Šablony v 1.1

Hradlové pole

```
Family      Cyclone IVE
Device      EP4CE6E22C8
```

Projekt – zdrojový soubor

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity name is
    Port (
--          in, out ports
   );
end name;

architecture Behavioral of name is
--           signals
begin
--          architecture body
end Behavioral;
```

Komponenta

Vkládá se do části **signály**.

```
component <name>
Port (
   <name> :  in std_logic;
   <name> :  in std_logic_vector(x downto 0);
   <name> :  out std_logic;
   <name> :  out std_logic_vector(x downto 0)
   );
end component;
```

Vkládá se do části **architecture**.

```
<instance>:<name>
   Port map(  <io_komponenta1> => <signal1>,
              <io_komponenta2> => <signal2>);
```

Porty

```
<name> :  in std_logic;
<name> :  in std_logic_vector(x downto 0);
<name> :  out std_logic;
<name> :  out std_logic_vector(x downto 0);
```

Signály

```
signal  <name> :  std_logic;
signal  <name> :  std_logic := '0';
signal  <name> :  std_logic_vector(x downto 0);
signal  <name> :  std_logic_vector(4 downto 0) := "10010";
signal  <name> :  std_logic_vector(x downto 0) := (others => '0');
```

Proměnné

```
variable  <name> :  std_logic_vector(x downto 0);
signal  <name> :  std_logic_vector(4 downto 0) := "10010";
signal  <name> :  std_logic_vector(x downto 0) := (others => '0');
```

Nástupná a sestupná hrana

```
rising_edge(<clk>)
falling_edge(<clk>)
```

Kombinační obvody

Základní hradla

```
<output> <= <input_1> and <input_2> and <input_3>
<output> <= <input_1> or <input_2> or <input_3>
<output> <= <input_1> xor <input_2> xor <input_3>
<output> <= not <input>
<output> <= not (<input_1> and <input_2> and <input_3>)
<output> <= <input_1> nand <input_2>
<output> <= not (<input_1> or <input_2> or <input_3>)
<output> <= <input_1> nor <input_2>
<output> <= <input_1> xnor <input_2> xnor <input_3>
```

Operátory

```
Přiřazení                          <=
Sloučení (sjednocení)              &
-- Spojení dvou signálů.
B <= A & C(7 downto 1);
```

Podmínky - asynchronní

Select /When Statement

```
with <choice_expression> select
   <name> <= <expression> when <choices>,
      <expression> when <choices>,
      <expression> when others;
```

When /Else Statement

```
<name> <= <expression> when <condition> else
      <expression> when <condition> else
      <expression>;
```

Podmínky – synchronní

Case

```
process(<clock>)
begin
if ( <clock>'event and <clock> ='1') then
   case (<2-bit select>) is
      when "00" =>
            <statement>;
      when "01" =>
            <statement>;
      when "10" =>
             <statement>;
      when "11" =>
            <statement>;
      when others =>
            <statement>;
   end case;
end if;
end process;
```

Case se synchronním resetem

```
process(<clock>)
begin
if ( <clock>'event and <clock> ='1') then
   if ( <reset> = '1') then
      <statement>;
   else
      case (<2-bit select>) is
            when "00" =>
                  <statement>;
            when "01" =>
                  <statement>;
            when "10" =>
                  <statement>;
            when "11" =>
                  <statement>;
            when others =>
                  <statement>;
      end case;
   end if;
end if;
end process;
```

If/Elsif/Else

```
process(<clock>)
begin
if ( <clock>'event and <clock> ='1') then
   if <condition> then
      <statement>
   elsif <condition> then
      <statement>
   else
      <statement>
    end if;
end if;
end process;
```

Komparátor

```
process(<clock>)
begin
if (<clock>'event and <clock> ='1') then
   if ( <input1> = <input2> ) then
      <output> <= '1';
   else
      <output> <= '0';
   end if;
end if;
end process;
```

Relační operátory

Rovnost	**=**
Větší než	**>**
Větší nebo rovno	**>=**
Menší než	**<**
Menší nebo rovno	**<=**
Nerovnost	**/=**

Dekodér

```
process(<clock>)
begin
   if ( <clock>'event and <clock> ='1') then
      if ( <reset> = '1') then
            <output> <= "0000";
      else
            case <input> is
                  when "00" => <output> <= "0001";
                  when "01" => <output> <= "0010";
                  when "10" => <output> <= "0100";
                  when "11" => <output> <= "1000";
                  when others => <output> <= "0000";
            end case;
      end if;
   end if;
end process;
```

Kodér

```
process(<clock>)
begin
if ( <clock>'event and <clock> ='1') then
   if (<reset> = '1') then
      <output> <= "00";
   else
      case <input> is
            when "0001" =>
                  <output> <= "00";
            when "0010" =>
                  <output> <= "01";
            when "0100" =>
                  <output> <= "10";
            when "1000" =>
                  <output> <= "11";
            when others =>
                  <output> <= "00";
      end case;
   end if;
end if;
end process;
```

Převodník kódu

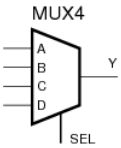
HEX/7 segment

```
--HEX-to-seven-segment decoder
--   HEX:   in    STD_LOGIC_VECTOR (3 downto 0);
--   LED:   out   STD_LOGIC_VECTOR (6 downto 0);
--
-- segment encoinputg
--      0
--     ---
--  5 |   | 1
--     ---   <- 6
--  4 |   | 2
--     ---
--      3

with HEX select
LED <= "1111001" when "0001",--1
   "0100100" when "0010",    --2
   "0110000" when "0011",    --3
   "0011001" when "0100",    --4
   "0010010" when "0101",    --5
   "0000010" when "0110",    --6
   "1111000" when "0111",    --7
   "0000000" when "1000",    --8
   "0010000" when "1001",    --9
   "0001000" when "1010",    --A
   "0000011" when "1011",    --b
   "1000110" when "1100",    --C
   "0100001" when "1101",    --d
   "0000110" when "1110",    --E
   "0001110" when "1111",    --F
   "1000000" when others;    --0
```

Multiplexer - asynchronní

Jednovstupý




```
Y <= A when (SEL = "00") else
     B when (SEL = "01") else
     C when (SEL = "10") else
     D;
```

Zápis multiplexoru pomocí konstrukce with-select

```
with SEL select
   Y <= A when "00",
        B when "01",
        C when "10",
        D when others;
```

Multiplexer - synchronní

Zápis pomocí konstrukce case uvnitř procesu

```
process(A,B,C,D,SEL)
begin
   case SEL is
       when "00" =>
         Y <= A;
       when "01" =>
         Y <= B;
       when "10" =>
         Y <= C;
       when others =>
         Y <= D;
   end case;
end process;
```

Zápis pomocí konstrukce if - elsif uvnitř procesu

```
process(A,B,C,D,SEL)
begin
   if (SEL="00") then
      Y <= A;
   elsif (SEL="01") then
      Y <= B;
   elsif (SEL="10") then
```

```
      Y <= C;
   else
      Y <= D;
   end if;
end process;
```

Dvouvstupový

```
<output> <= <input1> WHEN <selector>  = '1'  ELSE
                 <input2>;
```

Čtyřvstupový

```
process (<selector>,<input1>,<input2>,<input3>,<input4>)
begin
   case <selector> is
      when "00" => <output> <= <input1>;
      when "01" => <output> <= <input2>;
      when "10" => <output> <= <input3>;
      when "11" => <output> <= <input4>;
      when others => <output> <= <input1>;
   end case;
end process;
```

Demultiplexer - asynchronní

Dvouvstupový

```
<output_1>  <=  <input>  WHEN  <selector>  = '1'  ELSE
                              '0';
<output_2>  <=  <input>  WHEN  <selector>  = '1'  ELSE
                              '0';
```

Čtyřvstupový

```
process (<selector>,<input>)
begin
      case <selector> is
                  when "00" =>
                        <output_1> <= <input>;
                        <output_2> <= '0';
                        <output_3> <= '0';
                        <output_4> <= '0';
                  when "01" =>
                        <output_1> <= '0';
                        <output_2> <= <input>;
                        <output_3> <= '0';
                        <output_4> <= '0';
                  when "10" =>
                        <output_1> <= '0';
                        <output_2> <= '0';
                        <output_3> <= <input>;
                        <output_4> <= '0';
                  when "11" =>
                        <output_1> <= '0';
                        <output_2> <= '0';
                        <output_3> <= '0';
                        <output_4> <= <input>;
                when others =>
                        <output_1> <= '0';
                        <output_2> <= '0';
                        <output_3> <= '0';
                        <output_4> <= '0';

      end case;
end process;
```

Generátor paritního bitu

Lichá parita

```
LP  <= 1  xor  bit0  xor  bit1  xor  bit2  xor ................ xor  bitn
```

Sudá parita

```
SP <= 0  xor  bit0 xor  bit1  xor  bit2  xor ................ xor  bitn
```

Sekvenční obvody

Klopný obvod

Klopný obvod D

```
process (<clock>)
begin
   if rising_edge(<clock>) then
      <output> <= <input>;
   end if;
end process;
```

Klopný obvod D – s asynchronním resetem

```
process (<clock>, <reset>)
begin
   if <reset>='1' then
      <output> <= '0';
   elsif (<clock>'event and <clock>='1') then
      <output> <= <input>;
   end if;
end process;
```

Klopný obvod D – se synchronním resetem

```
process (<clock>)
begin
   if <clock>'event and <clock>='1' then
      if <reset>='1' then
            <output> <= '0';
      else
            output> <= <input>;
      end if;
   end if;
end process;
```

Klopný obvod D – hradlovaný (clock enable), s asynchronním resetem

```
process (<clock>, <reset>)
begin
   if <reset>='1' then
      <output> <= '0';
   elsif (<clock>'event and <clock>='1') then
      if <clock_enable> = '1' then
            <output> <= <input>;
      end if;
   end if;
end process;
```

Klopný obvod D – hradlovaný (clock enable), se synchronním resetem

```
process (<clock>)
begin
   if <clock>'event and <clock>='1' then
      if <reset>='1' then
            <output> <= '0';
      elsif <clock_enable> ='1' then
            <output> <= <input>;
      end if;
   end if;
end process;
```

Debounce –hrana stisku

```
process(<clock>)
begin
   if (<clock>'event and <clock> = '1') then
      if (<reset> = '1') then
            Q1 <= '0';
            Q2 <= '0';
            Q3 <= '0';
      else
            Q1 <= D_IN;
            Q2 <= Q1;
            Q3 <= Q2;
      end if;
      Q_OUT <= Q1 and Q2 and (not Q3);
   end if;
end process;
```

Debounce – trvání stisku

```
process(clk)
begin
   if (clk'event and clk = '1') then
      if ce = '1' then
            Q1 <= btn;
            Q2 <= Q1;
            Q3 <= Q2;
      end if;
      Q_OUT <= Q1 and Q2;
   end if;
end process;
```

Čítač

Čítač – binární, vzestupný

```
process (<clock>)
begin
   if <clock>='1' and <clock>'event then
            <count> <= <count> + 1;
   end if;
end process;
```

Čítač - binární, vzestupný, hradlovaný (clock enable)

```
process (<clock>)
begin
   if <clock>='1' and <clock>'event then
      if <clock_enable>='1' then
            <count> <= <count> + 1;
      end if;
   end if;
end process;
```

Čítač –binární, vzestupný, hradlovaný, s asynchronním resetem

```
process (<clock>, <reset>)
begin
   if <reset>='1' then
      <count> <= (others => '0');
   elsif <clock>='1' and <clock>'event then
      if <clock_enable>='1' then
            <count> <= <count> + 1;
      end if;
   end if;
end process;
```

Čítač –binární, vzestupný, hradlovaný, se synchronním resetem

```
process (<clock>)
begin
   if <clock>='1' and <clock>'event then
      if <reset>='1' then
         <count> <= (others => '0');
      elsif <clock_enable>='1' then
         <count> <= <count> + 1;
      end if;
   end if;
end process;
```

Čítač –binární, vzestupný, hradlovaný, s nastavením a asynchronním resetem

```
process (<clock>, <reset>)
begin
   if <reset>='1' then
      <count> <= (others => '0');
   elsif <clock>='1' and <clock>'event then
      if <clock_enable>='1' then
         if <load_enable>='1' then
            <count> <= <input>;
         else
            <count> <= <count> + 1;
         end if;
      end if;
   end if;
end process;
```

Čítač –binární, vzestupný, hradlovaný, s nastavením a synchronním resetem

```
process (<clock>)
begin
   if <clock>='1' and <clock>'event then
      if <reset>='1' then
         <count> <= (others => '0');
      elsif <clock_enable>='1' then
         if <load_enable>='1' then
            <count> <= <input>;
         else
            <count> <= <count> + 1;
         end if;
      end if;
   end if;
end process;
```

Čítač – binární, reverzibilní

```
process (<clock>)
begin
   if <clock>='1' and <clock>'event then
      if <count_direction>='1' then
         <count> <= <count> + 1;
      else
         <count> <= <count> - 1;
      end if;
   end if;
end process;
```

Čítač –binární, reverzibilní, hradlovaný, s nastavením a asynchronním resetem

```
process (<clock>, <reset>)
begin
   if <reset>='1' then
      <count> <= (others => '0');
   elsif <clock>='1' and <clock>'event then
      if <clock_enable>='1' then
         if <load_enable>='1' then
            <count> <= <input>;
         else
            if <count_direction>='1' then
               <count> <= <count> + 1;
            else
               <count> <= <count> - 1;
            end if;
         end if;
      end if;
   end if;
end process;
```

Čítač –binární, reverzibilní, hradlovaný, s nastavením a asynchronním resetem

```
process (<clock>)
begin
   if <clock>='1' and <clock>'event then
      if <reset>='1' then
         <count> <= (others => '0');
      elsif <clock_enable>='1' then
         if <load_enable>='1' then
            <count> <= <input>;
         else
            if <count_direction>='1' then
               <count> <= <count> + 1;
            else
               <count> <= <count> - 1;
            end if;
         end if;
      end if;
   end if;
end process;
```

Posuvný registr

Posuvný registr – kruhový, vlevo

```
process (<clock>)
begin
   if <clock>='1' and <clock>'event then
            <reg> <= <reg>(x-1 downto 0) &  <reg>(x);
   end if;
end process;
```

Posuvný registr – kruhový, vpravo

```
process (<clock>)
begin
   if <clock>='1' and <clock>'event then
            <reg> <= <reg>( 0) &  <reg>(x downto 1);
   end if;
end process;
```

Posuvný registr – vlevo s nasouváním nuly

```
process (<clock>)
begin
   if <clock>='1' and <clock>'event then
            <reg> <= <reg>(x-1 downto 0) &  ‘0‘;
   end if;
end process;
```

Posuvný registr – vpravo s nasouváním jedničky

```
process (<clock>)
begin
   if <clock>='1' and <clock>'event then
            <reg> <= ‘1‘ &  <reg>(x downto 1);
   end if;
end process;
```

Posuvný registr – vlevo s nasouváním nuly a přenosem

```
process (<clock>)
begin
   if <clock>='1' and <clock>'event then
            <reg> <= <reg>(x-1 downto 0) &  ‘0‘;
      <c> <= <reg>(x);
   end if;
end process;
```

Intel

```
NET "btnstop" CLOCK_DEDICATED_ROUTE = FALSE;

NET "btn0" LOC = B18;
NET "btn1" LOC = D18;
NET "btn2" LOC = E18;
NET "btn3" LOC = H13;

NET "clk" LOC = B8;

NET "seg[0]" LOC = L18;
NET "seg[1]" LOC = F18;
NET "seg[2]" LOC = D17;
NET "seg[3]" LOC = D16;
NET "seg[4]" LOC = G14;
NET "seg[5]" LOC = J17;
NET "seg[6]" LOC = H14;
NET "seg[7]" LOC = C17;

NET "cislo[0]" LOC = F17;
NET "cislo[1]" LOC = H17;
NET "cislo[2]" LOC = C18;
NET "cislo[3]" LOC = F15;

NET "ld[0]" LOC = J14;
NET "ld[1]" LOC = J15;
NET "ld[2]" LOC = K15;
NET "ld[3]" LOC = K14;
NET "ld[4]" LOC = E16;
NET "ld[5]" LOC = P16;
NET "ld[6]" LOC = E4;
NET "ld[7]" LOC = F4;

NET "sw1[0]" LOC = G18;
NET "sw1[1]" LOC = H18;
NET "sw1[2]" LOC = K18;
NET "sw1[3]" LOC = K17;

NET "sw2[0]" LOC = L14;
NET "sw2[1]" LOC = L13;
NET "sw2[2]" LOC = N17;
NET "sw2[3]" LOC = R17;
```