

# Computação Reconfigurável

## Aula 7

Valeri Skliarov, Prof. Catedrático

Email: [skl@ua.pt](mailto:skl@ua.pt)

URL: <http://sweet.ua.pt/skl/>

Departamento de Eletrónica, Telecomunicações e Informática  
Universidade de Aveiro

<http://elearning.ua.pt/>

# Aula 6

- Paralelismo e concorrência.
- Implementação de redes de procura.
- Implementação de redes de ordenação.
- Computação de *popcount* (peso e distância de *Hamming*).

Bibliografia: V.Sklyarov,  
I.Skliarova, A.Barkalov,  
L.Titarenko. Synthesis and  
Optimization of FPGA-Based  
Systems. Springer, 2014.

## Paralelismo e concorrência

1. Frequência de FPGAs é essencialmente mais baixa que a frequência de processadores de uso geral (e.g. PC). Compare 4 GHz de PC e 100 MHz da placa Nexys-4.
2. Para tirar vantagens de FPGAs paralelismo deve ser aplicado largamente.
3. Paralelismo tem mais relações com várias operações executadas ao mesmo tempo. Ex. redes de procura e ordenação.
4. Concorrência tem mais relações com circuitos que são ativos ao mesmo tempo. Ex. atribuições " $\leq$ " no corpo de arquitetura e processos.
5. Outro tipo de aceleração possível é a utilização de circuitos combinatórios com pequena profundidade em vez de circuitos sequenciais.
6. FPGA permite acelerar resolução de problemas quando só *software* é menos rápido de que *software + hardware (FPGA) + interação entre hardware (FPGA) e software*.

## Vamos discutir sistemas seguintes:

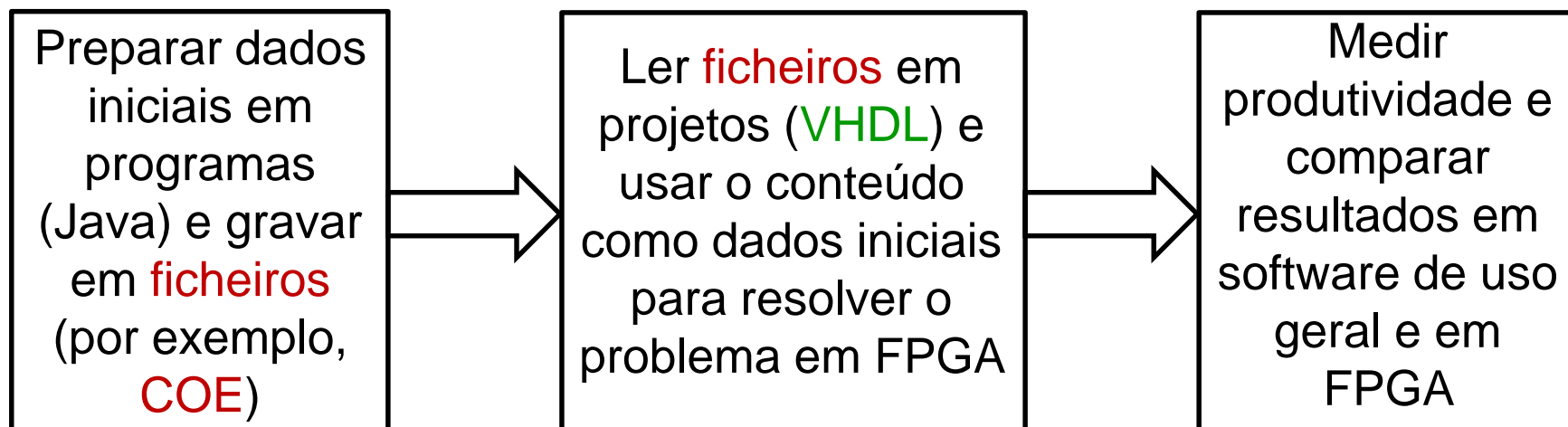
Software de computador de uso geral: *desktop PC* ou portátil; programas escritos em linguagens de uso geral, por exemplo JAVA

Vamos falar sobre interação na segunda parte de aulas

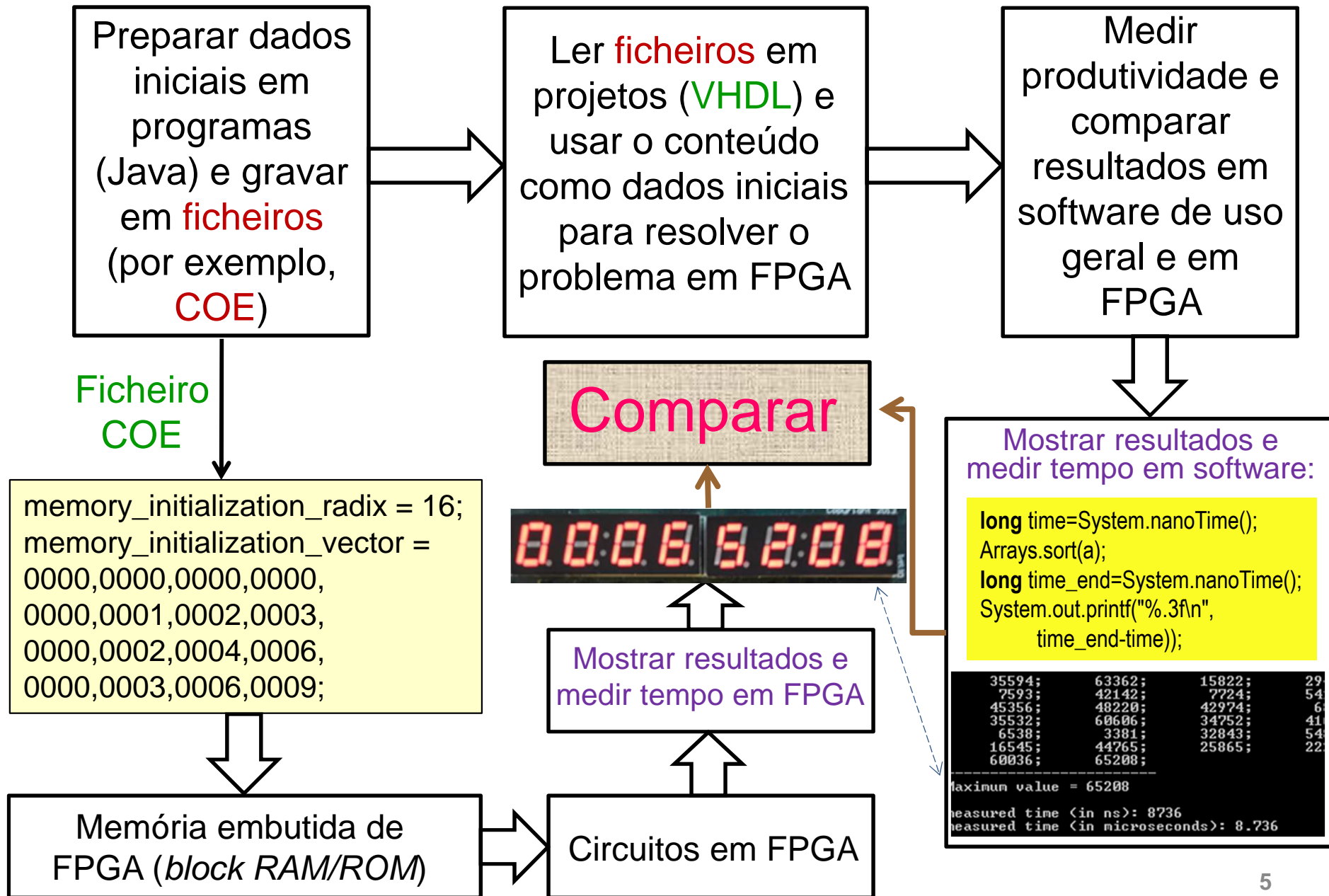
## comparando com

FPGA sem considerar o tempo de interações (*communication overheads*)

## Metodologia geral:



# Metodologia geral:



## Exemplos

1. Computações tabulares (computações baseadas em tabelas):
  - a) Preencher ficheiro do tipo COE, utilizando uma linguagem de alto nível.
  - b) Configurar memória embutida de tipo ROM utilizando o ficheiro COE.
  - c) Utilizar operandos como partes de endereço e ler os resultados durante só um (ou dois) ciclo(s) de relógio.

A <sub>2</sub>	A <sub>10</sub>	B <sub>2</sub>	B <sub>10</sub>	Resultado binário	Resultado decimal
00	0	00	0	0000	0
01	1	00	0	0000	0
10	2	00	0	0000	0
11	3	00	0	0000	0
00	0	01	1	0000	0
01	1	01	1	0001	1
10	2	01	1	0010	2
11	3	01	1	0011	3



A <sub>2</sub>	A <sub>10</sub>	B <sub>2</sub>	B <sub>10</sub>	Resultado binário	Resultado decimal
00	0	10	2	0000	0
01	1	10	2	0010	2
10	2	10	2	0100	4
11	3	10	2	0110	6
00	0	11	3	0000	0
01	1	11	3	0011	3
10	2	11	3	0110	6
11	3	11	3	1001	9



# Programa em Java

```
import java.util.*; import java.io.*;
public class ROM_for_multiplication {
    static final int MaxOp1 = 256;           // valor máximo+1 para primeiro operando
    static final int MaxOp2 = 256;           // valor máximo+1 para segundo operando
    public static void main (String args[]) throws IOException {
        File fout = new File("coe_for_multiplication.coe");
        PrintWriter pw = new PrintWriter(fout);
        pw.println("memory_initialization_radix = 16;");
        pw.println("memory_initialization_vector = ");
        for(int i = 0; i < MaxOp1; i++)
            for(int j = 0; j < MaxOp1; j++)
                pw.printf((i==(MaxOp1-1)) & (j==(MaxOp2-1)) ? "%04x;" : "%04x,",i*j);
        pw.println();
        pw.close();
    }
}
```

Java

```
static final int MaxOp1 = 4;           // valor máximo-1 para primeiro operando
static final int MaxOp2 = 4;           // valor máximo-1 para segundo operando
```

```
memory_initialization_radix = 16;
memory_initialization_vector =
0000,0000,0000,0000,0000,0001,0002,0003,0000,0002,0004,0006,0000,0003,0006,0009;
```

endereço

0000,0001,0010,0011,0100,0101,0110,0111,1000,1001,1010,1011,1100,1101,1110,1111

## Pode comparar valores

A <sub>2</sub>	A <sub>10</sub>	B <sub>2</sub>	B <sub>10</sub>	Resultado binário	Resultado decimal
00	0	00	0	0000	0
01	1	00	0	0000	0
10	2	00	0	0000	0
11	3	00	0	0000	0
00	0	01	1	0000	0
01	1	01	1	0001	1
10	2	01	1	0010	2
11	3	01	1	0011	3

A <sub>2</sub>	A <sub>10</sub>	B <sub>2</sub>	B <sub>10</sub>	Resultado binário	Resultado decimal
00	0	10	2	0000	0
01	1	10	2	0010	2
10	2	10	2	0100	4
11	3	10	2	0110	6
00	0	11	3	0000	0
01	1	11	3	0011	3
10	2	11	3	0110	6
11	3	11	3	1001	9

```

static final int MaxOp1 = 4;           // valor máximo-1 para primeiro operando
static final int MaxOp2 = 4;           // valor máximo-1 para segundo operando

memory_initialization_radix = 16;
memory_initialization_vector =
0000,0000,0000,0000,0000,0001,0002,0003,0000,0002,0004,0006,0000,0003,0006,0009;

```



# VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all; use IEEE.STD_LOGIC_UNSIGNED.all;
```

```
entity Multiplier_ROM is
```

```
generic (  NOp1      : integer := 8;
           NOp2      : integer := 8;
port (      clk       : in  std_logic;
          sw         : in  std_logic_vector(15 downto 0);
          seg        : out std_logic_vector (6 downto 0);
          sel_disp   : out std_logic_vector (7 downto 0) );
end Multiplier_ROM;
```

```
architecture Behavioral of Multiplier_ROM is
```

```
signal data_in  : std_logic_vector(NOp1+NOp2-1 downto 0);
```

```
component blk_mem_gen_1 is
```

```
port (      clka      : in  std_logic;
          addra       : in  std_logic_vector(NOp1+NOp2-1 downto 0);
          douta       : out std_logic_vector(NOp1+NOp2-1 downto 0) );
```

```
end component;
```

```
signal Op1      : std_logic_vector(NOp1-1 downto 0);
```

```
signal Op2      : std_logic_vector(NOp2-1 downto 0);
```

```
signal address  : std_logic_vector(NOp1+NOp2-1 downto 0) := Op1 & Op2;
```

```
signal BCD4, BCD3, BCD2, BCD1, BCD0 : std_logic_vector(3 downto 0);
```

```
begin
```

```
-- corpo da arquitetura ↓↓↓↓
```



VHDL

Cada multiplicação vai ser executada durante só um ciclo de relógio

# VHDL

```
Op1 <= sw(15 downto 8);
Op2 <= sw(7 downto 0);

block_ROM : blk_mem_gen_1                                -- criado por IP core generator
    port map (clk, address, data_in );                    -- mapeamento posicional

disp_cont: entity work.EightDisplayControl
    port map ( clk=>clk,  leftL=>"0000",      near_leftL=>"0000",
                near_rightL=>"0000",  rightL=>BCD4,
                leftR=>BCD3,          near_leftR=>BCD2,
                near_rightR=>BCD1,  rightR=>BCD0,
                select_display=>sel_disp,segments=>seg);

BCD_dec:  entity work.BinToBCD16
port map (      clk      => clk,
              reset      => '0',
              ready       => open,
              binary      => data_in,
              request     => '1',
              BCD4        => BCD4,
              BCD3        => BCD3,
              BCD2        => BCD2,
              BCD1        => BCD1,
              BCD0        => BCD0);

end Behavioral;
```

VHDL

Ainda não expliquei o código deste componente que permite converter o código binário para o código decimal. Pode utilizar este componente para mostrar os resultados em decimal ou não e mostrar os resultados em hexadecimal

Todos os projetos e códigos estão disponíveis em [elearning.ua.pt](http://elearning.ua.pt). Pode encontrar explicações em *P.P. Chu. FPGA Prototyping Using VHDL with Examples: Xilinx Spartan-3 Version. Jonh, Willey & Sons, 2008.*

Cada multiplicação vai ser executada durante só um (**dois**) ciclo(s) de relógio

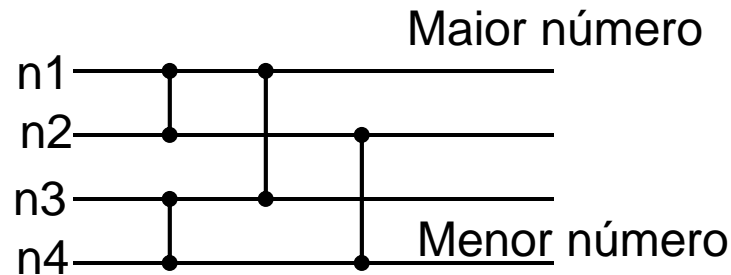
## Exemplos

2. Computação do peso de *Hamming* em circuitos combinatórios.
3. Redes de procura.
4. Redes de ordenação

### Informação adicional:

1. *Al-Haj Baddar, S.W., & Batcher, K.E. (2011). Designing Sorting Networks. A New Paradigm. Springer.*
2. *D.E. Knuth, The Art of Computer Programming. Sorting and Searching. vol. III. Addison-Wesley, 2011.*
3. *V.Sklyarov, I.Skliarova, Parallel Processing in FPGA-based Digital Circuits and Systems. TUT Press, Tallinn, 2013.*
4. *V.Sklyarov, I.Skliarova, A.Barkalov, L.Titareenko. Synthesis and Optimization of FPGA-Based Systems. Springer, 2014.*

# Implementação de redes de procura (searching networks)

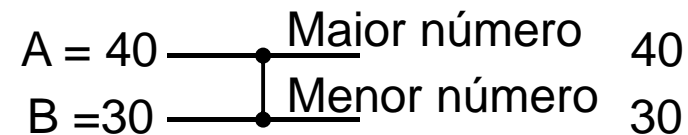
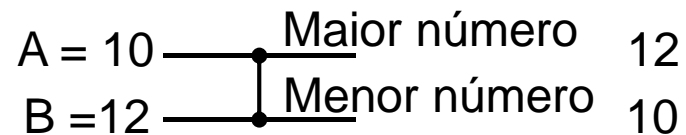


n1, n2, n3, n4 são valores que podem ser ordenados: inteiros, reais, caracteres, *strings*, etc.

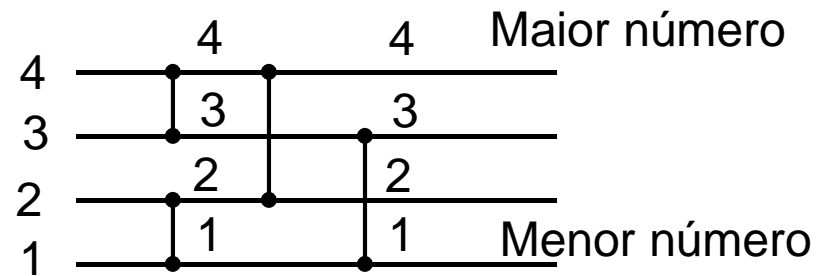
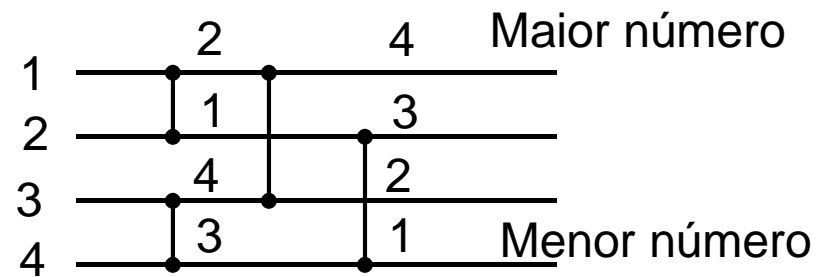
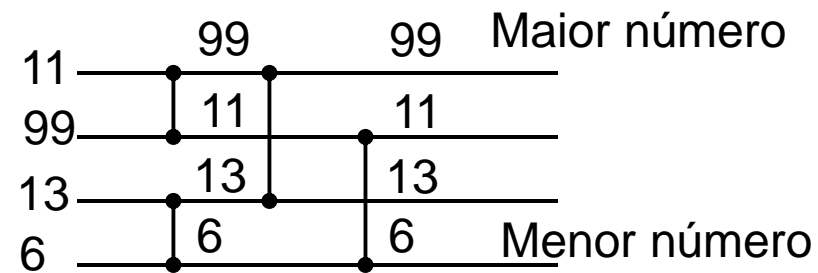


Um elemento para comparar e trocar se necessário

## Exemplos:



## Exemplos:



## VHDL:

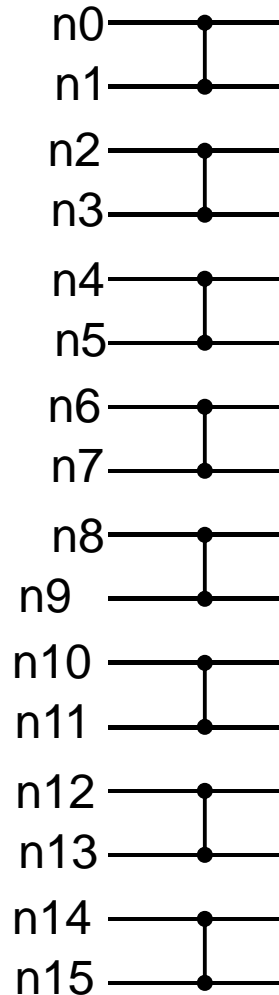
```
for i in 0 to L/2-1 loop
    if MyAr(2*i) < MyAr(2*i+1) then
        N_MyAr(2*i) <= MyAr(2*i+1);
        N_MyAr(2*i+1) <= MyAr(2*i);
    else null;
    end if;
end loop;
```

A \_\_\_\_\_ Maior número  
B \_\_\_\_\_ Menor número

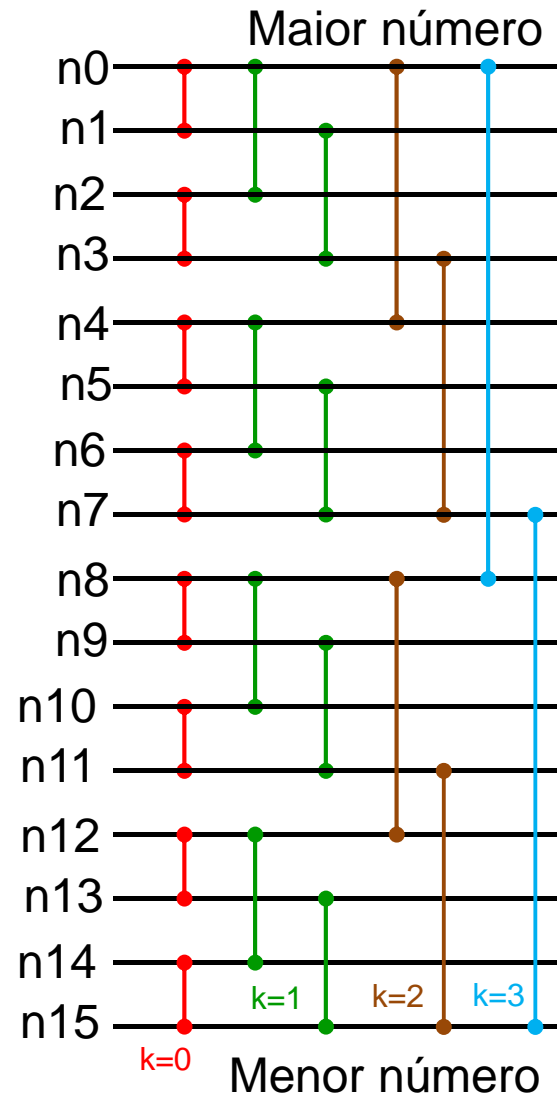
FPGA disponível na placa **Nexys4** pode executar 1024 operações deste tipo em paralelo sobre inteiros de 32 bits com frequência 100 MHz, i.e. durante só 10 ns.

**Todas as operações vão ser executadas em paralelo no circuito combinatório**

**L = 16**



# Exemplo com implementação:



```
measured time <in ns>: 5724
measured time <in microseconds>: 5.724
```

```
import java.util.*;

public class max_min_network_parameterization_withoutMath {
    static final int N = 16; // N = 16,32,64,128,256,...
    static final int p = 4; // p = 4,5,6,7,8, ...
    static final int s[] = {1,2,4,8,16}; // para cada p s deve ser alterado
    static Random rand = new Random();
    public static void main(String[] args)
    {
        int result, a[] = new int[N];
        for(int x = 0; x < N; x++)
            a[x] = rand.nextInt(50); //Integer.MAX_VALUE;
        for(int i = 0; i < a.length; i++) {
            System.out.printf("%10d; ", a[i]);
            if (((i+1)%5) == 0) System.out.println();
        }

        long time=System.nanoTime();
        result = SN(N,a);
        long time_end=System.nanoTime();
        System.out.printf("\n\nMaximum value = %5d\n\n", result);
        System.out.printf("measured time (in ns): %d\n", time_end-time);
        System.out.printf("measured time (in microseconds): %.3f\n",
            (double)(time_end-time)/1000.);
    }

    public static int SN(int N, int a[])
    {
        int tmp;
        for(int k = 0; k < p; k++)
            for(int i = 0; i < a.length/s[k+1]; i++)
                if (a[s[k+1]*i+s[k]-1] > a[s[k+1]*i+s[k+1]-1])
                {
                    tmp = a[s[k+1]*i+s[k]-1];
                    a[s[k+1]*i+s[k]-1] = a[s[k+1]*i+s[k+1]-1];
                    a[s[k+1]*i+s[k+1]-1] = tmp;
                }
        return a[N-1];
    }
}
```

```
4798; 48711; 11758; 28539; 50738;
55401; 53767; 29906; 23325; 42132;
26222; 28773; 52733; 26006; 22247;
29933;
Maximum value = 55401
measured time <in ns>: 5724
measured time <in microseconds>: 5.724
Press any key to continue . . .
```

resultados

Java

```
public static int SN(int N, int a[])
```

```
{ int tmp;
```

```
for(int k = 0; k < p; k++)
```

```
for(int i = 0; i < a.length/s[k+1]; i++)
```

```
if (a[s[k+1]*i+s[k]-1] > a[s[k+1]*i+s[k+1]-1])
```

```
{ tmp = a[s[k+1]*i+s[k]-1];
```

```
a[s[k+1]*i+s[k]-1] = a[s[k+1]*i+s[k+1]-1];
```

```
a[s[k+1]*i+s[k+1]-1] = tmp;
```

```
return a[N-1];
```

Java

O programa em Java pode ser convertido no código VHDL

```
max_f : process(data_in)
```

```
variable MyAr : in_data;
```

```
variable tmp : std_logic_vector(M-1 downto 0);
```

```
begin
```

```
for i in N-1 downto 0 loop
```

```
MyAr(i) := data_in(M*(i+1)-1 downto M*i);
```

```
end loop;
```

```
for k in 0 to p-1 loop
```

```
for i in 0 to N/(2**(k+1))-1 loop
```

```
if ( MyAr( 2**(k+1)*i+(2**k)-1 ) > MyAr( 2**(k+1)*i+2**(k+1)-1 ) ) then
```

```
tmp := MyAr( 2**(k+1)*i+(2**k)-1 );
```

```
MyAr( 2**(k+1)*i+(2**k)-1 ) := MyAr( 2**(k+1)*i+2**(k+1)-1 );
```

```
MyAr( 2**(k+1)*i+2**(k+1)-1 ) := tmp;
```

```
end if;
```

```
end loop;
```

```
end loop;
```

```
max_value <= MyAr(N-1);
```

```
end process;
```

VHDL



```
library IEEE; use IEEE.STD_LOGIC_1164.all; use IEEE.STD_LOGIC_UNSIGNED.a
```

```
entity Max_circuit is
```

```
generic ( N          : integer := 32;  
          M          : integer := 16;  
          p          : integer := 5);  
port (    data_in    : in std_logic_vector(N*M-1 downto 0);  
          max_value   : out std_logic_vector(M-1 downto 0) );
```

```
end Max_circuit;
```

```
architecture Behavioral of Max_circuit is
```

```
    type in_data is array (N-1 downto 0) of std_logic_vector(M-1 downto 0);
```

```
begin
```

```
max_f : process(data_in)
```

```
    variable MyAr : in_data;
```

```
    variable tmp : std_logic_vector(M-1 downto 0);
```

```
begin
```

```
    for i in N-1 downto 0 loop
```

```
        MyAr(i) := data_in(M*(i+1)-1 downto M*i);
```

```
    end loop;
```

```
    for k in 0 to p-1 loop
```

```
        for i in 0 to N/(2**(k+1))-1 loop
```

```
            if ( MyAr( 2**(k+1)*i+(2**k)-1 ) > MyAr( 2**(k+1)*i+2**(k+1)-1 ) ) then
```

```
                tmp := MyAr( 2**(k+1)*i+(2**k)-1 );
```

```
                MyAr( 2**(k+1)*i+(2**k)-1 ) := MyAr( 2**(k+1)*i+2**(k+1)-1 );
```

```
                MyAr( 2**(k+1)*i+2**(k+1)-1 ) := tmp;
```

```
            end if;
```

```
        end loop;
```

```
    end loop;
```

```
    max_value <= MyAr(N-1);
```

```
end process;
```

```
end Behavioral;
```

**Código completo**

**VHDL**

# Verificação

```
35594;    63362;    15822;    29496;    49737;
7593;     42142;    7724;     54164;    38001;
45356;    48220;    42974;    6855;     43994;
35532;    60606;    34752;    41670;    31648;
6538;     3381;     32843;    54841;    33063;
16545;    44765;    25865;    22244;    46945;
60036;    65208;
```

Maximum value = 65208

measured time (in ns): 8736

measured time (in microseconds): 8.736

```
50462;    2503;     55844;    27984;    31840;
11360;    18089;    40565;    15521;    14529;
10719;    10500;    47034;    20630;    31073;
50407;    33871;    39797;    64962;    23472;
19914;    30634;    9754;     22011;    11736;
4182;     10868;    18483;    31722;    18810;
15221;    57364;
```

Maximum value = 64962

measured time (in ns): 5423

measured time (in microseconds): 5.423

```
207;      63909;    5333;     33904;    27119;
32346;    42665;    64811;    42431;    58537;
58414;    51109;    16391;    36040;    3595;
47724;    59812;    18780;    19137;    25567;
34493;    4317;     54945;    44243;    58567;
14264;    57841;    55319;    48583;    2071;
61520;    18537;
```

Maximum value = 64811

measured time (in ns): 5423

measured time (in microseconds): 5.423

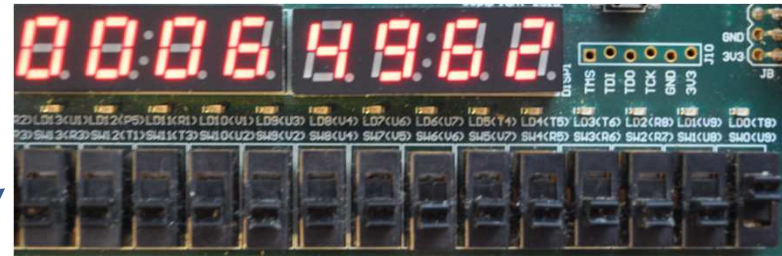
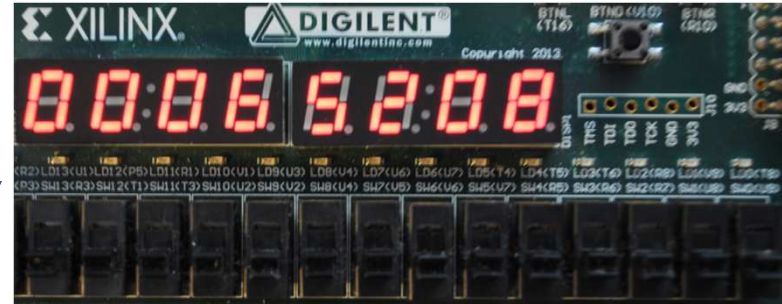
```
57487;    7049;     43245;    10673;    37132;
61934;    28609;    17123;    56423;    63474;
55855;    31009;    16854;    32785;    23745;
49538;    35476;    38906;    34316;    3767;
48333;    48898;    53128;    52537;    35414;
45429;    11745;    27030;    27647;    558;
32493;    1842;
```

Maximum value = 63474

measured time (in ns): 6929

measured time (in microseconds): 6.929

**Execução do programa**



Para todos os exemplos o tempo de execução é menor que 5 níveis de comparadores (< 100 ns)

# Simulação



1. É melhor utilizar sistema hexadecimal.
2. O tempo da simulação é 3000 ns.
3. Todos outros passos são iguais aos passos que já foram explicados várias vezes.

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

```
entity stimulus is
```

```
  generic (
    N      : integer := 8;
    M      : integer := 4;
    p      : integer := 3);
```

```
end stimulus;
```

```
architecture behavior of stimulus is
```

```
  signal data_in      : std_logic_vector(N*M-1 downto 0);
  signal max_value     : std_logic_vector(M-1 downto 0);
```

```
begin
```

```
  uut: entity work.Max_circuit
        generic map (N,M,p)
        port map (data_in, max_value);
```

```
  stim_proc: process
```

```
  begin
```

```
    data_in <= x"481a9e75";
```

```
    wait for 1000 ns;
```

```
    data_in <= x"12134261";
```

```
    wait for 1000 ns;
```

```
    data_in <= x"abc7fe02";
```

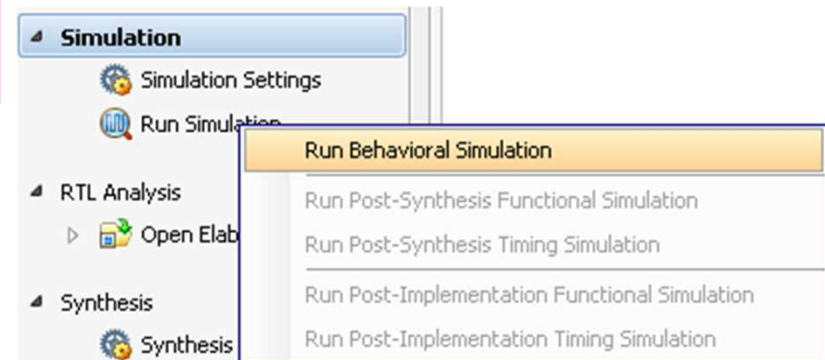
```
    wait for 1000 ns;
```

```
  end process;
```

```
end behavior;
```

Name	Value	0 us	1 us	2 us
data	481a9e75	481a9e75	12134261	abc7fe02
max_value	e	e	6	f
N	1000		1000	
M	100		100	
p	11		11	

VHDL



```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity Max_finder is
generic (   N           : integer := 32;
           M           : integer := 16;
           p           : integer := 5);
port (  clk           : in std_logic;
       sw            : in std_logic_vector(2 downto 0);
       seg           : out std_logic_vector (6 downto 0);
       sel_disp      : out std_logic_vector (7 downto 0) );
end Max_finder;

architecture Behavioral of Max_finder is
signal data_in       : std_logic_vector(N*M-1 downto 0);
signal max_value      : std_logic_vector(M-1 downto 0);
signal BCD4, BCD3, BCD2, BCD1, BCD0 : std_logic_vector(3 downto 0);
component blk_mem_gen_0 is
port (
  clka      : in std_logic;
  addra     : in std_logic_vector(2 downto 0);
  douta     : out std_logic_vector(N*M-1 downto 0)
);
end component;

begin
block_ROM : blk_mem_gen_0
  port map (clk, sw, data_in);

disp_cont: entity work.EightDisplayControl
  port map ( clk=>clk,   leftL=>"0000", near_leftL=>"0000",
            near_rightL=>"0000", rightL=>BCD4,
            leftR=>BCD3, near_leftR=>BCD2,
            near_rightR=>BCD1, rightR=>BCD0,
            select_display=>sel_disp, segments=>seg);

```

## Código VHDL completo

```

BCD_dec: entity work.BinToBCD16
port map (  clk      => clk,
           reset     => '0',
           ready      => open,
           binary     => max_value,
           request    => '1',
           BCD4       => BCD4,
           BCD3       => BCD3,
           BCD2       => BCD2,
           BCD1       => BCD1,
           BCD0       => BCD0);

my_max : entity work.Max_circuit
generic map ( N=>N,M=>M,p=>p)
port map( data_in=>data_in,
          max_value=>max_value );

end Behavioral;

```

VHDL

```

import java.util.*; import java.io.*;
public class max_min_network_parameterization_withoutMath
{
    static final int N = 32;
    static final int p = 5;
    static final int s[] = {1,2,4,8,16,32};
    static final int M = 16;
    static final int D = 8;
    static Random rand = new Random();
    public static void main(String[] args) throws IOException
    {
        int result, a[] = new int[N];

        File fout = new File("coe_for_max.coe");
        PrintWriter pw = new PrintWriter(fout);
        pw.println("memory_initialization_radix = 16;");
        pw.println("memory_initialization_vector = ");

        for(int d = 0; d < D; d++) {
            for(int x = 0; x < N; x++) a[x] = rand.nextInt((int)Math.pow(2,16)-1);
            System.out.println();
            for(int i = 0; i < a.length; i++) {
                System.out.printf("%10d; ",a[i]);
                if (((i+1)%5) == 0) System.out.println();
            }
            for(int l=0; l<N-1; l++) pw.printf("%04x",a[l]);
            pw.printf((d == D-1) ? "%04x;\n" : "%04x,\n",a[N-1]);
            long time=System.nanoTime();
            result = SN(N,a);
            long time_end=System.nanoTime();
            System.out.printf("\n-----\nMaximum value = %5d\n",result);
            System.out.printf("measured time (in ns): %d\n",time_end-time);
            System.out.printf("measured time (in microseconds): %.3f\n",(double)(time_end-time)/1000.);
            pw.println();
            pw.close();
        }

        public static int SN(int N, int a[])
        {
            int tmp;
            for(int k = 0; k < p; k++)
                for(int i = 0; i < a.length/s[k+1]; i++)
                    if (a[s[k+1]*i+s[k]-1] > a[s[k+1]*i+s[k+1]-1])
                    {
                        tmp = a[s[k+1]*i+s[k]-1];
                        a[s[k+1]*i+s[k]-1] = a[s[k+1]*i+s[k+1]-1];
                        a[s[k+1]*i+s[k+1]-1] = tmp;
                    }
            return a[N-1];
        }
    }
}

```

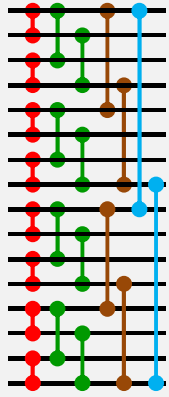
// N pode ser qualquer valor de  $2^R$ ,  $R = 0,1,2,3,4,5,6,7,8,9,10,\dots$   
 // p is the number of levels in MAX  
 // change this array for greater numbers of N  
 // M is the size of one word. DO NOT FORGET TO CHANGE FORMAT WHEN CHANGE M  
 // D is the depth of memory (how many addresses are allocated for the memory)

// CHANGE FORMAT HERE 4 =  $\text{intlog } M$ ,  
 // CHANGE FORMAT HERE 4 =  $\text{intlog } M$ ,

i.e.  $2^4 = M = 16$   
 i.e.  $2^4 = M = 16$

**Java**

$p=4$

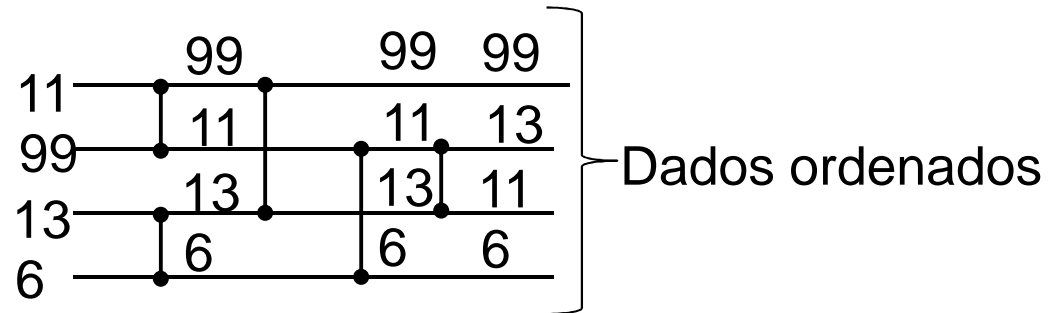


**Código Java completo**

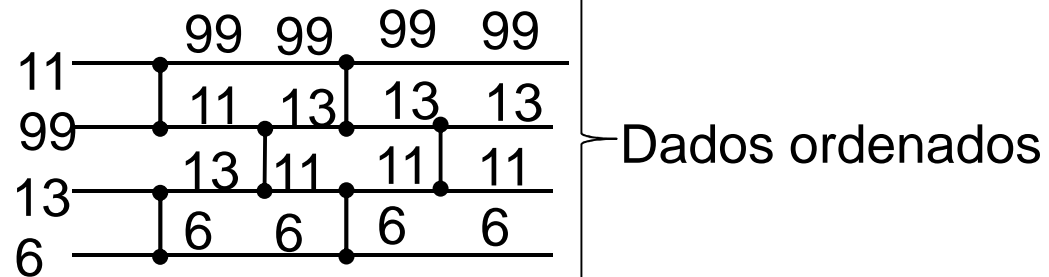


# Implementação de redes de ordenação (sorting networks)

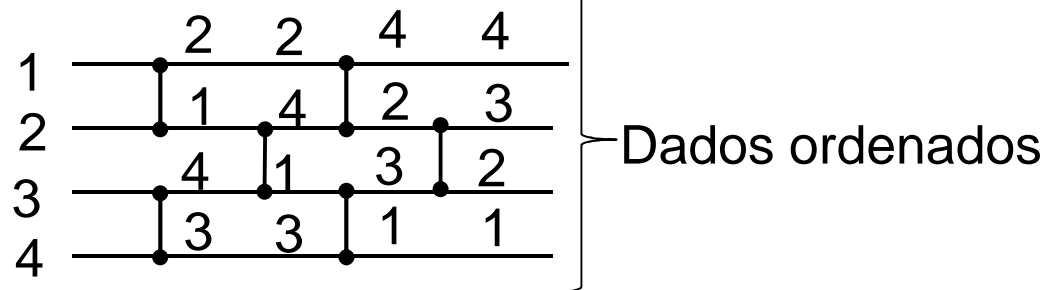
*Rede even-odd merge*



*Rede even-odd transition*

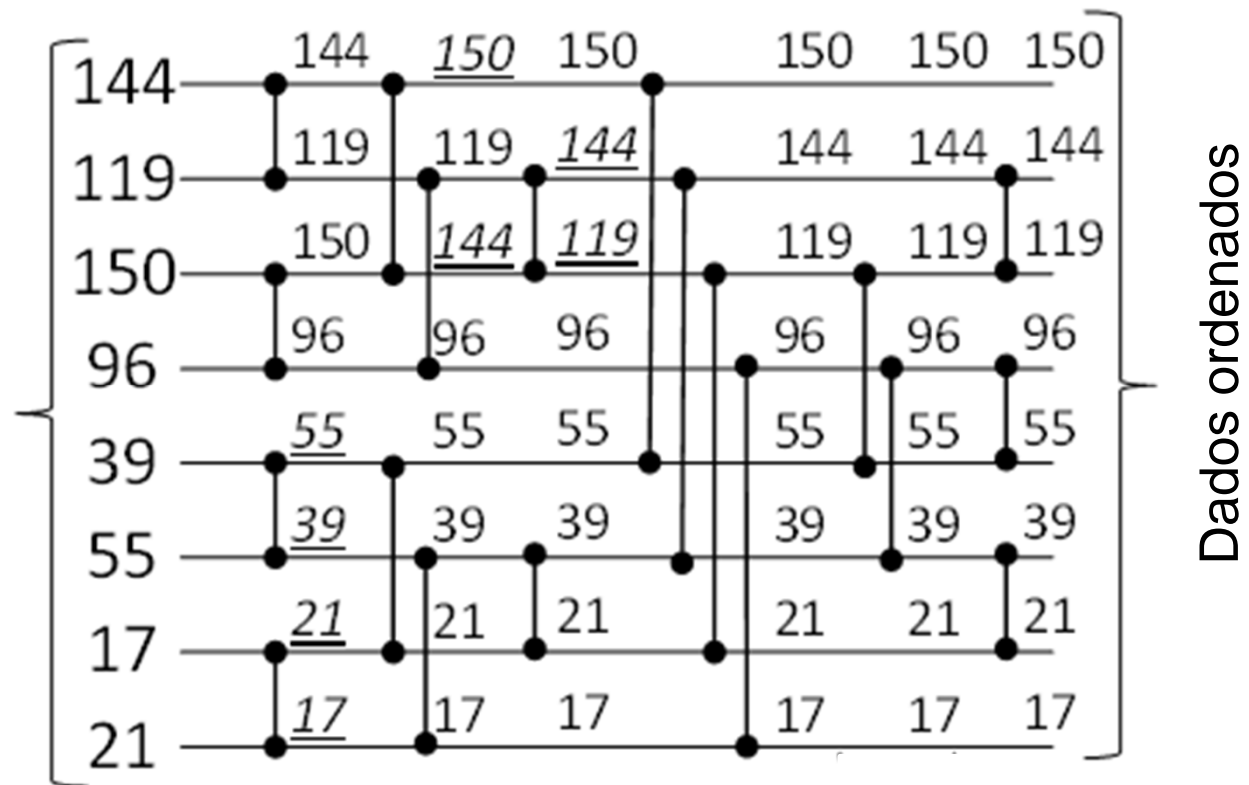


*Rede even-odd transition*



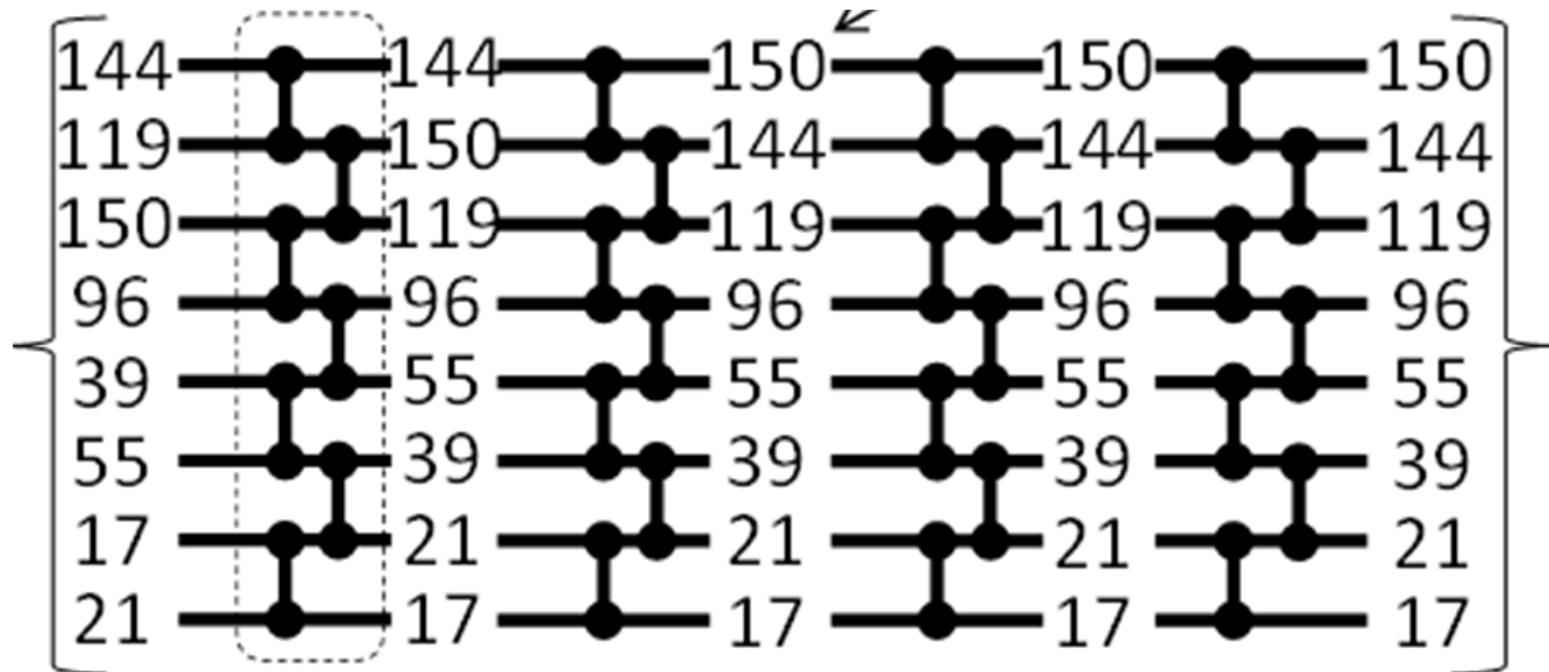
# Implementação de redes de ordenação (sorting networks)

Rede *even-odd merge*  
para oito variáveis



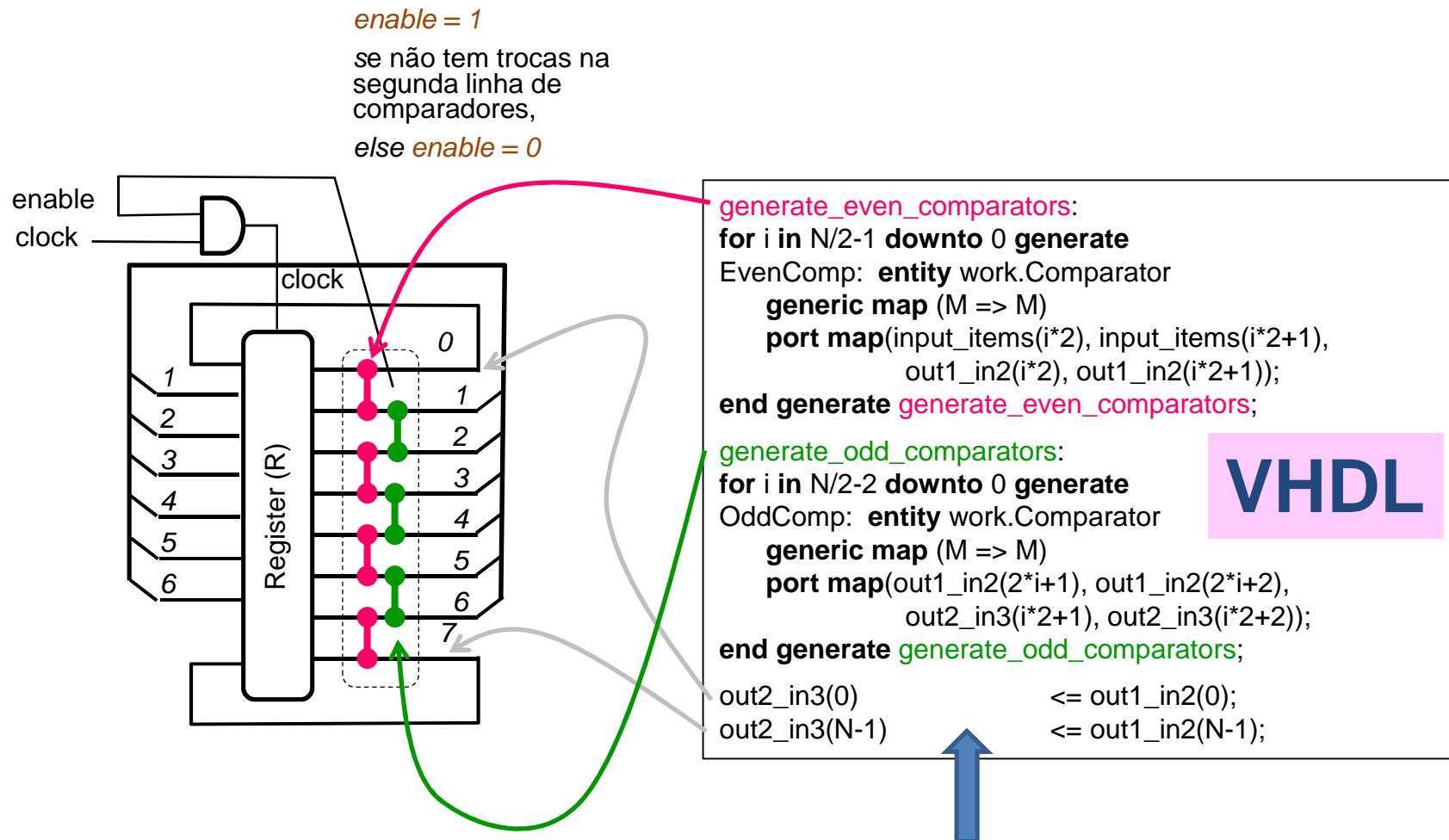
# Implementação de redes de ordenação (sorting networks)

Rede *even-odd transition* para oito variáveis





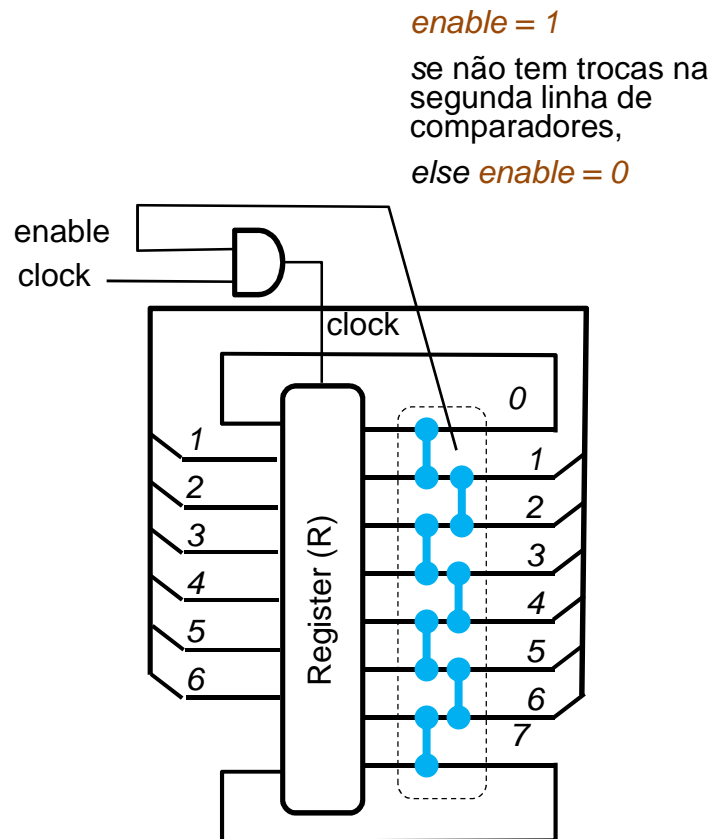
# Implementação de redes de ordenação (redes iterativas)



Para descrição da rede podemos utilizar: 1) descrição estrutural; 2) descrição comportamental; 3) descrição mista

A rede de busca pode ser descrita numa função e a rede de ordenação num procedimento

# Implementação de redes de ordenação (redes iterativas)



Descrição possível do circuito parametrizável  
para comparar/trocar dados:

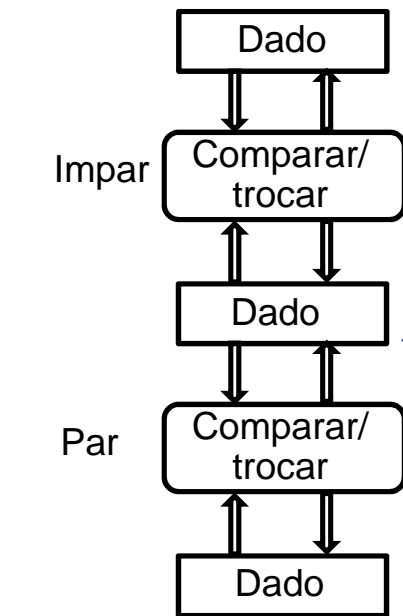
```
entity Comparator is
generic (    M          : integer := 4);
port(       Op1, Op2    : in std_logic_vector(M-1 downto 0);
          MaxValue      : out std_logic_vector(M-1 downto 0);
          MinValue      : out std_logic_vector(M-1 downto 0));
end Comparator;

architecture Behavioral of Comparator is
begin
process(Op1, Op2)
begin
    if Op1 >= Op2 then      MaxValue <= Op1; MinValue <= Op2;
    else                   MaxValue <= Op2; MinValue <= Op1;
    end if;
end process;
end Behavioral;
```

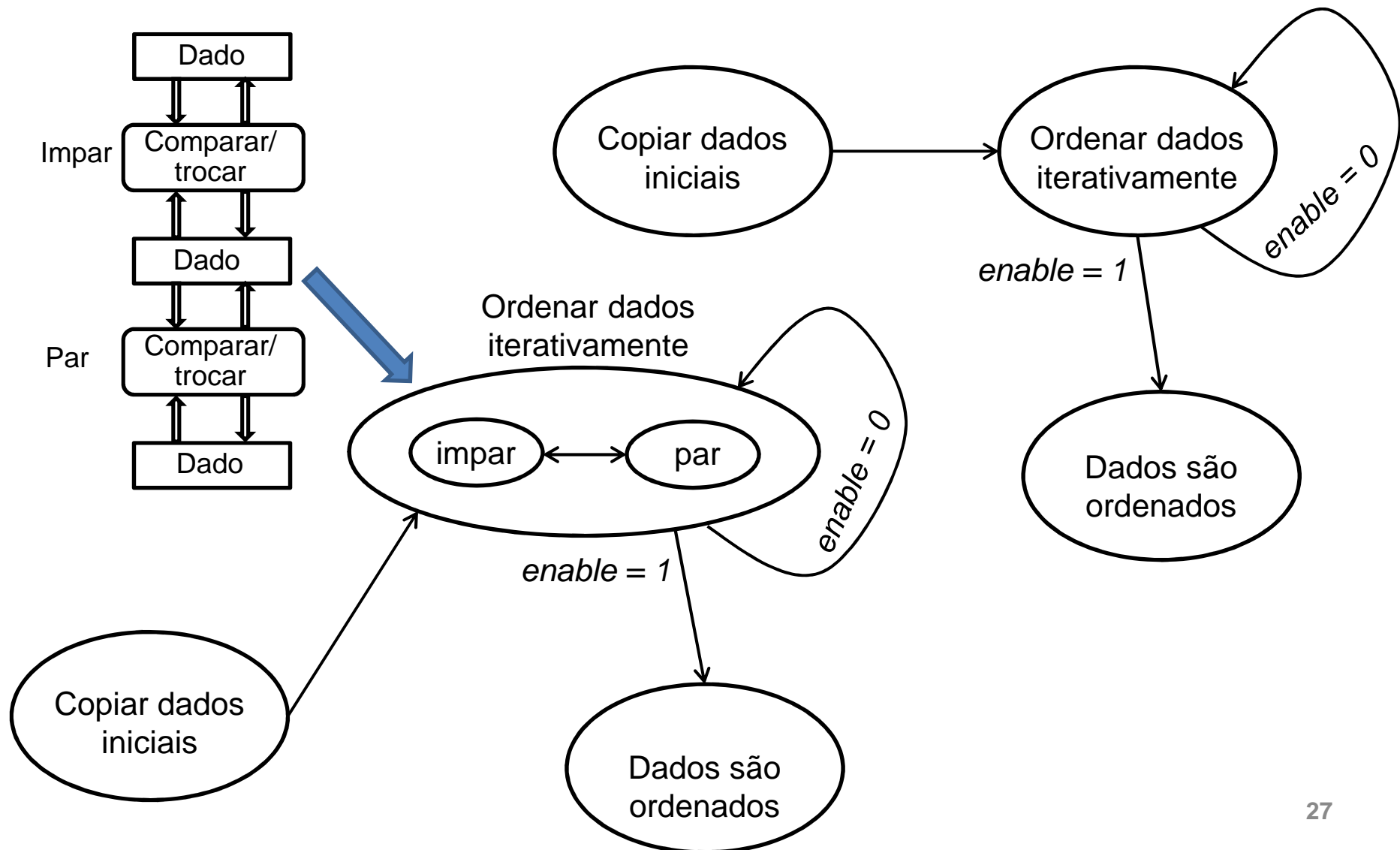
**VHDL**

# Implementação de redes de ordenação (redes iterativas)

Rede que tem só um nível



Máquina de estados finitos simples:



# Implementação de redes de ordenação (redes iterativas)



Exemplo

Vivado

# Implementação de redes de ordenação (redes iterativas)

```
library IEEE; use IEEE.std_logic_1164.all; use IEEE.STD_LOGIC_ARITH.ALL; use IEEE.std_logic_unsigned.all;
```

```
entity IterativeSorterFSM is
generic(      L:integer:=8; M          :integer:=8);
port (        clk           : in std_logic;
         reset            : in std_logic;
         led             : out std_logic_vector(7 downto 0);
         data_in          : in std_logic_vector(L*M-1 downto 0);
         data_out         : out std_logic_vector(L*M-1 downto 0));
end entity IterativeSorterFSM;
```

Código completo para  
ordenador de dados  
iterativo

```
architecture Behavioral of IterativeSorterFSM is
type state_type is (initial_state, even, odd, completed); -- enumeration type for the FSM states
signal C_S, N_S          : state_type;
type in_data is array (L-1 downto 0) of std_logic_vector(M-1 downto 0);
signal MyAr, N_MyAr       : in_data;
signal sorting_completed, N_sorting_completed : std_logic;
signal counter, N_counter : integer range 0 to 2*L-1 := 0;
begin
process (clk) -- this is a sequential process
begin
if rising_edge(clk) then
if (reset = '1') then C_S <= initial_state; counter <= 0; MyAr <= (others=>(others => '0'));
else
C_S <= N_S;
MyAr <= N_MyAr;
counter <= N_counter;
sorting_completed <= N_sorting_completed;

end if;
end if;
end process;
```

VHDL

Exemplo

# Implementação de redes de ordenação (redes iterativas)

```
process (C_S, data_in, sorting_completed, counter, MyAr) -- this is a combinational process
begin
  N_S                <= C_S;
  N_MyAr              <= MyAr;
  N_counter           <= counter;
  N_sorting_completed <= sorting_completed;
case C_S is
when initial_state => N_S <= even; N_sorting_completed <= '0'; N_counter <= 0;
  for i in L-1 downto 0 loop
    N_MyAr(i) <= data_in(M*(i+1)-1 downto M*i);
  end loop;
when even => N_S <= odd;
  if (sorting_completed = '0') then N_counter <= counter+1; N_sorting_completed <= '1';
    for i in 0 to L/2-1 loop
      if MyAr(2*i) < MyAr(2*i+1) then
        N_MyAr(2*i)  <= MyAr(2*i+1);      N_MyAr(2*i+1) <= MyAr(2*i);
      else null;
      end if;
    end loop;
  else N_S <= completed;
  end if;
when odd => N_S <= even;
  for i in 0 to L/2-2 loop
    if MyAr(2*i+1) < MyAr(2*i+2) then
      N_sorting_completed <= '0'; N_MyAr(2*i+1) <= MyAr(2*i+2); N_MyAr(2*i+2) <= MyAr(2*i+1);
    else null;
    end if;
  end loop;
when completed => N_S <= completed;
when others => N_S <= initial_state;
end case;
end process;
```

VHDL

Exemplo

# Implementação de redes de ordenação (redes iterativas)

```
process (MyAr)
begin
  for i in L-1 downto 0 loop
    data_out(M*(i+1)-1 downto M*i) <= MyAr(i);
  end loop;
end process;

led <= conv_std_logic_vector(counter,8);

end Behavioral;
```

**VHDL**

## Importante:

1. Utilização correta de variáveis e sinais.
2. Conversão de tipos.
3. Descrição da máquina.

**Exemplo**

# Implementação de redes de ordenação (redes iterativas)

```
library IEEE; use IEEE.STD_LOGIC_1164.all; use IEEE.STD_LOGIC_UNSIGNED.all;
```

```
entity embedded_ROM is
```

```
generic ( M      : integer := 16;
```

```
          L      : integer := 64;
```

```
          Dlog    : integer := 3);
```

```
port (      clk      : in std_logic;
```

```
          btnC       : in std_logic;
```

-- btnC é reset

```
          sw         : in std_logic_vector(3 downto 0);
```

```
          seg        : out std_logic_vector (6 downto 0);
```

```
          sel_disp   : out std_logic_vector (7 downto 0) );
```

```
end embedded_ROM;
```

```
architecture Behavioral of embedded_ROM is
```

```
    signal address      : std_logic_vector(Dlog-1 downto 0);
```

```
    signal data_in      : std_logic_vector(L*M-1 downto 0);
```

```
    signal data_out     : std_logic_vector(L*M-1 downto 0);
```

```
    signal scan         : integer range 0 to L-1;
```

```
    signal count        : std_logic_vector(7 downto 0);
```

```
component blk_mem_gen_1 is
```

```
    port (      clka      : in std_logic;
```

```
            addra       : in std_logic_vector(Dlog-1 downto 0);
```

```
            douta       : out std_logic_vector(L*M-1 downto 0) );
```

```
end component;
```

```
    signal bin_i          : std_logic_vector(M-1 downto 0);
```

```
    signal BCD4, BCD3, BCD2, BCD1, BCD0 : std_logic_vector(3 downto 0);
```

```
    signal C_MS, C_Middle, C_LS        : std_logic_vector(3 downto 0);
```

```
    signal divided_clk      : std_logic;
```

```
begin
```

VHDL

Exemplo



# Implementação de redes de ordenação (redes iterativas)

```
address <= sw(Dlog-1 downto 0);
block_ROM :          blk_mem_gen_1
                    port map (clk, address, data_in );
div:      entity work.clock_divider
          port map ( clk, '0',divided_clk);
disp_cont: entity work.EightDisplayControl
          port map ( clk=>clk, leftL=>C_MS,          near_leftL=>C_Middle,
                    near_rightL=>C_LS,   rightL=>BCD4,
                    leftR=>BCD3, near_   leftR=>BCD2,
                    near_rightR=>BCD1,  rightR=>BCD0,
                    select_display=>sel_disp,segments=>seg);
bin_i <=  data_in(M*(scan+1)-1 downto M*scan) when sw(3) = '0' else
          data_out(((scan+1)*M)-1 downto scan*M);
scan <= scan+1 when rising_edge(divided_clk);
BCD_dec: entity work.BinToBCD16
        port map ( clk => clk,reset => '0', ready => open, binary => bin_i, request => '1',
                  BCD4 => BCD4, BCD3 => BCD3,BCD2 => BCD2,BCD1 => BCD1,BCD0 => BCD0);
sorter:  entity work.IterativeSorterFSM
        generic map (L => L, M => M)
        port map ( clk=> clk, reset => btnC, led => count, data_in => data_in, data_out => data_out);
BCD_count: entity work.BinToBCD8
          port map ( clk      => clk,
                    reset    => '0',
                    ready    => open,
                    binary   => count,
                    BCD2     => C_MS,
                    BCD1     => C_Middle,
                    BCD0     => C_LS);

end Behavioral;
```

## Importante:

1. Parametrização.
2. Utilização de relógio.

**VHDL**

**Exemplo**

# Geração do ficheiro COE de Java

```
import java.util.*; import java.io.*;
public class Random_to_file {
    static Random rand = new Random();
    static final int M = 16;           // M is the size of one word. DO NOT FORGET TO CHANGE FORMAT WHEN CHANGE M
    static final int L = 256;         // L is the number of blocks
    static final int D = 8;           // D is the depth of memory (how many addresses are allocated for the memory)
    public static void main (String args[]) throws IOException {
        int a[] = new int[L];         // the size of input data
        File fout = new File("coe_from_java256.coe");
        PrintWriter pw = new PrintWriter(fout);
        pw.println("memory_initialization_radix = 16;");
        pw.println("memory_initialization_vector = ");
        for(int d = 0; d < D; d++) {
            for(int i = 0; i < a.length; i++) a[i] = rand.nextInt(16383);
            for(int l=0; l<L-1; l++) pw.printf("%04x",a[l]); // CHANGE FORMAT HERE 4 = intlog M
            pw.printf((d == D-1) ? "%04x;" : "%04x",a[L-1]); // CHANGE FORMAT HERE 4 = intlog M
            pw.println();
        }
        pw.close();
    }
}
```

Java

Exemplo

# Comparação de software e hardware

Java

```
import java.util.*;    import java.io.*;
public class SortMeasureTime {
    static Random rand = new Random();
    static final int L = 64;
public static void main (String args[]) throws IOException {
    int a[] = new int[L];
    for(int i = 0; i < L; i++) a[i] = rand.nextInt(16383);
    long time=System.nanoTime();
    Arrays.sort(a);
    long time_end=System.nanoTime();
    System.out.printf("measured time (in microseconds): %.3f\n", (double)(time_end-time)/1000.);
}
}
```

// L is the size of data

// Measuring initial time in nanoseconds

// java sorting

// Measuring final time in nanoseconds



measured time (in microseconds): 109.056  
Press any key to continue . . . \_

Tempo em  
ciclos de  
relógio



Frequência é 100 MHz, período é 10 ns

Então ordenação foi feita durante 600 ns, i.e. 181 vezes mais rápido que no computador (3.4 GHz)

# Computação de popcount (peso e distância de Hamming)

Usado na área de bioinformática, informática química, procura combinatória, telecomunicações, etc.

1. O peso  $w$  de *Hamming* do vetor binário é o número de uns no vetor.
2. A distância de *Hamming*  $d(A,B)$  entre dois vetores A e B é o número de elementos em que os vetores são diferentes:  $d(A,B) = w(A \text{ XOR } B)$
3. Comparadores de pesos de *Hamming* são usados em várias tarefas de filtragem.
4. Por exemplo,  $w(A)$  frequentemente deve ser comparado com um limite (*threshold*)  $\kappa$ , ou com  $w(B)$ , onde  $B = \{b_0, \dots, b_{Q-1}\}$  é um outro vetor binário e estes vetores podem ter tamanhos iguais ou diferentes.

$A \rightarrow "01010011101100": w(A) = 7. B \rightarrow "01001000101101": w(B) = 6.$

$d(A,B) = w(A \text{ XOR } B) = w("00011011000001") = 5$

# Computação de popcount (peso e distância de Hamming)

## 1. Utilização de processo combinatório:

```
cp3: process(input)
variable HammingWeightCount : integer range 0 to 1023;
begin
    HammingWeightCount := 0;
    for i in input'range loop
        if input(i) = '1' then HammingWeightCount := HammingWeightCount+1;
        end if;
    end loop;
    led <= conv_std_logic_vector(HammingWeightCount,16);
end process cp3;
```

- a) Onde **cp3** é uma etiqueta opcional;
- b) O objeto **HammingWeightCount** deve ser uma variável e não pode ser um sinal.
- c) Vetor *input* pode ser lido do ficheiro *COE*.
- d) Saída *led* pode ser ligada com leds ou com displays de segmentos.
- e) Desvantagem: profundidade grande do circuito combinatório com atrasos respetivos.
- f) Quando o processo vai calcular o peso de *Hamming* sequencialmente não pode usar uma frequência alta. Para o exemplo em cima e placa Nexys-4 menos que 1 MHz.
- g) Recursos do circuito são elevados.

# Computação de popcount (peso e distância de Hamming)

## 2. Utilização numa função:

```
function HammingWeight (input: std_logic_vector) return integer is  
    variable HammingWeightCount : integer range 0 to input'length;  
begin  
    HammingWeightCount := 0;  
    for i in input'range loop  
        if input(i) = '1' then HammingWeightCount := HammingWeightCount+1;  
        end if;  
    end loop;  
    return HammingWeightCount;  
end HammingWeight;
```

- a) O objeto HammingWeightCount deve ser uma variável e não pode ser um sinal.
- b) Vetor *input* pode ser lido do ficheiro COE.
- c) Saída *led* pode ser ligada com leds ou com displays de segmentos.
- d) Desvantagem: profundidade grande do circuito combinatório para vetores grandes.
- e) Vantagem: parametrização através de utilização do argumento e o resultado sem limites (*unconstrained inputs and outputs*).

## Computação de popcount (peso e distância de Hamming)

3. Utilização duma máquina de estados finitos (ver aula teórica 5 – slide 17).

- a) Vetor de entrada pode ser lido do ficheiro COE.
- b) Saída pode ser ligada com leds ou com displays de segmentos.
- c) Desvantagem: o número elevado de ciclos de relógio.
- d) Vantagem: parametrização extremamente simples.
- e) Vantagem: pode usar frequência alta.

Métodos melhores para contar o peso de *Hamming* em FPGAs são baseados em:

- 1. Somadores.
- 2. LUTs.
- 3. Redes especiais (*counting networks*).
- 4. FPGA DSPs – *digital signal processing slices*

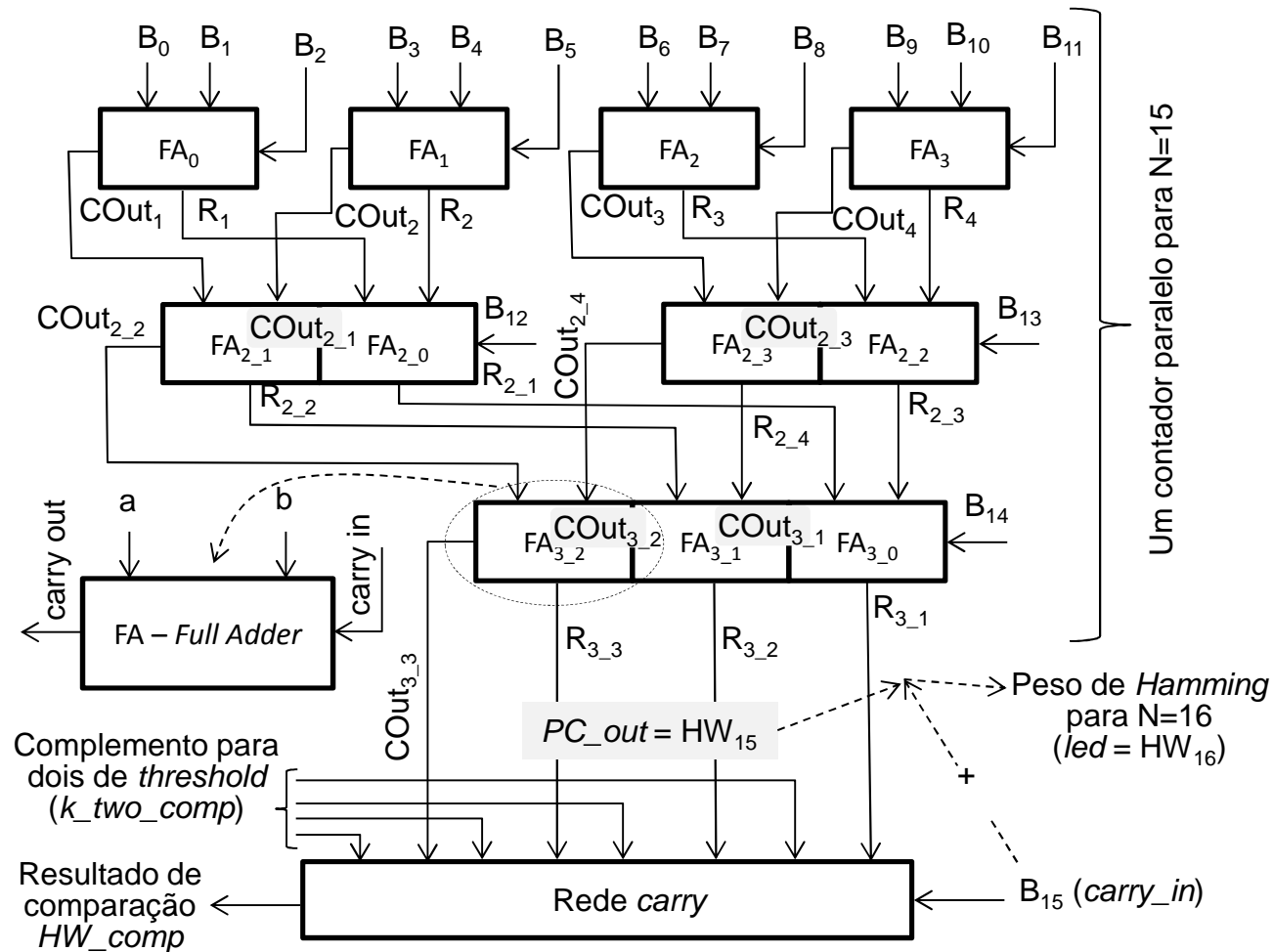
## Utilização de somadores (contadores paralelos)

Informação adicional:

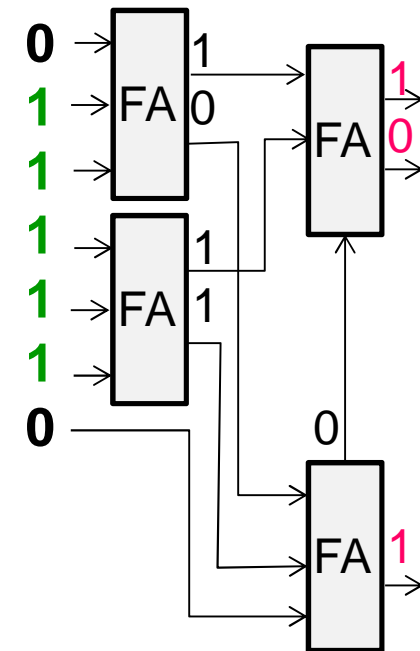
- 1. B. Parhami, *Efficient Hamming Weight Comparators for Binary Vectors Based on Accumulative and Up/Down Parallel Counters*, *IEEE Trans. on Circuits and Systems—II: Express Briefs*, vol. 56, no. 2, pp. 167-171, Feb. 2009.
- 2. V. Sklyarov, I. Skliarova, A. Barkalov, L. Titarenko. *Synthesis and Optimization of FPGA-Based Systems*. Springer, 2014.

# Computação de popcount (peso e distância de Hamming)

## Utilização de somadores (contadores paralelos)



Ideia:





# Computação de popcount (peso e distância de Hamming)

## Descrição de FA (*Full Adder*)

**entity** FullAdder **is**

**port**(    A                   : **in** std\_logic;  
          B                   : **in** std\_logic;  
          CarryIn            : **in** std\_logic;  
          Result             : **out** std\_logic;  
          CarryOut           : **out** std\_logic);

**end** FullAdder;

**architecture** Behavioral **of** FullAdder **is**  
**begin**

CarryOut <= (A **and** B) **or** (A **and** CarryIn) **or** (B **and** CarryIn);

Result    <= A **xor** B **xor** CarryIn;

**end** Behavioral;

## Utilização de somadores (código misto completo)

```
library IEEE; use IEEE.std_logic_1164.all; use IEEE.std_logic_unsigned.all;
entity ParallelCounterComparator is
    port (
        sw      : in std_logic_vector(15 downto 0);
        led     : out std_logic_vector(4 downto 0) );
end ParallelCounterComparator;
```

**architecture Behavioral of ParallelCounterComparator is**

```
    signal COut1, COut2, COut3, COut4      : std_logic;
    signal COut2_1, COut2_2, COut2_3, COut2_4 : std_logic;
    signal COut3_1, COut3_2, COut3_3      : std_logic;
    signal B                                : std_logic_vector(15 downto 0);
    signal PC_out                           : std_logic_vector(3 downto 0);
    signal R1, R2, R3, R4, R2_1, R2_2, R2_3, R2_4, R3_1, R3_2, R3_3 : std_logic;
```

**begin**

```
B <= sw;
```

```
FA0 : entity work.FullAdder
```

```
FA1 : entity work.FullAdder
```

```
FA2 : entity work.FullAdder
```

```
FA3 : entity work.FullAdder
```

```
FA2_0 : entity work.FullAdder
```

```
FA2_1 : entity work.FullAdder
```

```
FA2_2 : entity work.FullAdder
```

```
FA2_3 : entity work.FullAdder
```

```
FA3_0 : entity work.FullAdder
```

```
FA3_1 : entity work.FullAdder
```

```
FA3_2 : entity work.FullAdder
```

```
PC_out <= COut3_3 & R3_3 & R3_2 & R3_1;
```

```
led <= PC_out + ("0000" & B(15));
```

**end Behavioral;**

```
port map(B(0), B(1), B(2), R1, COut1);
```

```
port map(B(3), B(4), B(5), R2, COut2);
```

```
port map(B(6), B(7), B(8), R3, COut3);
```

```
port map(B(9), B(10), B(11), R4, COut4);
```

```
port map(R1, R2, B(12), R2_1, COut2_1);
```

```
port map(COut1, COut2, COut2_1, R2_2, COut2_2);
```

```
port map(R3, R4, B(13), R2_3, COut2_3);
```

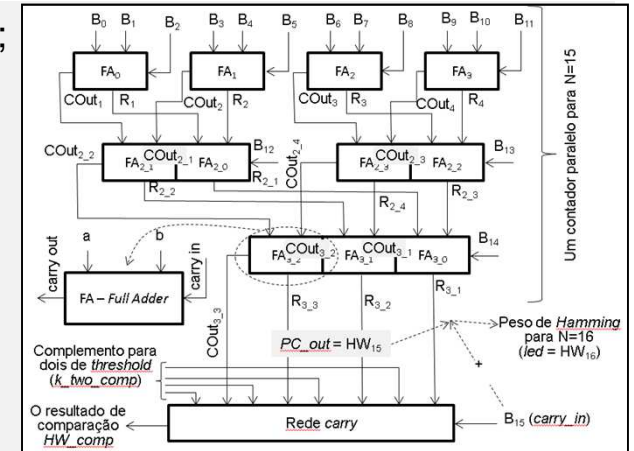
```
port map(COut3, COut4, COut2_3, R2_4, COut2_4);
```

```
port map(R2_1, R2_3, B(14), R3_1, COut3_1);
```

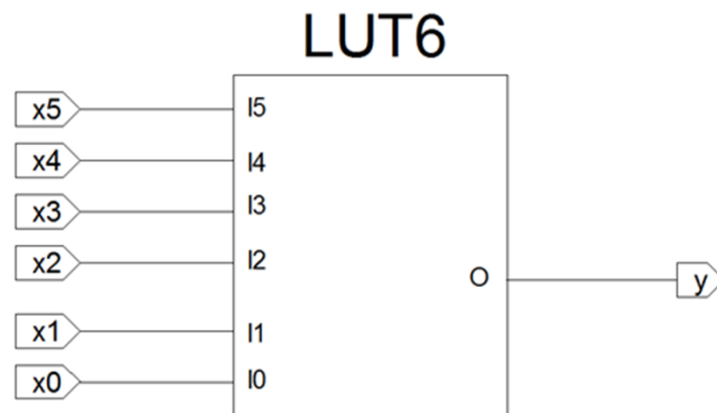
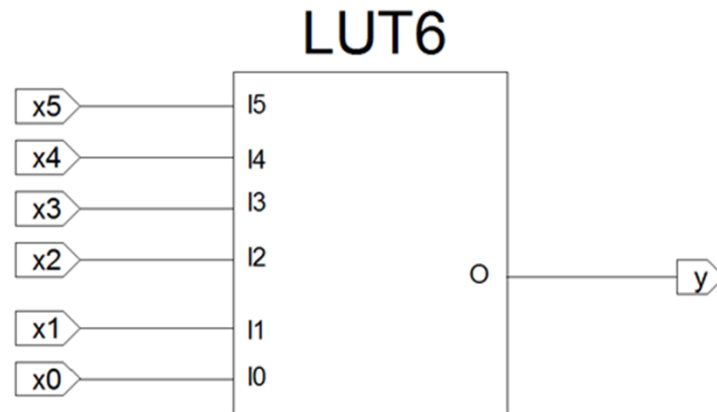
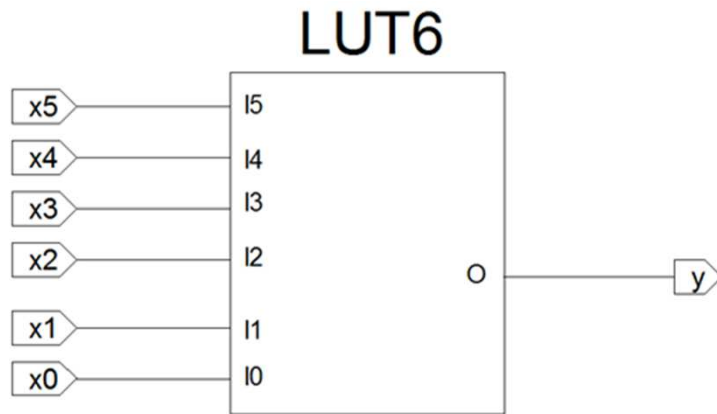
```
port map(R2_2, R2_4, COut3_1, R3_2, COut3_2);
```

```
port map(COut2_2, COut2_4, COut3_2, R3_3, COut3_3);
```

-- peso de Hamming



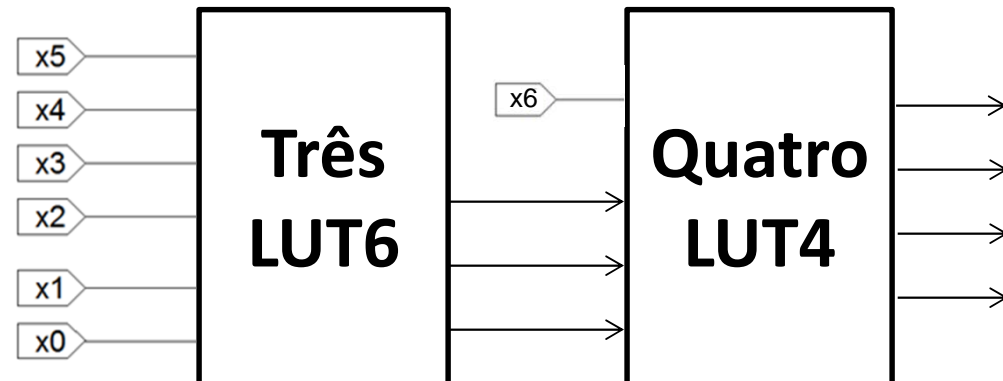
# Computação de popcount (peso e distância de Hamming)



## Utilização de LUTs

Três *LUTs* (i.e. menos que um *slice*) podem ser usadas para encontrar o peso de *Hamming* de qualquer vetor binário com 6 bits

Como encontrar o peso de *Hamming* para vetores binários com mais de que 6 bits



# Computação de popcount (peso e distância de Hamming)

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity LUT_6to3 is
  port (
    sw      : in  std_logic_vector (5 downto 0);
    led     : out std_logic_vector (2 downto 0);
  end LUT_6to3;

  architecture Behavioral of LUT_6to3 is

    type LUT is array (2 downto 0) of std_logic_vector(63 downto 0);
    constant conf_LUT : LUT := (
      X"fee8e880e8808000",
      X"8117177e177e7ee8",
      X"6996966996696996");

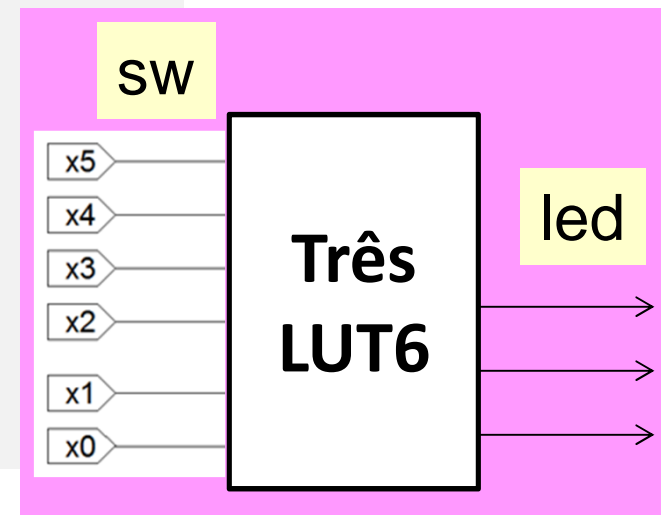
  begin

    led      <=      conf_LUT(2)(conv_integer(sw)) &
                    conf_LUT(1)(conv_integer(sw)) &
                    onf_LUT(0)(conv_integer(sw));

    -- alternativamente o código seguinte pode ser usado
    --gen: for i in conf_LUT'range generate
    --      led(i) <= conf_LUT(i)(conv_integer(sw));
    --end generate gen;

  end Behavioral;
```

## Utilização de LUTs



# Como utilizar projetos disponíveis em elearning.ua.pt

1. Só os projetos que usam IP cores (memórias *distributed* ou *block*) estão disponíveis completamente.
2. A partir da próxima aula (aula 8) para projetos com IP cores só vão ser disponibilizados códigos VHDL e IP cores devem ser gerados por alunos.

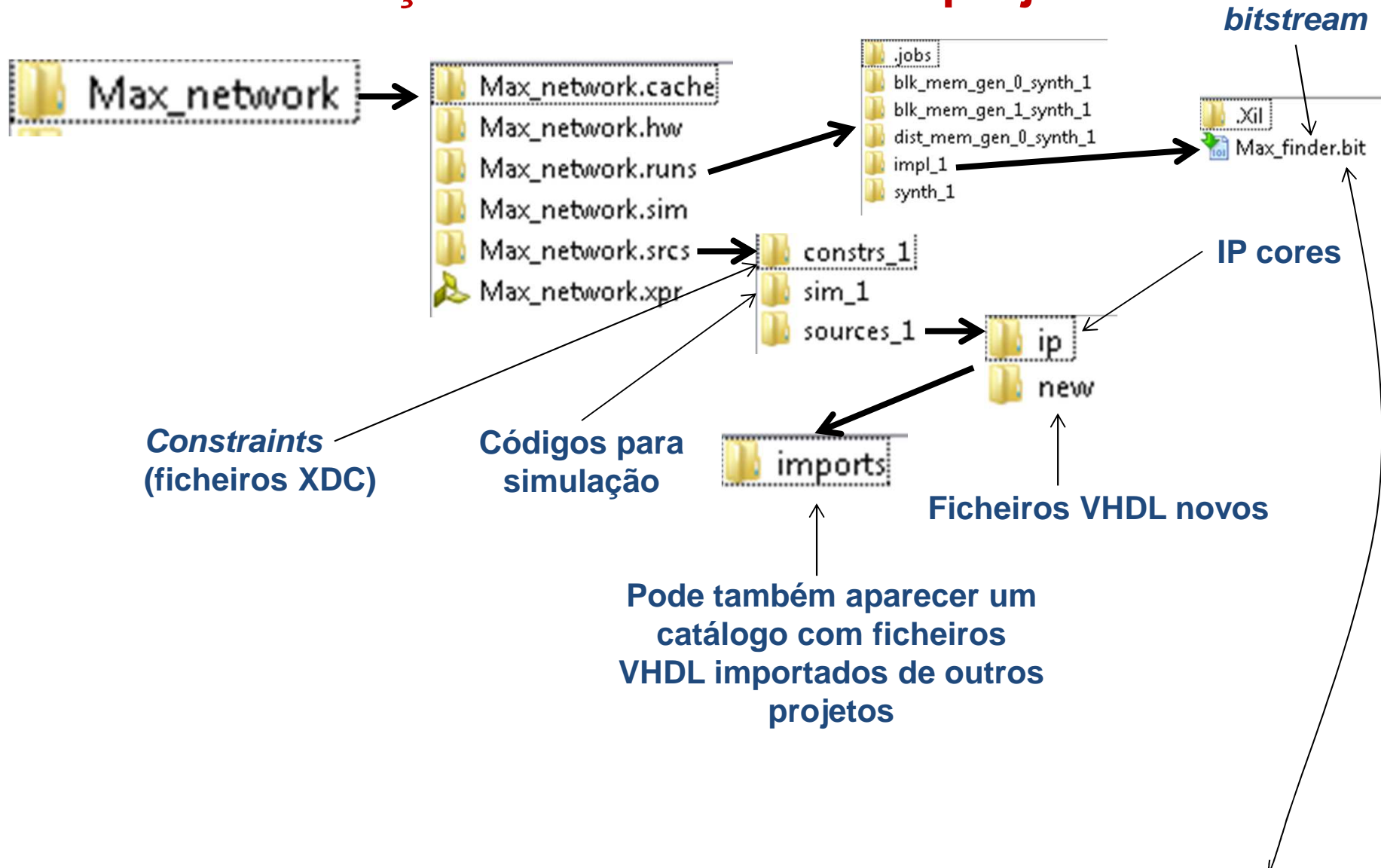
Use projetos para que só VHDL códigos estão disponíveis de modo seguinte:

1. Criar um projeto novo.
2. Adicionar no projeto ficheiros com códigos VHDL disponíveis.
3. Adicionar o ficheiro XDC completo com todas as linhas comentadas.
4. Remover comentários para linhas com *constraints* correspondentes aos portos do módulo VHDL do nível de topo, por exemplo, para entradas e saídas indicadas em baixo por **cor vermelha**:

```
entity LUT_6to3 is
  port (   sw           : in std_logic_vector (5 downto 0);
          led          : out std_logic_vector (2 downto 0));
end LUT_6to3;
```

5. Sintetizar o circuito, fazer implementação e gerar *bitstream*.
6. Ligar placa, abrir *target*, e configurar a FPGA. Verificar o projeto.
7. Não tentar abrir *target* ou programar a placa que está desligada. Pode aparecer um problema e para resolver este problema é necessário sair do Vivado e executar Vivado mais uma vez. Depois abrir *target* para a placa ligada e configurar a FPGA.
8. Um problema frequente é utilização de diretórios com longo caminho. Para evitar este problema não use caminhos longos ou copie o *bitstream* no *Decktop* e configure a FPGA utilizando o *bitstream* da *Decktop*.

# Informação sobre diretórios do projeto



**Bitstream** para configurar FPGA tem o tipo **bit**, por exemplo, **Max\_finder.bit**

## Exame prático: dias 18 e 19 de abril

1. Duração é 1h30m. Peso 30%.
2. Trabalhos podem ser feitos em grupos (máximo 2 alunos).
3. O tempo para esta avaliação não pode ser superior a 1h30m.
4. Entrega de trabalhos deve ser feita por e-mail [skl@ua.pt](mailto:skl@ua.pt) durante 10 minutos depois de 1h30m. Assunto é **CR exame prático** e **números. mecanográficos** dos alunos. Pode enviar o projeto completo em único ficheiro *rar* ou *zip*.
5. Aceito os resultados de simulação ou implementação para Nexys-4.
6. O ficheiro XDC deve ser preparado obrigatoriamente independentemente do tipo de projeto (ver p. 5).
7. As tarefas são baseadas em aulas teóricas 1-9 e aulas práticas 1-8.
8. A aula teórica 9 é uma aula de revisão da matéria anterior com muitos exemplos práticos úteis.
9. Só uma tarefa vai ser proposta mas implementação deve ser feita de acordo com os requisitos. Por exemplo, quando o requisito é implementação de **procedimento** ou **código estrutural** não pode entregar um **processo** ou **código comportamental**. Penalização é 100%.

## Exame prático: exemplo:

1. Gerar vetores de 128 bits aleatoriamente.
2. Extrair do vetor palavras de 8 bits.
3. Fazer ordenação de palavras utilizando uma **rede iterativa com 2 níveis de comparadores**. Usar **descrição comportamental e uma máquina de estados finitos** para a rede.  $\Rightarrow$  **simulação**
4. Mostrar os resultados em leds em binário sequencialmente com frequência  $\sim 1$  Hz.  $\Rightarrow$  **simulação**
5. Mostrar os resultados em displays em hexadecimal com frequência  $\sim 1$  Hz.
6. Entregar o projeto com os estímulos para simulação comportamental ou com *bitstream* para implementação na placa Nexys-4.
7. No caso de estímulos só entregue o código VHDL para o ponto 5 sem verificação do projeto na placa.
8. O ficheiro final XDC deve ser entregue sem compromissos.

Pode mostrar o funcionamento do projeto até ao final da avaliação. Se o tempo permitir, o projeto pode ser avaliado até ao final da avaliação

Só é permitido utilizar código para controlar displays de segmentos e para divisor de frequência