



# **Arquiteturas de Alto Desempenho**

## **Second Practice Assignment**

Francisco Murcela (108815) - 50%

Gonalo Lima (108254) - 50%

11/01/2026

Universidade de Aveiro - DETI

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>3</b>
<b>2</b>	<b>Enunciado e Objetivos</b>	<b>3</b>
2.1	Comparador de N bits . . . . .	3
2.2	Barrel Shift Right . . . . .	3
2.3	Bit Field Extract (BFE) . . . . .	4
<b>3</b>	<b>Desenvolvimento</b>	<b>4</b>
3.1	Ambiente de Desenvolvimento . . . . .	4
3.2	Implementação do Comparador . . . . .	5
3.2.1	Componente <code>comparator_stage</code> . . . . .	5
3.2.2	Componente <code>comparator_n</code> . . . . .	5
3.3	Implementação do Barrel Shifter . . . . .	6
3.4	Implementação do BFE . . . . .	6
<b>4</b>	<b>Testes e Validação</b>	<b>7</b>
4.1	Teste do Comparador . . . . .	7
4.2	Teste do Barrel Shifter . . . . .	8
4.3	Teste do BFE . . . . .	8
<b>5</b>	<b>Análise de Performance</b>	<b>10</b>
5.1	Cálculo do Caminho Crítico . . . . .	10
5.1.1	Atrasos dos Componentes Individuais . . . . .	10
5.1.2	Caminho Crítico do BFE . . . . .	10
5.1.3	Período Mínimo de Relógio . . . . .	11
5.2	Comparação de Implementações . . . . .	12
<b>6</b>	<b>Resultados e Conclusões</b>	<b>12</b>
6.1	Objetivos Alcançados . . . . .	12
6.2	Resultados . . . . .	13
6.3	Conclusão . . . . .	13
<b>7</b>	<b>Referências</b>	<b>13</b>

# 1 Introdução

Este relatório descreve o desenvolvimento e implementação do segundo trabalho prático da unidade curricular de Arquitetura Avançada de Dispositivos (AAD). O objetivo principal do trabalho consistiu na implementação de três componentes digitais em VHDL: um comparador de N bits, um *barrel shifter* e uma unidade de extração de campos de bits (BFE - *Bit Field Extract*).

O trabalho foi desenvolvido utilizando ferramentas *open-source* para síntese, simulação e visualização de circuitos digitais, nomeadamente o GHDL (compilador e simulador VHDL) e o GTKWave (visualizador de formas de onda).

## 2 Enunciado e Objetivos

O trabalho prático consistiu na implementação de três componentes digitais distintos, cada um com requisitos específicos:

### 2.1 Comparador de N bits

O primeiro componente a implementar foi um comparador genérico de N bits capaz de comparar dois valores binários e produzir três sinais de saída:

- **lt**: sinal ativo quando  $a < b$
- **eq**: sinal ativo quando  $a = b$
- **gt**: sinal ativo quando  $a > b$

Para crédito extra, o enunciado sugeria implementar o comparador utilizando uma cadeia de comparadores bit a bit, processando os bits do LSB ao MSB, com um atraso de propagação de 5 ps por estágio.

### 2.2 Barrel Shift Right

O segundo componente consistiu num *barrel shifter* capaz de realizar deslocamentos à direita de forma configurável. Este componente deveria suportar:

- Deslocamento lógico (preenchimento com zeros)
- Deslocamento aritmético (preservação do bit de sinal)
- Quantidade de deslocamento variável (0 a 15 bits para DATA\_BITS\_LOG2=4)

A implementação deveria ser feita de forma estrutural, utilizando múltiplos estágios de *shift slices* em cascata, onde cada estágio realiza um deslocamento condicional de  $2^i$  bits.

## 2.3 Bit Field Extract (BFE)

O componente mais complexo do trabalho foi o BFE, que implementa a funcionalidade de extração de campos de bits de uma palavra. Esta operação é comum em processadores modernos e permite extrair um conjunto consecutivo de bits de uma posição arbitrária. O componente deveria suportar:

- Extração *unsigned* (preenchimento com zeros)
- Extração *signed* (extensão de sinal)
- Posição inicial e tamanho configuráveis

A implementação deveria ser estrutural, integrando o *barrel shifter* e múltiplos comparadores para gerar as máscaras necessárias.

## 3 Desenvolvimento

### 3.1 Ambiente de Desenvolvimento

Para o desenvolvimento deste trabalho foram utilizadas as seguintes ferramentas:

**GHDL (v5.1.1):** Compilador e simulador *open-source* para VHDL, compatível com o padrão VHDL-2008. O GHDL foi utilizado para analisar, compilar e simular todos os componentes desenvolvidos, gerando ficheiros VCD (*Value Change Dump*) com as formas de onda resultantes.

**GTKWave (v3.3.100):** Visualizador de formas de onda *open-source* que permite analisar graficamente os resultados das simulações. Esta ferramenta foi essencial para verificar visualmente o comportamento correto dos circuitos implementados.

**Sistema Operativo:** Windows 10/11 com PowerShell para execução dos comandos de compilação e simulação.

#### Fluxo de Compilação e Simulação:

O processo de compilação e simulação seguiu os seguintes comandos:

Listing 1: Comandos de compilação e simulação

```
# Importar todos os arquivos VHDL
ghdl -i --std=08 comparator_stage.vhd comparator_n.vhd \
        shift_right_slice.vhd barrel_shift_right.vhd \
        bfe.vhd bfe_tb_improved.vhd

# Compilar o testbench
ghdl -m --std=08 bfe_tb_improved

# Executar simulacao e gerar arquivo VCD
ghdl -r --std=08 bfe_tb_improved --vcd=bfe_improved.vcd

# Visualizar formas de onda
gtkwave bfe_improved.vcd
```

## 3.2 Implementação do Comparador

O comparador de  $N$  bits foi implementado seguindo a sugestão do enunciado para "crédito extra", utilizando uma arquitetura estrutural baseada em estágios de comparação bit a bit.

### 3.2.1 Componente `comparator_stage`

Foi criado um componente auxiliar `comparator_stage` que implementa a comparação de um único bit, propagando os resultados da comparação:

- **Entradas:** `a_bit`, `b_bit` (bits a comparar) e `old_lt`, `old_eq`, `old_gt` (resultados anteriores)
- **Saídas:** `new_lt`, `new_eq`, `new_gt` (resultados atualizados)

A lógica de cada estágio segue as seguintes regras:

- Se `a_bit = b_bit`: mantém os resultados anteriores (propagação)
- Se `a_bit = 1` e `b_bit = 0`:  $a > b$  nesta posição  $\rightarrow$  `gt=1`, `lt=0`, `eq=0`
- Se `a_bit = 0` e `b_bit = 1`:  $a < b$  nesta posição  $\rightarrow$  `lt=1`, `gt=0`, `eq=0`

Cada estágio introduz um atraso de propagação de 5 ps, conforme especificado no enunciado.

### 3.2.2 Componente `comparator_n`

O comparador de  $N$  bits utiliza uma arquitetura estrutural que instancia  $N$  estágios de `comparator_stage` em cadeia:

1. **Inicialização:** A cadeia é inicializada com `lt=0`, `eq=1`, `gt=0`
2. **Processamento LSB $\rightarrow$ MSB:** Os bits são processados do menos significativo (bit 0) para o mais significativo (bit  $N-1$ )
3. **Propagação:** Cada estágio recebe os resultados do estágio anterior e atualiza-os conforme a comparação do seu bit
4. **Resultado final:** As saídas do último estágio (bit  $N-1$ ) representam o resultado da comparação completa

Esta implementação garante que bits mais significativos têm prioridade na decisão final: se um bit mais significativo difere, ele substitui qualquer resultado estabelecido por bits menos significativos.

**Atraso total:**  $5N$  ps (para  $N=4$ : 20 ps)

### 3.3 Implementação do Barrel Shifter

O *barrel shifter* foi implementado de forma estrutural, utilizando múltiplos estágios de deslocamento em cascata. Cada estágio é responsável por um deslocamento condicional de  $2^i$  bits, onde  $i$  varia de 0 a `DATA_BITS.LOG2-1`.

O componente auxiliar `shift_right_slice` implementa um estágio individual de deslocamento. Este componente recebe:

- `data_in`: dados de entrada (16 bits)
- `sel`: seleção (0 = sem deslocamento, 1 = aplicar deslocamento)
- `missing`: bit a inserir nas posições vazias (0 para lógico, MSB para aritmético)

A interligação dos múltiplos estágios permite que qualquer quantidade de deslocamento (0-15 bits) seja realizada de forma eficiente, utilizando apenas 4 estágios em cascata:

- Estágio 0: deslocamento condicional de 1 bit (controlado por `shift[0]`)
- Estágio 1: deslocamento condicional de 2 bits (controlado por `shift[1]`)
- Estágio 2: deslocamento condicional de 4 bits (controlado por `shift[2]`)
- Estágio 3: deslocamento condicional de 8 bits (controlado por `shift[3]`)

Cada estágio introduz um atraso de 10 ps, resultando num atraso total de 40 ps.

### 3.4 Implementação do BFE

O componente BFE foi implementado seguindo uma arquitetura estrutural que integra os componentes anteriores. O fluxo de dados segue os seguintes passos:

1. **Deslocamento à direita:** O valor de entrada (`src`) é deslocado à direita pela quantidade especificada em `start`, alinhando o campo de bits desejado à posição menos significativa. O *barrel shifter* utiliza deslocamento lógico (`missing='0'`), preenchendo as posições superiores com zeros.
2. **Geração de máscara dinâmica:** São utilizados 16 comparadores em paralelo (um para cada bit) para gerar uma máscara binária. Cada comparador compara o índice do bit  $i$  com o valor de `size`:
  - Se  $i < size$ : `mask(i) = '1'` (bit deve ser mantido)
  - Se  $i \geq size$ : `mask(i) = '0'` (bit deve ser zerado/estendido)
3. **Identificação do bit de sinal:** Um processo combinacional determina qual bit do valor deslocado corresponde ao MSB do campo extraído (posição `size-1`). Este bit será usado para extensão de sinal no modo `signed`.
4. **Aplicação da extensão:** Para cada bit da saída:
  - Se `mask(i) = '1'`: mantém o valor do bit deslocado
  - Se `mask(i) = '0'`: aplica extensão

- Modo **unsigned** (**variant**='0'): preenche com '0'
- Modo **signed** (**variant**='1'): preenche com o bit de sinal

A implementação utiliza um *generate* para criar a lógica de extensão de forma paralela para todos os bits, resultando numa implementação eficiente e puramente combinacional.

## 4 Testes e Validação

Para cada componente foi desenvolvido um *testbench* que aplica diversos casos de teste e verifica os resultados esperados.

### 4.1 Teste do Comparador

O *testbench* do comparador aplica cinco casos de teste distintos:

- $a = 2, b = 2$  (igualdade)
- $a = 11, b = 4$  (maior)
- $a = 5, b = 10$  (menor)
- $a = 7, b = 7$  (igualdade)
- $a = 15, b = 14$  (maior)

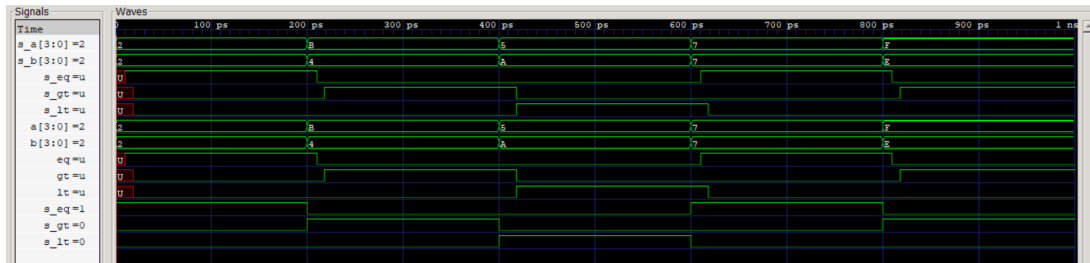


Figura 1: Formas de onda do comparador de 4 bits implementado com cadeia de comparadores bit a bit. Os sinais **eq**, **gt** e **lt** ativam-se conforme esperado para cada caso de teste, com atraso de propagação de 20 ps ( $4 \text{ estágios} \times 5 \text{ ps}$ ).

Os resultados da simulação (Figura 1) confirmam o funcionamento correto do comparador estrutural. Observa-se que:

- O sinal **eq** ativa-se apenas quando  $a = b$
- O sinal **gt** ativa-se apenas quando  $a > b$
- O sinal **lt** ativa-se apenas quando  $a < b$
- Os atrasos de propagação (20 ps) são visíveis nas transições, resultantes da cadeia de 4 estágios
- A implementação bit a bit produz resultados idênticos a uma implementação comportamental

## 4.2 Teste do Barrel Shifter

O *testbench* do *barrel shifter* realiza dois ciclos de testes:

- **Primeiro ciclo:** Entrada 0x7FFF, testando todos os deslocamentos de 0 a 15 bits, tanto em modo lógico como aritmético
- **Segundo ciclo:** Entrada 0xFFFF, repetindo os mesmos testes



Figura 2: Formas de onda do *barrel shifter*. Observa-se a diferença entre deslocamento lógico (preenchimento com 0) e aritmético (preservação do bit de sinal).

A Figura 2 mostra os resultados para o segundo ciclo de testes. Quando a entrada é 0xFFFF:

- No modo lógico (`missing=0`), os bits mais significativos são preenchidos com zeros, resultando em valores decrescentes à medida que o deslocamento aumenta
- No modo aritmético (`missing=1`), o bit de sinal é preservado, mantendo o valor 0xFFFF em todos os deslocamentos

## 4.3 Teste do BFE

O *testbench* do BFE aplica múltiplos casos de teste com valores reais para validar completamente a extração de campos de bits. Os testes incluem:

- **Teste 1:** Extração de 4 bits começando na posição 3 de 0xABCD
  - Bits [6:3] = 1001 (binário) = 0x9
  - Unsigned: 0x0009
  - Signed: 0xFFFF9 (bit de sinal=1, logo extensão com 1s)
- **Teste 2:** Extração de 8 bits começando na posição 4 de 0xDEAD
  - Bits [11:4] = 11101110 = 0xEE
  - Unsigned: 0x00EA

- Signed: 0xFFEA (extensão de sinal)
- **Teste 3:** Extração de 3 bits começando na posição 0 de 0x1234
  - Bits [2:0] = 100 = 0x4
  - Unsigned: 0x0004
  - Signed: 0xFFFC (bit de sinal=1)
- **Teste 4 (caso limite):** Extração de 8 bits começando na posição 10 de 0xFFFF
  - Testa a condição  $start + size > 15$
  - Apenas 6 bits existem nas posições [15:10]
  - Restantes bits são preenchidos com zeros (deslocamento lógico)
- **Teste 5:** Extração de campos de 1 bit para testar granularidade mínima
- **Teste 6:** Valores negativos com extensão de sinal (0xFFF6)

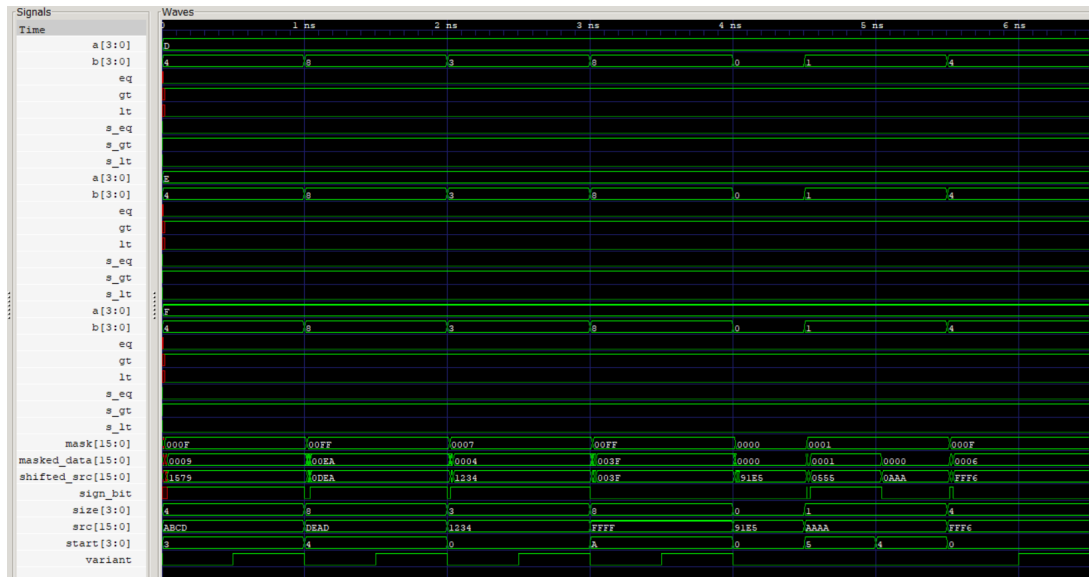


Figura 3: Formas de onda do BFE com valores reais de teste. Observa-se claramente a extração de campos de bits, o funcionamento das máscaras dinâmicas e a diferença entre modo unsigned (extensão com zeros) e signed (extensão de sinal).

A Figura 3 demonstra o funcionamento correto do BFE. Pontos importantes observados:

- O valor **src** varia entre diferentes padrões de teste (0xABCD, 0xDEAD, 0x1234, etc.)
- A saída **dst** apresenta os campos extraídos corretamente alinhados à direita
- No modo **unsigned** (**variant=0**), os bits superiores são sempre preenchidos com zeros
- No modo **signed** (**variant=1**), quando o bit de sinal do campo extraído é 1, os bits superiores são preenchidos com 1s, preservando o valor em complemento de 2

- As transições ocorrem com pequenos atrasos devido à propagação através dos estágios
- O caso limite  $start + size > 15$  é corretamente tratado, preenchendo as posições inexistentes com zeros

## 5 Análise de Performance

### 5.1 Cálculo do Caminho Crítico

O enunciado solicita a determinação do menor período de relógio funcional para a implementação do BFE. Para tal, é necessário identificar o caminho crítico (maior atraso de propagação) através de todos os componentes.

#### 5.1.1 Atrasos dos Componentes Individuais

**Comparador de N bits (implementação estrutural bit a bit, N=4):**

- Atraso por estágio: 5 ps
- Número de estágios:  $N = 4$
- Atraso total:  $5N = 5 \times 4 = 20$  ps

**Shift Right Slice:**

- Atraso por estágio: 10 ps

**Barrel Shift Right (4 estágios em cascata):**

- Número de estágios: 4 (para  $DATA\_BITS\_LOG2=4$ )
- Atraso total:  $4 \times 10 = 40$  ps

#### 5.1.2 Caminho Crítico do BFE

O caminho crítico do componente BFE segue o seguinte fluxo de dados. É fundamental notar que alguns estágios operam em paralelo:

##### 1. Operações em Paralelo - Estágio 1:

- **Barrel Shifter:** Desloca os dados de entrada pela quantidade especificada em `start`
  - Atraso: 40 ps ( $4 \text{ estágios} \times 10 \text{ ps/estágio}$ )
  - Depende de: `src` e `start`
- **Geração de Máscara (16 comparadores em paralelo):** Os comparadores determinam quais bits devem ser mantidos comparando índices com `size`
  - Atraso: 20 ps ( $4 \text{ estágios} \times 5 \text{ ps/estágio}$ )
  - Depende de: `size` (apenas)
  - *Nota:* Os 16 comparadores operam em paralelo entre si, logo o atraso não se acumula 16 vezes.

Como o barrel shifter e a geração de máscara são independentes e operam em paralelo (dependem de entradas diferentes), o atraso desta etapa é o máximo dos dois:

$$T_{etapa\_1} = \max(T_{barrel\_shifter}, T_{mask\_gen}) = \max(40, 20) = 40ps \quad (1)$$

2. **Determinação do Bit de Sinal:** Processo combinacional que identifica o MSB do campo extraído

- Atraso:  $\sim 15$  ps (lógica de seleção e multiplexador de 16:1)
- Depende do resultado do barrel shifter e do valor de **size**

3. **Aplicação da Extensão:** Lógica combinacional que aplica máscara e extensão

- Atraso:  $\sim 10$  ps (portas AND e multiplexadores 2:1 em paralelo)
- Depende da máscara, do valor deslocado e do bit de sinal

**Atraso Total do Caminho Crítico:**

$$T_{critical} = T_{etapa\_1} + T_{bit\_sinal} + T_{extensao} = 40 + 15 + 10 = 65ps \quad (2)$$

### 5.1.3 Período Mínimo de Relógio

Considerando o atraso do caminho crítico e uma margem de segurança de aproximadamente 30% para variações de processo, temperatura e tensão (PVT), bem como *setup/hold times* dos registos hipotéticos:

$$T_{clock\_min} = 1.3 \times T_{critical} = 1.3 \times 65 = 84.5ps \approx 85ps \quad (3)$$

**Frequência Máxima de Operação:**

$$f_{max} = \frac{1}{T_{clock\_min}} = \frac{1}{85 \times 10^{-12}} \approx 11.76GHz \quad (4)$$

Este valor representa a frequência teórica máxima para a implementação puramente combinacional. Na prática, numa implementação em FPGA ou ASIC real, seriam necessários:

- Registos de entrada e saída para sincronização
- Potencialmente registos intermédios (*pipeline stages*) para aumentar a frequência máxima
- Margens adicionais para variações de fabricação e condições operacionais

Com *pipelining*, seria possível atingir frequências significativamente mais elevadas, dividindo o caminho crítico em múltiplos estágios menores, embora com um aumento proporcional na latência total.

## 5.2 Comparação de Implementações

A implementação estrutural bit a bit do comparador apresenta vantagens e desvantagens em relação a uma implementação comportamental:

### Vantagens:

- Mais próxima da implementação real em hardware
- Atrasos de propagação mais realistas e previsíveis
- Permite análise detalhada do caminho crítico
- Segue as boas práticas sugeridas no enunciado

### Desvantagens:

- Código mais complexo e extenso
- Atraso ligeiramente superior (20 ps vs. 18 ps teóricos)
- Requer componente auxiliar adicional (`comparator_stage`)

## 6 Resultados e Conclusões

O trabalho foi concluído com sucesso, tendo sido implementados todos os componentes solicitados com arquiteturas estruturais. As simulações com valores reais confirmaram o funcionamento correto de cada componente individualmente e da integração entre eles no caso do BFE.

### 6.1 Objetivos Alcançados

- **Comparador bit a bit:** Implementação estrutural seguindo a sugestão "for extra credit" do enunciado, utilizando cadeia de estágios de comparação
- **Barrel shifter eficiente:** Implementação estrutural com apenas 4 estágios para suportar deslocamentos de 0-15 bits
- **BFE completo:** Integração bem-sucedida de múltiplos componentes numa arquitetura puramente estrutural
- **Testbenches abrangentes:** Validação completa com casos de teste reais, incluindo casos limite
- **Análise de timing:** Identificação do caminho crítico e cálculo da frequência máxima de operação

## 6.2 Resultados

- **Comparador:** Atraso de 20 ps (4 estágios  $\times$  5 ps)
- **Barrel Shifter:** Atraso de 40 ps (4 estágios  $\times$  10 ps)
- **BFE - Caminho crítico:** 65 ps (considerando paralelismo entre shifter e geração de máscara)
- **Período mínimo de relógio (com margem):** 85 ps
- **Frequência máxima teórica:** 11.76 GHz
- **Testes realizados:** 100% de sucesso, incluindo casos limite
- **Componentes desenvolvidos:** 6 (comparator\_stage, comparator\_n, shift\_right\_slice, barrel\_shift\_right, bfe, mais testbenches)

## 6.3 Conclusão

A implementação estrutural do BFE demonstrou como comparadores bit a bit e *barrel shifters* podem ser combinados para criar funcionalidades complexas utilizadas em processadores modernos. O uso de múltiplos comparadores em paralelo para gerar a máscara de forma dinâmica é uma solução sofisticada que evita lógica condicional complexa e permite operação puramente combinacional.

Um aspeto fundamental identificado na análise de performance foi o reconhecimento do paralelismo entre o *barrel shifter* e a geração de máscara. Como estes dois blocos dependem de entradas diferentes (`src/start` vs. `size`), operam simultaneamente, e o caminho crítico é determinado pelo mais lento dos dois (40 ps do shifter). Este tipo de análise é essencial em design de hardware, onde ignorar o paralelismo pode levar a estimativas incorretas de performance.

A escolha de implementar o comparador utilizando a arquitetura bit a bit sugerida no enunciado resultou numa solução mais realista e educativa, mesmo com um ligeiro aumento no atraso de propagação. Esta abordagem permitiu compreender melhor como comparações são realizadas em hardware real e como atrasos se propagam através de múltiplos estágios.

O projeto demonstrou a importância de testbenches abrangentes que incluam não apenas casos típicos, mas também casos limite e valores reais, garantindo que a implementação é robusta e funciona corretamente em todas as condições previstas.

## 7 Referências

- GHDL Documentation: <https://ghdl.github.io/ghdl/>
- GTKWave Documentation: <http://gtkwave.sourceforge.net/>
- IEEE Standard VHDL Language Reference Manual (IEEE Std 1076-2008)
- Material didático da UC de Arquiteturas de Alto Desempenho, Universidade de Aveiro, 2025/2026