en, load : in  std_logic;
      d     : in  std_logic_vector(3 downto 0);
      q     : out std_logic_vector(3 downto 0)
   );
end bin_count_f;
architecture arch of bin_count_f is
signal r_reg   : std_logic_vector(3 downto 0);
begin
   process(clk,reset)
   begin
      if reset = '1' then
         r_reg <= (others => '0');
      elsif (clk'event and clk = '1') then
         if syn_clr = '1' then
            r_reg <= (others => '0');
         elsif load = '1' then
            r_reg <= d;
         elsif en = '1' then
            r_reg <= r_reg + 1;
         end if;
      end if;
   end process;
   q  <= r_reg;
end arch;
```

Еще один важный пример счетчика с программируемым пределом счета, который задается как внешний сигнал. Тут сигнал синхронного сброса мы создаем сами с помощью дополнительной комбинационной схемы:

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity bin_count_m is
   port (
      clk, reset        : in  std_logic;
      m     : in  std_logic_vector(3 downto 0);
      q     : out std_logic_vector(3 downto 0)
   );
end bin_count_m;
architecture arch of bin_count_m is
signal r_reg   : std_logic_vector(3 downto 0);
signal syn_clr : std_logic;
begin
   process(clk,reset)
   begin
      if reset = '1' then
         r_reg <= (others => '0');
      elsif (clk'event and clk = '1') then
         if syn_clr = '1' then
            r_reg <= (others => '0');
         else
            r_reg <= r_reg + 1;
         end if;
      end if;
   end process;
   syn_clr <= '1' when (r_reg = m) else '0';
   q  <= r_reg;
end arch;
```