

Laboratório de Sistemas Digitais

Trabalho Prático nº 1

Introdução às FPGAs, VHDL, ferramentas de projeto e kit de desenvolvimento

Objetivos

- Familiarização com as ferramentas de projeto e com o *kit* de desenvolvimento com FPGA que vai ser usado nas aulas práticas.
- Captura de diagramas esquemáticos (esquemas lógicos), modelação em VHDL, simulação, implementação em FPGA e teste de componentes elementares.

Sumário

Este trabalho prático está dividido em 4 partes. Na parte I pretende-se introduzir os aspectos básicos das ferramentas e do *kit* de desenvolvimento que vai ser usado nas aulas práticas de Laboratório de Sistemas Digitais (LSDig), com base num projeto simples baseado em captura de diagrama esquemático. São abordadas as diversas fases de projeto, desde a descrição até ao teste, passando pela simulação e implementação. Na parte II é ilustrado o projeto hierárquico baseado inteiramente na linguagem de descrição de hardware VHDL, através da construção gradual de uma porta lógica NAND a partir de uma AND e de um inverter. A parte III é dedicada ao projeto híbrido baseado em diagramas esquemáticos e descrições VHDL para diferentes partes do sistema e introduz-se os operadores lógicos em VHDL. Na parte IV é abordado um exemplo, baseado num comparador de igualdade, que permite ilustrar algumas das vantagens da linguagem VHDL, ao nível da abstração e produtividade.

Advertências muito importantes:

- A placa de desenvolvimento usada nas aulas práticas de Laboratório de Sistemas Digitais possui uma FPGA e diversos componentes que se podem danificar devido a descargas de eletricidade estática, pelo que deve ser manuseada com cuidado. Em particular, não deve tocar com qualquer parte do corpo ou objetos (incluindo vestuário) nos seus contactos elétricos e conetores.
- No final da aula, desligue o *kit* e arrume-o adequadamente juntamente com os cabos e alimentador na respetiva caixa.

Resumo do Fluxo de Projeto para Sistemas Baseados em FPGA

A Figura 1 ilustra o fluxo de projeto para sistemas baseados em FPGA. Os diversos passos são resumidos de seguida.

A etapa de *design entry* consiste na modelação, codificação ou introdução da funcionalidade pretendida, podendo para tal o projetista usar linguagens de descrição de hardware, diagramas esquemáticos, diagramas de transição de estado ou outros métodos. No caso de LSDig vai ser utilizada captura de diagramas esquemáticos e/ou a linguagem de descrição de hardware VHDL consoante o que for mais adequado para cada sistema. No caso das descrições em VHDL pode ser utilizado qualquer editor de texto para este efeito, embora por conveniência seja usado o editor integrado no ambiente de desenvolvimento (*Integrated Development Environment – IDE*) que vamos adotar (“Altera Quartus Prime”), o qual possui também editores de diagramas esquemáticos e de diagramas de estados, além de outras ferramentas necessárias para realizar todos os passos do fluxo de projeto.

Depois de modelado o sistema, o passo seguinte é a sua síntese (lógica), isto é, a compilação do modelo de forma a criar uma *netlist* (i.e. um conjunto de portas lógicas, flip-flops, multiplexadores, outros componentes e respetivas interligações) que implementa a funcionalidade pretendida. Este passo é realizado por ferramentas de software normalmente desenvolvidas pelo fabricante da FPGA usada.

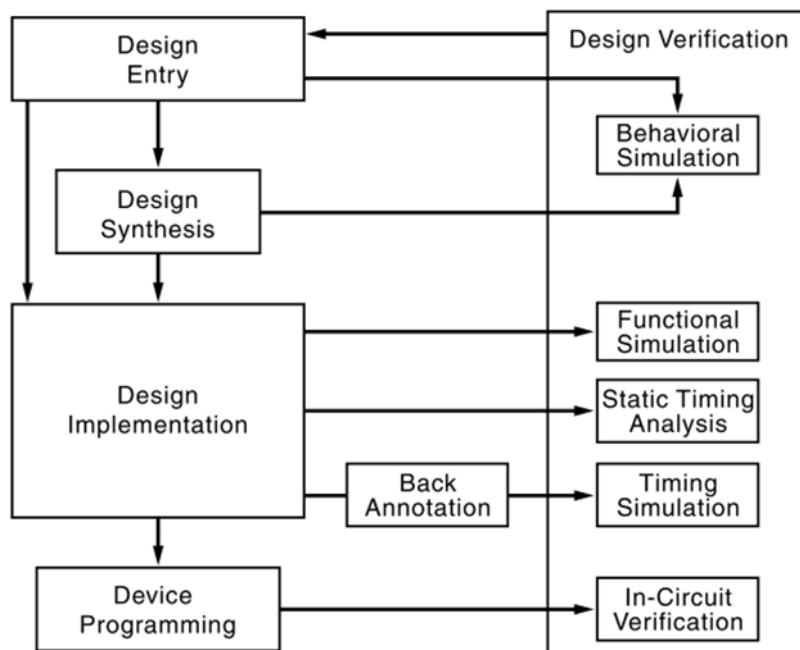


Figura 1 – Fluxo de projeto para sistemas digitais baseados em FPGA (fonte: www.xilinx.com).

Seguidamente, a *netlist* vai ser implementada (compilada) para uma determinada família de FPGAs por uma ferramenta de software (desenvolvida pelo fabricante da FPGA) que realiza os seguintes passos:

- Mapeamento da *netlist* em primitivas da FPGA (tabelas de verdade, portas lógicas, multiplexadores, registos, etc.);
- Posicionamento das primitivas em localizações específicas da FPGA;
- Encaminhamento das interconexões (estabelecimento das ligações) entre as primitivas da FPGA.

O resultado da implementação é um ficheiro de configuração da FPGA que deve ser usado para a sua programação através de software e um cabo de programação adequado.

A validação do sistema pode ser realizada em diversas etapas do fluxo de projeto, por simulação ou verificação em hardware real. São também normalmente disponibilizadas pelo fabricante diversas ferramentas de análise temporal, energética, recursos lógicos utilizados, etc. A maior parte destes passos vai ser ilustrada ao longo deste trabalho prático.

Nota importante: devido a limitações de velocidade da rede no acesso ao diretório pessoal em arca.ua.pt, nos PCs das salas de aula recomenda-se a utilização de uma pendrive para guardar os seus projetos e ficheiros. Caso isso não seja possível, guarde e aceda ao seu trabalho a partir de um diretório da drive Z: Em qualquer dos casos não utilize espaços nem

caracteres especiais (e.g. acentos) nas *paths* dos projetos e ficheiros, uma vez que isso causará problemas na utilização das ferramentas (isto significa que não deve gravar os seus projetos em sub-diretórios do “Ambiente de Trabalho” ou dos “Meus Documentos”). **Sugestão:** Em Windows crie, por exemplo em C:\Users\<Utilizador>, uma estrutura de diretórios para gravar os seus projetos de LSDig (e.g. C:\Users\<Utilizador>\LSDig\Aula1\Parte1\Projeto); em Linux poderá fazê-lo em /home/<utilizador>/LSDig/Aula1/Parte1/Projeto).

Parte I

Demonstração das etapas fundamentais do fluxo de projeto com base em captura de diagrama esquemático

1. Abra a aplicação “Altera Quartus Prime” e crie um novo projeto (menu “File→New Project Wizard”) de acordo com os passos seguintes (Figuras 2 a 9).

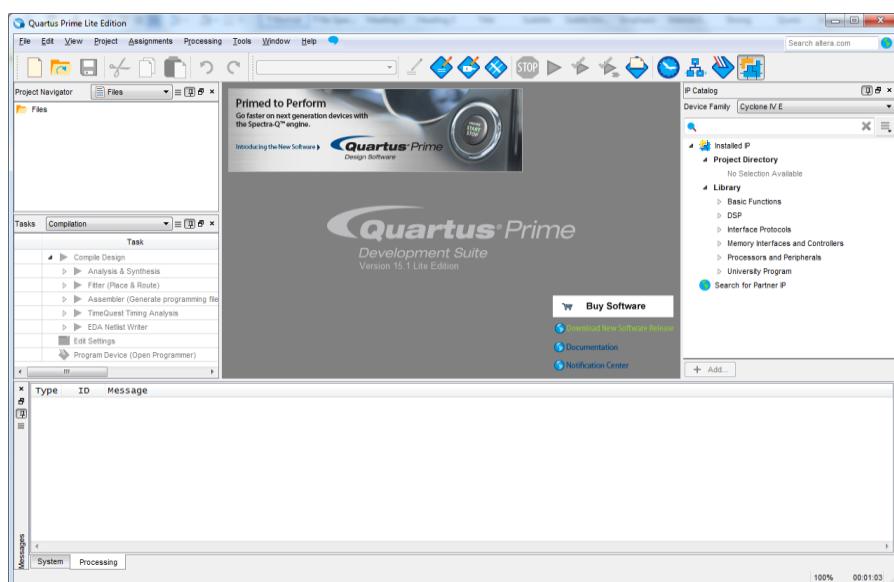


Figura 2 – Aspetto inicial da aplicação “Altera Quartus Prime” (sem qualquer projeto aberto).

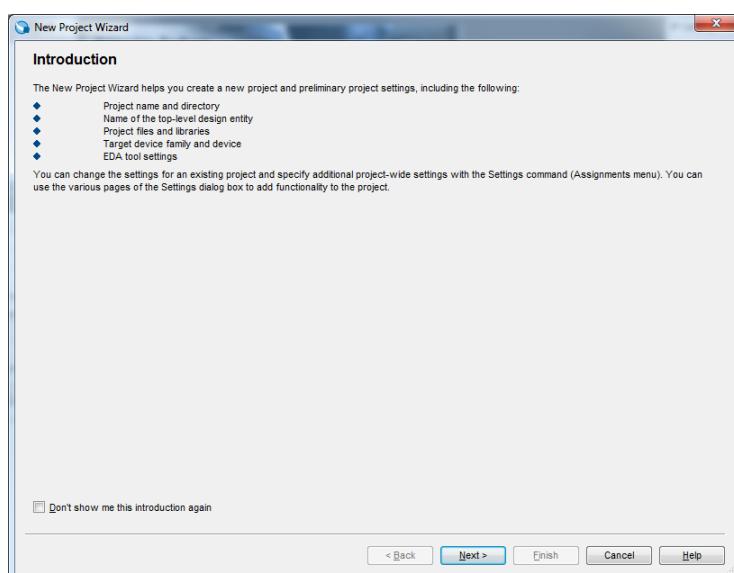


Figura 3 – Passo inicial introduutorio (pode ser desativado).

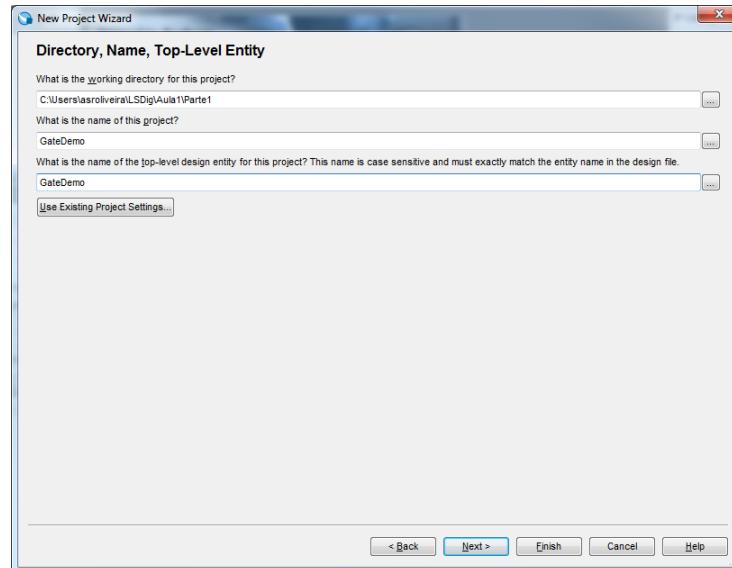


Figura 4 – Passo 1 – identificação e localização do projeto no sistema de ficheiros – adaptar de acordo com o diretório usado, o qual não pode conter no caminho (*path*) espaços nem caracteres especiais, e.g. acentos.

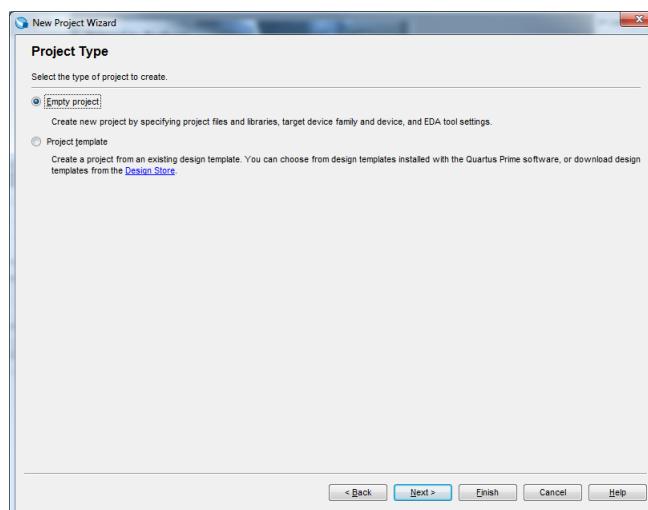


Figura 5 – Passo 2 – seleção do tipo de projeto a criar (projeto vazio neste caso).

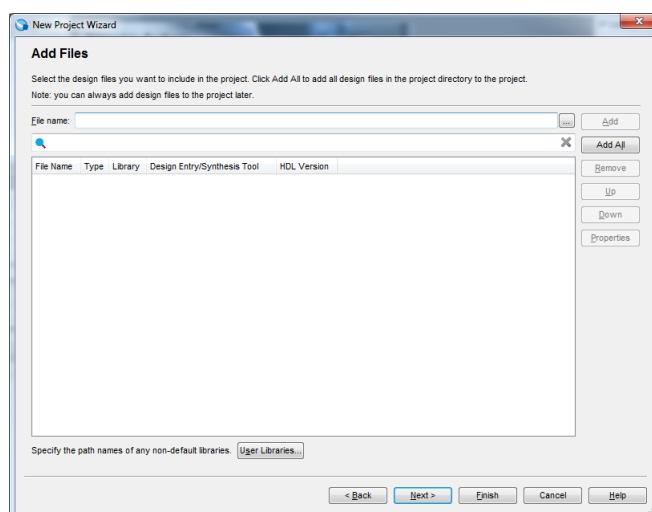


Figura 6 – Passo 2 – adição de ficheiros pré-existentes (não usado neste projeto).

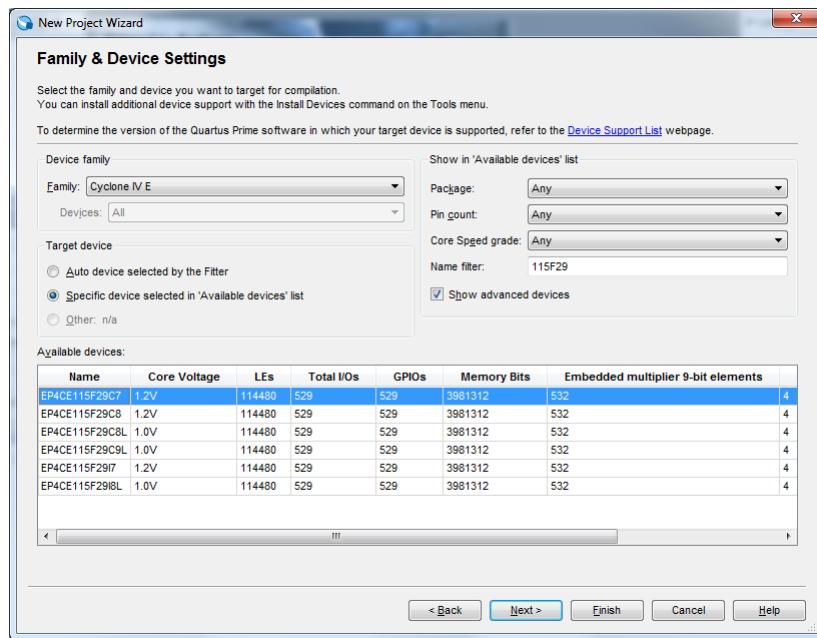


Figura 7 – Passo 3 – seleção da FPGA usada (**Altera Cyclone IV EP4CE115F29C7**) – a especificação do filtro “115F29” (campo “Name filter”) facilita a seleção da FPGA correta (primeira da lista).

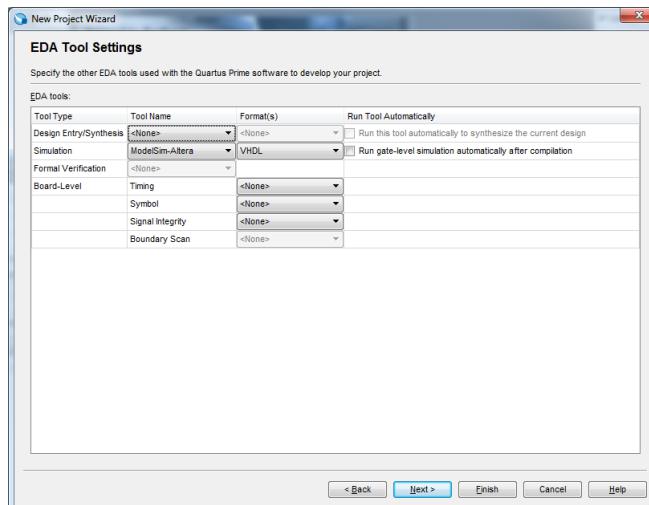


Figura 8 – Passo 4 – seleção das ferramentas e linguagens usadas no fluxo de projeto (valores por omissão).

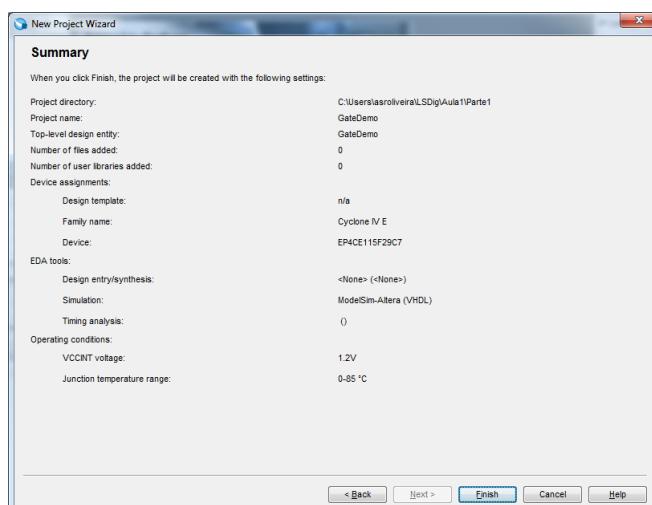


Figura 9 – Passo 5 – sumário final da criação do projeto.

2. Após premir “Finish” o IDE “Altera Quartus Prime” deve apresentar o aspeto da Figura 10.

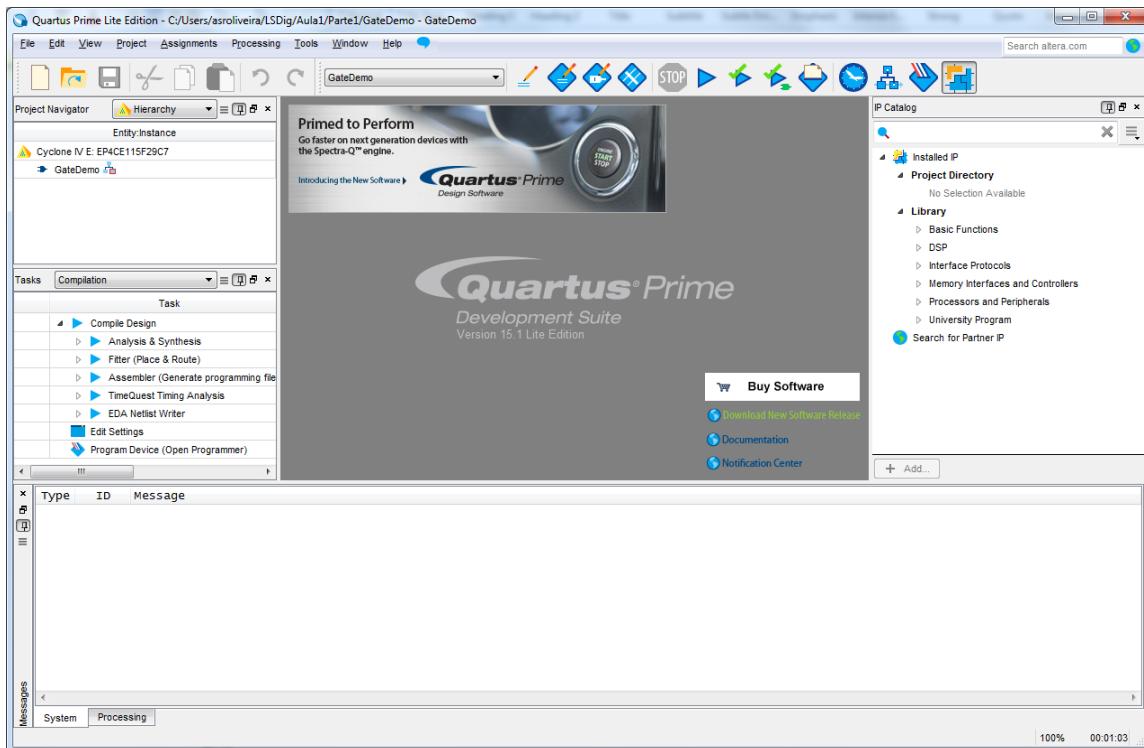


Figura 10 – Aspetto do “Altera Quartus Prime” IDE após a criação do projeto.

3. Crie um novo ficheiro para um diagrama esquemático (menu “File→New”), de acordo com a Figura 11.

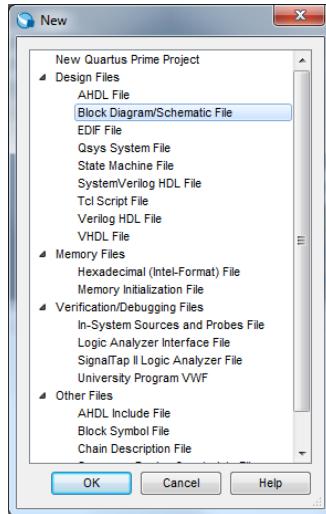


Figura 11 – Seleção do tipo de ficheiro a criar (Block Diagram / Schematic File).

4. Adicione uma porta lógica AND de 2 entradas usando o botão “Symbol Tool” da barra de ferramentas (Figura 12 a)) e escolhendo o componente de acordo com a Figura 12 c).

5. Adicione dois portos de entrada e um de saída usando o botão “Pin Tool” da barra de ferramentas (Figura 12 b)).

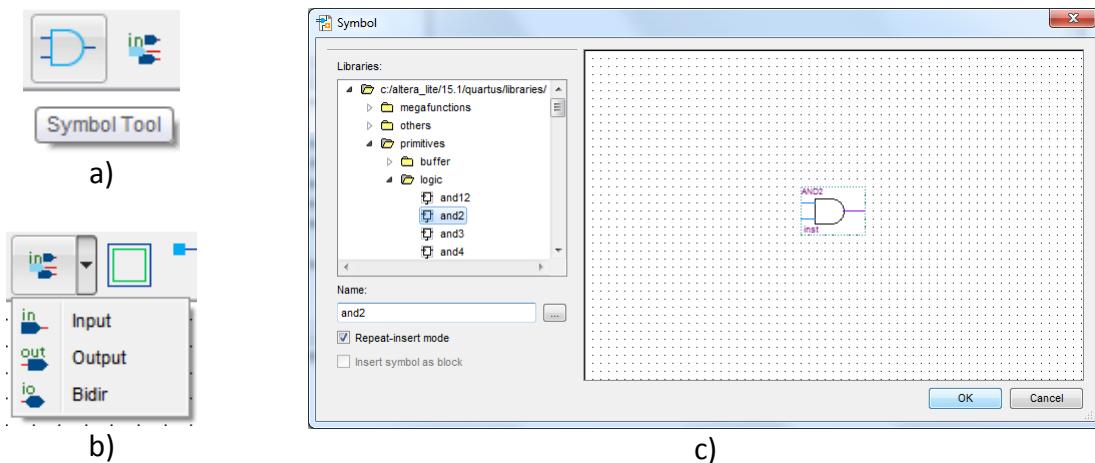


Figura 12 – Botões “Symbol Tool” e “Pin Tool” da barra de ferramentas e seleção da porta lógica AND na biblioteca de componentes do “Altera Quartus Prime”.

6. Interligue os portos de entrada, de saída e a porta lógica de acordo com a Figura 13. Identifique cada um dos elementos de acordo com os nomes apresentados na Figura 13.

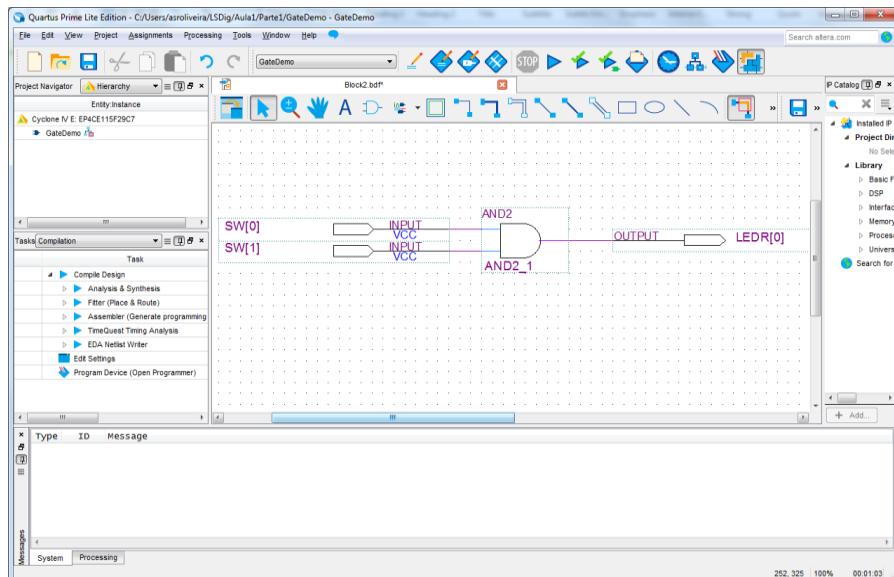


Figura 13 – Interligação da porta lógica AND e dos portos de entrada e de saída e identificação dos diversos elementos do circuito (para mudar o nome de um componente ou porto faça duplo clique sobre o nome atual).

7. Grave o ficheiro, cujo nome deverá ser “GateDemo.bdf” (Figura 14).

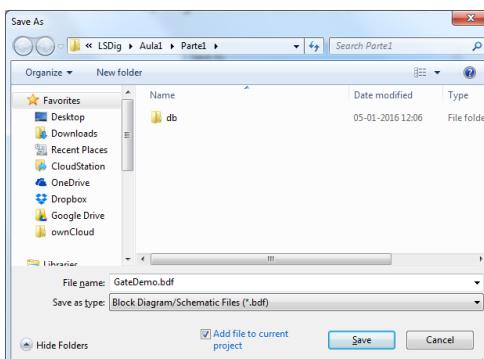


Figura 14 – Caixa de diálogo para gravação do ficheiro “GateDemo.bdf”.

8. Seguidamente vai ser validado por simulação o comportamento da porta lógica utilizada. No entanto, antes de efetuar a simulação, execute a opção “Analysis & Synthesis” para que seja analisada a correção estrutural do projeto. Após a execução da “Analysis & Synthesis” o IDE deve apresentar o aspeto da Figura 15.

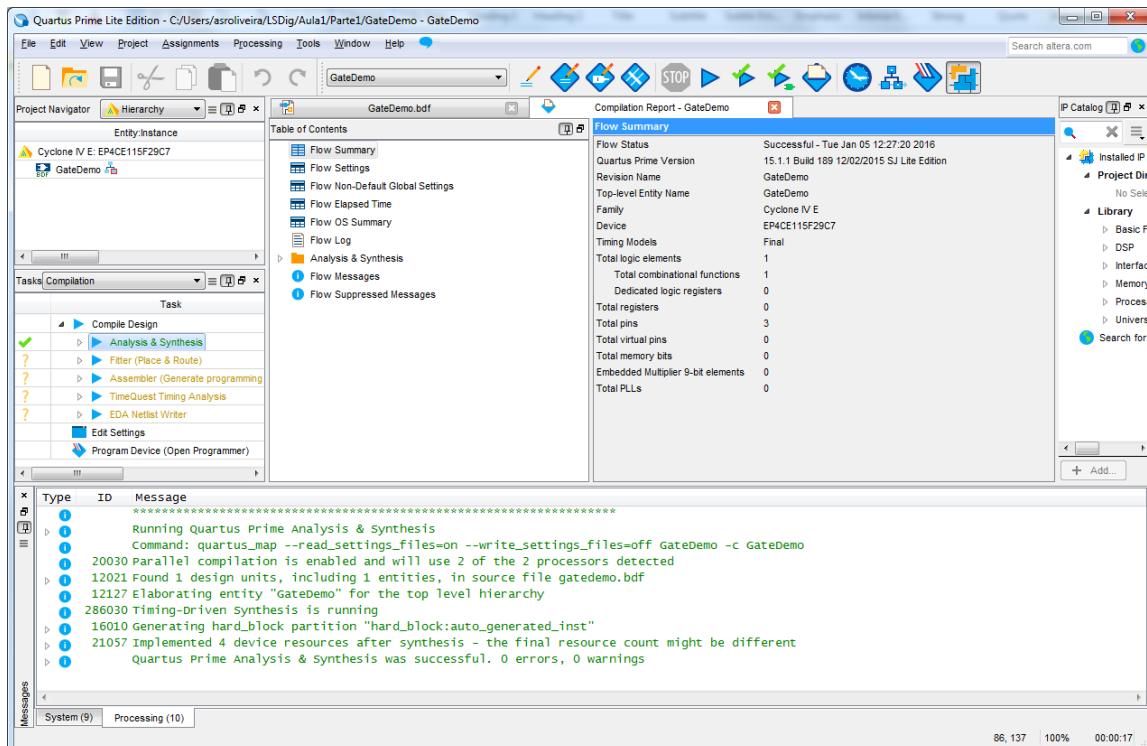


Figura 15 – “Altera Quartus Prime” IDE após a execução da “Analysis & Synthesis”.

9. Simule o comportamento da porta lógica, criando para tal um ficheiro VWF (menu “File→New”) de acordo com os passos descritos nos pontos seguintes (descritos nas Figuras 16 a 18).

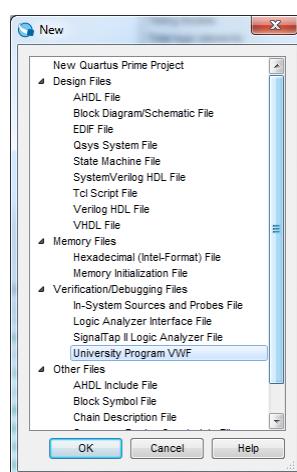


Figura 16 – Seleção do tipo de ficheiro a criar (“University Program VWF”).

10. Após premer “OK” a janela seguinte deverá abrir, onde deverão ser indicados os sinais a usar na simulação (Figura 17).

11. Através do menu “Edit→Insert→Insert Node or Bus”, premindo de seguida os botões “Node Finder” e “List”, selecione todos os portos de entrada e saída do circuito (Figuras 17 e 18).

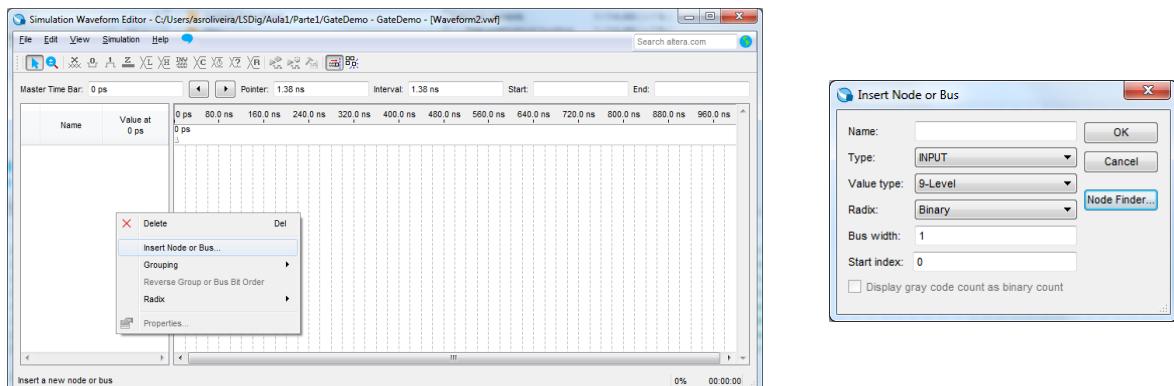


Figura 17 – Janela do simulador antes da especificação dos sinais de entrada e de saída a usar na simulação.

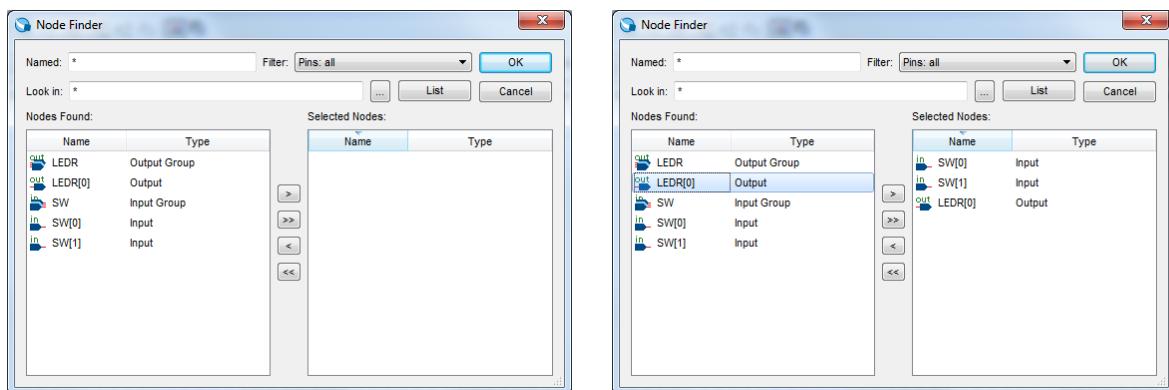


Figura 18 – Especificação dos sinais de entrada e de saída a usar na simulação.

12. Após premer “OK” poderá especificar os valores pretendidos para as entradas do circuito ao longo do tempo – o(s) valor(es) da(s) saída(s) será(ão) determinado(s) durante a simulação. Utilize para tal o rato, selecionando os troços do diagrama temporal do sinal pretendido com o valor lógico que deseja que ele assuma (Figura 19).

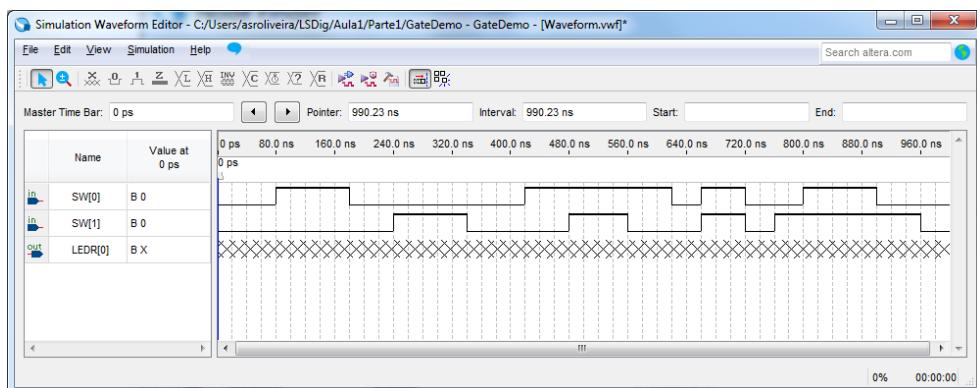


Figura 19 – Janela do simulador após a especificação das formas de onda dos sinais (vetores) de entrada.

13. Após especificar os valores (vetores) de simulação, grave o ficheiro com o nome “GateDemo.vwf” (Figura 20).

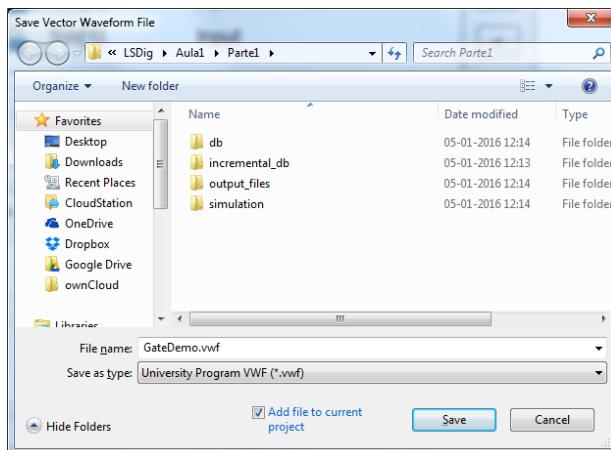


Figura 20 – Caixa de diálogo para gravação do ficheiro “GateDemo.vwf”.

14. Execute a simulação através do menu “*Simulation→Run Functional Simulation*”, o qual deve abrir uma janela semelhante à Figura 21. Após a simulação deve obter o valor da saída da porta lógica correspondente às entradas que especificou (Figura 22).

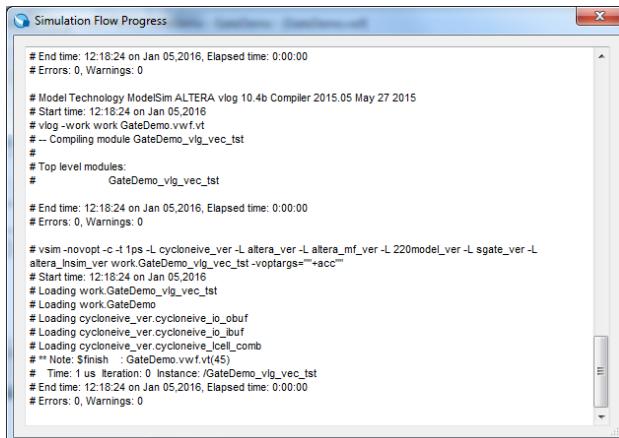


Figura 21 – Janela de compilação e execução da simulação do circuito com os vetores de entrada especificados.

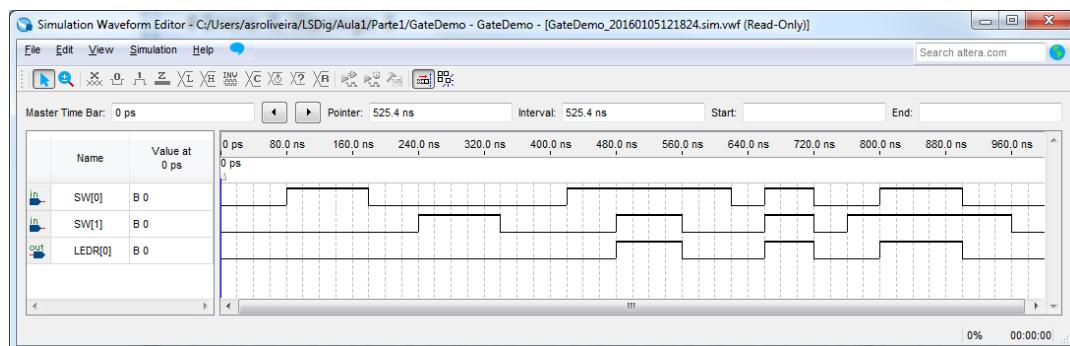


Figura 22 – Janela do simulador com a forma de onda do sinal de saída em função dos vetores de entrada especificados e do comportamento do circuito.

15. Para implementar o circuito em hardware real (FPGA), importe agora as definições de pinos da FPGA da placa de desenvolvimento que vai usar (localização física dos pinos aos quais estão ligados à FPGA os vários dispositivos da placa – e.g. LEDs, interruptores, displays, etc.). Para tal use o menu “*Assignments→Import Assignments*” (Figura 23).

O ficheiro que contém todas as definições de pinos da FPGA da placa DE2-115 é o “master.qsf”. Este ficheiro está disponível para *download* no site da UC (em elearning.ua.pt) e não deve ser alterado. Este ficheiro será usado ao longo do semestre, pelo que deve ser colocado num diretório base e comum a todos os projetos.

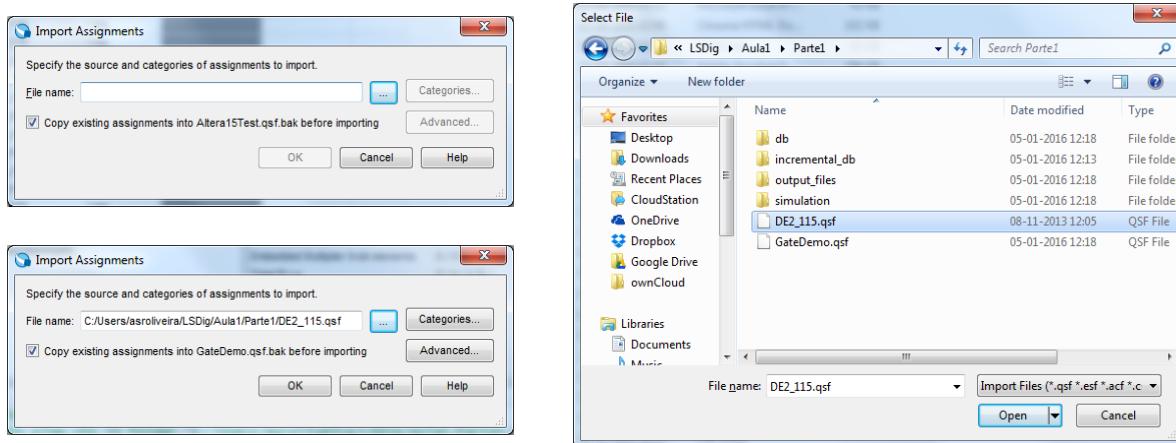


Figura 23 – Seleção e importação do ficheiro “master.qsf” com as definições dos pinos da FPGA ligados aos dispositivos do *kit*.

16. Efetue a síntese e implementação do projeto através do comando “*Compile Design*”. No final da compilação o IDE deve apresentar o aspeto da Figura 24.

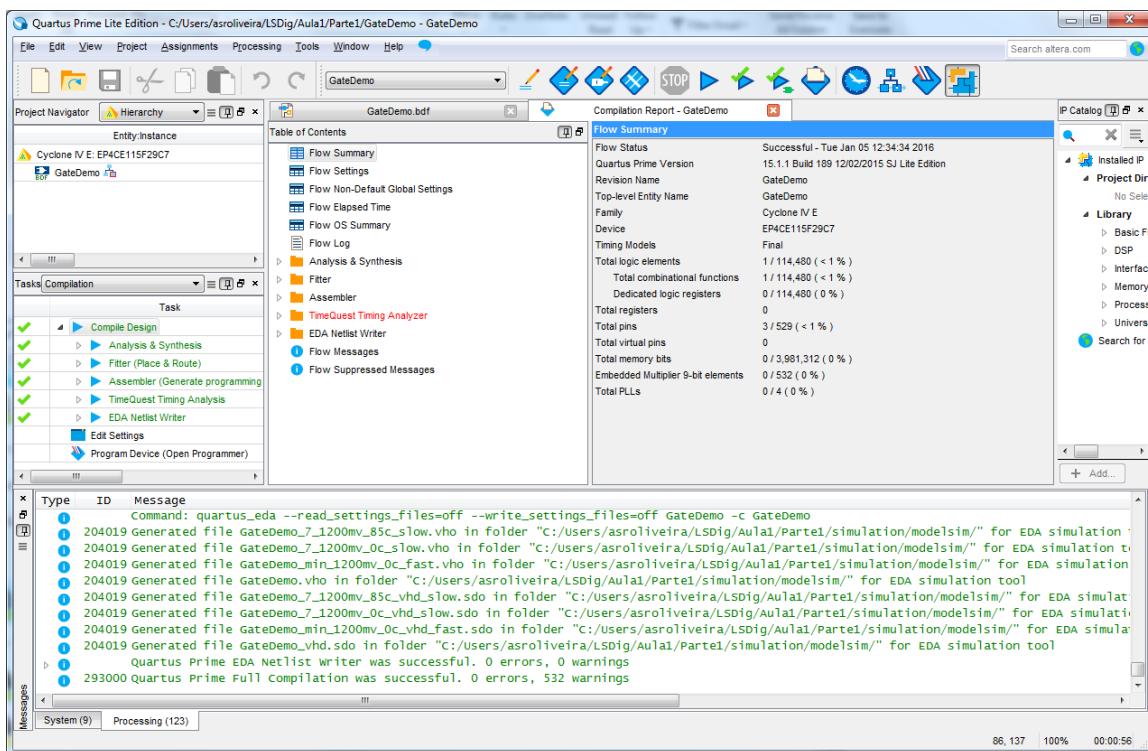


Figura 24 – “Altera Quartus Prime” IDE após compilação (implementação) completa do projeto.

17. No final do processo de compilação, programe a FPGA através do comando “*Program Device*” que deverá abrir a janela da Figura 25.

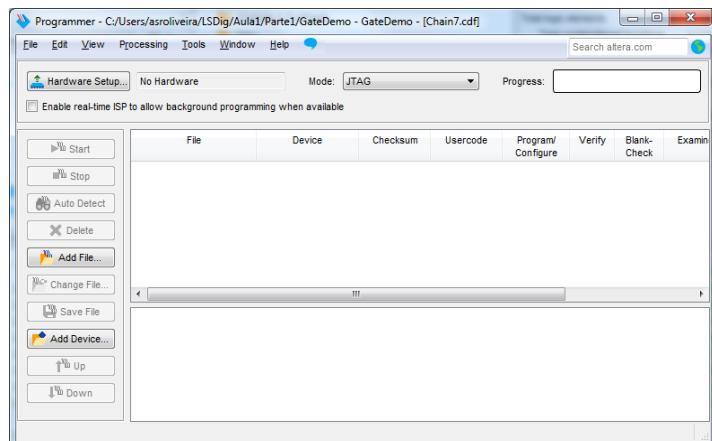


Figura 25 – Janela inicial da aplicação de programação da FPGA.

18. Configure a interface usada para programação da FPGA, premindo o botão “Hardware Setup” e selecionando a opção “USB-Blaster” (Figura 26). Caso esta opção não esteja disponível, verifique se a placa possui a alimentação ligada e está conectada ao computador através do cabo USB.

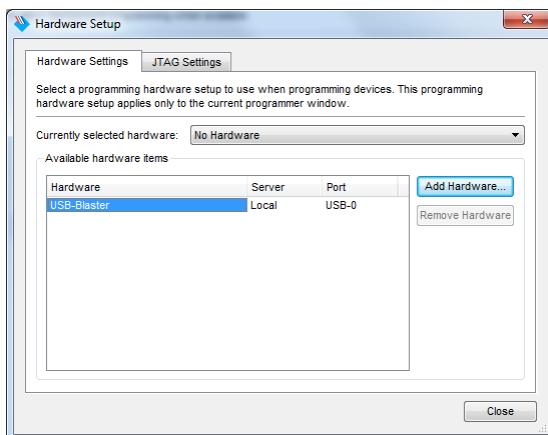


Figura 26 – Configuração da interface de programação da FPGA (USB-Blaster).

19. Premindo o botão “Add File”, abra o ficheiro de programação da FPGA com a configuração relativa ao circuito construído. O ficheiro deverá possuir o nome “GateDemo.sof” e encontrar-se no sub-diretório “output_files” do projeto (Figura 27).

Nota: se ao abrir a aplicação de programação, esta já apresentar o aspeto da Figura 28, não necessita de realizar a ação “Add File” descrita neste ponto.

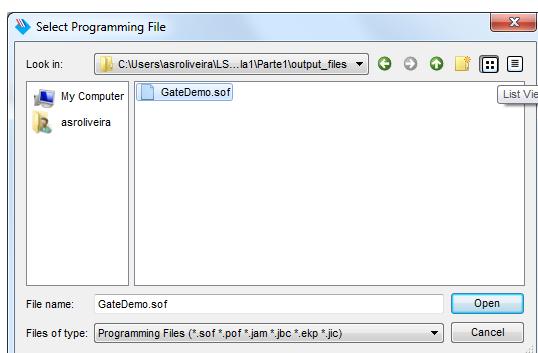


Figura 27 – Seleção do ficheiro de programação da FPGA (ficheiro “GateDemo.sof” no sub-diretório “output_files” do projeto).

20. Para programar a FPGA prima o botão “Start”. Quando a programação da FPGA estiver concluída (com sucesso), a aplicação deve apresentar o aspeto da Figura 29.

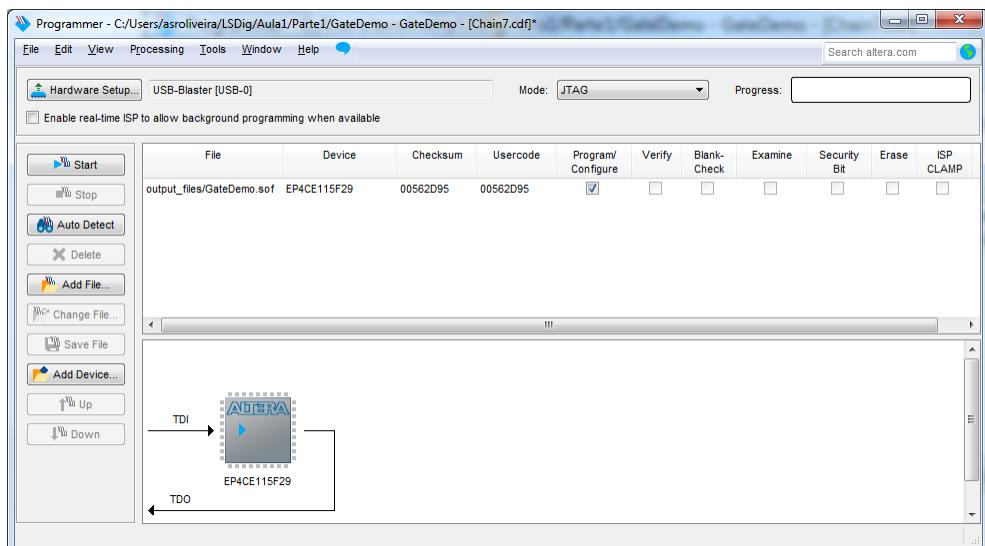


Figura 28 – Janela da aplicação antes da programação da FPGA (com indicação da FPGA usada e ficheiro de programação selecionado).

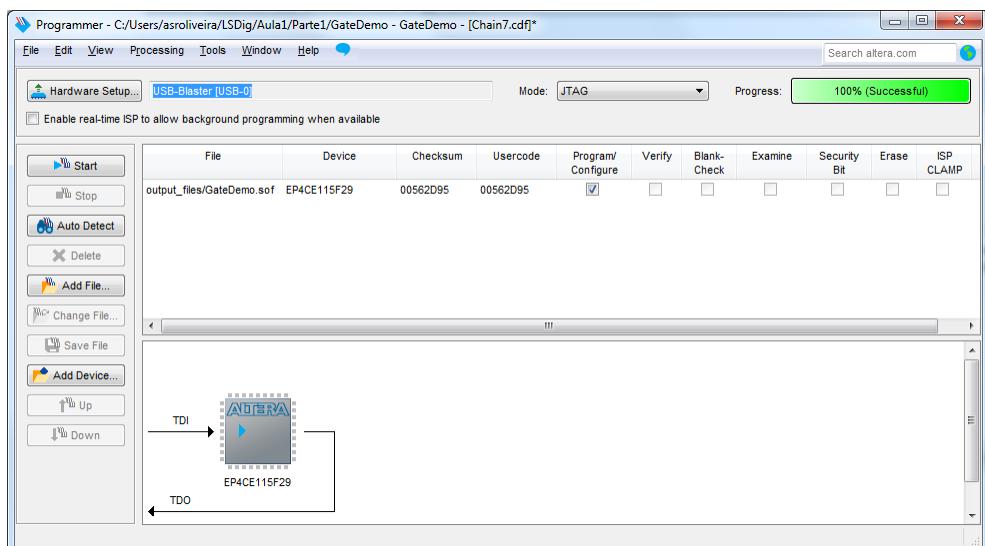


Figura 29 – Janela da aplicação após programação da FPGA.

21. Teste a porta lógica AND no *kit* de desenvolvimento aplicando diversos vetores de teste através dos interruptores usados e observando no LED o valor da saída.

22. Feche a aplicação de programação da FPGA. Recomenda-se que grave as configurações da aplicação de programação num ficheiro com o mesmo nome do projeto (neste caso “GateDemo.cdf”). Desta forma, quando voltar a abrir este ficheiro através da janela principal do “Altera Quartus Prime” a aplicação de programação já estará corretamente configurada.

23. Finalmente feche o projeto (menu “File→Close Project”).

Parte II

Demonstração do projeto hierárquico com base na linguagem de descrição de hardware VHDL

- 1.** Crie para a FPGA do *kit DE2-115 (Cyclone IV EP4CE115F29C7)* um novo projeto, seguindo os mesmos passos do ponto 1 da parte I deste guião. Considere a informação de identificação do projeto apresentada na Figura 30. O sumário da descrição do projeto deve ser semelhante ao da Figura 31.

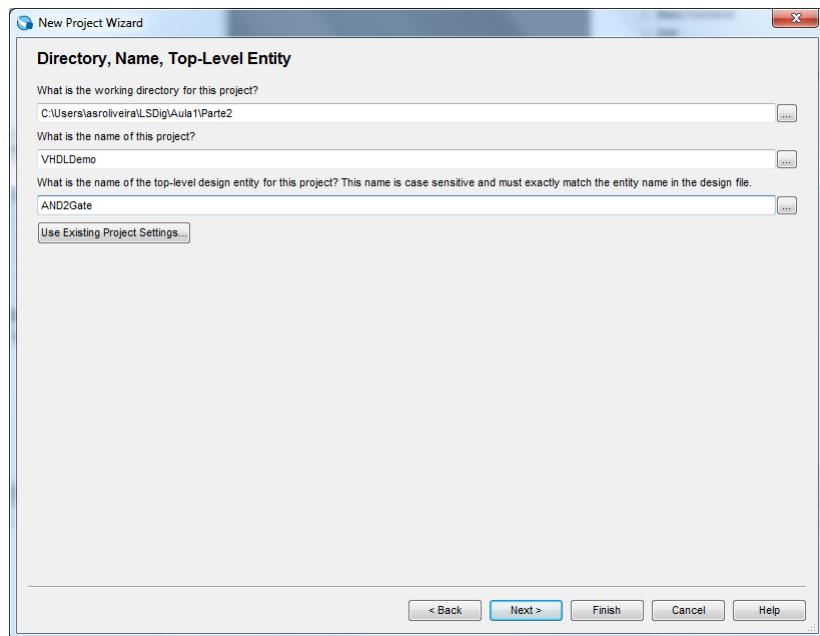


Figura 30 – Identificação e localização do projeto no sistema de ficheiros – adaptar de acordo com o diretório usado, o qual não deve conter no caminho (*path*) espaços nem caracteres especiais, e.g. acentos.

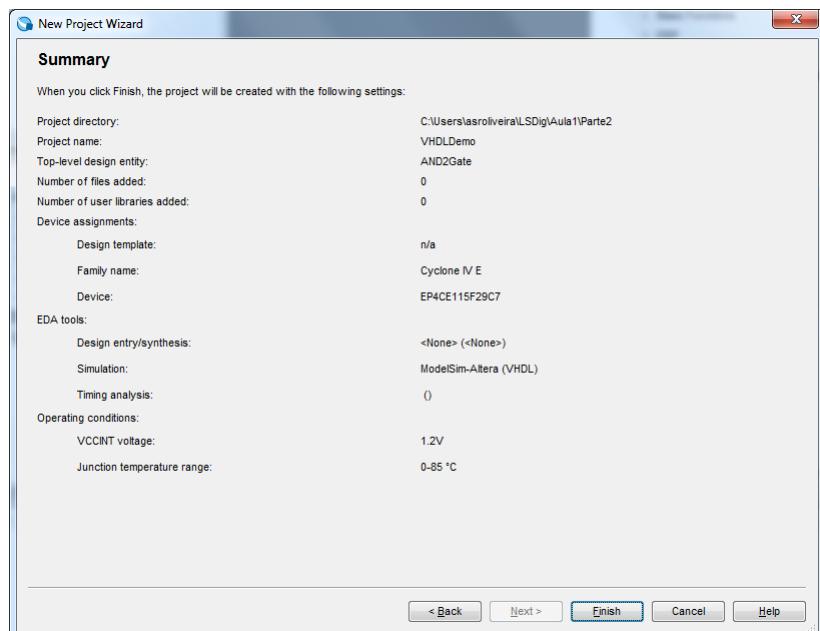


Figura 31 – Sumário final da criação do projeto.

2. Após premir “Finish” o IDE “Altera Quartus Prime” deve apresentar o aspeto da Figura 32.

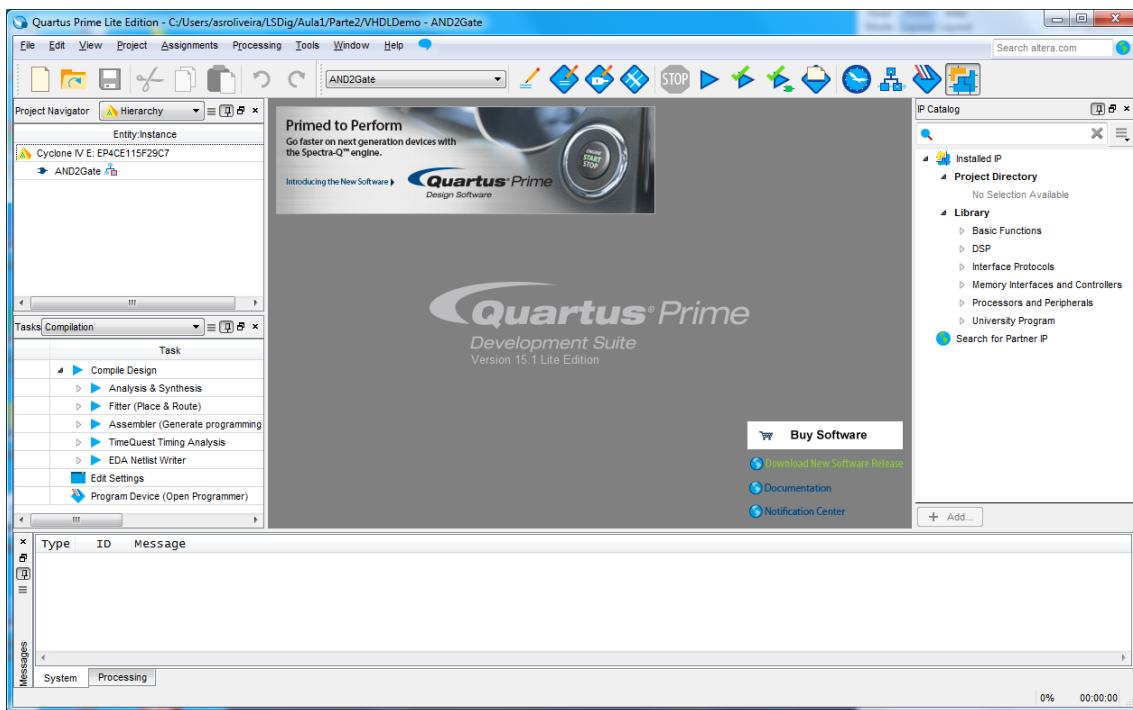


Figura 32 – “Altera Quartus Prime” IDE após a criação do projeto.

3. Crie um novo ficheiro para código fonte VHDL (menu “File→New”), de acordo com a Figura 33.

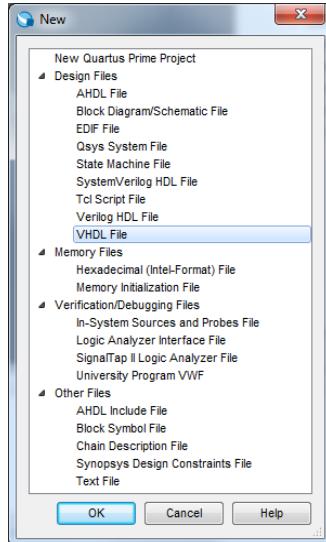


Figura 33 – Seleção do tipo de ficheiro a criar (VHDL File).

4. Introduza no ficheiro que acabou de criar o código VHDL correspondente a uma porta lógica AND de 2 entradas (mostrado na Figura 34).

The screenshot shows the Quartus Prime Lite Edition interface. The main window displays the VHDL code for an AND gate:

```

1 -- Bibliotecas
2 library IEEE;
3 use IEEE.STD_LOGIC_1164.all;
4
5 -- Interface (portos)
6 entity AND2Gate is
7     port(inPort0 : in std_logic;
8          inPort1 : in std_logic;
9          outPort : out std_logic);
10 end AND2Gate;
11
12 -- Implementação (descrição do funcionalidade)
13 architecture Behavioral of AND2Gate is
14 begin
15     outPort <= inPort0 and inPort1;
16 end Behavioral;
17
18
19

```

The Project Navigator shows the project structure with an Entity named "AND2Gate". The IP Catalog pane lists various Altera IP cores. The Tasks pane shows the compilation status.

Figura 34 – Código fonte da porta lógica AND de 2 entradas (“AND2Gate.vhd”).

5. Grave o ficheiro, cujo nome deverá ser “AND2Gate.vhd” (Figura 35).

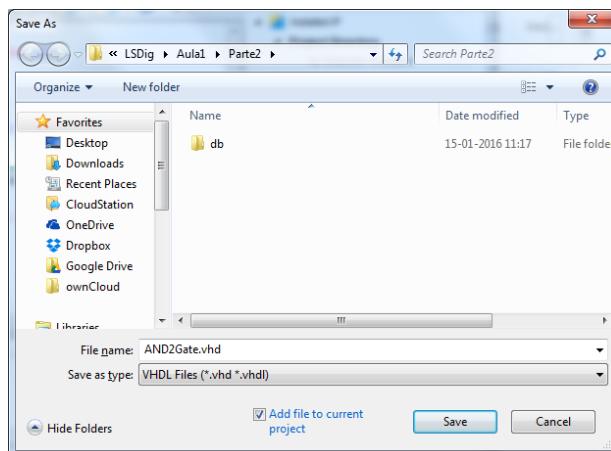


Figura 35 – Caixa de diálogo para gravação do ficheiro “AND2Gate.vhd”.

6. Seguidamente vai ser validado por simulação o comportamento da porta lógica modelada. No entanto, antes de efetuar a simulação, execute a opção “*Analysis & Synthesis*” para que a correção sintática e estrutura do projeto sejam analisadas. Após a execução da “*Analysis & Synthesis*” o IDE deve apresentar o aspeto da Figura 36.

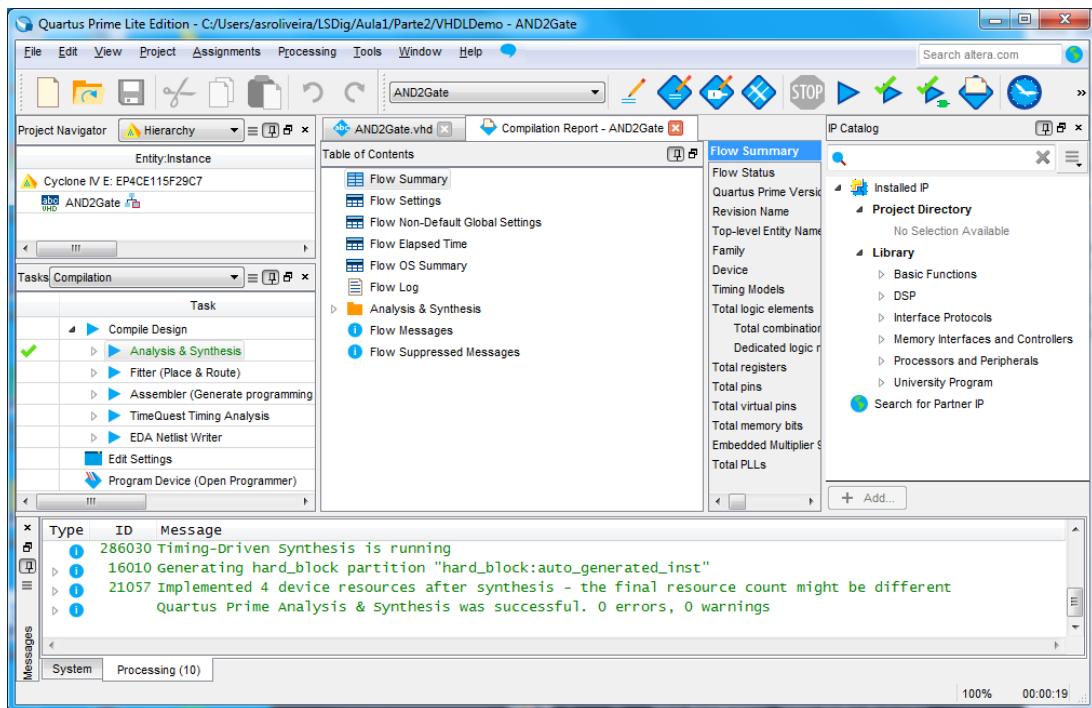


Figura 36 – “Altera Quartus Prime” IDE após a execução da “Analysis & Synthesis”.

7. Simule o comportamento da porta lógica que acabou de descrever, criando para tal um ficheiro VWF (menu “File→New”) (Figura 37).

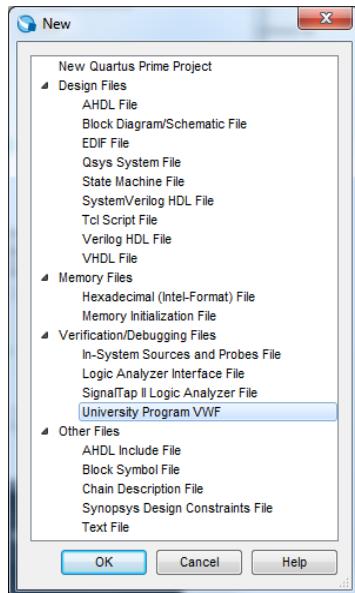


Figura 37 – Seleção do tipo de ficheiro a criar (University Program VWF).

8. Após premer “OK” abrirá uma janela onde deverão ser indicados os sinais usados na simulação. À semelhança da simulação realizada na primeira parte do guião, através do menu “Edit→Insert→Insert Node or Bus”, premindo de seguida os botões “Node Finder” e “List”, selecione todos os portos de entrada e saída da “AND2Gate”. Após premer “OK” poderá especificar os valores pretendidos ao longo do tempo para as entradas da “AND2Gate”. Utilize para tal o rato, selecionando os troços do diagrama temporal do sinal pretendido com o valor lógico que deseja que ele assuma (Figura 38).

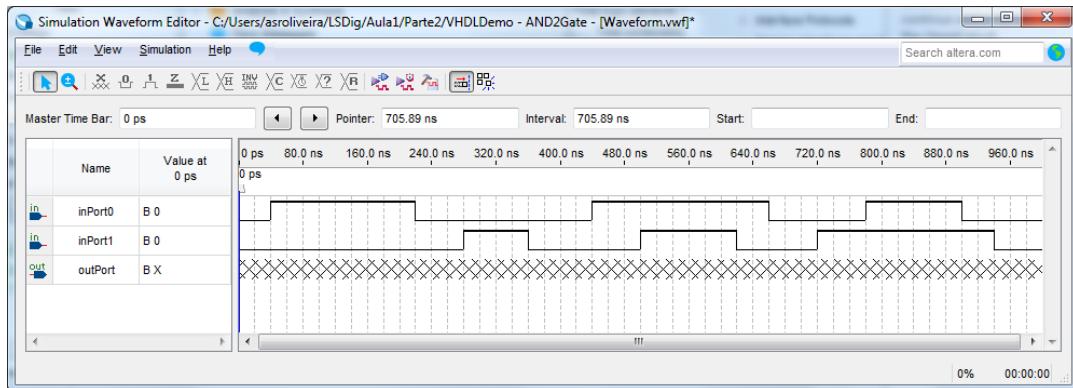


Figura 38 – Janela do simulador após a especificação das formas de onda (vetores) de entrada.

9. Após especificar os valores (vetores) de simulação, grave o ficheiro com o nome “AND2Gate.vwf”.

10. Execute a simulação através do menu “*Simulation→Run Functional Simulation*”. Após a simulação deve obter o valor da saída da porta lógica AND correspondente às entradas que especificou (Figura 39).

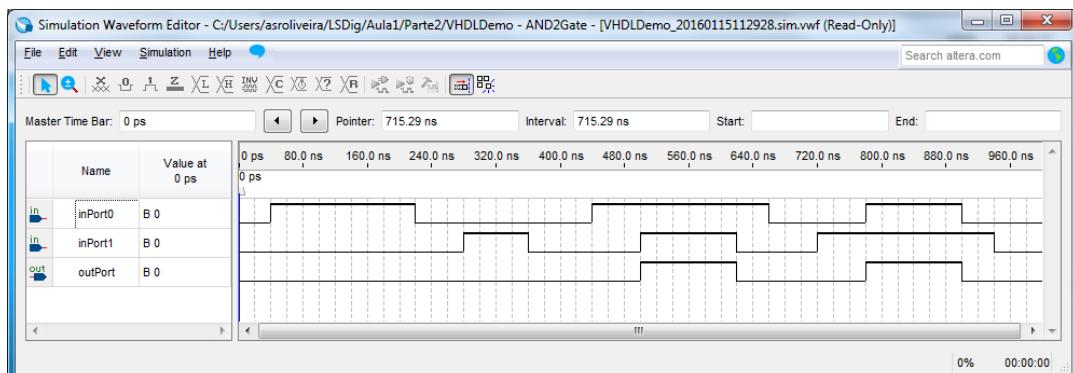


Figura 39 – Janela do simulador após a simulação.

11. Crie um novo ficheiro VHDL, chamado “GateDemo.vhd”, que irá servir para instanciar a porta lógica que acabou de descrever nos pontos anteriores e associá-la a pinos concretos da FPGA do *kit* de desenvolvimento que vai usar para a testar. Introduza o código VHDL mostrado na Figura 40 no ficheiro que acabou de criar, para instanciar a porta lógica e ligá-la a pinos adequados da FPGA (neste caso as entradas vão ser ligadas a interruptores e a saída ligada a um LED). Após editar, grave o ficheiro com o nome “GateDemo.vhd”.

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;

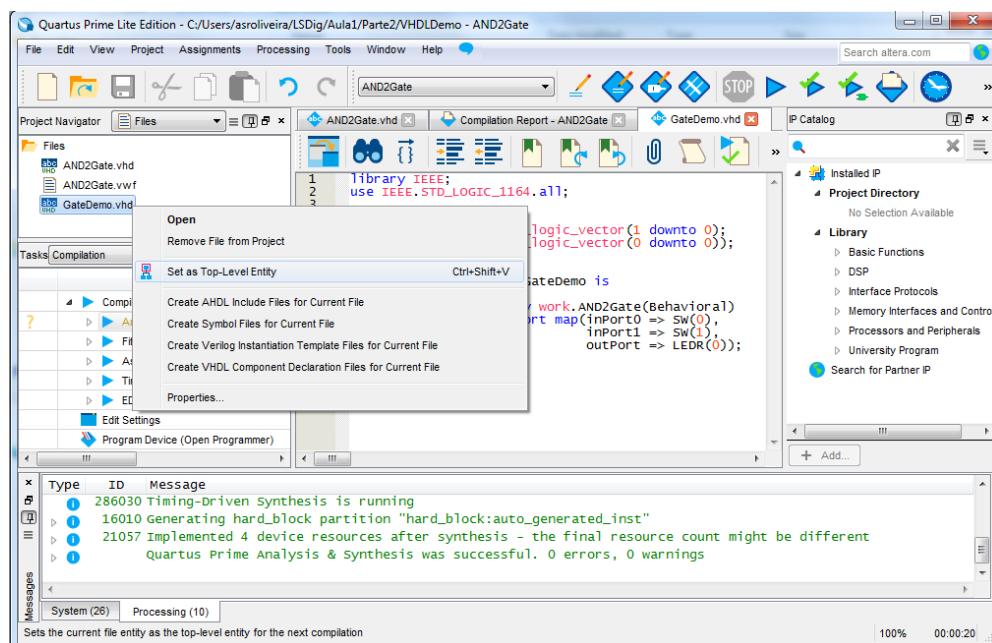
entity GateDemo is
    port(SW : in std_logic_vector(1 downto 0);
        LEDR : out std_logic_vector(0 downto 0));
end GateDemo;

architecture shell of GateDemo is
begin
    system_core : entity work.AND2Gate(Behavioral)
        port map(inPort0 => SW(0),
                  inPort1 => SW(1),
                  outPort => LEDR(0));
end shell;

```

Figura 40 – Código fonte do módulo *top-level* (“GateDemo.vhd”).

12. Altere o ficheiro *top-level* do projeto (ficheiro “principal” que inclui todos os outros e portanto está no nível hierárquico mais elevado do projeto) de acordo com a Figura 41. O *top-level* deixa de ser o “AND2Gate.vhd” e passa a ser o “GateDemo.vhd”.

Figura 41 – Alteração do módulo *top-level* para o “GateDemo.vhd”.

13. Importe as definições de pinos da FPGA da placa de desenvolvimento, usando o menu “Assignments→Import Assignments” (Figura 42).

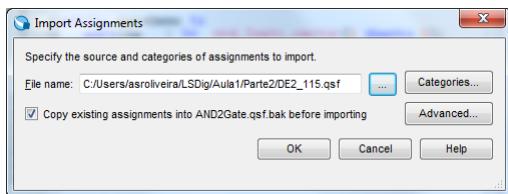


Figura 42 – Importação do ficheiro “master.qsf” com as definições dos pinos da FPGA ligados aos dispositivos do kit.

14. Efetue a síntese e implementação do projeto através do comando “*Compile Design*”. No final da compilação o IDE deve apresentar o aspeto da Figura 43.

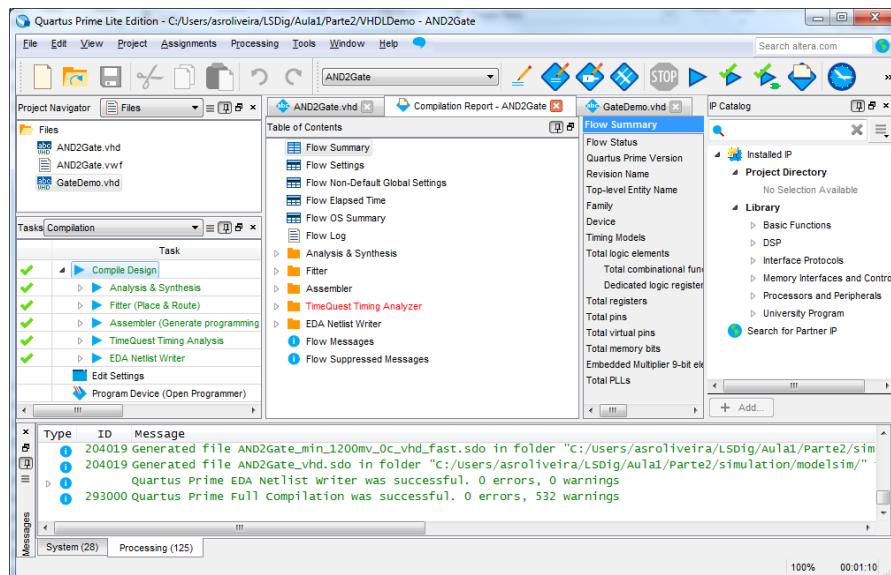


Figura 43 – “Altera Quartus Prime” IDE após compilação (implementação) completa do projeto.

15. No final do processo de compilação, programe a FPGA. Quando estiver concluída (com sucesso) a programação da FPGA, a aplicação de programação deve apresentar o aspeto da Figura 44.

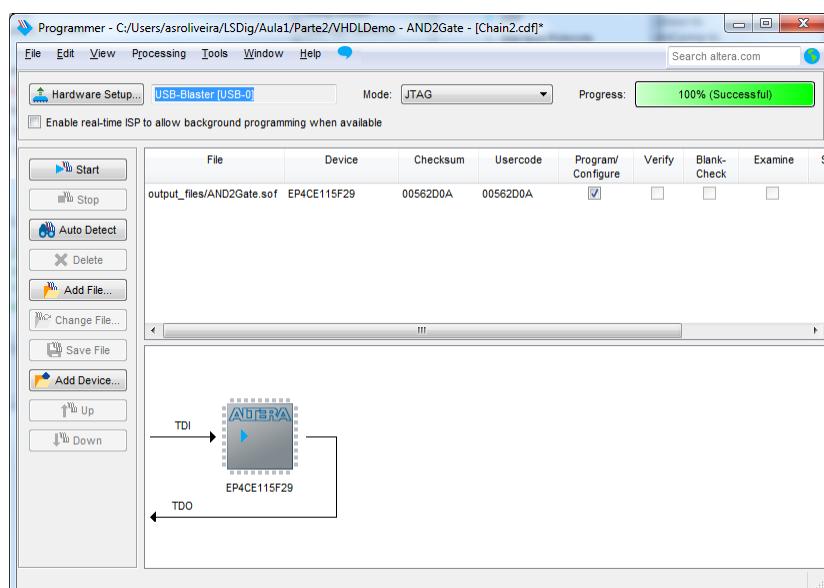
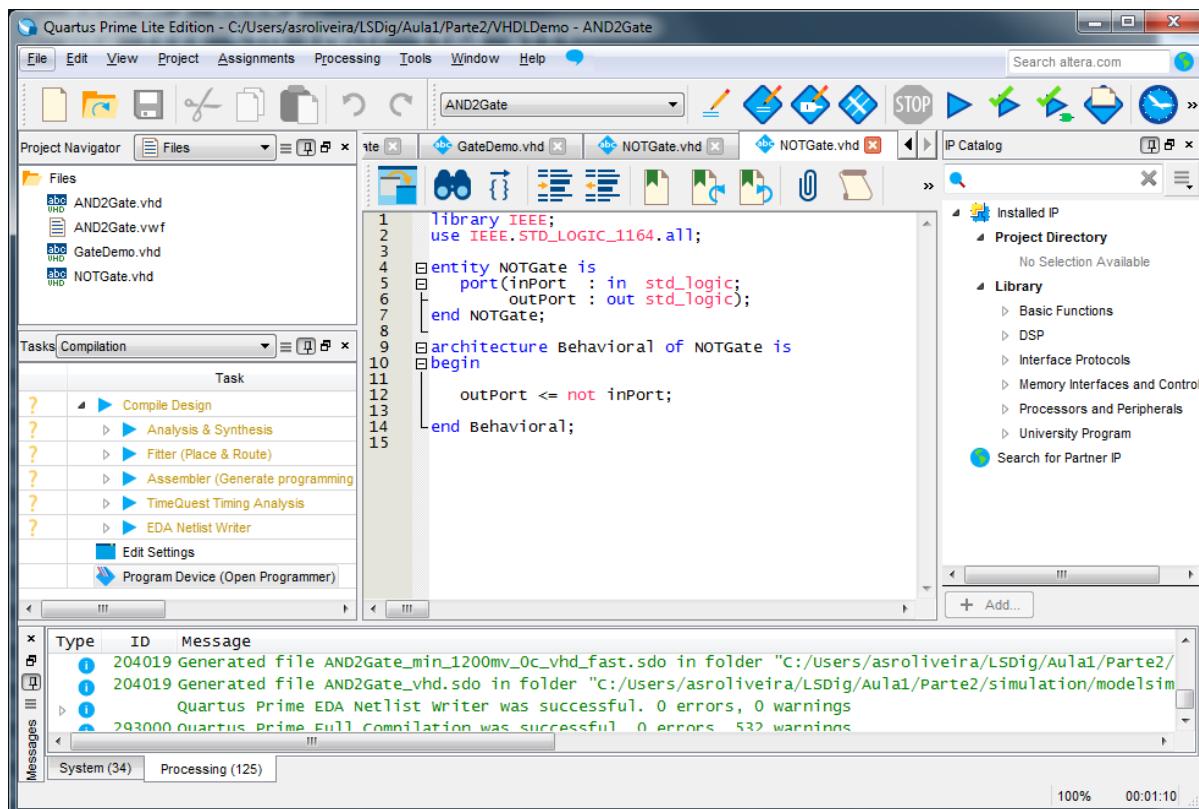


Figura 44 – Janela da aplicação após programação da FPGA.

16. Teste a porta lógica AND no *kit* de desenvolvimento aplicando diversos vetores de teste através dos interruptores usados e observando no LED o valor da saída.

17. Crie um novo ficheiro VHDL, contendo o código fonte correspondente a um inversor e no final grave com o nome “NOTGate.vhd” (Figura 45).



The screenshot shows the Quartus Prime Lite Edition interface. The main window displays the VHDL code for a NOT gate:

```

1 Library IEEE;
2 use IEEE.STD_LOGIC_1164.all;
3
4 entity NOTGate is
5     port(inPort : in std_logic;
6          outPort : out std_logic);
7 end NOTGate;
8
9 architecture Behavioral of NOTGate is
10 begin
11
12     outPort <= not inPort;
13
14 end Behavioral;
15 
```

The Project Navigator on the left shows files: AND2Gate.vhd, AND2Gate.wvf, GateDemo.vhd, and NOTGate.vhd. The Messages panel at the bottom shows compilation logs:

- 204019 Generated file AND2Gate_min_1200mv_0c_vhd_fast.sdo in folder "C:/users/asroliveira/LSDig/Aula1/Parte2/
- 204019 Generated file AND2Gate_vhd.sdo in folder "C:/users/asroliveira/LSDig/Aula1/Parte2/simulation/modelsim"
- Quartus Prime EDA Netlist Writer was successful. 0 errors, 0 warnings
- 293000 Quartus Prime Full compilation was successful. 0 errors, 532 warnings

Figura 45 – Código fonte do inversor (“NOTGate.vhd”).

18. Crie um novo ficheiro VHDL, contendo o código correspondente a uma porta lógica NAND de 2 entradas, construída a partir das duas portas lógicas (AND e inversor) implementadas nos pontos anteriores, instanciadas e interligadas de acordo com a estrutura ilustrada na Figura 46 e o código da Figura 47.

Nota: este ponto pretende ilustrar apenas a instanciação e interligação de componentes descritos em VHDL, com base num exemplo simples, não sendo a forma mais eficaz de criar uma porta lógica NAND, uma vez que a linguagem VHDL também disponibiliza o operador “nand”. No entanto, esta forma tem a vantagem de ilustrar o projeto hierárquico em VHDL, com um exemplo simples, a instanciação e interligação textual de componentes, de forma análoga aos métodos de captura de diagrama esquemático.

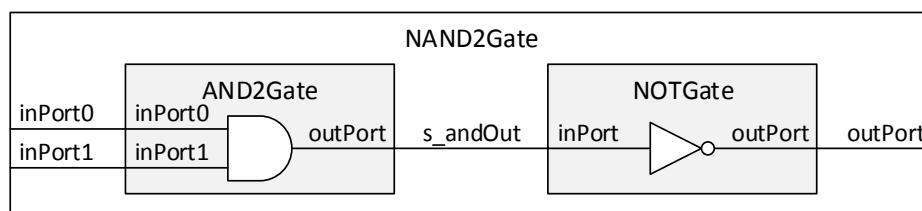
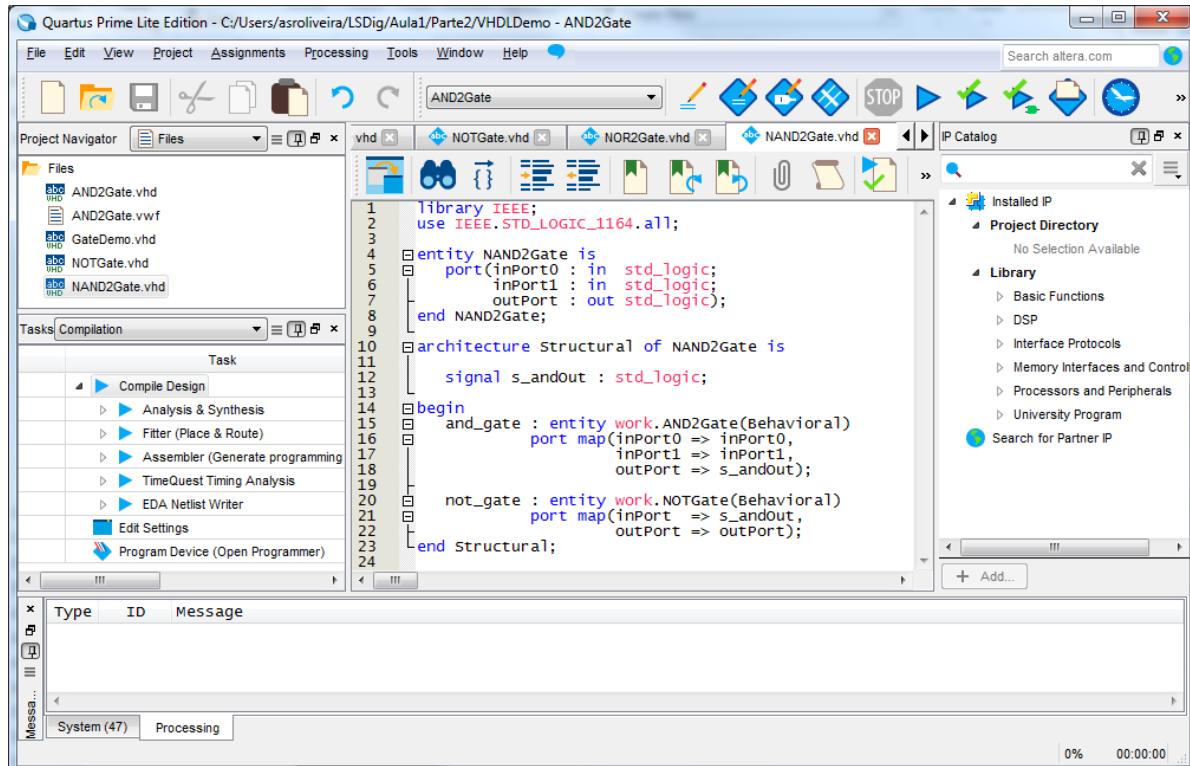


Figura 46 – Estrutura da porta lógica NAND de 2 entradas com indicação das ligações dos portos de entrada e de saída.



```

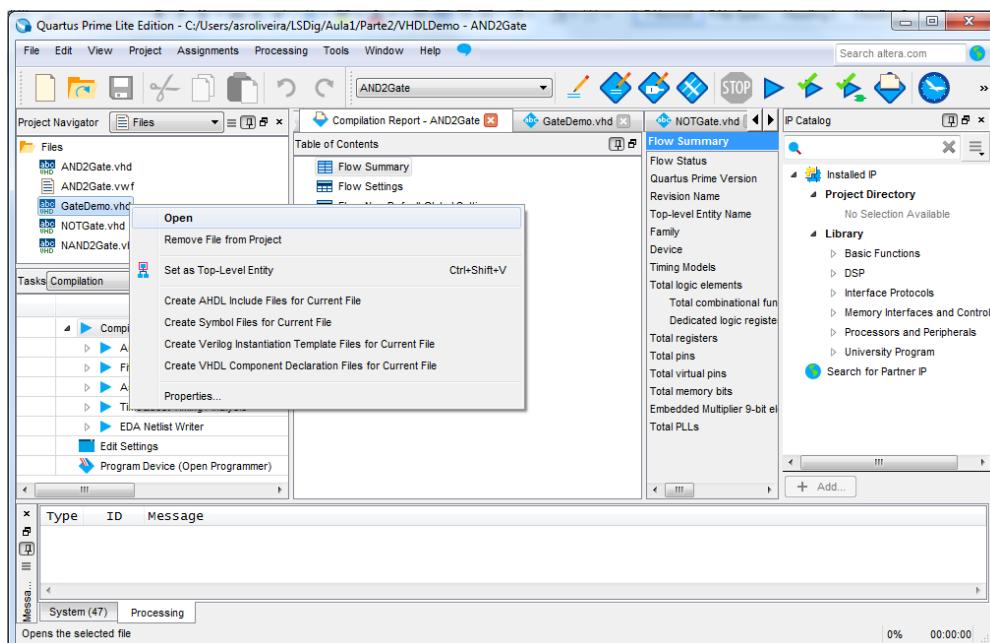
1 Library IEEE;
2 use IEEE.STD_LOGIC_1164.all;
3
4 entity NAND2Gate is
5     port(inPort0 : in std_logic;
6          inPort1 : in std_logic;
7          outPort : out std_logic);
8 end NAND2Gate;
9
10 architecture Structural of NAND2Gate is
11
12     signal s_andout : std_logic;
13
14     begin
15         and_gate : entity work.AND2Gate(Behavioral)
16             port map(inPort0 => inPort0,
17                       inPort1 => inPort1,
18                       outPort => s_andout);
19
20         not_gate : entity work.NOTGate(Behavioral)
21             port map(inPort => s_andout,
22                       outPort => outPort);
23
24     end structural;

```

Figura 47 – Código fonte da porta lógica NAND de 2 entradas (“NAND2Gate.vhd”).

19. Grave o ficheiro com o nome “NAND2Gate.vhd”.

20. Edite o ficheiro *top-level* “GateDemo.vhd” de forma a que seja usado o componente NAND2Gate (arquitetura Structural) em vez do AND2Gate (arquitetura Behavioral) usado anteriormente – Figuras 48 e 49.

Figura 48 – Abertura do módulo *top-level* “GateDemo.vhd”.

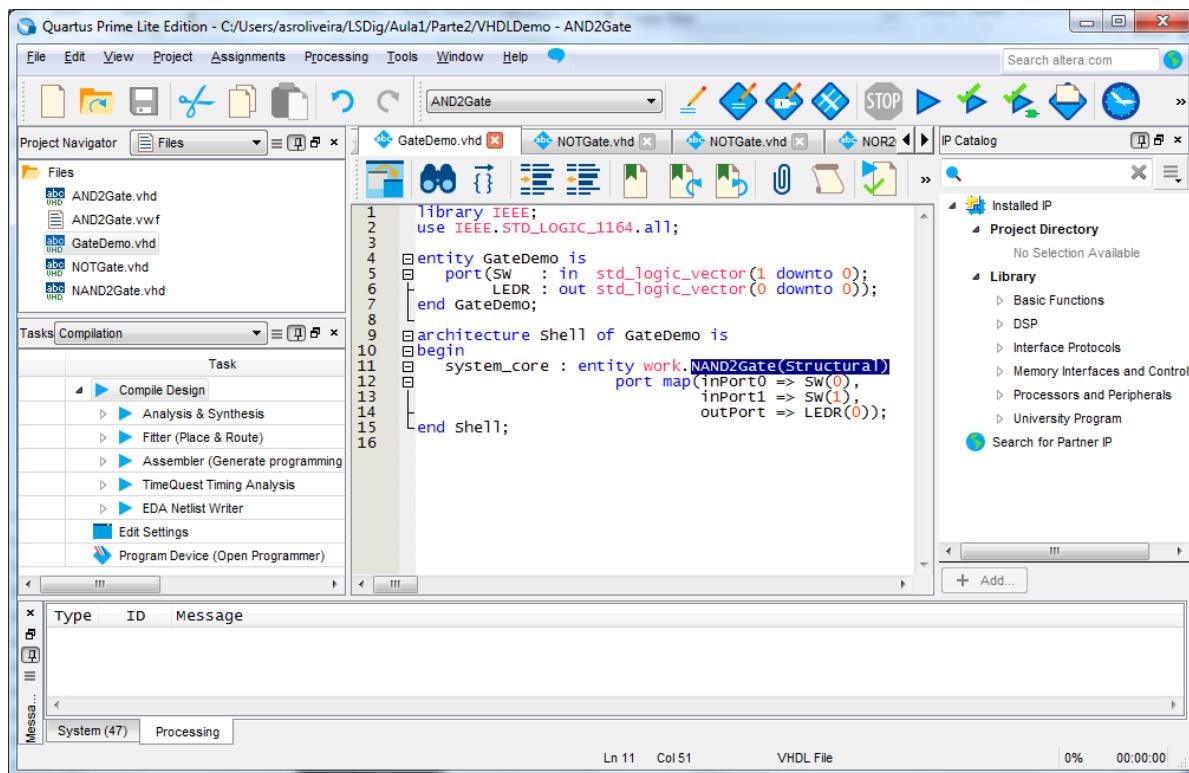


Figura 49 – Edição e alteração do módulo top-level “GateDemo.vhd”.

21. Volte a importar o ficheiro “master.qsf” com as definições dos pinos da FPGA na placa de desenvolvimento.

22. Efetue a síntese e implementação do sistema através do comando “*Compile Design*”. No final da compilação o IDE deve apresentar o aspeto da Figura 50.

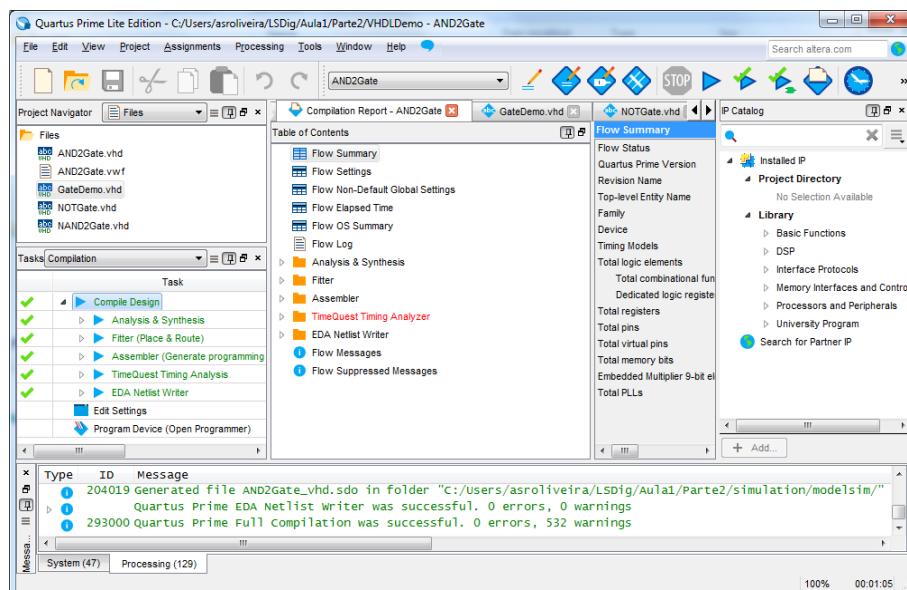


Figura 50 – “Altera Quartus Prime” IDE após compilação completa do projeto.

23. Programe a FPGA e teste no *kit* o funcionamento da porta lógica NAND implementada.

24. Feche a aplicação de programação da FPGA e seguidamente o projeto.

Parte III

Demonstração do projeto híbrido baseado em diagramas esquemáticos e descrições VHDL

- Crie para a FPGA do *kit DE2-115 (Cyclone IV EP4CE115F29C7)* um novo projeto, seguindo os mesmos passos do ponto 1 da parte I deste guião. Considere a informação de identificação do projeto apresentada na Figura 51. O sumário da descrição do projeto deve ser semelhante à Figura 52.

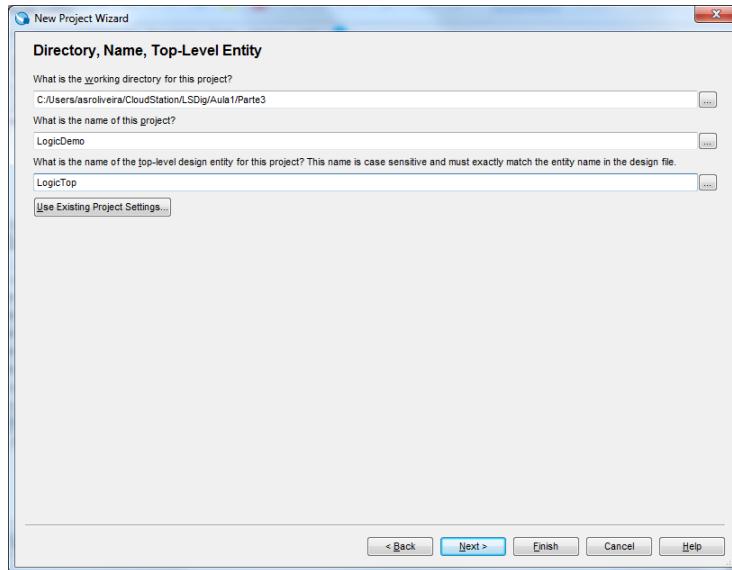


Figura 51 – Identificação e localização do projeto no sistema de ficheiros.

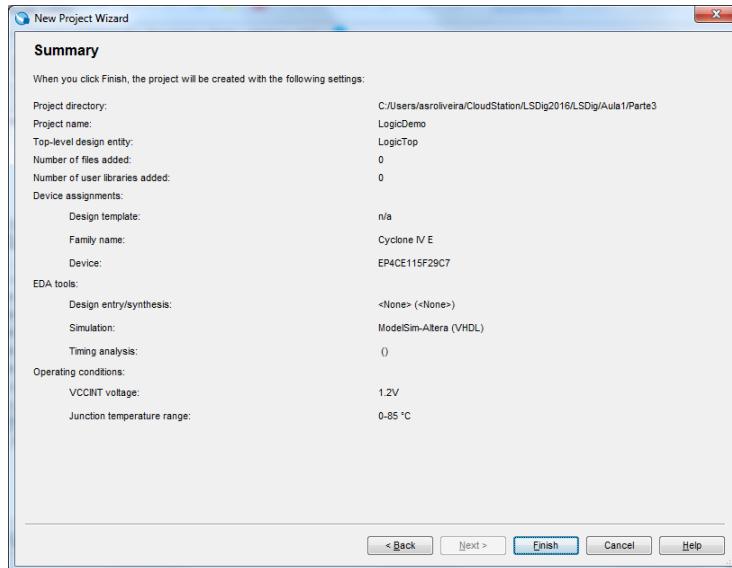


Figura 52 – Sumário final da criação do projeto.

- Crie um novo ficheiro para código fonte VHDL (menu “File→New”).
- Introduza no ficheiro que acabou de criar o código VHDL da Figura 53, correspondente a uma unidade que realiza diversas operações lógicas.
- Grave o ficheiro, cujo nome deverá ser “LogicUnit.vhd”.

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity LogicUnit is
    port(input0 : in std_logic;
          input1 : in std_logic;
          invOut : out std_logic;
          andOut : out std_logic;
          orOut : out std_logic;
          xorOut : out std_logic;
          nandOut : out std_logic;
          norOut : out std_logic);
end LogicUnit;

architecture Behavioral of LogicUnit is
begin
    invOut <= not input0;
    andOut <= input0 and input1;
    orOut <= input0 or input1;
    xorOut <= input0 xor input1;
    nandOut <= input0 nand input1;
    norOut <= input0 nor input1;
end Behavioral;

```

Figura 53 – Código fonte do módulo “LogicUnit” (exemplo em VHDL com operadores lógicos).

5. Para poder instanciar num diagrama esquemático o módulo “LogicUnit.vhd” escrito em VHDL é necessário criar um símbolo tal como ilustrado na Figura 54.

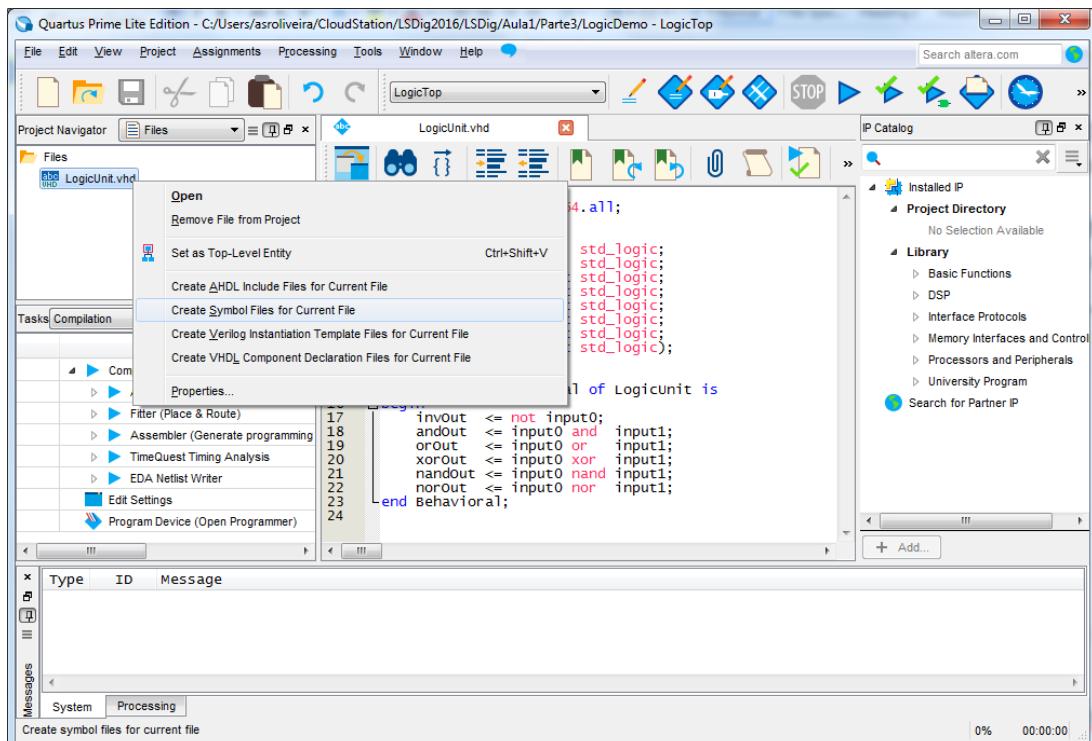


Figura 54 – Criação de um símbolo para um módulo de forma a poder ser utilizado num diagrama esquemático.

6. Crie um novo ficheiro para um diagrama esquemático (menu “File→New”), que irá servir para instanciar a entidade “LogicUnit” e associá-la a pinos adequados da FPGA (entradas ligadas aos interruptores e saídas ligada a LEDs) do kit de desenvolvimento que vai usar para a testar.

7. Uma vez criado o símbolo, um módulo definido pelo utilizador pode ser usado num diagrama esquemático da mesma forma que qualquer bloco predefinido, usando o botão “Symbol Tool” da barra de ferramentas. Crie uma instância do módulo “LogicUnit” (Figura 55).

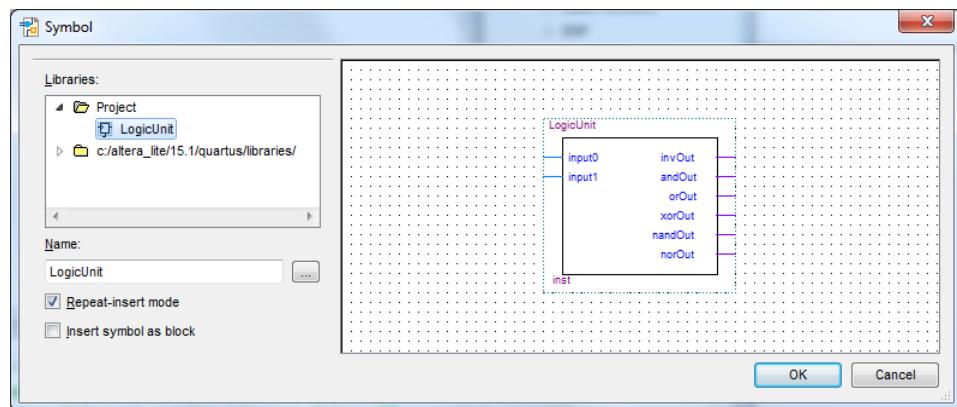


Figura 55 – Instanciação num diagrama esquemático do módulo “LogicUnit”.

8. Ligue as entradas e as saídas do módulo “LogicUnit” a portos de entrada e de saída tal como ilustrado na Figura 56 (entradas ligadas a interruptores e saída a LEDs do kit).

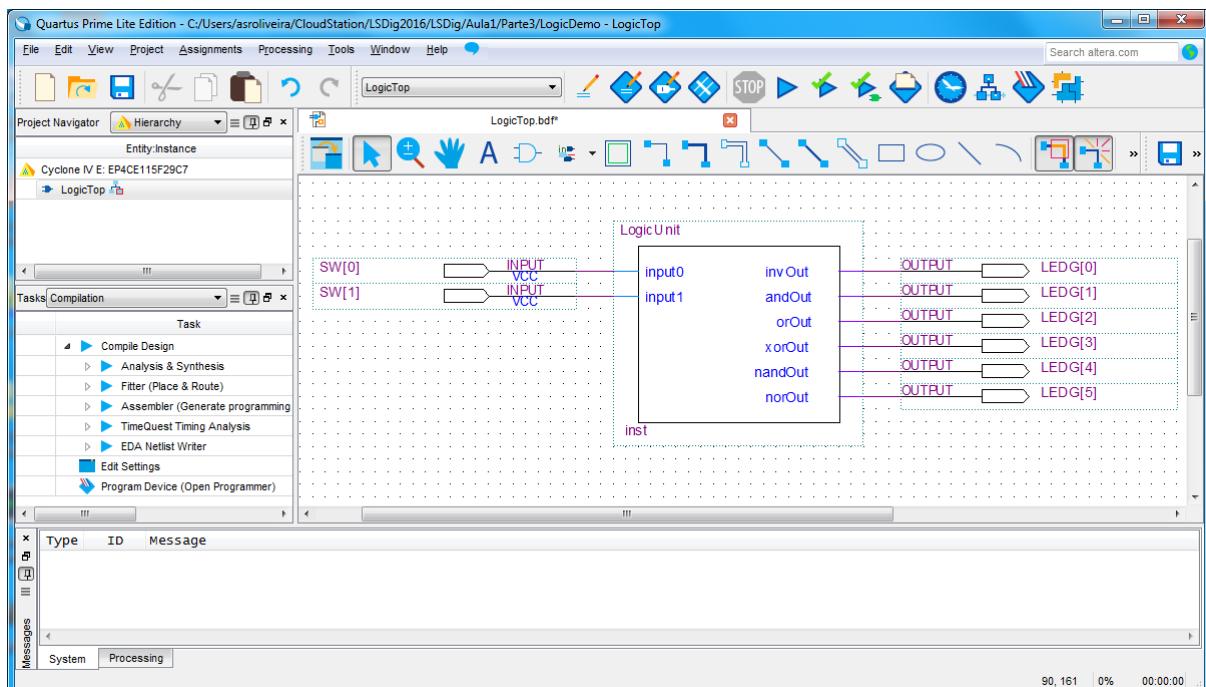


Figura 56 – Interligação do módulo “LogicUnit” e dos portos de entrada e de saída e identificação dos diversos elementos do circuito no módulo “LogicDemo”.

9. Grave o ficheiro, cujo nome deverá ser “LogicTop.bdf”.

10. Importe o ficheiro “master.qsf” com as definições dos pinos da FPGA na placa de desenvolvimento. Após a importação, o diagrama esquemático deverá apresentar o aspeto da Figura 57.

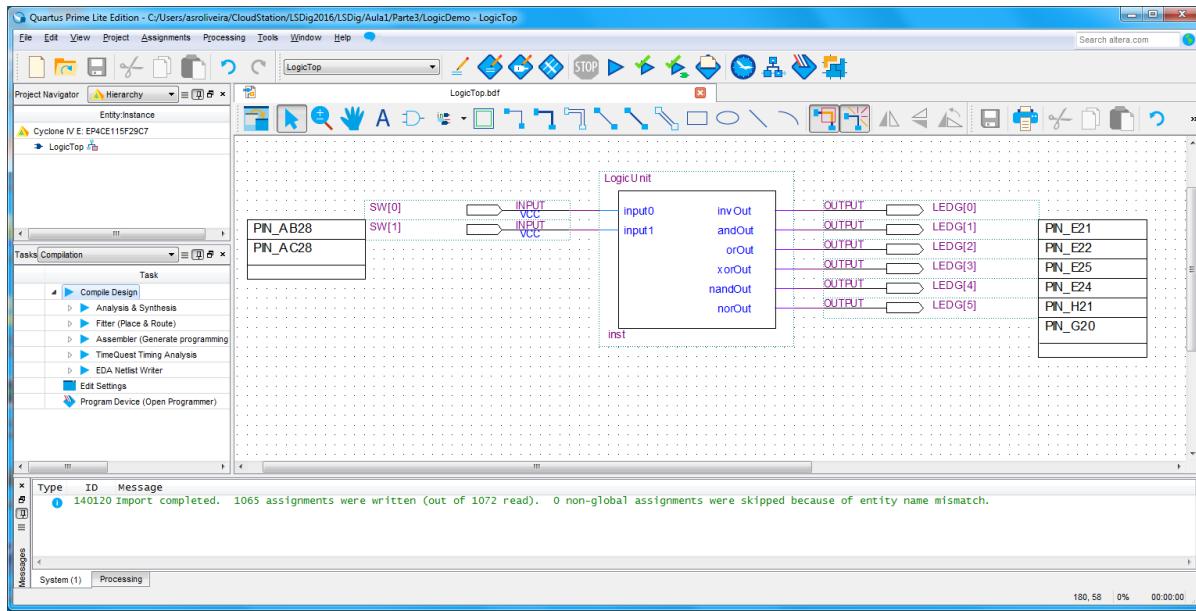


Figura 57 – Módulo “LogicUnit” e portos de entrada e de saída com identificação dos pinos usados da FPGA.

11. Efetue a síntese e implementação do projeto através do comando “*Compile Design*”. No final da compilação o IDE deve apresentar o aspeto da Figura 58.

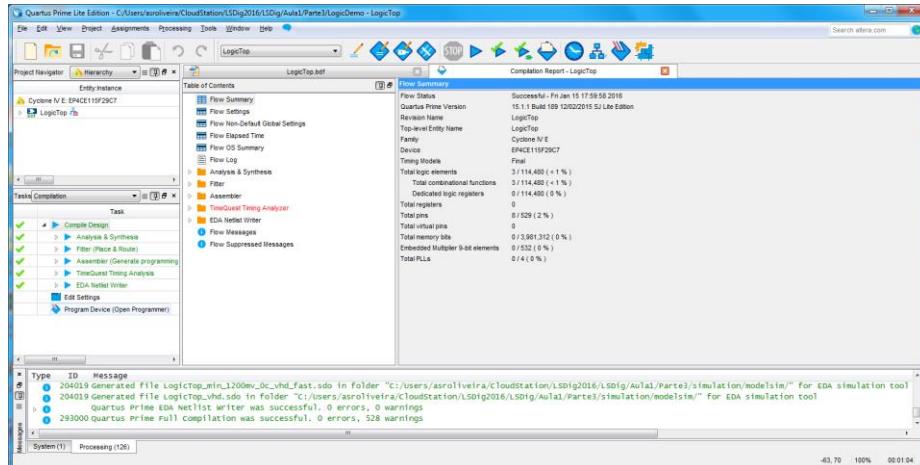


Figura 58 – “Altera Quartus Prime” IDE após compilação (implementação) completa do projeto.

12. No final do processo de compilação, programe a FPGA através do comando “*Program Device*”.

13. Teste o projeto no *kit* de desenvolvimento aplicando diversos vetores de teste através dos interruptores usados e observando nos LEDs os valores das saídas para as diversas funções lógicas.

14. Feche a aplicação de programação da FPGA e seguidamente o projeto.

Parte IV

Demonstração das vantagens de VHDL na modelação comportamental de componentes

1. Crie para a FPGA do *kit DE2-115 (Cyclone IV EP4CE115F29C7)* um novo projeto, chamado “EqCmpDemo”, cuja entidade *top-level* deverá chamar-se “EqCmpDemo” e seguindo os mesmos passos do ponto 1 da parte I deste guião.
2. Crie um novo ficheiro para um diagrama esquemático, chamado “EqCmp4.bdf”, para implementar um comparador de igualdade de palavras de 4 bits (Figura 59). No final da edição do esquema lógico, o circuito deve ser semelhante ao da Figura 60. Utilize os componentes “xnor” e “and4” da biblioteca do “Altera Quartus Prime” acessível através do “Symbol Tool”.

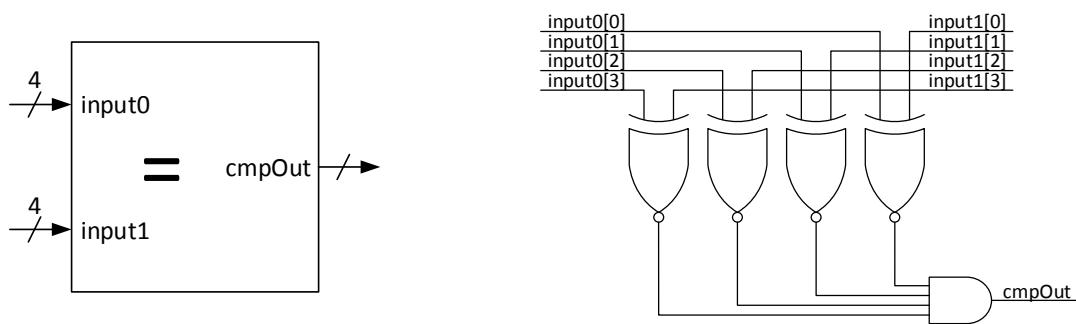


Figura 59 – Diagrama lógico do comparador de igualdade de palavras de 4 bits.

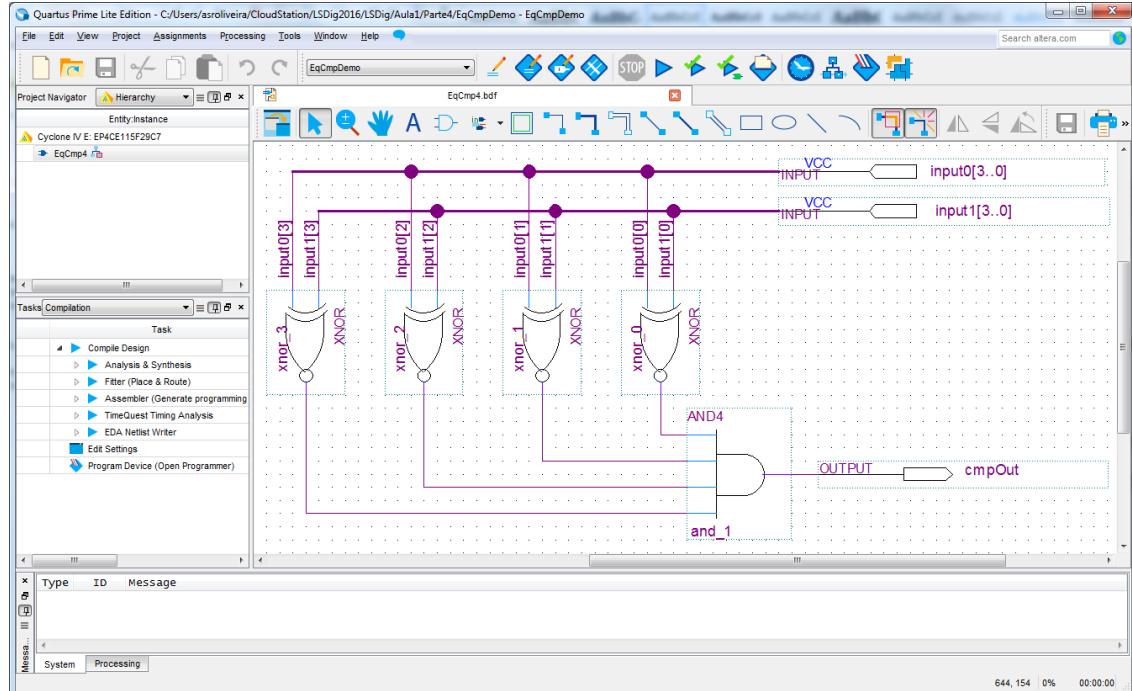


Figura 60 – Diagrama lógico do comparador de igualdade de palavras de 4 bits com interligação e identificação dos diversos elementos do circuito.

3. Crie um símbolo para o módulo “EqCmp4”, de forma a poder ser usado num diagrama esquemático, tal como ilustrado na Figura 61, e grave-o com o nome “EqCmp4.bsf” (Figura 62).

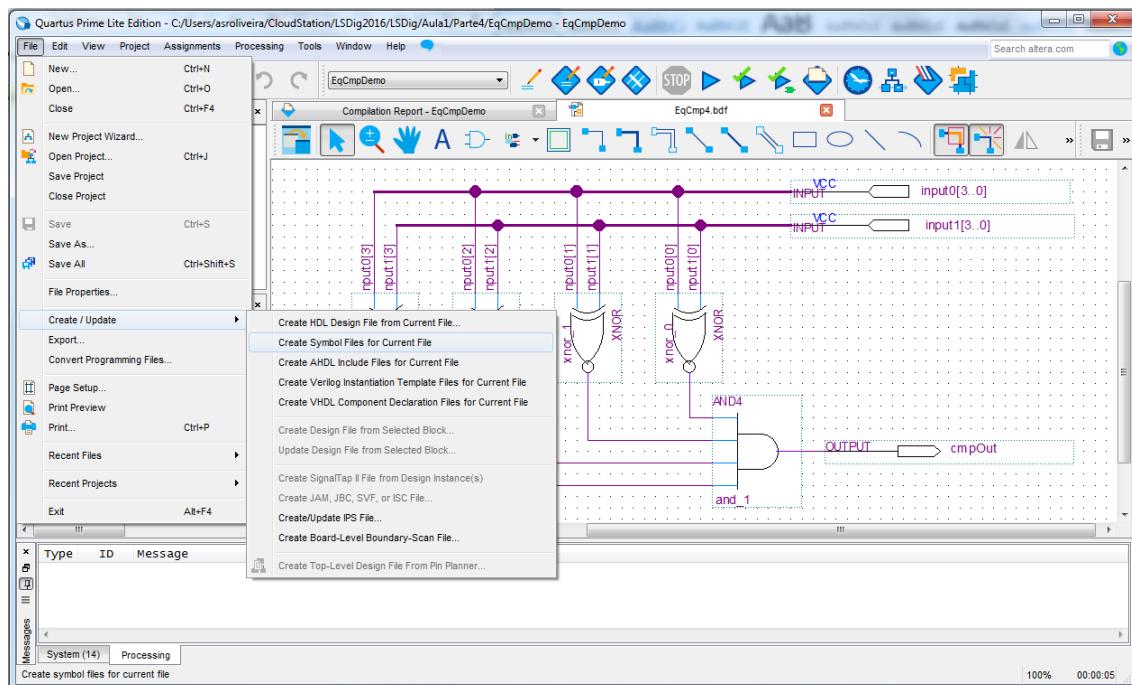


Figura 61 – Criação de um símbolo para o módulo “EqCmp4”.

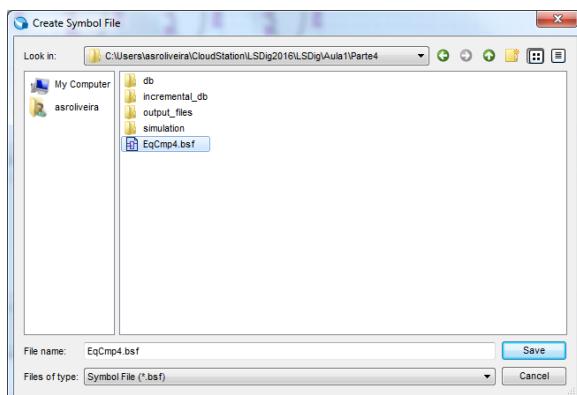


Figura 62 – Gravação do ficheiro “EqCmp4.bsf” relativo ao símbolo do módulo “EqCmp4”.

4. Crie um novo ficheiro para um diagrama esquemático, chamado “EqCmpDemo.bdf”, para instanciar o comparador de igualdade construído no ponto anterior (Figura 63) e efetuar a sua interligação a pinos da FPGA. Ligue a entrada “input0[3..0]” aos interruptores SW[3..0], a entrada “input1[3..0]” aos interruptores SW[7..4] e a saída “cmpOut” a LEDG[0] tal como ilustrado na Figura 64.

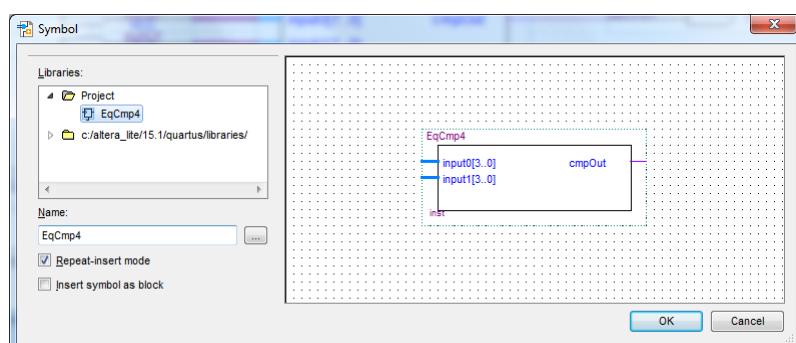


Figura 63 – Instanciação num diagrama esquemático do módulo “EqCmp4”.

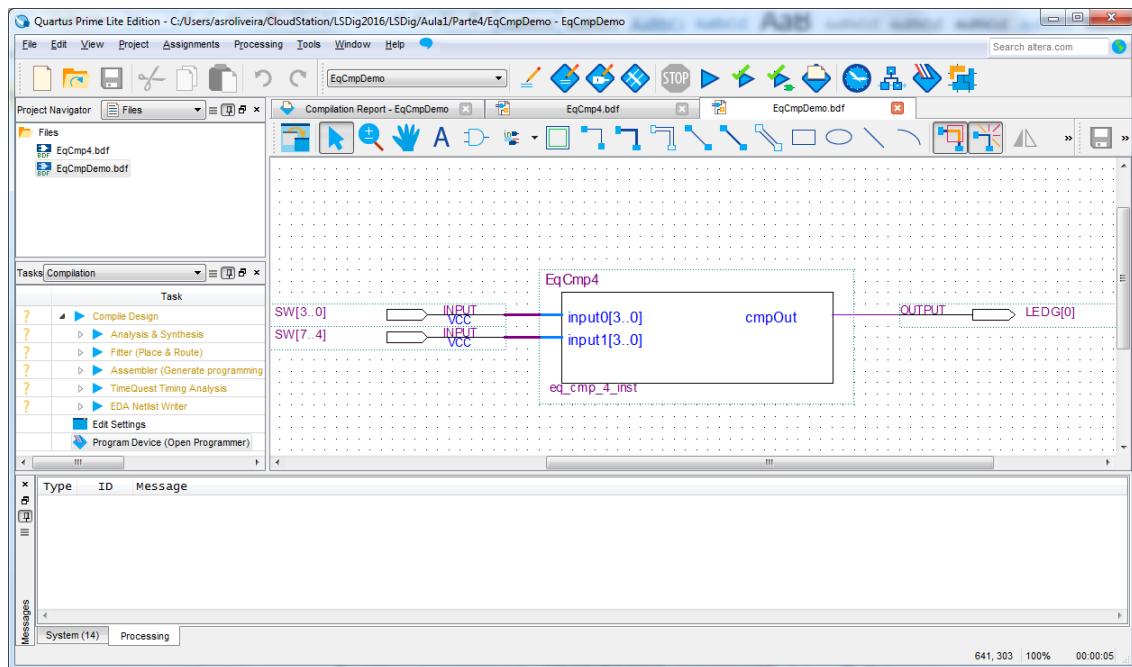


Figura 64 – Interligação do módulo “EqCmp4” e dos portos de entrada e de saída e identificação dos diversos elementos do circuito no módulo “EqCmpDemo”.

5. Importe o ficheiro “master.qsf” com as definições dos pinos da FPGA na placa de desenvolvimento.
6. Efetue a síntese e implementação do projeto através do comando “*Compile Design*”.
7. No final do processo de compilação, programe a FPGA e teste o comparador no *kit* de desenvolvimento aplicando diversos vetores de teste através dos interruptores usados e observando no LED o valor da saída.
8. Feche a aplicação de programação da FPGA.
9. Crie um novo ficheiro VHDL, chamado “EqCmp8.vhd” com o código VHDL da Figura 65, correspondente a um comparador de igualdade de duas entradas de 8 bits.

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity EqCmp8 is
    port(input0 : in std_logic_vector(7 downto 0);
          input1 : in std_logic_vector(7 downto 0);
          cmpOut : out std_logic);
end EqCmp8;

architecture Behavioral of EqCmp8 is
begin
    cmpOut <= '1' when (input0 = input1) else
                  '0';
end Behavioral;

```

Figura 65 – Código fonte do módulo “EqCmp8” (comparador de igualdade de 8 bits).

10. Crie um símbolo para o módulo descrito no ficheiro “EqCmp8.vhd”, tal como ilustrado na Figura 66, de forma a poder instanciá-lo num diagrama esquemático.

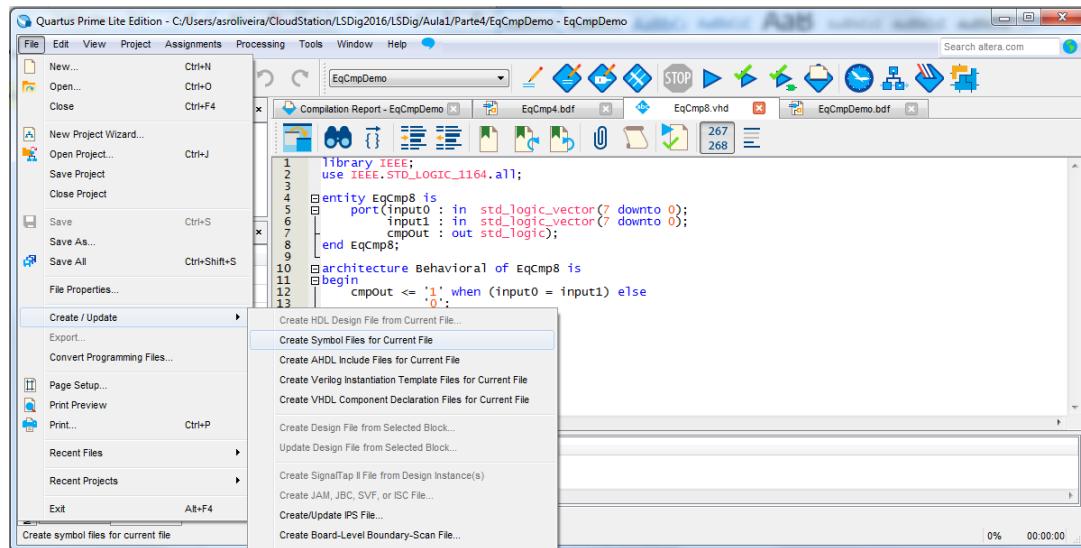


Figura 66 – Criação de um símbolo para o módulo “EqCmp8”.

11. Troque no ficheiro *top-level* “EqCmpDemo” a instanciação do módulo “EqCmp4” pelo módulo “EqCmp8” (Figura 67) e ligue-o tal como ilustrado na Figura 68.

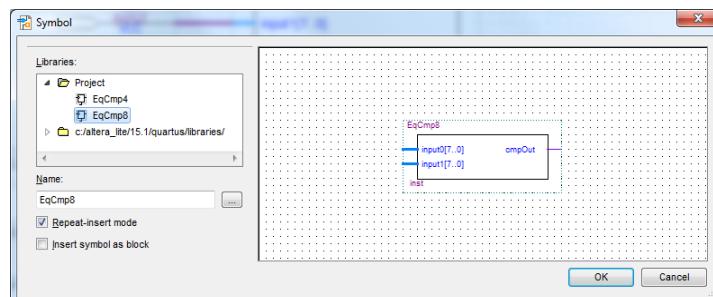


Figura 67 – Instanciação num diagrama esquemático do módulo “EqCmp8”.

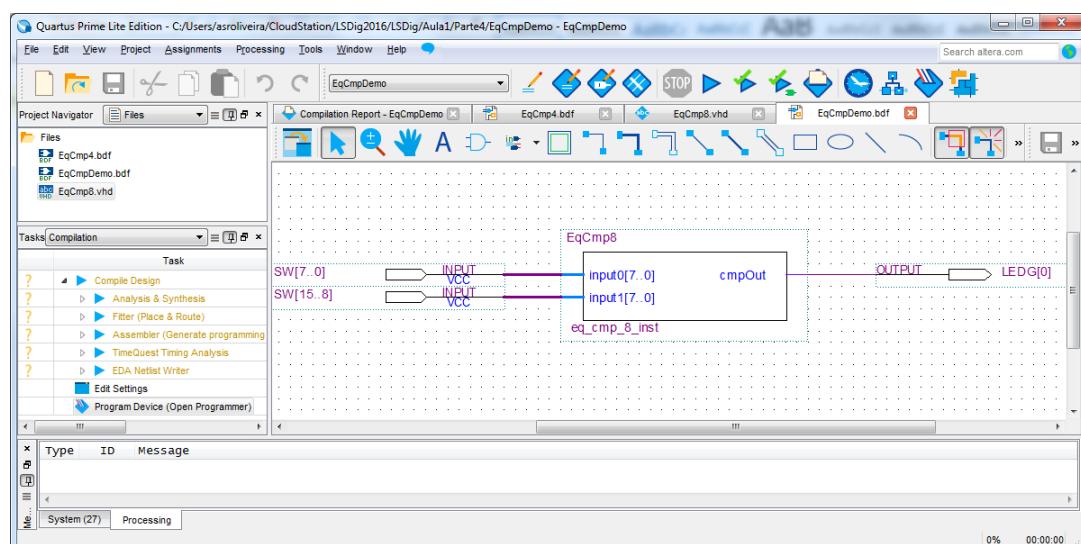


Figura 68 – Interligação do módulo “EqCmp8” e dos portos de entrada e de saída e identificação dos diversos elementos do circuito no módulo “EqCmpDemo”.

12. Volte a importar o ficheiro “master.qsf” com as definições dos pinos da FPGA na placa de desenvolvimento.

13. Efetue a síntese e implementação do projeto através do comando “*Compile Design*”.

14. Programe a FPGA através do comando “*Program Device*”.

15. Teste o comparador no *kit* de desenvolvimento aplicando diversos vetores de teste através dos interruptores usados e observando no LED o valor da saída.

16. Feche a aplicação de programação da FPGA e seguidamente o projeto.

Para refletir

De acordo com a experiência adquirida com a realização deste guião, em que circunstâncias prefere usar VHDL e em que casos prefere a captura de diagramas esquemáticos (esquemas lógicos)?

TPC

1. Acrescente ao projeto da parte II deste guião um ficheiro que permita realizar na forma de um diagrama esquemático a instanciação e interligação da porta lógica AND e do inversor de forma a construir a porta NAND. Compile e teste o projeto resultante.

2. Valide por simulação o módulo “LogicUnit” da parte III deste guião.

3. Valide por simulação os módulos “EqCmp4” e “EqCmp8” da parte IV deste guião.

PDF criado em 11/02/2017 às 02:15:54