UNIVERSIDADE FEDERAL DE SANTA CATARINA

Laboratório 2: Introdução à Linguagem VHDL

EEL5105 – Circuitos e Técnicas Digitais

Objetivos

- Primeiros passos em **VHDL**
- Estudar **exemplos básicos de descrição de hardware** em VHDL
- Implementar circuitos usando **VHDL** no **Emulador do kit DE2**

Introdução

- **VHDL - Visão Geral**
  - **VHDL** é uma linguagem para descrição de hardware.
  - **VHDL** = **V**HSIC **H**ardware **D**escription **L**anguage.
  - No final da década de 80, **VHDL** se tornou uma linguagem padrão para o **IEEE** (***Institute of Electrical and Electronic Engineers***).
  - Existem diversas ferramentas para simular e sintetizar (gerar hardware) circuitos descritos em **VHDL**.
  - Outras linguagens de descrição de hardware: Verilog, SystemC, AHDL, Handel-C, System Verilog, Abel, Ruby, ...

Introdução

- **VHDL - Visão Geral**
  - Permite descrever um circuito digital de diferentes formas (ex.: **estrutural**, **comportamental**, **fluxo de dados**).
  - Descrições em **VHDL** podem então ser utilizadas para gerar **hardware** (arquivo para configuração de um FPGA ou projeto de um circuito integrado, por exemplo).
  - Descrições em **VHDL** podem ser **simuladas**, permitindo eliminar problemas antes da **síntese do hardware**.
  - A geração de estímulos para simulação **VHDL** é comumente realizada por intermédio de **testbenches**, onde são definidos estímulos a serem aplicados ao circuito, dentre outras coisas.

Introdução

- **VHDL – Exemplo de código: *Majority Detector***
  - ***Majority Detector***: saída em nível lógico alto sempre que a maioria dos bits de entrada estiver em nível lógico alto

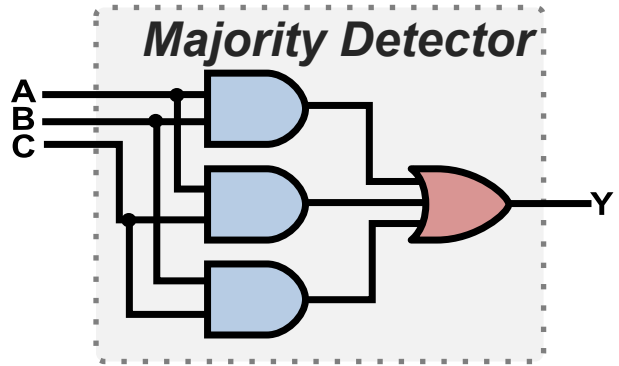


Tabela verdade

A B C	Y
0 0 0	0
0 0 1	0
0 1 0	0
0 1 1	1
1 0 0	0
1 0 1	1
1 1 0	1
1 1 1	1

Y = (A and B) or (A and C) or (B and C)

Y = (A . B) + (A . C) + (B . C)

Introdução

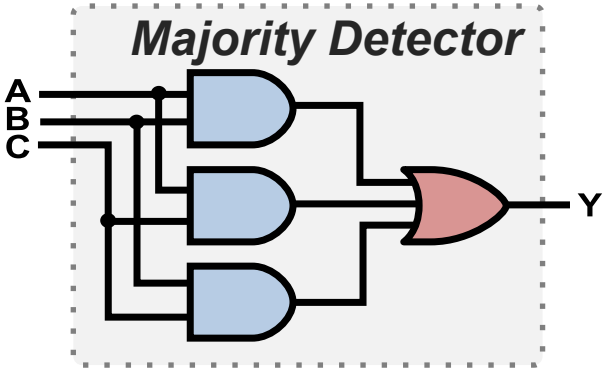
- VHDL – *Majority Detector*

LIBRARIES

```
library IEEE;
use IEEE.Std_Logic_1164.all;
```

ENTITY

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



ARCHITECTURE

```
architecture circuito_logico of majority is
  signal D,E,F: std_logic;
begin
  Y <= D or E or F;
  D <= A and B;
  E <= A and C;
  F <= B and C;
end circuito_logico;
```

Introdução

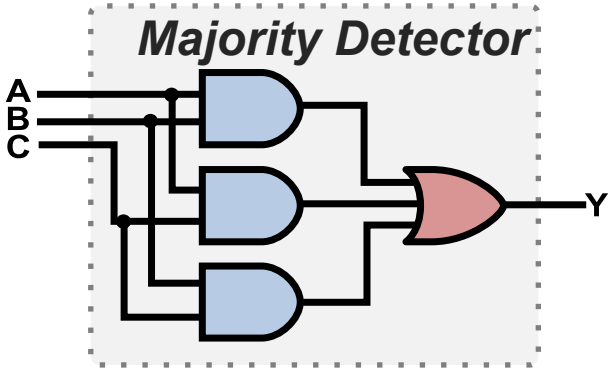
- **VHDL – *Majority Detector***
  - **LIBRARIES**: bibliotecas necessárias.

```
library IEEE;
use IEEE.Std_Logic_1164.all;
```

  - Basicamente, essa biblioteca define os tipos **std_logic** e **std_logic_vector**, os quais são versões aperfeiçoadas dos tipos nativos **bit** e **bit_vector** do VHDL.
  - **std_logic**: '0' ou '1' (com aspas simples).
  - **std_logic_vector**: "001010" ou "011" ou "01110" etc… (com aspas duplas).

Introdução

- VHDL – ***Majority Detector***
  - **ENTITY**: define as **portas** do circuito digital, ou seja, a **interface** entre a lógica implementada e o mundo externo.



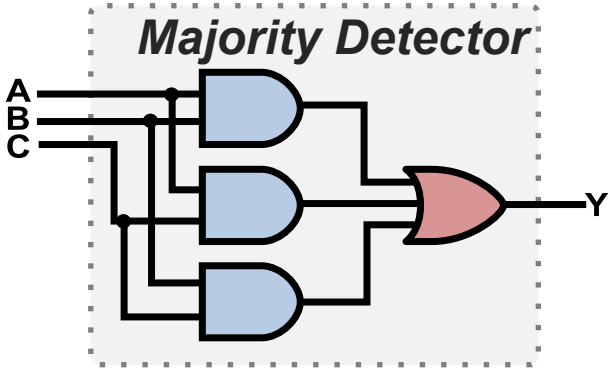

```
entity majority is
port (A: in std_logic;
      B: in std_logic;
      C: in std_logic;
      Y: out std_logic );
end majority;
```

Poderia ser também:

```
entity majority is
port (A, B, C: in std_logic;
      Y: out std_logic );
end majority;
```

Introdução

- **VHDL – *Majority Detector***
  - **ENTITY**: define as **portas** do circuito digital, ou seja, a **interface** entre a lógica implementada e o mundo externo.

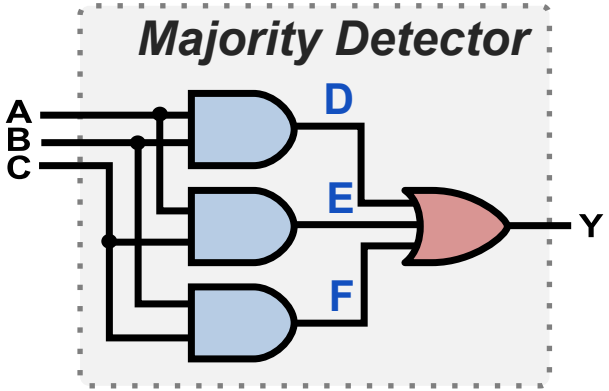



```
entity majority is
port (A: in  std_logic;
      B: in  std_logic;
      C: in  std_logic;
      Y: out std_logic );
end majority;
```

nome direção tipo

Introdução

- VHDL – *Majority Detector*
  - **ARCHITECTURE** : define a funcionalidade do circuito digital, utilizando as **portas** listadas na **ENTITY**, além de **signals** para fazer as conexões internas.




```
architecture circuito of majority is
  signal D,E,F: std_logic;
begin
  Y <= D or E or F;
  D <= A and B;
  E <= A and C;
  F <= B and C;
end circuito;
```

Introdução

- VHDL – *Majority Detector*
  - **ARCHITECTURE**

```
architecture circuito of majority is
  signal D,E,F: std_logic;
begin
  Y <= D or E or F;
  D <= A and B;
  E <= A and C;
  F <= B and C;
end circuito;
```

Declaração dos *signals*

Operador de atribuição

Operadores lógicos: not, and, nand, or, nor, xor e xnor.

Introdução

- **Exemplo com std_logic_vector:**

```
library IEEE;
use IEEE.Std_Logic_1164.all;
entity silly is
port (A: in std_logic_vector(7 downto 0);
      Y: out std_logic_vector(7 downto 0)
      );
end silly;
architecture myarch of silly is
  signal AUX: std_logic_vector(3 downto 0);
begin
  Y <= A(7 downto 4) & AUX;
  AUX <= not A(3 downto 0);
end myarch;
```

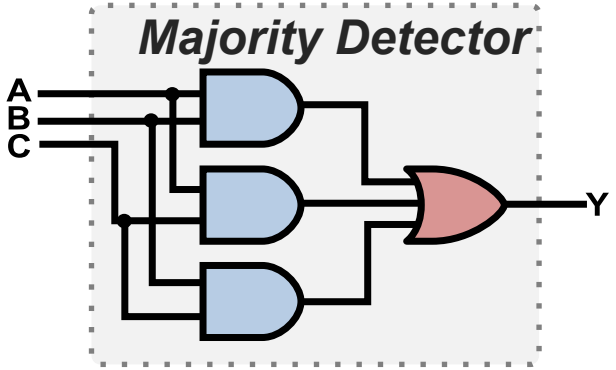
Operador de concatenação

Pergunta que você deve ser capaz de responder: o que faz esse circuito para A = 11110000?

Pouco importa a sequência das atribuições aqui, pois elas são concorrentes.

Tarefas

- **Tarefa 1**: Adaptar o código mostrado anteriormente visando implementar o **Majority Detector** no **Emulador do DE2**
  - **Atenção:** se o nome da **entity**/**arquivo** não for **usertop**/**usertop.vhd**, é preciso aplicar **Set Top Level** ao arquivo utilizado.
  - Além disso, as **ports** devem ser adaptadas de tal forma a serem mapeadas pelo emulador para **chaves** e **leds**. Utilize então as seguintes **ports** para sua entidade **usertop**:




```
SW: in std_logic_vector(17 downto 0);
LEDR: out std_logic_vector(17 downto 0)
```

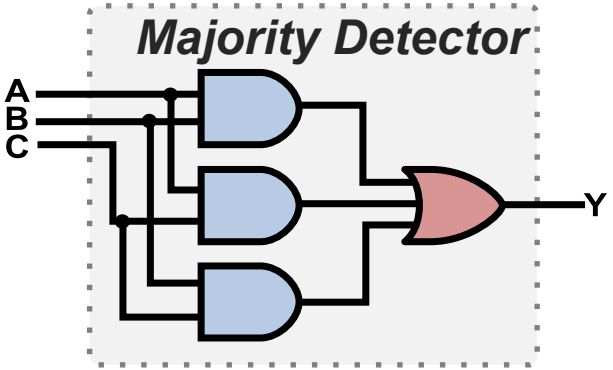
Você usará apenas 3 bits de SW e 1 bit de LEDR,
mas mesmo assim a porta toda precisa ser declarada no Emulador.

Tarefas

- **Tarefa 1**: Adaptar o código mostrado anteriormente visando implementar o **Majority Detector** no **Emulador do DE2**
  - **Dica:** Se quiser tornar o código mais legível, você pode declarar **A**, **B**, **C** e **Y** (entradas e saída do **majority detector**) como **signals**, e fazer as seguintes atribuições dentro da sua arquitetura:

```
A <= SW(0);
B <= SW(1);
C <= SW(2);
LEDR(0) <= Y;
```

Assim, o seu código do **majority detector** poderá ser escrito em função de **A**, **B**, **C** e **Y**.



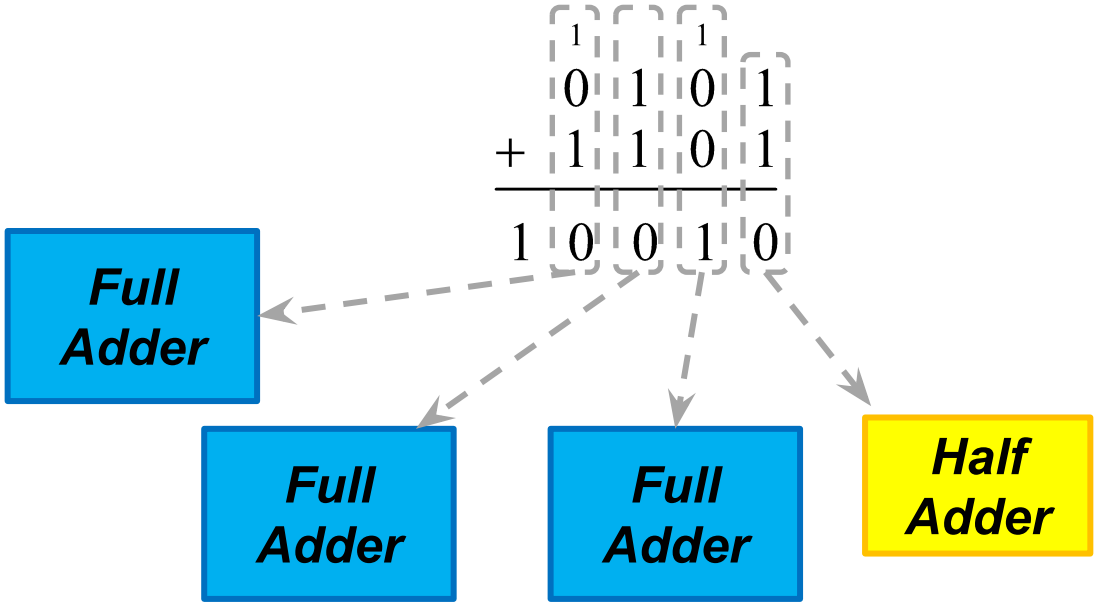
Tarefas

- **Tarefa 2**: Adaptar o código do exemplo "***silly***" para observar seu funcionamento no **Emulador**.
  - Novamente, se o nome da **entity/arquivo** não for **usertop**/**usertop.vhd**, é preciso aplicar **Set Top Level** ao arquivo utilizado. ,
  - **Ports** precisam ser adaptadas como na Tarefa 1.


```
SW: in std_logic_vector(17 downto 0);
LEDR: out std_logic_vector(17 downto 0)
```

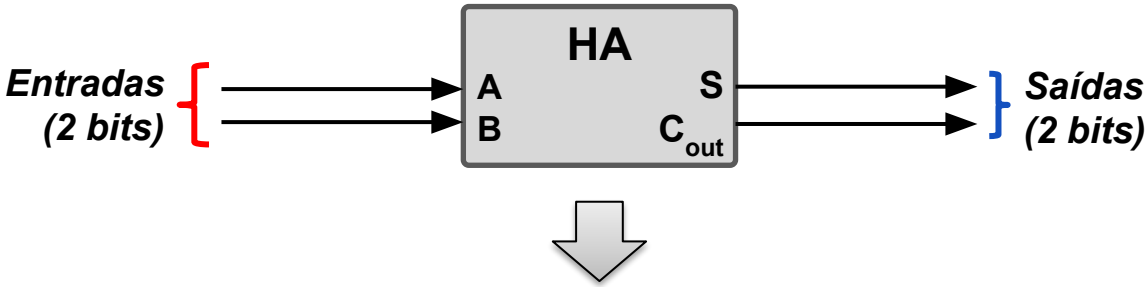
Tarefas

- **Tarefa 3**: Implementar um meio somador ou ***half adder*** (**HA**) e um somador completo ou ***full adder*** (**FA**).

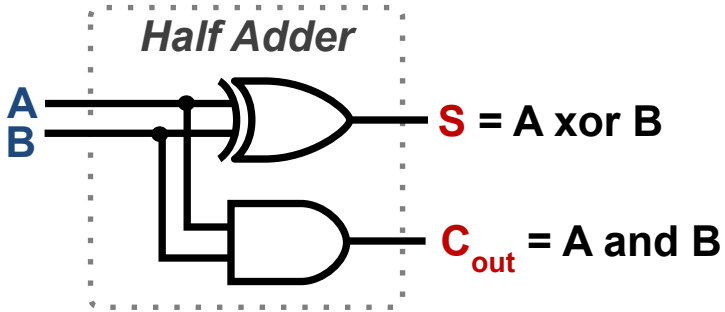


Tarefa

- ***Half Adder***

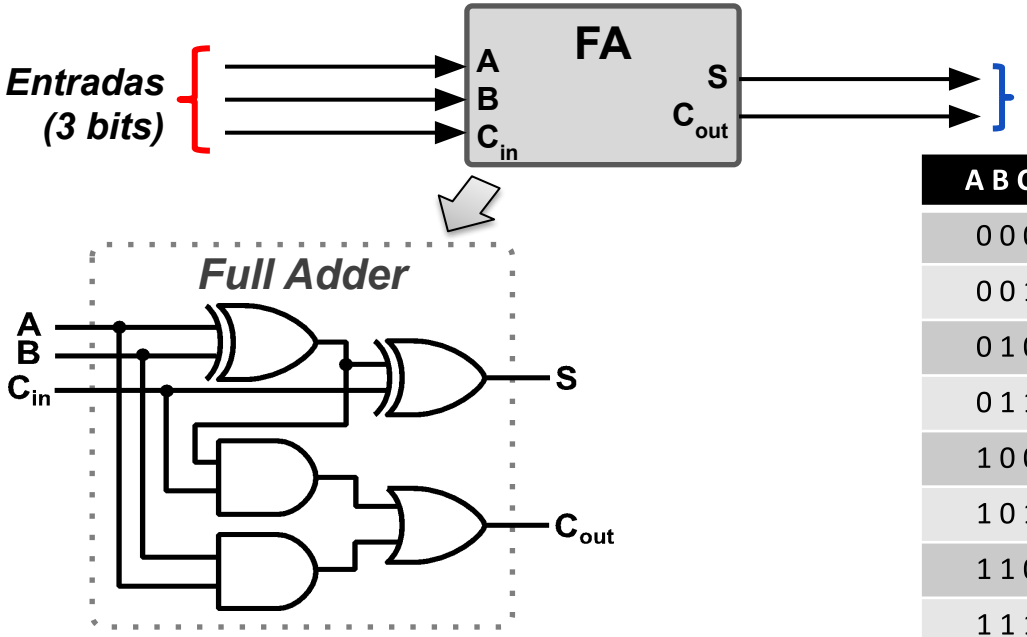


A	B	S	C_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Tarefa

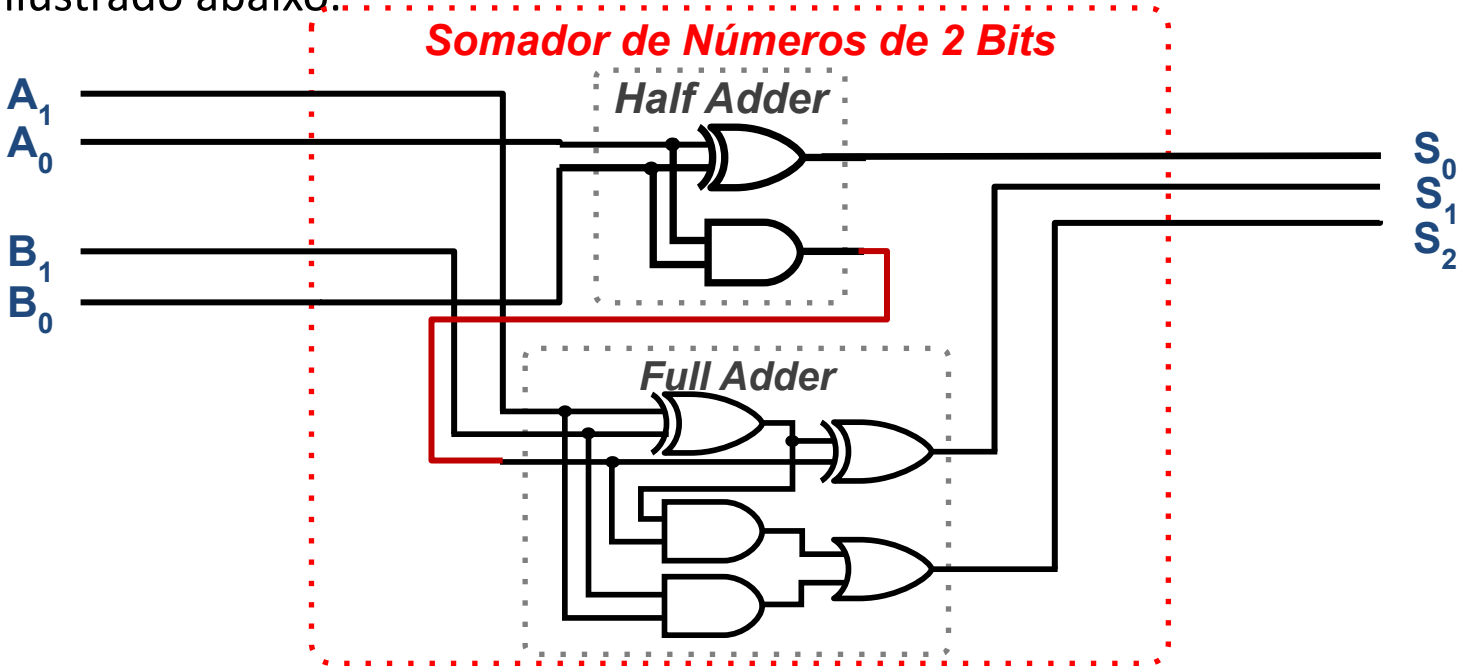
- ***Full Adder***
  - **Observação: signals** são necessários para conexões internas



A B C_{in}	S	C_{out}
0 0 0	0	0
0 0 1	1	0
0 1 0	1	0
0 1 1	0	1
1 0 0	1	0
1 0 1	0	1
1 1 0	0	1
1 1 1	1	1

Tarefa Avançada

- **Tarefa 4**: Faça a implementação um somador de números de 2 bits. Tal circuito é construído associando um ***half adder*** com um ***full adder*** conforme ilustrado abaixo.



Tarefa Avançada

- **Tarefa 5**: Universalidade das operações **NAND** e **NOR**.
  - Obtenha a implementação equivalente somente com portas **NAND** do **Majority Detector** e faça a implementação de tal circuito no **Emulador do DE2**.
  - Realize o mesmo procedimento, agora com a representação equivalente com portas **NOR**.