



NOVA SCHOOL OF  
SCIENCE & TECHNOLOGY

DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA E  
DE COMPUTADORES

Mestrado em Engenharia Eletrotécnica e de Computadores

---

## Laboratório 1

---

### Co-design e Sistemas Reconfiguráveis

---

*Alunos:*

João Pedro Antunes - **70380**  
Júlio Lopes- **70512**  
Marcos Romão - **71348**

*Docentes:*

Aniko Costa  
Filipe Moutinho

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>1</b>
1.1	Contextualização . . . . .	1
1.2	Objetivos . . . . .	1
<b>2</b>	<b>Análise Teórica</b>	<b>2</b>
2.1	Redes de Petri . . . . .	2
2.2	Execução síncrona vs. GALS . . . . .	2
2.3	Implementação em FPGA e Arduino . . . . .	3
<b>3</b>	<b>Redes de Petri</b>	<b>4</b>
3.1	IOPT-Tools . . . . .	4
3.1.1	Rede de Petri — Tapete Singular . . . . .	4
3.1.2	Rede de Petri — Modelo Completo . . . . .	5
3.1.3	Rede de Petri — Fusion set . . . . .	6
3.1.4	Rede de Petri — Node set . . . . .	7
3.2	Simulação e Análise . . . . .	7
3.2.1	Simulação com token-player . . . . .	7
3.2.2	Simulação temporal . . . . .	8
3.2.3	Geração e análise do espaço de estados . . . . .	8
<b>4</b>	<b>Implementação em FPGA</b>	<b>9</b>
4.1	Geração do código VHDL . . . . .	9
4.2	Configuração de entradas/saídas . . . . .	9
4.3	Testes experimentais . . . . .	9
<b>5</b>	<b>Implementação em Arduino</b>	<b>10</b>
5.1	Geração do código C . . . . .	10
5.2	Testes . . . . .	10
5.3	Comparação com simulação e implementação FPGA . . . . .	10
<b>6</b>	<b>Controlador Distribuído em regime Síncrono</b>	<b>11</b>
6.1	Cutting Set . . . . .	11
6.2	Decomposição SPLIT . . . . .	11
6.3	Modelo com canais síncronos . . . . .	11
6.4	Modelo com canais assíncronos . . . . .	11
6.5	Simulação e Análise . . . . .	11
6.6	Decomposição GALS . . . . .	11
<b>7</b>	<b>Implementação em FPGA - Distribuição Síncrona</b>	<b>12</b>
7.1	Geração do código VHDL . . . . .	12
7.2	Implementação do LFSR . . . . .	12
7.3	Deployment na FPGA . . . . .	12
7.4	Testes e análise de resultados . . . . .	12

<b>8 Buffers de Capacidade Finita</b>	<b>13</b>
8.1 Modificação do modelo . . . . .	13
8.2 Interconexão dos componentes . . . . .	13
8.3 Módulo de visualização do número de objetos . . . . .	13
8.4 Simulação e validação . . . . .	13
8.5 Implementação centralizada em FPGA . . . . .	13
8.6 Análise de resultados . . . . .	13
<b>9 Distribuído com Buffers (Síncrono)</b>	<b>14</b>
9.1 Aplicação síncrona ao buffer de capacidade finita . . . . .	14
9.2 Decomposição e implementação distribuída . . . . .	14
9.3 Execução síncrona global . . . . .	14
9.4 Testes e análise . . . . .	14
<b>10 Distribuído com Buffers (Assíncrono)</b>	<b>15</b>
10.1 Aplicação assíncrona ao buffer de capacidade finita . . . . .	15
10.2 Comunicações não-instantâneas com buffers . . . . .	15
10.3 Deployment e testes . . . . .	15
10.4 Análise comparativa . . . . .	15
<b>11 Comparação de abordagens</b>	<b>16</b>
11.1 Centralizado vs Distribuído . . . . .	16
11.2 Síncrono vs Assíncrono . . . . .	16
11.3 Impacto dos atrasos de comunicação . . . . .	16
11.4 Vantagens e desvantagens de cada abordagem . . . . .	16
11.5 Análise de escalabilidade . . . . .	16
<b>12 Conclusões</b>	<b>17</b>
12.1 Resultados alcançados . . . . .	17
12.2 Dificuldades encontradas e soluções adotadas . . . . .	17

# 1 Introdução

## 1.1 Contextualização

Este trabalho insere-se no âmbito da unidade curricular de Co-design e Sistemas Reconfiguráveis, focando-se no desenvolvimento e implementação de controladores para sistemas de automação industrial. O caso de estudo considerado é um sistema de produção composto por 3 células em cascata, onde cada célula integra um braço de robô e um tapete rolante.

A complexidade destes sistemas justifica a utilização de metodologias formais de modelação e verificação, permitindo validar o comportamento do controlador antes da sua implementação física. Adicionalmente, a possibilidade de implementar o controlador de forma centralizada ou distribuída oferece diferentes trade-offs entre complexidade de implementação, desempenho e escalabilidade do sistema.

## 1.2 Objetivos

O presente trabalho tem como objetivo principal o desenvolvimento completo de um controlador para um sistema de automação com três células em cascata ( $N=3$ ), desde a fase de modelação até à implementação física em hardware reconfigurável.

Os objetivos específicos incluem:

- Modelar o comportamento do sistema utilizando redes de Petri Input-Output Place-Transition (IOP), explorando as capacidades de composição modular através de operações de adição e fusão de redes;
- Validar e analisar o modelo através de simulação e análise do espaço de estados, verificando propriedades comportamentais críticas do sistema;
- Implementar o controlador em plataformas de hardware distintas (FPGA e Arduino), avaliando a abordagem centralizada de controlo;
- Desenvolver uma arquitetura de controlo distribuído através da decomposição do modelo global, explorando paradigmas de execução síncrona e GALS (Globally Asynchronous Locally Synchronous);
- Analisar o impacto de comunicações não-instantâneas entre controladores distribuídos, introduzindo atrasos pseudo-aleatórios;
- Estender o modelo para suportar buffers de capacidade finita superior a uma unidade, aumentando a flexibilidade do sistema;
- Comparar as diferentes abordagens de implementação, avaliando vantagens, desvantagens e adequação a diferentes contextos aplicacionais.

## 2 Análise Teórica

### 2.1 Redes de Petri

As redes de Petri constituem uma ferramenta formal de modelação adequada para sistemas concorrentes e distribuídos. Uma rede de Petri é definida por uma quádrupla  $PN = (P, T, F, M_0)$ , onde P representa lugares, T transições, F arcos direcionados, e  $M_0$  a marcação inicial. A dinâmica baseia-se no disparo de transições, que removem tokens dos lugares de entrada e adicionam aos de saída.

**Redes IOPT (Input-Output Place-Transition)** estendem as redes clássicas com capacidade de interagir com o ambiente através de:

- Eventos de entrada associados a sinais externos (sensores);
- Sinais de saída para controlo de atuadores;
- Guardas e prioridades para resolução de conflitos;
- Semântica temporal para modelar delays.

**Redes de Petri Coloridas** permitem representar de forma compacta sistemas com estrutura repetitiva. Os tokens possuem "cores" (tipos de dados), permitindo que um único modelo represente múltiplas instâncias de subsistemas idênticos. No sistema estudado, tokens  $\langle 1 \rangle$ ,  $\langle 2 \rangle$  e  $\langle 3 \rangle$  identificam cada célula.

**Composição Modular:** A construção de sistemas complexos utiliza operações de:

- *Merge Nets*: união de múltiplas redes mantendo nós distintos;
- *Fusion Sets*: identificação de nós a fundir criando sincronização;
- *Net Addition*: combinação de merge e fusão num modelo integrado.

### 2.2 Execução síncrona vs. GALS

Em contexto síncrono, todos os componentes operam com um relógio global comum. Oferece simplicidade de design e determinismo, mas apresenta limitações de escalabilidade, consumo energético elevado (relógio sempre ativo) e dificuldades na distribuição do sinal de relógio (clock skew) em sistemas grandes.

Já num paradigma GALS, este divide o sistema em domínios síncronos locais que comunicam assincronamente. Cada domínio opera com relógio independente, comunicando através de protocolos de handshaking ou FIFOs assíncronas.

Vantagens do GALS incluem escalabilidade, modularidade, eficiência energética e tolerância a falhas. As desvantagens são a maior complexidade de interface, latência variável na comunicação e necessidade de tratamento de metastabilidade nas fronteiras entre domínios.

### 2.3 Implementação em FPGA e Arduino

Utilizar-se-á uma **FPGA** (Field-Programmable Gate Array). A descrição do sistema é feita em VHDL, uma linguagem de descrição de hardware que permite especificar comportamento concorrente. Recorrer-se-á também a um microcontrolador **Arduino**.

A framework IOPT-Tools permite geração automática de código VHDL para FPGA e código C para Arduino, facilitando a comparação entre as duas abordagens de implementação.

Critério	FPGA	Arduino
Paralelismo	Hardware real	Software (sequencial)
Tempo de resposta	ns	ms

Tabela 1: Comparaçāo entre plataformas FPGA e Arduino

hyperref

### 3 Redes de Petri

#### 3.1 IOPT-Tools

##### 3.1.1 Rede de Petri — Tapete Singular

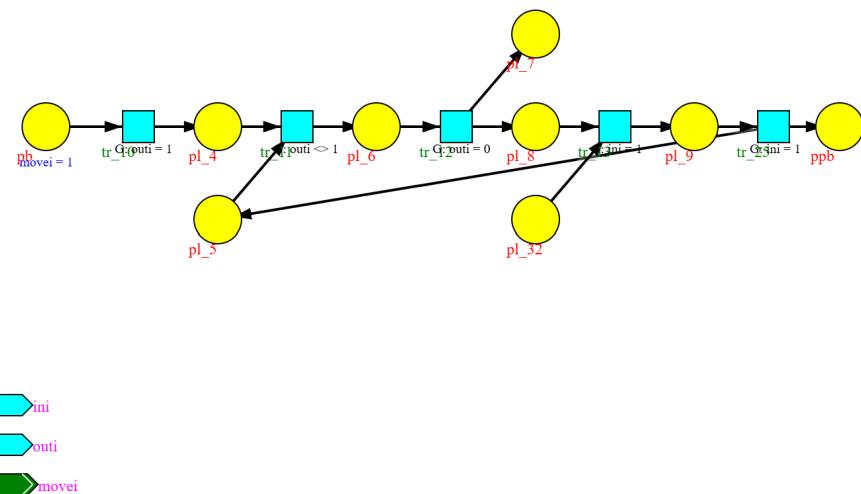


Figura 1: Rede de Petri — Tapete Singular

### 3.1.2 Rede de Petri — Modelo Completo

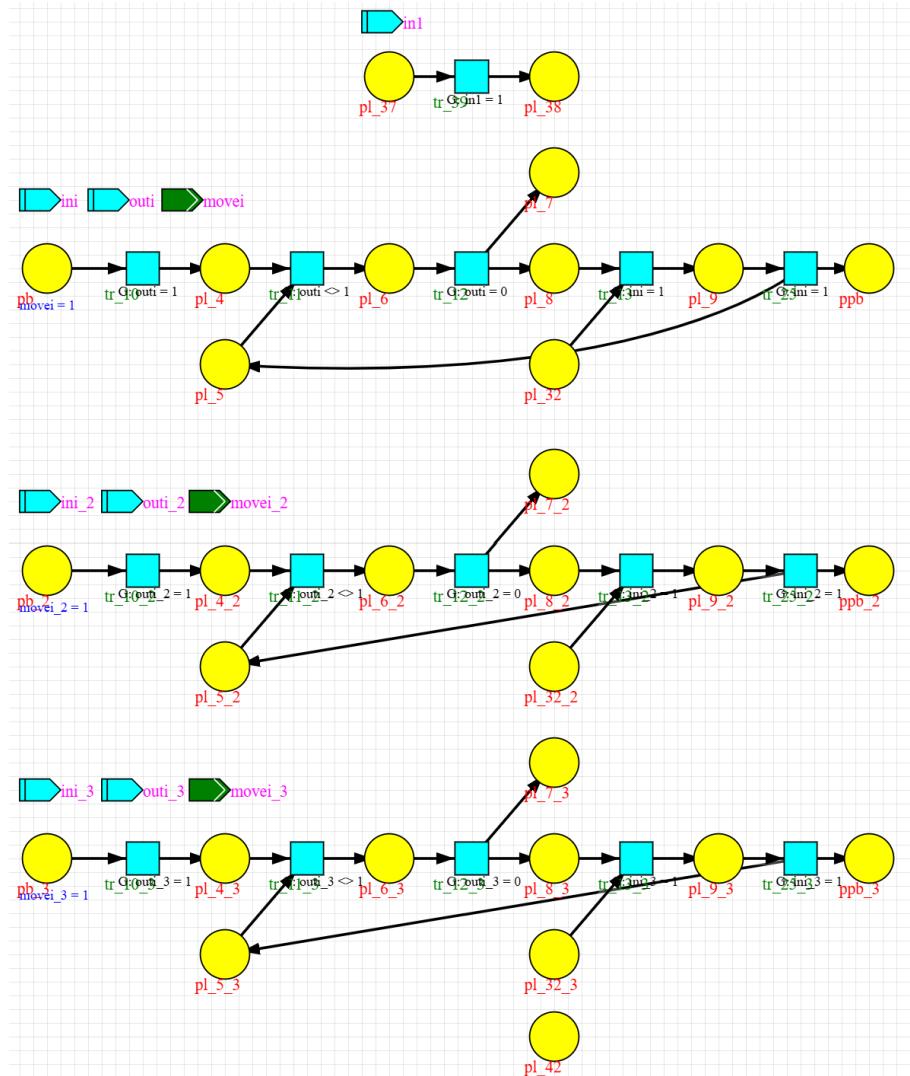


Figura 2: Rede de Petri — Modelo Completo

### 3.1.3 Rede de Petri — Fusion set

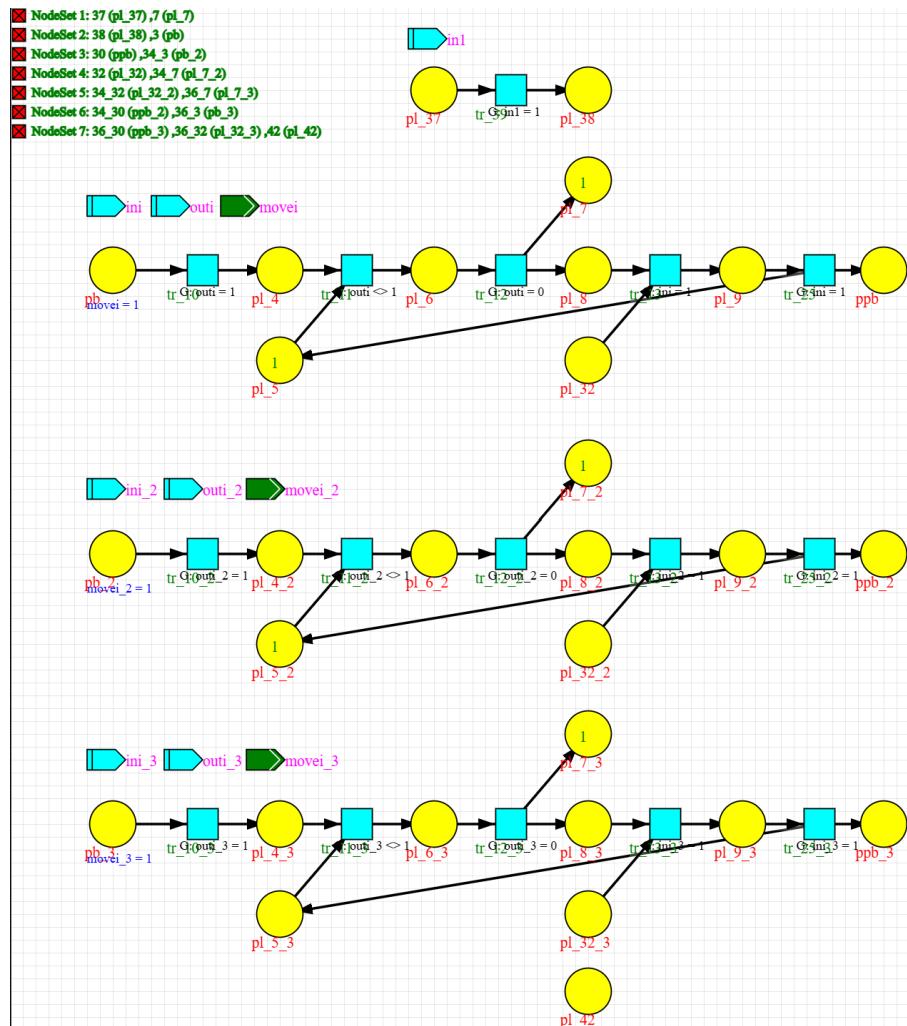


Figura 3: Rede de Petri — Fusion set

### 3.1.4 Rede de Petri — Node set

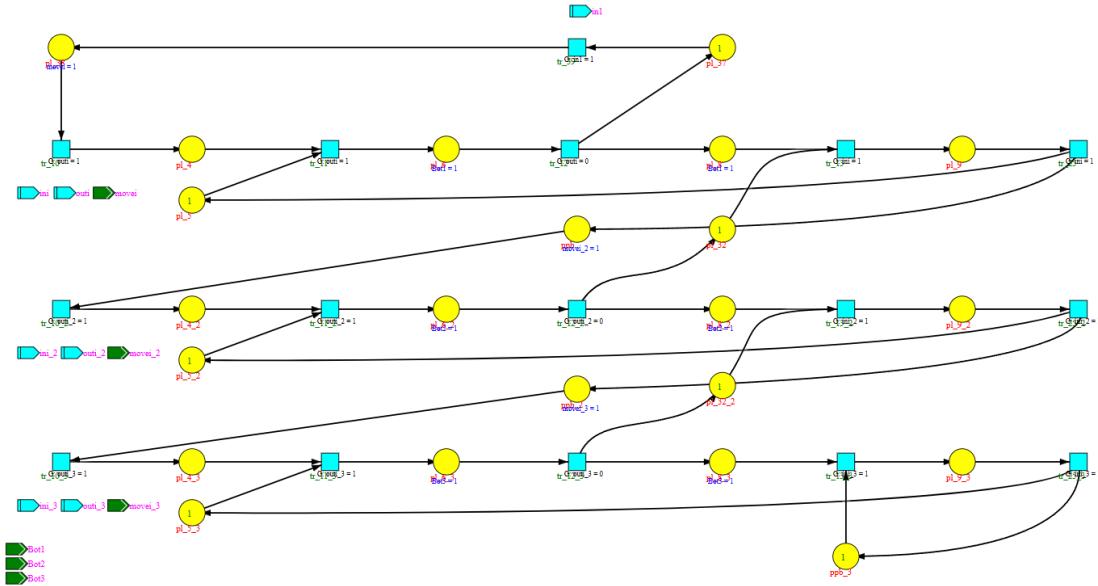


Figura 4: Rede de Petri - Node set

## 3.2 Simulação e Análise

### 3.2.1 Simulação com token-player

Neste link tem acesso ao vídeo da simulação com token-player:

[https://unlpt-my.sharepoint.com/:v/g/personal/jafe\\_lopes\\_fct\\_unl\\_pt/EUvkbe\\_KlbRFrrdI0gsSh98Bu2URieTdJFFzwpahj5HS6A?e=i5XMNg&nav=eyJyZWZlcnJhbEluZm8iOns3D](https://unlpt-my.sharepoint.com/:v/g/personal/jafe_lopes_fct_unl_pt/EUvkbe_KlbRFrrdI0gsSh98Bu2URieTdJFFzwpahj5HS6A?e=i5XMNg&nav=eyJyZWZlcnJhbEluZm8iOns3D)

### 3.2.2 Simulação temporal

### 3.2.3 Geração e análise do espaço de estados

```

586884 Started state-space generation.
Cycle 1: 1 states + 0 links
Cycle 2: 2 states + 0 links
Cycle 3: 3 states + 0 links
Cycle 4: 5 states + 0 links
Cycle 5: 5 states + 0 links
Cycle 6: 8 states + 0 links
Cycle 7: 13 states + 4 links
Cycle 8: 18 states + 10 links
Cycle 9: 25 states + 18 links
Cycle 10: 34 states + 30 links
Cycle 11: 51 states + 48 links
Cycle 12: 70 states + 91 links
Cycle 13: 93 states + 174 links
Cycle 14: 146 states + 306 links
Cycle 15: 211 states + 510 links
Cycle 16: 304 states + 808 links
Cycle 17: 351 states + 1396 links
Cycle 18: 432 states + 1852 links
Cycle 19: 474 states + 2393 links
Cycle 20: 524 states + 2769 links
Cycle 21: 561 states + 3141 links
Cycle 22: 570 states + 3373 links
Cycle 23: 598 states + 3485 links
Cycle 24: 606 states + 3601 links
Cycle 25: 614 states + 3657 links
Cycle 26: 622 states + 3713 links

MIN Bounds: p1_32=0 p1_32_2=0 p1_37=0 p1_38=0 p1_4=0 p1_4_2=0 p1_4_3=0 p1_5=0 p1_5_2=0 p1_5_3=0 p1_6=0 p1_6_2=0 p1_6_3=0 p1_8=0 p1_8_2=0 p1_8_3=0 p1_9=0 p1_9_2=0 p1_9_3=0 ppb=0 ppb_2=0 ppb_3=0
MAX Bounds: p1_32+1 p1_32_2+1 p1_37+1 p1_38+1 p1_4+1 p1_4_2+1 p1_4_3+1 p1_5+1 p1_5_2+1 p1_5_3+1 p1_6+1 p1_6_2+1 p1_6_3+1 p1_8+1 p1_8_2+1 p1_8_3+1 p1_9+1 p1_9_2+1 p1_9_3+1 ppb=1 ppb_2=1 ppb_3=1

#####
Total States: 622
Total Links: 3741
#####

Executing queries...
Done: Found 0 query matching states.

Generation time (sec): 0.00 (when 0.00 it is smaller than 0.01sec)
Generating output file.
Done.

#####
Total States: 622
Total Links: 3741
Deadlock count: 0
Conflict count: 0
Invalid count: 0
#####

```

Figura 5: Rede de Petri — Espaço de Estados

## 4 Implementação em FPGA

- 4.1 Geração do código VHDL
- 4.2 Configuração de entradas/saídas
- 4.3 Testes experimentais

## 5 Implementação em Arduino

### 5.1 Geração do código C

### 5.2 Testes

### 5.3 Comparação com simulação e implementação FPGA

## 6 Controlador Distribuído em regime Síncrono

### 6.1 Cutting Set

A identificação do conjunto de corte (cutting set) é um passo crucial na decomposição de sistemas distribuídos. Este conjunto consiste em um grupo de canais ou conexões que, quando removidos, dividem o sistema em subsistemas independentes. A escolha adequada do cutting set permite a implementação eficiente de controladores distribuídos, minimizando a complexidade da comunicação entre os módulos.

### 6.2 Decomposição SPLIT

A decomposição usando SPLIT (Synchronous Parallel Logic Interconnect Technology) permite a criação de uma arquitetura mais flexível e escalável, dividindo o sistema em módulos independentes que se comunicam através de canais síncronos. Essa abordagem facilita a implementação de sistemas complexos, permitindo a reutilização de componentes e a redução do tempo de desenvolvimento.

### 6.3 Modelo com canais síncronos

A modelação com canais síncronos envolve a definição de canais de comunicação que operam em sincronia, permitindo a troca de informações entre os módulos de forma coordenada. Essa abordagem é fundamental para garantir a consistência e a previsibilidade no comportamento do sistema.

### 6.4 Modelo com canais assíncronos

A modelação com canais assíncronos envolve a definição de canais de comunicação que operam de forma independente, permitindo a troca de informações entre os módulos sem a necessidade de sincronização. Essa abordagem é útil em sistemas onde a latência e a largura de banda são variáveis, proporcionando maior flexibilidade na comunicação.

### 6.5 Simulação e Análise

### 6.6 Decomposição GALS

A decomposição GALS (Globally Asynchronous, Locally Synchronous) envolve a criação de três controladores separados que operam de forma independente, permitindo uma maior flexibilidade e escalabilidade no sistema. Essa abordagem facilita a integração de diferentes módulos e a adaptação a variações nas condições de operação.

## 7 Implementação em FPGA - Distribuição Síncrona

- 7.1 Geração do código VHDL
- 7.2 Implementação do LFSR
- 7.3 Deployment na FPGA
- 7.4 Testes e análise de resultados

## 8 Buffers de Capacidade Finita

### 8.1 Modificação do modelo

A modificação do modelo para múltiplas peças envolve a adaptação dos componentes do sistema para lidar com a entrada e saída de várias peças simultaneamente. Isso inclui a implementação de buffers de capacidade finita que permitem o armazenamento temporário das peças, garantindo que o sistema possa operar de forma eficiente mesmo quando há variações na taxa de entrada ou saída.

### 8.2 Interconexão dos componentes

### 8.3 Módulo de visualização do número de objetos

### 8.4 Simulação e validação

### 8.5 Implementação centralizada em FPGA

### 8.6 Análise de resultados

## 9 Distribuído com Buffers (Síncrono)

### 9.1 Aplicação síncrona ao buffer de capacidade finita

A aplicação síncrona ao buffer de capacidade finita envolve a utilização de canais de comunicação síncronos para gerir o fluxo de dados entre os diferentes módulos do sistema. Esta abordagem permite que o buffer opere de forma eficiente, garantindo que as peças sejam armazenadas e transferidas de acordo com um clock global, o que facilita a coordenação entre os componentes e minimiza o risco de perda ou congestionamento de dados.

### 9.2 Decomposição e implementação distribuída

A decomposição e implementação distribuída do sistema com buffers de capacidade finita envolve a divisão do sistema em módulos independentes que se comunicam através de canais síncronos. Cada módulo é responsável por uma parte específica do processamento, como a gestão do buffer, a entrada e saída de peças, e o controlo dos atuadores. Esta abordagem permite uma maior flexibilidade e escalabilidade, facilitando a adaptação do sistema a diferentes requisitos operacionais.

### 9.3 Execução síncrona global

### 9.4 Testes e análise

## 10 Distribuído com Buffers (Assíncrono)

- 10.1 Aplicação assíncrona ao buffer de capacidade finita
- 10.2 Comunicações não-instantâneas com buffers
- 10.3 Deployment e testes
- 10.4 Análise comparativa

## 11 Comparação de abordagens

11.1 Centralizado vs Distribuído

11.2 Síncrono vs Assíncrono

11.3 Impacto dos atrasos de comunicação

11.4 Vantagens e desvantagens de cada abordagem

11.5 Análise de escalabilidade

## 12 Conclusões

### 12.1 Resultados alcançados

### 12.2 Dificuldades encontradas e soluções adotadas