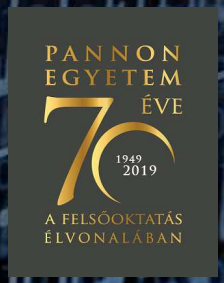
Pannon Egyetem, MIK-VIRT, Veszprém



Dr. Vörösházi Zsolt
voroshazi.zsolt@virt.uni-pannon.hu

Tervezési módszerek programozható logikai eszközökkel

4. A VHDL alapjai I.

Nyelvi típusok. Kifejezések, operátorok.

Frisítve: 2020. 10. 07.

Tárgyalt ismeretkörök
4. előadás

A VHDL alapjai I.:

- Nyelvi típusok:
  - Elemi és összetett típusok.
- Kifejezések, operátorok.

IEEE 1076

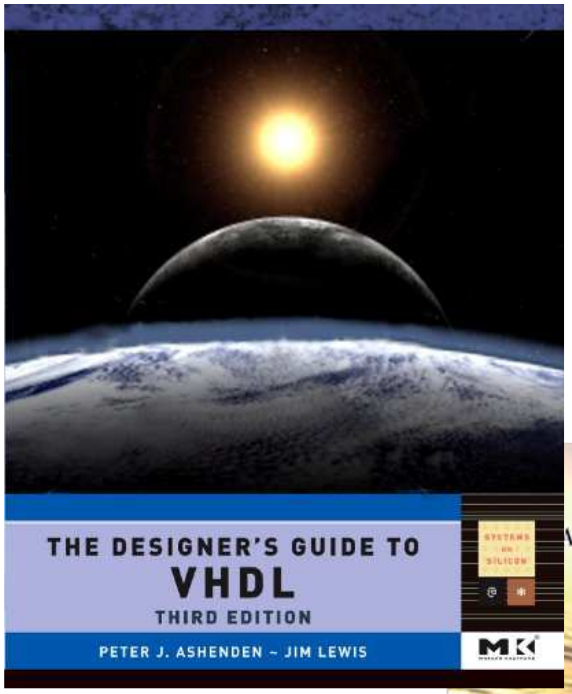
Bevezetés

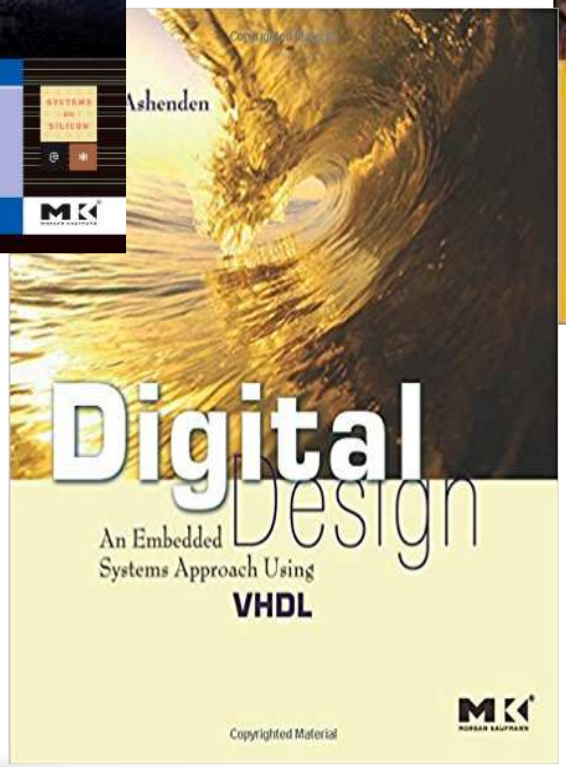
VHDL ALAPJAI

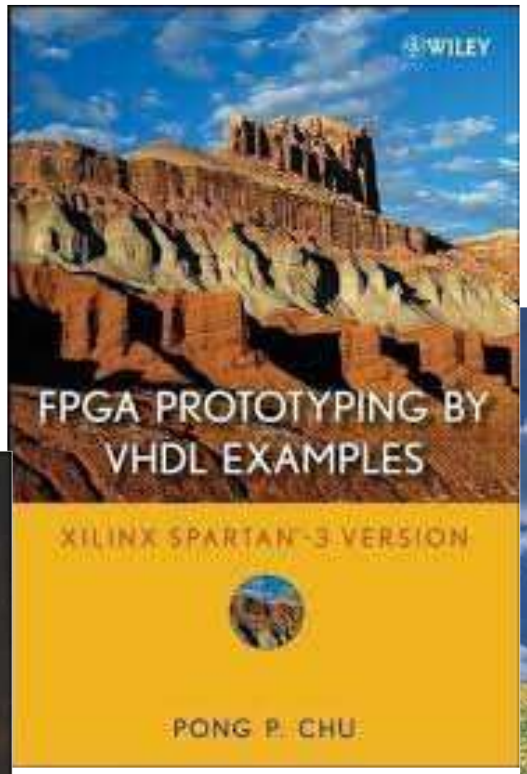
Felhasznált irodalom:

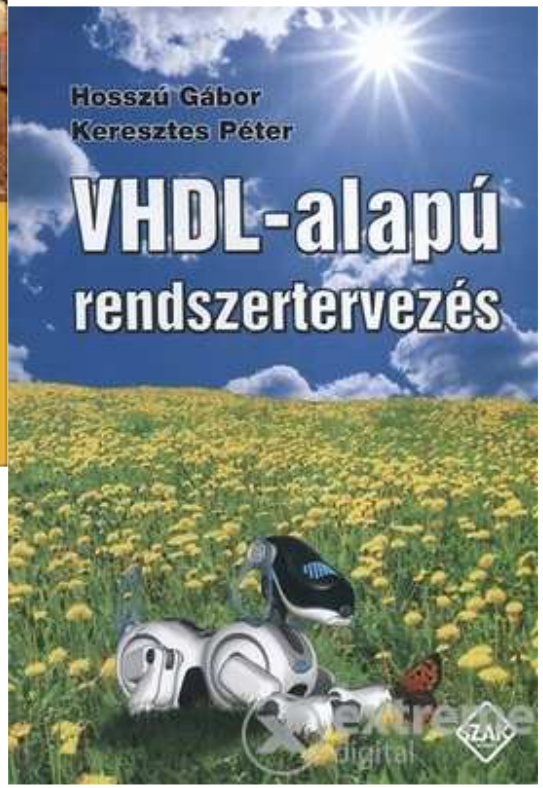
- 🕮 Vivado Design Suite – User Guide – Synthesis (UG901)
  https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_1/ug901-vivado-synthesis.pdf
- 🕮 Pong P. Chu - FPGA Prototyping by VHDL Examples: Xilinx Spartan-3
  http://academic.csuohio.edu/chu_p/rtl/fpga_vhdl.html
- 🕮 Hosszú Gábor - Keresztes Péter: VHDL ALAPÚ RENDSZERTERVEZÉS (2012 © Szak kiadó)
  http://www.szak.hu/konyvek_htm/vhdl.html
- 🕮 Horváth – Harangozó - VHDL VHSIC HARDWARE DESCRIPTION LANGUAGE - BME SEGÉDLET (2006)
  http://www.fsz.bme.hu/~tom/vhdl/vhdl_s.pdf
- 🕮 IEEE 1076-2008 Standard VHDL Reference Manual:
  https://ieeexplore.ieee.org/document/4772740
- 🕮 Richard E. Haskell & Darrin M. Hanna - Introduction to Digital Design VHDL (Digilent Inc.)
  https://reference.digilentinc.com/_media/textbooks:intro_digital_design-digilent-vhdl_online.pdf
- 🕮 *Real Digital - A hands-on approach to digital design* (Digilent Inc.)
  https://reference.digilentinc.com/textbooks:real_digital

Felhasznált irodalom:









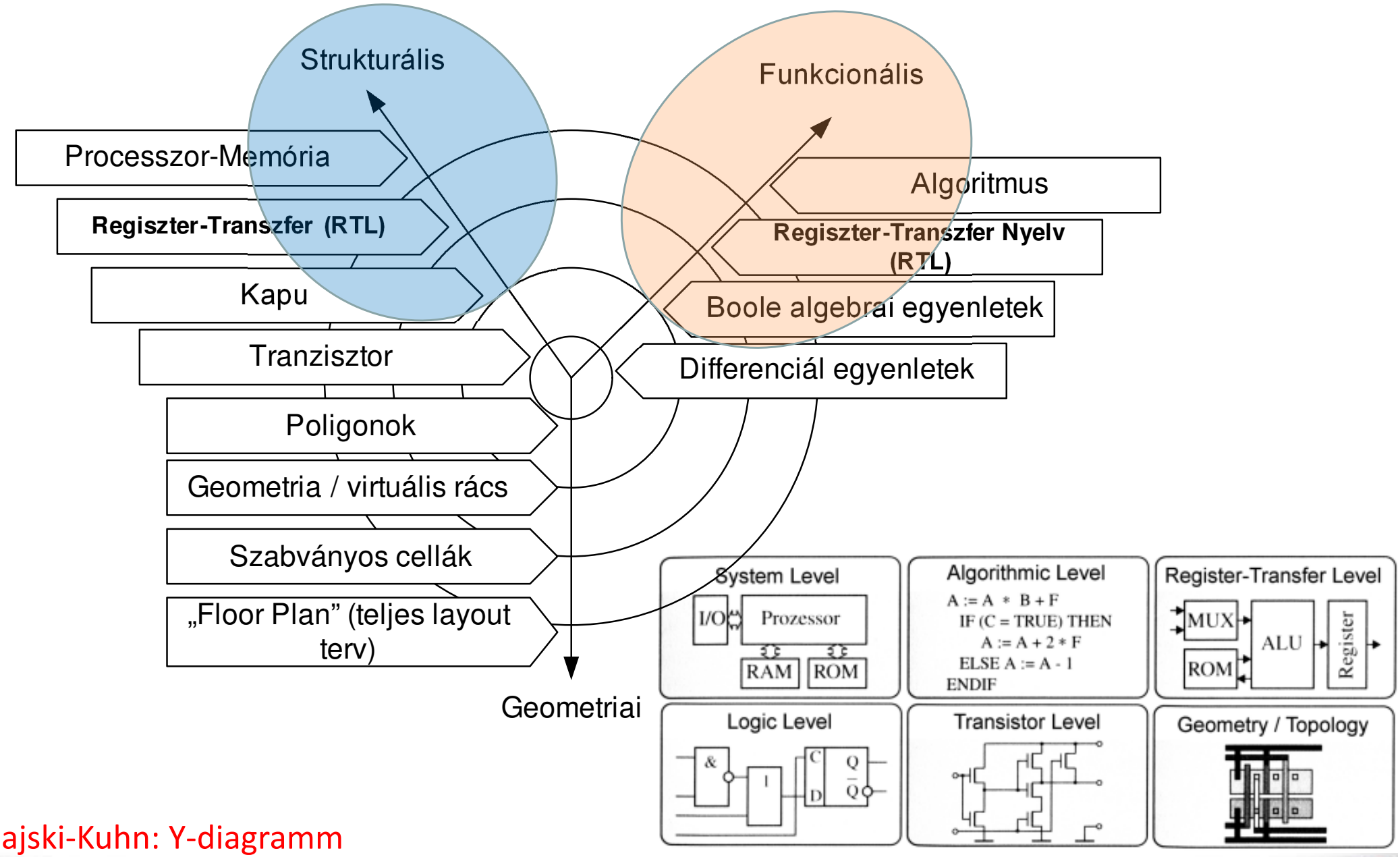
VHDL

- DARPA: **VHSIC** (**V**ery **H**igh **S**peed **I**ntegrated **C**ircuits) program
- Eszközfüggetlen alapleírás, viszont különböző FPGA gyártók logikai szintézis-eszközei készülhetnek hozzá (pl. Xilinx ISE/Vivado, Intel Quartus, Synopsys Synplify, …)
- Szabványos
  - **Behavior/Viselkedés**: what does it do?
  - **Structure/Strukturális**: what is it composed of?
  - **Functional/Funkcionális**: how do I interface to it?
  - **Fizikai**: how fast is it?
- VHDL (VHSIC Hardware Description Language)
- ADA, és Pascal nyelvekből származik

VHDL

- Támasztott követelmények kezdetektől
  - Struktúra leírása
  - Specifikáció (előírás)
- Későbbi követelmények (kb. 80-as évek végétől)
  - Szimuláció
  - Szintézis!
- 1980-as években jelent meg
- 1987: **IEEE-1076 szabvánnyá** vált, (VHDL-87)
  - Verziók:
    - 1992-1993, VHDL-93
    - 1998-2001, VHDL-2001
    - 2001-2008, VHDL-2008 (Vivado 2015.3-tól támogatott)
    - 2019, VHDL-2019

Tartományok (Y-diagram)



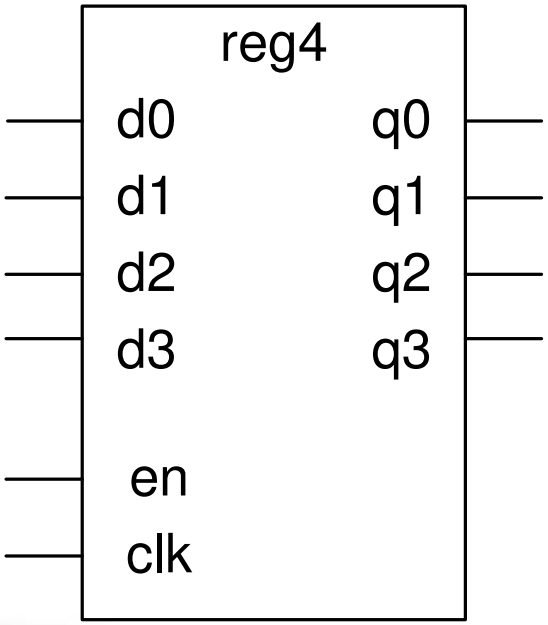
Gajski-Kuhn: Y-diagramm

Feladat 1.)

- Készítsünk egy új projektet Xilinx Vivado segítségével: „reg4”. Implementáljuk vagy ***a.) strukturális***, vagy ***b.) viselkedési*** VHDL modellben.
  - Legyen „d_latches” and „and2” gate (két-szintű hierarchia)
  - Készítsünk Testbench-et is („reg4_tb”) majd pedig szimuláljuk le a modellt Vivado Simulator segítségével.

VHDL modellezési koncepciók

```
--Entity deklarációja egy 4-bites parallel regiszternek (CLK,  Enable bemenetekkel)
entity reg4 is
port ( d0, d1, d2, d3, en, clk : in std_logic;
   q0, q1, q2, q3 : out std_logic );
end reg4;
```

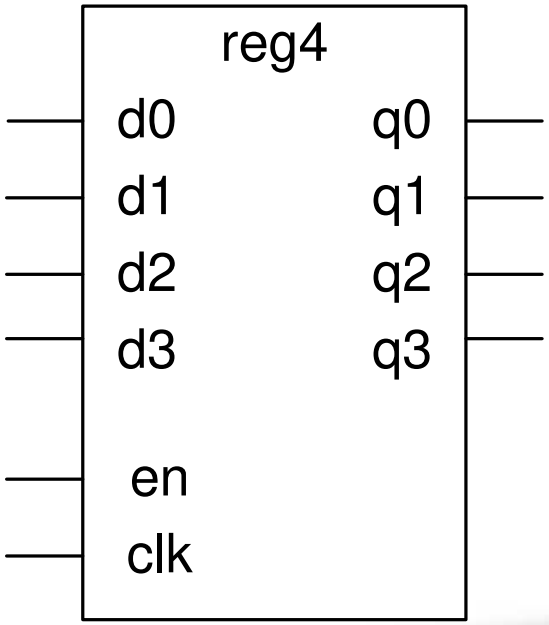


- VHDL modell leírása:
  - a.) viselkedési,
  - funkcionális,
  - b.) strukturális,
  - Időzítési.

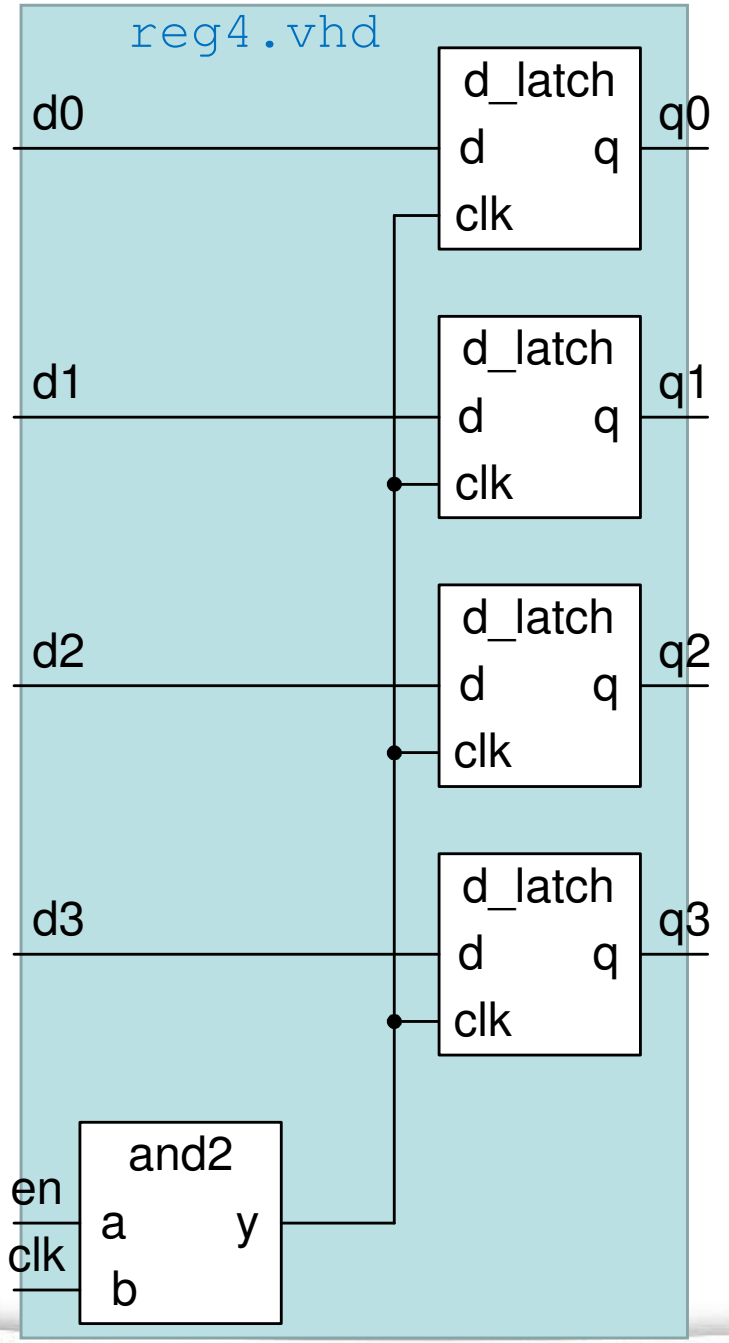
a.) Viselkedési modell

```
architecture Behavioral of reg4 is
begin
   storage : process (d0, d1, d2, d3, en, clk) is
   -- az érzékenységi lista megadása!
   variable stored_d0, stored_d1, stored_d2, stored_d3 : std_logic;
  begin
     if en = '1' and clk = '1' then
       stored_d0 := d0;
       stored_d1 := d1;
       stored_d2 := d2;
       stored_d3 := d3;
    end if;
       q0 <= stored_d0 after 5 ns;
       q1 <= stored_d1 after 5 ns;
       q2 <= stored_d2 after 5 ns;
       q3 <= stored_d3 after 5 ns;
    end process storage;
end architecture Behavioral;
```

szekvenciális process() utasítások



b.) Strukturális modell



Strukturális = egy digitális rendszer teljes belső struktúrájának, illetve az összeköttetések hierarchikus felépítése VHDL-ben.

b.) Strukturális modell (folyt.)

D tároló (latch)

```
entity d_latch is
  port ( d, clk : in std_logic;
  q : out std_logic );
end d_latch;

architecture basic of d_latch is
begin
  proc_latch: process (clk, d) is
  begin
    if clk = '1' then
      q <= d after 2 ns;
    end if;
  end process proc_latch;
end architecture basic;
```

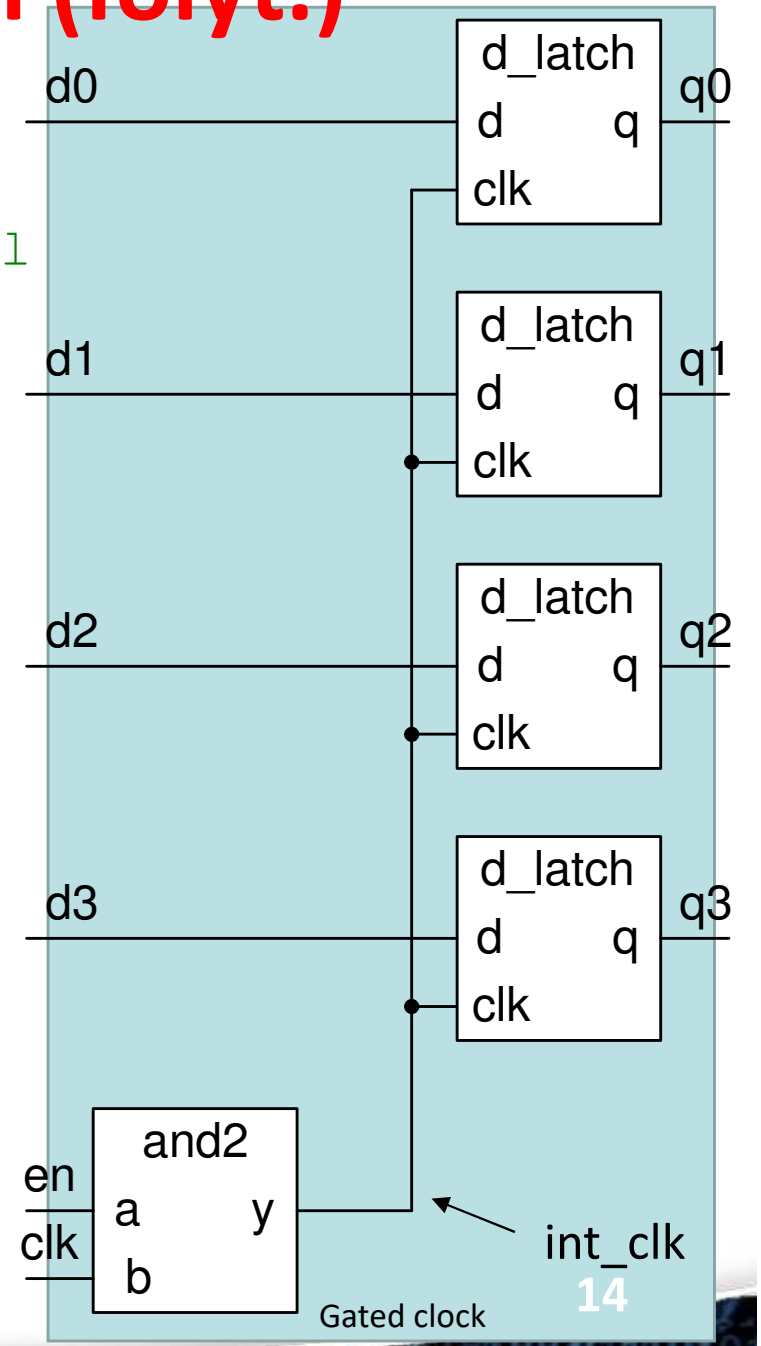
AND2 kapu

```
entity and2 is
  port ( a, b : in std_logic;
  y : out std_logic);
end and2;

architecture basic of and2 is
begin
  proc_and2: process(a, b) is
  begin
    y <= a and b after 2 ns;
     --wait on a, b;
  end process proc_and2;
end architecture basic;
```

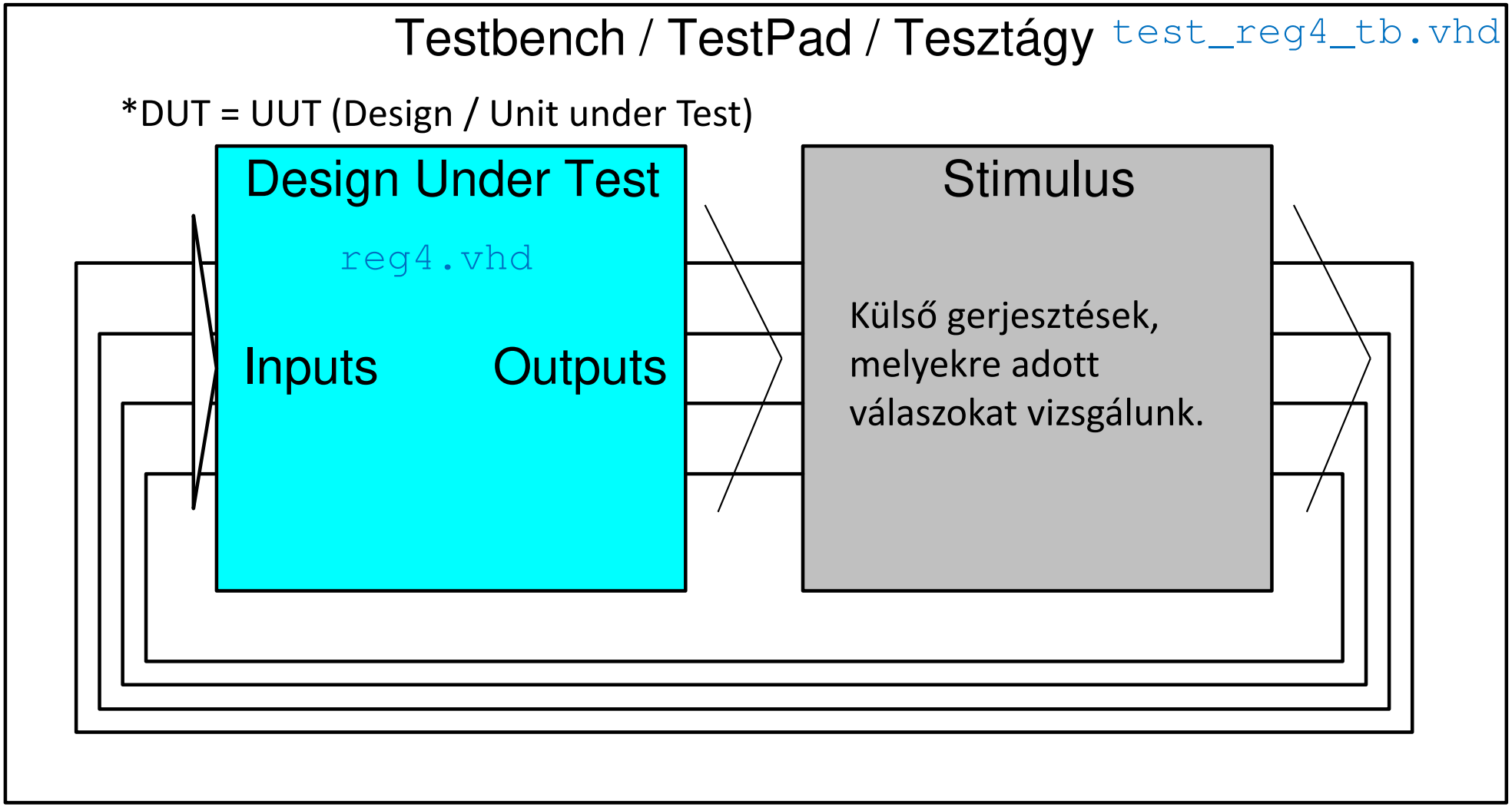
Nem lett volna szükséges process()-t használni!

b.) Strukturális modell (folyt.)

```
architecture struct of reg4 is
   signal int_clk : std_logic;  --internal
    signal
begin
   bit0 : entity work.d_latch(basic)
     port map (d0, int_clk, q0);
   bit1 : entity work.d_latch(basic)
     port map (d1, int_clk, q1);
   bit2 : entity work.d_latch(basic)
     port map (d2, int_clk, q2);
   bit3 : entity work.d_latch(basic)
     port map (d3, int_clk, q3);
   gate : entity work.and2(basic)
     port map (en, clk, int_clk);
end architecture struct;
```



TestBench: VHDL szimuláció



*Lásd korábbi Vivado **Simulator** bevezető példa

VHDL TestBench (tesztágy)

```
entity test_bench is
end entity test_bench;          -- testbench-nek nincs portlistája

architecture reg4_tb of test_bench is
  signal d0, d1, d2, d3, en, clk, q0, q1, q2, q3 : std_logic;
begin
  dut : entity work.reg4(Behavioral)
    port map ( d0, d1, d2, d3, en, clk, q0, q1, q2, q3 );

  stimulus : process is
  begin
    d0 <= '1'; d1 <= '1'; d2 <= '1'; d3 <= '1';
    en <= '0';
    wait for 20 ns;
    en <= '1'; wait for 20 ns;
    d0 <= '0'; d1 <= '0'; d2 <= '0'; d3 <= '0'; wait for 20 ns;
    en <= '0'; wait for 20 ns;
    -- . . .
    wait;
  end process stimulus;
end architecture reg4_tb;
```

VHDL alapjai I.

NYELVI TÍPUSOK, KIFEJEZÉSEK

VHDL lexikai elemei

- VHDL egy erősen típusos nyelv!
  - *Variable* és *signal* deklarálhatóak akár ‚`n-bites`' objektumként is
- **Comment (megjegyzések)**

```
variable data : std_logic; --data változó deklarációja
--ez egy több soros...
--... komment
```

- **Identifiers (azonosítók)** neve lehet
  - `'A'…'Z', 'a'…'z', '0'…'9' és '_'`
  - Betűvel kell, hogy kezdődjenek!
  - Nem lehet egymást követő ___, és nem végződhet '_' -re
  - **Case in-sensitive!**: Nem tesz különbséget pl. `Cat, CAT, cat, CaT` között. (De használjuk konzekvens módon őket!)

VHDL - Foglalt szavak

abs	configuration	impure	null	rem	type
access	constant	in	of	report	unaffected
after	disconnect	inertial	on	return	units
alias	downto	inout	open	rol	until
all	else	is	or	ror	use
and	elsif	label	others	select	variable
architecture	end	library	out	severity	wait
array	entity	linkage	package	signal	when
assert	exit	literal	port	shared	while
attribute	file	loop	postponed	sla	with
begin	for	map	procedure	sll	xnor
block	function	mod	process	sra	xor
body	generate	nand	pure	srl	
buffer	generic	new	range	subtype	
bus	group	next	record	then	
case	guarded	nor	register	to	
component	if	not	reject	transport	

Számok (literals)

- Decimális számok (alapértelmezett):
  - `23, 0, 146, 46E5`
- Valós számok: (mantissza-exponens decimális alapú)
  - `23.1, 0.0, 3.14159`
  - `46.E5, 1.0E+12, 34.0e-08`
- N-alapú számok (n#x#):
  - `2#1111_1101#, 16#FD#, 16#0fd#, 8#0375#` `-- dec(253)`
  - `2#0.1000#, 8#0.4#, 16#0.F#` `-- =dec(0.5)`
- Számok szeparációval ('_'), és lehetséges változataik:
  - `123_456, 3.141_592_6, 2#1111_1100#`

Karakterek, string-ek, bit string-ek

- Karakterek
  - `'A', 'z', ',', '', ' '`
- String-ek (karakter füzérek)
  - `"A string", "" --üres string`
- Bit string-ek (bit füzérek)
  (**B/O/X/D**-alappal, és '_' szeparátorral kombinálhatók)
  - Dec, `D"23", d"1000_0003"` (alap)
  - Binary, `B"0110001", b"0110_0111"`
  - Octal, `O"372", o"00"`
  - Hex, `X"FA", x"0d"`

VHDL objektumok

- Konstans (constant)
- Változó (variable) – vigyázat, mert önmagában tárol is (~regiszter)! ,
- Jel (signal) – „csak” összeköttetés, de nem tárol (~vezeték / globális „változó”),
- Fájl (file).

Konstans /változó / jel deklarációk

```
constant identifier {, ...}: subtype_indication [:=
   initial expression];
variable identifier {, ...}: subtype_indication [:=
   initial expression];
Signal identifier {, ...}: subtype_indication [:= initial
   expression];
```

```
constant NUM_OF_BYTES: integer := 4;
constant NUM_OF_BITS : integer := 8* NUM_OF_BYTES;
constant E : real := 2.718281828;
constant PROP_DELAY : time := 3 ns;
constant SIZE_LIMIT, COUNT_LIMIT : integer := 255;

variable index : natural:= 0;
variable sum, average, largest : real;
signal wire: std_logic:= '0';
```

Megj: a *constant* és *variable* deklarációknak mindig a *begin… end* ELŐTTI részen kell szerepeltetni egy VHDL modulhoz tartozó architektúra részen!!

VHDL adattípusok

- **I. Elemi (skalár):**
  - egész (integer),
  - lebegőpontos / valós (real),
  - fizikai mennyiségek (pl: idő, hossz mérték egys.)
  - felsorolás (enum.),
  - egyéb elemi adattípusok.
- **II. Összetett:**
  - tömb (egy-, ill. több-dimenziós),
  - rekord
- **III. File** (nem tárgyaljuk)

I. Elemi (skalár) adattípusok

- **Előredefiniált elemi adattípusok és értékkészleteik:**

```
-- egész típus
type integer is range -2147483648 to
2147483648 ;

-- lebegőpontos típus (max 6 tizedesjegy
pontosságig) :
type real is range -16#0.7FFFFFF8#+32 to
16#0.7FFFFFF8#+32 ;

-- felsorolás / enum. típusok:
type boolean is (False, True) ;
type bit is ('0', '1') ;
```

Típus (type) deklarációk

- **type** identifier **is** type_definition

- Egész típusra

```
type_definition <= range expr. (to | downto)
  expr.

type apples is range 100 downto 0;
type oranges is range 100 downto 0;
oranges := apples; --érvénytelen típus egyeztetés!!!

constant NUM_OF_BITS : integer :=32;
type bit_index is range 0 to NUM_OF_BITS-1;
```

Típus (type) deklarációk

- Valós típusra

```
type_definition <= range expr. (to | downto)
  expr.
```

```
type input_level is range -10.0 to +10.0;
type probability is range 1.0 downto 0.0;
```

Előredefiniált elemi (skalár) adattípusok

- **Értékkészleteik:**

```
-- karakter típus
type character is(
NUL, SOH, STX, ETX, EOT, ENQ, ACK, BEL, BS, HT, LF, VT,
FF, CR, SO, SI, DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB,
CAN, EM, SUB, ESC, FSP, GSP, RSP, USP,
' ', '!', '"', '#', '$', '%', '&', ''', '(', ')', '*', '+',
',', '-', '.', '/', '0', '1', '2', '3', '4', '5', '6', '7',
'8', '9', ':', ';', '<', '=', '>', '?', '@', 'A', 'B', 'C',
'D', 'E', 'F',  'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N',
'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z',
'[', '\', ']', '^', '_', '`', 'a', 'b', 'c', 'd', 'e', 'f',
'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r',
's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '{', '|', '}', '~',
DEL);
```

Előredefiniált elemi (skalár) adattípusok

```
--Fizikai típus (általánosan)
range expr. (to | downto) expr.
   units
      identifier;
      {identifier = physical_literal;}
   end units [identifier];

physical_literal <= [decimal_literal
   | based_literal] unit_name
```

Előredefiniált elemi (skalár) adattípusok

```
-- Fizikai típus: pl. idő
type time is range impl. def.
  units
    fs; --elsődleges m.egység az [fs]
    ps=1000 fs;
    ns=1000 ps;
    us=1000 ns;
    ms=1000 us;
    sec=1000 ms;
    min=60 sec;
    hr=60 min;
end units;
```

Előredefiniált elemi (skalár) adattípusok

```
-- Fizikai típus: pl. ellenállás
type resistance is range 0 to 1E9
  units
    ohm;                        --elsődleges m.egys [ohm]
    kohm = 1000 ohm;
    Mohm = 1000 kohm;
end units resistance;
-- Fizikai típus: pl. hosszúság
type length is range 0 to 1E9
  units
    um;                          --elsődleges m.egys [um]
    mm=1000 um;                  --metrikus egységek
    m=1000 mm;
    inch=25400 um;               --angolszász egységek
    foot=12 inch;
end units length;
```

Elemi (skalár) adattípusok

```
--felsorolás/enumerated típus
type alu_function is (disable, pass, add,
  subtract, multiply, divide);
type octal_digit is ('0', '1', '2', '3',
  '4', '5', '6', '7');
variable alu_op : alu_function;
variable last_digit : octal_digit := '0';

alu_op := subtract;
last_digit := '7';
```

```
type boolean is (false, true);
```

„Bit” típusok: STANDARD_LOGIC_1164

- **Bit** type

```
type bit is ('0', '1'); --származtatott típus (de nem ezt
  használjuk), mivel inout esetén 'Z'-nem támogatott  !
```

- **standard_logic type** --hagyományos num-std típus

```
type std_logic is (
 '0', --Forcing 0
 '1', --Forcing 1
 'Z', --High impedance (tri-state buffer)

 'U', --Uninitialized /nincs kezdőértéke
 'X', --Forcing unknown / ismeretlen
  (jelet két különböző értékre kényszerítenénk egyszerre)
 'W', --Weak unknown
 'L', --Weak 0
 'H', --Weak 1
 '-', --Don't care
);
```

Hagyományos TTL szintek

Vivado/ISE szintézis eszköz korlátozottan támogatja.

Feloldhatók ún. rezolúciós függvények segítségével

Megj: VHDL-ben csak a '0', '1' és 'Z' jelent szintetizálható értékeket!

Altípusok (subtype) deklarációk

- **subtype** identifier **is** type_mark [**range** expr. (**to** | **downto**) expr.]

- Előre-definiált altípusok VHDL-ben:
  - **subtype natural is** integer **range** 0 **to** highest_integer;
  - **subtype positive is** integer **range** 1 **to** highest_integer;

Típus minősítők

```
type logic_level is (unknown, low, undriven,
  high);
type system_state is (unknown, ready, busy);

subtype valid_level is logic_level range low
  to high;

P1:
logic_level'(unknown),
system_state'(unknown)
logic_level'(high),
valid_level'(high)
```

Explicit típuskonverziók a *std_logic_vector* és a *numeric* adattípusok között

- `real(123)` `-- 123.0`
- `integer(1.23)` `-- rounding to 1`

Mit szeretnénk? **Hogyan kell megadni?**

VHDL adattípus (x)	Eredmény VHDL adat-típusa	konverziós VHDL függvény (type cast)
unsigned, signed	std_logic_vector	std_logic_vector(x)
signed, std_logic_vector	unsigned	unsigned(x)
unsigned, std_logic_vector	signed	signed(x)
unsigned, signed	integer	to_ingeger(x)
natural	unsigned	to_unsigned(x,size)
integer	signed	to_signed(x, size)

VHDL-93 és IEEE.STD_LOGIC_1164 csomagban

VHDL operátor	jelentés	operandus VHDL adat típusa	eredmény VHDL adat típusa
a ** b	hatványozás	interger (egész)	integer (egész)
a + b	összeadás		
a - b	kivonás		
a * b	szorzás		
a / b	osztás		
a & b	konkatenáció (összefű-zés)	1D tömb, illetve tömbelemek	1D tömb
a = b	egyenlő	tetszőleges	boolean
a /= b	nem egyenlő		
a < b	kisebb, mint	skalár, 1D tömb	boolean
a <= b	kisebb egyenlő, mint		
a > b	nagyobb, mint		
a >= b	nagyobb egyenlő, mint		
not a	negálás	boolean, std_logic, std_logic_vector	boolean, std_logic, std_logic_vector
a and b	ÉS		
a or b	VAGY		
a nand b	NEM-ÉS		
a nor b	NEM-VAGY		
a xor b	Kizáró VAGY, Antivalencia		
a xnor b	Ekvivalencia		

LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;

Szimulálható, de nem mind szintetizálható operátorok!

VHDL-93 és IEEE.STD_LOGIC_1164 csomagban

abs	abszolútérték	integer,	integer
mod	maradékos osztás (modulo)	fizikai	fizikai
rem	maradék (remainder)		
sll	balra léptetés (logikai)		
srl	jobbra léptetés (logikai)		
sla	balra léptetés (arit)	1D tömb, illetve	1D tömb, illetve
sra	jobbra léptetés (arit)	tömbelemek	tömbelemek
rol	forgatás (rotate) balra		
ror	forgatás (rotate) jobbra		

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
```

Szimulálható, de nem mind szintetizálható operátorok!

Szintetizálható operátorok: VHDL

VHDL operátor	jelentés	operandus VHDL adattípusa	eredmény VHDL adattípusa
a + b	összeadás	unsigned, natural signed, integer	unsigned, signed
a - b	kivonás		
a * b	szorzás		
a = b	egyenlő	unsigned, natural signed, integer	unsigned, signed
a /= b	nem egyenlő		
a < b	kisebb, mint		
a <= b	kisebb egyenlő, mint		
a > b	nagyobb, mint		
a >= b	nagyobb egyenlő, mint		

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
```

Ezek már Szimulálható és egyben Szintetizálható operátorok!

Általunk használt STD csomagok

Melyek szimulálhatók és szintetizálhatók is egyben:

- **LIBRARY IEEE;**
- **USE IEEE.STD_LOGIC_1164.ALL;**
  --std_logic, std_logic_vector támogatása
- **USE IEEE.NUMERIC_STD.ALL;**
  --aritmetikai operátorok támogatása, de unsigned, signed típusokon!
- **USE IEEE.STD_LOGIC_UNSIGNED.ALL;**
  -- inkrementálás támogatása std_logic_vector típusokon

Attribútumok

- Attribútum, mint valamely objektum névvel ellátott jellemző tulajdonsága.
- Attribútuma lehet:
  - típusnak, altípusnak,
  - tömbnek,
  - jelnek, stb.
- Megadása: `objektum_név'attribútum`

Típusok attribútumai

`T'left`	-- első (*left-most*) érték T-ben
`T'right`	-- utolsó (*right-most*) érték T-ben
`T'low`	-- legkisebb érték T-ben
`T'high`	-- legnagyobb érték T-ben
`T'ascending`	-- igaz, ha T egy növekvő sorrendű
`T'image(x)`	-- *stringként* ábrázolja a T-ben lévő *x értékét* *(*value*)*
`T'value(s)`	-- egy *value* (numerikus értékként) ábrázolja az *s* (string-et)

Példa: Tartomány (típus) attribútumai

```
type set_index_range is range 21
  downto 11;
```

T ≡ set_index_range (típus)

```
set_index_range'left = 21
set_index_range'right = 11
set_index_range'low = 11
set_index_range'high = 21
set_index_range'ascending = false
set_index_range'image(14)= "14" --string
set_index_range'value("20")= 20 --érték
```

Példa: Felsorolt típusok attribútumai

```
type logic_level is (unknown, low, undriven,
high);

logic_level'left = unknown
logic_level'right = high
logic_level'low = unknown
logic_level'high = high
logic_level'ascending = true
logic_level'image(undriven) = "undriven"
logic_level'value("LOW") = low   -- =1
```

Diszkrét és fizikai típusok attribútumai

```
T'pos(x)              -- x pozíciója T-ben
T'val(x)              -- x pozíción lévő érték T-ben
T'succ(x)             -- x-nél egyel nagyobb pozíción
lévő érték T-ben (successor of x)
T'pred(x)             -- x-nél egyel kisebb pozíción
lévő érték T-ben (predecessor of x)
T'leftof(x)           -- x-től egyel balra lévő érték T-
ben (left of x)
T'rightof(x)          -- x-től egyel jobbra lévő érték
T-ben (right of x)
```

Példa: Felsorolt típus attribútumok

```
type logic_level is (unknown, low, undriven, high);

logic_level'pos(unknown)= 0
logic_level'val(3)= high
logic_level'succ(unknown)= low
logic_level'pred(undriven)= low
logic_level'ascending= true
logic_level'image(undriven)= "undriven" --string
logic_level'val("LOW")= low                    -- =1, érték

time'pos(4 ns)=4_000_000      -- mivel [fs] alapegység
```

II. Összetett típus: Tömbök (arrays)

- **array** (discrete_range {, …}) **of** element_subtype_indication
  - discrete_range <= discrete_subtype_indication | expr. (**to** | **downto**) expr.
  - subtype_indication <= type_mark [**range** expr. (**to** | **downto**) expr.]

- **type** BE_word **is array** (0 **to** 31) **of** **std_logic**;
  --BIG endian
- **type** LE_word **is array** (31 **downto** 0) **of** **std_logic**;
  --LITTLE endian

Tömbök

```
type controller_state is (initial, idle,
  active, error);

type state_counts is array (idle to error)
  of natural;

type state_counts is array (controller_state
  range idle to error) of natural;
variable counters : state_counts;

  counters(active):=counters(active)+1;
```

Tömbök

```
type BE_word is array (0 to 31) of
  std_logic; --BIG_END

subtype coeff_ram_address is integer range 0
  to 63;
type coeff_array is array(coeff_ram_address)
  of real;

signal coeff : coeff_array;
variable buffer_register, data_register :
  BE_word;
  coeff(0):=0.0;
  data_register:=buffer_register;
```

Tömbök (aggregált)

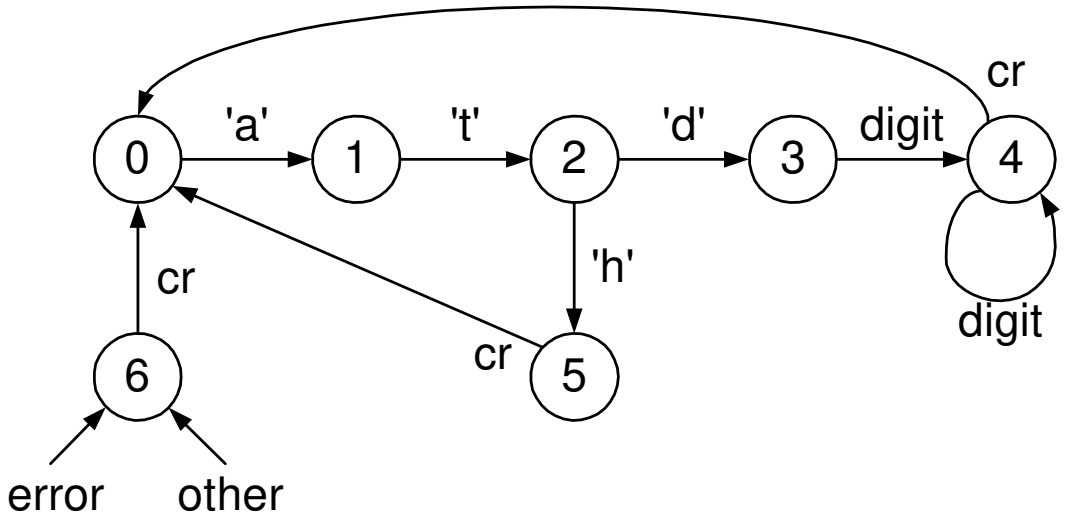
```
aggregate <= (([choices => ] expr. ) {, …})
choices <= (expr. | discrete_range | others)
   {| …}
```

```
type coeff_array is array (coeff_ram_address) of
   real;
variable coeff : coeff_array:=(0=>1.6, 1=>2.3,
   2=>1.6, 3 to 63=>0.0);
variable coeff : coeff_array:=(0=>1.6, 1=>2.3,
   2=>1.6, others=>0.0);
variable coeff : coeff_array:=(0|2=>1.6, 1=>2.3,
   others=>0.0);
```

Összetett típus: Több-dimenziós tömbök

```
type state is range 0 to 6;
type symbol is ('a', 't', 'd', 'h',
  digit, cr, error, other);


type transition_matrix is array
  (state, symbol) of state;
variable transition_table :
  transition_matrix;
Pl: transition_table(2,'d');   --indexelés
```

Pl: Több-dimenziós tömb



FSM: Finite State Machine

Pl. MODEM

- 'atd' -> digit(s) -> cr
- 'ath' -> cr
- 'other' = minden más karakter ('atdh', 'cr' kívül)

```
constant next_state : transition_matrix :=
(
   0 => ('a' => 1, others => 6),
   1 => ('t' => 2, others => 6),
   2 => ('d' => 3, 'h' => 5, others => 6),
   3 => (digit => 4, others => 6),
   4 => (digit => 4, cr => 0, others => 6),
   5 => (cr => 0, others => 6),
   6 => (cr => 0, others => 6)      );
```

Tömb attribútumok

```
A'left(N)              -- left bound of index range N
A'right(N)             -- right bound of index range N
A'low(N)               -- lower bound of index range N
A'high(N)              -- upper bound of index range N
A'range(N)             -- index range of dim. N of A
A'reverse_range(N)     -- reverse of index range N
A'length(N)            -- length of index range N
A'ascending(N)         -- true if index range is ascending
```

A: tömb

Tömb (több-dimenziós) attribútumok

```
type A is array (1 to 4, 31 downto 0)
  of boolean;
```

```
A'left(1)=1
A'right(2)=0
A'length(1)=4
A'ascending(1)=true
A'ascending(2)=false
A'low(2)=0
```

```
A'low(1)=1
A'high(2)=31
A'length(2)=32
```

Tömb attribútumok

```
type free_map is array (0 to 5) of
  std_logic_vector(15 downto 0);
variable count := 0;
…
for index in free_map'range loop
   if free_map(index) then
      count := count + 1;
   end if;
end loop;
```

Process() –be kell tenni!

Dimenzió-nélküli (<>) tömb típusok

```
type sample is array (natural range <>) of integer;
variable short_sample_buf : sample(0 to 63);

type string is array (positive range <>) of character;

type bit_vector is array (natural range <>) of std_logic;
subtype byte is bit_vector(7 downto 0);

type std_ulogic_vector is array (natural range <> ) of
   std_ulogic;
subtype std_ulogic_word is std_ulogic_vector(0 to 31);
signal csr_offset : std_ulogic_vector(2 downto 1);
variable current_test : std_ulogic_vector(0 to 13) :=
   "ZZZZZZZZZZ----";
constant all_ones : std_ulogic_vector(15 downto 0) :=
   X"FFFF";
```

Tömb szeletek (array slices)

```
entity byte_swap is
port(
   input : in  std_logic_vector(15 downto 0);
   output : out std_logic_vector(0 to 15));
end entity byte_swap;
-------------------------------------
architecture behavior of byte_swap is
begin
   swap : process (input)
   begin
      output(8 to 15) <= input(7 downto 0);
      output(0 to 7)  <= input(15 downto 8);
   end process swap;
end architecture behavior;
```

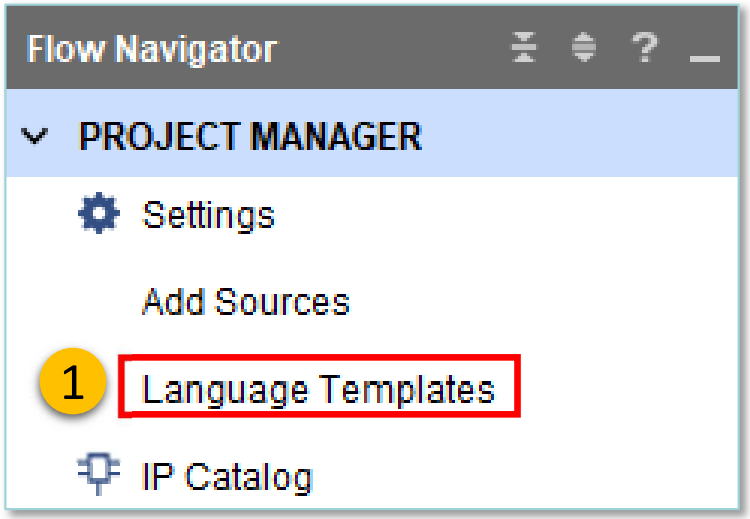
Összetett típus: rekord

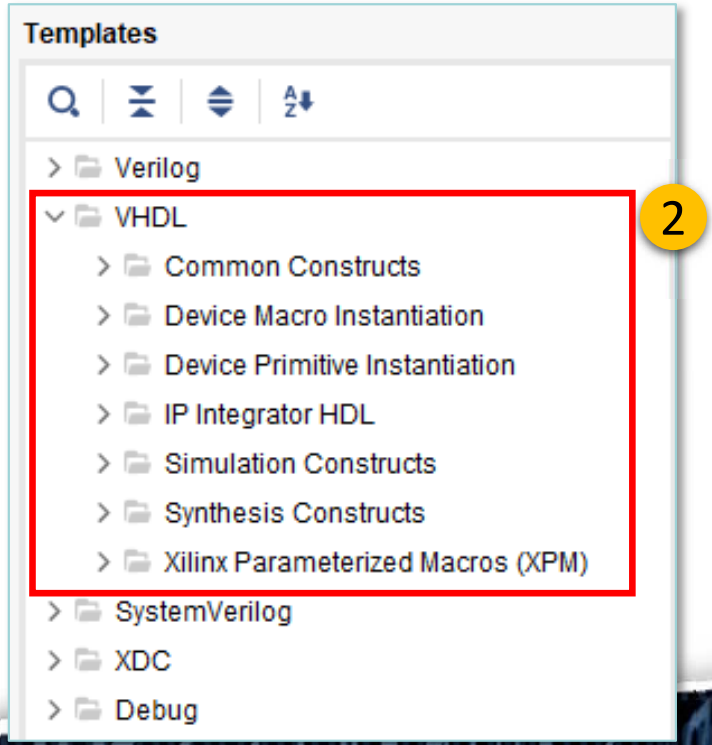
```
type time_stamp is record
       seconds : integer range 0 to 59;
       minutes : integer range 0 to 59;
       hours   : integer range 0 to 23;
end record time_stamp;

variable sample_time, current_time :
  time_stamp;
sample_time:= current_time;
sample_hour:= sample_time.hours;
constant midday : time_stamp:=(0,0,12);
```

```
constant midday : time_stamp := (hours =>
  12, minutes => 0, seconds => 0);
```

Xilinx Vivado Project Manager

- VHDL Language Templates (beépített nyelvi sablonok)
  - Project Manager → Language Templates…
  - Keresés a „VHDL” kulcsszóra
    - Common Constructs (pl. entitás, architektúra stb.)
    - Simulation Constructs (csak szimulálható kódrészletek)
    - Synthesis Contstructs (szintetizálható kódrészletek, ill. primitívek)





Általunk használt STD csomagok

Melyek szimulálhatók és szintetizálhatók is egyben:

- **LIBRARY IEEE;**
- **USE IEEE.STD_LOGIC_1164.ALL;**
  --std_logic, std_logic_vector támogatása
- **USE IEEE.NUMERIC_STD.ALL;**
  --aritmetikai operátorok támogatása, de unsigned, signed típusokon!
- **USE IEEE.STD_LOGIC_UNSIGNED.ALL;**
  -- inkrementálás támogatása std_logic_vector típusokon

Feladat 2.)

- Tervezzen egy ***3-bites Összeadó áramkört*** a **numeric_std** VHDL könyvtári csomag használatával („*osszeado_3bit.vhd*”)!
- Használjon *natural* típust az **ADAT_BIT** konstanshoz melynek kezdőértéke:= 3 (ez a bitszélessége az a_in, és b_in bemeneteknek, amelyek legyenek std_logic_vector típusúak).
- Megj: alkalmazza az *explicit* típus-konverziót unsigned és, std_logic_vector között! (**USE IEEE.NUMERIC_STD.ALL;**)
- Készítsen egy test-bench-et, majd szimulálja le a VHDL modul viselkedését („*osszeado_3bit_tb.vhd*”)!
- Adja meg a fizikai lábkiosztást (*.xdc*), majd töltse le FPGA-ra *(.bit):*
  - a_in: jobboldali 3 kapcsolóra, b_in: jobboldali 3 nyomógombra
  - carry_out és sum_out: **1+3=4** LED
    (carry_out az MSB átvitelbit)

Emlékeztető: Explicit típuskonverziók a *std_logic_vector* és a *numeric* adattípusok között

Mit szeretnénk? Hogyan kell megadni?

VHDL adattípus (x)	Eredmény VHDL adat-típusa	konverziós VHDL függvény (type cast)
unsigned, signed	std_logic_vector	std_logic_vector(x)
signed, std_logic_vector	unsigned	unsigned(x)
unsigned, std_logic_vector	signed	signed(x)
unsigned, signed	integer	to_ingeger(x)
natural	unsigned	to_unsigned(x,size)
integer	signed	to_signed(x, size)

Feladat 2.a.) VHDL module Megoldás (3-bites összeadó)

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL; -- std_logic_vector típusokhoz
USE IEEE.NUMERIC_STD.ALL;    -- csomag a '+' operátor használatához !

entity osszeado_3_bit is
   port(a_in, b_in: in std_logic_vector(2 downto 0);
        carry_out : out std_logic;
        sum_out   : out std_logic_vector(2 downto 0)  );
end osszeado_3_bit;

architecture behav of osszeado_3_bit is
   constant ADAT_BIT: natural:= 3; --3-bites konstans kifejezés
   signal a_sig, b_sig, sum_sig: unsigned(ADAT_BIT downto 0); --1 bittel szélesebb
begin
   a_sig <= unsigned('0' & a_in);   --'a' kiterjesztése előjel nelkülivé!
   b_sig <= unsigned('0' & b_in);
   sum_sig <= a_sig + b_sig;        --összeadás unsigned típuson értelmezett!
   -- eredmény visszaalakítása (ADAT_BIT) szélességűre
   sum_out <= std_logic_vector(sum_sig(ADAT_BIT-1 downto 0));
   --eredmény MSB bitje lesz a generált carry out, átvitel
   carry_out <= std_logic(sum_sig(ADAT_BIT));
end behav;
```

Feladat 2.b.) VHDL testbench Megoldás I. (3-bites összeadó)

Testbench – néhány esetre

```
-- Stimulus process (osszeado_3bit_tb.vhd)
 stim_proc: process
 begin
            a_in <= (others => '0');
            b_in <= (others => '0');
            wait for 20 ns;

            a_in <= "001";
            b_in <= "100";
            wait for 20 ns;

            a_in <= "010";
            b_in <= "010";
            wait for 20 ns;

            a_in <= "101";
            b_in <= (others => '1');
            wait for 20 ns;
   wait;
 end process;
```

Megj.: itt nem az összes lehetséges gerjesztési kombináció lett megadva a teszteléshez

Feladat 2.c.) VHDL testbench Megoldás II. (3-bites összeadó)

Testbench – for ciklussal

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use IEEE.STD_LOGIC_unsigned.all;  -- inkrementálás std_logic_vector típuson
. . .
 -- Stimulus process (osszeado_3bit_tb.vhd)
stim_proc: process
  begin
            a_in <= "000";
            b_in <= "111";
            wait for 20 ns;
            for index in 0 to (2**3 - 1) loop     -- vagy (2**(a_in'lenght) -1)
                        a_in <= a_in + 1;
                        wait for 20 ns;
                        b_in <= b_in - 1;
                        wait for 20 ns;
              -- assert a_in /= "110" report "a_in egyenlo =110"   severity warning;
            end loop;
           wait;
end process;
```

Megj.: a gerjesztési teszt vektorok ***for*** ciklus segítségével lettek generálva.

Feladat 2.) Szimulációs eredmény