TÉCNICO LISBOA

Instituto Superior Técnico

Sistemas Digitais

LEEC $\|LEFT\|LEAer$

Workshop de Workflow em AMD Vivado

Autores:

Filipe Piçarra — filipepicarra@tecnico.ulisboa.pt
João Barreiros Coelho Rodrigues — joaobarreiroscoelhorodrigues@tecnico.ulisboa.pt

Com o apoio de:

Prof. António Grilo — antonio.grilo@inov.pt
Prof. Paulo Lopes — paulo.lopes@tecnico.ulisboa.pt

1º Semestre 2024/2025

Conteúdo

1 Introdução

Este *Workshop* tem como objetivo a aprendizagem do *workflow* em **AMD Vivado** através de um simples exemplo, para tal, é necessária uma breve introdução aos conceitos de lógica booleana.

1.1 O Problema

Suponha que se quer desenvolver um sistema para ativar a do seu jardim. Este sistema de rega dispara a uma hora exata do dia; de modo a poupar água, o sistema só dispara se o solo estiver seco ou não for chover nas próximas horas.

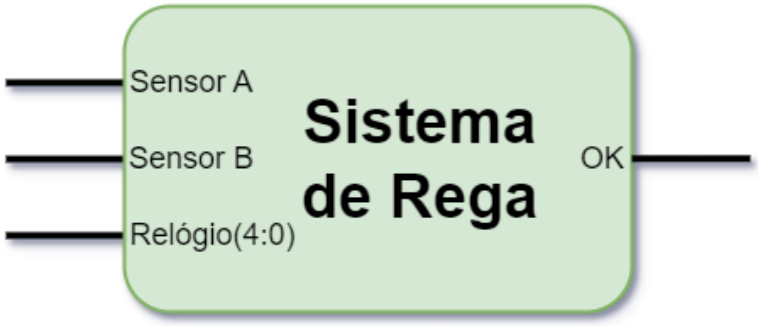


Figura 1: Esquema do Sistema de Rega.

As entradas do nosso sistema são as seguintes:

1. **Sensor A** - Sinal de **1 bit** que representa o Sensor de humidade no solo, que nos diz se o solo está seco.
2. **Sensor B** - Sinal de **1 bit** que representa o Pluviómetro, sensor que mede se irá chover nas próximas horas.
3. **Relógio** - Sinal de **5 bits** que nos indica as horas que já passaram no dia. No nosso caso, queremos que o sistema dispare às 19:00 (19 horas). Note-se que este relógio não faz uso da componente dos minutos.

Temos apenas uma única saída, um sinal **OK**, sinal de **1 bit** que ativa a rega do jardim.

2 Relembrar lógica Booleana

2.1 Representação Binária

Em sistemas digitais, trabalhamos com sinais binários, estes são dígitos que podem assumir o valor **0 (LOW)** ou **1 (HIGH)**. Esta notação facilita o entendimento do *output* dos sensores, se o **Sensor A** estiver a **1 (HIGH)**, então temos que está o solo está seco. De um ponto de vista eletrónico, os circuitos integrados trabalham com valores booleanos, por isso, operações com números (somas, multiplicações, etc.) têm de ser feitas em binário.

Relembremos a notação binária - para motivos de simplicidade vamos restringir-nos a trabalhar com os números naturais. Tal como na notação em base decimal onde cada digito vale uma potência de 10:

$$28473_{10} = 2 \times 10^4 + 8 \times 10^3 + 4 \times 10^2 + 7 \times 10^1 + 3 \times 10^0 \tag{1}$$

Em base dois (notação binária), cada digito vale uma potência de 2:

$$0110\ 1110_2 = 0 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 110_{10} \tag{2}$$

Posto isto, para representarmos o nosso relógio vamos precisar de um número de bits suficiente, cuja sua soma consiga representar todos as horas no dia - **24**. A maneiras mais prática de ver, é saber qual a potência de dois mais próxima de 24, para tal:

$$\log_2(24) = 4.58 \tag{3}$$

Logo precisamos de 4.58 bits, obviamente que isto não faz sentido, por isso dizemos que precisamos de *5 bits* para representar o nosso número.

Agora, para se obter o nosso número em binário que representa a hora do dia, em que o sistema ativa - 19 - fazemos sucessivas subtrações das maiores potências de base dois que "cabem"dentro do nosso número, observe:

$$\begin{aligned} &19 - 2^{\lfloor \log_2(19) \rfloor} = 3 && : \text{bit 4 ativo} \\ &3 - 2^{\lfloor \log_2(3) \rfloor} = 1 && : \text{bit 1 ativo} \\ &1 - 2^0 = 0 && (\text{fim}) : \text{bit 0 ativo} \end{aligned} \tag{4}$$

Ficamos então com o valor alvo do relógio, representado em 5 bits, dado por:

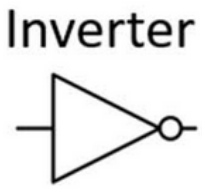
$$19_{10} = 1\ 0011_2 \tag{5}$$

2.2 Portas Lógicas Básicas

Aos blocos que implementam funções lógicas combinatórias chamamos *portas lógicas*. Estas são blocos que recebem N sinais de entrada e geram M sinais de saída, de acordo com as suas regras/tabelas de verdade. No workshop vamos abordar as seguintes portas lógicas:

2.2.1 NOT

Uma porta **NOT** é uma porta de negação, cuja função é inverter o sinal de entrada: se entra **1** sai **0** e se entra **0** sai **1**.

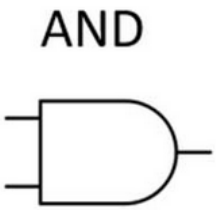


Input	Output
0	1
1	0

Figura 2: Diagrama e tabela de verdade da porta NOT.

2.2.2 AND

Uma porta **AND** é uma porta que faz o "e"lógico entre dois sinais, ambas as entradas têm de ser **1 (HIGH)** para a saída ser **1**, em qualquer outro caso a saída é **0**.

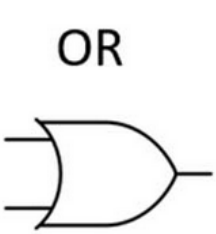


A	B	Output
0	0	0
1	0	0
0	1	0
1	1	1

Figura 3: Diagrama e tabela de verdade da porta AND.

2.2.3 OR

Uma porta **OR** realiza o "ou"lógico entre dois sinais, estando a saída ativa quando um deles está a **1**.



A	B	Output
0	0	0
1	0	1
0	1	1
1	1	1

Figura 4: Diagrama e tabela de verdade da porta OR.

3 Lógica do Circuito

No que toca ao processamento dos sinais, temos que analisar o sistema que nos foi pedido e as suas condições:

3.1 Sensores

"o sistema só dispara se o solo estiver seco **ou** se **não** for chover nas próximas horas"

Ou seja, os sensores permitem a rega se o **A** estiver **ativo ou B** estiver **desativado**. Sendo o diagrama:

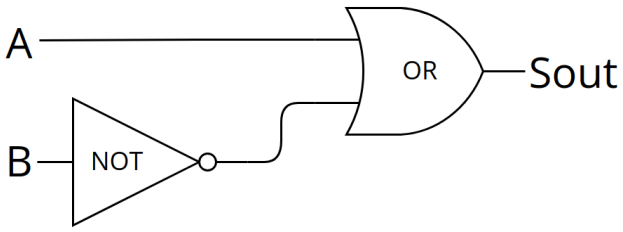


Figura 5: Diagrama dos Sensores

A	B	(A ∨ ¬B)
F	F	**T**
F	T	**F**
T	F	**T**
T	T	**T**

Figura 6: Tabela de Verdade dos Sensores

3.2 Relógio

Para o relógio, utilizamos um bloco de comparação, onde verificamos que a nossa entrada é igual a 10011_2 (19_{10}), para tal podemos utilizar portas **AND** que verifiquem os valores de cada bit e juntem:

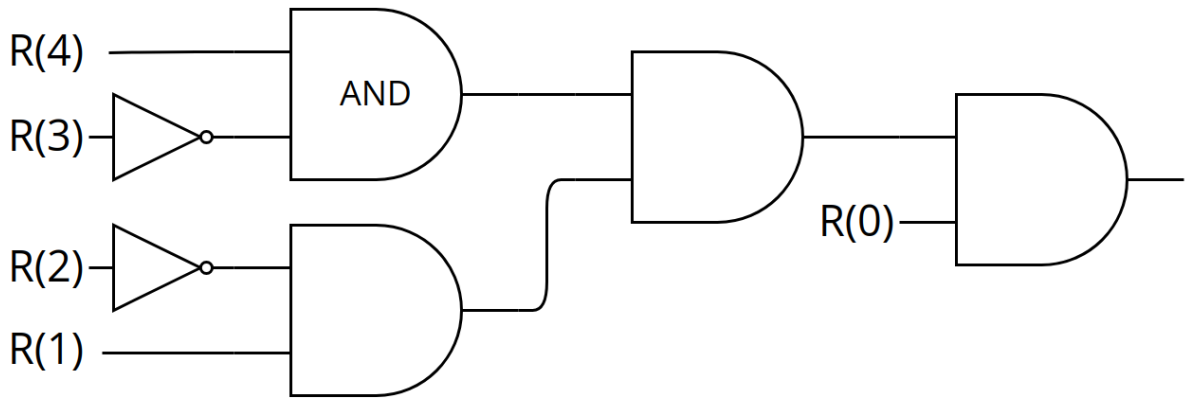


Figura 7: Diagrama do verificador do relógio.

3.3 Circuito Final

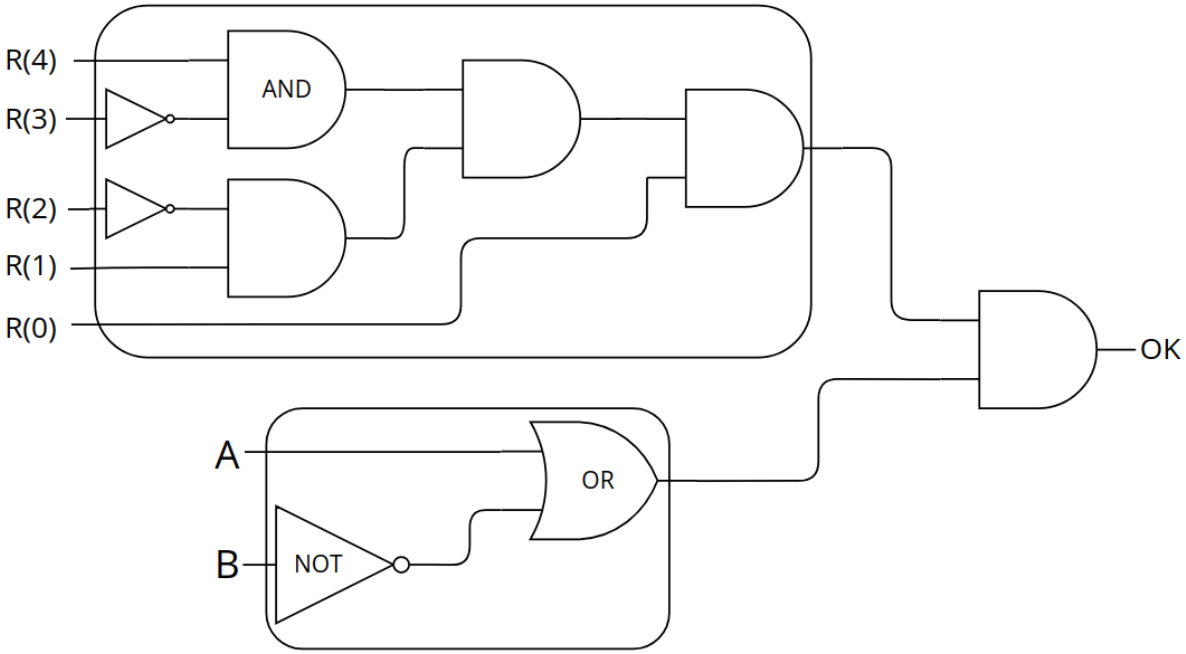
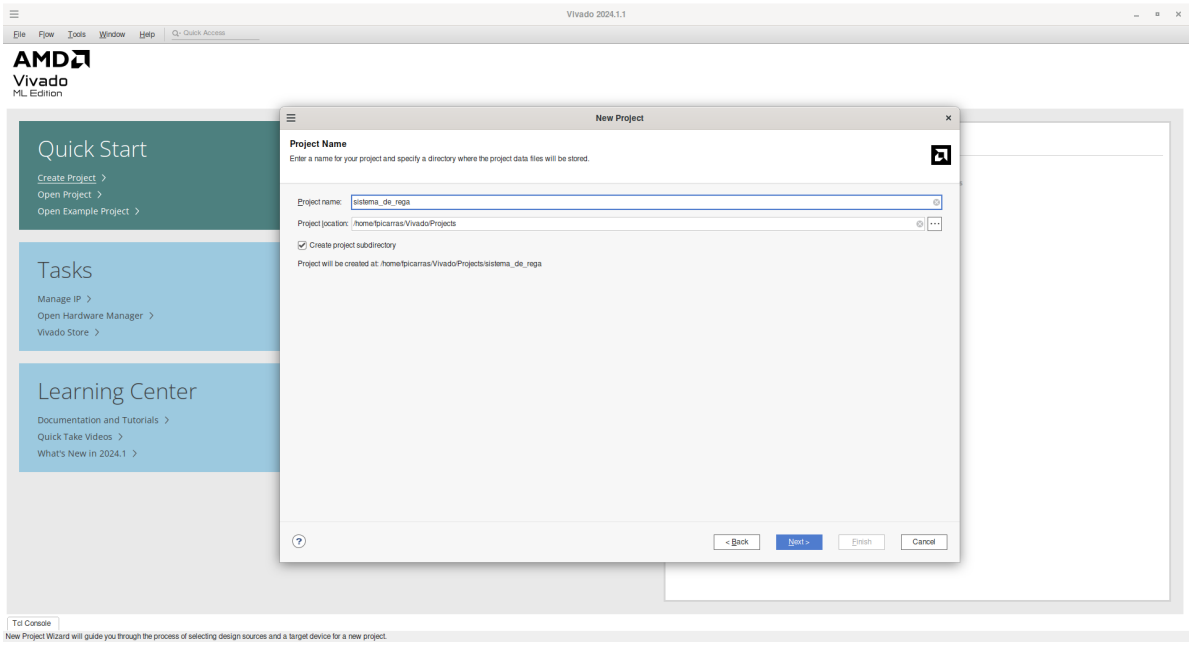


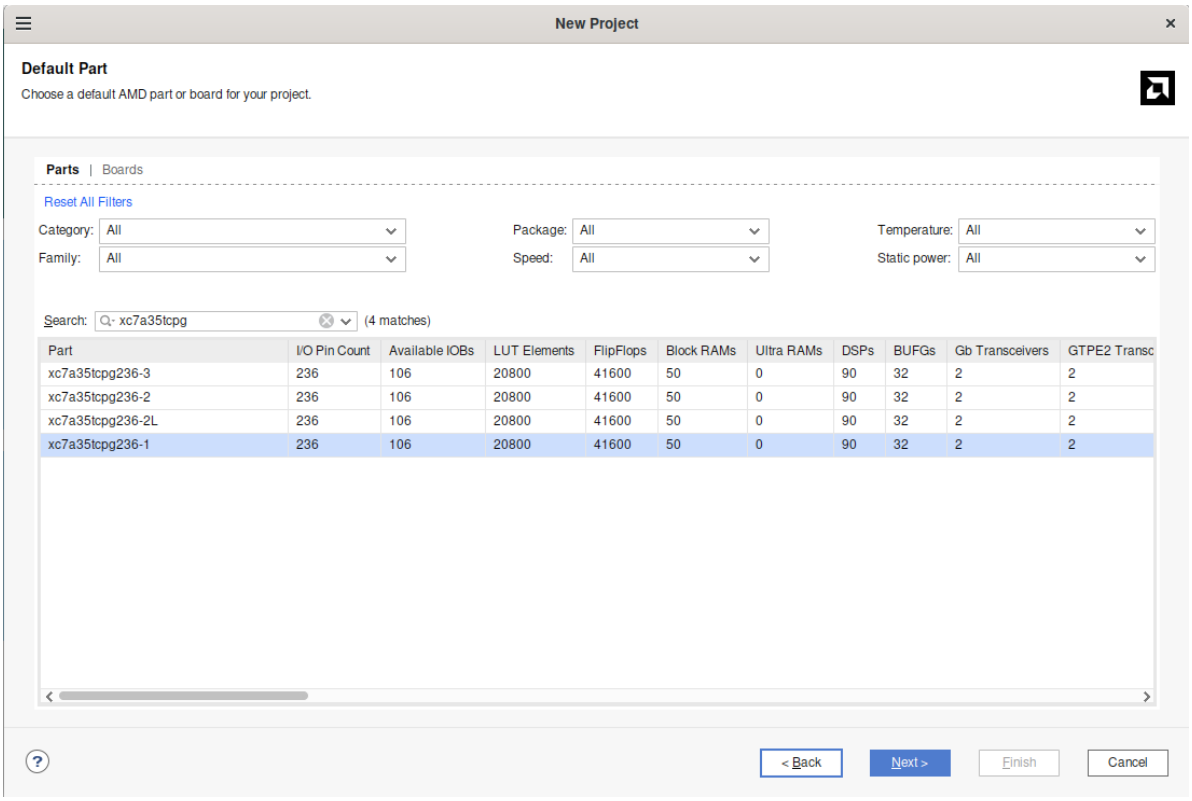
Figura 8: Diagrama do curcuito final.

4 Implementação

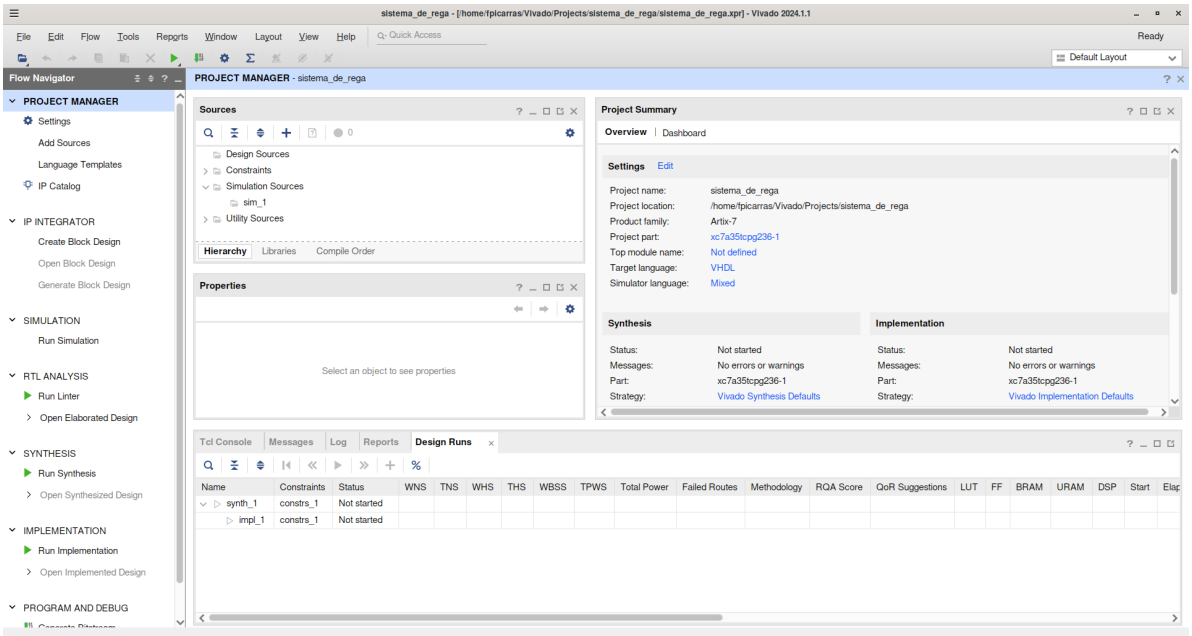
Para implementarmos o nosso sistema de rega, temos agora de abrir o *AMD Vivado*; clicar em **Create Project**; colocar o nome **sistema_de_rega** e escolher a directoria de trabalho, ou seja aonde iremos ter o design do sistema digital assim como os ficheiros resultantes da sintese, implementação e simulação.



Clicar em **next**; escolher **RTL Project** e clicar em **next** até aparecer a seleção **Default Part**, aqui é onde selecionamos a placa FPGA onde o circuito poderia ser implementado - matéria a ser abordada nos laboratórios de Sistemas Digitais. Por motivos de coerência selecionamos a mesma que consta nos guias de laboratório - **xc7a35tcpg236-1**:



Clicar em **next** e depois **finish**. Esta será a interface que vamos ter à frente:

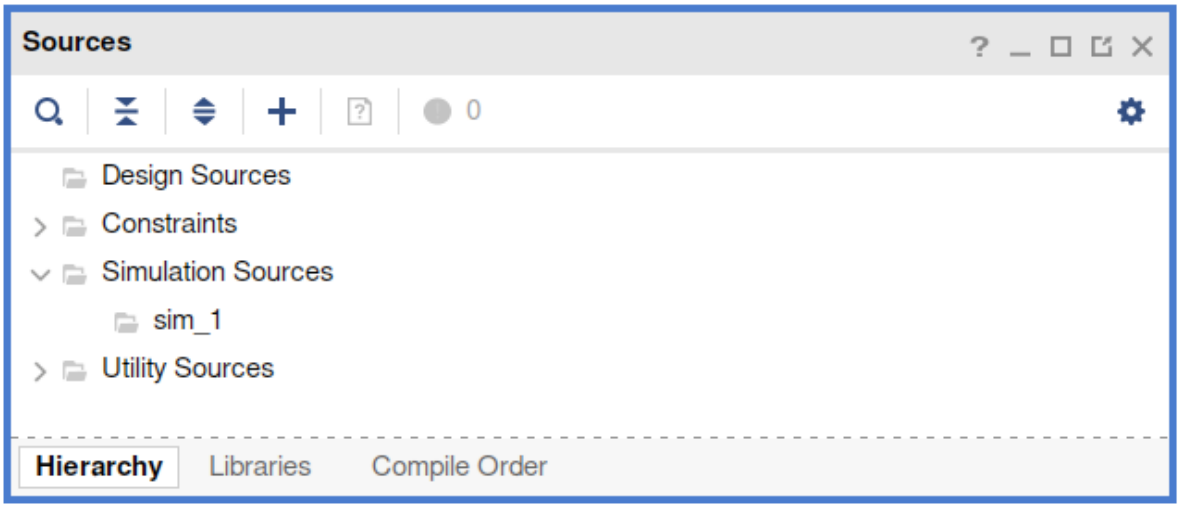


4.1 Design Sources

No Vivado, podemos criar várias descrições de hardware em arquivos (.vhd) separados, no nosso caso, iremos 3 *design sources*:

1. **sensors** - Circuito que recebe **A** e **B** e devolve um sinal que corresponde à lógica dos sensores;
2. **relogio** - Circuito que recebe o sinal de *5 bits* do relógio e devolve um sinal lógico **HIGH** quando atingimos o valor alvo;
3. **rega** - Circuito que combina os dois anteriores;

Comecemos pelo *sensors*. Podemos adicionar *design sources* clicando no "mais"no painel **Sources**:



Selecionamos **Add or create design sources**; **Create File** e aqui introduzimos o nome do nosso circuito, neste caso comecemos pelo *sensors* - certifique-se de quando inserir o nome, selecionar o *File type* **VHDL**!

O programa irá gerar um arquivo com a seguinte estrutura. Daqui conseguimos extrair as secções essenciais de um arquivo **VHDL**: declaração das bibliotecas (linhas 22 e 23), declaração da **entity** (linhas 27 a 29) e declaração da **architecture** (linhas 31 a 36). Na **entity** é onde definimos as nossas portas de entrada e saída e na **architecture** é onde definimos a lógica interna.

```
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24
25  -- Este é um exemplo de um comentário
26
27  entity sensors is
28
29  end sensors;
30
31  architecture Behavioral of sensors is
32
33  begin
34
35
36  end Behavioral;
```

Assim as portas de entra e saída de *sensors* são declaradas da seguinte forma:

```
27 entity sensors is
28     port(
29         A, B : in  std_logic;
30         O    : out std_logic
31         );
32 end sensors;
```

Para declararmos os sentidos das portas, quer entrada ou saída, utilizamos as declarações, respetivamente, **in** ou **out**. Como estas entradas/saídas são apenas de 1bit, dizemos que o seu tipo de sinal é **std_logic** - pode-se pensar como sendo apenas um fio a levar sinal.

Já a lógica de funcionamento interna é definida escrevendo-a diretamente como ela está no diagrama 6, ficando:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Este é um exemplo de um comentário

entity sensors is
    port(
        A, B : in  std_logic;
        O    : out std_logic
        );
end sensors;

architecture Behavioral of sensors is
begin
    O <= A or not B;
end Behavioral;
```

Repetindo os mesmos passos para *relogio* ficamos com um ficheiro *relogio.vhd* com o seguinte conteúdo:

```
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24
25 entity relogio is
26     port(
27         R : in  std_logic_vector(4 downto 0);
28         O : out std_logic
29     );
30 end relogio;
31
32 architecture Behavioral of relogio is
33 begin
34     O <= R(4) and (not R(3)) and (not R(2)) and R(1) and R(0);
35
36 end Behavioral;
```

Repara como a porta de entrada **R** tem uma declaração diferente, isso é porque ela está representada como um vetor de 5 bits, ou seja, um conjuntos de vários sinais **std_logic**. Nós designers especificamos o tamanho deste vetor/*bus* através da declaração **std_logic_vector(N-1 downto 0)** sendo N o tamanho do vetor.

Temos tudo para montarmos o circuito final, para isso criemos o ficheiro *rega.vhd*, que recebe nas suas entradas o vetor **R** com 5 bits e os sinais dos sensores **A** e **B**, com a saída **OK**:

```
25 entity rega is
26     port(
27         A, B : in  std_logic;
28         R    : in  std_logic_vector(4 downto 0);
29         O    : out std_logic
30     );
31 end rega;
```

Para ligarmos os outros componentes entram aqui novos conceitos:

1. Temos que instanciar os dois componentes (*sensors* e *relogio*);
2. Temos ainda que criar sinais que se liguem a estes dois componentes.

O que temos à nossa direita consiste na declaração dos componentes *sensors* e *relogio*, onde dizemos ao programa quais são as portas destes componentes. Abaixo temos a criação do sinais que se ligarão à saída destes componentes - pensemos neles como fios auxiliares que podem ser colocados a ligar um ponto X a um ponto Y.
Resta nos agora a declaração dos componentes e junção de sinais

```
33 architecture Behavioral of rega is
34     -- Declaração do componente sensors;
35     component sensors
36     port(
37         A, B : in  std_logic;
38         O    : out std_logic
39         );
40     end component;
41     -- Declaração do componente relogio;
42     component relogio
43     port(
44         R : in  std_logic_vector(4 downto 0);
45         O : out std_logic
46     );
47     end component;
48     -- Declaração dos sinais que unem as saídas
49     --dos componentes ao todo.
50     signal out_sensors, out_relogio : std_logic;
51
52 begin
```

Os aspeto final da arquitetura será:

```
33  architecture Behavioral of rega is
34      -- Declaração do componente sensors;
35      component sensors
36      port(
37          A, B : in  std_logic;
38          O    : out std_logic
39          );
40      end component;
41      -- Declaração do componente relogio;
42      component relogio
43      port(
44          R : in  std_logic_vector(4 downto 0);
45          O : out std_logic
46      );
47      end component;
48      -- Declaração dos sinais que unem as saídas
49      --dos componentes ao todo.
50      signal out_sensors, out_relogio : std_logic;
51  begin
52      -- Instanciação de sensors
53      sensors_i : sensors
54      port map(
55          A => A,
56          B => B,
57          O => out_sensors
58      );
59      -- Instaciação de relogio
60      relogio_i : relogio
61      port map(
62          R => R,
63          O => out_relogio
64      );
65      -- AND entre os dois outputs
66      OK <= out_sensors and out_relogio;
67  end Behavioral;
```

Repare que existe uma **diferença** entre a declaração e instanciação dos componentes.

A declaração vem da *keyword* **component**, onde se indica que este existe, por exemplo, uma **entity** *sensors* na **entity** a projectar *rega* e qual o I/O da **entity** *sensors*. É importante recordar que a descrição do arquitectura comportamental da **entity** *sensors* foi feita no sensors.vhd.

A instanciação é feita da descrição do comportamento d (*keyword* **begin**) e é onde indicamos que existe um componente do tipo *sensors* de nome *sensors_i*, adicionalmente neste bloco de instanciação é utilizada a *keyword* **portmap();** para indicar quais as ligações feitas no I/O desta instaciação.

A instaciação da mesma declaração pode ser múltiplas vezes para, por exemplo, criar um sistema em que há múltiplos componentes independentes mas de arquitectura igual.