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 4

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 alarmes utilizando apenas a linguagem de descrição de Hardware Verilog. 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 principal deste projeto foi projetar e implementar um relógio digital capaz de exibir horas, minutos e segundos em um Display de Cristal Líquido (LCD), permitindo a configuração e acionamento de três alarmes com avisos sonoros musicais, além de exibir a temperatura ambiente. O controle das funcionalidades do relógio é realizado por meio de um controle remoto.

Os objetivos específicos incluem o desenvolvimento de um contador de horas, minutos e segundos; a criação de um circuito de controle que permita a programação tanto do relógio quanto dos alarmes via controle remoto; a integração de todos os componentes em um sistema funcional capaz de acionar os alarmes nos horários programados; a adição de melodias aos alarmes; e a correta exibição das informações no LCD.

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

O desenvolvimento do projeto foi dividido em várias etapas, que serão apresentadas em tópicos neste capítulo, visando facilitar o entendimento do processo.

É importante destacar que o FPGA utilizado no projeto possui entradas e saídas invertidas, ou seja, suas entradas e saídas são ativadas no estado lógico 0 e desativadas no estado lógico 1. Essa característica impactou diretamente a configuração das saídas dos LEDs e displays, exigindo a inversão lógica dessas saídas dentro dos códigos desenvolvidos.

2.1 Linguagem utilizada

Para o desenvolvimento deste projeto, foi utilizada a linguagem de descrição de hardware Verilog, amplamente empregada na área de projetos digitais e síntese de circuitos integrados. Verilog é uma linguagem robusta e eficiente, projetada para descrever, simular e sintetizar sistemas de hardware em níveis que variam de comportamental a estrutural.

Uma das principais vantagens de Verilog é sua sintaxe concisa e semelhante à das linguagens de programação tradicionais, como C, o que facilita o aprendizado e a compreensão por parte dos desenvolvedores. Além disso, Verilog é extremamente flexível, permitindo descrever circuitos digitais tanto em nível de abstração mais alto, com foco em funcionalidades lógicas, quanto em nível mais baixo, detalhando as interconexões entre portas lógicas.

Entre os pontos fortes da linguagem, destacam-se:

- Modularidade: Verilog permite a criação de módulos reutilizáveis e hierárquicos, o que simplifica o desenvolvimento de sistemas complexos, como o relógio digital deste projeto.
- Suporte a simulação e síntese: A linguagem é compatível com ferramentas de simulação de hardware, permitindo testar e validar o comportamento dos circuitos antes de sua implementação física. Além disso, o código Verilog pode ser sintetizado diretamente em FPGA ou

ASIC, o que garante sua aplicabilidade em ambientes industriais.

- Controle temporal preciso: Verilog possibilita a descrição de eventos temporais com precisão, o que é essencial para sistemas que dependem de temporizações rigorosas, como contadores de tempo e acionamento de alarmes.

Esses fatores tornam Verilog uma escolha adequada para projetos de sistemas embarcados complexos e tempo-real, como o relógio digital desenvolvido neste trabalho.

2.2 Divisor de clock

Em sistemas digitais, especialmente aqueles implementados em FPGA ou ASIC, o clock é um sinal essencial que sincroniza todas as operações do circuito. No entanto, a frequência do clock fornecida pelo hardware muitas vezes é muito alta para determinados módulos ou operações, como contadores, temporizadores e interfaces de display. Para resolver isso, é comum utilizar um divisor de clock.

Um divisor de clock é um circuito responsável por reduzir a frequência do sinal de clock original, gerando clocks mais lentos, ajustados para a operação de certos módulos. No caso deste projeto, em que o relógio digital exibe segundos, minutos e horas, o divisor de clock é fundamental para gerar um sinal de 1 segundo a partir de um clock de alta frequência, como 50 MHz, utilizado no FPGA.

2.2.1 Funcionalidade

A principal função do divisor de clock é dividir a frequência do sinal de clock de entrada. Por exemplo, se o clock fornecido ao sistema é de 50 MHz (50 milhões de ciclos por segundo), e é necessário um sinal com um período de 1 segundo, o divisor de clock precisa contar 50 milhões de ciclos do clock de entrada e gerar um pulso de saída após esse tempo.

Isso garante que o contador de horas, minutos e segundos, bem como outras operações que dependem de intervalos de tempo, funcionem de maneira precisa, com base em um clock que atenda às suas necessidades temporais. Sem o divisor de clock, o sistema operaria em uma velocidade muito alta, impossibilitando o controle adequado do tempo e o acionamento dos alarmes na hora correta.

2.2.2 Importância

Os divisores de clock são essenciais em projetos de sistemas embarcados e digitais, pois permitem a adaptação da frequência do clock para diversos módulos e dispositivos que operam em diferentes velocidades. No caso do relógio digital deste projeto, o divisor de clock é crucial para que as operações temporais sejam sincronizadas corretamente, garantindo a exibição precisa do tempo e a programação dos alarmes.

Além disso, o uso de divisores de clock melhora a eficiência do projeto, pois evita a necessidade de operar todos os módulos com um clock de alta frequência, o que pode aumentar o consumo de energia e a complexidade do design.

2.2.3 Estrutura

O divisor criado, contido do Apêndice A, 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 (registrador) 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.

2.2.4 Utilizações

Foram utilizados dois divisores de clock no projeto. O primeiro gera um clock de um segundo (Figura 2.1) e, o segundo, um clock de meio segundo (Figura 2.2), suas funcionalidade serão evidenciadas nos próximos capítulos.

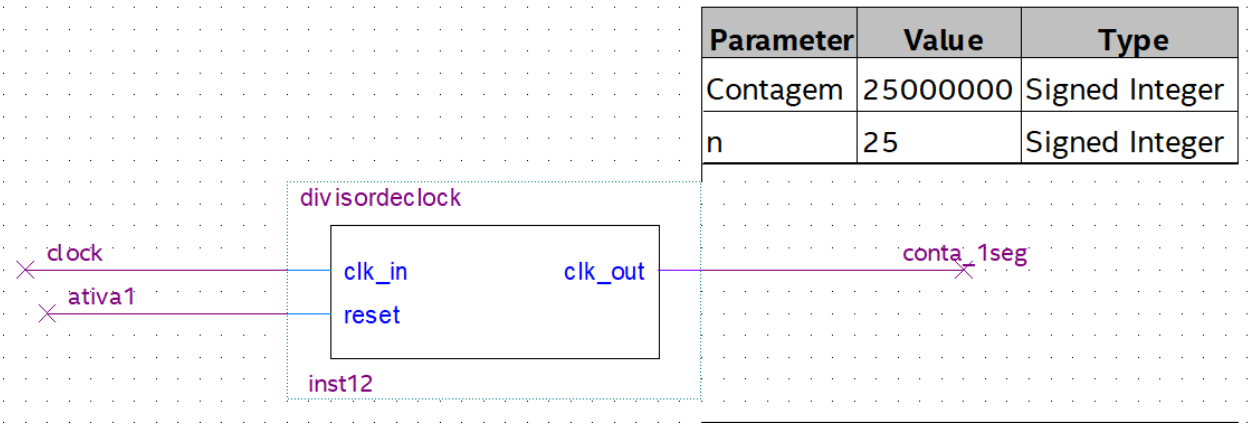


Figura 2.1: Divisor de clock: contador de um segundo.

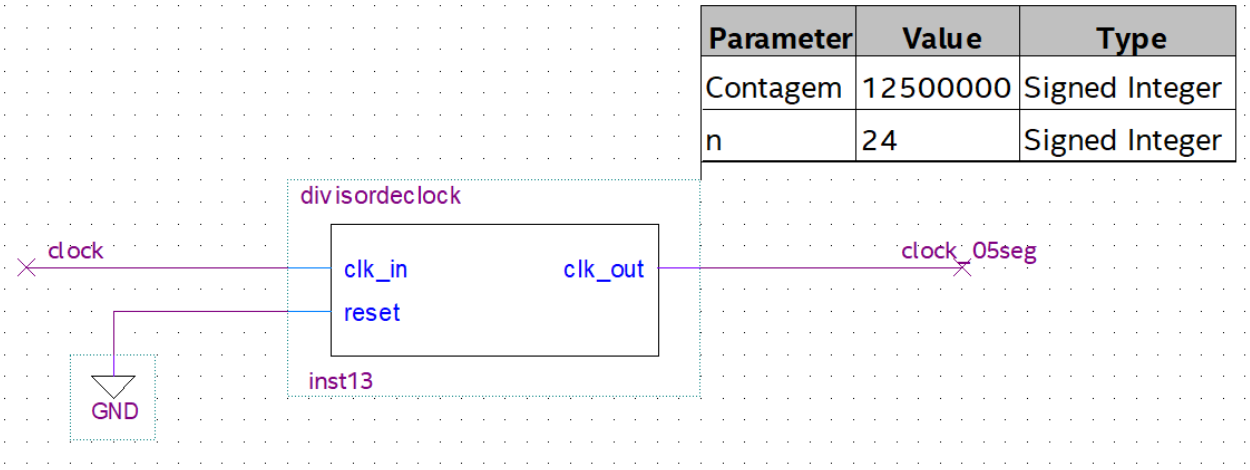


Figura 2.2: Divisor de clock: contador de meio segundo.

2.3 Contador de 4 bits

O módulo disponível no Apêndice B implementa um contador de 4 bits configurável que pode operar em modo crescente ou decrescente. Ele recebe como entradas o sinal de clock, um sinal de controle que define a direção da contagem (crescente ou decrescente), um sinal para ativar o reset e o valor máximo de contagem. A saída é representada por um sinal que contém os 4 bits da contagem em binário, além de um sinal de controle de reset, que é ativado quando o contador atinge seu valor máximo ou mínimo, dependendo do modo de operação.

A lógica do contador é baseada em um bloco sensível à borda de subida do clock. No modo decrescente, o contador diminui seu valor a cada ciclo de clock, e quando atinge o valor mínimo (sempre zero), ele é resetado para o valor máximo (definido na entrada). O sinal reset é ativado nesse ponto para indicar que o contador foi reinicializado. No modo crescente, o comportamento é inverso: o contador aumenta a cada ciclo de clock e, ao atingir o valor máximo, ele é resetado para o valor mínimo (zero), com o sinal de reset sendo ativado nesse momento.

Esse contador é fundamental para controlar ciclos de temporização e contagem em várias partes do sistema, como no relógio digital, onde diferentes unidades de tempo (segundos, minutos, horas) precisam de contagem precisa e controlada. A flexibilidade oferecida pelo modo crescente/decrescente e a parametrização do valor máximo tornam o contador adequado para diversas aplicações dentro do projeto.

2.4 Relógio

O relógio, descrito no Apêndice C, utiliza instâncias do módulo de contador de 4 bits, apresentado no capítulo anterior, para gerenciar cada unidade de tempo — segundos, minutos e horas. Esses contadores podem operar tanto em modo crescente quanto decrescente, dependendo do sinal de controle. Cada contador recebe um sinal de reset e um valor máximo, que define o momento em que o contador deve ser reinicializado e o próximo contador deve ser incrementado.

O código também recebe um seletor para o ajuste das unidades de tempo, variando entre os dígitos conforme sua ativação. Para cada unidade de tempo, o valor máximo é pré-definido: "9"para as unidades de segundos e minutos, "5"para as dezenas de segundos e minutos, "9"ou "3"para as unidades de horas e "2"para as dezenas de horas. Quando o contador atinge seu valor máximo, ele é resetado, e o contador da próxima unidade é incrementado.

A variável 'relogio-completo' é continuamente atualizada com os valores de horas, minutos e segundos, concatenando as unidades e dezenas em um único vetor de 24 bits. Isso facilita a exibição completa do horário em sistemas externos. A utilização do bloco do relógio é demonstrada na Figura 2.3.

Figura 2.3: Bloco esquemático do relógio.

2.5 Alarme

O código apresentado no Apêndice D implementa a lógica de um sistema de alarme digital. A entidade possui entradas para o clock, sinal de habilitação que determina o modo de contagem (crescente ou decrescente), e botões de controle. As saídas representam o tempo (segundos, minutos e horas) em formato de 4 bits.

A arquitetura utiliza seis contadores de 4 bits para representar os componentes do alarme, seguindo uma abordagem semelhante à usada no módulo do relógio. A lógica de controle monitora o estado de um seletor de ajuste, que determina qual unidade de tempo será modificada. Quando o seletor está ativo, o incremento ou decremento de cada dígito é controlado pelo clock de alteração, enquanto os demais contadores permanecem estabilizados. O seletor de display é responsável por alternar entre as unidades de tempo (segundos, minutos e horas), permitindo o ajuste específico de cada um.

Além disso, o código implementa condições específicas para resetar os contadores quando atingem o valor máximo. Por exemplo, quando a unidade das horas atinge "2"e a dezena das horas atinge "5", ocorre um *carry* para a unidade das horas, semelhante à lógica do código do relógio. Essa abordagem garante que o alarme funcione corretamente, incrementando ou decrementando as unidades de tempo conforme necessário.

A variável ‘alarme-completo‘ é continuamente atualizada com os valores de horas, minutos e

segundos, concatenando as unidades e dezenas em um único vetor de 24 bits. Isso facilita a utilização em códigos subsequentes que processam as saídas do alarme e integram as funções de exibição e controle. A utilização destes blocos é demonstrada na Figura 2.4.

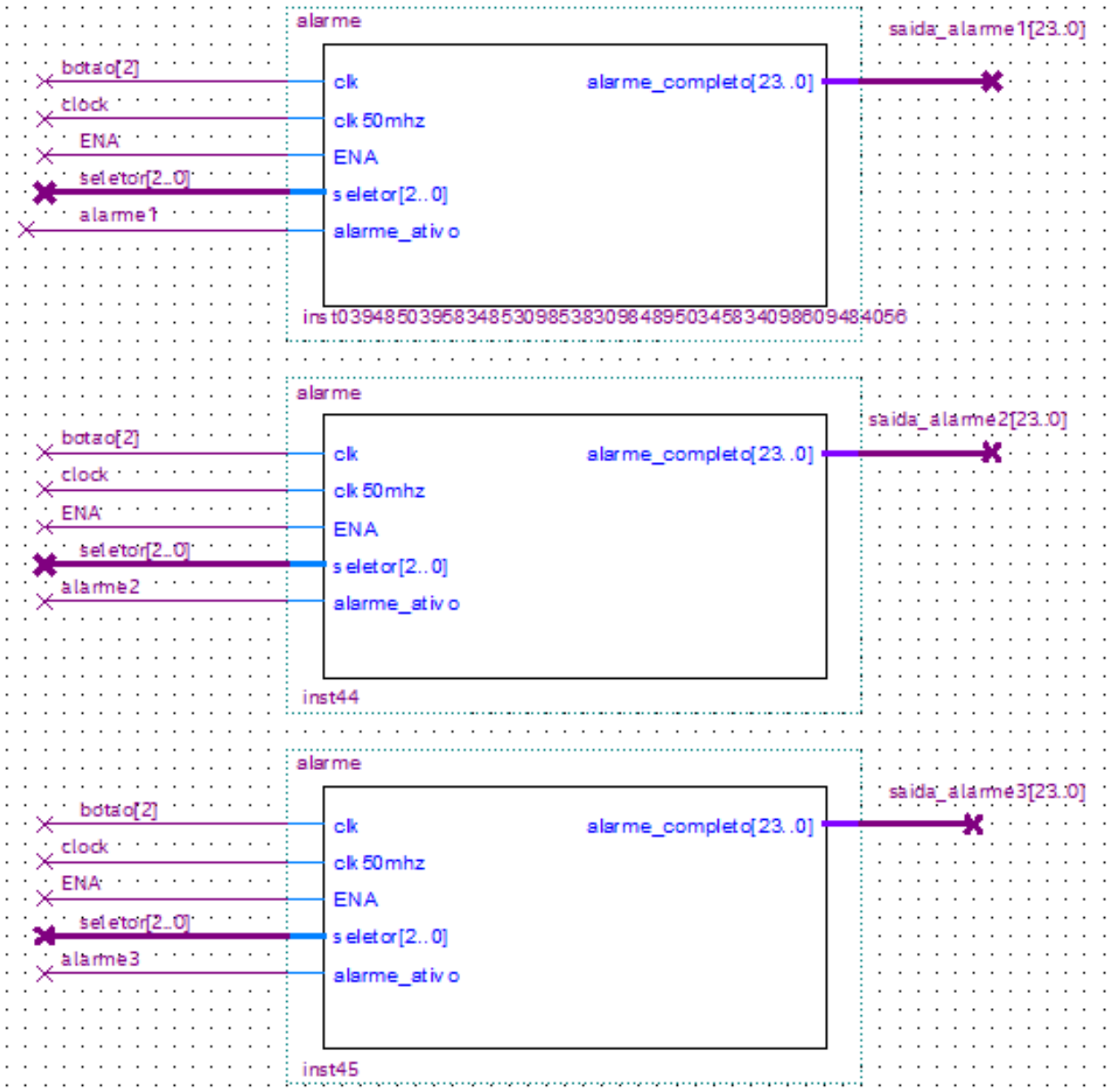


Figura 2.4: Bloco esquemático dos alarmes.

2.6 Ajustes

Todos os ajustes e interações com o relógio digital são feitas através de um controle remoto infravermelho, explicado posteriormente. Internamente, para ajustar as horas do relógio e dos alarmes, bem como alternar entre esses modos, é necessário implementar uma função de seleção. Para isso, foram criados dois seletores, compostos por contadores de módulo 8 e módulo 4, conforme descrito nos Apêndices E e F, respectivamente.

2.6.1 Seletor display

Na primeira função (Apêndice E), o chip ilustrado na Figura 2.5 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 no ajuste dos alarmes.

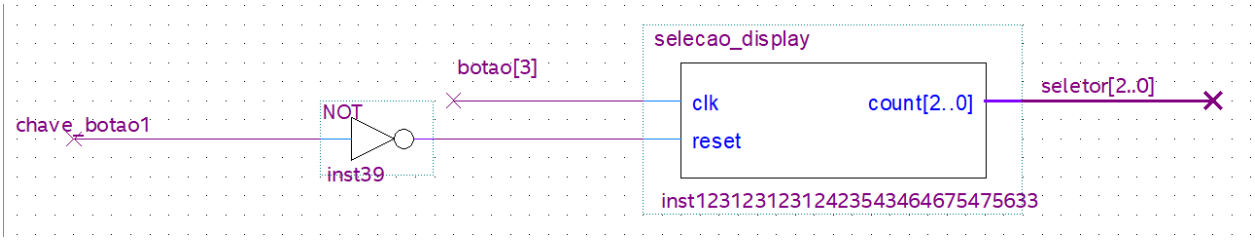


Figura 2.5: Seletor do dígido a ser alterado.

2.6.2 Seletor função

Na segunda função (Apêndice F), o chip mostrado na Figura 2.6 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 é utilizado em dois contextos: no código que seleciona o ajuste a ser realizado e no código de exibição do LCD, que indica o modo disponível para alteração.

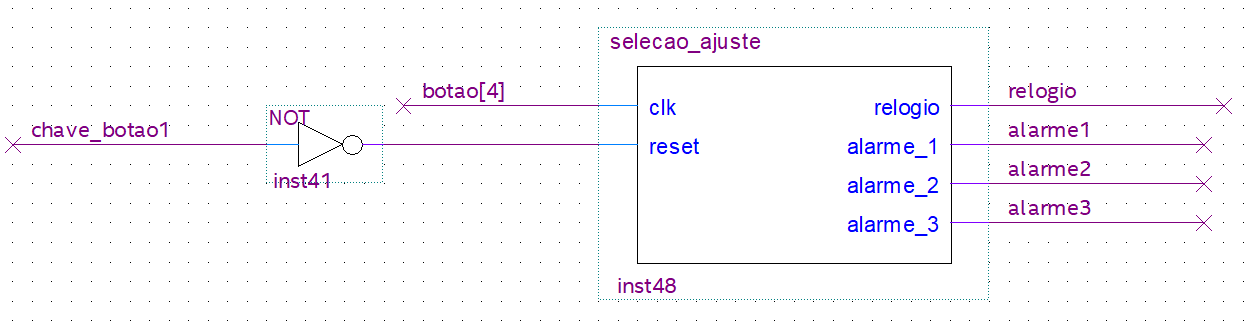
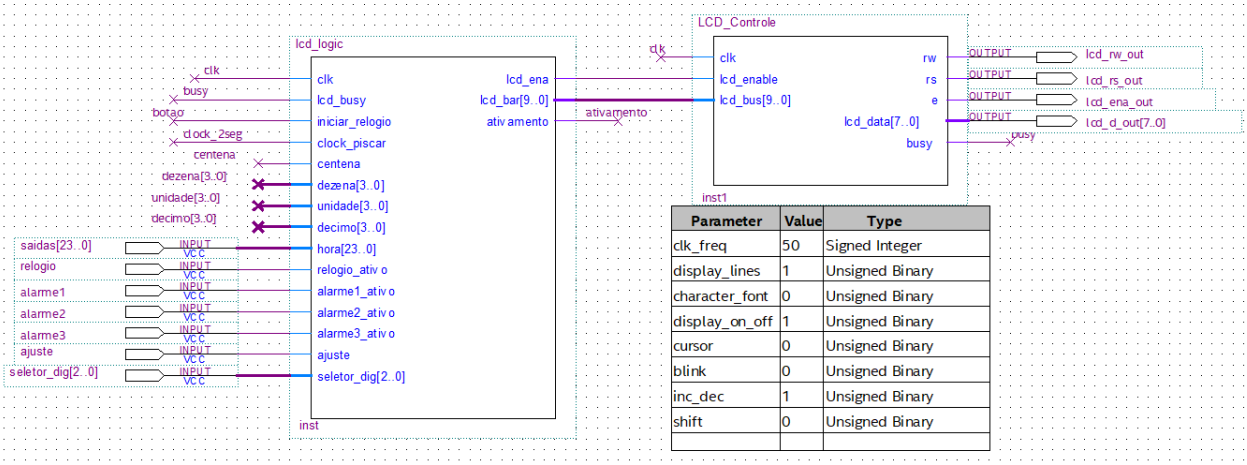


Figura 2.6: Seletor do modo a ser ajustado.

2.7 Display de Cristal Líquido

As interfaces do LCD (Controlador e Lógica) foram integradas em dois blocos esquemáticos, conforme ilustrado na Figura 2.7.



Parameter	Value	Type
clk_freq	50	Signed Integer
display_lines	1	Unsigned Binary
character_font	0	Unsigned Binary
display_on_off	1	Unsigned Binary
cursor	0	Unsigned Binary
blink	0	Unsigned Binary
inc_dec	1	Unsigned Binary
shift	0	Unsigned Binary

Figura 2.7: Estrutura interna do LCD.

Esses blocos, explicados a seguir, representam as duas principais aplicações do controle do display e foram inseridos no arquivo principal do projeto. A integração desses elementos no sistema completo pode ser observada na Figura 2.8.

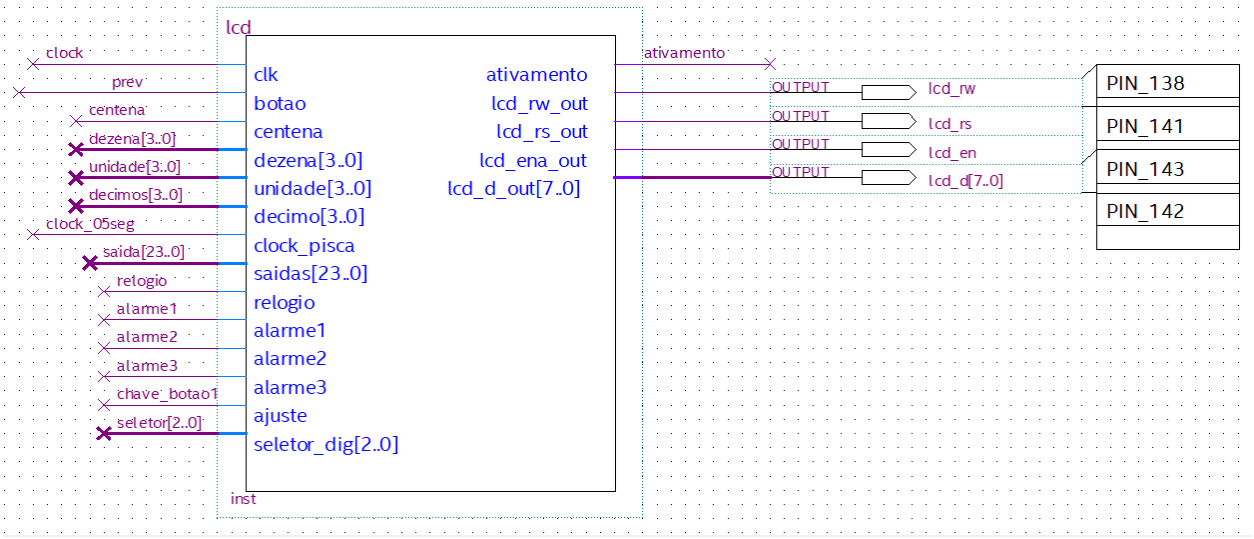


Figura 2.8: Estrutura externa do LCD.

2.8 Gerenciador de saídas

O módulo descrito no Apêndice G desempenha duas funções essenciais: a seleção do horário a ser exibido no sistema e a comparação dos horários do relógio com os alarmes programados.

O bloco de multiplexação permite escolher entre quatro entradas de horário: o horário atual do relógio e os horários programados para três alarmes distintos. Dependendo dos sinais de controle ("relogio", "alarme1", "alarme2", "alarme3"), o multiplexador ajusta a saída ("horario-saida"), exibindo o horário correspondente. A cada ciclo de clock, o multiplexador identifica qual sinal de controle está ativo e atualiza a saída para exibir o horário correto, alternando entre o relógio e os alarmes conforme o estado do sistema.

Além disso, o módulo também implementa um sistema de comparação, que verifica se o horário atual do relógio coincide com algum dos horários dos alarmes. A cada ciclo de clock, o horário do relógio é comparado com os três horários dos alarmes. Caso haja coincidência, o comparador correspondente é ativado: se o horário do relógio coincidir com o primeiro alarme, o ‘comparador1‘ será ativado, enquanto os outros comparadores (‘comparador2‘ e ‘comparador3‘) permanecerão desativados. O mesmo processo se aplica para os alarmes 2 e 3. Dessa forma, o sistema identifica qual alarme foi acionado.

Além disso, o módulo inclui uma funcionalidade de pausa, ativada por um botão. Quando o bo-

tão de pausa é pressionado, todos os comparadores são desativados, interrompendo temporariamente a verificação dos alarmes.

Em resumo, o código garante uma seleção eficiente entre os horários do relógio e dos alarmes, além de realizar comparações precisas entre os tempos. Isso assegura o funcionamento integrado e confiável do sistema de alarmes e do relógio digital, conforme ilustrado na Figura 2.9.

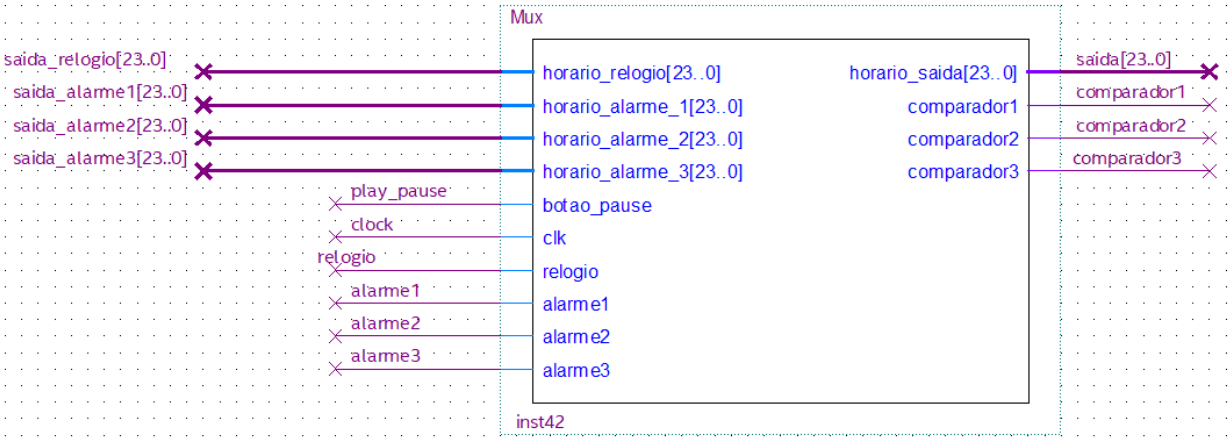


Figura 2.9: Gerenciador das saídas (multiplexador).

2.8.1 Controlador do LCD

O código descrito no Apêndice H é responsável por gerenciar a comunicação entre um sistema digital e um display LCD, utilizando uma máquina de estados finitos (FSM) para controlar as operações de inicialização e envio de dados/comandos. O módulo recebe parâmetros configuráveis, como a frequência do clock, o número de linhas do display, o tipo de fonte e configurações adicionais, como o estado do cursor, o modo de incremento/decremento e o comportamento de shift. Esses parâmetros tornam o controle do display LCD flexível, permitindo sua adaptação a diferentes tipos de displays.

O sinal de clock é utilizado para sincronizar todas as operações, enquanto o sinal de habilitação controla quando o LCD pode receber novos dados ou comandos. Além disso, uma entrada de dados carrega as informações que serão enviadas ao display. Os estados da máquina são os seguintes:

- **Energização:** A FSM inicia no estado de energização, em que o sistema aguarda 50 ms para garantir que o LCD esteja devidamente alimentado.

- **Inicialização:** Em seguida, entra na fase de inicialização, durante a qual diversas instruções de configuração são enviadas ao display. Essas instruções incluem a definição do modo de operação (como a escolha entre 8 bits, número de linhas e tipo de fonte), ativação do display, controle do cursor e limpeza da tela. Cada instrução possui um tempo de execução específico, garantindo que o display tenha tempo suficiente para processá-las corretamente.

- **Pronto:** Após a inicialização, o módulo entra no estado "Pronto", aguardando o sinal de habilitação para enviar dados ou comandos ao LCD. Quando habilitado, o barramento de dados é usado para determinar se o conteúdo a ser enviado é um comando ou um dado, de acordo com o bit mais significativo (‘rs‘) e o sinal de leitura/escrita (‘rw‘). O dado ou comando é então carregado no barramento de dados do display e o sistema avança para o próximo estado.

- **Enviar:** No último estado, o módulo gera um pulso de habilitação, necessário para que o LCD execute o comando ou exiba os dados. Um temporizador controla precisamente o tempo de ativação e desativação desse pulso, garantindo o funcionamento adequado do display. Após a conclusão da operação, o sistema retorna ao estado "Pronto", aguardando o próximo comando ou dado.

Durante todo o processo, o sinal ‘busy‘ é utilizado para indicar se o LCD está ocupado processando um comando ou disponível para novas instruções, assegurando o correto sincronismo entre o sistema e o display.

2.8.2 Lógica do LCD

O módulo descrito no Apêndice I controla a exibição de informações em um display LCD por meio de uma máquina de estados finitos (FSM) e é sincronizado com um sinal de clock. Ele é responsável por gerenciar a exibição de mensagens e valores no LCD, dependendo do estado atual do sistema e das entradas fornecidas pelo controle remoto ou botões. A lógica implementada também interage com um botão para alternar entre os estados, até que o sistema atinja o estado final, onde o relógio e a temperatura são exibidos continuamente.

A FSM possui quatro estados principais. Nos três primeiros, o LCD exibe mensagens iniciais ou informações pré-definidas. A transição entre esses estados ocorre ao pressionar um botão. No último estado, o display exibe os valores atualizados do relógio digital e da temperatura ambiente.

Para cada estado, um contador é utilizado para controlar a sequência de caracteres que será exibida no display LCD. Cada caractere é representado por seu valor ASCII, e a cada ciclo de clock, o contador incrementa, enviando o próximo caractere para o LCD, até que a mensagem esteja completamente exibida.

O módulo ativa o LCD somente quando este está pronto para receber dados, o que é indicado por ele não estar ocupado. O barramento ‘lcd-bus‘ carrega os comandos e dados a serem enviados ao display, como instruções para movimentação do cursor ou para escrever textos. Essa abordagem garante que as informações sejam exibidas corretamente, com o LCD recebendo e processando os dados de forma ordenada e sincronizada.

- **Primeiro estado:** Exibe a mensagem "Eletronica"na primeira linha e "Digital"na segunda linha do LCD.
- **Segundo estado:** Exibe a mensagem "Projeto 4"na primeira linha e "Verilog"na segunda linha.
- **Terceiro estado:** Exibe a mensagem "Alunos: Alysson,"na primeira linha e "Felipe, Victoria"na segunda linha.
- **Quarto estado:** Exibe a hora atual ou os alarmes, representados pelo vetor ‘hora‘, na primeira linha, e a temperatura na segunda linha.

2.9 Controle remoto

O módulo descrito no Apêndice J implementa o controle de um sistema digital com entrada de sinal de um controle remoto infravermelho (IR). O código utiliza um clock de 50 MHz para sincronizar as operações e controla um display BCD de 7 segmentos, exibindo o valor correspondente à tecla pressionada no controle remoto. O sistema recebe, como entradas, o sinal de reset e os dados do receptor IR, e possui saídas que controlam qual dos quatro displays BCD de 7 segmentos está ativo, além dos dados enviados para o display, representando a tecla pressionada.

2.9.1 Estrutura

O sistema processa 32 bits de dados recebidos do controle remoto, que incluem o endereço e o comando. A arquitetura interna utiliza registradores para armazenar esses valores, enquanto uma máquina de estados finitos (FSM) controla a recepção e o processamento dos dados.

Sincronização dos Dados de IR

O código começa sincronizando os sinais de entrada IR para garantir a estabilidade dos dados. Três registradores são utilizados para "amortecer"o valor do IR em estágios consecutivos, permitindo a detecção confiável das bordas de subida e descida. Essas transições são capturadas pelos sinais "*irda-negedge*"e "*irda-posedge*", que indicam, respectivamente, as bordas de descida e de subida do sinal IR.

Contagem e Temporização

Dois contadores são usados para medir a duração dos pulsos do sinal IR: um contador de 11 bits divide o clock por 1750, fornecendo uma base de tempo para a detecção dos sinais IR. Um segundo contador, de 9 bits, monitora os ciclos de alta e baixa do sinal IR.

Esses contadores permitem identificar com precisão os tempos característicos de cada pulso, como o pulso de 900 µs (que indica o início da comunicação), o pulso de 450 µs (usado durante a transmissão de dados), e os tempos dos sinais lógicos "1"e "0", que têm duração total de 2,25 ms e 1,125 ms, respectivamente.

Máquina de Estados Finitos (FSM)

A FSM implementada controla o fluxo de recepção dos dados. Ela é composta por quatro estados principais:

- **Ocioso (IDLE):** Estado inicial em que o sistema aguarda uma transição no sinal IR.
- **Atraso de 900 µs:** Estado que aguarda 900 µs para verificar se a transição inicial do sinal IR é válida.
- **Atraso de 450 µs:** Estado que aguarda 450 µs para detectar o início da transmissão de dados.
- **Recepção de Dados (DATA):** Estado no qual os 32 bits de dados são recebidos do controle remoto.

A FSM monitora o sinal IR e realiza transições entre os estados com base nos tempos dos pulsos detectados e na ausência de erros. Caso algum erro seja detectado durante a transmissão dos dados, a FSM retorna ao estado inicial, reiniciando o processo de recepção.

Recepção de Dados

Durante o estado de recepção de dados, o sistema começa a receber os 32 bits de informações enviados pelo controle remoto. A FSM interpreta se o pulso recebido representa um "0"ou um "1", com base nos tempos medidos pelos contadores. Esses dados são armazenados em um registrador, enquanto um contador de 6 bits garante que exatamente 32 bits sejam recebidos.

2.9.2 Decodificação dos Comandos

Com base no comando recebido do controle remoto, o código exibe o número correspondente no display de 7 segmentos. A decodificação é realizada por um bloco lógico que, ao identificar o comando, ajusta os valores de saída em "*led-db*", exibindo o dígito correto no display BCD. Por exemplo, o comando 8'h68 corresponde ao dígito "0", enquanto o comando 8'h30 exibe o dígito "1".

Entretanto, para que esses comandos possam ser enviados corretamente ao display de cristal líquido (LCD), é necessário um segundo decodificador, que traduz os sinais de 7 segmentos para atuarem como botões e gerarem pulsos conforme o comando. A implementação deste decodificador adicional está descrita no Apêndice K.

2.9.3 Integração

Os códigos apresentados neste capítulo foram integrados para que as entradas do controle remoto infravermelho possam gerar pulsos de controle para o sistema. A integração dessa solução é ilustrada na Figura 2.10.

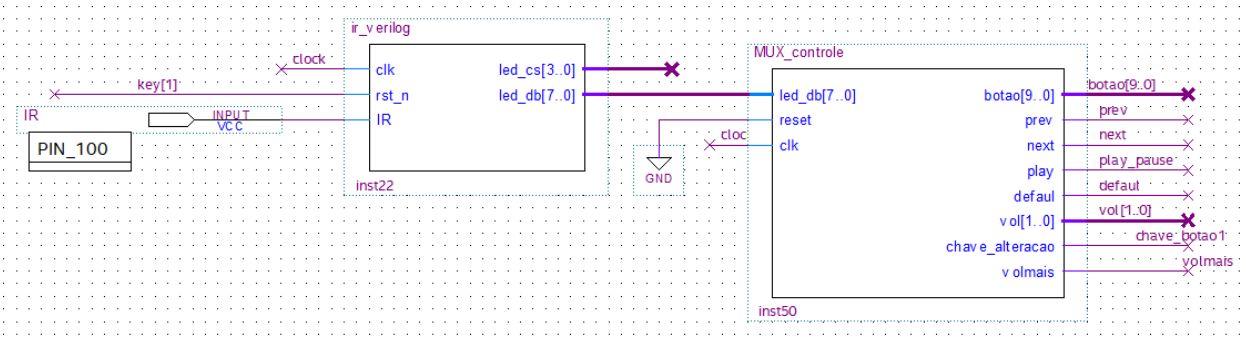


Figura 2.10: Bloco esquemático do controle remoto.

2.10 Sensor de temperatura

Um dos objetivos do projeto é exibir, junto às horas, a temperatura ambiente, utilizando um sensor de temperatura presente no FPGA. Para essa funcionalidade, três módulos foram utilizados, conforme descritos no Anexo A, Anexo B e Apêndice L, os quais implementam uma interface de comunicação I2C para ler dados de temperatura do sensor LM75A, decodificá-los e exibi-los no LCD.

2.10.1 Estrutura

O módulo descrito no Anexo A é responsável por gerar o clock do barramento I2C (SCL), controlar o barramento de dados I2C (SDA) e implementar a máquina de estados finitos (FSM) que gerencia a leitura dos dados do sensor a cada segundo. Ele recebe como entradas o clock de 50 MHz e o sinal de reset, e suas saídas incluem o clock I2C gerado para o barramento SCL e o barramento de dados SDA, usado para comunicação bidirecional com o sensor. Além disso, o módulo gera a saída de temperatura, contendo os dois bytes de leitura (MSB e LSB) obtidos do sensor LM75A.

Geração do Clock I2C

O primeiro bloco do código é responsável pela geração do clock para o barramento I2C. A frequência do clock é gerada por um contador que divide o clock de 50 MHz, resultando em um período de aproximadamente 4 µs (atendendo ao mínimo de 2,5 ns exigido pelo protocolo I2C). O clock I2C é produzido em quatro fases: nível alto, borda de descida, nível baixo e borda de subida, cada uma representando uma porção do ciclo do sinal SCL.

Contador de Tempo para Leitura de Temperatura

Um segundo contador controla a frequência das leituras de temperatura, configurado para realizar a leitura a cada 1 segundo. O contador é incrementado a cada ciclo de clock e, ao atingir o valor limite, ativa a FSM para iniciar o processo de leitura da temperatura.

Máquina de Estados Finitos (FSM)

A FSM segue a sequência de comunicação padrão do protocolo I2C, conforme especificado no datasheet do sensor LM75A. Os principais estados da FSM incluem:

- **IDLE**: Estado inicial, aguardando o término do temporizador de 1 segundo para iniciar a comunicação.
- **START**: Inicia a comunicação I2C com o sensor, gerando a condição de início (START condition).
- **ADDRESS**: Envia o endereço do dispositivo I2C (neste caso, o LM75A) e o bit de leitura.
- **ACK1**: Aguarda a confirmação (ACK) do sensor.
- **READ1**: Lê o primeiro byte de dados (MSB da temperatura).
- **ACK2**: Envia uma nova confirmação ao sensor.
- **READ2**: Lê o segundo byte de dados (LSB da temperatura).
- **NACK**: Finaliza a comunicação sem enviar uma nova confirmação.
- **STOP**: Gera a condição de parada (STOP condition) para encerrar a comunicação.

Operação de Leitura

Nos estados de leitura "READ1"e "READ2", os dados lidos do barramento I2C são armazenados em um registrador, convertendo o sinal de formato serial para paralelo à medida que os dados são recebidos. O primeiro byte corresponde ao MSB da leitura de temperatura, enquanto o segundo byte contém o LSB. Após a leitura dos dois bytes, o registrador é atribuído à saída do módulo.

Controle do Barramento SDA

O barramento de dados SDA é controlado por um registrador e uma flag, que definem sua direção. O módulo alterna entre saída durante a transmissão de dados (quando o mestre envia dados ao escravo) e entrada durante a recepção de dados do sensor. Quando o barramento SDA não está em uso, ele é colocado em alta impedância, permitindo que o sensor controle a linha.

2.10.2 Decodificação da Leitura

O módulo descrito no Anexo B implementa a decodificação dos dados de temperatura no formato BCD (0 a 125°C), exibindo-os em displays de 7 segmentos. O módulo recebe como entradas o clock, o sinal de reset e os dados de temperatura de 16 bits (MSB e LSB) fornecidos pelo sensor LM75A.

As saídas correspondem aos sinais que controlam os segmentos do display e a seleção de qual display estará ativo.

Geração do Clock de 1kHz

O código começa gerando um clock de aproximadamente 1 kHz para alternar rapidamente entre os displays de 7 segmentos, de modo que todos pareçam ligados simultaneamente ao olho humano. Isso é feito por um contador que divide o clock de entrada até 50.000 ciclos, reiniciando então para gerar a frequência de 1 kHz. Esse clock define o tempo de atualização de cada display.

Seleção do Display

A lógica de controle dos displays é implementada por um registrador que define qual display será ativado em cada intervalo de tempo. O barramento seleciona o display ativo, alternando rapidamente entre os quatro displays para garantir que todos mostrem suas informações de forma contínua.

Decodificação BCD para Display de 7 Segmentos

A decodificação dos valores BCD para os displays de 7 segmentos é feita através de um decodificador que ativa os segmentos apropriados para exibir os números de 0 a 9. O ponto decimal é ativado no display 3, que exibe a unidade da temperatura, para representar a casa decimal (por exemplo, 25.5°C).

Conversão dos Dados de Temperatura

Os dados de temperatura lidos do sensor LM75A são processados para serem exibidos em quatro dígitos nos displays de 7 segmentos. A temperatura é representada em dois bytes:

- **data[15]**: Indica o sinal da temperatura (positivo ou negativo).
- **data[14:8]**: Contém a parte inteira da temperatura.
- **data[7]**: Representa a parte decimal (0 para .0, 1 para .5).

O valor correspondente é extraído e processado para cada um dos quatro displays, exibindo a temperatura da seguinte forma:

- **Display 4**: Mostra a parte decimal (.0 ou .5).

- **Display 3**: Exibe a unidade da temperatura, baseada em **data[14:8]**.
- **Display 2**: Exibe a dezena da temperatura.
- **Display 1**: Exibe a centena (0 ou 1, dado que a faixa vai de 0 a 125°C).

O código ajusta corretamente os valores de exibição conforme a leitura de temperatura e a conversão BCD, garantindo a exibição precisa no display.

2.10.3 Decodificação de 7 Segmentos para Bits

O módulo descrito no Apêndice L converte os sinais dos displays de 7 segmentos em valores binários de 4 bits, representando cada dígito da temperatura (centena, dezena, unidade e décimos). Ele recebe como entradas o clock, os segmentos e a seleção do dígito ativo, e as saídas correspondem aos valores de 4 bits para cada posição.

Processo de Conversão

O módulo converte os sinais de 7 segmentos para valores binários de 4 bits, mapeando os segmentos em seus equivalentes numéricos. Dependendo do seletor, o valor convertido é armazenado em um dos quatro registradores intermediários, representando centena, dezena, unidade e décimos.

Atribuição das Saídas

Os valores convertidos são atribuídos às saídas correspondentes, garantindo a exibição correta da temperatura em quatro dígitos.

2.10.4 Integração

A integração dos três módulos permite que a leitura da temperatura seja convertida para um formato binário e manipulada no sistema, como ilustrado na Figura 2.11.

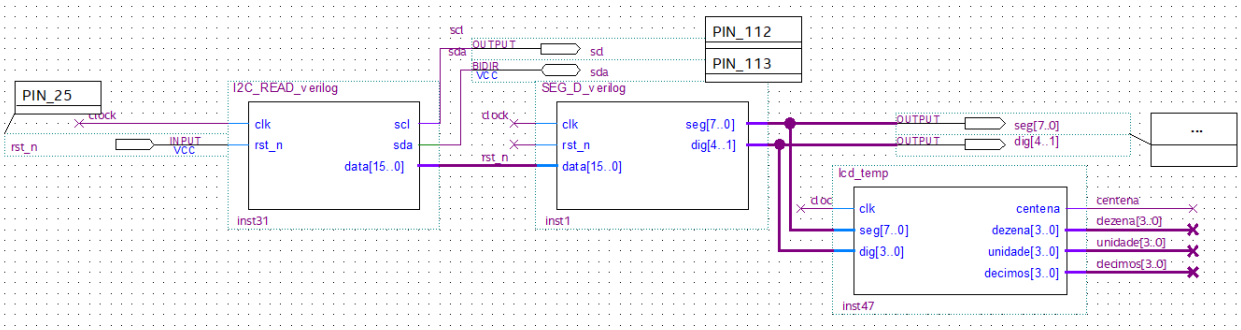


Figura 2.11: Bloco esquemático do controle remoto.

3 Manual de Operação

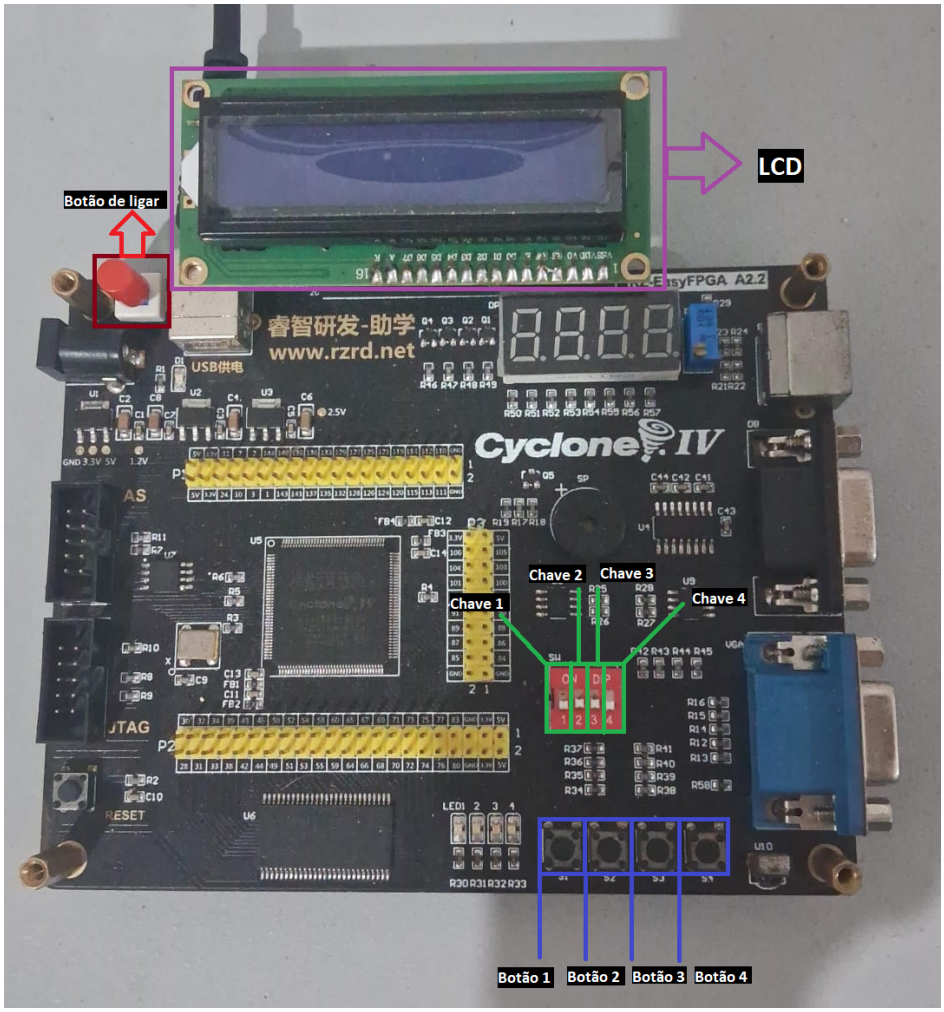


Figura 3.1: Identificação dos itens no FPGA.


CHANNEL
CH−
CH
CH+
PREV
NEXT
PLAY/PAUSE
VOL−
VOL+
EQ
0
100+
200+
1
2
3
4
5
6
7
8
9

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 e temperatura atuais e permitir o ajuste de três alarmes programáveis. O display de cristal líquido, por onde são exibidos o relógio, a temperatura e as informações do projeto, está marcado em roxo na Figura 3.1.

   - A ferramenta utilizada para operar o relógio digital é o controle remoto da Figura 3.2, cujos botões são referenciados no próprio controle.

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 a frase "Eletronica Digital";

   - Aperte o botão "Prev"para que o display varie para a segunda frase "Projeto 4 Verilog";

   - Aperte o botão "Prev", novamente, para que o display varie para a terceira frase "Alunos: Alysson, Felipe, Victória";

   - Aperte o botão "Prev", novamente, para que o relógio inicie a contagem; o LCD passará a apresentar o relógio digital no formato HH:MM:SS (Horas:Minutos:Segundos) a a temperatura no formado CDU.D^{o}C (Centena Dezena Unidade . Décimo ^{o} C).

3. Ajustes do Relógio e Alarmes

   (a) Entrando no Ajuste do Relógio

      - Para entrar no modo de ajuste do relógio, aperte o botão 1;

      - Será exibida a mensagem "Ajuste Rel"e o display correspondente ao dígito de ajuste começará a piscar;

      - A alteração será feita no modo crescente, caso deseje alterar o dígito no modo decrescente, pressione o botão "Vol+";

   (b) Alterando o Dígito

      - Para alterar o dígito que está selecionado, 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 selecione (pisque) 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;

- Será exibida a mensagem "Ajuste A1"e o display correspondente ao dígito de ajuste começará a piscar;

- 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, aperte o botão 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;

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

- Para desativar o alarme, pressione o botão "Play/Pause"

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 resultou na criação de um relógio digital programado em Verilog e implementado em um Display de Cristal Líquido (LCD), exibindo o horário no formato HH:MM:SS. Ao ligar o sistema, o display mostra informações iniciais, como o nome da equipe desenvolvedora. Através de interações com um botão, é possível navegar por essas informações até que o relógio seja exibido, momento em que o sistema começa a contar o tempo.

O relógio desenvolvido possui uma função de ajuste de horário, ativada ao pressionar o botão 1. Nesse modo de ajuste, o usuário pode utilizar o botão 2 para alterar os valores de horas, minutos e. Baseado nas interações com o botão "volume+"o sistema incrementa os valores ou decrementa com o botão 2. Para ajudar na navegação, o display que está sendo ajustado pisca, indicando que aquela parte está selecionada.

Além do ajuste do horário, o relógio conta com três alarmes independentes. No modo de ajuste, pressionando o botão 4, o usuário pode alternar entre o ajuste de cada alarme. O botão 2 permite incrementar ou decrementar os valores do alarme, enquanto o botão 3 permite trocar entre a configuração de horas e minutos. Quando o horário do relógio coincide com o horário configurado para algum dos alarmes, um som é emitido, sendo que cada alarme possui um som distinto. Existe também um botão dedicado para silenciar o buzzer quando o alarme dispara. O display LCD informa em qual estado o relógio se encontra, seja no modo de exibição do horário ou no modo de ajuste (relógio ou alarmes).

Adicionalmente, o relógio possui uma funcionalidade que mostra a temperatura ambiente. Utilizando um sensor presente na placa FPGA, o sistema é capaz de medir a temperatura e exibi-la junto com o horário, oferecendo uma funcionalidade prática extra

5 Discussão dos Resultados

O desenvolvimento do relógio digital foi satisfatório e alcançou o objetivo de criar um dispositivo funcional, prático e fácil de usar. A interface com o LCD foi bem projetada, permitindo ao usuário acompanhar o estado do sistema e realizar ajustes de forma intuitiva, graças ao uso de indicadores visuais como o piscar do display.

O destaque do projeto foi a implementação dos alarmes com sons diferenciados para cada um, uma solução que agrega versatilidade e praticidade ao dispositivo. A possibilidade de ajustar até três alarmes permite que o usuário tenha mais controle sobre sua rotina diária, e o fato de existir um botão para silenciar o alarme sem complicações mostra uma preocupação com a usabilidade do sistema.

Outro aspecto interessante foi a integração do sensor de temperatura, uma função extra que pode tornar o relógio útil em ambientes onde o monitoramento da temperatura é necessário. Isso acrescenta um valor prático ao projeto, já que o relógio não se limita apenas a marcar as horas, mas também fornece informações úteis sobre o ambiente.

Por fim, o projeto alcançou os resultados esperados, oferecendo um sistema completo e funcional, que combina a exibição de tempo com a funcionalidade de alarme e monitoramento de temperatura. A integração dessas funcionalidades com uma interface simples e eficaz garante que o relógio seja útil em diversas situações cotidianas.

6 Conclusão

O desenvolvimento do projeto de um relógio digital com controle remoto e função de despertador, programado na linguagem de descrição de hardware Verilog, resultou em um sistema operacional que atingiu os principais objetivos propostos. As funcionalidades implementadas incluem a contagem de 24 horas, ajuste de horário, configuração de alarmes, funções de incremento e decremento para o modo de ajuste, integração com sensor de temperatura, alterações no relógio através do controle remoto e interação por meio de um display de Cristal Líquido (LCD). O sistema demonstrou alta eficiência ao empregar técnicas de multiplexação e divisores de clock, oferecendo uma interface visual clara e operações consistentes.

Apesar dos resultados alcançados, a implementação do aviso sonoro musical para os alarmes não foi concluída devido à complexidade do projeto e à limitação de tempo disponível.

Embora as limitações encontradas, o projeto atingiu seus principais objetivos, estabelecendo uma base sólida para futuras melhorias. A experiência obtida durante a implementação em Verilog ampliou o entendimento sobre sistemas embarcados e controle de hardware. Com mais tempo e conhecimento técnico, seria viável implementar as funcionalidades pendentes e aprimorar o desempenho e a usabilidade do sistema.

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 Divisor de clock

```
module divisordeclock #(
    parameter Contagem = 10,      // m dulo da contagem
    parameter n = 5                // N mero de bits do contador
) (
    input wire clk_in,
    input wire reset,
    output reg clk_out
);
   reg [n-1:0] aux_contagem = Contagem;
    reg [n-1:0] counter = 0;         // Contador din mico de acordo com o n mero
         de bits definido
                        // Processo s ncrono controlado pela borda de subida do
                            clock de entrada
    always @(posedge !clk_in or posedge reset) begin
        if (reset) begin
            counter <= 0;             // Reset do contador
            clk_out <= 0;             // Reset do clock de sa da
        end else begin
            if (counter == aux_contagem-1) begin
                counter <= 0;         // Reinicia o contador
                clk_out <= (~clk_out);  // Inverte o clock de sa da
            end else begin
                counter <= counter + 1; // Incrementa o contador
            end
        end
    end

endmodule
```

Apêndice B Contador de 4 bits

```
module contador4bits(
   input clk,                    // Sinal de clock
   input decrescente,            // Sinal de controle (0 para contagem crescente, 1
       para decrescente)
   input resetar,                // Sinal de reset
   input [3:0] max_value,        // Valor m ximo da contagem
   output reg reset,             // Sinal de reset da contagem
   output reg [3:0] count        // Sa da do contador de 4 bits
);

   // L gica do contador
   always @(posedge clk or posedge resetar) begin
      if (resetar) begin
         count <= 4'b0000;       // Reseta o contador para zero
         reset <= 1'b0;          // Indica que houve um reset
      end else begin
         if (decrescente) begin
            if (count == 4'b0000) begin
               count <= max_value; // Reseta para o valor m ximo quando atinge o
                   m nimo
               reset <= 1'b1;
            end else begin
               count <= count - 1; // Contagem decrescente
               reset <= 1'b0;
            end
         end else begin
            if (count == max_value) begin
               count <= 4'b0000;   // Reseta para zero ao atingir o valor m ximo
               reset <= 1'b1;
            end else begin
               count <= count + 1; // Contagem crescente
               reset <= 1'b0;
            end
         end
      end
```

```
34     end
35 endmodule
```

Apêndice C Relógio

```
1  module relogio(
2     input wire clk, clk50mhz,                    // Sinal de clock
3    input wire ENA,          // Sinal de controle (0 para contagem crescente, 1 para
         decrescente)
4    input wire [3:1] botao,
5    input wire chave_alteracao,
6    input wire[2:0] seletor,
7    input wire relogio_ativo,
8     output reg [23:0] relogio_completo
9  );
10
11     // Sinais intermedi rios para o reset dos contadores
12 wire  carry_min;
13 reg clock[6:1];
14 wire reset[6:1];
15 reg incremento = 1'b0;
16 wire [23:0] relogio_parcial;
17 wire [3:0] seg_unidade, seg_dezena, min_unidade, min_dezena, hora_unidade,
     hora_dezena;    // Dezena das horas (0-2)
18 assign relogio_parcial = {hora_dezena[3:0],hora_unidade[3:0],min_dezena[3:0],
     min_unidade[3:0],seg_dezena[3:0],seg_unidade[3:0]};
19
20
21
22     // Contador de unidades dos segundos (0-9)
23 contador4bits contador_seg_unidade (
24    .clk(clock[1]),
25    .decrescente(ENA),
26   .resetar(1'b0), //condi  o para resetar caso necess rio
27    .max_value(4'd9),       // Valor m ximo para unidades dos segundos
28   .reset(reset[1]),
29    .count(seg_unidade)
30 );
```

```
31
32      // Contador de dezenas dos segundos (0-5)
33 contador4bits contador_seg_dezena (
34    .clk(clock[2]),
35    .decrescente(ENA),
36   .resetar(1'b0), //condi  o para resetar caso necess rio
37    .max_value(4'd5),       // Valor m ximo para dezenas dos segundos
38    .reset(reset[2]),
39   .count(seg_dezena)
40    );
41
42
43      // Contador de unidades dos minutos (0-9)
44 contador4bits contador_min_unidade (
45    .clk(clock[3]),
46    .decrescente(ENA),
47   .resetar(1'b0), //condi  o para resetar caso necess rio
48    .max_value(4'd9),       // Valor m ximo para unidades dos minutos
49    .reset(reset[3]),
50   .count(min_unidade)
51 );
52
53      // Contador de dezenas dos minutos (0-5)
54 contador4bits contador_min_dezena (
55    .clk(clock[4]),
56    .decrescente(ENA),
57   .resetar(1'b0), //condi  o para resetar caso necess rio
58    .max_value(4'd5),       // Valor m ximo para dezenas dos minutos
59    .reset(reset[4]),
60   .count(min_dezena)
61 );
62
63 assign carry_min = ((relogio_completo[23:20] == 4'd2) && (relogio_completo[19:16]
       >= 4'd4)) ;  // Carry para horas
64
65      // Contador de unidades das horas (0-9) ou (0-3)
66 contador4bits contador_hora_unidade (
67    .clk(clock[5]),
68    .decrescente(ENA),
```

```
69    .resetar(carry_min), //condi  o para resetar caso necess rio
70     .max_value(4'd9),       // Valor m ximo para unidades das horas
71     .reset(reset[5]),
72    .count(hora_unidade)
73  );
74
75      // Contador de dezenas das horas (0-2)
76  contador4bits contador_hora_dezena (
77     .clk(clock[6]),
78     .decrescente(ENA),
79    .resetar(carry_min), //condi  o para resetar caso necess rio
80     .max_value(4'd2),       // Valor m ximo para dezenas das horas
81     .reset(reset[6]),
82    .count(hora_dezena)
83  );
84
85
86  always @(posedge clk50mhz or posedge seletor)
87  begin
88    relogio_completo <= relogio_parcial;
89
90
91    case (chave_alteracao)
92      0:
93      begin
94        clock[1] <= clk;
95        clock[2] <= reset[1];
96        clock[3] <= reset[2];
97        clock[4] <= reset[3];
98        clock[5] <= reset[4];
99        clock[6] <= reset[5];
100     end
101     1:
102     begin
103       if (relogio_ativo) begin
104         incremento <= botao[2]; //define o incremento como o bot o 2
105       end
106       else begin
107         incremento <= 1'b0;
```

```
108       end
109       case (seletor)
110
111           3'd0: begin    clock[1] <= incremento;clock[2] <= 1'b0;clock[3] <= 1'b0;
                  clock[4] <= 1'b0;clock[5] <= 1'b0;clock[6] <= 1'b0; end
112
113           3'd1: begin    clock[1] <= 1'b0;clock[2] <= incremento;clock[3] <= 1'b0;
                  clock[4] <= 1'b0;clock[5] <= 1'b0;clock[6] <= 1'b0; end
114
115           3'd2: begin    clock[1] <= 1'b0;clock[2] <= 1'b0;clock[3] <= incremento;
                  clock[4] <= 1'b0;clock[5] <= 1'b0;clock[6] <= 1'b0; end
116
117           3'd3: begin    clock[1] <= 1'b0;clock[2] <= 1'b0;clock[3] <= 1'b0;clock
                  [4] <= incremento;clock[5] <= 1'b0;clock[6] <= 1'b0; end
118
119           3'd4: begin    clock[1] <= 1'b0;clock[2] <= 1'b0;clock[3] <= 1'b0;clock
                  [4] <= 1'b0;clock[5] <= incremento;clock[6] <= 1'b0; end
120
121           3'd5: begin    clock[1] <= 1'b0;clock[2] <= 1'b0;clock[3] <= 1'b0;clock
                  [4] <= 1'b0;clock[5] <= 1'b0;clock[6] <= incremento; end
122
123           default: begin clock[1] <= 1'b0;clock[2] <= 1'b0;clock[3] <= 1'b0;clock
                  [4] <= 1'b0;clock[5] <= 1'b0;clock[6] <= 1'b0; end
124
125       endcase
126
127     end
128   endcase
129 end
130
131
132 endmodule
```

Apêndice D Alarme

```
1 module alarme(
2    input wire clk, clk50mhz,                    // Sinal de clock
```

```
  input wire ENA,          // Sinal de controle (0 para contagem crescente, 1 para
      decrescente)
  input wire[2:0] seletor,
  input alarme_ativo,
   output reg [23:0] alarme_completo
);

    // Sinais intermedi rios para o reset dos contadores
wire  carry_min;
reg clock[6:1];
wire reset[6:1];
reg incremento;
wire [23:0] alarme_parcial;
wire [3:0] seg_unidade, seg_dezena, min_unidade, min_dezena, hora_unidade,
    hora_dezena;    // Dezena das horas (0-2)
assign alarme_parcial = {hora_dezena[3:0],hora_unidade[3:0],min_dezena[3:0],
    min_unidade[3:0],seg_dezena[3:0],seg_unidade[3:0]};



    // Contador de unidades dos segundos (0-9)
contador4bits contador_seg_unidade (
   .clk(clock[1]),
   .decrescente(ENA),
  .resetar(1'b0), //condi  o para resetar caso necess rio
   .max_value(4'd9),       // Valor m ximo para unidades dos segundos
  .reset(reset[1]),
   .count(seg_unidade)
);

    // Contador de dezenas dos segundos (0-5)
contador4bits contador_seg_dezena (
   .clk(clock[2]),
   .decrescente(ENA),
  .resetar(1'b0), //condi  o para resetar caso necess rio
   .max_value(4'd5),       // Valor m ximo para dezenas dos segundos
   .reset(reset[2]),
  .count(seg_dezena)
   );
```

```
39
40
41      // Contador de unidades dos minutos (0-9)
42 contador4bits contador_min_unidade (
43     .clk(clock[3]),
44     .decrescente(ENA),
45    .resetar(1'b0), //condi  o para resetar caso necess rio
46     .max_value(4'd9),       // Valor m ximo para unidades dos minutos
47     .reset(reset[3]),
48    .count(min_unidade)
49 );
50
51      // Contador de dezenas dos minutos (0-5)
52 contador4bits contador_min_dezena (
53     .clk(clock[4]),
54     .decrescente(ENA),
55    .resetar(1'b0), //condi  o para resetar caso necess rio
56     .max_value(4'd5),       // Valor m ximo para dezenas dos minutos
57     .reset(reset[4]),
58    .count(min_dezena)
59 );
60
61 assign carry_min = (hora_dezena == 4'd2 && hora_unidade <= 4'd5);  // Carry para
       horas
62
63      // Contador de unidades das horas (0-9) ou (0-3)
64 contador4bits contador_hora_unidade (
65     .clk(clock[5]),
66     .decrescente(ENA),
67    .resetar(carry_min), //condi  o para resetar caso necess rio
68     .max_value(4'd9),       // Valor m ximo para unidades das horas
69     .reset(reset[5]),
70    .count(hora_unidade)
71 );
72
73      // Contador de dezenas das horas (0-2)
74 contador4bits contador_hora_dezena (
75     .clk(clock[6]),
76     .decrescente(ENA),
```

```
77    .resetar(1'b0), //condi  o para resetar caso necess rio
78     .max_value(4'd2),       // Valor m ximo para dezenas das horas
79     .reset(reset[6]),
80    .count(hora_dezena)
81  );
82
83
84  always @(posedge clk50mhz or posedge seletor)
85  begin
86    alarme_completo <= alarme_parcial;
87    case (alarme_ativo)
88      0:
89      begin
90        clock[1] <= 1'd0;
91        clock[2] <= 1'd0;
92        clock[3] <= 1'd0;
93        clock[4] <= 1'd0;
94        clock[5] <= 1'd0;
95        clock[6] <= 1'd0;
96      end
97      1:
98      begin
99        incremento <= clk; //define o incremento como o bot o 2
100
101     case (seletor)
102
103         3'd0: begin    clock[1] <= incremento;clock[2] <= 1'b0;clock[3] <= 1'b0;
                clock[4] <= 1'b0;clock[5] <= 1'b0;clock[6] <= 1'b0; end
104
105         3'd1: begin    clock[1] <= 1'b0;clock[2] <= incremento;clock[3] <= 1'b0;
                clock[4] <= 1'b0;clock[5] <= 1'b0;clock[6] <= 1'b0; end
106
107         3'd2: begin    clock[1] <= 1'b0;clock[2] <= 1'b0;clock[3] <= incremento;
                clock[4] <= 1'b0;clock[5] <= 1'b0;clock[6] <= 1'b0; end
108
109         3'd3: begin    clock[1] <= 1'b0;clock[2] <= 1'b0;clock[3] <= 1'b0;clock[4]
                 <= incremento;clock[5] <= 1'b0;clock[6] <= 1'b0; end
110
```

```
111         3'd4: begin    clock[1] <= 1'b0;clock[2] <= 1'b0;clock[3] <= 1'b0;clock[4]
                <= 1'b0;clock[5] <= incremento;clock[6] <= 1'b0; end
112
113         3'd5: begin    clock[1] <= 1'b0;clock[2] <= 1'b0;clock[3] <= 1'b0;clock[4]
                <= 1'b0;clock[5] <= 1'b0;clock[6] <= incremento; end
114
115         default: begin clock[1] <= 1'b0;clock[2] <= 1'b0;clock[3] <= 1'b0;clock[4]
                <= 1'b0;clock[5] <= 1'b0;clock[6] <= 1'b0; end
116
117     endcase
118
119     end
120   endcase
121 end
122
123
124 endmodule
```

Apêndice E Seletor display

```
1  module selecao_display (
2     input clk,                    // Sinal de clock
3     input reset,                  // Sinal de reset
4    output reg [2:0] count      // contador 3 bits
5
6  );
7
8
9     // L gica do contador com reset ass ncrono
10 always @(posedge clk or posedge reset)
11 begin
12    if (reset)
13       count <= 3'b000;
14    else if (count == 3'd5)
15       count <= 3'b000;
16    else
17       count <= count + 1;  // Incremento da contagem
18 end
```

```
19
20
21
22 endmodule
```

Apêndice F Seletor função

```
1  module selecao_ajuste (
2     input clk,                    // Sinal de clock
3     input reset,                  // Sinal de reset
4     output reg relogio,
5     output reg alarme_1,
6     output reg alarme_2,
7     output reg alarme_3
8  );
9     reg [1:0] count;
10
11    // L gica do contador com reset ass ncrono
12    always @(posedge clk or posedge reset) begin
13       if (reset)
14          count <= 2'd0;
15       else if (count == 2'd3)
16          count <= 2'd0;
17       else
18          count <= count + 1;  // Incremento da contagem
19    end
20
21    // Sele  o das sa das com base no valor do contador
22    always @(*) begin
23       case (count)
24          2'd0 : begin relogio = 1'b1; alarme_1 = 1'b0; alarme_2 = 1'b0; alarme_3 =
                1'b0;
25          end
26          2'd1 : begin relogio = 1'b0; alarme_1 = 1'b1; alarme_2 = 1'b0; alarme_3 =
                1'b0;
27          end
28          2'd2 : begin relogio = 1'b0; alarme_1 = 1'b0; alarme_2 = 1'b1; alarme_3 =
                1'b0;
```

```
29          end
30          2'd3 : begin relogio = 1'b0; alarme_1 = 1'b0; alarme_2 = 1'b0; alarme_3 =
                1'b1;
31          end
32          default: begin relogio = 1'b0; alarme_1 = 1'b0; alarme_2 = 1'b0; alarme_3
                = 1'b0;
33          end
34      endcase
35   end
36 endmodule
```

Apêndice G Gerenciador de saídas

```
1  module Mux (
2      input wire [23:0] horario_relogio,
3      input wire [23:0] horario_alarme_1,
4      input wire [23:0] horario_alarme_2,
5      input wire [23:0] horario_alarme_3,
6      input wire botao_pause,
7      output reg [23:0] horario_saida,
8      output reg comparador1,
9      output reg comparador2,
10     output reg comparador3,
11     input wire clk,
12     input wire relogio,
13     input wire alarme1,
14     input wire alarme2,
15     input wire alarme3
16 );
17
18 // Multiplexador
19 always @(posedge clk) begin
20     if (relogio)
21         horario_saida <= horario_relogio;
22     else if (alarme1)
23         horario_saida <= horario_alarme_1;
24     else if (alarme2)
25         horario_saida <= horario_alarme_2;
```

```
26      else if (alarme3)
27          horario_saida <= horario_alarme_3;
28  end
29
30  // Sistema de compara  o
31  always @* begin
32      // Reset das sa das do comparador no in cio do bloco para evitar valores
            indesejados
33      // Verifica a condi  o do bot o pause primeiro
34      if (botao_pause) begin
35          comparador1 <= 1'b0;
36          comparador2 <= 1'b0;
37          comparador3 <= 1'b0;
38      end
39      else if (horario_relogio == horario_alarme_1) begin
40          comparador1 <= 1'b1;
41      end
42      else if (horario_relogio == horario_alarme_2) begin
43          comparador2 <= 1'b1;
44      end
45      else if (horario_relogio == horario_alarme_3) begin
46          comparador3 <= 1'b1;
47      end
48  end
49
50  endmodule
```

Apêndice H Controlador LCD

```
1  module LCD_Controle
2
3  //Par metros
4  #(
5    parameter clk_freq       = 50,  //clock principal em MHz
6    parameter display_lines  = 1'b1,   //n mero de linhas do display (0 = uma linha
         , 1 = duas linhas)
7    parameter character_font = 1'b0,   //fonte (0 = 5x8 pontos, 1 = 5x10 pontos)
8    parameter display_on_off = 1'b1,   //display on/off (0 = off, 1 = on)
```

```
9   parameter cursor         = 1'b0,   //cursor on/off (0 = off, 1 = on)
10  parameter blink          = 1'b0,   //blink on/off (0 = off, 1 = on)
11  parameter inc_dec        = 1'b1,   //incremento/decremento (0 = decremento, 1 =
        incremento)
12  parameter shift          = 1'b0    //shift on/off (0 = off, 1 = on)
13 )
14
15 (
16  input clk, lcd_enable,
17  input [9:0] lcd_bus,
18
19  output reg rw, rs, e,
20  output reg [7:0] lcd_data,
21  output reg busy = 1'b1
22 );
23
24 //declara  o de estados da fsm
25 reg [1:0] estado = 2'd0;
26
27 always@(posedge clk) begin
28     //estado <= 2'd0; //ENERGIZACAO(0)
29
30  reg [31:0] clk_count = 1'b0; //contador para temporiza  o de eventos
31
32  case(estado)
33  //Espera 50ms para garantir a energiza  o do LCD
34    0 : begin
35       busy <= 1'b1;                          //Quando for ENERGIZA  O
36       if(clk_count < (50000 * clk_freq)) begin //Espera 50ms
37         clk_count = clk_count + 1;
38         estado   <= 2'd0;                    //Enquanto estiver ocupado,
               permane o na energiza  o
39       end
40       else begin                             //Energiza  o Completa
41         clk_count = 0;
42         rs <= 1'b0;                          //opera  o de instru  o
43         rw <= 1'b0;                          //Opera  o de escrita
44         lcd_data <= 8'b00110000;             //8-bits 1L*16 5*8 function_set
45         estado <= 2'd1;                      //INICIALIZA  O(1)
```

```
46        end
47      end
48    //sequ ncia de inicializa  o do display LCD
49      1 : begin//INICIALIZA  O(1)
50         busy <= 1'b1;                                  //lcd ocupado
51         clk_count = clk_count + 1;                     //function set
52         if(clk_count < (10 * clk_freq)) begin
53          lcd_data <= {4'b0011, display_lines, character_font, 2'b00};
54          e <= 1'b1;                                    //habilita o LCD (executa o
                comando)
55          estado <= 2'd1;
56         end
57         else if(clk_count < (60 * clk_freq)) begin     //espera 50 us
58          lcd_data <= 8'b00000000;                      //nenhuma nova instru  o ,
                apenas aguarda
59          e <= 1'b0;                                    //desabilita o LCD
60          estado <= 2'd1;
61         end
62         else if(clk_count < (70 * clk_freq)) begin     //display on/off control
63          lcd_data <= {5'b00001, display_on_off, cursor, blink};
64          e <= 1'b1;                                    //habilita o LCD (executa o
                comando)
65          estado <= 2'd1;
66         end
67         else if(clk_count < (120 * clk_freq)) begin    //espera 50 us
68            lcd_data <= 8'b00000000;
69            e <= 1'b0;
70            estado <= 2'd1;
71         end
72         else if(clk_count < (130 * clk_freq)) begin    //display clear
73            lcd_data <= 8'b00000001;
74            e <= 1'b1;                                 //habilita o LCD (executa o
                  comando)
75            estado <= 2'd1;
76         end
77         else if(clk_count < (2130 * clk_freq)) begin   //wait 2 ms
78            lcd_data <= 8'b00000000;
79            e <= 1'b0;
80            estado <= 2'd1;
```

```
81        end
82        else if(clk_count < (2140 * clk_freq)) begin  //entry mode set
83           lcd_data <= {6'b000001, inc_dec, shift};
84           e <= 1'b1;                                 //habilita o LCD (executa o
                 comando)
85           estado <= 2'd1;
86        end
87        else if(clk_count < (2200 * clk_freq)) begin  //wait 60 us
88           lcd_data <= 8'b00000000;
89           e <= 1'b0;
90           estado <= 2'd1;
91        end
92        else begin                                     //initialization complete
93           clk_count = 0;
94           busy <= 1'b0;
95           estado <= 2'd2;                            //PRONTO(2)
96        end
97     end
98     2 :begin//PRONTO
99       if (lcd_enable) begin
100        busy <= 1'b1;
101        rs <= lcd_bus[9];
102        rw <= lcd_bus[8];
103        lcd_data <= lcd_bus;  //vamo verrrr
104        clk_count = 0;
105        estado <= 2'd3;
106      end
107      else begin
108        busy <= 1'b0;
109        rs <= 1'b0;
110        rw <= 1'b0;
111        lcd_data <= 8'b00000000;                     //O mesmo que uma instru  o
               de fazer nada
112        clk_count = 0;
113        estado <= 2'd2;
114      end
115    end
116    3 :begin//ENVIAR
117      busy <= 1'b1;
```

```
118       if (clk_count < (50 * clk_freq)) begin          //espera 50 us
119           if (clk_count < clk_freq) begin           //enable negativo
120             e <= 1'b0;                            //desabilita o LCD
121           end
122           else if(clk_count < (14 * clk_freq)) begin //enable positivo em metade
                  do ciclo (25us)
123             e <= 1'b1;                              //habilita o LCD (executa o
                    comando)
124           end
125           else if(clk_count < (27 * clk_freq)) begin //enable negativo na outra
                  metade do ciclo (25us)
126             e <= 2'b0;                              //desabilita o LCD
127           end
128           clk_count = clk_count + 1;
129           estado <= 2'd3;
130       end
131       else begin
132         clk_count = 0;
133         estado <= 2'd2;
134       end
135      end
136     endcase
137   end
138 endmodule
```

Apêndice I Lógica LCD

```
1  module lcd_logic
2  (
3    input wire clk, lcd_busy, iniciar_relogio,clock_piscar, // clock50MHZ, VERIFICA
       SE LCD DISPON VEL, BOT O PARA INICIAR RELOGIO, CLOCK_2SEG
4    input centena,
5    input [3:0]dezena,
6    input [3:0]unidade,
7    input [3:0]decimo,
8    input [23:0]hora,  //vetor de horas e alarmes
9    input relogio_ativo,
10   input alarme1_ativo,
```

```
11    input alarme2_ativo,
12    input alarme3_ativo,
13    input ajuste,
14    input [2:0] seletor_dig,
15    output lcd_ena,
16    output [9:0] lcd_bar,
17    output ativamento
18  );
19
20  reg [1:0] estado = 2'b00;
21  reg [1:0] seletor = 2'b00;
22  reg aux = 1'b1;
23  reg lcd_enable;
24  reg [9:0] lcd_bus;
25
26  assign lcd_ena = lcd_enable;
27  assign lcd_bar = lcd_bus;
28  assign ativamento = aux;
29
30
31  always @(posedge iniciar_relogio) begin
32
33    case (seletor)
34      0: seletor <= 2'b01;
35      1: seletor <= 2'b10;
36      2: seletor <= 2'b11;
37      3: seletor <= 2'b11;
38    endcase
39
40  end
41
42  always @(posedge clk)
43  begin
44
45    reg [5:0] char = 6'd0; //em caso de erro, usar o assign.
46
47
48    if(seletor == 2'b00) begin //Seleciona qual frase estar  mostrando no lcd
        referente a determinada musica.
```

```
49      estado <= 2'b00;
50    end
51    else if(seletor == 2'b01) begin
52      estado <= 2'b01;
53    end
54    else if(seletor == 2'b10) begin
55      estado <= 2'b10;
56    end
57    else if(seletor == 2'b11) begin
58      estado <= 2'b11;
59      aux <= 1'b0;
60    end
61
62    case(estado)
63      2'b00: begin
64        if((!lcd_busy) & (!lcd_enable)) begin
65          lcd_enable <= 1'b1; //Habilita o LCD
66          if(char < 34) begin
67            char = char + 1;//Incrementa o estado
68          end
69          else begin
70            char = 0;
71          end
72          case (char)
73            0  : lcd_bus <= {2'b00, 8'b10000000}; //inst. linha 1
74            1  : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
75            2  : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
76            3  : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
77            4  : lcd_bus <= {2'b10, 8'b01000101}; //E
78            5  : lcd_bus <= {2'b10, 8'b01101100}; //l
79            6  : lcd_bus <= {2'b10, 8'b01100101}; //e
80            7  : lcd_bus <= {2'b10, 8'b01110100}; //t
81            8  : lcd_bus <= {2'b10, 8'b01110010}; //r
82            9  : lcd_bus <= {2'b10, 8'b01101111}; //o
83            10 : lcd_bus <= {2'b10, 8'b01101110}; //n
84            11 : lcd_bus <= {2'b10, 8'b01101001}; //i
85            12 : lcd_bus <= {2'b10, 8'b01100011}; //c
86            13 : lcd_bus <= {2'b10, 8'b01100001}; //a
87            14 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
```

```
88          15 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
89          16 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
90          17 : lcd_bus <= {2'b00, 8'b11000000}; //inst. linha 2
91          18 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
92          19 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
93          20 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
94          21 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
95          22 : lcd_bus <= {2'b10, 8'b01000100}; //D
96          23 : lcd_bus <= {2'b10, 8'b01101001}; //i
97          24 : lcd_bus <= {2'b10, 8'b01100111}; //g
98          25 : lcd_bus <= {2'b10, 8'b01101001}; //i
99          26 : lcd_bus <= {2'b10, 8'b01110100}; //t
100         27 : lcd_bus <= {2'b10, 8'b01100001}; //a
101         28 : lcd_bus <= {2'b10, 8'b01101100}; //l
102         29 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
103         30 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
104         31 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
105         32 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
106         33 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
107         default: lcd_enable <= 1'b0; //desabilita o LCD
108       endcase
109     end
110     else
111     begin
112       lcd_enable <= 1'b0; //desabilita o LCD
113     end
114   end //end do estado 1 estado
115
116   2'b01: begin
117     if((!lcd_busy) & (!lcd_enable)) begin
118       lcd_enable <= 1'b1; //Habilita o LCD
119       if(char < 34) begin
120         char = char + 1;//Incrementa o estado
121       end
122       else begin
123         char = 0;
124       end
125       case (char)
126         0  : lcd_bus <= {2'b00, 8'b10000000}; //inst. linha 1
```

```
127           1  : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
128           2  : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
129           3  : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
130           4  : lcd_bus <= {2'b10, 8'b01010000}; //P
131           5  : lcd_bus <= {2'b10, 8'b01110010}; //r
132           6  : lcd_bus <= {2'b10, 8'b01101111}; //o
133           7  : lcd_bus <= {2'b10, 8'b01101010}; //j
134           8  : lcd_bus <= {2'b10, 8'b01100101}; //e
135           9  : lcd_bus <= {2'b10, 8'b01110100}; //t
136           10 : lcd_bus <= {2'b10, 8'b01101111}; //o
137           11 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
138           12 : lcd_bus <= {2'b10, 8'b00110100}; //4
139           13 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
140           14 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
141           15 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
142           16 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
143
144           17 : lcd_bus <= {2'b00, 8'b11000000}; //inst. linha 2
145           18 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
146           19 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
147           20 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
148           21 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
149           22 : lcd_bus <= {2'b10, 8'b01010110}; //V
150           23 : lcd_bus <= {2'b10, 8'b01100101}; //e
151           24 : lcd_bus <= {2'b10, 8'b01110010}; //r
152           25 : lcd_bus <= {2'b10, 8'b01101001}; //i
153           26 : lcd_bus <= {2'b10, 8'b01101100}; //l
154           27 : lcd_bus <= {2'b10, 8'b01101111}; //o
155           28 : lcd_bus <= {2'b10, 8'b01100111}; //g
156           29 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
157           30 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
158           31 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
159           32 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
160           33 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
161           default: lcd_enable <= 1'b0; //desabilita o LCD
162         endcase
163       end
164       else
165       begin
```

```
166         lcd_enable <= 1'b0; //desabilita o LCD
167       end
168     end //end do estado 2 estado
169
170     2'b10: begin
171       if((!lcd_busy) & (!lcd_enable)) begin
172         lcd_enable <= 1'b1; //Habilita o LCD
173         if(char < 34) begin
174           char = char + 1;//Incrementa o estado
175         end
176         else begin
177           char = 0;
178         end
179         case (char)
180           0  : lcd_bus <= {2'b00, 8'b10000000}; //inst. linha 1
181           1  : lcd_bus <= {2'b10, 8'b01000001}; //A
182           2  : lcd_bus <= {2'b10, 8'b01101100}; //l
183           3  : lcd_bus <= {2'b10, 8'b01110101}; //u
184           4  : lcd_bus <= {2'b10, 8'b01101110}; //n
185           5  : lcd_bus <= {2'b10, 8'b01101111}; //o
186           6  : lcd_bus <= {2'b10, 8'b01110011}; //s
187           7  : lcd_bus <= {2'b10, 8'b00111010}; //:
188           8  : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
189           9  : lcd_bus <= {2'b10, 8'b01000001}; //A
190           10 : lcd_bus <= {2'b10, 8'b01101100}; //l
191           11 : lcd_bus <= {2'b10, 8'b01111001}; //y
192           12 : lcd_bus <= {2'b10, 8'b01110011}; //s
193           13 : lcd_bus <= {2'b10, 8'b01110011}; //s
194           14 : lcd_bus <= {2'b10, 8'b01101111}; //o
195           15 : lcd_bus <= {2'b10, 8'b01101110}; //n
196           16 : lcd_bus <= {2'b10, 8'b00101100}; //,
197
198           17 : lcd_bus <= {2'b00, 8'b11000000}; //inst. linha 2
199           18 : lcd_bus <= {2'b10, 8'b01000110}; //F
200           19 : lcd_bus <= {2'b10, 8'b01100101}; //e
201           20 : lcd_bus <= {2'b10, 8'b01101100}; //l
202           21 : lcd_bus <= {2'b10, 8'b01101001}; //i
203           22 : lcd_bus <= {2'b10, 8'b01110000}; //p
204           23 : lcd_bus <= {2'b10, 8'b01100101}; //e
```

```
205           24 : lcd_bus <= {2'b10, 8'b00101100}; //,
206           25 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
207           26 : lcd_bus <= {2'b10, 8'b01010110}; //V
208           27 : lcd_bus <= {2'b10, 8'b01101001}; //i
209           28 : lcd_bus <= {2'b10, 8'b01100011}; //c
210           29 : lcd_bus <= {2'b10, 8'b01110100}; //t
211           30 : lcd_bus <= {2'b10, 8'b01101111}; //o
212           31 : lcd_bus <= {2'b10, 8'b01110010}; //r
213           32 : lcd_bus <= {2'b10, 8'b01101001}; //i
214           33 : lcd_bus <= {2'b10, 8'b01100001}; //a
215           default : lcd_enable <= 1'b0; //desabilita o LCD
216         endcase
217       end
218       else begin
219         lcd_enable <= 1'b0; //desabilita o LCD
220       end
221     end //end do estado 3 estado
222
223
224     2'b11: begin
225       if((!lcd_busy) & (!lcd_enable)) begin
226         lcd_enable <= 1'b1; //Habilita o LCD
227         if(char < 34) begin
228           char = char + 1;//Incrementa o estado
229         end
230         else begin
231           char = 0;
232         end
233         if (ajuste==0 & relogio_ativo==1) begin   // frase rel gio
234           case (char)
235             0  : lcd_bus <= {2'b00, 8'b10000000}; //inst. linha 1
236             1  : lcd_bus <= {2'b10, 8'b01010010}; //R
237             2  : lcd_bus <= {2'b10, 8'b01100101}; //e
238             3  : lcd_bus <= {2'b10, 8'b01101100}; //l
239             4  : lcd_bus <= {2'b10, 8'b01101111}; //o
240             5  : lcd_bus <= {2'b10, 8'b01100111}; //g
241             6  : lcd_bus <= {2'b10, 8'b01101001}; //i
242             7  : lcd_bus <= {2'b10, 8'b01101111}; //o
243             8  : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
```

```
244             9  : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
245             10 : lcd_bus <= {2'b10, 8'b01010100}; //T
246             11 : lcd_bus <= {2'b10, 8'b01100101}; //e
247             12 : lcd_bus <= {2'b10, 8'b01101101}; //m
248             13 : lcd_bus <= {2'b10, 8'b01110000}; //p
249             14 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
250             15 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
251             16 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
252
253             17 : lcd_bus <= {2'b00, 8'b11000000}; //inst. linha 2
254             18 :                                        //H
255               begin
256                 if ((ajuste) & (clock_piscar) & (seletor_dig== 3'd5))
257                 begin
258                   lcd_bus <= {2'b10, 8'b00100000};
259                 end
260                 else begin
261                   lcd_bus <= {2'b10, 4'b0011, hora[23:20]};
262                 end
263               end
264             19 :
265               begin                                     //H
266                 if ((ajuste) & (clock_piscar) & (seletor_dig== 3'd4))
267                 begin
268                   lcd_bus <= {2'b10, 8'b00100000};
269                 end
270                 else begin
271                   lcd_bus <= {2'b10, 4'b0011, hora[19:16]};
272                 end
273               end
274             20 : lcd_bus <= {2'b10, 8'b00111010};           //:
275             21 :
276               begin                                     //M
277                 if ((ajuste) & (clock_piscar) & (seletor_dig== 3'd3))
278                 begin
279                   lcd_bus <= {2'b10, 8'b00100000};
280                 end
281                 else begin
282                   lcd_bus <= {2'b10, 4'b0011, hora[15:12]};
```

```
283                 end
284               end
285             22 :
286               begin                                    //M
287                 if ((ajuste) & (clock_piscar) & (seletor_dig== 3'd2))
288                 begin
289                   lcd_bus <= {2'b10, 8'b00100000};
290                 end
291                 else begin
292                   lcd_bus <= {2'b10, 4'b0011, hora[11:8]};
293                 end
294               end
295             23 : lcd_bus <= {2'b10, 8'b00111010};           //:
296             24 :
297               begin                                    //S
298                 if ((ajuste) & (clock_piscar) & (seletor_dig== 3'd1))
299                 begin
300                   lcd_bus <= {2'b10, 8'b00100000};
301                 end
302                 else begin
303                   lcd_bus <= {2'b10, 4'b0011, hora[7:4]};
304                 end
305               end               //S
306             25 :
307               begin                                    //S
308                 if ((ajuste) & (clock_piscar) & (seletor_dig== 3'd0))
309                 begin
310                   lcd_bus <= {2'b10, 8'b00100000};
311                 end
312                 else begin
313                   lcd_bus <= {2'b10, 4'b0011, hora[3:0]};
314                 end
315               end
316             26 : lcd_bus <= {2'b10, 8'b00100000};           //Espa o
317             27 : lcd_bus <= {2'b10, 7'b0011000, centena};   //Centena
318             28 : lcd_bus <= {2'b10, 4'b0011, dezena[3:0]};  //Dezena
319             29 : lcd_bus <= {2'b10, 4'b0011, unidade[3:0]}; //Unidade
320             30 : lcd_bus <= {2'b10, 8'b00101110};         //.
321             31 : lcd_bus <= {2'b10, 4'b0011, decimo[3:0]};  //Decimo
```

```
322             32 : lcd_bus <= {2'b10, 8'b11011111};          //
323             33 : lcd_bus <= {2'b10, 8'b01000011};          //C
324           endcase
325         end
326         else if (ajuste==1 & relogio_ativo==1) begin    // frase ajuste alarme 1
327           case (char)
328             0  : lcd_bus <= {2'b00, 8'b10000000}; //inst. linha 1
329             1  : lcd_bus <= {2'b10, 8'b01000001}; //A
330             2  : lcd_bus <= {2'b10, 8'b01101010}; //j
331             3  : lcd_bus <= {2'b10, 8'b01110101}; //u
332             4  : lcd_bus <= {2'b10, 8'b01110011}; //s
333             5  : lcd_bus <= {2'b10, 8'b01110100}; //t
334             6  : lcd_bus <= {2'b10, 8'b01100101}; //e
335             7  : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
336             8  : lcd_bus <= {2'b10, 8'b01010010}; //R
337             9  : lcd_bus <= {2'b10, 8'b01100101}; //e
338             10 : lcd_bus <= {2'b10, 8'b01101100}; //l
339             11 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
340             12 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
341             13 : lcd_bus <= {2'b10, 8'b01010100}; //T
342             14 : lcd_bus <= {2'b10, 8'b01100101}; //e
343             15 : lcd_bus <= {2'b10, 8'b01101101}; //m
344             16 : lcd_bus <= {2'b10, 8'b01110000}; //p
345
346             17 : lcd_bus <= {2'b00, 8'b11000000}; //inst. linha 2
347             18 :                                          //H
348               begin
349                 if ((ajuste) & (clock_piscar) & (seletor_dig== 3'd5))
350                 begin
351                   lcd_bus <= {2'b10, 8'b00100000};
352                 end
353                 else begin
354                   lcd_bus <= {2'b10, 4'b0011, hora[23:20]};
355                 end
356               end
357             19 :
358               begin                                     //H
359                 if ((ajuste) & (clock_piscar) & (seletor_dig== 3'd4))
360                 begin
```

```
361                   lcd_bus <= {2'b10, 8'b00100000};
362                 end
363                 else begin
364                   lcd_bus <= {2'b10, 4'b0011, hora[19:16]};
365                 end
366               end
367             20 : lcd_bus <= {2'b10, 8'b00111010};           //:
368             21 :
369               begin                                     //M
370                 if ((ajuste) & (clock_piscar) & (seletor_dig== 3'd3))
371                 begin
372                   lcd_bus <= {2'b10, 8'b00100000};
373                 end
374                 else begin
375                   lcd_bus <= {2'b10, 4'b0011, hora[15:12]};
376                 end
377               end
378             22 :
379               begin                                     //M
380                 if ((ajuste) & (clock_piscar) & (seletor_dig== 3'd2))
381                 begin
382                   lcd_bus <= {2'b10, 8'b00100000};
383                 end
384                 else begin
385                   lcd_bus <= {2'b10, 4'b0011, hora[11:8]};
386                 end
387               end
388             23 : lcd_bus <= {2'b10, 8'b00111010};           //:
389             24 :
390               begin                                     //S
391                 if ((ajuste) & (clock_piscar) & (seletor_dig== 3'd1))
392                 begin
393                   lcd_bus <= {2'b10, 8'b00100000};
394                 end
395                 else begin
396                   lcd_bus <= {2'b10, 4'b0011, hora[7:4]};
397                 end
398               end               //S
399             25 :
```

```
400             begin                                   //S
401               if ((ajuste) & (clock_piscar) & (seletor_dig== 3'd0))
402               begin
403                 lcd_bus <= {2'b10, 8'b00100000};
404               end
405               else begin
406                 lcd_bus <= {2'b10, 4'b0011, hora[3:0]};
407               end
408             end
409           26 : lcd_bus <= {2'b10, 8'b00100000};           //Espa o
410           27 : lcd_bus <= {2'b10, 7'b0011000, centena};   //Centena
411           28 : lcd_bus <= {2'b10, 4'b0011, dezena[3:0]};  //Dezena
412           29 : lcd_bus <= {2'b10, 4'b0011, unidade[3:0]}; //Unidade
413           30 : lcd_bus <= {2'b10, 8'b00101110};         //.
414           31 : lcd_bus <= {2'b10, 4'b0011, decimo[3:0]};  //Decimo
415           32 : lcd_bus <= {2'b10, 8'b11011111};         //
416           33 : lcd_bus <= {2'b10, 8'b01000011};         //C
417         endcase
418       end
419       else if (alarme1_ativo) begin   // frase ajuste alarme 1
420         case (char)
421           0  : lcd_bus <= {2'b00, 8'b10000000}; //inst. linha 1
422           1  : lcd_bus <= {2'b10, 8'b01000001}; //A
423           2  : lcd_bus <= {2'b10, 8'b01101010}; //j
424           3  : lcd_bus <= {2'b10, 8'b01110101}; //u
425           4  : lcd_bus <= {2'b10, 8'b01110011}; //s
426           5  : lcd_bus <= {2'b10, 8'b01110100}; //t
427           6  : lcd_bus <= {2'b10, 8'b01100101}; //e
428           7  : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
429           8  : lcd_bus <= {2'b10, 8'b01000001}; //A
430           9  : lcd_bus <= {2'b10, 8'b00110001}; //1
431           10 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
432           11 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
433           12 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
434           13 : lcd_bus <= {2'b10, 8'b01010100}; //T
435           14 : lcd_bus <= {2'b10, 8'b01100101}; //e
436           15 : lcd_bus <= {2'b10, 8'b01101101}; //m
437           16 : lcd_bus <= {2'b10, 8'b01110000}; //p
438
```

```
439             17 : lcd_bus <= {2'b00, 8'b11000000}; //inst. linha 2
440             18 :                                        //H
441               begin
442                 if ((ajuste) & (clock_piscar) & (seletor_dig== 3'd5))
443                 begin
444                   lcd_bus <= {2'b10, 8'b00100000};
445                 end
446                 else begin
447                   lcd_bus <= {2'b10, 4'b0011, hora[23:20]};
448                 end
449               end
450             19 :
451               begin                                     //H
452                 if ((ajuste) & (clock_piscar) & (seletor_dig== 3'd4))
453                 begin
454                   lcd_bus <= {2'b10, 8'b00100000};
455                 end
456                 else begin
457                   lcd_bus <= {2'b10, 4'b0011, hora[19:16]};
458                 end
459               end
460             20 : lcd_bus <= {2'b10, 8'b00111010};           //:
461             21 :
462               begin                                     //M
463                 if ((ajuste) & (clock_piscar) & (seletor_dig== 3'd3))
464                 begin
465                   lcd_bus <= {2'b10, 8'b00100000};
466                 end
467                 else begin
468                   lcd_bus <= {2'b10, 4'b0011, hora[15:12]};
469                 end
470               end
471             22 :
472               begin                                     //M
473                 if ((ajuste) & (clock_piscar) & (seletor_dig== 3'd2))
474                 begin
475                   lcd_bus <= {2'b10, 8'b00100000};
476                 end
477                 else begin
```

```
478                 lcd_bus <= {2'b10, 4'b0011, hora[11:8]};
479               end
480             end
481           23 : lcd_bus <= {2'b10, 8'b00111010};           //:
482           24 :
483             begin                                    //S
484               if ((ajuste) & (clock_piscar) & (seletor_dig== 3'd1))
485               begin
486                 lcd_bus <= {2'b10, 8'b00100000};
487               end
488               else begin
489                 lcd_bus <= {2'b10, 4'b0011, hora[7:4]};
490               end
491             end               //S
492           25 :
493             begin                                    //S
494               if ((ajuste) & (clock_piscar) & (seletor_dig== 3'd0))
495               begin
496                 lcd_bus <= {2'b10, 8'b00100000};
497               end
498               else begin
499                 lcd_bus <= {2'b10, 4'b0011, hora[3:0]};
500               end
501             end
502           26 : lcd_bus <= {2'b10, 8'b00100000};           //Espa o
503           27 : lcd_bus <= {2'b10, 7'b0011000, centena};   //Centena
504           28 : lcd_bus <= {2'b10, 4'b0011, dezena[3:0]};  //Dezena
505           29 : lcd_bus <= {2'b10, 4'b0011, unidade[3:0]}; //Unidade
506           30 : lcd_bus <= {2'b10, 8'b00101110};         //.
507           31 : lcd_bus <= {2'b10, 4'b0011, decimo[3:0]};  //Decimo
508           32 : lcd_bus <= {2'b10, 8'b11011111};         //
509           33 : lcd_bus <= {2'b10, 8'b01000011};         //C
510         endcase
511       end
512       else if (alarme2_ativo) begin   // frase ajuste alarme 2
513         case (char)
514           0  : lcd_bus <= {2'b00, 8'b10000000}; //inst. linha 1
515           1  : lcd_bus <= {2'b10, 8'b01000001}; //A
516           2  : lcd_bus <= {2'b10, 8'b01101010}; //j
```

```
517             3  : lcd_bus <= {2'b10, 8'b01110101}; //u
518             4  : lcd_bus <= {2'b10, 8'b01110011}; //s
519             5  : lcd_bus <= {2'b10, 8'b01110100}; //t
520             6  : lcd_bus <= {2'b10, 8'b01100101}; //e
521             7  : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
522             8  : lcd_bus <= {2'b10, 8'b01000001}; //A
523             9  : lcd_bus <= {2'b10, 8'b00110010}; //2
524             10 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
525             11 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
526             12 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
527             13 : lcd_bus <= {2'b10, 8'b01010100}; //T
528             14 : lcd_bus <= {2'b10, 8'b01100101}; //e
529             15 : lcd_bus <= {2'b10, 8'b01101101}; //m
530             16 : lcd_bus <= {2'b10, 8'b01110000}; //p
531
532             17 : lcd_bus <= {2'b00, 8'b11000000}; //inst. linha 2
533             18 :                                  //H
534               begin
535                 if ((ajuste) & (clock_piscar) & (seletor_dig== 3'd5))
536                 begin
537                   lcd_bus <= {2'b10, 8'b00100000};
538                 end
539                 else begin
540                   lcd_bus <= {2'b10, 4'b0011, hora[23:20]};
541                 end
542               end
543             19 :
544               begin                             //H
545                 if ((ajuste) & (clock_piscar) & (seletor_dig== 3'd4))
546                 begin
547                   lcd_bus <= {2'b10, 8'b00100000};
548                 end
549                 else begin
550                   lcd_bus <= {2'b10, 4'b0011, hora[19:16]};
551                 end
552               end
553             20 : lcd_bus <= {2'b10, 8'b00111010};       //:
554             21 :
555               begin                             //M
```

```
556                 if ((ajuste) & (clock_piscar) & (seletor_dig== 3'd3))
557                 begin
558                   lcd_bus <= {2'b10, 8'b00100000};
559                 end
560                 else begin
561                   lcd_bus <= {2'b10, 4'b0011, hora[15:12]};
562                 end
563               end
564             22 :
565               begin                                    //M
566                 if ((ajuste) & (clock_piscar) & (seletor_dig== 3'd2))
567                 begin
568                   lcd_bus <= {2'b10, 8'b00100000};
569                 end
570                 else begin
571                   lcd_bus <= {2'b10, 4'b0011, hora[11:8]};
572                 end
573               end
574             23 : lcd_bus <= {2'b10, 8'b00111010};           //:
575             24 :
576               begin                                    //S
577                 if ((ajuste) & (clock_piscar) & (seletor_dig== 3'd1))
578                 begin
579                   lcd_bus <= {2'b10, 8'b00100000};
580                 end
581                 else begin
582                   lcd_bus <= {2'b10, 4'b0011, hora[7:4]};
583                 end
584               end               //S
585             25 :
586               begin                                    //S
587                 if ((ajuste) & (clock_piscar) & (seletor_dig== 3'd0))
588                 begin
589                   lcd_bus <= {2'b10, 8'b00100000};
590                 end
591                 else begin
592                   lcd_bus <= {2'b10, 4'b0011, hora[3:0]};
593                 end
594               end
```

```
595             26 : lcd_bus <= {2'b10, 8'b00100000};           //Espa o
596             27 : lcd_bus <= {2'b10, 7'b0011000, centena};   //Centena
597             28 : lcd_bus <= {2'b10, 4'b0011, dezena[3:0]};  //Dezena
598             29 : lcd_bus <= {2'b10, 4'b0011, unidade[3:0]}; //Unidade
599             30 : lcd_bus <= {2'b10, 8'b00101110};         //.
600             31 : lcd_bus <= {2'b10, 4'b0011, decimo[3:0]};  //Decimo
601             32 : lcd_bus <= {2'b10, 8'b11011111};         //
602             33 : lcd_bus <= {2'b10, 8'b01000011};         //C
603           endcase
604         end
605         else if (alarme3_ativo) begin   // frase ajuste alarme 3
606           case (char)
607             0  : lcd_bus <= {2'b00, 8'b10000000}; //inst. linha 1
608             1  : lcd_bus <= {2'b10, 8'b01000001}; //A
609             2  : lcd_bus <= {2'b10, 8'b01101010}; //j
610             3  : lcd_bus <= {2'b10, 8'b01110101}; //u
611             4  : lcd_bus <= {2'b10, 8'b01110011}; //s
612             5  : lcd_bus <= {2'b10, 8'b01110100}; //t
613             6  : lcd_bus <= {2'b10, 8'b01100101}; //e
614             7  : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
615             8  : lcd_bus <= {2'b10, 8'b01000001}; //A
616             9  : lcd_bus <= {2'b10, 8'b00110011}; //3
617             10 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
618             11 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
619             12 : lcd_bus <= {2'b10, 8'b00100000}; //Espa o
620             13 : lcd_bus <= {2'b10, 8'b01010100}; //T
621             14 : lcd_bus <= {2'b10, 8'b01100101}; //e
622             15 : lcd_bus <= {2'b10, 8'b01101101}; //m
623             16 : lcd_bus <= {2'b10, 8'b01110000}; //p
624
625             17 : lcd_bus <= {2'b00, 8'b11000000}; //inst. linha 2
626             18 :                                    //H
627               begin
628                 if ((ajuste) & (clock_piscar) & (seletor_dig == 3'd5))
629                 begin
630                   lcd_bus <= {2'b10, 8'b00100000};
631                 end
632                 else begin
633                   lcd_bus <= {2'b10, 4'b0011, hora[23:20]};
```

```
634                 end
635               end
636             19 :
637               begin                                    //H
638                 if ((ajuste) & (clock_piscar) & (seletor_dig== 3'd4))
639                 begin
640                   lcd_bus <= {2'b10, 8'b00100000};
641                 end
642                 else begin
643                   lcd_bus <= {2'b10, 4'b0011, hora[19:16]};
644                 end
645               end
646             20 : lcd_bus <= {2'b10, 8'b00111010};           //:
647             21 :
648               begin                                    //M
649                 if ((ajuste) & (clock_piscar) & (seletor_dig== 3'd3))
650                 begin
651                   lcd_bus <= {2'b10, 8'b00100000};
652                 end
653                 else begin
654                   lcd_bus <= {2'b10, 4'b0011, hora[15:12]};
655                 end
656               end
657             22 :
658               begin                                    //M
659                 if ((ajuste) & (clock_piscar) & (seletor_dig== 3'd2))
660                 begin
661                   lcd_bus <= {2'b10, 8'b00100000};
662                 end
663                 else begin
664                   lcd_bus <= {2'b10, 4'b0011, hora[11:8]};
665                 end
666               end
667             23 : lcd_bus <= {2'b10, 8'b00111010};           //:
668             24 :
669               begin                                    //S
670                 if ((ajuste) & (clock_piscar) & (seletor_dig== 3'd1))
671                 begin
672                   lcd_bus <= {2'b10, 8'b00100000};
```

```
673                 end
674                 else begin
675                   lcd_bus <= {2'b10, 4'b0011, hora[7:4]};
676                 end
677               end               //S
678             25 :
679               begin                                    //S
680                 if ((ajuste) & (clock_piscar) & (seletor_dig== 3'd0))
681                 begin
682                   lcd_bus <= {2'b10, 8'b00100000};
683                 end
684                 else begin
685                   lcd_bus <= {2'b10, 4'b0011, hora[3:0]};
686                 end
687               end
688             26 : lcd_bus <= {2'b10, 8'b00100000};           //Espa o
689             27 : lcd_bus <= {2'b10, 7'b0011000, centena};   //Centena
690             28 : lcd_bus <= {2'b10, 4'b0011, dezena[3:0]};  //Dezena
691             29 : lcd_bus <= {2'b10, 4'b0011, unidade[3:0]}; //Unidade
692             30 : lcd_bus <= {2'b10, 8'b00101110};         //.
693             31 : lcd_bus <= {2'b10, 4'b0011, decimo[3:0]};  //Decimo
694             32 : lcd_bus <= {2'b10, 8'b11011111};         //
695             33 : lcd_bus <= {2'b10, 8'b01000011};         //C
696           endcase
697         end
698       end
699       else begin
700         lcd_enable <= 1'b0; //desabilita o LCD
701       end
702     end //end do estado 4
703
704   endcase // end case dos estados
705 end //end do always
706
707 endmodule
```

Apêndice J Estrutura Controle Remoto

```
/*----------------------------------
Arquivo:   ir_verilog.v
Modulo:    ir_verilog(clk,rst_n,IR,led_cs,led_db)
Descri  o: Controle de Remoto Infravemelho
Autor:     Malki- edheq  Benjamim
Data:      20/01/2022
----------------------------------*/
module ir_verilog(clk,rst_n,IR,led_cs,led_db);

  input   clk; //clock de 50MHz
  input   rst_n; //reset ativo baixo
  input   IR; //entrada de dados irda
  output reg [3:0] led_cs; //4 displays BCD7SEG
  output reg [7:0] led_db; //7 segmentos e ponto de cada display BCD7SEG

  reg [7:0] led1,led2,led3,led4; //representa cada display com 7 segmentos
  reg [15:0] irda_data;    // armazena o dado do irda, e ent o envia para os 7
     segmentos
  reg [31:0] get_data;     // armazena os 32 bits do dado do irda
  reg [5:0]  data_cnt;     // contador para os 32 bits do dado do irda
  reg [2:0]  estado_atual, prox_estado; //registradores de estado da FSM
  reg error_flag;          // flag de erro durante os 32 bits de dados do irda

  //-------------------------------------------------------------------------
  reg irda_reg0;       //valor inst vel
  reg irda_reg1;       //recebe irda_reg0, para estabiliza  o
  reg irda_reg2;       //recebe irda_reg1, auxilia a determinar a borda do irda
  wire irda_negedge; //determina a borda de descida do irda
  wire irda_posedge; //determina a borda de subida do irda
  wire irda_change;    //determina a transi  o de borda do irda

  //reg[15:0] cnt_scan;//

  always @ (posedge clk) //sincroniza os registradores do irda
    if(!rst_n) //reset assincrono dos registradores do irda
      begin
        irda_reg0 <= 1'b0; //limpa o registrador irda_reg0
        irda_reg1 <= 1'b0; //limpa o registrador irda_reg1
        irda_reg2 <= 1'b0; //limpa o registrador irda_reg2
```

```
39      end
40    else
41      begin
42        led_estado_atual<= 4'b0000; //atualiza os registradores do irda na borda
              de subida do clk
43        irda_reg0 <= IR; //recebe o valor lido irda
44        irda_reg1 <= irda_reg0; //atualiza com valor est vel
45        irda_reg2 <= irda_reg1; // atualiza garantindo valor est vel de IR
46      end
47
48  assign irda_change = irda_negedge | irda_posedge; //atribui 1 numa transi  o
      de borda de irda
49  assign irda_negedge = irda_reg2 & (~irda_reg1);   //atribui 1 na borda de
      descida de irda
50  assign irda_posedge = (~irda_reg2) & irda_reg1;   //atribui 1 na borda de
      descida de irda
51
52  reg [10:0] cnt1;      //Divisor de frequ ncia por 1750
53  reg [8:0]  cnt2;      //conta o n mero de pontos ap s o cnt1
54  wire verifica_900us;  // verifica a dura  o de 9ms = 900us
55  wire verifica_450us;  // verifica a dura  o de 4.5ms = 450us
56
57  //L gico '1'     uma rajada de pulso de 562,5 s  seguida por um espa o de
      1,6875ms, com um tempo total de transmiss o de 2,25ms
58  wire high;            // verifica  data="1"
59  //L gico '0'     uma rajada de pulso de 562,5 s  seguida por um espa o de
      562,5 s , com um tempo total de transmiss o de 1,125ms
60  wire low;             // verifica  data="0"
61
62
63  //----------------------------------------------------------------------------
64  always @ (posedge clk)
65    if (!rst_n) //reset ass ncrono
66      cnt1 <= 11'd0; //reinicia o contador 1
67    else if (irda_change) //na transi  o de borda de irda
68      cnt1 <= 11'd0; //reinicia o contador 1
69    else if (cnt1 == 11'd1750) //caso contador estoure
70      cnt1 <= 11'd0; //reinicia o contador 1
71    else
```

```
72      cnt1 <= cnt1 + 1'b1; // incrementa o contador 1
73    //-------------------------------------------------------------------------
74
75    //-------------------------------------------------------------------------
76    always @ (posedge clk)
77      if (!rst_n) //reset ass ncrono
78        cnt2 <= 9'd0; //reinicia o contador 2
79      else if (irda_change) //na transi  o de borda de irda
80        cnt2 <= 9'd0; //reinicia o contador 2
81      else if (cnt1 == 11'd1750) //1750 pulso n vel baixo e 1750 pulsos n vel alto
82        cnt2 <= cnt2 +1'b1; //incrementa o contador 2
83     //-------------------------------------------------------------------------
84
85    //Garante a estabilidade avaliando intervalo de contagem inves do valor exato
86    assign verifica_900us = ((217 < cnt2) & (cnt2 < 297));// valor exato esperado
          256
87    assign verifica_450us = ((88 < cnt2) & (cnt2 < 168)); // valor exato esperado
          128
88    assign high = ((38 < cnt2) & (cnt2 < 58));            // valor exato esperado 48
89    assign low  = ((6 < cnt2) & (cnt2 < 26));             // valor exato esperado 16
90
91    //-------------------------------------------------------------------------
92    // Declara  o da FSM
93    localparam IDLE_STATE   = 3'b000, //estado inicial
94               ATRASO_900us  = 3'b001, //atraso de 900us
95               ATRASO_450us  = 3'b010, //atrado de 450us
96               DATA_STATE    = 3'b100; //estado de transfer ncia de dados
97
98    //FSM L gica para controle do estado atual (sequencial)
99    always @ (posedge clk)
100     if (!rst_n) //reset ass ncrono
101       estado_atual <= IDLE_STATE; //reinicia a FSM
102     else
103       estado_atual <= prox_estado; //atualiza o estado atual
104
105   //FSM L gica para controle do estado atual (combinacional)
106   always @ ( * )
107     case (estado_atual)
108       IDLE_STATE://quando no estado inicial
```

```
109         if (~irda_reg1)
110           prox_estado = ATRASO_900us; //passa ao estado seguinte
111         else
112           prox_estado = IDLE_STATE; //reinicia a FSM
113 
114       ATRASO_900us: //quando no estado de atraso de 900us
115         if (irda_posedge)
116           begin
117             if (verifica_900us)
118               prox_estado = ATRASO_450us; //passa ao estado seguinte
119             else
120               prox_estado = IDLE_STATE; //reinicia a FSM
121           end
122         else  //previne infer ncia de latches
123           prox_estado = ATRASO_900us; //permanece no estado atual
124 
125       ATRASO_450us: //quando no estado de atraso de 450us
126         if (irda_negedge)
127           begin
128             if (verifica_450us)
129               prox_estado = DATA_STATE; //passa ao estado seguinte
130             else
131               prox_estado = IDLE_STATE; //reinicia a FSM
132           end
133         else //previne infer ncia de latches
134           prox_estado = ATRASO_450us; //permanece no estado atual
135 
136       DATA_STATE: //quando no estado de transfer ncia de dados
137         if ((data_cnt == 6'd32) & irda_reg2 & irda_reg1) //verifica se recebeu os
              32 bits
138           prox_estado = IDLE_STATE; //passa ao estado seguinte
139         else if (error_flag) //caso apresente erro nos dados de irda
140           prox_estado = IDLE_STATE;  //reinicia a FSM
141         else
142           prox_estado = DATA_STATE; //permanece no estado atual
143       default:
144         prox_estado = IDLE_STATE; //recupera de estado inv lido , reiniciando a
              FSM
145     endcase
```

```
146
147   //FSM L gica para controle das sa das
148   always @ (posedge clk)
149     if (!rst_n) //reset ass ncrono
150       begin
151         data_cnt <= 6'd0; //reinicia o contador de bits de dados irda
152         get_data <= 32'd0; //limpa os registradores para os 32 bits de dados irda
153         error_flag <= 1'b0; //limpa a flag de erro de dados irda
154       end
155     else if (estado_atual == IDLE_STATE) //quando no estado inicial
156       begin
157         data_cnt <= 6'd0; //reinicia o contador de bits de dados irda
158         get_data <= 32'd0; //limpa os registradores para os 32 bits de dados irda
159         error_flag <= 1'b0; //limpa a flag de erro de dados irda
160       end
161     else if (estado_atual == DATA_STATE) //quando no estado de transmiss o de
          dados
162       begin
163         if (irda_posedge)  //verifica se    borda de subida
164           begin
165             if (!low)  //caso n o seja n vel l gico baixo (n vel l gico 0,
                  560us)
166               error_flag <= 1'b1; //define flag de erro
167           end
168         else if (irda_negedge)  //verifica se    borda de descida
169           begin
170             if (low) //caso seja n vel l gico baixo (n vel l gico 0, 560us)
171               get_data[0] <= 1'b0;
172             else if (high) //caso seja n vel l gico alto (n vel l gico 1, 1680
                  us)
173               get_data[0] <= 1'b1;
174             else
175               error_flag <= 1'b1; //caso contr rio , define flag de erro
176
177             get_data[31:1] <= get_data[30:0]; //atualiza o registrador de dados,
                  deslocando 1 bit
178             data_cnt <= data_cnt + 1'b1; //incrementa o contador de dados
179           end
180       end
```

```
181
182
183   //L gica para o controle dos displays BCD7SEG ----------------------------
184   always @ (posedge clk)
185     if (!rst_n)
186       irda_data <= 16'd0;
187     else if ((data_cnt ==6'd32) & irda_reg1)
188   begin
189     led1 <= get_data[7:0];  //Complemento dos dados
190     led2 <= get_data[15:8]; //C digo dos dados
191     led3 <= get_data[23:16];//C digo de usu rio
192     led4 <= get_data[31:24];
193   end
194
195   //Exibe nos display de BCD7SEG a tecla pressionado no controle remoto IR
196   always@(led2)
197   begin
198     case(led2)
199     //c digo no controle IR : decodifica  o BCD7SEG
200       8'b01101000: led_db = 8'b1100_0000;  //exibe 0 no display
201       8'b00110000: led_db = 8'b1111_1001;  //exibe 0 no display
202       8'b00011000: led_db = 8'b1010_0100;  //exibe 0 no display
203       8'b01111010: led_db = 8'b1011_0000;  //exibe 0 no display
204       8'b00010000: led_db = 8'b1001_1001;  //exibe 0 no display
205       8'b00111000: led_db = 8'b1001_0010;  //exibe 0 no display
206       8'b01011010: led_db = 8'b1000_0010;  //exibe 0 no display
207       8'b01000010: led_db = 8'b1111_1000;  //exibe 0 no display
208       8'b01001010: led_db = 8'b1000_0000;  //exibe 0 no display
209       8'b01010010: led_db = 8'b1001_0000;  //exibe 0 no display
210       default:     led_db = 8'b1000_1110;  //exibe F no display
211     endcase
212   end
213
214 endmodule
```

Apêndice K Decodificador Controle Remoto

```
1 module MUX_controle(
```

```
    input wire [7:0] led_db,     // Sinal de entrada do controle IR
    input wire reset, clk,       // Sinais de reset e clock
    output reg [9:0] botao,      // Sa das para os bot es num ricos (0-9)
    output reg prev, next, play, defaul, // Sa das para bot es de controle
    output reg [1:0] vol,        // Sa das para controle de volume
    output wire chave_alteracao, // Sa da para chave de altera  o
   output wire volmais
);

    reg count = 1'b0;
   reg aux2 = 1'b0;

    assign chave_alteracao = count;
   assign volmais = aux2;

    always @(posedge clk)
   begin
        if (led_db == 8'b1_111_1001) begin  // Serve para o bot o 1 ativar a
            chave de altera  o
            count <= ~count;
        end
       if (led_db == 8'b1_000_0110) begin  // Serve para o bot o 1 ativar a chave
            de altera  o
            aux2 <= ~aux2;
        end
    end

    always @(posedge clk) begin
        // Resetando as sa das para garantir que o pulso seja gerado corretamente
        botao <= 10'b0;
        prev <= 1'b0;
        next <= 1'b0;
        play <= 1'b0;
        defaul <= 1'b0;
        vol <= 2'b0;

        case (led_db)
            // C digo do controle IR : decodifica  o BCD7SEG
            8'b1_100_0000: botao[0] <= 1'b1;   // Pulso para o bot o 0
```

```
            8'b1_111_1001: botao[1] <= 1'b1;   // Pulso para o bot o 1
            8'b1_010_0100: botao[2] <= 1'b1;   // Pulso para o bot o 2
            8'b1_011_0000: botao[3] <= 1'b1;   // Pulso para o bot o 3
            8'b1_001_1001: botao[4] <= 1'b1;   // Pulso para o bot o 4
            8'b1_001_0010: botao[5] <= 1'b1;   // Pulso para o bot o 5
            8'b1_000_0010: botao[6] <= 1'b1;   // Pulso para o bot o 6
            8'b1_111_1000: botao[7] <= 1'b1;   // Pulso para o bot o 7
            8'b1_000_0000: botao[8] <= 1'b1;   // Pulso para o bot o 8
            8'b1_001_0000: botao[9] <= 1'b1;   // Pulso para o bot o 9
            8'b1_010_0000: prev <= 1'b1;       // Pulso para o bot o PREV
            8'b1_000_0011: next <= 1'b1;       // Pulso para o bot o NEXT
            8'b1_100_0110: play <= 1'b1;       // Pulso para o bot o PLAY
            8'b1_010_0001: vol[0] <= 1'b1;     // Pulso para o bot o VOL-
            8'b1_000_0110: vol[1] <= 1'b1;     // Pulso para o bot o VOL+
            default: defaul <= 1'b1;           // Pulso para a condi   o default
        endcase
    end
endmodule
```

Apêndice L Decodificador de 7 segmentos Sensor de Temperatura

```
module lcd_temp (
    input clk,
    input [7:0] seg,      // Ajustado para 7 bits
    input [3:0] dig,
    output reg centena,  // Ajustado para 4 bits
    output reg [3:0] dezena,
    output reg [3:0] unidade,
    output reg [3:0] decimos
);

reg [3:0] valor_transformado1;
reg [3:0] valor_transformado2;
reg [3:0] valor_transformado3;
reg [3:0] valor_transformado4;

always @(posedge clk) begin
```

```
17    case (seg[6:0])                          // transforma  o de 7 segmentos para
          valores de 4 bits
18        7'b1000000 : begin  // 0
19            if (dig == 4'b1110) valor_transformado1 <= 4'b0000;
20            if (dig == 4'b1101) valor_transformado2 <= 4'b0000;
21            if (dig == 4'b1011) valor_transformado3 <= 4'b0000;
22            if (dig == 4'b0111) valor_transformado4 <= 4'b0000;
23        end
24        7'b1111001 : begin  // 1
25            if (dig == 4'b1110) valor_transformado1 <= 4'b0001;
26            if (dig == 4'b1101) valor_transformado2 <= 4'b0001;
27            if (dig == 4'b1011) valor_transformado3 <= 4'b0001;
28            if (dig == 4'b0111) valor_transformado4 <= 4'b0001;
29        end
30        7'b0100100 : begin  // 2
31            if (dig == 4'b1110) valor_transformado1 <= 4'b0010;
32            if (dig == 4'b1101) valor_transformado2 <= 4'b0010;
33            if (dig == 4'b1011) valor_transformado3 <= 4'b0010;
34            if (dig == 4'b0111) valor_transformado4 <= 4'b0010;
35        end
36        7'b0110000 : begin  // 3
37            if (dig == 4'b1110) valor_transformado1 <= 4'b0011;
38            if (dig == 4'b1101) valor_transformado2 <= 4'b0011;
39            if (dig == 4'b1011) valor_transformado3 <= 4'b0011;
40            if (dig == 4'b0111) valor_transformado4 <= 4'b0011;
41        end
42        7'b0011001 : begin  // 4
43            if (dig == 4'b1110) valor_transformado1 <= 4'b0100;
44            if (dig == 4'b1101) valor_transformado2 <= 4'b0100;
45            if (dig == 4'b1011) valor_transformado3 <= 4'b0100;
46            if (dig == 4'b0111) valor_transformado4 <= 4'b0100;
47        end
48        7'b0010010 : begin  // 5
49            if (dig == 4'b1110) valor_transformado1 <= 4'b0101;
50            if (dig == 4'b1101) valor_transformado2 <= 4'b0101;
51            if (dig == 4'b1011) valor_transformado3 <= 4'b0101;
52            if (dig == 4'b0111) valor_transformado4 <= 4'b0101;
53        end
54        7'b0000010 : begin  // 6
```

```
55              if (dig == 4'b1110) valor_transformado1 <= 4'b0110;
56              if (dig == 4'b1101) valor_transformado2 <= 4'b0110;
57              if (dig == 4'b1011) valor_transformado3 <= 4'b0110;
58              if (dig == 4'b0111) valor_transformado4 <= 4'b0110;
59          end
60          7'b1111000 : begin  // 7
61              if (dig == 4'b1110) valor_transformado1 <= 4'b0111;
62              if (dig == 4'b1101) valor_transformado2 <= 4'b0111;
63              if (dig == 4'b1011) valor_transformado3 <= 4'b0111;
64              if (dig == 4'b0111) valor_transformado4 <= 4'b0111;
65          end
66          7'b0000000 : begin  // 8
67              if (dig == 4'b1110) valor_transformado1 <= 4'b1000;
68              if (dig == 4'b1101) valor_transformado2 <= 4'b1000;
69              if (dig == 4'b1011) valor_transformado3 <= 4'b1000;
70              if (dig == 4'b0111) valor_transformado4 <= 4'b1000;
71          end
72          7'b0010000 : begin  // 9
73              if (dig == 4'b1110) valor_transformado1 <= 4'b1001;
74              if (dig == 4'b1101) valor_transformado2 <= 4'b1001;
75              if (dig == 4'b1011) valor_transformado3 <= 4'b1001;
76              if (dig == 4'b0111) valor_transformado4 <= 4'b1001;
77          end
78      endcase
79
80      // Atribui os valores corretos para os displays
81      decimos <= valor_transformado1;
82      unidade <= valor_transformado2;
83      dezena  <= valor_transformado3;
84      centena <= valor_transformado4;
85  end
86
87  endmodule
```

9 Anexos

Anexo A I2C READ Sensor de Temperatura

```
1  //Arquivo:   I2C_READ_verilog.v
2  //Modulo:    I2C_READ_verilog(clock, rst_n, scl,sda, data)
3  //Descri  o: Interface de Comunica  o I2C LM75A
4  //Autor:     Malki- edheq  Benjamim
5  //Data:      27/01/2022
6  module I2C_READ_verilog(
7    input clk,       //clock FPGA 50MHz
8    input rst_n,     //Reset ass ncrono ativo-baixo
9    output reg scl,  //SCL (clock do barramento I2C)
10   inout  sda,      //SDA (barramento de dados I2C)
11   output [15:0] data //Dado de temperatura
12 );
13
14   //Declara  o de Registradores
15   reg [15:0] data_r;    //Registrador para dado de temperatura
16   reg sda_r;          //Registrador para SDA
17   reg sda_link;       //Flag para dire  o SDA (in/out)
18   reg [7:0] scl_cnt;    //contador gerar clock SCL
19   reg [2:0] cnt;        //contador auxiliar para clock SCL
20   reg [25:0] timer_cnt; //timer para ler temperatura a cada 1s
21   reg [3:0] data_cnt;   //Registrador para convers o Serial-Paralela
22   reg [7:0] addr_reg;   //Endere o do dispositivo I2C
23   reg [8:0] estado;     //Registrador para estados
24
25   /*INICIO: B1, B2 e B3 - Gera  o de clk para barramento SCL*/
26   //Conta um per odo de clock de SCL
27   `define CNT_OVER 8'd199
28   always@(posedge clk or negedge rst_n) begin : B1
```

```
29    if (!rst_n) scl_cnt <= 8'd0;
30    else if (scl_cnt == `CNT_OVER) scl_cnt <= 8'd0; //4us (min 2.5ns)
31    else scl_cnt <= scl_cnt + 1'b1;
32  end
33  //Define transi  es e n veis do clock SCL, com base no per odo
34  always@(posedge clk or negedge rst_n) begin : B2
35    if (!rst_n) cnt <= 3'd4;
36    else
37      case(scl_cnt)
38        8'd49  : cnt <= 3'd1;//SCL n vel alto  (1us)
39        8'd99  : cnt <= 3'd2;//borda de descida SCL (2us)
40        8'd149 : cnt <= 3'd3;//SCL n vel baixo  (3us)
41        8'd199 : cnt <= 3'd0;//borda de subida SCL (4us)
42         default: cnt <= 3'd4;//SCL indefinido
43      endcase
44  end
45  `define SCL_HIG (cnt == 3'd1) //SCL n vel alto (1us)
46  `define SCL_NEG (cnt == 3'd2) //na borda de descida SCL (2us)
47  `define SCL_LOW (cnt == 3'd3) //SCL n vel baixo (3us)
48  `define SCL_POS (cnt == 3'd0) //na borda de subida SCL (4us)
49
50  //produz o sinal de clock na sa da SCL
51  always@(posedge clk or negedge rst_n) begin : B3
52    if (!rst_n) scl <= 1'b0;
53    else if (`SCL_POS) scl <= 1'b1;//ap s borda subida, scl alto
54    else if (`SCL_NEG) scl <= 1'b0;//ap s borda descida, scl baixo
55  end
56  /*FIM: B1, B2 e B3 : Gerar clk para barramento SCL*/
57
58  /*INICIO: B4 - Leitura de dados de temperatura a cada 1s */
59  `define TIMER_OVER 26'd49999999
60  always@(posedge clk or negedge rst_n) begin : B4
61    if (!rst_n) timer_cnt <= 26'd0;  //1Hz -> 1seg
62    else if (timer_cnt == `TIMER_OVER) timer_cnt <= 26'd0;
63    else timer_cnt <= timer_cnt + 1'b1;
64  end
65  //Defini  o da fsm (formato one-hot)
66  // vide datasheet LM75A Texas Instrument
67  // SNOS808P   JANUARY   2000  REVISED   DECEMBER 2014 PAG 7-8
```

```
68  // Fig(a) Typical 2-Byte Read From Preset Pointer Location
69  localparam  IDLE    = 9'b0_0000_0000,
70          START   = 9'b0_0000_0010, //inicia comunica  o com escravo
71          ADDRESS = 9'b0_0000_0100, //envia endere o do escravo
72          ACK1    = 9'b0_0000_1000, //confirma  o pelo escravo
73          READ1   = 9'b0_0001_0000, //leitura do 1byte MSB (temperatura)
74          ACK2    = 9'b0_0010_0000, //confirma  o pelo mestre
75          READ2   = 9'b0_0100_0000, //leitura do 1byte LSB (temperatura)
76          NACK    = 9'b0_1000_0000, //mestre n o responde ao escravo
77          STOP    = 9'b1_0000_0000; //finaliza comunica  o com escravo
78
79  //Endere o do dispositivo _ opera  o leitura
80  `define DEVICE_ADDRESS 8'b1001_0001
81
82  //B5 : Descri  o da FSM
83  always@(posedge clk or negedge rst_n) begin : B5
84    if (!rst_n)
85      begin
86        data_r      <= 16'd0; //limpa o registrador de dados
87        sda_r       <= 1'b1;  //transmite um bit 1
88        sda_link    <= 1'b1;  //habilita sa da SDA (escrita)
89        estado      <= IDLE;  //reinicia a FSM
90        addr_reg  <= 8'd0;  //endere o inicial 0000_0000
91        data_cnt    <= 4'd0;  //reinicia contador de dados
92      end
93    else
94      case(estado)
95        IDLE: //Estado Inicial
96          begin
97            sda_r     <= 1'b1; //transmite um bit 1
98            sda_link <= 1'b1; //habilita sa da SDA (escrita)
99            if (timer_cnt == `TIMER_OVER) //periodo conclu do
100             estado <= START; //inicia a FSM
101           else  estado <= IDLE;
102         end
103       START://Mestre inicia comunica  o com escravo
104         begin
105           if (`SCL_HIG) begin //caso SCL n vel alto
106             sda_r       <= 1'b0; //transmite um bit 0
```

```
107             sda_link    <= 1'b1; //habilita sa da SDA (escrita)
108             addr_reg  <= `DEVICE_ADDRESS;
109             estado      <= ADDRESS; //pr ximo estado endere amento
110             data_cnt    <= 4'd0; //reinicia o contador de dados
111           end
112           else estado <= START; //permanece no estado atual
113         end
114       ADDRESS://Mestre envia endere o e opera  o para escravo
115         begin
116           if(`SCL_LOW) begin//caso SCL n vel baixo
117             //Endere amento conclu do, o SDA muda de dire  o
118             //e o dispositivo est  pronto para emitir um sinal de resposta
119             if(data_cnt == 4'd8) begin
120                 estado   <= ACK1; //pr ximo estado, confirma  o (ACK)
121                 data_cnt <= 4'd0; //reinicia o contador de dados
122                 sda_r    <= 1'b1;
123                 sda_link <= 1'b0; //SDA alta imped ncia (leitura)
124             end
125             else begin//Durante o endere amento, o SDA atua como entrada

126               estado   <= ADDRESS; //permanence no estado
127               data_cnt <= data_cnt + 1'b1; //incremente o contador de dados
128               case(data_cnt) //convers o paralela-serial, SDA transmite o
                      endere o
129                 4'd0: sda_r <= addr_reg[7];  //transmite o MSB do endere o
130                 4'd1: sda_r <= addr_reg[6];
131                 4'd2: sda_r <= addr_reg[5];
132                 4'd3: sda_r <= addr_reg[4];
133                 4'd4: sda_r <= addr_reg[3];
134                 4'd5: sda_r <= addr_reg[2];
135                 4'd6: sda_r <= addr_reg[1];
136                 4'd7: sda_r <= addr_reg[0]; //transmite o LSB do endere o
137                 default: ;
138               endcase
139             end
140           end
141           else  estado <= ADDRESS;  //permanece no estado enquanto SCL n vel
                    alto
142         end
```

```
143         ACK1://Aguarda confirma  o do escravo
144           begin
145             //inicia a leitura do LSByte
146             if ((!sda && (`SCL_HIG))||(`SCL_NEG)) estado <= READ1;
147             else estado <= ACK1; //permanece no estado
148           end
149         READ1://Leitura de dados do escravo, MSB (1byte)
150           begin
151             //Ao concluir a leitura do MSB 1byte,
152             //o SDA muda de dire  o e
153             //o host est  pronto para emitir um sinal de resposta
154             if((`SCL_LOW) && (data_cnt == 4'd8))
155               begin
156                 estado   <= ACK2;
157                 data_cnt <= 4'd0; //reinicia o contador de dados
158                 sda_r    <= 1'b1; //transmite um bit 1
159                 sda_link <= 1'b1; //habilita sa da SDA (escrita)
160               end
161             //Durante a leitura de dados, o dispositivo atua como uma sa da
162             else if(`SCL_HIG)
163               begin
164                 data_cnt <= data_cnt + 1'b1; //incrementa o contador de dados
165                 case(data_cnt) //convers o serial-paralela
166                   4'd0: data_r[15] <= sda; //MSB da leitura (do byte lido)
167                   4'd1: data_r[14] <= sda;
168                   4'd2: data_r[13] <= sda;
169                   4'd3: data_r[12] <= sda;
170                   4'd4: data_r[11] <= sda;
171                   4'd5: data_r[10] <= sda;
172                   4'd6: data_r[9]  <= sda;
173                   4'd7: data_r[8]  <= sda; //LSB da leitura (do byte lido)
174                   default: ; //sem a   o , aguarda pr ximo valor de data_cnt
175                 endcase
176               end
177             else  estado <= READ1;  //permanece no estado
178           end
179         ACK2://Mestre responde ao escravo
180           begin
181             if(`SCL_LOW) sda_r <= 1'b0; //transmite um bit 0
```

```
182             else if(`SCL_NEG)
183               begin
184                 sda_r    <= 1'b1;
185                 sda_link <= 1'b0; //SDA alta imped ncia (leitura)
186                 estado   <= READ2; //pr ximo estado, leitura LSB
187               end
188             else estado <= ACK2; //permanece no estado
189           end
190         READ2://Leitura de dados do escravo, LSB (1byte)
191           begin
192             //Ao concluir a leitura do LSB 1byte,
193             //o SDA muda de dire   o e
194             //o host est  pronto para emitir um sinal de resposta
195             if((`SCL_LOW) && (data_cnt == 4'd8)) begin
196               estado   <= NACK; //pr ximo estado
197               data_cnt <= 4'd0; //reinicia o contador de dados
198               sda_r    <= 1'b1; //transmite um bit 1
199               sda_link <= 1'b1; //habilita sa da SDA (escrita)
200             end
201             else if(`SCL_HIG)
202               begin
203                 data_cnt <= data_cnt + 1'b1; //incrementa o contador de dados
204                 case(data_cnt)  //convers o serial-paralela
205                   4'd0: data_r[7] <= sda; //MSB da leitura (do byte lido)
206                   4'd1: data_r[6] <= sda;
207                   4'd2: data_r[5] <= sda;
208                   4'd3: data_r[4] <= sda;
209                   4'd4: data_r[3] <= sda;
210                   4'd5: data_r[2] <= sda;
211                   4'd6: data_r[1] <= sda;
212                   4'd7: data_r[0] <= sda; //LSB da leitura (do byte lido)
213                   default: ;  //sem a   o , aguarda pr ximo valor de data_cnt
214                 endcase
215               end
216             else estado <= READ2; //permanece no estado
217           end
218         NACK://Mestre para de responder ao escravo
219           begin
220             if(`SCL_LOW) begin
```

```
221                 estado <= STOP; //pr ximo estado
222                 sda_r <= 1'b0; //transmite um bit 0
223               end
224             else estado <= NACK; //permanece no estado
225           end
226         STOP: //Finaliza a comunica  o com o escravo
227           begin
228             if(`SCL_HIG) begin
229               estado <= IDLE; //reinicia a FSM
230               sda_r <= 1'b1; //transmite um bit 1
231             end
232             else estado <= STOP; //permanece no estado
233           end
234         default: estado <= IDLE; //recupera de estado inv lido
235       endcase
236   end
237   /*FIM: B4, B5 - Leitura de dados de temperatura a cada 1s */
238
239   //Atribui  o cont nua
240   //se sda_link==1 ent o sda transmite sda_r, caso contr rio alta imped ncia
241   assign sda  = sda_link ? sda_r : 1'bz; //infere triestado
242   assign data = data_r; //atualiza a sa da com os dados lidos (2 Bytes)
243
244 endmodule
```

Anexo B Decodificador I2C Sensor de Temperatura

```
1  //Arquivo:   SEG_D_verilog.v
2  //Modulo:    SEG_D_verilog(clock, rst_n, data, seg, dig)
3  //Descri  o: Decodificador  BCD7SEG LM75A de 0-125 C  POSITIVO
4  //Autor:     Malki- edheq  Benjamim
5  //Data:      27/01/2022
6  module SEG_D_verilog (
7    input clk,rst_n,
8    input [15:0] data, //2bytes
9    output reg [7:0] seg, //seg[7] -> dp
10   output reg [4:1] dig //seleciona display
11
```

```
);

  reg [3:0] dataout_buf;
  reg [1:0] disp_dat;
  reg [16:0] delay_cnt;

  //gera clk de 1kHz (sugest o: clk externo)
  always@(posedge clk,negedge rst_n) begin
    if(!rst_n) delay_cnt <= 16'd0;
    else if(delay_cnt == 16'd50000)
      delay_cnt <= 16'd0;
    else delay_cnt<= delay_cnt + 1'b1;
  end

  /* IN CIO: Controle do Display 7seg */
  //redefine o display ativo a cada 1ms (delay)
  always@(posedge clk,negedge rst_n) begin
    if(!rst_n) disp_dat <= 2'd0;
    else if (delay_cnt == 16'd50000)
     disp_dat <= disp_dat + 1'b1;
    else disp_dat <= disp_dat;
  end
  //seleciona o display a ser ativo
  always@(disp_dat) begin
    case(disp_dat)
      2'b00: dig = 4'b1110; //habilita o display 4
      2'b01: dig = 4'b1101; //habilita o display 3
      2'b10: dig = 4'b1011; //habilita o display 2
      2'b11: dig = 4'b0111; //habilita o display 1
      default dig = 4'b1111;//todos desabilitados
    endcase
  end
  // Decoder BCD7seg (sugest o: decoder externo)
  always@(dataout_buf, dig) begin
    case(dataout_buf)
      4'h0 : seg[6:0] = 7'b1000000; //0
      4'h1 : seg[6:0] = 7'b1111001; //1
      4'h2 : seg[6:0] = 7'b0100100; //2
      4'h3 : seg[6:0] = 7'b0110000; //3
```

```
51      4'h4 : seg[6:0] = 7'b0011001; //4
52      4'h5 : seg[6:0] = 7'b0010010; //5
53      4'h6 : seg[6:0] = 7'b0000010; //6
54      4'h7 : seg[6:0] = 7'b1111000; //7
55      4'h8 : seg[6:0] = 7'b0000000; //8
56      4'h9 : seg[6:0] = 7'b0010000; //9
57      default : seg[6:0] = 7'b1000000;//0
58    endcase
59    if (dig == 4'b1101) //Unidade da temperatura
60      seg[7] = 1'b0; // dp ativo  para dig unidade
61    else seg[7] = 1'b1; //dp inativo demais digs
62  end
63  /* FIM: Controle do Display 7seg */
64
65  /* INICIO: MSB+LSB 9bits para 4BCD  */
66  always@(dig, data) begin
67    case(dig)
68      //dado display 4 (decimal da temperatura)
69      //data[7] decimal da temperatura (0 ->.0  , 1->.5  )
70      4'b1110:  dataout_buf = data[7] ? 4'h5 : 4'h0;
71      //dado display 3 (unidade da temperatura)
72      4'b1101: begin //data[15:8] MSB+LSB da temperatura, data[15] sinal (0 -> +,
            1-> -)
73        if (data[15]) dataout_buf = 4'd0; //caso temp negativa valor exibido 0
74        else
75          if (data[14:8] >= 10 && data[14:8] < 20) dataout_buf = data[14:8] - 7'
              d10;
76          else if (data[14:8] >= 20 && data[14:8] < 30) dataout_buf = data[14:8] -
                7'd20;
77          else if (data[14:8] >= 30 && data[15:8] < 40) dataout_buf = data[14:8] -
                7'd30;
78          else if (data[14:8] >= 40 && data[15:8] < 50) dataout_buf = data[14:8] -
                7'd40;
79          else if (data[14:8] >= 50 && data[15:8] < 60) dataout_buf = data[14:8] -
                7'd50;
80          else if (data[14:8] >= 60 && data[15:8] < 70) dataout_buf = data[14:8] -
                7'd60;
81          else if (data[14:8] >= 70 && data[15:8] < 80) dataout_buf = data[14:8] -
                7'd70;
```

```
82          else if (data[14:8] >= 80 && data[15:8] < 90) dataout_buf = data[14:8] -
                7'd80;
83          else if (data[14:8] >= 90 && data[15:8] < 100) dataout_buf = data[14:8]
               - 7'd90;
84          else if (data[14:8] >= 100 && data[15:8] < 110) dataout_buf = data[14:8]
                - 7'd100;
85          else if (data[14:8] >= 110 && data[15:8] < 120) dataout_buf = data[14:8]
                - 7'd110;
86          else if (data[14:8] >= 120) dataout_buf = data[14:8] - 7'd120;
87          else dataout_buf = data[11:8]; //apenas unidade
88      end
89      //dado display 2 (dezena da temperatura)
90      4'b1011: begin //data[15:8] MSB+LSB da temperatura, data[15] sinal (0 -> +,
            1-> -)
91        if (data[15]) dataout_buf = 4'd0; //caso temp negativa valor exibido 0
92        else
93          if (data[14:8] >= 10 && data[14:8] < 20) dataout_buf = 4'h1;
94          else if (data[14:8] >= 20 && data[14:8] < 30) dataout_buf = 4'h2;
95          else if (data[14:8] >= 30 && data[14:8] < 40) dataout_buf = 4'h3;
96          else if (data[14:8] >= 40 && data[14:8] < 50) dataout_buf = 4'h4;
97          else if (data[14:8] >= 50 && data[14:8] < 60) dataout_buf = 4'h5;
98          else if (data[14:8] >= 60 && data[14:8] < 70) dataout_buf = 4'h6;
99          else if (data[14:8] >= 70 && data[14:8] < 80) dataout_buf = 4'h7;
100         else if (data[14:8] >= 80 && data[14:8] < 90) dataout_buf = 4'h8;
101         else if (data[14:8] >= 90 && data[14:8] < 100) dataout_buf = 4'h9;
102         else if (data[14:8] >= 110 && data[14:8] < 120) dataout_buf = 4'h1;
103         else if (data[14:8] >= 120) dataout_buf <= 4'h2;
104         else dataout_buf <= 4'b0;//dezena zero
105     end
106     //dado display 1  (centena da temperatura)
107     4'b0111: begin //data[15:8] MSB+LSB da temperatura, data[15] sinal (0 -> +,
            1-> -)
108       if (data[15]) dataout_buf = 4'd0; //caso temp negativa valor exibido 0
109       else dataout_buf = (data[14:8] >= 100) ? 4'h1 : 4'h0;
110     end
111     default : dataout_buf = 4'b0;
112   endcase
113 end
114 /* INICIO: MSB+LSB 9bits para 4BCD  */
```

```
115
116 endmodule
```