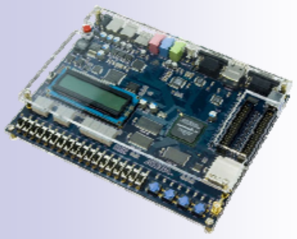
Struktury počítačových systémů

Přednáší: Richard Šusta

20. října 2011 verze 1.0



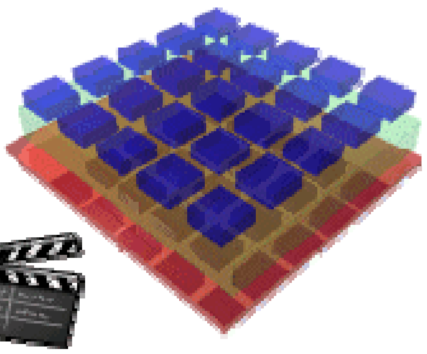
ČVUT-FEL v Praze – kód předmětu A0B35SPS

Graphic/Memory Arrays

...and their addressing

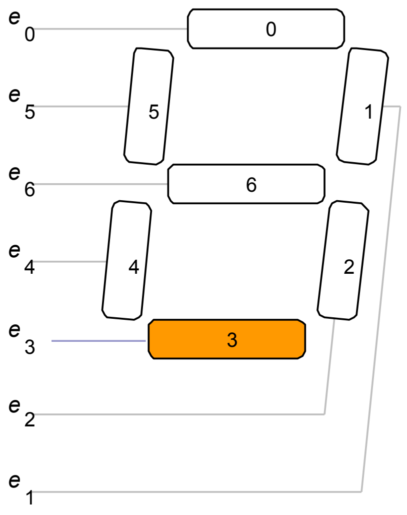
Reading/writing data matrices

- direct addressing - e.g. 7 segment, CRT
- shift registers - e.g. CCD
- row/column - e.g. LCD, memories



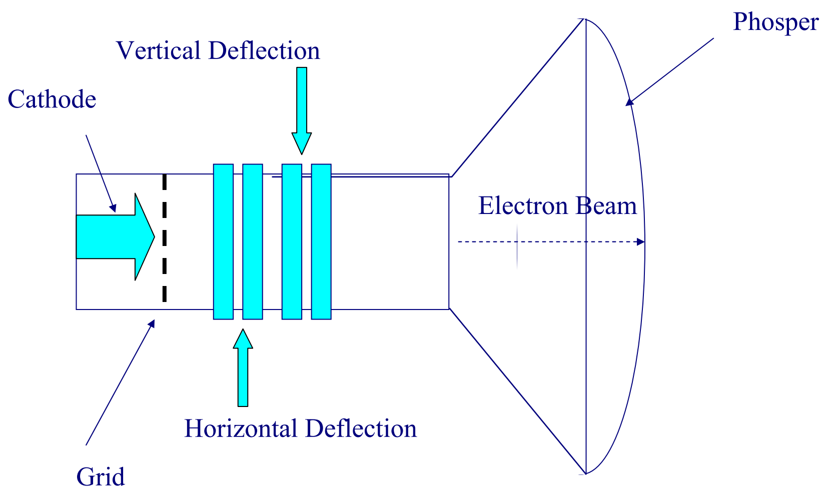
Method-1:Direct addressing

Pixel are connected by direct wires

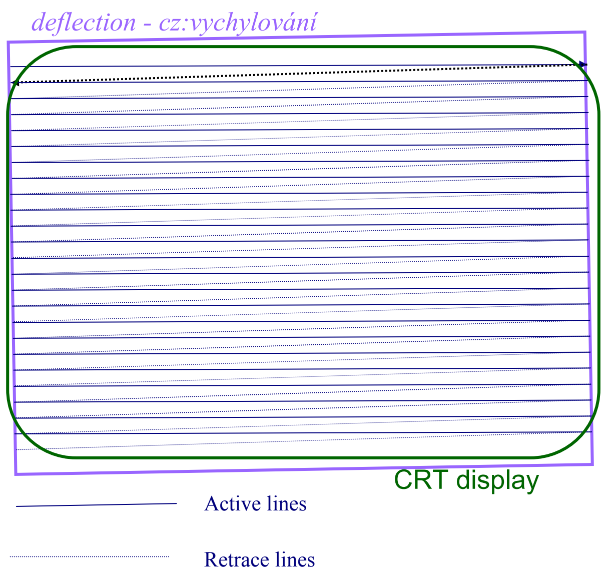


Method-1:Cathode Ray Tubes

- Direct addressing by deflected electron bean



Raster Scan



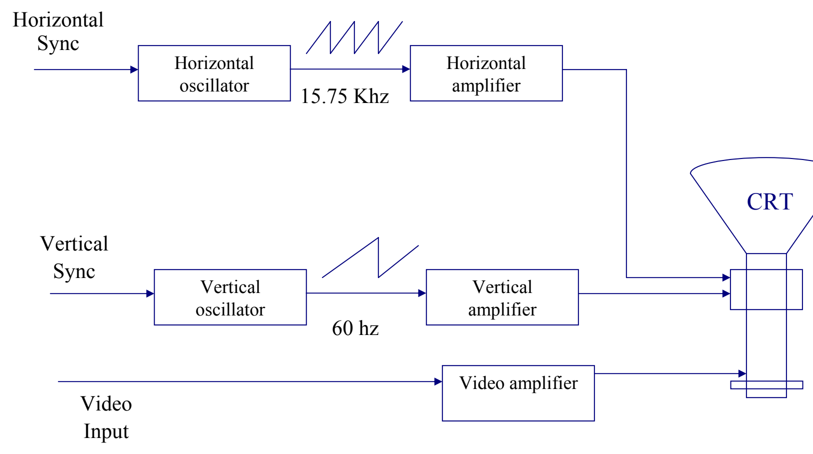
The beam is off during retrace.

***Horizontal* - beam returns to left edge**

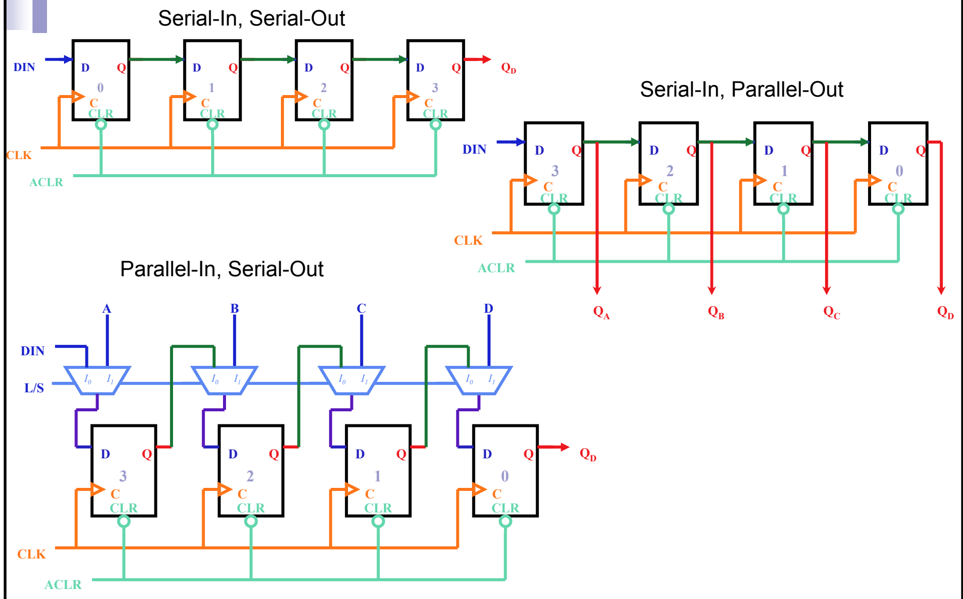
***Vertical* - beam returns to upper left corner.**

Interlaced monitor scans odd lines in odd pictures and even lines in even pictures

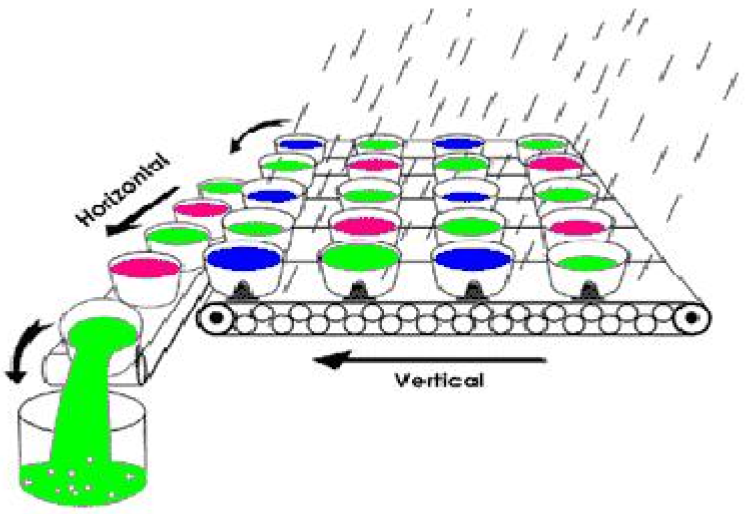
Horizontal, Vertical Syncs



Serializace posuvnými registry

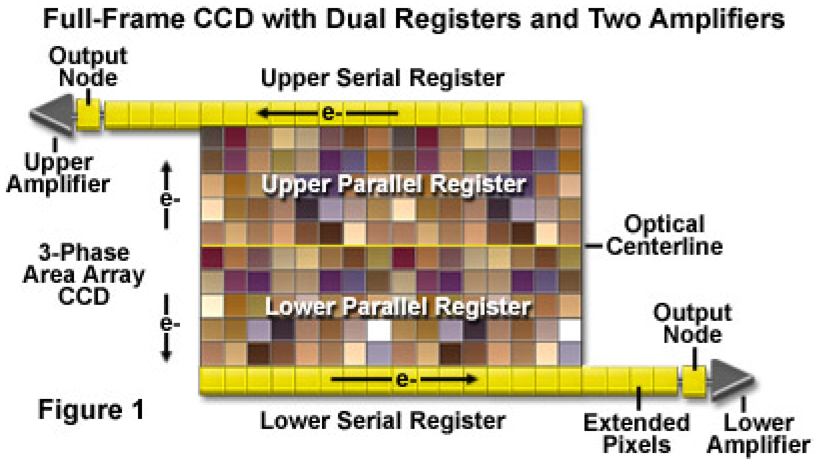


Method-2:Shift Registers - CCD Readout 1/3



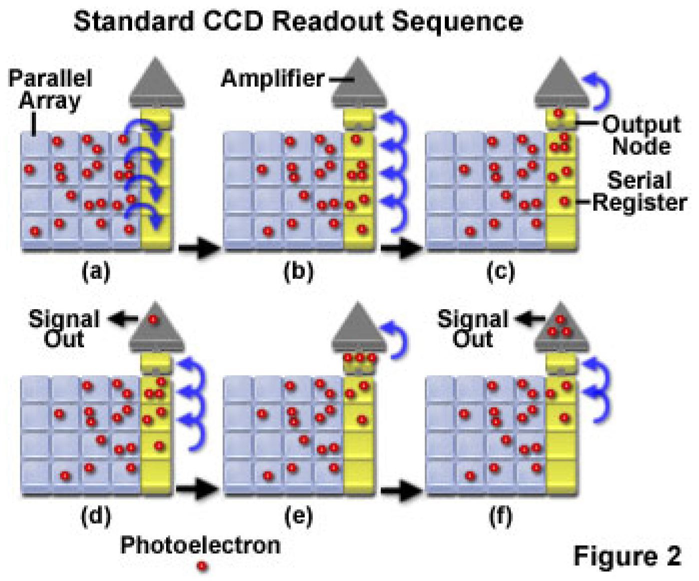
[Image source: Digital Photography Review]

Method-2:Shift Registers - CCD Readout 2/3



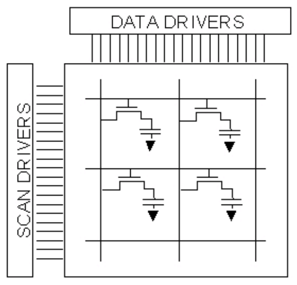
[Image source: Digital Photography Review]

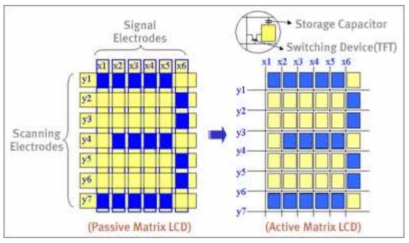
Method-2:Shift Registers - CCD Readout 3/3

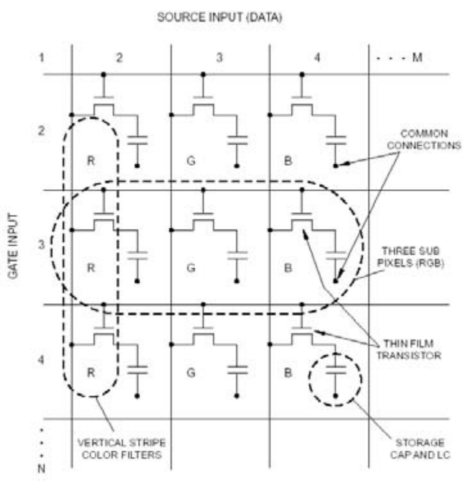


[Image source: Digital Photography Review]

Method-3:Row-Column





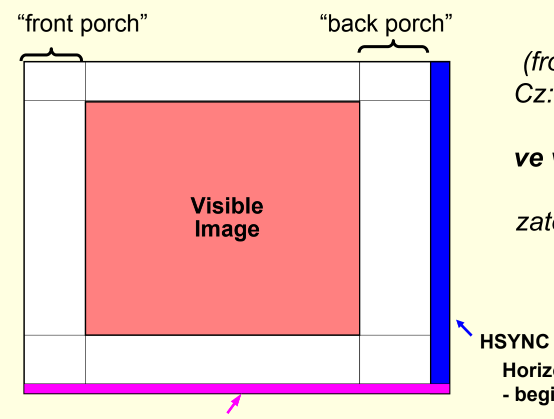


e.g. LCD matrices, memories

[Picture from: LCDs conceptos, Sony]

Introduced by IBM 1987

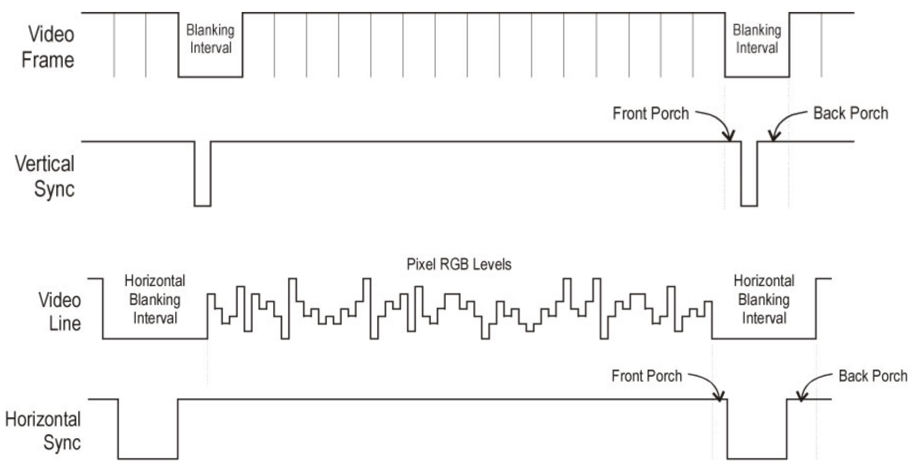
Video Graphic Array (VGA) Signal

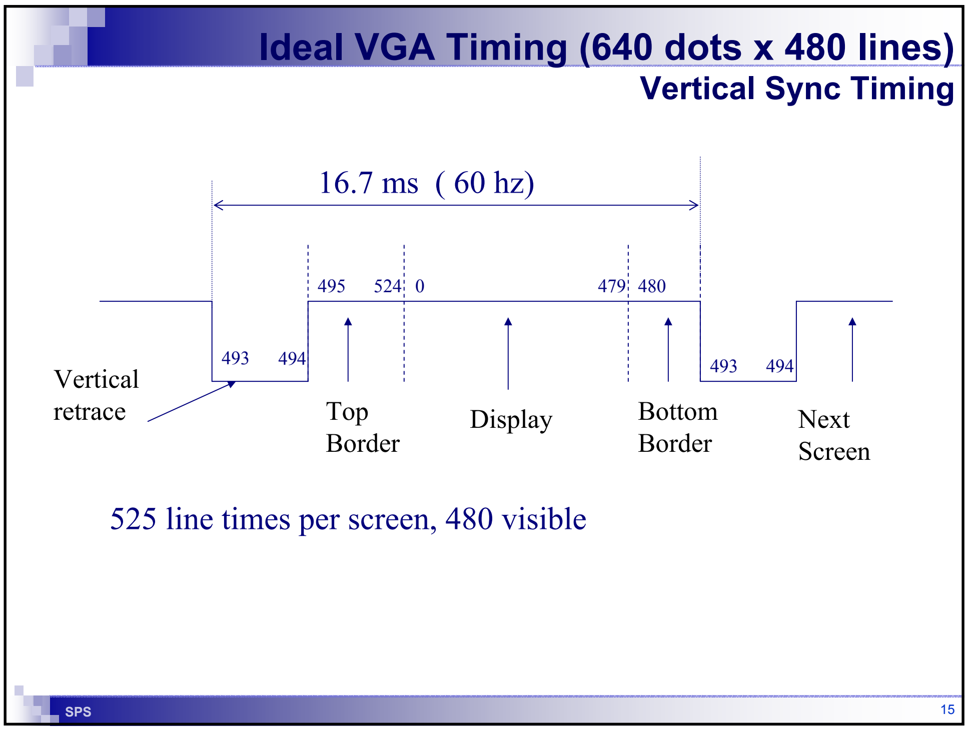


VSYNC - Vertical Synchronization- beginning of frames

porch
(from latin "porta" = gate)
Cz:krytý vchod, přístřešek, veranda
ve video: *front/back porch přední/zadní doba zatemnění - poskytuje čas pro inicializaci*

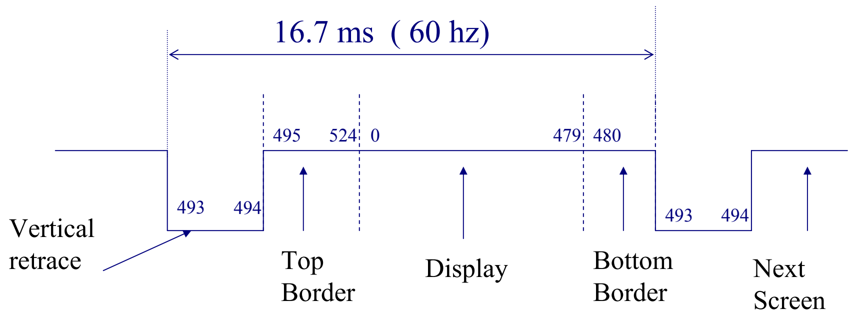
VGA signal



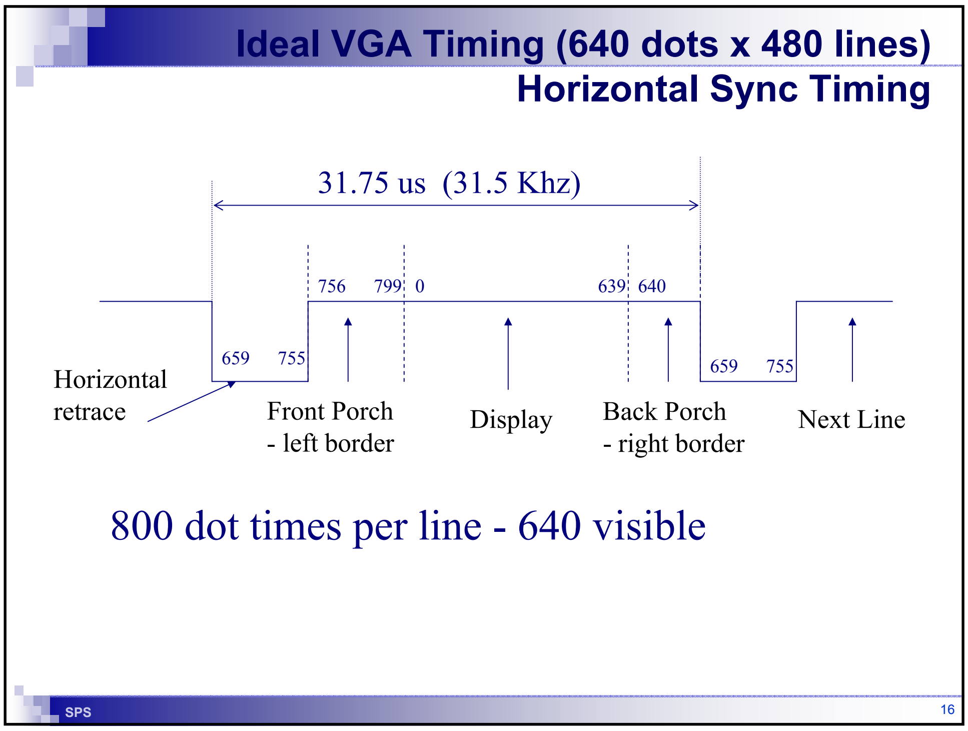


Ideal VGA Timing (640 dots x 480 lines)

Vertical Sync Timing

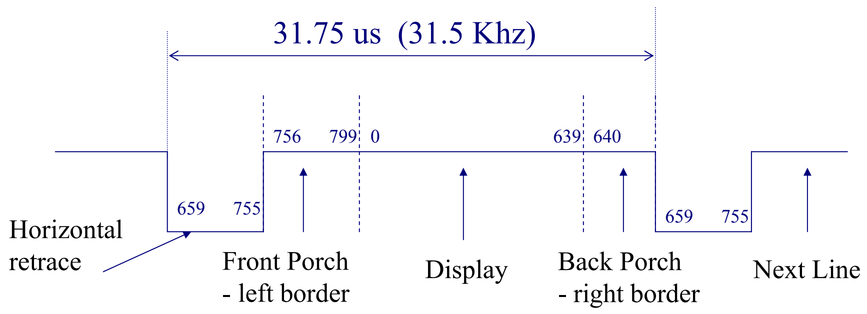


525 line times per screen, 480 visible

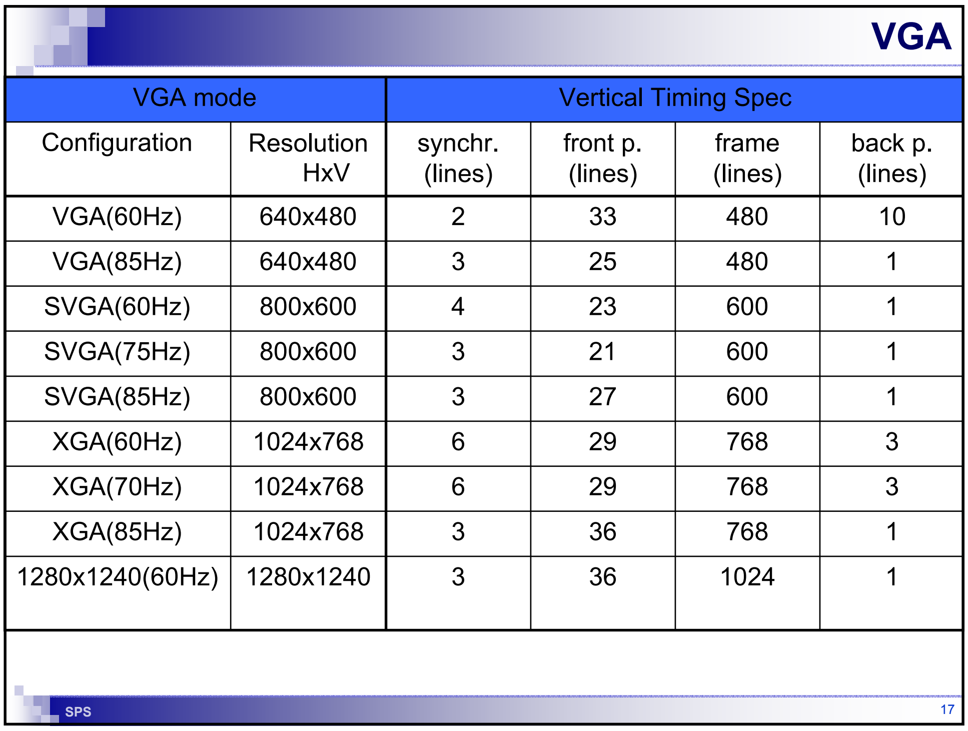


Ideal VGA Timing (640 dots x 480 lines)

Horizontal Sync Timing

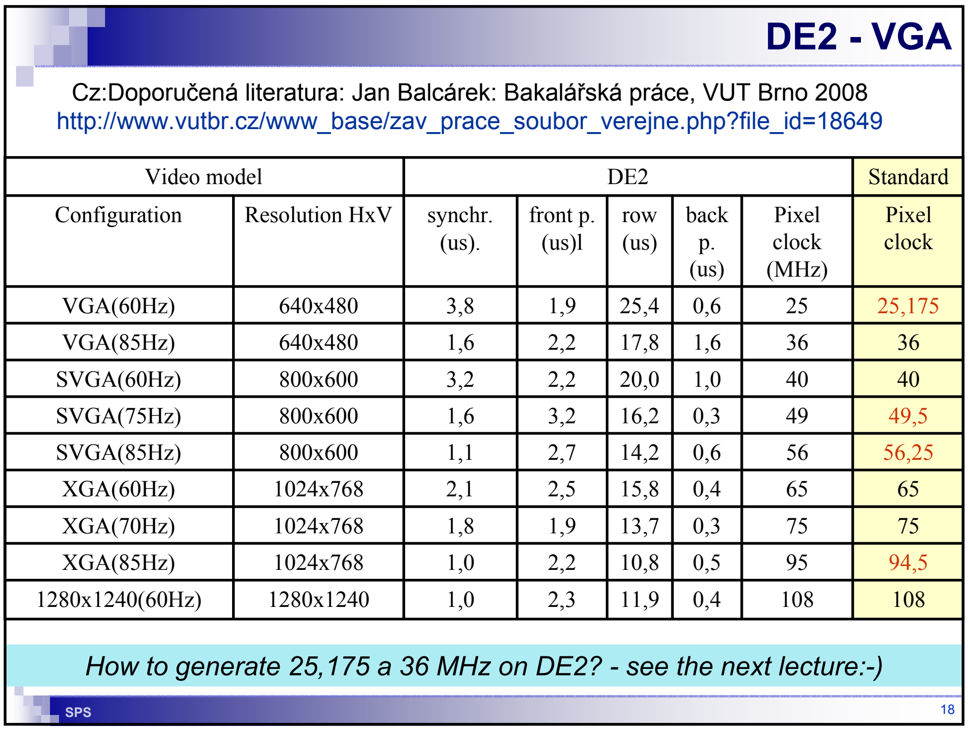


800 dot times per line - 640 visible



VGA

VGA mode		Vertical Timing Spec			
Configuration	Resolution HxV	synchr. (lines)	front p. (lines)	frame (lines)	back p. (lines)
VGA(60Hz)	640x480	2	33	480	10
VGA(85Hz)	640x480	3	25	480	1
SVGA(60Hz)	800x600	4	23	600	1
SVGA(75Hz)	800x600	3	21	600	1
SVGA(85Hz)	800x600	3	27	600	1
XGA(60Hz)	1024x768	6	29	768	3
XGA(70Hz)	1024x768	6	29	768	3
XGA(85Hz)	1024x768	3	36	768	1
1280x1240(60Hz)	1280x1240	3	36	1024	1



DE2 - VGA

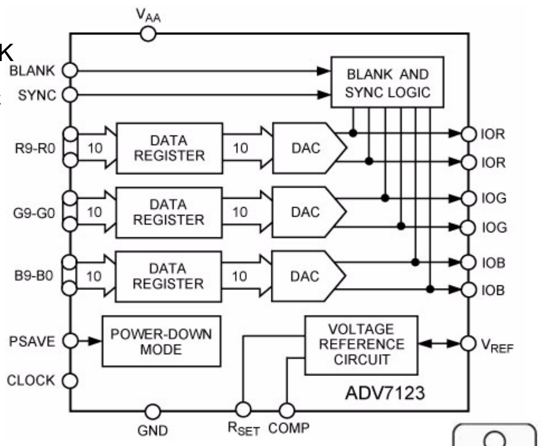
Cz:Doporučená literatura: Jan Balcárek: Bakalářská práce, VUT Brno 2008
http://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=18649

Video model		DE2					Standard
Configuration	Resolution HxV	synchr. (us).	front p. (us)l	row (us)	back p. (us)	Pixel clock (MHz)	Pixel clock
VGA(60Hz)	640x480	3,8	1,9	25,4	0,6	25	25,175
VGA(85Hz)	640x480	1,6	2,2	17,8	1,6	36	36
SVGA(60Hz)	800x600	3,2	2,2	20,0	1,0	40	40
SVGA(75Hz)	800x600	1,6	3,2	16,2	0,3	49	49,5
SVGA(85Hz)	800x600	1,1	2,7	14,2	0,6	56	56,25
XGA(60Hz)	1024x768	2,1	2,5	15,8	0,4	65	65
XGA(70Hz)	1024x768	1,8	1,9	13,7	0,3	75	75
XGA(85Hz)	1024x768	1,0	2,2	10,8	0,5	95	94,5
1280x1240(60Hz)	1280x1240	1,0	2,3	11,9	0,4	108	108

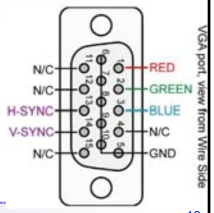
How to generate 25,175 a 36 MHz on DE2? - see the next lecture:-)

DE2 ADV7173

VGA BLANK	VGA_BLANK
VGA SYNC	VGA_SYNC
10-bit Red level	VGA_R[9:0]
10-bit Green level	VGA_G[9:0]
10-bit Blue level	VGA_B[9:0]
VGA Clock	VGA_CLK



VGA H_SYNC	VGA_HS
VGA V_SYNC	VGA_VS



DVI versus VGA

DVI-D

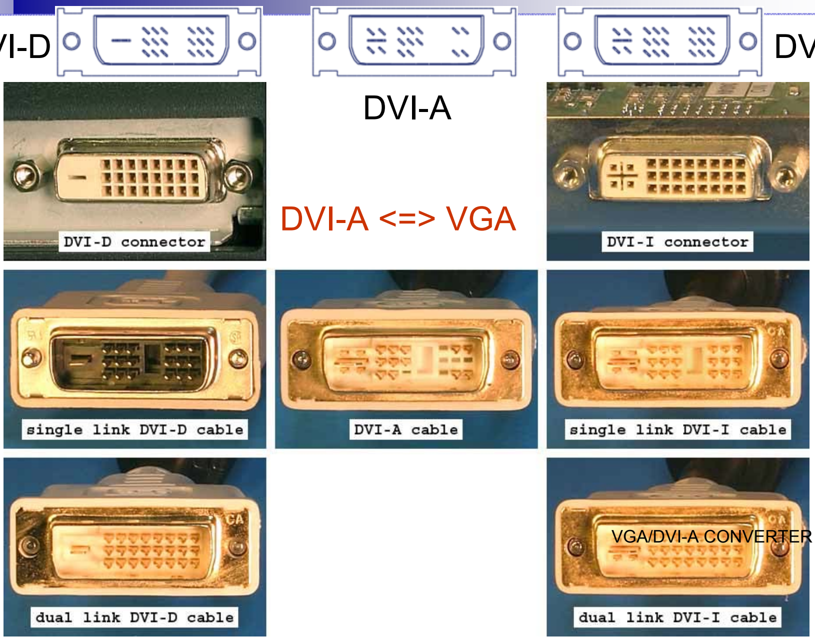
DVI-A

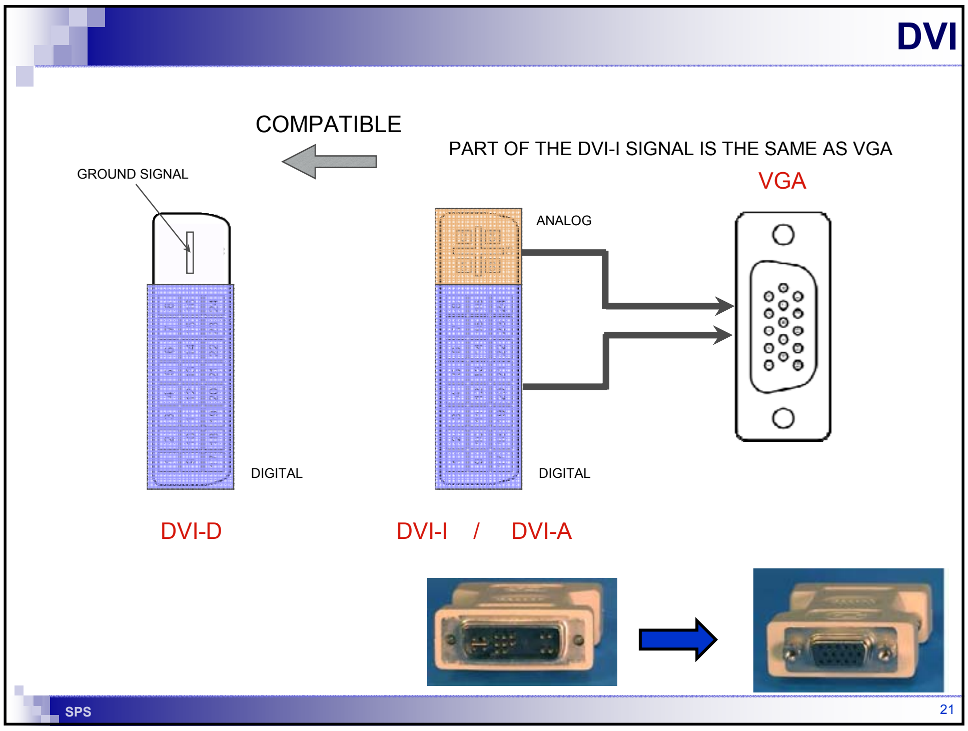
DVI-I

DVI-A <=> VGA

pixels are sent as dot streams

pixels are sent as analog values





DVI

VHDL

jazyk logických obvodů

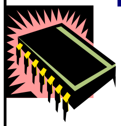
VHDL Jazyk

- Grafická schémata nedovolují snadno porovnávat verze návrhu, hledat změny
- **Hardware Description Language** (HDL)
  - Programovací jazyk pro modelování, simulaci a syntézu číslicových obvodů a systémů.
- Historie
  - 1980: US Department of Defense
    Very High Speed Integrated Circuit program (VHSIC)
  - 1987: Institute of Electrical and Electronics Engineers
    - IEEE Standard 1076 (VHDL'87)
  - 1993: VHDL revize a update
- **Verilog** - další významný HDL jazyk
  - podobný C, používaný hlavně v Americe a Japonsku
- Skoro všechny vývojové nástroje povolují Verilog i VHDL
- Programátoři obvodů se časem musí naučit oba jazyky
- **České firmy a školy používají primárně VHDL**

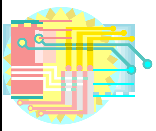
VHDL vlastnosti/omezení...

- ***VDHL nerozlišuje malá a velká písmena***
- ***Používané implementace většinou nerozumí česky, nelze tedy psát s diakritikou, a to ani komentáře.***
- ***Silně typový jazyk s velmi omezenými implicitními konverzemi***
- ***Silně upovídaný jazyk***
- ***Silně neprogramátorský jazyk***
  – sestavili ho návrháři obvodů, nikoliv programátoři
  – píše se v stylem vhodným pro obvody a simulace
- ***Každý obvod může být popsán pomocí VHDL a znovu použitý pro syntézu složitějších obvodů ve VHDL nebo symbolickém editoru***

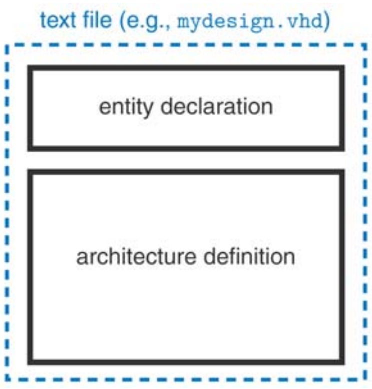
Entity / Architecture



- Entity
  - Component interface
    - Inputs
    - Outputs

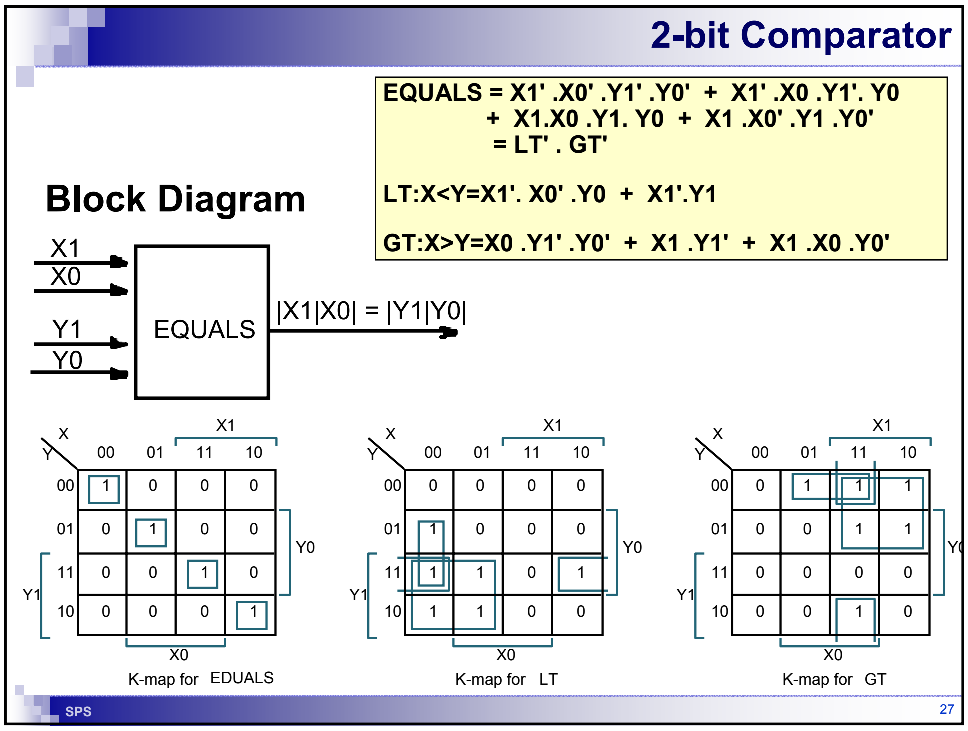


- Architecture
  - Structural
    - Non-procedural dataflow
  - Behavioral
    - Non-procedural dataflow
    - Procedural



Terminologie

- Structural modeling
  - Komponenta je popsaná svým složením z primitiv
- Behavioral modeling
  - Popisuje funkčnost, dynamické chování, ale neobsahuje implementaci...
  - ...překladač se pokusí najít možnou implementaci, výsledek bude - optimální / neoptimální / nelze zapojit ???
- Styl - Register Transfer Level – RTL
  - Styl návrhu v "behavior modeling"
  - Rozložení obvodu na menší jednotky
  - Používají se synchronní registry, hodiny...
  - ...při návrhu RTL myslíme v hardwaru.
  - Jedině RTL lze syntetizovat –> vše ostatní se na RTL převádí



2-bit Comparator

EQUALS = X1' .X0' .Y1' .Y0' + X1' .X0 .Y1'. Y0
+ X1.X0 .Y1. Y0 + X1 .X0' .Y1 .Y0'
= LT' . GT'

LT:X<Y=X1'. X0' .Y0 + X1'.Y1

GT:X>Y=X0 .Y1' .Y0' + X1 .Y1' + X1 .X0 .Y0'

Block Diagram

X1
X0
Y1
Y0
EQUALS
|X1|X0| = |Y1|Y0|

K-map for EDUALS

Y \ X	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

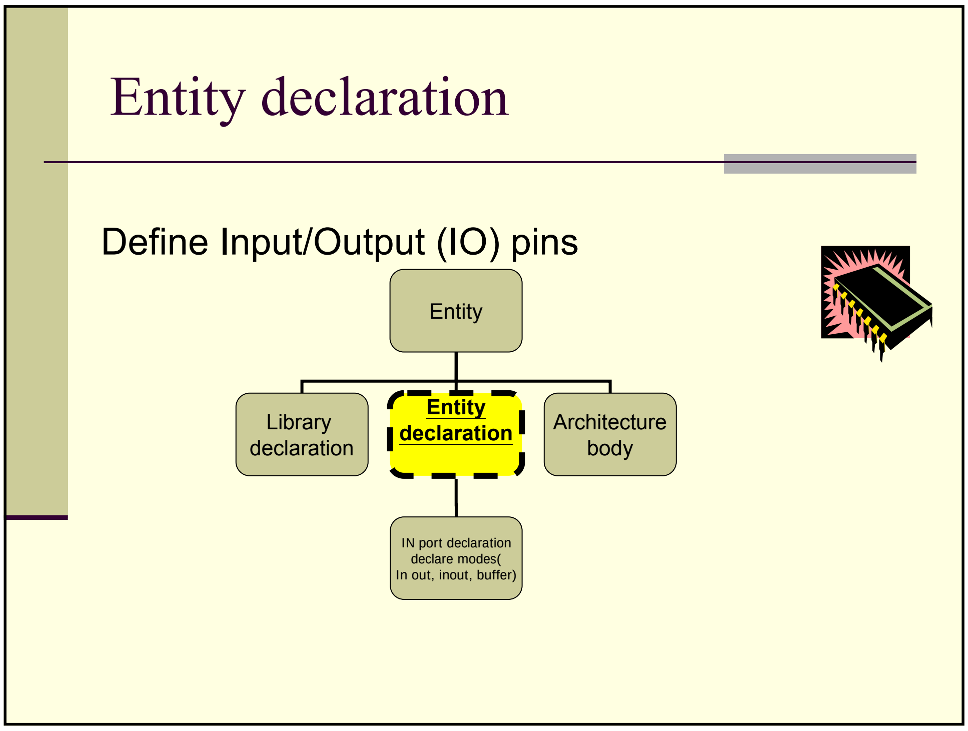
K-map for LT

Y \ X	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

K-map for GT

Y \ X	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	1	0

SPS 27



Entity declaration

Define Input/Output (IO) pins

Entity
Library declaration
Entity declaration
Architecture body
IN port declaration declare modes(In out, inout, buffer)

Entity

```
entity entity-name is
   port (signal-names  :  mode  signal-type;
         signal-names  :  mode  signal-type;
         . . .

         signal-names  :  mode  signal-type);
end entity-name;
```

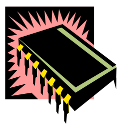
pozor, před) není už středník
jeho napsání je chybou!

Entity defines IO pins

Entity enclosed by the entity name – ***eq2*** (entered by the user)

Case insensitive code!

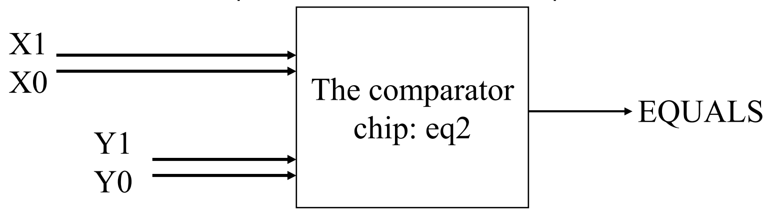
```
entity eq2 is
port (x1, x0, y1, y0:   in std_logic;
         equals:  out std_logic);
end eq2;
```

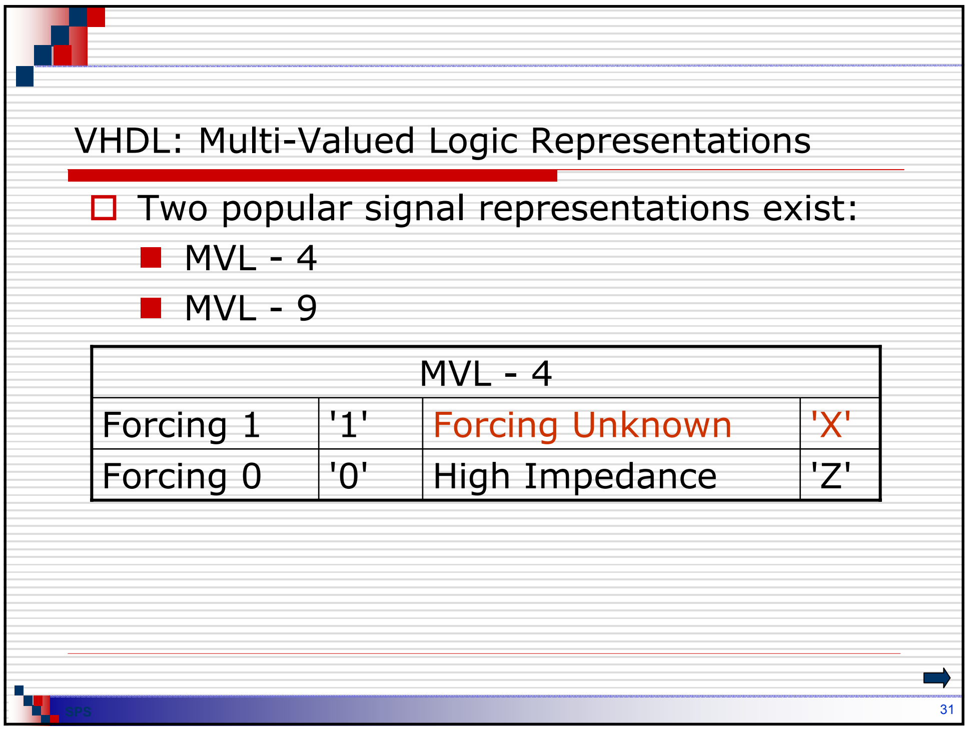


Port defines the I/O pins.

(Standard logic, an IEEE standard)

4 inputs X0, X1, Y1, Y0 and 1 output EQUALS

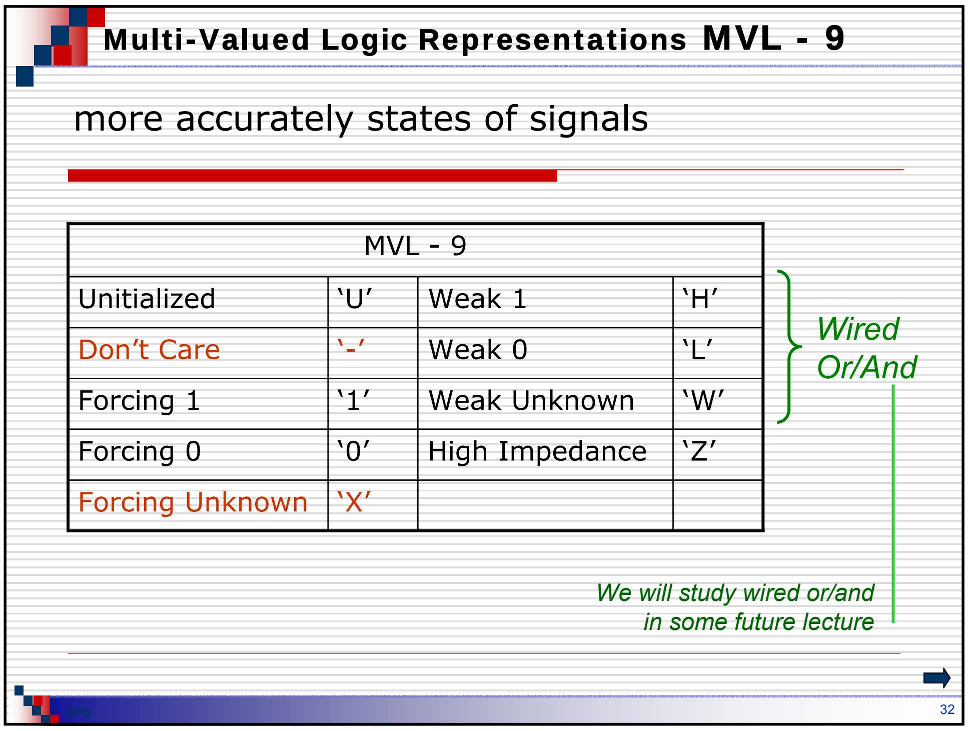




VHDL: Multi-Valued Logic Representations

- Two popular signal representations exist:
  - MVL - 4
  - MVL - 9

MVL - 4			
Forcing 1	'1'	Forcing Unknown	'X'
Forcing 0	'0'	High Impedance	'Z'



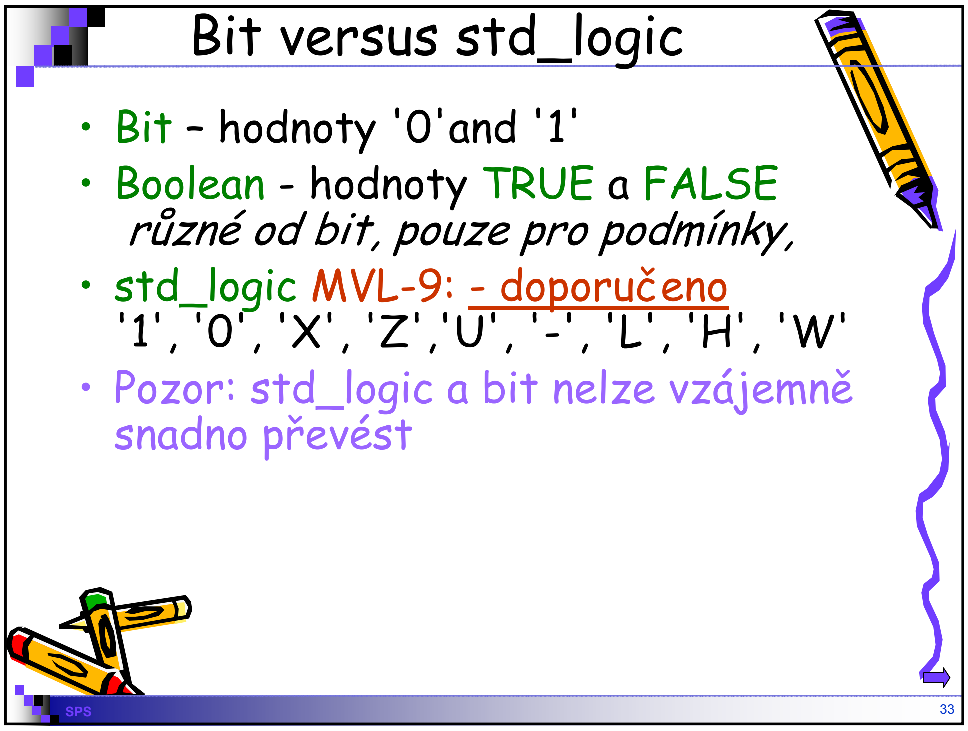
Multi-Valued Logic Representations MVL - 9

more accurately states of signals

MVL - 9			
Unitialized	‘U’	Weak 1	‘H’
Don’t Care	‘-’	Weak 0	‘L’
Forcing 1	‘1’	Weak Unknown	‘W’
Forcing 0	‘0’	High Impedance	‘Z’
Forcing Unknown	‘X’		

Wired Or/And

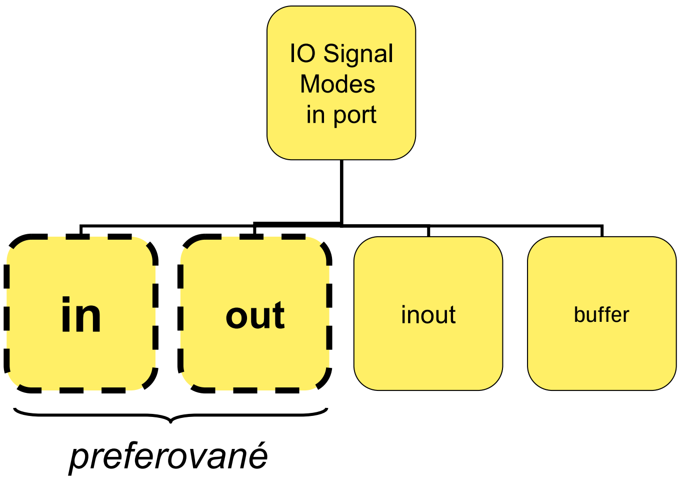
We will study wired or/and in some future lecture



Bit versus std_logic

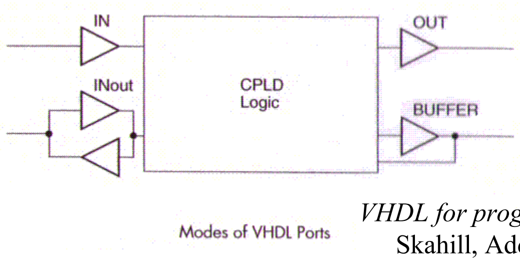
- Bit – hodnoty '0'and '1'
- Boolean - hodnoty TRUE a FALSE
  různé od bit, pouze pro podmínky,
- std_logic MVL-9: - doporučeno
  '1', '0', 'X', 'Z','U', '-', 'L', 'H', 'W'
- Pozor: std_logic a bit nelze vzájemně snadno převést

4 typy IO signalů v deklaraci port



IN, OUT, INOUT, BUFFER modes

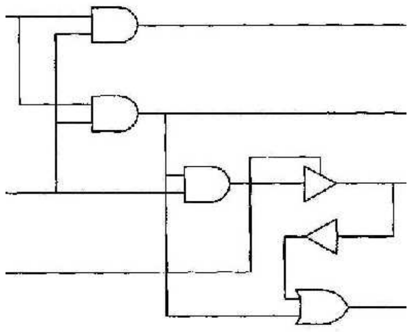
- IN: data proudí pouze dovnitř - vstup
- OUT: data vystupují ven, pouze výstup.
  Entita nemůže číst výstupy
- INOUT: obousměrný pro datové linky.
- BUFFER: podobný OUT, **ale entita smí číst zpět data**. Používáno pro adresy a řídicí signály CPU.



VHDL for programmable logic, Skahill, Addison Wesley

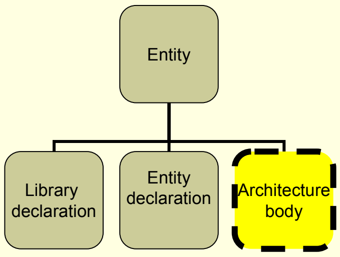
Miniotázky: IN, OUT, INOUT, BUFFER

- Identifikujte módy IO pins ve schématu



VHDL for programmable logic, Skahill, Addison Wesley

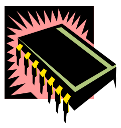
The architecture body



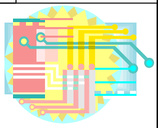
Define the internal architecture/operation

Example of a Comparator

```
entity eq2 is
  port (x1, x0, y1, y0: in std_logic;
        equals: out std_logic   );
end eq2;
```



```
architecture dataflow1 of eq2 is
begin
      equals <= '1' when (x1 = y1) and (x0 = y0) else '0';
   -- "comment" equals is active high
end dataflow1;
```



Jak číst?

```
architecture dataflow1 of eq2 is
 begin
            equals <= '1' when (x1 = y1) and (x0 = y0) else '0';
   -- "comment" equals is active high
 end dataflow1;
```

Operaci: *equals* <= '*1*' when (*x1* = *y1*) and (*x0* = *y0*) else '*0*';
čteme jako: *Equals pin gets value* '*1*'
when (*x1* = *y1*) and (*x0* = *y0*)
else it gets '*0*';
"--" znamená komentář

equals, x1, x0, y1, y0 *jsou "I/O signal pins" deklarované uživatelem*

Relational Operators

- *they have no relative precedences and*
- their results has always type Boolean (TRUE,FALSE)

Symbol	Operator
=	Equal
/=	Not equal
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

equals <= '*1*' when (*x1* = *y1*) and (*x0* = *y0*) else '*0*';

Logical operators

defined for types: ***boolean, std_logic, std_logic_vector*** ***and*** *std_ulogic, std_ulogic_vector, bit, bit_vector*

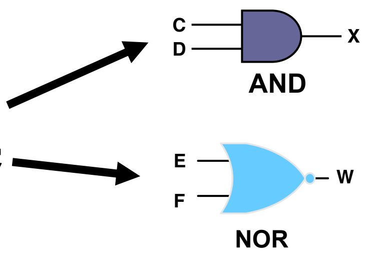
and	*logical and*
or	*logical or*
nand	*logical complement of and*
nor	*logical complement of or*
not	*logical not*
xor	*logical exclusive or*
xnor	*logical complement of exclusive or*

equals <= '*1*' when (*x1* = *y1*) and (*x0* = *y0*) else '*0*';

„Concurrent“ Assigment

(cz:současné, souběžné)

```
Z <= not A;
Y <= A or B;
X <= C and D;
W <= E nor F;
V <= B nand D;
U <= C xor F;
```

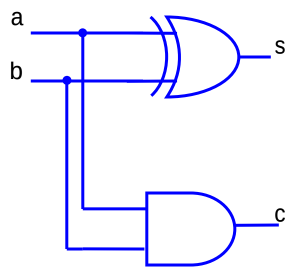


equals <= '*1*' when (*x1* = *y1*) and (*x0* = *y0*) else '*0*';

Concurrent Assignments

Jaké přiřazení se provede dříve ve VHDL programu, to na prvním řádku, nebo na druhém řádku?

```
s <= a xor b;
c <= a and b;
```



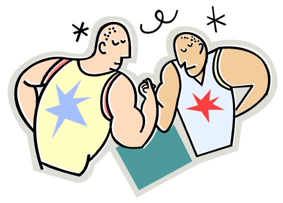
•V implementaci: obě současně!
- **operace v hardwaru probíhají paralelně**!

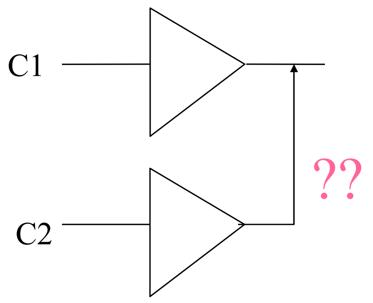
•V simulaci: sice postupně, ale v nespecifikovaném pořadí
– **simulujeme hardware**…

equals <= '*1*' when (*x1* = *y1*) and (*x0* = *y0*) else '*0*';

Can the outputs be connected together?

Sometimes somewhere

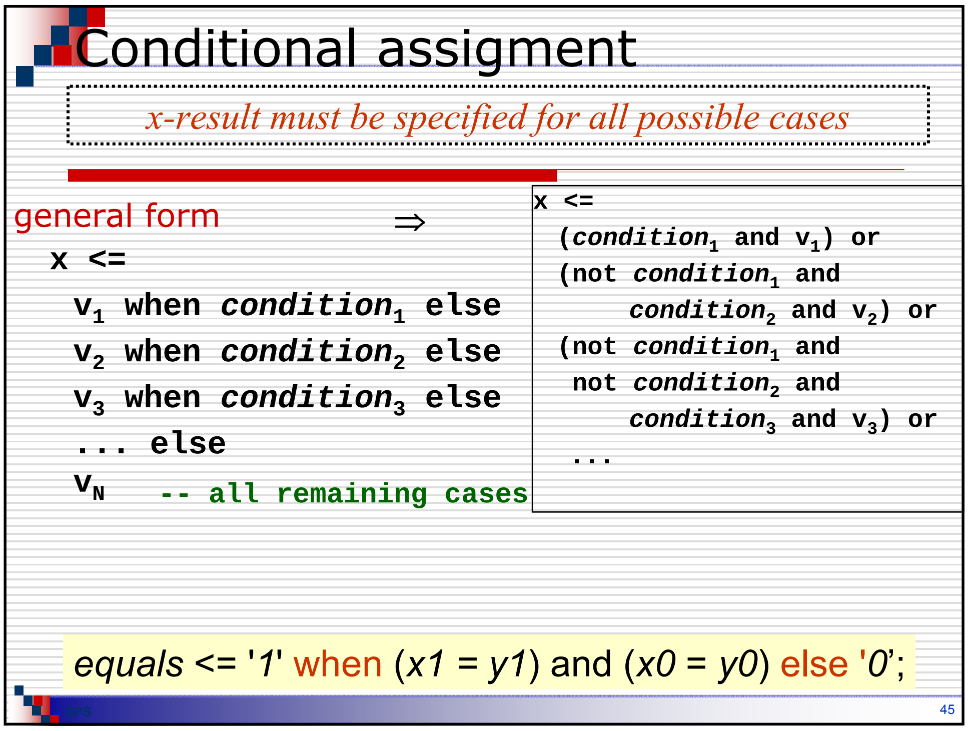




Signal *c1*,*c2*, *b1*: std_logic;

~~*b1<=c1;*~~
~~*b1<=c2;*~~

Cyclon II and majority of other FPGA allow only one assignment each signal in program.



Conditional assigment

x-result must be specified for all possible cases

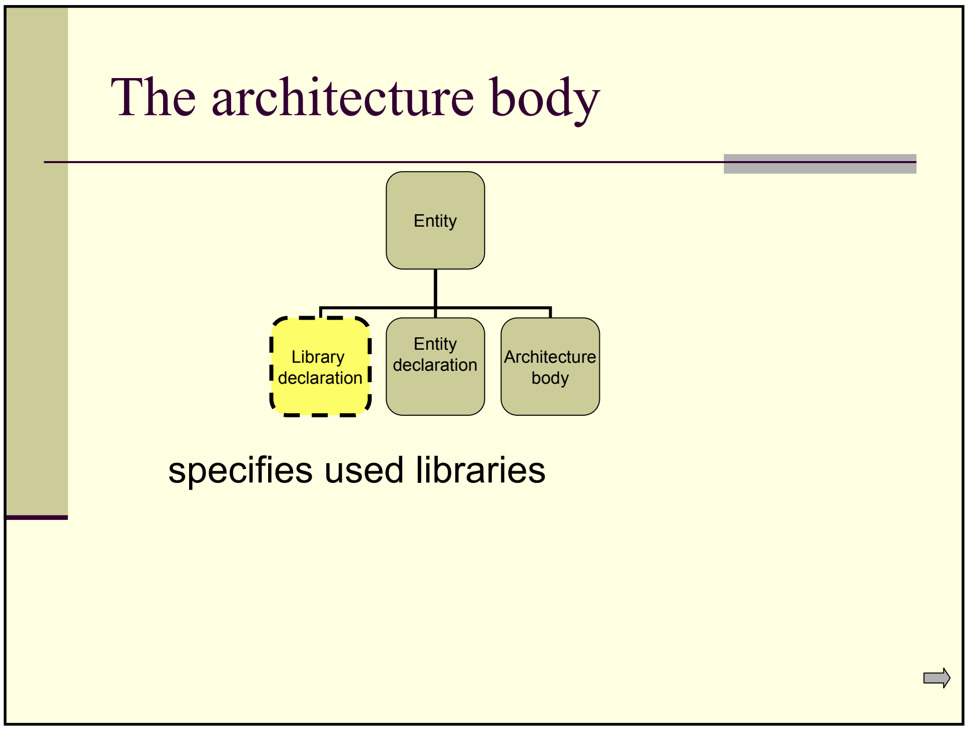
general form ⇒

```
x <=
  v_1 when condition_1 else
  v_2 when condition_2 else
  v_3 when condition_3 else
  ... else
  v_N    -- all remaining cases
```

```
x <=
 (condition_1 and v_1) or
 (not condition_1 and
      condition_2 and v_2) or
 (not condition_1 and
  not condition_2 and
      condition_3 and v_3) or
  ...
```

equals <= '*1*' when (*x1* = *y1*) and (*x0* = *y0*) else '*0*';

45



The architecture body

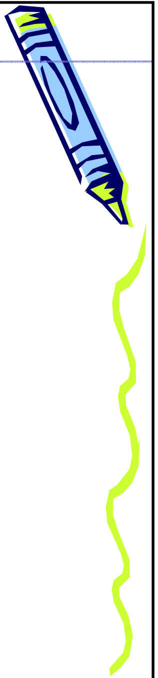
Entity
Library declaration
Entity declaration
Architecture body

specifies used libraries

Library clause

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

entity eq2 is
   port (x1, x0, y1, y0: in std_logic;
          equals: out std_logic   );
  end eq2;

  architecture dataflow1 of eq2 is
  begin
         equals <= '1' when (x1 = y1) and (x0 = y0) else '0';
    -- “comment” equals is active high
  end dataflow1;
```

Mini otázky



```
entity test1 is
port (in1,in2:    in bit;
    out1: out bit;
end test1;

architecture test1arch of test1 is
begin
        out1<= in1 or in2;
end test1_arch;
```

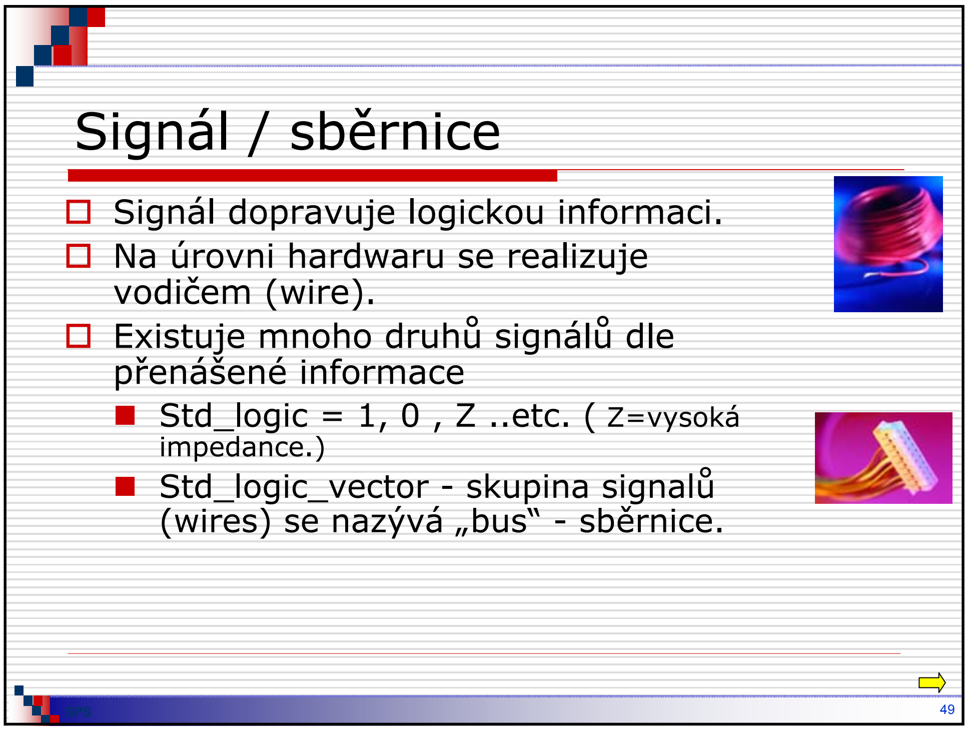
Řekněte, které řádky obsahují deklaraci entity a které architekturu.

Nalezněte chybu v kódu.

Jaká je funkce deklarace a architektury u entity?

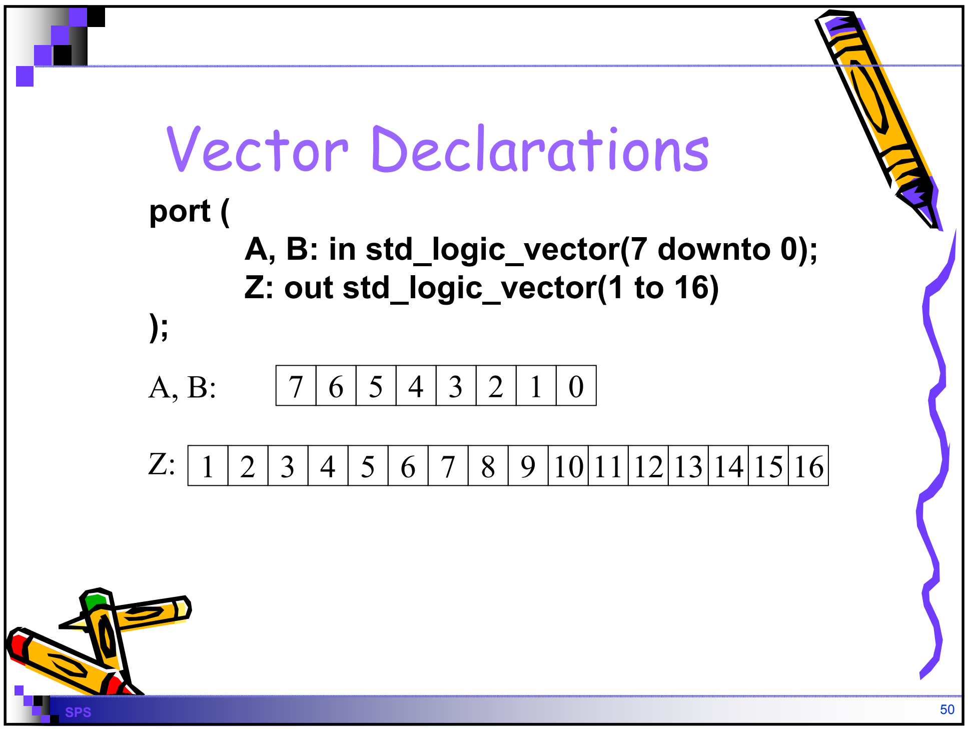
Nakreslete obvod se jmény pinů (pins).

Podtrhněte slova definovaná uživatelem v kódu nahoře.



Signál / sběrnice

- Signál dopravuje logickou informaci.
- Na úrovni hardwaru se realizuje vodičem (wire).
- Existuje mnoho druhů signálů dle přenášené informace
  - Std_logic = 1, 0 , Z ..etc. (Z=vysoká impedance.)
  - Std_logic_vector - skupina signalů (wires) se nazývá „bus“ - sběrnice.



Vector Declarations

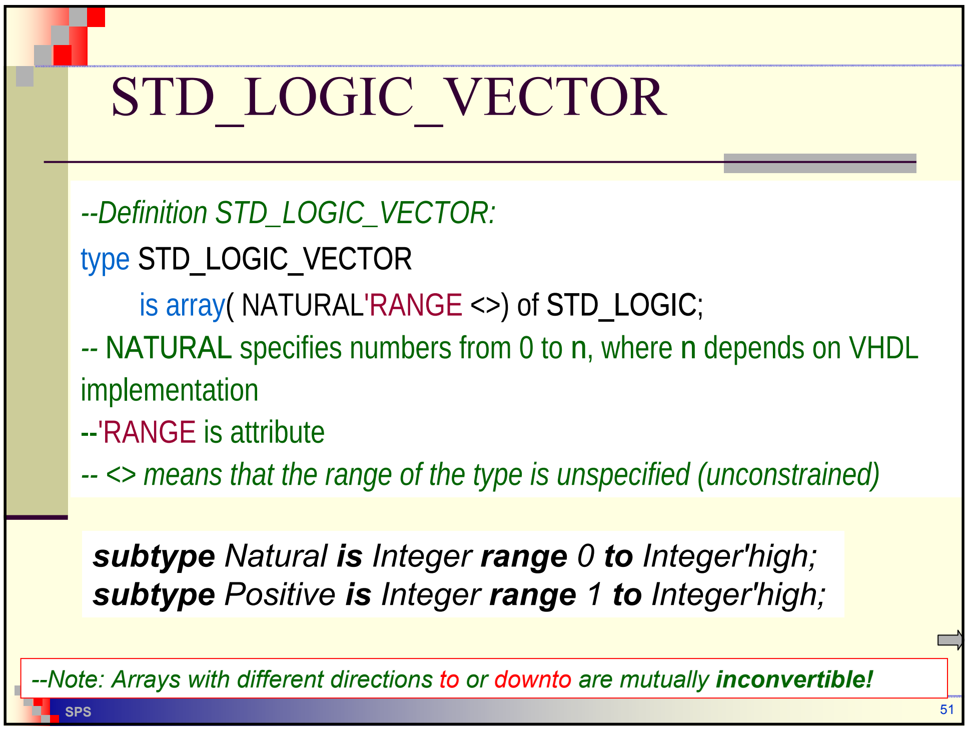
```
port (
        A, B: in std_logic_vector(7 downto 0);
        Z: out std_logic_vector(1 to 16)
);
```

A, B:

7	6	5	4	3	2	1	0

Z:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16



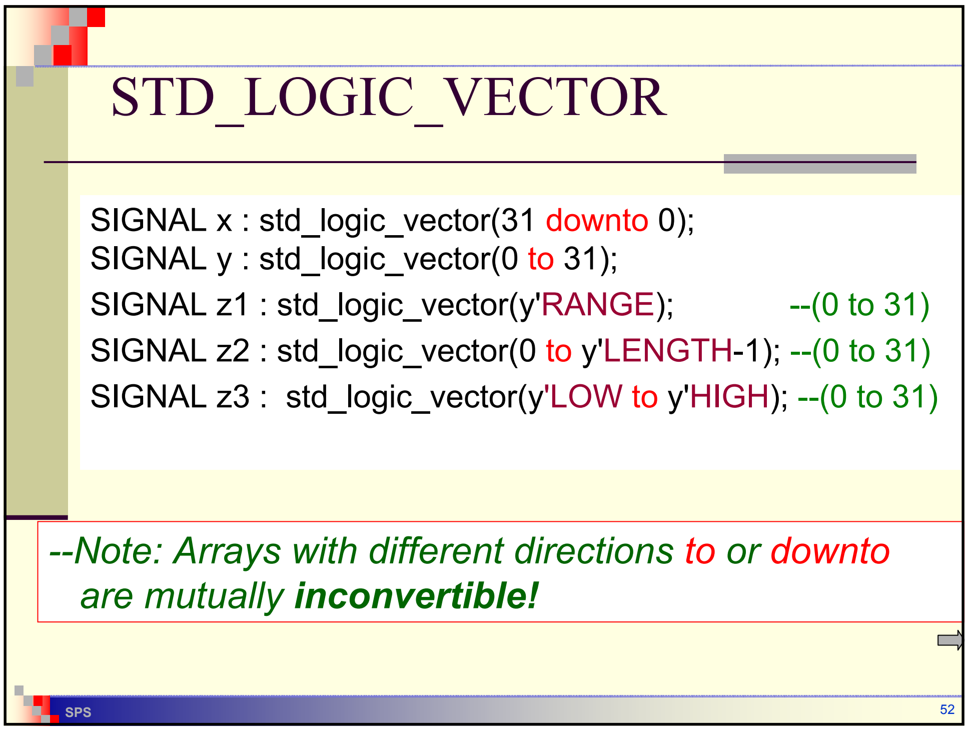
STD_LOGIC_VECTOR

```
--Definition STD_LOGIC_VECTOR:
type STD_LOGIC_VECTOR
      is array( NATURAL'RANGE <>) of STD_LOGIC;
-- NATURAL specifies numbers from 0 to n, where n depends on VHDL
implementation
--'RANGE is attribute
-- <> means that the range of the type is unspecified (unconstrained)
```

***subtype** Natural **is** Integer **range** 0 **to** Integer'high;*
***subtype** Positive **is** Integer **range** 1 **to** Integer'high;*

*--Note: Arrays with different directions to or downto are mutually **inconvertible!***

SPS 51



STD_LOGIC_VECTOR

```
SIGNAL x : std_logic_vector(31 downto 0);
SIGNAL y : std_logic_vector(0 to 31);
SIGNAL z1 : std_logic_vector(y'RANGE);            --(0 to 31)
SIGNAL z2 : std_logic_vector(0 to y'LENGTH-1); --(0 to 31)
SIGNAL z3 :  std_logic_vector(y'LOW to y'HIGH); --(0 to 31)
```

*--Note: Arrays with different directions to or downto are mutually **inconvertible!***

SPS 52

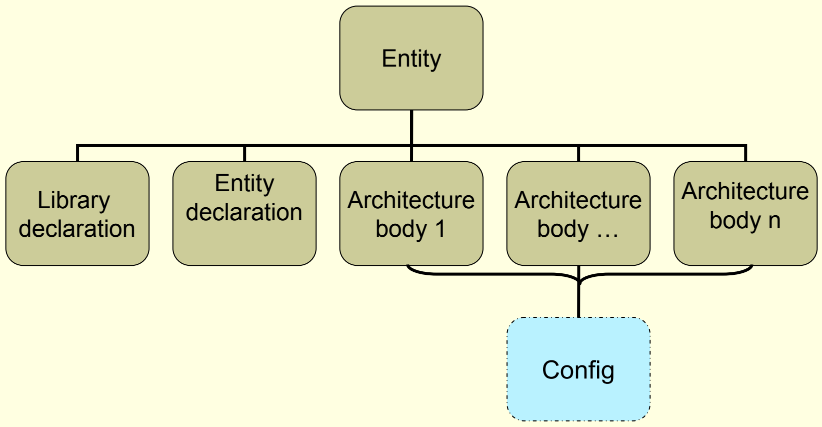
Assignment Statements

```
SIGNAL x, y, z : STD_LOGIC;
SIGNAL a, b, c : STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL sel     : STD_LOGIC_VECTOR(2 DOWNTO 0);

-- Concurrent Signal Assignment Statements
x <= y AND z;  a <= b OR c;

-- Alternatively, signals may be assigned constants
x <= '0';
y <= '1';
z <= 'Z';
a <= "00111010"; -- Assigns 0x3A to a
b <= X"3A"; -- Assigns 0x3A to b
c <= X"3" & X"A"; -- Assigns 0x3A to c
sel <= a(5 DOWNTO 4) & '1';
```

EQ2 with Bus - cz:sběrnice

```
architecture dataflow2 of eq2 is
   signal x,y:std_logic_vector(1 downto 0); -- internal signals
   begin
      x <= x1&x0; y<= y1&y0;    -- concatenate & by downto style
      equals <= '1' when x=y else '0';
end dataflow2;

architecture dataflow3 of eq2 is
   signal x,y:std_logic_vector(0 to 1); -- internal signals
   begin
      x <= x0&x1; y<= y0&y1;    -- concatenate & by to style
      equals <= '1' when x=y else '0';
end dataflow3;
```

Entita a struktura VHDL souboru

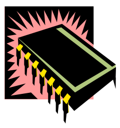


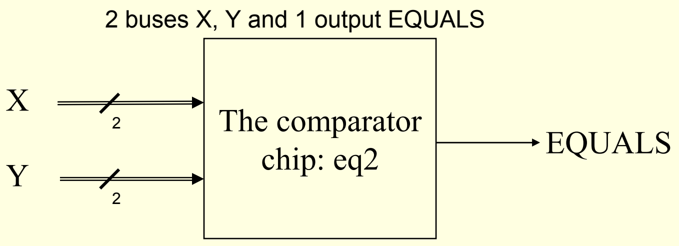
Configuration

```
configuration eq2config of eq2 is
   for dataflow3
   end for;
end eq2config;
```

Entity – bus inputs

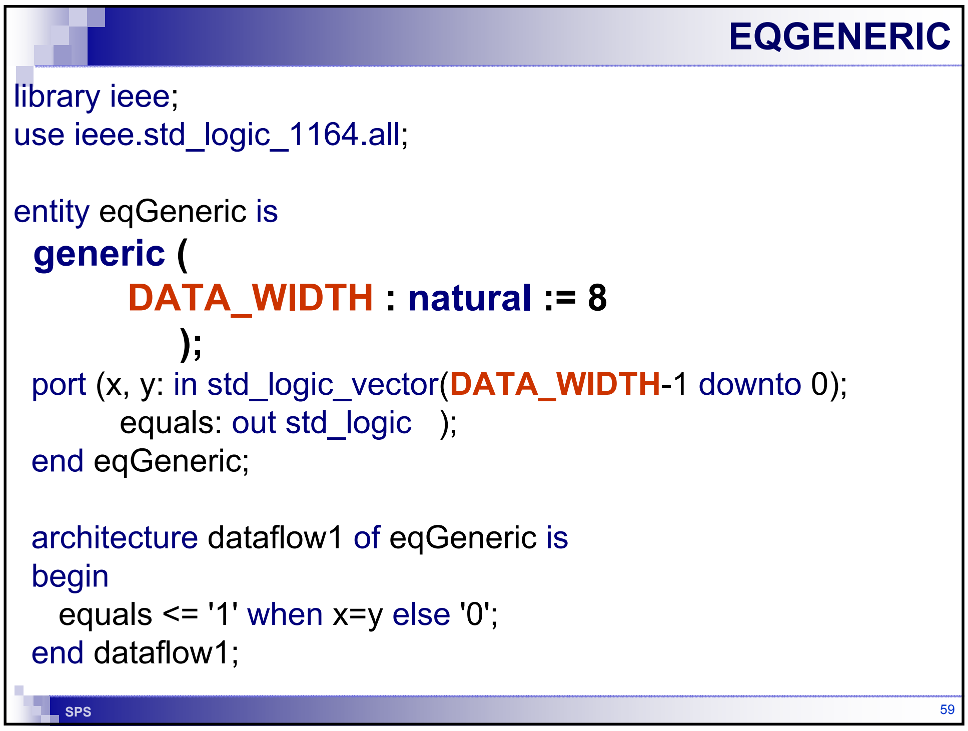
```
entity eq2bus is
port (x, y:      in std_logic_vector(1 downto 0);
        equals:  out std_logic);
end eq2bus;
```





EQ2BUS

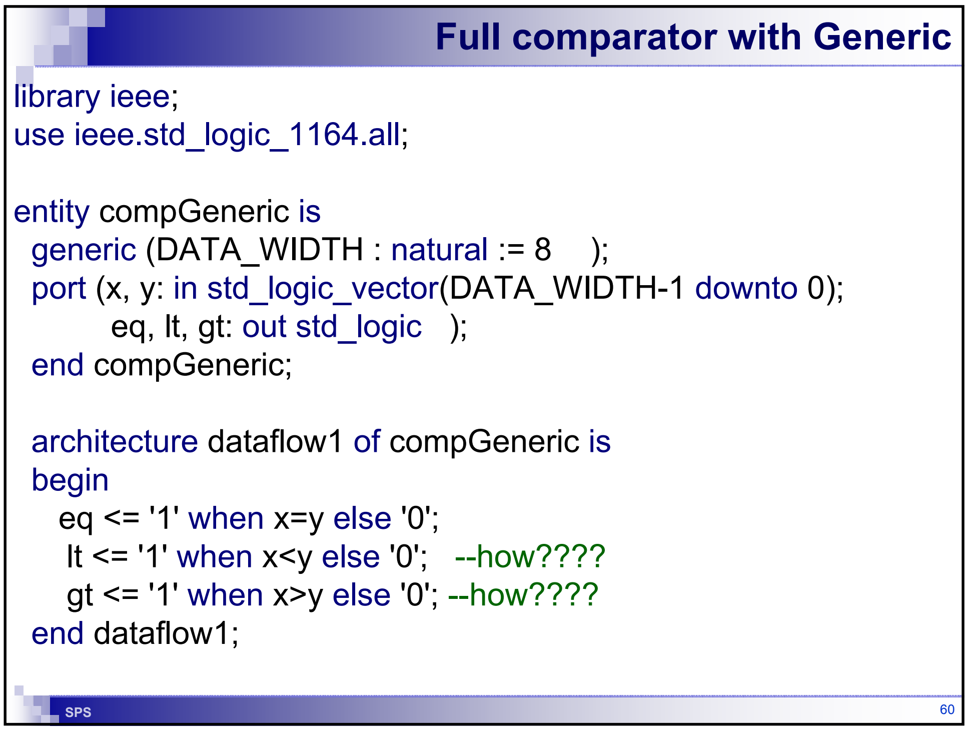
```
library ieee;
use ieee.std_logic_1164.all;

entity eq2bus is
  port (x, y: in std_logic_vector(1 downto 0);
          equals: out std_logic   );
  end eq2bus;

  architecture dataflow1 of eq2bus is
  begin
    equals <= '1' when x=y else '0';
  end dataflow1;
```



EQGENERIC

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

entity eqGeneric is
 generic (
        DATA_WIDTH : natural := 8
             );
 port (x, y: in std_logic_vector(DATA_WIDTH-1 downto 0);
        equals: out std_logic   );
 end eqGeneric;

 architecture dataflow1 of eqGeneric is
 begin
   equals <= '1' when x=y else '0';
 end dataflow1;
```

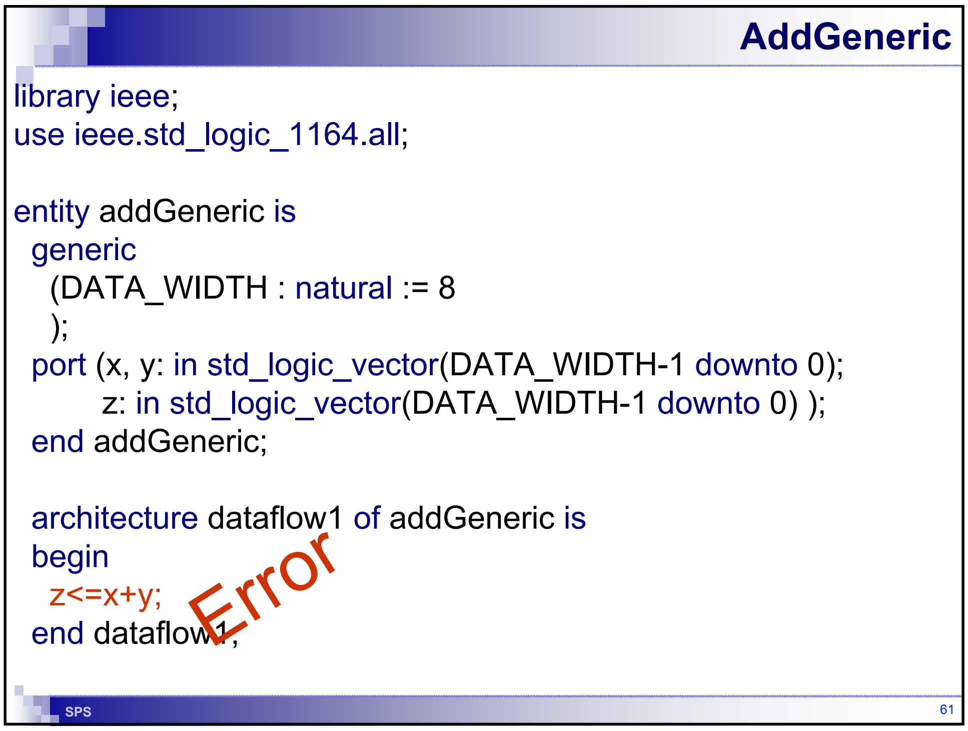
SPS 59



Full comparator with Generic

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

entity compGeneric is
 generic (DATA_WIDTH : natural := 8    );
 port (x, y: in std_logic_vector(DATA_WIDTH-1 downto 0);
        eq, lt, gt: out std_logic    );
 end compGeneric;

 architecture dataflow1 of compGeneric is
 begin
   eq <= '1' when x=y else '0';
    lt <= '1' when x<y else '0';   --how????
    gt <= '1' when x>y else '0'; --how????
 end dataflow1;
```

SPS 60

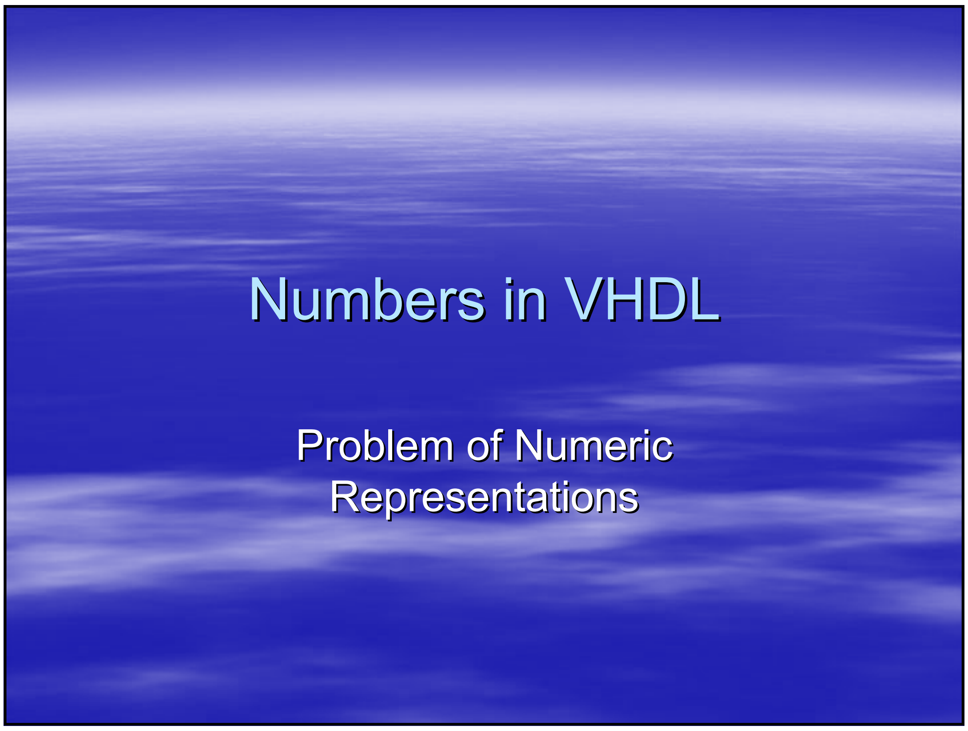


AddGeneric

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

entity addGeneric is
  generic
    (DATA_WIDTH : natural := 8
    );
  port (x, y: in std_logic_vector(DATA_WIDTH-1 downto 0);
        z: in std_logic_vector(DATA_WIDTH-1 downto 0) );
  end addGeneric;

  architecture dataflow1 of addGeneric is
  begin
    z<=x+y;
  end dataflow1;
```

Error

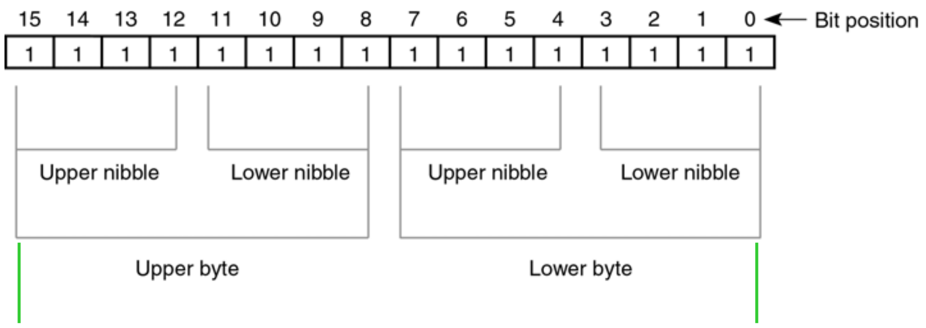
SPS 61



Numbers in VHDL

Problem of Numeric Representations

Bytes, Nibbles, and Bits in 16bit Word

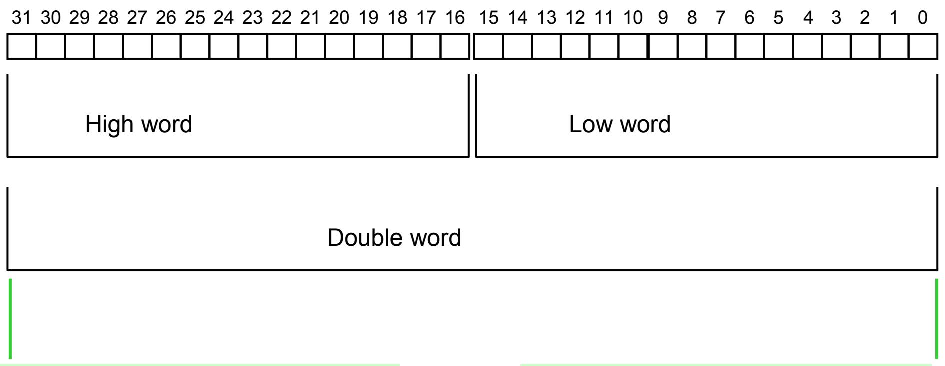


MSB – Most Significant Bit

Least Significant Bit - **LSB**

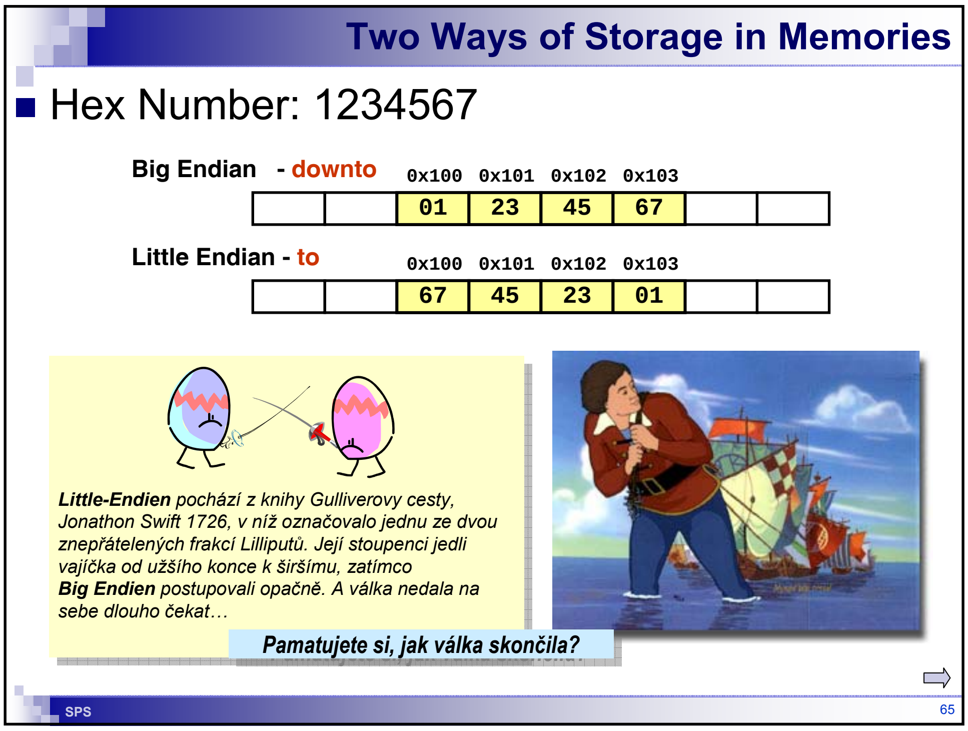
BIT=a BInary digiT

32 bit Word



MSB – Most Significant Bit

Least Significant Bit - **LSB**



Two Ways of Storage in Memories

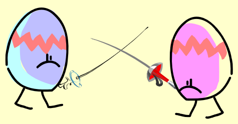
- Hex Number: 1234567

Big Endian - downto

		0x100	0x101	0x102	0x103		
		01	23	45	67		

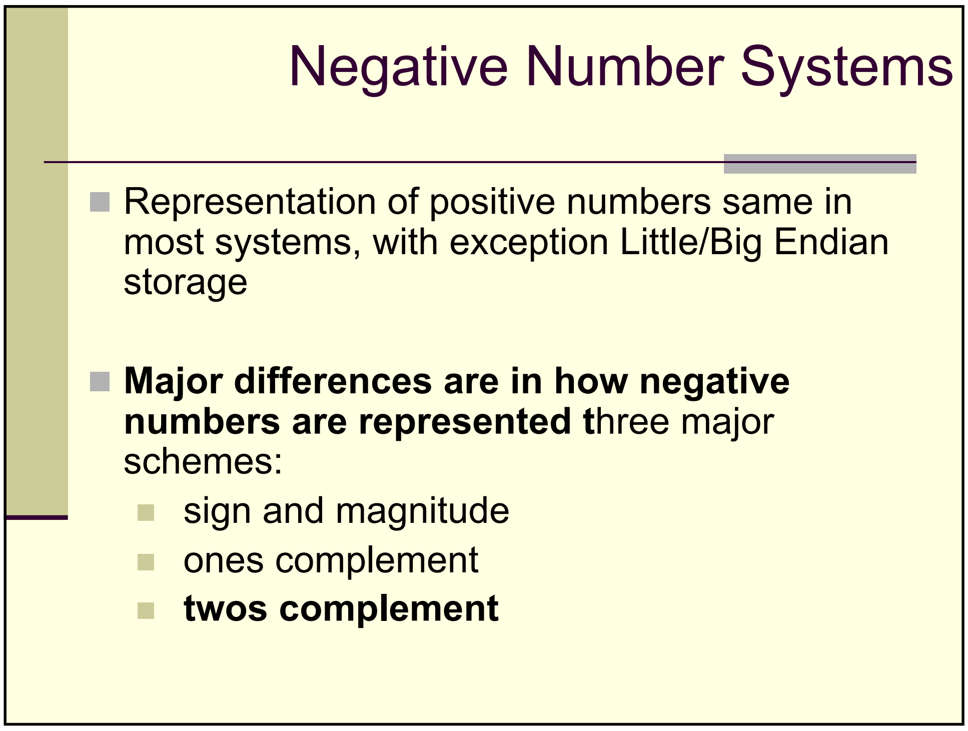
Little Endian - to

		0x100	0x101	0x102	0x103		
		67	45	23	01		



Little-Endien *pochází z knihy Gulliverovy cesty, Jonathon Swift 1726, v níž označovalo jednu ze dvou znepřátelených frakcí Lilliputů. Její stoupenci jedli vajíčka od užšího konce k širšímu, zatímco* ***Big Endien*** *postupovali opačně. A válka nedala na sebe dlouho čekat…*

Pamatujete si, jak válka skončila?



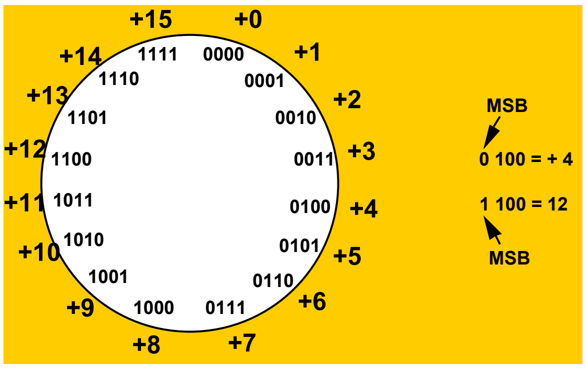
Negative Number Systems

- Representation of positive numbers same in most systems, with exception Little/Big Endian storage
- **Major differences are in how negative numbers are represented t**hree major schemes:
  - sign and magnitude
  - ones complement
  - **twos complement**

Number Systems

Assumptions:we'll assume a 4 bit machine word

Unsigned 4-bit numbers

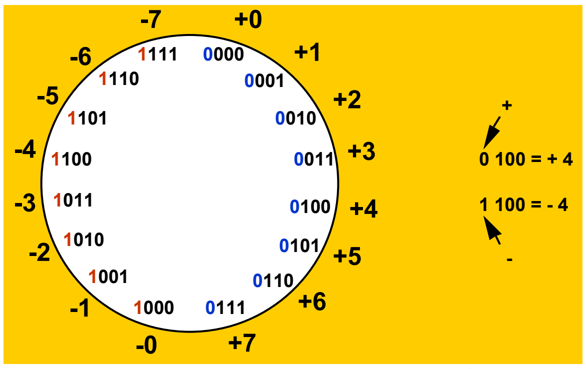


- Cumbersome addition/subtraction

[Seungryoul Maeng:Digital Systems]

Number Systems

Sign and Magnitude Representation

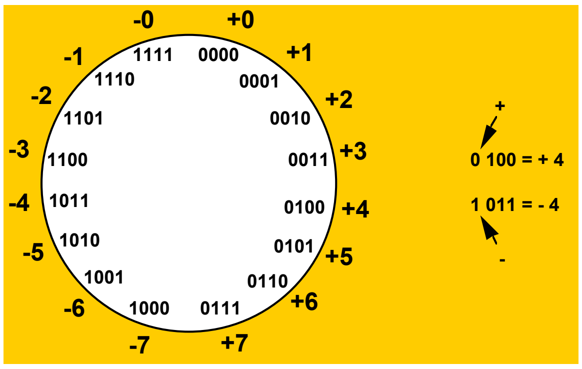


- Cumbersome addition/subtraction
- Sign+Magnitude usually used only for float point numbers

[Seungryoul Maeng:Digital Systems]

Number Systems

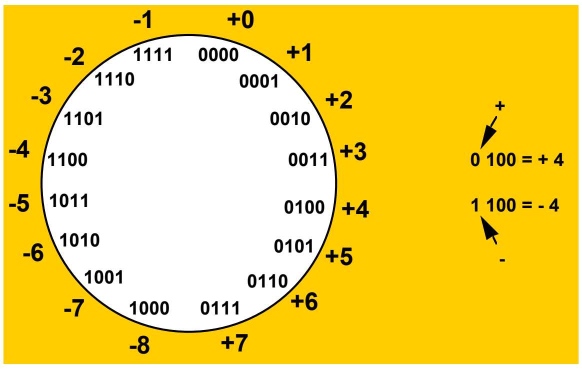
Ones Complement
(In Czech: První doplněk)



- Still two representations of 0! This causes some problems
- Some complexities in addition

Number Representations

Twos Complement
(In Czech: Druhý doplněk)



- Only one representation for 0
- One more negative number than positive number

Poznámka: Bit versus std_logic v publikacích?

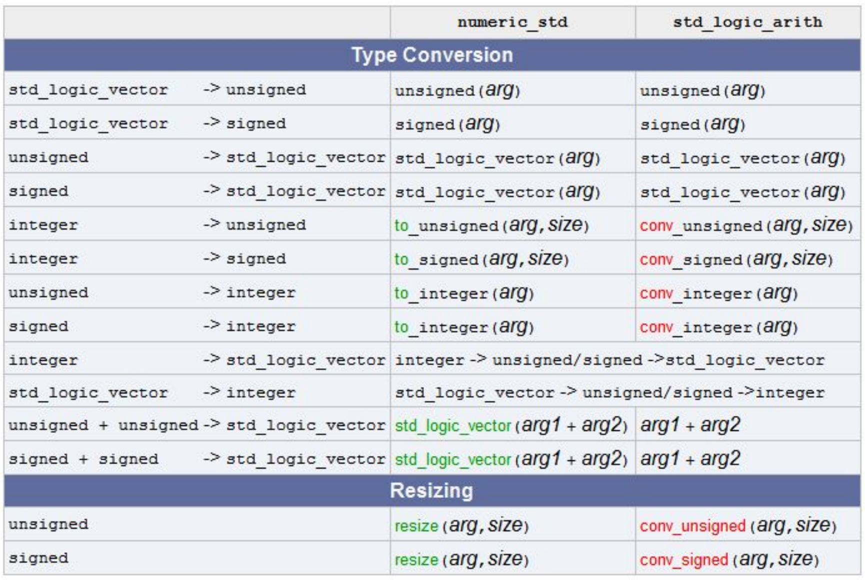
- Bit – hodnoty '0'and '1'
- std_logic MVL-9: '1', '0', 'X', 'Z','U', '-', 'L', 'H', 'W'
- **std_logic_arith** – *verze pro aritmetické výpočty rozlišuje signed / unsigned – původně vyvinutá firmou Synopsys, čtyři různí výrobci si udělali nepatrně odlišné verze, později IEEE udělalo oficiální verzi of std_logic_arith called numeric_std*
- **numeric_std** formální IEEE náhrada std_logic_artih
- nutno použít buď std_logic_arith nebo *numeric_std, nikdy ne oboje!*

Add STD_LOGIC_ARITH

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity addGeneric is
 generic
   ( DATA_WIDTH : natural := 8   );
 port (x, y: in std_logic_vector(DATA_WIDTH-1 downto 0);
      z: out std_logic_vector(DATA_WIDTH-1 downto 0)   );
 end addGeneric;

 architecture dataflow1 of addGeneric is
 begin
   z<=x+y;
 end dataflow1;
```

```
library ieee;use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity unsigned_adder is
generic ( DATA_WIDTH : natural := 8      );
port  ( a, b   : in unsigned  ((DATA_WIDTH-1) downto 0);
         result : out unsigned ((DATA_WIDTH-1) downto 0)
   );
end entity;
architecture rtl of unsigned_adder is
begin result <= a + b;
end rtl;
```

Conversion functions



	numeric_std	std_logic_arith
Type Conversion		
std_logic_vector -> unsigned	unsigned(*arg*)	unsigned(*arg*)
std_logic_vector -> signed	signed(*arg*)	signed(*arg*)
unsigned -> std_logic_vector	std_logic_vector(*arg*)	std_logic_vector(*arg*)
signed -> std_logic_vector	std_logic_vector(*arg*)	std_logic_vector(*arg*)
integer -> unsigned	to_unsigned(*arg*, *size*)	conv_unsigned(*arg*, *size*)
integer -> signed	to_signed(*arg*, *size*)	conv_signed(*arg*, *size*)
unsigned -> integer	to_integer(*arg*)	conv_integer(*arg*)
signed -> integer	to_integer(*arg*)	conv_integer(*arg*)
integer -> std_logic_vector	integer -> unsigned/signed ->std_logic_vector	
std_logic_vector -> integer	std_logic_vector -> unsigned/signed ->integer	
unsigned + unsigned -> std_logic_vector	std_logic_vector(*arg1* + *arg2*)	*arg1* + *arg2*
signed + signed -> std_logic_vector	std_logic_vector(*arg1* + *arg2*)	*arg1* + *arg2*
Resizing		
unsigned	resize(*arg*, *size*)	conv_unsigned(*arg*, *size*)
signed	resize(*arg*, *size*)	conv_signed(*arg*, *size*)

From: http://dz.ee.ethz.ch/support/ic/vhdl/vhdlsources.en.html

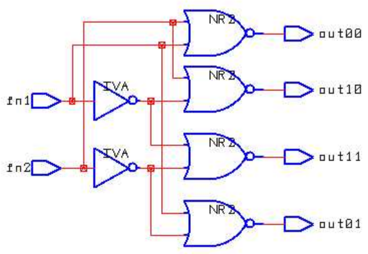
Miniotázky : Nakreslete schéma obvodu

```
entity test is
  port (in1 : in std_logic_vector (2 downto 0);
        out1 : out std_logic_vector (3 downto 0));
 end test;
architecture test_arch of test is
 begin
   out1(0)<=in1(1);
   out1(1)<=in1(2);
   out1(2)<=in1(0) and in1(1);
   out1(3)<='1';
 end test_arch ;
```

From VHDL for programmable logic, Skahill, Addison Wesley

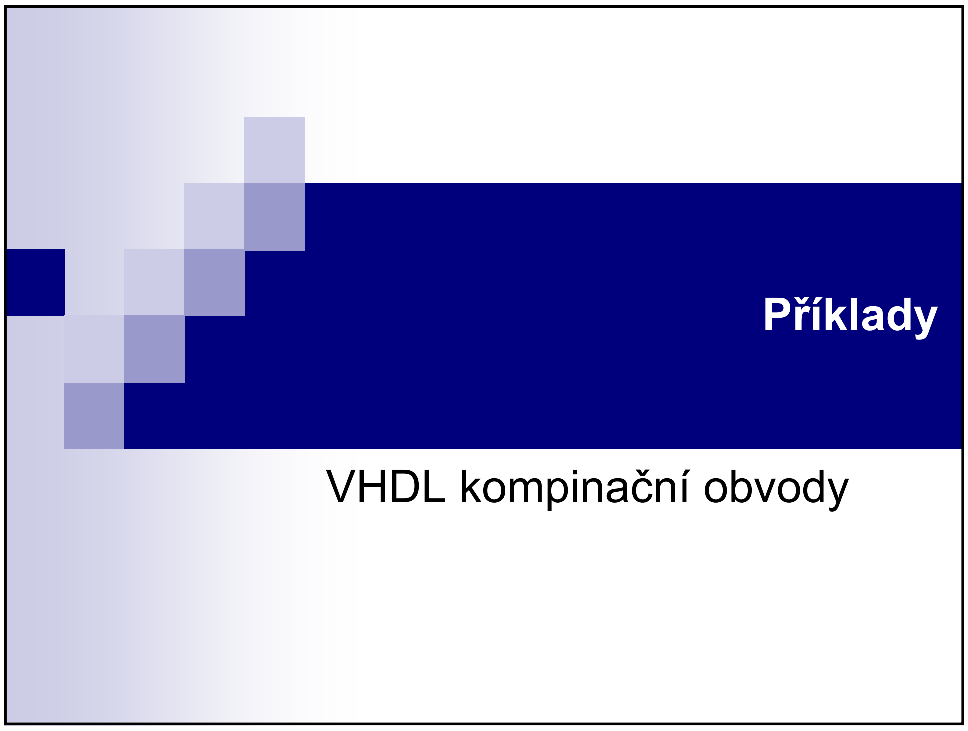
Miniotázky: Napište entitu

- Vyplňte pravdivostní tabulku a napiště VHDL kód



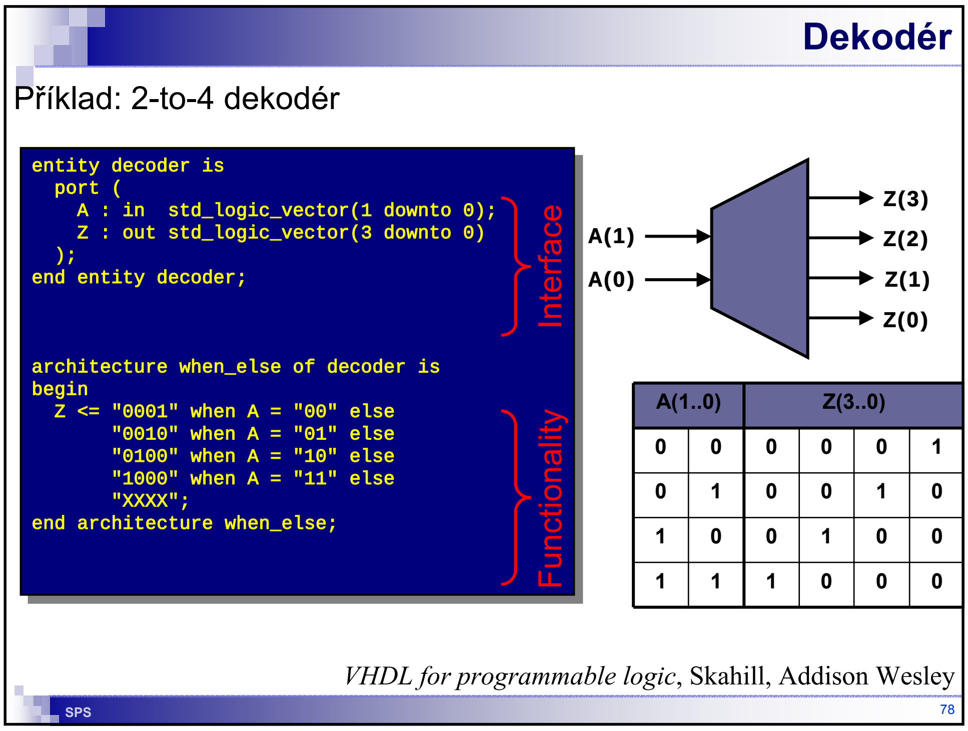
In1	in2	out0 0	out1 0	out1 1	out0 1

From VHDL for programmable logic, Skahill, Addison Wesley



Příklady

VHDL kompinační obvody



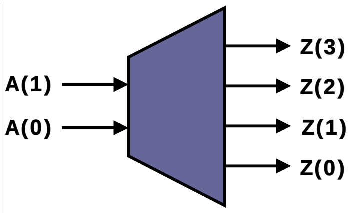
Dekodér

Příklad: 2-to-4 dekodér

```
entity decoder is
  port (
    A : in  std_logic_vector(1 downto 0);
    Z : out std_logic_vector(3 downto 0)
  );
end entity decoder;


architecture when_else of decoder is
begin
  Z <= "0001" when A = "00" else
       "0010" when A = "01" else
       "0100" when A = "10" else
       "1000" when A = "11" else
       "XXXX";
end architecture when_else;
```

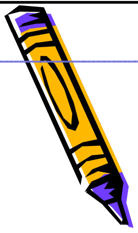
Interface

Functionality

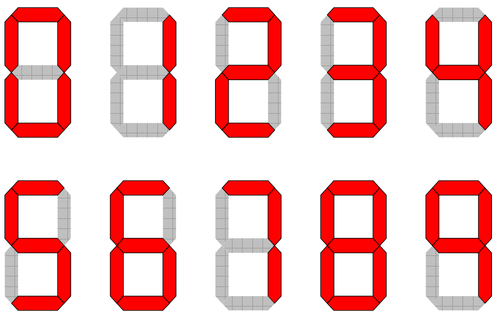


A(1..0)		Z(3..0)			
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

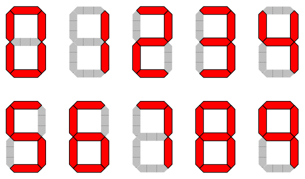
VHDL for programmable logic, Skahill, Addison Wesley

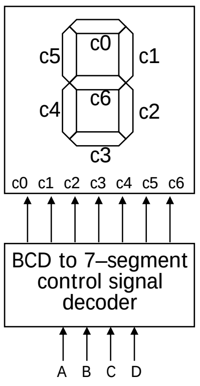


Decoder for 7-segment display



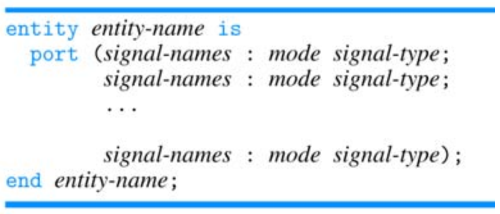
BCD / 7-segment





A	B	C	D	C0	C1	C2	C3	C4	C5	C6
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1
1	0	1	–	–	–	–	–	–	–	–
1	1	–	–	–	–	–	–	–	–	–

7segment Entity



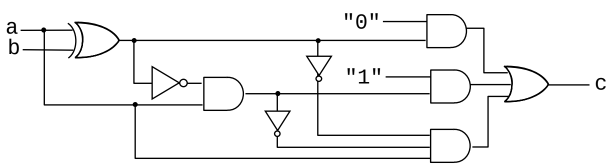
```
entity entity-name is
  port (signal-names : mode signal-type;
           signal-names : mode signal-type;
           . . .

           signal-names : mode signal-type);
end entity-name;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
-- 7segment driver hex display of a number
ENTITY SevenSegment IS
PORT( D, C, B, A :IN STD_LOGIC;
  segmentsOut: OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
        );
END SevenSegment;
```

Conditional Signal Assignment

```
c <= "0" when a /= b else
     "1"  when a = '1' else
     b;
```

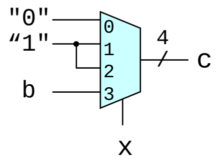
means



Selected Signal Assignment

```
with x select
  c <=    "0" when "00" ,
          "1" when "01" | "10" ,
      "b" when others;
```

means



- Resulting circuit is more compact and faster than circuit produced by conditional assignment.

Assignment Statements (summary)

```
SIGNAL x, y, z  :STD_LOGIC;
SIGNAL a, b, c  :STD_LOGIC_VECTOR( 7 DOWNTO 0);
SIGNAL sel      :STD_LOGIC_VECTOR( 2 DOWNTO 0);

-- Conditional Assignment Statement
  x <='0'   WHEN sel = "000" ELSE
      y     WHEN sel = "011" ELSE
      z     WHEN x = '1' ELSE
     '1';

-- Selected Signal Assignment Statement
-- NOTE: The selection values must be constants
WITH sel SELECT
   x <= '0'  WHEN "000",
        y  WHEN "011",
        z  WHEN "100",
       '1' WHEN OTHERS;

-- Selected signal assignments also work with vectors
WITH x SELECT
      a <= "01010101"WHEN '1',
      b   WHEN OTHERS;
```

7segment ARCHITECTURE

```
architecture Behavioral of SevenSegment IS
Signal dataIn:STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN  dataIn <= D & C & B & A; -- concatenate operator (&)
with dataIn SELECT-- LSB is A
   segmentsOut <="1000000" WHEN "0000",-- 0
                 "1111001" WHEN "0001",-- 1
                 "0100100" WHEN "0010", -- 2
                 "0110000" WHEN "0011", -- 3
                 "0011001" WHEN "0100",-- 4
                 --
                 --
                 "0000011" WHEN "1011",-- b
                 "0100111" WHEN "1100",-- c
                 "0100001" WHEN "1101",-- d
                 "0000110" WHEN "1110",-- E
                 "0001110" WHEN others;-- for F "111" and simulation
END Behavioral;
```

7segment with ripple blanIn / blankOut

```
ENTITY SevenSegment IS
    PORT( D, C, B, A :IN STD_LOGIC;  blankIn :IN STD_LOGIC;
            blankOut : OUT STD_logic;
           segmentsOut: OUT STD_LOGIC_VECTOR(6 DOWNTO 0) );
END SevenSegment;

architecture Behavioral of SevenSegment IS
 Signal dataIn:STD_LOGIC_VECTOR(3 DOWNTO 0);
 Signal segments: STD_LOGIC_VECTOR(6 DOWNTO 0);
BEGIN
    dataIn <= D & C & B & A;
    with dataIn SELECT-- LSB is A
    segments <="1000000" WHEN "0000",-- 0
                "1111001" WHEN "0001",-- 1
                "0100100" WHEN "0010", -- 2
                --
                --
                "0000110" WHEN "1110",-- E
                "0001110" WHEN others;-- for F "111" and simulation
   blankOut <= '1' when dataIn="0000" and blankIn='1' else '0';
   segmentsOut <= segments when dataIn /= "0000" or blankIn='0' else "1111111

END Behavioral;
```

...KONEC...

a nashledanou příště