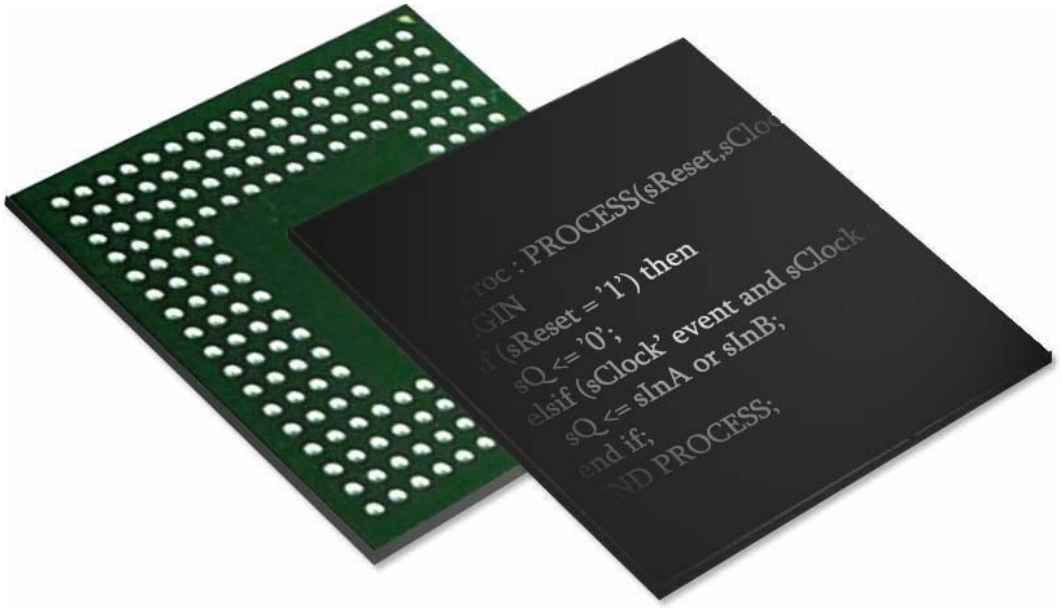
HDL alapú tervezés



2012.04.21.

Milotai Zsolt

HDL alapú tervezés

Cél

- Megismerkedni a harverleíró nyelvek világával és a hardverleírás alapjaival

Tartalom

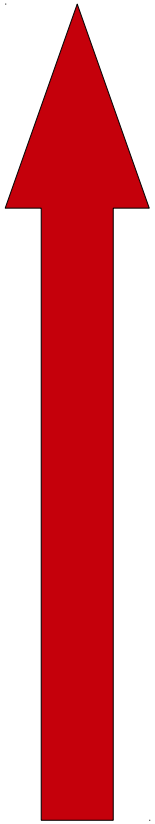
- Absztrakciós szintek és tervezési stratégiák
- A fontosabb hardverleíró nyelvek áttekintése
- A hardverleírás alapjai

HDL alapú tervezés

I.
A tervezés absztrakciós szintjei, stratégiák

Fizikai tervezés

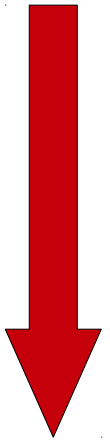
- Rétegszint
  - Inkább csak elvi lehetőség
- Tranzisztorok szintje
  - példa: Parallax Propeller, 8 év
- Kapuszint
- RTL (Register-Transfer Level)
- Viselkedési szint



- Bonyolultság
- Tervezési idő
- Lehetőség az optimalizálásra

Nyelvi leírási szintek

- Dataflow (adatfolyam)
- Structural (szerkezeti, strukturális)
- Behavioral (viselkedési, algoritmikus)



Absztrakció nő

Tervezési stratégiák

- Top-down
  - „felülről lefelé építkezés” vagy „lépésenkénti finomítás” elve
  - Előnye: felesleges részletek elrejtése
- Bottom-up
  - „lentről felfelé” vagy „téglánkénti építkezés” elve

Tervezési stratégiák

- Kevert
  - A 2 módszer együttes, értelemszerű használata
  - *„Nincs pótszere az intelligenciának, a tapasztalatnak, a józan észnek és a jó ízlésnek.”* - Bjarne Stroustrup

HDL alapú tervezés

II.
A hardverleíró nyelvek áttekintése

HDL áttekintés

- ABEL (Advenced Boolean Expression Language)
- VHDL
- Verilog HDL, SystemVerilog
- SystemC
- JHDL
- MyHDL
- Handel-C
- stb.

HDL bevezető

- HDL: Hardware Description Language
- Tervezési / megvalósítási lehetőségek:
  - sematikus ábra
  - HDL
  - szekvencia diagram
  - stb.
- HDL előnye:
  - Univerzalitás: nem okoz gondot bonyolult és / vagy nagy kiterjedésű rendszerek leírása (gondoljunk pl. egy 4K mélységű RAM-ra)

VHDL

VHDL ≠ **V**ery **H**ard and **D**ifficult **L**anguage

☺

VHDL

- VHDL: VHSIC HDL, azaz Very High Speed Integrated Circuit Hardware Description Language
- 1981: Amerikai Védelmi Minisztérium (US DoD) kezdeményezése
- Céljai:
  - Rugalmas leírást nyújtson
  - Minden szimulátorral ugyanazt az eredményt adja
  - Technológia függetlenség

VHDL

- 1983-85: fejlesztés: Intermetrics, IBM és TI
- 1987: első szabvány (IEEE Std. 1076-1987)
- Módosítások:
  - IEEE Std. 1076-1993
  - IEEE Std. 1076-2000
  - IEEE Std. 1076-2002
  - IEEE Std. 1076-2008
- A szabvány fenntartója a non-profit Accellera (Open Verilog International (OVI) + VHDL International)

VHDL

- 2007: VHPI (VHDL Procedural Interface): ezen interfészen keresztül pl. C-ben írt programmal hozzáférhetünk a VHDL modellhez
  - Alkalmazás: például processzor utasítás-készletének szimulációja
- Kiterjesztések:
  - VHDL-200X: HDL & HVL (Hardware Verification Language)
  - VHDL-AMS: Analog & Mixed-Signal

Verilog HDL

- Phil Moorby, Prabhu Goel, 1983/1984, Automated Integrated Design Systems (Gateway)

Phil Moorby

Prabhu Goel

Verilog HDL

- 1989: a Cadence Design Systems felvásárolta a Gateway-t
- 1990: a Verilogot közkinccsé tették
- IEEE Std. 1364-1995; IEEE Std. 1364-2001
- Ma a szabvány fenntartója a non-profit Accellera (Open Verilog International (OVI) + VHDL International)
- Verilog kiterjesztések:
  - SystemVerilog (IEEE Std. 1800-2005): HDL & HVL
  - Verilog-AMS (draft): Analog & Mixed-Signal

SystemC

- 1999: az OSCI (Open SystemC Initiative) által került bejelentésre
- Fejlesztői: ARM Ltd., CoWare, Synopsys, CynApps
- 2005: IEEE Std. 1666-2005
- C++ szintaxis
- Kiterjesztései:
  - SystemC AMS

JHDL

- JHDL: Java HDL, később Just-Another HDL
- Brigham Young University (BYU) Configurable Computing Laboratory, 1997
- Elsősorban OOP megközelítés
- Valójában egy eszköztár és class library Java fölött

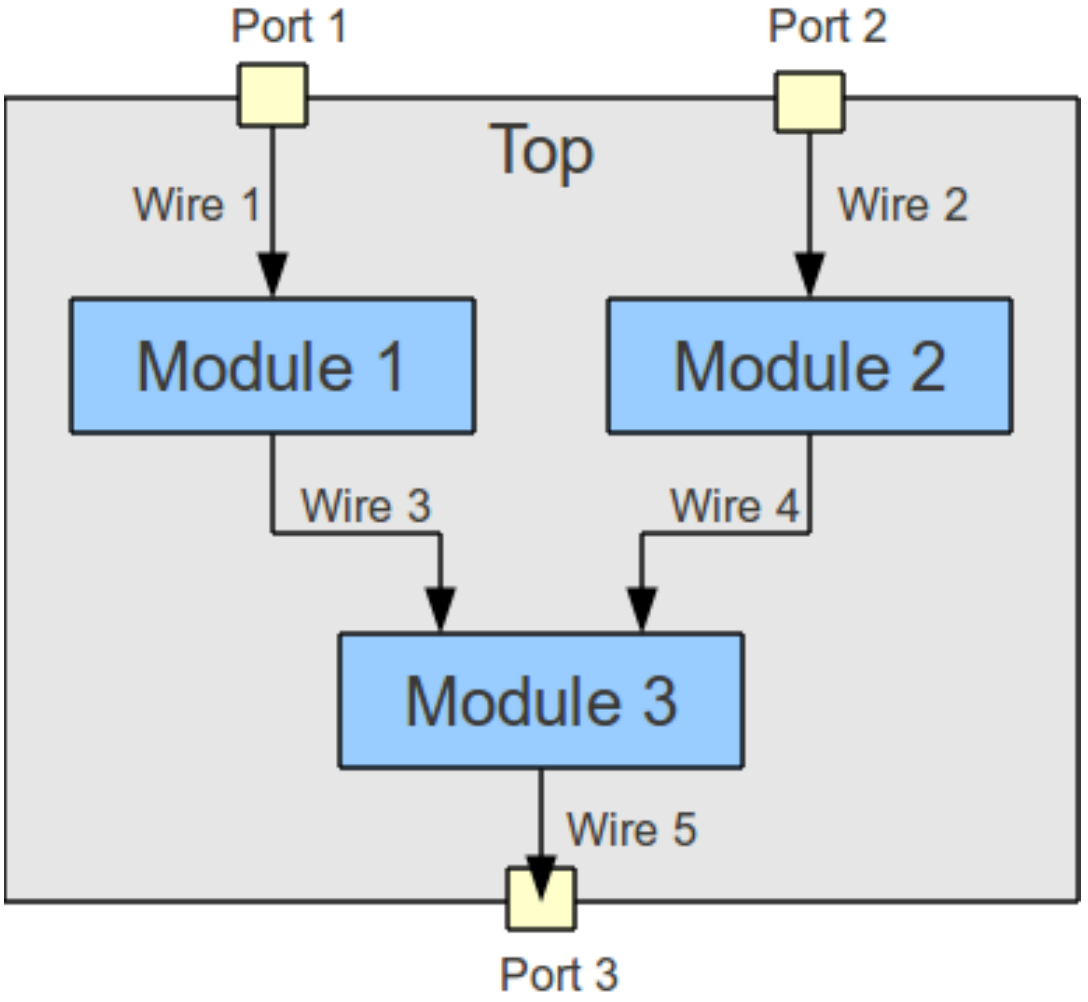
III.
VHDL alapok

Lexikai elemek

- Megjegyzések: -- (2 db egymást követő kötőjel)
  - pl.: `-- comment`
- Azonosítók:
  - A fejlesztő által definiált nevek
  - Szintaxis: betű { [aláhúzás] betű_vagy_szám }
  - példa: `sig; Sig; SIG; S_sig34`
- Karakterek: `''`
  - példa: `'c'; 'C'; '5'`
- Stringek: `" "`
  - példa: `"string"`

A hardverleírás megközelítése

- Design részei:
  - Modulok
  - Vezetékek
  - Portok (speciális vezetékek)



Vezetékek / Jelek

- IEEE Std. 1164: logikai érték reprezentálására alkották. A következő 9 értéket tartalmazza:
  - 'U' – uninitialized
  - 'X' – strong unknown
  - '0' – strong 0
  - '1' – strong 1
  - 'Z' – high impedance
  - 'W' – weak unknown
  - 'L' – weak 0 (wired-OR)
  - 'H' – weak 1 (wired-AND)
  - '-' - don't care

Vezetékek / Jelek

- Az IEEE Std. 1164 definícióinak használatához a kódban szükséges az alábbi „include”:

```
library IEEE;
use IEEE.std_logic_1164.all;
```

Vezetékek / Jelek

- A vezeték / jel neve a VHDL-ben: **signal**
- Akár vezetékről, akár buszról van szó, **signal** objektumról beszélünk, csak típusban különböznek
- Feladatuk: a modulok közötti információ-továbbítás biztosítása (az egységek összekötése)
- Egy signal alaptípusai (vezeték esetén):
  - std_logic: IEEE Std. 1164 szerinti 9 érték bármelyikét felveheti
  - bit: csak '0' és '1' értéket vehet fel

Vezetékek / Jelek

- Egy signal alaptípusai (busz esetén):
  - std_logic_vector: IEEE Std. 1164 szerinti 9 érték bármelyikét felvehetik az egyes vezetékek
  - bit_vector: csak '0' és '1' értékek valamelyikét vehetik fel a busz elemei
- Jel deklarációja:
  - **signal** azonosító : típus [:= kifejezés];

Vezetékek / Jelek

- Példa jelek deklarálására:
  - `signal a : std_logic;`
  - `signal b : bit := '0';`
  - `signal c : bit_vector(7 downto 0);`
  - `signal d : std_logic_vector(31 downto 0) := X”0000FFFF”;`
  - `signal e : std_logic_vector(0 to 15);`
  - `signal f : bit_vector(15 downto 12) := B”0000”;`
- Példa jel értékadására:
  - `c <= X”5F”;`
- Megjegyzés: radix megadása:
  - B - bináris
  - O - oktális
  - X - hexadecimális

Fontosabb adattípusok I.

- Típusdefiniálás: **type** típus_neve **is** kifejezés;
- Skalár típusok (példákkal):
  - Egész típus:

```
type integer is range -(2**31) to 2**31 – 1;
```

  - Lebegőpontos típus:

```
type real is range -1E38 to 1E38;
```

  - Felsorolás típus:

```
type bit is ('0', '1');
type boolean is (false, true);
```

Fontosabb adattípusok II.

- Skalár típusok (példákkal):
  - Fizikai típus:

```
type time is range -(2**31 – 1) to (2**31 – 1)
   units
      fs;
      ps  = 1000 fs;
      ns  = 1000 ps;
      us  = 1000 ns;
      ms  = 1000 us;
      sec = 1000 ms;
      min = 60 sec;
      hr  = 60 min;
   end units;
```

Fontosabb adattípusok III.

- Összetett típusok (példákkal):
  - Tömb típus:

```
type word is array (31 downto 0) of bit;
```

  - Rekord típus:

```
type bus_type is record
   addr  : std_logic_vector(31 downto 0);
   data  : std_logic_vector(63 downto 0);
   rd_wr : std_logic;
   frame : std_logic;
   ack   : std_logic;
end record;
```

Altípusok (subtypes)

- Célja: egy típus értékkészletének leszűkítése
- Szintaxis: **subtype** azonosító **is** altípus_megjelölés;
- Példák:

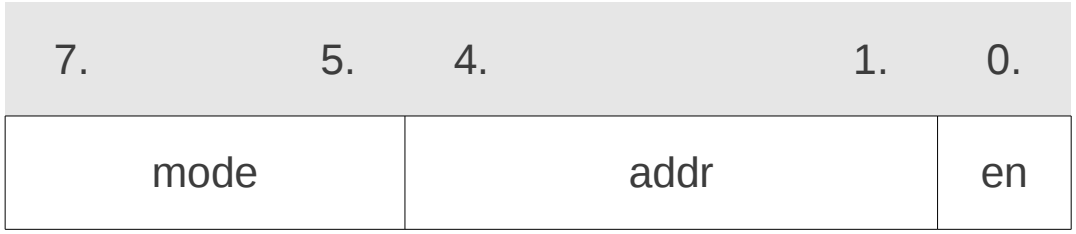
```
subtype natural is integer range 0 to integer'high;
subtype positive is integer range 1 to integer'high;
```

- Példa: signal 16 bites véletlenszám (rnd) tárolására:

```
-- definiálunk egy rnd_type nevű altípust
subtype rnd_type is std_logic_vector(15 downto 0);
-- deklarálunk egy rnd nevű, rnd_type típusú signalt
signal rnd : rnd_type;
```

Aliasok

- Aliasok használatával egy objektumnak vagy az objektum egy részének alternatív nevet adhatunk
- Szintaxis: **alias** azonosító : típus_megjelölés **is** objektum_név;
- Példa: egy regiszter bitjeinek elnevezése:




```
-- deklaráljuk a 8 bites regisztert
signal register : std_logic_vector(7 downto 0);
-- definiáljuk az aliasokat
alias mode : std_logic_vector(2 downto 0) is register(7 downto 5);
alias addr : std_logic_vector(3 downto 0) is register(4 downto 1);
alias en   : std_logic is register(0);
```

Konstansok

- Alkalmazása: olyan objektumok esetén, melyek értéke a kezdeti értékadás után nem módosítható
- Szintaxis: **constant** azonosító_lista : típus_megjelölés [:= kifejezés];
- Példák:

```
constant c : std_logic_vector(7 downto 0) := X"3F";

constant timescale : time := 10 ns;
```

Konstansok

- Példa: LUT létrehozása:

```
-- a LUT paraméterei
constant data_width : integer := 12;
constant element_num : integer := 15;

-- a LUT típusának definiálása
subtype value_type is std_logic_vector(data_width - 1 downto 0);
type lut_type is array(element_num - 1 downto 0) of value_type;

-- a konstans LUT létrehozása és feltöltése
constant lut : lut_type :=
    (
     X"000", X"0cc", X"197", X"262", X"32b",
     X"3f2", X"4b7", X"578", X"636", X"6f0",
     X"7a6", X"857", X"902", X"9a8", X"a48"
    );
```

Változók

- Alkalmazása: olyan objektumok esetén, amelyek a kezdeti értékadás után megváltoztathatják értéküket
- Szintaxis: **variable** azonosító_lista : típus_megjelölés [:= kifejezés];
- Egy változó értékadása azonnal kiértékelődik
- Az értékadás operátora a ':=' szemben a jelek '<=' operátorával
- Példák:

```
variable v : std_logic_vector(7 downto 0);
variable cnt : integer;
variable i : integer range 0 to 10 := 0;
-- értékadás változónak
i := 5;
```

Attribútumok

- Az attribútumok egy nyelvi eszközről kiegészítő információt adnak
- Használatuk szintaktikája: nyelvi_eszköz'kiegészítő_információ
- Léteznek előre definiált attribútumok, de a felhasználó is definiálhat sajátot
- Példák attribútumok használatára:
  - Típus és altípus attribútumok

    `T'high`: egy T skalár típus felső korlátja

    `T'low`: egy T skalár típus alsó korlátja
  - Tömb attribútumok

    `A'range(N)`: az A tömb N-edik elem típusának értékkészlete
  - Jel attribútumok

    `S'event`: az S jel értéke megváltozott

Attribútumok

- Példa élvezérlés megvalósítására:
  - Felfutó vagy lefutó él (esemény) „figyelése” egy signalon (clk)

```
if (clk'event) then
   ...
end if;
```

  - Felfutó él „figyelése” egy signalon (clk)

```
if (clk'event and clk = '1') then
   ...
end if;
```

  - Lefutó él „figyelése” egy signalon (clk)

```
if (clk'event and clk = '0') then
   ...
end if;
```

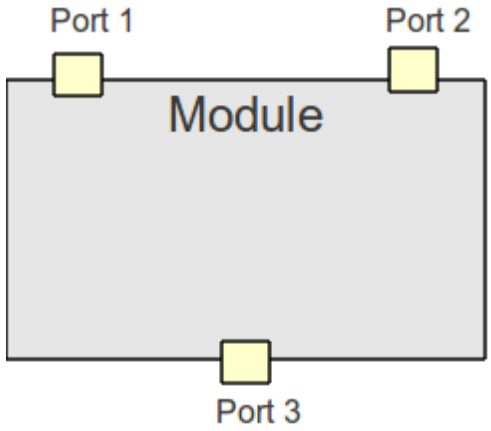
Operátorok

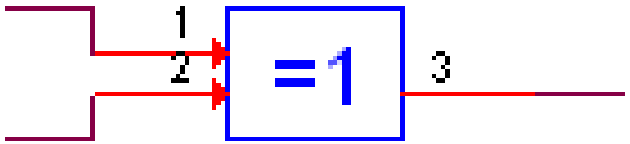
Aritmetikai (kétoperandusú)	**+** **-** ***** **/** **mod** **rem** ******	összeadás kivonás szorzás osztás modulus maradékképzés hatványozás
Aritmetikai (egyoperandusú)	**+** **-** **abs**	plusz előjel mínusz előjel abszolút érték
Relációs	**=** **/=** **<** **>** **<=** **>=**	egyenlő nem egyenlő kisebb nagyobb kisebb vagy egyenlő nagyobb vagy egyenlő
Logikai (kétoperandusú)	**and; or; nand; nor; xor; xnor**	
Logikai (egyoperandusú)	**not**	
Összefűző	**&**	összefűzés (concatenate)

Vissza a hardver tervezéséhez...

Mi az, amit elengedhetetlen jól specifikálni?

- A külvilággal való kapcsolatot, azaz az interfészt
- A tényleges működést





entitás

architektúra

Egy VHDL forráskód felépítése

```
library IEEE;
use IEEE.std_logic_1164.all;


entity module_1 is
        ...          -- az entitás leírása
end module_1;

architecture arch_name of module_1 is
        ...          -- az architektúra leírása
end arch_name;
```

Az entitás

- Írjuk le a modulunknak, mint entitásunknak az interfészét a **port lista** segítségével!

```
entity module_1 is
    port(
         A : in  std_logic;
         B : in  std_logic;
         C : out std_logic
        );
end module_1;
```

Az entitás

- Figyeljük meg a szintaxist!
  - Egy blokkot vagy listát pontosvessző (;) zár le
  - A port listában a pontosvessző (;) az egyes kivezetések felsorolásának elválasztásáért felelős, ezért nem szabad pontosvesszőt (;) írnunk a legutolsó kivezetés után
  - A port listát akár 1 sorban, felsorolás jelleggel is írhattunk volna, de így sokkal áttekinthetőbb

Az entitás

- Általánosabban is leírhatunk egy komponenst a **generic lista** segítségével, például:

```
entity ram is
   generic(
           width    : integer := 32;
           depth    : integer := 1024;
           addr_len : integer := 10
          );
   port(
        clk      : in  std_logic;
        rst      : in  std_logic;
        rd       : in  std_logic;
        wr       : in  std_logic;
        addr     : in  std_logic_vector(addr_len – 1 downto 0);
        data_in  : in  std_logic_vector(width - 1 downto 0);
        data_out : out std_logic_vector(width - 1 downto 0)
       );
end ram;
```

Az entitás

- A port listában lévő portok kizárólag signal-ok (ezért a **signal** kulcsszó elhagyható), és speciális módosítóval rendelkeznek:
  - in
  - out
  - inout
  - buffer (bufferelt kimenet, mely az entitás számára olvasható)
- A generic listában szereplő paraméterek kizárólag bemenő paraméterek és értékük konstans
- Így a **constant** kulcsszó és az **in** módosító elhagyható

Az architektúra felépítése

```
architecture arch_name of module_1 is

        ...          -- deklarációs rész

begin

        ...          -- definíciós rész

end arch_name;
```

- A deklarációs és definíciós részeket a **begin** kulcsszó választja el egymástól

Az architektúra felépítése

- A deklarációs részben bejelenthetőek:
  - signalok
  - konstansok
  - megosztott változók
  - komponensek
- A definíciós rész:
  - tartalmazza a tényleges működési leírást
  - konkurens formában

Példák deklarációra

- `signal s : std_logic_vector(15 downto 0);`

- `constant c : integer := 8;`
- `constant d : bit_vector(3 downto 0) := B"1011";`
- `constant e : bit_vector := B"1011";`

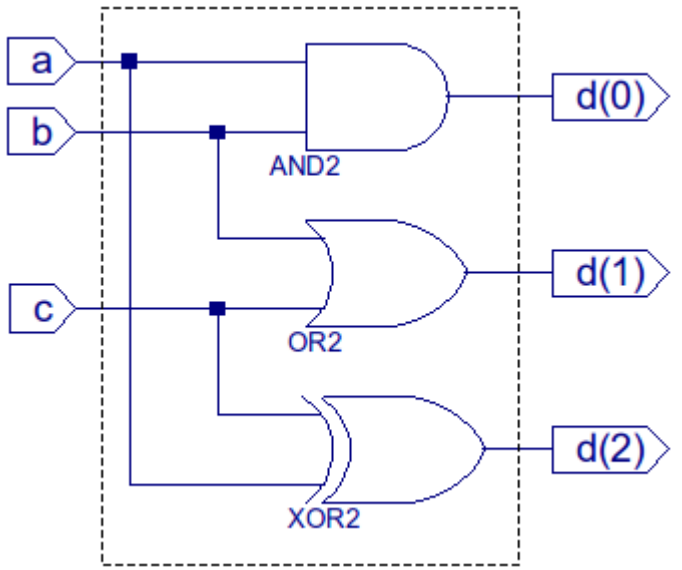
- `shared variable shv : integer;`
- `shared variable cnt : integer range 0 to 127 := 0;`

Egy egyszerű VHDL példa

```
library IEEE;
use IEEE.std_logic_1164.all;

entity simple_logic is
   port(
        a : in  std_logic;
        b : in  std_logic;
        c : in  std_logic;
        d : out std_logic_vector(2 downto 0)
       );
end simple_logic;

architecture dataflow of simple_logic is

begin

    d(0) <= a and b;
    d(1) <= b or c;
    d(2) <= a xor c;

end dataflow;
```

entitás

architektúra



Konkurens signal értékadások

- Feltételes értékadás:
  - Szintaxis:

    target <= { waveform **when** condition **else** }
    waveform

  - Példa:

```
out <= c when (a = '0' and b = '1') else '0';
```

- Kiválasztó értékadás:
  - Szintaxis:

    with expression **select**

    s <= waveform_1 **when** choice_list_1,
    waveform_2 **when** choice_list_2,
    ...
    waveform_n **when** choice_list_n;

Konkurens signal értékadások

- Példa kiválasztó értékadásra (3 - 8 dekóder):

```
signal dec_in  : std_logic_vector(2 downto 0);
signal dec_out : std_logic_vector(7 downto 0);

with dec_in select
   dec_out <= B"00000001" when B"000",
              B"00000010" when B"001",
              B"00000100" when B"010",
              B"00001000" when B"011",
              B"00010000" when B"100",
              B"00100000" when B"101",
              B"01000000" when B"110",
              B"10000000" when B"111";
```

Tervezés komponensekkel

- Egy komponens deklarációja az architektúrában:

```
component comp_1
    generic(
             p : integer;
             r : integer
           );
    port(
          a : in  std_logic;
          b : out std_logic
        );
end component;
```

Tervezés komponensekkel

- Komponens példányosítása:

```
comp_1_inst: comp_1
generic map(
              p => p_top,
              r => r_top
            )
port map(
           a => a_top,
           b => b_top
         );
```

Tervezés komponensekkel

- Komponens többszöri példányosítása:

```
comp_1_16_inst: for i in 0 to 15 generate
    comp_1_inst: comp_1
    generic map(
                 p => p_top,
                 r => r_top
               )
    port map(
              a => a_top(i),
              b => b_top(i)
            );
end generate comp_1_16_inst;
```

Szekvenciális vezérlés

- Az architektúrában szekvenciális vezérlést **process**-en belül valósíthatunk meg
- Több, egymással konkurens process is szerepelhet ugyanazon architektúrában
- Általános felépítése:

```
label: process(signal_1, signal_n)
          ...              -- deklarációs rész
begin
          ...              -- definíciós rész
end process label;
```

A process triggerelése

- Jel értékének megváltozása (szenzitív lista)
  - **`process`**`(signal_1, signal_n)`
  - **`wait on`**
- Logikai feltétel teljesülése
  - **wait until**
- Időtartam letelte (csak szimulációhoz)
  - **wait for**

Szekvenciális vezérlési szerkezetek

- Feltételes utasítások
  - if szerkezet
  - case szerkezet
- Ciklusok
  - loop
  - while
  - for

if szerkezet

- if szerkezet szintaxisa:

if feltétel **then**

utasítás_sorozat

{**elsif** feltétel **then**

utasítás_sorozat}

[**else**

utasítás_sorozat]

end if;

case szerkezet

- case szerkezet szintaxisa:

```
case kifejezés is
    when választás =>
            utasítás_sorozat;
        ...
    when választás =>
            utasítás_sorozat;
end case;
```

loop ciklus

- Feltétel nélküli végrehajtást tesz lehetővé
- loop ciklus szintaxisa:

  [label:] **loop**

  utasítás_sorozat;

  end loop [label];

- Példa: órajel generálás szimulációhoz

```
loop
   clk <= '0';
   wait for 10 ns;
   clk <= '1';
   wait for 10 ns;
end loop;
```

for ciklus

- for ciklus szintaxisa:

  [label:] **for** ciklusváltozó **in** diszkrét_tartomány

  utasítás_sorozat;

  end loop [label];

- Példa: vektortömb alapállapotba állítása

```
type d_type is array(63 downto 0) of std_logic_vector(15 downto 0);
signal data : d_type;


for i in 0 to 127 loop
    data(i) <= (others => '0');
end loop;
```

HDL alapú tervezés

IV.
Példák

Források, további szakirodalom

- Peter J. Ashenden: The VHDL Cookbook
- Peter J. Ashenden: The Designer's Guide to VHDL (3rd Edition)
- Enoch O. Hwang: Digital Logic and Microprocessor Design with VHDL
- Doulos KnowHow
- Doulos VHDL PaceMaker (interactive tutorial)
- dr. Horváth Tamás, Harangozó Gábor: VHDL segédlet

HDL alapú tervezés

Q & A