

TRABALHO DE LABORATÓRIO III

CIRCUITOS SEQUENCIAIS: ENIGMA (VIVADO TUTORIAL)

VERSÃO 1.3

1. INTRODUÇÃO

Pretende-se com este trabalho que os alunos se familiarizem com a utilização de ferramentas de simulação e prototipagem de circuitos digitais. Para o efeito, será realizada uma apresentação tutorial da ferramenta *Vivado WebPack* da *Xilinx* (versão 2016.3), que será utilizada para simular um circuito constituído por elementos básicos de memória (flip-flops) agregados a circuitos aritméticos.

Este trabalho é considerado para **avaliação de conhecimentos**. Na aula, cada grupo deverá **impreterivelmente** mostrar ao docente a resposta a todas as questões referidas na folha de respostas. Recomenda-se que **realize todo o trabalho em casa, usando a aula de laboratório apenas para testar o circuito na placa de prototipagem**. A folha de respostas (em formato PDF) e o projeto arquivado^a, deverão ser colocados num único ficheiro (em formato **ZIP**). Este ficheiro deverá ser entregue (no Fénix) **até às 23h59m de domingo, dia 20 de Novembro de 2016^b**.

A preparação prévia do trabalho de laboratório está dependente da instalação (bem-sucedida) da ferramenta de Vivado WebPack (versão 2016.3). A instalação deste software pode ser realizada com a ajuda do [Guia de Instalação do Vivado Design Suite WebPack](#), disponível na página da cadeira. Durante a realização deste trabalho, poderá ser útil consultar os seguintes slides das aulas teóricas:

1. Aula 12: [Linguagens de Descrição e Simulação de Circuitos Digitais](#)
2. Aula 14: Circuitos Sequenciais Básicos: Simbologia e Descrição em VHDL

Para conseguir realizar a prototipagem dos circuitos implementados no laboratório, é **fundamental** consultar o **Guia de Implementação de Circuitos na Placa de Desenvolvimento** (*Digilent Basys 3*), disponível na página da cadeira. Recomenda-se vivamente que **tenha estes documentos consigo durante a aula**.

2. ENIGMA: MÁQUINA DE (DES)CODIFICAÇÃO DOS CÓDIGOS

O objetivo deste trabalho consiste na implementação e simulação de uma versão simplificada da máquina *Enigma* utilizando a ferramenta *Xilinx Vivado WebPack* (utilizando o VHDL como linguagem de descrição de circuitos digitais) e a placa de prototipagem *Digilent Basys 3*.

A [máquina Enigma](#) (ilustrada na Figura 1) é uma máquina eletromecânica, utilizada tanto para criptografar (*codificar*) como para descriptografar (*descodificar*) códigos (*mensagens*) de guerra. Esta máquina adquiriu notabilidade por ter sido adaptada pelas forças militares alemãs durante a Segunda Guerra Mundial por razões de facilidade de uso e (supostamente) pela indecifrabildade das mensagens por ela codificadas.

^a Para arquivar o projeto no Vivado, clique *File*→*Archive Project*. Será aberta uma janela, onde deve indicar o nome ("*Archive name*") e localização em disco do projeto arquivado ("*Archive location*") - ex: no ambiente do trabalho. Clique *OK* para guardar o projeto em formato *zip*.

^b Em virtude de o 1º teste da cadeira se realizar na mesma semana que decorre este trabalho (dia 18 de Novembro), a entrega do relatório pode ser feita até ao domingo dessa semana (em vez de ser feita na 6ª feira, como é habitual).



Figura 1. Máquina enigma, utilizada pela marinha Alemã (II Guerra Mundial).

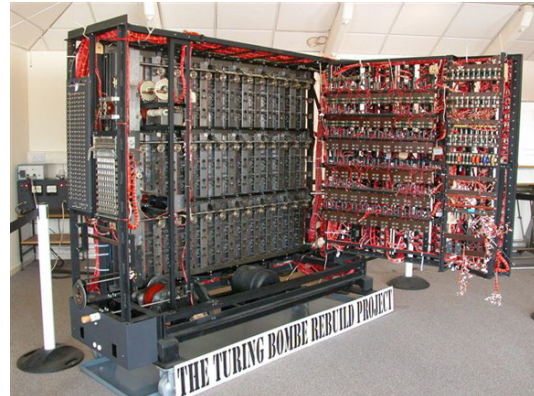


Figura 2. Réplica da máquina desenvolvida por Alan Turing.

Sob a liderança do matemático [Alan Turing](#), cientistas britânicos criaram uma máquina que visava acelerar o processo de descodificação das Enigmas usadas pela marinha alemã (ver Figura 2). Por isso mesmo, a máquina de Turing é habitualmente considerada como sendo o primeiro computador moderno. Usando a máquina do Turing, o código da marinha alemã foi decifrado, e é geralmente aceite que o acesso (pelos aliados) a informação contida nas mensagens que a Enigma não protegeu foi responsável pelo fim da Segunda Guerra Mundial pelo menos um ano antes do que seria de prever.

A versão simplificada da máquina Enigma, a implementar neste laboratório, é apresentada na Figura 3. A parte esquerda da imagem mostra a **enigma** (vista numa perspetiva global, i.e., definição do componente), indicando as 3 entradas de um bit (*clk*, *reset* e *start*) e 3 saídas de 8 bits (*mess_orig*, *mess_codif* e *mess_descod*). NOTA: a mensagem original a codificar (*mess_orig*) é gerada internamente.

A entrada *clk* (o sinal de relógio) da **enigma** é síncrona com o flanco ascendente. A entrada *start* (ativa a 1) serve para inicializar o processo da geração duma mensagem original (*mess_orig*), seguido pelos processos de geração da mensagem codificada (*mess_codif*) e da mensagem descodificada (*mess_descod*). A entrada *reset* é assíncrona e inicializa a *mess_orig* a zero.

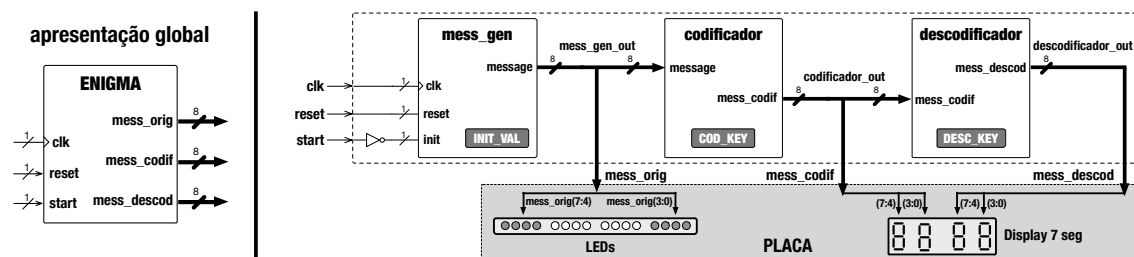


Figura 3. Máquina Enigma: definição do componente, implementação e ligações à placa Basys3

Ao nível da implementação (parte direita da Figura 3), a **enigma** é constituída por 3 componentes: **mess_gen**, **codificador** e **descodificador**. O componente **mess_gen** serve para gerar uma sequência de mensagens de teste aleatórias. Sempre que o sinal *start* da **enigma** for 1, o componente **mess_gen** gera uma mensagem aleatória na saída *message* (8-bits), sincronamente com o sinal de relógio (*clk*). Contudo, antes deste processo é necessário atribuir o valor inicial da mensagem (i.e., *INIT_VAL* ≠ 0). Para isso, deve ser colocado o sinal *start* da **enigma** a 0, que ativa a entrada assíncrona *init* do componente **mess_gen**. O valor de *INIT_VAL* é definido com base no número de aluno de menor valor, através dos dois dígitos menos significativos *K* (em decimal) e aplicando a fórmula seguinte:

$$INIT_VAL_DEC = \left\lfloor \frac{K}{2} \right\rfloor + 1.$$

Por exemplo, para o aluno com o número 84877, o valor de *K* é 77 e o *INIT_VAL_DEC* é 39 (38+1). Finalmente, o valor de *INIT_VAL* a utilizar é obtido através da conversão de *INIT_VAL_DEC* para um número binário de 6-bits.

A mensagem gerada pelo componente **mess_gen** (*mess_gen_out*) entra no **codificador** para ser codificada (saída *mess_codif* de 8-bits do codificador). Para esse efeito, o codificador criptografa a mensagem usando operações lógicas e aritméticas (em complemento para dois) de acordo com a fórmula:

$$MESS_CODIF = ROR((MESSAGE + COD_KEY), 4).^c$$

COD_KEY é um número binário de 8-bits definido com base na primeira letra do nome (x_1) e do apelido (x_2) do aluno com o número de maior valor. Para obter o valor de COD_KEY, é necessário concatenar os valores binários correspondentes aos x_1 e x_2 de acordo com a tabela em baixo:

Letra	A,N	B,O	C,P	D,Q	E,R	F,S	G,T	H,U	I,V	J,W	K,X	L,Y	M,Z
Código	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110

Por exemplo, para o aluno Diogo Silva, $x_1=D$ (0101) e $x_2=S$ (0111), o COD_KEY é 01010111.

A mensagem codificada (*codificador_out*) entra no **descodificador** para ser decifrada (saída *mess_descod* de 8-bits). Para esse efeito, o descodificador aplica operações lógicas e aritméticas (em complemento para dois) de acordo com a fórmula seguinte:

$$MESS_DESCOD = ROL(MESS_CODIF, 4) + DESC_KEY.^d$$

O valor do DESC_KEY deverá ser definido pelos alunos de forma a garantir que a mensagem obtida após a descodificação (*descodificador_out*) seja igual à mensagem original (i.e., *mess_gen_out* do componente **mess_gen**).

Responda à Pergunta 1 na folha de respostas.

Adicionalmente, a Figura 3 mostra ainda as ligações das saídas da máquina **enigma** implementada aos elementos de saída da placa *Digilent Basys 3*, onde os bits da mensagem original são ligados aos LEDs, enquanto a mensagem codificada e a mensagem descodificada são ligadas aos 4 displays de 7 segmentos.

3. GERADOR DE MENSAGENS (MESS_GEN)

O objetivo desta secção do tutorial consiste na análise de um circuito sequencial, realizada no âmbito de uma descrição tutorial da ferramenta Xilinx Vivado WebPack.

3.1. Descarregue o ficheiro **lab3.zip** disponível na página da cadeira. Guarde-o no seu ambiente de trabalho e descompacte-o para uma outra pasta de nome **lab3**. A pasta **lab3** tem 3 subdiretorias: *design_sources*, *simulation_sources* e *placa*.

- A pasta **design_sources** contém os ficheiros “.vhd” que representam a descrição (em VHDL) dos vários circuitos utilizados para implementar a enigma. Os *Design Sources* fornecidos têm sempre a definição do circuito no nível do componente (*entity*), bem como a sua implementação detalhada (*architecture*).
- A pasta **simulation_sources** contém os ficheiros “.vhd” utilizados para a simulação (em VHDL) de todos os circuitos utilizados, i.e., *Simulation Sources* ou *Test Benches* (*tb*). As *Simulation Sources* fornecidas incluem uma sequência de teste para simular (verificar) o comportamento do circuito.
- A pasta **placa** contém vários ficheiros necessários para estabelecer as ligações entre a implementação da **Enigma** (em VHDL) e a placa de desenvolvimento *Digilent Basys 3*.

3.2. Inicie o ambiente de projeto Vivado (na lista de programas procure *Xilinx Design Tools*→*Vivado 2016.3* e clique na aplicação *Vivado 2016.3*).

^c $ROR(y, 4)$ é a operação *rotate right* (rotação à direita) de 4 bits do valor y .

Exemplo: $ROR(01010111, 4) = 01110101$.

^d $ROL(y, 4)$ é a operação *rotate left* (rotação à esquerda) de 4 bits do valor y .

Exemplo: $ROL(01100111, 4) = 01110110$.

3.3. Criação de um projeto novo.

Clique *File→New Project*. Abrir-se-á uma janela e clique *Next*. Na janela seguinte indique o nome do projeto (*Project name*) e localização em disco do projeto (*Project location*). Deixe a opção *Create project subdirectory* ativa (desde modo, será criada uma subdiretoria com o nome do projeto). Após o preenchimento destes campos, pressione em *Next*. Na janela seguinte (*Project Type*) selecione a opção *RTL Project* e ative a opção *Do not specify sources at this time*. Clique *Next*.

3.4. Especificação do dispositivo.

Na janela seguinte (*Default Type*) é preciso especificar a placa utilizada no laboratório, i.e., *Digilent Basys 3* baseada na FPGA da família Artix-7 com a referência *XC7A35TCPG236-1*. Preencha os campos *Family*, *Package* e *Speed Grade* de acordo com a Figura 4 e selecione o dispositivo *XC7A35TCPG236-1*. Clique *Next*.

Em seguida, complete a criação do projeto pressionado *Finish*.

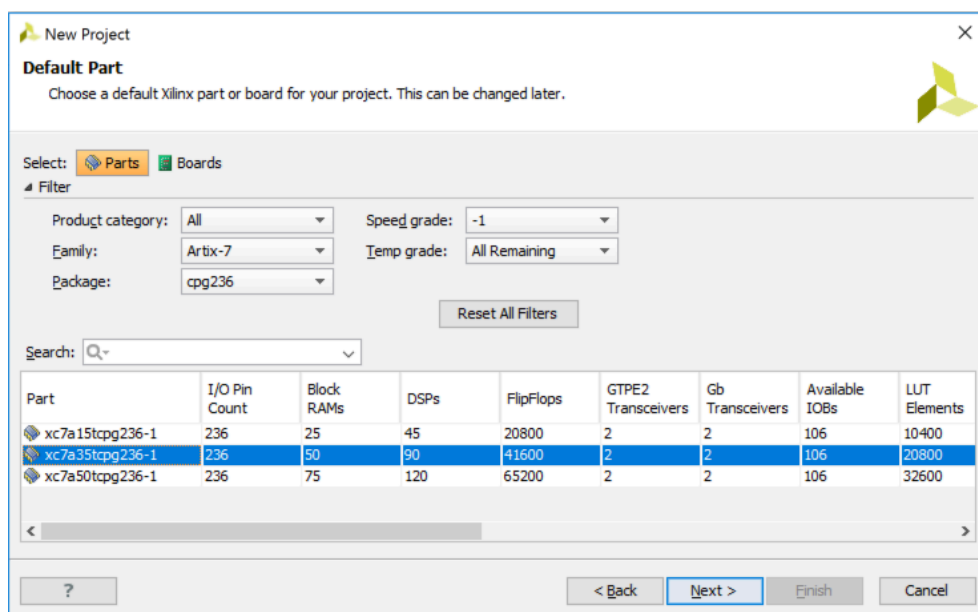


Figura 4. Opções do projeto: especificação do dispositivo.

3.5. Clique *Tools→Options...* e verifique se a opção *Copy sources into Project* está ativada na secção *Source Files* e a opção *VHDL* na secção *Target Language*. Clique *OK* (Figura 5).

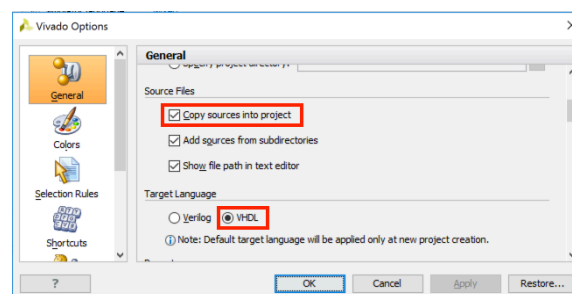



Figura 5. Opções do Vivado: Configuração

3.6. Inclusão de um ficheiro de descrição do circuito (*Design Source*)

Faça *File→Add Sources...*. Será aberta uma janela e selecione a opção *Add or create design sources*. Clique *Next*. Pressione o ícone , selecione *Add files...* e navegue até à pasta com os ficheiros descompactados (reveja o ponto 3.1). Na pasta *design_sources*, selecione *ff_elem*, *ff_d* e *m21*. Complete a importação clicando *OK* e depois *Finish*.

- 3.7. Após inserir os *Design Sources*, o ambiente do Vivado deverá ter o aspeto indicado na Figura 6. Note ainda que este aspeto poderá ser modificado, alterando a dimensão e localização de cada um dos cinco painéis (P1, P2, P3, P4 e P5).

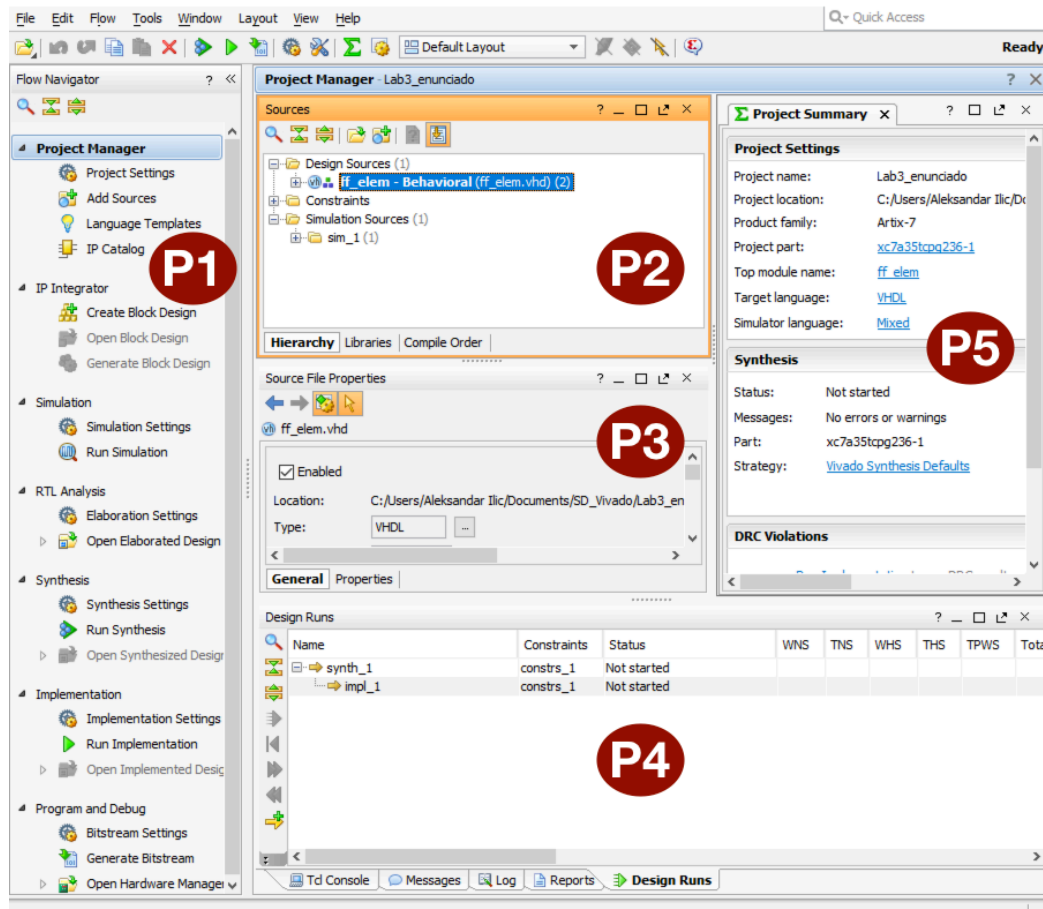




Figura 6. Ambiente de desenvolvimento no Xilinx Vivado.

- 3.8. Observe o painel P2: o ficheiro `ff_elem` foi incluído na pasta "Design Sources". **Expandir o circuito** `ff_elem` clicando no símbolo  (à esquerda) para visualizar as instâncias dos componentes `ff_dem21` que o `ff_elem` engloba.
- 3.9. **Definir o circuito como o módulo de topo.**
 Se apresentação for diferente, é preciso definir o circuito como módulo de topo. No painel P2, faça clique direito no nome do circuito (e.g., `ff_elem`) e clique "Set as Top".
- 3.10. **Abrir/editar o Design Source do circuito.**
 Na pasta "Design Sources" (painel P2), clique duas vezes no nome do circuito (e.g., `ff_elem`), no painel P5 abrirá um novo *tab* com o nome do circuito e o código em VHDL será apresentado.
- 3.11. **Visualização do esquema do circuito (logograma).**
 No painel P1, expanda a opção "Open Elaborated Design" da secção "RTL Analysis" e clique uma vez em  Schematic. Na janela seguinte clique OK. Será aberto (no painel P5) um *tab* "Schematic" apresentando o logograma do circuito (ver Figura 7, parte direita). A parte esquerda da Figura 7 representa o componente `ff_elem` na sua visão de componente, i.e., definição ou *entity* em VHDL.

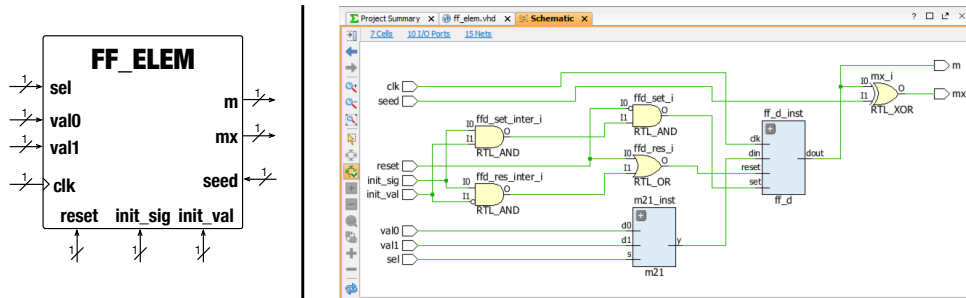


Figura 7. FF_ELEM: Definição e implementação (logigrama) do circuito.

Ao ativar a opção Schematic, a ferramenta começa por realizar a verificação da correção do circuito, i.e., verifica se existe algum erro na descrição nos ficheiros adicionados. Em caso de sucesso, não serão mostradas mensagens de erro no tab *Messages* no painel P4 (as mensagens de *info* e *status* podem aparecer e podem, em geral, ser ignoradas). Após a visualização do logigrama, clique no tab “Sources” do painel P2 para voltar à hierarquia do projeto, com todos os ficheiros adicionados.

Responda às perguntas 2, 3, 4 e 5 na folha de respostas.

- 3.12. Repetindo o procedimento descrito na alínea 3.6, adicione o *Design Source* correspondente ao componente `mess_gen`.
- 3.13. Após adicionar um ficheiro ao projeto, verifique se este fica configurado como sendo o módulo de topo (reveja o passo 3.9). Por vezes, em cima do painel P2 aparecerá a mensagem de aviso “*Elaborated design out of date...*”. Clique em *Reload* e observe o logigrama do circuito (reveja o passo 3.11). No painel P2, clique no tab *Sources* para voltar para a hierarquia do projeto, e clique duas vezes no nome do ficheiro para visualizar a código em VHDL (reveja o passo 3.10). Inspeccione o conteúdo, comparando a descrição em VHDL com o logigrama apresentado.
- 3.14. **Inclusão de um ficheiro para efetuar a simulação do circuito (*Simulation Source*).**
Faça “*File→Add Sources...*”. Será aberta uma janela e selecione a opção “*Add or create simulation sources*”. Clique *Next*. Pressione e selecione “*Add files...*”. Navegue até à pasta com os ficheiros descompactados (reveja o ponto 3.1). Selecione o ficheiro `tb_mess_gen.vhd` na subdiretoria `simulation_sources`. Complete a importação clicando *OK* e depois *Finish*. Repare no painel P2: o ficheiro foi incluído na pasta “*Simulation Sources*”. Se for preciso, expanda a pasta “*Simulation Sources*” e todas subdiretorias clicando no símbolo (habitualmente, o *Simulation Source* é colocado na subdiretoria `sim1`).
- 3.15. **Abrir/editar o *Simulation Source* do circuito.**
Utilizando o painel P2 e a pasta “*Simulation Sources*”, clique no nome do *TestBench* (e.g., `tb_mess_gen`). No painel P5 abrir-se-á um novo *tab* com o código VHDL do *TestBench* selecionado. Inspeccione o seu conteúdo. Para observar a variação do valor da(s) saída(s), é preciso iniciar a simulação do circuito.
- 3.16. **Inicialização da simulação.**
Certifique-se que o ficheiro da simulação é o módulo de topo (reveja o passo 3.9). No painel P1 e na pasta “*Simulation*”, clique Run Simulation e em seguida em “*Run Behavioral Simulation*”. Abrir-se-á um novo separador com nome *Behavioral Simulation* no lugar do anterior separador *Project Manager* com um aspeto semelhante ao representado na Figura 8.

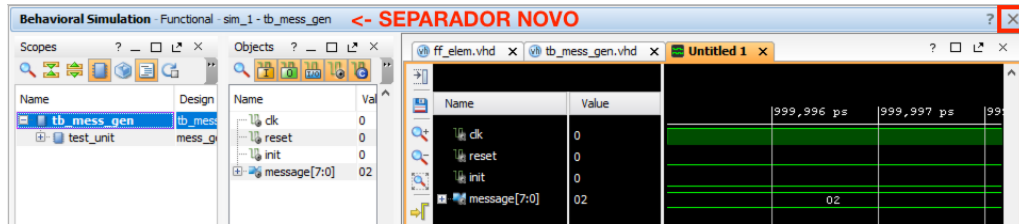



Figura 8. Simulação do circuito mess_gen.



Clique no botão  (Zoom Fit) para poder observar a simulação na sua totalidade. Verifique ainda o tab *Messages* no painel P4 para saber se a ferramenta detetou algum erro durante a elaboração ou execução da simulação.

Após acabar a simulação, feche o separador *Behavioral Simulation* para voltar ao separador *Project Manager* (ou *Elaborated Design*).

- 3.17. Utilizando o procedimento descrito na alínea 3.10, abra o *Design Source* do circuito mess_gen. O valor inicial da mensagem é definido através do sinal interno "init_val" (ver linha 61). Mude o valor deste sinal para o valor INIT_VAL calculado na Secção 2 (i.e., pergunta 1).

Grave o ficheiro e verifique se não são mostradas mensagens de erro no tab *Messages* no painel P4 (ignorando as mensagens de *info* e *status*) e corrija os erros. Nesse caso, a ferramenta deteta erros de sintaxe, identificados como *critical warnings*!

Quando tiver resolvido os erros de sintaxe apresentados pela ferramenta, volta a gravar o ficheiro para a ferramenta atualizar a informação relativa aos erros.

- 3.18. É importante ter em consideração que nem todos os erros são apresentados quando se grava o ficheiro. Para fazer a verificação completa é necessário fazer a **síntese do circuito**, clicando na opção  *Run Synthesis* na pasta "*Synthesis*" do painel P1 (pode demorar algum tempo). Para verificar o estado da síntese, observe o *canto superior direito*, onde o progresso da operação será reportado *Running synth_design*  *Cancel* e quando acabar aparecerá a mensagem **Synthesis Complete**. Será então aberta a janela "*Synthesis Complete*". Clique *Cancel*. No caso de existirem erros no código, aparecerá a janela "*Synthesis Failed*". Clique *OK*. No painel P4, clique no tab "*Messages*", onde a ferramenta poderá indicar um conjunto de avisos (*warnings*) e erros. Os erros deverão ser **todos** corrigidos, enquanto os *infos*, *statuses* e alguns *warnings* podem, em geral, ser ignorados.

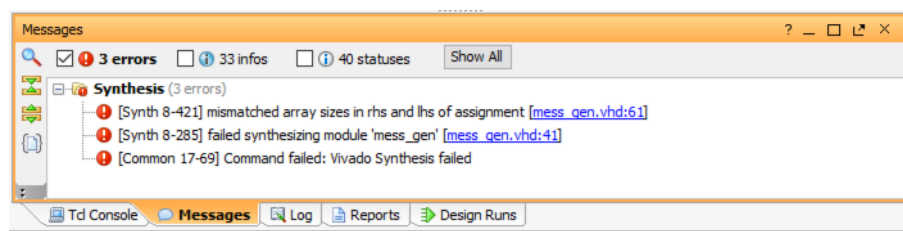






Figura 9. Síntese do circuito: Erros na descrição VHDL

A Figura 9 apresenta um exemplo de um erro detectado durante a síntese do circuito. Para localizar o erro, clique no *hyperlink* representado a azul (ver Figura 9) para se deslocar para a linha de código respetiva (por vezes, o erro encontra-se numa linha de código acima da referenciada). É de notar que a descrição dos erros apresentada pela ferramenta nem sempre descreve, de forma clara, a causa do erro e nesses casos é necessário consultar a descrição dos *warnings*. Após corrigir os erros, corra o processo de síntese do circuito até não obter nenhum erro.

- 3.19. Reinicie a simulação usando a *Simulation Source* tb_mess_gen, repetindo o passo 3.16.

- 3.20. Para **aumentar o tempo de simulação** (para além dos 1000 ns) é necessário indicar o período do tempo desejado no campo  1000 ns (por cima do separador *Behavioral Simulation*). Depois, é necessário recomençar a simulação (clicando no ) e clicar no botão . Clique no botão  (Zoom Fit) para poder observar a simulação na sua totalidade.
- 3.21. Na simulação, apenas aparecem os sinais de entrada e saída do circuito testado. No entanto, será mais fácil verificar o correto funcionamento do circuito se adicionar alguns sinais internos.

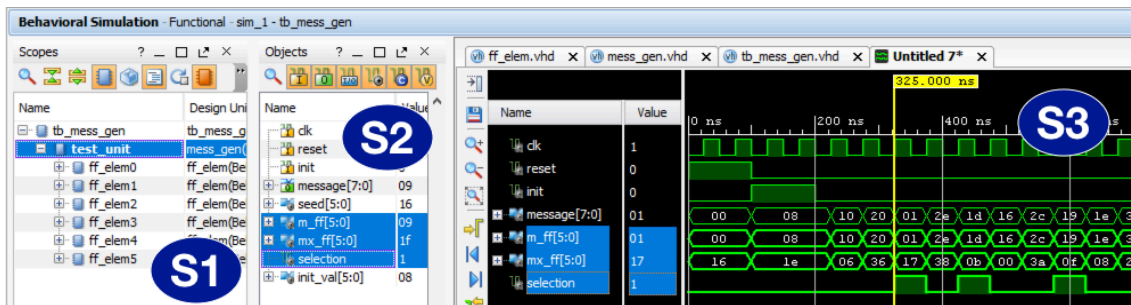


Figura 10. Ambiente de simulação no Xilinx Vivado.

Como ilustrado na Figura 10, existem 3 painéis no ambiente da simulação: S1, S2 e S3. Para adicionar sinais internos à simulação, selecione a instância desejada no painel S1 (*Scope*) (e.g., *test_unit*) e em seguida selecione os sinais de interesse no painel S2 (*Objects*) (e.g., *m_ff*, *mx_ff* e *selection*). Arraste-os para o painel S3 (ou então faça clique direito e no drop-menu selecione “Add To Wave Window”).

Terá de recomençar a simulação (reveja o passo 3.20) para que os sinais adicionados sejam atualizados no painel S3. Para modificar a representação dum sinal, faça clique direito sobre o sinal em questão (no painel S3), no *drop-menu* selecione “Radix” e no menu seguinte selecione a representação desejada (binário, hexadecimal, octal, número inteiro com ou sem sinal etc.).

Responda à Pergunta 6 na folha de respostas.

4. CODIFICADOR

O objetivo desta secção do tutorial é implementar o módulo codificador do enigma que crie uma mensagem codificada (*mess_codif*) com base na mensagem original recebida através do módulo *mess_gen* (entrada *message*). Para este fim, o codificador realiza a operação seguinte:

$$MESS_CODIF = ROR((MESSAGE + COD_KEY), 4).$$

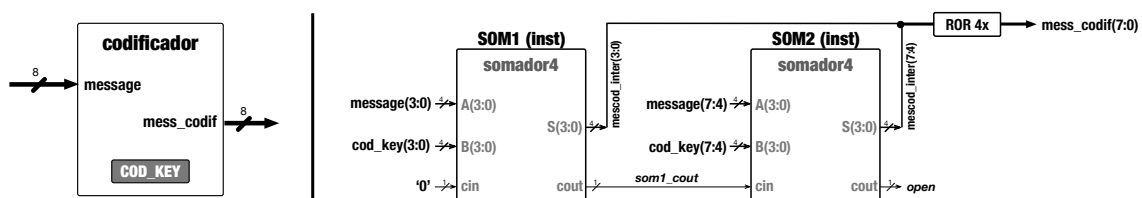


Figura 11. Codificador: Definição e logograma (implementação) do componente

Conforme ilustrado na Figura 11, a soma (em complemento para dois) entre a mensagem original e o parâmetro *COD_KEY* é realizada utilizando duas instâncias de somador de 4-bits, i.e., *SOM1* e *SOM2*, ligadas em cascata usando o sinal de transporte *som1_cout*. O resultado da soma (*mescod_inter*) é então deslocado à direita (*rotate right* (ROR)) em 4 posições, de modo a produzir a mensagem codificada (*mess_codif*).

- 4.1. Comece por adicionar os *Design Sources* (reveja o passo 3.6) dos componentes `full_adder` e `somador4`. Defina o `somador4` como o módulo de topo (reveja o passo 3.9) e faça o *Reload* do “*Elaborated Design*”, repetindo o passo 3.13. Expanda o `somador4` no painel P2 (reveja o passo 3.8) e abra os *Design Sources* adicionados (rever passo 3.10). Como se pode observar, o somador de 4 bits é implementado utilizando 4 *full adders*, tal como descrito na [aula teórica nº12](#).
- 4.2. Adicione o *Design Source* `codificador` utilizando o mesmo procedimento descrito no passo 4.1 e abra-o. Observe a correspondência entre a definição do componente apresentada na Figura 11 (à esquerda) e a parte do código VHDL entre “`entity codificador is`” e “`end codificador;`”. Observe, também, a correspondência entre o logograma apresentado na Figura 11 (à direita) e a sua descrição em VHDL, i.e., a parte do código entre “`architecture Behavioral of codificador is`” e “`end Behavioral;`”. Crie o logograma do `codificador` no Vivado (reveja o passo 3.11) e compare o esquema obtido com o logograma apresentado na Figura 11.
- 4.3. Adicione o *Simulation Source* `tb_codificador` (reveja o passo 3.14). Defina este ficheiro como módulo de topo (reveja o passo 3.9) e execute a simulação (reveja o passo 3.16).
- 4.4. Abra o *Design Source* do `codificador` (ou clique no tab `codificador.vhd` no painel P5, se já se encontra aberto). Na linha 53 é necessário alterar o valor do sinal `COD_KEY` para o valor calculado na Secção 2 (i.e., pergunta 1). Grave o ficheiro e faça a síntese do circuito, repetindo o passo 3.18.
- 4.5. Se nenhum erro for reportado, execute a simulação do circuito (veja o passo 3.16).

Responda à Pergunta 7 na folha de respostas.

5. DESCODIFICADOR

O objetivo desta secção do tutorial consiste na implementação do descodificador do enigma, que recebe a mensagem codificada na entrada `mess_codif` e transforma-a na mensagem original. Desta forma, a mensagem descodificada (`mess_descod`) deverá ter o mesmo valor da mensagem que saiu do módulo `mess_gen`. Assim, para concretizar a sua função, o descodificador deverá realizar a seguinte operação:

$$MESS_DESCOD = ROL(MESS_CODIF, 4) + DESC_KEY.$$

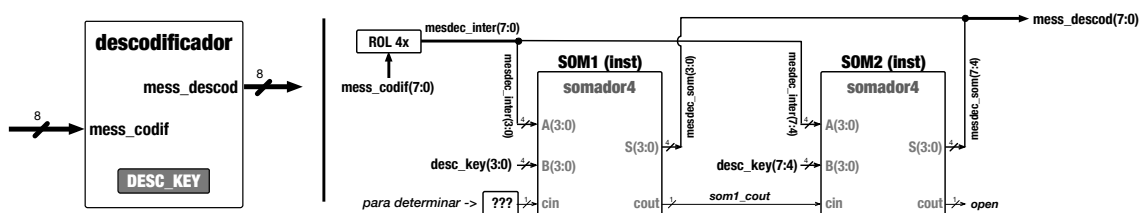


Figura 12. Descodificador: Definição do componente e logograma (implementação) parcial

Como mostrado na Figura 12, é necessário realizar em primeiro lugar a operação de deslocamento à esquerda (*rotate left* (ROL)) de 4 posições da mensagem codificada, de modo a obter o sinal interno `mesdec_inter` de 8 bits. Depois, o valor de `mesdec_inter` deverá ser somado (em complemento para dois) com o valor de `DESC_KEY`, de modo a obter o sinal interno `mesdec_som` de 8 bits. Para este efeito, é necessário instanciar dois somadores de 4-bits, i.e., `SOM1` e `SOM2`, ligados em cascata através do sinal interno `som1_cout`. No final, o valor do sinal `mesdec_som` deverá ser ligado directamente à saída do módulo descodificador (`mess_descod`).

- 5.1. Comece por adicionar o *Design Source* do `descodificador` (reveja o passo 3.6). Torne-o o módulo de topo (reveja o passo 3.9), faça o *Reload* do “*Elaborated Design*” repetindo o

passo 3.13, e abra o ficheiro (reveja o passo 3.10). Conforme poderá observar, o circuito está parcialmente implementado. É fornecida a definição do descodificador (i.e., “*entity*” que corresponde à Figura 12, parte esquerda), bem como a declaração de todos os sinais internos (parte direita da Figura 12) que estão apenas interligados na implementação (i.e., entre o “*begin*” e “*end Behavioral;*” na parte “*architecture*” em VHDL).

- 5.2. Adicione o *Simulation Source* `tb_descodificador` (reveja o passo 3.14) e torne-o o módulo de topo (reveja o passo 3.9). Este *Simulation Source* está completamente preenchido, pelo que não é necessário introduzir nenhuma modificação. Por conseguinte, este ficheiro pode ser utilizado durante a implementação, para verificar o correcto funcionamento do descodificador.
- 5.3. Complete a implementação do descodificador de modo a obter o funcionamento acima descrito. Tal como anteriormente, é necessário alterar o valor do `DESC_KEY` na linha 46 para o valor calculado na Secção 2 (i.e., pergunta 1). Depois, ainda neste ficheiro, deverá declarar o componente `somador4` antes do “*begin*” na parte “*architecture*”. Em seguida, depois do “*begin*”, ou seja, na parte da implementação, comece por realizar a operação ROL sobre a entrada `mess_codif`, de modo a formar o sinal interno `mesdec_inter`. Depois, é necessário criar duas instâncias do componente `somador4` e fazer as ligações necessárias de acordo com a Figura 12 (como exemplificado na implementação do `codificador` e nos slides da [aula teórica nº12](#)). As saídas das duas instâncias de somador (`mesdec_som`) deverão então ser ligadas à saída `mess_descod` do módulo descodificador.
Quando terminar a implementação, faça a síntese do circuito (reveja o passo 3.18) e corrija **todos** os erros. Crie o logograma do descodificador implementado (reveja o 3.11).

Nota importante: Não pode alterar a definição do descodificador (ou seja, a “*entity*” em VHDL) e não é recomendável mudar os nomes de fios internos!

Responda à Pergunta 8 na folha de respostas.

- 5.4. Efetue a simulação do módulo descodificador (reveja o passo 3.16) usando o *Simulation Source* `tb_descodificador`.

Responda à Pergunta 9 na folha de respostas.

6. ENIGMA: PUTTING IT ALL TOGETHER

O objetivo desta secção do tutorial é finalizar a implementação da *enigma*, ligando todos os circuitos previamente elaborados nas Secções 3, 4 e 5 e de acordo com a Figura 3.

- 6.1. Comece por incluir o *Design Source* da *enigma* (reveja o passo 3.6). Torne-o o módulo de topo (reveja o passo 3.9) e faça o *Reload* do “*Elaborated Design*”, repetindo o passo 3.13. Expanda o circuito da *enigma* no painel P2 (reveja o passo 3.8) e abra-o (reveja o passo 3.10). Observe a correspondência entre o código `vhdl` e a engima apresentado na Figura 3.
- 6.2. Adicione o *Simulation Source* `tb_enigma` (reveja o passo 3.6). Torne-o módulo de topo (reveja o passo 3.9) e abra o ficheiro (reveja o passo 3.15).
Após concluir estes dois passos (6.1 e 6.2), o painel P2 deverá ter o aspeto apresentado na Figura 13.

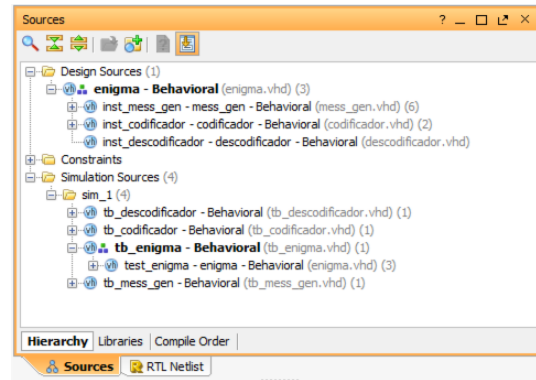


Figura 13. O painel P2 (Sources)

6.3. Faça a simulação da enigma (reveja o passo 3.16) usando o *Simulation Source* `tb_enigma`.

Responda à Pergunta 10 na folha de respostas.

7. TESTE DO CIRCUITO NA PLACA DE PROTOTIPAGEM

Nota importante: Antes de iniciar o teste do circuito é **fundamental consultar (em casa)** o **Guia de Implementação de Circuitos na Placa de Desenvolvimento** (Digilent Basys 3), disponível na página da cadeira.

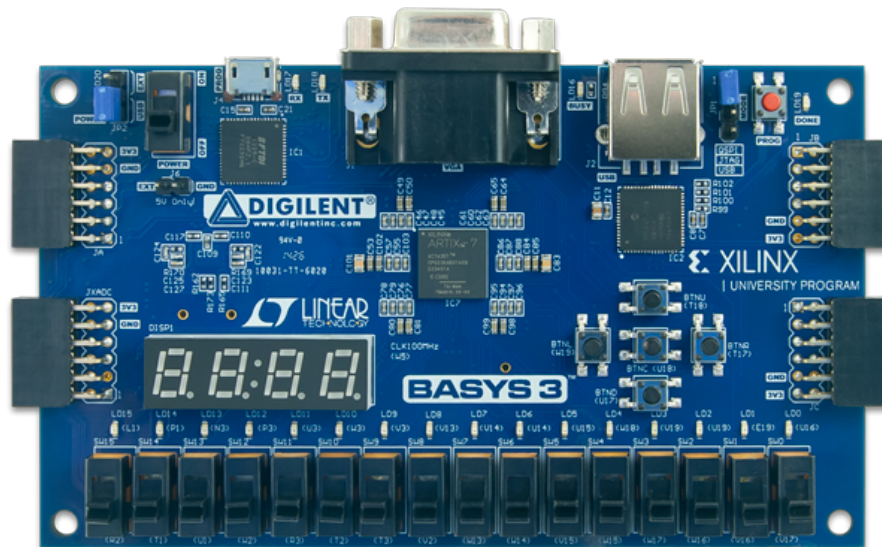


Figura 14. Placa de prototipagem Basys 3.

Para realizar o teste da máquina *enigma* projetada utilizando a placa de prototipagem (*Digilent Basys 3*, equipada com a FPGA *Artix-7* da Xilinx – ver Figura 14), foi disponibilizado um conjunto de ficheiros na pasta **placa** (veja a descrição dos ficheiros descompactados no ponto 3.1), que deverá utilizar nesta parte do trabalho:

- **sd.vhd** – descrição do circuito principal (da placa)
- **Basys3_Master.xdc** – configuração dos portos (da placa)
- **clkdiv.vhd** – divisor de frequência (especificação)
- **disp7.vhd** – bloco do controlo do display de 7 segmentos (especificação).

Não modifique os nomes destes ficheiros!

- 7.1. Adicione os ficheiros `sd.vhd`, `clkdiv.vhd` e `disp7.vhd` ao projeto, fazendo *Add Sources*→*Add or create **design** sources* (reveja o passo 3.6).
- 7.2. Adicione o ficheiro `Basys3_Master.xdc` ao projeto, fazendo *Add Sources*→*Add or create **constraints***.
- 7.3. Verifique se o ficheiro `sd.vhd` está definido como módulo de topo (faça clique direito no ficheiro e selecione a opção “*Set as Top*”). Verifique também se a hierarquia do projeto inclui os componentes `clkdiv`, `disp7` e `Basys3_Master.xdc`, conforme indicado na Figura 15. A inclusão destes componentes é obrigatória e deve ser sempre verificada (**a não inclusão do ficheiro `Basys3_Master.xdc`, pode DESTRUIR o dispositivo, e caso isso aconteça ser-lhe-ão pedidas responsabilidades**).

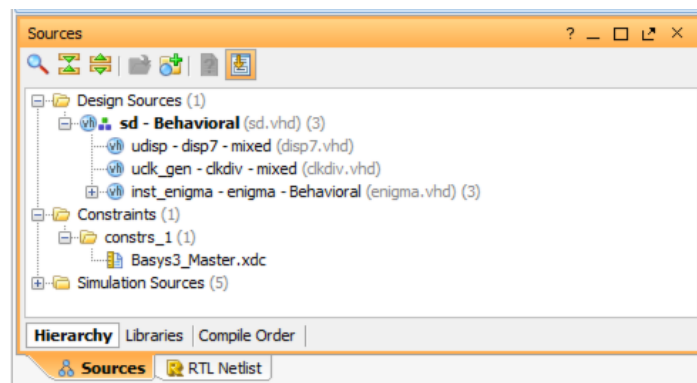


Figura 15. Hierarquia do projeto incluindo `sd`, `clkdiv`, `disp7` e `Basys3_Master.xdc`

- 7.4. Abra o módulo `sd`, clicando duas vezes em cima do ficheiro `sd.vhd`.
Este projeto não é mais do que uma interface da placa que é disponibilizada ao aluno: as entradas e saídas *já estão configuradas* de acordo com o modelo do dispositivo utilizado na placa de desenvolvimento. Por conseguinte, este módulo funciona como uma placa de prototipagem virtual.
Nota: Não altere o conteúdo do código neste ficheiro a não ser que tal lhe seja pedido pelo docente.
- 7.5. Conforme ilustrado na Figura 3, as seguintes ligações foram estabelecidas de forma a possibilitar a correta interação do utilizador com o circuito:
 - a) O sinal de relógio ***clk*** está ligado ao sinal ***clk_slow*** (este sinal tem uma frequência fixa de 1,5 Hz);
 - b) A entrada ***reset*** está ligada ao buffer do botão de pressão ***BTN(4)***, i.e., o botão central;
 - c) O sinal de entrada ***init*** está ligado ao interruptor ***SW(0)***, i.e., o primeiro interruptor (switch) do lado direito da placa;
 - d) Os bits da mensagem original (***mess_orig***) estão ligados aos buffers dos ***LEDs***, conforme ilustrado na Figura 3: ***mess_orig(3:0)*** aos ***LED(3:0)*** (os LEDs mais à direita) e ***mess_orig(7:4)*** aos ***LED(15:12)*** (os LEDs mais à esquerda);
 - e) A mensagem codificada (***mess_codif***) é apresentada nos dígitos 3 e 2 do display de 7 segmentos em formato hexadecimal, i.e., ***disp3*** e ***disp2***;
 - f) A mensagem decodificada (***mess_descod***) é apresentada nos dígitos 1 e 0 do display de 7 segmentos em formato hexadecimal, i.e., ***disp1*** e ***disp0***;
 - g) A escrita nos dígitos do display de 7 segmentos é ativada pelas entradas ***aceso(3:0) = "1111"***.

- 7.6. Implemente o circuito na placa de desenvolvimento. Para tal, siga as instruções disponibilizadas no "**Guia de Implementação de Circuitos na Placa de Desenvolvimento**". Note que o interruptor ON/OFF da placa deve estar na posição ON.
Nota: durante a síntese do circuito, a ferramenta poderá indicar um conjunto de avisos (warnings) e erros. Os erros deverão ser **todos** corrigidos; os warnings podem, em geral, ser ignorados, sendo que alguns são originados pelo facto de ter entradas/saídas no ar.
- 7.7. Verifique o correto funcionamento do circuito. Mostre-o ao docente. Comente.
- 7.8. Comente o funcionamento do interruptor (switch) $SW(0)$. Explique o que observa nos LEDs e justifique a informação que visualiza nos displays.

Responda à Pergunta 11 na folha de respostas.