Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática

LABORATÓRIO DE ORGANIZAÇÃO DE COMPUTADORES

TEXTO DE APOIO PARA LABORATÓRIOS

Autores: CÉSAR A. M. MARCON
EDUARDO A. BEZERRA
FERNANDO G. MORAES
NEY L. V. CALAZANS

2003/1 (Versão 1.0)

SUMÁRIO

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 Foundation 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. Não use utilitários externos de compressão/descompressão, pois a hierarquia pode não ser salva de forma correta!!!

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 existe 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 pela 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 partículas metálicas. Ainda, sistemas ópticos como discos compactos (CDs) usam propriedades reflexivas de um plástico. E assim por diante.

Neste curso, 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, sendo a medida de valores digitais associados a estas grandezas um caso especial das correspondentes analógicas sobretudo 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 (denominada *função resposta* ou *resposta* do dispositivo) que estes dispositivos definem sobre as grandezas. As grandezas elétricas mais relevantes são a **tensão** elétrica (ou **voltagem**) e a **corrente elétrica**. Além desta existe a **resistência**, a **capacitância**, 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** (ver definições das três últimas grandezas ao final desta Seção), etc. 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 a lei de Ohm (**V=R.I**), que relaciona tensão e corrente. Esta equação define uma reta num gráfico **IxV**, cuja inclinação é definida pelo valor de resistência do resistor, **R**. Tipicamente, a tensão (medida em Volts: V) aplicada ao resistor gera uma corrente elétrica **I** (medida em Ampères: A) que é proporcional ao valor da resistência (medida em Ohms: 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.

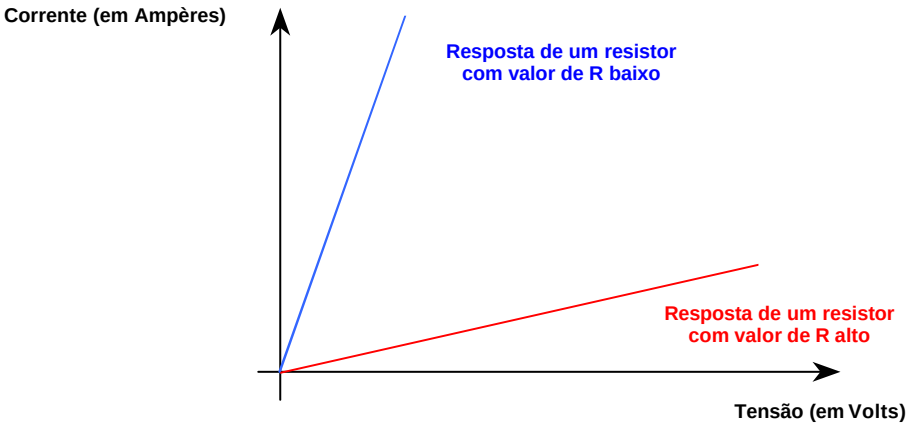


Figura 1 Gráficos típicos IxV para resistores, conforme a Lei de Ohm. Uma reta vertical corresponderia em tese a um resistor de valor 0, ou um curto-circuito. Uma linha horizontal corresponderia em tese a um resistor de valor infinito, ou circuito aberto.

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.

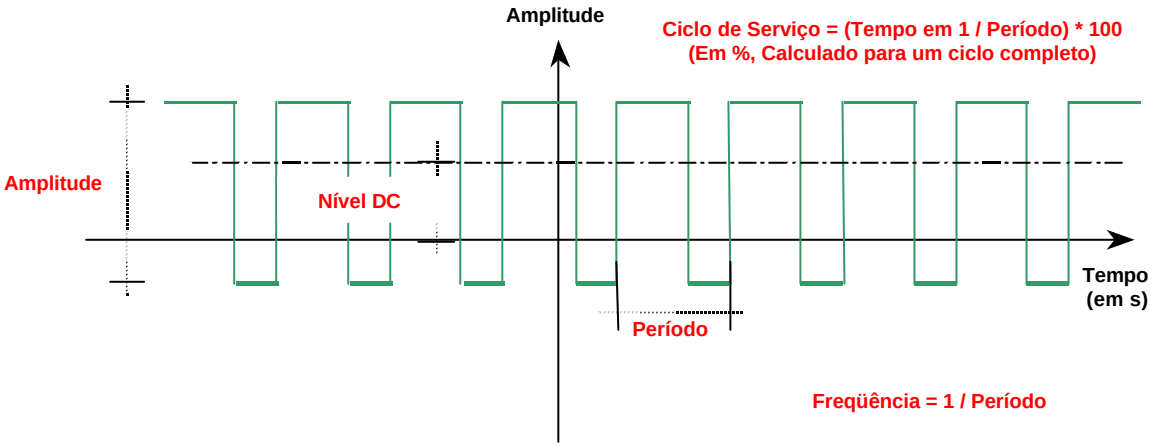


Figura 3 - Gráfico típico Amplitude versus Tempo para forma de onda periódica quadrada, ilustrando as grandezas associadas: freqüência, período, ciclo de serviço, amplitude e nível DC.

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 ao 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.

1.2.1. Medida de Sinais Elétricos – o Multímetro

Para gerar sinais elétricos é necessário ter controle sobre o valor de cada uma das grandezas do sinal a gerar. Por exemplo, para gerar a tensão elétrica equivalente a 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 a tensão (Corrente Alternada, CA ou Corrente Contínua, CC) a corrente (CA ou CC) e a resistência. Outras medidas que podem ser eventualmente medidas são a capacitância elétrica, a freqüência de um sinal periódico, a continuidade elétrica (caso especial de medida de resistência), o 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

[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.

do valor medido, e, quando aplicável, uma indicação de que a medida exceda a capacidade de representação (representado pelo texto **OL.** representando **Off-Limits value**);

- 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 deste laboratório;
- 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 um preto (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.

1.2.2. 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.

1.2.3. 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, computar 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.

A Figura 2 apresenta um gerador de ondas fornecendo uma freqüência de 7.84 kHz e esta freqüência sendo medida 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, o TDS-210 da Tektronix, são os seguintes:

- Uma tela de cristal líquido, contendo: 1 área gráfica de 10*8 quadrados de 1cm 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 engatilhada) (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 Entregar

Tarefa 1: Identifique na Figura 2 o multímetro. Se já não estiver montado, monte o multímetro na sua bancada, e estude seu painel.

Tarefa 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: 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: 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: 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: Para testar a compreensão do funcionamento do aparelho, use-o para gerar uma onda específica. Na Tabela 1 existem 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: 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: 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 pontas às pontas da saída 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*, 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? Depois de 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 Introdução a Sistemas de CAD, Projeto com Esquemáticos e Circuitos Combinacionais

Prática: Implementação de somador de 8 bits – captura, hierarquia e simulação funcional

Recursos: Sistema de CAD Foundation da empresa Xilinx, Inc.

2.1 Objetivos

O objetivo deste laboratório é fornecer aos alunos ferramental básico usado no projeto, validação e implementação de sistemas digitais mediante emprego de software do tipo sistema de Projeto Auxiliado por Computador (PAC, do inglês, *Computer Aided Design*, ou CAD) para sistemas digitais eletrônicos (em inglês, Electronic CAD ou ECAD). Este conjunto de ferramentas é fundamental para o restante da disciplina. Ao final deste laboratório o aluno deve estar apto a usar os recursos básicos de um sistema comercial de CAD eletrônico para realizar a captura de projeto e a validação do mesmo através de simulação no nível lógico de abstração, bem como dominar os princípios de projeto hierárquico no nível lógico de abstração de sistemas digitais. O objetivo específico aqui é implementar de forma modular e hierárquica um somador de 8 bits, a partir de somadores de 1 bit. Será enfatizado durante a aula o processo de simulação lógica ***funcional*** (sem atraso nas portas lógicas) e utilização de ***scripts***.

2.2 Uso do Foundation para Edição de Esquemáticos

1. Para abrir o FOUNDATION (Figura 4), clique no ícone correspondente. Se não houver este ícone no desktop, executar c:/fndtn/active/exe/pcm.exe. Caso não esteja instalado na máquina, chamar o professor.

- *seta 1: indica chamada ao editor de portas lógicas ou editor de esquemáticos.*

- *seta 2: indica chamada ao simulador de portas lógicas ou simplesmente simulador lógico.*

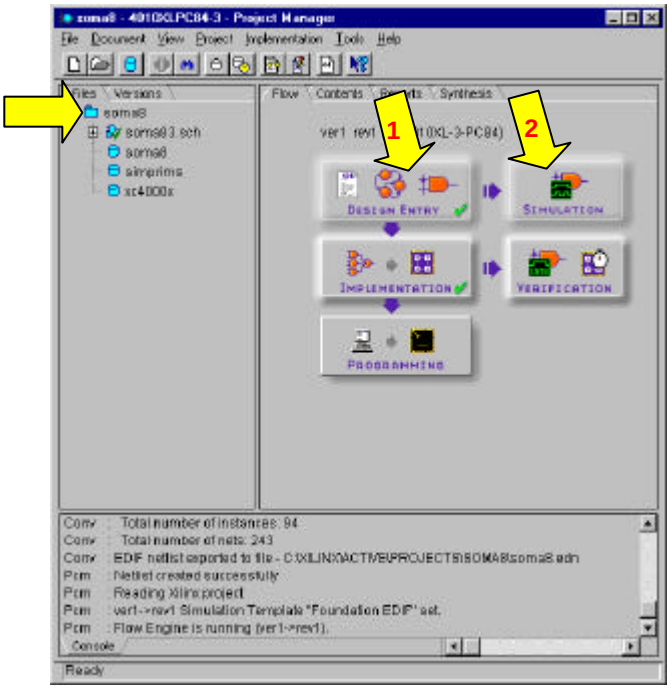


Figura 4 – Janela Principal doGerenciador de Projeto.

2. Para abrir um novo projeto (Figura 5), vá para o menu ➔ **File** ➔ **New project** (o símbolo ➔ indica menus e/ou submenus).

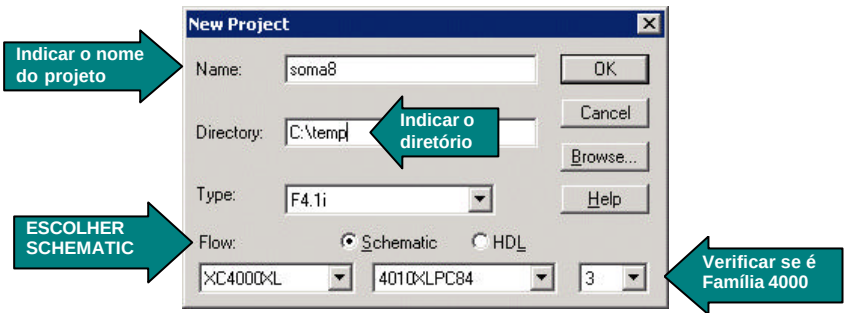


Figura 5 - Criação de um novo projeto.

MUITO IMPORTANTE:

- Trabalhar sempre no disco local (c:) (evite o disco h:, pois este é muito lento. Após o projeto realizado é possível armazenar o projeto em um arquivo .zip e guardá-lo no seu disco h:). Não utilizar disquete, exceto para fazer cópias de salvaguarda ao final do trabalho.
- Verificar se o dispositivo corresponde ao dispositivo da placa (família XC4000XL, componente 4010XLPC84 velocidade -3).

3. Para abrir o editor de esquemáticos, clicar no botão "Schematic Editor" da janela principal (*ver seta 1* - Figura 4).
4. Os comandos a serem utilizados na edição de esquemáticos são de 3 classes: inserção de símbolos, inserção de fios e inserção de pinos (os pontos de entrada e saída do circuito, ou seja, sua conexão ao mundo exterior). A janela inicial do editor é apresentada na Figura 6.

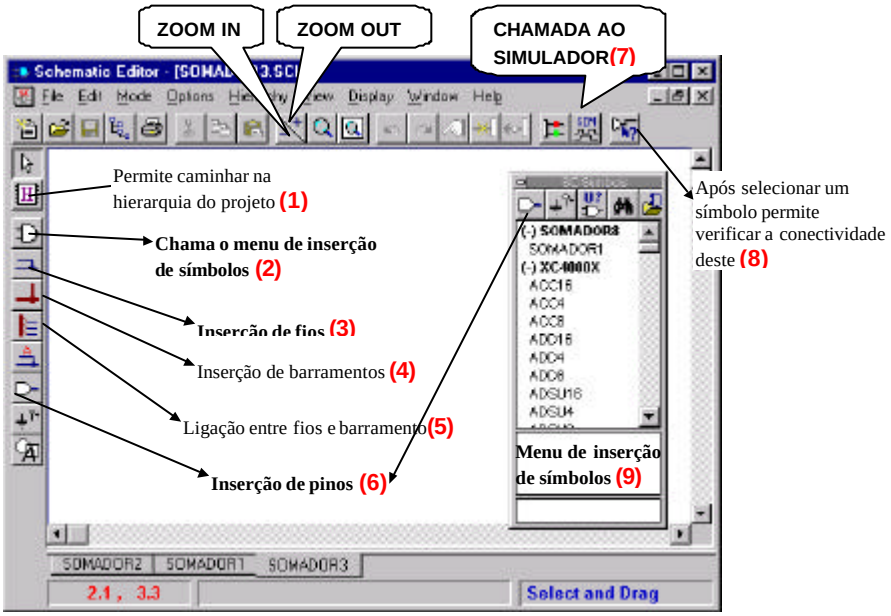


Figura 6 - Editor de Esquemático

Para criar um esquemático:

a. insira as portas lógicas: clique no botão (2), no menu (9), selecione a porta lógica desejada ou escreva seu nome na parte inferior do menu.
b. insira os pinos de entrada/saída do somador, via botão (6). A janela da Figura 7 deve ser preenchida para cada pino.

Figura 7 - Inserção de Pino.

c. realize as conexões entre as portas lógicas e os pinos, clicando no botão (3) - Figura 6, depois na primeira extremidade do fio, arrastando-se o mouse até a extremidade seguinte (podes clicar em pontos intermediários da conexão para realizar quebras), terminando-se com um clique na extremidade final da conexão.

5. Uma vez concluído o esquemático pode-se chamar o simulador via o botão **(7)** no editor de esquemático ou via seta 2 no gerenciador de projeto.

2.3 Uso do Foundation para Simulação Lógica

O objetivo deste item é verificar o comportamento de um circuito digital usando simulação exaustiva. Para realizar a simulação utilizaremos um ***script***. O script permite definir: (1) quais sinais serão definidos como entrada; (2) quais sinais serão observados e (3) o tempo de simulação. Para invocar o editor de script selecione:

tools → script editor → create empty script

O *script* da Figura 8 exemplifica os comandos necessários à simulação de um circuito.

```
;simulacao relativa ao somador full-adder        COMENTÁRIO

delete_signals
restart                                          comandos de inicialização
stepsize 10 ns

wfm a 0ns=0 (5ns=1 5ns=0)*1000
wfm b 0ns=0 (10ns=1 10ns=0)*1000                 formas de onda (entradas)
wfm cin 0ns=0 (20ns=1 20ns=0)*1000

watch sum                                        sinais de saída
watch cout

sim 200ns                                        tempo de simulação
```

Figura 8 - Script para simulação do somador completo.

O comando básico para definir formas de onda periódicas é waveform ou wfm. A sintaxe do comando é: wfm <nome do sinal> <tempo inicial>=<valor> (<intervalo1>=<valor> < intervalo2> =<valor> < intervalo3>=<valor> ……..)*<número de repetições

Uma vez escrito e armazenado o script, deve-se executá-lo. Para isto, fechar o editor de scripts e no simulador lógico fazer: ***file → run script file → [nome dado na edição do script]***

A Figura 9 apresenta as formas de onda de um circuito digital, assim como alguns botões úteis para a simulação.

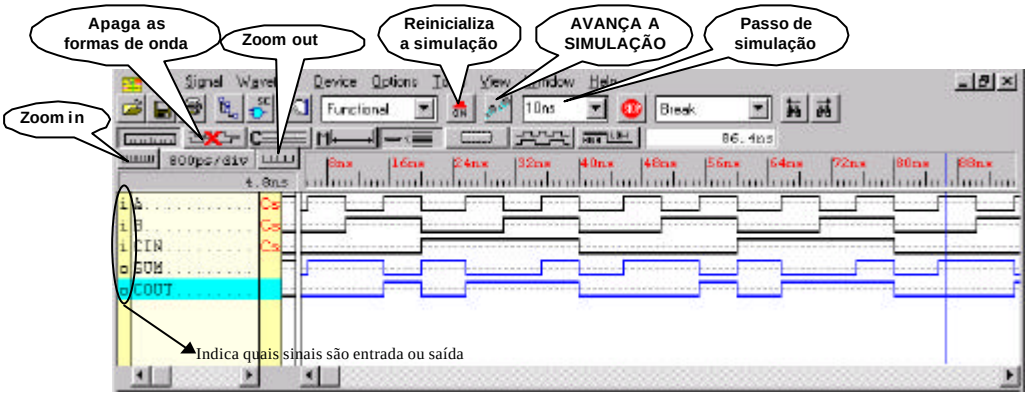


Figura 9 – Janela principal do simulador lógico.

Algumas dicas úteis:

1. Para reinicializar a simulação clicar no botão correspondente a esta função, e depois pode-se executar a simulação passo a passo no botão correspondente.
2. Para apagar as formas geradas antes de reinicializar uma simulação também há um botão correspondente.
3. Para visualizar um sinal interno do circuito, não definido no script. Ir no editor de esquemáticos e colocar um *probe* (ponteira) sobre o sinal desejado - Figura 10. No editor de esquemático é inserido um retângulo sobre o sinal, e no simulador um novo sinal. Reinicialize a simulação, apague as formas de onda e execute passo a passo a simulação. **Faça este procedimento para os sinais internos do somador. ESTE É UM PROCEDIMENTO IMPORTANTE PARA DEPURAR CIRCUITOS COMPLEXOS.**

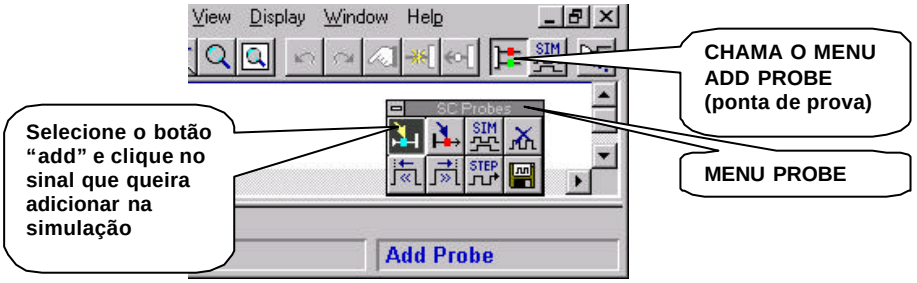


Figura 10 - Inserção de pontas de prova (probes).

2.4 A Fazer e Entregar

Tarefa 1: O circuito somador completo de 1 bit é capaz de somar três bits quaisquer, possuindo 3 entradas: A, B e Cin (Carry in, ou vai-um de entrada) e duas saídas: Soma e Cout (Carry-out, ou vai-um de saída). Determinar as equações do FA, utilizando tabela verdade e mapas de Karnaugh.

Tarefa 2: Faça a implementação do FA na forma de um diagrama de esquemáticos.

Tarefa 3: A partir do script da Figura 8 realize a simulação do FA. As formas de onda obtidas devem corresponder as formas de onda da Figura 9 (Observe que as palavras reservadas são escritas em vermelho pelo ***script editor***. Para conhecer a função de cada uma, simplesmente selecione o comando e pressione F1. Usar o help).

Tarefa 4: A partir do diagramas de esquemáticos do somador de 1 bit (FA) implemente um somador de 8 bits de forma hierárquica. Siga os passos abaixo:

1. A partir do esquemático do FA, criar um símbolo referente a este para ser utilizado pelo esquemático pai (somador de 8 bits).

- Escolha a opção de menu (**Hierarchy ➔ Create Macro Symbol from Current Sheet**). A janela da Figura 11 é apresentada, e normalmente deve-se apenas clicar em OK.
- Você pode editar as caraterísticas do novo símbolo (nome, comentário, etc), clicando em seguida sobre o botão OK. Dica: responda *não* à pergunta "queres editar símbolo?" após o OK.
- Agora, existe um novo símbolo que representa este circuito disponível na biblioteca, com o nome indicado na opção *symbol name*.

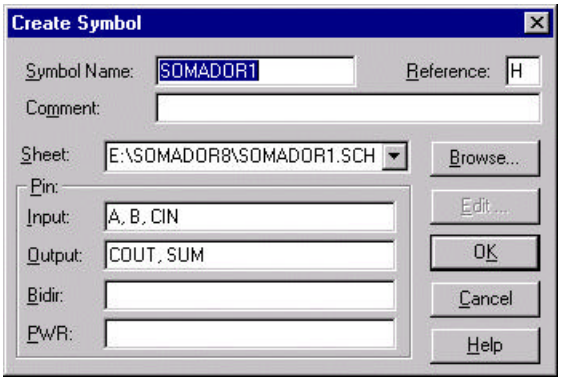
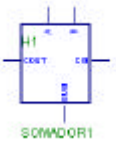


Figura 11- Criação de um símbolo a partir de uma folha de esquemáticos.

2. Abrir nova folha, (opção de menu File ➔ New Sheet do editor de esquemáticos) e inserir nela OITO (8) células iguais, instâncias do símbolo que você acabou de criar no item 1. Para colocar as instâncias do somador recém criado, ir no menu de inserção de símbolos (botão 2 - Figura 6), e selecione o somador pela lista ou escreva seu nome na parte inferior do menu.
3. Como a disposição das entradas e saídas dificulta a edição de esquemático, podemos modificar o símbolo gerado automaticamente para o somador de um bit, a fim de simplificar a edição (Figura 12). Utilizar para isto a ferramenta de editor de símbolos.

Símbolo original do somador



Modificar o símbolo, clicando duas vezes no símbolo original, selecionado a opção *symbol editor*. No editor de símbolos colocar as entradas na parte superior, cin na direita alinhado com o cout na esquerda e a saída sum na parte inferior. Salvar o símbolo e voltar para o editor de esquemático.

Figura 12 - Modificação de um símbolo através do editor de símbolos.

4. O esquemático neste ponto deve ser semelhante ao esquemático da Figura 13.

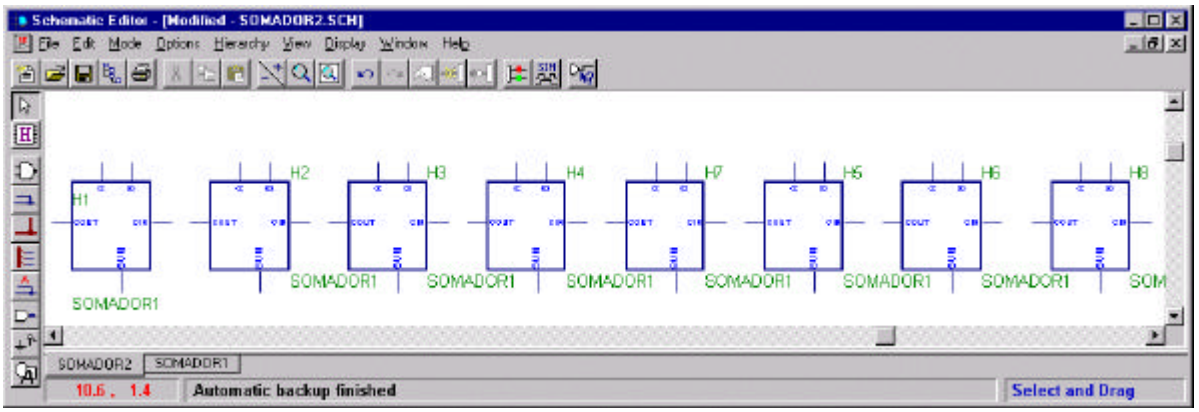


Figura 13 – Esquemático do somador de 8 bits, com 8 instâncias do FA.

5. Realizar as conexões relativas a propagação do vai um (carry out).

6. Inserir os pinos Cin e Cout.
7. Inserir os barramentos de 8 bits referentes as entradas A e B, e a saída Soma (botão 4 - Figura 6). Atenção à direção dos sinais (input/output). Procedimento: um clique no início do barramento e um duplo clique no final deste. Após o duplo clique aparecerá a janela da Figura 14, devendo-se indicar: nome do barramento, direção e número de bits.

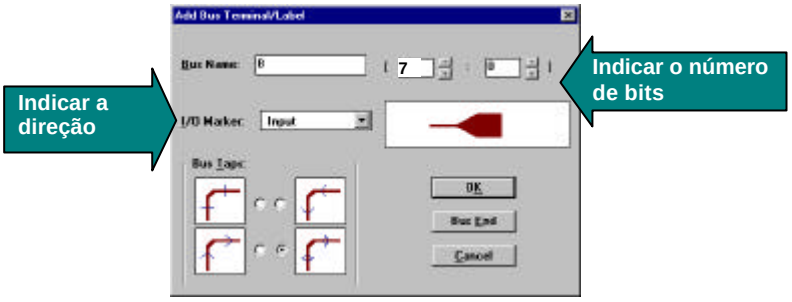


Figura 14 - Janela de inserção de barramento.

Neste ponto, o esquemático deve ser semelhante ao esquemático da Figura 15.

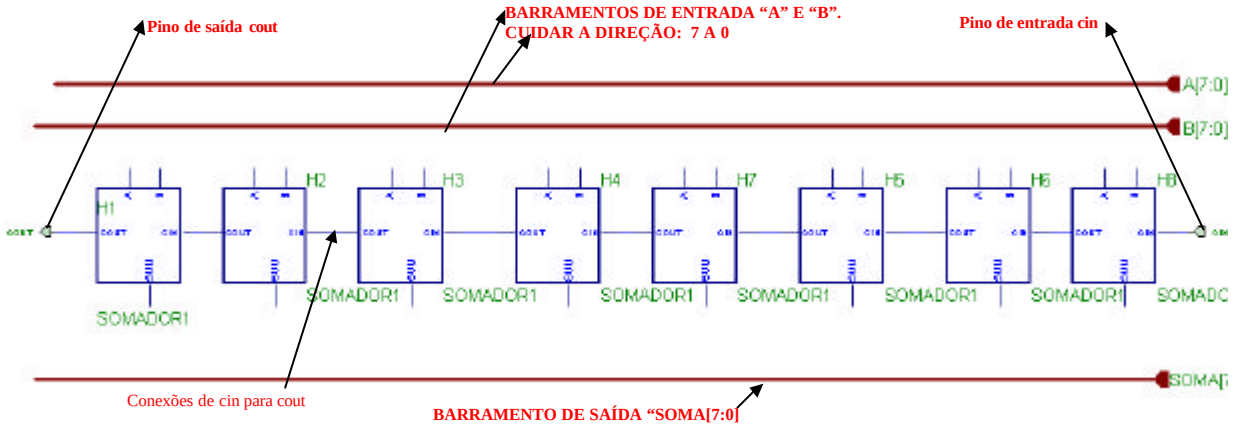


Figura 15 - Esquemático do somador de 8 bits, com inserção dos barramentos.

8. Após a inserção dos barramentos, ligar os fios dos FA aos barramentos.

***** parte mais propensa a erros *****

Procedimentos:

Método 1: clicar no botão (5), clicar no barramento para selecioná-lo (**observar a mensagem "expand: bus tap A"** na parte inferior da janela), clicar nos pinos dos FA. A conexão se faz automaticamente.

Método 2: caso o método 1 não realize a conexão automaticamente: insira um fio entre o pino e o barramento, clique no botão (5), clicar no barramento para selecioná-lo (**observar a mensagem "expand: bus tap A"** na parte inferior da janela), selecionar o sinal (A0, A1, A2, ..) com os teclas ↑ ou ↓, clicar no pino do FA. Observar o texto que será inserido sobre a conexão.

- Uma dica é, após a realização das conexões, verificá-las utilizando o botão do editor correspondente (procedimento: selecionar o símbolo e depois clicar no botão de conectividade).

O resultado final é apresentado na Figura 16.

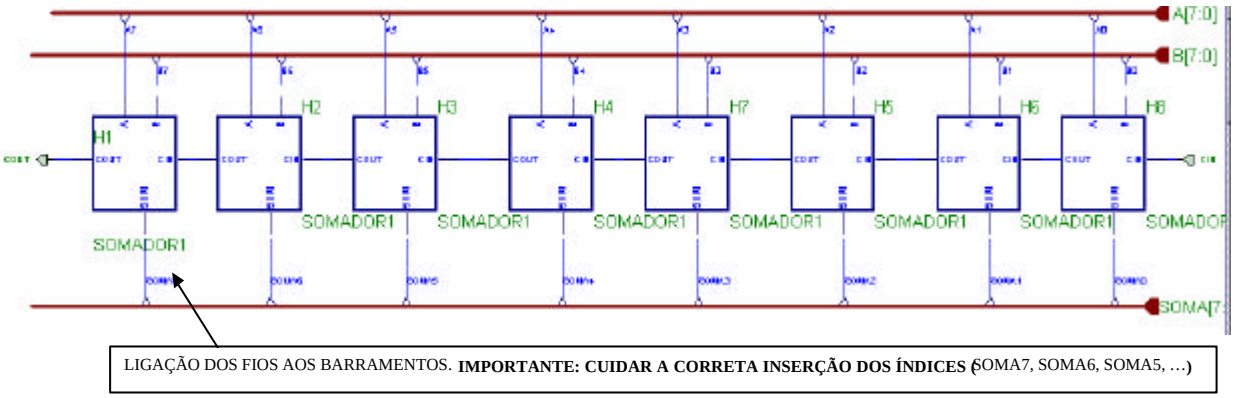


Figura 16 - Esquemático final do somador de 8 bits.

Tarefa 5: Simule o somador de 8 bits projetado anteriormente. Siga os passos abaixo.

1. Deve-se primeiro definir o *script* para simular este circuito. Um possível *script* é apresentado na Figura 17.

```
; simulacao relativa ao projeto somador de 8 bits
delete_signals
restart
stepsize 10 ns

; define os vetores a serem visualizados
vector A     a[7:0]
vector B     b[7:0]
vector SOMA  soma[7:0]

wfm A 0ns=10\H  40ns=15\H  40ns=30\H  40ns=50\H  40ns=60\H  40ns=05\H  40ns=09\H
wfm B 0ns=30\H  40ns=AB\H  40ns=FF\H  40ns=01\H  40ns=8C\H  40ns=8E\H  40ns=EE\H
wfm cin 0ns=0  140ns=1

watch cout
sim 280ns
```

Figura 17 - Script de simulação para o somador de 8 bits.

2. O resultado esperado para esta simulação é apresentado na Figura 18 (a interpretação desta simulação deverá constar no relatório).

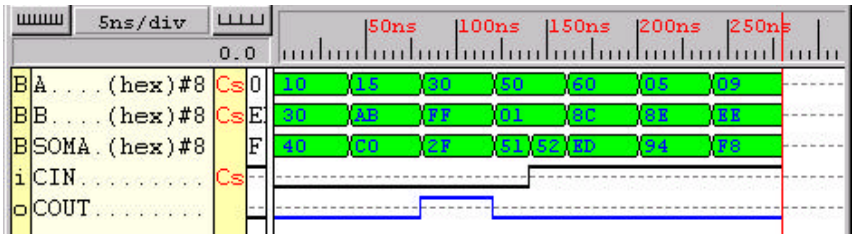


Figura 18 - Janela de simulação para o somador de 8 bits.

Tarefa 6: Inserir no esquemático a lógica e um pino que identifica transbordo (overflow). Não se esqueça que o conceito de transbordo apenas faz sentido se interpretam as entradas do somador como números representados em complemento de 2.

Tarefa 7: Simular. O grupo de alunos deve considerar no *script* casos onde haverá/não haverá vai um e transbordo, indicando na simulação estes casos.

3 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 diagramas de esquemáticos

Recursos: Sistema de CAD Foundation da empresa Xilinx, Espresso da Universidade de Berkeley

3.1 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** do circuito, 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** (em inglês, *finite state machines*, ou **FSMs**), também denominadas **autômatos finitos**, revisado nas aulas teórica. Lembre-se que a implementação de FSMs sob a forma de hardware síncrono pode ser realizada seguindo o modelo estrutural da Figura 19, onde 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 (I_i) e do estado atual armazenado no registrador de estado (S_i). Com isto, dada uma codificação dos estados internos, basta gerar o hardware combinacional que implementa as duas funções para implementar a FSM.

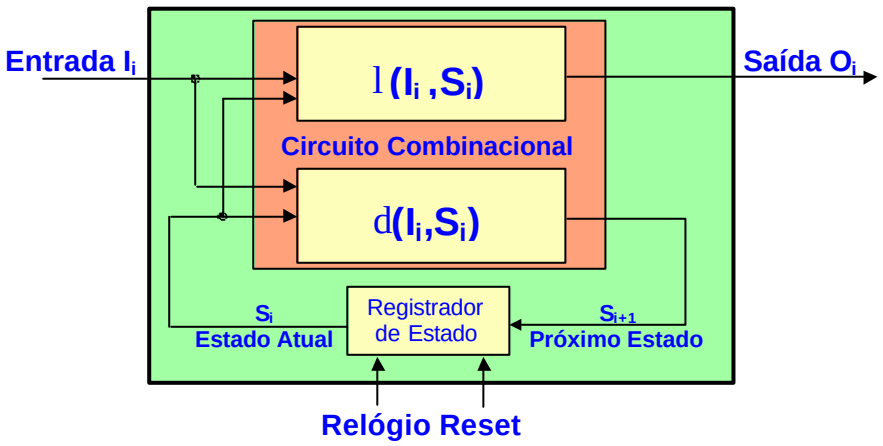


Figura 19 – Modelo estrutural para FSMs síncronas.

3.2 Melhorando a Capacidade de implementar Funções Booleanas - Espresso

O programa **espresso**, desenvolvido na década de 80 por pesquisadores da Universidade de Berkeley (Califórnia, USA) é de domínio público, inclusive sendo seu código fonte disponível. Este programa recebe uma especificação de **n** funções discretas, todas dependentes de um conjunto de **p** variáveis discretas, e fornece como saída uma descrição equivalente sob a forma de uma soma de produtos, onde o número de produtos é minimizado em relação à descrição original (se possível), e onde há um compartilhamento máximo de utilização de produtos por funções distintas.

Este programa deve estar instalado nas máquinas do laboratório sob o diretório **C:\XSTOOLS\BIN**. Neste diretório deve existir o programa (**espresso.exe**), um diretório com diversos exemplos (diretório **PLA_ex**). Além destes, devem existir os dois manuais da ferramenta (arquivos **user_manual.txt** e **io_manual.txt**). Espresso é um programa DOS, que trabalha a partir de uma interface de linha de comando. Como o programa opera com entrada e saída padrão, a maneira mais simples de executá-lo é digitar na linha de comando DOS o seguinte:

DOS_Prompt> espresso Esta linha dispara o espresso e espera que o usuário entre com as funções linha a linha via teclado (ou seja, a entrada padrão), até a linha contendo .e (comando que indica o final da descrição). A seguir o programa é executado e o resultado é jogado na saída padrão (normalmente a tela). Para usar arquivos de entrada e saída digita-se a linha DOS:

DOS_Prompt> espresso arq_entrada >arq_saída Neste caso, o programa lê a descrição do arquivo arq_entrada e coloca a saída minimizada no arquivo arq_saída.

3.3 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 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. Uma terceira hipótese simplificadora consiste em ignorar a composição exata das moedas inseridas na máquina, atendo-se apenas ao montante total inserido.

3.4 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 20.

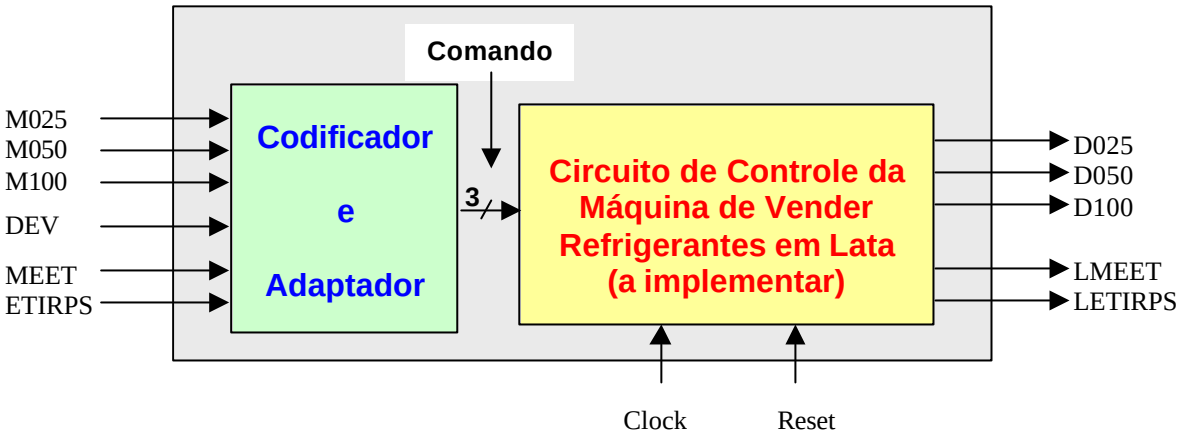


Figura 20 – Diagrama de blocos da máquina de venda de refrigerantes, mostrando a interface com o mundo externo.

Note-se na Figura 20 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:

Comando=000 – nenhuma tecla apertada;

Comando=001 – chegou moeda de R$0,25;

Comando=010 – chegou moeda de R$0,50;

Comando=011 – chegou moeda de R$1,00;

Comando=100 – pressionada a tecla para devolver as moedas colocadas;

Comando=101 – apertada a tecla de pedido de lata de refrigerante MEET;

Comando=110 – apertada a tecla de pedido de lata de refrigerante ETIRPS;

Comando=111 – nunca deve aparecer, constituindo-se assim em uma condição de inespecificação, ou *don´t-care*. Este último pode e deve ser usado para simplificar o projeto do circuito.

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 precisaremos 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 vetor **Comando**. Não se esqueça de considerar a ação do circuito quando nenhuma das entradas está ativa (entrada **Nenhum**). Assim, temos a Tabela 2, 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 2: **se nenhum sinal de saída aparece, isto significa que todos são gerados com valor '0'** (por exemplo, quando se está no estado S000 e a entrada é M025). Note ainda que muitas entradas da tabela não estão preenchidas.

A codificação dos estados já está implicitamente feita no arquivo exemplo (**Refri.pla**). Perceba que esta escolha pode afetar o tamanho do hardware gerado. Cada codificação distinta pode gerar um hardware combinacional distinto. É importante salientar que existem codificações corretas e incorretas. Uma maneira simples de garantir a corretude de uma codificação é simplesmente atribuir códigos distintos a estados distintos.

3.5 A Fazer e Entregar

Tarefa 1: Complete a tabela de transição de estados (Tabela 2) corretamente.

Tabela 2 – Tabela de transição de estados que descreve o comportamento da máquina de venda de refrigerantes.

Estado Atual	Comando						
	Nenhum=000	M025=001	M050=010	M100=011	DEV=100	MEET=101	ETIRPS=110
S000		S025			S000	S000	
S025		S050			S000, D025=1		
S050						S050	
S075							
S100	S100			S150, D050=1			
S125							
S150							

Tarefa 2: Usar o arquivo disponível na *homepage* da disciplina (arquivo **refri.pla**), contendo o início da descrição das funções combinacionais da máquina de vender refrigerantes. Tomem este arquivo, e leiam a introdução em comentários sobre o formato de especificação de entrada do **espresso**. A partir daí, completem a especificação de entrada para que reflita exatamente a especificação da parte combinacional da máquina de vender refrigerantes e executem o **espresso** sobre este arquivo, gerando um arquivo de saída. Analisem este arquivo para verificar qual o resultado da simplificação conduzida pelo programa.

- Quantos bits tem a codificação de estado?
- Porque a escolha da atribuição da codificação de estados pode afetar o tamanho do hardware gerado?
- Qual o número de entradas e saídas de cada uma das funções combinacionais (função de saída e função próximo estado – ver Figura 19)? Esta resposta determina quantas funções Booleanas deverão ser efetivamente implementadas e de quantas variáveis cada uma delas depende.

Tarefa 3: Gere uma implementação em diagrama de esquemáticos do circuito de controle da máquina de vender refrigerantes em lata, utilizando as equações obtidas no item acima, usando como referência a Figura 19. A Figura 21 ilustra como deve ficar o esquemático. Não esqueça de colocar flip-flops nas saídas para estabilizar os sinais (transforma em *Moore* a máquina de estados).

Tarefa 4: Simule o circuito implementado. A Figura 22 ilustra um exemplo de simulação. Para tanto, gere um *script* 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.

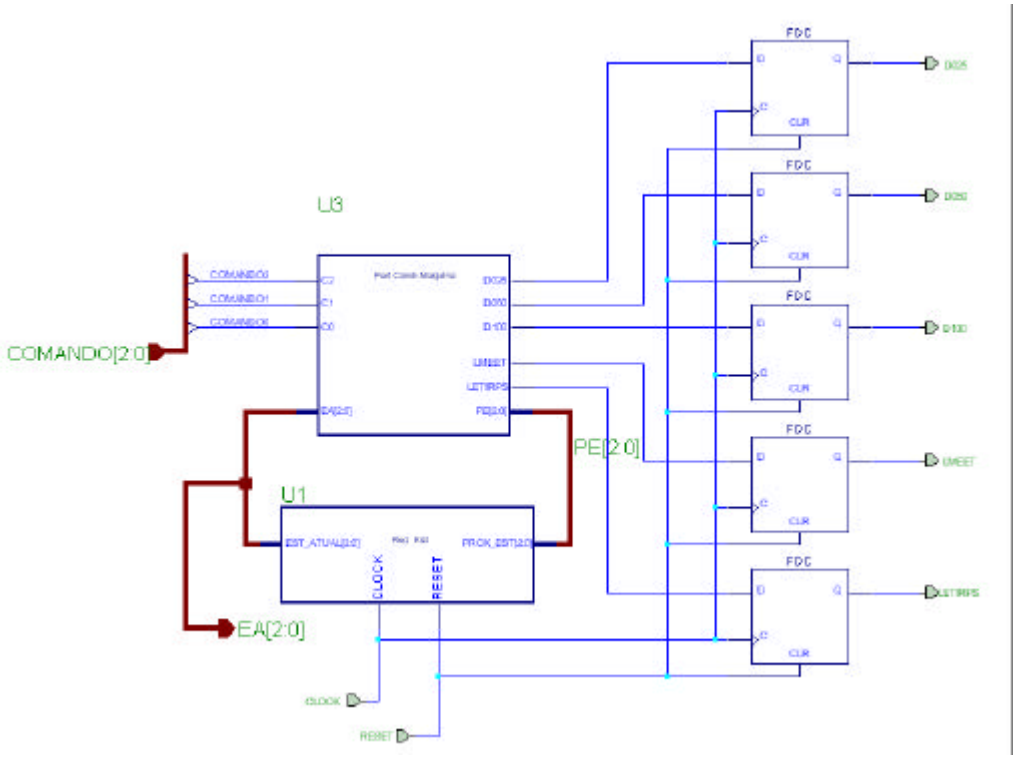
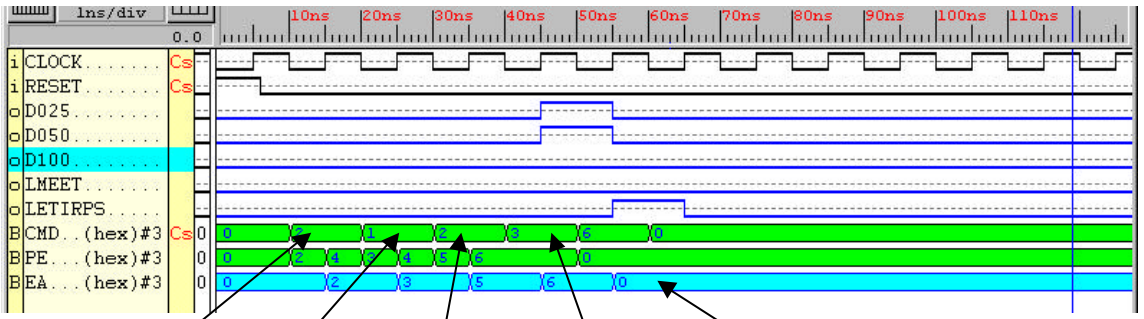


Figura 21 - Esquemático com registradores na saída.



Moeda de 50 | Moeda de 25 | Moeda de 50 | Moeda de 100, devolve 75 | Pediu refri e voltou para o estado 0

```
| ----------------------- Initial settings ----------------------
delete_signals
set_mode functional
restart
stepsize 10 ns

|-----------------Watched Signals and Vectors ------------------------
watch CLOCK RESET D025 D050 D100 LMEET LETIRPS

|----------------------- Vectors defintions ----------------------
vector CMD COMANDO[2:0]
vector PE  PE[2:0]
vector EA  EA[2:0]

|----------------------- Clock and Reset defintions ----------------
wfm clock 0ns=0 (5ns=1 5ns=0)*20
wfm reset 0ns=1 6ns=0

|-- COMANDO
|        NADA        50        25          50          100      LETIRPS      NADA
wfm CMD 0ns=000\B 10NS=010\B 10NS=001\B 10NS=010\B  10NS=011\B 10NS=110\B 10NS=000\B

|----------------- run the simulation  -------------------------------
sim 150ns
```

Figura 22 - Exemplo de simulação da máquina de refrigerantes e respectivo *script* de simulação.

4 Implementação Física de Circuitos Digitais

Prática: O exemplo da máquina de venda de refrigerantes em lata: implementação em hardware do circuito usando diagramas de esquemáticos

Recursos: Sistema de CAD Foundation da empresa Xilinx, Inc, Plataforma XS40/XST1 da empresa Xess, Inc, software Xstools

4.1 Objetivos

Este trabalho é continuação do **Laboratório sobre Circuitos Seqüenciais e Máquinas de Estados Finitas** .

O objetivo deste laboratório é **aprender como, a partir de um projeto validado funcionalmente, chegar a um hardware que efe tivamente funcione**. Isto será feito usando-se ferramentas de síntese e implementação automatizada de hardware disponíveis, bem como através da plataforma de prototipação disponível no ambiente de aula.

4.2 Adaptação do Projeto da Máquina de Venda de Refrigerantes para Implementação na Placa de Prototipação XS40/XSt1 da Empresa Xess Inc.

Descreve-se a seguir, passo a passo, as ferramentas a usar para a implementação em hardware da máquina de venda de refrigerantes sobre uma plataforma específica disponível.

1. Plataforma XS40/XSt1 da Xess (Figura 23)

Trata-se de um Plataforma de Hardware adequada para uso educacional, formada por duas placas. Estas contêm um dispositivo de hardware reconfigurável do tipo FPGA (Field-Programmable Gate Array), memória, microprocessador (na placa XS40) e diversos recursos para realizar Entrada e Saída: displays de 7 segmentos, leds, conectores para teclado e vídeo, codificadores/decodificadores de áudio, e área de prototipação para montagens de hardware (sobretudo na placa inferior, a XSt1). Para trabalhar nesta e nas partes seguintes deste trabalho, utilize uma da plataformas XS40/XSt1 disponíveis no ambiente de aula.

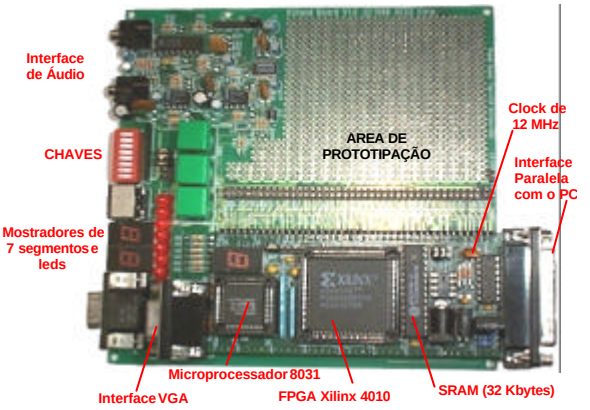


Figura 23 – Plataforma XS40/XSt1.

2. Para a implementação do circuito da máquina de vender refrigerantes na plataforma XS40/XSt1, são necessários alguns blocos adicionais de hardware, que adaptem os sinais e formatos de entrada e saída para utilização por seres humanos, devido às características físicas dos dispositivos de entrada e saída que queremos usar. Vamos agora estudar estes blocos:

– *Debounce*: este é o nome do circuito que "limpa" o sinal binário de uma chave mecânica, para evitar erros tais como interpretar que apertar uma vez a tecla seja considerado como apertar diversas vezes. No laboratório, mostrar-se-á na prática como este problema ocorre, usando equipamento de teste e medida tal como osciloscópio. Para maiores detalhes consultar o Anexo II.

– Conversor *bin_7seg*: Trata-se de um circuito combinacional, que recebe como entrada um número composto por 4 bits, correspondendo a um dígito hexadecimal entre 0H e FH. Produz como saídas 7 sinais para acionar um mostrador de 7 segmentos, acendendo os segmentos que identificam o número binário representado na entrada. Estes dois circuitos serão fornecidos na *homepage* da disciplina, através de sua descrição em VHDL. Use-os inicialmente como células de biblioteca do tipo caixas pretas e insira-os no seu projeto na medida do necessário. O processo de inserção é detalhado mais adiante.

4.3 Síntese, Implementação e Teste da Máquina de Vender Refrigerantes

Uma vez editada a descrição abstrata do projeto (esta fase se chama tradicionalmente de *fase de projeto* ou, mais detalhadamente, *fase de captura do projeto*), passa-se a executar a *fase de validação de projeto*, através de simulação funcional (componentes considerados sem atrasos). Para se chegar ao produto final, é necessário passar ainda pela fase onde a descrição abstrata é traduzida para uma descrição física (a chamada *fase de síntese*). Após a síntese, procede-se à implementação do hardware (*fase de implementação*), que pode ser feita via fabricação do "chip" ou, como será o caso aqui, pela configuração de um dispositivo especial, através do processo de "descarga" (em inglês, *download*) da implementação sobre o FPGA da placa XS40/XSt1. Finalmente, deve-se proceder ao teste do circuito implementado (no que se denomina *fase de teste*).

A fase de síntese para o sistema que utilizamos é normalmente realizada de forma muito simples, pois o sistema de CAD (Computer Aided Design) Foundation possui um ferramental automatizado poderoso e de interface com o usuário amigável. No entanto, algumas tarefas devem ser cumpridas antes de realizar a síntese. Estas tarefas são aquelas dependentes do tipo de dispositivo físico a ser utilizado. Até a fase de validação, todo o processo de projeto é independente da tecnologia de implementação (exceto a escolha do circuito integrado XC4010XLPC84-3, que não afeta a *captura do projeto*). Agora, deve-se fornecer as informações específicas do circuito integrado, bem como da plataforma de implementação onde este circuito se encontra (placa e dispositivos a ele conectados, tais como cristal de relógio, chaves dipswitch e push-button, leds e mostradores de 7 segmentos). A mais fundamental destas informações consiste na associação de sinais de entrada e saída (E/S) do circuito aos pinos de entrada e saída do FPGA. Vejamos como se faz esta associação.

1. Existem dois métodos básicos para fazer a associação pinos E/S do projeto aos pinos E/S de FPGA no CAD Foundation para esquemáticos:

– Método dependente do projeto – Dentro do esquemático de topo (no nível mais alto da hierarquia), usar componentes do tipo PAD (um jargão técnico usado para designar cada ponto do chip onde se soldam fios microscópicos que conectam algum ponto deste ao mundo externo através dos pinos do encapsulamento) da biblioteca XC4000X, tais como IPAD, OPAD, no lugar dos conectores de entrada e saída usados acima. Após, associar um número de pino do FPGA (no nosso caso, os pinos do XC4010PC84-3 são numerados de P1 a P84) a cada PAD. Este é o método padrão da ferramenta de esquemáticos, que não usaremos aqui;

– Método independente de projeto – Inserir a relação entre E/S do projeto e do FPGA como uma tabela no arquivo **<nome do projeto>.ucf**, produzido automaticamente durante a criação dos arquivos de projeto. A vantagem deste método está na portabilidade dos arquivos de projeto isolando a região de projeto dependente de implementação num arquivo específico (conforme dito antes, UCF é um acrônimo para User Constraints File, ou Arquivo de Restrições do Usuário) . Este é o método que escolhemos aqui.

2. Para associar os pinos externos ao esquemático, aconselha-se a utilizar o segundo método. Para isto, no gerenciador de projeto Foundation abra a janela de diálogo de opções de síntese, escolhendo **Implementation→Options**. A seguir, clique no botão **Edit Options** da linha **Implementation**. Agora, escolha a orelha **Translate** e lá marque a caixa **Create I/O Pads from Ports**, se ela já não estiver marcada. Esta opção anuncia para o CAD Foundation que a associação de pinos deve ser feita relacionando-se os terminais do nível mais alto de sua hierarquia de projeto aos pinos mencionados para estes no *arquivo.ucf*. A Figura 24 mostra como deve aparecer a tela após a escolha. Aceite as opções assim feitas, clicando nos botões **OK** até todas as janelas de diálogo estarem fechadas.

3. Agora tudo está pronto para a síntese. Para tanto, basta clicar em **Implementation→Implement Design** no gerenciador de projetos. Se tudo estiver correto, surge uma janela de diálogo **Implement Design**. Aguarde o final da síntese. Se ocorrerem erros, estude as mensagens e resolva-os, em geral são causados por bobagem, tais como erro de sintaxe no arquivo .ucf. Se necessitar re-executar a chamada da síntese automática depois que esta foi pelo menos uma vez concluída, você pode optar por clicar na opção **Overwrite current version**, para poupar espaço em disco. Cinco passos têm lugar durante a fase de síntese:

– *Translate* - tradução dos dados de projeto do formato independente de tecnologia para o formato proprietário Xilinx;

- *Map* – particionamento do seu projeto em um conjunto de pedaços que caibam cada um nos blocos físicos do FPGA, os chamados Configurable Logic Blocks ou CLBs e os Input/Output Blocks, ou IOBs;
- *Place&Route* – Posicionamento dos CLBs/IOBs em posições físicas específicas do chip em questão e conexão via fios que têm de ser conectados entre dois CLBs dois IOBs e entre CLBs e IOBs;
- *Timing* – Cálculo dos valores precisos de atrasos associados a fios e blocos de todo o circuito, visando uma ressimulação do projeto com atrasos detalhados. Este passo pode dar mais segurança que a implementação final vai funcionar (não usado aqui, reservado para cursos mais avançados);
- *Configure* – Geração do arquivo binário ***<nome do seu projeto>.bit***, a ser usado na configuração do FPGA para que este funcione como o hardware projetado por você.

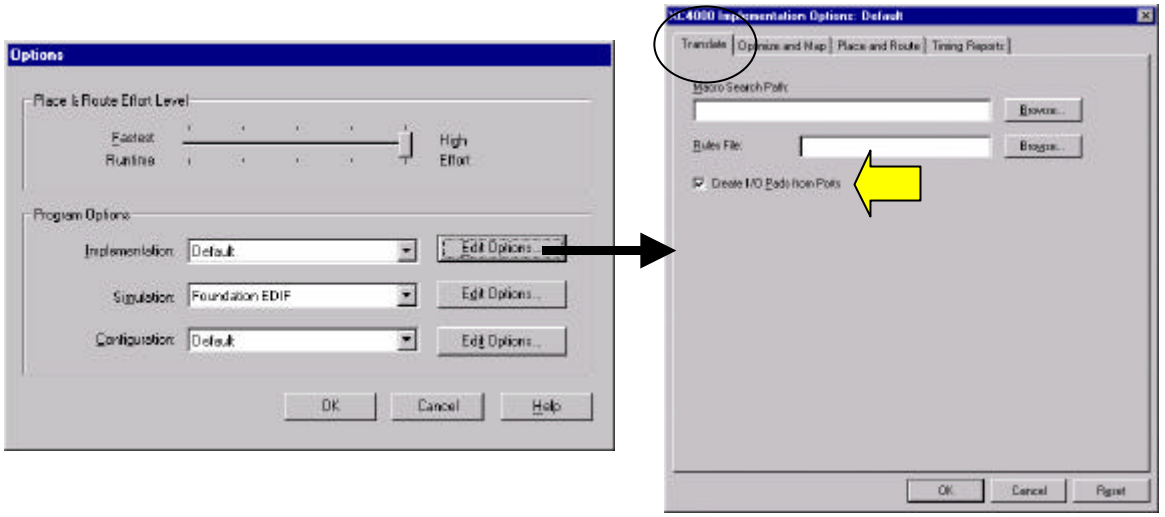


Figura 24 – Janela de escolha de método de associação de pinos E/S de projeto e FPGA.

4. Se a síntese tiver ocorrido sem problemas, surge a janela de diálogo na Figura 25. Clique **OK** nela.

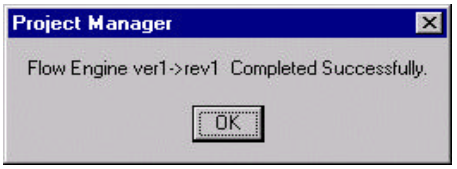


Figura 25 – Janela de diálogo final da implementação.

5. Caso deseje, você pode agora observar a implementação física do projeto através do editor de "layout" de FPGAs Use a opção de menu **Tools → Implementation → FPGA Editor** do gerenciador de projeto Foundation. Lá pode-se observar e editar cada detalhe do mapeamento, do posicionamento e do traçado de rotas realizado. Não edite seu projeto nesta ferramenta, ele pode não funcionar mais depois disto!!!

6. Agora vem a fase de implementação, mais simples que todas as anteriores. Para implementar seu projeto sobre o FPGA da placa XS40/XSt1 existe um conjunto de ferramentas que não está integrado com o CAD Foundation, e que deve ser executado separadamente (esta é uma característica particular da plataforma que empregamos). O único arquivo de todo o seu projeto que interessa é o arquivo binário gerado no penúltimo passo da síntese acima, ***<nome do seu projeto>.bit***. A versão mais atual deste arquivo está sempre disponível diretamente no diretório de mesmo nome que o nome de seu projeto. O objetivo da implementação é descarregar os conteúdos deste arquivo, via porta paralela do PC, para o FPGA, assim configurando-o. Mais uma vez há duas maneiras de fazê-lo:

- *Via Interface Gráfica* – verifique se o programa GXSLOAD.EXE executa em sua máquina. Caso afirmativo, basta executá-lo. Surge uma janela para a qual basta arrastar o arquivo *.bit* desejado e a configuração é realizada. A janela do GXSLOAD tem a aparência da Figura 26;

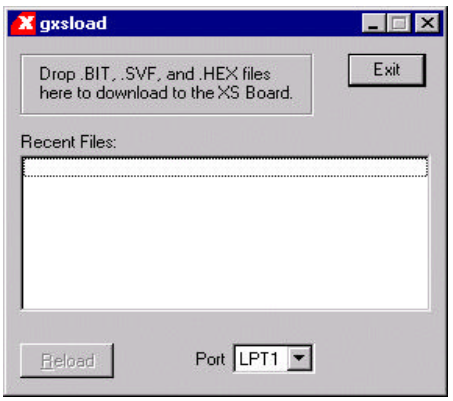


Figura 26 – Janela da interface gráfica para configuração do FPGA da placa XS40/XSt1.

– *Via comandos DOS* – O programa não gráfico denomina-se XSLOAD.EXE, e aceita como parâmetro o nome do arquivo *.bit* a ser usado na configuração do FPGA da placa. Cuidado contudo deve ser tomado pois o programa espera que a variável de ambiente XSTOOLS_BIN_DIR esteja apontando para o diretório contendo os arquivos de configuração do fabricante, tipicamente em C:\XSTOOLS\BIN ou em \XSTOOLS. Verifique se esta variável está disponível na sua máquina, senão use o comando DOS SET para criá-la ou alterá-la para seu valor correto. Após isto, basta executar a linha DOS: **XSLOAD <*nome do seu projeto*>*.bit***

4.4 A Fazer e Entregar

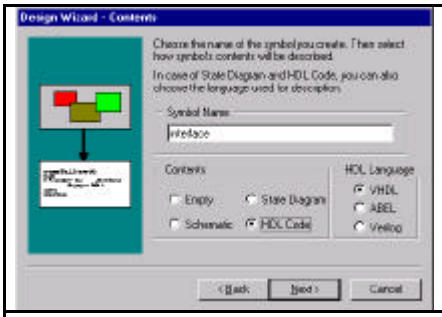
Tarefa 1: Crie um novo projeto a partir de uma cópia de seu projeto original da máquina de vender refrigerantes realizado na fase anterior deste laboratório (Note que seu projeto tem de estar já simulado **ok** antes de usá-lo aqui). Use o Comando **File →Copy Project** do Gerenciador de Projeto do CAD Foundation, criando uma cópia do seu projeto com outro nome, por exemplo **refr_hw**.

Tarefa 2: Crie um macro-símbolo de sua máquina de venda de refrigerantes. Instancie-o em uma folha vazia. Observe o macro-símbolo referente ao circuito na Figura 28. Neste macro-símbolo temos como entradas o vetor de comando, *clock* e *reset*. Como saídas o estado atual e os sinais D025 a D100 e LMEET e LETIRPS.

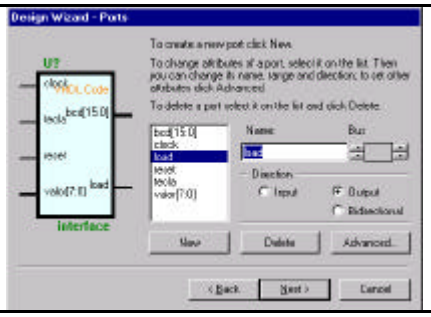
Tarefa 3: Faça a criação de dois novos símbolos do tipo VHDL. Para isto crie dois símbolos vazios, via opção *Tools* ➔ *Symbol Wizard*, um com nome **debounce** e outro com nome **bin_7seg**.

- O módulo **debounce** tem três entradas de 1 bit (**tecla**, **reset**, **clock**) e uma saída de um bit (**teclaf**, significando tecla filtrada).
- O módulo **bin_7seg** possui uma entrada de quatro bits (**valor[3:0]**) e uma saída de 7 bits (**setseg[6:0]**).
- O nome do módulo criado tem de ser exatamente o mesmo do arquivo VHDL correspondente, ou seja, *debounce* e *bin_7seg*. Um exemplo de detalhamento deste procedimento aparece na Figura 27, ***para outro tipo de circuito***. Siga os mesmos passos para seu caso, *mutatis mutandis* (em latim, mudando o que tem de ser mudado).

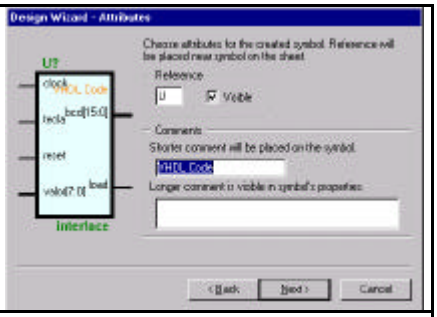
Tarefa 4: Insira os símbolos recém criados, **bin_7seg** e **debounce,** no esquemático contendo o macro-símbolo da máquina de vender refrigerantes. O bloco **debounce** deve ter sua saída *teclaf* conectada ao pino de *clock* de sua máquina, sua entrada *tecla* conectada ao um pino externo tal como *Tclk* através de um buffer de entrada (*Tclk* significa Tecla de clock, e vai ser um dos botões verdes da plataforma de hardware, a tecla SPARE). Os pinos de controle do debounce, por outro lado, devem ser ligados aos pinos externos *Treset* (Tecla de Reset, o botão verde do meio, na plataforma de hardware) e Ck12MHz (o pino conectado ao clock de 12MHz da plataforma de hardware (ambos via buffers).



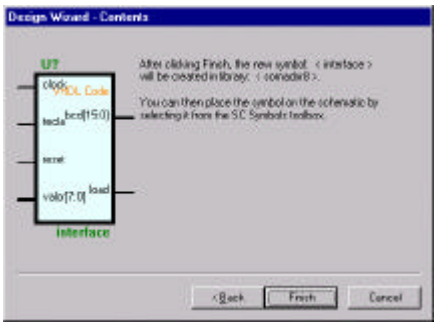
1. Nomeie o nome do bloco como *interface*
2. Indique que o conteúdo é HDL
3. Indique VHDL como a HDL que será utlizada.



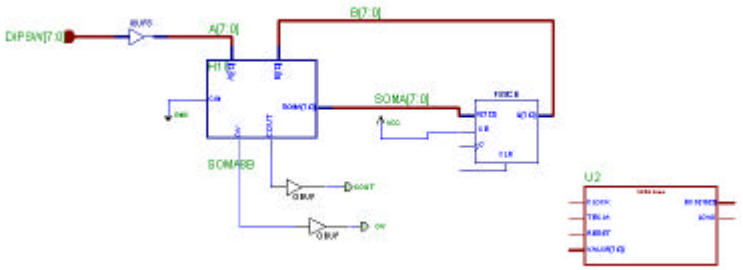
1. Defina os pinos de entrada: clock, reset, tecla e valor (indicar no quadrado bus o intervalo 7:0)
2. Defina os pinos de saída: load e bcd (indicar intervalo 15:0)



Clique next para avançar.



Clique Finish para terminar.



No esquemático em edição, instancie o módulo, e verifique se corresponde ao esperado.

Figura 27 - Criação de um novo símbolo VHDL vazio, para outro tipo de circuito.

Tarefa 5: Realize as conexões finais. Responda à seguintes questões:

- Existem 4 entradas no circuito (CK12MHZ, TCLK, TRESET, COMANDO[2:0]); porque pede-se aqui então que apenas a entrada TCLK tenha seu circuito de debounce? Não há problemas similares para os sinais de TRESET e os 3 bits de COMANDO?
- Porque na Figura 28 são inseridos inversores entre as saídas e os OBUF? Dica: olhar no manual da placa de prototipação.

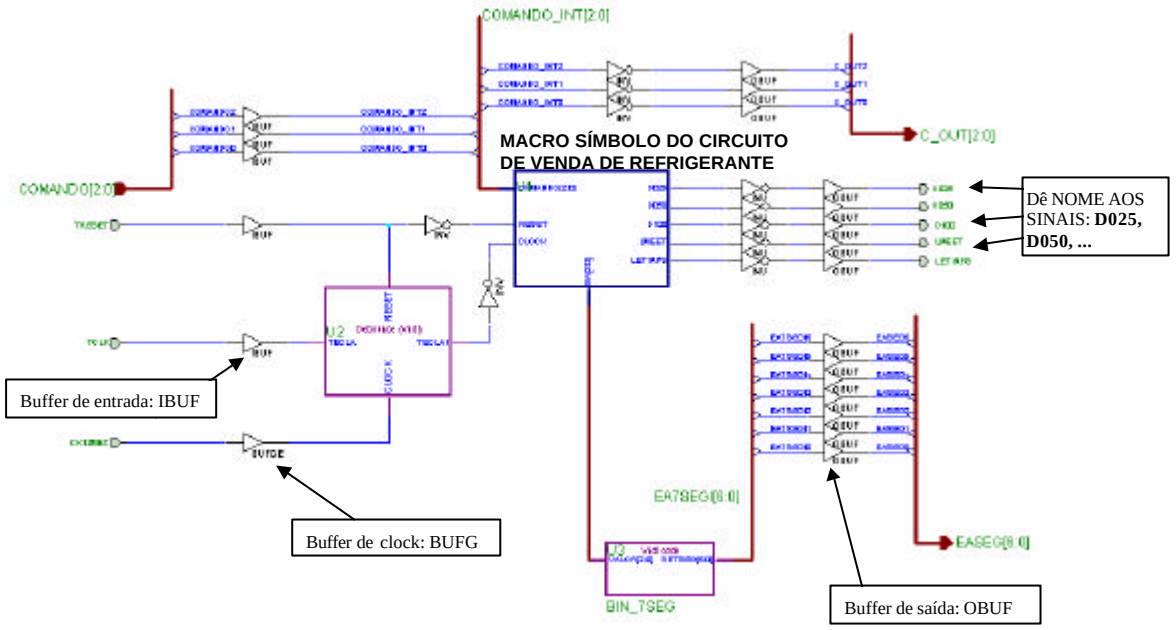


Figura 28 – Um exemplo de circuito completo, com buffers e terminais adequados.

ATENÇÃO:

- cuidado para ligar *buffers* de entrada em pinos de entrada e de saída em pinos de saída;
- o *buffer* para o sinal de relógio **Clock** denomina-se **BUFG**, não use **IBUF**;
- note que cada entrada e cada saída é isolada do mundo externo via *buffers* adequados;
- a saída do módulo **bin_7seg** deve ser ligada ao estado de sua máquina. Se o estado de sua máquina não for uma saída do seu módulo, ***altere*** sua macro para disponibilizá-lo como uma saída. Apesar disto não ser necessário ao funcionamento, será uma boa ilustração enxergar a troca de estados internos. Encare isto como uma informação útil para a depuração do circuito.

Tarefa 6: Insira o código VHDL dado nos blocos criados na tarefa 3 (até então vazios). Selecione o botão para percorrer a hierarquia (botão "H"), clicando duas vezes na instância do módulo criado (*debounce* e/ou *bin_7seg*). O editor de VHDL é aberto com um esqueleto do circuito. O procedimento para inserção do código VHDL é: (1) apague todo o conteúdo do arquivo, ou seja, o esqueleto do circuito; (2) abra os arquivos VHDL disponíveis na homepage da disciplina, usando um editor de texto; (3) para cada arquivo no editor de texo selecionar todo o seu conteúdo, copie-o e cole-o no arquivo VHDL em edição; (4) salve e o arquivo VHDL. Ao sair do editor de VHDL, será feita automaticamente a verificação sintática e a síntese de hardware do módulo VHDL. Poderá levar alguns minutos. Paciência ...

Tarefa 7: Recupere o arquivo "lab03011.ucf" disponível na homepage da disciplina. Insira todo o conteúdo deste arquivo texto ao final do arquivo .ucf disponível no seu projeto e salve o seu .ucf. Adicione o arquivo assim obtido ao seu projeto (opção de menu Document → Add do Gerenciador de Projeto do Foundation). O arquivo "lab03011.ucf" possui as definições de todos os pinos necessários a este projeto específico. Ele foi montado com base nas tabelas do arquivo xst40mdl.pdf, que contém a descrição completa do modelo do usuário da placa XS40/XSt1, e está contido nos manuais da plataforma XS40/XSt1 disponíveis no laboratório. Observe que se você usou nomes de terminais no seu projeto que não correspondem exatamente aos nomes usados no arquivo .ucf, ou você deve editar seu esquemático, ou editar o arquivo de restrições, para compatibilizá-los.

Tarefa 8: Gere o arquivo binário de configuração

Tarefa 9: Observe o layout do circuito obtido.

Tarefa 10: Realize o download na placa de prototipação.

Tarefa 11: Finalmente, realizada a fase de implementação vem a fase de teste do circuito. Exercite o seu

circuito, verificando a correção dos resultados da operação da máquina. Calcule manualmente o valor do resultado e confira usando o circuito. Use os dados de seu *script* de simulação, por exemplo. As informações para executar a contento esta fase são agora abordadas:

- Configuração das teclas – Use a Figura 28 para localizar os elementos da placa XS40/XSt1. Os três bits da palavra de comando C[2:0] são entrados usando as 3 teclas mais à direita do conjunto de dip-switches (8 chaves brancas num suporte vermelho. Posição OPEN indica chave aberta. As teclas verdes grandes possuem nomes SPARE, RESET e PROGRAM. Atenção: não aperte esta última tecla (a que está mais à direita), a menos que você deseje desconfigurar ("desimplementar") seu projeto do FPGA. A tecla SPARE (tecla verde mais à esquerda) corresponde ao sinal Tclock de seu projeto, enquanto que a tecla do meio RESET, corresponde ao Treset do seu projeto. Comece inicializando seu circuito por esta tecla.
- *Mostradores e LEDs* – o mostrador esquerdo da placa XSt1 (situado ao lado do conector de teclado) mostrará o estado atual da máquina (entre 0, correspondente a S000 e 6, correspondente a S150). Quando você pressiona RESET, este deve mostrar 0. Dos 8 leds disponíveis acima dos mostradores (numerados de D8 a D1), apenas os cinco mais à esquerda são usados, sendo D8 o sinal D025, D7 o sinal D050 e assim por diante até LETIRPS.
- *Observação*: Se você quiser saber o mapeamento dos pinos do FPGA para pinos de dispositivos da placa XS40/XSt1, existem manuais da placa disponíveis.

Tarefa 12: Usem o osciloscópio para obter o diagrama de tempos da Figura 63 (página 70), conforme o Anexo II, mostrando o resultado obtido ao professor.

Tarefa 13: Demostrar o funcionamento do circuito operando corretamente na placa de prototipação.

5 Laboratório sobre Implementação de Sistemas Digitais Mediante Esquemáticos - Experimento Avançado

Prática: Implementação de um Circuito Medidor Digital de Freqüência

Recursos: Sistema de CAD Foundation da empresa Xilinx, Inc., Gerador de freqüências e Osciloscópio

5.1 Introdução e Objetivos

Este laboratório tem por objetivo consolidar os conhecimentos adquiridos em projetos com esquemáticos, através do projeto de um circuito digital capaz de medir a freqüência de sinais digitais periódicos (freqüencímetro digital). Neste projeto, o aluno deve elaborar, através de esquemáticos, um circuito que permita a medição de várias freqüências, apresentando o valor da freqüência medida nos displays de 7 segmentos disponíveis na placa de prototipação. Este laboratório, ao contrário dos anteriores não fornece extensa informação sobre o que implementar, cabendo aos grupos descobrir, eventualmente com o auxílio do professor em sala de aula, como realizar as tarefas de implementação e teste.

Para realizar o laboratório, os grupos devem usar o gerador de freqüências, e o osciloscópio, que permitirá comprovar os resultados, obtidos nas medições, e apresentados através do display de 7 segmentos.

5.2 Princípio de funcionamento do freqüencímetro proposto

Para a implementação do freqüencímetro será utilizada uma freqüência previamente conhecida (freqüência de referência - **Fr**). A partir de **Fr** e de dois contadores, que são incrementados a cada pulso de relógio, é possível obter o valor da freqüência de entrada, **Fin**. Cada freqüência tem o seu próprio contador. O contador de **Fr** será denominado de **ContFr**, e o contador de **Fin** será denominado de **ContIn**.

Para demonstrar o princípio de funcionamento do freqüencímetro, considere que o contador **ContIn** sempre é incrementado quando **Fin** alterar o seu valor de 0 para 1 (biestável sensível à borda de subida), e que o contador **ContFr** sempre é incrementado quando **Fr** alterar o seu valor de 0 para 1 (idem para a sensibilidade à borda). Neste caso, se **Fr** for igual a **Fin**, ambos contadores terão o mesmo valor de contagem para qualquer instante em que forem avaliados. Porém, se **Fr** for maior que **Fin**, então o **ContFr** terá um valor maior que **ContIn**. Se ocorrer a situação inversa (**Fr** menor que **Fin**), **ContFr** terá um valor menor que **ContIn**. Contudo, existe uma relação constante, já que os contadores informam o número de vezes que as freqüências tiveram transições, assim como pode ser observado na Equação 1, que estabelece uma proporção entre as freqüências envolvidas a partir dos valores de contagem:

$$\frac{\mathbf{ContFr}}{\mathbf{Fr}} = \frac{\mathbf{ContIn}}{\mathbf{Fin}}$$

Equação 1 – Proporção entre Fr e Fin a partir de valores de ContFr e ContIn.

Transformando a Equação 1 é possível obter uma relação de **Fin** em função do valor dos contadores e de **Fr**, como pode ser observado na Equação 2.

$$\mathbf{Fin} = \frac{\mathbf{ContIn} \cdot \mathbf{Fr}}{\mathbf{ContFr}}$$

Equação 2 – Mesma equação 1, com isolamento do fator Fin.

Como artifício de implementação, pode-se utilizar uma freqüência constante para **Fr**, e usar o conteúdo de

ContFr apenas em momentos pré determinados quando este atingir um valor pré definido e conhecido. Uma maneira simples de fazer isto é escolher um dos bits (ou uma configuração de bits) de **ContFr** para servir de indicador dos momentos a usar para computar a freqüência a medir.

A forma para utilizar **ContFr** é fazer com que o cálculo de **Fin** seja disparado *sempre e apenas* quando o **ContFr** tiver contado o valor pré definido. Desta forma, tanto **Fr**, quanto **ContFr** passam a ser constantes no momento da amostragem e **Fin** passa a depender apenas de **ContIn** (ver Figura 29). Por exemplo, supondo que **Fr** seja 10MHz e o **ContFr** dispare a análise de **ContIn** quando o mesmo tiver contado 10.000 bordas da freqüência de referência, a Equação 2 pode ser reescrita sob as formas das Equações 3 e 4 abaixo:

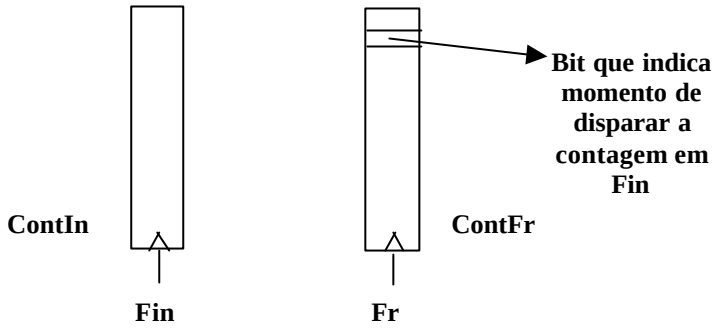


Figura 29 - Ilustração de como os contadores podem ser usados para disparar a contagem. Na Figura, mostra-se apenas a versão que usa um bit de Fr para computar o momento de calcular Fin.

$$\mathbf{Fin} = \frac{\text{ContIn} \cdot 10000000}{10000}$$

Equação 3.

$$\mathbf{Fin} = \mathbf{ContIn} \cdot 1000$$

Equação 4.

Note que com tal esquema simplificado de medida, o circuito seria adequado para medir freqüências com aproximadamente a mesma ordem de grandeza. Logo, **ContIn** deverá armazenar valores na ordem de 10000.

5.3 Detalhes de implementação

O grupo deve atentar ao fato de que quem dispara o processo de cálculo de **Fin** é o valor armazenado em **ContFr**. Logo, deve ser implementado um circuito para detectar quando **ContFr** tem o valor desejado.

Uma vez disparado o cálculo de **Fin**, o resultado deve ser armazenado em alguma memória para que seja possível apresentar o resultado no display de 7 segmentos, e esta memória não deve ser alterada até um novo valor válido de **Fin** ser obtido, conforme descrito acima.

Uma vez calculado o valor de **Fin**, uma nova contagem deve ser efetuada. Logo, é necessário então reinicializar os contadores.

A Figura 30 mostra um esquemático parcial que ilustra o funcionamento do freqüêncímetro.

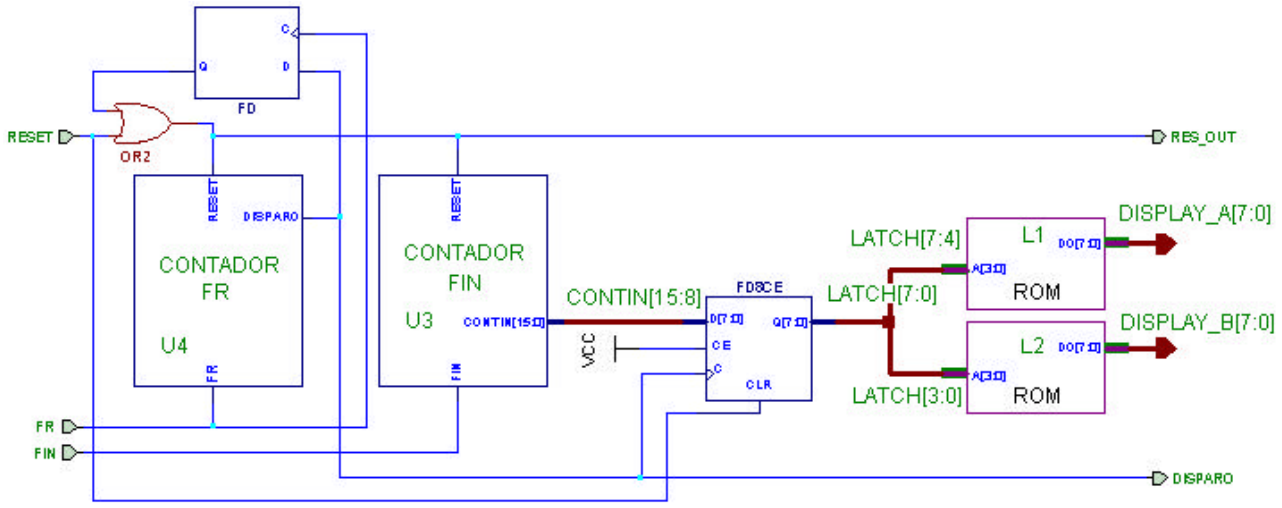


Figura 30 - Esquemático do freqüencímetro digital.

A implementação dos contadores, quanto dos decodificadores dos displays de 7 segmentos, pode ser realizada com logiBLOX (menu Tool – item LogiBLOX Module Generator). A seguir está apresentada a implementação do decodificador:

```
;
; memfile Dec7Seg.mem for LogiBLOX symbol Dec7Seg
; Created on Saturday, September 30, 2000 17:52:27
;
; Header Section
RADIX 10
DEPTH 16
WIDTH 8
DEFAULT 0
;
; Data Section
; Specifies data to be stored in different addresses
; e.g., DATA 0:A, 1:0
RADIX 16
DATA
88 ;1 0 0 0 1 0 0 0   ; 0
ED ;1 1 1 0 1 1 0 1   ; 1
A2 ;1 0 1 0 0 0 1 0   ; 2
A4 ;1 0 1 0 0 1 0 0   ; 3
C5 ;1 1 0 0 0 1 0 1   ; 4
94 ;1 0 0 1 0 1 0 0   ; 5
90 ;1 0 0 1 0 0 0 0   ; 6
AD ;1 0 1 0 1 1 0 1   ; 7
80 ;1 0 0 0 0 0 0 0   ; 8
84 ;1 0 0 0 0 1 0 0   ; 9
; end of LogiBLOX memfile
```

5.4 A Fazer e Entregar

Tarefa 1: Considerando a estrutura da plataforma de prototipação XS40/XST1, qual ou quais as freqüências de referência disponível(is) nesta? Escolha a freqüência com base nesta disponibilidade, pois esta pautará a implementação em hardware.

Tarefa 2: O grupo deve implementar em esquemático um contador para a freqüência de referência **Fr** e outro para a freqüência de entrada **Fin**. Devido à forma como será exibida **Fin,** e à forma de efetuar a contagem de **ContFr**, analise qual o melhor tipo de contador (hexadecimal ou decimal), tanto para **ContIn**, quanto para **ContFr**.

Tarefa 3: Simule funcionalmente o seu projeto com um *script* adequado, apresentando a correta simulação para duas freqüências de entrada distintas (após gerar a forma de onda, use a opção de menu File → Save Simulation State...). Esta simulação não deve ser muito simples de ser realizada, devido à temporização do freqüencímetro. Aponte quais as características do projeto que dificultam a simulação, e como estas dificuldades foram superadas para que a simulação fosse executada corretamente.

Tarefa 4: O grupo deve implementar o circuito na placa XS40/XST-1, criando o arquivo **.ucf** adequado para o seu projeto e testar o mesmo com um gerador de freqüências e osciloscópio no laboratório.

Tarefa 5: Demonstrar o projeto funcionando em sala de aula.

6 Modelo de Computador de Programa Armazenado e Programação em Linguagem de Montagem

Prática: Programação em Linguagem de Montagem do Processador Cleópatra

Recursos: Ambiente de Desenvolvimento de Software para a Arquitetura Cleópatra

6.1 Introdução e Objetivos

O presente Laboratório tem por objetivo estratégico efetivamente iniciar os trabalhos práticos específicos de Organização de Computadores (Unidade 02 da Disciplina, bem como da disciplina teórica companheira). Este primeiro passo consistirá em trabalhar em níveis de abstração mais altos, independentes de hardware, ignorando hardware e lidando com a abstração denominada Arquitetura de Conjunto de Instruções (em inglês, *Instruction Set Architecture*, ou *ISA*), a primeira camada de software acima do hardware de um sistema computacional (caso ignore-se e/ou identifique-se esta camada com o nível de código de máquina). Em resumo, trabalhar-se-á neste Laboratório exclusivamente sobre o tema de programação em Linguagem de Montagem.

Dado o contexto do parágrafo anterior, em acordo com o conteúdo visto de forma concomitante na disciplina teórica, adota-se aqui a arquitetura Cleópatra como alvo do trabalho. O objetivo específico deste laboratório é fixar os conteúdos de programação em linguagem de montagem (em inglês, *assembly language*) para a arquitetura Cleópatra, através do uso do sistema integrado de desenvolvimento, composto de 3 ferramentas acessíveis via **interface gráfica unificada**: um **editor de textos especializado**, o **montador Cleópatra** (**cleoasm.exe**) e o **simulador Cleópatra** (**cleosim.exe**). Todo o sistema é de domínio público, e foi inicialmente desenvolvido por Daniel Carvalho Liedke, quando de sua participação como aluno da disciplina, em 1998/II, e mais tarde como bolsista do Grupo de Apoio ao Projeto de Hardware (GAPH) da FACIN.

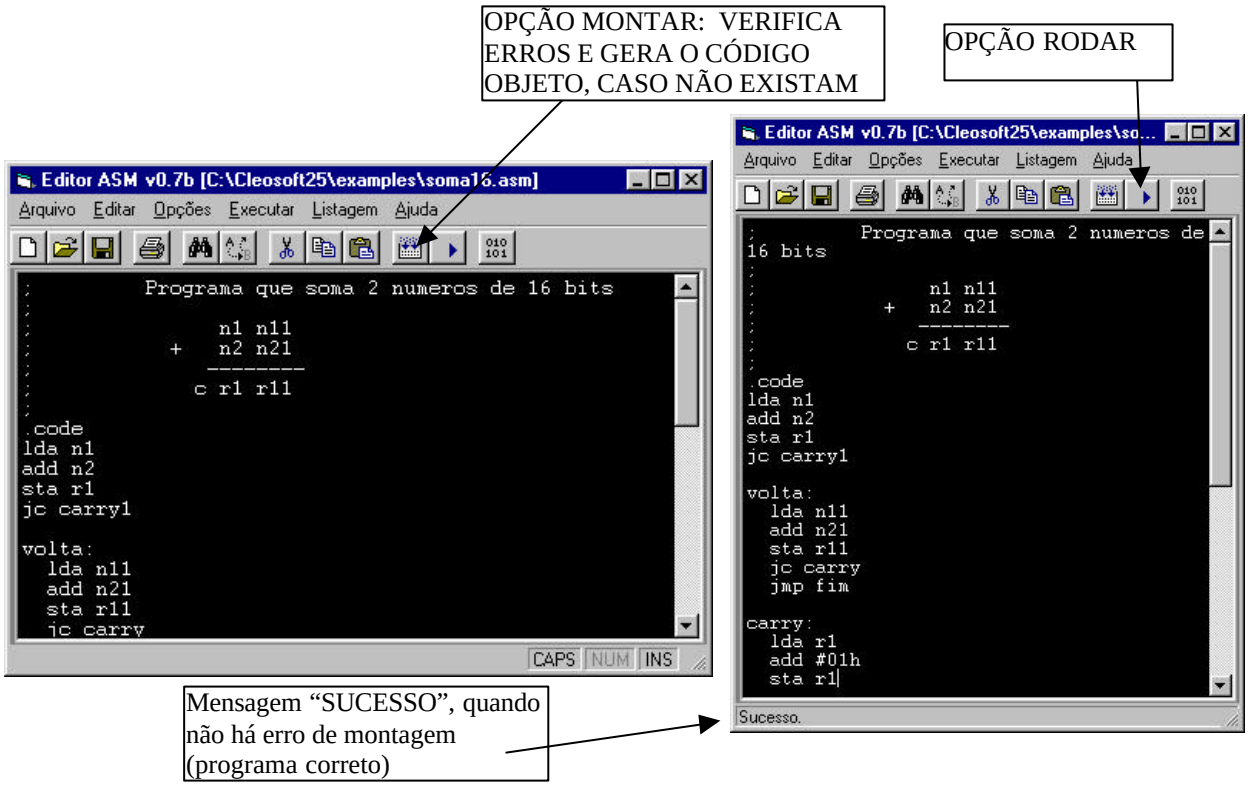
Obs: O ambiente de desenvolvimento Cleópatra já se encontra devidamente instalado nas máquinas do Laboratório. Caso deseje usá-lo fora do mesmo, faça o seguinte:

Copiar a distribuição Cleópatra da URL http://www.inf.pucrs.br/~moraes/org/cleosoft.zip. Basta descomprimir **cleosoft.zip** e executar o arquivo setup.exe para iniciar a instalação do simulador.

Abaixo detalhar-se-á a utilização básica dos programas do ambiente Cleópatra. A instalação básica vem com exemplos de programas que funcionam, programas estes de vários graus de complexidade. Mais exemplos de enunciados e soluções na página de download da disciplina.

6.2 Editor Cleópatra e Montador Cleópatra

- Função do programa montador: gerar o código objeto (arquivos **.cle*) a partir de arquivos texto contendo descrições de programas válidos em linguagem de montagem (assembly language) da Arquitetura Cleópatra. Se o programa não for sintaticamente correto, **cleoasm.exe** indica os erros (para uma descrição detalhada da sintaxe e da semântica da linguagem de montagem, ver o documento o_cleo2.01.pdf).
  - Entrada do montador: descrição em linguagem de montagem (assembly language) em texto ASCII, correspondendo a arquivos com terminação .asm.
  - Saídas do montador:
    1. código objeto (binário), terminação .cle;
    2. arquivos de listagem, terminação .txt;
    3. .hex para execução na placa de prototipação.
- Chamar o simulador a partir de "Start" → "Programs" → "Cleo Simulator 2.0" → "Cleo Simulator 2.0", ou caminho similar (depende da instalação e versão). A janela mostrada na Figura 31.a é exibida. Se não houver caminho, procurar diretório de nome **cleosoft20** ou similar no drive C. O nome do executável inicial do ambiente é **cleosim.exe**.



(a) janela inicial do montador. (b) montador com o programa soma16 carregado.

Figura 31 - Janela do programa Montador.

- A idéia aqui é abrir um programa fornecido com o software, a título de exemplo. Ir no menu “Arquivo”, opção “Abrir”. Escolher o arquivo **soma16.asm**, localizado no subdiretório **examples** da instalação do ambiente. Observar a estrutura do programa, com as *diretivas de montagem*[3] para designar início e fim da área de código (*.code* e *.endcode*), início e fim da área de dados (*.data* e *.enddata*), e definição de nome de área de dados e sua inicialização (*db*).
- Utilizar a opção de menu “Executar” → “Montar” (Tecla de atalho F5 ou antepenúltimo botão mais à esquerda) para gerar o código objeto. Caso o programa esteja corretamente escrito, a mensagem ***sucesso*** é exibida na parte inferior esquerda da janela do montador. Senão, uma mensagem de erro aparece no mesmo lugar. Alterar o programa para que isto ocorra, re-executando o montador e analisando a mensagem gerada. Por exemplo, na primeira linha após o rótulo *volta*, trocar o mnemônico *lda* por *lad*. Corrigir o erro e montar o programa outra vez.
- O programa em código objeto, com terminação *.cle*, será utilizado pelo simulador. Ele se encontra no mesmo diretório do programa em linguagem de montagem.

6.3 Simulador Cleópatra

- Após montar com sucesso o programa, clicar na opção de menu “Executar” → “Rodar” (Tecla de atalho *shift* F5 ou penúltimo botão mais à esquerda) na janela do editor. Ao efetuar o comando deverá aparecer uma janela equivalente a Figura 32:

[3] Uma *diretiva de montagem* contém informações para o programa montador, orientando-o na geração do código objeto. Não se deve confundir diretivas de montagem com *mnemônicos da linguagem de montagem*, que são textos que fazem com que o programa montador gere código objeto executável pelo processador.

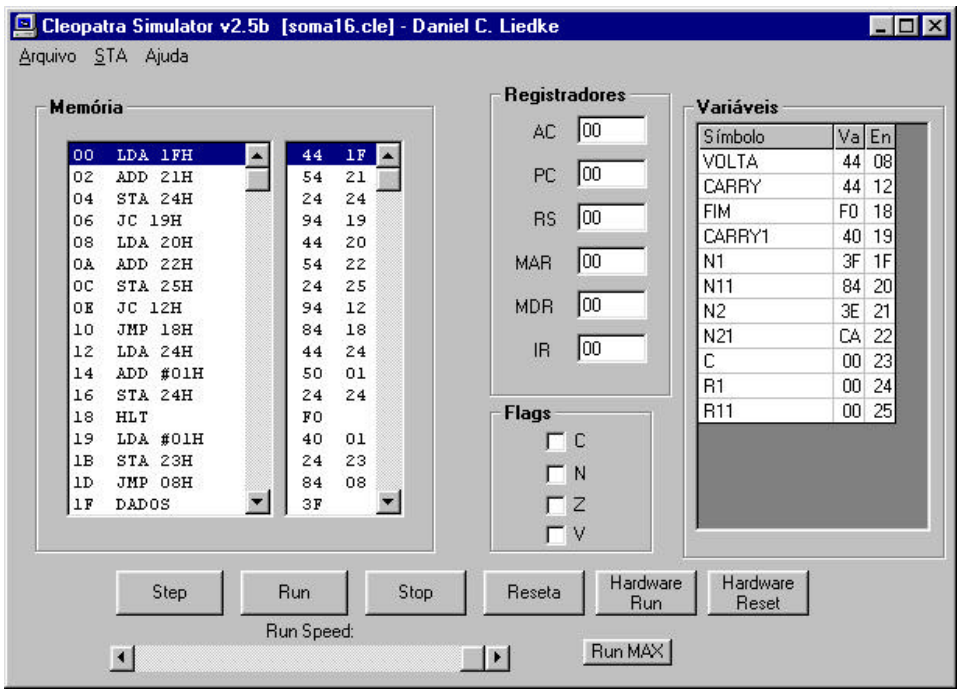


Figura 32 - Simulador da Cleópatra, com o arquivo soma16.cle carregado.

- Observar as 5 áreas desta janela:
    1. Conteúdo da memória principal do processador Cleópatra (áreas de dados e programas): a coluna da esquerda contém o código fonte produzido pela interpretação do código objeto em suas áreas de instruções e dados. A primeira área é delimitada pelas diretivas *.code*/*.endcode*, enquanto que a segunda área o é pelas diretivas *.data*/*.enddata*. A memória possui posições numeradas de 00H até 0FFH. Note que as instruções podem ocupar uma ou duas palavras de memória, enquanto que os dados são sempre mostrados palavra a palavra (identificados na coluna da esquerda pelo texto *DADOS* e na da direita pelo valor destes. A coluna da direita mostra o conteúdo total da memória em hexadecimal. A primeira coluna dá informação sobretudo sobre memória de programa (instruções), e a segunda sobretudo sobre a memória de dados.
    2. Registradores internos da arquitetura: acumulador (AC), contador de programa (PC), registrador de endereço de retorno de subrotina (RS), registrador de endereço de memória (MAR), registrador de dados da memória (MDR) e registrador de instrução corrente (IR).
    3. Qualificadores (flags): carry (C), negativo (N), zero (Z) e transbordo, ou overflow (V).
    4. Rótulos definidos pelo usuário (os rótulos de posições do programa e também os identificadores na memória de dados, criados usando a diretiva *db*).
    5. Controle de execução: do passo-a-passo até execução direta (até que se encontre a primeira instrução *hlt*). A opção “RESETA” permite reinicializar o estado do programa.

- Execute as três primeiras linhas do programa exemplo, utilizando a opção “STEP”.
    1. *LDA n1*. Foi traduzida para *LDA 1FH.* Observe se o conteúdo do acumulador após a execução desta primeira linha confere, ou seja, se AC está com 3FH.
    2. *ADD n2*. Instrução traduzida como *ADD 21H*. Após este passo executado, observe o conteúdo do acumulador, 7DH, que corresponde à soma de 3F + 3E.
    3. *STA r1.* Use o mouse para visualizar a posição de memória 24H (valor de r1) e observe que esta posição de memória recebeu o valor 7DH após a execução deste passo, e está marcada com um “*”, indicando que o endereço foi alterado pelo programa.

6.4 A Fazer e Entregar

Tarefa 1: Diga qual(is) a(s) diretiva(s) aceitas pelo montador Cleópatra, qual(is) sua(s) utilidade(s) e como funciona(m)?

Tarefa 2: Onde começam e terminam as memórias de instruções e de dados, respectivamente? O que acontece com todas as posições de memória não referidas explicitamente como memória de dados ou de programa no simulador?

Tarefa 3: Como se dá a geração dos endereços associados aos rótulos? Dê pelo menos dois exemplos de geração, pelo menos um de geração de rótulo para posição do programa e pelo menos um para posição da memória de dados.

Tarefa 4: Serão indicados, no momento da aula, quais exercícios cabem a cada equipe de alunos. Cada equipe deverá realizar 3 exercícios, 1 de cada grupo da lista definida na Tabela 3. Os grupos referem-se a exercícios com grau de dificuldade similar. O grau de dificuldade da lista abaixo é crescente (Grupo 1 < Grupo 2 < Grupo 3). Para cada exercício implementado deve ser entregue o **fluxograma**, o **código fonte** e **código objeto** em versão magnética, contendo o programa e dados sobre os quais este executa de maneira correta.

Tabela 3 - Especificação de programas para a Tarefa 4 (1 coluna por equipe, o professor atribui).

Grupo/Equipe	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10
Grupo 1	2	3	4	5	6	7	8	9	10	9
Grupo 2	16	15	14	13	12	11	12	13	14	15
Grupo 3	17	18	19	19	18	17	17	18	19	19

6.5 Lista de exercícios a serem resolvidos

IMPORTANTE: palavras em itálico na lista são rótulos associados a endereços específicos de memória.

- **GRUPO 1:**

1. Subtrair uma constante a um vetor armazenado à partir da posição de memória cujo rótulo é *end1*. O número de elementos está armazenado na posição de memória cujo rótulo é *end2*.
2. Fazer um algoritmo que lê um array, calcula o número de elementos pares e ímpares e informa qual é o maior par e o maior ímpar.
3. Escrever um programa para mover um vetor armazenado entre os endereços de memória com rótulos *inicio1* e *fim1* para os endereços cujos rótulos são *inicio2* e *fim2*. Assumir que as seguintes condições são verdadeiras: valor(*inicio1*) < valor (*fim1*), valor (*inicio2*) < valor (*fim2*), (*fim1*-*inicio1*) = (*fim2*-*inicio2*) = (tamanho do vetor –1).
4. Contar o número de posições de memória com conteúdo igual a uma constante, armazenada em uma posição de memória cujo rótulo é CTE, no vetor armazenado entre os endereços 080H e 0F5H.
5. Dados dois inteiros, A e B, armazenados, respectivamente, nas posições de memória cujos rótulos são *n1* e *n2*, armazenar na posição de memória com rótulo *max* o max(A,B) (valor máximo entre A e B) e na posição de memória com rótulo *min* o min(A,B) (definido de forma similar ao max), levar em consideração números positivos e negativos.
6. Descobrir se um número **n** é múltiplo de 4
7. Descobrir se um número **n** é múltiplo de 3.
8. Descobrir se um número **n** é múltiplo de um número **m** qualquer. Pressupor que **n** possa ser um número positivo ou negativo.
9. Faça um algoritmo que calcula a divisão de um número de 8 bits (armazenado na posição de memória com rótulo *v1*) por 2, através de subtrações sucessivas. Ao final do algoritmo a parte inteira da divisão

deve estar na posição de memória com rótulo *int* e o resto na posição de memória com rótulo *resto*.

10. Faça um algoritmo que calcula a divisão de dois números de 8 bits (armazenados nas posições de memória com rótulos *v1* e *v2*) através de subtrações sucessivas. Ao final do algoritmo a parte inteira da divisão deve estar na posição de memória com rótulo *int* e o resto na posição de memória com rótulo *resto*.

- **GRUPO 2:**

11. A transmissão de dados binários é o que viabiliza a existência de tecnologias como a Internet. A transmissão de dados a longas distâncias é uma tarefa muito propensa a erros, devido a efeitos ambientais externos (raios, interferências eletromagnéticas nas linhas de transmissão devidas a equipamentos elétricos, raios cósmicos ou manchas solares que interferem nas transmissões de satélites, etc.). Uma maneira de detectar erros de transmissão é acrescentar bits de controle ao dado transmitido, de forma que o receptor possa verificar a validade dos dados transmitidos. Um esquema simples de detecção de erros são as técnicas de *bit de paridade*. A idéia consiste em contar o número de bits de um determinado valor e acrescentar um bit na mensagem que diz se a contagem destes valores na mensagem é par ou ímpar. Assim, pode-se imaginar alguns tipos básicos de cálculo de paridade: paridade de 0s ou paridade de 1s, paridade par ou paridade ímpar, paridade ativa em 0 ou paridade ativa em 1. Implemente um programa que recebe *n*, o número de bits de uma mensagem (assuma, para facilitar, que 0<n<8), *msg*, a mensagem de tamanho n e produz uma nova mensagem *msgp* com n+1 bits, onde o bit mais significativo é a paridade da mensagem. Escolha o tipo de paridade e documente-o.

12. Multiplicar por somas sucessivas 2 inteiros positivos de 8 bits, armazenando o resultado em 16 bits. O multiplicando e o multiplicador devem estar armazenados nas posições de memória com rótulos *n1* e *n2*, respectivamente. O resultado deverá ser armazenado nas posições de memória com rótulos *mh* (a parte mais significativa do resultado) e *ml* (a parte menos significativa do resultado). O número de somas sucessivas deve ser o menor possível.

13. Fazer um programa que gere o *n* primeiros números da seqüência de Fibonnacci, e armazenar a seqüências em endereços consecutivos à partir da posição de memória com rótulo *idx*. Assuma que *n* está entre 0 e 14 e explique como se resolveria o problema surgido a partir de valores de *n* acima de 14. Primeiro defina qual é este problema.

14. Escreva um programa para criar um vetor *novo*, à partir de dois vetores *v1* e *v2* de mesma dimensão *n*, segundo a relação estabelecida na equação abaixo. A interpretação da equação é: cada elemento de *novo*, na posição *k*, receberá o somatório dos máximos dos vetores *v1* e *v2*, entre *0* e *k*.

$$novo_k = \sum_{i=0}^{k} \max(v1_i, v2_i), \quad 0 \le k < n$$

15. Escreva um programa para criar um vetor *novo*, à partir de dois vetores *v1* e *v2* de mesma dimensão *n*, segundo a relação estabelecida na equação abaixo. A interpretação da equação é: cada elemento de *novo*, na posição *k*, receberá o somatório dos mínimos dos vetores *v1* e *v2*, entre *0* e *k*.

$$novo_k = \sum_{i=0}^{k} \min(v1_i, v2_i), \quad 0 \le k < n$$

16. Dados dois vetores, *a* e *vet*, de dimensão *n*, implemente um algoritmo para preencher *vet* a partir de *a*, segundo a equação abaixo. A interpretação da equação é: cada elemento de *vet*, na posição *k*, receberá o somatório de todos os elementos do vetor *a*, entre os índices *0* e *k*.

$$vet_k = \sum_{i=0}^{k} a_i, \quad 0 \le k < n$$

- **GRUPO 3:**

17. Seja dado um ponto inicial (x0, y0) e uma seqüência de **n** valores inteiros (o valor de *n* deve estar armazenado em uma posição de memória com rótulo *n*, e a seqüência de valores deve estar armazenada como um vetor iniciando na posição de memória com rótulo *seq*). Implemente um algoritmo que desloque o ponto inicial alternadamente na horizontal e na vertical (iniciando com um deslocamento horizontal). Suponha que cada valor da seqüência dá a magnitude do deslocamento seguinte. Considere

que os deslocamentos são tais que o deslocamento atual e o imediatamente anterior circunscrevem um arco que avança no sentido anti-horário se o deslocamento atual for negativo, e no sentido horário se o deslocamento atual for positivo. Note que o deslocamento inicial pode ser em qualquer sentido horizontal, pois não há deslocamento anterior. Documente sua solução para indicar a escolha feita pelo seu programa neste caso.

Exemplo: dados de entrada [-6, -6, -4, 2, 3, 4] (ver Figura 33). Resultado -5 para x e -4 para y. Decisão: primeiro deslocamento para a esquerda se negativo, para a direita se positivo.

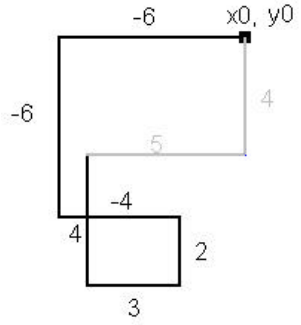


Figura 33 – Representação gráfica da entrada de dados [-6, -6, -4, 2, 3, 4].

18. Implemente um algoritmo simples de ordenação **crescente**. O vetor de origem deve estar armazenado entre as posições de memória de rótulos *ini1* e *fim1*, respectivamente, e o vetor ordenado deve estar armazenado entre as posições de memória *ini2* e *fim2*, respectivamente.
19. Implemente um algoritmo simples de ordenação **decrescente**. O vetor de origem deve estar armazenado entre as posições de memória de rótulos *ini1* e *fim1*, e o vetor ordenado deve estar armazenado entre as posições de memória *ini2* e *fim2*.

- **GRUPO AVANÇADO (PARA QUEM DESEJA IR MAIS LONGE):**

20. Faça um programa para ordenar 20 valores hexadecimais, que estão armazenados em 10 bytes. Sendo que dentro de cada byte, o nibble (conjunto de 4 bits) menos significativo deve conter o menor valor.

    Exemplo: Vetor de entrada: 034h, 0FAh, 0BCh, 077h, 01Ch
    Vetor ordenado: 013h, 047h, 07Ah, 0BCh, 0CFh

21. Escrever um algoritmo que calcule os primeiros *n* números primos e os armazene seqüencialmente, a partir da posição de memória cujo rótulo é *nprimos.*
22. Fazer um programa para automatizar uma agência de casamentos. Considere que a agência dispõe do cadastro de n homens e n mulheres. Neste cadastro existe uma ordenação de preferencia que os homens e as mulheres tem uns pelos outros. Selecione os n casais de forma a favorecer a melhor escolha do casal.
    Exemplo:

Número	Homens	Mulheres
1	1, 2, 3	3, 2, 1
2	1, 3, 2	3, 1, 2
3	3, 2, 1	1, 3, 2

    Neste caso o homem 1 e 2 tem como preferência maior a mulher 1, mas esta tem como maior preferência o homem 3

23. Dadas **n** caixas de 3 dimensões, fazer um programa para descobrir qual é o maior número de caixas que podem ser colocadas umas dentro das outras. Cada caixa pode ser representada, neste problema, como um vetor de três dimensões (**dx**, **dy**, **dz**) e as **n** caixas formam uma matriz **n** x 3.
24. Seja dada uma máquina que tem somente duas operações **inteiras**: multiplicar por 2 e dividir por 3, e que conhece inicialmente o número 1. Fazer um programa que gere qualquer número de 1 a 18 como uma expressão: número = (2^X)/(3^Y).

    Exemplo: 2 = 2^1/3^0 3 = 2^5/3^2 4 = 2^2/4^0 5 = 2^4/3^1

7 Modelo de Computador de Programa Armazenado – Arquitetura Cleópatra: Bloco de Dados, Bloco de Controle

Prática: Finalização e Simulação do Processador Cleópatra

Recursos: Projeto da Arquitetura Cleópatra (esquemático) e CAD Foundation

7.1 Introdução e Objetivos

O **Laboratório sobre o Modelo de Computador de Programa Armazenado e Programação em Linguagem de Montagem** envolveu a prática com a linguagem de montagem (em inglês, *assembly language*) do processador Cleópatra, mediante emprego do sistema de desenvolvimento de software do mesmo. Neste Laboratório, a ênfase muda para a implementação em esquemáticos do processador Cleópatra, passando de preocupações arquiteturais para problemas organizacionais.

Neste Laboratório, partir-se-á de uma versão parcialmente implementada da Cleópatra. O trabalho prático incluirá completar o projeto do bloco de controle do processador e simulá-lo corretamente, para um pequeno programa em linguagem de montagem. A parte a completar é a ROM de tradução de conteúdos do IR para microendereços. Após terminado o diagrama de esquemáticos da Cleópatra, deve-se realizar a simulação do mesmo, usando um script de simulação a ser implementado pelo grupo de alunos. Um exemplo de script (arquivo ***Completa.cmd***) e de suas formas de onda de saída (arquivo ***Completa.des***) estão disponíveis no projeto **Cleo_p.zip**.

7.2 A Fazer e Entregar

Tarefa 1: Buscar o arquivo Cleo_p.zip da homepage da disciplina. (não descompacte este arquivo manualmente). Abra o Foundation . No menu "File", escolha a opção "*Restore Project*". Escolha o arquivo *Cleo_p.zip* no seu diretório de trabalho. Esta opção irá descompactar o arquivo "*Cleo_p.zip*". Cuidar para indicar no "*wizard*" o local correto onde instalar o projeto. Caso o projeto não seja aberto após recuperação, vá ao menu File → Open Project. Abrir o projeto cleo_p

Tarefa 2: Percorrer o esquemático reconhecendo os blocos de dados e de controle.

O Bloco de Dados deste projeto está completo, mas o Bloco de Controle não. Entrando três níveis na hierarquia, chega-se até o interior do BC. A Figura 34 à esquerda mostra como deve aparecer a tela neste nível, que apresenta as duas partes principais do BC: a MicroROM e o seqüenciador. A MicroROM está completa, falta apenas descrever uma das partes do seqüenciador, qual seja, o a tradução de IR e qualificadores para microendereços. Entrando dentro do seqüenciador, é possível visualizar esta parte do projeto em nível de interface. A Figura 34 à direita mostra como deve aparecer a tela do Editor de Esquemáticos, após descer na hierarquia como indicado. O retângulo abaixo e ao centro do esquemático da Figura 34 é o que deve ser implementado nesta aula.

O bloco **IR2END** teve apenas sua interface e lógica definidas no projeto parcial. Para implementar seus conteúdos, será necessário implementar uma ROM. Para tanto, existe um utilitário do sistema de CAD Foundation que automatiza a criação de blocos regulares complexos, e que deverá ser empregado. Trata-se da ferramenta denominada **LogiBlox Module Generator**. Os conteúdos da citada ROM devem ser inferidos a partir da Tabela da última página da especificação Cleópatra (Versão 2.0 ou mais atual).

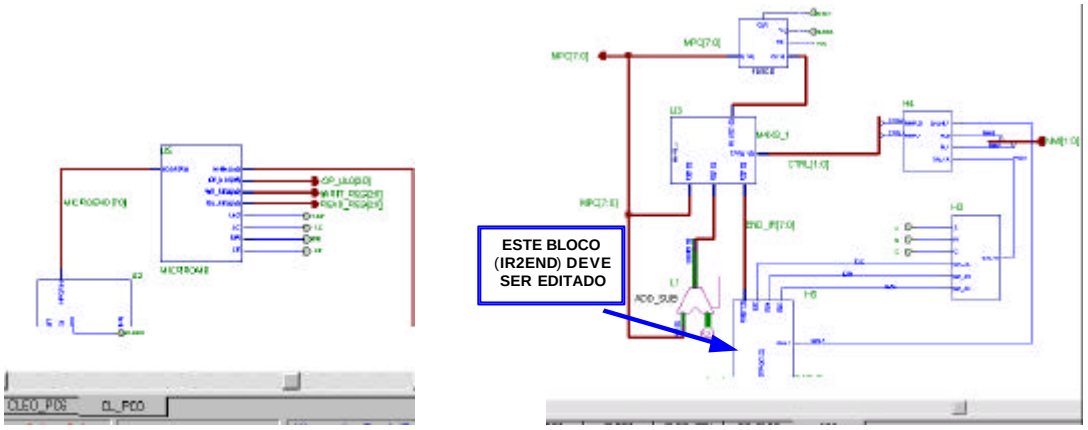


Figura 34 – Esquemáticos intermediários do Bloco de Controle Cleópatra.

Tarefa 3: Finalizar a implementação do bloco de controle, completando o módulo IR2END. Para criar a ROM em questão, inicie pela escolha, no editor de esquemáticos, do menu: *Tools* ➔ *LogiBLOX Module Generator*. Esta escolha abre uma janela de diálogo para criar diversos blocos de forma automática. Clique no campo *Module Types* e escolha *Memories*. A partir daí a janela de diálogo deve apresentar-se como na Figura 35. **ENTREGAR: listagem do arquivo que define a ROM de tradução, com comentários e a janela do esquemático contendo a ROM.**

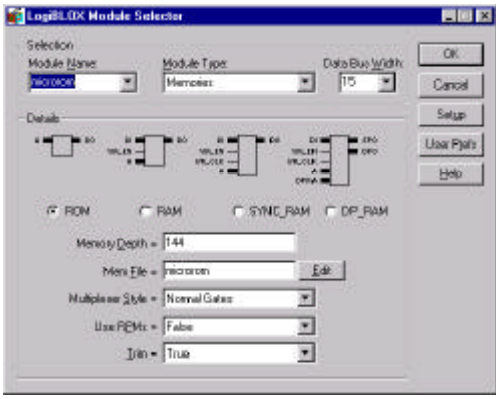


Figura 35 - Janela de parametrização de módulos da ferramenta LogiBlox Module Generator.

Para prosseguir, **configure** os campos desta janela (os números da janela estão errados):

- especificando um nome para a ROM;
- garantindo que a opção ROM é a escolhida;
- especificando o número de saídas (campo *Data Bus Width*);
- o número de linhas da ROM (campo *Memory Depth*);
- um nome do arquivo que conterá o texto de especificação da ROM (pode ser o mesmo nome do módulo).

No Apêndice deste laboratório há detalhes de como preencher a ROM. Caso haja dúvidas sobre como preencher os campos, clique em *Help*, depois clique em *Index*, e digite ROM para chegar na opção de ajuda para módulos do tipo ROM. Após configuradas todas as opções, e não antes, deve-se editar o arquivo, clicando-se na opção *Edit*. As instruções de como preencher o arquivo com os dados da ROM encontram-se nas mesmas páginas de ajuda mencionadas antes. Terminando a edição do arquivo, basta sair do editor e clicar em *Ok* para que a geração da ROM tenha lugar. Caso haja erros, o arquivo foi mal preenchido. Deve-se, então, corrigir os erros e re-executar o processo de geração.

Erros comuns:

1. O número de linhas da ROM deve ser múltiplo de 16;
2. O número de dados especificados no arquivo deve ser exatamente igual ao número de linhas da ROM.

Se a geração da ROM tiver sucesso, uma nova célula é inserida na biblioteca do projeto, com o nome do módulo escolhido. Agora basta inserir este módulo e conectá-lo aos pinos de entrada e saída do bloco IR2END, não esquecendo verificar se tudo está certo com as portas lógicas detectoras de HLT/JN/JZ/JC/JV.

Tarefa 4: Simulação do processador. Recuperar o arquivo ***Completa.cmd*** do projeto e estude-o. **RESPONDA: quais são os sinais necessários para definir a execução de um programa completo?** A Figura 36 apresenta o resultado da simulação das 4 primeiras instruções do trecho de programa abaixo.

```
;                       clocks    object code
prgtst: lda #0FFH     ;  6     00 40 FF   AC initialized with FF
        sta minus1    ;  7     02 24 80   memory write using direct AM
        lda minus1,R  ;  8     04 4C 7A   restore using relative AM, setting N
        add pone,I    ; 10     06 58 83   add with indirect AM setting flags C/Z

 Data area            ;  Total of 71 clocks
        org #80h      ;
minus1: db  #00h      ;        80 00        intialized to 0
xaxa:   db  xexe      ;        81 14        generates indirect address
one:    db  #01h      ;        82 01        the constant 1
pone:   db  one       ;        83 82        pone is a pointer to the constant 1
```

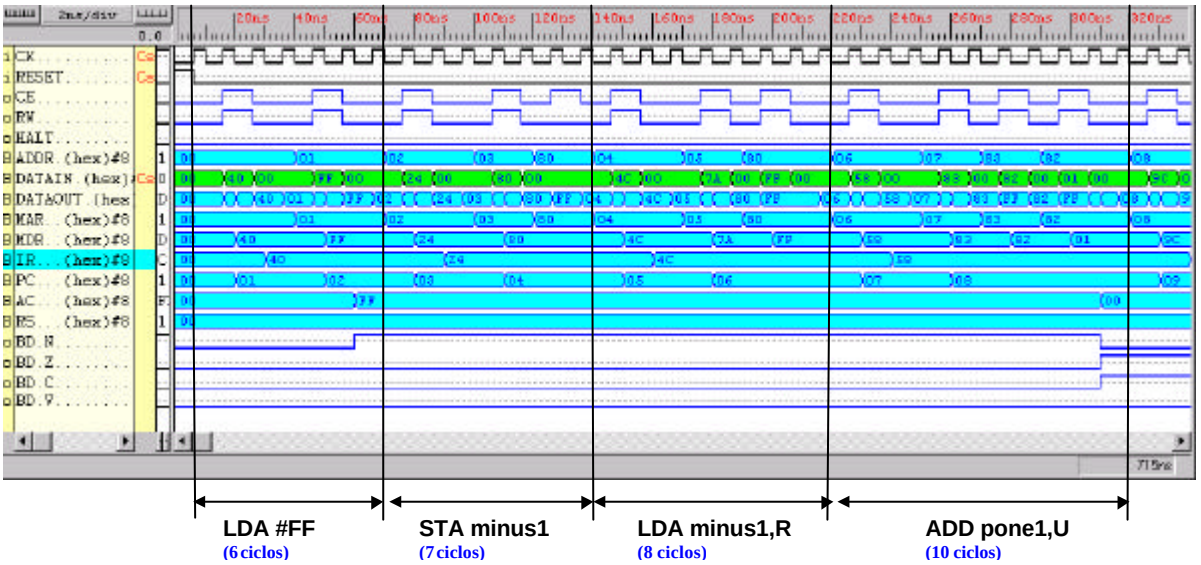


Figura 36 - Exemplo de simulação de 4 instruções.

- Durante a aula será escolhido um conjunto de exercícios da Tabela 4 para cada grupo. Os grupos devem realizar a simulação completa de trecho de programa. **ENTREGAR: *script* de simulação e janela com a simulação das 4 instruções. Deve-se mostrar na simulação o que ocorre a cada ciclo de clock, em cada registrador.**

Tabela 4 - Grupo de Instruções para serem simuladas.

Conjunto 1	Conjunto 2	Conjunto 3	Conjunto 4	Conjunto 5	Conjunto 6	Conjunto 7	Conjunto 8	Conjunto 9	Conjunto 10
LDA #ABh	LDA #1Bh	LDA #ABh	LDA #3Eh	LDA #22h	LDA #11h	LDA #67h	LDA #E3h	LDA #FFh	LDA #12h
ADD 0F1h	OR 12h,I	AND 0A5h,I	ADD 0C5h	OR 56h,I	AND 77h	ADD 48h,I	OR 0F9h,I	AND 0D0h,I	ADD 0CCh
ADD 0E0h,I	AND 25h	STA 91h	AND 56h,I	ADD 69h	STA 91h,I	OR 34h	ADD 0F8h	STA 0F8h	ADD 18h,I
JC 0FAh,I	JN 0FAh,R	JZ 0FAh	JN 0FAh,R	JV 0FAh,I	JN 0FAh	JZ 0FAh,I	JV 0FAh,R	JN 0FAh	JC 0FAh,R

Obs.1: Para os modos direto e indireto o aluno deverá escolher os conteúdos de memória.

Obs.2: Escolher os dados de forma a forçar que sempre ocorra o desvio na última instrução. *Exemplo*: supondo que o seu seja o "Conjunto 1", vocês teriam o seguinte mapa de memória:

Endereço	0	1	2	3	4	5	6	7	20h	E0h	F1h
Conteúdo	LDA	ABH	ADD	F1h	ADD	E0h	JC	FAh	42h	20h	34h

Ou seja, vocês atribuiriam o conteúdo 20h à posição 0E0h, o conteúdo 34h à posição 0F1h e o conteúdo 12h à posição 20h. No final da micro-simulação, vocês deverão ter no acumulador a soma ABh+34h+42h, ou seja, 21h e carry.

ATENÇÃO1: Note que existe uma defasagem de meio período de relógio entre ações no BD e no BC, para que os comandos do BC e os qualificadores do BD estabilizem após uma mudança, antes de serem usados pelo outro bloco (ver inversor entre o sinal de clock de entrada e o BD).

ATENÇÃO2: Existem alguns problemas na simulação, devido ao pessimismo do modelo usado pelo simulador lógico do Foundation para barramentos tristate. Caso sua simulação acuse erro de conflito nos barramentos BUSA e BUSB, escolha ignorar o erro até o fim da simulação. Após, analise se existe algum problema na simulação (linhas em X ou Z). Se estas não existirem, não há problemas com a simulação.

7.3 Apêndice - O que deve ser colocado na ROM IR2END

- O endereço da ROM é formado pelos 6 bits mais significativos do IR. Supor:

  STA: 24H ou 0010 0100
  O endereço na ROM serão os 6 bits mais significativos, ou 001001 = 9 (decimal) ou 09H (hexadecimal)

- Na posição 9 da ROM deverá ser escrito o valor 04. Atenção: a posição 9 corresponde à linha 10, pois o endereço começa em 0.

A Tabela 5 ilustra a relação entre as instruções e o endereço correspondente para a sua execução em ROM.

Tabela 5 – Relação entre a instrução e o endereço em ROM.

Instrução	Valor em Hexa da Instrução	Endereço correspondente p/ a ROM (decimal)	Conteúdo da ROM (μendereço de Início)
NOT	**00**	**0**	**03H**
STA	**24**	**9**	**04H**
STA ,I	**28**	**10**	**08H**
STA ,R	**2C**	**11**	**0EH**
LDA #	**40**	**16**	**12H**
LDA ,D	**44**	**17**	**15H**
LDA ,I	**48**	**18**	**1AH**
LDA ,R	**4C**	**19**	**21H**

Continuar …

- No arquivo texto deverá ser inserido o conteúdo da ROM após o campo DATA:

```
; Data Section
; Specifies data to be stored in different addresses
; e.g., DATA 0:A, 1:0
RADIX 16
DATA  ..........................................
03                ; posição 0
03
03
03
03
03
03
03
04                ; endereço correspondente ao STA
08                ; endereço correspondente ao STA ,I
 ……
```

Conteúdo que deve ser preenchido a mão pelo aluno.

Porque nas 8 primeiras posições da ROM se armazena o mesmo valor (03H)? Isto ocorre em outras partes desta ROM? Onde?

8 Implementação de Sistemas Digitais com HDLs Ferramentas de Captura e Validação

Prática: Implementação de um Circuito Acumulador

Recursos: Ambiente de Desenvolvimento Active-HDL da Aldec, Inc.

8.1 Introdução e Objetivos

Os laboratórios anteriores referiam-se às Unidade I e II da disciplina. Eles serviram para dar embasamento na captura de projeto, simulação e síntese de sistemas digitais baseados em diagramas de esquemáticos, diagramas de transição de estados e geradores automáticos de módulos. Também foi exercitada, nestes laboratórios, a síntese física de sistemas digitais visando sua implementação sobre FPGAs e a configuração e teste destes dispositivos sobre plataforma educacionais, em particular sobre a plataforma XS40/XST-1 da empresa Xess, disponível no ambiente da disciplina. O último destes laboratórios visou consolidar os conhecimentos adquiridos, através de um projeto específico. A partir do presente laboratório, inicia-se a Unidade III da disciplina, onde o foco passa a ser uma forma mais abstrata e mais moderna de desenvolver sistemas digitais, qual seja, 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 exemplo, um acumulador de valores de 8 bits.

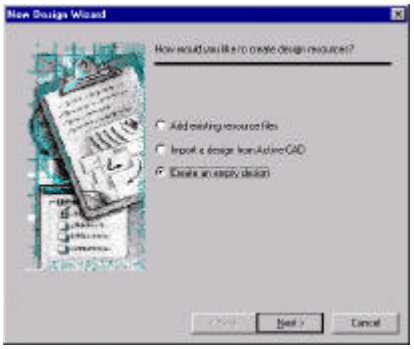
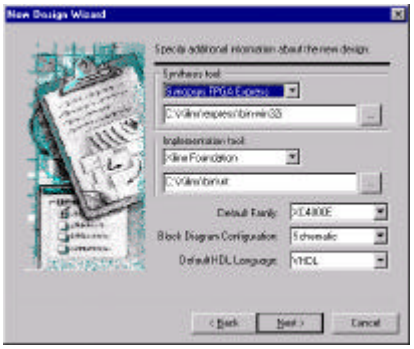
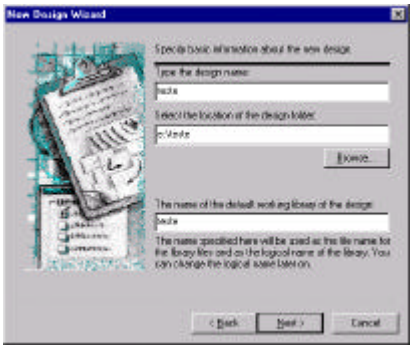
8.2 Localização do Ambiente de Desenvolvimento Active-HDL

- Verificar se o simulador está instalado. Normalmente, há o ícone do "**Active-HDL 4.2**" no *desktop*. Se o ícone não estiver disponível no seu ambiente de trabalho, verifique em **C:\Program Files\Aldec\Active-HDL 4.2\Bin\avhdl.exe**, ou tente localizá-lo via menu **Start** do Windows.

Active-HDL 4.2.lnk

8.3 Início do Projeto

- Crie um novo projeto (Figura 37). Para tanto, na janela de diálogo inicial, escolha o marcador *Create New Design* e clique no botão Ok.

		
Crie um projeto vazio.	Janelas de parâmetros – não é importante agora.	Defina o nome do projeto e o diretório onde este será armazenado (Use seu diretório H:/ ou algum diretório no disco local C:).

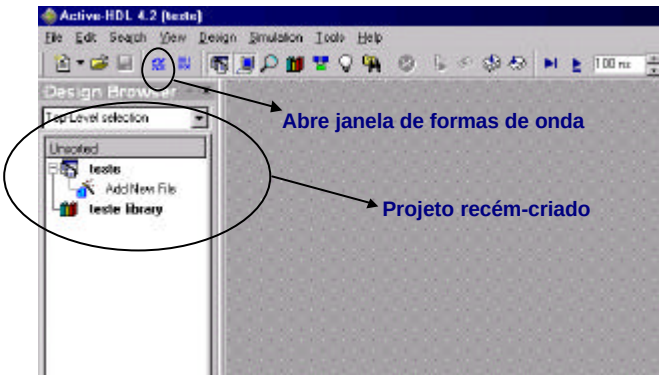


Figura 37 - Criação de um novo projeto VHDL.

8.4 Implementação de um Circuito Acumulador

- Acumuladores são circuitos digitais básicos muitos empregados, compostos de um registrador que guarda resultados de um processo de acumulação e um operador combinacional que realizada as operações parciais cujos resultados são acumulados. Os operadores mais comumente usados são somadores ou multiplicadores ou combinações destes, mas outros são possíveis. No caso, trabalharemos com um somador.
- Um diagrama de blocos sucinto do circuito que deverá ser implementado está mostrado na Figura 38. Os sinais com nomes em caracteres pretos indicam entradas e/ou saídas externas (parte da entity do módulo VHDL), enquanto que os sinais com nome em azul são sinais internos do circuito. O somador e o registrador constituem o sistema acumulador, enquanto que o contador de oito bits é um exemplo muito simples de um gerador de parcelas da soma.

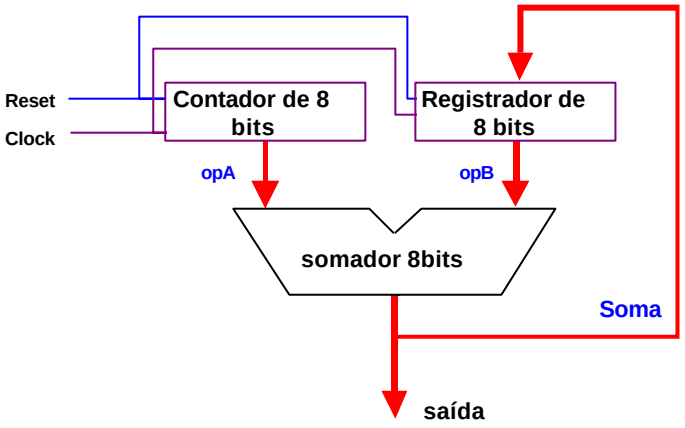


Figura 38 - Diagrama de blocos do sistema digital acumulador.

Criação do arquivo VHDL

- Seu projeto deve ter como entradas os sinais de **reset** e **clock**, e como saída um vetor **saída** de 8 bits.
- No menu File→New→VHDL Source. Siga os seguintes passos para a criação do arquivo:
  1. Verifique se a opção "*Add the generated file to the design*" está selecionada e "*next*".
  2. Indique o nome do arquivo, por exemplo "acumulador", e "*next*".
  3. Inicia-se agora o procedimento de inserção de pinos. Insira os pinos "clock", "reset" e "saída" conforme a descrição que segue e a Figura 39:
     - "new", "name" clock, direção "in";
     - "new", "name" reset, direção "in";
     - "new", "name" saída, direção "out", 8 bits.

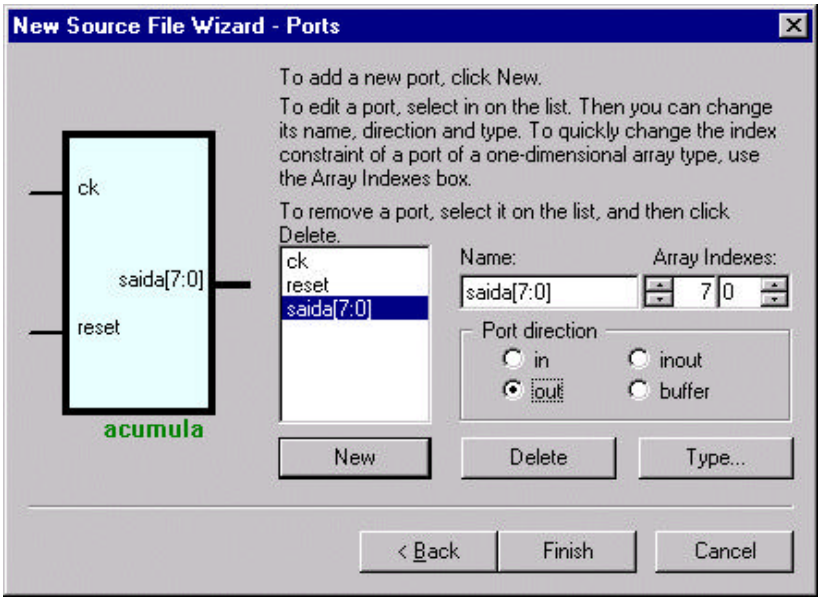


Figura 39 - Janela de inserção do barramento de saída chamado "saída".

- Ao final da inserção dos pinos é aberta a janela com o modelo do arquivo VHDL. Observar na Figura 40 que o arquivo criado já contém a referência à biblioteca IEEE e ao *package* pertinente, onde os tipos *std_logic* estão definidos. Também aparece a definição da *entity* do seu módulo principal, faltando apenas completar a parte referente à arquitetura (*architecture*), cujo cabeçalho já está disponível.

```
--
--  File: D:\home\moraes\acumulador\SRC\acumulador.VHD
--  created by Design Wizard: 05/03/99 08:44:19
--
library IEEE;
use IEEE.std_logic_1164.all;
entity acumulador is
    port (
        clock: in STD_LOGIC;
        reset: in STD_LOGIC;
        saida: out STD_LOGIC_VECTOR (7 downto 0)
    );
end acumulador;
architecture acumulador of acumulador is
begin
  -- <<enter your statements here>>
end acumulador;
```

Figura 40 - Arquivo aberto após a inserção dos pinos.

Projeto do Acumulador

O projeto deste acumulador será decomposto em 3 partes: a definição dos elementos comuns, o circuito propriamente dito e o gerador de teste ou *test bench.*

- **Primeira parte do projeto**: *Definição do elementos comuns.* Aqui utilizaremos o conceito de "*package*" da linguagem VHDL, onde serão definidos o tipo *regsize* (vetor de 8 bits) e uma subrotina (*procedure*) para realizar a soma, denominada *somaAB*. O código encontra-se na Figura 41. Utilizar o *help* da ferramenta para compreender o significado das palavras reservadas de VHDL.

  Importante: toda a descrição do *package* é inserida no início do arquivo, na linha 1, **antes dos comentários inseridos pelo *wizard***. Não se esqueça de que a cada nova **design unit** do tipo *package* ou *entity*, é necessário definir as bibliotecas usadas pela entidade em questão. Ou seja, o contexto de uma declaração de uso de uma biblioteca restringe-se à unidade de projeto (design unit) definida imediatamente após a declaração.

```
library IEEE;
use IEEE.Std_Logic_1164.all;

package comum is

    subtype regsize is std_logic_vector(7 downto 0);
    procedure somaAB  (     signal A,B: in  regsize;   signal Cin: in STD_LOGIC;
                            signal S:   out regsize;   signal Cout:out STD_LOGIC);
end comum;

package body comum is

    procedure somaAB  (     signal A,B:  in  regsize;  signal Cin:  in STD_LOGIC;
                                signal S:    out regsize;  signal Cout: out STD_LOGIC) is
    variable carry : STD_LOGIC;
    begin
          for w in 0 to 7 loop
                if w=0 then carry:=Cin;  end if;
                S(w) <= A(w) xor B(w) xor carry;
                carry := (A(w) and B(w)) or (A(w) and carry) or (B(w) and carry);
          end loop;
          Cout <= carry;
    end somaAB;
end comum;
```

Figura 41 - Estado do texto VHDL após a definição dos elementos comuns.

- **<u>Segunda parte do projeto</u>**: *O circuito propriamente dito.* Iniciar logo após o término do package (*end comum*). Acrescentar, às bibliotecas inseridas pelo *Design Wizard*, a biblioteca "IEEE.std_logic_unsigned.all", que habilita o uso do operador '+' entre tipos *std_logic* e *std_logic_vector*, a serem utilizados no contador, e o package "*comum*", que conterá o somador e a definição do tipo *regsize*. O texto deve ficar como na Figura 42.

```
library IEEE;                                  } Já inserido pelo wizard
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;               } Bibliotecas adicionais que devem ser inseridas
use work.comum.all;

entity acumulador is                           }
    port (                                     }
        clock: in STD_LOGIC;                   } Já inserido pelo wizard
        reset: in STD_LOGIC;                   }
        saida: out STD_LOGIC_VECTOR (7 downto 0)}
    );                                         }
end acumulador;
architecture acumulador of acumulador is ...... } Início do resto do projeto
```

Figura 42 - Estado do texto VHDL após a definição das bibliotecas adicionais.

- Declarar entre a *architecture* e o *begin* os sinais internos do circuito. Para este exemplo serão necessários três barramentos - *opA*, *opB* e *soma* - e dois sinais para o somador - *cin* e *cout*.

```
architecture acumulador of acumulador is
   signal soma, opA, opB : regsize;
   signal cin, cout : std_logic;
begin
```

- Agora deve-se montar o circuito entre o *begin* e o "*end acumulador*". Teremos 4 módulos executando em paralelo: dois registradores, o somador e a conexão da saída do registrador à saída do circuito. Ver Figura 43.

```
cin <= '0';                                    -- somador
s1: somaAB( opA, opB, cin, soma, cout);

saida <= soma;                                 -- conexão da saída do registrador à saída do circuito

process (clock, reset)                         --- registrador incrementador de 8 bits, com entrada e saída opA
begin
   if reset ='1' then   opA <= (others=>'0');
      elsif (clock'event and clock='1') then  opA <= opA + 1;
   end if;
end process;

process (clock, reset)                         --- registrador de 8 bits, com entrada soma e saída opB
begin
   if reset ='1' then   opB <= (others=>'0');
      elsif (clock'event and clock='1') then  opB <= soma;
   end if;
end process;
```

Figura 43 - Estado do texto VHDL após a definição da arquitetura do acumulador.

- **Terceira parte do projeto**: *O gerador de teste ou test bench*. Sua função é gerar os estímulos para o circuito descrito. Para isto, declara-se o circuito implementado (no caso o acumulador) como um componente do test bench, utilizando-o na *architecture* deste último. Depois gera-se o *reset* com uma atribuição e o *clock* com um processo. O processo de geração do test bench está descrito na Figura 44.

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.comum.all;

entity tb is                                   -- entidade sem pinos
end tb;

architecture teste of tb is

   component acumulador                        -- declara que vai utilizar a entidade acumulador
      port (copiar aqui as portas do acumulador );
   end component;

   signal reset, ck : std_logic;               -- sinais internos
   signal saida : regsize;

begin

  -- instância do acumulador
  a1:  acumulador port map(clock=>ck, reset=>reset, saida=>saida);

  reset <= '1' , '0' after 10 ns;              -- gera o reset com atribuição

  process                                       -- gera o clock
     begin
         ck <= '1' after 10ns, '0' after 20ns;
         wait for 20ns;
    end process;

end teste;
```

Figura 44 - Conteúdo do arquivo VHDL que descreve o test bench do circuito acumulador.

8.5 Compilação e Simulação

- Uma vez descrito o test bench, compila-se a descrição até não haverem mais erros. Use a opção Menu Design → Compile All.

- Depois deve-se realizar a simulação. Os passos para isto são:
  1. Opção de menu *Simulation* → *Initialize Simulation*. Nesta parte deve-se escolher qual módulo iremos simular. Deve-se escolher o módulo *tb*, pois este é o módulo de mais alto nível na hierarquia.
  2. Abrir o diagrama de tempos, clicando no botão relativo a esta função (ver Figura 37).
  3. Inserir na coluna da esquerda do diagrama de tempos os sinais que se deseja visualizar. O procedimento é simples: seleciona-se o módulo desejado e arrasta-se seu ícone com o mouse apertado para o diagrama de tempos, o que acrescenta todos os sinais na janela de visualização da simulação. Processo similar pode ser usado para mostrar um ou mais sinais individuais.
  4. Escolha a opção de menu *Simulation* → *Run For* tantas vezes quantas for necessário para enxergar uma porção significativa do comportamento do circuito.

- Verifique se a simulação está funcionando corretamente, e relate as observações no relatório da aula.

8.6 A fazer e a entregar

Tarefa 1: **Estude, Pense e Responda**: O quê realiza o circuito da Figura 38 na sua opinião? Uma resposta possível é que se trata de um circuito capaz de calcular a soma dos **n** primeiros termos de uma **progressão aritmética** de **termo inicial** 0 e **razão** 1. Porquê? Que parte do circuito gera **n**?

Tarefa 2: Na descrição apresentada na Figura 43 é possível usar ao invés de dois processos (*process*), um para cada registrador, apenas um processo que cria simultaneamente os dois registradores, uma vez que ambos são controlados pelos mesmos sinais de clock e o reset. Faça isto.

Tarefa 3: Qual a freqüência gerada pelo processo do teste bench que produz o sinal de clock? Altere o teste bench para produzir uma freqüência de clock de 33,333MHz e refaça a simulação. Salve ambas as formas de onda de simulação para entrega junto com o projeto.

Tarefa 4: Arquivar o projeto completo (opção de menu Design → Archive Design... do Ambiente Active-HDL), com as formas de onda salvas para duas freqüências distintas de simulação (conforme Tarefa 3);

Tarefa 5: Fazer o relatório com as respostas às questões aqui colocadas dentro das Tarefas;

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

A Figura 45 apresenta uma visão geral do ambiente de desenvolvimento da ferramenta Active-HDL. Alguns itens importantes estão comentados em azul.

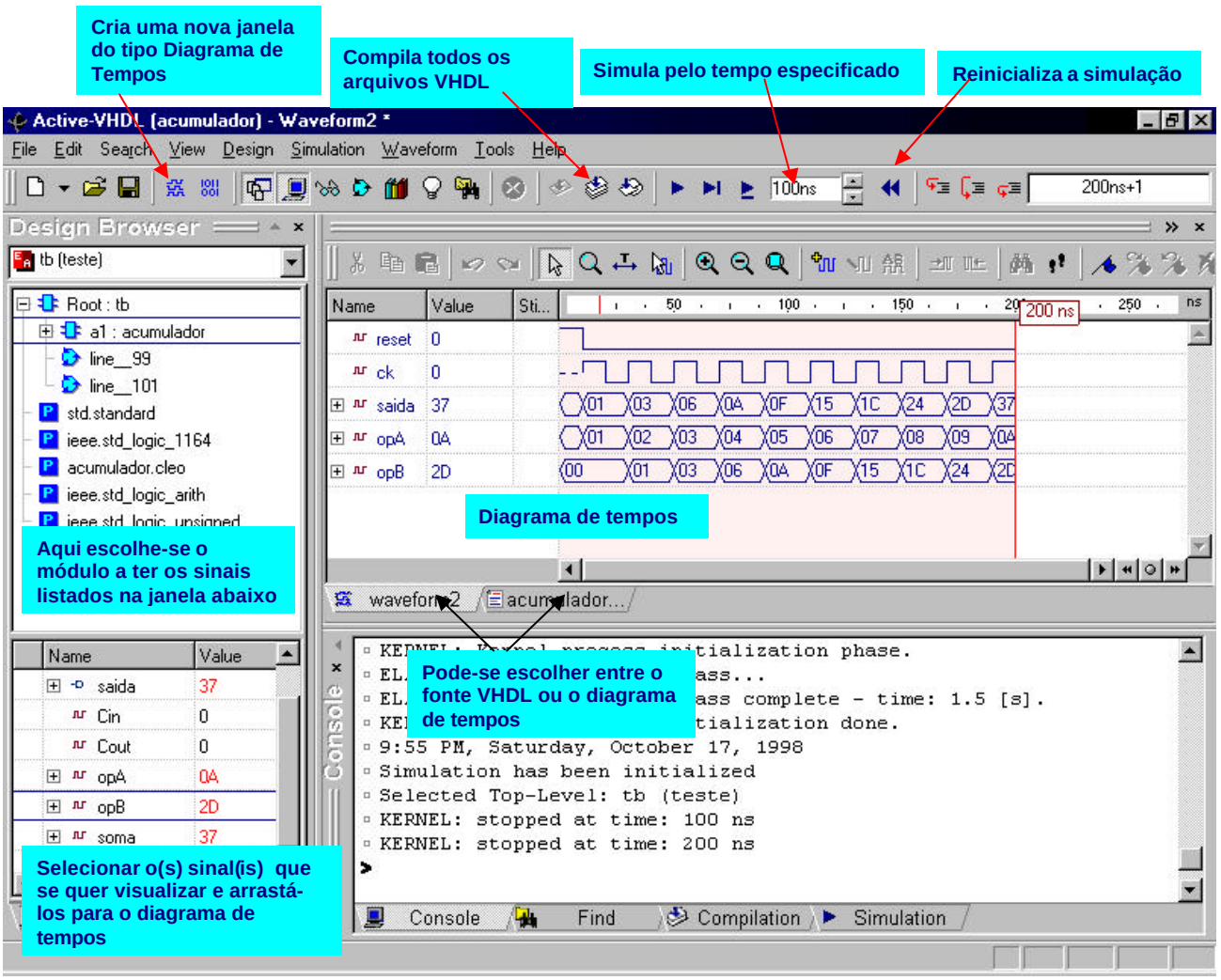


Figura 45 - Visão Geral do Ambiente de Desenvolvimento Active-HDL

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.

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

Prática: Ativação dos leds e displays da plataforma XS40/XST-1

Recursos: CAD Foundation e Ferramentas XSTools da Xess, Inc.

9.1 Introdução e Objetivos

O primeiro Laboratório sobre HDLs 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 exemplo, um acumulador de somas.

O objetivo específico do presente Laboratório será o de investigar um fluxo de projeto a partir de uma descrição inicial em linguagem de descrição de hardware, desde a captura do projeto até a sua implementação em hardware. Isto será feito mediante um exemplo simples, que deverá ser capturado, validado funcionalmente via simulação e sintetizado para a plataforma educacional XS40/XST-1 da empresa Xess. O exemplo deverá então ser descarregado na plataforma e testado. Finalmente, um exercício de alteração do projeto original, sua validação, síntese, descarga e novo teste devem ser realizados pelos alunos. Adicionalmente, este Laboratório apresenta uma breve introdução a FPGAs, os dispositivos sobre os quais as plataformas de prototipação usadas em aula estão baseadas.

9.2 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 HDLs.

- Síntese Lógica
  - Processo de transformar a descrição VHDL em uma descrição a nível de transferência entre registradores (designada em inglês por *Register Transfer Level* ou *RTL*) equivalente, menos abstrata. A descrição RTL é um arquivo textual descrevendo uma interconexão de portas lógicas e flip-flops, independente de dispositivo.
- Síntese Física
  - Processo de mapear a descrição RTL nos elementos do FPGA (no caso da Xilinx e do Foundation. Em outros ambientes, poderia ser para outro tipo de implementação, e.g. um *chip* fabricado sob encomenda).
  - As ferramentas básicas são: 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), posicionamento (que distribui as CLBs nas LUTs específicas do FPGA, em linhas e colunas da matriz do dispositivo escolhido) roteamento (que conecta as CLBs entre si, por meio da interconexões reconfiguráveis) e geração da imagem binária da configuração do FPGA (um arquivo com denominação **<nome do projeto>.bit** que contém todos os bits que irão configurar bits da RAM de controle do FPGA).
- Download
  - Processo de descarregar o resultado da síntese física (arquivo **.bit**) no FPGA.
  - Será utilizada uma ferramenta específica do fabricante da placa XS40/XST-1, a empresa Xess (http://www.xess.com), não integrada ao ambiente Foundation. Esta ferramenta (XSTools) está instalada nas máquinas do Laboratório. São dois programas: XSLOAD ou GXSLOAD (descarrega arquivo .BIT para o FPGA e opcionalmente programa .HEX para o microprocessador 8051 da XS40) e XSPORT ou GXSPORT (comunicação entre PC e placa durante execução).
- Placa XS40, placa autônoma de hardware, fabricada pela Xess Corporation, dotada de:
  - um FPGA XC40005XL;
  - um microprocessador 80C51 da Intel;
  - 32Kbytes de RAM e slot para PROM de download;
  - Interface Paralela para conexão ao computador hospedeiro;
  - Conector VGA para capturar sinal de vídeo;

– 1 Display de 7 segmentos;
– Vários “jumpers” de configuração, conectores adicionais;
– regulador de tensão programável (3,3V, 5V, etc).

- Placa XST-1

– Placa auxiliar à XS40, do mesmo fabricante, contendo:
– Possibilidade de acrescentar mais 64Kbytes de RAM;
– 3 chaves, duas das quais programáveis (entrada de dados e reset do FPGA);
– dip-switch de 8 chaves (entrada de dados);
– 8 leds e 2 displays de 7 segmentos;
– codificador/decodificador estéreo de sinal de som com 2 conectores (in/out);
– conector VGA adicional e conector para teclado PS2;
– área de prototipação para montagens adicionais;
– conector para placa XS40 ou XS95 e conector para todos os sinais da XS40.

9.3 A Fazer e Entregar

Tarefa 1: Recuperar os arquivos “leds.vhd” e “leds.ucf” da homepage da disciplina Abrir o Foundation, iniciando um novo projeto, tendo o cuidado de indicar projeto tipo HDL, pois faremos o projeto de hardware a partir de VHDL. Copie os arquivos “leds.vhd” e “leds.ucf” para o diretório do seu projeto. **Atenção**: se o projeto criado não é denominado **leds**, o arquivo ucf copiado deverá ter seu nome alterado para coincidir com o nome do projeto e ter a extensão .ucf. Após inseridos os arquivos, dê duplo clique no arquivo ucf , e confira no final do arquivo a relação entre os sinais e os pinos. Deve existir um arquivo com extensão ucf no diretório do projeto, que deve ser substituído pelo arquivo ucf fornecido

- Insira no projeto o fonte vhdl “leds.vhd”, via menu *Document* → *Add Document*. Selecione o “leds.vhd”. Execute um duplo clique neste arquivo. Estude-o, para compreender sua funcionalidade.

  RESPONDER: O que faz o hardware descrito por leds.vhd?

- Chamar a ferramenta de síntese lógica automática. Execute a síntese lógica (Botão ***Run***). A caixa da ferramenta de síntese lógica conterá um “!” ao final da síntese (há um *warning* apenas). Aparecerá o menu da Figura 46, o qual deve ser preenchido corretamente.

1. Defina a ***entity*** TESTE como aquela a ser gerada. **ATENÇÃO: é muito importante escolher a entity correta!!**

2. Defina a família de FPGAs, no caso é a família XC4000XL (3.3 Volts)

3. Defina o dispositivo da família – 4010 com 84 pinos externos e encapsulamento plástico (PC)

4. Defina a velocidade do dispositivo (deve ser xl – 3)

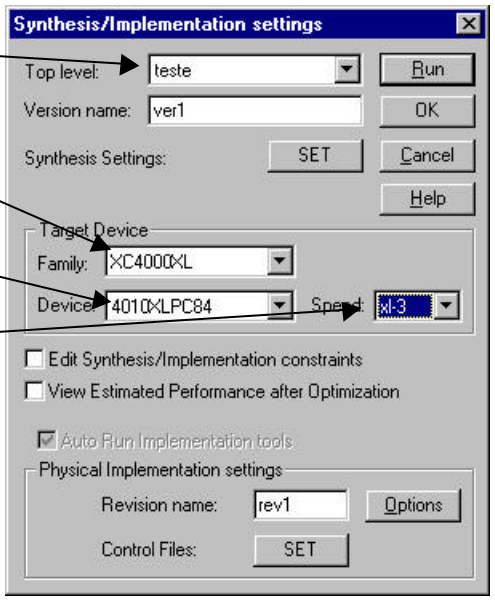


Figura 46 - Definição dos parâmetros para síntese física.

Tarefa 2: Chamar a síntese física automática. Não há nenhum parâmetro a ser inserido. Aparecerá a janela apresentada na Figura 47, com o progresso da síntese física. Observar as diversas

etapas: mapeamento, posicionamento e roteamento, análise de *timing* (cálculo automático do atraso no caminho mais longo no circuito – crítico) e geração do arquivo de download (extensão **bit**).

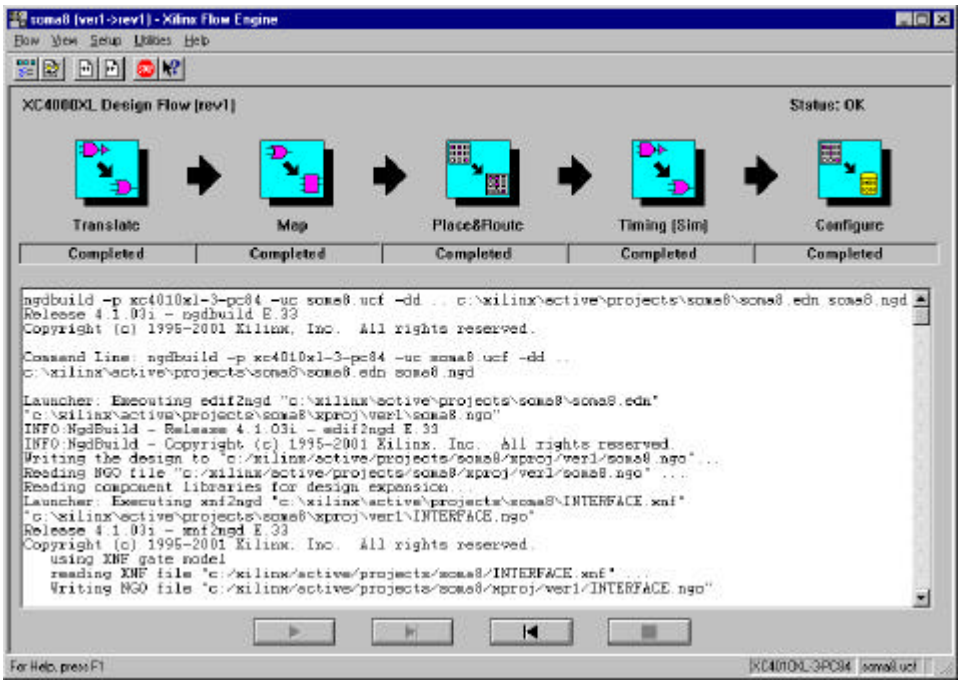


Figura 47 - Janela de Síntese Física.

Tarefa 3: Agora, analise os relatórios gerados pela tarefas de síntese, seja a síntese lógica (ou independente de tecnologia), seja a síntese física (também chamada de dependente de tecnologia). Existe no software Foundation uma orelha denominada Reports, na mesma sub-janela da janela de fluxo de projeto. Lá existem vários relatórios (ou diretórios de relatórios). **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.

Tarefa 4: Observar, por exemplo, que durante a análise de caminho crítico, é indicada uma freqüência máxima de funcionamento. **RESPONDER**: Qual foi esta freqüência no seu caso? Procure o relatório onde consta esta informação. Descubra e diga também nos relatórios qual é o fio de maior atraso (delay). Qual a relação deste fio com a freqüência máxima de operação do circuito? É bom que o valor da freqüência máxima seja alto ou baixo? Porquê?

Tarefa 5: Antes de realizar o *download*, observar a arquitetura interna do FPGA. Ir no menu *Tools* → *Implementation* → *FPGA Editor* do Gerenciador de Projetos Foundation. Se aparecer uma janela de diálogo, escolha o arquivo que possui como prefixo o nome do seu projeto. A aparência do circuito mostrado deve ser similar à Figura 48.

RESPONDER: Na sua opinião, qual o impacto do número de CLBs na implementação física? Por exemplo, existe limite para o tamanho de circuito implementável?

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?

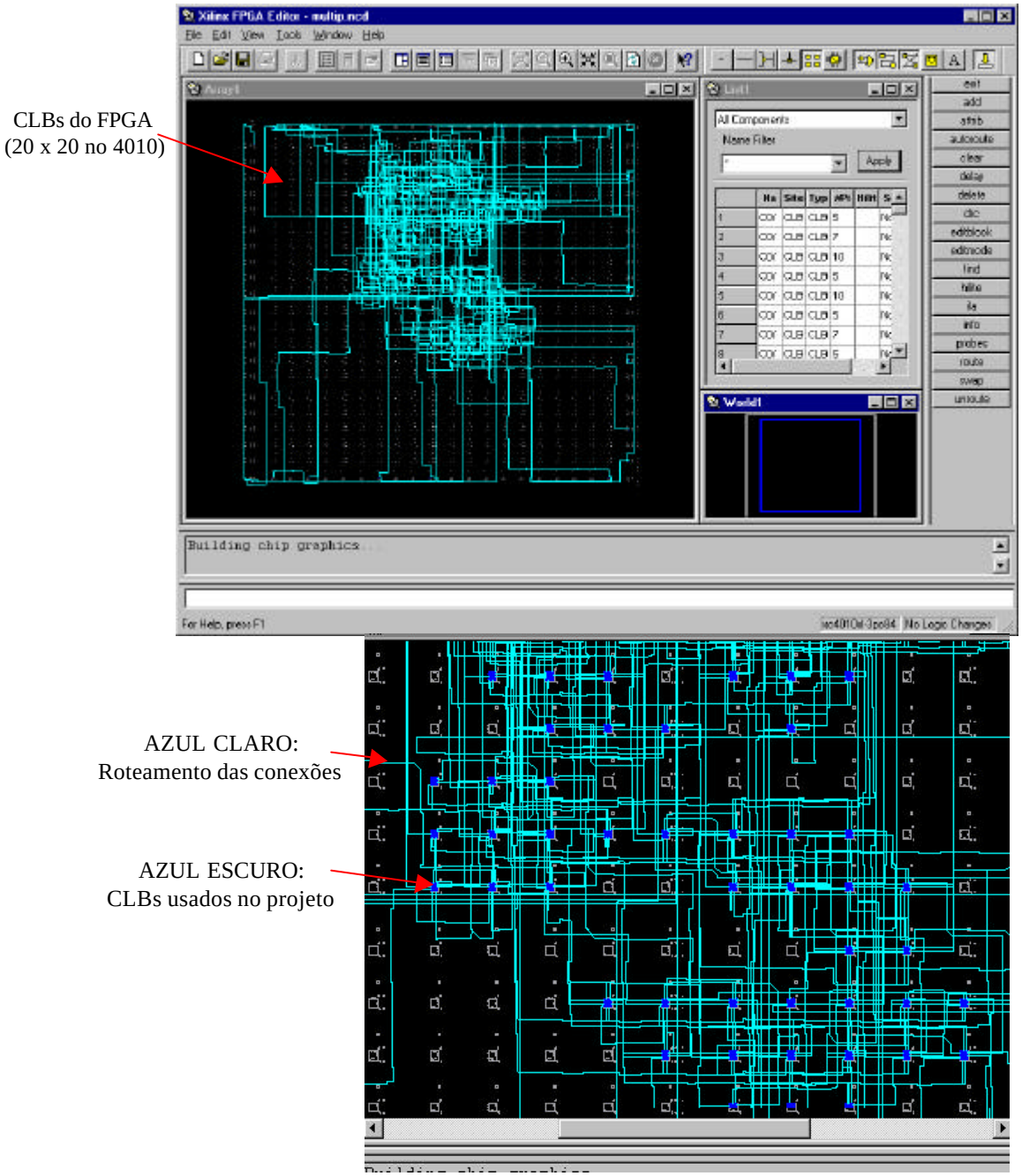


Figura 48 - Representação da planta baixa do circuito leds.pdf implementado internamento no FPGA.

Tarefa 6: Descarregue o arquivo binário com seu projeto implementado na plataforma de prototipação. ***Pronto! Estamos com o algoritmo descrito em VHDL implementado em hardware!***

Tarefa 7: Altere o hardware implementado para fazer o seguinte: a contagem mostrada nos mostradores de sete segmentos evolui com uma determinada freqüência. Pede-se:

- Calcule na teoria (a partir das informações no código VHDL) e na prática, qual é esta freqüência de contagem (provavelmente diferente, devido à imprecisão do oscilador interno do FPGA. Não é necessário calcular exatamente na versão prática, apenas de forma aproximada, usando, por exemplo um relógio com contagem de segundos;
- Altere a freqüência de contagem, de forma que esta passe a ser o mais próxima possível de 1Hz;
- Inverta a direção aparente de movimento dos leds.

Tarefa 8: Mostrar o projeto funcionando ao professor.

10 Implementação de Sistemas Digitais com VHDL - Multiplicação por somas sucessivas

Prática: Implementação estrutural de multiplicação por somas de produtos

Recursos: Ambiente de Desenvolvimento Active-HDL da Aldec, Inc, CAD Foundation e Ferramentas XSTools e Plataforma XS40/XST-1 da Xess, Inc.

10.1 Introdução e Objetivos

Os objetivos deste laboratório são:

- Implementar um multiplicador por somas sucessivas em VHDL.
- Implementar uma máquina de estados simples para o controle do multiplicador.
- Realizar o test_bench para a simulação da máquina de controle do multiplicador.
- Realizar o download do multiplicador no hardware para verificar o correto funcionamento do sistema.

10.2 Implementação do Bloco de Dados do Módulo Multiplicador

A Figura 49 ilustra a estrutura geral de um multiplicador de dois vetores de 8 bits representando números naturais (ou seja binários puros). Dois níveis da hierarquia do projeto aparecem, correspondendo às descrições da entidade e da arquitetura em VHDL. Este é o circuito a ser implementado. Mostra-se na Figura 49 apenas um possível diagrama de blocos do hardware para esta computação (importante: não estão sendo mostrados no diagrama os sinais de inicialização e relógio dos registradores, mas estes são obviamente necessários).

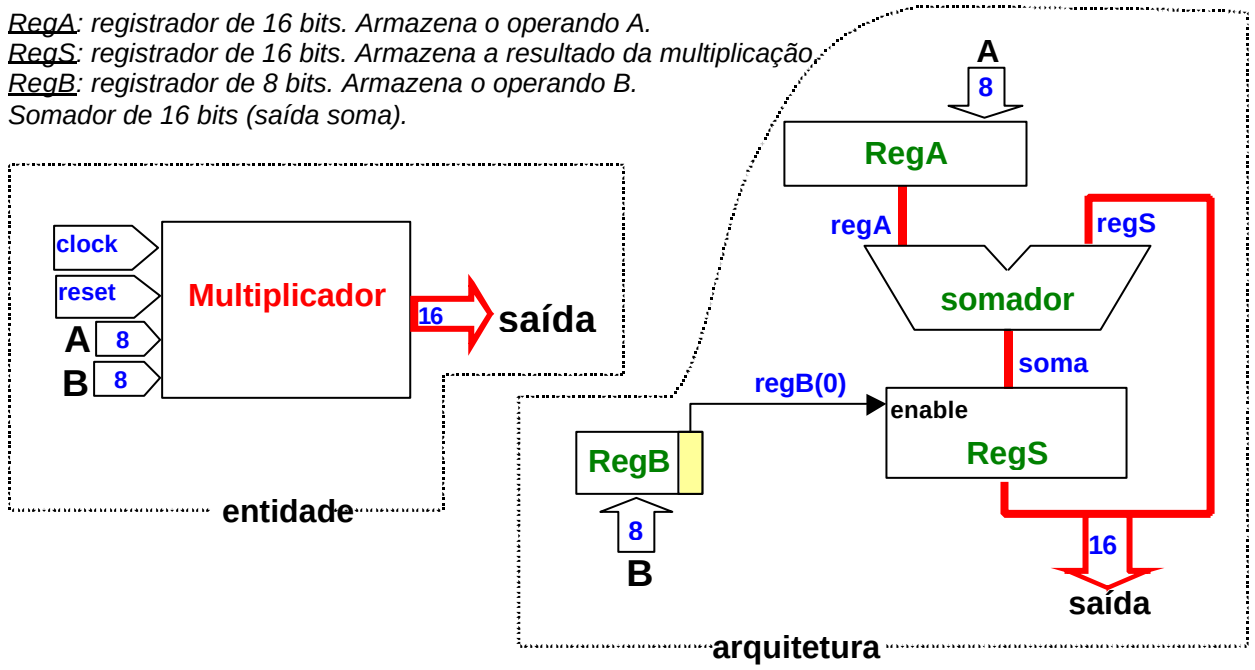


Figura 49 - Arquitetura de um multiplicador por somas sucessivas.

Esta versão é uma das mais simples formas de implementar um multiplicador em hardware. Para

outras versões (mais eficientes) recomenda-se a leitura Seção 4.6 do livro “Organização e Projeto de Computadores - a interface hardware/software”, de D. A. Patterson e J. L. Hennessy, Segunda Edição, a partir da página 144.

O funcionamento do multiplicador por somas sucessivas é o seguinte. Quando o bit menos significativo do operando B for 1 (denominado regB(0) na Figura 49) soma-se o resultado acumulado até o momento ao operando A, deslocado quantas vezes for necessário. A cada ciclo de relógio, ambos operandos são deslocados, porém em direções *contrárias* (“A” para a esquerda e “B” para a direita). Exemplo para palavra de 4 bits, para a multiplicação 0110 (6) *x* 1011 (11):

Ciclo		RegA	RegB	RegS	Soma
0		0000 0110	101**1**	0000 0000	0000 0110
1	↑			0000 0110	
	↓	0000 1100	010**1**		0001 0010
2	↑			0001 0010	
	↓	0001 1000	001**0**		0010 1010
3	↑			*mantém*	
	↓	0011 0000	000**1**		0100 0010
4	↑			0100 0010	
	↓	0110 0000	0000		

Observação: os registradores RegA e RegS tem o dobro de bits, no caso 8.

- Inicialmente o reset é aplicado, o que deve ter como efeito as seguintes ações (seu código VHDL deve garantir isto):
  - inicialização do registrador A: regA <= "00000000" & A
    (A é de 8 bits , mas regA é de 16 bits, deve-se usar concatenação em VHDL);
  - inicialização do registrador B: regB <= B;
  - reset do regS: regS <= (others=>'0');
- A cada ciclo de relógio, durante exatamente 8 ciclos:
  - *Na **subida** do relógio* armazena-se o resultado da soma se regB(0) for igual a ‘1’;
  - *Na **descida** do relógio* desloca-se os registradores regA e regB

```
dica: para um dos registradores: regA <= regA(14 downto 0) & ‘0’;
```

- Como está sendo feito o procedimento, são necessários 8 ciclos de relógio após o sinal de reset para obter o resultado da multiplicação de 2 números de 8 bits.

Faça um teste de mesa e verifique se o algoritmo funciona corretamente.

10.3 Implementação do Bloco de Controle do Módulo Multiplicador

Devido às limitações de recursos de entrada e saída na plataforma XS40/XST-1, deve-se implementar um bloco de controle do multiplicador como um máquina de 4 estados, ativada por uma tecla de pressão, onde os estados possuem as seguintes funções:

- Estado 1: armazena o operando A, a partir do dado de entrada contido nas chaves dip-switch;
- Estado 2: armazena o operador B, a partir do dado de entrada contido nas chaves dip-switch;
- Estado 3: inicializa o multiplicador;
- Estado 4: realiza a multiplicação (resultado pronto após 8 ciclos de relógio neste estado).

O código abaixo ilustra uma possível implementação do dito bloco de controle em VHDL:

```
-----------------------------------------------------------------
-- CONTROLE DO MULTIPLICADOR
-----------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use work.mult.all;        -- usar o nome do package definido pelo grupo

entity ctrl_mul is        -- NÃO ALTERAR OS NOMES DOS PINOS
         port( data : in reg8;
               key, ck, start : in std_logic;
               saida : out reg16);
end;

architecture a1 of ctrl_mul is

 component multiplicador is
         port( completar a descrição dos pinos );
 end component;

 type State_type is (S1, S2, S3, S4);
 signal EA: State_type;

  -- declare os sinais necessários

 begin

 -- maquina de estados para controlar o multiplicador
 process (reset, key)
 begin
    if start ='0'then
       EA <= S1;
       mulreset <= '0';
    elsif key'event and key='0' then
      case EA is
           when S1 => EA <= S2; A<=data;
           when S2 => EA <= S3; B<=data; mulreset <= '1';
           when S3 => EA <= S4; mulreset <= '0';
           when S4 => EA <= S4;
     end case;
   end if;
 end process;

 -- instanciacao do multiplicador
 X1: multiplicador port map( completar a descrição );

end a1;
```

O circuito de controle pode ser visto como composto de 4 partes:

1. Declaração da entidade:

– 3 entradas tipo std_logic: *clock*, *reset* e *key*;
– 1 entrada de 8 bits: *data*, que será utilizada pelos operandos “A” e “B”;
– *saída*, de 16 bits, que conterá o resultado da multiplicação.

2. Declaração do multiplicador (*component*); definição do tipo State_type, que será utilizado pela máquina de estados; e declaração dos sinais.
3. Processo responsável pela implementação do algoritmo de controle, representado pela máquina de estados.
4. Instanciamento do multiplicador (*port map*).

10.4 Simulação do Módulo Multiplicador

O circuito para testar o multiplicador deve controlar 4 sinais: *ck*, *start*, *key* (chave, ativa na borda de descida) e *data* (dados – 8 bits). Há apenas uma saída, o valor da multiplicação através do sinal *saida.*

Uma possível implementação do *test_bench* está ilustrada abaixo:

```
library IEEE;
use IEEE.Std_Logic_1164.all;
use work.mult.all;

entity tb is
end;

architecture rtl of tb is
 component ctrl_mul is
         port(completar a descrição dos pinos);
 end component;
 signal data : reg8;
 signal saida : reg16;
 signal ck, start, key: std_logic ;
begin
 m1 : ctrl_mul port map(completar a descrição dos pinos);
--- fazer o processo do clock
 start <='0', '1' after 5 ns;
 key <= '1', '0' after 20ns, '1' after 25 ns, '0' after 40ns, '1' after 45 ns,
         '0' after 50ns , '1' after 55 ns;
 data <= x"F3", x"57" after 40ns, x"67" after 450ns, x"9B" after 460ns;
end rtl;
```

A simulação deste test_bench está ilustrada no diagrama de tempos da Figura 50.

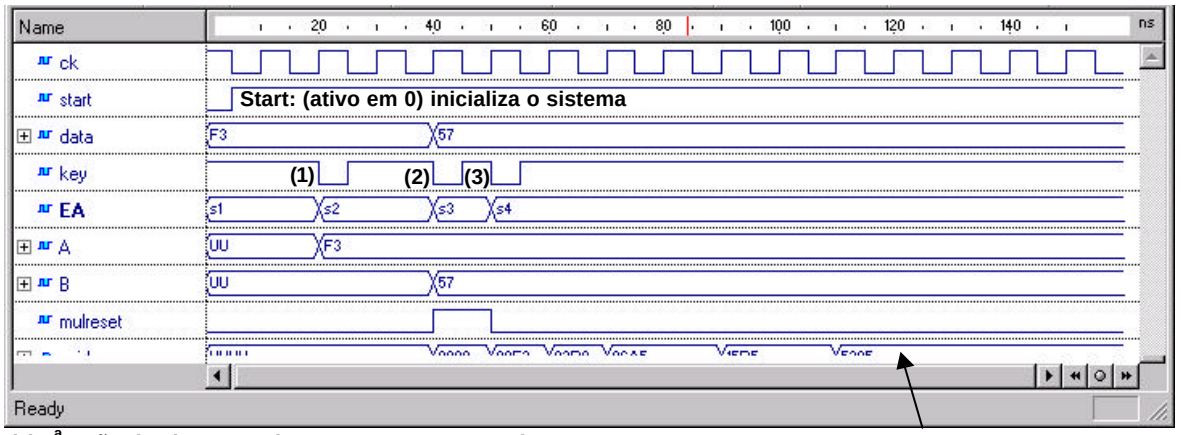


Figura 50 - Diagrama de tempos da simulação do multiplicador por somas sucessivas.

10.5 Síntese , Implementação e Teste do Módulo Multiplicador

Após obter uma simulação correta (ou seja, após validar funcionalmente o multiplicador), feche o Active-HDL e abra a ferramenta Foundation, escolhendo a opção novo projeto (ou File → new project), tendo o cuidado de especificar que o projeto a criar seja do tipo VHDL (botão HDL).

Passos a realizar para obter a versão executável em hardware:

1. Obter o arquivo multip.ucf na homepage da disciplina e substituir o seu arquivo **.ucf** original (na raiz do projeto), por este, com a definição correta dos pinos. **ATENÇÃO**: Não se esqueça de alterar o nome do arquivo, caso o seu projeto não se chame **multip**.
2. Obter o arquivo top.vhd da homepage da disciplina e o inseri-lo no diretório raiz do projeto.

3. Copiar o arquivo criado na **Parte II** (contendo package, blocos de dados e de controle do multiplicador), no diretório raiz do projeto. Para que esta descrição seja sintetizável, descartar o arquivo de test_bench.
4. Inserir os dois arquivos fontes VHDL no projeto usando a opção de menu: Document → Add do gerenciador de projeto do Foundation.

Resumindo: deve-se acrescentar 2 arquivos VHDL e um UCF na raiz do projeto.

Chamar a ferramenta de síntese lógica automática, tomando o cuidado da definição do módulo "*top level*", que no presente exemplo denomina-se "top". Execute a síntese lógica (*Run*). A caixa da ferramenta de síntese lógica conterá um "!" ao final da síntese, pois existirão muitos *warnings*, relacionados ao set/reset global. Após, chamar a síntese física automática. Não há nenhum parâmetro a ser inserido.

Agora, analise os relatórios gerados pela tarefas de síntese, seja a síntese lógica (ou independente de tecnologia), seja a síntese física (também chamada de dependente de tecnologia). Existe no software Foundation uma orelha denominada Reports, na mesma sub-janela da janela de fluxo de projeto. Lá existem vários relatórios (ou diretórios de relatórios).

Observar, em "*implementation report files*" o relatório "*post layout timing report*", que durante a análise de caminho crítico, é indicada uma freqüência máxima de operação do circuito, em MHz.

Certifique-se que a plataforma XS40/XST-1 de sua bancada esteja devidamente conectada ao computador via cabo paralelo, e que a plataforma está eletricamente alimentada. A seguir, abra o programa "gxsload", e arraste o arquivo "multip.bit" (que se encontra na raiz do projeto) para a janela do gxsload. Esta ação realiza o download do circuito na placa.

Como utilizar o multiplicador:

1. Pressione reset (tecla do meio), até que todos os leds acendam.
2. Selecione o operando "A", através das dip-switch (um número binário de 8 bits).
3. Pressione spare (tecla da esquerda). Apenas os 4 leds da direita permanecem acesos.
4. Selecione o operando "B", através das dip-switch (outro número binário de 8 bits).
5. Pressione spare. Todos os leds apagam, e o display exibe 00. Significa multiplicador inicializado.
6. Pressione spare pela terceira vez. O resultado da multiplicação deve ser então exibido. Os 8 bits mais significativos são exibidos nos leds e os 8 bits menos significativos são exibidos nos displays 7 segmentos, como dois dígitos em hexadecimal.

10.6 A Fazer e a Entregar

Tarefa 1: Implemente o multiplicador em VHDL, utilizando como modelo o circuito acumulador.

- **Dicas:**

– Implemente um package que contenha apenas a declaração de barramentos assim:

```
library IEEE;
use IEEE.Std_Logic_1164.all;
package mult is
     subtype reg16 is std_logic_vector(15 downto 0);
     subtype reg8 is std_logic_vector(7 downto 0);
end mult;
```

– Implemente 3 processos, um relativo a cada registrador, tendo como controle os sinais de reset e clock. Atenção: 2 registradores são sensíveis à borda de subida do clock e 1 é sensível à borda de descida do clock.

Tarefa 2: O que ocorre se todos os registradores operarem na **mesma** borda de clock? Funciona ou não? Justifique.

Tarefa 3: Justifique o tamanho do multiplicador. Porquê são necessários 16 bits de saída, se os operandos A e B são de 8 bits?

Tarefa 4: Faça um diagrama de blocos mostrando a conexão entre os circuitos *multiplicador* / *control_mul* / *top*.

Tarefa 5: Desenhe a máquina de estados do controlador do multiplicador. Diga o que ocorre em cada estado.

Tarefa 6: Modifique o test_bench para realizar 3 multiplicações, ao invés de apenas 1. Apresente no relatório a simulação comentada. Atenção: deixar ao menos 8 ciclos de clock para a realização da multiplicação.

Tarefa 7: Qual a freqüência de operação do circuito obtido (ver o nome do relatório indicado acima)?

Tarefa 8: Confira no "*implementation report files*" o relatório "*map report*". Lá está relatado que foram utilizados X CLBs (Y% do FPGA). Qual o valor de X e Y no seu projeto?

Tarefa 9: Mostre o circuito funcionando para o professor.

11 Implementação de Sistemas Digitais com VHDL - Acesso à Memória Externa na Plataforma de Prototipação

Prática: Implementação de uma aplicação que faz acesso à memória externa da plataforma XS40/XST-1

Recursos: Ambiente de Desenvolvimento Active-HDL da Aldec, Inc, CAD Foundation e Ferramentas XSTools e Plataforma XS40/XST-1 da Xess, Inc.

11.1 Introdução e Objetivos

Este Laboratório tem por objetivo exercitar na prática os conceitos aprendidos pelo uso de recursos que empregam comunicação assíncrona como modo de trocar informações. Trata-se, neste caso, da comunicação entre o FPGA da plataforma de prototipação XS40/XST-1 e a memória estática (SRAM) disponível na mesma plataforma. Esta comunicação assíncrona é bastante simples, uma vez que funciona de forma unidirecional. Especificamente, a memória da XS40/XST-1 não tem como informar que conseguiu ler o dado colocado pelo FPGA no barramento no tempo que o FPGA o disponibilizou. Assim, a temporização de escrita, e sobretudo leitura da memória deve ser cuidadosamente planejada.

Ao final do laboratório, os alunos deverão ter compreendido uma forma particular de operação da comunicação assíncrona, bem como a descrição e implementação de sistemas assíncronos em VHDL. Os objetivos específicos envolvem o desenvolvimento de máquina de estados para controle de operações e a utilização de memórias externas.

11.2 Uma palavra sobre a plataforma de prototipação XS40/XST-1 e o algoritmo de acesso a memória

A plataforma XS40/XST-1 possui um chip de memória RAM estática W24257AK-15, fabricado pela empresa Winbond Electronics Corp. Trata-se de uma memória estática de com organização 32.768 palavras (2^{15}) de 8 bits (32Kx8). O acesso se faz via um barramento bidirecional de 8 bits (1 palavra de cada vez), um barramento de endereços de 15 bits e três sinais de controle ativos em zero (CE-“chip enable”, WE-“write enable” e OE-“output enable”).

A plataforma de prototipação XS40/XST-1 usa multiplexação do barramento de endereços da memória com os fios para os mostradores de 7 segmentos, e do barramento de dados com os leds (Figura 51). Desta forma, não é possível fazer acesso simultâneo aos recursos de saída (mostradores de 7 segmentos e leds) e a memória.

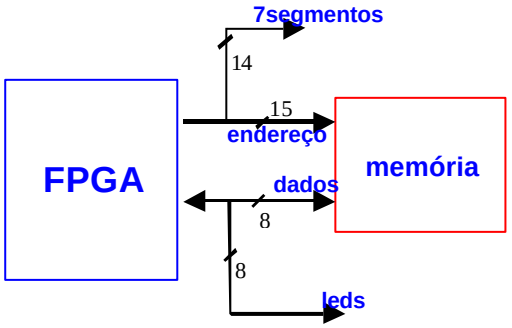


Figura 51 - Multiplexação dos barramentos na placa de prototipação.

Mostra-se aqui um algoritmo muito simples de acesso à memória, iniciando por escritas seqüenciais (endereço 0 recebe valor, endereço 1 recebe valor+1, etc.), procedendo-se à leitura destes valores após 16

escritas. Este algoritmo pode ser assim descrito:

1. Aguardar o pressionamento da tecla 'spare'. A seguir, iniciar processo de escrita:
   Registrador **mdr** recebe o valor das chaves dip-switch.
   Registrador **adram** recebe o valor 0.

2. Escrever, no endereço de memória indicado por **adram**, o conteúdo do registrador **mdr**.

3. Incrementar o valor dos registradores **mdr** e **adram**.
   Se o valor de **adram** for maior que 16, ir para o passo 4. Senão, voltar para o passo 2[4].

4. Esperar que seja pressionada a tecla 'spare'. A seguir, zerar o registrador **adram**.

5. Ler o conteúdo de memória apontado por **adram** armazenando-o no registrador **mdr**.

6. Incrementar o registrador **adram**. Exibir o valor nos displays de sete-segmentos. Se o valor de **adram** for maior que 16 passar para o passo 7. Senão, voltar ao passo 5[5].

7. Terminar a execução, reiniciando o processo de escrita apenas quando for pressionada a tecla 'reset'.

11.3 Análise do circuito

Cada passo do algoritmo apresentado anteriormente pode ser implementado como um estado da máquina de estados de controle (S1 a S7), descrita entre as linhas **161** e **197** no código VHDL fornecido.

A Figura 52 ilustra a estrutura do circuito proposto. O barramento *adram* (endereçamento da memória e mostradores de 7 segmentos) pode receber dados de 3 fontes distintas: registrador *mar*, quando desejo endereçar a memória; constante, quando desejo exibir uma constante nos displays; e o valor de *mdr* convertido para 7-segmentos, quando desejo exibir o valor lido da memória. A escolha de qual fonte será utilizada pelo *adram* é feita através de um multiplexador. O registrador *mar* tem por função armazenar e incrementar o endereço que vai ser utilizado para acesso à memória. O funcionamento dos barramentos *adram* e *mar* estão descritos entre as linhas **202** e **216** no código VHDL fornecido.

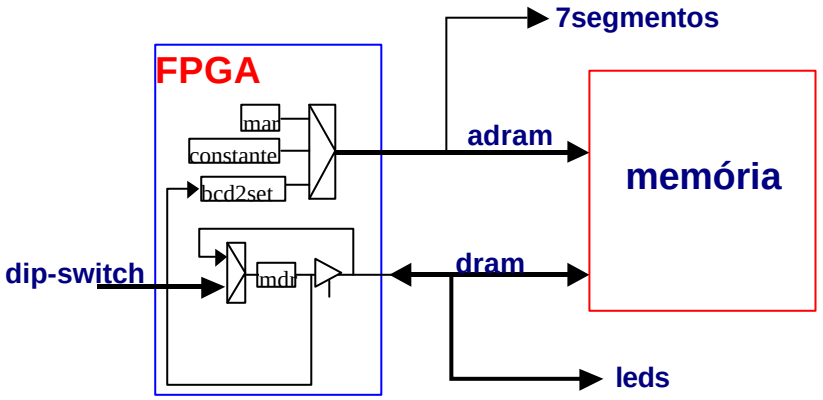


Figura 52 - Registradores internos do circuito proposto.

O barramento *dram* pode ser tanto entrada como saída, dependendo da ação efetuada sobre a memória. O registrador *mdr* tem por objetivos: (1) armazenar o valor contido nas chaves (*dip-switch*); (2) auto-incrementar seu valor; (3) receber dado proveniente da memória durante ação de **leitura** na memória. Quando se está **escrevendo** na memória *dram* recebe *mdr*, acionando-se uma chave *tri-state*. Caso não esteja sendo feita escrita na memória, esta chave fica aberta (alta impedância), não interferindo com a

[4] Os passos 2 e 3 escrevem seqüencialmente na memória, a partir do endereço de memória 0000H, o valor indicado pelas chaves, incrementado este endereço de 1 a cada novo endereço.
[5] Os passos 5 e 6 lêem seqüencialmente da memória, a partir do endereço de memória 0000H, exibindo cada valor lido nos mostradores de 7 segmentos.

operação da memória. Os barramentos *dram* e *mdr* têm sua funcionalidade descrita entre as linhas **221** e **234** no código VHDL fornecido.

Como dito antes, A memória tem 3 sinais de controle ativos em zero: *OE*, para indicar leitura; *WE*, para indicar escrita; e *CE* para habilitar a pastilha de memória a funcionar (para leitura e/ou escrita). A geração destes sinais está descrita entre as linhas **239** e **241** no código VHDL fornecido.

11.4 A Fazer e Entregar

Tarefa 1: Buscar o arquivo com a descrição VHDL do projeto inicial, denominado **access.vhd**, e o arquivo com a restrição da pinangem, denominado **access.ucf**. Iniciar um novo projeto no FOUNDATION. Neste laboratório não será utilizado o simulador Active-HDL. Faça a síntese e o download na placa e verifique se o funcionamento está de acordo com o algoritmo apresentado acima.

Tarefa 2: Analisar o código VHDL, desenhando um diagrama de blocos mostrando os componentes e suas interconexões. ENTREGAR NO RELATÓRIO O DESENHO.

Tarefa 3: Desenhar a máquina de estados e entregar no relatório.

Tarefa 4: Faça as modificações no projeto referentes à Tabela 6, sintetize e teste a nova versão. No início, ao invés de ler apenas uma informação das chaves, leia três

- Valor inicial a ser gravado na memória - *v0*;
- Passo entre dois elementos consecutivos - *step*.
- Número de elementos a gravar - *nb*;

Tabela 6 – Exemplo de conteúdo de memória.

	Conteúdo da Memória	
Endereço	**Versão original**	**Versão modificada**
0	**key**	***v0***
1	**key+1**	***v0+step***
2	**key+2**	***v0+2*step***
3	**key+3**	***v0+3*step***
4	**key+4**	***v0+4*step***
5	**key+5**	***v0+5*step***
6	**key+6**	***v0+6*step***
…	**…**	
15	**key+15**	

nb

Dica para a implementação: acrescentar apenas 2 estados adicionais na máquina de estados.

Tarefa 5: Mostrar ao professor o projeto original e o projeto alterado, ambos funcionando na placa de prototipação.

12 Implementação de Sistemas Digitais com VHDL - Comunicação Assíncrona entre Dispositivos

Prática: Implementação de um projeto VHDL simulável que realiza a comunicação assíncrona entre dois módulos de hardware

Recursos: Ambiente de Desenvolvimento Active-HDL da Aldec, Inc, CAD Foundation

12.1 Introdução e Objetivos

Este Laboratório tem por objetivo exercitar o processo de comunicação assíncrona entre dispositivos. O princípio de comunicação assíncrona é importantíssimo sempre que não se pode ter um relógio global que comanda todas as ações de um sistema computacional. Isto ocorre, por exemplo, quando os dispositivos que compõe o sistema possuem velocidade de operação muito diferentes, como ocorre entre CPU e disco e entre disco e impressora.

Ao final do laboratório, os alunos deverão ter compreendido o princípio básico de operação da comunicação assíncrona, bem como a descrição de sistemas assíncronos em VHDL. Para realizar o laboratório o aluno irá utilizar a ferramenta Active-HDL. Todos os resultados serão avaliados apenas através de simulação.

12.2 Comunicação Assíncrona

Na comunicação assíncrona, os dispositivos comunicantes operam sem a existência de um sinal comum de sincronização dos eventos em suas respectivas interfaces. Ou seja, não existe um sinal que opere como relógio global do sistema. Como os dispositivos neste caso estão sendo executados de forma completamente independente um do outro, não é possível que um dispositivos saiba o estado do outro, para eventualmente trocar informações com este no momento correto. Desta forma devem ser inseridos mecanismos que permitam a comunicação entre os dispositivos, sem que haja falha na informação que será trocada.

Em geral, os mecanismos de comunicação assíncrona são compostos por duas classes de sinais de controle, os que qualificam a informação (**sinais qualificadores**. Exemplo: leitura ou escrita) e sinais que sincronizam a troca de informação (**sinais sincronizadores**. Exemplo: inicie ou pare a comunicação, reconhecimento de dado transmitido, etc.).

Os sistemas de comunicação assíncrona, muitas vezes operam parcialmente com sinais qualificadores e sincronizadores e parcialmente com temporização. A chamada **operação temporizada** se caracteriza por um dispositivo transmissor que disponibiliza uma informação em um determinado sinal e aguarda um instante de tempo pré-determinado antes de prosseguir nas suas tarefas. Independentemente do dispositivo receptor ter reconhecido a informação, o dispositivo transmissor estará liberado para processar novas informações. Na prática, o tempo de espera pré-determinado deve ser o especificado pelo dispositivo receptor. Exemplo típico de operação temporizada ocorre quando um microprocessador lento conecta-se a uma memória rápida.

A quantidade de sinais qualificadores em uma determinada comunicação está intimamente ligada à confiabilidade e ao desempenho do sistema. Em outras palavras, quanto maior o número de qualificadores, maior o número de informações que permitem dar confiabilidade à informação trocada. Em contrapartida, maior será o tempo gasto para garantir que uma informação foi transmitida com sucesso.

A Figura 53 ilustra dois dispositivos (CPU e periférico) que se comunicam assincronamente. O controle da comunicação é efetuado através de quatro sinais (receive, acpt, send e ack) que têm a função de qualificadores e temporizadores da informação. A Figura 53 mostra, também, um barramento para a troca da informação propriamente dita. As Figura 54 e Figura 55 apresentam a troca de informação entre a CPU e o periférico, com um diagrama de tempos ilustrativo.

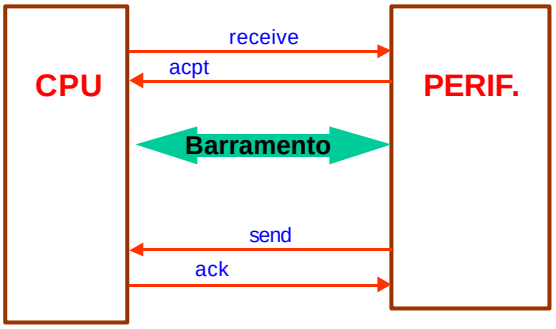


Figura 53 - Comunicação assíncrona.

O método de comunicação assíncrona mostrado nas Figura 54 e Figura 55 caracteriza-se por uma comunicação em quatro partes:

1- aviso de dado válido (transmissor, borda de subida do sinal **receive**);

2- aviso de dado recebido (receptor, borda de subida do sinal acpt);

3- aviso de fim de dado válido (transmissor, borda de descida do sinal **receive**);

4- aviso de fim da comunicação (receptor, borda de descida do sinal acpt).

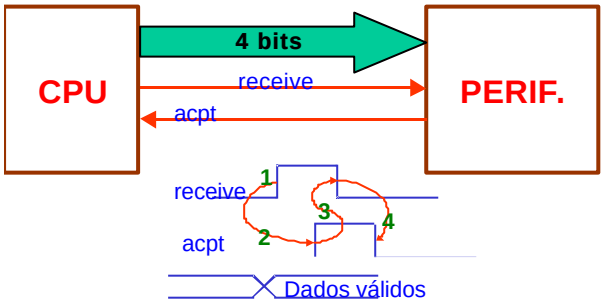


Figura 54 - Envio de dados CPU → periférico.

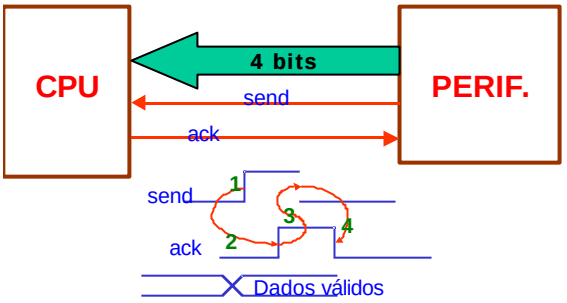


Figura 55 - Envio de dados periférico → CPU.

12.3 Especificação do Projeto

Neste Laboratório, deve ser descrito em VHDL um sistema de comunicação conforme ilustrado na Figura 56, que controla um fluxo de dados unidirecional, denominado **Sistema B**. Este sistema pode receber um vetor de 12 bits (*InVector*) do **Sistema A**, e transmiti-lo para o **Sistema C** sempre que a condição descrita abaixo for satisfeita. Caso a condição não seja satisfeita, o sistema deve filtrar a informação fornecida (ou seja, descartá-la sem transmitir ao **Sistema C**) e requisitar um novo vetor.

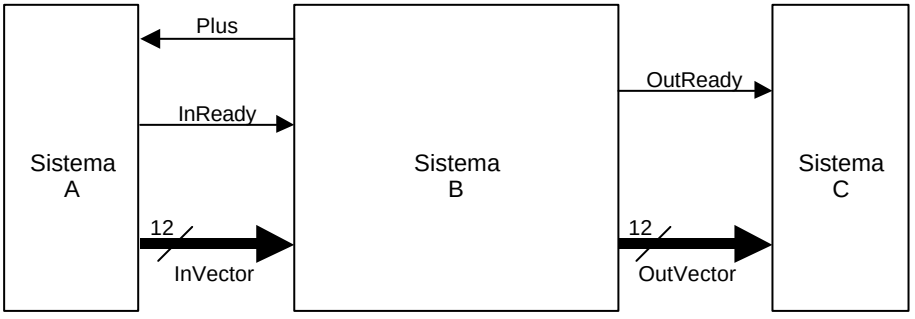


Figura 56 - Diagrama de blocos de um sistemas de comunicação entre dois sistemas.

Sempre que o **Sistema A** detectar o sinal *Plus* em 1 e tiver um vetor para transmitir para o **sistema C**, ele

deve colocar o sinal *InReady* em 1 e permanecer com este valor enquanto o **Sistema B** não colocar o sinal de *Plus* em 0.

Sempre que o vetor de entrada deva ser repassado para o **Sistema C**, o seu conteúdo deve ser colocado no vetor de saída *OutVector* (vetor de 12 bits) e o sinal *OutReady* (1 bit) deve permanecer em 1 **por exatamente dois ciclos do relógio interno do sistema B** (supõe-se que isto permita que o **sistema C** tenha tempo suficiente para adquirir o conteúdo do *OutVector*).

Abaixo está descrita a condição para filtrar a seqüência de bits:

InVector[0, 3] and *InVector*[4, 7] and *InVector*[8, 11] = "0000"

Exemplos:

1 - *InVector* = 0100 0011 0111, o vetor de 12 bits será filtrado, pois o resultado é "0000"

2 - *InVector* = 0011 0011 0011, o vetor de 12 bits será transmitido, pois o resultado é "0011"

Abaixo está exemplificada a entrada de 2 vetores (representados em hexadecimal, 437H e 333H). Como pode ser observado, o vetor 437 foi filtrado, já que satisfaz a equação.

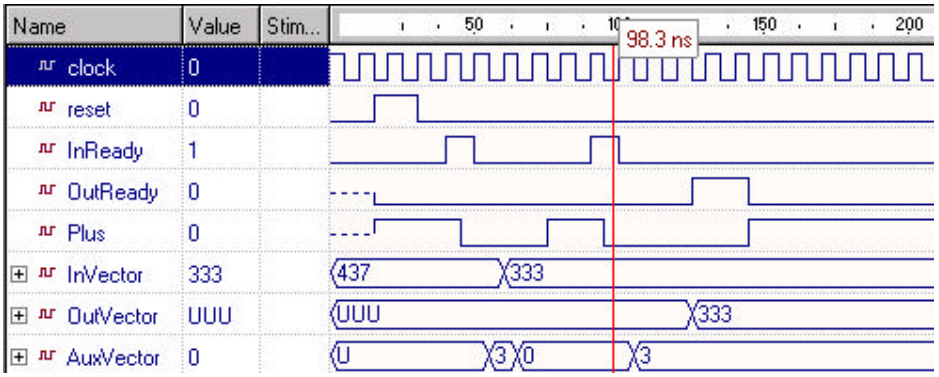


Figura 57 - Simulação do funcionamento do sistema B.

Na Figura 57 aparecem, além dos sinais apresentados na Figura 56, um sinal de **reset**, que indica o início de funcionamento dos sistemas e um registrador auxiliar **AuxVector** (de 4 bits), que faz parte do Sistema B e serve para realizar a avaliação do vetor de entrada.

12.4 A Fazer e a Entregar

Tarefa 1: Faça a descrição VHDL de todo o **Sistema B** e o test bench (o que equivale a uma descrição parcial da operação dos **Sistemas A** e **C**) que deve simular a entrada de pelo menos 10 valores para o vetor InVector, dos quais pelo menos 3 devem ser filtrados pelo Sistema B, e pelo menos a mesma quantidade deve ser transmitida ao **Sistema C**. Incluir no relatório final os fontes e a captura das janelas com as simulações comentadas.

Tarefa 2: **Responda**: quantos ciclos de relógio é necessário, na sua implementação, para o **Sistema B** realizar uma operação completa de filtragem e/ou transmissão?

Tarefa 3: **Responda**: A interface entre o **Sistema A** e o **Sistema B** caracteriza-se por ter uma operação síncrona ou assíncrona? E a interface entre o **Sistema B** e o **Sistema C**?

Tarefa 4: Fazer o diagrama de transição de estados do **Sistema B.**

13 VHDL: Simulação da arquitetura cleopatra

1. Abra um novo projeto no active VHDL denominado “cleo”, sem fontes.

   Create new desing ➔ *nome: cleo* ➔ *create an empty desiing* ➔ *concluir*

2. Copie da página da disciplina o arquivo o arquivo VHDL e o exemplo de teste de simulação.
3. Abra uma janela de simulação e clicar em *simulation* e *initialize simulation*. Selecionar como raiz do projeto a entidade “tb” (test bench).
4. Colocar nesta janela todos os sinais da clopatra (abaixo do tb) e os sinais AC, PC, IR, MDR (registradores da parte de controle) do *datapah.*
5. **Antes** de simular abra o arquivo “teste.txt” e observe a primeira linha: 00 48. Significa: vai executar um LDA indireto (3 ciclos para a busca e 7 para a execução → total de 10 ciclos de clock para esta instrução), tal com o apresentado na Figura 58.
   - LEMBRAR: a micro-instrução é ativa na borda de subida do clock (observar na simulação) e a mudança no valor dos registradores na borda de descida do clock.
   - SIMULAR. Nesta primeira iteração observar o sinal início, que demarca o início de uma instrução.

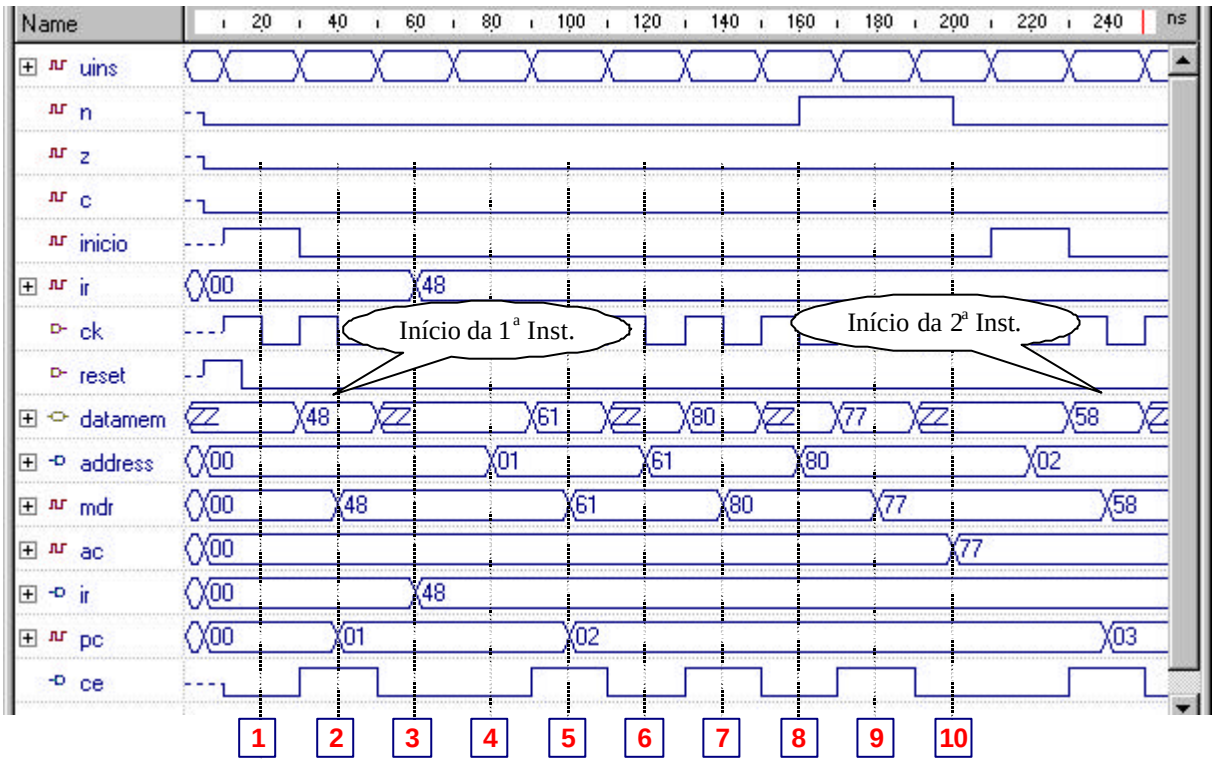


Figura 58 – Simulação da arquitetura Cleópatra.

```
1. MAR ← PC  2. MDR ← PMEM (MAR); PC ← PC +1;  3. IR ← MDR
4. MAR ← PC  5. MDR ← PMEM (MAR); PC ← PC +1;
6. MAR ← MDR 7. MDR ← PMEM (MAR)
8. MAR ← MDR 9. MDR ← PMEM (MAR)
10. AC ← MDR
```

13.1 A Fazer e a Entregar

Escreva um programa que encontre o somatório de um vetor. Especificação dos endereços:

- Endereço 20H: endereço do início do vetor
- Endereço 21H: número de elementos do vetor
- Endereço 22H: endereço onde deverá ser armazenado o somatório.

Sugestão: trabalhem com vetores pequenos, exemplo, 5 elementos.

Procedimento:

1. Escreva o programa em linguagem assembly para a Cleópatra.
2. Traduza para código objeto.
3. Escreva o código objeto no arquivo "program2.txt" (substitua o antigo programa). Não esqueça de colocar os dados no arquivo.
4. Simular. Exibir na simulação os registros internos do processador.
5. Durante a simulação verificar a memória, ou seja, verificar se realmente ocorre leituras escritas.
6. Ao final da simulação verificar se o conteúdo do endereço 22H contém realmente o valor do somatório do vetor.
7. Entregar: código assembly, código objeto, janelas da simulação indicando pontos relevantes da simulação.
8. Responda: quantos ciclos de clock o programa implementado gasta para executar este programa. Expresse também o tempo de execução na forma de: tempo = k1 + n*k2, onde k1 e k2 são constantes e n o número de elementos do vetor.

Anexo 1 – Circuitos Lógicos Programáveis – FPGA

FPGA – A sigla vem do inglês, e significa Field Programmable Gate Array

- **Definição** – Trata-se de um dispositivo eletrônico (*chip*) configurável pelo usuário (donde a palavra Field no nome, que significa No Campo, significando no campo de aplicação), através da descarga (*download*) da sua funcionalidade por algum meio (gravador de EPROM, cabo serial ou paralelo, leitura de ROM/PROM/EPROM na própria placa do dispositivo, etc.).
- Utilizamos FPGAs baseados em RAM da família Xilinx XC4000XL, cujos membros são compostos por várias partes configuráveis e reconfiguráveis, entre estes os blocos lógicos, as conexões entre estes e a interface com o mundo externo.

Componentes de um FPGA (Figura 59):

- **CLBs**: Configurable Logic Blocks –blocos lógicos configuráveis.
- **Interconexão configurável**: permite conectar CLBs entre si.
- **IOBs**: interface configurável com o mundo externo. Cada pino pode ser configurado como entrada, saída ou ambos.

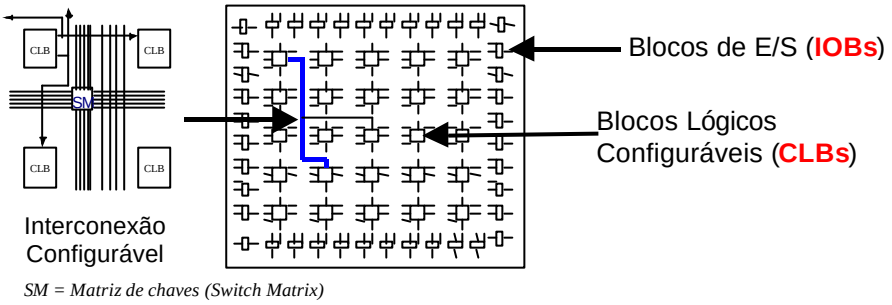


Figura 59 - Componentes de um FPGA.

A Figura 60 ilustra a estrutura interna de um CLB. Cada CLB contém 2 tabelas-verdade configuráveis (*Look-Up-Tables* ou *LUTs*) de 4 entradas, podendo implementar qualquer função Booleana de 4 variáveis (LUTs F e G) e uma com 3 entradas, podendo implementar qualquer função de 3 variáveis (LUT H). Unindo as LUTs, pode-se implementar uma única função de 5 variáveis. O CLB contém também 2 flip-flops e lógica de propagação rápida de vai-um para circuitos aritméticos.

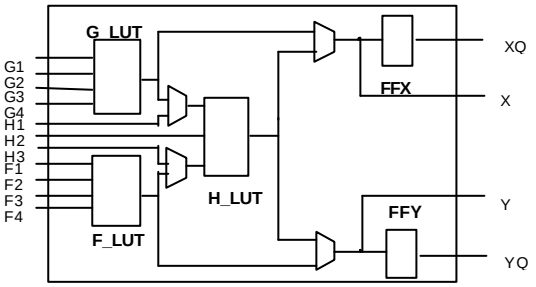


Figura 60 - Arquitetura de um Dispositivo Reconfigurável FPGA da família Xilinx XC4000XL

Como capacidade de implementação, cita-se o dispositivo XC4010XL da placa XS40. Este possui 400 CLBs. Segundo a Xilinx, este chip pode implementar circuitos com o equivalente a algo entre 3000 e 9000 portas lógicas. (Exemplo: o processador Cleópatra gasta 170 CLBs do XC4010XL).

A conexão entre o FPGA e o mundo externo é feita pela definição da posição dos pinos de entrada/saída.

- É responsabilidade do usuário definir a interface de seu circuito com o mundo externo, através da atribuição dos sinais de sua entidade mais externa (*entity*) aos pinos do FPGA.

- Consultar o esquemático da placa XS40 nos manuais da plataforma XS40/XST-1 para informação sobre as posições dos pinos específicos e suas conexões a periféricos das placas.
- O usuário entra com a informação do número dos pinos no arquivo com extensão ***ucf***, na raiz do *Foundation*.(Figura 61). Sempre adicionar este arquivo ao projeto em desenvolvimento.
- Exemplo da parte final do arquivo ***ucf***, onde são descritos os pinos de E/S:

```
…NET LED<3>  LOC = P38 ;
  NET LED<4>  LOC = P35 ;
  NET LED<5>  LOC = P81 ;
  NET LED<6>  LOC = P80 ; …..
```

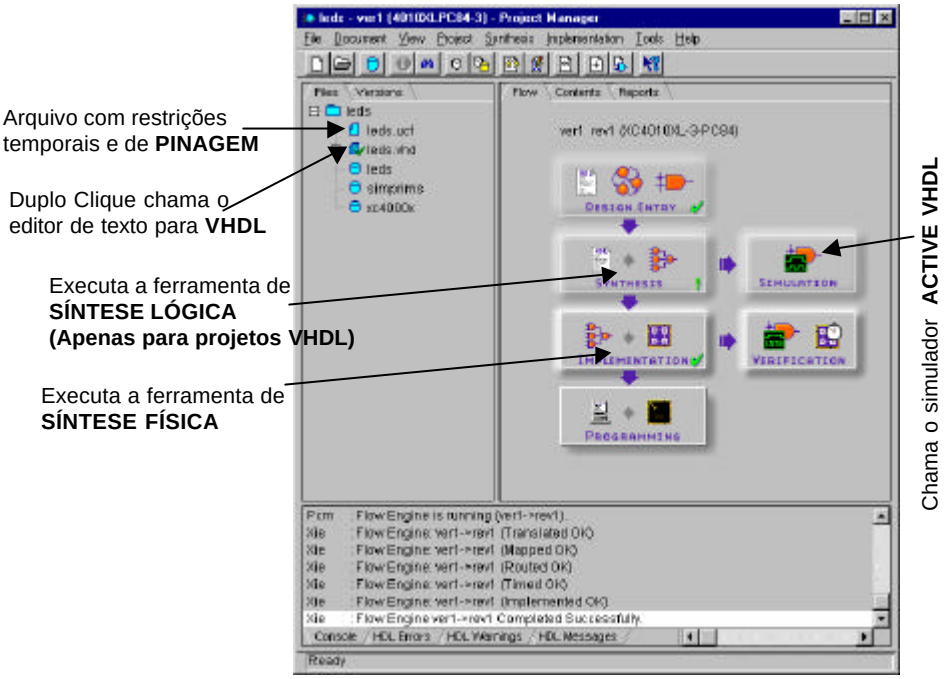


Figura 61 - Janela Principal do CAD Foundation, com configurações de projeto VHDL salientadas.

Anexo 2 – Definição Física do Circuito de Debounce

Fenômenos eletrônicos são rápidos. Fenômenos mecânicos são muito mais lentos. Considere-se uma tecla do tipo push-button, que possui uma mola impulsionando-a para o estado aberto, conforme a desenho da Figura 62. Quando não há forças além daquela da mola agindo, a chave está aberta e não passa corrente no resistor R1, fazendo com que a tensão na saída seja igual à tensão de alimentação do circuito, no desenho 5Volts. Isto se deve à famosa lei de Ohm V=RI, que relaciona tensão, corrente e resistência num circuito elétrico. R no caso de chave aberta é infinita (R=R1 + ∞ = ∞), pois a resistência de um circuito série de dois resistores é a soma das resistências individuais. Uma chave ideal funciona ora como uma resistência infinita (chave aberta) ora como uma resistência nula (chave fechada). Quando a chave está aberta não pode passar corrente por R1, logo não pode haver queda de tensão sobre esta, pois a lei de Ohm diz R1*0=0V. Assim, para garantir que o circuito respeite a lei das malhas (soma das tensões sobre todos os elementos de um circuito é 0), toda a tensão está aplicada sobre a chave, e a saída é uma tensão de 5 Volts em relação ao pólo negativo da fonte de alimentação. Quando uma força mecânica (gerada pelo dedo do usuário, por exemplo) atua sobre a chave, fechando-a, ela passa a agir como um fio sem resistência, e a saída está na tensão 0 Volts em relação ao pólo negativo da fonte de alimentação.

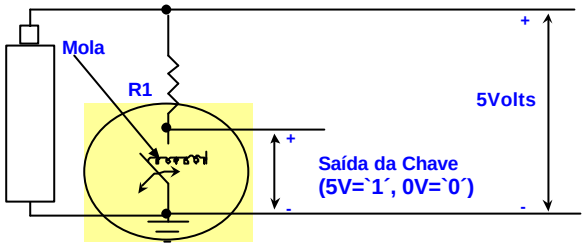


Figura 62 – Diagrama elétrico de uma chave mecânica.

Assim, quando uma tecla push-button é pressionada (o raciocínio é análogo para qualquer outro tipo de contato mecânico), há idealmente uma transição da tensão de 1 para 0, ou seja, uma borda descida. Contudo a realidade é mais complicada, pois o contato dos dois pólos da chave não é perfeito, sempre podendo haver um ou mais "repiques" do contato, provocando não uma, mas diversas transições 0-1, 1-0 até estabilizar-se a saída no valor de chave apertada (0). Medindo com equipamento suficientemente sensível (e.g. osciloscópio) a relação Tensão versus Tempo na saída da chave, chega-se a um gráfico similar ao da Figura 63, supondo que o contato inicial dos pólos da chave ocorreu no instante de tempo 100 microssegundos.

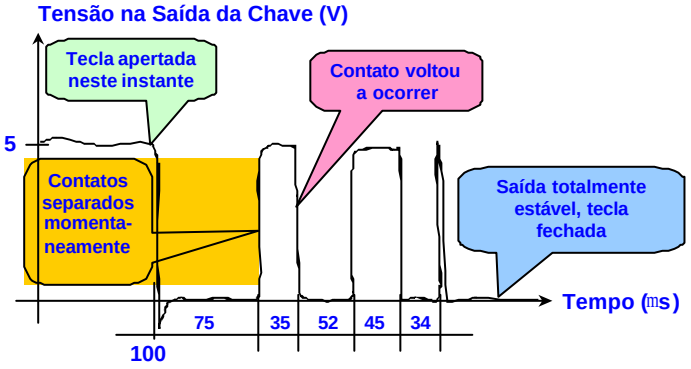


Figura 63 – Exemplo de Gráfico de Tensão versus Tempo típico de uma chave mecânica.

Note-se que ao invés de uma transição 0-1, foram obtidas 4 transições. Se este sinal é o relógio de um registrador sensível à borda de descida, seriam executadas 4 escritas neste último ao invés de 1 apenas, que é o que se pretende apertando a tecla apenas uma vez. Note-se também que o tempo ao longo do qual se distribuem as transições indesejadas (241 microssegundos, ou quase ¼ de segundo) é muito grande em relação ao tempo necessário para o sistema digital efetuar operações (na ordem de nanossegundos, ou seja, um bilionésimo de segundo, ou seja, 1/1.000.000.000 segundos). O circuito que recebe esta entrada "suja", elimina as transições espúrias e produz uma única transição a cada aperto de tecla se denomina **debounce** (em inglês, **bounce** significa picar ou repicar, ou seja debounce significa retirar os repiques).