Лабораторная работа 8

Синтаксические конструкции

Конечный автомат Мура

Пример реализации конечного автомата Мура.

```
type state_type is (init_state,state1,state2,state3);
signal state, next_state : state_type;
signal output_in : std_logic;

process (clk) begin
   if rising_edge(clk) then
      if srst = '1' then
         state <= init_state; -- Инициализация конечного автомата
         output <= '0';  -- Инициализация выходных сигналов
      else
         state <= next_state;
         output <= output_in;
      end if;
   end if;
end process;

-- Управление выходными сигналами
GET_OUTPUT: process (state) begin
   if state = state3 then
      output_in <= '1';
   else
      output_in <= '0';
   end if;
end process;
-- Управление состояниями конечного автомата
GET_NEXT_STATE: process (state, input1, input2) begin
   next_state <= state;  -- по умолчанию остаемся в текущем состоянии
   case (state) is
      when init_state =>
          next_state <= state1;
      when state1 =>
         if input1 = '1' then
            next_state <= state2;
         end if;
      when state2 =>
         if input2 = '1' then
            next_state <= state3;
         end if;
      when state3 =>
         next_state <= state1;
      when others =>
         next_state <= init_state;
   end case;
end process;
```

Конечный автомат Мили

Пример реализации конечного автомата Мили. Отличия от предыдущего примера – только во втором процессе.

```
type state_type is (init_state,state1,state2,state3);
```

```
signal state, next_state : state_type;
signal output_in : std_logic;

process (clk) begin
   if rising_edge(clk) then
      if srst = '1' then
         state <= init_state; -- Инициализация конечного автомата
         output <= '0';  -- Инициализация выходных сигналов
      else
         state <= next_state;
         output <= output_in;
      end if;
   end if;
end process;

-- Управление выходными сигналами
GET_OUTPUT: process (state, input1, input2) begin
   if (state = state3 and input1 = '1') then
      output_in <= '1';
   else
      output_in <= '0';
   end if;
end process;

-- Управление состояниями конечного автомата
GET_NEXT_STATE: process (state, input1, input2) begin
   next_state <= state;  -- по умолчанию остаемся в текущем состоянии
   case (state) is
      when init_state =>
          next_state <= state1;
      when state1 =>
         if input1 = '1' then
            next_state <= state2;
         end if;
      when state2 =>
         if input2 = '1' then
            next_state <= state3;
         end if;
      when state3 =>
         next_state <= state1;
      when others =>
         next_state <= init_state;
   end case;
end process;
```

Особенности построения конечных автоматов на VHDL[1]

Приведенные выше примеры реализации конечных автоматов не являются единственными в своем роде. Возможна реализация конечных автоматов в двух и даже в одном процессе, а вывод данных можно размещать в тактируемом процессе. Более того, можно управлять кодированием состояний конечного автомата выполняя традиционный размен быстродействия на количество используемых ресурсов. Далее на примерах будут рассмотрены различные варианты реализации одного и того же конечного автомата. Диаграммы состояний построены в среде HDL Designer компании Mentor Graphics.

[1] Материал основан на документе Predicting the output of Finite State Machines компании Mentor Graphics.

Простой автомат Мура

Рассмотрим сначала простой конечный автомат Мура. На рис. 1а приведена диаграмма состояний такого автомата, а на рис. 1б – соответствующая ему структурная схема. Ниже представлен VHDL-код, автоматически сгенерированный из редактора конечных автоматов среды HDL Designer.

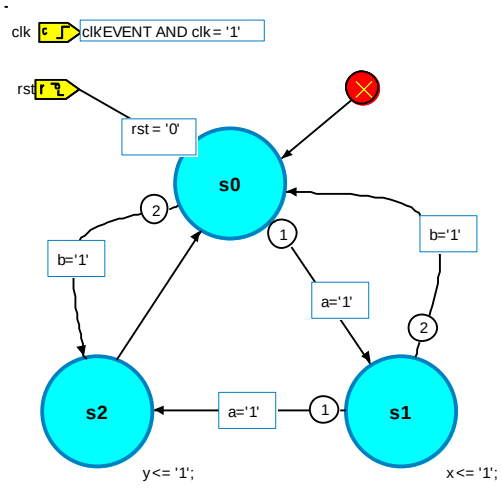

Рис. 1а. Пример конечного автомата Мура

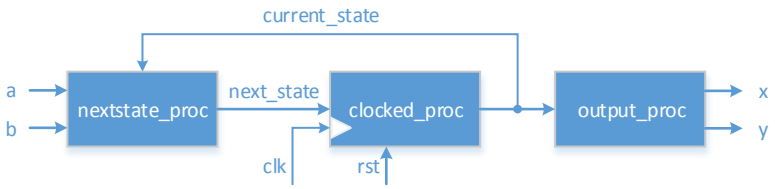

Рис. 1б. Структурная схема конечного автомата на рис. 1а.

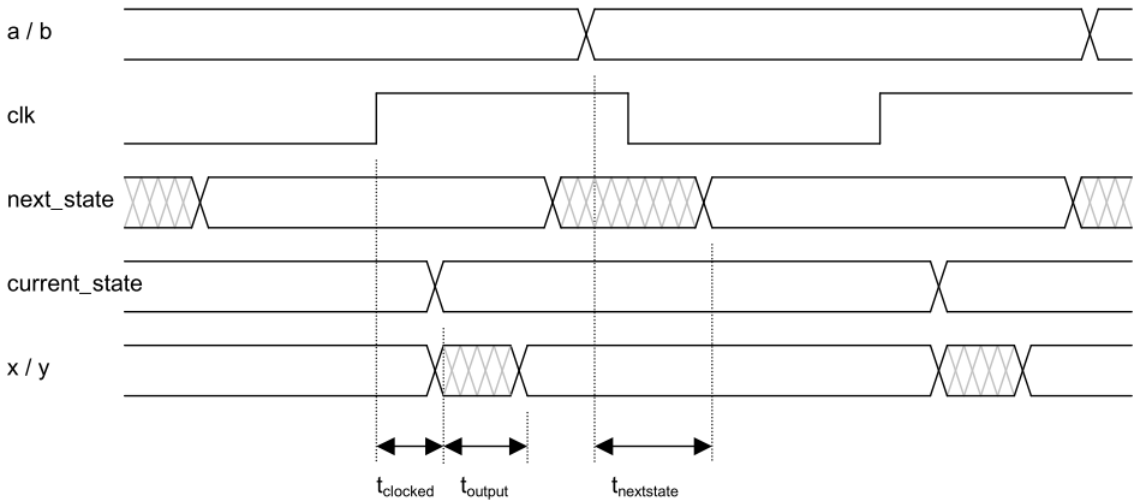

Рис. 1в. Временные диаграммы работы конечного автомата на рис. 1а

В приведенном примере значения сигналов на входах a и b совместно с current_state определяют значение следующего состояния автомата next_state. По значению current_state определяются выходные сигналы x и y.

Важно отметить, что присваивание значений выходным сигналам привязано к **состояниям**, а не ветвям диаграммы состояний (автомат Мура). Присваивание значений выходным сигналам **комбинационное**.

В выходном процессе явное обновление значений сигналов x и y выполняется только в интересующих состояниях (внутри оператора case). В остальных случаях выходным сигналам присваиваются значения по умолчанию – нули (секция Default values). Такой подход с одной стороны сокращает объем исходного кода и улучшает его читаемость, а с другой – предотвращает появление триггеров-защелок в проекте.

В тактируемом процессе (clocked_proc) присутствует только сигнал current_state, все остальные сигналы, включая выходы, – комбинаторные.

На рис. 1.в показаны временные диаграммы работы конечного автомата на рис. 1а. Задержка на выходах x и y определяется двумя составляющими: время реакции на фронт сигнала тактирования и время установления – $t_{clocked} + t_{output}$. Выходные сигналы могут искажаться. Все искажения закончатся, если $t_{clocked} + t_{output} + t_{setup} < T_{clk}$.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY moore_example IS
   PORT(
      a   : IN     std_logic;
      b   : IN     std_logic;
      clk : IN     std_logic;
      rst : IN     std_logic;
      x   : OUT    std_logic;
      y   : OUT    std_logic
   );
END moore _example ;

ARCHITECTURE fsm OF sm_example IS
   TYPE STATE_TYPE IS (s0,s1,s2);
   -- Declare current and next state signals
   SIGNAL current_state : STATE_TYPE;
   SIGNAL next_state : STATE_TYPE;
BEGIN
   -----------------------------------------------------------------
   clocked_proc : PROCESS (clk) BEGIN
      IF rising_edge(clk) THEN
         IF (rst = '0') THEN
            current_state <= s0;
         ELSE
            current_state <= next_state;
         END IF;
      END IF;
   END PROCESS clocked_proc;
   -----------------------------------------------------------------
   nextstate_proc : PROCESS (a,b,current_state) BEGIN
      CASE current_state IS
         WHEN s0 =>
            IF (a='1') THEN
               next_state <= s1;
            ELSIF (b='1') THEN
               next_state <= s2;
            ELSE
               next_state <= s0;
            END IF;
         WHEN s1 =>
            IF (a='1') THEN
               next_state <= s2;
            ELSIF (b='1') THEN
               next_state <= s0;
            ELSE
```

```
                next_state <= s1;
             END IF;
          WHEN s2 =>
             next_state <= s0;
          WHEN OTHERS =>
             next_state <= s0;
       END CASE;
    END PROCESS nextstate_proc;
    -----------------------------------------------------------------
    output_proc : PROCESS (current_state) BEGIN
       -- Default Assignment
       x <= '0';
       y <= '0';
       -- Combined Actions
       CASE current_state IS
          WHEN s1 =>
             x <= '1';
          WHEN s2 =>
             y <= '1';
          WHEN OTHERS =>
             NULL;
       END CASE;
    END PROCESS output_proc;
END fsm;
```

Простой автомат Мили

В конечном автомате Мура в предыдущем примере присутствуют задержки при обновлении выходных сигналов, также выходные сигналы могут искажаться. Аналогичный конечный автомат может быть реализован как автомат Мили, в котором выходные сигналы обновляются по фронту сигнала тактирования (registered).

На рис. 2а приведена диаграмма состояний такого автомата, а на рис. 2б – соответствующая ему структурная схема. Ниже представлен VHDL-код, автоматически сгенерированный из редактора конечных автоматов среды HDL Designer.

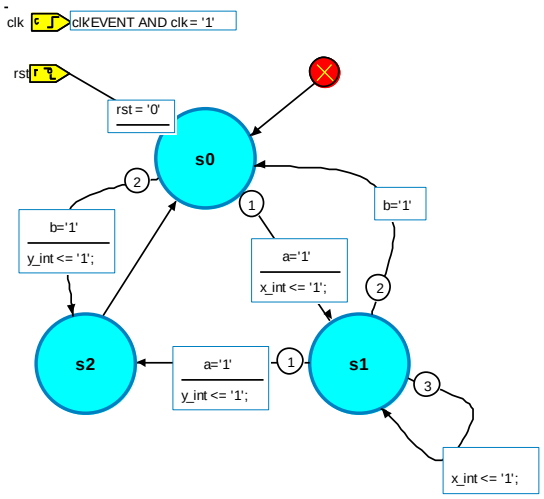

Рис. 2а. Пример конечного автомата Мили

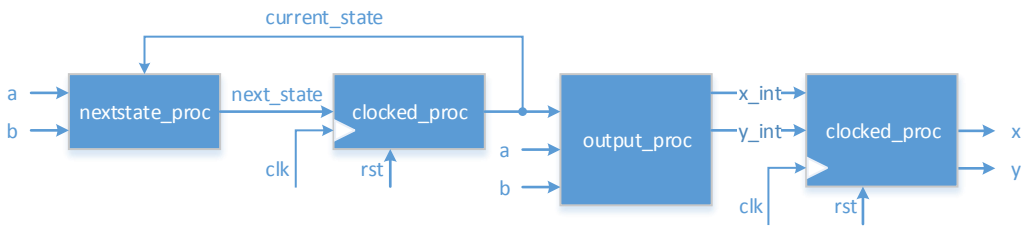

Рис. 2б. Структурная схема конечного автомата на рис. 2а.

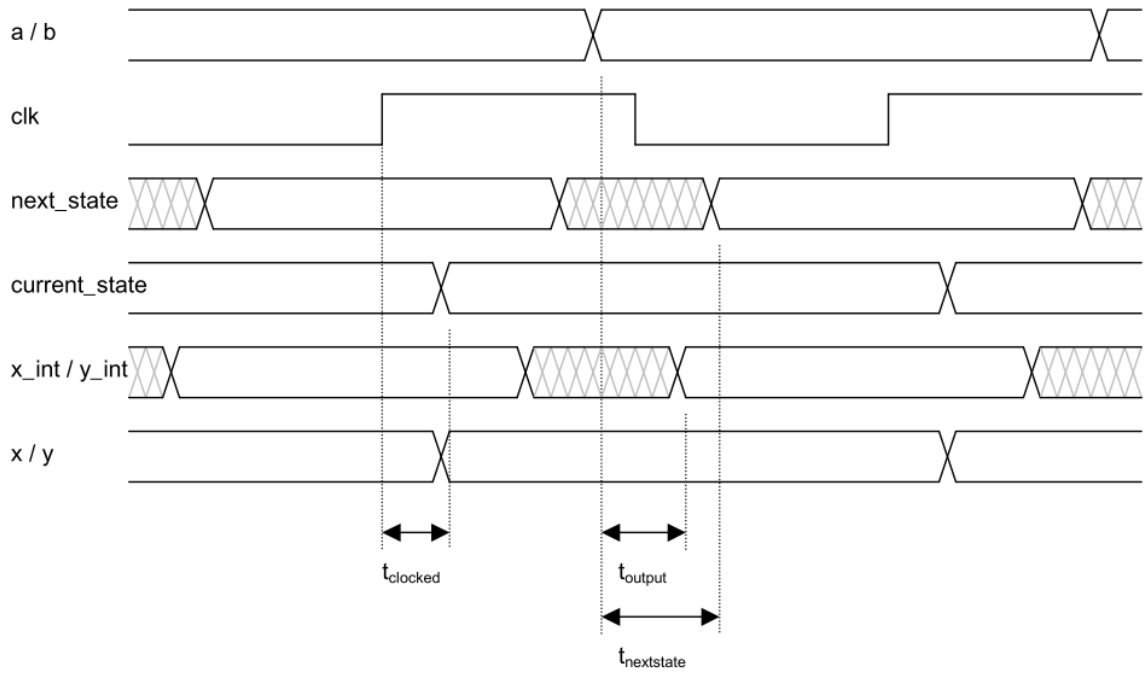

Рис. 2в. Временные диаграммы работы конечного автомата на рис. 2а

В приведенном примере значения сигналов на входах a и b совместно с current_state определяют значение следующего состояния автомата next_state. По значениям current_state и входов a и b определяются внутренние версии выходных сигналов x_int и y_int. Далее эти сигналы защелкиваются по фронту тактовой частоты и выводятся из модуля (выходы x и y).

В отличие от предыдущего примера, здесь присваивание значений выходным сигналам привязано к **ветвям (переходам)**, а не состояниям диаграммы состояний (автомат Мили). Также появились внутренние версии выходных сигналов (*_int). Присваивание значений внутренним версиям выходных сигналов по-прежнему **комбинаторное**, а вывод данных из автомата – по фронту.

Зафиксируем внимание на том, что из состояния s1 выходит третья ветвь, замкнутая в этом же состоянии. Это необходимо для того, чтобы в состоянии s1 на выходе x была единица все то время, пока автомат находится в состоянии s1 (если убрать этот переход, то единица на выходе x будет присутствовать только на одном такте, потом выход будет сброшен в состояние по умолчанию).

На рис. 2.в показаны временные диаграммы работы конечного автомата на рис. 2а. Задержка на выходах x и y равна $t_{clocked}$. Искажения выходных сигналов **отсутствуют**. При этом для каждого выхода потребуется поставить (синтезатору) дополнительный триггер. Описание конечного автомата немного усложнилось за счет того, что теперь на каждом переходе необходимо указывать значения выходных сигналов.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY mealy_example IS
   PORT(
      a   : IN     std_logic;
      b   : IN     std_logic;
      clk : IN     std_logic;
```

```
      rst : IN     std_logic;
      x   : OUT    std_logic;
      y   : OUT    std_logic
   );
END mealy_example ;

ARCHITECTURE fsm OF mealy_example2 IS
   TYPE STATE_TYPE IS (s2,s0,s1);
   -- Declare current and next state signals
   SIGNAL current_state : STATE_TYPE;
   SIGNAL next_state : STATE_TYPE;
   -- Declare any pre-registered internal signals
   SIGNAL x_int : std_logic ;
   SIGNAL y_int : std_logic ;
BEGIN
   -----------------------------------------------------------------
   clocked_proc : PROCESS (clk) BEGIN
      IF rising_edge(clk) THEN
         IF (rst = '0') THEN
            current_state <= s0;
            -- Default Reset Values
            x <= '0';
            y <= '0';
         ELSE
            current_state <= next_state;
            -- Registered output assignments
            x <= x_int;
            y <= y_int;
         END IF;
      END IF;
   END PROCESS clocked_proc;

   -----------------------------------------------------------------
   nextstate_proc : PROCESS (a,b,current_state) BEGIN
      CASE current_state IS
         WHEN s2 =>
            next_state <= s0;
         WHEN s0 =>
            IF (a='1') THEN
               next_state <= s1;
            ELSIF (b='1') THEN
               next_state <= s2;
            ELSE
               next_state <= s0;
            END IF;
         WHEN s1 =>
            IF (a='1') THEN
               next_state <= s2;
            ELSIF (b='1') THEN
               next_state <= s0;
            ELSE
               next_state <= s1;
            END IF;
         WHEN OTHERS =>
            next_state <= s0;
      END CASE;
   END PROCESS nextstate_proc;

   -----------------------------------------------------------------
   output_proc : PROCESS (a,b,current_state) BEGIN
      -- Default Assignment
      x_int <= '0';
      y_int <= '0';
```

```
      -- Combined Actions
      CASE current_state IS
         WHEN s0 =>
            IF (a='1') THEN
               x_int <= '1';
            ELSIF (b='1') THEN
               y_int <= '1';
            END IF;
         WHEN s1 =>
            IF (a='1') THEN
               y_int <= '1';
            ELSIF (b='1') THEN
            ELSE
               x_int <= '1';
            END IF;
         WHEN OTHERS =>
            NULL;
      END CASE;
   END PROCESS output_proc;

END fsm;
```

Простой автомат Мили с тактируемыми выходными сигналами

В предыдущем примере выходные сигналы (их внутренние версии) формировались сначала комбинаторно, а затем подавались на выход через триггеры. Возможна реализация, когда выходные сигналы сразу формируются по фронту сигнала тактирования (а не комбинаторно).

На рис. 3а приведена диаграмма состояний такого автомата, а на рис. 3б – соответствующая ему структурная схема. Ниже представлен VHDL-код, автоматически сгенерированный из редактора конечных автоматов среды HDL Designer.

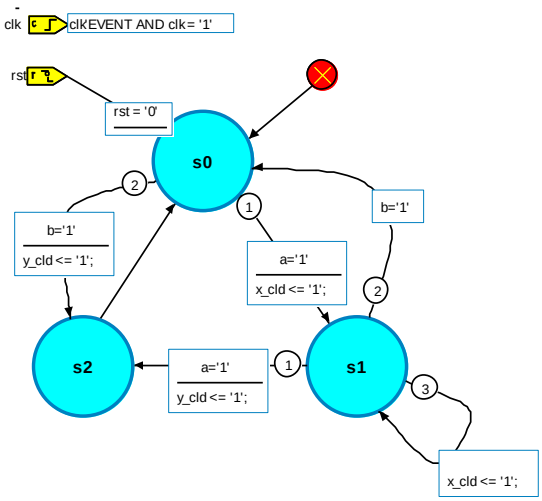

Рис. 3а. Пример конечного автомата Мили

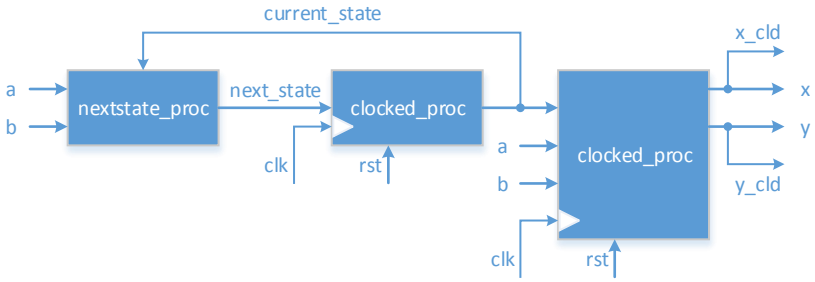

Рис. 3б. Структурная схема конечного автомата на рис. 3а.

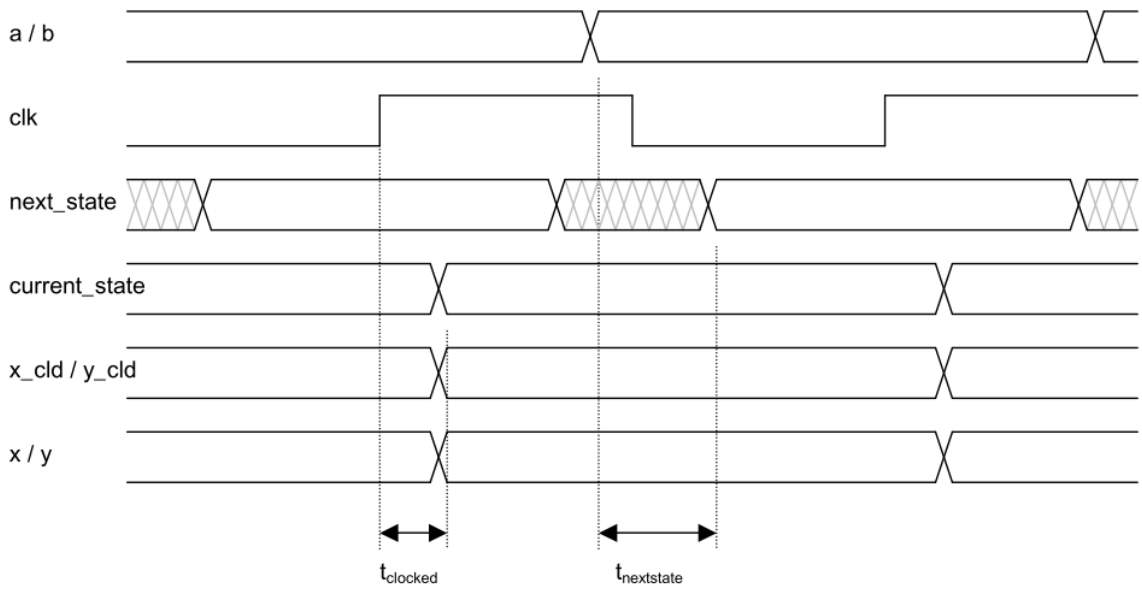

Рис. 3в. Временные диаграммы работы конечного автомата на рис. 3а

В отличие от предыдущего примера, в текущем присваивание значений внутренним версиям (*_cld) выходных сигналов выполняется **по фронту сигнала тактирования**, а не комбинаторно. Весь конечный автомат описан двумя процессами. При этом внутри конечного автомата доступна защелкнутая по фронту версия выходных сигналов. Иногда это бывает полезным для повышения тактовой частоты устройства (естественно, ценой дополнительных ресурсов).

На рис. 3.в показаны временные диаграммы работы конечного автомата на рис. 3а. Задержка на выходах x и y равна $t_{clocked}$. Искажения выходных сигналов **отсутствуют.**

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY mealy_example2 IS
   PORT(
      a   : IN     std_logic;
      b   : IN     std_logic;
      clk : IN     std_logic;
      rst : IN     std_logic;
      x   : OUT    std_logic;
      y   : OUT    std_logic
   );
END mealy_example2 ;

ARCHITECTURE fsm OF mealy_example2 IS
   TYPE STATE_TYPE IS (s2,s0,s1);
   -- Declare current and next state signals
   SIGNAL current_state : STATE_TYPE;
   SIGNAL next_state : STATE_TYPE;
   -- Declare any pre-registered internal signals
   SIGNAL x_cld : std_logic ;
   SIGNAL y_cld : std_logic ;

BEGIN
   -----------------------------------------------------------------
   clocked_proc : PROCESS (clk) BEGIN
      IF rising_edge(clk) THEN
         IF (rst = '0') THEN
            current_state <= s0;
            -- Default Reset Values
```

```
            x_cld <= '0';
            y_cld <= '0';
         ELSE
            current_state <= next_state;
            -- Default Assignment To Internals
            x_cld <= '0';
            y_cld <= '0';
            -- Combined Actions
            CASE current_state IS
               WHEN s0 =>
                  IF (a='1') THEN
                     x_cld <= '1';
                  ELSIF (b='1') THEN
                     y_cld <= '1';
                  END IF;
               WHEN s1 =>
                  IF (a='1') THEN
                     y_cld <= '1';
                  ELSIF (b='1') THEN
                  ELSE
                     x_cld <= '1';
                  END IF;
               WHEN OTHERS =>
                  NULL;
            END CASE;
         END IF;
      END IF;
   END PROCESS clocked_proc;

   -----------------------------------------------------------------------
   nextstate_proc : PROCESS (a,b,current_state) BEGIN
      CASE current_state IS
         WHEN s2 =>
            next_state <= s0;
         WHEN s0 =>
            IF (a='1') THEN
               next_state <= s1;
            ELSIF (b='1') THEN
               next_state <= s2;
            ELSE
               next_state <= s0;
            END IF;
         WHEN s1 =>
            IF (a='1') THEN
               next_state <= s2;
            ELSIF (b='1') THEN
               next_state <= s0;
            ELSE
               next_state <= s1;
            END IF;
         WHEN OTHERS =>
            next_state <= s0;
      END CASE;
   END PROCESS nextstate_proc;

   -- Concurrent Statements
   -- Clocked output assignments
   x <= x_cld;
   y <= y_cld;
END fsm;
```

Задание к лабораторной работе 8

Реализовать конечные автоматы по словесному описанию диаграмм состояний.

Необходимо реализовать:

1. * Реализовать кодер сверточного кода (7,5) и тестбенч к нему.

   Кодер реализовывать не на основе регистра сдвига, а на основе общего синтаксиса конечных автоматов Мили/Мура.
   Разрабатываемое устройство должно иметь входы сброса, тактирования и информационный и двухразрядный выход для кодированной последовательности.
   Начальное состояние регистра сдвига кодера – все нули.
   Начальное значение выходного сигнала – все нули.
   Результат сложения по модулю 2 ячеек 7 отправлять в старший разряд выхода, результат сложения ячеек 5 – в младший разряд выхода.
   Использовать вариант реализации конечного автомата на основе двух процессов.
   При моделировании в качестве информационной (кодируемой) последовательности подавать 1. Напомним, что в процессе работы кодера на каждый входной бит генерируется два выходных бита. Это значит, что разрядность информационного входа должна быть равна 1, а выхода – 2.
   Тестбенч должен генерировать сигнал сброса и сигнал тактирования. Результатом выполнения подзадания являются корректные временные диаграммы работы кодера и прошивка FPGA.

   Использовать следующий шаблон для объявления интерфейса модуля и архитектуры:

```
--------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity lab81 is
    Port ( clk  : in STD_LOGIC;
           srst : in STD_LOGIC;
           info : in STD_LOGIC;
           coded : out STD_LOGIC_VECTOR (1 downto 0));
end lab81;

architecture a of lab81 is
begin
...
--------------------------------------
```

2. **Разработать конечный автомат, который имеет четыре состояния, и тестбенч к нему.

   Устройство должно иметь следующие входы: тактирования, синхронного сброса, на трехразрядных беззнаковых числа, кнопка. Устройство должно иметь один 8-разрядный выход.
   Состояния конечного автомата:
   - 0 – начальное состояние ожидания, в котором с периодом примерно 1 с мигает один светодиод (младший из восьми);
   - 1 – состояние “Суммирование”, в котором выполняется комбинаторное (не в процессе) суммирование двух беззнаковых трехразрядных чисел;
   - 2 – состояние “Умножение”, в котором выполняется комбинаторное умножение двух трехразрядных беззнаковых чисел;
   - 3 – состояние “Счетчик Грея”, в котором выполняется 8-ми разрядный счет в коде Грея с периодом ровно 0,25 с на основе перекодировки прямого суммирующего счетчика. Для определенности автопроверки, перекодировку выполнять комбинаторно, инкрементирование перекодируемого счетчика выполнять при равенстве нулю счетчика, задающего период 0,25 с.
   Из каждого состояния возможен переход только в следующее состояние: из 0 в 1; из 1 в 2; из 2 в 3 и из 3 в 0. Переход в следующее состояние должен выполняться по нажатию кнопки. Для предотвращения дребезга контактов состояние кнопки считывать с частотой примерно 0,5 с.
   Результаты выводить на светодиоды, в качестве информационных входов использовать слайдеры.
   Для ускорения времени моделирования и/или автопроверки отключить антидребезг (т.е. вход Кнопка подключен прямо на конечный автомат), а период мигания светодиода и период счета счетчика Грея устанавливать равными 256 периодам сигнала тактирования.

Каждый модуль (мигание светодиодом, суммирование и т.д.) должен работать непрерывно независимо от состояния конечного автомата. Конечный автомат только подключает к выходным светодиодам выход того или иного модуля. Например, счетчик Грея считает, но на выходе результат перемножения входов.

Обратить внимание, что выходные сигналы некоторых модулей необходимо дополнить нулями для получения на выходе 8-разрядного числа.

Использовать следующий шаблон для объявления интерфейса модуля и архитектуры:

```
--------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE ieee.std_logic_arith.all;

entity lab82 is
    Port ( clk  : in STD_LOGIC;
           srst : in STD_LOGIC;
           x1   : in STD_LOGIC_VECTOR (2 downto 0);
           x2   : in STD_LOGIC_VECTOR (2 downto 0);
           btn  : in STD_LOGIC;
           y    : out STD_LOGIC_VECTOR (7 downto 0));
end lab82;

architecture Behavioral of lab82 is
    TYPE STATE_TYPE IS (s0,s1,s2,s3);
    -- Declare current and next state signals
    SIGNAL current_state : STATE_TYPE;
    SIGNAL next_state : STATE_TYPE;
    -- Declare any pre-registered internal signals
    SIGNAL y_cld : std_logic_vector(7 downto 0);

    signal sum   : std_logic_vector (7 downto 0);
    signal mult  : std_logic_vector (7 downto 0);
    signal gcntr : std_logic_vector (7 downto 0);
    signal led   : std_logic;

    signal ibtn  : std_logic;

    component lab62 is
        Port ( clk  : in STD_LOGIC;
               dout : out STD_LOGIC);
    end component lab62;

    component gray_cntr is
        Port ( clk   : in STD_LOGIC;
               srst  : in STD_LOGIC;
               gcntr : out STD_LOGIC_VECTOR (7 downto 0));
    end component gray_cntr;
begin
    ibtn <= btn; -- For simulation
...
--------------------------------------
```