UNIVERSIDADE FEDERAL DE PERNAMBUCO

ENGENHARIA ELETRÔNICA

DEPARTAMENTO DE ELETRÔNICA E SISTEMAS

CENTRO DE TECNOLOGIA E GEOCIÊNCIAS

DOCENTE RESPONSÁVEL: DR. MARCO AURÉLIO BENEDETTI RODRIGUES

DISCIPLINA: ES441

SEMESTRE 2024.1

TURMA 01A

Eletrônica Digital

Projeto 3

Alysson Lucas Pontes Cavalcante da Silva

Felipe Rafael Barros da Silva

Maria Victória Martins Neves

Recife, 2024

Sumário

1 Introdução

Neste trabalho, foi desenvolvido um projeto de um relógio digital com alarme utilizando apenas a linguagem de descrição de Hardware VHDL. Este projeto faz parte da disciplina de Eletrônica Digital, em que o objetivo principal é aplicar os conceitos estudados na construção de um sistema funcional que demonstre o entendimento prático dos componentes eletrônicos digitais.

O objetivo geral deste projeto é projetar e implementar um relógio digital que, além de exibir horas, minutos e segundos no Display de Cristal Líquido (LCD), seja capaz de programar e acionar três alarmes com avisos musicais.

Os objetivos específicos do projeto incluem a criação de um contador de horas, minutos e segundos; o desenvolvimento de um circuito de controle que permita a programação do relógio e do alarme através de chaves e botões; e a integração de todos esses elementos em um único sistema funcional que acione o alarme no horário programado.

A estrutura deste relatório está organizada da seguinte forma: Introdução, apresenta o projeto e os seus objetivos; Desenvolvimento, expõe os passos seguidos para o projeto e implementação do relógio digital, incluindo o diagrama de blocos e a explicação dos códigos desenvolvidos; Manual de Operação, descreve a forma de operar o circuito gravado no FPGA; Resultados, discute o desempenho do relógio e dos alarmes, bem como os desafios enfrentados durante o desenvolvimento; Conclusão, avaliação do cumprimento dos objetivos do projeto, pontuação das limitações, apontamento das dificuldades na relação teoria versus prática e destaque da contribuição das atividades para o aprendizado da equipe; Apêndices, com a inserção dos códigos autorais desenvolvidos para o projeto; Anexos, com a inserção dos códigos não autorais utilizados no projeto.

2 Desenvolvimento

A criação do projeto se dividiu em muitas etapas, que serão apresentadas em tópicos neste capítulo, a fim do melhor entendimento. Uma informação de extrema importância para a continuidade da leitura do relatório é que o FPGA utilizado possui as entradas e saídas invertidas, em outras palavras, suas entradas e saídas são ativadas no estado 0 e desativadas no estado 1. Essa configuração influenciou em todas as entradas de botões ou chaves utilizadas e também nas saídas das LEDs e displays, necessitando a inversão das entradas e saídas dentro dos códigos.

2.1 Linguagem utilizada

Para a implementação do relógio digital com configuração de alarmes, a linguagem utilizada foi o VHDL (VHSIC Hardware Description Language). O VHDL é uma linguagem de descrição de hardware padronizada pela IEEE (Institute of Electrical and Electronics Engineers) que permite modelar e simular o comportamento de circuitos digitais em níveis diversos de abstração, como o nível comportamental, estrutural e de transferência de dados.

Optou-se pelo VHDL devido à sua ampla utilização em projetos digitais que envolvem sistemas complexos, como o desenvolvimento de circuitos integrados e o uso de FPGA (Field Programmable Gate Arrays). A flexibilidade da linguagem permitiu a implementação dos componentes fundamentais do relógio, como contadores, flip-flops e módulos de controle, além de possibilitar a inclusão de funcionalidades extras, como a configuração de três alarmes distintos.

Outra vantagem de utilizar VHDL foi a facilidade de testbenching e simulação, que viabilizou a validação dos módulos de forma eficiente antes de gravar o código no FPGA. A natureza descritiva do VHDL facilita a transição de uma visão funcional de alto nível do circuito para a implementação detalhada do hardware, o que foi crucial no desenvolvimento e depuração do relógio digital.

Além disso, o VHDL oferece suporte a um projeto modular, permitindo que o relógio e seus alarmes fossem desenvolvidos em blocos individuais (como contadores de segundos, minutos e horas,

além dos módulos de alarme), promovendo a reutilização de código e a escalabilidade do sistema.

2.2 Conversores de clock

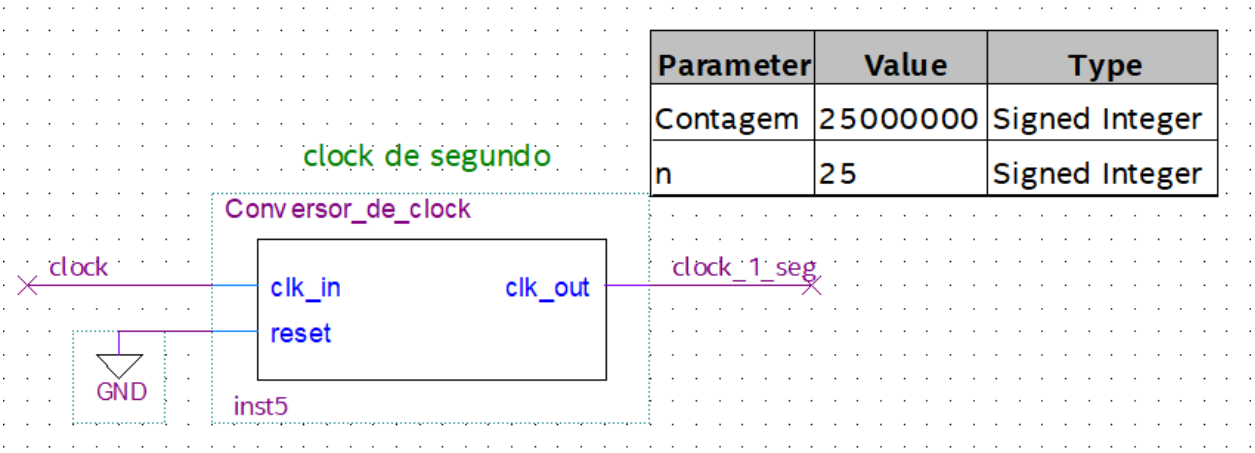
O conversor de clock é fundamental em projetos de eletrônica digital por várias razões. O conversor de clock é crucial para a contagem do tempo em um relógio digital, permitindo uma amostragem precisa e a atualização dos valores de tempo de acordo com a frequência desejada.

Além disso, no caso do debounce, o conversor reduz a frequência do clock principal para estabilizar sinais de entrada, minimizando os problemas de flutuação rápida e ruídos. Ele também é útil para ajustar a frequência do sinal sonoro, permitindo a geração de ritmos. Outro uso importante é no controle de displays, tanto para mantê-los acesos simultanemente, quanto para piscar os displays com a frequência desejada.

Em resumo, o conversor de clock permite a adaptação da frequência do sinal principal para atender às necessidades específicas do projeto, assegurando o funcionamento correto e eficiente do sistema.

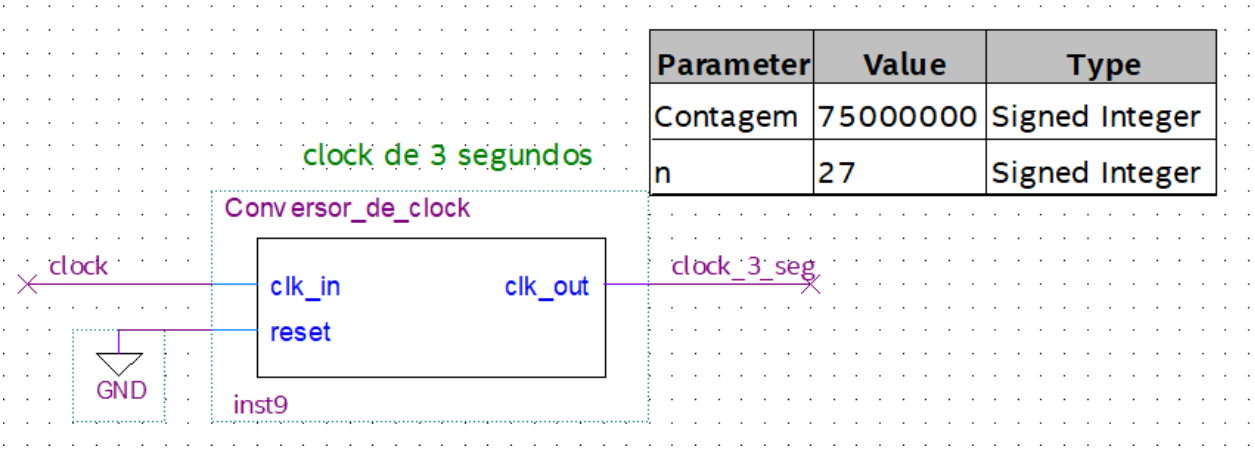
O conversor é configurado com dois parâmetros: ‘n‘, que define o número de bits do contador, e ‘Contagem‘, que especifica o valor limite da contagem. O código reduz a frequência do clock principal utilizando um contador binário de ‘n‘ bits. A cada borda de subida do clock de entrada, o contador é incrementado. Quando o valor do contador atinge o limite especificado por ‘Contagem‘, o clock de saída é alternado (invertido) e o contador é reiniciado. Esse processo gera um sinal de clock com frequência reduzida na saída.

São utilizados cinco conversores de clock no projeto. Os dois primeiros geram um clock de um segundo (Figura 2.1) e um clock de três segundos (Figura 2.2). Os outros três conversores controlam o Buzzer, atribuindo uma frequência sonora específica a cada alarme. Dessa forma, cada conversor é configurado com parâmetros distintos, ativando os alarmes 1, 2 e 3, respectivamente, conforme ilustrado nas Figuras 2.3 a 2.5. O código correspondente a esses conversores está disponível no Apêndice A.



Parameter	Value	Type
Contagem	25000000	Signed Integer
n	25	Signed Integer

Figura 2.1: Conversor de clock: contador de um segundo.



Parameter	Value	Type
Contagem	75000000	Signed Integer
n	27	Signed Integer

Figura 2.2: Conversor de clock: contador de três segundos.

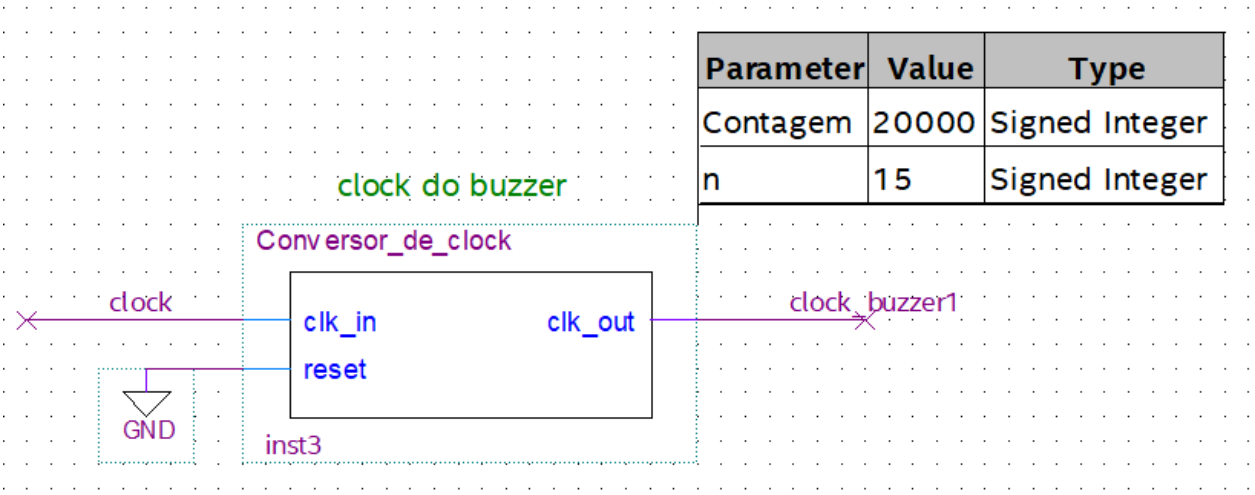


Figura 2.3: Conversor de clock: frequência para o primeiro alarme.

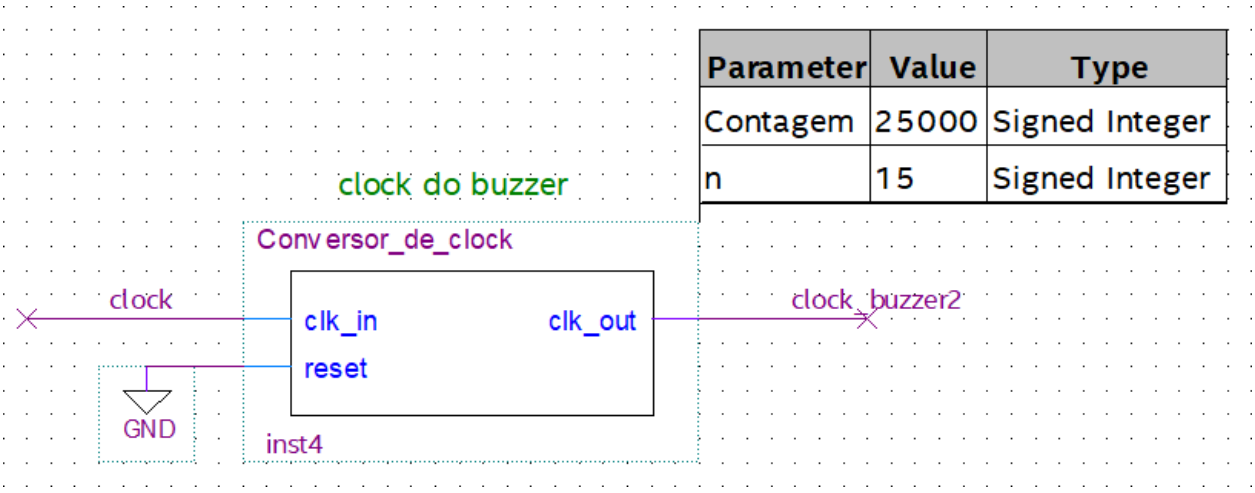


Figura 2.4: Conversor de clock: frequência para o segundo alarme.

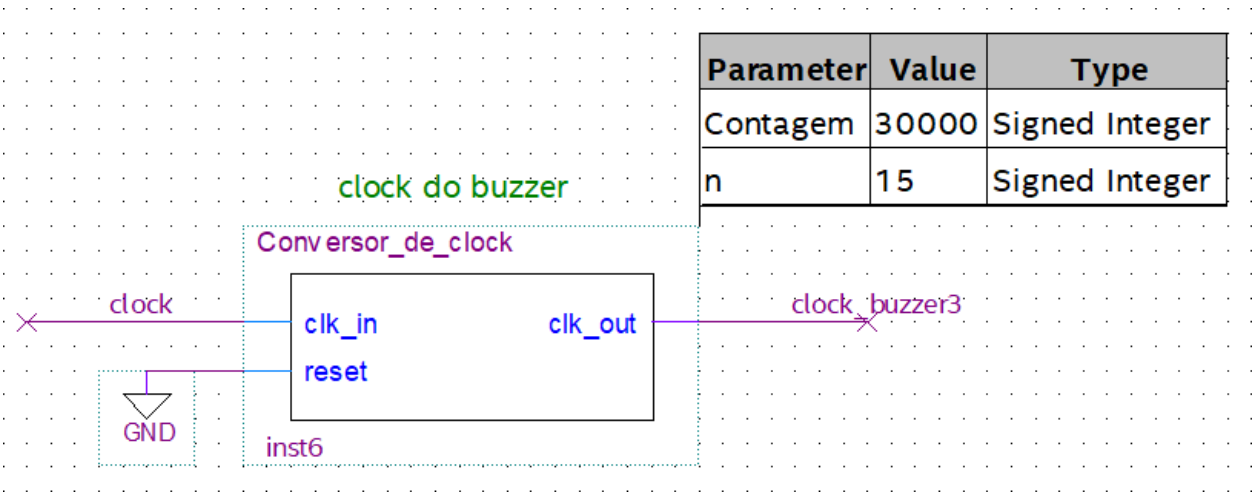


Figura 2.5: Conversor de clock: frequência para o terceiro alarme.

2.3 Contador de 4 bits

O código VHDL para o contador síncrono de quatro bits, descrito no Apêndice B, opera com base em dois sinais de controle: clock e reset. A saída é um vetor de quatro bits (módulo 16) que representa o valor atual da contagem, incrementando em 1 a cada ciclo de clock, utilizando aritmética binária.

O comportamento do contador é controlado pelo sinal de reset. Quando ativo, o reset reinicia a contagem; quando inativo, a contagem é incrementada a cada borda de subida do clock. Embora o contador conte de 0 a 15 (equivalente a ‘0000‘ a ‘1111‘ em binário), o reset permite ajustar o valor em que o contador reinicia, controlando o ciclo de contagem conforme o contexto. Isso é particularmente útil em aplicações de temporização, como horas, minutos e segundos, que exigem diferentes limites de contagem (por exemplo, 0 a 59 para segundos/minutos e 0 a 23 para horas).

No projeto do relógio, esse contador será instanciado para representar cada dígito, desde os segundos até as horas. O reset garantirá que a contagem de cada unidade de tempo seja reiniciada no valor correto, permitindo personalizar os ciclos conforme necessário para cada parte do relógio.

Dessa forma, o contador módulo 16 é a base da contagem do relógio, com múltiplas instâncias criadas para diferentes dígitos. O dígito menos significativo (LSD) é o primeiro contador, e o mais significativo (MSD), o último, com condições específicas para os resets de cada dígito, conforme o ciclo necessário para cada unidade de tempo.

Cada dígito exige condições particulares:

- Para os dígitos menos significativos dos **segundos** e **minutos**, os contadores são configurados como módulo 10.
- Os dígitos mais significativo dos **segundos** e **minutos** utilizam um contador módulo 6.
- Para o LSD das **horas**, o contador pode operar em módulo 10 ou em módulo 4, a depender do MSD das horas. Há também um reset configurado para o caso em que se tenta alterar o primeiro dígito das horas de 1 para 2, mas o segundo dígito já é maior que 3. Nesse cenário, o reset é aplicado ao segundo dígito para evitar que o horário ultrapasse 23:59.
- O MSD das **horas** é resetado quando Q4=3.

Foram criados dois contadores de seis dígitos utilizando a mesma lógica: um para o relógio e outro para o alarme. A necessidade de dois contadores distintos surge porque, no caso do relógio, o clock de um dígito está conectado ao reset do dígito anterior. Isso significa que, quando o dígito menos significativo (LSD) atinge 9 e é resetado, o dígito à esquerda é incrementado em 1, e assim sucessivamente. Em contraste, no alarme, os dígitos funcionam de maneira independente, e o clock de cada um é controlado manualmente por um botão.

2.4 Display de Cristal Líquido

2.4.1 Interface para o controlador do LCD

O código apresentado no Anexo A define um pacote VHDL que contém duas funções utilitárias e um componente responsável pelo controle de um display de cristal líquido (LCD), utilizando bibliotecas específicas para manipulação de sinais lógicos e vetores de bits.

Uma das principais funcionalidades desse código é a conversão de strings em vetores de bits, onde cada caractere é transformado em seu correspondente binário de 8 bits (código ASCII). Essa conversão facilita o envio de informações textuais ao display.

Além disso, o pacote define a interface de controle do LCD, que inclui sinais básicos como "*clock*"e "*reset*", bem como sinais específicos para habilitar o envio de dados e comandar o transporte de instruções ou dados ao LCD. Entre esses sinais, destacam-se os de controle de leitura/escrita, indicação de comando ou dado, habilitação do display, monitoramento do estado do LCD (ocupado ou disponível) e envio dos dados a serem exibidos.

Em resumo, este pacote fornece as funções essenciais para a manipulação e envio de dados ao LCD, além de definir a interface de comunicação com o controlador do display.

2.4.2 Controlador do LCD

O código do Anexo B é responsável por gerenciar a comunicação e controle de um display de cristal líquido (LCD) por meio de uma máquina de estados. A máquina gerencia a inicialização e operação do LCD, alternando entre estados como energização, configuração e envio de dados.

Os parâmetros genéricos permitem configurar diversas características do display, como o número de linhas, o tipo de fonte, a exibição de cursor e piscagem, e o modo de incremento ou deslocamento de caracteres. O controlador utiliza sinais como "*clock*", "*reset*"e "*enable*"para sincronizar a comunicação e enviar instruções através de um barramento.

A FSM garante a inicialização correta do LCD e o envio dos comandos e dados. Utiliza temporizações precisas para que o display processe os comandos de forma adequada, alternando o sinal de habilitação conforme necessário para realizar operações como habilitar o display, limpar a tela, e configurar o modo de entrada.

2.4.3 Exibição no LCD

O código responsável pela lógica de controle do display LCD no relógio digital implementado em VHDL, contido no Anexo C, é projetado para alternar entre dois estados principais: o estado **INICIAL** e o estado **RELOGIO**. No estado inicial, o display exibe mensagens predefinidas em dois conjuntos de frases (uma por linha), enquanto no estado de relógio, o display apresenta as horas, minutos e segundos, que podem ser ajustados por meio de botões.

O código utiliza um processo de controle para alternar entre cinco conjuntos de frases que fornecem informações sobre o projeto e os integrantes do grupo durante o estado inicial, com cada conjunto sendo exibido por 3 segundos antes da transição para o próximo. O ciclo de mensagens segue a seguinte sequência:

- "ELETRONICA"e "Digital UFPE"
- "RELOGIO DIGITAL"e "PROJETO 3"
- "ALYSSON"e "CAVALCANTE"
- "FELIPE"e "BARROS"

- "MARIA"e "VICTORIA"

No estado "**RELOGIO**", o display apresenta as horas, minutos e segundos. A lógica de controle verifica as entradas dos botões e do seletor de exibição para permitir o ajuste do relógio. Dependendo do botão pressionado e da seleção do usuário, o relógio pode ser ajustado em diferentes dígitos (horas, minutos ou segundos), atualizando o LCD de acordo com as entradas fornecidas.

As conversões de caracteres no código estão intimamente ligadas à tabela ASCII. O sistema armazena strings de texto em vetores, onde cada caractere é representado pelo seu valor ASCII em formato binário. O controlador do LCD interpreta esses valores para exibir os caracteres desejados no display. Essa abordagem permite uma comunicação eficaz entre o FPGA e o LCD, utilizando a padronização fornecida pela tabela ASCII para garantir que os caracteres corretos sejam exibidos.

A comunicação entre o FPGA e o display LCD ocorre por meio de um barramento de dados e um sinal de habilitação. O LCD é ativado quando o controlador está disponível, e as instruções para escrever nas linhas do display são enviadas em sequência através de uma máquina de estados. O código também inclui um processo para gerenciar o envio dos caracteres para as linhas do LCD, com cada caractere sendo transmitido conforme o clock principal.

As interfaces do LCD são integradas em dois blocos esquemáticos mostrados nas Figuras 2.6 e 2.7.

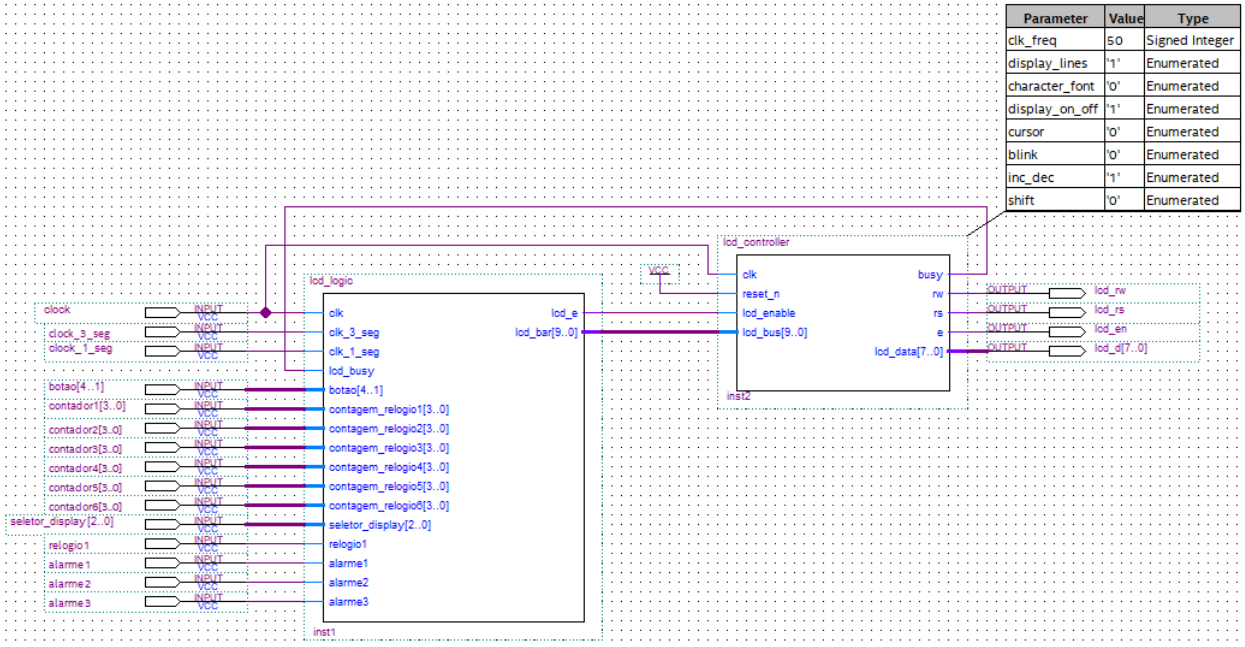


Figura 2.6: Estrutura interna do LCD.

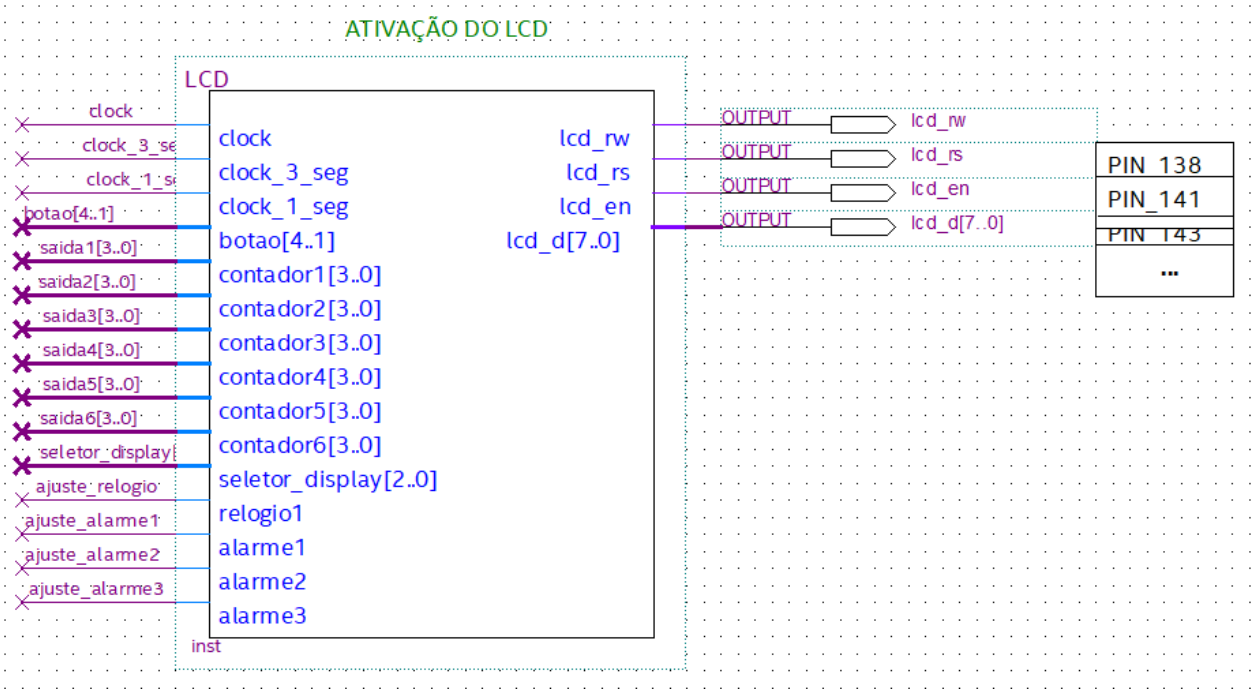


Figura 2.7: Estrutura externa do LCD.

2.5 Ajustes

Para ajustar as horas do relógio e dos alarmes, bem como alternar entre esses modos, é necessário implementar uma função. Para isso, foram criados dois seletores, compostos por contadores de módulo 8 e módulo 4, conforme descrito nos Apêndices C e D, respectivamente. Ambas as funções dependem da ativação do modo de ajuste e do botão correspondente (botão 4, neste caso).

2.5.1 Seletor display

Na primeira função (Apêndice C), o chip ilustrado na Figura 2.8 permite selecionar o dígito a ser ajustado. O botão utilizado para alternar entre os dígitos é o botão 3. Como o relógio está no formato HH:MM:SS, há um total de seis dígitos, o que justifica o uso de um contador de módulo 8, permitindo a variação entre todos os dígitos, tanto no ajuste do relógio quanto nos ajustes doa alarmes.

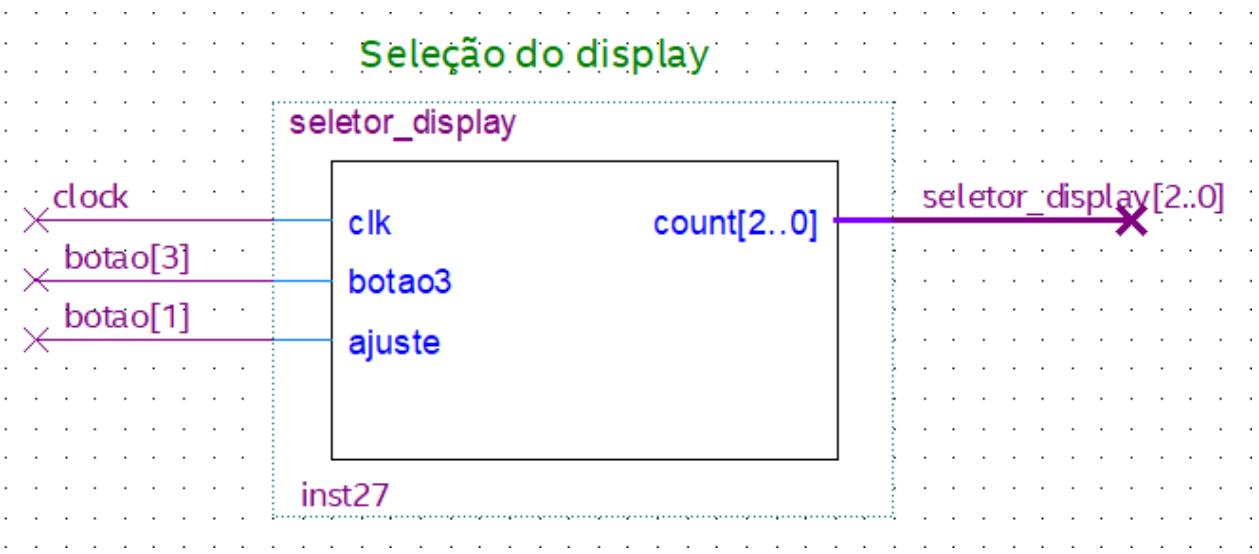


Figura 2.8: Seletor do dígido a ser alterado.

2.5.2 Seletor modo ajuste

Na segunda função (Apêndice D), o chip mostrado na Figura 2.9 permite alternar entre os diferentes modos de ajuste (relógio, alarme 1, alarme 2 e alarme 3). Neste caso, o botão 4 é responsável pela mudança entre os modos.

Este seletor também é utilizado em dois contextos: no código que exibe qual ajuste está sendo realizado e no código de exibição do LCD, que indica o modo disponível para alteração.

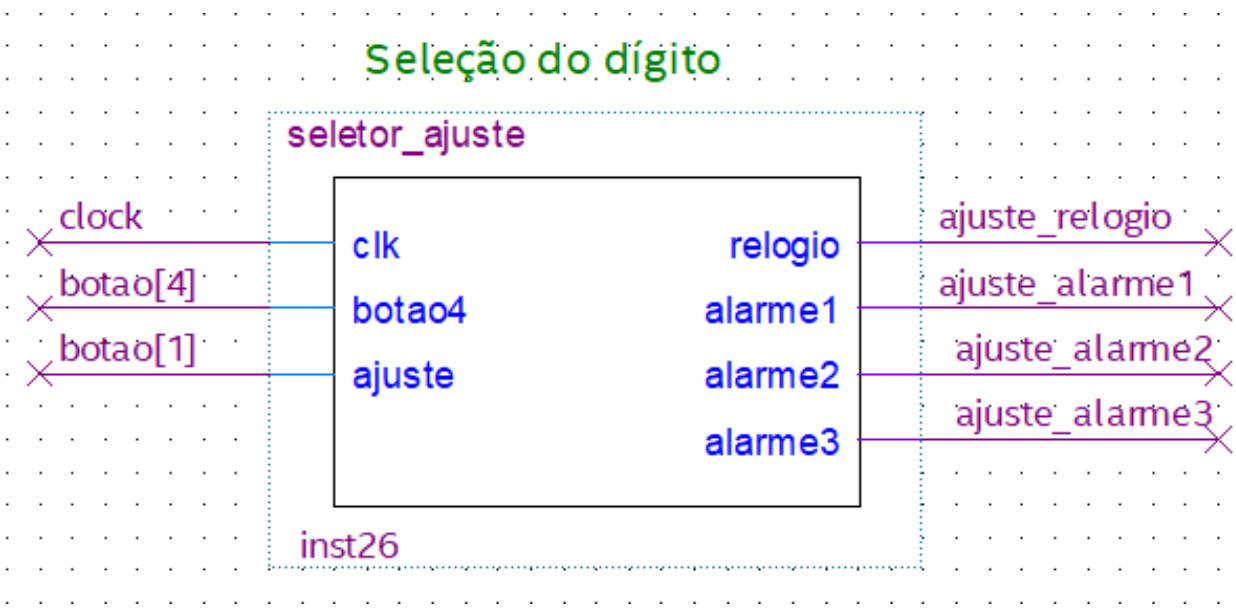


Figura 2.9: Seletor do modo a ser ajustado.

2.6 Relógio

O código apresentado no Apêndice E implementa a lógica de um relógio digital em VHDL, operando em dois estados principais: **INICIAL** e **RELOGIO1**. No estado inicial, o relógio aguarda a ativação do botão 1. Assim como na máquina de estados utilizada para a exibição no LCD, conforme discutido no capítulo "Display de Cristal Líquido", o sistema permanece em espera até que a entrada do relógio seja iniciada. Durante esse estado inicial, são exibidos apenas os dados do projeto e os nomes dos integrantes, e a contagem do relógio ainda não é iniciada.

A arquitetura do código utiliza os contadores de 4 bits descritos no capítulo "Contador de 4 bits"para representar os diferentes componentes do relógio: segundos, minutos e horas, com dois contadores dedicados a cada unidade de tempo.

A lógica de controle verifica o sinal de ajuste e o seletor de dígito para habilitar os contadores apropriados, permitindo a alteração de valores com base na entrada do usuário. Quando o modo de ajuste está ativo, o clock de cada dígito é determinado pelo seletor, e os resets dependem dos valores pré-definidos. Quando o modo de ajuste é desligado, o relógio opera de forma síncrona, recebendo o clock de entrada no dígito menos significativo e propagando os pulsos de clock para os dígitos subsequentes, de acordo com os resets dos contadores anteriores, que possuem valores individuais

pré-definidos.

Os valores contados são transmitidos como saídas ("saida1"a "saida6"), possibilitando que os tempos atualizados sejam utilizados em um código subsequente que processa todas as saídas do relógio e dos alarmes. A implementação, mostrada na Figura 2.10, assegura a ativação e desativação adequadas dos contadores, garantindo um funcionamento estável e confiável do relógio digital.

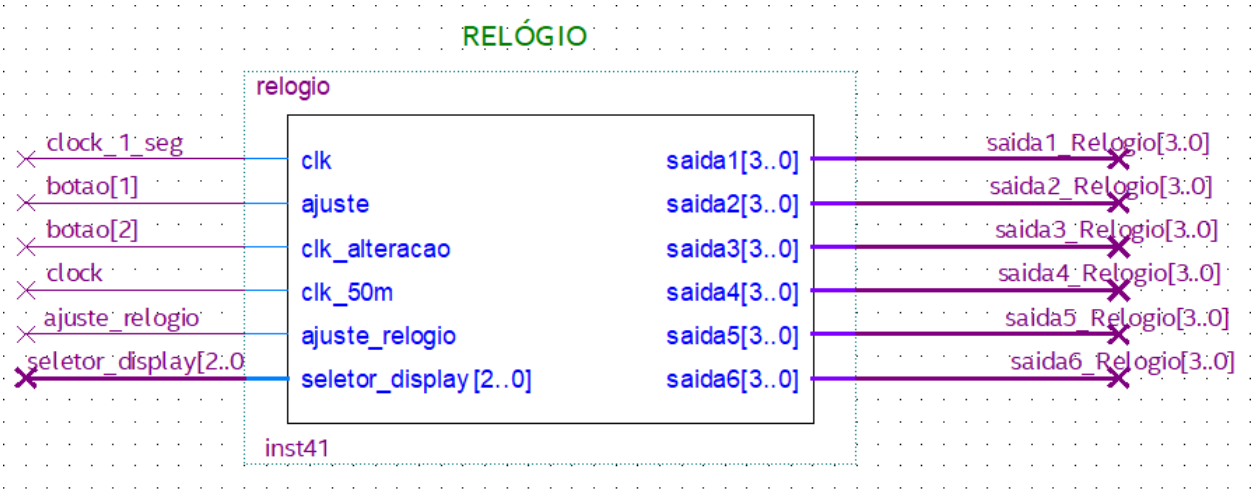


Figura 2.10: Bloco esquemático do relógio.

2.7 Alarme

O código apresentado no Apêndice F implementa a lógica de um sistema de alarme em VHDL. A entidade possui entradas para o clock de alteração, um seletor de display e um seletor de ajuste, além de fornecer saídas que representam o tempo em formato de 4 bits.

A arquitetura utiliza seis contadores de 4 bits para modelar os componentes do alarme, seguindo uma abordagem semelhante à do relógio. A lógica de controle, encapsulada em um processo, monitora o estado do seletor de ajuste. Quando este está ativo, o incremento em cada dígito é controlado por um clock de alteração, enquanto os demais contadores permanecem estabilizados. O seletor de display determina qual dígito será ajustado. Além disso, o código implementa condições para resetar os contadores em momentos específicos, conforme explicado no capítulo "Contador de 4 bits".

Os valores contados são transmitidos como saídas (denominadas "saida1"a "saida6"), permitindo que os tempos atualizados sejam utilizados em códigos subsequentes que processam as saídas do relógio e dos alarmes. Essa implementação, abordada na Figura 2.11, garante a ativação e de-

sativação adequadas dos contadores, assegurando um funcionamento estável e confiável do sistema de alarme.

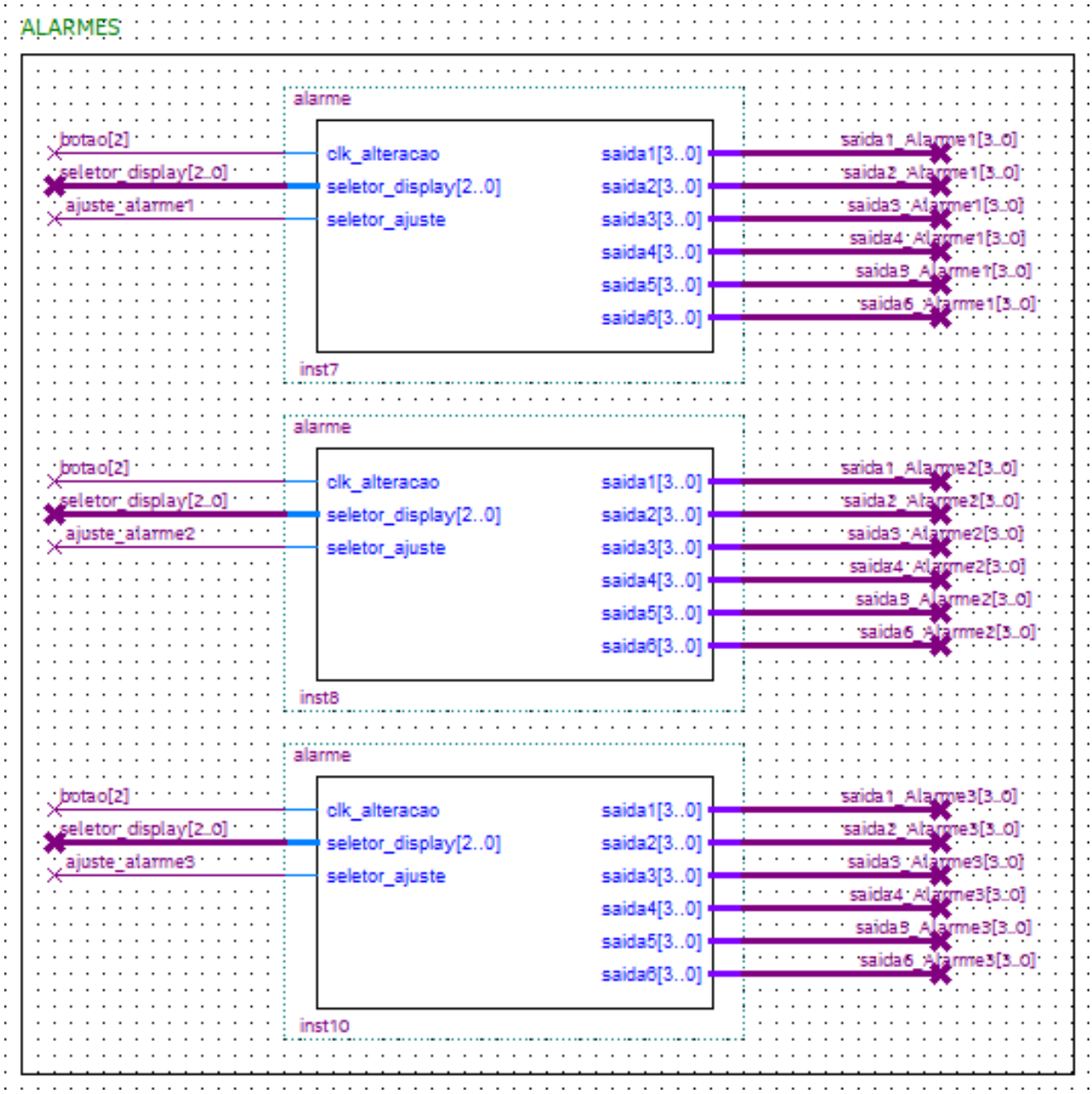


Figura 2.11: Bloco esquemático dos alarmes.

2.8 Gerenciamento de saídas

O código apresentado no Apêndice G implementa uma espécie de multiplexador que combina a funcionalidade de um relógio digital com múltiplos alarmes. Este módulo, presente na Figura 2.12, recebe como entradas os contadores do relógio e dos três alarmes, além de sinais de controle, como o clock, o modo de ajuste e os botões que acionam as diversas funções do sistema.

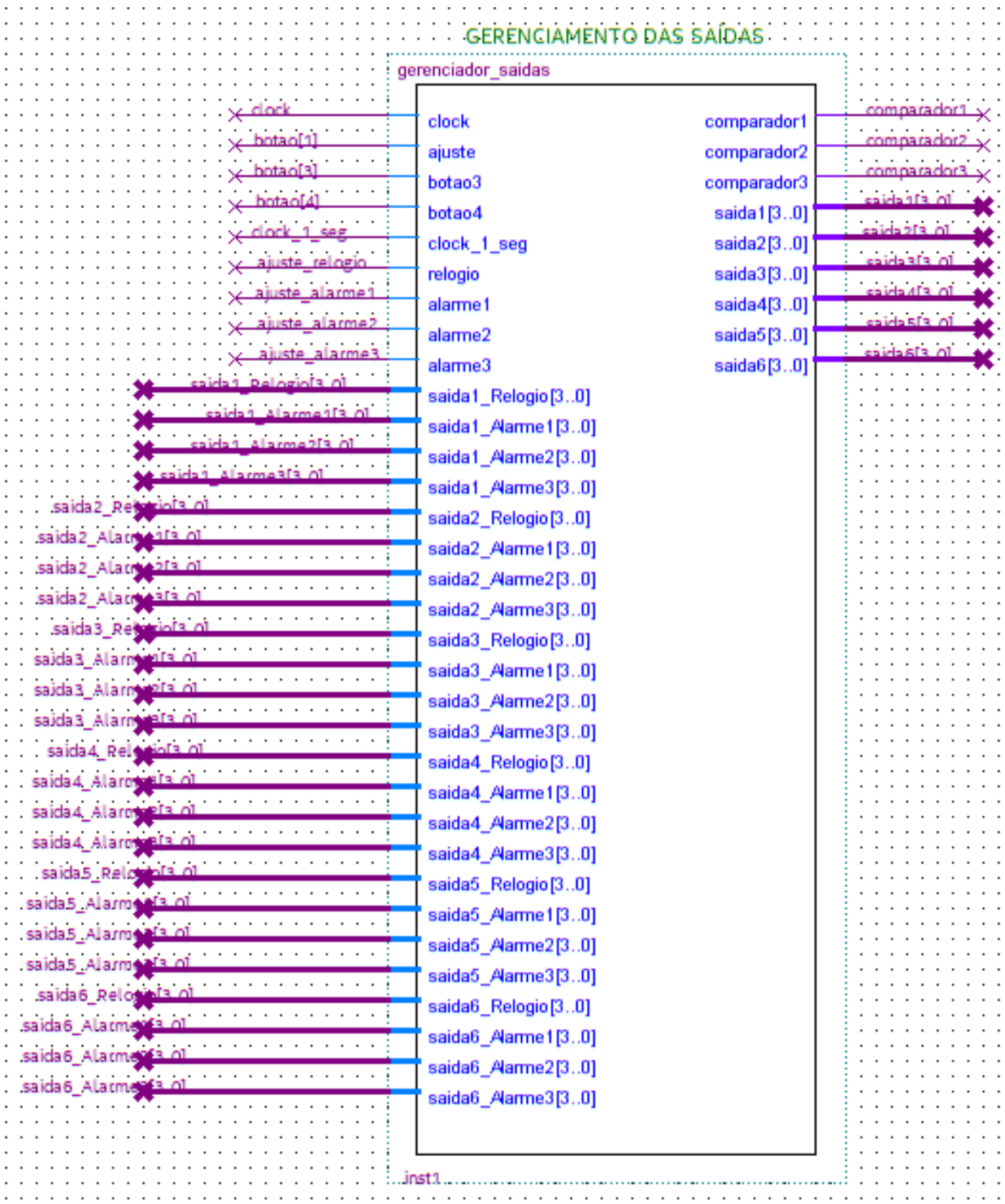


Figura 2.12: Gerenciador das saídas (multiplexador).

A entidade é projetada para receber sinais que representam o estado atual do relógio e de cada

um dos três alarmes, permitindo que o sistema determine qual função deve ser ajustada. As saídas do multiplexador são compostas por seis vetores de 4 bits, que representam exclusivamente os dados de uma das funções — seja do relógio, do alarme 1, do alarme 2 ou do alarme 3. Além disso, os sinais de comparação, identificados como "comparador1", "comparador2"e "comparador3", indicam se há coincidência entre os tempos do relógio e cada um dos alarmes.

A arquitetura do módulo define uma máquina de estados que contempla os estados "**INICIAL**"e "**RELOGIO1**". Essa máquina de estados interage com as outras duas já implementadas no projeto, mencionadas no subcapítulo "Controlador do LCD"e no capítulo "Relógio".

O processo de multiplexação é ativado na borda de subida do sinal de clock. No estado inicial, o sistema aguarda a ativação do relógio por meio do botão 1. Uma vez no segundo estado, o sistema verifica qual função está ativa, seja o relógio ou um dos alarmes. Se o sinal ‘relogio‘ estiver ativo, as saídas correspondem às do relógio. Caso um dos alarmes esteja ativado, as saídas são ajustadas de acordo com os dados do alarme correspondente.

A lógica de comparação, contida no Apêndice H e mostrado na Figura 2.13, é crucial para a funcionalidade dos alarmes. O código compara os valores do relógio com os programados para cada alarme, com a condição de que o sistema não esteja no modo de ajuste. Se houver coincidência entre as saídas do relógio e as saídas de um alarme, o sinal do comparador correspondente é ativado, garantindo que, quando o relógio atingir a hora programada de um alarme, o sinal adequado será acionado. O sistema foi projetado com três comparadores independentes para permitir que cada alarme tenha um toque distinto.

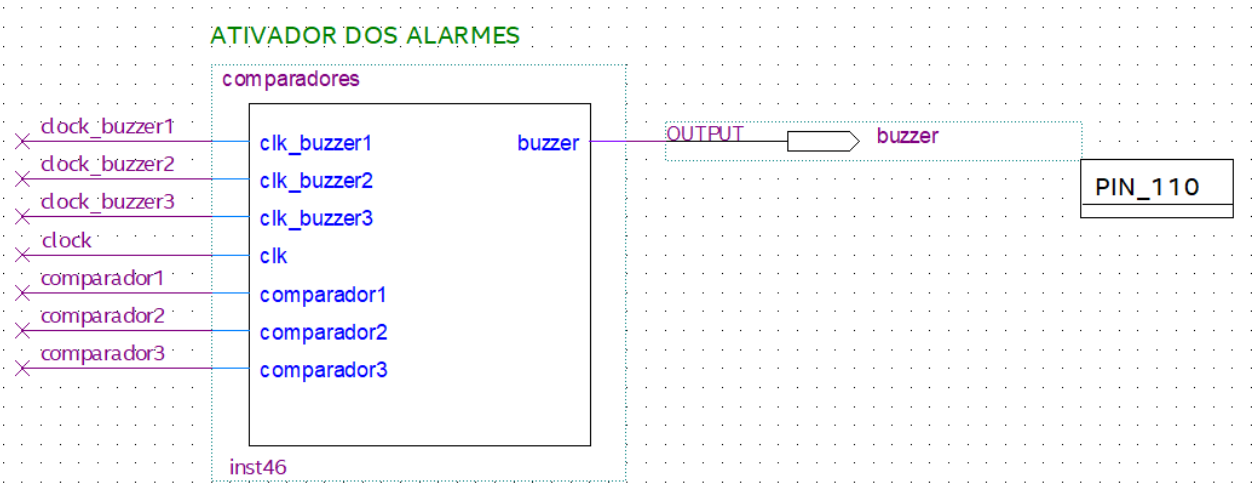


Figura 2.13: Sistema de comparação.

Adicionalmente, se ambos os botões específicos (botões 3 e 4) forem pressionados, os sinais dos comparadores são desativados. Por outro lado, se o sistema estiver no modo de ajuste, todos os sinais de comparação são igualmente desativados, assegurando que a lógica de comparação não interfira nas configurações dos alarmes.

Em resumo, o código possibilita uma seleção eficiente entre as saídas do relógio e dos alarmes, garantindo uma comparação precisa entre os tempos. Isso promove um funcionamento coeso e confiável do sistema de alarme e do relógio digital.

3 Manual de Operação

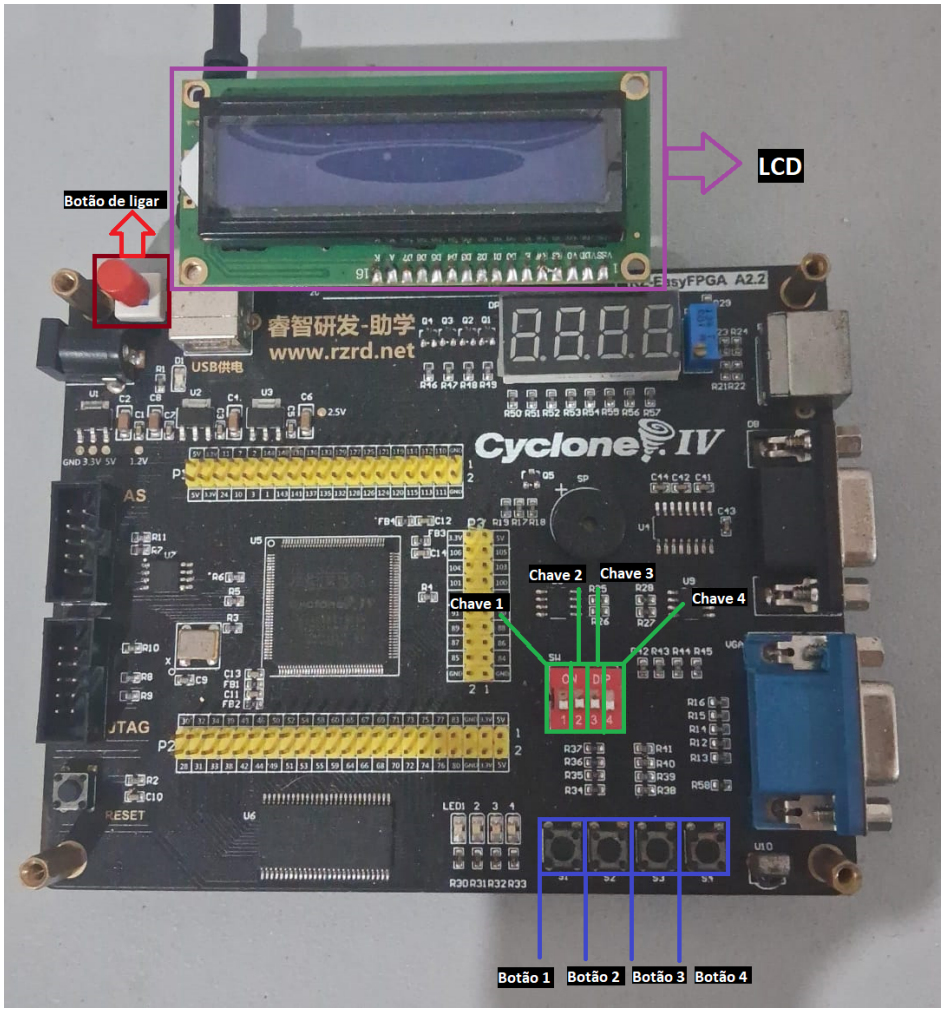


Figura 3.1: Identificação dos itens no FPGA.

1. Introdução

- Este manual descreve o funcionamento e o procedimento de ajuste do relógio digital com alarme. O dispositivo é projetado para fornecer a hora atual e permitir o ajuste de um alarme

programável. O display de cristal líquido, por onde são exibidos o relógio e as informações do projeto, está marcado em amarelo na Figura 3.1.

- As ferramentas utilizadas para operar o relógio digital são as chaves (marcadas em verde na Figura 3.1) e os botões (marcados em azul na Figura 3.1). É importante observar que as chaves e os botões possuem funções equivalentes. Em outras palavras, ativar a chave 1 é funcionalmente equivalente a pressionar e segurar o botão 1. Esta correspondência garante que as operações realizadas com as chaves e os botões produzam os mesmos efeitos no ajuste e controle do dispositivo.

2. Ligando o Circuito

   - Aperte o botão vermelho (marcado em vermelho na Figura 3.1) para ligar o FPGA;

   - Grave o circuito na placa;

   - O relógio iniciará no modo de apresentação em que mostrará os dados do projeto e o nome dos integrantes do grupo;

   - Aperte o botão 1 para que o relógio inicie a contagem; o LCD passará a apresentar o relógio digital no formato HH:MM:SS (Horas:Minutos:Segundos) com início da contagem em 00:00:01.

3. Ajustes do Relógio e Alarmes

   (a) Entrando no Ajuste do Relógio

   - Para entrar no modo de ajuste do relógio, habilite a chave 1;

   - O display correspondente ao dígito de ajuste começará a piscar.

   (b) Alterando o Dígito de Ajuste

   - Para alterar o dígito que está piscando, aperte o botão 2 repetidamente até que o número desejado seja exibido no display.

   (c) Selecionando o Dígito para Alteração

   - Para mudar o dígito que está sendo ajustado, pressione o botão 3 até que o display mostre o dígito desejado para a alteração.

   (d) Entrando no Ajuste dos Alarmes

   - Caso deseje entrar no modo de ajuste dos alarmes, aperte o botão 4;

   - O sistema sairá do modo de ajuste do relógio e entrará no modo de ajuste do alarme 1;

   - Repita o processo de seleção e alteração do dígito de ajuste;

- Use o botão 4 para selecionar o próximo alarme até que tenha feito todos os ajustes desejados.

(e) Saindo do Ajuste

- Para sair do modo de ajuste e registrar os horários dos alarmes e relógio, desligue a chave 1;

- O horário do alarme será salvo e o display retornará a mostrar a hora atual do relógio.

4. Desligar o alarme

- Quando o alarme disparar, um sinal sonoro será emitido e o display começará a piscar;

- Cada alarme terá uma frequência sonora diferente, gerando diferentes sons;

- Para desativar o alarme, pressione simultaneamente os botões 3 e 4.

5. Notas Finais

- Certifique-se de seguir os passos corretamente para ajustar tanto o horário do relógio quanto os horários dos alarmes.

4 Resultados

O projeto desenvolvido resultou em um relógio digital com despertador programado em VHDL, exibido no formato HH:MM:SS em um Display de Cristal Líquido (LCD), permitindo a contagem de 24 horas. O relógio inicia sua contagem no tempo 00:00:01 (1 segundo) e oferece funções específicas para ajuste de horário e configuração de 3 alarmes através de botões e chaves na placa FPGA.

Ao inicializar o sistema, o usuário é apresentado a frases no LCD que variam a cada 3 segundos mostrando o nome dos integrantes do projeto, disciplina e título do projeto. Para iniciar a exibição do relógio é necessário que o usuário pressione o primeiro botão podendo ser chamado de botão 1. Logo após pressionado, o relógio é apresentado e sua contagem começa, aparecendo na linha de baixo do LCD o nome "RELOGIO". Caso o usuário queira entrar no modo de ajuste de horas, o sistema será ativado por meio da chave 1. Nesse modo, o pixel selecionado para alteração começará a piscar e na linha inferior do LCD a frase "AJUSTE RELOGIO C"é apresentada, o usuário pode selecionar qual dos pixel deseja ajustar por meio do botão 3. O valor exibido pode ser incrementado pressionando o botão 2, com o formato crescente e após o máximo da contagem do pixel em foco a contagem retorna a zero. Após concluir o ajuste, o usuário pode desligar a chave de ajuste e retorna para o relógio ou, alternativamente, pressionar o botão 4 para entrar no modo de configuração de alarmes, onde é apresentado 3 diferentes alarmes a cada clique no botão 4 e retorna para o relógio no 4 clique, o ajuste dos alarmes segue o mesmo padrão de ajuste do relógio para incremento e seleção do display.

O programa possui a função despertador que foi programada de forma a acionar um alerta sonoro quando a hora do relógio coincide com qualquer um dos alarmes configurados. Neste caso, o buzzer da placa FPGA é ativado. O buzzer permanece ativo até que os botões 3 e 4 sejam pressionados simultaneamente para desativá-lo, garantindo que o usuário interaja fisicamente com o sistema para interromper o alarme. Vale salientar que, cada alarme possui sua própria música predefinida.

O sistema possui um código para o LCD que trabalha para mostrar as horas do relógio e horas do alarme na linha um do LCD e na linha dois do LCD mostrar frases indicando o que está

sendo mostrado na linha um. Além disso, cria a percepção de que todos os pixels estão ativos simultaneamente, embora fisicamente apenas um esteja aceso em um dado momento. Ademais, foi implementado um sistema de conversor de clock, que ajusta o clock nativo da placa de 50 MHz para gerar um clock adequado ao funcionamento do relógio e seus demais sistemas. Clocks separados foram usados para diferentes funções, como gerar pulsos de um segundo, controlar o piscar dos pixels no modo de ajuste e ativar o buzzer em caso de alarme.

5 Discussão dos Resultados

O projeto desenvolvido atingiu a maior parte dos objetivos, com um relógio digital funcional programado em VHDL, oferecendo contagem de 24 horas, ajuste de horário e configuração de três alarmes com despertadores distintos. No entanto, algumas funcionalidades exigidas pelo projeto não puderam ser implementadas devido à complexidade e à carga de trabalho associada ao desenvolvimento e integração dessas funções na linguagem VHDL.

Um dos principais desafios foi a implementação de uma função decremental para o ajuste de horas e alarmes. Enquanto o sistema permite incrementar os valores exibidos de forma crescente, não conseguimos incorporar a funcionalidade de decremento. Isso foi devido à dificuldade em estruturar essa função em VHDL, dentro do tempo disponível, sem comprometer outras funcionalidades já implementadas.

Outro ponto crítico foi a ausência de um sistema de debounce para lidar com os cliques dos botões. O debounce é essencial para evitar que múltiplos comandos indesejados sejam captados quando um botão é pressionado. A implementação dessa função em VHDL, que exige a criação de um filtro de software para eliminar "ruídos"nos sinais, mostrou-se desafiadora devido à complexidade envolvida e ao nível de detalhamento necessário para lidar com a temporização correta. Como resultado, o projeto ficou suscetível a múltiplos acionamentos de botões em um único clique, o que pode afetar a precisão e a experiência de uso do sistema.

Além disso, planejamos desenvolver uma rotina de músicas para o buzzer que utilizaria frequências distintas convertidas pelo conversor de clock, criando sons diferentes para cada alarme. No entanto, essa funcionalidade também não foi alcançada, principalmente devido à dificuldade em manipular frequências em VHDL para gerar sons de forma controlada e diferenciada. A complexidade de adaptar a lógica necessária para o controle de sons, juntamente com o tempo necessário para ajustar corretamente os conversores de clock, superou os conhecimentos que tínhamos no momento sobre a linguagem.

Essas falhas na implementação das funções adicionais refletem tanto a carga de trabalho do

projeto quanto a curva de aprendizado da linguagem VHDL, que, por ser de baixo nível, exige um controle detalhado sobre a temporização e os sinais. Apesar disso, o sistema principal do relógio, incluindo as funções de ajuste e alarme, foi concluído com sucesso, proporcionando uma base sólida para futuras melhorias. Em projetos subsequentes, poderíamos explorar com mais profundidade a incorporação dessas funcionalidades, dado um maior tempo de desenvolvimento e refinamento do conhecimento técnico.

Esses desafios nos mostraram a importância de planejamento adicional e de aprofundar o domínio das ferramentas de desenvolvimento, para que futuras versões do sistema possam incluir essas funcionalidades ausentes de forma eficaz.

6 Conclusão

O projeto de relógio digital com despertador programado em VHDL resultou em um sistema funcional que atingiu os principais objetivos estabelecidos, como a contagem de 24 horas, ajuste de horário, configuração de alarmes, despertador com 3 sinais sonoros distintos pré programados e interação por meio de um display de Cristal Líquido (LCD). O sistema demonstrou ser eficiente ao utilizar multiplexação e conversão de clock, proporcionando uma interface visual clara e operações confiáveis.

No entanto, algumas funcionalidades desejadas, como o ajuste decremental, o debounce para controle de botões e a criação de rotinas sonoras diferenciadas para os alarmes, não puderam ser implementadas. Essas funções exigiam um nível de complexidade superior, que, diante da carga do projeto e dos limites de conhecimento sobre VHDL, não foi possível integrar adequadamente.

Mesmo com essas limitações, o projeto mostrou-se bem-sucedido dentro de seu escopo, fornecendo uma plataforma sólida para ajustes e melhorias futuras. A experiência adquirida ao enfrentar as dificuldades de implementação em VHDL reforçou o entendimento sobre o desenvolvimento de sistemas embarcados e as especificidades do controle de hardware em linguagem de descrição de hardware. Com um maior domínio técnico e tempo adicional, seria possível explorar as funcionalidades não implementadas e otimizar ainda mais o desempenho e usabilidade do sistema.

No contexto acadêmico e prático, este projeto destaca-se como um exemplo significativo de aplicação de lógica digital, demonstrando a capacidade de controlar de forma precisa um relógio digital e seus alarmes, ao mesmo tempo em que evidenciou desafios que devem ser considerados em versões futuras.

7 Referências Bibliográficas

[1] Tocci, Ronald J., Neal S. Widmer, e Gregory L. Moss. *Sistemas Digitais: Princípios e Aplicações*. 11ª ed., Pearson Prentice Hall, 2007.

8 Apêndices

Apêndice A Conversor de Clock

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

entity Conversor_de_clock is
    generic (
        Contagem : integer := 10000000;  -- m dulo de contagem de clocks
        n : integer := 32            -- N mero de bits do contador
    );
    port (
        clk_in : in std_logic;
        reset : in std_logic := '0';
        clk_out : out std_logic
    );
end Conversor_de_clock;

architecture Behavioral of Conversor_de_clock is
    -- Contador din mico de acordo com o n mero de bits definido
    signal counter : unsigned(n-1 downto 0) := (others => '0');
    signal clk_out_reg : std_logic := '0';  -- Registrador do clock de sa da
begin

    -- Processo s ncrono controlado pela borda de subida do clock de entrada
    process(clk_in, reset)
    begin
        if reset = '1' then
            counter <= (others => '0');      -- Reset do contador
            clk_out_reg <= '0';              -- Reset do clock de sa da
```

```
29      elsif rising_edge(clk_in) then
30         if counter = (Contagem - 1) then
31            counter <= (others => '0');   -- Reinicia o contador
32            clk_out_reg <= not clk_out_reg;  -- Inverte o clock de sa da
33
34         else
35            counter <= counter + 1;       -- Incrementa o contador
36         end if;
37      end if;
38   end process;
39   -- Atribui o valor do registrador    sa da do clock
40   clk_out <= clk_out_reg;
41
42end Behavioral;
```

Apêndice B Contador de 4 bits

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4
5 entity contador_4bits is
6    port (
7       clk : in std_logic;
8       reset : in std_logic := '0';
9       count : out std_logic_vector(3 downto 0)
10   );
11end contador_4bits;
12
13architecture Behavioral of contador_4bits is
14   signal count_reg : std_logic_vector(3 downto 0);
15begin
16
17   -- Processo s ncrono controlado pela borda de subida do clock e reset
18   process(clk, reset)
19   begin
20      if reset = '1' then
21         count_reg <= (others => '0');  -- Reset do contador
```

```
22      elsif rising_edge(clk) then
23         count_reg <= std_logic_vector(unsigned(count_reg) + 1);  -- Incrementa o
               contador
24      end if;
25   end process;
26
27   -- Atribui o valor do registrador    sa da
28   count <= count_reg;
29
30 end Behavioral;
```

Apêndice C Seletor display

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4
5  -- Declara  o da entidade
6  entity seletor_display is
7     port (
8        clk,botao3,ajuste : in std_logic; -- Sinal de clock para mudar a sa da
             (00,01,10,11)
9        count : out std_logic_vector(2 downto 0)  -- Sa da do contador (2 bits)
10    );
11 end seletor_display;
12
13 -- Descri  o do comportamento do contador
14 architecture Behavioral of seletor_display is
15    signal count_reg : std_logic_vector(2 downto 0);
16 begin
17    -- Processo s ncrono controlado pela borda de subida do clock
18    process(clk)
19    begin
20     if rising_edge(clk) then
21       if (botao3='1') then
22         if count_reg = "101" then    -- Valor bin rio de 5, contagem m xima do
               nosso contador.
23           count_reg <= (others => '0');       -- Reinicia o contador
```

```
24          else
25            count_reg <= std_logic_vector(unsigned(count_reg) + 1);  -- Incrementa o
                  contador
26          end if;
27        end if;
28
29
30        if ajuste = '0' then
31          count_reg <= (others => '0');
32        end if;
33      end if;
34    end process;
35    -- Atribui o valor do registrador     sa da
36    count <= count_reg;
37
38 end Behavioral;
```

Apêndice D Seletor modo ajuste

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4
5  entity seletor_ajuste is
6     port (
7        clk,botao4,ajuste: in std_logic;
8      relogio, alarme1,alarme2,alarme3 :out std_logic
9
10    );
11 end seletor_ajuste;
12
13 architecture Behavioral of seletor_ajuste is
14    signal count_reg : std_logic_vector(1 downto 0);
15 begin
16    -- Processo s ncrono controlado pela borda de subida do clock
17    process(clk)
18    begin
19     if rising_edge(clk) then
```

```
20      if botao4 = '1' then
21        if (count_reg = "11") then    -- Valor bin rio de 3, contagem m xima do
              nosso contador.
22          count_reg <= (others => '0');       -- Reinicia o contador
23        else
24          count_reg <= std_logic_vector(unsigned(count_reg) + 1);  -- Incrementa o
                contador
25        end if;
26      end if;
27
28
29      if (ajuste = '0') then
30        count_reg <= (others => '0');
31      end if;
32    end if;
33
34    case count_reg is
35      when "00" => relogio <= '1';
36                alarme1  <='0';
37                alarme2  <='0';
38                alarme3  <='0';
39
40      when "01" => relogio <= '0';
41                alarme1  <='1';
42                alarme2  <='0';
43                alarme3  <='0';
44
45      when "10" => relogio <= '0';
46                alarme1  <='0';
47                alarme2  <='1';
48                alarme3  <='0';
49
50      when "11" => relogio <= '0';
51                alarme1  <='0';
52                alarme2  <='0';
53                alarme3  <='1';
54    end case;
55  end process;
56
```

```
57
58 end Behavioral;
```

Apêndice E Relógio

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity relogio is
7     port (
8        clk,ajuste,clk_alteracao,clk_50m, ajuste_relogio : in std_logic;
9      seletor_display : std_logic_vector(2 downto 0);
10       saida1, saida2, saida3, saida4,saida5,saida6 : out std_logic_vector(3 downto
             0)
11    );
12 end relogio;
13
14 architecture Behavioral of relogio is
15
16   TYPE ESTADOS IS (INICIAL, RELOGIO1);
17   SIGNAL  estado  : ESTADOS := INICIAL;
18
19   signal temp_saida1, temp_saida2, temp_saida3, temp_saida4,temp_saida5,
        temp_saida6 : std_logic_vector(3 downto 0);
20   Signal clock1, clock2, clock3, clock4,clock5,clock6, reset1, reset2, reset3,
        reset4,reset5,reset6 : std_logic;
21   component contador_4bits
22       port (
23          clk : in std_logic;
24          reset : in std_logic;
25          count : out std_logic_vector(3 downto 0)
26       );
27    end component;
28
29 begin
30
```

```
31  contador1 : contador_4bits PORT MAP (clk => clock1,reset => reset1, count =>
        temp_saida1); -- segundo menos significativo
32
33  contador2 : contador_4bits PORT MAP (clk => clock2,reset => reset2, count =>
        temp_saida2); -- segundo mais significativo
34
35  contador3 : contador_4bits PORT MAP (clk => clock3,reset => reset3, count =>
        temp_saida3); -- hora menos significativo
36
37  contador4 : contador_4bits PORT MAP (clk => clock4,reset => reset4, count =>
        temp_saida4); -- hora mais significativo
38
39  contador5 : contador_4bits PORT MAP (clk => clock5,reset => reset5, count =>
        temp_saida5); -- hora menos significativo
40
41  contador6 : contador_4bits PORT MAP (clk => clock6,reset => reset6, count =>
        temp_saida6); -- hora mais significativo
42
43  process(clk_50m)
44  begin
45    IF rising_edge(clk_50m) THEN
46      IF (ajuste = '1') THEN
47        estado <= RELOGIO1;
48      END IF;
49      case estado is
50        when INICIAL =>
51          if temp_saida1 = "0000" then
52            clock1 <= clk;
53          else clock1 <= '0'; end if;
54
55        when RELOGIO1 =>
56          if ajuste_relogio = '1' then
57            case seletor_display is
58              when "000" => clock1 <= clk_alteracao; clock2 <= '0'; clock3 <= '0';
                    clock4 <= '0'; clock5<= '0'; clock6 <= '0';
59              when "001" => clock2 <= clk_alteracao; clock1 <= '0'; clock3 <= '0';
                    clock4 <= '0'; clock5<= '0'; clock6 <= '0';
60              when "010" => clock3 <= clk_alteracao; clock1 <= '0'; clock2 <= '0';
                    clock4 <= '0'; clock5<= '0'; clock6 <= '0';
```

```
61              when "011" => clock4 <= clk_alteracao; clock1 <= '0'; clock2 <= '0';
                    clock3 <= '0'; clock5<= '0'; clock6 <= '0';
62              when "100" => clock5 <= clk_alteracao; clock1 <= '0'; clock2 <= '0';
                    clock3 <= '0'; clock4<= '0'; clock6 <= '0';
63              when "101" => clock6 <= clk_alteracao; clock1 <= '0'; clock2 <= '0';
                    clock3 <= '0'; clock4<= '0'; clock5 <= '0';
64              when others =>
65                clock1 <= '0';clock2 <= '0';clock3 <= '0';clock4 <= '0';clock5 <=
                     '0';clock6 <= '0';
66            end case;
67
68              -- reset segundo menos significativo
69            if (temp_saida1 = "1010") then -- reseta no 10
70              reset1 <= '1';
71            else
72              reset1 <= '0';
73            end if;
74              -- reset segundo mais significativo
75            if (temp_saida2 = "0110") then -- reseta no 6
76              reset2 <= '1';
77            else
78              reset2 <= '0';
79            end if;
80            -- reset minuto menos significativo
81            if (temp_saida3 = "1010") then -- reseta no 10
82              reset3 <= '1';
83            else
84              reset3 <= '0';
85            end if;
86            -- reset minuto mais significativo
87            if (temp_saida4 = "0110") then -- reseta no 6
88              reset4 <= '1';
89            else
90              reset4 <= '0';
91            end if;
92            -- reset hora menos significativo
93            if (temp_saida5 = "1010") or (temp_saida6 = "0010" and temp_saida5 >=
                  "0100" ) then
94              reset5 <= '1';
```

```
95              else
96                reset5 <= '0';
97              end if;
98              -- reset hora mais significativo
99              if (temp_saida6 = "0011")  then
100               reset6 <= '1';
101             else
102                 reset6 <= '0';
103             end if;
104           end if;
105
106           if (ajuste = '0') then
107             clock1 <= clk;
108             clock2 <= reset1;
109             clock3 <= reset2;
110             clock4 <= reset3;
111             clock5 <= reset4;
112             clock6 <= reset5;
113               -- reset segundo menos significativo
114             if (temp_saida1 = "1010") then -- reseta no 10
115               reset1 <= '1';
116             else
117               reset1 <= '0';
118             end if;
119
120               -- reset segundo mais significativo
121             if (temp_saida2 = "0110") then -- reseta no 6
122               reset2 <= '1';
123             else
124               reset2 <= '0';
125             end if;
126             -- reset minuto menos significativo
127             if (temp_saida3 = "1010") then -- reseta no 10
128               reset3 <= '1';
129             else
130               reset3 <= '0';
131             end if;
132             -- reset minuto mais significativo
133             if (temp_saida4 = "0110") then -- reseta no 6
```

```
134                reset4 <= '1';
135              else
136                reset4 <= '0';
137              end if;
138              -- reset hora menos significativo
139              if (temp_saida5 = "1010") or (temp_saida6 = "0010" and temp_saida5 >=
                     "0100" ) then
140                reset5 <= '1';
141              else
142                reset5 <= '0';
143              end if;
144              -- reset hora mais significativo
145              if (temp_saida6 = "0011")  then
146                reset6 <= '1';
147              else
148                  reset6 <= '0';
149              end if;
150            end if;
151            saida1 <= temp_saida1;
152            saida2 <= temp_saida2;
153            saida3 <= temp_saida3;
154            saida4 <= temp_saida4;
155            saida5 <= temp_saida5;
156            saida6 <= temp_saida6;
157        end case;
158      end if;
159    end process;
160 end;
```

Apêndice F Alarme

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity alarme is
7     port (
```

```
8       clk_alteracao   : in std_logic;
9     seletor_display : in std_logic_vector(2 downto 0);
10    seletor_ajuste  : in std_logic;
11      saida1, saida2, saida3, saida4,saida5,saida6 : out std_logic_vector(3 downto
            0)
12
13
14   );
15 end alarme;
16
17 architecture Behavioral of alarme is
18   signal temp_saida1, temp_saida2, temp_saida3, temp_saida4,temp_saida5,
       temp_saida6 : std_logic_vector(3 downto 0);
19   Signal clock1, clock2, clock3, clock4,clock5,clock6, reset1, reset2, reset3,
       reset4,reset5,reset6 : std_logic;
20   component contador_4bits
21       port (
22          clk   : in std_logic;
23          reset : in std_logic;
24          count : out std_logic_vector(3 downto 0)
25       );
26    end component;
27
28 begin
29
30   contador1 : contador_4bits PORT MAP (clk => clock1,reset => reset1, count =>
       temp_saida1); -- segundo menos significativo
31
32   contador2 : contador_4bits PORT MAP (clk => clock2,reset => reset2, count =>
       temp_saida2); -- segundo mais significativo
33
34   contador3 : contador_4bits PORT MAP (clk => clock3,reset => reset3, count =>
       temp_saida3); -- minuto menos significativo
35
36   contador4 : contador_4bits PORT MAP (clk => clock4,reset => reset4, count =>
       temp_saida4); -- minuto mais significativo
37
38   contador5 : contador_4bits PORT MAP (clk => clock5,reset => reset5, count =>
       temp_saida5); -- hora menos significativo
```

```
39
40  contador6 : contador_4bits PORT MAP (clk => clock6,reset => reset6, count =>
        temp_saida6); -- hora mais significativo
41
42  process(seletor_display,clk_alteracao)
43  begin
44    if seletor_ajuste = '1' THEN
45
46      case seletor_display is
47        when "000" => clock1 <= clk_alteracao; clock2 <= '0'; clock3 <= '0';
              clock4 <= '0'; clock5<= '0'; clock6 <= '0';
48        when "001" => clock2 <= clk_alteracao; clock1 <= '0'; clock3 <= '0';
              clock4 <= '0'; clock5<= '0'; clock6 <= '0';
49        when "010" => clock3 <= clk_alteracao; clock1 <= '0'; clock2 <= '0';
              clock4 <= '0'; clock5<= '0'; clock6 <= '0';
50        when "011" => clock4 <= clk_alteracao; clock1 <= '0'; clock2 <= '0';
              clock3 <= '0'; clock5<= '0'; clock6 <= '0';
51        when "100" => clock5 <= clk_alteracao; clock1 <= '0'; clock2 <= '0';
              clock3 <= '0'; clock4<= '0'; clock6 <= '0';
52        when "101" => clock6 <= clk_alteracao; clock1 <= '0'; clock2 <= '0';
              clock3 <= '0'; clock4<= '0'; clock5 <= '0';
53        when others =>
            clock1 <= '0';clock2 <= '0';clock3 <= '0';clock4 <= '0';clock5 <= '0';
                clock6 <= '0';
55      end case;
56    end if;
57
58      saida1 <= temp_saida1;
59      saida2 <= temp_saida2;
60      saida3 <= temp_saida3;
61      saida4 <= temp_saida4;
62      saida5 <= temp_saida5;
63      saida6 <= temp_saida6;
64
65      -- reset segundo menos significativo
66
67      if (temp_saida1 = "1010") then -- reseta no 10
68        reset1 <= '1';
69      else
```

```
70          reset1 <= '0';
71        end if;
72
73          -- reset segundo mais significativo
74
75        if (temp_saida2 = "0110") then -- reseta no 6
76          reset2 <= '1';
77        else
78          reset2 <= '0';
79        end if;
80
81        -- reset minuto menos significativo
82
83        if (temp_saida3 = "1010") then -- reseta no 10
84          reset3 <= '1';
85        else
86          reset3 <= '0';
87        end if;
88
89        -- reset minuto mais significativo
90
91        if (temp_saida4 = "0110") then -- reseta no 6
92          reset4 <= '1';
93        else
94          reset4 <= '0';
95        end if;
96
97        -- reset hora menos significativo
98
99        if (temp_saida5 = "1010") or (temp_saida6 = "0010" and temp_saida5 >= "0100"
              ) then
100         reset5 <= '1';
101       else
102         reset5 <= '0';
103       end if;
104
105       -- reset hora mais significativo
106
107       if (temp_saida6 = "0011")  then
```

```
108         reset6 <= '1';
109       else
110           reset6 <= '0';
111       end if;
112
113
114   end process;
115 end;
```

Apêndice G Gerenciamento de saídas

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4
5  entity gerenciador_saidas is
6     port (
7      clock,ajuste,botao3,botao4,clock_1_seg : in std_logic;
8        relogio,alarme1,alarme2,alarme3 : in std_logic;  -- Seleciona qual fun   o
             ajustar
9      comparador1, comparador2,comparador3: out std_logic;
10       saida1_Relogio,saida1_Alarme1,saida1_Alarme2,saida1_Alarme3: in
             std_logic_vector(3 downto 0);
11     saida2_Relogio,saida2_Alarme1,saida2_Alarme2,saida2_Alarme3: in
           std_logic_vector(3 downto 0);
12     saida3_Relogio,saida3_Alarme1,saida3_Alarme2,saida3_Alarme3: in
           std_logic_vector(3 downto 0);
13     saida4_Relogio,saida4_Alarme1,saida4_Alarme2,saida4_Alarme3: in
           std_logic_vector(3 downto 0);
14     saida5_Relogio,saida5_Alarme1,saida5_Alarme2,saida5_Alarme3: in
           std_logic_vector(3 downto 0);
15     saida6_Relogio,saida6_Alarme1,saida6_Alarme2,saida6_Alarme3: in
           std_logic_vector(3 downto 0);
16     saida1,saida2,saida3,saida4,saida5,saida6: out std_logic_vector (3 downto 0)
17    );
18 end gerenciador_saidas;
19
20 architecture Behavioral of gerenciador_saidas is
```

```
21   TYPE ESTADOS IS (INICIAL, RELOGIO1);
22   SIGNAL  estado  : ESTADOS := INICIAL;
23 begin
24
25    -- Processo de multiplexa  o controlado por 'acender_displays'
26    process(clock)
27    begin
28     IF rising_edge(clock) THEN
29
30
31       case estado is
32       when INICIAL =>
33         IF (ajuste = '1') THEN
34         estado <= RELOGIO1;
35         END IF;
36
37       when RELOGIO1 =>
38           if relogio = '1' then
39             saida1 <= saida1_Relogio; saida2 <= saida2_Relogio; saida3 <=
                   saida3_Relogio; saida4 <= saida4_Relogio; saida5 <= saida5_Relogio;
                    saida6 <= saida6_Relogio;
40
41           elsif alarme1 = '1' then
42
43             saida1 <= saida1_alarme1; saida2 <= saida2_alarme1; saida3 <=
                   saida3_alarme1; saida4 <= saida4_alarme1; saida5 <= saida5_alarme1;
                    saida6 <= saida6_alarme1;
44
45
46           elsif alarme2 = '1' then
47             saida1 <= saida1_alarme2; saida2 <= saida2_alarme2; saida3 <=
                   saida3_alarme2; saida4 <= saida4_alarme2; saida5 <= saida5_alarme2;
                    saida6 <= saida6_alarme2;
48
49
50           elsif alarme3 = '1' then
51             saida1 <= saida1_alarme3; saida2 <= saida2_alarme3; saida3 <=
                   saida3_alarme3; saida4 <= saida4_alarme3; saida5 <= saida5_alarme3;
                    saida6 <= saida6_alarme3;
```

```
52          end if;
53
54
55             -- Processo principal ou dentro da arquitetura, dependendo de onde o
                   comparador    definido
56
57          if ajuste = '0' then  -- Verifica se n o est  em modo de ajuste
58            -- Verifica se Rel gio coincide com o Alarme 1, 2 ou 3
59            if (((saida1_Relogio = saida1_alarme1) and
60               (saida2_Relogio = saida2_alarme1) and
61               (saida3_Relogio = saida3_alarme1) and
62               (saida4_Relogio = saida4_alarme1)and
63               (saida5_Relogio = saida5_alarme1) and
64               (saida6_Relogio = saida6_alarme1)) and (clock_1_seg = '0')) then
65
66                comparador1 <= '1';  -- Se houver coincid ncia com algum alarme
67                comparador2 <= '0';
68                comparador3 <= '0';
69
70            elsif botao3 = '1' and botao4 = '1' then
71              comparador1 <= '0';  -- Se n o houver coincid ncia
72            end if;
73
74            if  (((saida1_Relogio = saida1_alarme2) and
75               (saida2_Relogio = saida2_alarme2) and
76               (saida3_Relogio = saida3_alarme2) and
77               (saida4_Relogio = saida4_alarme2)and
78               (saida5_Relogio = saida5_alarme2) and
79               (saida6_Relogio = saida6_alarme2)) and (clock_1_seg = '0')) then
80
81                comparador1 <= '0';  -- Se houver coincid ncia com algum alarme
82                comparador2 <= '1';
83                comparador3 <= '0';
84
85            elsif botao3 = '1' and botao4 = '1' then
86              comparador2 <= '0';  -- Se n o houver coincid ncia
87            end if;
88
89            if  (((saida1_Relogio = saida1_alarme3) and
```

```
90                (saida2_Relogio = saida2_alarme3) and
91                (saida3_Relogio = saida3_alarme3) and
92                (saida4_Relogio = saida4_alarme3)and
93                (saida5_Relogio = saida5_alarme3) and
94                (saida6_Relogio = saida6_alarme3)) and (clock_1_seg = '0')) then
95
96                 comparador1 <= '0';
97                 comparador2 <= '0';
98                 comparador3 <= '1';  -- Se houver coincid ncia com algum alarme
99
100
101
102             elsif botao3 = '1' and botao4 = '1' then
103               comparador3 <= '0';  -- Se n o houver coincid ncia
104             end if;
105           else
106             comparador1 <= '0';  -- Se estiver em modo de ajuste
107             comparador2 <= '0';  -- Se estiver em modo de ajuste
108             comparador3 <= '0';  -- Se estiver em modo de ajuste
109           end if;
110       end case;
111     end if;
112   end process;
113
114 end Behavioral;
```

Apêndice H Comparadores

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity comparadores is
7     port (
8        clk_buzzer1,clk_buzzer2,clk_buzzer3, clk, comparador1,comparador2,
             comparador3 : in std_logic;
9     buzzer : out std_logic
```

```
10    );
11 end comparadores;
12
13 architecture Behavioral of comparadores is
14 begin
15
16   process(clk)
17   begin
18     if comparador1 = '1' then
19       buzzer <= clk_buzzer1;
20     elsif comparador2 = '1' then
21       buzzer <= clk_buzzer2;
22     elsif comparador3 = '1' then
23       buzzer <= clk_buzzer3;
24     else
25       buzzer <= '0';
26     end if;
27   end process;
28
29 end Behavioral;
```

9 Anexos

Anexo A Interface para o controlador do LCD

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

PACKAGE lcd_vhdl_package IS

  FUNCTION to_std_logic_vector( s : string ) RETURN std_logic_vector;
  FUNCTION reverse( s : string ) RETURN string;

  COMPONENT lcd_controller IS
    PORT(
       clk        : IN  STD_LOGIC; --clock principal
       reset_n    : IN  STD_LOGIC; --ativo-baixo reinicializa o lcd
       lcd_enable : IN  STD_LOGIC; --(1) envia dados para o controlador LCD
       lcd_bus    : IN  STD_LOGIC_VECTOR(9 DOWNTO 0); --instru  o (9)rs, (8)rw e
           (7..0)char
       busy       : OUT STD_LOGIC; --feedback do controlador de LCD (1)ocupado (0)
          dispon vel
       rw, rs, e  : OUT STD_LOGIC; --leitura/escrita, instru  o/dados, habilita
          LCD
       lcd_data   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)); -- char enviado para o LCD(
          D7..D0)
   END COMPONENT;

END PACKAGE lcd_vhdl_package;

PACKAGE BODY lcd_vhdl_package IS
  --converte uma string em uma array de vetores de 8bits
```

```
25   FUNCTION to_std_logic_vector( s : string ) RETURN std_logic_vector
26   IS --variavel auxiliar para armazenamento tempor rio
27     VARIABLE r : std_logic_vector( 0 TO s'LENGTH * 8 - 1) ;
28   BEGIN
29     FOR i IN 1 TO s'HIGH LOOP --percorre todos os caracteres da string
30       --converte cada caractere em um vetor de 8bits
31       --e armazena na vari vel auxiliar em ordem crescente
32       r((i - 1) * 8  TO i * 8 - 1) := std_logic_vector( to_unsigned( character'POS
             (s(i)) , 8 ) ) ;
33     END loop ;
34   RETURN r ; --retorna a array de vetores de 8bits
35   END FUNCTION ;
36   --inverte a sequ ncia de caracteres numa string
37   FUNCTION reverse( s : string ) RETURN string
38   IS --variavel auxiliar para armazenamento tempor rio
39     VARIABLE r : string(s'HIGH DOWNTO s'LOW) ;
40   BEGIN
41     FOR i IN 1 TO s'HIGH LOOP --percorre todos os caracteres da string
42       --inverte a posi  o de cada caractere
43       --eg. 8bits r(7) := s(0) e r(0):= s(7)
44       r(s'HIGH + 1 - i) := s(i) ;
45     END LOOP ;
46   RETURN r ;
47   END FUNCTION ;
48
49 END PACKAGE BODY lcd_vhdl_package ;
```

Anexo B Controlador do LCD

```
1  LIBRARY IEEE;
2  USE IEEE.std_logic_1164.ALL;
3  ENTITY lcd_controller IS
4    GENERIC(
5      clk_freq       :  INTEGER    := 50;    --clock principal em MHz
6      display_lines  :  STD_LOGIC  := '0';   --n mero de linhas do display (0 = uma
           linha, 1 = duas linhas)
7      character_font :  STD_LOGIC  := '0';   --fonte (0 = 5x8 pontos, 1 = 5x10
         pontos)
```

```
8     display_on_off :  STD_LOGIC  := '1';   --display on/off (0 = off, 1 = on)
9     cursor         :  STD_LOGIC  := '0';   --cursor on/off (0 = off, 1 = on)
10    blink          :  STD_LOGIC  := '0';   --blink on/off (0 = off, 1 = on)
11    inc_dec        :  STD_LOGIC  := '1';   --incremento/decremento (0 = decremento
          , 1 = incremento)
12    shift          :  STD_LOGIC  := '0');  --shift on/off (0 = off, 1 = on)
13  PORT(
14    clk        : IN   STD_LOGIC; --clock principal
15    reset_n    : IN   STD_LOGIC;                    --ativo-baixo, reinicializa o
          LCD
16    lcd_enable : IN   STD_LOGIC;                    --retem os dados no
          controlador LCD
17    lcd_bus    : IN   STD_LOGIC_VECTOR(9 DOWNTO 0);  --(9) rs (8) rw (7..0) dado
          char
18    busy       : OUT  STD_LOGIC := '1';             --feedback do controlador (1)
          ocupado/(0)dispon vel
19    rw, rs, e  : OUT  STD_LOGIC;                    --leitura/escrita,
          instru  o/dado, habilita LCD ativo-alto
20    lcd_data   : OUT  STD_LOGIC_VECTOR(7 DOWNTO 0)); --sinal de dado (char) para o
          LCD
21 END lcd_controller;
22 ARCHITECTURE bhv OF lcd_controller IS
23   --Declara  o de estados da FSM
24   TYPE ESTADOS IS(ENERGIZACAO, INICIALIZACAO, PRONTO, ENVIAR);
25   SIGNAL  estado  : ESTADOS;
26 BEGIN
27   --FSM do
28   PROCESS(clk)
29     VARIABLE clk_count : INTEGER := 0; --contador para temporiza  o de eventos
30   BEGIN
31     IF rising_edge(clk) THEN
32       --reinicializa a FSM
33       IF(reset_n = '0') THEN
34         estado <= ENERGIZACAO;
35       END IF;
36       CASE estado IS
37         -- espera 50ms para garantir a energiza  o do LCD
38       WHEN ENERGIZACAO =>
39         busy <= '1';
```

```
40          IF(clk_count < (50000 * clk_freq)) THEN     --espera 50 ms
41            clk_count := clk_count + 1;
42            estado <= ENERGIZACAO;
43          ELSE                                        --energiza  o completa
44            clk_count := 0;
45            rs <= '0';
46            rw <= '0';
47            lcd_data <= "00110000"; --8-bits 1L*16 5*8 function_set
48            estado <= INICIALIZACAO;
49          END IF;
50        --sequ ncia de inicializa  o do display LCD
51        WHEN INICIALIZACAO =>
52          busy <= '1'; --LCD ocupado
53          clk_count := clk_count + 1;
54          IF(clk_count < (10 * clk_freq)) THEN        --function set
55            lcd_data <= "0011" & display_lines & character_font & "00";
56            e <= '1'; --habilita o LCD (executa o comando)
57            estado <= INICIALIZACAO;
58          ELSIF(clk_count < (60 * clk_freq)) THEN     --espera 50 us
59            lcd_data <= "00000000"; --nenhuma nova instru  o , apenas aguarda
60            e <= '0'; --desabilita o LCD
61            estado <= INICIALIZACAO;
62          ELSIF(clk_count < (70 * clk_freq)) THEN     --display on/off control
63            lcd_data <= "00001" & display_on_off & cursor & blink;
64            e <= '1'; --habilita o LCD (executa o comando)
65            estado <= INICIALIZACAO;
66          ELSIF(clk_count < (120 * clk_freq)) THEN    --espera 50 us
67            lcd_data <= "00000000";
68            e <= '0';
69            estado <= INICIALIZACAO;
70          ELSIF(clk_count < (130 * clk_freq)) THEN    --display clear
71            lcd_data <= "00000001";
72            e <= '1'; --habilita o LCD (executa o comando)
73            estado <= INICIALIZACAO;
74          ELSIF(clk_count < (2130 * clk_freq)) THEN   --wait 2 ms
75            lcd_data <= "00000000";
76            e <= '0';
77            estado <= INICIALIZACAO;
78          ELSIF(clk_count < (2140 * clk_freq)) THEN   --entry mode set
```

```
79            lcd_data <= "000001" & inc_dec & shift;
80            e <= '1'; --habilita o LCD (executa o comando)
81            estado <= INICIALIZACAO;
82          ELSIF(clk_count < (2200 * clk_freq)) THEN  --wait 60 us
83            lcd_data <= "00000000";
84            e <= '0';
85            estado <= INICIALIZACAO;
86          ELSE                                      --initialization complete
87            clk_count := 0;
88            busy <= '0';
89            estado <= PRONTO;
90          END IF;
91        --wait for the enable signal and then latch in the instruction
92        WHEN PRONTO =>
93          IF(lcd_enable = '1') THEN
94            busy <= '1';
95            rs <= lcd_bus(9);
96            rw <= lcd_bus(8);
97            lcd_data <= lcd_bus(7 DOWNTO 0);
98            clk_count := 0;
99            estado <= ENVIAR;
100         ELSE
101           busy <= '0';
102           rs <= '0';
103           rw <= '0';
104           lcd_data <= "00000000";
105           clk_count := 0;
106           estado <= PRONTO;
107         END IF;
108       --envia instru  o para o LCD
109       WHEN ENVIAR =>
110         busy <= '1'; --LCD ocupado
111         IF(clk_count < (50 * clk_freq)) THEN       --espera 50 us
112           IF(clk_count < clk_freq) THEN            --enable negativo
113             e <= '0'; --desabilita o LCD
114           ELSIF(clk_count < (14 * clk_freq)) THEN  --enable positivo em metade do
                ciclo (25us)
115             e <= '1'; --habilita o LCD (executa o comando)
```

```
116           ELSIF(clk_count < (27 * clk_freq)) THEN  --enable negativo na outra
                  metade do ciclo (25us)
117             e <= '0'; --desabilita o LCD
118           END IF;
119           clk_count := clk_count + 1;
120           estado <= ENVIAR;
121         ELSE
122           clk_count := 0;
123           estado <= PRONTO;
124         END IF;
125       END CASE;
126     END IF;
127   END PROCESS;
128 END bhv;
```

Anexo C Exibição no LCD

```
1  LIBRARY IEEE;
2  USE IEEE.std_logic_1164.all;
3  USE WORK.lcd_vhdl_package.ALL;
4  ENTITY lcd_logic IS
5    PORT(   clk       : IN  STD_LOGIC;  --clock principal
6         clk_3_seg: IN  STD_LOGIC;  --clock 3 seg
7         clk_1_seg: IN  STD_LOGIC;  --clock 1 seg
8         lcd_busy : IN  STD_LOGIC;  --feedback do controlador (1)ocupado/(0)
              dispon vel
9         botao    : IN  STD_LOGIC_VECTOR(4 DOWNTO 1); --bot es
10        lcd_e    : OUT STD_LOGIC;  --retem os dados no controlador LCD
11        lcd_bar : OUT STD_LOGIC_VECTOR(9 DOWNTO 0);  --(9) rs (8) rw (7..0) dado
              char
12        contagem_relogio1, contagem_relogio2, contagem_relogio3, contagem_relogio4,
              contagem_relogio5, contagem_relogio6 : IN std_logic_vector(3 downto 0);
13        seletor_display : IN  STD_LOGIC_VECTOR(2 DOWNTO 0);
14        relogio1,alarme1,alarme2,alarme3 : IN   STD_LOGIC
15     );
16 END lcd_logic;
17 ARCHITECTURE bhv OF lcd_logic IS
18   --Registradores
```

```
19  SIGNAL lcd_enable : STD_LOGIC;
20  SIGNAL lcd_bus    : STD_LOGIC_VECTOR(9 DOWNTO 0);
21  --Barramento de dados do display
22  SIGNAL L1 : std_logic_vector (127 DOWNTO 0):= to_std_logic_vector("   ELETRONICA
        ");--16 caracteres!!!;
23  SIGNAL L2 : std_logic_vector (127 DOWNTO 0):= to_std_logic_vector("  Digital
    UFPE  ");--16 caracteres!!!;
24  SIGNAL LR : std_logic_vector (127 DOWNTO 0);--16 caracteres!!!;
25  SIGNAL frase_Relogio : std_logic_vector (127 DOWNTO 0):= to_std_logic_vector("
        RELOGIO     ");--16 caracteres!!!;
26  SIGNAL frase_Ajuste_Relogio  : std_logic_vector (127 DOWNTO 0):=
    to_std_logic_vector("AJUSTE RELOGIO C");--16 caracteres!!!;
27  SIGNAL frase_Ajuste_Alarme1  : std_logic_vector (127 DOWNTO 0):=
    to_std_logic_vector("AJUSTE ALARME1 C");--16 caracteres!!!;
28  SIGNAL frase_Ajuste_Alarme2  : std_logic_vector (127 DOWNTO 0):=
    to_std_logic_vector("AJUSTE ALARME2 C");--16 caracteres!!!;
29  SIGNAL frase_Ajuste_Alarme3  : std_logic_vector (127 DOWNTO 0):=
    to_std_logic_vector("AJUSTE ALARME3 C");--16 caracteres!!!;
30
31
32
33
34  --constantes
35  SIGNAL frase1_1 : std_logic_vector (127 DOWNTO 0) := to_std_logic_vector("
    ELETRONICA   ");--16 caracteres!!!
36  SIGNAL frase1_2 : std_logic_vector (127 DOWNTO 0) := to_std_logic_vector("
    Digital UFPE  ");--16 caracteres!!!
37
38  SIGNAL frase2_1 : std_logic_vector (127 DOWNTO 0) := to_std_logic_vector("
    RELOGIO DIGITAL");--16 caracteres!!!
39  SIGNAL frase2_2 : std_logic_vector (127 DOWNTO 0) := to_std_logic_vector("
    PROJETO 3   ");--16 caracteres!!!
40
41  SIGNAL frase3_1 : std_logic_vector (127 DOWNTO 0) := to_std_logic_vector("
    ALYSSON    ");--16 caracteres!!!
42  SIGNAL frase3_2 : std_logic_vector (127 DOWNTO 0) := to_std_logic_vector("
    CAVALCANTE   ");--16 caracteres!!!
43
```

```
44   SIGNAL frase4_1 : std_logic_vector (127 DOWNTO 0) := to_std_logic_vector("
        FELIPE     ");--16 caracteres!!!
45   SIGNAL frase4_2 : std_logic_vector (127 DOWNTO 0) := to_std_logic_vector("
        BARROS     ");--16 caracteres!!!
46
47   SIGNAL frase5_1 : std_logic_vector (127 DOWNTO 0) := to_std_logic_vector("
        MARIA      ");--16 caracteres!!!
48   SIGNAL frase5_2 : std_logic_vector (127 DOWNTO 0) := to_std_logic_vector("
        VICTORIA    ");--16 caracteres!!!
49
50
51
52   TYPE ESTADOS IS (INICIAL, RELOGIO);
53   SIGNAL  estado  : ESTADOS := INICIAL;
54
55 BEGIN
56 --atribui  o  cont nua das sa das registradas
57  lcd_e <= lcd_enable;
58  lcd_bar <= lcd_bus;
59
60
61
62  PROCESS(clk) -- Mudar as frases do estado inicial
63   VARIABLE mudanca  :  INTEGER RANGE 0 TO 4 := 0;
64  BEGIN
65
66     IF (rising_edge(clk_3_seg)) AND (estado = INICIAL) THEN
67       CASE mudanca IS
68          WHEN 0 => L1 <= frase1_1; L2 <= frase1_2;
69          WHEN 1 => L1 <= frase2_1; L2 <= frase2_2;
70          WHEN 2 => L1 <= frase3_1; L2 <= frase3_2;
71          WHEN 3 => L1 <= frase4_1; L2 <= frase4_2;
72          WHEN 4 => L1 <= frase5_1; L2 <= frase5_2;
73       END CASE;
74       IF (mudanca < 4) THEN
75         mudanca := mudanca + 1; --incrementa o estado
76       ELSE mudanca := 0; --reinicia o estado
77
78       END IF;
```

```
79      END IF;
80
81
82
83
84  END PROCESS;
85
86
87
88
89   PROCESS(clk)
90    VARIABLE char  :  INTEGER RANGE 0 TO 34 := 0; --6 bits
91   BEGIN
92    IF rising_edge(clk) THEN
93     IF (botao(1) = '1') THEN
94      estado <= RELOGIO;
95     END IF;
96    CASE estado IS
97      WHEN INICIAL => -- 1   ESTADO DA M QUINA
98          IF (lcd_busy = '0' AND lcd_enable = '0') THEN
99            lcd_enable <= '1'; --habilita o LCD
100           IF (char < 34) THEN
101             char := char + 1; --incrementa o estado
102           ELSE char := 0; --reinicia o estado
103           END IF;
104           CASE char IS --verifica o estado atual
105             WHEN 0  => lcd_bus <= "00" & "10000000"; --inst. linha 1
106             WHEN 1  => lcd_bus <= "10" & L1(127 DOWNTO 120); --prim. char da linha
                    1
107             WHEN 2  => lcd_bus <= "10" & L1(119 DOWNTO 112);
108             WHEN 3  => lcd_bus <= "10" & L1(111 DOWNTO 104);
109             WHEN 4  => lcd_bus <= "10" & L1(103 DOWNTO 96);
110             WHEN 5  => lcd_bus <= "10" & L1(95 DOWNTO 88);
111             WHEN 6  => lcd_bus <= "10" & L1(87 DOWNTO 80);
112             WHEN 7  => lcd_bus <= "10" & L1(79 DOWNTO 72);
113             WHEN 8  => lcd_bus <= "10" & L1(71 DOWNTO 64);
114             WHEN 9  => lcd_bus <= "10" & L1(63 DOWNTO 56);
115             WHEN 10 => lcd_bus <= "10" & L1(55 DOWNTO 48);
116             WHEN 11 => lcd_bus <= "10" & L1(47 DOWNTO 40);
```

```
117             WHEN 12 => lcd_bus <= "10" & L1(39 DOWNTO 32);
118             WHEN 13 => lcd_bus <= "10" & L1(31 DOWNTO 24);
119             WHEN 14 => lcd_bus <= "10" & L1(23 DOWNTO 16);
120             WHEN 15 => lcd_bus <= "10" & L1(15 DOWNTO 8);
121             WHEN 16 => lcd_bus <= "10" & L1(7 DOWNTO 0); --ult char da linha 1
122
123             WHEN 17 => lcd_bus <= "00" & "11000000"; --inst. linha 2
124             WHEN 18 => lcd_bus <= "10" & L2(127 DOWNTO 120); --prim. char da linha
                    2
125             WHEN 19 => lcd_bus <= "10" & L2(119 DOWNTO 112);
126             WHEN 20 => lcd_bus <= "10" & L2(111 DOWNTO 104);
127             WHEN 21 => lcd_bus <= "10" & L2(103 DOWNTO 96);
128             WHEN 22 => lcd_bus <= "10" & L2(95 DOWNTO 88);
129             WHEN 23 => lcd_bus <= "10" & L2(87 DOWNTO 80);
130             WHEN 24 => lcd_bus <= "10" & L2(79 DOWNTO 72);
131             WHEN 25 => lcd_bus <= "10" & L2(71 DOWNTO 64);
132             WHEN 26 => lcd_bus <= "10" & L2(63 DOWNTO 56);
133             WHEN 27 => lcd_bus <= "10" & L2(55 DOWNTO 48);
134             WHEN 28 => lcd_bus <= "10" & L2(47 DOWNTO 40);
135             WHEN 29 => lcd_bus <= "10" & L2(39 DOWNTO 32);
136             WHEN 30 => lcd_bus <= "10" & L2(31 DOWNTO 24);
137             WHEN 31 => lcd_bus <= "10" & L2(23 DOWNTO 16);
138             WHEN 32 => lcd_bus <= "10" & L2(15 DOWNTO 8);
139             WHEN 33 => lcd_bus <= "10" & L2(7 DOWNTO 0);--ult. char da linha 2

140             WHEN OTHERS => lcd_enable <= '0'; --desabilita o LCD
141           END CASE;
142
143         ELSE lcd_enable <= '0'; --desabilita o LCD
144         END IF;
145
146     WHEN RELOGIO =>   -- 2   ESTADO DA M QUINA
147         IF (lcd_busy = '0' AND lcd_enable = '0') THEN
148             lcd_enable <= '1'; --habilita o LCD
149             IF (char < 34) THEN
150               char := char + 1; --incrementa o estado
151             ELSE char := 0; --reinicia o estado
152             END IF;
153             CASE char IS --verifica o estado atual
```

```
154                 WHEN 0  => lcd_bus <= "00" & "10000000"; --inst. linha 1
155                 WHEN 1  => lcd_bus <= "10" & "00100000";--prim. char da linha 1
156                 WHEN 2  => lcd_bus <= "10" & "00100000";
157                 WHEN 3  => lcd_bus <= "10" & "00100000";
158                 WHEN 4  => lcd_bus <= "10" & "00100000";
159                 WHEN 5  =>
160                         if ((clk_1_seg = '1') and (botao(1) = '1') and (
                                seletor_display = "101") ) or (botao(1) = '0' or (
                                seletor_display /= "101")) then
161                           lcd_bus <= "10" & "0011" & contagem_relogio6;  -- Hora
                                mais significativo
162                         else
163                           lcd_bus <= "10" & "00100000";
164                         end if;
165                 WHEN 6  => lcd_bus <= "10" & "0011" & contagem_relogio5;  -- Hora
                    menos significativo
166                         if ((clk_1_seg = '1') and (botao(1) = '1') and (
                                seletor_display = "100") ) or (botao(1) = '0' or (
                                seletor_display /= "100")) then
167                           lcd_bus <= "10" & "0011" & contagem_relogio5;  -- Hora
                                menos significativo
168                         else
169                           lcd_bus <= "10" & "00100000";
170                         end if;
171
172                 WHEN 7  => lcd_bus <= "10" & "00111010";
173                 WHEN 8  =>
174                         if ((clk_1_seg = '1') and (botao(1) = '1') and (
                                seletor_display = "011") ) or (botao(1) = '0' or (
                                seletor_display /= "011")) then
175                           lcd_bus <= "10" & "0011" & contagem_relogio4;  -- minuto
                                mais significativo
176                         else
177                           lcd_bus <= "10" & "00100000";
178                         end if;
179
180                 WHEN 9  =>
181                         if ((clk_1_seg = '1') and (botao(1) = '1') and (
                                seletor_display = "010") ) or (botao(1) = '0' or (
```

```
                        seletor_display /= "010")) then
182                   lcd_bus <= "10" & "0011" & contagem_relogio3;  -- minuto
                          menos significativo
183                 else
184                   lcd_bus <= "10" & "00100000";
185                 end if;
186             WHEN 10 => lcd_bus <= "10" & "00111010";
187             WHEN 11 =>
188                 if ((clk_1_seg = '1') and (botao(1) = '1') and (
                        seletor_display = "001") ) or (botao(1) = '0' or (
                        seletor_display /= "001")) then
189                   lcd_bus <= "10" & "0011" & contagem_relogio2;  -- segundo
                          mais significativo
190                 else
191                   lcd_bus <= "10" & "00100000";
192                 end if;
193
194
195             WHEN 12 =>
196                 if ((clk_1_seg = '1') and (botao(1) = '1') and (
                        seletor_display = "000") ) or (botao(1) = '0' or (
                        seletor_display /= "000") ) then
197                   lcd_bus <= "10" & "0011" & contagem_relogio1;  -- Segundo
                          menos significativo
198                 else
199                   lcd_bus <= "10" & "00100000";
200                 end if;
201             WHEN 13 => lcd_bus <= "10" & "00100000";
202             WHEN 14 => lcd_bus <= "10" & "00100000";
203             WHEN 15 => lcd_bus <= "10" & "00100000";
204             WHEN 16 => lcd_bus <= "10" & "00100000"; --ult char da linha 1
205
206
207             CASE botao(1) IS
208             WHEN '1' =>
209               IF   relogio1 = '1'  THEN LR <= frase_Ajuste_Relogio;
210               ELSIF alarme1 = '1'  THEN LR <= frase_Ajuste_Alarme1;
211               ELSIF alarme2 = '1'  THEN LR <= frase_Ajuste_Alarme2;
212               ELSIF alarme3 = '1'  THEN LR <= frase_Ajuste_Alarme3;
```

```
213                 END IF;
214               WHEN '0' =>
215                 LR <= frase_Relogio;
216               END CASE;
217
218
219
220               WHEN 17 => lcd_bus <= "00" & "11000000"; --inst. linha 2
221               WHEN 18 => lcd_bus <= "10" & LR(127 DOWNTO 120); --prim. char da
                      linha 2
222               WHEN 19 => lcd_bus <= "10" & LR(119 DOWNTO 112);
223               WHEN 20 => lcd_bus <= "10" & LR(111 DOWNTO 104);
224               WHEN 21 => lcd_bus <= "10" & LR(103 DOWNTO 96);
225               WHEN 22 => lcd_bus <= "10" & LR(95 DOWNTO 88);
226               WHEN 23 => lcd_bus <= "10" & LR(87 DOWNTO 80);
227               WHEN 24 => lcd_bus <= "10" & LR(79 DOWNTO 72);
228               WHEN 25 => lcd_bus <= "10" & LR(71 DOWNTO 64);
229               WHEN 26 => lcd_bus <= "10" & LR(63 DOWNTO 56);
230               WHEN 27 => lcd_bus <= "10" & LR(55 DOWNTO 48);
231               WHEN 28 => lcd_bus <= "10" & LR(47 DOWNTO 40);
232               WHEN 29 => lcd_bus <= "10" & LR(39 DOWNTO 32);
233               WHEN 30 => lcd_bus <= "10" & LR(31 DOWNTO 24);
234               WHEN 31 => lcd_bus <= "10" & LR(23 DOWNTO 16);
235               WHEN 32 => lcd_bus <= "10" & LR(15 DOWNTO 8);
236               WHEN 33 => lcd_bus <= "10" & LR(7 DOWNTO 0);--ult. char da linha 2

237               WHEN OTHERS => lcd_enable <= '0'; --desabilita o LCD
238             END CASE;
239           ELSE
240             lcd_enable <= '0'; --desabilita o LCD
241           END IF;
242
243   END CASE;
244   END IF;
245  END PROCESS;
246 END ARCHITECTURE;
```