

Laboratório de Sistemas Digitais

Aula Teórico-Prática 11

Ano Letivo 2017/18

Bibliotecas, *Packages* e Funções em VHDL
Records em VHDL

Resumo dos Tipos de Dados em VHDL
Macros/Funções de Conversão entre Tipos

Arnaldo Oliveira, Augusto Silva, Guilherme Campos, Tomás Oliveira e Silva



Universidade
de Aveiro

Conteúdo

- Bibliotecas (*libraries*) e Pacotes (*packages*) em VHDL para definição de
 - Tipos e constantes
 - Funções (*functions*) e procedimentos (*procedures*)
- *Records* em VHDL (vs. *Arrays*)
- Resumo dos tipos de dados em VHDL
- Modelação e utilização de situações de alta impedância (*tri-state*)
- Macros/Funções de conversão entre tipos

Bibliotecas e Pacotes em VHDL

- Uma biblioteca (*library*) consiste num ou mais pacotes (*packages*)
 - Uma biblioteca contém definições (tipos, constantes, componentes, funções, etc.) úteis para vários projetos
 - Um pacote contém definições úteis para vários projetos e/ou módulos
- Bibliotecas standard
 - STD
 - IEEE
- Library WORK
 - Biblioteca do projeto “atual”
- Library ALTERA_MF
 - Biblioteca da Altera com definição de Megafunctions (blocos pré-definidos)

- Packages VHDL standard

```
library STD;  
use STD.textio;
```

```
library IEEE;  
use IEEE.std_logic_1164.all;  
use IEEE.numeric_bit.all;  
use IEEE.numeric_std.all;  
use IEEE.std_logic_signed.all;  
use IEEE.std_logic_unsigned.all;  
use IEEE.std_logic_textio.all;  
use IEEE.std_logic_arith.all;  
use IEEE.math_real.all;  
use IEEE.math_complex.all;
```

Síntese

Simulação

Pouco usadas/deprecated/
não recomendadas atualmente

Exemplo de *Package Header* c/ Tipos, Constantes e Protótipos de Funções

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

Ficheiro
LSDPack.vhd

```
package LSDPack is
```

```
    subtype Tbyte      is std_logic_vector(7 downto 0);  
    subtype TWord      is std_logic_vector(31 downto 0);
```

```
    constant WORD_MAX : TWord := (others => '1');
```

```
    function min(a, b : integer) return integer;  
    function boolean2std_logic(v : boolean) return std_logic;
```

```
end LSDPack;
```

Para evitar situações de síntese “complicadas”, vamos restringir a utilização de funções a cálculos auxiliares, com parâmetros de entradas constantes (conhecidos em *compile/synthesis time*)

Exemplo de *Package Body* com a Implementação de Funções

```
package body LSDPack is
```

```
  function min(a, b : integer) return integer is
```

```
  begin
```

```
    if (a < b) then
```

```
      return a;
```

```
    else
```

```
      return b;
```

```
    end if;
```

```
  end min;
```

```
  function boolean2std_logic(v : boolean) return std_logic is
```

```
  begin
```

```
    if (v = false) then
```

```
      return '0';
```

```
    else
```

```
      return '1';
```

```
    end if;
```

```
  end boolean2std_logic;
```

```
end LSDPack;
```

Para evitar situações de síntese “complicadas”, vamos restringir a utilização de funções a cálculos auxiliares, com parâmetros de entradas constantes (conhecidos em *compile/synthesis time*)

Ficheiro
LSDPack.vhd

Utilização de *Packages* em Módulos VHDL e Outras *Packages*

- Alguns exemplos

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

```
library WORK;  
use WORK.LSDPack.all;
```

(se a *package* LSDPack estiver definida no projeto atual)

```
library LSD;  
use LSD.LSDPack.all;
```

(se a *package* LSDPack estiver definida noutra biblioteca independente do projeto, e.g. LSD)

Tipos de Dados Agregados em VHDL

- Tipos de dados capazes de armazenar diversos valores
 - Podem ser usados como portos, sinais ou constantes
 - Declarados em *packages* ou na parte declarativa de *architectures*
 - *Arrays*
 - Todos os elementos são do mesmo tipo
 - Exemplos
 - `std_logic_vector`
 - Arrays usados na definição das memórias
 - *Records*
 - Os elementos podem ser de diversos tipos (semelhante a uma *struct* em C/C++/Java)
 - Os vários elementos de um *record* designam-se campos
 - Exemplos
 - Agregação de vários sinais de controlo, estado, endereço e dados de um barramento
 - Agregação de um conjunto de valores correlacionados (e.g. parâmetros de cada modo VGA – fornecido na package “`vga_config.vhd`” do “Material de apoio aos projetos finais (pacote completo)”)

Exemplos de Records em VHDL

(copiados do código fonte fornecido no “Material de apoio aos projetos finais (pacote completo)”)

```
type vga_rgb_t is record
    r : std_logic_vector(7 downto 0);
    g : std_logic_vector(7 downto 0);
    b : std_logic_vector(7 downto 0);
end record vga_rgb_t;
```

```
type vga_data_t is record
    h_sync      : std_logic;
    v_sync      : std_logic;
    blank_n     : std_logic;
    x           : vga_x_t;
    y           : vga_y_t;
    end_of_line  : std_logic;
    end_of_frame : std_logic;
end record vga_data_t;
```

- Aspectos a analisar
 - Definição dos *records* (ver *package* “vga_config.vhd”)
 - Declaração de *records* como
 - Constantes (ver *package* “vga_config.vhd”)
 - Sinais (ver módulo “ir_vga_logic_analyzer_tl.vhd”)
 - Portos (ver módulo “vga_controller.vhd”)
 - Sintaxe para atribuição/leitura de valores dos diversos campos (ver *package* “vga_config.vhd”)

Resumo dos Tipos de Dados em VHDL

- Vamos de seguida resumir a definição e utilização típica dos seguintes tipos de dados em VHDL
 - `std_logic(_vector)`
 - `bit(_vector)`
 - `integer`
 - `natural`
 - `positive`
 - `unsigned`
 - `signed`
 - `boolean`
 - `character`
 - `string`
 - `real`
 - `time`
 - Tipos enumerados

... começando pelo `std_logic(_vector)`

std logic 1164 – Valores Predefinidos

“Resolved” standard logic type - std_logic / std_logic_vector

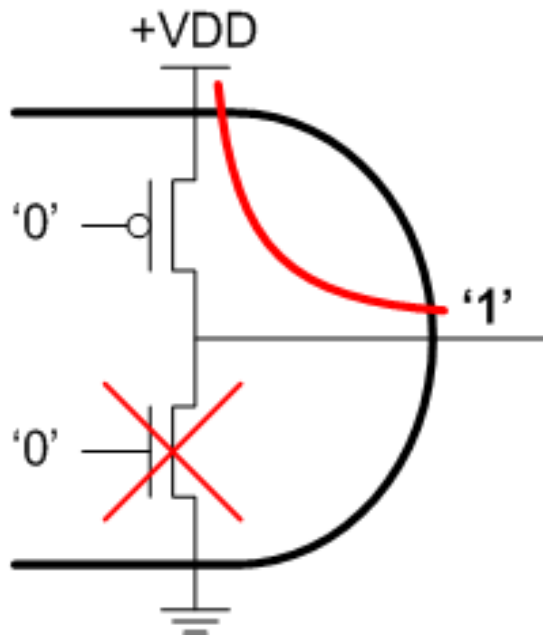
- ‘0’ – “Strong” Low level (active driver / saída ativa)
- ‘1’ – “Strong” High level (active driver / saída ativa)
- ‘L’ – “Weak” Low level (passive driver / saída passiva)
- ‘H’ – “Weak” High level (passive driver / saída passiva)
- ‘Z’ – Tristate (high impedance / alta impedância)
- ‘-’ – Don’t care (útil para especificar condições de don’t care para efeitos de otimização)

Valores geridos pelo simulador (não atribuídos explicitamente!)

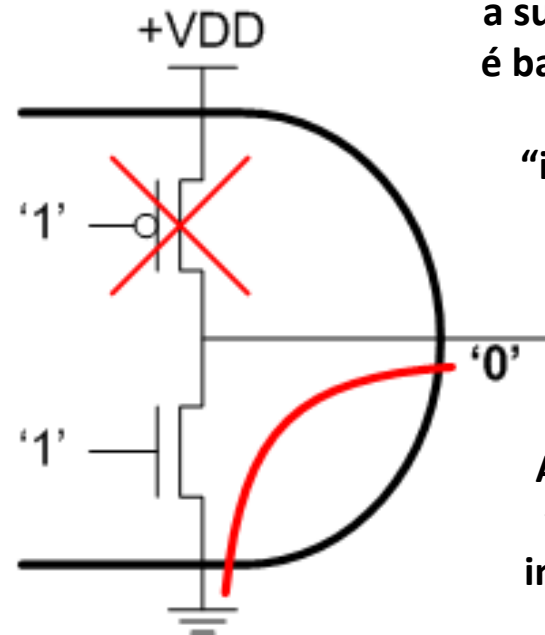
- ‘X’ – “Strong” Conflict (devido a active drivers)
- ‘W’ – “Weak” Conflict (devido a passive drivers)
- ‘U’ – Uninitialized (útil para detetar sinais não inicializados durante a simulação – valor inicial por omissão de um std_logic)

std logic 1164 – Strong (Active) Drivers

'1' – Strong High Level



'0' – Strong Low Level



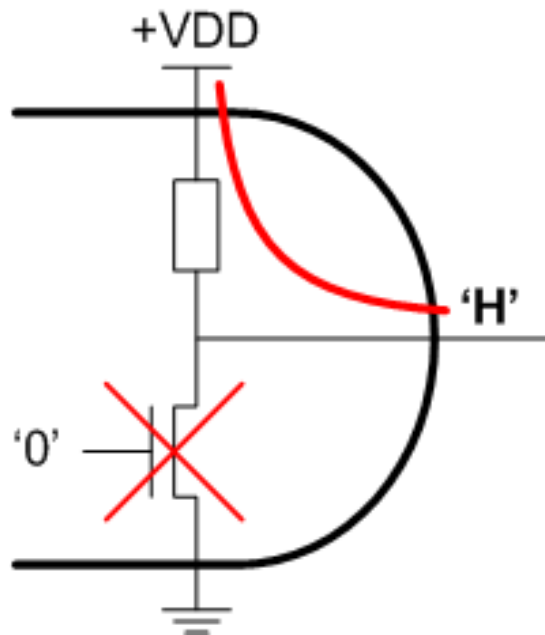
Quando o transistor conduz a sua resistência é baixa, atuando como um "interruptor fechado"

As linhas a vermelho indicam por onde flui corrente

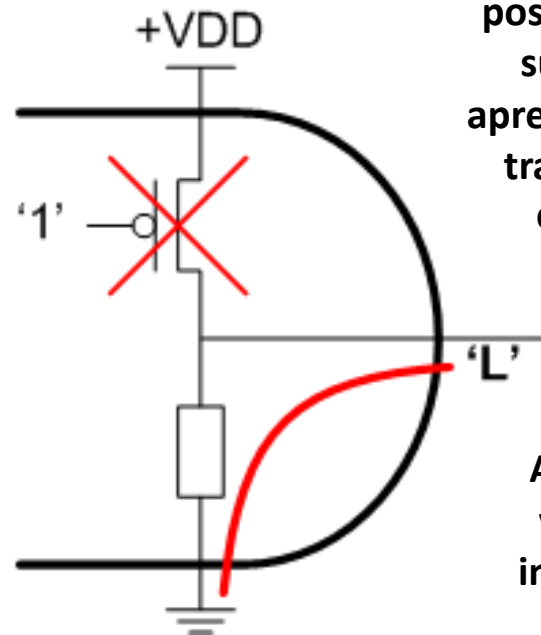
Podem ser testados (incluído em condições) e usados em atribuições a sinais, portas e variáveis

std logic 1164 –Weak (Passive) Drivers

'H' – Weak High Level



'L' – Weak Low Level

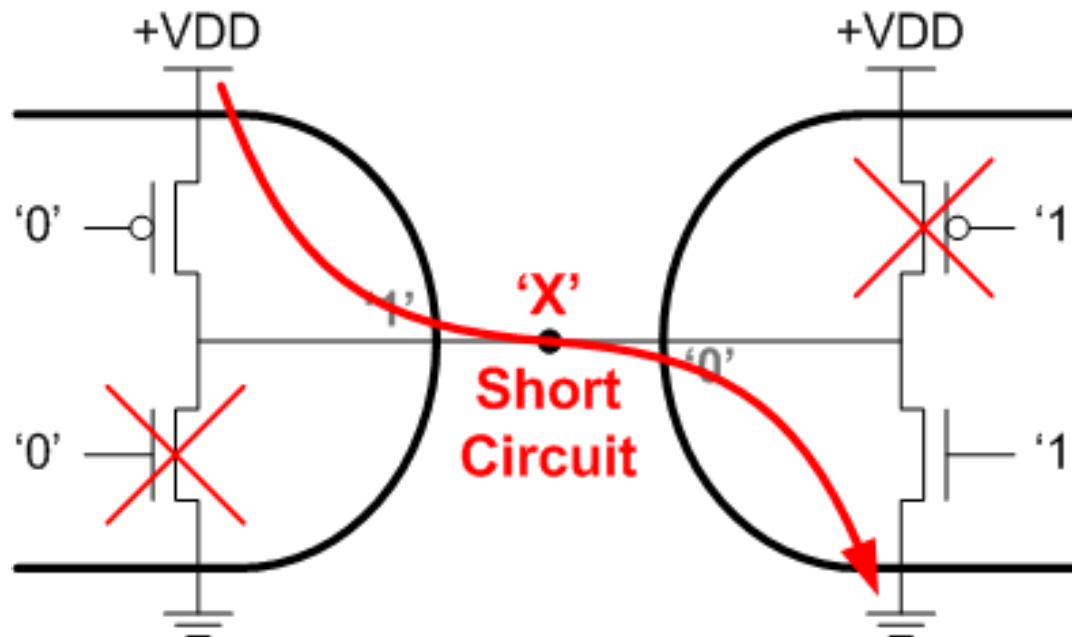


A resistência de pull-up/down possui um valor superior ao apresentado pelo transistor em condução

As linhas a vermelho indicam por onde flui corrente

Podem ser usados em atribuições a sinais, portas e variáveis
Devem ser testados (em condições) como '0' ou '1'

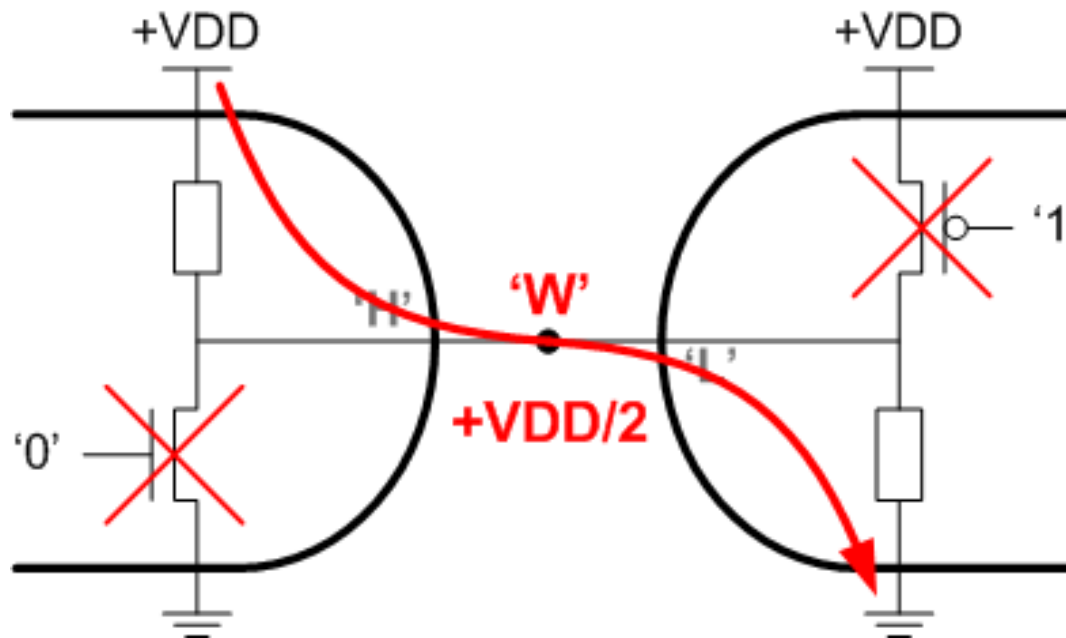
std logic 1164 – Conflitos Ativos ('X')



Gerado pelo simulador devido a conflitos entre *drivers* ativos ou como resultado da propagação de valores *Uninitialized*

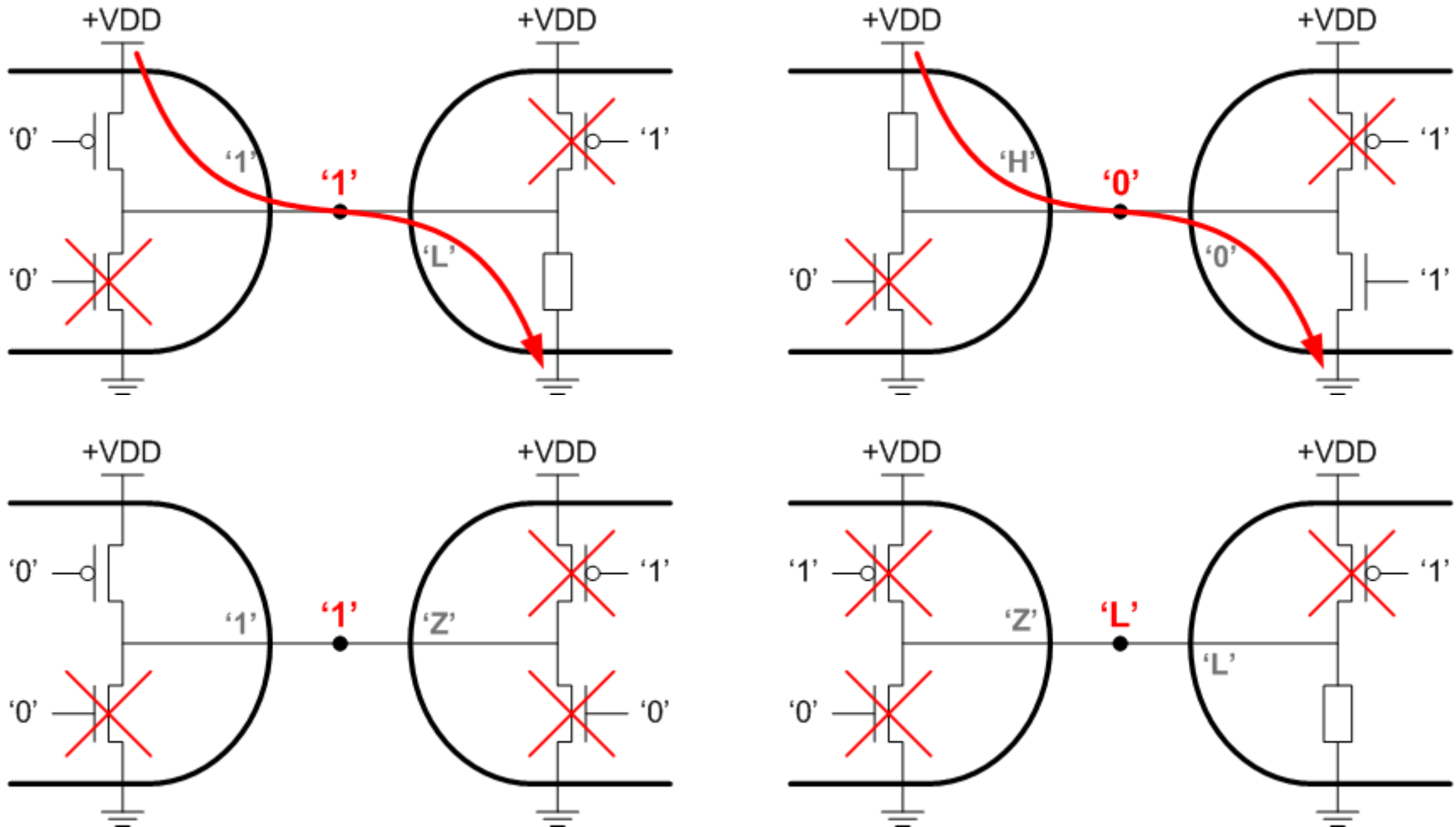
Não faz sentido ser testado (em condições) nem usado em atribuições a sinais, portos ou variáveis

std logic 1164 – Conflitos Passivos ('W')

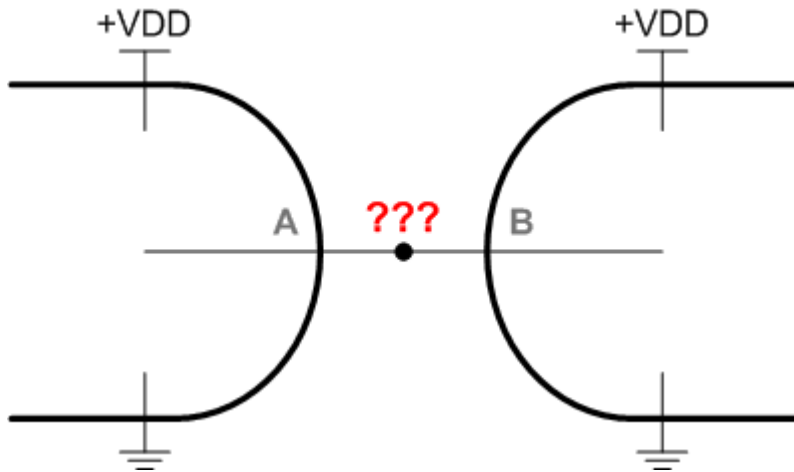


Gerado pelo simulador devido a conflitos entre *drivers* passivos
Não faz sentido ser testado (em condições) nem usado em atribuições a sinais,
portos ou variáveis

std logic 1164 – Outros Casos que não Resultam em Conflitos



std logic 1164 – Tabela de Resolução



Ver nota sobre sub-tipos
“resolved” no slide seguinte.

		B								
		U	X	0	1	Z	W	L	H	-
A	U	U	U	U	U	U	U	U	U	U
	X	U	X	X	X	X	X	X	X	X
	0	U	X	0	X	0	0	0	0	X
	1	U	X	X	1	1	1	1	1	X
	Z	U	X	0	1	Z	W	L	H	X
	W	U	X	0	1	W	W	W	W	X
	L	U	X	0	1	L	W	L	W	X
	H	U	X	0	1	H	W	W	H	X
	-	U	X	X	X	X	X	X	X	X



Definição std_logic(_vector) na *Package std_logic_1164*

```
type std_ulogic is ('U', -- Uninitialized
                   'X', -- Forcing Unknown
                   '0', -- Forcing 0
                   '1', -- Forcing 1
                   'Z', -- High Impedance
                   'W', -- Weak Unknown
                   'L', -- Weak 0
                   'H', -- Weak 1
                   '-' -- Don't care);
```

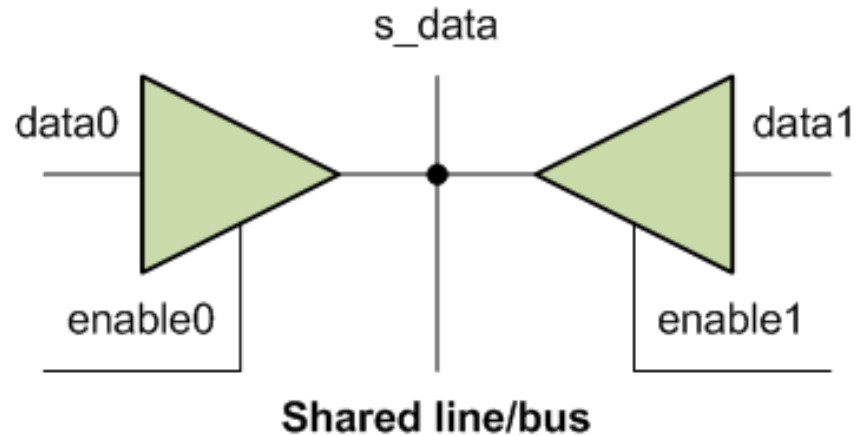
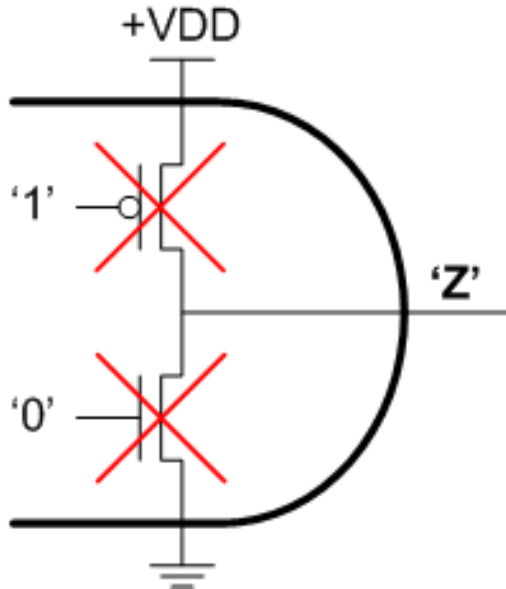
```
subtype std_logic is resolved std_ulogic;
type std_logic_vector is array(natural range <>) of std_logic;
```

Um sub-tipo “resolved” permite resolver conflitos de múltiplas instâncias do tipo hierarquicamente superior, através da evocação de uma função específica para o efeito. (e.g. num std_ulogic, qualquer atribuição simultânea, a um mesmo sinal, a partir de duas fontes distintas, resulta num erro de compilação).

– Utilização

- Abstração dos níveis lógicos possíveis em hardware e em simulação
- Operações sobre vetores de bits

std logic 1164 – Utilização de Tri-state



Modelação de linhas partilhadas (*shared*)

exemplo com atribuições condicionais

```
s_data <= data0 when (enable0 = '1') else  
    'Z' ;
```

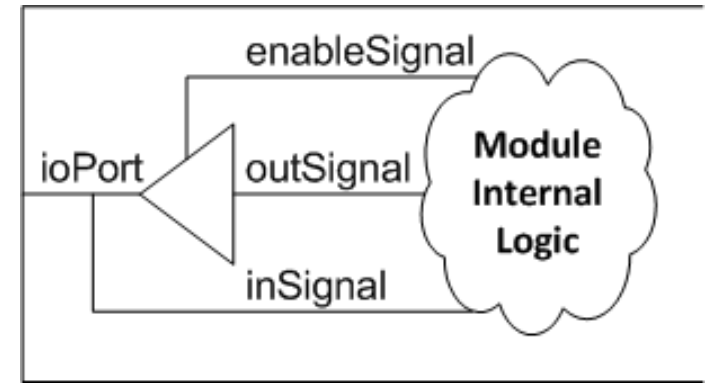
```
s_data <= data1 when (enable1 = '1') else  
    'Z' ;
```

Pode também ser usado com barramentos do tipo `std_logic_vector`

Portos Bidirecionais e Lógica Tri-state

```
entity EntityName
  port(...
    ioPort : inout std_logic;
    ...);
end EntityName;
```

```
architecture Behavioral of EntityName
  signal inSignal, outSignal, enableSignal : std_logic;
begin
  ...
  ioPort <= outSignal when (enableSignal = '1') else
    'Z';
  inSignal <= ioPort;
  ...
end Behavioral;
```



Tipicamente usados com pinos externos da FPGA (portos da entidade *top-level*).
Podem também ser usados com barramentos do tipo `std_logic_vector`



Resumo dos Tipos de Dados em VHDL (mais frequentes)

- integer

- Definição (na *package* STANDARD)

- ```
type integer is range -2147483647 to 2147483647;
```

- Utilização típica

- Indexação de arrays e como segundo operando de deslocamentos (*shifts*) e rotações (*rotates*) – número de posições a deslocar

- natural

- Definição (na *package* STANDARD)

- ```
subtype natural is integer range 0 to integer'high;
```

- Utilização típica

- Semelhante ao tipo integer, mas para valores naturais

- positive

- Definição (na *package* STANDARD)

- ```
subtype positive is integer range 1 to integer'high;
```

- Utilização típica

- Semelhante ao tipo integer, mas para valores positivos

# Resumo dos Tipos de Dados em VHDL (mais frequentes)

- **unsigned**

- Definição (na *package* NUMERIC\_STD)

- ```
type UNSIGNED is array (NATURAL range <>) of STD_LOGIC;
```

- Utilização típica

- Operações aritméticas e lógicas em quantidades inteiras **sem** sinal

- **signed**

- Definição (na *package* NUMERIC_STD)

- ```
type SIGNED is array (NATURAL range <>) of STD_LOGIC;
```

- Utilização típica

- Operações aritméticas e lógicas em quantidades inteiras **com** sinal

- **tipos enumerados**

- Definição

- Pelo utilizador num módulo ou package

- Utilização típica

- Definição dos estados simbólicos de uma FSM

# Outros Tipos de Dados

- **boolean**

- Definição (na *package* STANDARD)

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

- Utilização típica

- Resultado de condições e expressões booleanas

- **character / string**

- Definição (na *package* STANDARD)

- Utilização típica

- Manipulação de caracteres e arrays de caracteres

- **real**

- Definição (na *package* STANDARD)

- `type real is range -1.0E308 to 1.0E308;`

- Utilização típica

- Operações aritméticas em quantidades reais – **apenas em simulação ou na síntese, mas quando os valores são estáticos**

# Outros Tipos de Dados

- **time**

- Definição (na package STANDARD)

```
type time is range -2147483648 to 2147483647
 units
 fs;
 ps = 1000 fs;
 ns = 1000 ps;
 us = 1000 ns;
 ms = 1000 us;
 sec = 1000 ms;
 min = 60 sec;
 hr = 60 min;
 end units;
```

- Utilização típica

- Simulação e construção de testbenches

- **bit e bit\_vector**

- Definição (na package STANDARD)

```
type bit is ('0', '1');
```

```
Type bit_vector is array (natural range <>) of bit;
```

- Pouco usado devido à existência do tipo `std_logic(_vector)`

# Macros/Funções de Conversão entre Tipos

- Para simplificar a interface entre módulos deve-se utilizar sempre portos do tipo **std\_logic** ou **std\_logic\_vector**
  - Exceto se forem usados *records*
- Se necessário, as conversões são efetuadas dentro dos módulos para os tipos requeridos pelas operações a realizar
- Macros de conversão **(un)signed <-> std\_logic\_vector**
  - **unsigned**(parâmetro do tipo **std\_logic\_vector**)
    - macro/operador de conversão de **std\_logic\_vector** para **unsigned**
  - **signed**(parâmetro do tipo **std\_logic\_vector**)
    - macro/operador de conversão de **std\_logic\_vector** para **signed**
  - **std\_logic\_vector**(parâmetro do tipo **signed/unsigned**)
    - macro/operador de conversão de **signed** ou **unsigned** para **std\_logic\_vector**



# Macros/Funções de Conversão entre Tipos

- Funções de conversão `integer <-> (un)signed`
  - `to_integer(arg : unsigned) return natural;`
  - `to_integer(arg : signed) return integer;`
  - `to_unsigned(arg : natural, size: natural)`  
`return unsigned;`
  - `to_signed(arg : integer, size: natural)`  
`return signed;`
- Consoante a conversão pretendida, pode ser necessário uma ou duas conversões em cascata
  - e.g. `std_logic_vector->integer`
    - `to_integer(unsigned(Vetor_de_Bits_a_Conv))`

# Comentários Finais

- No final desta aula deverá ser capaz de:
  - Conhecer a finalidade das packages em VHDL
  - Usar os tipos de dados agregados em VHDL (*arrays* e *records*)
  - Usar adequadamente os principais tipos de dados suportados pelo VHDL (tanto para síntese como para simulação)
  - Compreender em profundidade o tipo `std_logic` e o significado dos seus valores predefinidos
  - Modelar sinais e portos tri-state
  - Utilizar corretamente as macros e funções de conversão entre tipos