**UFRN - UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE**

**DIMAP - DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA**

**BACHARELADO EM TECNOLOGIA DA INFORMAÇÃO**

**DIM0128 - CIRCUITOS LÓGICOS**

CARLOS HENRIQUE CORTEZ GOMES

HANIEL LUCAS MACHADO ROCHA

JAMILLY EMILLY DA SILVA CAMPELO

PEDRO VINÍCIUS BARBOSA PEREIRA

# PROJETO IMPLEMENTAÇÃO EM VHDL DE UM SISTEMA DE CONTROLE PARA 3 ELEVADORES (CONTROLE EM 2 NÍVEIS) - UNIDADE 1

NATAL/RN

2025

## SUMÁRIO

# 1. RESUMO

Este trabalho foi desenvolvido para implementar em VHDL um sistema que resolva o problema de controle de 3 elevadores em um prédio de 32 andares, incluindo chamadas internas dos elevadores e externas em cada andar, controle de portas e indicação do andar atual em displays. A solução resultou num sistema em 2 níveis (controlador local do elevador e escalonador/supervisor para gerenciar qual elevador atende cada chamada externa) usando a estratégia de prioridade por direção combinada com o algoritmo nearest-car (que leva em consideração a distância dos elevadores ao andar da chamada). Também apresentamos a validação do projeto com a simulação das formas de onda de 3 cenários distintos no software GTKWave.

# 2. INTRODUÇÃO

## 2.1. Contextualização

O problema de controle de elevadores é uma questão complexa que envolve muitas variáveis que podem aumentar ainda mais a complexidade do problema, como a temporização das portas, controle do motor em casos de emergência, se as chamadas externas são por andar ou por elevador, a estratégia de gerenciamento das chamadas externas, e assim por diante.

No caso que estamos tratando neste relatório, consideramos um prédio de 31 andares (com o número de andares sendo um generic que pode ser alterado caso seja desejado), com 3 elevadores, chamadas externas por andar (dois botões, um para subir e outro para descer, por andar), ausência de temporização das portas, dentre outros.

A solução foi desenvolvida dividindo a lógica em dois níveis:

- **Nível 1 - controlador local do elevador:** representa um elevador individual, com as chamadas internas pelos botões, controle de motor e porta, monitoramento da porta por um led e visualização do andar atual por meio de displays;

- **Nível 2 - escalonador/supervisor:** é o componente responsável por gerenciar as chamadas externas e decidir qual elevador atende cada uma delas de acordo com uma lógica de prioridade por direção combinada com o algoritmo *nearest car*, enviando pedidos aos controladores locais dos elevadores.

Assim, temos um problema simplificado de controle de elevadores solucionado usando máquinas de estados finitas para cada um dos níveis junto com lógica combinacional e sequencial.

### 2.2. Objetivos

O objetivo principal deste projeto é o projeto, implementação e validação de um sistema de controle em VHDL para múltiplos elevadores, baseado nos requisitos funcionais especificados. Para alcançar este objetivo, foram definidos objetivos específicos, como: projetar e implementar em VHDL um sistema de controle de dois níveis para gerenciar 3 elevadores em um edifício parametrizável de 31 andares; desenvolver o Controlador Local (nível 1), capaz de gerenciar solicitações internas, controlar o motor (subir/descer), operar a porta (aberta no estado IDLE, fechada em movimento) e exibir o andar atual em displays de 7 segmentos; implementar um Escalonador (nível 2) para gerenciar todas as chamadas externas (subir/descer), aplicando uma estratégia de otimização de tempo e viagens (a combinação de prioridade por direção e nearest-car); validar o funcionamento do sistema através de simulações digitais (testbenches), cobrindo cenários de chamadas simples, chamadas concorrentes e atendimento por prioridade; e, por fim, documentar a arquitetura, as máquinas de estado e os resultados das simulações neste relatório técnico.

## 3. ARQUITETURA DO SISTEMA

### 3.1. Arquitetura geral

Como contextualizado previamente, existem dois níveis que possuem funcionalidades diferentes. No primeiro nível, o **elevator_controller** (auxiliado do **queue_registers** e **det_andar**) ordena o **motor** a subir ou descer, dependendo do lugar de onde a chamada precisa ser atendida. Enquanto isso, as **portas** abrem ou

fecham a depender do movimento do elevador (quando parado, elas abrem, quando em movimento, elas fecham) e o **display** atualiza sua saída a depender do andar atual do elevador. Por último, tem um **detector de andar** e um **controlador de andar**, onde o primeiro emite o andar que o elevador está e o outro. No segundo nível, o sistema recebe como entrada um andar com uma direção atrelada. Esse pedido de andar é processado primeiro pelo componente **fila** que recolhe esse pedido e repassa para o **call_dispatcher**, que também repassa para o **cost_calculator** e para o **distance_calculator**. Após isso, o **cost_comparator** compara o custo de cada elevador e escolhe um ganhador. Esse ganhador é recebido pelo **output_generator**, que "manda" o pedido de andar (e direção) para o elevador ganhador do **cost_comparator**. Por fim, a ligação entre esses dois níveis é feita conectando as saídas do Nível 2 (o andar pedido e a direção) com as entradas do Nível 1.

### 3.2. Interface de sinais

O sistema é composto por um escalonador (nível 2) que controla as chamadas externas e envia solicitações para os três elevadores (instâncias do nível 1). As entradas são divididas em:

- **Sinais de controle**
  - **clk:** o sinal de clock global que sincroniza todos os componentes sequenciais do sistema (as máquinas de estados finitas, os registradores de fila, os contadores de andar);
  - **reset:** o sinal de reset global para colocar todos os componentes sequenciais em um estado inicial conhecido.

- **Chamadas externas (para o Escalonador)**
  - **call_up [0..4]:** representa uma chamada externa para subir no andar representado por 5 bits que dura apenas um ciclo de clock;
  - **call_down [0..4]:** análogo ao call_up, só que sendo uma chamada externa para descer.

- **Chamadas internas (para cada elevador)**

Como o sistema possui 3 elevadores, precisamos de uma entrada que representa o painel interno para cada um deles.

- **dest_request_elev0 [0..31]:** o painel de 32 botões do Elevador 0;
- **dest_request_elev1 [0..31]:** o painel de 32 botões do Elevador 1;
- **dest_request_elev2 [0..31]:** o painel de 32 botões do Elevador 2.

Já as saídas são divididas em:

- **Indicadores de andar (displays)**
  - **seg7_D0_elev0 [6..0], seg7_D1_elev0 [6..0]:** displays de unidade e dezena do Elevador 0;
  - **seg7_D0_elev1 [6..0], seg7_D1_elev1 [6..0]:** displays de unidade e dezena do Elevador 1;
  - **seg7_D0_elev2 [6..0], seg7_D1_elev2 [6..0]:** displays de unidade e dezena do Elevador 2.

- **Indicadores das portas (LEDs)**
  - **door_open_closed_elev0:** LED de estado da porta do Elevador 0 ('1' = aberta, '0' = fechada);
  - **door_open_closed_elev1:** LED de estado da porta do Elevador 1;
  - **door_open_closed_elev2:** LED de estado da porta do Elevador 2.

- **Indicadores de movimento dos elevadores (LEDs)**
  - **moving_elev0 [1..0]:** LEDs de movimento do Elevador 0 ("00" = parado, "10" = subindo e "01" = descendo);
  - **moving_elev1 [1..0]:** LEDs de movimento do Elevador 1;
  - **moving_elev2 [1..0]:** LEDs de movimento do Elevador 2.

## 4. METODOLOGIA E IMPLEMENTAÇÃO

### 4.1. Estratégia de escalonamento

Uma decisão de projeto muito importante do sistema de controle de elevadores trata-se do algoritmo escolhido para a estratégia de escalonamento, isto

é, o critério que o Nível 2 irá adotar para escolher os elevadores que irão atender cada chamada externa nos andares.

No projeto em questão, foi decidido pelo algoritmo de prioridade por direção combinado com o *nearest-car*. O algoritmo final funciona da seguinte forma: a cada chamada em questão (composta por andar e direção da chamada), o componente Cost Calculator, que possui uma instância para cada elevador, calcula o custo que cada um teria para atender àquela chamada. Esse custo inicial é calculado com base apenas no estado do elevador: se ele está na mesma direção da chamada e ainda não passou pelo andar, o custo é ótimo, se ele está parado o custo é intermediário e, se ele está na direção contrária ou já passou do andar (teria que mudar a direção pelo menos uma vez para atender à chamada), o custo é o pior.

Paralelamente a isso, há também três instâncias do Distance Calculator, uma para cada elevador, de forma a calcular a distância atual entre cada um deles e o andar da chamada ativa, com uma saída de 5 bits (já que a distância pode variar de 0 a 31 andares). Tendo essas 6 saídas (*distance* e *cost_elevator* para cada um dos três elevadores) como entradas do componente Cost Comparator, ele decide qual elevador vai atender cada chamada usando o seguinte algoritmo: 1) ele seleciona os elevadores com melhor custo, se só há um, esse é o escolhido (por exemplo, apenas um com o custo “ótimo” ou “intermediário”); 2) se dois ou mais elevadores empatam nesse critério, o desempate ocorre pela menor distância ao elevador da chamada.

Esse foi o algoritmo escolhido levando em consideração que combina tanto o estado dos elevadores no momento da chamada quanto a sua distância física ao andar da chamada, além de ser o mais próximo de um sistema real, atendendo ao requisito de "otimização de tempo/viagens"

## 4.2. Máquinas de Estado Finito (FSMs)

### 4.2.1. FSM do Elevador (Nível 1)

O Nível 1 possui três estados: IDLE (parado), MOVING_UP (subindo) e MOVING_DOWN (descendo). O sistema inicia no estado IDLE, com o elevador parado. Nesse estado, se houver uma chamada acima, ele vai para o estado

MOVING_UP, se houver abaixo, vai para o MOVING_DOWN, se houver tanto chamadas acima quanto abaixo, a FSM prioriza chamadas para descer, considerando que, no geral, há mais pessoas querendo descer do que subir, então segue para o estado MOVING_DOWN. Em ambos os estados MOVING_UP e MOVING_DOWN, há o retorno para IDLE apenas quando o elevador passa num andar com chamada ativa, então ele para pra atendê-la.

Essa máquina de estados não controla a porta, dado que o componente Door a controla de forma a mantê-la aberta sempre que o elevador está parado (IDLE) e fechada quando o elevador está em movimento (MOVING_UP ou MOVING_DOWN).

#### 4.2.2. FSM do Escalonador (Nível 2)

No nível 2, semelhante ao nível 1, há 3 estados. O primeiro é quando é recebida uma chamada de pedido para subir, o segundo é quando recebe duas chamadas, e o último é quando recebe nenhuma chamada. No primeiro, o pedido é delegado para a fila (a que representa pedidos de subida ou a de pedidos de descida), o call dispatcher recebe (na próxima subida) o andar e direção desse novo pedido e repassa, mais uma vez para os calculadores de custo e de distância, que repassam para o comparador de custo para produzir uma saída que representa o melhor elevador para atender o pedido. No Gerador de saída, o melhor elevador recebe o andar e direção para o pedido, e assim termina o trabalho do nível 2 e repassa para o nível 1. No segundo estado, quando há duas chamadas concorrentes, a fila recebe os dois, mas guarda os de subida em uma fila, e os de descida em outra fila. Depois, esse componente alterna entre a subida e entre a descida, fazendo com que os pedidos (e direções do pedido) passem pelos mesmos caminhos, mas em ordens diferentes. E por último, no último estado, o nível 2 não emite nenhum sinal para o nível 1.

# 5. SIMULAÇÕES E RESULTADOS

## 5.1. Instruções para reproduzir simulações

Para reproduzir a simulação, é necessário ter as seguintes ferramentas instaladas na sua máquina:

- **GHDL:** o compilador e simulador VHDL
- **GTKWave:** o visualizador de forma de ondas

A maneira mais simples de reproduzir a simulação é utilizando o script **simular_nivel_1.sh**, localizado no diretório raiz do projeto.

Para executá-lo, abra um terminal e navegue até a pasta raiz do projeto. Em seguida, conceda permissão de execução ao script com o comando:

**chmod +x simular_nivel_1.sh**

Após isso, execute o script com:

**./simular_nivel_1.sh**

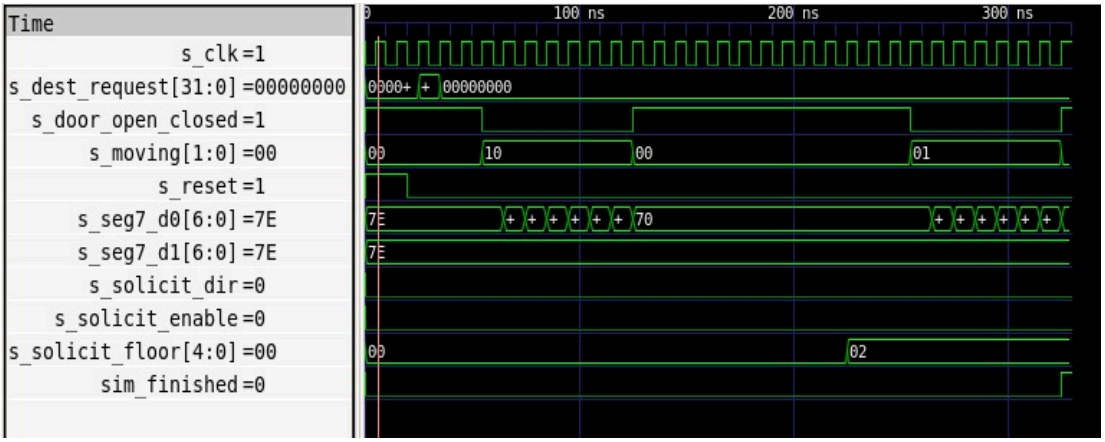
O script irá compilar todos os componentes, executar a simulação, gerar o arquivo **simulacao_nivel_1.vcd** e abrir o GTKWave para análise.

## 5.2. Cenário 1 (chamada simples)

Este cenário testa a funcionalidade do Nível 1: a chamada interna simples. O teste simula o sistema saindo do estado ocioso no andar 0, após o reset. Em seguida, um usuário pressiona o botão para o andar 5 no painel interno (s_dest_request(5)). O objetivo é verificar se o controlador identifica a chamada, fecha a porta, aciona o motor para subir, atualiza os displays durante a subida e, ao chegar no destino, para o motor e abre a porta.

Abaixo está a captura de onda do cenário.

*Figura 1*




*Captura da forma de onda do cenário 1. Reprodução autoral.*

Os principais eventos da simulação são:

No início da simulação, o sinal **s_reset** é ativado (nível '1'), forçando o sistema ao estado inicial.

Após o *reset*, o sistema permanece ocioso no andar 0. Isso é confirmado pelos sinais: **s_moving** está em **“00”** (Parado) e **s_door_open_closed** está em **‘1’** (Aberta). Os displays (**s_seg7_D0** e **s_seg7_D1**) mostram o valor "0" (representado pelo hexadecimal **7E** na simulação).

Neste ponto, o testbench simula o usuário pressionando o botão do 5º andar. Isso é visto como uma breve atividade (indicada por **+** no GTKWave) no barramento **s_dest_request[31:0]**.

A lógica interna (FSM **elevator_controller**) detecta esta nova solicitação na fila de subida (**queue_up**).

A FSM reage à chamada e comanda o motor para subir. Isso é visto na forma de onda como **s_moving** mudando de **"00"** para **"10"** (Subindo).

Consequentemente, o módulo da porta (**door.vhd**) reage ao movimento e fecha a porta, como visto em **s_door_open_closed** mudando de **'1'** para **'0'** (Fechada).

Enquanto **s_moving** está em **"10"**, o sensor de andar interno (**det_andar**) incrementa o andar a cada ciclo de clock.

Os displays reagem a essa mudança, mostrando a contagem (indicada pelos **+** na forma de onda), provando que os módulos **det_andar** e displays estão funcionando em conjunto.

Quando o sensor de andar interno atinge o valor "5", a FSM do **elevator_controller** detecta a chegada. Ela imediatamente comanda a parada do motor, o que é visto como **s_moving** retornando a **"00"** (Parado).

No mesmo instante, a porta é comandada a abrir, com **s_door_open_closed** retornando a **'1'** (Aberta). O display **s_seg7_D0** agora se estabiliza no valor que representa "5"

### 5.3. Cenário 2 (chamada concorrente)

Este cenário simula o primeiro de uma série de chamadas concorrentes e testa a capacidade do escalonador de alocar a chamada ao elevador ocioso mais próximo.

A chamada de teste é:

- **Chamada:** Para **Subir (UP)** no **Andar 2**.
- **Estado dos Elevadores:**
    - **Elevador 0:** Parado no Andar 2.
    - **Elevador 1:** Parado no Andar 0.
    - **Elevador 2:** Parado no Andar 0.

O resultado esperado é que o **Elevador 0** seja escolhido, pois tem a menor distância (zero) e está em um estado "Bom" (parado).

Abaixo está a captura de onda do cenário, que também será usada para o cenário 3.

*Figura 2*


*Captura da forma de onda dos cenários 2 e 3. Reprodução autoral.*

Os principais eventos da simulação são:

O sinal **chamada_ativa_in** (vindo do **call_disp**) muda para **"02"**, indicando que o Andar 2 está sendo processado.

Os três pares de **cost_calculator** e **custo_distancia** calculam os custos para esta chamada:

- **Elevador 0:** Custo 32 (Status "Parado" + Distância 0).
- **Elevador 1:** Custo 34 (Status "Parado" + Distância 2).
- **Elevador 2:** Custo 34 (Status "Parado" + Distância 2).

O **comparador_custo** analisa os valores (32, 34, 34) e define o vencedor. Na onda, o sinal **elevador_vencedor_in** muda para **"00".**

Resultado: O **gerador_saida** recebe o vencedor **"00"** e a chamada **"02".** Ele então roteia a chamada para a saída correta, como visto no sinal **solicite_0_out** (saída para o Elevador 0), que muda para **"02"**. As saídas **solicite_1_out** e **solicite_2_out** permanecem em **"00"**.

### 5.4. Cenário 3 (prioridade de direção)

Este cenário simula uma chamada concorrente onde a prioridade de direção é mais importante que a distância.

- **Chamada:** Para **Descer** no **Andar 10**.
- **Estado dos Elevadores:**

- **Elevador 0:** Parado no Andar 1.
- **Elevador 1:** Subindo ("10") no Andar 2.
- **Elevador 2:** Descendo ("01") no Andar 11.

O resultado esperado é que o **Elevador 2** seja escolhido. Apesar do Elevador 0 estar mais próximo, o Elevador 2 já estava se movendo na direção correta e a chamada estava em seu caminho.

Os principais eventos da simulação são:

O sinal **chamada_ativa_in** muda para **"0A"** (10), indicando que a próxima chamada da fila está sendo processada.

Os calculadores de custo avaliam a nova situação:

- **Elevador 0:** Custo 41 (Status "Parado" + Distância 9).
- **Elevador 1:** Custo 72 (Status "Ruim" - direção oposta + Distância 8).
- **Elevador 2:** Custo 1 (Status "Melhor" - na direção certa e no caminho + Distância 1).

O **comparador_custo** analisa os valores (41, 72, 1) e corretamente identifica o Elevador 2 como o vencedor. Na onda, **elevador_vencedor_in** muda para **"10"**.

Resultado: O **gerador_saida** roteia a chamada para o Elevador 2. Isso é comprovado pelo sinal **solicite_2_out** (saída para o Elevador 2) mudando para **"0A"**. As outras saídas permanecem em **"00"**.

## 6. DISCUSSÕES E CONCLUSÃO

### 6.1. Problemas encontrados e decisões de projeto

Para o desenvolvimento desse projeto, o grupo se encontrou para estabelecer um diagrama de blocos que guiou a construção e elaboração desse projeto. Certas decisões, como a lógica de qual elevador recebe certos pedidos de andar e a concordância de que as chances de dois pedidos serem feitos ao mesmo tempo são praticamente zero, foram tomadas e refletiram na criação dos

componentes e testbenches. Além disso, o grupo encontrou dificuldades em unificar os dois níveis, não conseguindo fazer a comunicação entre a saída do nível 1 e a saída do nível 2, apesar dos dois níveis funcionarem bem quando testados independentemente.

### 6.2. Conclusão

O presente projeto teve êxito em projetar, implementar e validar um sistema de controle para 3 elevadores em VHDL, atendendo aos requisitos funcionais de uma arquitetura de dois níveis. O sistema demonstrou uma separação clara de responsabilidades: os Controladores Locais (nível 1) executaram com sucesso as operações de movimento (MOVING_UP, MOVING_DOWN), parada (IDLE) e controle de portas, enquanto o Escalonador (nível 2) gerenciou eficientemente as chamadas externas. A estratégia de escalonamento adotada, combinando prioridade por direção com o algoritmo nearest-car, provou ser eficaz na otimização de viagens, selecionando o elevador mais apto (baseado em estado e distância) para cada solicitação, conforme validado pelos cenários de simulação. Os testes em simulação (testbenches), cobrindo cenários de chamada simples, concorrência e prioridade de direção, validaram o comportamento esperado do sistema e a correta interação entre os níveis 1 e 2. Desta forma, conclui-se que todos os objetivos propostos foram alcançados, resultando em um sistema de controle de elevadores funcional, modular e devidamente validado.