Primer implementacije CPE v vezju FPGA

- CPE lahko implementiramo (tudi) v programirljivem digitalnem vezju FPGA (Field-Programmable Gate Array)
  - Komercialni procesorji so sicer implementirani v tehnologiji ASIC (Application-Specific Integrated Circuits), ki pa ni lahko dostopna
- FPGA sestavlja polje celic CLB (Configurable logic block) s povezavami
  - oboje uporabnik lahko določi ('programira')

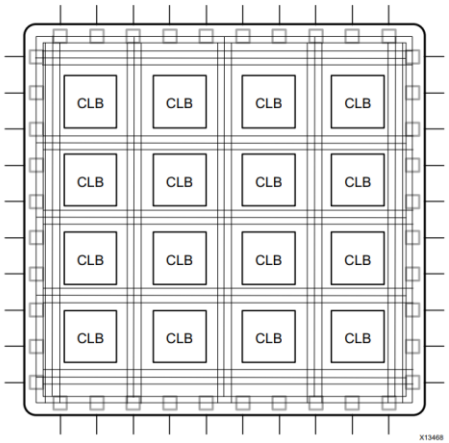
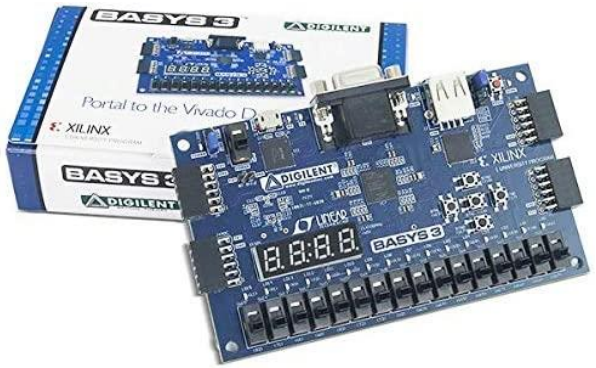


Figure 2-1: **Basic FPGA Architecture**

Primer preproste demo ploščice:



CLB vsebuje vpogledno tabelo (LUT) in D flip-flop

- LUT lahko implementira poljubno logično funkcijo 4-6 spremenljivk

Vse slike so iz: Introduction to FPGA Design with VivadoHigh-Level Synthesis. UG998 (v1.1) January 22, 2019, Xilinx.

The hardware implementation of a LUT can be thought of as a collection of memory cells connected to a set of multiplexers. The inputs to the LUT act as selector bits on the multiplexer to select the result at a given point in time. It is important to keep this representation in mind, because a LUT can be used as both a function compute engine and a data storage element. Figure 2-3 shows this functional representation of the LUT.

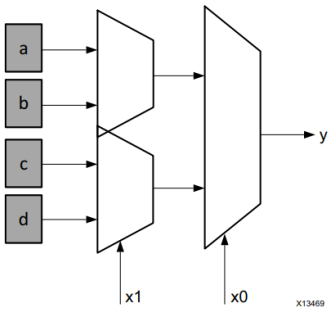


Figure 2-3: **Functional Representation of a LUT as Collection of Memory Cells**

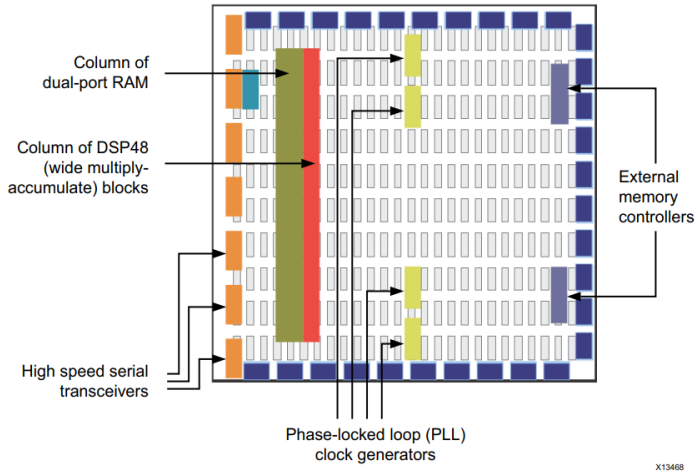


Figure 2-2: **Contemporary FPGA Architecture**

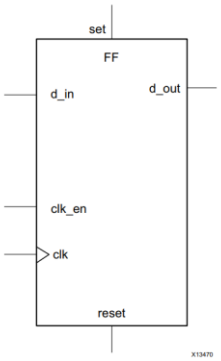


Figure 2-4: **Structure of a Flip-Flop**

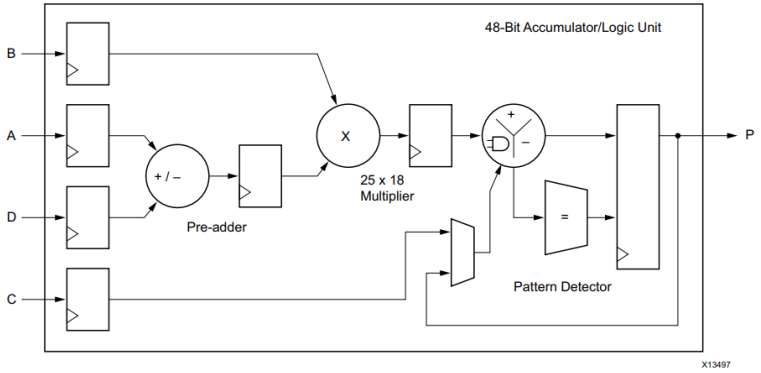
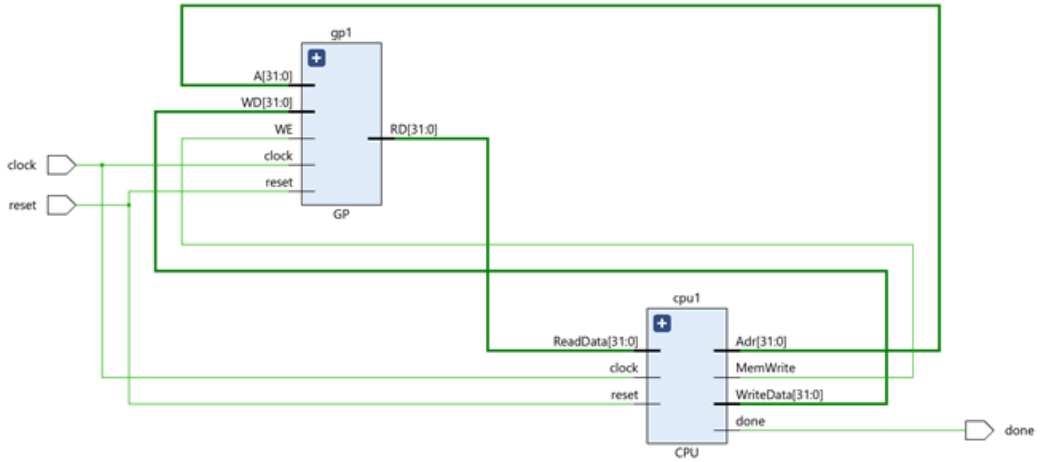


Figure 2-5: **Structure of a DSP Block**

Primer: Večcikelna CPE (poenostavljen RISC-V iz poglavja 6-CPE)

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

entity Computer is
    Port (  reset: in std_logic;
            clock: in std_logic;
            done: out std_logic);
end Computer;

architecture Behavioral of Computer is
component CPU is
  port (reset: in std_logic;
        clock: in std_logic;
        MemWrite: inout std_logic;
        Adr: out std_logic_vector(31 downto 0);
        ReadData: in std_logic_vector(31 downto 0);
        WriteData: out std_logic_vector(31 downto 0);
        done: out std_logic
        );
end component;
component GP is
  port (reset: in std_logic;
        clock: in std_logic;
        WE: in std_logic;
        A : in std_logic_vector(31 downto 0);
        RD : out std_logic_vector(31 downto 0);
        WD : in std_logic_vector(31 downto 0)
        );
end component;
signal MemWrite: std_logic := '0';
signal A : std_logic_vector(31 downto 0) := (others => '0');
signal ReadData : std_logic_vector(31 downto 0) := (others => '0');
signal WriteData : std_logic_vector(31 downto 0) := (others => '0');
begin
cpu1: CPU port map( reset => reset, clock => clock, MemWrite => MemWrite,
        Adr => A, ReadData => ReadData, WriteData => WriteData, done => done);
gp1: GP port map( reset => reset, clock => clock, WE => MemWrite,
        A => A, RD => ReadData, WD => WriteData);
end Behavioral;
```



CPE (1. del)

```
entity CPU is
  port (
        reset: in std_logic;
        clock: in std_logic;
        MemWrite: out std_logic;
        Adr: out std_logic_vector(31 downto 0);
        ReadData: in std_logic_vector(31 downto 0);
        WriteData: out std_logic_vector(31 downto 0);
        done: out std_logic
        );
end entity CPU;

architecture top of CPU is
component KE is...
component ALU is...

type RF_array is array (0 to 31) of std_logic_vector(31 downto 0);
    signal RF: RF_array := (others => (others => '0'));

    signal op: std_logic_vector(6 downto 0) := (others => '0');
    signal funct3: std_logic_vector(2 downto 0) := (others => '0');
    signal funct7_5: std_logic := '0';
    signal Zero: std_logic := '0';
    signal PCWrite: std_logic := '0';
    signal IRWrite: std_logic := '0';
    signal AdrSrc: std_logic := '0';
    signal ResultSrc: std_logic_vector(1 downto 0) := (others => '0');
    signal ALUControl: std_logic_vector(2 downto 0) := (others => '0');
    signal ALUSrcB:  std_logic_vector(1 downto 0) := (others => '0');
    signal ALUSrcA: std_logic_vector(1 downto 0) := (others => '0');
    signal ImmSrc: std_logic_vector(1 downto 0) := (others => '0');
    signal RegWrite: std_logic := '0';

    signal PC, OldPC: std_logic_vector(31 downto 0) := (others => '0');
    signal IR: std_logic_vector(31 downto 0) := (others => '0');
    signal srcA, srcB: std_logic_vector(31 downto 0) := (others => '0');
```

```
begin

muxAdr: Adr <= PC when AdrSrc = '0' else Result;

procPC: process (clock, reset) is
    begin
        if reset = '1' then
            PC <= (others => '0');
        elsif rising_edge(clock) then
            if PCWrite = '1' then
                PC <= Result;
            end if;
        end if;
    end process;

procIR: process (clock, reset) is
    begin
        if reset = '1' then
            IR <= (others => '0');
            OldPC <= (others => '0');
        elsif rising_edge(clock) then
            if IRWrite = '1' then
                IR <= ReadData;
                OldPC <= PC;
            end if;
        end if;
    end process;

procData: process (clock, reset) is
    begin
        if reset = '1' then
            Data <= (others => '0');
        elsif rising_edge(clock) then
            Data <= ReadData;
        end if;
    end process;

    op <= IR(6 downto 0);
```

CPE (2. del)

```
    op <= IR(6 downto 0);
    funct3 <= IR(14 downto 12);
    funct7_5 <= IR(30);

    Rs1 <= IR(19 downto 15);
    Rs2 <= IR(24 downto 20);
    Rd <= IR(11 downto 7);

procKE: KE port map( reset => reset, clock => clock, op => op, funct3 => funct3,
          funct7_5 => funct7_5, Zero => Zero, PCWrite => PCWrite, MemWrite => MemWrite,
          IRWrite => IRWrite, AdrSrc => AdrSrc, ResultSrc => ResultSrc,
          ALUControl => ALUControl, ALUSrcB => ALUSrcB, ALUSrcA => ALUSrcA,
          ImmSrc => ImmSrc, RegWrite => RegWrite, done => done);

procRFWrite: process( reset, clock)
    begin
        if reset = '1' then
            RF <= (others => (others => '0'));
        elsif rising_edge(clock) then
            if RegWrite = '1' then
                RF(to_integer(unsigned(Rd))) <= Result; --WD3
            end if;
        end if;
    end process;

procRFRead: process( Rs1, Rs2)
    begin
        RD1 <= RF(to_integer(unsigned(Rs1)));
        RD2 <= RF(to_integer(unsigned(Rs2)));
    end process;

    Imm12 <= IR(31 downto 20) when ImmSrc = "00" --lw
        else (IR(31 downto 25) & IR(11 downto 7)) when ImmSrc = "01" --sw
        else (IR(31) & IR(7) & IR(30 downto 25) & IR(11 downto 8)) when ImmSrc = "10" --beq
        else (others => '0');
    ImmExt <= (std_logic_vector(to_unsigned(0,20)) & Imm12) when Imm12(11) = '0'
         else (std_logic_vector(to_unsigned(1,20)) & Imm12)  when Imm12(11) = '1'
         else std_logic_vector(to_unsigned(0,32));
```

```
procRegAB: process( reset, clock)
    begin
        if reset = '1' then
            regA <= (others => '0');
            regB <= (others => '0');
        elsif rising_edge(clock) then
            regA <= RD1;
            regB <= RD2;
        end if;
    end process;
    WriteData <= regB;

    srcA <= PC when ALUSrcA = "00" else OldPC when ALUSrcA = "01"
        else regA when ALUSrcA = "10";
    srcB <= regB when ALUSrcB = "00" else ImmExt when ALUSrcB = "01"
        else std_logic_vector(to_unsigned(4,32)) when ALUSrcB = "10";

procALU: ALU port map( ALUControl => ALUControl, srcA => srcA, srcB => srcB,
            ALUResult => ALUResult, Zero => Zero);

procALUOut: process (clock, reset) is
    begin
        if reset = '1' then
            ALUOut <= (others => '0');
        elsif rising_edge(clock) then
            ALUOut <= ALUResult;
        end if;
    end process;

    Result <= ALUOut when ResultSrc = "00"
        else Data when ResultSrc = "01"
        else ALUResult when ResultSrc = "10";
end architecture top;
```

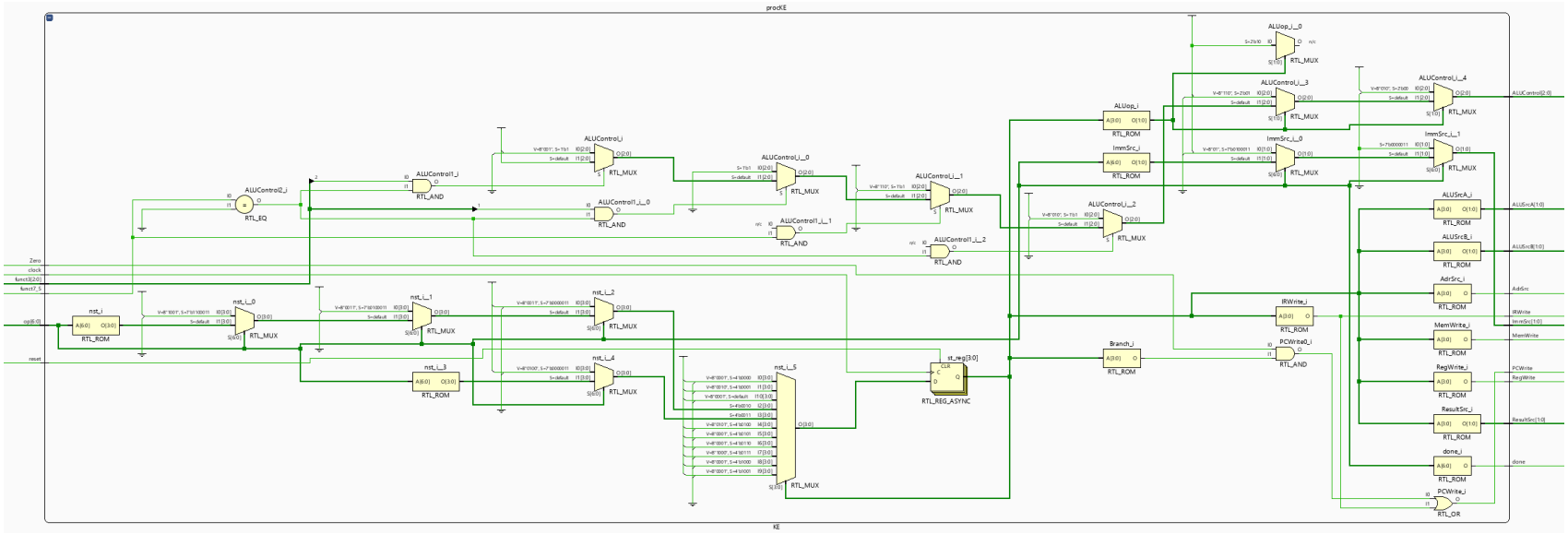
Kontrolna enota (KE) je podvezje CPE

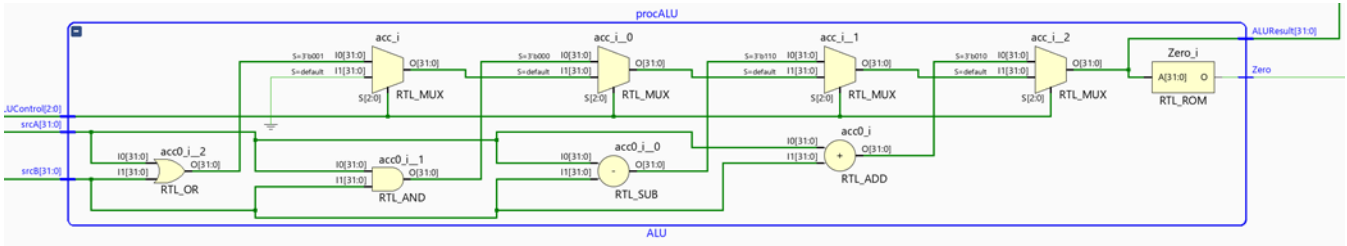
- njen glavni del je končni avtomat (FSM), ki ustreza diagramu prehajanja stanj v pog. 6

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

entity KE is
  port (
          reset: in std_logic;
          clock: in std_logic;
          op: in std_logic_vector(6 downto 0);
          funct3: in std_logic_vector(2 downto 0);
          funct7_5: in std_logic;
          Zero: in std_logic;
          PCWrite: out std_logic;
          MemWrite: out std_logic;
          IRWrite: out std_logic;
          AdrSrc: out std_logic;
          ResultSrc: out std_logic_vector(1 downto 0);
          ALUControl: out std_logic_vector(2 downto 0);
          ALUSrcB: out std_logic_vector(1 downto 0);
          ALUSrcA: out std_logic_vector(1 downto 0);
          ImmSrc: out std_logic_vector(1 downto 0);
          RegWrite: out std_logic;
          done: out std_logic
        );
end KE;

architecture rtl of KE is
    type state_type is (Sr, S0, S1, S2, S3, S4, S5, S6, S7, S10, Serr);
    signal st, nst: state_type;
    signal ALUop: std_logic_vector(1 downto 0) := (others => '0');
    signal Branch: std_logic := '0';
    signal PCUpdate: std_logic := '0';
begin
    PCWrite <= (Zero and Branch) or PCUpdate;

 InstrDecoder:
    ImmSrc <= "00" when op = "0000011"
         else "01" when op = "0100011"
         else "10" when op = "1100011"
         else "11";

 ALUDecoder:
```

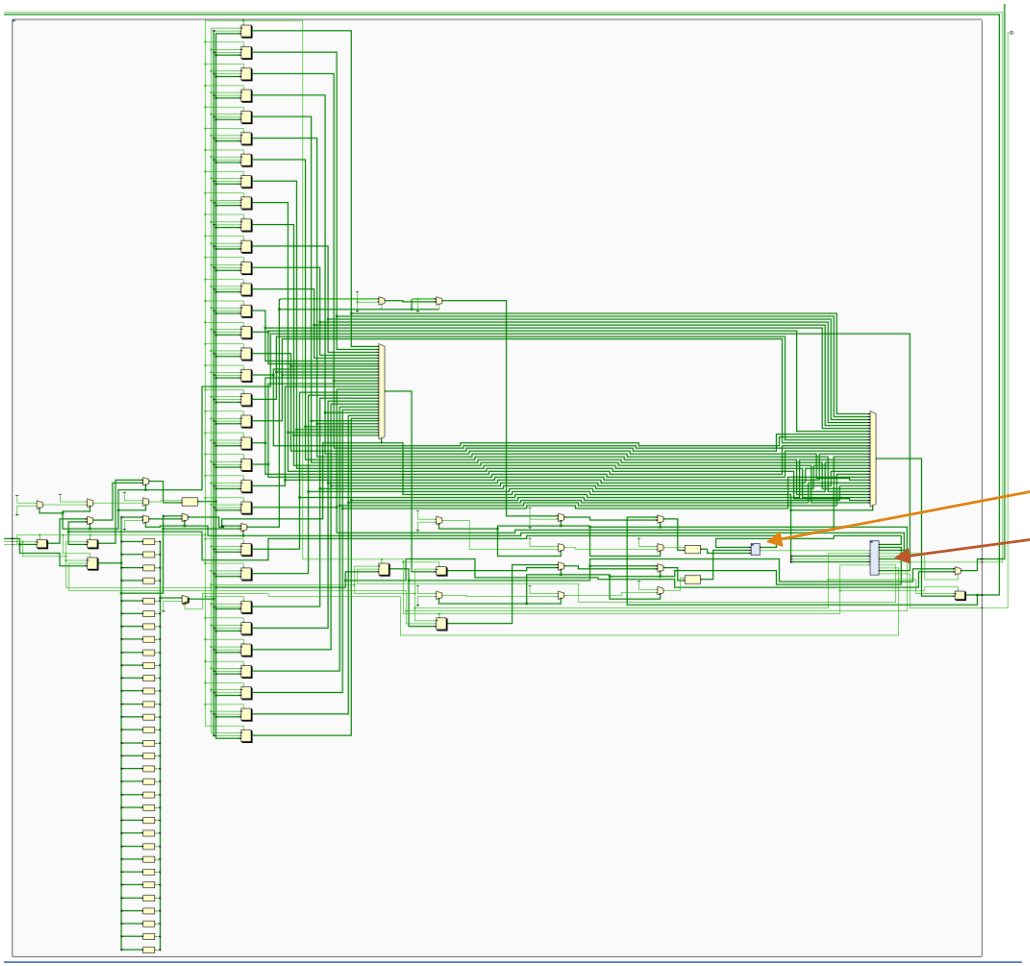
```
when S2 =>  -- MemAdr
    ALUSrcA <= "10";
    ALUSrcB <= "01";
    ALUop <= "00";
    if op = "0000011" then -- lw
        nst <= S3;
    elsif op = "0100011" then -- sw
        nst <= S5;
    else
        nst <= Serr;
    end if;

when S3 =>      -- MemRead
    ResultSrc <= "00";
    AdrSrc <= '1';
    nst <= S4;

when S4 =>      -- MemWB
    ResultSrc <= "01";
    RegWrite <= '1';
    nst <= S0;

when S5 =>      -- MemWrite
    ResultSrc <= "00";
    AdrSrc <= '1';
    MemWrite <= '1';
    nst <= S0;

when S6 =>      -- Execute R
    ALUSrcA <= "10";
    ALUSrcB <= "00";
    ALUop <= "10";
    nst <= S7;

when S7 =>      -- ALUWB
    ResultSrc <= "00";
    RegWrite <= '1';
    nst <= S0;
```

Shema KE in ALE:

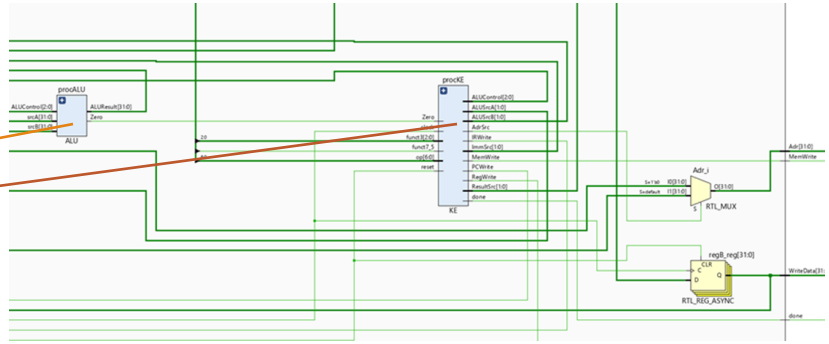




Shema CPE:



Del CPE (KE in ALE):



- HDL-jeziki (Hardware Description Language – jezik za opis hardvera):
  - VHDL
  - Verilog
- 2 načina modeliranja v HDL-jeziku:
  - RTL: register transfer level
    - tudi v našem primeru
  - Gate-level
- Postopek implementacije vezja v FPGA:
  - Sinteza
    - Program za sintezo iz opisa v HDL-jeziku sklepa, kakšno vezje želimo implementirati
  - Mapiranje
    - Preslikava elementov vezja v elemente v FPGA (CLB, LUT, FF, BRAM, itd.)
  - Razmestitev in povezovanje (placement, routing)
    - vpliva na zakasnitve in posledično največjo možno frekvenco delovanja
  - Datoteka ‘bitstream’ se naloži v FPGA
    - celice, ki se jih programira, so tipa SRAM (določajo vsebine tabel LUT, konfiguracijo elementov in povezave v FPGA)

Testiranje (Simulacija)

- funkcionalno (v tej fazi se ne upošteva dejanskih zakasnitev)
- časovno (upošteva izračunane zakasnitve)

Testna VHDL datoteka (Testbench file) podaja potek vhodnih signalov:

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

entity testComputer is
end entity testComputer;

architecture tb of testComputer is
component Computer is
    Port ( reset: in std_logic;
           clock: in std_logic;
           done: out std_logic);
end component Computer;
constant clk_period: time := 10 ns;
signal reset: std_logic := '0';
signal clock: std_logic := '0';
signal done: std_logic := '0';

begin
comp1: Computer port map (reset => reset, clock => clock, done => done);

clk_process: process
    begin
        clock <= not clock;
        wait for clk_period/2;
    end process;

stim_process: process
    begin
        reset <= '1';
        wait for 1.5*clk_period;
        reset <= '0';
        wait for 25*clk_period;
        wait until done = '0';
        report "Test Done."  severity failure;
        assert (FALSE) report "End of simulation." severity Failure;
    end process;
end architecture tb;
```

Rezultat simulacije, prikazan grafično: