
**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 2

Alysson Lucas Pontes Cavalcante da Silva

Felipe Rafael Barros da Silva

Maria Victória Martins Neves

Recife, 2024

# Sumário

# 1 Introdução

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

O objetivo geral deste projeto é projetar e implementar um relógio digital que, além de exibir horas e minutos, seja capaz de programar e acionar três alarmes.

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

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

# 2 Desenvolvimento

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

## 2.1 Linguagem utilizada

Para a implementação deste projeto, foi utilizada a linguagem AHDL (Altera Hardware Description Language), uma linguagem de descrição de hardware específica para a família de dispositivos FPGA da Altera. O AHDL é uma ferramenta poderosa para a criação e simulação de circuitos digitais, permitindo a descrição do comportamento e da estrutura de um circuito de forma textual.

Uma das principais vantagens da AHDL é a sua simplicidade e flexibilidade, o que a torna ideal para projetos que envolvem circuitos lógicos, como o relógio digital desenvolvido. A linguagem permite a definição de funções lógicas e temporizações de forma clara e objetiva, facilitando a modelagem dos diferentes módulos que compõem o relógio, como o contador de segundos, minutos e horas, bem como os alarmes.

Além disso, a AHDL oferece um bom suporte para integração com outros componentes e módulos desenvolvidos dentro do ambiente de design da Altera (Quartus), o que possibilitou o controle preciso de entradas e saídas, como as chaves e botões para configuração dos alarmes e o display de 7 segmentos para visualização do tempo.

Essa escolha de linguagem proporcionou a flexibilidade necessária para a implementação do relógio digital com múltiplos alarmes, permitindo que o circuito fosse ajustado de acordo com os requisitos do projeto e as limitações do hardware disponível.

## 2.2 Conversores de clock

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

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

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

O conversor é configurado com dois parâmetros: n, que define o número de flip-flops para a contagem binária, e count, que especifica o valor de contagem. O código reduz a frequência do clock principal usando flip-flops tipo T. Cada flip-flop alterna seu estado a cada pulso do clock, com o clock de cada flip-flop subsequente alimentado pela saída do flip-flop anterior. A contagem vai até o valor especificado (count) e gera um sinal de clock com frequência reduzida como saída. O flip-flop auxiliar gera o clock convertido, que alterna após atingir a contagem desejada e reinicia o contador. Quando a contagem (count) é alcançada, todos os flip-flops são resetados para reiniciar a contagem.

São necessários quatro conversores de clock. Os três primeiros geram: um clock de um segundo, um clock muito rápido para acionar os displays simultaneamente, e outro igualmente rápido para controlar o buzzer. O código desses três conversores está incluído no Apêndice A, e suas implementações estão destacadas na Figura 2.1.

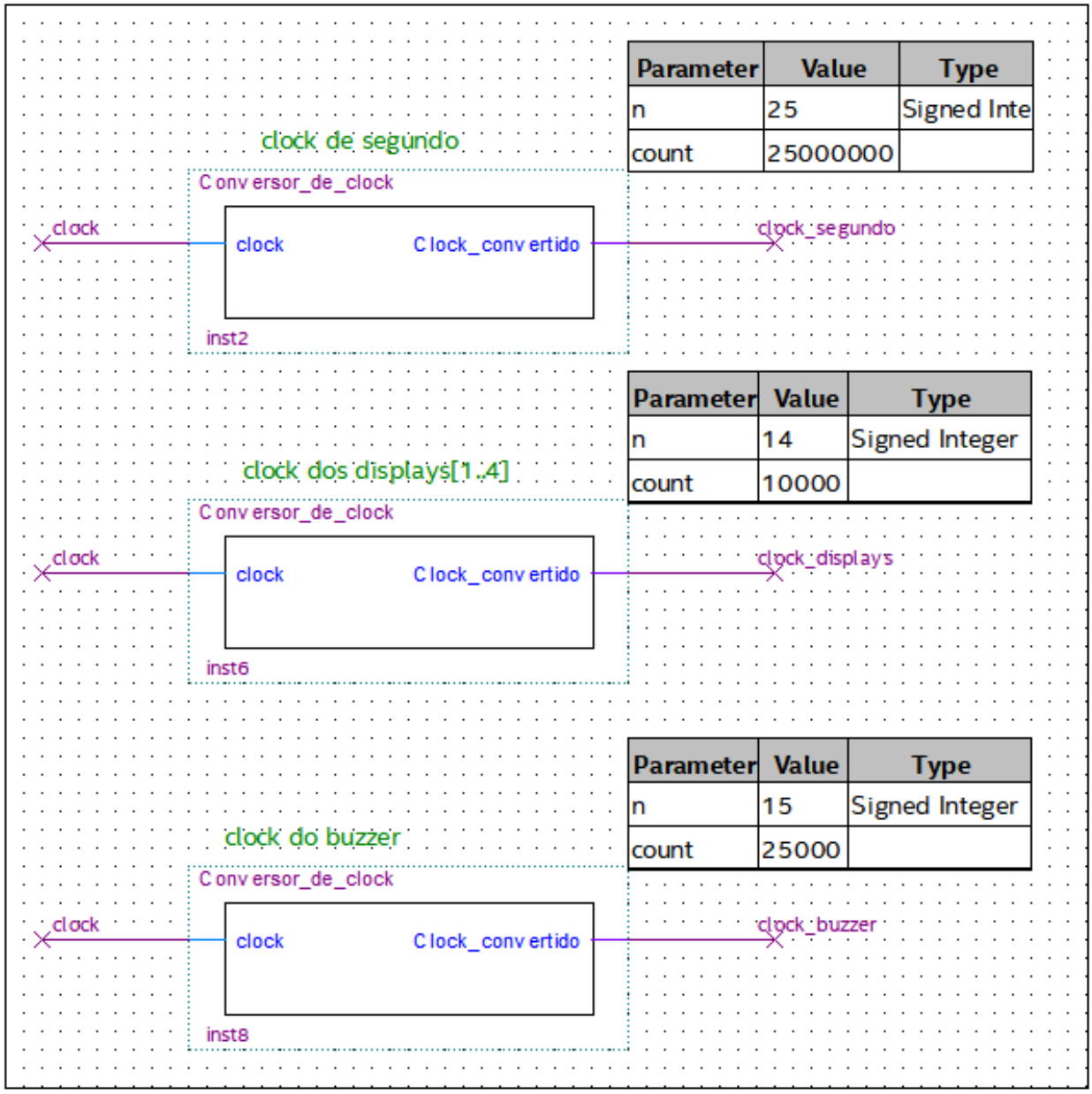


Figura 2.1: Conversores de clock: contador de segundo, ativador dos displays simultâneos e ativador do buzzer.

O quarto conversor, presente no Apêndice B, foi criado especificamente para o relógio, com a diferença principal sendo a inclusão de uma máquina de estados. Essa máquina permite que a contagem comece em um, da seguinte forma: inicialmente, o relógio recebe um clock para iniciar a contagem em um e, em seguida, entra no segundo estado, em que opera com base no clock

configurado pelos parâmetros fornecidos no código. Este último conversor não foi utilizado como símbolo no diagrama esquemático e sim incluído diretamente no código do relógio.

## 2.3 Debounce

Em eletrônica, "debounce"é o processo de eliminar ruídos eletromecânicos indesejados que ocorrem quando um interruptor é acionado. Durante o acionamento de um botão, o movimento das partes internas pode gerar pulsos elétricos instáveis antes que o contato se estabilize. Isso pode levar a comandos errados em circuitos digitais. O debounce garante que apenas uma transição estável seja reconhecida, aumentando a precisão do sistema.

No caso do projeto atual, já em ciência da possível problemática em relação ao uso dos botões, foi implementado um debounce com o objetivo de "atrasar"o sinal recebido, reconhecendo como entrada apenas um pulso.

O código em AHDL, contido no Apêndice C, implementa um circuito de debounce para quatro entradas digitais, utilizando flip-flops e conversores de clock para estabilizar os sinais e gerar clocks mais lentos. Cada flip-flop é acionado por um clock reduzido e inverte o sinal da entrada correspondente. As saídas são definidas pelos flip-flops, com a saida2 tendo um clock diferente para atender a funcionalidades específicas necessárias para o botão 2. Isso assegura que todas as entradas sejam lidas de forma estável e sem ruídos. O bloco esquemático que utiliza o Debounce para filtrar os sinais dos botões é o da Figura 2.2.

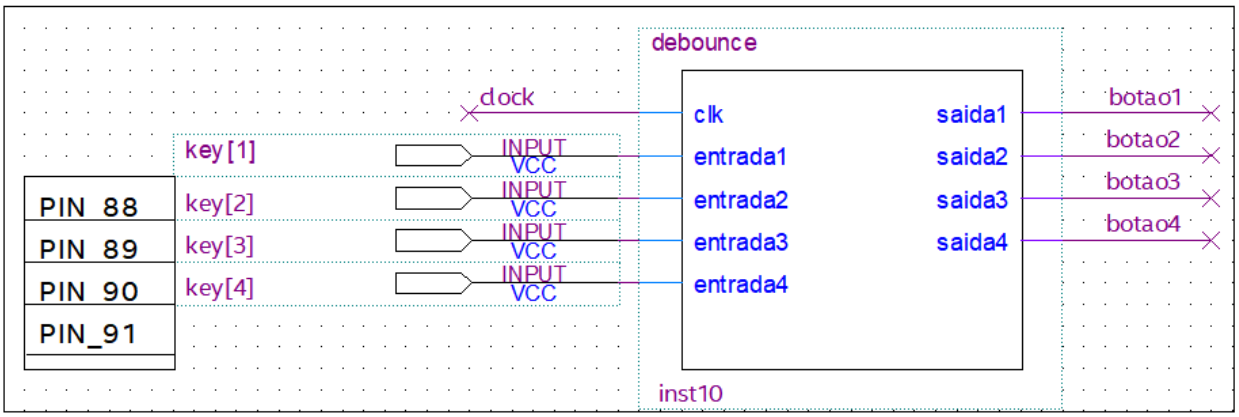


Figura 2.2: Diagrama esquemático do Debounce.

## 2.4 Contador de 4 bits

Existem dois tipos de contadores, assíncronos os síncronos, que possuem a mesma função, mas não o mesmo desempenho. Os problemas associados aos contadores assíncronos surgem devido ao acúmulo dos atrasos de propagação dos flip-flops, visto que nem todos os flip-flops mudam de estado em sincronia com os pulsos de entrada. A solução, para melhor eficiência da contagem, é a utilização de contadores síncronos ou paralelos, nos quais todos os flip-flops são acionados simultaneamente pelos pulsos de clock de entrada. Como os pulsos de clock são aplicados a todos os flip-flops, é necessário um mecanismo para controlar quando um flip-flop deve mudar de estado ou permanecer inalterado ao receber um pulso de clock.

Através dessa implementação, cria-se o contador módulo 16, que funciona como a base para a contagem do relógio. Nesse processo, são criados quatro contadores, sendo Q1 o dígito menos significativo (LSD, do inglês "Least Significant Digit") e Q4 o dígito mais significativo (MSD, do inglês "Most Significant Digit"), com condições específicas estabelecidas para os resets de cada dígito.

Cada dígito exige condições particulares, inclusive dependendo do formato de hora definido (12h ou 24h):

- Formato 24h
  - Para o dígito menos significativo dos minutos, o contador é configurado como módulo 10, a partir da condição de reset Q1=10.
  - O dígito mais significativo dos minutos utiliza um contador módulo 6, que é reiniciado em Q2=6.
  - Para o LSD (Menor Dígito Significativo) das horas, o contador pode operar em módulo 10, reiniciando quando Q3=10, ou em módulo 4, quando Q3=4 e Q4=2. Há também um reset configurado para o caso em que se tenta alterar o primeiro dígito das horas de 1 para 2, mas o segundo dígito já é maior que 3. Nesse cenário, o reset é aplicado ao segundo dígito para evitar que o horário ultrapasse 23:59.
  - O MSD (Mais Significativo Dígito) das horas é resetado quando Q4=3.
- Formato 12h

- Para o dígito menos significativo dos minutos, o contador é configurado como módulo 10, a partir da condição de reset Q1=10.
- O dígito mais significativo dos minutos utiliza um contador módulo 6, que é reiniciado em Q2=6.
- Para o LSD (Menor Dígito Significativo) das horas, o contador pode operar em módulo 10, reiniciando quando Q3=10, ou em módulo 2, quando Q3=2 e Q4=1. Há também um reset configurado para o caso em que se tenta alterar o primeiro dígito das horas de 0 para 1, mas o segundo dígito já é maior que 1. Nesse cenário, o reset é aplicado ao segundo dígito para evitar que o horário ultrapasse 11:59.
- O MSD (Mais Significativo Dígito) das horas é resetado quando Q4=2.

Foram criados dois contadores utilizando a mesma lógica: um para o relógio e outro para o alarme. A necessidade de dois contadores distintos surge porque, no caso do relógio, o clock de um dígito está conectado ao reset do dígito anterior. Isso significa que, quando o dígito menos significativo (LSD) atinge 10 e é resetado, o dígito à esquerda é incrementado em 1, e assim sucessivamente. Em contraste, no alarme, os dígitos funcionam de maneira independente, e o clock de cada um é controlado manualmente por um botão.

O trecho de código responsável pela contagem é utilizado tanto nos arquivos do relógio quanto nos do alarme. Esses arquivos serão explicados em detalhes mais adiante e estão incluídos integralmente nos Apêndices H e I.

## 2.5 Transformação de bits em segmentos

Os contadores criados realizam a contagem de 0 (zero) a 15 (quinze) em binário. No entanto, o objetivo final é exibir esses números em um display de 7 segmentos. Para isso, é necessário desenvolver um decodificador que converta os bits de saída do contador nos sinais apropriados para ativar os segmentos correspondentes do display. A Tabela 2.1 apresenta a correspondência entre cada número decimal, sua representação binária e os segmentos ativos no display de 7 segmentos.

| Número decimal | Número binário | Segmentos ativos | Entradas no display de 7 segmentos |
|---|---|---|---|
| 0 | 0000 | abcdef | 1111110 |
| 1 | 0001 | bc | 0110000 |
| 2 | 0010 | abdeg | 1101101 |
| 3 | 0011 | abcdg | 1111001 |
| 4 | 0100 | bcfg | 0110011 |
| 5 | 0101 | acdfg | 1011011 |
| 6 | 0110 | acdefg | 1011111 |
| 7 | 0111 | abc | 1110000 |
| 8 | 1000 | abcdefg | 1111111 |
| 9 | 1001 | abcfg | 1111011 |

Tabela 2.1: Decimal, binário e segmentos

O código, presente no Apêndice D, implementa um decodificador para display de 7 segmentos, que converte números binários de 4 bits em sinais que controlam os segmentos do display. As entradas representam o valor binário (de 0 a 9), e as saídas controlam os sete segmentos do display, identificados pelas letras A a G. Dessa forma, para cada combinação binária de entrada, é especificada a ativação dos segmentos correspondentes para exibir os números de 0 a 9 no display. Por exemplo, a entrada "0000"ativa os segmentos para formar o número "0", enquanto "1001"ativa os segmentos para exibir o número "9".

De forma muito mais simples do que seria necessário ao usar portas lógicas, a decodificação de bits em segmentos em AHDL é facilitada, pois a linguagem reconhece diretamente os números em binário e permite a relação direta entre os valores binários e os padrões de ativação dos segmentos. Isso simplifica a implementação e reduz a complexidade do código.

Cada decodificador é responsável por traduzir um dígito binário em sua representação decimal. Por isso, foram utilizados quatro chips, exibidos na Figura 2.3, que implementam o código do decodificador, convertendo os números binários para decimal nas saídas, que, por sua vez, serão enviados aos displays para exibição.

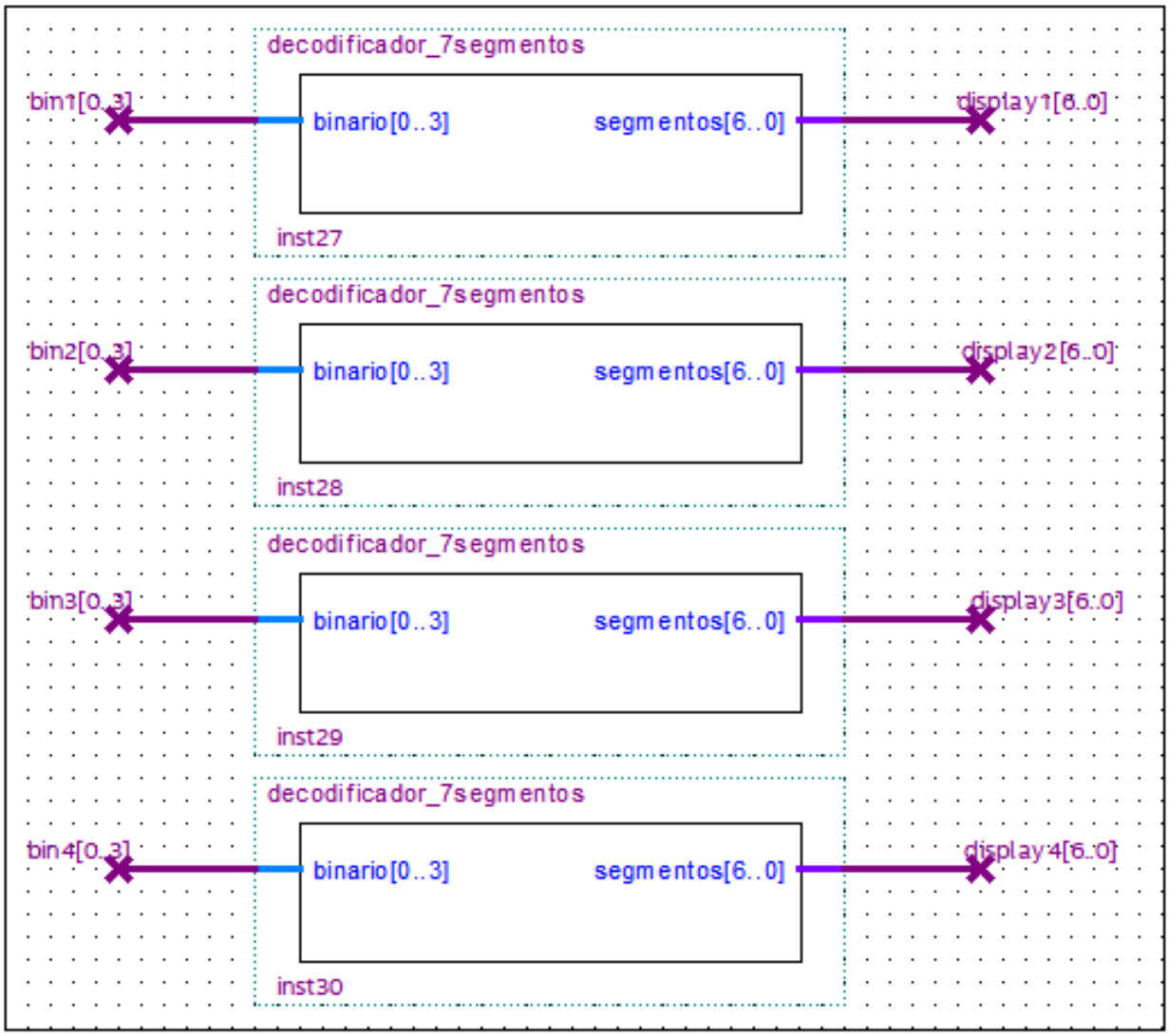


Figura 2.3: Diagrama esquemático do Decodificador.

## 2.6 Ilusão dos displays

Com os contadores e o circuito decodificador já aptos a serem implementados, é possível visualizar as contagens no display. No entanto, devido à mesma configuração de pinagem, o número exibido é idêntico nos quatro displays. Para resolver esse problema, é necessário alternar a ativação dos displays, de modo que cada um acenda individualmente em sequência. Essa alternância, idealmente, ocorre de forma tão rápida que se torna imperceptível ao olho humano, criando a ilusão de que todos os displays estão acesos simultaneamente.

Um multiplexador, ou MUX, é um dispositivo de comutação digital que seleciona uma entre várias entradas de dados e a encaminha para uma única saída. O multiplexador opera com base em sinais de controle ou seleção. Dependendo da combinação desses sinais, uma das entradas é conectada à saída.

Para garantir o funcionamento adequado do multiplexador, é necessário implementar um mecanismo que permita alternar entre os displays de forma sequencial e eficiente. Esse mecanismo é realizado através da criação de um contador, incluso no Apêndice E, que atua como um seletor do display ativo. O código utiliza flip-flops tipo T para alternar entre os quatro displays de 7 segmentos, ativando um display por vez em sequência rápida, o que cria a ilusão de que todos estão acesos simultaneamente. A cada pulso do clock de entrada, os flip-flops geram uma contagem binária de 0 a 3, e, com base nessa contagem, um display é ativado enquanto os outros permanecem desativados. Esse ciclo contínuo permite a ativação sequencial dos displays, garantindo que eles sejam atualizados de forma eficiente e seu chip é apresentado na Figura 2.4.

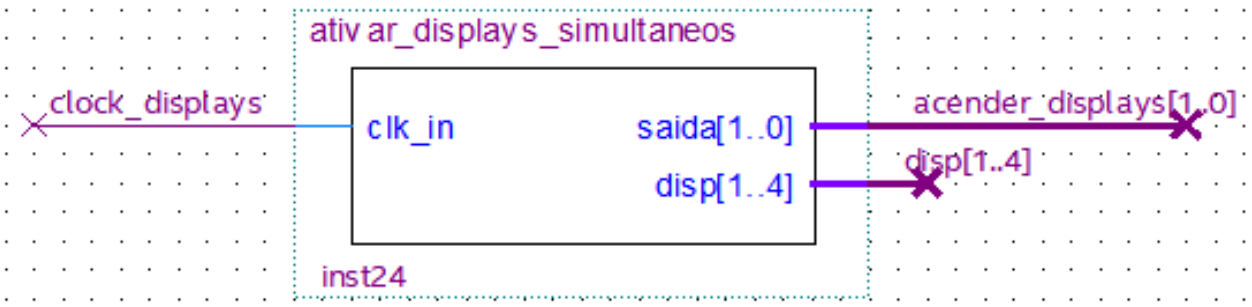


Figura 2.4: Contador seletor para o multiplexador.

À medida que o contador avança, suas saídas são encaminhadas para o multiplexador através do chip criado com essa funcionalidade mostrado na Figura 2.5. Este multiplexador utiliza essas saídas para selecionar o display apropriado, direcionando os dados correspondentes para o display escolhido. O clock do contador é gerado por um conversor de clock de módulo baixo, descrito anteriormente e incluído no Apêndice A, que proporciona uma frequência de contagem muito alta.

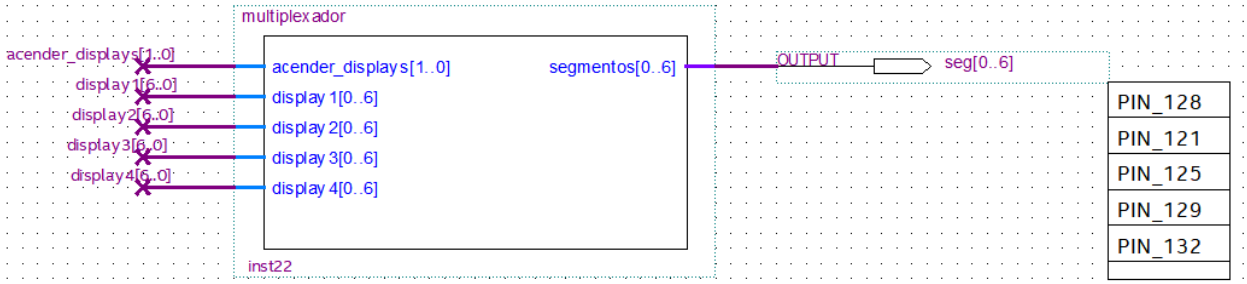


Figura 2.5: Diagrama esquemático do Multiplexador.

O código presente no Apêndice F define um multiplexador que seleciona qual dos quatro displays de 7 segmentos deve ser exibido a cada vez. O multiplexador recebe como entradas o seletor e o clock rápido mencionados, além dos sinais correspondentes a cada segmento de display.

A saída do multiplexador corresponde aos pinos dos segmentos que serão ativados no display selecionado. Com base no valor do seletor, o multiplexador aciona o display adequado, iluminando apenas os segmentos que correspondem ao valor binário específico. Assim, somente o display escolhido é ativado, enquanto os outros permanecem inativos. Embora o sistema alterne entre os displays, a alta frequência do clock torna essa mudança imperceptível aos olhos humanos.

## 2.7 Ajustes

Para ajustar as horas do relógio e dos alarmes, bem como alternar entre esses modos, é necessário implementar uma função, conforme descrito no Apêndice G. Para isso, foi criado um seletor composto por um contador de módulo 4, no qual as quatro saídas são ativadas individualmente, permitindo a seleção adequada. O ajuste do relógio é ativado com o acionamento da chave 1.

Esse seletor foi implementado em um chip que atua de duas formas principais:

1. Seleção de Display: Na primeira função, o chip permite a seleção do display que será ajustado. O botão utilizado para alternar entre os displays é o botão 3.

2. Seleção de Modo de Ajuste: Na segunda função, o chip permite alternar entre os diferentes modos de ajuste (relógio, alarme 1, alarme 2, ou alarme 3). Nesse caso, o botão 4 é responsável por alternar entre os modos.

Ambas as funções dependem da ativação do modo de ajuste e do botão correspondente, conforme ilustrado na Figura 2.6.

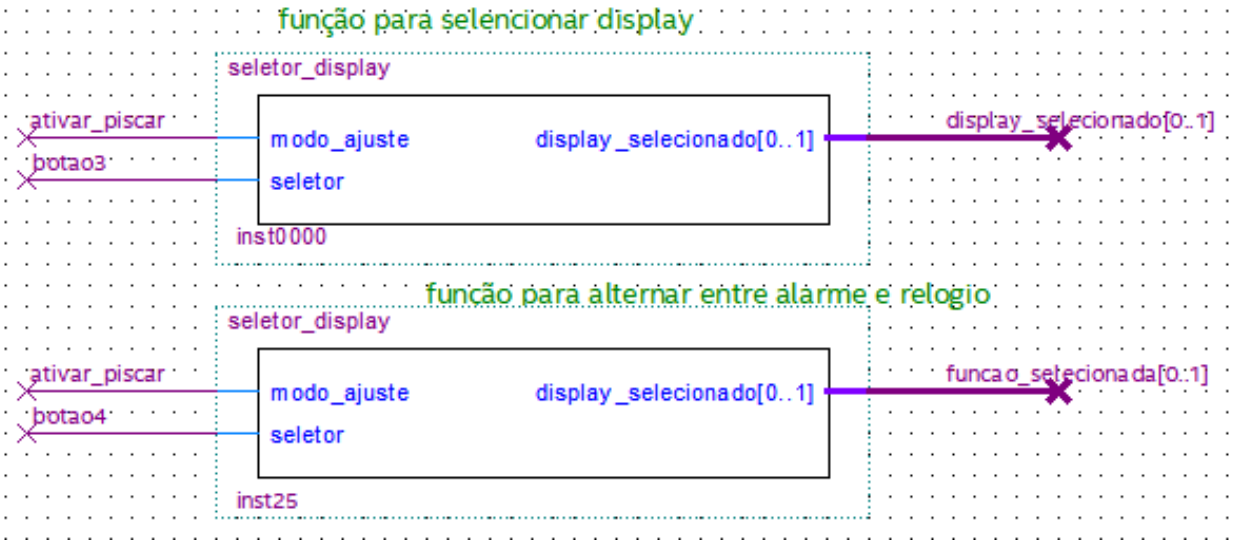


Figura 2.6: Seletor do display a ser alterado e Seletor do modo a ser ajustado.

## 2.8 Relógio

O código do relógio, inserido no Apêndice H, implementa um sistema digital que permite ajustes nos formatos de 12 ou 24 horas, controlando quatro contadores responsáveis pelas diferentes unidades de tempo (segundos, minutos, horas). As entradas incluem dois sinais de clock: um para o funcionamento normal e outro para o modo de ajuste, além de um sinal que define se o relógio está em modo normal ou de ajuste. O seletor de dois bits controla qual parte do relógio será ajustada, enquanto a entrada "n"define se o relógio opera no formato de 12 ou 24 horas. As saídas consistem em quatro conjuntos de 4 bits que representam o valor atual de cada contador, além de um LED que indica o período do dia (AM/PM) quando configurado no modo de 12 horas.

O arquivo do relógio é integrado posteriormente a um arquivo principal que coordena esse módulo e outros, sincronizando suas operações. No modo normal (quando o modo de ajuste está desativado), os contadores operam em cascata, de modo que o avanço de um contador depende do reset do anterior, garantindo a correta sequência de contagem de tempo. Esses contadores seguem o modelo discutido no capítulo "Contador de 4 bits". Quando o modo de ajuste é ativado, o seletor determina qual dos quatro contadores será ajustado via um botão que aciona o clock de alteração, permitindo modificar os dígitos correspondentes.

Se o relógio estiver configurado para operar no formato de 24 horas, o LED permanece apagado.

No formato de 12 horas, o LED indica "AM"quando aceso e "PM"quando apagado, sendo controlado por um flip-flop que alterna de estado a cada ciclo de 12 horas. Tanto os resets quanto o controle dos flip-flops são cuidadosamente sincronizados para assegurar a correta contagem do tempo, usando tanto o clock principal quanto o clock de ajuste.

## 2.9 Alarme

O código do alarme, apresentado no Apêndice I, implementa um sistema digital com quatro contadores de 4 bits, responsáveis por diferentes unidades de tempo, como horas e minutos, de forma similar ao código do relógio. O sistema utiliza um seletor de 2 bits para determinar qual contador será ajustado, direcionando os sinais de clock. Dependendo da configuração do seletor, o clock é aplicado a um dos contadores, enquanto os demais ficam inativos. Cada contador é constituído por flip-flops de 4 bits, cujas entradas e saídas são sincronizadas pelo clock correspondente.

Os contadores operam de forma sequencial e possuem lógica de resets que garante a reinicialização automática ao atingir valores predefinidos, como detalhado no capítulo "Contador de 4 bits". O comportamento do sistema ajusta-se conforme o formato de tempo configurado: 12 ou 24 horas, baseado no sinal de entrada "n". Quando "n"está configurado para o formato de 24 horas, o LED permanece apagado. No modo de 12 horas, o LED alterna entre ligado (AM) e desligado (PM), controlado por um flip-flop que muda de estado a cada ciclo de 12 horas.

Os contadores são conectados em cascata, com resets independentes para assegurar a contagem precisa dos intervalos de tempo. O flip-flop responsável pelo controle do LED alterna o estado entre AM e PM no modo de 12 horas. As saídas representam os valores binários de cada contador e são atualizadas conforme o clock de ajuste atua sobre o contador selecionado. Dessa forma, o sistema permite a alternância entre a configuração dos diferentes displays e o controle do tempo nos formatos de 12 ou 24 horas.

O arquivo do relógio é integrado posteriormente a um arquivo principal que coordena esse módulo e outros, sincronizando suas operações.

## 2.10 Exibição e alteração

A partir dos códigos previamente desenvolvidos, é possível integrá-los para configurar a exibição do relógio e alarmes, além de seus ajustes, bem como a ativação do buzzer e dos LEDs. Assim, um

único código, contido no Apêndice J, abrangente cobre todos esses aspectos. O bloco esquemático gerado com todas essas funcionalidades é apresentado na Figura 2.7.

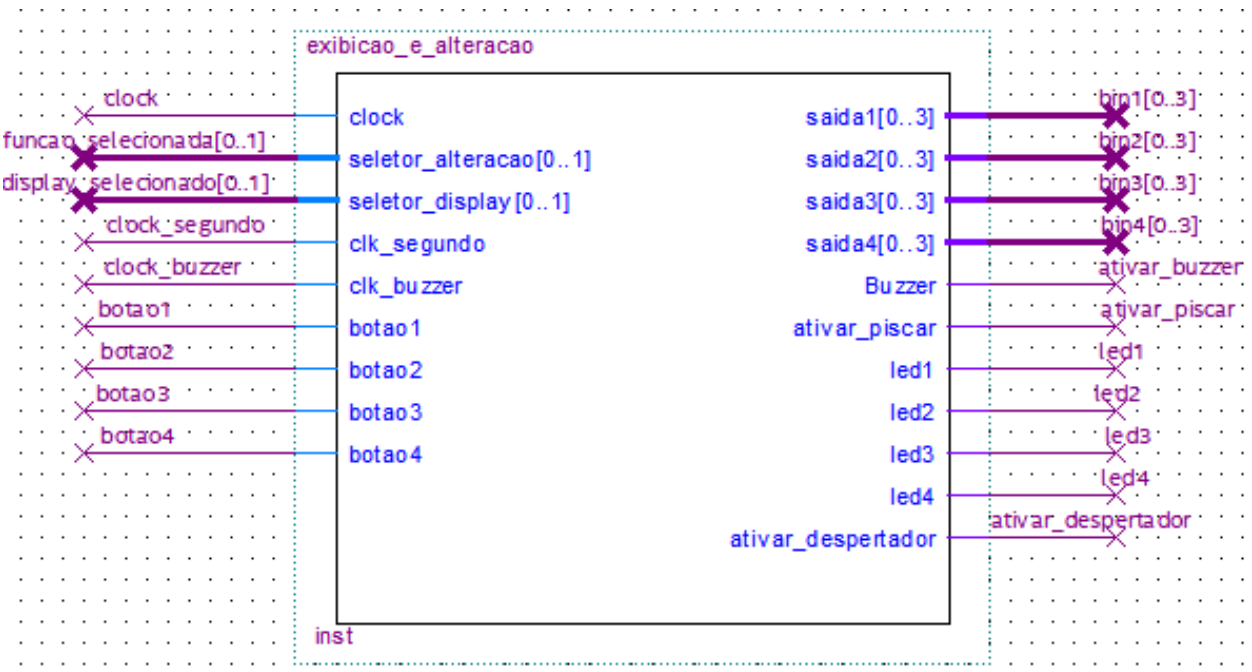


Figura 2.7: Diagrama esquemático do Exibição e Alteração.

O sistema implementa uma instância para o relógio e três para os alarmes, que operam de forma independente, controlados por uma máquina de estados que alterna entre os formatos de 12 e 24 horas, dependendo da entrada dos botões.

### 2.10.1 Definição do formato

Inicialmente, a máquina de estados verifica os botões "botao3"e "botao4"para selecionar o formato de horas. No modo de 24 horas, o relógio e os alarmes são ajustados conforme esse formato, com a variável "n"indicando a configuração. Da mesma forma, no modo de 12 horas, o comportamento é similar, mas "n"representa o formato de 12 horas. O primeiro estado da máquina serve como inicialização, exibindo no display as opções de formato. Dependendo da seleção, a máquina transita para o segundo estado, iniciando o relógio no formato de 24 horas, ou para o terceiro estado, iniciando-o no formato de 12 horas.

### 2.10.2 Alteração do relógio e alarmes

O sistema também permite alternar entre a exibição do relógio e dos três alarmes para ajustes, utilizando um seletor. No início, o relógio recebe o clock padrão enquanto os alarmes permanecem inativos. Quando o relógio é selecionado para ajuste, seu clock é pausado, recebendo apenas o clock proveniente do botão de alteração. Os dígitos do relógio são exibidos no display, e todos os LEDs permanecem desligados. Ao selecionar um alarme, o sistema de ajuste é ativado para esse alarme específico, enquanto os outros permanecem inativos. Dessa forma, apenas o alarme selecionado recebe o clock de alteração, enquanto o relógio retoma seu clock definido, garantindo que a contagem continue durante o ajuste. Para cada alarme, um LED correspondente é ativado, indicando qual alarme está sendo ajustado, com os LEDs 1 a 3 representando os alarmes 1 a 3, respectivamente.

### 2.10.3 Comparação

Outra função do código é a comparação, que verifica se o horário atual do relógio coincide com algum dos alarmes configurados. Em caso de coincidência, um comparador ativa o alarme.

O sistema é controlado por um flip-flop do tipo T, cujo clock é acionado pela saída do comparador ou pela ativação simultânea dos botões 3 e 4. O primeiro pulso do flip-flop T ocorre quando o comparador detecta que o horário do relógio coincide com o do alarme, ativando a saída do flip-flop.

### 2.10.4 Sinal sonoro

A ativação do buzzer depende de três condições: o modo de ajuste deve estar desligado (uma vez que o buzzer não deve tocar durante esse modo), a saída do flip-flop deve estar ativa, e deve haver um clock de alta frequência. Essas condições são suficientes para acionar o alarme, mas um clock de um segundo foi adicionado às entradas para que o buzzer emita um som alternado a cada segundo.

Para desativar o alarme, o próximo pulso de clock precisa desativar a saída do flip-flop. Isso ocorre quando três condições são atendidas: a saída do flip-flop T está ativada (com o buzzer tocando) e os botões 3 e 4 são pressionados simultaneamente. Quando essas condições são satisfeitas, o clock é acionado, desativando a saída do flip-flop T e silenciando o alarme.

## 2.11 Acender e piscar

O multiplexador tem a função de direcionar os bits de cada display para os segmentos correspondentes, conforme descrito anteriormente. No entanto, além dessa função, é essencial garantir a ativação adequada dos LEDs e gerenciar os momentos em que o display deve piscar. Para isso, foi desenvolvido um código que modula a ativação dos displays de acordo com a configuração do relógio, permitindo uma visualização correta dos valores e um controle eficiente do piscar do relógio.

Quando o modo de ajuste está ativo, uma estrutura de controle determina qual display deve piscar com base no valor do seletor. Dependendo da seleção, o dígito correspondente é ativado, fazendo com que ele pisque a cada pulso do clock de um segundo.

Se o modo de ajuste estiver desativado e o buzzer estiver ativado, ou seja, se o alarme estiver tocando, todos os dígitos do display serão ativados para piscar simultaneamente, utilizando a mesma lógica combinada com o clock de um segundo.

Esse sistema, apresentado no Apêndice K, assegura que, ao selecionar um display específico ou quando um alarme toca, o display selecionado ou todos os displays pisquem de acordo com as condições estabelecidas. O chip referente a esse código é exibido na Figura 2.8.

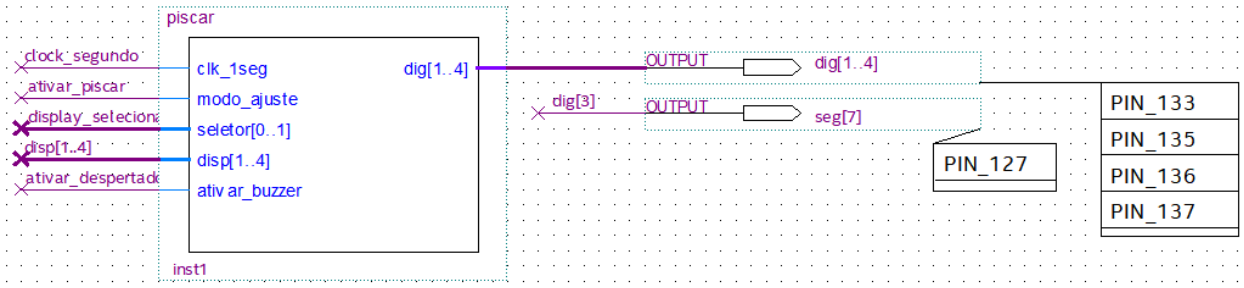


Figura 2.8: Diagrama esquemático do Acender e Piscar.

# 3 Manual de Operação

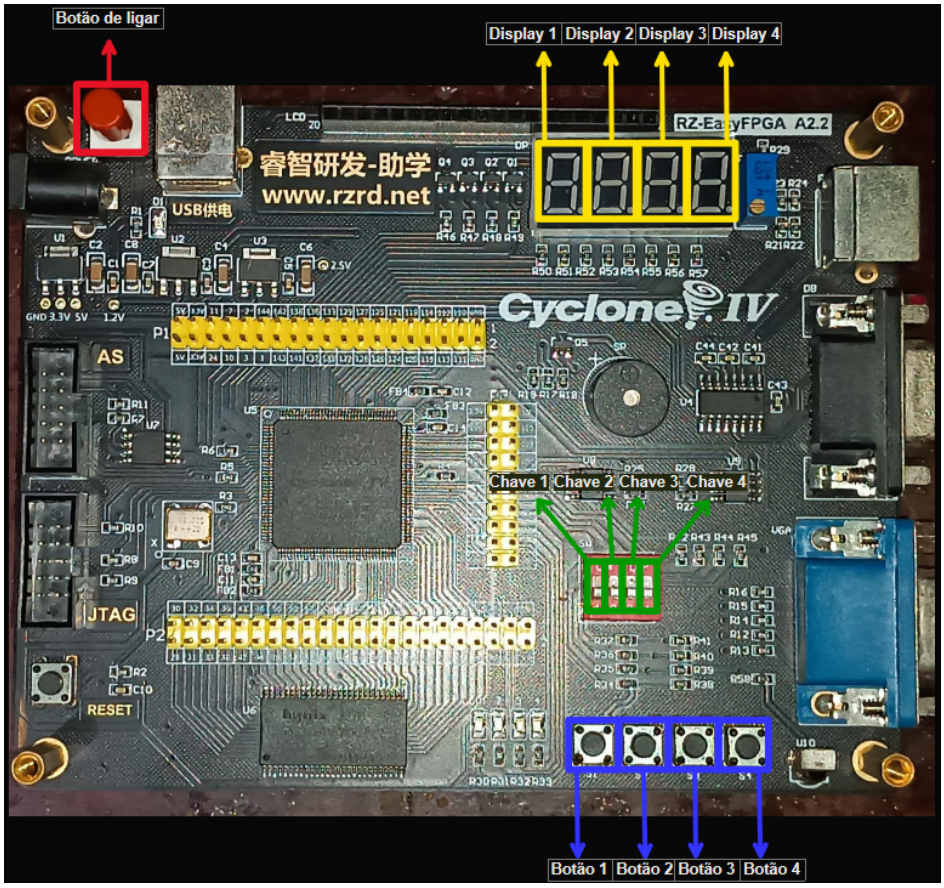


Figura 3.1: Identificação dos itens no FPGA.

1. Introdução

   - Este manual descreve o funcionamento e o procedimento de ajuste do relógio digital com alarme. O dispositivo é projetado para fornecer a hora atual e permitir o ajuste de um alarme programável. Os displays, por onde são exibidos os dígitos do relógio, estão marcados em amarelo na Figura 3.1.

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

2. Ligando o Circuito

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

   - Ao gravar o circuito na placa, o relógio iniciará no modo de escolha do formato de horas, teremos o modo de 12 horas e 24 horas. O botão 3 seleciona o relógio no formato de 24 horas e o botão 4 seleciona o formato de 12 horas.

   - Após a seleção do formato de horas o relógio digital iniciará a contagem em 00:01, no formato HH:MM (Horas:Minutos).

3. Ajuste do Relógio

   (a) Entrando no Modo de Ajuste

   - Para entrar no modo de ajuste do relógio, ligue a chave 1.

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

   (b) Alterando o Dígito de Ajuste

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

   (c) Selecionando o Dígito para Alteração

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

   (d) Finalizando o Ajuste do Relógio

   - Caso deseje entrar no modo de ajuste do alarme, aperte o botão 4.

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

   - Para sair do modo ajuste, desligue a chave 1.

4. Ajuste do Alarme

(a) Entrando no Modo de Ajuste do Alarme

- O processo para ajustar o alarme é semelhante ao ajuste do relógio.

- Repetir o procedimento para cada dígito do alarme:

- Use o botão 2 para alterar o dígito piscante.

- Use o botão 3 para selecionar o dígito a ser alterado.

- Use o botão 4 para selecionar o próximo alarme

(b) Finalizando o Ajuste do Alarme

- Para sair do modo de ajuste do alarme e registrar o horário do alarme, desligue a chave 1.

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

5. Desligar o alarme

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

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

6. Notas Finais

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

# 4 Resultados

O projeto desenvolvido resultou em um relógio digital funcional programado em AHDL, com estrutura no formato HH:MM, permitindo a contagem do tempo em dois formatos: de 0 a 24 horas ou de 0 a 12 horas. No formato de 12 horas, foi implementado um indicador visual de AM/PM por meio de uma LED, que acende para indicar o período da manhã (AM) e apaga para o período da tarde/noite (PM), alternando a cada 12 horas. O relógio inicia sua contagem no tempo 00:01 (1 minuto) e oferece funções específicas para ajuste de horário e configuração de alarmes através de botões e chaves na placa FPGA.

Ao ligar o sistema, o usuário pode escolher entre os dois formatos de hora (12 ou 24 horas) e, uma vez selecionado, o relógio começa a funcionar. Caso o usuário queira entrar no modo de ajuste de horas, o sistema será ativado por meio de uma chave dedicada. Nesse modo, o usuário pode selecionar qual dos display deseja ajustar, sendo que o display ativo pisca para indicar que está em modo de alteração. O valor exibido pode ser incrementado pressionando um botão, com o formato crescente e após o máximo da contagem do display em foco a contagem retorna a zero, vale lembrar que se segurarmos o botão de incremento o sistema vai acrescentar de forma constante valores ao display. Após concluir o ajuste, o usuário pode desligar a chave de ajuste ou, alternativamente, pressionar o botão 4 para entrar no modo de configuração de alarmes, que segue o mesmo padrão de ajuste de horas.

A função de comparação de alarmes-relógio foi programada de forma a acionar um alerta visual e sonoro quando a hora do relógio coincide com qualquer um dos alarmes configurados. Neste caso, os quatro displays HH:MM piscam de forma sincronizada, enquanto o buzzer da placa FPGA é ativado. O buzzer permanece ativo até que os botões 3 e 4 sejam pressionados simultaneamente para desativá-lo, garantindo que o usuário interaja fisicamente com o sistema para interromper o alarme.

O sistema utiliza multiplexadores para controlar a ativação dos displays de 7 segmentos, permitindo que cada segmento seja atualizado individualmente em alta velocidade, criando a percepção de que

todos os displays estão ativos simultaneamente, embora fisicamente apenas um segmento esteja aceso em um dado momento. Além disso, foi implementado um sistema de divisor de clock, que ajusta o clock nativo da placa de 50 MHz para gerar um clock adequado ao funcionamento do relógio. Clocks separados foram usados para diferentes funções, como gerar pulsos de um minuto, controlar o piscar dos displays no modo de ajuste e ativar o buzzer em caso de alarme.

Por fim, para garantir que a operação dos botões e chaves fosse confiável e livre de ruídos elétricos, foi implementado um sistema de debouncing. Este mecanismo evita leituras errôneas de múltiplos acionamentos de botões e chaves devido a instabilidades elétricas naturais que ocorrem em sistemas mecânicos. A exibição dos números no relógio foi realizada por meio de um decodificador de 7 segmentos, que traduz os sinais binários gerados pelo contador de 4 bits para uma representação numérica visual no display de 7 segmentos.

A Figura 4.1 apresenta o instante inicial de funcionamento do relógio.

# 5 Discussão dos Resultados

A implementação do relógio digital foi concluída com sucesso, demonstrando a viabilidade da construção de sistemas de controle e temporização em AHDL, utilizando hardware reconfigurável (FPGA). O design modular permitiu que o sistema atendesse às funcionalidades principais, como a exibição do tempo, ajuste de horário e configuração de alarmes, enquanto incorporava recursos adicionais como o indicador AM/PM e a interação com o usuário por meio de botões e chaves.

O estágio de sincronização dos contadores dos displays do relógio revela ser um fator extremamente importante para evitarmos erros de atraso durante as nossas contagem, fazendo com que o sistema fique mais preciso e eficaz ao longo do tempo. Evitando assim, qualquer tipo de imprecisão ocasionada por Delay na comunicação do sistema.

Um dos principais desafios foi garantir a precisão e estabilidade do relógio, especialmente devido à alta frequência de operação da placa FPGA (50 MHz). Para resolver esse problema, o uso de divisores de clock foi crucial, convertendo o clock de 50 MHz para uma frequência mais baixa, adequada para a contagem dos segundos e a ativação do buzzer. Esses conversores garantiram que as diferentes partes do sistema operassem de forma sincronizada e estável, sem sobrecarregar o hardware.

A escolha de implementar multiplexadores para o controle dos displays foi uma decisão acertada, pois possibilitou o controle eficiente dos 4 displays com um número limitado de sinais. Isso não apenas otimizou o uso dos recursos disponíveis na FPGA, mas também garantiu uma comutação rápida o suficiente para que o piscar dos displays fosse perceptível e claro, tanto no modo de ajuste quanto durante a ativação dos alarmes.

O mecanismo de debouncing implementado para os botões e chaves demonstrou ser fundamental para evitar leituras incorretas, que poderiam comprometer a funcionalidade de ajuste do relógio. Sem o debouncing, a leitura dos botões poderia resultar em múltiplos incrementos acidentais ao ajustar o horário, o que comprometeria a experiência do usuário. O sistema de detecção de AM/PM no formato de 12 horas também foi bem-sucedido, com a LED representando de forma clara o período do dia, tornando o relógio mais intuitivo.

O processo de desativação do alarme com a interação dos botões foi executado com êxito, fazendo com sempre o sistema sempre entenda que deve desativar o buzzer e o piscar dos displays após pressionar os 2 botões.

# 6 Conclusão

O projeto do relógio digital em AHDL demonstrou ser uma implementação eficaz de um sistema de temporização e controle em uma FPGA, integrando múltiplas funcionalidades como ajustes de horas e alarmes, além de uma interface clara com o usuário. A arquitetura do relógio, incluindo a escolha entre os formatos de 12 e 24 horas, com a indicação visual de AM/PM, contribuiu para a versatilidade do sistema, atendendo a diferentes preferências de uso.

A aplicação de técnicas como a multiplexação para controle de displays, a divisão do clock para geração de pulsos em diferentes frequências e o debouncing para estabilização de entradas mecânicas, foram essenciais para garantir o desempenho eficiente e a confiabilidade do sistema. O projeto demonstrou a capacidade de aplicar conceitos de eletrônica digital em um contexto prático, utilizando o AHDL para controlar um dispositivo complexo em hardware reconfigurável.

Outro ponto importante foi a modularidade do sistema. A abordagem modular do design facilitou a integração de diferentes componentes, como a lógica de contagem de horas, os ajustes de alarmes e o controle do buzzer, sem comprometer a complexidade geral do sistema.

A experiência de desenvolver o projeto em AHDL também proporcionou um maior entendimento sobre a programação de hardware reconfigurável e a importância de técnicas de controle de fluxo, como o uso adequado de flip-flops e contadores para garantir que os sinais fossem processados de forma correta e em tempo hábil. A interação direta com os componentes físicos, como botões e LEDs, destacou a importância de lidar com as limitações de sistemas reais, como o debouncing para evitar leituras incorretas e a necessidade de controlar cuidadosamente a temporização dos sinais.

# 7 Referências Bibliográficas

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

# 8 Apêndices

## Apêndice A Conversor de clock

```
PARAMETERS(
  n = 1 , -- n mero de flip-flops para contagem bin ria.
  count = 1 -- valor da contagem
);
SUBDESIGN Conversor_de_clock
(
    clock  : INPUT;             -- Entrada de clock principal
    Clock_convertido : OUTPUT;  -- Sa da do clock convertido
)
VARIABLE
    Flip[n-1..0] : TFF;      -- Vetor de flip-flops tipo T
   Flip_Flop_saida :TFF;    -- Flip-Flop T auxiliar para converter o clock
      principal
   clrn : NODE; -- Clear/reset para o contador (ativo em 0)
BEGIN

   Flip_Flop_saida.clk = NOT clrn;   -- Configurando o Flip-Flop Auxiliar
    Flip_Flop_saida.t = VCC;
    Flip_Flop_saida.clrn = VCC;
    Flip_Flop_saida.prn = VCC;

    -- O primeiro flip-flop recebe o clock diretamente
    Flip[0].clk = NOT clock;
    Flip[0].t = VCC;         -- O flip-flop T sempre alterna
    Flip[0].clrn = clrn;     -- Reset para o contador
    Flip[0].prn = VCC;       -- Preset sempre ativo

    -- La o para gerar os outros flip-flops
```

```
    FOR i IN 1 TO n-1 GENERATE
        Flip[i].clk = NOT Flip[i-1].q; -- Clock de cada flip-flop    a sa da do
            anterior
        Flip[i].t = VCC;
        Flip[i].clrn = clrn;
        Flip[i].prn = VCC;
    END GENERATE;

    -- Contagem crescente

   -- Sistema de reset dos flip-flops  ap s  chegar na contagem desejada

   IF count == Flip[].q THEN
    clrn = GND; -- Ativou o Reset de todos os Flip-Flops
   ELSE
     clrn = VCC; -- Ap s o reset desetivou o n   de reset de todos os Flip-Flops
   END IF;

   -- Sistema que da clock no Flip-Flop auxiliar  ap s  a contagem concluida

   Clock_convertido = NOT Flip_Flop_saida.q; -- cock convertido  s e r   a saida do
       Flip-Flop Auxiliar

END;
```

## Apêndice B Conversor de clock para o relógio

```
PARAMETERS(
  n = 32 , -- n mero de flip-flops para contagem bin ria.
  count = 3000000000 -- valor da contagem
);
SUBDESIGN Conversor_relogio
(
  clock : INPUT;              -- Entrada de clock principal
   Clock_convertido : OUTPUT;  -- Sa da do clock convertido
)
VARIABLE
  Flip[n-1..0] : TFF;       -- Vetor de flip-flops tipo T
```

```
12   estado_pulso : MACHINE WITH STATES (MINUTO1, MINUTO2);
13 BEGIN
14
15   estado_pulso.clk=clock;
16
17
18    Flip[0].clk = NOT clock;
19    Flip[0].t = VCC;         -- O flip-flop T sempre alterna
20    Flip[0].clrn = NOT (Flip[].q == count);     -- Reset para o contador
21    Flip[0].prn = VCC;       -- Preset sempre ativo
22
23     -- La o para gerar os outros flip-flops
24    FOR i IN 1 TO n-1 GENERATE
25     Flip[i].clk = NOT Flip[i-1].q; -- Clock de cada flip-flop    a sa da do
           anterior
26       Flip[i].t = VCC;
27       Flip[i].clrn = NOT (Flip[].q == count);
28       Flip[i].prn = VCC;
29    END GENERATE;
30
31   CASE estado_pulso IS
32     WHEN MINUTO1=>
33       IF  (Flip[].q == 100000) THEN
34         Clock_convertido = vcc; -- Ativou o Reset de todos os Flip-Flops
35         estado_pulso=MINUTO2;
36       ELSE
37         Clock_convertido = GND; -- Ap s o reset desetivou o n   de reset de todos
               os Flip-Flops
38       END IF;
39     WHEN MINUTO2=>
40       IF  (Flip[].q == count) THEN
41         Clock_convertido = vcc; -- Ativou o Reset de todos os Flip-Flops
42       ELSE
43         Clock_convertido = GND; -- Ap s o reset desetivou o n   de reset de todos
               os Flip-Flops
44       END IF;
45   END CASE;
46
47 END;
```

# Apêndice C Debounce

```
TITLE "Debounce";

INCLUDE "Conversor_de_clock.inc";

SUBDESIGN debounce(
  clk, entrada1, entrada2, entrada3, entrada4   : INPUT;
  saida1, saida2, saida3, saida4      : OUTPUT;
)

VARIABLE
  ffd1, ffd2, ffd3, ffd4 : DFF;
  conversor : Conversor_de_clock WITH (n=21,count=2000000);
  conversor_05seg : Conversor_de_clock WITH (n=23,count=7500000);

BEGIN
  conversor.clock = clk;
  conversor_05seg.clock = clk;

  ffd1.clk = conversor.Clock_convertido;
  ffd1.d = NOT entrada1;
  ffd1.prn = VCC;
  ffd1.clrn = VCC;
  saida1 = ffd1.q;

  ffd2.clk = conversor.Clock_convertido;
  ffd2.d = NOT entrada2;
  ffd2.prn = VCC;
  ffd2.clrn = VCC;
  saida2 = (ffd2.q AND conversor_05seg.Clock_convertido);

  ffd3.clk = conversor.Clock_convertido;
  ffd3.d = NOT entrada3;
  ffd3.prn = VCC;
  ffd3.clrn = VCC;
  saida3 = ffd3.q;

  ffd4.clk = conversor.Clock_convertido;
```

```
38    ffd4.d = NOT entrada4;
39    ffd4.prn = VCC;
40    ffd4.clrn = VCC;
41    saida4 = ffd4.q;
42
43  END;
```

## Apêndice D Decodificador

```
1  SUBDESIGN decodificador_7segmentos
2  (
3     binario[0..3] : INPUT;     --[Q3 Q2 Q1 Q0]
4     segmentos[6..0] : OUTPUT; --[A B C D E F G]
5  )
6
7  BEGIN
8
9     TABLE
10       binario[] =>  segmentos[];
11       B"0000"    =>  B"1111110"; -- 0
12       B"0001"    =>  B"0110000"; -- 1
13       B"0010"    =>  B"1101101"; -- 2
14       B"0011"    =>  B"1111001"; -- 3
15       B"0100"    =>  B"0110011"; -- 4
16       B"0101"    =>  B"1011011"; -- 5
17       B"0110"    =>  B"1011111"; -- 6
18       B"0111"    =>  B"1110000"; -- 7
19       B"1000"    =>  B"1111111"; -- 8
20       B"1001"    =>  B"1111011"; -- 9
21    END TABLE;
22
23 END;
```

## Apêndice E Ativar displays simultaneos

```
1  SUBDESIGN ativar_displays_simultaneos
2  (
```

```
    clk_in        : INPUT;
    saida[1..0], disp[1..4]    : OUTPUT;
)


VARIABLE
  Q[1..0] : TFF;

BEGIN

  Q[0].t=vcc;
  Q[1].t = Q[0].q;
  Q[].clk=clk_in;
  Q[].clrn=vcc;
  Q[].prn=vcc;

   saida[] = Q[].q;

  CASE Q[] IS
    WHEN B"00" =>
      disp[1] = GND;
      disp[2] = VCC;
      disp[3] = vcc;
      disp[4] = vcc;
      WHEN B"01" =>
      disp[1] = VCC;
      disp[2] = GND;
      disp[3] = vcc;
      disp[4] = vcc;
      WHEN B"10" =>
      disp[1] = VCC;
      disp[2] = VCC;
      disp[3] = GND;
      disp[4] = vcc;
      WHEN B"11" =>
      disp[1] = VCC;
      disp[2] = VCC;
      disp[3] = vcc;
      disp[4] = GND;
```

```
42    END CASE;
43 END;
```

## Apêndice F Multiplexador

```
1  TITLE "Multiplexador";
2
3  SUBDESIGN multiplexador -- Nessa se o declaramos os pinos de entrada e saida
4  ( -- O nome do arquivo .tdf tem que ser igual ao declarado no sub
5     acender_displays[1..0], display1[0..6], display2[0..6], display3[0..6],
          display4[0..6] : INPUT;  -- pinos de entrada
6     segmentos[0..6] : OUTPUT;  -- pinos de saida
7  )
8
9  BEGIN
10   CASE acender_displays[] IS
11     WHEN B"00" => segmentos[]=!display1[];
12     WHEN B"01" => segmentos[]=!display2[];
13     WHEN B"10" => segmentos[]=!display3[];
14     WHEN B"11" => segmentos[]=!display4[];
15   END CASE;
16 END;
```

## Apêndice G Seletor display

```
1  TITLE "Seletor Display";
2
3  SUBDESIGN seletor_display -- Nessa se o declaramos os pinos de entrada e saida
4  ( -- O nome do arquivo .tdf tem que ser igual ao declarado no sub
5     modo_ajuste, seletor : INPUT;  -- pinos de entrada
6     display_selecionado[0..1] : OUTPUT;  -- pinos de saida
7  )
8
9  VARIABLE
10    Q[0..1] : TFF;  -- Cria o da conexo eletrica, variaveis daqui no so o
          da mesma forma que python ou c
11
```

```
BEGIN  -- a linguagem n o    compilada em cascata vertical mas sim horizontal

  Q[].clk = seletor;
  Q[1].t = modo_ajuste;
  Q[0].t = Q[1].q;
  Q[].clrn = modo_ajuste;
  Q[].prn = vcc;

  if modo_ajuste==VCC then
    display_selecionado[]=Q[].q;
  END IF;
END;
```

## Apêndice H Relógio

```
TITLE "Rel gio";

SUBDESIGN relogio
(
   clk, clk_alteracao, modo_ajuste, seletor[0..1], n: INPUT;
   saida1[3..0], saida2[3..0], saida3[3..0], saida4[3..0], led4 : OUTPUT;
)

VARIABLE

   A[8..1],reset[1..4], clock[1..4] : NODE;

  am_pm,Q1[3..0],Q2[3..0],Q3[3..0],Q4[3..0]: TFF;

BEGIN

  IF modo_ajuste==GND THEN --somente conecta o clock com reset do anterior se
      estiver fora do modo ajuste

    clock1=clk;
    clock2= reset1;
    clock3=reset2;
    clock4=reset3;
```

```
23      saida4[] = Q4[].q;
24      saida3[] = Q3[].q;
25      saida2[] = Q2[].q;
26      saida1[] = Q1[].q;
27
28    ELSE
29
30      CASE seletor[] IS
31        WHEN B"00" =>
32          clock1=clk_alteracao;
33          clock2=GND;
34          clock3=GND;
35          clock4=GND;
36        WHEN B"01" =>
37          clock1=GND;
38          clock2=clk_alteracao;
39          clock3=GND;
40          clock4=GND;
41        WHEN B"10" =>
42          clock1=GND;
43          clock2=GND;
44          clock3=clk_alteracao;
45          clock4=GND;
46        WHEN B"11" =>
47          clock1=GND;
48          clock2=GND;
49          clock3=GND;
50          clock4=clk_alteracao;
51      END CASE;
52      saida4[] = Q4[].q;
53      saida3[] = Q3[].q;
54      saida2[] = Q2[].q;
55      saida1[] = Q1[].q;
56
57    END IF;
58
59
60    Q1[].clk= clock1; Q2[].clk= clock2; Q3[].clk= clock3; Q4[].clk= clock4; --
          sincroniza  o de clocks
```

```
61
62
63  A1 = Q1[0].q AND Q1[1].q;
64  A2 = Q1[2].q AND A1;
65  Q1[0].t= VCC; Q1[1].t = Q1[0].q; Q1[2].t = A1; Q1[3].t = A2;        --contador de
        4bits 1 (menos significativo)
66
67  Q1[].clrn = NOT reset1; Q1[].prn = VCC;
68
69  A3 = Q2[0].q AND Q2[1].q;
70  A4 = Q2[2].q AND A3;
71  Q2[0].t= VCC; Q2[1].t = Q2[0].q; Q2[2].t = A3; Q2[3].t = A4;        --contador de
        4bits 2
72
73  Q2[].clrn = NOT reset2; Q2[].prn = VCC;
74
75  A5 = Q3[0].q AND Q3[1].q;
76  A6 = Q3[2].q AND A5;
77  Q3[0].t= VCC; Q3[1].t = Q3[0].q; Q3[2].t = A5; Q3[3].t = A6;        --contador de
        4bits 3
78
79  Q3[].clrn = NOT reset3; Q3[].prn = VCC;
80
81  A7 = Q4[0].q AND Q4[1].q;
82  A8 = Q4[2].q AND A7;
83  Q4[0].t= VCC; Q4[1].t = Q4[0].q; Q4[2].t = A7; Q4[3].t = A8;        --contador de
        4bits 4 (mais significativo)
84
85  Q4[].clrn = NOT reset4; Q4[].prn = VCC;
86
87
88  CASE n is
89
90    when VCC =>
91
92      IF Q1[].q==B"1010" THEN --reset para o flipflop 1
93        reset1=VCC;
94      ELSE
95        reset1=GND;
```

```
96        END IF;
97        IF Q2[].q==B"0110" THEN --reset para o flipflop 2
98          reset2=VCC;
99        ELSE
100         reset2=GND;
101       END IF;
102
103       IF ((Q3[].q == B"1010") OR ((Q3[].q == B"0100") & (Q4[].q == B"0010"))) THEN
              --reset para o flipflop 3
104         reset3=VCC;
105       ELSE
106         reset3=GND;
107       END IF;
108       IF (Q4[].q == B"0010" AND Q3[].q >= B"0100") THEN
109         reset3=VCC;
110       ELSE
111         reset3=GND;
112       END IF;
113       IF Q4[].q == B"0011" THEN --reset para o flipflop 4
114         reset4=VCC;
115       ELSE
116         reset4=GND;
117       END IF;
118         led4=VCC;         -- Se o formato 24h estiver ativo a led fica apagada
119
120     WHEN GND =>
121
122       IF Q1[].q==B"1010" THEN --reset para o flipflop 1
123         reset1=VCC;
124       ELSE
125         reset1=GND;
126       END IF;
127       IF Q2[].q==B"0110" THEN --reset para o flipflop 2
128         reset2=VCC;
129       ELSE
130         reset2=GND;
131       END IF;
132       IF (Q3[].q==B"1010") OR (Q3[].q==B"0010" & Q4[].q==B"0001") OR (Q3[].q>B"
              0001" & Q4[].q==B"0001") THEN --reset para o flipflop 3
```

```
133         reset3=VCC;
134       ELSE
135         reset3=GND;
136       END IF;
137       IF Q4[].q==B"0010" THEN --reset para o flipflop 4
138         reset4=VCC;
139         am_pm.clk = reset4;
140       ELSE
141         am_pm.clk = reset4;
142         reset4=GND;
143       END IF;
144       led4=am_pm.q;        -- a led recebe o am_pm indicando manh  ou tarde (
              ligado manh )
145
146
147   END CASE;
148
149   am_pm.t = VCC;
150   am_pm.prn = VCC;
151   am_pm.clrn = VCC;
152
153 END;
```

# Apêndice I Alarme

```
1  TITLE "Alarme";
2
3  SUBDESIGN alarme
4  (
5     seletor_display[0..1], clk_alarme, n : INPUT;
6     saida1[3..0], saida2[3..0], saida3[3..0], saida4[3..0], led4 : OUTPUT;
7  )
8
9  VARIABLE
10    A[8..1],reset[4..1], aux[1..4] : NODE;
11
12   Q1[3..0],Q2[3..0],Q3[3..0],Q4[3..0],am_pm : TFF;
13
```

```
BEGIN

  CASE seletor_display[] IS
    WHEN B"00" =>
      aux1=clk_alarme;
      aux2=GND;
      aux3=GND;
      aux4=GND;
    WHEN B"01" =>
      aux1=GND;
      aux2=clk_alarme;
      aux3=GND;
      aux4=GND;
    WHEN B"10" =>
      aux1=GND;
      aux2=GND;
      aux3=clk_alarme;
      aux4=GND;
    WHEN B"11" =>
      aux1=GND;
      aux2=GND;
      aux3=GND;
      aux4=clk_alarme;
  END CASE;


  Q1[].clk= aux1; Q2[].clk= aux2; Q3[].clk= aux3; Q4[].clk= aux4; --
      sincroniza  o de clocks


  A1 = Q1[0].q AND Q1[1].q;
  A2 = Q1[2].q AND A1;
  Q1[0].t= VCC; Q1[1].t = Q1[0].q; Q1[2].t = A1; Q1[3].t = A2;       --contador de
       4bits 1 (menos significativo)

  Q1[].clrn = NOT reset1; Q1[].prn = VCC;

  A3 = Q2[0].q AND Q2[1].q;
  A4 = Q2[2].q AND A3;
```

```
51  Q2[0].t= VCC; Q2[1].t = Q2[0].q; Q2[2].t = A3; Q2[3].t = A4;        --contador de
        4bits 2
52
53  Q2[].clrn = NOT reset2; Q2[].prn = VCC;
54
55  A5 = Q3[0].q AND Q3[1].q;
56  A6 = Q3[2].q AND A5;
57  Q3[0].t= VCC; Q3[1].t = Q3[0].q; Q3[2].t = A5; Q3[3].t = A6;        --contador de
        4bits 3
58
59  Q3[].clrn = NOT reset3; Q3[].prn = VCC;
60
61  A7 = Q4[0].q AND Q4[1].q;
62  A8 = Q4[2].q AND A7;
63  Q4[0].t= VCC; Q4[1].t = Q4[0].q; Q4[2].t = A7; Q4[3].t = A8;        --contador de
        4bits 4 (mais significativo)
64
65  Q4[].clrn = NOT reset4; Q4[].prn = VCC;
66
67
68  CASE n is
69
70    when VCC =>
71
72      IF Q1[].q==B"1010" THEN --reset para o flipflop 1
73        reset1=VCC;
74      ELSE
75        reset1=GND;
76      END IF;
77      IF Q2[].q==B"0110" THEN --reset para o flipflop 2
78        reset2=VCC;
79      ELSE
80        reset2=GND;
81      END IF;
82
83      IF ((Q3[].q == B"1010") OR ((Q3[].q == B"0100") & (Q4[].q == B"0010")) OR ((
            Q3[].q > B"0011") & (Q4[].q == B"0010"))) THEN --reset para o flipflop 3
84        reset3=VCC;
85      ELSE
```

```
86          reset3=GND;
87        END IF;
88
89        IF Q4[].q == B"0011" THEN --reset para o flipflop 4
90          reset4=VCC;
91        ELSE
92          reset4=GND;
93        END IF;
94          led4=VCC;         -- Se o formato 24h estiver ativo a led fica apagada
95
96      WHEN GND =>
97
98        IF Q1[].q==B"1010" THEN --reset para o flipflop 1
99          reset1=VCC;
100       ELSE
101         reset1=GND;
102       END IF;
103       IF Q2[].q==B"0110" THEN --reset para o flipflop 2
104         reset2=VCC;
105       ELSE
106         reset2=GND;
107       END IF;
108       IF (Q3[].q==B"1010") OR (Q3[].q==B"0010" & Q4[].q==B"0001") OR (Q3[].q>B"
              0001" & Q4[].q==B"0001") THEN --reset para o flipflop 3
109         reset3=VCC;
110       ELSE
111         reset3=GND;
112       END IF;
113       IF Q4[].q==B"0010" THEN --reset para o flipflop 4
114         reset4=VCC;
115       ELSE
116         reset4=GND;
117       END IF;
118       am_pm.clk = reset4;
119       led4=am_pm.q;        -- a led recebe o am_pm indicando manh  ou tarde (
              ligado manh )
120
121
122   END CASE;
```

```
123
124   am_pm.t = VCC;
125   am_pm.prn = VCC;
126   am_pm.clrn = VCC;
127
128   saida4[] = Q4[].q;
129   saida3[] = Q3[].q;
130   saida2[] = Q2[].q;
131   saida1[] = Q1[].q;
132 END;
```

## Apêndice J Exibição e Alteração

```
1  TITLE "Exibi  o e Altera  o de Hora e Alarmes";
2
3  INCLUDE "alarme.inc";
4  INCLUDE "relogio.inc";
5  INCLUDE "conversor_relogio.inc";
6
7  SUBDESIGN exibicao_e_alteracao
8  (
9
10   clock,seletor_alteracao[0..1], seletor_display[0..1], clk_segundo, clk_buzzer,
       botao1, botao2, botao3, botao4 : INPUT;
11
12   saida1[0..3], saida2[0..3], saida3[0..3], saida4[0..3], Buzzer, ativar_piscar,
       led1, led2, led3, led4, ativar_despertador : OUTPUT;
13 )
14
15 VARIABLE
16
17   relogio1 : relogio;   --instanciamento das fun  es rel gio e alarme
18   alarme1 : alarme;
19   alarme2 : alarme;
20   alarme3 : alarme;
21   --clk_relogio : conversor_relogio WITH (n=32,count=3000000000); -- MINUTO
22   clk_relogio : conversor_relogio WITH (n=24,count=10000000);  -- TEMPO PARA TESTE
       0,2 SEGUNDOS
```

```
23
24   clock_relogio, clk_alteracao, formato24, formato12, modo_ajuste, aux, n,
         comparador : NODE;  --n s auxiliares
25
26   fft : TFF; -- flipflop T auxiliar
27
28   state : MACHINE WITH STATES (s1, s2, s3); --maquina de estados (selecionar 24 ou
         12, fun o 24h, fun o 12h)
29
30 BEGIN
31
32   state.clk=clock; --CLOCK DA MAQUINA DE ESTADOS
33
34   CASE state IS    --Case da maquina de estados
35
36     WHEN s1 =>
37
38       saida4[]=B"0010";
39       saida3[]=B"0100";
40       saida2[]=B"0001";
41       saida1[]=B"0010";
42
43       formato24=botao3;
44       formato12=botao4;
45
46       IF (formato24) THEN   -- formato24 => BOTAO3
47         state=s2;
48       END IF;
49
50       IF (formato12) THEN   -- formato12 => BOTAO4
51         state=s3;
52       END IF;
53
54     WHEN s2 =>      -- Modo 24h
55
56
57       clk_relogio.clock = clock;
58
59       n=VCC; --indica o formato de hrs
```

```
60
61      modo_ajuste = botao1;
62      clk_alteracao = botao2;
63      ativar_piscar = modo_ajuste;
64
65    WHEN s3 =>       -- Modo 12h
66
67
68      clk_relogio.clock = clock;
69        n=GND; --indica o formato de hrs
70
71      modo_ajuste = botao1;
72      clk_alteracao = botao2;
73      ativar_piscar = modo_ajuste;
74
75  END CASE;
76
77  clock_relogio = clk_relogio.Clock_convertido;
78
79  relogio1.n=n;                --REL GIO E ALARMES RECEBEM O FORMATO DE HRS A
        SER SEGUIDO
80  alarme1.n=n;
81  alarme2.n=n;
82  alarme3.n=n;
83
84
85  relogio1.modo_ajuste = modo_ajuste;
86  led4=relogio1.led4;
87  relogio1.seletor[] = seletor_display[];
88
89  relogio1.clk_alteracao = GND;
90  relogio1.clk = clock_relogio;
91  alarme1.clk_alarme = GND;
92  alarme2.clk_alarme = GND;
93  alarme3.clk_alarme = GND;
94
95  CASE seletor_alteracao[] IS
96    WHEN B"00" =>                          --SELECIONA O REL GIO 1
97
```

```
98        alarme1.clk_alarme = GND;
99        alarme2.clk_alarme = GND;
100       alarme3.clk_alarme = GND;
101       relogio1.clk_alteracao = clk_alteracao;
102
103       relogio1.clk = GND;
104
105       saida1[]=relogio1.saida1[];
106       saida2[]=relogio1.saida2[];
107       saida3[]=relogio1.saida3[];
108       saida4[]=relogio1.saida4[];
109
110       led1=VCC;
111       led2=VCC;
112       led3=VCC;
113       led4=relogio1.led4;
114
115
116
117     WHEN B"01" =>                          -- SELECIONA O ALARME 1
118       alarme1.clk_alarme = clk_alteracao;
119       alarme2.clk_alarme = GND;
120       alarme3.clk_alarme = GND;
121
122       relogio1.clk_alteracao = GND;
123       relogio1.clk = clock_relogio;
124       relogio1.modo_ajuste=GND;
125
126       alarme1.seletor_display[] = seletor_display[];
127       saida1[]=alarme1.saida1[];
128       saida2[]=alarme1.saida2[];
129       saida3[]=alarme1.saida3[];
130       saida4[]=alarme1.saida4[];
131
132       led1=GND;
133       led2=VCC;
134       led3=VCC;
135       led4=alarme1.led4;
136
```

```
137     WHEN B"10" =>                                --SELECIONA O ALARME 2
138
139       alarme1.clk_alarme = GND;
140       alarme2.clk_alarme = clk_alteracao;
141       alarme3.clk_alarme = GND;
142
143       relogio1.clk_alteracao = GND;
144       relogio1.clk = clock_relogio;
145       relogio1.modo_ajuste=GND;
146
147       alarme2.seletor_display[] = seletor_display[];
148       saida1[]=alarme2.saida1[];
149       saida2[]=alarme2.saida2[];
150       saida3[]=alarme2.saida3[];
151       saida4[]=alarme2.saida4[];
152
153       led1=VCC;
154       led2=GND;
155       led3=VCC;
156       led4=alarme2.led4;
157
158     WHEN B"11" =>                               --SELECIONA O ALARME 3
159
160       alarme1.clk_alarme = GND;
161       alarme2.clk_alarme = GND;
162       alarme3.clk_alarme = clk_alteracao;
163
164       relogio1.clk_alteracao = GND;
165       relogio1.clk = clock_relogio;
166       relogio1.modo_ajuste=GND;
167
168       alarme3.seletor_display[] = seletor_display[];
169         --saidas
170       saida1[]=alarme3.saida1[];
171       saida2[]=alarme3.saida2[];
172       saida3[]=alarme3.saida3[];
173       saida4[]=alarme3.saida4[];
174
175       led1=VCC;
```

```
176       led2=VCC;
177       led3=GND;
178       led4=alarme3.led4;
179   END CASE;
180
181   IF fft.q == gnd THEN
182
183     IF (((relogio1.saida1[] == alarme1.saida1[]) AND     --COMPARADOR DE BITS PARA
          TOCAR O DESPERTADOR
184       (relogio1.saida2[] == alarme1.saida2[]) AND
185       (relogio1.saida3[] == alarme1.saida3[]) AND
186       (relogio1.saida4[] == alarme1.saida4[]) AND
187       (relogio1.led4 == alarme1.led4)) OR
188
189       ((relogio1.saida1[] == alarme2.saida1[]) AND
190       (relogio1.saida2[] == alarme2.saida2[]) AND
191       (relogio1.saida3[] == alarme2.saida3[]) AND
192       (relogio1.saida4[] == alarme2.saida4[]) AND
193       (relogio1.led4 == alarme2.led4)) OR
194
195       ((relogio1.saida1[]==alarme3.saida1[]) AND
196       (relogio1.saida2[]==alarme3.saida2[]) AND
197       (relogio1.saida3[]==alarme3.saida3[]) AND
198       (relogio1.saida4[]==alarme3.saida4[]) AND
199       (relogio1.led4 == alarme3.led4))) AND (clock_relogio == VCC)  THEN
200
201       comparador=VCC;
202
203     ELSE
204       comparador=GND;
205     END IF;
206   END IF;
207   -- saida do ativar_buzzer
208   fft.t = VCC;
209   fft.prn = VCC;
210   fft.clrn = VCC;
211   fft.clk = aux;
212   aux = comparador $ (botao3 AND botao4 AND fft.q);  --sistema para parar o alarme
          com 2 bot es
```

```
213   IF modo_ajuste==GND THEN
214     Buzzer = fft.q AND clk_buzzer AND clk_segundo;
215     ativar_despertador = fft.q;
216   ELSE
217     ativar_despertador = GND;
218     Buzzer = GND;
219   END IF;
220 END;
```

## Apêndice K Acender e Piscar

```
1  TITLE "PISCAR DISPLAY"; -- Quando selecionar o display a ser alterado e quando o
       alarme tocar
2                 -- Al m de piscar, deve ligar os outros displays simultaneamente
3
4  SUBDESIGN piscar
5  (
6    clk_1seg, modo_ajuste, seletor[0..1], disp[1..4], ativar_buzzer : INPUT;
7    dig[1..4] : OUTPUT;
8  )
9
10 VARIABLE
11
12   display[1..4] : NODE;
13
14 BEGIN
15
16
17   display[]=disp[];
18   dig[]=disp[];
19
20
21   IF (modo_ajuste==VCC) THEN
22     CASE seletor[] IS
23
24       WHEN B"00" =>
25         dig[1]= (NOT display[1] AND clk_1seg);
26       WHEN B"01" =>
```

```
27          dig[2]= (NOT display[2] AND clk_1seg);
28        WHEN B"10" =>
29          dig[3]= (NOT display[3] AND clk_1seg);
30        WHEN B"11" =>
31          dig[4]= (NOT display[4] AND clk_1seg);
32
33      END CASE;
34    END IF;
35
36    IF (modo_ajuste==GND AND ativar_buzzer==VCC) THEN
37
38      dig[]= ((NOT display[]) AND clk_1seg);
39
40    END IF;
41
42  END;
```