

Laboratório de Sistemas Digitais**Trabalho Prático nº 7****Construção e utilização de *testbenches* em VHDL****Simulação comportamental e temporal****Depuração de circuitos em FPGA****Objetivos**

- Construção e utilização de *testbenches* para simulação em VHDL de circuitos combinatórios e sequenciais.
- Simulação comportamental e temporal.
- Depuração de circuitos digitais implementados em FPGA com base em ferramentas de visualização de sinais capturados em tempo-real (analisadores lógicos).

Sumário

Este trabalho prático é dedicado à simulação comportamental, à simulação temporal e à depuração de um sistema em FPGA. Estas tarefas destinam-se a validar o projecto em diferentes etapas de desenvolvimento. A simulação comportamental pode ser usada logo após a modelação do sistema, para avaliar o seu funcionamento, considerando componentes ideais, i.e. sem atrasos, e independentemente da compilação (implementação) para uma FPGA específica. Por outro lado, a simulação temporal pode ser realizada após a compilação (implementação) do sistema para uma FPGA em concreto e já tem em conta estimativas para os atrasos dos componentes do sistema, dados o seu posicionamento e interligação na FPGA, pelo que permite assim uma simulação muito mais próxima do funcionamento do sistema em *hardware* real. Por último, a depuração é realizada com o sistema a funcionar em *hardware*, i.e. estando a FPGA configurada com o sistema-alvo. Utiliza ferramentas que permitem capturar em tempo real os sinais internos, com o objetivo de observar a operação do circuito e a sua resposta a estímulos reais.

Este trabalho prático está dividido em seis partes. A primeira e a segunda são dedicadas à simulação comportamental com base nos exemplos de *testbenches* apresentados nas aulas teórico-práticas (a primeira baseada num decodificador, como exemplo de um componente combinatório; a segunda baseada num contador, como exemplo de um circuito sequencial). Na terceira parte é utilizada uma *testbench* para um componente combinatório (ALU) de forma a realizar primeiro a sua simulação comportamental e depois a temporal.

A quarta parte é dedicada à depuração, apresentando o princípio de funcionamento e as capacidades dos analisadores lógicos integrados disponibilizados pelos fabricantes de FPGAs. Estes analisadores consistem num conjunto de blocos de *hardware* e ferramentas de software que permitem a captura de sinais em tempo real e a sua visualização. A apresentação é baseada no projeto de um contador binário, que permite ilustrar todos os passos necessários para a utilização do analisador lógico.

Na quinta parte pretende-se exercitar os três tipos de tarefas de validação (simulação comportamental, simulação temporal e depuração) ao longo do fluxo de projeto de um sistema baseado num temporizador, de forma a dar uma perspetiva global das tarefas de verificação de um sistema digital ao longo do fluxo de projeto.

Por último, na sexta parte, a realizar como TPC, pretende-se utilizar as ferramentas de depuração de forma didática para capturar e visualizar o *bounce* na comutação de contactos mecânicos.

Parte I

1. Abra a aplicação “*Quartus Prime*” e crie um novo projeto para a FPGA Cyclone IV EP4CE115F29C7. Poderá designar o projeto e a entidade *top-level* como “Dec2_4En”.
2. O código VHDL apresentado na Figura 1 implementa um decodificador binário de 2→4 com entrada de habilitação (*enable*). Crie um novo ficheiro VHDL, introduza esse código e grave o ficheiro com o nome “Dec2_4En.vhd”.
3. Execute o comando “*Analysis & Synthesis*” da aplicação “*Quartus Prime*” para realizar a análise e verificação sintática do projeto.
4. O código VHDL apresentado na Figura 2 implementa uma *testbench* para o decodificador da Figura 1. Crie um novo ficheiro VHDL, introduza esse código e grave o ficheiro com o nome “Dec2_4En_Tb.vhd”.
5. Verifique se a *path* “*ModelSim-Altera*” em “*Tools → Options (EDA Tool Options)*” está de acordo com a Figura 3 (localização no disco da ferramenta de simulação “*ModelSim-Altera*”: “C:\intelFPGA_lite\17.1\modelsim_ase\win32aloem” – em Windows ou tipicamente “/opt/intelFPGA_lite/17.1/modelsim_ase/linuxaloem/” – em Linux).
6. Execute a ferramenta de simulação “*ModelSim Intel FPGA Starter Edition*”, através do menu “*Tools → Run Simulation Tool → RTL Simulation*”.

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

entity Dec2_4En is
    port(enable : in std_logic;
          inputs : in std_logic_vector(1 downto 0);
          outputs : out std_logic_vector(3 downto 0));
end Dec2_4En;

architecture Behavioral of Dec2_4En is
begin
    process(enable, inputs)
    begin
        if (enable = '0') then
            outputs <= "0000";
        else
            if (inputs = "00") then
                outputs <= "0001";
            elsif (inputs = "01") then
                outputs <= "0010";
            elsif (inputs = "10") then
                outputs <= "0100";
            else
                outputs <= "1000";
            end if;
        end if;
    end process;
end Behavioral;
```

Figura 1 – Código VHDL de um decodificador binário 2→4 com *enable* (“Dec2_4En”).

NOTA: No contexto da simulação comportamental, o módulo “Dec2_4En” (ou outro que se pretenda simular e seja a “Unit Under Test – UUT” numa *testbench*) deve ser o *top-level* e compilado previamente no “Quartus Prime”. Por outro lado, o módulo relativo à *testbench* só pode ser compilado na ferramenta de simulação “ModelSim Intel FPGA Starter Edition”, atuando como *top-level* apenas no simulador.

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

-- Entidade sem portos
entity Dec2_4En_Tb is
end Dec2_4En_Tb;

architecture Stimulus of Dec2_4En_Tb is

    -- Sinais para ligar às entradas da UUT
    signal s_enable : std_logic;
    signal s_inputs : std_logic_vector(1 downto 0);

    -- Sinal para ligar às saídas da UUT
    signal s_outputs : std_logic_vector(3 downto 0);

begin
    -- Instanciação da Unit Under Test (UUT)
    uut: entity work.Dec2_4En(Behavioral)
        port map(enable => s_enable,
                  inputs => s_inputs,
                  outputs => s_outputs);

    --Process stim
    stim_proc : process
    begin
        wait for 100 ns;
        s_enable <= '0';
        wait for 100 ns;
        s_enable <= '1';
        wait for 100 ns;
        s_inputs <= "00";
        wait for 100 ns;
        s_inputs <= "10";
        wait for 100 ns;
        s_inputs <= "01";
        wait for 100 ns;
        s_inputs <= "11";
        wait for 100 ns;
    end process;
end Stimulus;
```

Figura 2 – Código VHDL de uma *testbench* (“Dec2_4En_Tb”) para o decodificador binário 2→4 com *enable* (“Dec2_4En”).

7. Quando executada, a ferramenta de simulação “ModelSim Intel FPGA Starter Edition” deve apresentar o aspeto ilustrado na Figura 4. Caso a janela “Wave” (destinada ao desenho das formas de onda) não apareça, poderá ativá-la através do menu “View → Wave”.

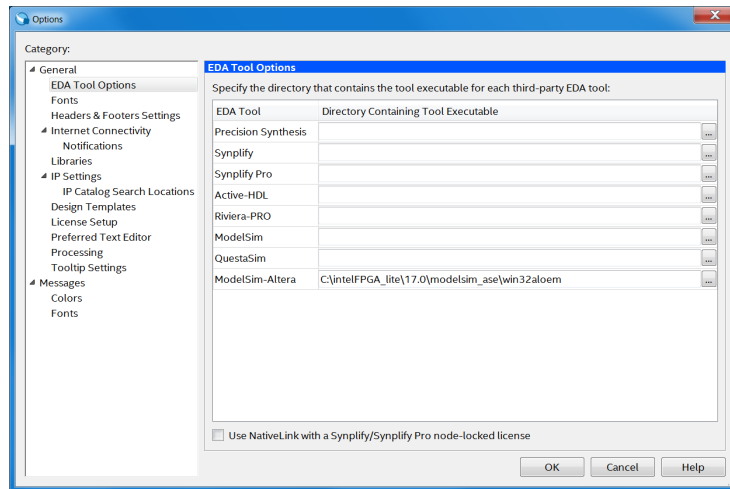


Figura 3 – Path correto para a ferramenta de simulação (em Windows).

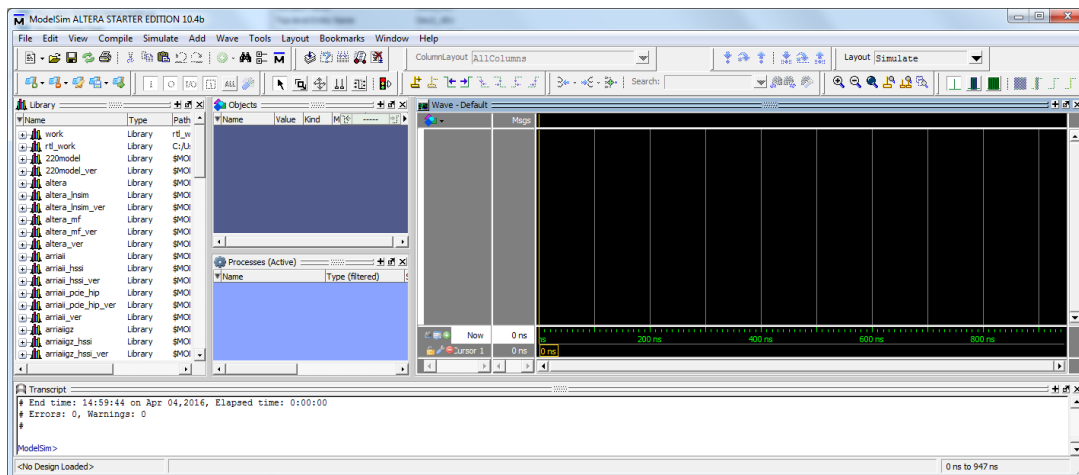


Figura 4 – Aspetto inicial da janela da ferramenta de simulação “ModelSim Intel FPGA Starter Edition”.

8. Compile a *testbench* através do menu “Compile → Compile”, selecionando o ficheiro “Dec2_4En_Tb.vhd” e premindo “Compile” (Figura 5).

9. Inicie a simulação através de um duplo clique na arquitetura “stimulus” da entidade “dec2_4en_tb” apresentada na biblioteca “work” (projeto atual) na janela “Library” da aplicação “ModelSim Intel FPGA Starter Edition” (Figura 6).

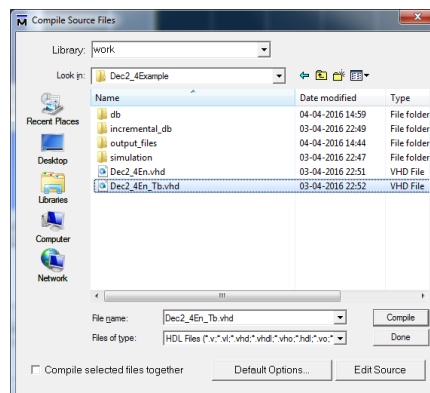


Figura 5 – Compilação da *testbench* “Dec2_4En_Tb.vhd”.

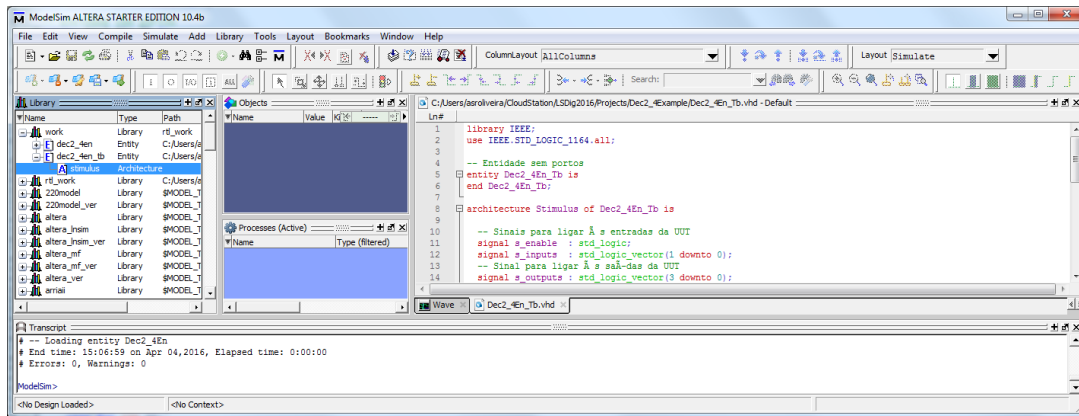


Figura 6 – Início da simulação da entidade **Dec2_4En_Tb** e arquitetura **Stimulus**.

10. Adicione à janela “Wave” (por *drag-and-drop* da janela “Objects”) os sinais da *testbench* ligados aos portos da entidade a simular (**s_enable**, **s_inputs** e **s_outputs**) conforme ilustrado na Figura 7.

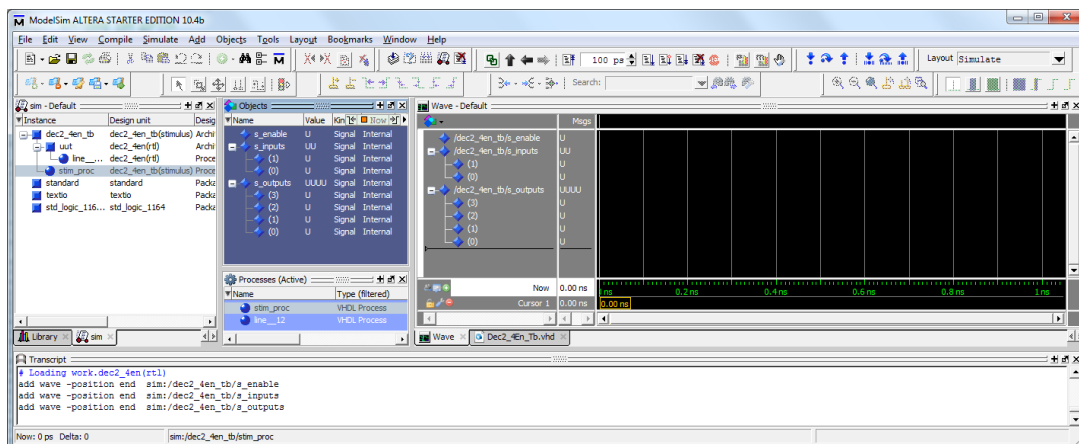


Figura 7 – Sinais **s_enable**, **s_inputs** e **s_outputs** adicionados à janela “Wave”.

11. Execute a simulação, especificando previamente o tempo pretendido (e.g. 1000 ns tal como ilustrado na Figura 8) e premindo “Run”. A janela “Wave” deve ser atualizada à medida que a simulação progride. Analise os resultados e no final feche a aplicação “*ModelSim Intel FPGA Starter Edition*” e de seguida o projeto.

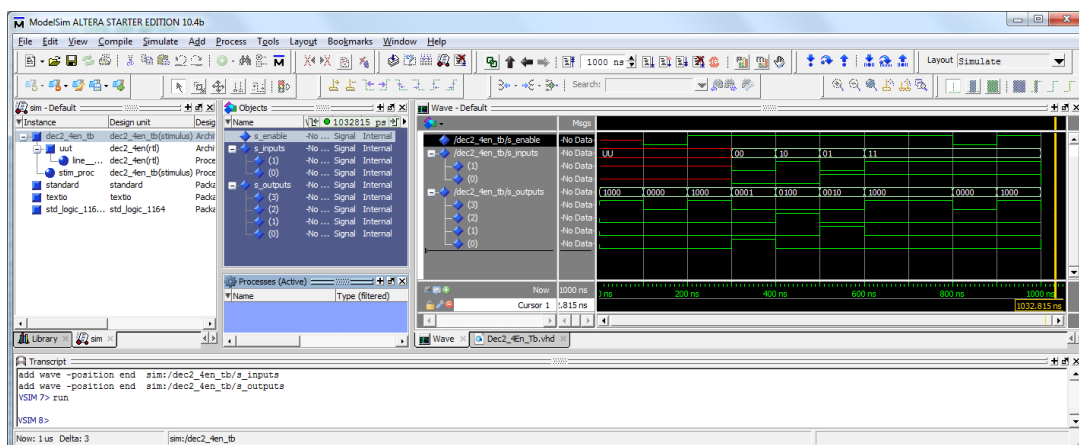


Figura 8 – Resultados de simulação do decodificador binário de 2→4 com *enable* “Dec2_4En.vhd” com a *testbench* “Dec2_4En_Tb.vhd”.

Parte II

Repita os passos da parte I deste guião para a simulação comportamental (com uma *testbench* em VHDL) do contador binário *up/down* de 4 bits apresentado nos slides da aula teórico-prática 7, como um exemplo de simulação de um componente sequencial.

Parte III

A simulação comportamental, realizada nas partes I e II deste guião, não toma em consideração os atrasos dos elementos lógicos da FPGA nem os atrasos devidos a recursos de encaminhamento; assume que todo o processamento é realizado em tempo zero, ou seja, que os elementos lógicos e recursos de interligação da FPGA são ideais.

Já a simulação temporal permite não só testar a função dum circuito mas também observar o cumprimento dos requisitos temporais, desde que sejam usados vetores de simulação adequados (que “exercitem” o caminho crítico). Para tal, utiliza o resultado da compilação, que incorpora o mapeamento da *netlist* em primitivas da FPGA e o seu posicionamento em localizações específicas, bem como o encaminhamento e interconexões entre as primitivas, o que permite obter estimativas bastante realistas dos atrasos envolvidos e assim modelar de forma mais precisa o comportamento do sistema em *hardware* real. Nesta parte, propõe-se a simulação temporal (com uma *testbench* em VHDL) de uma ALU de 16 *bits* semelhante à implementada (com 4 *bits*) na parte II do trabalho prático 3.

1. Abra a aplicação “*Quartus Prime*” e crie um novo projeto para a FPGA Cyclone IV EP4CE115F29C7. Poderá designar o projeto e a entidade *top-level* como “ALU16”.
2. O código VHDL apresentado na Figura 9 implementa uma ALU de 16 bits. Crie um novo ficheiro VHDL, introduza esse código e grave o ficheiro com o nome “ALU16.vhd”.
3. Execute o comando “*Analysis & Synthesis*” da aplicação “*Quartus Prime*” para realizar a análise e verificação sintática do projeto.
4. O código VHDL apresentado na Figura 10 implementa uma *testbench* para a ALU da Figura 9. Crie um novo ficheiro VHDL, introduza esse código e grave o ficheiro com o nome “ALU16_Tb.vhd”.
5. Realize a simulação comportamental da ALU com a *testbench* fornecida, executando a ferramenta “*ModelSim Intel FPGA Starter Edition*”, através do menu “*Tools → Run Simulation Tool → RTL Simulation*”, tal como na parte I deste guião. Apesar da *testbench* “ALU16_Tb.vhd” compilar com sucesso, a simulação não arranca devido a um erro. Sugestão: compile também o módulo “ALU16.vhd” na aplicação “*ModelSim Intel FPGA Starter Edition*” para ter mais informação sobre a causa do problema que não se manifestou anteriormente. **NOTA:** os módulos VHDL têm de ser compilados após terem sido alterados e antes de iniciar a simulação.
6. Analise os resultados da simulação que deverão ser semelhantes aos apresentados na Figura 11. Observe que a ALU não apresenta nesta simulação comportamental atrasos de propagação entre a alteração dos operandos de entrada e o resultado de saída. Para visualizar os valores dos sinais em hexadecimal, escolha a opção “*Radix → Hexadecimal*” do menu acessível com o botão direito do rato quando clica sobre o(s) sinal(is) pretendido(s) na janela “*Wave*”. No final feche a aplicação de simulação “*ModelSim Intel FPGA Starter Edition*”.

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

entity ALU16 is
    port(op : in std_logic_vector(2 downto 0);
          op0 : in std_logic_vector(15 downto 0);
          op1 : in std_logic_vector(15 downto 0);
          res : out std_logic_vector(15 downto 0);
          mHi : out std_logic_vector(15 downto 0));
end ALU16;

architecture Behavioral of ALU16 is

    signal s_mRes : std_logic_vector(31 downto 0);

begin

    s_mRes <= std_logic_vector(unsigned(op0) * unsigned(op1));

    process(op, op0, op1, s_mRes)
    begin
        case op is
            when "000" =>
                res <= std_logic_vector(unsigned(op0) + unsigned(op1));

            when "001" =>
                res <= std_logic_vector(unsigned(op0) - unsigned(op1));

            when "010" =>
                res <= s_mRes(15 downto 0);

            when "011" =>
                res <= std_logic_vector(unsigned(op0) / unsigned(op1));

            when "100" =>
                res <= std_logic_vector(unsigned(op0) rem unsigned(op1));

            when "101" =>
                res <= op0 and op1;

            when "110" =>
                res <= op0 or op1;

            when "111" =>
                res <= op0 xor op1;
            end case;
        end process;

        mHi <= s_mRes(31 downto 16) when (op = "010") else
            (others => '0');

    end Behavioral;
```

Figura 9 – Código VHDL de uma ALU de 16 bits ("ALU16").

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity ALU16_Tb is
end ALU16_Tb;

architecture Stimulus of ALU16_Tb is
    -- Sinais para ligar às entradas da UUT
    signal s_op   : std_logic_vector(2 downto 0);
    signal s_op0  : std_logic_vector(15 downto 0);
    signal s_op1  : std_logic_vector(15 downto 0);

    -- Sinais para ligar às saídas da UUT
    signal s_res  : std_logic_vector(15 downto 0);
    signal s_mHi  : std_logic_vector(15 downto 0);
begin
    uut: entity work.ALU16(Behavioral)
        port map(op   => s_op,
                 op0  => s_op0,
                 op1  => s_op1,
                 res  => s_res,
                 mHi  => s_mHi);

    --Process stim
    stim_proc : process
    begin
        s_op0 <= x"FEDC";
        s_op1 <= x"0123";
        s_op  <= "000"; -- +
        wait for 100 ns;
        s_op  <= "001"; -- -
        wait for 100 ns;
        s_op  <= "010"; -- *
        s_op1 <= x"89AB";
        wait for 100 ns;
        s_op  <= "011"; -- /
        s_op1 <= x"4567";
        wait for 100 ns;
        s_op  <= "100"; -- rem
        wait for 100 ns;
        s_op0 <= x"F30C";
        s_op1 <= x"F50A";
        s_op  <= "101"; -- and
        wait for 100 ns;
        s_op  <= "110"; -- or
        s_op1 <= x"0FA5";
        wait for 100 ns;
        s_op  <= "111"; -- xor
        wait for 100 ns;

        wait;

    end process;
end Stimulus;

```

Figura 10 – Código VHDL de uma *testbench* ("ALU16_Tb") para a ALU de 16 bits ("ALU16").

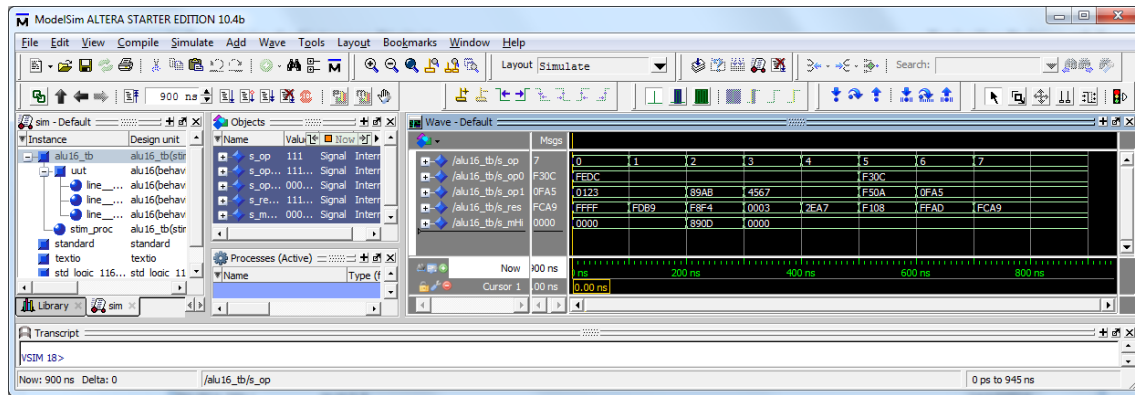


Figura 11 – Resultados da simulação comportamental da ALU de 16 bits com a *testbench* “ALU16_Tb”.

7. Para configurar as ferramentas de efetuar a simulação temporal da ALU, realize a seguinte sequência de passos:

- Especifique as opções de simulação através do menu “*Assignments → Settings...*” (que deverá abrir a janela da esquerda da Figura 12).
- Selecione a categoria “*EDA Tool Settings (Simulation)*”.
- Nas opções “*NativeLink Settings*” ative “*Compile test bench:*” e clique no botão “*Test Benches...*” (Figura 12).
- Na janela seguinte clique no botão “*New...*” e preencha todos os campos conforme ilustrado na Figura 13 (não se esquecendo de adicionar o ficheiro “*ALU16_Tb.vhd*” à lista de “*Test bench and simulation files*”).
- Voltando à janela anterior (Figura 12, janela da esquerda) selecione “*More EDA Netlist Writer Settings*”. Aparecem várias opções. Escolha “off” em “*Generate functional simulation netlist*”.
- Após configuração, as três janelas da Figura 12 devem apresentar o aspeto das Figuras 13 a 15.

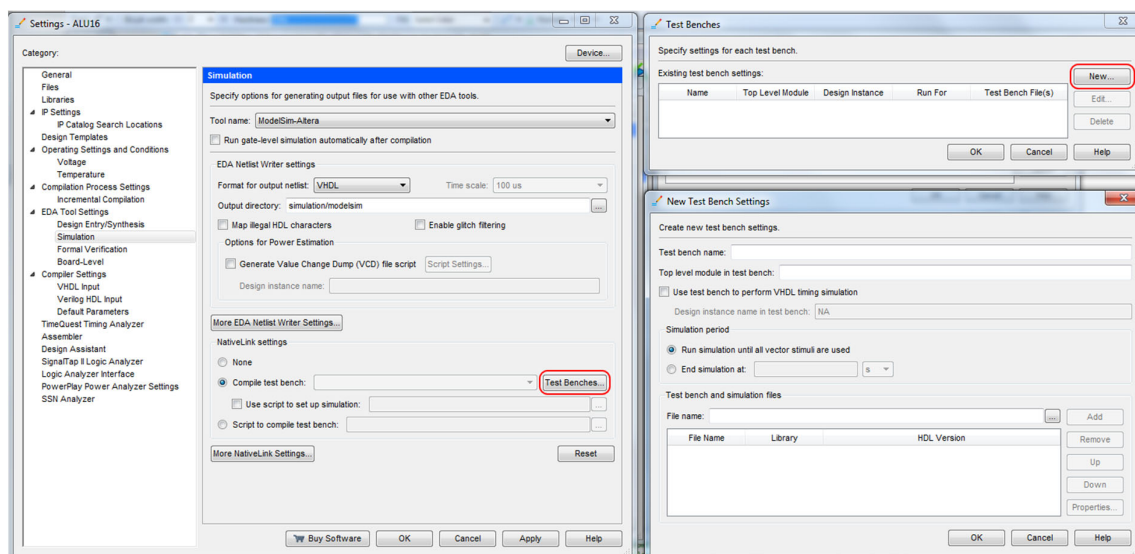


Figura 12 – Passos de configuração da *testbench* a usar na simulação temporal da ALU de 16 bits.

8. Execute a opção “*Compile Design*”. Para além de gerar o ficheiro de configuração da FPGA e os relatórios adequados, serão também criados o ficheiro “ALU16.vho” que contém a *netlist* sintetizada e o ficheiro “ALU16_vhd.sdo” com a informação temporal. Estes ficheiros encontram-se no subdiretório “simulation\modelsim” do projeto. Abra cada um destes

ficheiros e observe, no ficheiro “ALU16.vho”, que a arquitetura criada com o modelo da ALU pós-implementação possui o nome “**structure**”.

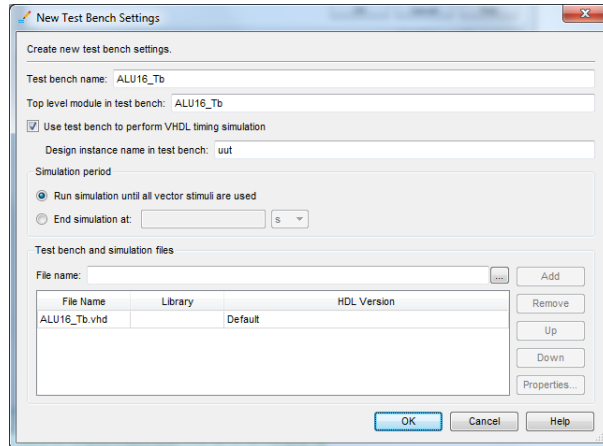


Figura 13 – Aspeto da janela “New Test Bench Settings” após configuração das opções.

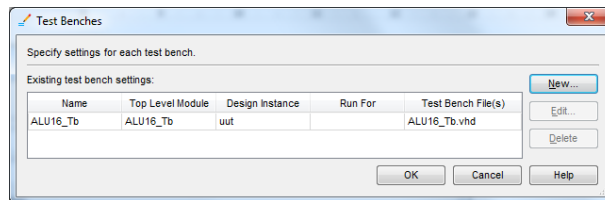


Figura 14 – Aspeto da janela “Test Benches” após configuração das opções.

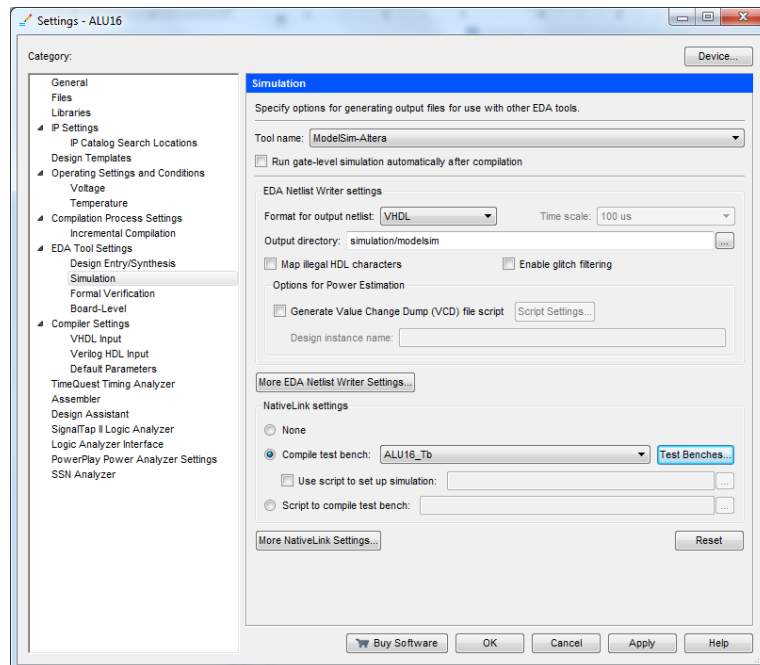


Figura 15 – Aspeto da janela “Settings” após configuração das opções.

9. Altere nome da arquitetura instanciada no módulo “ALU16_Tb” da *testbench* para “structure” (na linha “*uut: “entity work.ALU16(BehavioralStructure)”*”).

10. Execute a ferramenta de simulação “ModelSim Intel FPGA Starter Edition”, através do menu “Tools → Run Simulation Tool → Gate Level Simulation...”. Selecione o modelo temporal da simulação “Slow -7 1.2V 85 Model” e clique em “Run” (Figura 16).

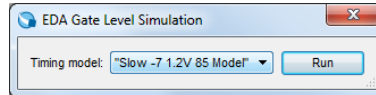


Figura 16 – Especificação do modelo temporal da simulação (*speed grade* da FPGA, tensão de alimentação e temperatura do dispositivo).

11. A simulação arrancará automaticamente, através de uma *script* gerada pelo “Quartus Prime” e de acordo com a configuração realizada no ponto 7. Analise os resultados da simulação (que deverão ser semelhantes aos apresentados na Figura 17) e observe os atrasos de propagação entre a modificação dos operandos de entrada e a atualização do resultado à saída da ALU. Faça “Zoom In” das forma de onda na janela “Wave” de forma a observar as múltiplas transições espúrias (*glitches*) que ocorrem sempre que o resultado da ALU é atualizado. A que se devem estes *glitches*?

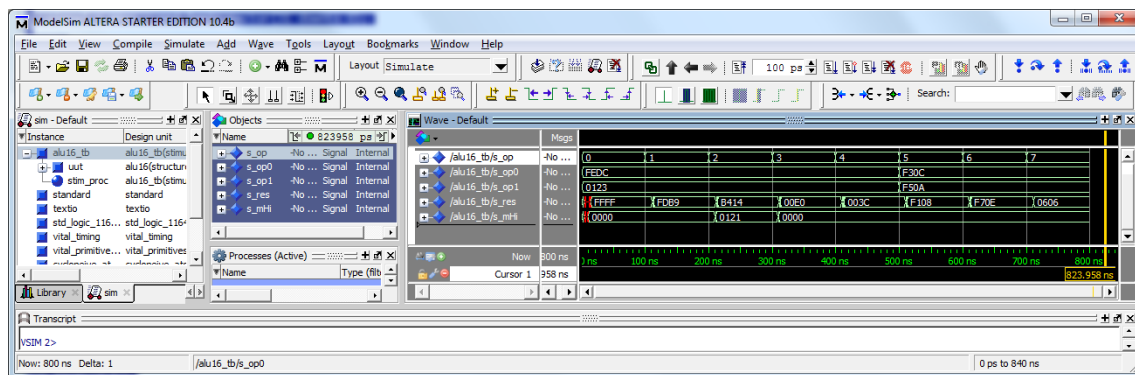


Figura 17 – Resultados da simulação temporal da ALU de 16 bits com a *testbench* “ALU16_Tb”.

Parte IV

1. Crie um novo projeto para a FPGA Cyclone IV EP4CE115F29C7. Poderá designar o projeto e a entidade *top-level* como “DebugDemo”. O projeto consistirá num contador binário crescente, apresentado na Figura 18 e descrito no ficheiro “CntUp.vhd”. O módulo *top-level* “DebugDemo.vhd” interliga os portos do contador a dispositivos do kit DE2-115 (Figura 19). Edite os ficheiros com o código fonte fornecido, grave-os com os nomes indicados, importe o ficheiro “master.qsf”, compile o projeto, programe a FPGA e responda às questões dos pontos seguintes.

- Qual a frequência de incremento (atualização) do contador?
- Que bits de saída do contador não estão ligados aos LEDs?
- Qual a frequência do bit de saída menos significativo do contador?
- Qual a frequência do bit de saída mais significativo do contador?

- e. Qual a frequência do bit de saída menos significativo do contador visível nos LEDs?
- f. Quanto tempo demora um ciclo completo de contagem (entre 2 passagens por zero)?

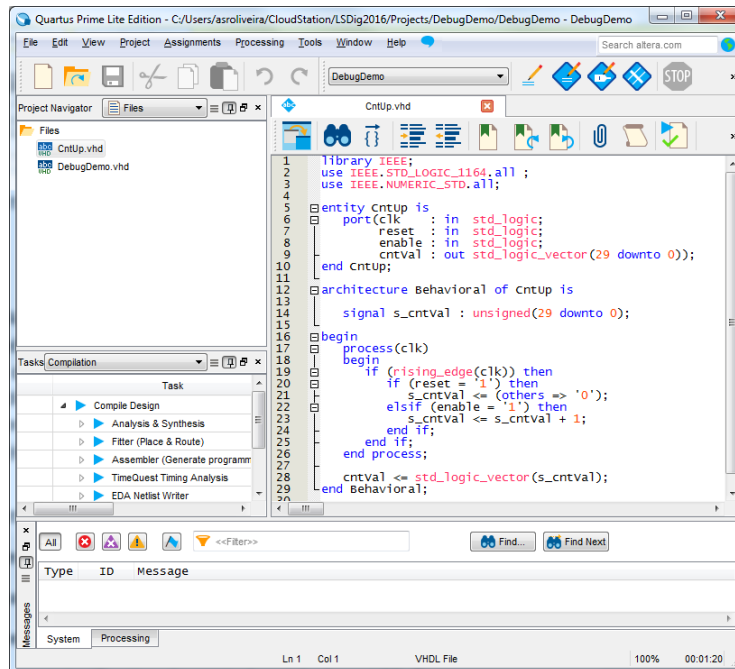


Figura 18 – Código fonte do módulo “CntUp.vhd”.

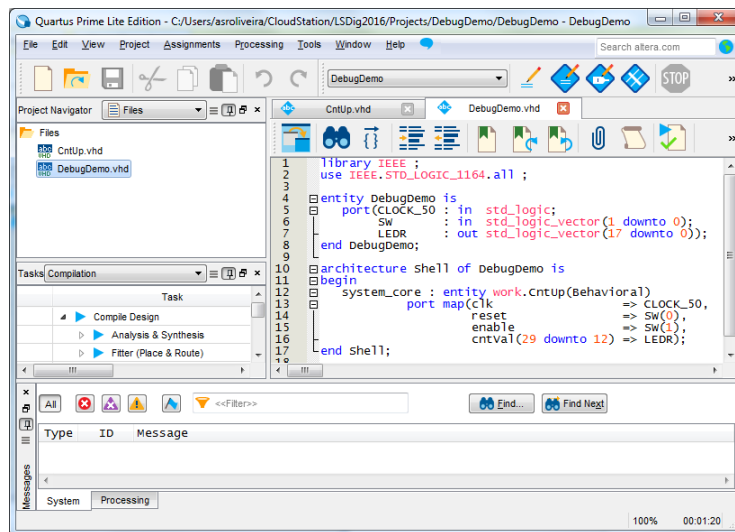


Figura 19 – Código fonte do módulo “DebugDemo.vhd”.

2. Devido à elevada frequência de comutação, a maioria dos bits do contador não podem ser devidamente observados a olho nu em dispositivos simples como os LEDs. Para resolver este problema e avaliar o funcionamento correto de todos os bits do contador vamos recorrer a uma ferramenta, genericamente designada por “Analisador Lógico Integrado”, tipicamente disponibilizada pelos fabricantes das FPGA. Estas ferramentas permitem especificar os sinais do sistema que pretendemos visualizar, adicionar de forma transparente e automática a lógica necessária para a sua captura, armazenamento e transferência para um computador de desenvolvimento onde será realizada a sua visualização. A lógica adicionada para este efeito utiliza os próprios recursos lógicos programáveis da FPGA. A interface para transferir os sinais capturados é também a usada na programação da FPGA (denominada JTAG e acessível através

do porto “USB Blaster”) o que é bastante conveniente. Para usar esta facilidade, o primeiro passo é a criação de um ficheiro no “Quartus Prime” do tipo “SignalTap II Logic Analyser File” (Figura 20).

NOTA: Para poder utilizar a ferramenta “SignalTap II Logic Analyser” disponibilizada pela Intel FPGA, deverá ativar a opção “Enable sending TalkBack data to Intel FPGA” disponível em “Tools → Options (Internet Connectivity → TalkBack Options)”.

3. Após a criação do ficheiro do tipo *SignalTap II Logic Analyser File* é apresentada a aplicação da Figura 21, onde devem ser especificados os sinais que se pretende capturar e visualizar, o sinal de *clock* usado para estabelecer a frequência de amostragem dos sinais, o número de amostras capturadas e as condições que disparam a amostragem. A configuração destes parâmetros vai ser descrita nos próximos pontos.

4. O primeiro passo é a adição dos nodos (sinais e portos) do sistema que se pretende capturar. Para tal, deve ser seleccionada a opção “Add Nodes...” do menu acessível com o botão direito do rato na área “Setup” mostrada na Figura 21. Os portos a seleccionar para captura e visualização no analisador lógico são os apresentados na Figura 22. Selecione a opção “Design Entry (all names)” no parâmetro “Filter”.

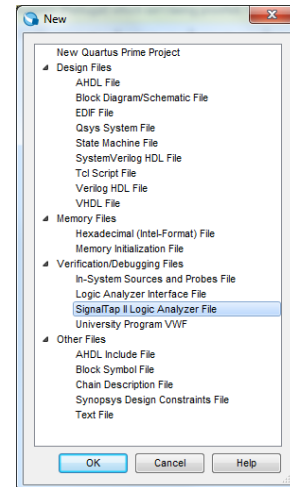


Figura 20 – Criação de um ficheiro do tipo “SignalTap II Logic Analyser File”.

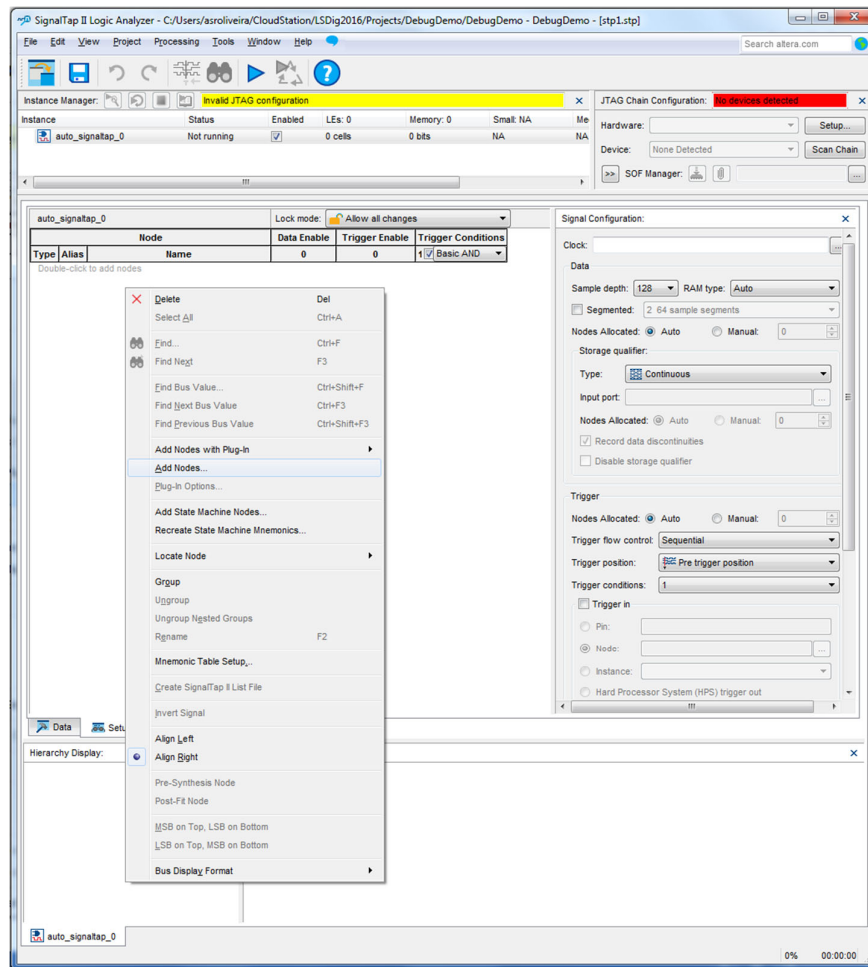


Figura 21 – Aspeto inicial da aplicação “SignalTap II Logic Analyser”.

5. Após premir *Insert* e fechar a janela de “Node Finder” deverá configurar na janela principal os seguintes parâmetros (Figura 23):

- **Hardware:** USB-Blaster (dispositivo/interface usado para programação e depuração)
- **Clock:** CLOCK_50 (sinal de relógio usado para estabelecer os instantes de amostragem / frequência de amostragem – 50 MHz)
- **Sample Depth:** 4K (número de amostras consecutivas a capturar)
- **Trigger Conditions** (condições que levam ao disparo da captura de amostras; neste caso $reset = 0$ e $enable = 1$ para capturar o instante em que é libertado o *reset*)
 - CntUp:system_core | reset = 0
 - CntUp:system_core | enable = 1

6. Seguidamente, grave o ficheiro com o nome “DebugDemo.stp”.

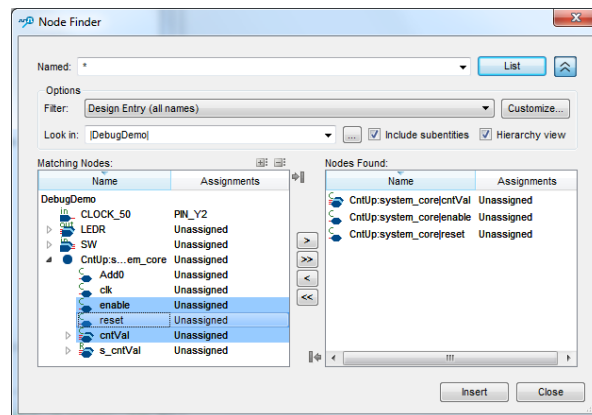


Figura 22 – Portos do contador selecionados (“reset”, “enable” e “cntVal”).

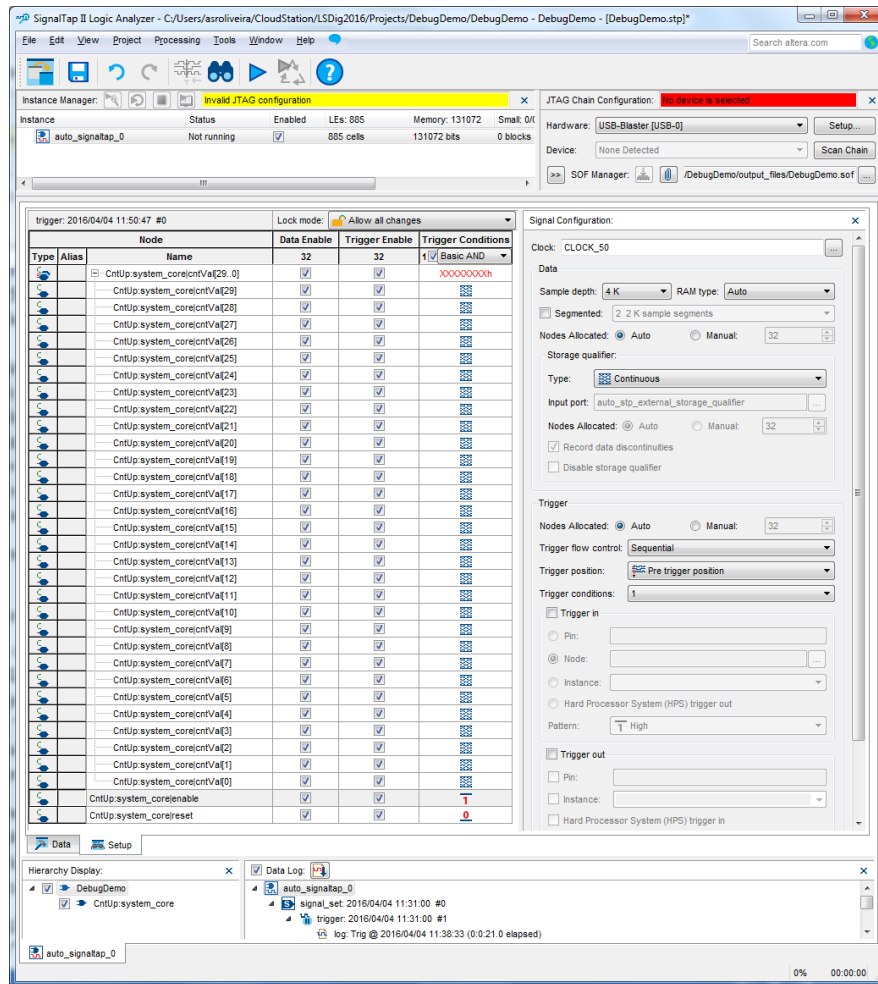



Figura 23 – Aspetto da aplicação “SignalTap II Logic Analyser” com os parâmetros configurados.

7. Uma vez que as componentes de hardware do analisador lógico usam recursos lógicos programáveis da FPGA, antes de efetuar a captura e visualização dos sinais, é necessário voltar a compilar todo o projeto no “Quartus Prime”. O projeto consiste nos ficheiros “DebugDemo.vhd”, “CntUp.vhd” e “DebugDemo.stp”. O ficheiro *top-level* deve continuar a ser o “DebugDemo.vhd”. Após compilação, execute a aplicação “SignalTap II Logic Analyser” (menu “Tools → SignalTap II Logic Analyser”).

8. Especifique o ficheiro SOF a usar para configurar a FPGA (“output_files/DebugDemo.sof”) e programe a FPGA (botão  no canto superior direito da Figura 23). O ficheiro SOF inclui quer a configuração da lógica do sistema desenvolvido, quer a configuração da lógica correspondente às componentes de hardware do analisador lógico.

9. Após a programação da FPGA (Figura 24) mude a janela principal da aplicação “SignalTap II Analyser” de “Setup” para “Data” (Figura 25). Neste ponto está tudo configurado e preparado para ser iniciada a captura de amostras dos sinais pretendidos. Para tal, basta selecionar o comando “Processing → Run Analysis” e de seguida desativar a entrada de “reset” e ativar a entrada de “enable” do contador de forma a que seja satisfeita a condição de *trigger* e iniciada a captura dos sinais pretendidos. Uma vez recolhidas 4K amostras de cada sinal, os respetivos valores são transferidos para o computador e visualizados na aplicação “SignalTap II Analyser” (Figura 26).

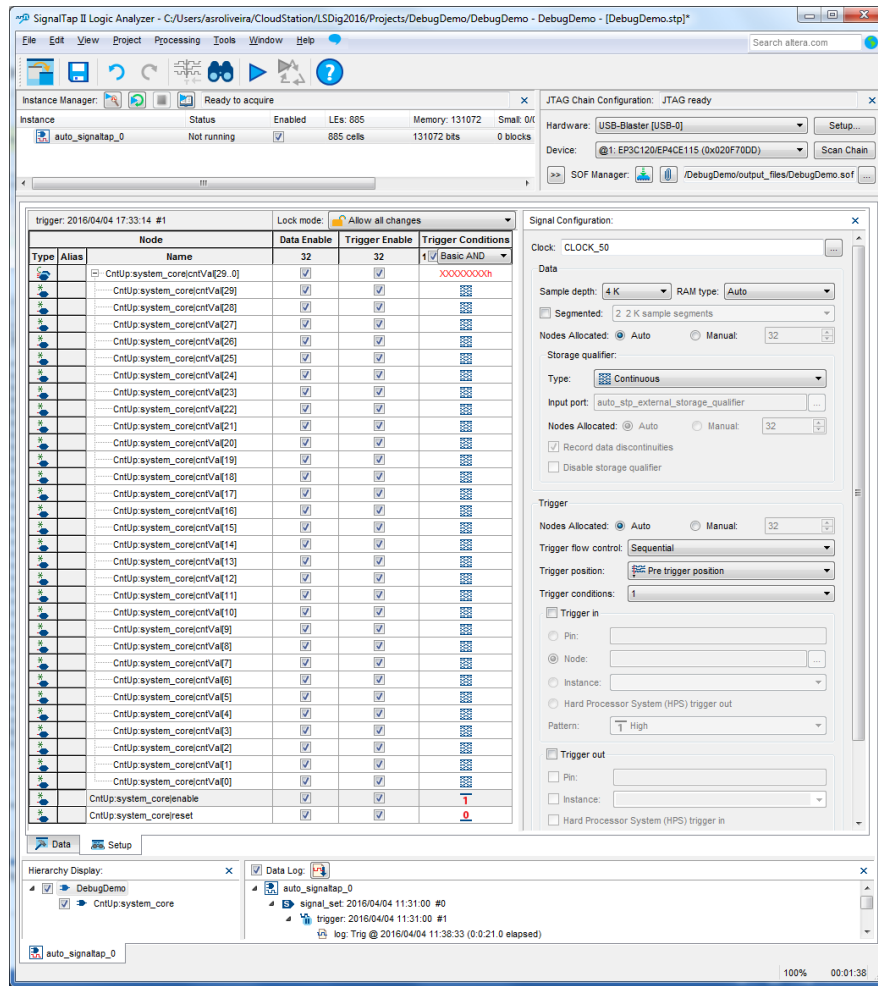


Figura 24 – Aspeto da aplicação “SignalTap II Logic Analyzer” após programação da FPGA.

10. Podem ser visualizados instantes particulares da contagem se forem definidas condições de *trigger* mais restritas. A título de exemplo, pode ser visualizado o instante de *wrap-around* do contador através da atribuição do valor “11...100” a *cntVal*, o que leva a que a captura seja disparada três ciclos de relógio antes do contador voltar à contagem “00...00” (Figura 27). Volte a executar o comando “Processing → Run Analysis” para efetuar a captura (Figura 28).

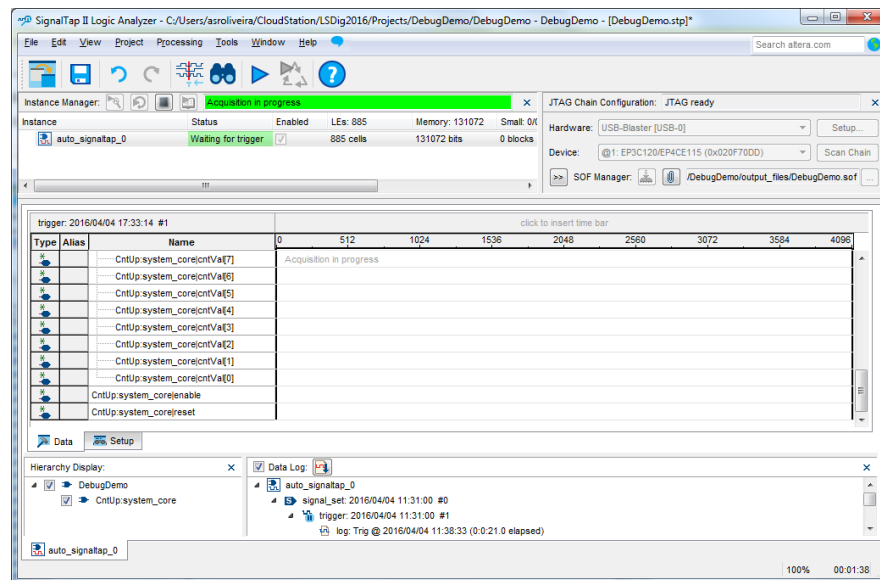


Figura 25 – Janela “Data” da aplicação “SignalTap II Logic Analyser” antes da captura (aguardar verificação da condição de disparo da captura).

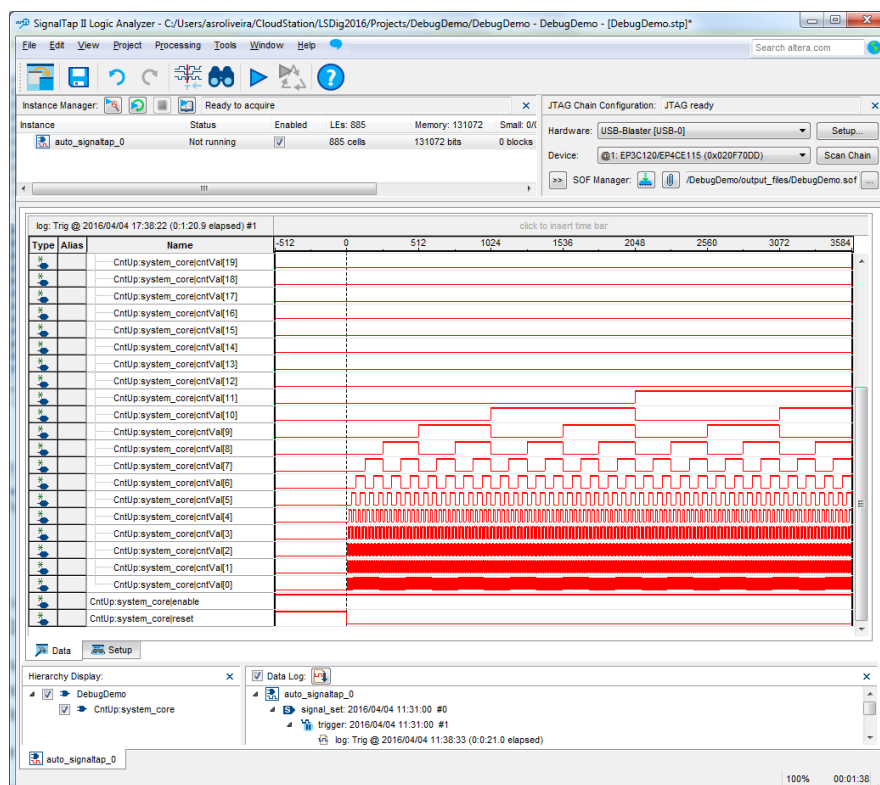


Figura 26 – Janela “Data” da aplicação “SignalTap II Logic Analyser” depois da captura.

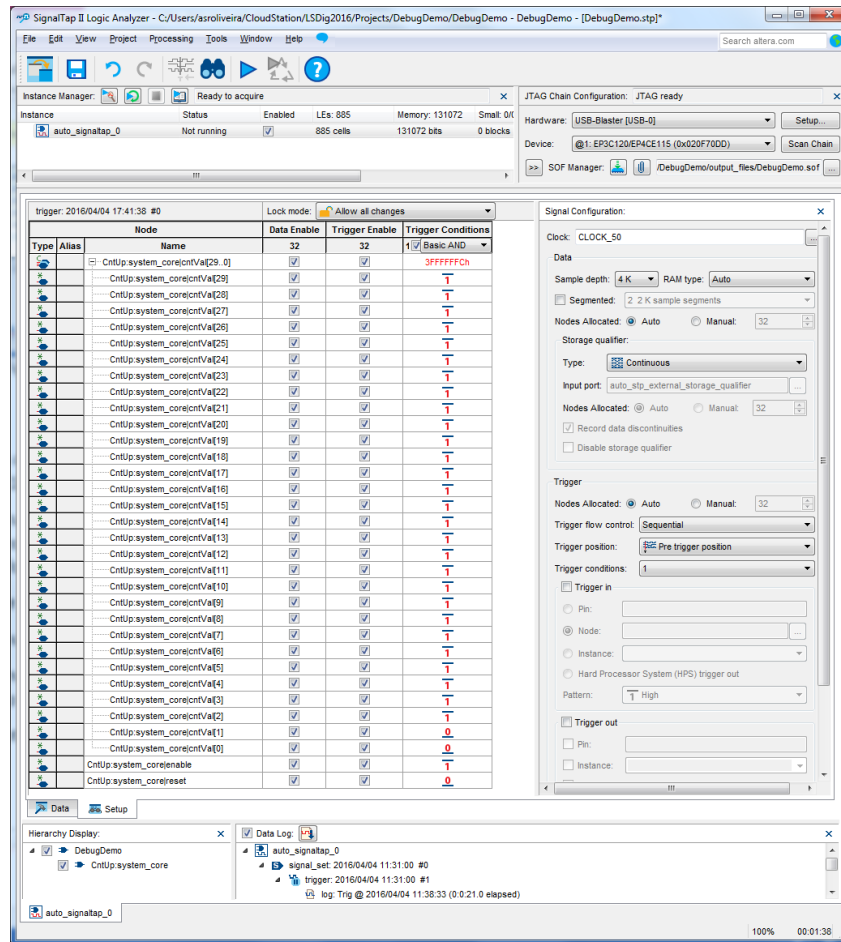


Figura 27 – Configuração do instante de disparo da captura (*trigger*) com base no valor dos sinais *reset*, *enable* e *cntVal*.

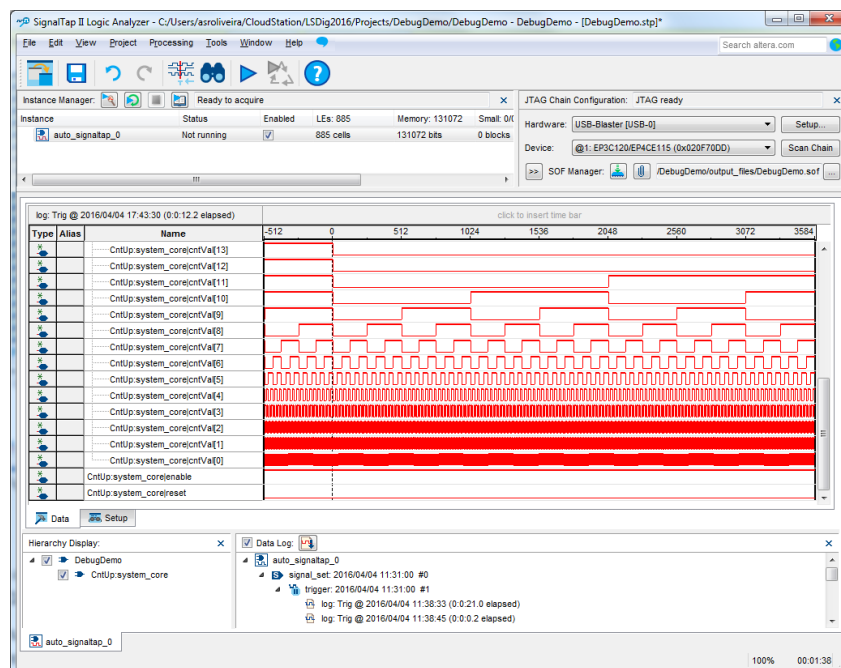


Figura 28 – Janela “Data” da aplicação “SignalTap II Logic Analyser” depois de uma nova aquisição, onde foi capturado o *wrap-around* do contador.

Parte V

1. Abra a aplicação “*Quartus Prime*” e crie um novo projeto para a FPGA Cyclone IV EP4CE115F29C7. Designe o projeto e a entidade *top-level* como “TimerTest”.
2. Construa um módulo relativo a um temporizador (parametrizável) de atraso à desoperação. Na sua instanciação utilize um sinal de relógio de 50 MHz e configure-o de modo a que, cada vez que for disparado, forneça um impulso na saída com a duração de 200 ns.
3. Realize todos os passos necessários para efetuar:
 - A simulação comportamental do sistema com uma *testbench* adequada.
 - A simulação temporal do sistema, após a sua compilação, com a mesma *testbench* do ponto anterior.
 - A captura dos sinais em tempo-real e depuração do circuito implementado em FPGA.

[TPC] Repita os pontos anteriores para um temporizador (parametrizável) de atraso à operação.

Parte VI

[TPC] Desenvolva e teste um projeto no “*Quartus Prime*”, que utilize a aplicação “*SignalTap II Logic Analyser*”, para capturar e visualizar o *bounce* na comutação de contactos mecânicos dos interruptores e botões de pressão do *kit* DE2-115.

PDF criado em 12/04/2023 às 16:56:30