Laboratorij za načrtovanje integriranih vezij

Fakulteta za elektrotehniko
Univerza v Ljubljani

Andrej Trost

Načrtovanje digitalnih el. sistemov

Implementacija in verifikacija vezij

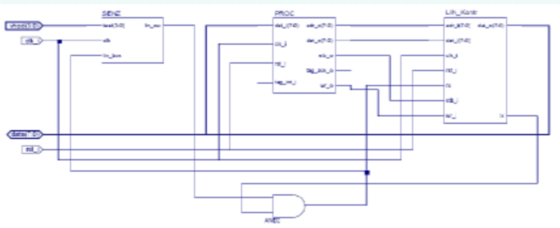
http://lniv.fe.uni-lj.si/ndes.html

Potek načrtovanja sistemov

- **specifikacija** zgradba in vmesniki
  - izdelava VHDL modelov
    - preverjanja sintakse in simulacija obnašanja vezja
  - sinteza vezja
    - funkcijska simulacija
  - tehnološka implementacija

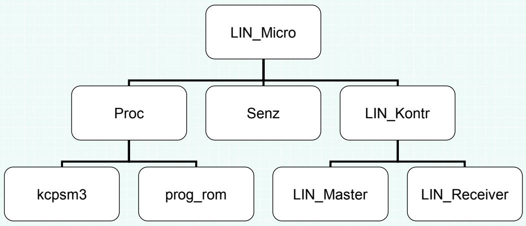
dokumentacija

Določanje zgradbe sistema



- določamo strukturo, vmesnike in relacije med posameznimi bloki

Hierarhična zgradba sistema



- hirarhično načrtovanje omogoča ločeno izvedbo in testiranje manjših blokov
- omogoča uporabo pripravljenih komponent IP

Določanje velikosti blokov

- pri shematskem načrtovanju so končni bloki zelo majhni > register, števec, primerjalnik
  - sistem je zelo razdrobljen in ga je težko popravljati
- v visokonivojskem jeziku so bloki zaključeni moduli > sprejemnik, FIFO, kotrolna enota
  - koda naj ne obsega več kot nekaj strani
  - posamezen blok naj bo primeren za testiranje
  - namesto obsežnih avtomatov raje uporabimo vgrajen mikrosekvenčnik ali mikroprocesor

Uporaba vgrajenih procesorjev

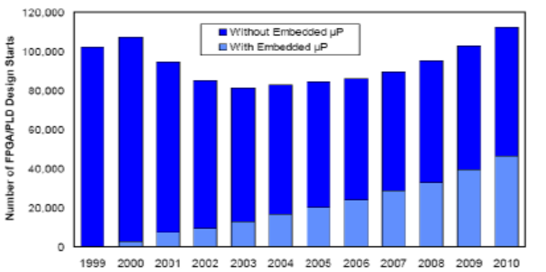


Figure 1 – FPGA Design Starts With Embedded µP
Source: Gartner, August 9, 2005

Prednosti vgrajenih procesorjev

- procesor uporabimo za krmiljenje, obdelavo podatkov pa naredimo z logiko
- spremembo funkcionalnosti naredimo le s spremembo algoritma
- pri vgrajenih procesorjih ni težav z dobavljivostjo oz. zastaranjem komponent
  - 8-bitni procesorji zasedejo zelo malo prostora (Picoblaze: 5% XC3S200 ali 0,5% XC3S5000)

Izdelava modelov vezij

- določitev funkcionalnosti
  - algoritmični diagram poteka
  - specifikacija dogodkov (registri in vodila)
- kodiranje v strojno-opisnem jeziku VHDL ali Verilog
  - algoritemski zapis v obliki primerni za RTL sintezo
  - enote opišemo s procesi in jih povežemo z vmesnimi signali ali stanji
    - sinhrono vezje bi lahko opisali le z enim procesom, vendar bi bil opis precej nepregleden

Priporočila za kodiranje

- VHDL koda je del dokumentacije!
- Upoštevaj priporočila sinhronega načrtovanja!
- razmisli o skupni rabi gradnikov (npr. števcev)
- izogibaj se globokemu gnezdenju if-elsif-else stavkov
- uporabi sklepne modele za večjo prenosljivost (npr. ROM) ter knjižnične instance za posebne gradnike (npr. Block RAM)

Uporaba komentarjev

- kratek opis posameznega modula
- kratek opis posameznih blokov (process)
  - npr. namen procesa, vhodi in izhodi
- opis signalov ob deklaraciji ali na instanci
- pojasnilo vejitev in konstant

```
-- delilnik za generiranje vzorcevalnih impulzov
-- input: clk = 100kHz
-- output: ce = 12.5kHz (LIN_Master), ce4 = 50kHz (LIN_Receiver)
P_timing: process(clk)
begin
  if rising_edge(clk) then
    if n=7 then            -- delitev vhodne ure z 8
```

Poimenovanje signalov in konstant

- sestavljena imena signalov opisujejo njihov namen, npr:
  - bit_cnt: števec bitov (cnt)
  - a_reg: register (reg)
  - wr_i: vhodni signal wr (i)
  - dat_o: izhodni signal dat (o)
- konstante pišemo z velikimi črkami, npr:
  - HRES, VRES

Načrtovanje sinhronih vezij

```
if st = SIdle or st = SActive then
  i <= 0;
elsif rising_edge(ce4) then
  i <= i + 1;
...
```

```
if rising_edge(clk) then
  if ce4='1' then
    if st=Ridle or st=Ractive then
      i <= 0;
    else
      i <= i + 1;
...
```

- če je mogoče, delamo sinhrono z uro
- izogibajmo se asinhr. resetu, ki ga krmili logika
  - izhod iz primerjalnika ali logike ima lahko špice ob spremembah stanja

Števci znotraj avtomata Lin_Master

```
signal dataBitNumber: integer range 0 to 8;
signal dataSyncNumber: integer range 0 to 14;
...
  when Ssyncbreak =>
    if ce = '1' then
          dataSyncNumber <= dataSyncNumber +1;
...
  when Sdata =>
    if ce = '1' then
          tx <= shift_r2(dataBitNumber);
          dataBitNumber <= dataBitNumber + 1;
```

- štejemo dolžino Sync impulza in število bitov
  - števca ne potrebujemo istočasno, zato lahko uporabimo le en števec!

Vgnezdene strukture v VGAsync

```
 if horizontal > 0 and horizontal < 800 then
   hsync <='0';
   blank <='0';
elsif horizontal >= 800 and horizontal < 856 then
   hsync <= '0';
   blank <='1';
elsif horizontal >= 856 and horizontal < 976 then
   hsync <= '1';
   blank <='1';
elsif horizontal >= 977 and horizontal < 1039 then
   hsync <= '0';
   blank <='1';
   ...
```

Rešitev VGAsync z manj gnezdenja

```
if hcount=HRES+HFP then
   hsync <= '1';
elsif hcount=HRES+HFP+HSY then
   hsync <= '0';
end if;

if hcount>HRES then
   blank <= '1';
else
   blank <= '0';
end if;
   ...
```

- pogojne stavke razdelimo
- uporabimo operator = namesto < ali >

Sklepni modeli (inference)

- program za sintezo sklepa o tem, kateri gradnik želimo uporabiti:
  - števec je vezje, ki ob uri in določenih pogojih prišteva oz. odšteva
  - ROM opišemo z zbirko konstant, ki jih beremo z naslavljanjem (indeksi)

```
type rom is array (0 to 511) of std_logic_vector(7 downto 0);
constant r : rom := ( "00111100", "01100110", ... );
...
rom_data <= r( conv_integer(adr) );
```

Knjižnične instance

- vezja FPGA vsebujejo vgrajene Block RAM enote, ki jih vključimo kot instance

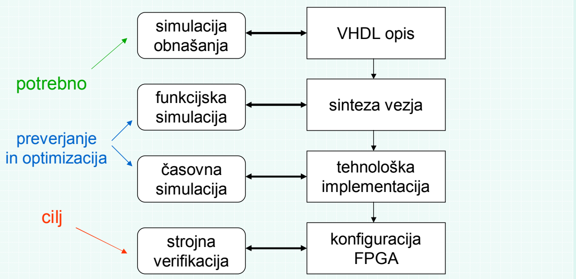
```
library UNISIM;     -- library used when instantiating Xilinx primitives
use UNISIM.VComponents.all;
```

```
VRAM : RAMB16_S4_S4
  generic map (
    INIT_A => X"0",   -- Value of output registers on Port A at startup
    ... )
  port map (
    DOA => open,     -- Port A 4-bit Data Output
    DOB => DataB,    -- Port B 4-bit Data Output
    ADDRA => adra,   -- Port A 12-bit Address Input
    ... )
```

Generator instanc IP

- Xilinx Core Generator ponuja bloke IP:
  - osnovni: registri, števci, pomnilniki, primerjalniki
  - komunikacijski: (de)kodirniki, korekcije napak, modulatorji
  - DSP: filtri, modulatorji, transformi, sintetizatorji
  - tehnološki gradniki: DCM, Rocket-IO
  - matematični: osnovne op., množenje in deljenje, koreni, trigonometrične funkcije, floating-point
  - pomnilni: CAM, FIFO, RAM, ROM

Simulacija vezij



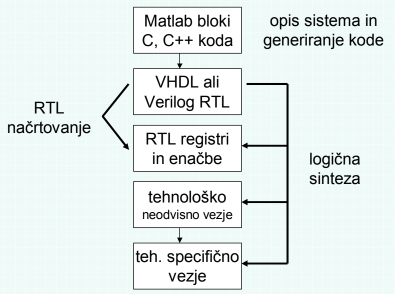
Priprava simulacije

- stimulatorji v obliki časovnega diagrama (waveform)
  - primerni so za preproste simulac. scenarije
  - vnesemo jih grafično ali v obliki makrojev
- izdelava testnih struktur (testbench)
  - omogočajo obsežne simulacije z različnimi scenariji
  - opisane so v istem jeziku kot testirano vezje, z vsemi konstrukti ki jih jezik omogoča
  - dobro napisane testne strukture so uporabne za vse vrste simulacij (funkcijsko, časovno)

Priprava načrta za sintezo

- sistem mora biti v celoti opisan
- uporabljati smemo le konstrukte jezika in način opisa, ki je primeren za sintezo
- izbrati moramo ciljno družino in vezje FPGA
- določimo optimizacije in časovne omejitve
  - frekvenco vhodne ure
  - zakasnitve na vhodih in izhodih

Sinteza



Matlab System Generator za DSP

- v okolju Matlab/Simulink naredimo model sistema iz vnaprej pripravljenih blokov
  - osnovni: števci, zakasnitve
  - pretvorba podatkov (fiksna vejica, rezanje)
  - DSP: Filter Design Tool, FFT, FIR
  - množilnik, akumulator, inverter
  - opis krmilne logike (MCode)
  - pomnilniki
- System Generator naredi tehnološke gradnike VHDL kodo za sintezo v Xilinx ISE okolju

Priprava za implementacijo

- avtomatizirana izvedba (tool driven)
- nastavljamo uporabniške zahteve in parametre (design constraints)
  - frekvenca ure
  - lokacija zunanjih priključkov
  - stopnja optimizacije
- floorplaning: določanje postavitve blokov
  - določimo območje v matriki za posamezen blok
  - izberemo vgrajene komponente za dol. bloke