Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática

LABORATÓRIO DE ORGANIZAÇÃO DE COMPUTADORES

TEXTO DE APOIO PARA LABORATÓRIOS

Autores: Ney Laert Vilar Calazans
Fernando Gehm Moraes

2006/1 (Versão 1.8)

SUMÁRIO

1 Equipamentos de Medida, Excitação, Teste e Aquisição de Grandezas Elétricas e Dados 1
1.1 Objetivos 1
1.2 Introdução 1
Medida de Sinais Elétricos – o Multímetro 4
Geração de Sinais Elétricos – o Gerador de Ondas 5
Medidas de Sinais Elétricos – o Osciloscópio 5
1.3 A Fazer e a Entregar 7
2 Implementação de Sistemas Digitais com HDLs - Ferramentas de Captura e Validação 9
2.1 Introdução e Objetivos 9
2.2 Localização do Ambiente de Desenvolvimento Active-HDL 9
2.3 Início do Projeto 9
2.4 Implementação de um Circuito Combinacional em VHDL 10
Criação do arquivo VHDL Inicial 11
Definição da Arquitetura do Módulo 12
Definição do ambiente de validação do módulo (testbench) 13
2.5 Compilação e Simulação 16
2.6 A Fazer e a Entregar 17
2.7 Apêndice – Visão Geral do Ambiente de Desenvolvimento Active-HDL 17
3 Implementação de Sistemas Digitais com HDLs Captura, Síntese, Implementação e Teste 19
3.1 Introdução e Objetivos 19
3.2 Conceitos Fundamentais de Ferramentas de CAD e Hardware Utilizado 19
3.3 Criação de um Projeto ISE 20
Criação do Arquivo Topo da Hierarquia 20
Inserção do Arquivo com a descrição do Circuito 21
3.4 Definição do Top, Pinagem Externa, Síntese Lógica e Síntese Física 22
Definição do Top 22
Definição da Pinagem Externa 22
Síntese Lógica 23
Síntese Física 24
3.5 Descarga do arquivo de Configuração e Teste do Circuito 25
Conexão e Alimentação da Plataforma no Hospedeiro 25
Configuração do FPGA 25
Teste do Circuito 26
3.6 A Fazer e a Entregar 26
4 Projeto e Validação de Hardware Combinacional em VHDL 28
4.1 Introdução e Objetivos 28
4.2 Especificação da ULA 29
4.3 A Fazer e a Entregar 29
5 Implementação de Hardware Combinacional em HDL Captura, Síntese, Implementação e Teste 32
5.1 Introdução e Objetivos 32
5.2 Atribuição de Recursos de E/S 32
5.3 Criação do Arquivo `top.vhd` 33
5.4 A Fazer e a Entregar 35
6 Processador Cleópatra: Organização VHDL e Programação 36
6.1 Introdução e Objetivos 36
6.2 A Fazer e a Entregar 36
7 Circuitos Seqüenciais e Máquinas de Estados Finitas 38
7.1 Objetivos 38
7.2 Especificação da Máquina 38
7.3 Projeto da Máquina 39
7.4 A Fazer e a Entregar 40
8 Modelo Bloco de Dados – Bloco de Controle: Princípios 42
8.1 Objetivos 42
8.2 Especificação do Projeto de Comunicação Serial Síncrona 43
8.3 A Fazer e a Entregar 44
9 Processador Cleópatra – Prototipação, Programação e Entrada e Saída 45
9.1 Objetivos 45
9.2 Prototipação do Processador Cleópatra 45
9.3 Acréscimo de um periférico ao Processador Cleópatra 45
9.4 A Fazer e a Entregar 46

Observações:

- Ao final de cada Laboratório há um conjunto de "tarefas" que devem ser executadas. Deve-se elaborar para cada Laboratório um relatório detalhando a execução das "tarefas" e as respostas às perguntas elaboradas, quando for o caso.
- **Atenção – Importantíssimo:** Cada projeto criado no sistema de CAD ISE e no ambiente de simulação Active-HDL cria uma quantidade enorme de arquivos de forma automática. Para tornar possível a correção adequada dos trabalhos, é indispensável que toda a hierarquia de projeto seja entregue ao professor. Para fazer isto de forma correta, os ambientes possuem um utilitário de arquivamento e compactação de projetos que deve ser usado. Quando o projeto estiver pronto para ser entregue, use a opção de menu **File →Archive Project...** do sistema Foundation (no sistema Active-HDL é a opção de menu **Design → Archive Design...**) para transformar toda a hierarquia de projeto num arquivo único com a denominação **<nome_do_seu_projeto>.zip**. Este arquivo deve ser entregue ao professor, quando se solicitar a entrega do projeto. A entrega de projetos incompletos ou em outro formato implicará a não avaliação do trabalho, com a conseqüente perda da nota do trabalho.

1 Equipamentos de Medida, Excitação, Teste e Aquisição de Grandezas Elétricas e Dados

Prática: Geração e Medida de Sinais Elétricos
Recursos: Multímetro, Gerador de Ondas e Osciloscópio

1.1 Objetivos

O objetivo deste laboratório é fornecer aos alunos maneiras de lidar com ferramental básico usado na geração e medida de grandezas elétricas, bem com a forma como estas grandezas elétricas são usadas para representar informação digital. Este ferramental servirá mais tarde para auxiliar na validação de projetos de sistemas digitais com os quais os alunos trabalharão. Os objetivos específicos compreendem a familiarização com instrumentos de geração e medida de sinais elétricos, em particular com geradores de forma de onda, multímetros e osciloscópios. Ao final deste laboratório, o aluno deve estar apto a usar os recursos básicos de instrumentos de medidas tais como osciloscópios e multímetros, e de aparelhos geradores de excitações, tal como geradores de formas de onda.

1.2 Introdução

Este laboratório dedica-se à apresentação de equipamentos utilizados para a geração de sinais elétricos e para a medição e interpretação destes.

A Ciência da Computação, com todo seu poder e impacto no mundo atual, está baseada na manipulação de ***informação*** que é ***representada*** sob forma ***digital binária***. A compreensão do sentido exato das palavras em itálico na frase anterior é crítica para esta introdução.

Intuitivamente, a *informação* existe sempre que há a possibilidade de escolha de um entre diversos valores possíveis. Por exemplo, a expressão uma das cores da bandeira brasileira transporta uma informação cujo valor, a cada instante, é um elemento escolhido de um conjunto de quatro elementos, representado matematicamente, por exemplo, por CBB={verde, amarelo, azul, branco}. Por outro lado, a expressão a cor do cavalo branco de Napoleão não transporta informação nenhuma. Isto ocorre porque o valor associado à expressão, a cada momento, só pode ser sempre o mesmo, branco. Matematicamente, o conjunto de onde se retira o valor desta expressão é um conjunto unitário.

A segunda palavra em itálico acima, *representada*, versa sobre como a informação pode ser armazenada e manipulada. No exemplo do conjunto CBB acima, os quatros elementos do conjunto foram *representados* por 4 seqüências ordenadas de símbolos do alfabeto usado na Língua Portuguesa, todas as seqüências sendo distintas entre si. Em última instância, nossa língua representa toda e qualquer palavra por seqüências finitas de caracteres retirados de um conjunto com 23 elementos, denominado **alfabeto**. Embora saibamos que o número de palavras da língua portuguesa é finito, isto não é uma limitação imposta pela regra de formação ou pelo alfabeto, que implicam obviamente um conjunto infinito. Logo, mesmo um número infinito de elementos poderia ser batizado, criando uma palavra para denominar cada um deles. Note que a escolha de 23 letras para criar qualquer palavra é menos o resultado de uma reflexão cuidadosa e mais de uma evolução histórica milenar das culturas e línguas. O mesmo efeito de representação poderia ser obtido com os 10 dígitos e a regras de formação de números, ou com os milhares de kanjis da língua japonesa ou os hieróglifos egípcios e as regras respectivas de combinação destes.

Finalmente, a expressão *digital binária* tem a ver com o final do parágrafo anterior, pois em Computação praticamente todos os dispositivos de processamento manipulam informação representada por seqüências finitas de símbolos de um conjunto de dois elementos, classicamente representados pelos *dígitos* 0 e 1, ou seja B={0,1}. A escolha de um alfabeto tão simples foi motivada por questões tecnológicas, econômicas e históricas. Sistemas eletrônicos como processadores e memórias semicondutoras representam 0 e 1 usando níveis de tensão elétrica (ou voltagem), enquanto que sistemas magnéticos como discos e disquetes representam estes valores por orientações magnéticas de aglomerados de partículas metálicas ferrosas. Ainda, sistemas ópticos como discos compactos (CDs) usam propriedades reflexivas de um plástico. E assim

por diante.

Nesta disciplina, ater-nos-emos apenas a sistemas eletrônicos, ignorando outras formas de representação de informação digital binária.

Sistemas eletrônicos podem ser divididos em *sistemas analógicos* e *sistemas digitais*, segundo a forma de manipulação de informação que empregam. Acima, foram feitas referências apenas aos princípios que estão por trás dos sistemas digitais, de interesse maior para Computação. Contudo, alguns princípios básicos de sistemas analógicos são interessantes de explorar aqui. Em particular, os instrumentos a serem estudados neste laboratório servem, sobretudo, para a medida de grandezas elétricas analógicas. A medida de valores digitais associados a estas grandezas é meramente um caso especial da correspondente analógica, em particular no caso da *tensão elétrica*.

Cada um dos dispositivos usados para construir sistemas eletro-eletrônicos (analógicos ou digitais) possui um comportamento característico. Por exemplo, um *resistor* opõe-se à passagem de corrente elétrica, reduzindo a corrente total consumida por um sistema. Algumas grandezas elétricas e as relações entre estas podem ser usadas para definir classes de dispositivos, baseado na forma da função matemática que estes dispositivos definem sobre as grandezas. Tal função matemática é denominada *função resposta* ou *resposta* do dispositivo. As grandezas elétricas mais relevantes são a **tensão** elétrica (ou **voltagem**) e a **corrente elétrica**. Além desta existem a **resistência**, a **capacitância**, e a **indutância** elétrica (de forma mais geral, agrupadas no termo ***impedância***), a **potência** elétrica (o produto entre tensão e corrente), a **freqüência**, o **período** e o **ciclo de serviço** de um sinal elétrico, a **amplitude**, o **nível DC**, etc. Recomenda-se que os alunos acessem as definições das três últimas grandezas constantes ao final desta Seção. De acordo com as relações entre tensão, corrente e impedância, pode-se classificar dispositivos eletro-eletrônicos em *lineares* ou *não-lineares*. Os lineares (resistores, capacitores e indutores) possuem como resposta uma função linear entre tensão e corrente. Os não-lineares (transistores, diodos, e outros) implicam respostas não-lineares. Por exemplo, no caso dos resistores, existe a famosa lei de Ohm (**V=R.I**), que relaciona tensão e corrente. Esta equação define uma reta em um gráfico corrente versus tensão (**IxV**), cuja inclinação é definida pelo valor de resistência do resistor, **R**. Tipicamente, a tensão (medida em Volts, símbolo V) aplicada ao resistor gera uma corrente elétrica **I** (medida em Ampères, símbolo A) que é proporcional ao valor da resistência (medida em Ohms, símbolo Ω). Um resistor ideal típico possui uma resistência constante. Assim, quanto maior a tensão, maior a corrente. A inclinação da reta depende do valor exato de R. Quanto maior o R, menor a corrente, para a mesma tensão, logo a reta aproxima-se de uma reta horizontal. Quanto menor o R, maior a corrente para a mesma tensão, e a reta está mais próxima de uma reta vertical. Este comportamento é ilustrado na Figura 1.

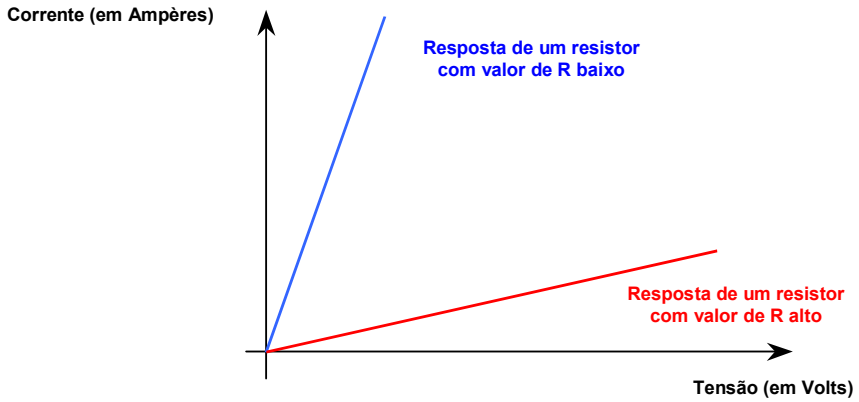


Figura 1 Gráficos IxV típicos para resistores, conforme a Lei de Ohm. Uma reta vertical corresponderia em tese a um resistor de valor 0, ou um curto-circuito. Uma linha horizontal corresponderia em tese a um resistor de valor infinito, ou circuito aberto.

A Figura 2 apresenta uma foto dos três equipamentos a serem estudados neste laboratório, o **multímetro**, o **gerador de ondas** e o **osciloscópio**.

Figura 2 - Equipamentos de excitação e medida de sinais elétricos: multímetro, gerador de ondas e osciloscópio.

Para finalizar esta Seção, é necessário introduzir conceitos relacionados a ondas periódicas de grandezas elétricas que podem ainda não ser de domínio dos alunos, e que são: **ciclo de serviço** (em inglês *duty cycle*), **amplitude** e **nível DC** (DC vem do inglês *Direct Current*, ou Corrente Contínua). A Figura 3 servirá de apoio para introduzir os conceitos e ilustrá-los. Uma onda periódica consiste num padrão de variação de grandeza elétrica que se repete, do ponto de vista conceitual, de forma infinita. Tais ondas podem ser representadas por um gráfico Amplitude *versus* Tempo, conforme ilustrado na Figura 3 para uma onda quadrada. Deve-se notar que a Amplitude refere-se à amplitude de alguma grandeza elétrica, tipicamente tensão, mas não necessariamente.

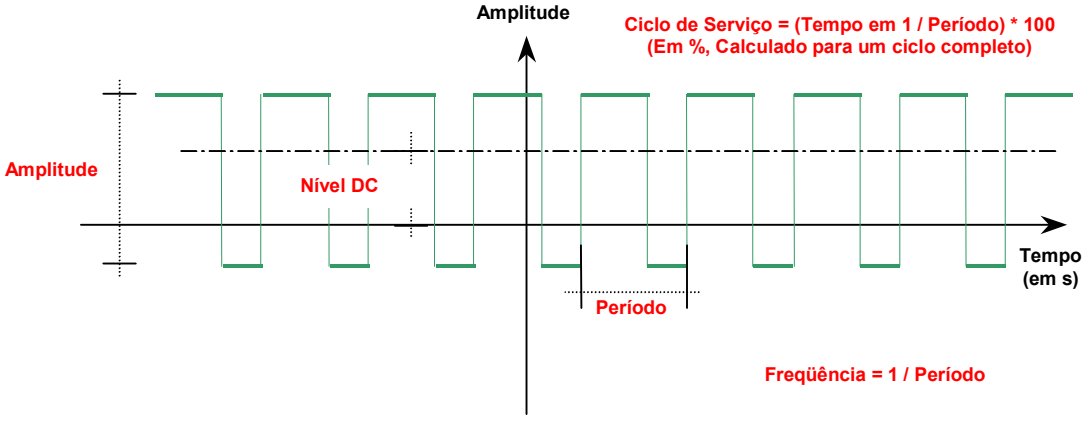


Figura 3 - Gráfico típico Amplitude versus Tempo para forma de onda periódica quadrada, ilustrando as grandezas associadas: freqüência, período, ciclo de serviço, amplitude e nível DC.

As principais características de uma onda periódica são sua freqüência e/ou seu período. Na Figura 3, ilustra-se a definição de período, que é o tempo necessário para que a onda realize um ciclo completo. No exemplo da Figura 3, suponha que a onda ciclicamente permanece na amplitude superior por 2ms (ms é a abreviatura de milisegundo, ou 10^{-3} segundos) e na amplitude inferior por 1ms; isto corresponde então a um período de 2ms+1ms= 3ms. Por outro lado, a freqüência de tal onda seria de (1/3ms)=333,33...Hz (Hz é a abreviatura de Hertz; 1Hz=1 ciclo por segundo). Vejamos então as definições restantes.

Definição: **Ciclo de serviço** em uma onda quadrada[1] é uma medida percentual da relação entre o tempo que a onda permanece na amplitude superior e o tempo que esta permanece na amplitude inferior.

No exemplo da Figura 3 e assumindo os valores estabelecidos no parágrafo anterior, o ciclo de serviço seria (2ms/3ms)*100, ou seja, 66,66...%. Isto indica que a onda, durante cada ciclo, permanece 66,66...% do tempo na amplitude alta e o restante do tempo na amplitude baixa. Note que uma onda simétrica tem um ciclo de serviço de exatamente 50%.

Definição: **Amplitude** de uma onda periódica é o valor absoluto da diferença entre a amplitude máxima e a amplitude mínima que a onda atinge.

No exemplo da Figura 3 suponha que a grandeza cuja amplitude é medida é tensão, e que o valor máximo de tensão atingido pela onda é +3,2Volts e que o valor mínimo de tensão atingido pela onda é -1Volt. Logo, a amplitude da onda seria |+3,2-(-1,0)| = 4,2Volts, também dito 4,2 Volts pico-a-pico (do "pico" superior ao "pico" inferior da onda). Note que a Amplitude de uma onda sempre é um valor positivo.

Definição: **Nível DC** de uma onda periódica é o valor de amplitude de uma onda fictícia de freqüência zero que possui a mesma integral definida ao longo de qualquer número inteiro de ciclos que a onda original.

Embora a definição formal pareça convoluta, a interpretação é simples. Estude a definição gráfica de Nível DC apresentada na Figura 3. Suponha uma onda periódica de ciclo de serviço 50% e simétrica em relação ao eixo do tempo. Como a integral definida mede a área de uma curva qualquer entre dois pontos do eixo das abscissas, se nos limitarmos a um número inteiro de ciclos como intervalo de cálculo da integral definida, todas as áreas positivas (acima do eixo do Tempo) cancelam as áreas negativas e a integral definida será sempre 0. Por outro lado, se o ciclo de serviço não for 50% e/ou se a onda não for simétrica em torno do eixo do Tempo, a integral definida pode não se anular ao longo de um número inteiro de ciclos. Se dividirmos o valor desta integral definida pelo número de ciclos e pelo período da onda, tem-se o valor do nível DC procurado. Voltando ao nosso exemplo, e assumindo os valores de ciclo de serviço e amplitude estabelecidos nos parágrafos anteriores, Teremos um nível DC para a onda da Figura 3 calculado como uma média ponderada pelo ciclo de serviço, ou seja 3,2*66,67%-1*33,33...% = 1,8Volts. Note-se que para uma onda periódica de tensão, o nível DC dá uma medida da energia líquida entregue pela onda ao circuito que recebe a onda.

Medida de Sinais Elétricos – o Multímetro

Para medir sinais elétricos, é necessário ter noção do valor de cada uma das grandezas do sinal a medir. Por exemplo, para medir a tensão elétrica de uma tomada caseira, é necessário ter uma noção de sua ordem de grandeza e de sua natureza. Neste caso particular, é necessário saber que se usa corrente alternada para distribuir eletricidade em casa, e que a tensão atinge normalmente valores acima de 100V. De fato, a tensão na tomada varia de forma senoidal com uma freqüência de 60Hz (60 ciclos de senóide por segundo).

Definição: Um multímetro é um equipamento capaz de medir uma dentre diversas grandezas elétricas a cada instante, donde seu nome. Alternativamente, cada uma das grandezas medidas pelo multímetro pode ser medida por um equipamento específico. No caso de voltagem trata-se de um **voltímetro**, no caso de corrente um **amperímetro**, e assim por diante. O usuário normalmente dispõe de meios de parametrizar o aparelho para selecionar o *tipo de grandeza* a medir, bem como a ordem de valor desta grandeza, denominada de *multiplicador*. As grandezas normalmente passíveis de medida com qualquer multímetro são tensão (Corrente Alternada, CA ou Corrente Contínua, CC), corrente (CA ou CC) e resistência. Outras medidas que podem ser eventualmente medidas são capacitância elétrica, freqüência de um sinal periódico, continuidade elétrica (caso especial de medida de resistência), teste de diodos e transistores, etc.

Os principais elementos do multímetro em questão, o ET-2060 da Minipa, são os seguintes:

- Um *mostrador* de 3 dígitos e 1/2[2], que apresenta a medida atual, bem como alguns indicadores de unidade (apenas para medidas de freqüência, nas grandezas restantes a unidade é apresentada na escolha feita no seletor), ponto decimal (fixo por escala, mas com posição variável de escala para escala), sinal do valor medido, e, quando aplicável, uma indicação de que a medida excede a capacidade de representação (representado pelo texto **OL.** representando **Off-Limits value**);

[1] Definições análogas existem para outras formas de onda como triangular, senoidal, etc.
[2] A notação 3 dígitos e ½ indica que o dígito mais à esquerda pode ser apenas 0 ou 1, logo o valor máximo mostrado no visor será 1999, donde as escalas estarem sempre associadas a limites com o dígito 2 seguido de zeros.

- Um *seletor* para ligar/desligar o aparelho, escolher o tipo de grandeza (no sentido anti-horário a partir da posição desliga, tem-se escalas para medir tensão, corrente, teste de transistor, resistência, capacitância e freqüência) e a precisão da medida. As escolhas dentro de cada grandeza estão marcadas em geral com o valor máximo permitido para a escala; posições especiais são providas para verificação de continuidade (com bip sonoro associado), teste de diodo, nível lógico TTL (0 e 1);
- Uma *chave* de escolha entre CA e CC, marcada com a nomenclatura em inglês, AC e DC, respectivamente (válida apenas para medidas de tensão e corrente);
- Duas linhas de 4 e 6 orifícios para inserção de transistor (cálculo de ganho) e de capacitores (cálculo de capacitância). Ignore esta parte, está fora do escopo desta disciplina;
- Quatro *conectores* para inserção das pontas de prova a conectar o dispositivo a medir ao aparelho. Use apenas os dois conectores mais à esquerda, ignorando os restantes;
- Um par de *pontas de prova*, sendo uma preta (**terra** ou **common**) e uma vermelha.

As medidas elétricas podem ser de dois tipos básicos, *intrínsecas* ou *transitórias*. As intrínsecas são imutáveis dentro de condições normais de funcionamento. Exemplos são a resistência de um resistor fixo, ou a saída de uma fonte de alimentação de um computador. Por exemplo, ao se ligar um computador, se a saída da fonte de alimentação marcada como sendo de 5Volts for medida como 0Volts ou 100Volts, pode-se ter certeza que a fonte está estragada. As transitórias são aquelas que variam ao longo do funcionamento do equipamento. Por exemplo, os pinos de dados de uma memória semicondutora durante a leitura estarão com uma tensão em relação à terra que depende do valor do dado. Tipicamente, os pinos que estiverem com 0Volts indicam um dado "0", enquanto que os pinos que estiverem ao redor de 5Volts indicam um dado "1".

Para informações adicionais, ler o manual do multímetro, disponível no armário do laboratório.

Geração de Sinais Elétricos – o Gerador de Ondas

Os sinais elétricos podem ser gerados de várias formas e mediante emprego de diversos instrumentos. Para geração de sinais, usaremos um gerador de ondas, também chamado de gerador de funções. Existem outros equipamentos para excitação, tais como geradores de padrões, geradores de corrente, ou geradores de tensão fixa, nenhum dos quais será abordado aqui.

Definição: Um gerador de ondas é um equipamento capaz de produzir uma forma de onda de tensão, normalmente periódica, e com determinados valores de grandezas elétricas, tais como freqüência, nível DC, amplitude, ciclo de serviço, simetria do sinal, etc.

Os principais elementos do gerador em questão, o MFG-4201 da Minipa, são os seguintes:

- Um mostrador de 6 dígitos, que apresenta a freqüência da onda sendo gerada, bem como alguns indicadores de unidade;
- Uma linha de 10 botões do tipo *push-button* com duas funções, selecionar o multiplicador de freqüências entre 1 e 1Mega (rótulo azul) e selecionar a forma da onda entre quadrada, triangular e senoidal (rótulo rosa);
- Uma linha de 4 botões do tipo *push-button* com funções especiais (ignore-os, deixando-os sempre não-pressionados);
- Uma linha de 7 potenciômetros para controles diversos da forma de onda gerada. Da esquerda para a direita: ajuste fino da freqüência, dois controles de varredura (ignore-os), ajuste de ciclo de serviço (permite controlar a assimetria do sinal periódico), ajuste de *offset* (nível DC), dois ajustes de amplitude do sinal (um para CMOS/TTL e um para a saída principal);
- Quatro saídas que transportam a onda gerada para conectar-se a algum sistema eletrônico que a usa como entrada: 1 para TTL/CMOS (será a mais usada aqui), uma saída principal de baixa impedância e duas com capacidade de trabalhar em tensões mais altas (ignore-as).

Para informações adicionais, ler o manual do gerador de ondas, disponível no armário do laboratório.

Medidas de Sinais Elétricos – o Osciloscópio

O Multímetro é um aparelho capaz de efetuar a medida de várias grandezas, como visto há duas seções atrás. Contudo, multímetros possuem uma limitação intrínseca, que é o fato de produzirem leituras apenas de

valores instantâneos de uma grandeza, ou, na melhor das hipóteses, computarem a média de um valor variável (o que funciona corretamente apenas para ondas periódicas e apenas para algumas das grandezas associadas a estas). Para medidas mais complexas, onde é necessário *visualizar* o comportamento temporal de uma grandeza elétrica, existe o **osciloscópio**. Osciloscópios são aparelhos de medição muito mais sofisticados que o multímetro. Em geral duas diferenças marcantes se destacam:

- a primeira é a capacidade do osciloscópio efetuar a medida simultânea de mais de uma grandeza, o que permite a comparação entre sinais elétricos (através de múltiplos **canais** de medida). Na maioria dos osciloscópios, o número de canais está limitado a 2;
- a segunda é a noção temporal que o osciloscópio fornece. Enquanto que o osciloscópio apresenta uma janela temporal, onde várias amostras do sinal podem ser observadas, o multímetro apresenta apenas uma amostra a cada instante de tempo. O osciloscópio pode apresentar formas de ondas semelhantes a um diagrama de tempos, e pela composição de mais de um canal pode ser possível detectar se um determinado sinal foi ativado no tempo certo, ou se o período de ativação do sinal é suficiente para que a operação desejada ocorra.

Essencialmente, o multímetro é útil para medir sinais que variam pouco com o tempo, tais como tensão em uma tomada ou a resistência de um material, ou quando a variação é "bem-comportada", ou seja, periódica pura e sem grandes discrepâncias no ciclo de serviço e/ou no nível DC. O osciloscópio, por outro lado, é utilizado para medir sinais que variam constantemente no tempo, tais como relógios (em inglês, *clocks*), ou sinais que variam aleatoriamente, tais como o sinal de habilitação de uma memória para a leitura e escrita.

Como exemplo comparativo, a Figura 2 apresenta um gerador de ondas fornecendo uma freqüência de 7.84 kHz e esta freqüência sendo medida simultaneamente por um osciloscópio e por um multímetro.

Definição: Um osciloscópio é um equipamento capaz de medir simultaneamente formas de onda de tensão (tipicamente), periódicas ou não, e apresentar esta medida usando gráficos de Tensão (em Volts ou múltiplos) versus Tempo (em segundos ou múltiplos/sub-múltiplos) em uma tela. O número mínimo (e típico) de medidas simultâneas que podem ser efetuadas é duas (2).

Outras grandezas, tais como corrente ou potências podem ser facilmente convertidas em tensão, via dispositivos simples, e isto elimina possíveis limitações em relação a multímetros.

Antes de tudo, deve-se perceber que a complexidade de osciloscópios é tal que seria impossível cobrir toda, ou mesmo a maior parte da funcionalidade destes aparelhos em uma aula apenas. Limitar-nos-emos aqui a explorar um mínimo de funcionalidades para permitir o seu uso. Para informações adicionais, ler o manual do osciloscópio, disponível no armário do laboratório.

Os principais elementos do osciloscópio TDS-210 da Tektronix são os seguintes:

- Uma tela de cristal líquido, contendo: 1 área gráfica de 10*8 quadrados de 1 cm de lado como unidades e informações sobre as ondas sendo mostradas; entre estas informações destacam-se 2 escalas de tensão (abaixo da área gráfica) a escala de tempo (abaixo da área gráfica, ao centro), a informação de onda estável (em inglês, *triggered*, significando gatilhada) (acima da área gráfica, ao centro, uma letra T branca em fundo escuro) e um menu para seleção de parâmetros do aparelho e das ondas atualmente medidas;
- Duas linhas botões para seleção de menus a aparecerem no lado direito da tela (ignore-os);
- Um botão de **Autoset**, útil para tentar parametrizar automaticamente uma onda sendo medida sem grandes complicações (muito, muito útil para aprendizes);
- Um botão de **Hardcopy** (ignore-o);
- Um botão de **Run/Stop**, usado para "congelar" a leitura do aparelho, memorizando o último trecho medido (ignore-o, por enquanto);
- Uma coluna de botões de acesso a opções de menu (cinco botões mais à esquerda, abaixo) (ignore-os, por enquanto);
- Duas colunas de controles verticais dos canais (CH1 e CH2) cada uma com dois potenciômetros e um botão de acesso ao menu do canal correspondente. Os potenciômetros servem para controlar a posição vertical da onda (botão menor) e para controlar a escala vertical do canal respectivo;
- Uma coluna de controle horizontal (afeta simultaneamente os dois canais), com dois potenciômetros e

um botão de acesso ao menu de controles horizontais. Os potenciômetros servem para controlar a posição horizontal da janela de medida (botão menor). Como a medida feita não cabe inteira na tela, pode-se mover a janela para a esquerda e para a direita (permitindo observar uma faixa de tempo maior que apenas uma tela), e controlar a escala horizontal da tela ("encolhendo" ou "espichando" a medida para caber mais ou menos tempo na tela, respectivamente);

- Uma coluna de controles de gatilho (em inglês, *trigger*) (ignore-os, por enquanto);
- Uma ponta de prova (identificada pelo texto Probe Comp no canto inferior esquerdo do painel direito), com uma onda padrão (quadrada, 1KHz, 5V pico-a-pico, ciclo de serviço 50%), útil para calibrar a medida do aparelho;
- Três entradas para as 2 ondas a medir (duas entradas com rótulos CH 1 e CH 2) e para uma onda de gatilho externo (ignore a entrada de gatilho).

1.3 A Fazer e a Entregar

Tarefa 1: (após ler Seção 1.2) Identifique na Figura 2 o multímetro. Se já não estiver montado, monte o multímetro na sua bancada, e estude seu painel.

Tarefa 2: (após ler Seção 1.2) Para testar sua compreensão do funcionamento do aparelho, use-o para medir a tensão de uma tomada comum. Faça o seguinte: 1) parametrize o aparelho para medir os teóricos 110 Volts da tomada (não se esqueça, em Corrente Alternada, CA); 2) insira as pontas de prova nos conectores adequados (leia as legendas!); 3) Insira a outra extremidade das pontas de prova na tomada; 4) Leia o valor de tensão obtido. Anote o valor exato de sua leitura.

Tarefa 3: (após ler Seção 1.2) Vamos agora realizar a medida de resistências. Tome uma plataforma de prototipação XS40/XST-1 do armário do laboratório. A medida de resistência é do tipo intrínseca, podendo ser realizada sem alimentar o sistema. Assim, identifique os resistores desta plataforma. Fisicamente, estes se assemelham a cilindros de cor bege ou cinza claro com dois terminais. No corpo do cilindro existem 4 anéis coloridos que consistem numa especificação do valor nominal do resistor. Para saber mais, estude o texto contendo o Código de Cores de Resistores na página da disciplina (**cod_cores_res.html**). A partir destas informações determine os valores nominais de 10 resistores da placa de extensão XST-01. A seguir, meça com o multímetro o valor exato de cada resistor, com a máxima precisão possível. Faça uma tabela listando a identificação, os valores nominais e os valores medidos dos resistores.

Tarefa 4: (após ler Seção 1.2) Realize a medida da freqüência de oscilação do cristal da placa XS40. Faça o seguinte: 1) parametrize o aparelho para medir freqüência, sabendo que a amplitude da onda está em torno de 5Volts em Corrente Contínua (CC); 2) identifique o cristal na placa XS40, ele encontra-se próximo do conector da porta paralela, sendo um dispositivo de 3 pinos de coloração marrom-claro, com um código X1 impresso ao seu lado na placa de circuito impresso; 3) Teste as três combinações de dois pinos com as pontas de prova e leia o valor de freqüência obtido (Não se esqueça de prestar atenção à unidade, Hz (Hertz), precedida eventualmente de um multiplicador (K ou M, indicando 10^3 ou 10^6)). Anote o valor exato de sua leitura.

Tarefa 5: (após ler Seção 1.2) Identifique na Figura 2 o gerador de ondas. Se já não estiver montado, monte o gerador de formas de onda na sua bancada, ligue-o (botão Power) e estude seu painel.

Tarefa 6: (após ler Seção 1.2) Para testar a compreensão do funcionamento do aparelho, use-o para gerar uma onda específica. A Tabela 1 descreve características de 10 ondas, uma para cada grupo de alunos do laboratório. O professor atribuirá uma tarefa para cada grupo de alunos durante a aula. Gere a onda atribuída ao seu grupo e meça as características da mesma. A freqüência pode ser medida com o multímetro, mas as medidas restantes (Amplitude e Forma da onda) somente podem ser verificadas com um osciloscópio. Em todas as ondas, assuma um ciclo de serviço de 50%. O nível DC pode ser medido indiretamente com o multímetro (fora do escopo deste laboratório).

Tabela 1 - Especificação de ondas a gerar para a Tarefa 6.

Grandeza	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
Freqüência (KHz)	35.5	46	1560	2,5	234	13,6	0,5	670	320	6
Amplitude (V)	3,5	5	10	7	8	6	4	1,5	13	7,5
Nível DC (V)	3	0	-2	-6	0,5	12	-6,5	-3,5	0	5
Forma	Quadrada	Senóide	Triangular	Quadrada	Senóide	Triangular	Quadrada	Senóide	Triangular	Quadrada

Tarefa 7: (após ler Seção 1.2) Identifique na Figura 2 o osciloscópio. Se já não estiver montado, monte o osciloscópio na sua bancada, ligue-o (botão Power, acima do painel) e estude seu painel.

Tarefa 8: (após ler Seção 1.2) Para testar a compreensão do funcionamento do aparelho, use-o para medir a onda gerada como parte da tarefa 6. Para tanto, escolha um canal e conecte as suas duas extremidades às pontas da saída respectivas do gerador de ondas, usando a ponta de prova adequada do osciloscópio. Conecte uma segunda ponta de prova do osciloscópio à onda padrão, interna ao osciloscópio. Após tudo conectado, aperte a tecla *Autoset*, o que deve fazer uma tela estável aparecer no osciloscópio. A partir daí, mude os parâmetros de visualização horizontal e vertical (de cada um dos canais) e verifique o efeito na tela. Como se pode demonstrar que a onda gerada pelo grupo está correta a partir das medidas que aparecem na tela do osciloscópio? Feitas as medidas e após a onda gerada corretamente aparecer na tela do osciloscópio, varie os parâmetros do gerador de ondas e observe o efeito no osciloscópio, mudando, pelo menos, a freqüência, a amplitude, e o tipo de onda.

2 Implementação de Sistemas Digitais com HDLs - Ferramentas de Captura e Validação

Prática: Implementação de um Circuito Digital Combinacional e sua simulação
Recursos: Ambiente de Desenvolvimento Active-HDL da Aldec, Inc.

2.1 Introdução e Objetivos

O laboratório anterior referia-se à Unidade I da disciplina. Ele serviu para dar embasamento no uso dos equipamentos de medida e geração de grandezas elétricas. A partir do presente laboratório, inicia-se a Unidade II da disciplina, onde o foco passa a ser uma forma abstrata e moderna de desenvolver sistemas digitais, aquela baseada no emprego de Linguagens de Descrição de Hardware (em inglês, *Hardware Description Languages*, donde deriva o acrônimo HDL).

Os objetivos específicos deste Laboratório são iniciar a utilização do ambiente de simulação Active-HDL, disponível para uso na disciplina, e realizar a simulação de um circuito combinacional exemplo, expresso por uma tabela verdade.

2.2 Localização do Ambiente de Desenvolvimento Active-HDL

Verificar se o simulador está instalado. Normalmente, há o ícone do "Active-HDL 6.2" no desktop. Ver exemplo da aparência do ícone abaixo. Se o ícone não estiver disponível no seu ambiente de trabalho, verifique em **C:\Program Files\Aldec\Active-HDL 6.2\BIN\avhdl.exe**, ou tente localizá-lo via menu **Start** do Windows. Lance o programa. Se aparecer uma janela de diálogo solicitando a abertura de um espaço de trabalho, cancele esta. Se for acusada inexistência de licença disponível, consulte o professor.


Active-HDL 4.2.lnk

2.3 Início do Projeto

Crie um novo projeto, seguindo os passos descritos abaixo. Primeiro, deve-se criar um espaço de trabalho para conter todos os projetos de hardware relacionados. Este é denominado no ambiente Active-HDL de "*workspace*". Para criá-lo, escolha a opção de menu **File→New→Workspace**. Escolha um nome para o espaço de trabalho e escolha o diretório onde criar, de preferência fora do diretório de instalação do software e de preferência em um disco local ao computador. A seguir clique OK. Ver exemplo na Figura 4.

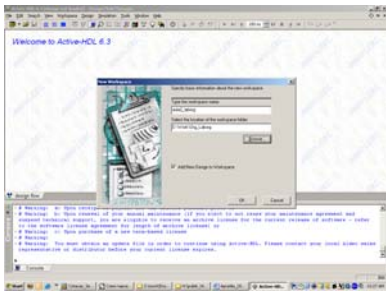
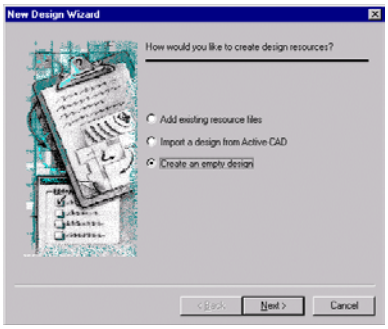
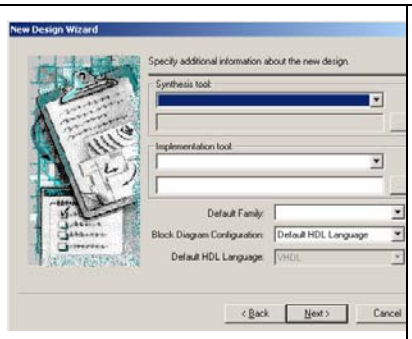


Figura 4 - Criação de um espaço de trabalho (*workspace*).

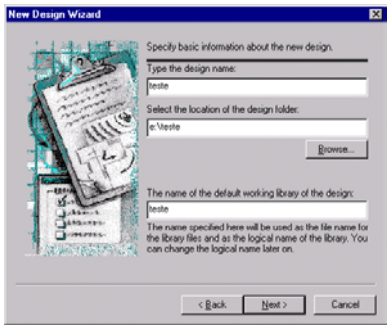
A partir da janela de diálogo a seguir siga os passos descritos na Figura 5.



Crie um projeto vazio.



Janelas de parâmetros – não é importante agora.



Defina o nome do projeto e o diretório onde este será armazenado (Use algum diretório no disco local C: de preferência, ou seu diretório H:/).

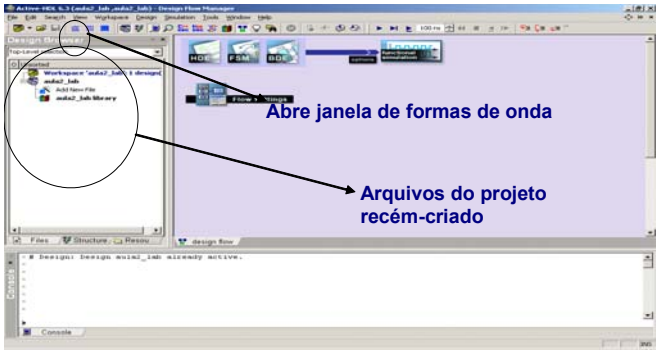


Figura 5 - Criação de um novo projeto VHDL.

2.4 Implementação de um Circuito Combinacional em VHDL

Os circuitos combinacionais mais simples são o inversor e as portas lógicas de duas entradas. A interconexão de portas lógicas usando fios permite criar circuitos combinacionais e seqüenciais. Se não houver laços de realimentação, nem elementos bidirecionais tais como *buffers tristate*, o circuito resultante é um circuito combinacional que pode ser descrito por uma tabela verdade. Aqui, ver-se-á como implementar uma tabela verdade usando a linguagem VHDL. A tabela verdade exemplo é dada na Tabela 2.

Tabela 2 - Tabela verdade do sistema digital a implementar.

Ent_A	Ent_B	Ent_C	Saída
0	0	0	01
0	0	1	01
0	1	0	10
0	1	1	11
1	0	0	11
1	0	1	10
1	1	0	00
1	1	1	10

A tabela acima especifica um circuito com 3 entradas binárias e duas saídas binárias.

Criação do arquivo VHDL Inicial

A criação do VHDL pode ser auxiliada por um "wizard" do ambiente Active-HDL conforme descrito abaixo. O projeto deve ter como entradas os sinais **Ent_A**, **Ent_B**, e **Ent_C**, e como saída um vetor **Saida** de 2 bits. Escolha a opção de menu **File→New→VHDL Source** ou clique no ícone **Add New File** do seu projeto. Escolha a aba "**Wizard**" da janela de diálogo, escolha o tipo de arquivo a criar ("**VHDL Source Code**"), e siga os seguintes passos para a criação do arquivo:

1. Verifique se a opção "*Add the generated file to the design*" está selecionada e escolha "*next*".
2. Indique o nome do arquivo, por exemplo "tab3e2s". O mesmo nome pode ser usado nos três campos desta janela. Em seguida clique no botão "*next*".
3. Inicia-se agora o procedimento de inserção de pinos de entrada e saída do projeto. Insira os pinos "Ent_A", "Ent_B", "Ent_C", e "Saida" conforme a descrição que segue, o que deve resultar na Figura 6:
    - "new", "name" Ent_A, direção "in";
    - "new", "name" Ent_B direção "in";
    - "new", "name" Ent_C, direção "in";
    - "new", "name" Saida, direção "out", 2 bits (índices 1 a 0).

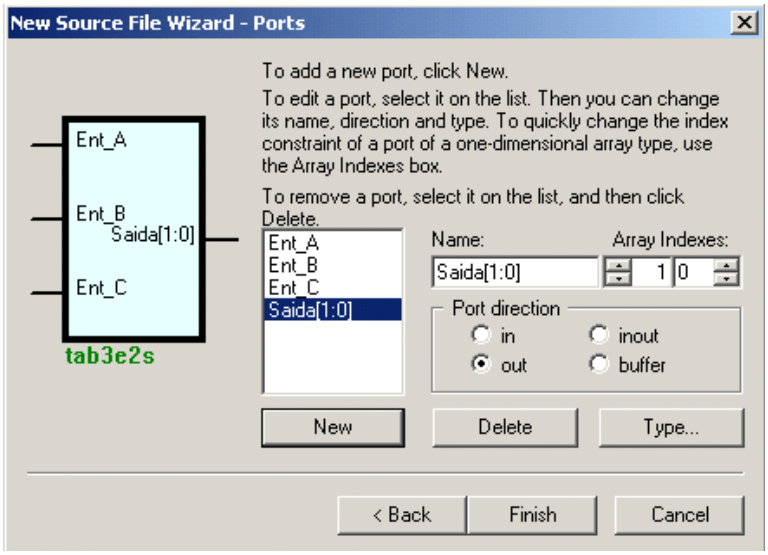


Figura 6 - Janela de inserção de pinos de uma descrição de hardware em VHDL.

Ao final da inserção dos pinos (clicando no botão *Finish*) é aberta a janela mostrando o conteúdo do arquivo VHDL gerado pelo "Wizard". Observar na Figura 7 que o arquivo criado já contém referência à biblioteca IEEE e ao *package* pertinente, onde os tipos *std_logic* e *std_logic_vector*, usados neste projeto, estão definidos. Um *package* VHDL é equivalente a um arquivo header (arquivos .h) de uma linguagem de programação tal como Java ou C, contendo definições úteis para um conjunto de diferentes programas.

A descrição VHDL obtida acima contém a definição das entradas e saídas do circuito (as declarações de nomes dentro da construção *port*, parte do bloco VHDL denominado *entity* ou entidade, em Português). O Bloco seguinte, denominado *architecture* (arquitetura, em Português) e associado à *entity* mencionada supra, conterá a descrição da estrutura e/ou comportamento do circuito (no nosso caso a tabela verdade da Tabela 2).

Deve-se observar as estruturas sintáticas de declaração do módulo de hardware, a relação entre a entidade ou interface (pinagem) do circuito e a arquitetura (descrição do comportamento e/ou estrutura do mesmo), a notação usada para introduzir comentários, entre outras características da descrição de hardware.

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

entity tab3e2s is
     port(
       Ent_A : in STD_LOGIC;
       Ent_B : in STD_LOGIC;
       Ent_C : in STD_LOGIC;
       Saida : out STD_LOGIC_VECTOR(1 downto 0)
           );
end tab3e2s;

--}} End of automatically maintained section

architecture tab3e2s of tab3e2s is
begin

       -- enter your statements here --

end tab3e2s;
```

Figura 7 - Arquivo aberto após a inserção dos pinos.

Definição da Arquitetura do Módulo

Os passos anteriores implementaram uma versão inicial do VHDL, que pode automaticamente ser gerado a partir da descrição das entradas e saídas do circuito, gerando um cabeçalho, e um par entidade-arquitetura (declarações *entity* e *architecture*). A entidade define a interface externa do circuito, enquanto que a arquitetura define a funcionalidade e/ou a estrutura de implementação do mesmo. Isto funciona de forma à classes em uma linguagem de programação orientada a objetos, onde esta é representada por uma interface de acesso (pública) e uma implementação (privada).

A definição da arquitetura é tarefa normalmente reservada normalmente ao projetista. No caso, basta substituir o comentário (`-- enter your statements here --`) por um texto VHDL que descreva a tabela verdade da Tabela 2 - Tabela verdade do sistema digital a implementar.

- . Para tanto, basta usar o comando VHDL `with`, que permite implementar qualquer tabela verdade simples. A Figura 8 mostra o texto que corresponde ao comando desejado.

```
with (Ent_A & Ent_B & Ent_C) select
      Saida <= "01" when "000",
               "01" when "001",
               "10" when "010",
               "11" when "011",
               "11" when "100",
               "10" when "101",
               "00" when "110",
               "10" when others;
```

Figura 8 - Texto VHDL descrevendo a tabela verdade da Tabela 2 - Tabela verdade do sistema digital a implementar.

.

Alguns comentários sobre a estrutura sintática do comando `with` são dadas a seguir, com o intuito de esclarecer seu uso. Primeiro, trata-se de um comando de atribuição condicional a um sinal. No caso, o sinal que recebe a atribuição é a saída do circuito, denominada `Saída`. Segundo, os valores atribuídos são um a um associados aos valores de entrada que lhe correspondem, de acordo com a especificação constante na tabela verdade, correspondendo a cada uma das 8 últimas linhas do comando da Figura 8. Terceiro, a

expressão entre parênteses estabelece uma estrutura a ser testada para realizar a atribuição condicional. No caso, esta estrutura corresponde à concatenação das três entradas da tabela verdade usando o operador VHDL `&`, de concatenação. Assim cada string após a palavra chave `when` é um vetor de três bits, onde o primeiro bit provém do valor da entrada `Ent_A`, o segundo do valor da entrada `Ent_B` e o terceiro o valor da entrada `Ent_C`. Finalmente, note-se o uso da expressão `when others`, para especificar um valor para todos os valores da expressão de avaliação que não sejam explicitamente mencionados como alternativas no comando `with`. No caso, esta cláusula corresponde à última linha da tabela verdade.

Para completar a descrição do hardware VHDL deste laboratório, basta inserir o texto VHDL da Figura 8 na descrição inicial gerada pelo *VHDL Source File Wizard*.

Definição do ambiente de validação do módulo (testbench)

Uma vez concluída a implementação do módulo de hardware é necessário definir um procedimento para validar este módulo. A linguagem VHDL possui alguns recursos para facilitar a definição de tal procedimento. Utiliza-se o termo *testbench* para referir-se a um ambiente que permita realizar este procedimento, que pode ser completamente descrito na própria linguagem VHDL. Obrigatoriamente, um *testbench* contém (a) uma instância do módulo de hardware a ser validado, (b) uma descrição de geradores de estímulos de entrada deste, (c) estruturas para capturar as saídas geradas pelo módulo de hardware a ser avaliado. Opcionalmente, *testbenches* contêm (a) repositórios de estímulos a aplicar no módulo de hardware, e (b) repositórios de resultados esperados e (c) repositórios de saídas geradas pelo módulo como resposta aos estímulos aplicados. Além disto, é possível definir um *testbench* onde resultados esperados possam ser automaticamente comparados a resultados obtidos. A Figura 9 ilustra a estrutura genérica de um *testbench*.

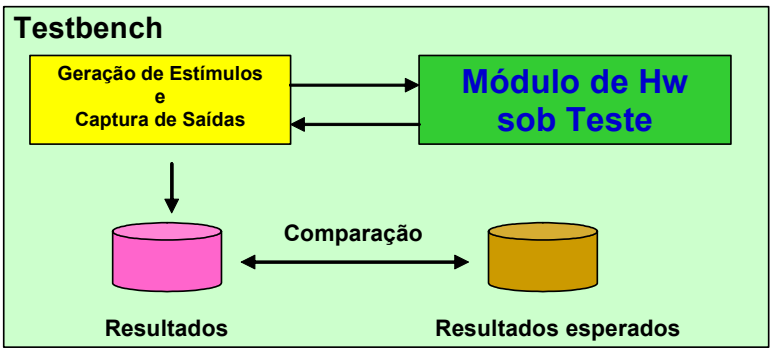


Figura 9 – Estrutura genérica de um *testbench* VHDL. Nesta estrutura, o repositório de estímulos é parte do módulo denominado Geração de estímulos e Captura de Saídas.

Aqui será mostrado como construir um *testbench* o mais simples possível para testar de forma exaustiva o hardware implementado. Para tanto o testbench deve conter (a) uma instância do módulo `tab3e2s`, (b) três geradores de estímulos (para as entradas `Ent_A`, `Ent_B` e `Ent_C`) e um capturador de saídas (para receber os valores produzidos no sinal `Saida` pelo módulo implementado).

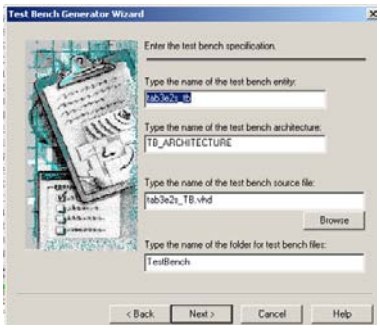
O procedimento de criação de testbenches pode ser simplificado pelo uso de um recurso do ambiente Active-HDL denominado *Test Bench Generator Wizard*. Vejamos como utilizá-lo, acompanhando o processo na Figura 10. Escolha a opção de menu **Tools→Generate Testbench** e siga os seguintes passos para a criação do arquivo:

1. Como só existe um par entidade-arquitetura no projeto não é necessário escolher estes. Apenas clique no botão "N*ext* >".
2. A janela seguinte serve para escolher um repositório de estímulos, que não será usado aqui (os estímulos serão criados diretamente dentro do texto VHDL). Logo, apenas clique no botão "N*ext* >".
3. A janela seguinte serve para definir nomes para os arquivos de *testbench*. Apenas aceite todas as sugestões e clique no botão "N*ext* >".
4. Na janela final, apenas clique no botão "Finish".

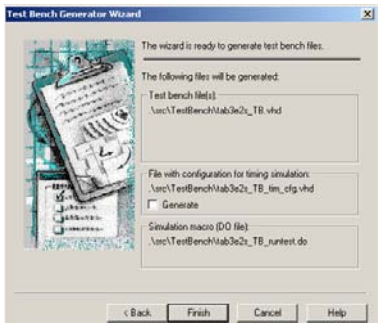
O resultado do processo é a criação de um diretório contendo dois arquivos (veja a localização deste na

janela Design Browser do ambiente Active-HDL, mostrada na Figura 10.

Primeira janela o Test Bench Generator Wizard. Se nada aparecer, clique Cancel, compile o arquivo VHDL com F11, voltando aqui depois.



Terceira janela, com sugestões de nomes de arquivos que compõem o testbench.



Janela final do Wizard.

Figura 10 - Criação inicial de arquivos que compõem um *testbench* básico, gerados pelo Test Bench Generator Wizard do ambiente Active-HDL.

Abra o diretório no Design Browser (use, por exemplo, duplo clique do mouse sobre o ícone do diretório) e perceba a existência neste de dois arquivos. O primeiro, tab3e2s_TB.vhd, contém a descrição ainda incompleta do *testbench*, enquanto o segundo, tab3e2s_TB_runtest.do é um arquivos contendo comandos para executar o *testbench*. Estes comandos usando a linguagem de scripts Tcl, enquanto o primeiro arquivo é uma descrição VHDL. Observe o conteúdo de ambos arquivos. O conteúdo do arquivo VHDL é reproduzido na Figura 11. A seguir, explicam-se as principais estruturas deste arquivo, aproveitando para introduzir novos comandos VHDL. Atente para os comentários gerados automaticamente, indicando pontos do texto VHDL onde o projetista pode ou deve acrescentar código VHDL adicional (comentários em verde).

As primeiras duas linhas correspondem à declaração de biblioteca e packages, como já apresentado na discussão da criação de qualquer módulo de hardware VHDL. As duas linhas seguintes contendo texto declaram a entidade do *testbench*. Note-se a característica de um *testbench* ser um sistema fechado, sem entradas nem saídas, ao contrário de qualquer módulo de hardware. Isto ocorre porque o *testbench* contém um sistema completo que gera os estímulos, aplica estes estímulos às entradas do hardware projetado e captura as saídas dos mesmos. Assim, nada entra e nada sai de um *testbench*. Todos os fios (signals) e pinos (ports) são internos ao *testbench*.

```
library ieee;
use ieee.std_logic_1164.all;
    -- Add your library and packages declaration here ...

entity tab3e2s_tb is
end tab3e2s_tb;

architecture TB_ARCHITECTURE of tab3e2s_tb is
    -- Component declaration of the tested unit
    component tab3e2s
    port(
           Ent_A : in std_logic;
           Ent_B : in std_logic;
           Ent_C : in std_logic;
           Saida : out std_logic_vector(1 downto 0) );
    end component;
    -- Stimulus signals - signals mapped to the input and inout ports of tested entity
    signal Ent_A : std_logic;
    signal Ent_B : std_logic;
    signal Ent_C : std_logic;
    -- Observed signals - signals mapped to the output ports of tested entity
    signal Saida : std_logic_vector(1 downto 0);
    -- Add your code here ...
begin
    -- Unit Under Test port map
    UUT : tab3e2s
           port map (
                  Ent_A => Ent_A,
                  Ent_B => Ent_B,
                  Ent_C => Ent_C,
                  Saida => Saida
           );
    -- Add your stimulus here ...
end TB_ARCHITECTURE;
```

Figura 11 – Parte do texto VHDL descrevendo o *testbench* parcialmente gerado de forma automática pelo Test Bench Generator Wizard do ambiente Active-HDL.

Após a entidade do testbench, segue-se a descrição da arquitetura do mesmo. Esta é composta de três partes principais: (a) a declaração da interface do módulo de hardware a ser testado, (entre as linhas iniciando com `component` e `end component`); (b) a declaração de sinais (fios) do testbench, onde serão conectados os gerados e capturadores de estímulos; e (c) o corpo da arquitetura (entre as linhas iniciando com `begin` e `end`), que contém por enquanto apenas a instanciação do módulo de hardware e sua conexão aos sinais do testbench. Finalmente existe um texto VHDL não mostrado na Figura 11, correspondendo ao comando configuration. Este comando somente é relevante quando mais de uma arquitetura existe associada a uma dada entidade, caso que jamais será usado nesta disciplina. Assim, este comando poderá sempre ser ignorado, e até mesmo seu texto pode ser eliminado do arquivo de *testbench* sem nenhuma conseqüência. Contudo, caso se remova esta linha deve-se editar o script Tcl, removendo a linha que inicia pelo comando `asim`.

Note que o *Wizard* que gerou este *testbench* foi capaz de, a partir do nome do par entidade-arquitetura a testar, encontrar a definição da interface do módulo e criar uma declaração compatível com este no *testbench* (usando o comando `component`). Ele também conseguiu gerar sinais de mesmos nomes que os dos pinos do módulo a testar (inserindo-os na arquitetura do *testbench*), instanciar o módulo de hardware a testar e conectá-lo (via o comando VHDL `port map`) aos sinais (fios) do *testbench*.

Resta ao projetista apenas especificar os estímulos a aplicar no módulo de hardware para completar o *testbench*. Isto é feito através do uso de um comando de atribuição aos sinais de entrada do módulo projetado, ou seja, `Ent_A, Ent_B e Ent_C`. Estas atribuições podem ser feitas de diversas maneiras. Lembra-se que se deseja testar exaustivamente o hardware. Como se trata de um circuito combinacional especificado por uma tabela verdade, seu teste exaustivo é relativamente simples, bastando reproduzir cada uma das condições de entrada possíveis, ou seja, uma combinação para cada linha da tabela verdade. Tal comando pode ser inserido em qualquer lugar do corpo da arquitetura, tipicamente sendo feito após o comentário `-- Add your stimulus here ...` do arquivo de *testbench*. Um exemplo de comandos que podem ser usados é mostrado na Figura 12. Deve-se notar que uma única atribuição pode especificar diversos valores que um sinal pode assumir ao longo do tempo, através do uso de valores a serem atribuídos separados por vírgulas e mediante uso da cláusula `after`, que implica passagem do tempo. O primeiro valor não possui cláusula `after` associada, logo vale a partir do inicio da simulação, este sempre considerado

como sendo o tempo (instante) 0 da execução. Note-se o uso da unidade ns, abreviatura de nanossegundo, ou 1 bilionésima parte de 1 segundo, uma unidade adequada para especificar tempo em circuitos digitais de alta velocidade, operando em freqüências de relógio na ordem de MegaHertz ou GigaHertz. Uma observação importante é que os tempos mencionados após a cláusula after são todos relativos ao momento de avaliação do comando e não relativos a uma cláusula `after` anterior.

```
Ent_A <= '0' , '1' after 40ns;
Ent_B <= '0' , '1' after 20ns, '0' after 40ns, '1' after 60ns;
Ent_C <= '0' , '1' after 10ns, '0' after 20ns, '1' after 30ns,
         '0' after 40ns, '1' after 50ns, '0' after 60ns, '1' after 70ns;
```

Figura 12 – Código VHDL especificando os estímulos de entrada para o circuito implementado. Este código aplica cada um dos valores da tabela verdade completa ao circuito, gerando uma simulação exaustiva.

Deixa-se aos alunos a tarefa de verificar que estes três comandos de atribuição geram cada uma das linhas da tabela verdade em ordem, a aplicação de cada linha durando exatamente 10ns, exceto o último valor que permanece em "111" do instante de tempo 70ns após o início da simulação até o final desta, seja este momento qual for.

2.5 Compilação e Simulação

Uma vez descrito o *testbench*, compila-se a descrição até não haverem mais erros. Use a opção Menu Design → Compile All, ou a tecla de atalho F11. Se o texto gerado for exatamente como descrito acima a compilação terá sucesso. Senão, mensagens de erro aparecem na janela de interação e devem ser usadas para corrigir os eventuais erros.

Em seguida, deve-se realizar a simulação. Uma forma de fazer isto é:

1. Execute o script Tcl criado pelo Test Bench Generator Wizard. Isto pode ser realizado da seguinte maneira: (a) selecionar a entidade a ser simulada, indo na barra de rolagem logo abaixo do título de janela Design Browser, escolhendo a entidade do *testbench* (**tab3e2s_tb(tb_architecture)**); (b) clicar com o botão direito do mouse no ícone do arquivo tab3e2s_TB_runtest.do, escolhendo a opção Execute. A conseqüência desta execução é a abertura de uma janela de simulação, contendo todos os sinais da interface externa do circuito a simular (`Ent_A`, `Ent_B`, `Ent_C` e `Saída`), um por linha da janela
2. Para realizar a simulação, escolha a opção de menu *Simulation* → *Run For*, ou a tecla de atalho F5, ou o ícone imediatamente à direita da janela que contém um número e uma unidade de tempo na barra de ícones mais perto do topo da janela do Active-HDL. Normalmente, esta janela deve conter o valor 100ns. Se este não for o caso, escreva 100ns nela e só então clique no botão à esquerda.

Verifique se a simulação está funcionando corretamente, e relate as observações no relatório deste laboratório. A Figura 13 ilustra como deve aparecer o resultado correto da simulação.

Algumas observações ajudam a compreender os resultados gráficos. A resposta da simulação está apresentada com um diagrama de tempos, onde se emprega uma escala de nanossegundos no eixo das abscissas. No eixo das ordenadas, apresentam-se os sinais do *testbench* sendo as entradas de 1 bit apresentadas com valores 0 e 1 (respectivamente linhas abaixo e acima do trecho reservado para cada sinal) e a saída de 2 bits apresentada com valores numéricos entre 0 e 3 e indicação do momento de transição para valores diferentes via cruzamento de linhas.

Verifique que este comportamento corresponde ao da tabela verdade. Em seguida, realize algumas alterações na descrição e no *testbench* re-compilando e re-simulando o *testbench* resultante. Exemplos de alterações simples são:

- Simular por mais tempo o testbench;
- Alterar os valores de saída gerados pela tabela verdade;
- Transformar as entradas em um vetor;
- Alterar o número de entradas e de saídas da tabela verdade.

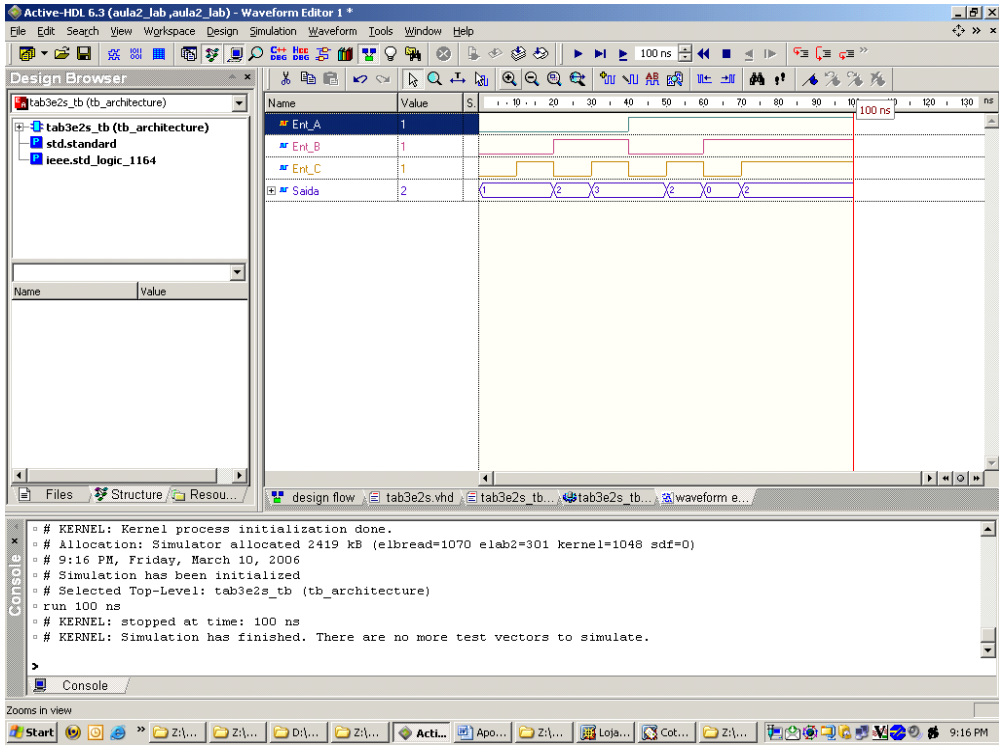


Figura 13 – Resultado da simulação do *testbench* para o circuito tab3e2s.

2.6 A Fazer e a Entregar

Tarefa 1: Leia o texto acima e siga as orientações para criar o circuito correspondente à tabela verdade dada, criar o *testbench* sugerido e gerar as simulações mencionadas.

Tarefa 2: Estude e experimente com o ambiente de edição/simulação. Observe a grande quantidade de material de referência disponível junto com o ambiente, por exemplo, a partir das opções de menu **Help→Online Documentation** e **Help→Interactive VHDL Tutorial**. Outra fonte importante de referência é o Assistente da Linguagem, acessível a partir da opção de menu **Tools→Language Assistant**, que permite acessar uma imensa quantidade de exemplos e templates para as linguagens suportadas pelo ambiente, tanto as de descrição de hardware como de simulação e de scripts.

Tarefa 3: Prepare um relatório descrevendo o aprendido neste laboratório, os experimentos livres realizados e eventual obtenção de informações extras, tais como sites pesquisados na Internet, etc.

2.7 Apêndice – Visão Geral do Ambiente de Desenvolvimento Active-HDL

A Figura 14 apresenta uma visão geral do ambiente de desenvolvimento da ferramenta Active-HDL. Alguns itens importantes estão comentados em azul.

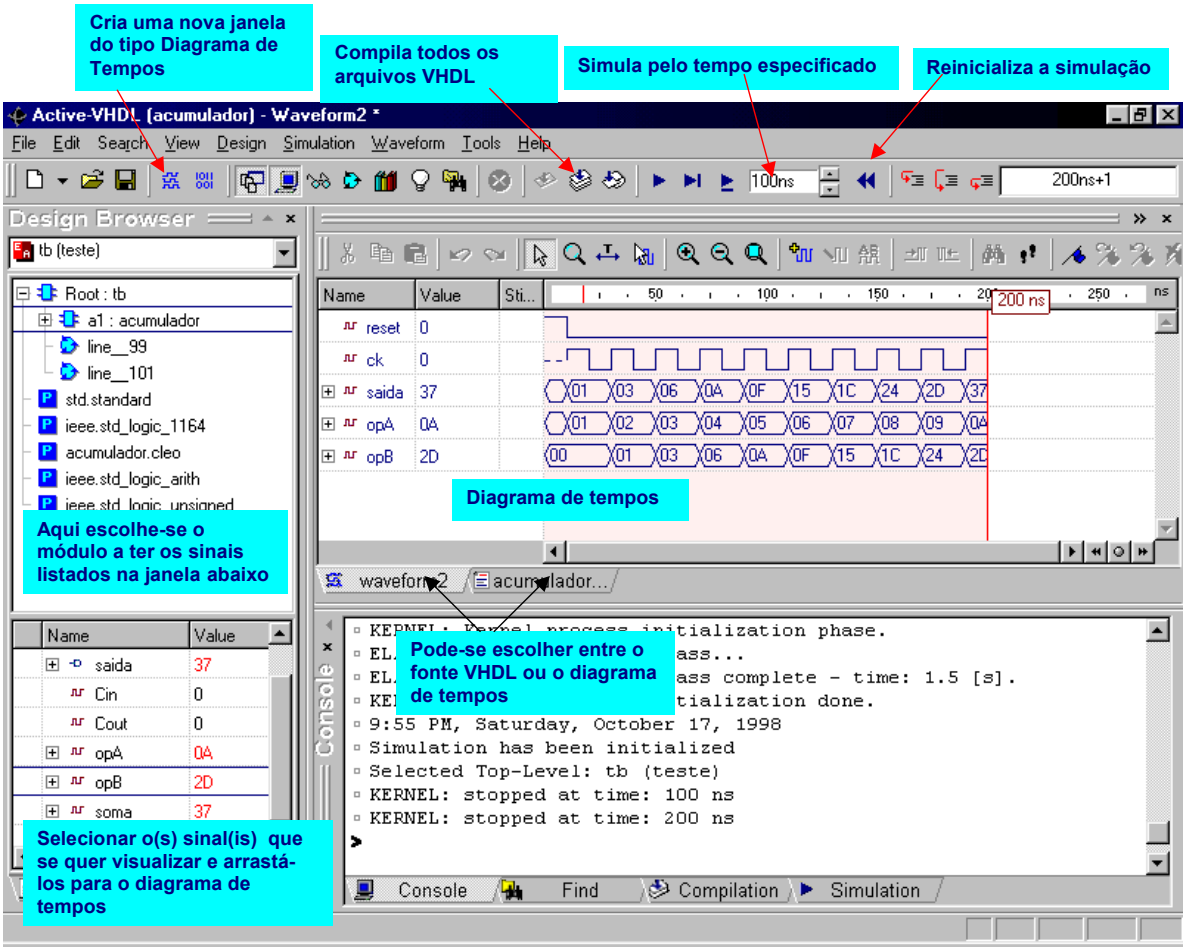


Figura 14 - Visão geral do ambiente de desenvolvimento Active-HDL.

Para melhorar a compreensão do tema, aconselha-se a realização, fora do horário de aula, de alguns tutoriais disponíveis a partir do ambiente Active-HDL, bem como a análise de documentações específicas. Exemplos importantes são:

- O tutorial Evita, acessível a partir da opção de menu Help → Interactive VHDL Tutorial do ambiente Active-HDL. Este tutorial também está disponível na área de *download* da disciplina;
- Os tutoriais HDL Entry and Simulation Tutorial e VHDL Testbench Tutorial, disponíveis a partir da opção de menu Help → On-Line Documentation;
- Exemplos selecionados do material denominado Application Notes e Package References, disponíveis a partir do mesmo local que os tutoriais supracitados.

3 Implementação de Sistemas Digitais com HDLs Captura, Síntese, Implementação e Teste

Prática: Implementação de um Circuito Digital Combinacional e sua simulação
Recursos: Ambiente de Desenvolvimento ISE da Xilinx, Inc. e Plataforma Digilent D2SB/DIO4

3.1 Introdução e Objetivos

O Laboratório 2 serviu para introduzir conceitos básicos sobre a utilização do ambiente de simulação Active-HDL, da empresa Aldec, bem como para realizar a simulação de um circuito combinacional exemplo.

O objetivo específico do presente Laboratório será o de investigar um fluxo de implementação de circuitos digitais em hardware reconfigurável, a partir de uma descrição inicial em linguagem de descrição de hardware (a mesma do Laboratório 2). Isto será feito mediante a síntese de hardware usando o ambiente ISE da Xilinx. Após sintetizado, o circuito deverá ser configurado e testado na plataforma de prototipação educacional formada pelas placas D2SB e DIO4 da empresa Digilent (www.digilentinc.com), conectados a um PC via cabo paralelo específico de configuração. Finalmente, um exercício de alteração do projeto original, sua validação, síntese, configuração em hardware e novo teste devem ser realizados pelos alunos. Adicionalmente, este Laboratório apresenta uma breve introdução a FPGAs, os dispositivos sobre os quais a plataforma de prototipação usada em aula está baseada.

3.2 Conceitos Fundamentais de Ferramentas de CAD e Hardware Utilizado

Nesta Parte, introduz-se alguma terminologia associada com o projeto de sistemas digitais, em particular o projeto via uso de linguagens de descrição de hardware (HDLs):

- Síntese Lógica
  - Processo de transformar a descrição VHDL em uma descrição em nível de transferência entre registradores (em inglês, *Register Transfer Level* ou *RTL*) equivalente, menos abstrata. A descrição RTL é um arquivo textual descrevendo uma interconexão de portas lógicas e elementos de memória. Esta descrição é independente de tecnologia de implementação em hardware.
- Síntese Física
  - Processo de mapear a descrição RTL nos elementos de algum dispositivo de hardware real. No caso da presente disciplina, este processo endereçará dispositivos VLSI reconfiguráveis comumente denominados de FPGAs. Um FPGA é um tipo de circuito integrado (em inglês, *integrated circuit* ou *chip*) onde existem blocos lógicos que implementam tabelas verdade preenchíveis (ditas reconfiguráveis e denominadas LUTs, do inglês *Look-Up Tables*), memórias, pinos configuráveis e fios configuráveis para interconectar todos estes elementos. Assim, a saída da síntese lógica é a entrada da síntese física, onde um diagrama de interconexão de portas lógicas é transformado em uma interconexão de componentes existentes no interior do FPGA. No caso dos dispositivos FPGA da empresa Xilinx, usa-se o ambiente de desenvolvimento *Integrated Systems Environment* ou ISE, desenvolvido pela empresa para realizar ambos, síntese lógica e síntese física. Outros ambientes podem ser usados para outros tipos de implementação, como por exemplo um *chip* fabricado sob encomenda.
  - Os principais passos de uma síntese física são: o *mapeamento*, que decompõe o projeto original em subconjuntos de portas lógicas, com cada subconjunto formado por um conjunto de portas que cabe dentro de exatamente uma LUT e/ou memórias; o *posicionamento*, que distribui LUTs e memórias em posições específicas do FPGA escolhido, em linhas e colunas da matriz do dispositivo escolhido; o *roteamento*, que conecta os elementos de hardware entre si, por meio da definição das interconexões reconfiguráveis; finalmente, há a *geração da imagem binária da configuração* do FPGA, um arquivo denominado **<nome do projeto>.bit** no caso do ambiente ISE da Xilinx. Este arquivo contém todos os bits usados para preencher a memória de controle que define a configuração do FPGA.
- Download
  - Processo de descarregar o resultado da síntese física (**<nome do projeto>.bit**) no FPGA.
  - Será utilizada uma ferramenta uma ferramenta disponível no ambiente ISE da Xilinx, o software **iMPACT**. Este software se comunica com o FPGA da placa D2SB através de um cabo paralelo, usado para enviar o arquivo de configuração para o FPGA, implementando desta forma o hardware descrito pelo arquivo.

- Placa D2SB, placa autônoma de hardware, fabricada pela empresa Digilent Inc. O manual da placa está disponível em http://www.digilentinc.com/Data/Products/D2SB/D2SB-rm.pdf. A placa contém:
  - Um FPGA Xilinx da família Spartan 2E, o XC2S200E, com capacidade para implementar circuitos com dimensão de aproximadamente 200.000 portas lógicas de 2 entradas. Este FPGA possui um encapsulamento PQ208, de 208 pinos, sendo que destes 143 pinos podem ser usados como pinos de entrada e/ou saída pelo hardware implementado no FPGA.
  - Um oscilador de 50 MHz e um soquete para acrescentar um segundo oscilador, ambos usados para gerar sinais de relógio para circuitos no FPGA.
  - Um soquete para uma memória Flash-ROM programável, capaz de armazenar uma configuração completa do FPGA.
  - Dois reguladores de tensão de 1A (3.3V and 1.8V).
  - Seis conectores de expansão, denominados A1, A2, B1, B2, C1 e C2.
  - 1 LED e 1 botão do tipo *pushbutton*, disponíveis para acionamento e leitura pelo FPGA, respectivamente.
  - Cabo paralelo de configuração e fonte de alimentação.
- Placa DIO4. Esta é uma placa auxiliar contendo dispositivos de entrada e saída. O manual da placa está disponível em http://www.digilentinc.com/Data/Products/DIO4/DIO4_rm.pdf. A placa contém:
  - Um mostrador de 4 dígitos de 7 segmentos.
  - 8 LEDs individuais.
  - 4 botões do tipo *pushbutton debounced*, ou seja, sem geração de transições espúrias.
  - 8 chaves deslizantes (*slide switches*).
  - Uma porta VGA de 3 bits.
  - Uma porta para teclado ou mouse PS/2.

3.3 Criação de um Projeto ISE

O ponto de partida do projeto é o arquivo de implementação da tabela verdade do Laboratório 2. Caso você não tenha sua própria versão do arquivo, é possível recuperar o arquivo "`tab3e2s.vhd`" da *homepage* da disciplina, existe um *link* para este na parte de Conteúdos da *homepage*, Aula 3. Mantenha este arquivo em um diretório de trabalho, ele será empregado abaixo.

Abrir o ambiente ISE, usando o caminho **Start Menu → Programs → Xilinx ISE 7.1i → Project Navigator**. Caso o programa inicie com algum projeto de um usuário anterior sendo aberto, feche o projeto, escolhendo a opção de menu **File → Close Project**. Em seguida, inicie um novo projeto, escolhendo a opção de menu **File → New Project**. Na janela inicial, primeiro escolha um diretório local (fora da área de instalação do programa, tipicamente em **C:\Temp**). Em seguida, escolha um nome para o projeto e certifique-se de indicar o **Top Level Module Type** como tipo HDL, pois far-se-á o projeto de hardware todo em VHDL. Após isto, clique **Next**.

Na janela seguinte deve ser especificado o dispositivo para o qual a síntese será realizada. Primeiro, escolha a família de dispositivos (**Device Family**) Spartan2E. Escolha em seguida o dispositivo (**Device**) xc2s200e. A seguir escolha o encapsulamento (**Package**) PQ208 e a Velocidade (**Speed Grade**) como -6. Todos estes dados podem ser obtidos no FPGA da placa D2SB. Para concluir esta janela, certifique-se que a ferramenta de síntese escolhida (**Synthesis Tool**) será o XST (VHDL/Verilog) da Xilinx. Clique **Next**.

Criação do Arquivo Topo da Hierarquia

A próxima janela permite acessar *wizards* para criação de arquivos de vários tipos. Apesar de nosso arquivo descrevendo o hardware estar pronto, é necessário criar um novo arquivo que vai ser o arquivo **topo da hierarquia**, pois haverá necessidade de adaptar o circuito VHDL para funcionar na plataforma de hardware escolhida, como explicado mais adiante.

Na fase de projeto e validação por simulação do hardware (estudada no Laboratório 2), ignorou-se a plataforma de prototipação destino do hardware, criando uma descrição independente de tecnologia. Este fato é bastante comum de ocorrer, e facilita o projeto inicial. Contudo, ao prototipar o hardware, com freqüência é necessário ajustar este para operar em uma plataforma já existente, o que normalmente requer hardware adicional. Apesar de ser possível alterar o arquivo VHDL já simulado para ajustar o hardware a funcionar na plataforma a ser usada, evita-se fazê-lo, para manter este portável entre diferentes plataformas. Assim, cria-se um novo arquivo, que insere o hardware simulado e provê as adaptações necessárias deste à plataforma a ser usada. Isto será feito agora, a partir da janela aberta.

Na janela que permite criar novos arquivos fonte (**Create New Source**) Clique **New Source**. Na janela a seguir, escolha o ícone **VHDL Module** da lista à esquerda, escolha o nome `top.vhd` e certifique-se que a opção **Add to Project** da janela **New Source** está selecionada, para que o arquivo seja criado no diretório de trabalho do projeto. Clique **Next**. Na janela seguinte, especifique 4 nomes de sinais exatamente iguais aos do circuito simulado (as entradas Ent_A, Ent_B e Ent_C de 1 bit e a saída Saida de 2 bits) e adicione um novo sinal de saída de 1 bit, denominado Hab_L. Este sinal é necessário pelo motivo descrito abaixo.

Em nosso protótipo, usaremos como entradas do circuito chaves deslizantes da placa DIO4 e como saídas LEDs da mesma placa. Se as duas placas D2SB e DIO4 são conectadas adequadamente (ver abaixo o que isto significa) as chaves deslizantes estarão conectadas diretamente a pinos do FPGA da placa D2SB. Contudo, o mesmo não ocorre com os LEDS. Estes têm suas entradas ligadas às saídas de uma memória (de fato um registrador), cujas entradas, estas sim, estão ligadas a pinos do FPGA. A memória possui um sinal de controle de 1 bit denominado LEDG, que quando em '1' permite fazê-la funcionar em *modo transparente*. Neste modo, o que aparece nas entradas é diretamente passado às saídas da memória, simulando o comportamento de um mero conjunto de fios. Como por omissão este sinal vale '0', o hardware deve prover este sinal além das saídas do circuito. O esquema de organização do projeto é ilustrado na Figura 15.

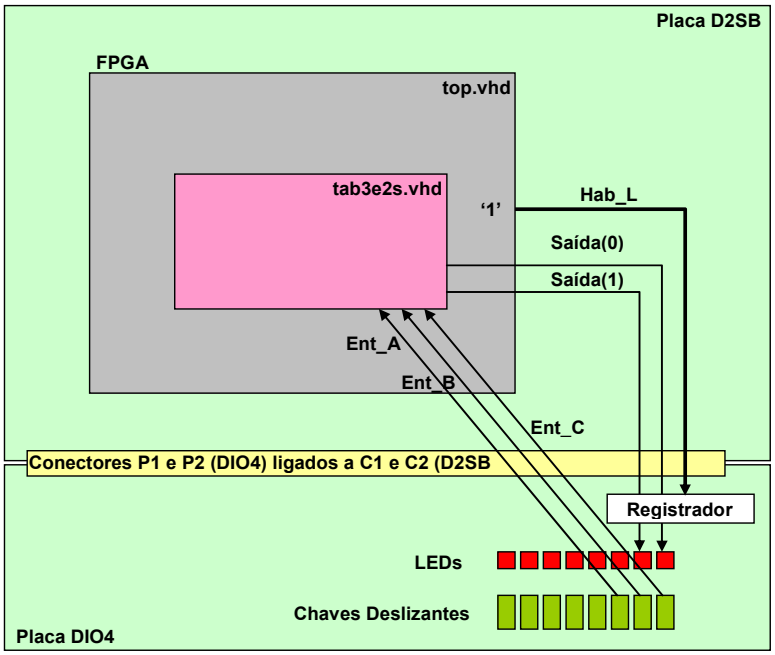


Figura 15 – Esquema geral da prototipação do hardware tab3e2s na plataforma D2SB/DIO4.

Depois de entrar com os nomes dos 5 sinais, clique **OK**, e na janela que volta a aparecer, clique **Next**.

Inserção do Arquivo com a descrição do Circuito

A janela a seguir permite incluir arquivos previamente existentes no projeto em definição. Clique no botão **Add Source** e selecione o arquivo com a descrição da tabela verdade do Laboratório 2 (por exemplo, o arquivo `tab3e2s.vhd` citado acima). Após escolher o arquivo, surge uma janela **Choose Source Type**. Esta serve para identificar o arquivo VHDL como um arquivo de projeto (que é o caso) ou como um arquivo de *testbench*. Este último tipo, como já foi visto serve apenas para simulação, e não para síntese. Escolha a opção VHDL Design File e aperte OK. A seguir, certifique-se que a opção **Copy to Project** da janela **New Project** está selecionada, para que o arquivo seja copiado para o diretório de trabalho do projeto, ao invés de se criar apenas um ponteiro para este. Clique **Next** e confira os dados do projeto a ser criado na janela seguinte. Se tudo estiver compatível com as parametrizações acima, basta clicar no botão **Finish**. A seguir, deve-se ter a janela principal do ambiente ISE mostrando conteúdo similar ao da Figura 16. Nesta janela, existem 4 sub-janelas: O *browser* para módulos de projeto, *snapshots* (versões congeladas de um projeto) e bibliotecas (intitulado **Sources in Project**), o *browser* de aplicativos, processos e relatórios de execução (intitulado **Processes for Source "top-top"**), a janela de edição e visualização de arquivos (a maior janela, acima à direita), e a janela de interação, com abas console, busca em arquivos, erros e alertas (**Warnings**).

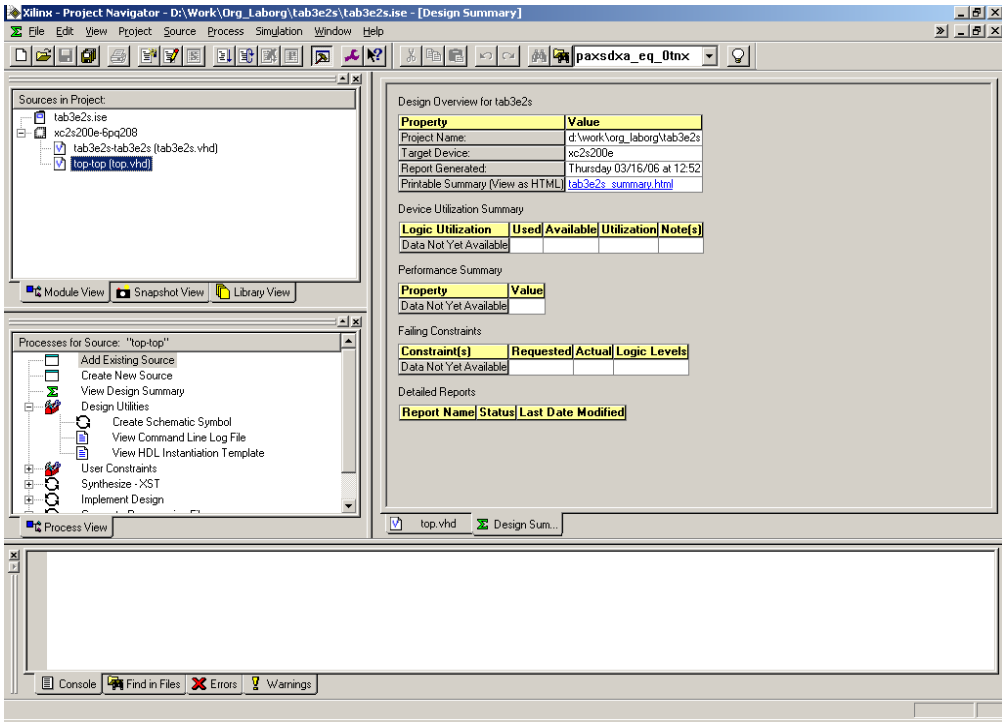


Figura 16 – Janela principal do ambiente ISE 7.1 após a criação do projeto tab3e2s.

3.4 Definição do Top, Pinagem Externa, Síntese Lógica e Síntese Física

Definição do Top

O arquivo topo foi criado pelo *wizard* usado acima e já possui definições de entidade e arquitetura, mas esta última encontra-se vazia. Neste caso, como pode ser observado na Figura 15, o arquivo topo contém apenas fios que repassam as entradas e saídas entre os pinos do FPGA e o circuito tab3e2s, e um fio que deve estar sempre em '1'. O conteúdo da arquitetura deve ser então como mostrado na Figura 17. Note-se que a arquitetura do módulo `top` instancia o circuito que implementa a tabela verdade. Note-se também que se utilizou uma forma mais compacta de instanciação, mencionando uma entidade já disponível na biblioteca de trabalho (**work**). Esta forma é o padrão da linguagem VHDL 1993, versão mais recente que o VHDL 1987. este último, mais antigo, usa a declaração component. Ambas as formas são aceitas pelos ambientes Active-HDL e ISE. Deve-se preferir a mais recente, por ser menos prolixa.

```
architecture top of top is
begin

   Hab_L <= '1';
   InstTab: entity work.tab3e2s port map (Ent_A=>Ent_A, Ent_B=>Ent_B,
                                          Ent_C=>Ent_C, Saida=> Saida);

end top;
```

Figura 17 – Descrição da arquitetura do arquivo `top.vhd` para o projeto ISE tab3e2s funcionar na plataforma D2SB/DIO4.

Definição da Pinagem Externa

Para implementar um hardware em um dispositivo físico real, é necessário atribuir a cada bit de cada entrada ou saída do circuito um pino do dispositivo externo. No presente caso, se quer implementar uma tabela verdade de três entradas e duas saídas, mais o pino de habilitação do registrador conectado aos LEDs. Uma forma simples de visualizar o funcionamento deste hardware é associar chaves às entradas e LEDS às saídas do mesmo, como já proposto na Figura 15. Como não há chaves ou LEDs suficientes na placa D2SB, é

preciso usar os recursos da placa DIO4, ligada à D2SB através de conectores compatíveis.

A tabela verdade será implementada no FPGA da placa D2SB. Este FPGA (observe na placa) possui 208 pinos externos. Como descrito antes, 143 destes pinos estão disponíveis para serem configurados como entradas e/ou saídas de circuitos implementados sobre o FPGA. Além disto, o *chip* está soldado na placa da plataforma D2SB, e cada pino foi conectado a fios da placa indo para lugares específicos, entre estes os pinos dos conectores fêmea (observar na placa) A1, A2, B1, B2, C1 e C2. A placa DIO4 possui dois conectores macho, P1 e P2, que devem ser ligados aos conectores C1 e C2 da placa D2SB, respectivamente. Esta conexão deve estar pronta nos conjuntos do laboratório. Verifique que isto se aplica, mudando a conexão se necessário, tomando cuidado para não danificar os pinos do lado macho, que facilmente pode ser quebrados.

Para facilitar o trabalho de identificação dos pinos do FPGA e sua relação com os elementos de hardware (chaves deslizantes e LEDS) a usar neste e em outros laboratórios, foi criada uma tabela de conversão de pinos que, assumindo a ligação de conectores descrita acima, associa pinos do FPGA com elementos da placa DIO4. Recupere este arquivo (tabela_conversao_D2SB_DIO4.doc) da página da disciplina, Conteúdo das Aulas, Aula 3. Abra e note que este arquivo contém uma tabela de 2x4 colunas, que associa pinos dos conectores P1/P2 (da placa DIO4) ao nome do sinal (que identifica o componente da DIO4 conectado a este fio), ao pino do conector C1/C2 (da placa D2SB) e ao pino do FPGA..

Neste Laboratório, o dado relevante é o pino do FPGA conectado a cada componente. Assim, as chaves deslizantes que serão usadas devem ser escolhidas dentre os sinais SW1 a SW8 da placa DIO4. Por exemplo, nota-se que a chave SW1 da placa DIO4 (em baixo, a chave mais à direita) está vinculada ao pino 23 do FPGA. Digamos que se quer associar as entradas `Ent_A, Ent_B` e `Ent_C` da tabela verdade às chaves SW3, SW2 e SW1, respectivamente. Da tabela de conversão, nota-se que se deve associar `Ent_A` ao pino 18 do FPGA, `Ent_B` ao pino 21 e `Ent_C` ao pino 23. Para as saídas usaremos os LEDs associados aos sinais LED2 e LED1 (pinos 109 e 111 do FPGA), sendo LED2 associado ao sinal `Saida<1>` e LED1 associado ao sinal `Saida<0>`. Na placa os LEDs citados estão identificados como LD2 e LD1. Finalmente, o sinal de controle de habilitação do registrador localizado entre o FPGA e os LEDs denomina-se LEDG na placa DIO4 e está associado ao pino 45 do FPGA. A maneira de informar estas e quaisquer outras associações de sinais do VHDL a pinos do FPGA no ambiente ISE 7.1 será descrita a seguir.

1. Na janela **Sources in Project**, selecione (se já não estiver selecionado) o arquivo VHDL com o topo da hierarquia de projeto (`top.vhd`). A seguir, na janela identificada como **Processes for source "...** identifique sob o título **User Constraints** o programa **Assign Package Pins (PACE)**.
2. Lance o programa **PACE** com duplo clique sobre o ícone associado. Como resposta a esta ação, uma janela de confirmação aparece, a qual se deve responder **Yes**. Trata-se de confirmar a criação de um arquivo de restrições de projeto, com extensão UCF (do inglês, *User Constraint File*), que pode já existir com nome diverso do padrão (que é `<nome do projeto>.ucf`). Após a confirmação, surge uma nova janela, com 5 sub-janelas. Usa-se aqui apenas a sub-janela intitulada **Design Object List – I/O Pins**. Nesta, perceba a presença de uma tabela onde cada linha está associada com as entradas e saídas da tabela verdade que descreve o circuito a implementar.
3. Use a coluna denominada **LOC** (do inglês, *location*). Clique na primeira posição desta coluna (deve estar associada a Ent_A) e digite **P18** (escolhendo o Pino 18 do FPGA para associar ao sinal `Ent_A`). repita a ação para os demais sinais. Note que à medida que associações são feitas, a janela ao lado mostra numa planta baixa do FPGA onde os pinos escolhidos se encontram fisicamente localizados (um pequeno retângulo azul marca cada pino, todos localizados na periferia do chip).
4. Uma vez terminada especificação de associação de sinais a pinos do FPGA, basta salvar o arquivo UCF gerado. Caso apareça uma janela de diálogo denominada **Bus Delimiter**, escolha a opção **XST Default** e clique **OK**.
5. Feche o programa **PACE**. A seguir, você pode visualizar o arquivo criado indo na janela **Processes for source "...**, identificando o título **User Constraints** e clicando no ícone associado ao texto **Edit Constraints (Text)**.

Síntese Lógica

Apenas uma ação precisa ser efetuada para realizar a síntese lógica, após a associação de sinais a pinos, e caso não haja erros no VHDL. Em nosso caso, entretanto haverá um erro.

Vá à janela **Processes for source "...,** identifique o título **Synthesize – XST** e dê um duplo clique no ícone associado. Aguarde o final da síntese. Uma mensagem de Erro deve aparecer na **Console**. Clique na aba **Errors** para saber detalhes do erro. Caso esteja usando o arquivo fornecido na *homepage*, o erro deve ocorrer na linha 42 do VHDL, com o comentário **Can not determine the type of the selector &**.

Este erro ocorre porque, como visto em aula teórica, nem toda construção válida em simulações VHDL pode ser sintetizada. No caso particular, a concatenação de vários sinais para gerar uma expressão de teste confunde o gerador de hardware, por permitir várias implementações possíveis, embora o comportamento do hardware esteja plenamente determinado. O que justifica ser a construção simulável.

Assim, será necessário editar o texto VHDL para eliminar o erro de síntese. Uma forma simples de resolver o problema é criar um sinal que seja a concatenação das três entradas (`Ent_A, Ent_B e Ent_C`), digamos `Ent`, e usar este para ser testado no comando `with`. Dê um duplo clique na linha que descreve o erro para abrir o arquivo e visualizar esta linha. Edite a arquitetura como mostrado na Figura 18. Em seguida, basta salvar o arquivo alterado e relançar a síntese lógica como antes. O ícone ao lado da janela **Synthesize – XST** deve apresentar, ao final da síntese, um *check mark* verde, indicando execução completa desta sem erros.

```
architecture tab3e2s of tab3e2s is
signal Ent : std_logic_vector (2 downto 0);
begin
    Ent <= Ent_A & Ent_B & Ent_C;
    with Ent select
           Saida <= "01" when "000", -- o restante é igual ao original
```

Figura 18 – Novo texto VHDL do arquivo `tab3e2s.vhd`, para que este não gere erro durante a síntese lógica.

Em seguida é possível observar o relatório da síntese lógica, com um duplo clique sobre o ícone associado à opção **View Synthesis Report** sob o título **Synthesize – XST** na janela **Processes for source "...**. No relatório, que é bem extenso, procure o título **Device Utilization Summary**, e note que, seja lá o que forem as unidades usadas, muito pouco do FPGA foi usado para implementar o circuito: 1 Slice de 2352 disponíveis, 2 LUTS de 4 entradas das 4704 disponíveis, e 6 pinos dos 146 disponíveis.

Síntese Física

Uma vez concluída a síntese lógica deve-se realizar a síntese física, como definido anteriormente. Para tanto, basta ir à janela **Processes for source "...,** identificar o título **Implement Design** e dar um duplo clique no ícone associado. Não se esqueça que na janela **Sources in Project**, deve estar selecionado o arquivo VHDL que descreve o seu circuito. Não devem ocorrer problemas durante esta síntese física. Pode-se em seguida visualizar a planta baixa do circuito gerado. Vá à janela identificada como **Processes for source "...** identifique sob o título **Implement Design** o programa **FPGA Editor** (identificado pelo texto **View/Edit Routed Design (FPGA Editor)**). Clique no ícone associado ao programa e visualize o projeto, que deve ser similar ao mostrado na Figura 19.

Em seguida à síntese física, basta gerar o arquivo de configuração. Para tanto, basta ir à janela **Processes for source "...,** identificar o título **Generate Programming File** e dar um duplo clique no ícone associado, aguardando o final da geração do arquivo (no caso do projeto mostrado aqui este arquivo se denomina `top.bit`). Pode-se constatar a correta geração deste arquivo abrindo o relatório identificado como **Programming File Generation Report** na janela **Processes for source "...**.

Uma observação interessante é que todo o processamento acima (síntese física, síntese lógica e geração do arquivo de configuração) pode ser disparado usando apenas a ação descrita no último parágrafo.

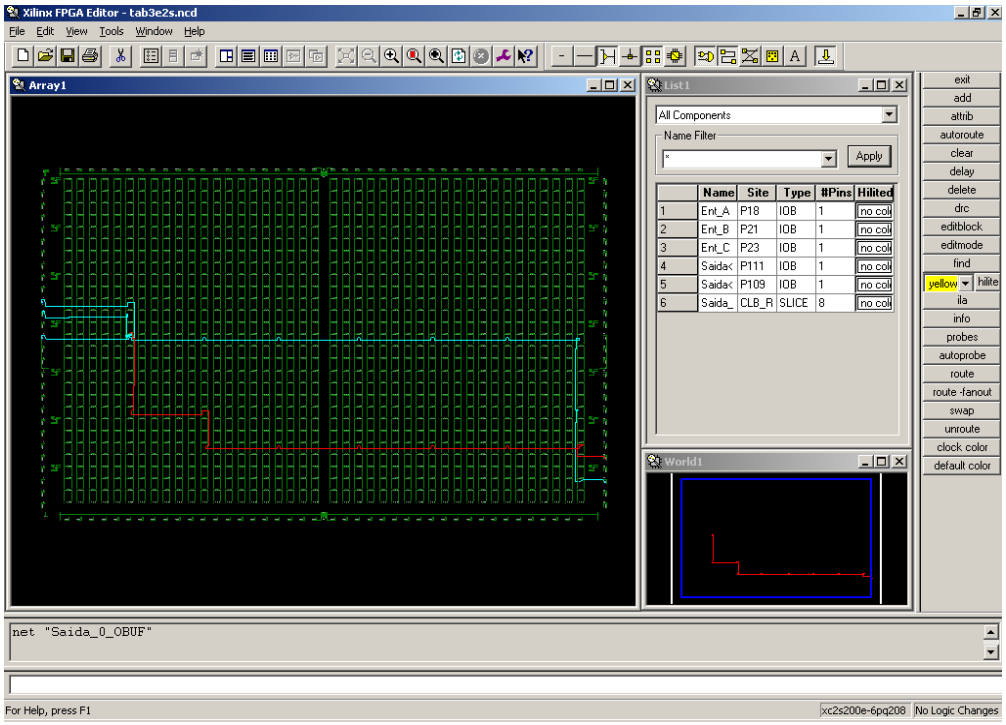


Figura 19 – Janela principal do software FPGA Editor após a síntese física do projeto `tab3e2s`.

3.5 Descarga do arquivo de Configuração e Teste do Circuito

Uma vez gerado o arquivo `tab3e2s.bit`, basta usar o mesmo para configurar o FPGA na plataforma de prototipação D2SB/DIO4 e testar se o circuito implementado funciona adequadamente. Estas ações serão descritas nesta Seção.

Conexão e Alimentação da Plataforma no Hospedeiro

Antes de configurar o FPGA é necessário alimentar a plataforma D2SB/DIO4 e conectá-la ao computador hospedeiro onde se está realizando o desenvolvimento com o ambiente ISE. Obtenha no armário do laboratório uma plataforma. Certifique-se que junto ao computador existem já instalados o cabo de alimentação e o cabo paralelo. Se este não for o caso, obtenha este par de cabos do armário ou com o professor da disciplina. Ligue o cabo paralelo ao conector JTAG da placa, verificando a correta polaridade do mesmo. Este conector encontra-se na parte superior esquerda da placa D2SB, logo abaixo do *push-button* BTN1. Em seguida alimente a placa, ligando o cabo de alimentação ao conector marcado 6V-12V DC na placa D2SB. Obviamente, certifique-se que o outro lado do carregador está conectado a uma tomada e que o outro lado do cabo paralelo está conectado à porta adequada do computador hospedeiro. Note-se que não é necessário alimentar a placa DIO4, ela recebe alimentação pelos conectores P1 e P2 da placa D2SB.

Após a alimentação, o único efeito perceptível deve ser o acendimento do LED LD2, posicionado no canto inferior esquerdo da placa D2SB. Se isto não acontecer, contacte o professor da disciplina.

Configuração do FPGA

Ir na janela **Processes for source "...,** identificar o título **Generate Programming File** e sob este ache o programa **iMPACT**, sinalizado pelo texto **Configure Device (iMPACT)** e ícone associado. Lance o programa com um duplo clique no ícone. O resultado é o surgimento de uma nova janela (a do programa **iMPACT**), e sobre esta uma janela de diálogo, mostrada na Figura 20.

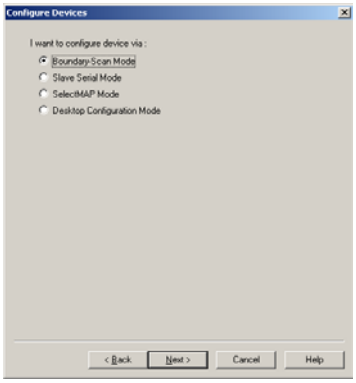


Figura 20 – Janela inicial de diálogo do software iMPACT.

Certifique-se que a opção **Boundary-Scan Mode** está selecionada e clique **Next**. Na próxima janela de diálogo, certifique-se que a opção de detecção automática da conexão está selecionada e clique **Next** (**Atenção: Este passo somente pode ser feito com a plataforma conectada ao hospedeiro e alimentada**). Como resultado, aparece um ícone com o formato de um chip e com nome **xcv200e**. Leia a mensagem da janela **Boundary-Scan Chain Contents Summary**, certifique-se de tê-la entendido, e clique **OK**. Após isto, deve surgir um *browser* do sistema operacional que lhe permite achar o arquivo com extensão **bit** a usar. Ache o arquivo `tab3e2s.bit` (se já não estiver no diretório certo) e o escolha, clicando a seguir em **Open**. Leia as mensagens e aceite-as todas (uma sobre a troca do nome do dispositivo e outra sobre a troca do tipo de Clock de configuração) com **OK**. Para iniciar a configuração do FPGA clique com o botão direito do *mouse* sobre o ícone do FPGA e escolha a opção **Program...**. Na janela de diálogo que surge, apenas clique em **OK**. Isto deve iniciar o processo de configuração do FPGA, que dura alguns segundos. A mensagem **Programming Succeeded** deve aparecer na janela do iMPACT. Sendo este software um tanto sensível, freqüentemente ocorrem condições que o levam a falhar na configuração do FPGA. Consulte o professor da disciplina caso isto ocorra.

Teste do Circuito

Para testar o circuito, basta usar as três chaves deslizantes mais à direita da placa DIO4 para gerar os valores de entrada da tabela verdade (chave para baixo é '0', chave para cima é '1') e verificar se os dois LEDs LD2 e LD1 revelam o comportamento correto das saídas (LED apagado é '0', LED aceso é '1'). Mostre o resultado ao professor da disciplina.

3.6 A Fazer e a Entregar

A maior parte das tarefas deste Laboratório foi realizada de forma concomitante com a discussão acima. Contudo, pode-se estender o Laboratório como segue abaixo.

Tarefa 1: Analise os relatórios gerados pelas tarefas de síntese, seja a síntese lógica (ou independente de tecnologia), seja a síntese física (também chamada de dependente de tecnologia). Existe no ambiente ISE um relatório (**Report**) para praticamente cada passo importante executado durante o projeto. **RESPONDER**: a partir dos relatórios, diga qual a taxa de ocupação dos CLBs do FPGA, e diga qual o tamanho do circuito final, em portas lógicas equivalentes (**equivalent gate count**). Procure descobrir o que significa esta medida e descreva isto no relatório do Laboratório.

Tarefa 2: Antes de realizar o *download*, foi mostrado o resultado da síntese física usando a ferramenta **FPGA Editor**. (ver item 3.4 acima). Como se pode ver na Figura 19, um FPGA é uma matriz bidimensional de blocos. Cada retângulo verde da Figura representa o espaço ocupado por um tal bloco, denominado **Configurable Logic Block** (CLB). Um CLB possui duas fatias (***slices***). Cada *slice* consiste a grosso modo de duas tabelas verdade configuráveis de 4 entradas e uma saída (as LUTs definidas antes) e dois bits de memória configuráveis. Em função disto, **RESPONDER**: Em sua opinião, qual o impacto do número de CLBs na implementação física? Por exemplo, existe limite para o tamanho de circuito implementável em cada FPGA? Todo o processo de geração do circuito acima foi automatizado, o software de síntese física

escolhendo onde colocar cada elemento, como gerar os fios que interconectam elementos distintos, etc. **RESPONDER**: Qual o impacto das ferramentas de CAD automáticas na implementação física? Por exemplo, observe o "layout" do circuito final obtido com o FPGA Editor. Há muito ou pouco espaço vago? Os fios implementados são sempre os mais curtos ou não? Por quê?

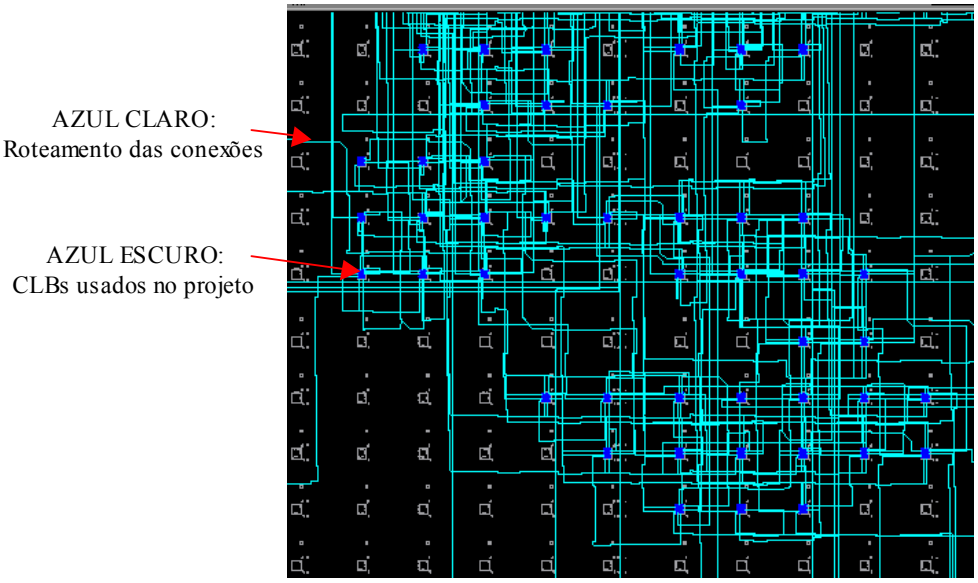


Figura 21 – Detalhe de uma planta baixa de circuito implementado sobre um FPGA Xilinx.

Tarefa 3: Altere o hardware implementado. A idéia é acrescentar uma entrada que escolhe se a função a ser mostrada nos LEDs é a função original, ou uma outra função de 3 variáveis diferente, que você deve escolher. Pede-se:

- Quais dos passos mostrados acima devem ser refeitos?
- Quanto tempo você gastou para realizar a alteração e ver o novo hardware funcionando?

Tarefa 4: Mostrar o projeto funcionando ao professor. **Mostrar o projeto funcionando ao professor!**

4 Projeto e Validação de Hardware Combinacional em VHDL

Prática: Projeto de uma ULA de 4 bits
Recursos: Ambiente de Desenvolvimento Active-HDL da Aldec, Inc.

4.1 Introdução e Objetivos

Os Laboratórios anteriores versaram sobre a capacitação da Turma para uso dos recursos disponíveis, incluindo manipulação de instrumentos de geração e medidas, ambientes de simulação e ambientes de síntese e prototipação de *hardware*. A partir deste Laboratório, inicia-se a prática de projeto e implementação de hardware em VHDL propriamente dita.

Sistemas digitais complexos são máquinas processadoras de informação codificada em binário. As principais funções de manipulação de informação podem ser classificadas em um de quatro grandes grupos, cada um contando com componentes especiais de hardware que viabilizam a realização destas tarefas essenciais:

- Armazenamento de informação – incluindo, por exemplo, o papel desempenhado por memórias e registradores;
- Transformação de informação – contemplando, sobretudo, operações aritméticas e lógicas;
- Transporte de informação – exemplos são as tarefas desempenhadas por componentes como multiplexadores, decodificadores, demultiplexadores e barramentos;
- Controle – desempenhado por elementos especificamente projetados para comandar as funções desempenhadas pelos demais grupos de funções. Exemplos de elementos que desempenham tais funções são as máquinas de estado finitas.

A transformação de informação se dá basicamente pela execução de operações aritméticas tais como soma, subtração, multiplicação e divisão e operações lógicas, tais como as operações E, OU, OU-EXCLUSIVO e INVERSÃO, entre outras. Em muitos elementos o uso destas funções é eventual e, portanto não se justifica o custo de implementar cada uma delas sob a forma de um hardware dedicado. Uma solução interessante costuma ser congregar em um único módulo de hardware configurável a capacidade de realizar cada uma destas funções em um dado instante de tempo. Tal módulo costuma ser denominado de Unidade Lógico-Aritmética, e possui uma estrutura genérica conforme mostrado na Figura 22. Neste componente, existem duas entradas e 1 saída de dados (em vermelho na Figura), um sinal de entrada de controle (**Operacao** na Figura), que escolhe uma dentre um conjunto finito de operações a executar, e um conjunto de saídas de status (**Qualificadores** na Figura), que qualificam o resultado da operação sendo realizada.

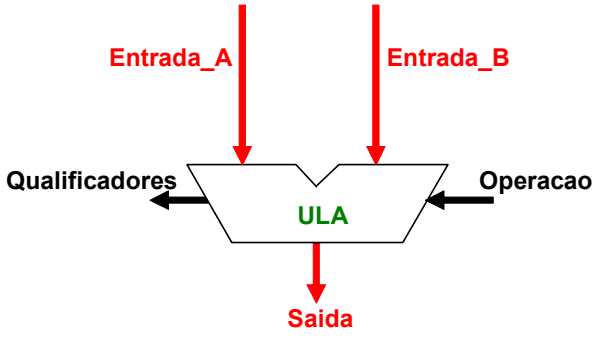


Figura 22 – Estrutura geral de uma Unidade Lógico-Aritmética.

O motivo de haverem duas entradas de dados é permitir realizar as operações mais comumente usadas, quais sejam operações binárias como soma e subtração e operações unárias como incremento e inversão. A **Operação** consiste tipicamente de um código binário distinto para cada operação, definido durante o projeto do componente. Os **Qualificadores** dão informações específicas tais como se o resultado de uma operação resultou no valor zero ou não, se uma operação aritmética gerou transbordo (vai-um) ou não, ou se o

resultado corresponde a um número negativo ou não. Cada uma destas informações binárias corresponde a um bit específico do vetor de bits **Qualificadores**. Um exemplo de implementação comercial de uma ULA é o antigo circuito integrado TTL, um CI de 24 pinos contendo uma ULA de 4 bits – com código 74LS181, cuja documentação está disponível por exemplo, no hiperlink da disciplina de Circuitos Digitais, em http://www.inf.pucrs.br/~calazans/undergrad/circdig/TTL/sn74ls181.pdf.

O objetivo específico deste Laboratório é implementar uma ULA de 4 bits em VHDL, gerar um *testbench* adequado para validar a implementação e simular o *testbench*, mostrando a correta funcionalidade do componente criado.

4.2 Especificação da ULA

A primeira característica da ULA a implementar é que esta deve operar sobre dados de 4 bits, significando que, referindo-se à Figura 22, as entradas (A e B) e a saída da ULA são vetores de 4 bits. A ULA a implementar está parcialmente especificada na Tabela 3. A estrutura de uma ULA como esta é tipicamente encontrada em circuitos integrados simples tais como os microprocessadores de primeira geração como o Intel 8080 ou o Motorola 6502.

Esta ULA será capaz de realizar 8 operações distintas, sendo 4 aritméticas e 4 lógicas. Para implementar 8 operações distintas, obviamente é necessário um mínimo de 3 bits para especificar os 8 diferentes códigos de operação, e este mínimo deve ser respeitado aqui. A Tabela 3 será completada de forma diferente para cada grupo durante a aula. Note-se que será especificado pelo professor um código distinto para cada uma das 8 operações do conjunto de **Conjunto de Operações Disponíveis**, e um subconjunto de qualificadores dentre os quatro possíveis a implementar constantes na Tabela. Das operações disponíveis, pode-se notar que várias são binárias e várias são unárias, algumas são lógicas e outras são aritméticas. Além disto, algumas operações têm mais de uma forma, tal como a subtração e o decremento. Os qualificadores são designados por letras maiúsculas com o significado seguinte (N=1 se resultado da operação é um número negativo em complemento de 2, N=0 caso contrário; Z=1 se resultado da operação é a constante 0, Z=0 caso contrário; C=1 se resultado da operação gerou um vai-um, C=0 caso contrário, sendo relevante apenas para operações aritméticas; V=1 se resultado da operação excedeu a capacidade de representação em complemento de 2, V=0 caso contrário, sendo relevante apenas para operações aritméticas).

Tabela 3 – Sumário incompleto da especificação da ULA a implementar.

Código da Operação	Associação para o Grupo	Conjunto de Operações Disponíveis
000		A + B (soma)
001		A – B ou B – A (subtração)
010		Decremento (A / B)
011		Incremento (A / B)
100		A or B / A nor B
101		A and B / A nand B
110		A xor B / A nxor B
111		not A / not B
Qualificadores		N, Z, C, V

4.3 A Fazer e a Entregar

Tarefa 1: Leia o texto acima e siga as orientações para criar a ULA especificada para seu grupo em VHDL. Inicialmente, use os recursos vistos no Laboratório 2 para gerar a entidade do seu hardware.

Tarefa 2: Concluída a entidade, pense como implementar a arquitetura da ULA, consultando por exemplo as transparências 87 e 88 disponíveis no hiperlink abaixo: http://www.inf.pucrs.br/~calazans/undergrad/orgarqI/Intro_VHDL_mar_2006_4pp.pdf.

Lembre-se que uma ULA nada mais é que um conjunto de módulos de hardware que realizam diferentes operações em momentos diferentes sobre um conjunto de sinais compartilhados. Assim, a ULA pode ser implementada como um comando de atribuição condicional ao seu sinal de saída e atribuições condicionais em paralelo a esta, uma para cada qualificador de saída, conforme a especificação. Concluída a captura da descrição da ULA, compile seu arquivo e certifique-se de remover todos os erros de sintaxe do mesmo, caso existam.
Observações sobre VHDL: O exemplo da ULA pode ser usado para abordar alguns conceitos ainda não vistos em VHDL. Primeiro, explore o conceito de tipos enumerados. Este conceito é muito útil para facilitar a depuração, pois pode-se usar valores simbólicos ao invés de códigos binários para representar informações. Um exemplo de definição de tipo enumerado em VHDL é: `type cor is (vermelho, laranja, amarelo, verde azul, ciano, violeta);` A partir desta definição, se pode definir sinais ou vetores de sinais deste tipo e usá-los no seu projeto, como por exemplo, `signal cor_do_carro : cor;` Use este tipo de estrutura para definir o sinal de entrada `operacao` da sua ULA. Como o tipo deve ser usado para definir uma das portas de sua entidade, coloque a definição dentro de um *package*. A seguir declare o uso deste *package* no seu par entidade-arquitetura. Detalhes sobre a sintaxe das construções citadas podem ser obtidos dentro do simulador Active-HDL, no *Language Assistant*, acessível na barra de ícones (uma lâmpada branca), na opção *Language Templates*. Outro conceito interessante que pode ser usado em conjunção com tipos enumerados (mas não limitado a estes, serve para outros tipos) é o de *atributos* de um tipo. Estude o texto sobre atributos de tipo disponível a partir da opção de menu **Help → On-line Documentation → (aba) Contents → References → VHDL Language Reference Guide → attributes (predefined)**. Atente para a sintaxe e semântica associadas aos atributos `Succ`, `High` e `Low`. Use estes para implementar o *testbench* (Tarefas 3 e 5 abaixo).
Observações sobre o projeto: a implementação da saída (Saida) da ULA deve ser um simples comando de atribuição condicional, mas os qualificadores exigem um pouco mais de reflexão para serem gerados de forma simples e eficiente. Os qualificadores N e Z não devem ser problemáticos, considerando que N é apenas o sinal do resultado em complemento de 2. Assim, basta tomar o último bit do resultado e dar a este o nome de N. Z é gerado a partir de um teste sobre o valor completo do resultado contra o vetor de 4 bits "0000". A geração do qualificador C é menos simples. O vai-um de operações aritméticas não é uma informação gerada pelo operador VHDL correspondente. Logo, deve ser gerado de alguma outra forma. Duas soluções possíveis são gerar uma tabela verdade mostrando a combinação de todos os valores possíveis dos dois dados de entrada e o valor do qualificador associado a cada combinação. Isto é simples, mas gera um hardware grande, pois a tabela terá 2^8= 256 linhas, uma para cada combinação de entradas (0 com 0, 0 com 1, etc até 15 com 15). Uma solução muito mais simples é usar o artifício de estender os vetores de dados de entrada com um bit (0 ou 1) adicional à esquerda, fazer a operação com os valores estendidos e considerar o quinto bit como o resultado do vai-um. Use este artifício na sua solução. Como dica, use o operador `&` para concatenar bits ou vetores de bits a outro bit ou vetor de bits. Finalmente, o qualificador de transbordo em complemento de 2 é o mais complexo de ser obtido. Um material de apoio que pode ser estudado para entender o problema de gerar V está disponível em http://www.inf.pucrs.br/~calazans/undergrad/arq1/aulas/aritcomp.pdf, em particular nas transparências 21 e 22 deste material, que descreve todos os casos distintos de cálculo de transbordo em complemento de 2. Se o assunto representação em complemento de 2 não for bem dominado, estude-o novamente em livros-texto clássicos de circuitos digitais ou organização de computadores. Este assunto é pré-requisito da disciplina e não será visto aqui. Uma constatação importante que pode ser usada na implementação do seu hardware é que o ocorre transbordo em uma operação aritmética se o sinal do resultado for incorreto. Outra observação importante é que somente a soma de números de mesmo sinal ou a subtração de números de sinais distintos pode gerar transbordo (V=1).

Tarefa 3: Use o *Wizard* gerador de *testbenches* visto na Laboratório 2 para produzir o esqueleto de *testbench* para sua ULA. Em seguida, preencha a arquitetura do *testbench* com os três geradores de estímulos (dois para as entradas e um para o código da operação).

Tarefa 4: Concluída a geração do *testbench*, realize a simulação, lançando o script de execução do testbench e avançando o tempo de forma a mostrar uma simulação que execute pelo menos

uma vez cada uma das operações. Valide o funcionamento das saídas do seu componente.

Tarefa 5: **Pense e responda**: A sua simulação é exaustiva? Justifique sua resposta. Senão, defina o que seria uma simulação exaustiva e compute o custo de uma tal simulação em termos de tempo, supondo que cada passo de simulação dura 10ns. Você pode gerar um *testbench* que produza uma simulação exaustiva? Como? **Dicas**: Estude a sintaxe e a semântica do comando VHDL `for...loop`. Veja, por exemplo, o **Language Assistant** do simulador Active-HDL (para sintaxe) e o **VHDL Language Reference Guide** (para sintaxe e semântica), ambos mencionados acima. É possível criar no *testbench* um comando `process` contendo laços aninhados para gerar a simulação exaustiva. Não se esqueça de usar comandos `wait` para permitir o avanço da simulação. Não se esqueça que o comando `for...loop` não pode ser usado fora do corpo de comandos `process`.

Tarefa 6: Documente seu projeto no relatório do Laboratório e guarde sua implementação da ULA, pois esta será usada na aula que vem, para prototipar uma versão em hardware da mesma.

5 Implementação de Hardware Combinacional em HDL Captura, Síntese, Implementação e Teste

Prática: Implementação de uma ULA de 4 bits
Recursos: Ambiente de Desenvolvimento ISE da Xilinx, Inc. e Plataforma Digilent D2SB/DIO4

5.1 Introdução e Objetivos

O Laboratório anterior propôs o projeto e a simulação de uma ULA de 4 bits simples em VHDL, abordando os detalhes de implementação de 8 funções distintas, a seleção de uma destas funções a ser executada pela ULA a cada instante e a forma de gerar os qualificadores N, Z, C e V para cada operação.

Naquele Laboratório, foi discutida e justificada a estrutura geral de uma ULA como elemento básico de transformação de informação.

O objetivo específico deste Laboratório é implementar a mesma ULA de 4 bits do Laboratório anterior sobre a plataforma Digilent D2SB/DIO4, realizando as alterações e acréscimos necessários no projeto até que este seja passível de gerar uma descrição VHDL suficientemente detalhada para implementação no FPGA da plataforma, usando os recursos de entrada e saída necessários para excitar o hardware e para mostrar os resultados da operação da ULA.

5.2 Atribuição de Recursos de E/S

Esta Seção propõe um conjunto de recursos de entrada e saída a usar na plataforma D2SB/DIO4 para representar as entradas e saídas da ULA de forma o mais clara possível. Esta proposta definitivamente não é a única possível, nem necessariamente é a melhor. Trata-se apenas de uma proposta razoável. A Figura 23 ilustra o esquema geral de uso de recursos de entrada e saída da plataforma D2SB/DIO4 para implementar a ULA de 4 bits e permitir que esta seja usada por um usuário humano. O resto desta Seção coloca a proposta de uso e justifica as escolhas desta. Na Seção 5.3, algumas sugestões de alteração e/ou melhoria desta proposta são colocadas. Cabe aos grupos divisar outras alterações possíveis e experimentá-las na plataforma disponível.

Como característica mais visível da ULA, esta deve operar sobre 1 ou 2 dados de 4 bits, gerando um resultado de 4 bits. Visto que um dado de 4 bits é diretamente mapeável para exatamente 1 dígito representado em hexadecimal, que existem 4 mostradores de 7 segmentos na placa DIO4, que cada um destes pode ser usado para representar exatamente um dígito hexadecimal, é natural associar as 2 entradas e a saída de dados da ULA a 3 dos 4 mostradores de 7 segmentos, digamos os dois mostradores mais à esquerda associados aos sinais `Entrada_A` e `Entrada_B`, e o mostrador mais à direita associado à saída da ULA, ficando um mostrador apagado para separar entradas de saídas. Como as entradas devem ser escolhidas pelo usuário, é natural usar 4 chaves deslizantes das 8 disponíveis para permitir ao usuário especificar o valor de `Entrada_A` e `Entrada_B`. Assim, cada vez que o usuário escolher um valor para as entradas de dados movendo as chaves deslizantes, os 2 mostradores mais à esquerda apresentam o número hexadecimal correspondente ao código destas. A associação a ser feita é usar as chaves SW8 a SW5 para `Entrada_A` e SW4 a SW1 para `Entrada_B`. Note-se que as chaves são conectadas às entradas dos FPGA e, por dentro deste, são redirecionadas para gerar os valores das entradas em hexadecimal nos mostradores de 7 segmentos associados a `Entrada_A` e `Entrada_B`.

Uma vez estabelecido como entradas e saída de dados são tratadas, resta decidir como lidar com a entrada de controle da operação da ULA (`Operacao`) e as saídas qualificadoras (`N, Z, C` e `V`). No caso da primeira, foi usado um tipo enumerado para especificar um dentre 8 possíveis operações. Assim, associe os três botões do tipo *push-button* `BTN3, BTN2` e `BTN1` para especificar um código de 3 bits de entrada. A conversão destes 3 bits para os valores específicos do tipo enumerado (por exemplo `Op`) é discutida na Seção 5.3. Finalmente cada saída qualificadora é associada a um diodo emissor de luz (LED) da placa DIO4. Associe `N` a `LED8`, `Z` a `LED7`, `C` a `LED6` e `V` a `LED5`.

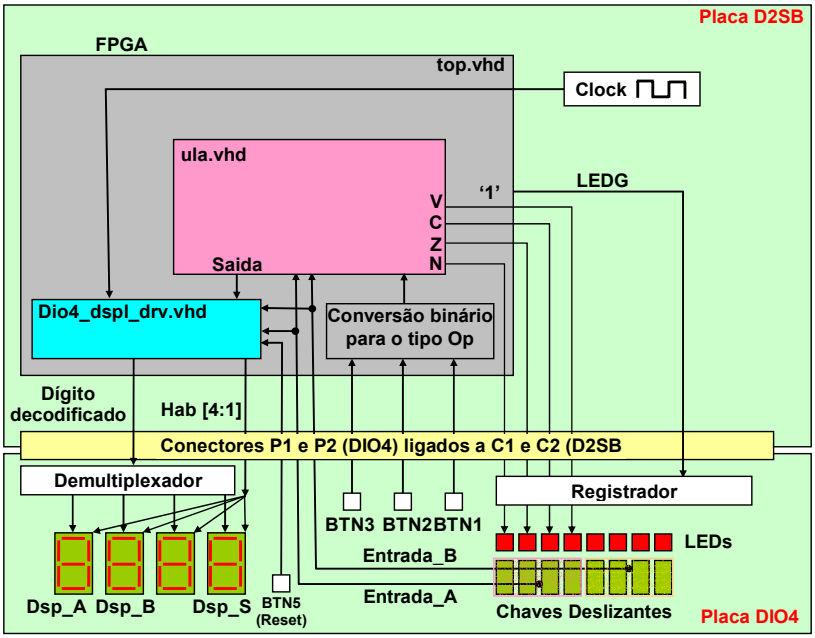


Figura 23 – Esquema geral da prototipação da ULA de 4 bits na plataforma D2SB/DIO4. Sinais Clock e Reset - ver Seção 5.3.

5.3 Criação do Arquivo `top.vhd`

Uma vez definido o uso de recursos de entrada e saída da plataforma D2SB/DIO4 para implementar a ULA de 4 bits, deve-se construir um arquivo `top.vhd`, conforme processo descrito no Laboratório 3. Este arquivo irá instanciar o projeto do módulo construído no Laboratório 4, mas não apenas isto, pois deverão ser feitas adaptações devido a particularidades da plataforma, conforme descrito a seguir.

A primeira particularidade é aquela já descrita no Laboratório 3, que dita a necessidade de criação de um sinal que deverá ser posto sempre em '1' e vinculado (via arquivo UCF) ao pino LEDG do conector entre placas D2SB e DIO4. Mais detalhes são encontrados no Laboratório 3. Isto deve ser feito sempre que pelo menos um dos LEDs da placa DIO4 deva ser usado, o que é o caso aqui.

A segunda particularidade diz respeito ao projeto, que usou um tipo enumerado para especificar a operação a realizada pela ULA, com óbvias vantagens de clareza obtidas na interpretação VHDL. Embora seja possível sintetizar hardware para tipos enumerados e obter inferência de vetores binários que os implementem de forma automática, isto requer conhecimentos avançados da ferramenta ISE que não se deseja explorar aqui. Assim, far-se-á uma operação explícita de conversão de `std_logic_vector (2 downto 0)` para o tipo enumerado denominado aqui de `Op` (os grupos podem ter dado outros nomes para este tipo no Laboratório 4). No arquivo `top.vhd`, deve-se criar na entidade a porta `Operacao` como sendo do tipo `std_logic_vector(2 downto 0)`. O código de conversão é apenas uma atribuição condicional dentro do arquivo top, devendo ser similar ao dado na Figura 24. Um sinal interno (`int_operacao`), do tipo Op, pode assim ser diretamente conectado à entrada `Operacao` da ULA.

```
-- Converte entrada std_logic_vector para tipo op, preparando instanciaçao da ULA
with Operacao select
       int_operacao <= soma          when "000",
                       subtracao     when "001",
                       incr          when "010",
                       decr          when "011",
                       op_and        when "100",
                       op_or         when "101",
                       op_xor        when "110",
                       op_not        when others;
```

Figura 24 – Exemplo de código de conversão do tipo `std_logic_vector` para `Op`.

A terceira particularidade diz respeito ao uso do *package* VHDL onde está definido o tipo enumerado. Este,

no Laboratório 4 fazia parte do mesmo arquivo que implementava a ULA. Contudo, a necessidade do arquivo `top.vhd` conectar-se a ULA faz com o que este último precise ter acesso ao *package*. Assim, deve-se alterar a forma original. O melhor (na maioria dos casos) é colocar o package em um arquivo VHDL separado e incluí-lo no projeto, fazendo referência ao mesmo nas entidades que necessitam de suas definições.

A quarta e última particularidade é mais complexa de tratar, sendo aquela ditada pela necessidade de uso do conjunto de 4 mostradores de 7 segmentos. Cada mostrador da placa DIO4 possui 9 pinos de controle de sua operação, 7 para especificar o estado de cada segmento, 1 para especificar o estado do ponto decimal à direita do número e 1 para habilitar/desabilitar o acendimento do mostrador como um todo. Isto perfaz um conjunto de 9x4=36 pinos do FPGA que necessitariam serem dedicados para comandar apenas o conjunto dos mostradores. Como isto é considerado um desperdício de pinagem do FPGA, os fabricantes da plataforma optaram por multiplexar os 4 mostradores em 1 único conjunto de 8 pinos e 4 sinais de 1 bit habilitadores da cada um dos mostradores, passando a usar apenas 1/3 da pinagem original requerida do FPGA (12 pinos). Por outro lado, todo circuito que necessite usar os mostradores a partir do FPGA deve internamente dispor de um hardware capaz de multiplexar temporalmente os até 4 valores de 8 bits em separado enviando-os para cada mostrador pelo mesmo conjunto de fios, em diferentes instantes de tempo. Quando um valor é enviado, apaga-se todos os mostradores, exceto aquele para onde o dado deve ser enviado. O processo funciona devido à diferença de velocidade entre o FPGA, operando na velocidade de MegaHertz (o *clock* de base da placa, por exemplo, é um sinal de 50MHz), e os fenômenos físicos que regem a funcionalidade dos mostradores. Acender e apagar um dos mostradores é uma operação que dura na ordem de dezenas ou centenas de milissegundos. O ideal é que o circuito de controle dos mostradores acenda cada mostrador por cerca de 1 a 2 milissegundos de forma circular. Uma freqüência de 512Hz a 1KHz é uma boa escolha. Considerando uma freqüência de 512Hz, a cada segundo cada um dos 4 mostradores fica aceso ¼ de segundo, mas é aceSo e apagado 128 vezes!! Como a elaboração de tal hardware é um tanto complexa, um módulo resolvendo o problema foi implementado e está disponibilizado na página da disciplina, no *link* http://www.inf.pucrs.br/~calazans/undergrad/laborg/dio4_dspl_drv.vhd. Este arquivo deve ser inserido no projeto e seu cabeçalho lido com bastante atenção pelo grupo para identificar como realizar a instanciação do *driver* de hardware dos 4 mostradores de 7 segmentos (**Nota: Não é estritamente necessário ler o código VHDL do arquivo, embora seja instrutivo procurar entender como o módulo funciona**). O módulo basicamente recebe 4 vetores de 6 bits (4 sendo o valor a ser mostrado em hexadecimal no mostrador, os outros dois sendo o valor do ponto decimal e uma habilitação do mostrador. Mais detalhes no cabeçalho do arquivo) e gera dois vetores, um de 8 bits contendo o valor decodificado a conectado a um dos 4 mostradores e um valor que habilita um dos 4 mostradores a receber este valor (ou em alguns casos, nenhum dos mostradores).

Após considerar todos os items descritos nas Seções 5.2 e 5.3, é possível obter um arquivo `top.vhd` com uma entidade que possui aparência similar àquela mostrada na Figura 25.

```
entity top is
    port ( clock, reset : in std_logic;
           Ent_A : in std_logic_vector(3 downto 0);
           Ent_B : in std_logic_vector(3 downto 0);
           Operacao : in std_logic_vector(2 downto 0);
           Anodo : out std_logic_vector(3 downto 0);
           Display : out std_logic_vector(7 downto 0);
           N, Z, C, V : out std_logic;
           LEDG: out std_logic -- só para placa D2SB/DIO4 Led Enable
           );
end top;
```

Figura 25 – Exemplo de aparência da entidade do arquivo `top.vhd`.

A arquitetura do arquivo `top.vhd` fica como tarefa para os grupos desenvolverem. Não se deve esquecer que esta arquitetura deve conter uma instância da ULA, uma instância do *driver* dos mostradores de 7 segmentos, um sinal que gera a habilitação do registrador dos LEDS da DIO4, a preparação dos sinais para conectar ao driver dos mostradores e sinais de *clock* e *reset*, necessários ao driver dos mostradores. O sinal de *clock* será o pino 182 do FPGA, conforme aparece no documento tabela_conversao_D2SB_DIO4.doc, enquanto que se deve associar ao sinal de reset ao *push-button* `BTN5` da placa DIO4.

5.4 A Fazer e a Entregar

Tarefa 1: Leiam o texto acima e sigam as orientações para criar o projeto da ULA no ambiente ISE. Usem os recursos vistos no Laboratório 3 para gerar o projeto. Este projeto deve conter, além do `top.vhdl` descrito na Seção 5.3, o arquivo VHDL simulado do Laboratório 4 (não se esqueçam de particionar o arquivo em dois, um para o *package* e um para a descrição da ULA em si, conforme descrito acima), o arquivo do *driver* dos mostradores, disponível na página da disciplina e, como já salientado no Laboratório 3, o essencial arquivo UCF com a associação de sinais a pinos. Sintetizem o projeto (sínteses lógica e física), gerem o arquivo `top.bit`, configurem o FPGA da placa e testem o projeto. Mostrem o projeto funcionando ao professor.

Tarefa 2: Uma vez funcionando o projeto alterem-no segundo algumas possibilidades lógicas. Por exemplo, no projeto proposto, dois mostradores de 7 segmentos apenas apresentam o valor das entradas constantes nas chaves deslizantes. E se se quisesse fazer o mesmo com a entrada de operação? Duas possibilidades ocorrem, usar o mostrador não usado para mostrar o código da operação em hexadecimal, ou usar alguns dos LEDs não usados para mostrar o mesmo valor em binário. Escolham uma destas opções e a implementem no projeto. Descrevam no relatório as alterações que esta mudança implicou no projeto.

Tarefa 3: Coloquem no relatório deste Laboratório os dados numéricos do projeto, incluindo a quantidade (absoluta e percentual) de *LUTs*, *Slices* e *Flip-flops* gastos pelo projeto, e o número de portas lógicas equivalentes.

6 Processador Cleópatra: Organização VHDL e Programação

Prática: Programação e simulação do processador Cleópatra
Recursos: Organização Cleópatra (VHDL), Simulador Active-HDL, Simulador Cleosim

6.1 Introdução e Objetivos

O objetivo fundamental deste Laboratório é introduzir a organização do processador Cleópatra, já estudada em aula na disciplina Organização de Computadores. Neste processo, objetiva-se fomentar o estudo da proposta desta organização por parte dos alunos, e conduzi-los à experimentação com a dita organização, mediante o emprego de simulação VHDL. Adicionalmente, objetiva-se desenvolver a capacidade de programação na linguagem de montagem do referido processador, através do emprego do ambiente Cleosim de edição, simulação e geração de código. Finalmente, objetiva-se exercitar o uso de operações de entrada e saída mapeadas em memória do processador, através do oferecimento de uma infra-estrutura mínima de suporte junto com o código VHDL do processador.

6.2 A Fazer e a Entregar

Tarefa 1: Busque os arquivos relativos ao presente laboratório, seja na *homepage* da disciplina ou diretamente em http://www.inf.pucrs.br/~calazans/undergrad/laborg/lab6_files.zip. Abra o simulador Active-HDL e use os arquivos da *homepage* como fontes para montar um novo workspace/projeto denominado, por exemplo, Cleo ou Cleopatra. Estude a estrutura deste projeto. O mesmo é composto por um arquivo readme_first.txt (leia ele imediatamente) explicando a estrutura geral do projeto, seis arquivos VHDL com a implementação do processador Cleópatra e de seu ambiente de execução, e um programa em linguagem de montagem do processador Cleópatra. Compile o projeto VHDL assim criado e certifique-se de que não há erros de sintaxe ou de elaboração do projeto.

Tarefa 2: Simule o projeto completamente. Com o testbench original fornecido, a simulação deve durar um pouco menos de 3700ns. Simule por este período. Certifique-se que todos os principais sinais aparecem na forma de onda, construindo esta manualmente. Se você não souber como isto pode ser feito, estude as referências disponíveis junto com o simulador, sob o item *Waveforms*. O principal tema a aprender aqui é como mostrar sinais de qualquer nível da hierarquia de projeto. **Responda à pergunta seguinte:** Quantos níveis possui a hierarquia deste projeto na sua parte mais profunda (que é o processador Cleópatra)? Justifique sua resposta. A aparência da forma de onda em termos de sinais visualizados deve ser exatamente igual à da Figura 26. Nesta Figura, apenas o final da simulação é apresentado (indicado pela subida do sinal *halt* no canto direito da Figura). Certifique-se de compreender bem o que significa cada um dos sinais mostrados na Figura, incluindo os sinais de controle do processador, os sinais de interface com a memória, os sinais que comandam a entrada e saída de dados entre o mundo externo e o processador, etc. Descreva estes grupos de sinais no relatório.

Tarefa 3: Modifique o programa dado como segue. Ao invés de calcular a soma de dois números de 16 bits, calcule o resultado da divisão de dois números inteiros sem sinal de 8 bits, gerando como saída o quociente (em saidah) e o resto da divisão (em saidal). Os números devem ser lidos da entrada de dados, primeiro o dividendo e depois o divisor. Comece gerando o código em linguagem de montagem da Cleópatra, simulando o mesmo no ambiente Cleosim. Após validar seu programa no ambiente Cleosim, simule o mesmo no ambiente Active-HDL. **Responda à pergunta seguinte:** é possível simular exatamente o mesmo código nos dois ambientes? Se não, explique porque e explique a solução empregada pelo grupo para validar o programa nos dois ambientes. Dica: lembre que os números a dividir podem ser distintos, e que somente existe uma posição de memória que mapeia a entrada de dados.

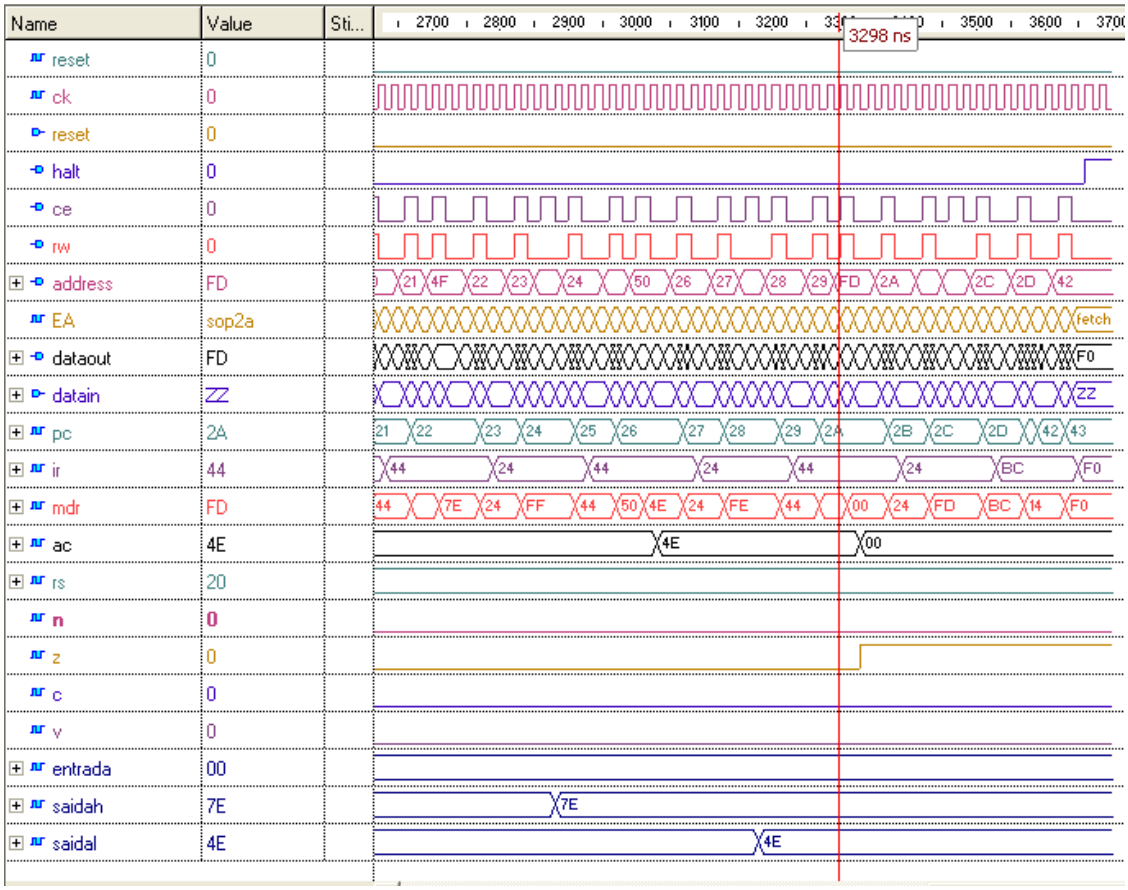


Figura 26 – Forma de onda da simulação do programa `soma_es.asm`.

7 Circuitos Seqüenciais e Máquinas de Estados Finitas

Prática: O exemplo da máquina de venda de refrigerantes em lata: especificação, projeto e implementação do circuito usando VHDL

Recursos: Simulador VHDL Active-HDL da empresa Aldec, Ambiente de Desenvolvimento ISE da Xilinx, Inc., e Plataforma Digilent D2SB/DIO4

7.1 Objetivos

Este trabalho tem como objetivo a familiarização com uma classe fundamental de circuitos digitais: os circuitos seqüenciais. Nestes, as saídas dependem não apenas do valor instantâneo das entradas, mas também de entradas passadas, ou melhor, da ordem de aparecimento destas. Esta ordem é armazenada internamente, normalmente de forma parcial, pelo circuito, e usada para definir as saídas junto com as entradas atuais. Em outras palavras, um circuito seqüencial possui uma memória interna, que armazena informações distintas em momentos distintos. Às informações armazenadas internamente por circuitos seqüenciais, dá-se o nome de estado interno ou simplesmente estado do circuito. Em particular, nos interessa aqui lidar apenas com circuitos seqüenciais síncronos, ou seja, aqueles onde a troca de um estado para outro é comandada por um sinal de relógio. Um modelo abstrato útil para representar circuitos seqüenciais são as máquinas de estados finitas (MEFs ou, em inglês, finite state machines, FSMs), também denominadas autômatos finitos, revisado nas aulas teóricas. Lembre-se que a implementação de FSMs sob a forma de hardware síncrono pode ser realizada seguindo o modelo estrutural da Figura 27. Nesta, nota-se que toda a parte seqüencial consiste de um registrador de estados controlado pelo sinal de relógio, e a parte combinacional consiste de dois conjuntos de funções Booleanas, uma para gerar a saída (função λ) e outra para calcular o próximo estado (função δ), ambas dependentes das entradas primárias atuais (Ii) e do estado atual armazenado no registrador de estado (Si). Com isto, dada uma codificação dos estados interna, basta gerar o hardware combinacional que implementa as duas funções para produzir uma descrição completa da FSM.

Em particular, este laboratório visa como atividade prática a implementação do hardware de uma máquina de venda de refrigerantes simplificada, como descrito mais adiante.

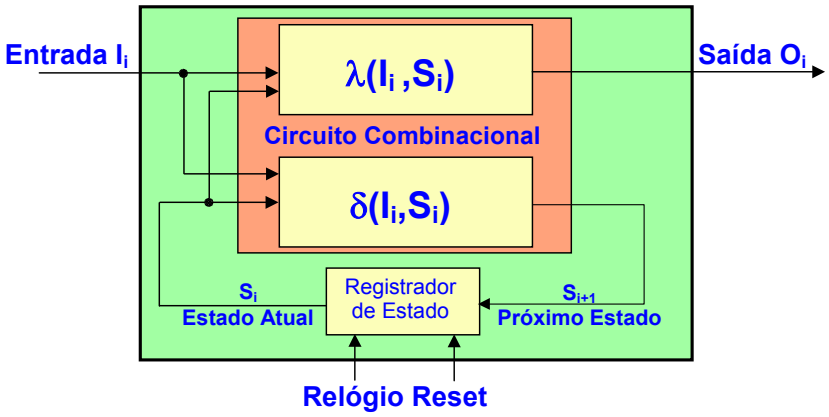


Figura 27 – Modelo estrutural genérico para FSMs síncronas.

7.2 Especificação da Máquina

A máquina deve ser capaz de fornecer dois tipos de refrigerantes, denominados MEET e ETIRPS. Estes estão disponíveis para escolha pelo usuário a partir de duas teclas no painel com o nome dos refrigerantes. Ambos refrigerantes custam R$1,50 e existe na máquina uma fenda para inserir moedas com uma mecânica capaz de reconhecer moedas de R$1,00, R$0,50 e R$0,25, e capaz de devolver automaticamente qualquer outro tipo de moeda ou objeto não reconhecido. Além disso, durante a compra, o usuário pode desistir da

transação e apertar a tecla DEV que devolve as moedas inseridas até o momento. Somente após acumular um crédito mínimo de R$1,50 o usuário pode obter um refrigerante. A devolução de excesso de moedas é automática sempre que o valor inserido antes de retirar um refrigerante ultrapassar R$1,50. Para evitar que situações esdrúxulas tenham de ser consideradas, tais como aquelas onde o usuário gera simultaneamente mais do que um evento de entrada, vamos supor que existe um codificador de prioridade que produz um código de uma única tecla (ou nenhuma tecla) sendo apertada a cada instante do tempo. Vamos supor também que existem circuitos para evitar que o fato de uma tecla ficar muito tempo apertada seja interpretado como vários apertos consecutivos da mesma tecla. As duas observações finais são hipóteses simplificadoras da especificação. No trabalho de implementação, deve-se assumir a existência externa do codificador de prioridade e dos circuitos de tratamento de tecla, veja o desenho na Figura 28. Uma terceira hipótese simplificadora consiste em ignorar a composição exata das moedas inseridas na máquina, atendo-se apenas ao montante total inserido.

7.3 Projeto da Máquina

O projeto inicial consiste em definir a interface do circuito de controle da máquina com o mundo externo e definir o comportamento deste circuito segundo uma máquina de estados finita. A interface pode ser depreendida a partir da especificação. Primeiro, cada tecla do painel corresponde a uma entrada de um bit. Tecla apertada será bit em '1' e tecla não apertada será bit em '0'. Isto vale para as teclas de pedido de refrigerante (MEET e ETIRPS) e para a tecla de devolução do dinheiro (DEV). O circuito de controle deve ser síncrono, e suponha que seu relógio será muito rápido comparado com as ações do usuário. Logo, cada uma destas teclas, uma vez apertada, irá gerar um sinal que fica muitos ciclos de relógio ativado. Outro ponto de entrada é a informação de moedas inseridas na fenda. Suponha que o circuito eletromecânico de reconhecimento de moedas gera um pulso de curta duração (longo apenas o suficiente para ser detectado pela próxima borda de relógio, e curto o bastante para não estar ativo em duas bordas consecutivas. Isto simplifica o projeto do controlador, constituindo-se em mais uma hipótese simplificadora) para cada moeda reconhecida como válida. Assim, isto corresponde a três outras entradas do circuito de controle, denominadas M025, M050, M100, cada uma destas entradas recebendo um pulso em '1' cada vez que moedas válidas de R$0,25, R$0,50 e R$1,00 são inseridas, respectivamente. As saídas da máquina correspondem a pulsos de devolução de moedas (D025, D050 e D100) e pulsos de liberação de refrigerante (LMEET e LETIRPS). Logo a interface do circuito de controle está completamente definida, tendo 8 entradas e 5 saídas. Esquematicamente, mostra-se o diagrama de blocos resultante desta discussão na Figura 28.

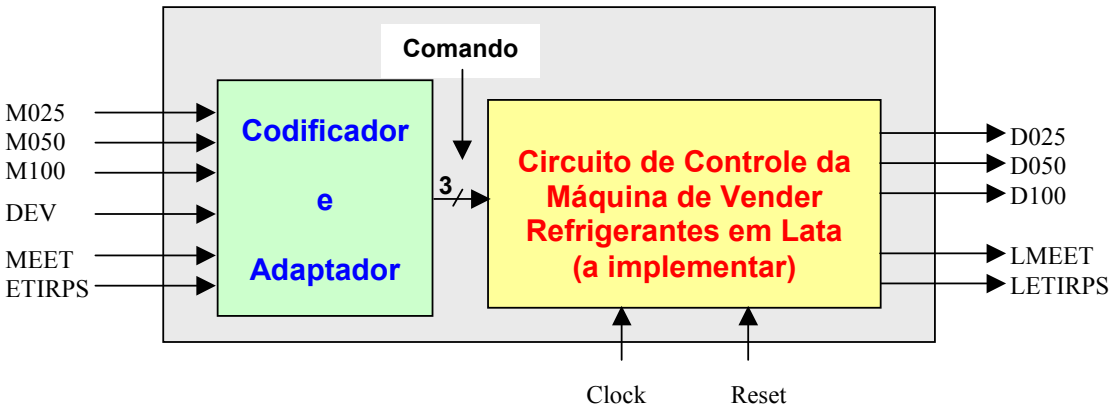


Figura 28 – Diagrama de blocos da máquina de venda de refrigerantes, mostrando a interface com o mundo externo.

Note-se na Figura 28 a separação do circuito Codificador e Adaptador, este responsável por realizar todas as tarefas das hipóteses simplificadoras. A saída do Codificador e Adaptador é um vetor de 3 bits que contém um **Comando**. Assuma a seguinte codificação para este vetor:

- **Nada**=nenhuma tecla apertada; **M025**=chegou moeda de R$0,25; **M050**=chegou moeda de R$0,50; **M100**=chegou moeda de R$1,00; **DEV**=pressionada a tecla para devolver as moedas colocadas; **MEET**=apertada a tecla de pedido de lata de refrigerante MEET; **ETIRPS**=apertada a tecla de pedido de lata de refrigerante ETIRPS.

O próximo passo é definir a tabela de transição de estados para o circuito de controle. Para fazer isto é necessário definir que informações serão usadas para indicar o estado da máquina. No caso, usar o valor total inserido até o momento é uma escolha adequada. Assim, a cada instante do tempo, podem ter sido inseridas moedas para perfazer qualquer valor entre R$0,00 e R$1,50, sem esquecer que sempre que o valor total exceder R$1,50 moedas em excesso são imediatamente devolvidas. Assim, a moeda de mais baixo valor (R$0,25) determina que precisa-se de 7 estados ao todo para representar qualquer combinação de moedas inseridas, inclusive nenhuma. Para gerar a tabela de transição de estados, basta definir as ações associadas a cada codificação do sinal **Comando**. Não se esqueça de considerar a ação do circuito quando nenhuma das entradas está ativa (**Nada**). Assim, temos a Tabela 4, que mostra a de transição de estados, onde as posições desta estão apresentando o próximo estado seguido da geração de sinais de saída. **Atenção** à convenção usada na Tabela 4: **se nenhum sinal de saída aparece, isto significa que todos valores inativos são gerados com para as saídas** (por exemplo, quando se está no estado S000 e a entrada é M025). Note ainda que muitas entradas da tabela não estão preenchidas.

7.4 A Fazer e a Entregar

Tarefa 1: Complete a tabela de transição de estados (Tabela 4) corretamente.

Tabela 4 – Tabela de transição de estados que descreve o comportamento da máquina de venda de refrigerantes.

Estado Atual	Comando de Entrada						
	Nada	M025	M050	M100	DEV	MEET	ETIRPS
S000		S025			S000	S000	
S025		S050			S000, D025		
S050						S050	
S075							
S100	S100			S150, D050			
S125							
S150							

Tarefa 2: Gere uma implementação em VHDL do circuito de controle da máquina de vender refrigerantes em lata, usando como referência a Figura 27.

Tarefa 3: Simule o circuito implementado. Para tanto, gere um *testbench* de simulação realista, testando algumas condições, tais como tentativa de tirar refrigerante sem crédito suficiente (pressionar a tecla de pedido de refrigerante sem crédito suficiente), tentativa de solicitar devolução de moedas com crédito nulo, etc.

Tarefa 4: Prototipe o circuito validado em VHDL na plataforma D2SB-DIO4. Escolha a pinagem adequada. Para visualizar o funcionamento na placa, o melhor é colocar para fora da máquina o estado desta. Assim, seu circuito vai ter como entradas os sinais de clock, reset, e o sinal comando. Como saídas, as cinco saídas binárias da máquina (podem ser associadas a leds da DIO4, sem esquecer do pino de habilitação destes), e o estado (que pode ser mostrado em um dos quatro mostradores de 7 segmentos (use o módulo acionador dos mostradores já visto em aulas anteriores dio4_dspl_drv). O sinal comando possui uma complicação adicional para ser prototipado. Lembrando que o clock da placa é de 50MHz, e que cada comando deve ser ativado am apenas uma borda de clock, é necessário usar um sinal que diga o instante exato em que o comando deve ser executado. Assim, adicionem um sinal que ative (chamado, por exemplo, **ativa**) o comando. Atribuam o sinal comando à três chaves deslizantes da DIO4 e o sinal ativa a um dos botões da DIO4. Além disso, quando alguém aperta o sinal ativa, este fica ativado por milhares ou milhões de ciclos de clock. Resolva este problema interpondo um módulo que recebe o clock da placa e a tecla ativa, e gera o clock da máquina de refrigerantes.

Um módulo testado é o dado pelo processo VHDL da Figura 29 abaixo:

```
gera_ck_int: process (clock, reset)
                    begin
                           if reset='1' then
                                  deb<=(others => '0');
                           elsif clock'event and clock='1' then
                                  deb <= ativa & deb(2 downto 1);
                           end if;
                  end process;

int_clock <= deb(1) and not deb(0);
```

Figura 29 – Processo e atribuição para, a partir de um clock rápido, e.g. 50MHz, gerar 1 pulso de clock único a cada vez que uma tecla lenta (ativa) é apertada.

O processo assume a existência de um sinal interno de 3 bits denominado deb (do inglês *debounce*), que opera como um registrador de deslocamento, e uma saída int_clock, que somente é '1' quando os dois últimos valores de déb são distintos, ou seja, quando há uma transição do valor de ativa. Para prototipar a máquina de refrigerantes, deve-se gerar um arquivo top, um arquivo UCF, alterar a máquina para que seu estado seja uma saída, não apenas um sinal interno, com possíveis conversões de tipo para mostrar o estado na placa.

8 Modelo Bloco de Dados – Bloco de Controle: Princípios

Prática: Comunicação síncrona: especificação, projeto, implementação e prototipação de circuito em VHDL seguindo o modelo Bloco de Dados – Bloco de Controle

Recursos: Simulador VHDL Active-HDL da empresa Aldec, Ambiente de Desenvolvimento ISE da Xilinx, Inc., e Plataforma Digilent D2SB/DIO4

8.1 Objetivos

Este trabalho tem como objetivo a familiarização com uma organização básica para a construção de sistemas digitais, o modelo Bloco de Dados/Bloco de Controle. Esta organização está por trás da maioria dos sistemas digitais empregados na prática, desde simples sistemas de comunicação serial como existem entre um mouse ou teclado e um computador hospedeiro, até processadores programáveis complexos. A essência do modelo é mostrada abaixo, na Figura 30. O modelo, como seu próprio nome expressa está dividido em duas partes essenciais. Primeiro há o Bloco de Controle, que recebe *Controles* do mundo externo e *Qualificadores* da outra parte do modelo e gera informações de *Status* para o mundo externo e/ou ordens para que *Ações* sejam realizadas no mundo externo. Esta parte constitui-se tipicamente de uma máquina de estados finita, como definido em laboratório anterior. A segunda parte, o Bloco de Dados, recebe do mundo externo *Dados de Entrada* e *Comandos* do Bloco de Controle, e produz para este último *Qualificadores*, que o Bloco de Controle pode usar para tomar decisões. Além destes o Bloco de Dados gera os *Dados de Saída* para o mundo externo.

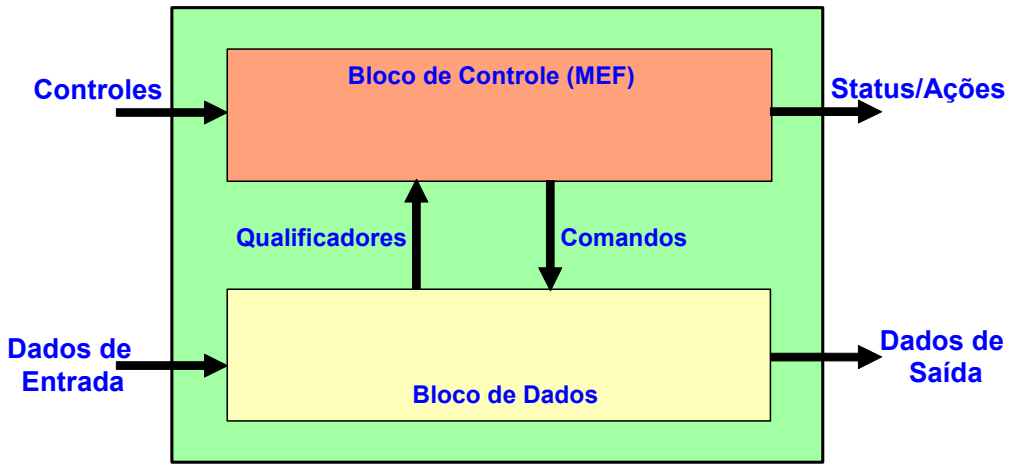


Figura 30 – Estrutura genérica do Modelo Bloco de Dados/Bloco de Controle de organização de sistemas digitais.

O Bloco de Dados é formado por três tipos de recursos: recursos de armazenamento de dados (registradores, memórias bidimensionais, etc.), recursos de processamento de dados (somadores, comparadores, ULAs, deslocadores, etc.) e recursos de interconexão entre os dois outros tipos anteriores. O Bloco de Dados é tipicamente um escravo do Bloco de Controle. O primeiro realiza em si o processamento de informação do sistema. O Bloco de Controle, por outro lado comanda a realização de ações e sincroniza a comunicação dos resultados do processamento de informação no Bloco de Dados com o mundo externo. Nem todos os fluxos de informação descritos na Figura precisam necessariamente existir em um sistema específico. Por exemplo, alguns sistemas digitais mais simples podem prescindir dos *Qualificadores*, se nenhum resultado do processamento for relevante para as ações do Bloco de Controle. Outro exemplo são sistemas internamente autônomos, que não precisam de *Controles* externos para realizar suas tarefas.

A experiência com implementação de sistemas digitais que sigam o modelo Bloco de Dados/Bloco de Controle é o objetivo maior deste Laboratório, preparando a experiência com processadores no Laboratório seguinte. Para tanto, utilizar-se-á o projeto de um sistema muito simples de comunicação síncrona entre dois sistemas digitais, assumindo que ambos são implementados segundo o modelo aqui em estudo. Poder-se-ia argumentar que o exemplo aqui é por demais simples para que seja necessário utilizar um esquema tão

elaborado quanto o modelo Bloco de Dados/Bloco de Controle, mas o objetivo aqui é didático. O estudo detalhado da implementação do processador Cleópatra, por exemplo, fornece uma ampla justificativa para separar fluxos de informação de controle e de dados, dada a complexidade de sua operação e o grau de inter-relação entre suas partes. E isto mesmo o processador sendo por demais simples, mesmo se comparado com o mais simples dos processadores comerciais como membros da família de processadores PIC da Microchip Technologies ou da família Intel MCS51 (e.g. o 8051).

8.2 Especificação do Projeto de Comunicação Serial Síncrona

O projeto a ser realizado consiste na implementação de dois sistemas que se comunicam de forma serial (bit a bit) através de um fio. Cada comunicação unitária consistirá no envio de 1 byte (8 bits) de dados gerado de um *sistema produtor* para um *sistema consumidor* através de um único fio. Assim, o byte de dados deve ser enviado 1 bit de cada vez pelo fio. Alguma forma de sincronização de envio deve ser empregada e alguma forma de identificar início e fim de transmissão deve ser provida, como descrito a seguir.

A forma de trabalho para este Laboratório é a seguinte. Cada grupo de alunos deve implementar e validar separadamente ambos produtor e consumidor em VHDL, validando cada um individualmente, através de dois testbenches. Em seguida, produtor e consumidor devem ser unidos em um projeto único, e simulados com outro testbench. A prototipação deve ser realizada por dois grupos de dois alunos, onde um grupo prototipa o produtor e o outro prototipa o consumidor, cada um em uma placa distinta. As duas placas devem estar previamente conectadas através de fios, de tal forma a permitir a comunicação de forma correta (Ver Seção *A Fazer e Entregar* para mais detalhes).

O produtor deve receber do mundo externo um dado de 8 bits (proveniente, por exemplo, das chaves deslizantes da placa DIO4), e um sinal de controle que diz quando este dado deve ser adquirido e enviado serialmente para o consumidor de forma serial. Além disto, o produtor deve possuir uma interface de transmissão serial para comunicar-se com o consumidor.

O consumidor deve possuir uma interface de recepção de dados compatível com a do produtor (ver discussão abaixo), e deve ter uma saída de 8 bits (conectada, por exemplo, a dois mostradores de 7 segmentos da placa DIO4), além de uma sinal que diga quando este dado é válido.

Cada um dos módulos deve seguir, para sua implementação, o modelo Bloco de Dados/Bloco de Controle. No relatório, os alunos devem classificar claramente cada um dos sinais externos e internos do produtor e do consumidor, segundo a nomenclatura da Figura 30.

Embora o projeto a ser desenvolvido seja extremamente simples, quase trivial, ele não é desprovido de riscos em sua realização. Primeiro, deve-se decidir como realizar em detalhe a comunicação entre produtor e consumidor. Mesmo que tenha sido especificado o uso de comunicação serial, existem várias formas de implementar tal comunicação.

Uma das formas é a comunicação assíncrona via sinais ditos de “aperto de mão” (do inglês, *handshake signals*), onde a cada dado a enviar tem sinalizada sua presença e requisição de envio pelo produtor através de um sinal de controle (que pode se chamar *req*, abreviatura do inglês *request*), e um sinal de status do consumidor, que anuncia a captura do dado com sucesso (que pode se chamar *ack*, abreviatura do inglês *acknowledge*). Esta forma é útil em sistemas síncronos apenas se estes podem garantir uma relação de fase entre os relógios de cada lado da comunicação. Este não é o caso ao fazer a comunicação entre duas placas que possuem fontes de relógio diferentes. Na placa D2SB, existe um oscilador a cristal de alta precisão com freqüência de 50MHz. Embora duas placas iguais, com o mesmo tipo de cristal possam garantir freqüências idênticas, nada pode ser dito da relação de fase entre os relógios de duas placas distintas, gerando incerteza quanto ao momento exato de amostrar um dado corretamente no envio entre placas. Isto pode ser contornado com sinais de controle adicionais, ou aumentado a complexidade das máquinas de estados de envio e recepção, mas em geral não se trata de um processo simples.

Um segunda forma que simplifica o projeto é enviar, além do dado, um sinal de sincronização do produtor para o consumidor, e fazer este sinal ser o relógio do consumidor. O próprio clock do produtor pode ser repassado como sinal de sincronização, enquanto que o cristal da outra placa é ignorado. Este é o esquema a ser usado aqui. Esta escolha simplifica a sincronização entre produtor consumidor, mas acrescenta um fio entre os sistemas, aumentando o custo da comunicação. Ademais, se os fios da comunicação forem longos ou de comprimentos distintos, os tempos de propagação do dado e do sinal de sincronismo podem ser

diferentes, causando novos problemas. Para reduzir estes últimos, pode-se usar bordas de relógio distintas no produtor e consumidor (por exemplo, fazer o produtor gerar/disponibilizar dados na borda de subida e fazer o consumidor adquirir dados na borda de descida do relógio). Uma questão adicional importante para sintetizar o projeto é que não é qualquer pino do FPGA que pode ser usado como pino de entrada de clock, devido à natureza especial deste sinal. O software ISE detecta que um determinado sinal é de relógio e, se o pino associado a este sinal pelo arquivo de restrições (.ucf) associar o sinal a um pino inadequado, ocorre um erro na fase de mapeamento do projeto (primeira fase da síntese física). Para evitar este erro, recomenda-se o uso do pino GCLK1 do FPGA, pois este não é usado pelo cristal da placa (que usa a entrada GCLK2), e está disponível em um dos conectores externos da placa D2SB.

Um outro cuidado muito importante ao prototipar o sistema como um todo é garantir a referência de tensão elétrica entre as placas do produtor e do consumidor. Para tanto, os sinais de terra (GND, do inglês *ground*) das duas placas podem ser interligados (e neste caso, devem).

8.3 A Fazer e a Entregar

Tarefa 1: Implementar em VHDL os módulos produtor e consumidor, e validá-los separadamente, cada um com um testbench. Guarde estes projetos

Tarefa 2: Criar um novo projeto que instancie o produtor e o consumidor, validando-os juntos, através de um novo testbench.

Tarefa 3: Prototipar o projeto como descrito antes, produtor e consumidor em placas distintas, comunicando-se através de fios externos. Mostre o projeto funcionando ao professor.

9 Processador Cleópatra – Prototipação, Programação e Entrada e Saída

Prática: Comunicação síncrona: especificação, projeto, implementação e prototipação de circuito em VHDL seguindo o modelo Bloco de Dados – Bloco de Controle

Recursos: Simulador VHDL Active-HDL da empresa Aldec, Ambiente de Desenvolvimento ISE da Xilinx, Inc., e Plataforma Digilent D2SB/DIO4

9.1 Objetivos

Os objetivos deste Laboratório são: (1) dominar o fluxo de prototipação do processador Cleópatra; (2) compreender os acréscimos feitos para permitir a interação entre entidades externas ao subsistema processador-memória e este subsistema; (3) implementar um novo periférico, integrá-lo ao subsistema processador-memória através da interface fornecida e mostrar sua utilização por software executando no processador.

9.2 Prototipação do Processador Cleópatra

Nesta parte do Laboratório, os grupos devem prototipar o processador Cleópatra sobre a plataforma D2SB/DIO4, a partir dos arquivos de distribuição constantes na homepage da disciplina, no link referente a este Laboratório (http://www.inf.pucrs.br/~calazans/undergrad/laborg/cleohw_distribution_2006.zip).

O primeiro passo é ler cuidadosamente o arquivo README_First.txt constante na distribuição citada. Lá o esquema geral de como o processador Cleópatra pode ser prototipado, programado e usado sobre esta plataforma é devidamente explicado. A distribuição inclui um exemplo de programa que usa os meios de entrada e saída fornecidos pela distribuição, na realidade um conjunto de acréscimos de hardware ao subsistema processador-memória para habilitar executar operações de entrada e saída de maneira quase transparente na programação. Estes acréscimos estão implementados parte no arquivo modulos.vhd e parte no arquivo top.vhd.

Uma vez compreendida a implementação do sistema de prototipação do processador, deve-se prototipar o processador e testar a execução do programa exemplo fornecido.

9.3 Acréscimo de um periférico ao Processador Cleópatra

Após prototipar com sucesso o processador e experimentar com o programa exemplo, vale a pena estudar também os conteúdos dos arquivos modulos.vhd e top.vhd, antes de empreender a criação de um novo periférico para acrescentar ao subsistema processador-memória.

Uma vez compreendido o processo de realização de entrada e saída via os acréscimos de hardware, pode-se partir para implementar o novo periférico. A funcionalidade do novo periférico não é em si tão importante quanto o processo de interação deste com o processador através da interface fornecida nesta distribuição. Na distribuição fornecida, os 4 últimos endereços do mapa de memória do processador são reservados para operações de entrada e saída, sendo que no esquema implementado, três posições correspondem à geração efetiva de saídas (escritas nos endereços FD, FE e FF) e duas posições correspondem à acesso efetivo a entradas (leitura dos endereços FC e FD). As saídas podem ser compartilhadas com entradas do periférico, e pode-se facilmente decodificar mais uma saída para gerar entrada para o periférico. Assim, o periférico pode receber até quatro entradas de 8 bits do processador. A saída do periférico não pode, no entanto, compartilhar linhas com os dispositivos de entrada. Contudo, é possível facilmente estender o processo de decodificação para aceitar até duas saídas de 8 bits do periférico. Eliminando-se os periféricos de entrada, as saídas do periférico pode ser até quatro com o esquema de decodificação proposto.

Cada grupo deve elaborar um periférico distinto, definir e documentar sua interface com o subsistema processador-memória. Uma vez feito isto, o periférico deve ser implementado em VHDL e validado via

testbench. Depois de validada a funcionalidade do periférico, este deve ser integrado ao subsistema processador-memória e o todo deve ser simulado via um testbench adequado.

Para ilustrar o processo completo de criação e integração de um periférico, apresenta-se aqui o esboço de um estudo de caso. Um dos tipos de periféricos mais simples são coprocessadores que aumentam o poder do processador Cleópatra. Por exemplo, sabe-se que o processador Cleópatra não possui uma instrução de subtração. Poder-se-ia então criar um periférico que realizasse esta operação. O processador, para usar este periférico teria um programa que em algum momento executa o seguinte: (1) escreve os dados a serem subtraídos em dois registradores de entrada e saída usando, por exemplo, as instruções STA 0FEh e STA 0FFh; (2) gera um comando de disparo da execução da subtração pelo periférico usando, por exemplo, STA 0FDh; (3) fica em laço lendo uma posição de entrada (usando, por exemplo, LDA 0FEh), até que o periférico escreva aí um código de controle informando a conclusão da sua operação. Ao mesmo tempo, neste laço se desabilita o comando de disparo da execução; (4) adquire o resultado da operação via, por exemplo LDA 0FFh e o escreve em um dispositivo de saída. Enquanto isto, o periférico possui uma máquina de estados que faz o seguinte: (1) lê um sinal na sua entrada até que este contenha o valor que dispara a sua execução; (2) calcula a subtração das duas entradas de dados, colocando o resultado na sua saída de dados; (3) gera uma informação avisando que concluiu a subtração e coloca em uma saída de status, voltando em seguida ao passo (1).

9.4 A Fazer e a Entregar

Tarefa 1: Definir o periférico a implementar, bem como sua interface com o subsistema processador-memória, propondo o mínimo possível de alterações no código fornecido.

Tarefa 2: Implementar em VHDL o periférico e validá-lo via testbench.

Tarefa 3: Integrar o novo periférico ao subsistema processador-memória, validando o todo via simulação, com um novo testbench. Prototipar o novo hardware. Mostrar o projeto funcionando ao professor.