Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Departamento de Engenharia Elétrica

# Trabalho de Conclusão de Curso

# Estudo de técnicas para processamento paralelo de sinais usando microcontroladores *dual-core*

**Lorran Marcos Dias de Faria**

**João Monlevade, MG**
**2023**

**Lorran Marcos Dias de Faria**

# Estudo de técnicas para processamento paralelo de sinais usando microcontroladores *dual-core*

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Ouro Preto como parte dos requisitos para obtenção do Título de Bacharel em Engenharia Elétrica pelo Instituto de Ciências Exatas e Aplicadas da Universidade Federal de Ouro Preto.
Orientador: Prof. Marcelo Moreira Tiago

**Universidade Federal de Ouro Preto**
**João Monlevade**
**2023**

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

F224e Faria, Lorran Marcos Dias de.
Estudos de técnicas para processamento paralelo de sinais usando microcontroladores dual-core. [manuscrito] / Lorran Marcos Dias de Faria. - 2023.
46 f.: il.: color., tab..

Orientador: Prof. Dr. Marcelo Moreira Tiago.
Monografia (Bacharelado). Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Aplicadas. Graduação em Engenharia Elétrica .

1. Algoritmos paralelos. 2. Microcontroladores. 3. Microprocessadores. 4. Processamento de sinais. 5. Programação paralela (Computação). I. Tiago, Marcelo Moreira. II. Universidade Federal de Ouro Preto. III. Título.

CDU 621.3

Bibliotecário(a) Responsável: Flavia Reis - CRB6-2431

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E APLICADAS
DEPARTAMENTO DE ENGENHARIA ELETRICA

# FOLHA DE APROVAÇÃO

**Lorran Marcos Dias de Faria**

**Estudo de técnicas para processamento paralelo de sinais usando microcontroladores *dual-core***

Monografia apresentada ao Curso de Engenharia Elétrica da Universidade Federal
de Ouro Preto como requisito parcial para obtenção do título de bacharel em Engenharia Elétrica

Aprovada em 11 de dezembro de 2023.

Membros da banca

Dr. Marcelo Moreira Tiago — Orientador — Universidade Federal de Ouro Preto
Dr. Igor Dias Neto de Souza — Universidade Federal de Ouro Preto
Dr. Welbert Alves Rodrigues — Universidade Federal de Ouro Preto

Marcelo Moreira Tiago, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 21/02/2024.

Documento assinado eletronicamente por **Marcelo Moreira Tiago**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 21/02/2024, às 11:32, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0671031** e o código CRC **0C5C1E45**.



R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: (31)3808-0818 - www.ufop.br

# Agradecimentos

Primeiramente agradeço a Deus pela concessão da oportunidade em me especializar com o título de Bacharel, me dando forças, sabedoria e conhecimento para minha permanência no curso. Também sou grato ao meu orientador, Dr. Marcelo Moreira Tiago pelos ensinamentos, paciência e apoio para o cumprimento do trabalho.

Sou grato ao meu pai Fábio, minha mãe Josiane e irmã Fábia Lorenna por todo o apoio emocional e nos momentos de fraqueza sempre acreditaram na minha capacidade, me motivando na realização dos meus objetivos e sonhos.

De forma especial quero agradecer aos meus amigos de estudos, Adeilson Gonçalves, Felipe Alves, Gabriel Machado, Guilherme Henrique, Guilherme Libardi e Gustavo Araújo, que me mostraram o valor de uma boa amizade e companheirismo.

Agradeço também a todos os professores e colegas do ICEA/UFOP que positivamente contribuíram para a minha formação acadêmica.

Externo também meus agradecimentos à minha companheira Martha Gomes, pelo apoio e motivação que foi de muita valia nesta etapa final.

*"Faça o teu melhor, na condição que você tem, enquanto você não tem condições melhores, para fazer melhor ainda!"*

*– Mário Sergio Cortella*

# Resumo

Este trabalho apresenta um estudo de técnicas de programação aplicadas a um microcontrolador de baixo custo. O dispositivo utilizado possui um microprocessador *dual-core* de 32 bits, com frequência de 240 MHz, unidade de ponto flutuante, memórias volátil e não-volátil, além de diversos periféricos destinados a aplicações envolvendo controle, automação e comunicação. Durante o estudo, foram analisadas tanto as características do *hardware* que compõe o dispositivo, tais como o gerenciamento e acesso concorrente à memória e periféricos, quanto as características do sistema operacional utilizado para controlar a ordem de prioridade de execução de tarefas nos dois núcleos de processamento disponíveis. A fim de se avaliar o desempenho do microcontrolador ao executar tarefas com alta complexidade computacional, foram realizados testes utilizando algoritmos para formação de imagens ultrassônicas. Para isso, foram utilizados sinais acústicos obtidos a partir da inspeção de um bloco de alumínio que apresentava um conjunto de defeitos com diâmetro e posições previamente conhecidas. A Técnica de Focalização por Abertura Sintética (do inglês, *Synthetic Aperture Focusing Technique*) (SAFT) foi utilizada para processar os sinais referentes a uma área limitada do bloco (20 mm$\times$ 20 mm). Foram geradas imagens com qualidade baixa (2601 *pixels*), média (5184 *pixels*) e alta (12544 *pixels*). Três abordagens diferentes foram utilizadas para implementar os algoritmos de processamento: uma implementação sequencial, onde um único núcleo é responsável por processar todas as informações; uma implementação paralela, com definição prévia do núcleo de processamento responsável por processar cada tarefa; e uma segunda implementação paralela, em que a definição do núcleo de processamento passa a ser feita de forma dinâmica, pelo Sistema Operacional em Tempo Real (do inglês, *Real-Time Operating System*) (RTOS). Os resultados obtidos mostram que, no melhor caso, o sistema pode processar imagens de baixa qualidade num tempo aproximado de 600 ms, limitado pelo fato de não ser possível realizar um acesso concorrente a memória e aos periféricos presentes no dispositivo.

**Palavras-chave**: Microcontroladores, Microprocessadores, RTOS, paralelização de algoritmos.

# Abstract

This work presents a study of programming techniques applied to a low-cost microcontroller. The used device features a 32-bit dual-core microprocessor with a frequency of 240 MHz, floating-point unit, volatile and non-volatile memories, as well as various peripherals designed for applications involving control, automation, and communication. During the study, both the hardware characteristics of the device, such as memory and peripheral management and concurrent access, and the characteristics of the operating system used to control the task execution priority in the two available processing cores were analyzed. To evaluate the microcontroller's performance in executing tasks with high computational complexity, tests were conducted using algorithms for ultrasonic image formation. Acoustic signals obtained from inspecting an aluminum block with a set of defects of known diameter and positions were used for this purpose. The Synthetic Aperture Focusing Technique (SAFT) was employed to process signals related to a limited area of the block (20 mm$\times$ 20 mm). Images with low (2601 pixels), medium (5184 pixels), and high (12544 pixels) quality were generated. Three different approaches were used to implement the processing algorithms: a sequential implementation, where a single core is responsible for processing all information; a parallel implementation with predefined processing core assignment for each task; and a second parallel implementation where processing core assignment is dynamically determined by the RTOS. The results obtained show that, in the best case, the system can process low-quality images in approximately 600 ms, limited by the inability to perform concurrent access to memory and peripherals present in the device.

**Keywords**: Microcontrollers, Microprocessors, RTOS, algorithm parallelization.

# Lista de ilustrações


Figura 1 – Diagrama comparativo entre arquiteturas de sistemas microprocessados de núcleo simples (*single-core*) e de múltiplos núcleos (*multi-core*). . . 2
Figura 2 – Arquitetura de processadores Xtensa LX6 representada a partir de uma diagrama de blocos. . . . . 9
Figura 3 – Diagrama esquemático com a organização dos processadores, dispositivos e periféricos presentes no microcontrolador ESP32. . . . . 10
Figura 4 – Estrutura de endereçamento de memória e periféricos utilizada por processadores Xtensa. . . . . 12
Figura 5 – Compartilhamento de memória entre dois processadores por meio de um barramento *multimaster*. . . . . 12
Figura 6 – Sistema de compartilhamento de memória de porta dupla por meio de barramentos locais separados. . . . . 13
Figura 7 – Compartilhamento na memória local através do barramento PIF usando operações PIF de entrada . . . . . 14
Figura 8 – Estrutura utilizada para conectar os dois núcleos de processamento aos bancos de memória cache. . . . . 15
Figura 9 – Organização do módulo controlador de DMA (DMA_ENGINE). . . . . 16
Figura 10 – Diagrama esquemático com um exemplo de controle de gerenciamento de múltiplas tarefas por um sistema RTOS. . . . . 18
Figura 11 – *Array* com $N$ elementos transdutores. . . . . 22
Figura 12 – *Array* de transdutores utilizados para identificação de um defeito num corpo de prova. . . . . 23
Figura 13 – *Array* de transdutores utilizados para caracterizar um determinado meio. 24
Figura 14 – Diagrama esquemático do circuito eletrônico utilizado ao longo do processo de aquisição dos sinais ultrassônicos utilizados neste trabalho. . . 27
Figura 15 – Peça de alumínio analisada por meio de ensaios não-destrutivos por ultrassom. . . . . 27
Figura 16 – Fluxograma do algoritmo utilizado para processar os dados sequencialmente, considerando apenas um núcleo de processamento. . . . . 33
Figura 17 – Fluxograma do algoritmo utilizado para processar os dados paralelamente, com tarefas pré-fixadas para cada núcleo de processamento. . . 34
Figura 18 – Fluxograma do algoritmo utilizado para processar os dados paralelamente, com alocação dinâmica das tarefas. . . . . 35
Figura 19 – Área de inspeção da peça de alumínio analisada a partir do ensaio não-destrutivo por ultrassom. . . . . 38

Figura 20 – Comparação entre os resultados obtidos ao processar imagens de baixa qualidade em processadores de 32 bits e 64 bits. . . . . 39
Figura 21 – Comparação entre os resultados obtidos ao processar imagens de qualidade intermediária em processadores de 32 bits e 64 bits. . . . . 40
Figura 22 – Comparação entre os resultados obtidos ao processar imagens de alta qualidade em processadores de 32 bits e 64 bits. . . . . 40

# Lista de tabelas

Tabela 1 – Memórias disponívies no microcontrolador ESP32-WROOM-32. . . . . 14
Tabela 2 – Modos de operação da memória cache disponível nos microcontroladores ESP32-WROOM-32. . . . . 15
Tabela 3 – Dados do ensaio com *array* de transdutores. . . . . 28
Tabela 4 – Definição do espaço necessário para armazenamento do algoritmo e dos dados necessários para gerar imagens com diferentes resoluções a partir do sistema microcontrolado. . . . . 32
Tabela 5 – Comparação entre os tempos de processamento necessários para executar os algoritmos propostos neste trabalho. . . . . 41

# Lista de siglas e abreviaturas

| | |
|---|---|
| **AMP** | Multiprocessamento Assimétrico (do inglês, *Asymmetric Multiprocessing*) |
| **AUTOSAR** | Arquitetura Aberta para Sistemas Automotivos (do inglês, *Automotive Open System Architecture*) |
| **BIST** | *Built-In Self Test* |
| **CPU** | Unidade Central de Processamento (do inglês, *Central Processing Unit*) |
| **DMA** | Acesso Direto à Memória (do inglês, *Direct Memory Access*) |
| **DSP** | Processador de Sinais Digitais (do inglês, *Digital Signal Processor*) |
| **FPU** | Unidade de Ponto Flutuante (do inglês, *Floating Point Unit*) |
| **FreeRTOS** | Sistema Operacional em Tempo Real Livre (do inglês, *Free Real-Time Operating System*) |
| **GPGPU** | Unidade de Processamento Gráfico de Propósito Geral (do inglês, *General Purpose Graphics Processing Unit*) |
| **GPU** | Unidade de Processamento Gráfico (do inglês, *Graphics Processing Unit*) |
| **MATLAB** | *Matrix Laboratory* |
| **MMU** | Unidade de Gerenciamento de Memória (do inglês, *Memory Management Unit*) |
| **PCNT** | Contador de Pulsos (do inglês, *Pulse Counter*) |
| **PID** | Controlador Proporcional Integral Derivativo (do inglês, *Proportional Integral Derivative Controller*) |
| **PIF** | Interface de Protocolo (do inglês, *Protocol Interface*) |
| **RAM** | Memória de Acesso Aleatório (do inglês, *Random Access Memory*) |
| **RISC** | (do inglês *Reduced Instruction Set Computer*) |
| **ROM** | Memória Somente de Leitura (do inglês, *Read-Only Memory*) |
| **RTOS** | Sistema Operacional em Tempo Real (do inglês, *Real-Time Operating System*) |
| **SAFT** | Técnica de Focalização por Abertura Sintética (do inglês, *Synthetic Aperture Focusing Technique*) |
| **SBST** | Teste Automático Baseado em Software (do inglês, *Software-Based Self Test*) |

| | |
|---|---|
| **SMP** | Multiprocessamento Simétrico (do inglês, *Symmetric Multiprocessing*) |
| **SOC** | Sistema em um Chip (do inglês, *System on Chip*) |
| **SRAM** | Memória de Acesso Aleatório Estática (do inglês, *Static Random-Access Memory*) |
| **ULA** | Unidade Lógica Aritmética (do inglês, *Arithmetic Logic Unit*) |
| **USART** | Receptor/Transmissor Universal Assíncrono/Síncrono (*Universal Synchronous/Asynchronous Receiver/Transmitter*) |
| **XLMI** | Interface de Memória Local Xtensa (do inglês, *Xtensa Local Memory Interface*) |

# Sumário


| | | |
|---|---|---|
| **1** | **INTRODUÇÃO** | **1** |
| **1.1** | **Motivações** | **3** |
| **1.2** | **Objetivos** | **4** |
| **1.3** | **Revisão bibliográfica** | **4** |
| **1.4** | **Estrutura do trabalho** | **6** |
| **2** | **REVISÃO TEÓRICA** | **8** |
| **2.1** | **Arquitetura do processador Xtensa LX6** | **8** |
| **2.2** | **Memória** | **11** |
| **2.3** | **Periféricos** | **16** |
| **2.4** | **Sistemas operacionais de tempo real** | **17** |
| **2.5** | **Ambientes de desenvolvimento** | **19** |
| **2.6** | **Técnica de focalização por abertura sintética** | **21** |
| **2.7** | **SAFT-TFM** | **23** |
| **2.8** | **Considerações parciais** | **25** |
| **3** | **METODOLOGIA** | **26** |
| **3.1** | **Dados utilizados para a realização dos ensaios** | **26** |
| **3.2** | **Testes iniciais usando computador pessoal** | **28** |
| **3.3** | **Preparação dos dados para utilização no sistema embarcado** | **30** |
| **3.4** | **Programação do sistema embarcado** | **32** |
| **3.5** | **Considerações parciais** | **36** |
| **4** | **RESULTADOS** | **37** |
| **4.1** | **Limitações do ambiente de desenvolvimento** | **37** |
| **4.2** | **Impacto da precisão das variáveis utilizadas para armazenamento de dados na imagem final** | **38** |
| **4.3** | **Comparação dos tempos de execução dos algoritmos implementados.** | **41** |
| **4.4** | **Considerações parciais** | **42** |
| **5** | **CONSIDERAÇÕES FINAIS** | **44** |
| **5.1** | **Propostas para trabalhos futuros** | **45** |
| | **REFERÊNCIAS** | **46** |

# 1 Introdução

O crescente aumento da complexidade dos sistemas de controle industriais tem exigido dos fabricantes um aumento da capacidade de processamento dos sistemas microprocessados e micro controlados. Os fabricantes constataram que, em vez de criar novos sistemas com um único núcleo, o que tornaria o projeto mais complexo e dispendioso devido às melhorias na arquitetura dos processadores, é mais eficiente usar vários núcleos paralelos simples, permitindo a execução simultânea de diferentes tarefas. (HARRINGTON; NG, 2012).

A necessidade de adaptação dos projetos relacionados a arquitetura dos microcontroladores visando o aumento do desempenho desses dispositivos no que diz repeito a capacidade de processamento digital de sinais em tempo real motivou fabricantes a atualizarem o projeto de seus dispositivos, que passaram a incorporar características antes presente apenas em microprocessadores como, por exemplo, arquiteturas com mais de um núcleo de processamento (*multi-core*), acesso simultâneo à memória e periféricos e compartilhamento de informações por meio de memória cache. Como consequência desse processo, novas arquiteturas e tecnologias de fabricação foram desenvolvidas para reduzir o consumo de energia e aumentar a eficiência desses processadores.

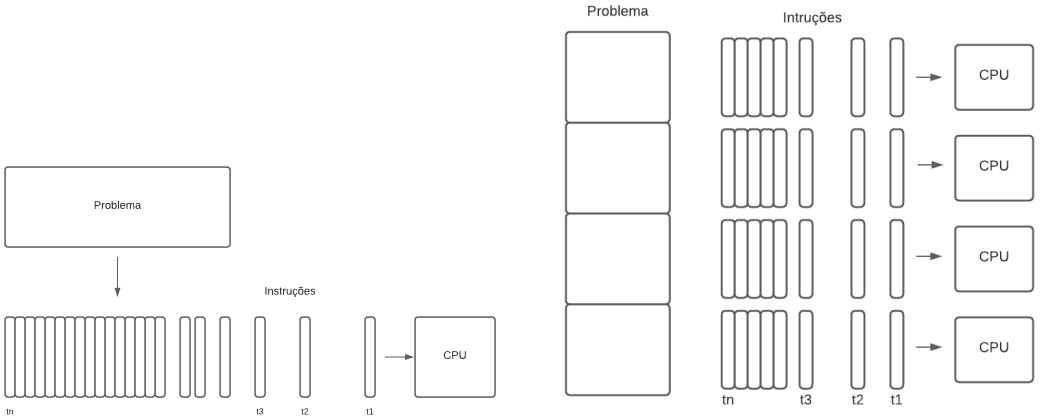
Um dos principais pontos que deve ser considerado para que se possa aumentar, de forma significativa, a velocidade de processamento de sistemas com mais de um núcleo diz respeito à forma como os algoritmos são implementados, que pode ser sequencial ou concorrente. Para que um sistema com mais de um núcleo opere eficientemente, processando dados simultaneamente nas unidades de processamento presentes no dispositivo, não deve haver dependência entre as funções executadas pelos núcleos do processador, seja com relação à comunicação entre eles ou à utilização de recursos compartilhados (JUNGKLASS et al., 2019). Trata-se de um problema que pode ocorrer em sistemas que possuem compartilhamento de memórias e periféricos a partir de um mesmo barramento de dados.

Um dos desafios da utilização de dispositivos com tais características é desenvolver um *software* capaz de gerenciar eficientemente os recursos disponíveis, evitando que um dos núcleos permaneça inoperante no momento em que outro dispositivo acessa o barramento compartilhado. Este *software* deve considerar a capacidade de processamento dos núcleos, a possibilidade de se realizar acessos múltiplos à memória ou não, e a possibilidade de se atribuir tarefas pré-fixadas (cada tarefa alocada a um núcleo) ou de forma pós-fixadas, onde a tarefa será alocada para o núcleo que estiver livre.

Essas diferenças na arquitetura e gerenciamento dos núcleos, memória e periféricos fizeram com que técnicas de paralelização, utilizadas com maior frequência em sistemas microprocessados, fossem utilizadas também em sistemas micro controlados. A Figura 1

apresenta uma comparação entre os dois tipos de arquitetura mencionados.

Figura 1 – Diagrama comparativo entre arquiteturas de sistemas microprocessados de núcleo simples (*single-core*) e de múltiplos núcleos (*multi-core*).




(a) Arquitetura *single-core*. (b) Arquitetura *multi-core*.

Fonte: do autor.

Observa-se que a vantagem de se utilizar uma arquitetura *multi-core* é que pode-se executar tarefas concorrentes simultaneamente. Dessa forma, considerando-se processadores com mesma frequência de operação, assumindo-se que as funções implementadas sejam independentes e o acesso à memória e periféricos possa ser realizado de forma simultânea, pode-se aumentar de maneira significativa a velocidade de execução desses algoritmos (BELL; WOOD, 2009).

Esse conjunto de fatores, aliados à boa relação custo-benefício desses microcontroladores, tem contribuído para popularização da utilização deste tipo de processador em aplicações que envolvam desde sistemas embarcados tradicionais (como controladores de veículos, eletrodomésticos e equipamentos industriais) até sistemas de automação conectados à internet (como sistemas IoT (do inglês, *Internet das coisas*) e aplicações relacionadas à indústria 4.0) (HENNESSY; PATTERSON, 2011).

A fim de avaliar as diferenças construtivas destes microcontroladores em relação aos modelos tradicionais de núcleo simples, este trabalho apresenta um estudo sobre a arquitetura e programação do microcontrolador *dual-core* (Espressif, modelo ESP32-WROOM-32). Para isso, são descritas as principais características construtivas do microcontrolador analisado, tais como organização de memória e periféricos, além das estratégias de compartilhamento de recursos entre os núcleos de processamento definidas pelo fabricante do dispositivo. Finalmente, são descritas as bibliotecas necessárias para programar o microcontrolador, com destaque para o sistema operacional de tempo real (RTOS), utilizado para gerenciar os recursos compartilhados e controlar os tempos de execução das tarefas, e as ferramentas de programação utilizadas ao longo do desenvolvimento das aplicações,

assim como o conjunto de bibliotecas.

## 1.1 Motivações

Para avaliar o desempenho desses dispositivos ao executar tarefas de processamento mais complexas, o microcontrolador em estudo foi utilizado como plataforma para processamento de algoritmos de focalização por abertura sintética (SAFT), utilizado para formação de imagens acústicas.

Esse algoritmo é bastante empregado em aplicações envolvendo ensaios não-destrutivos por ultrassom, tais como detecção de defeitos estruturais, falhas em pontos de solda, detecção de trincas, entre outros. A mesma estratégia também pode ser utilizada em aplicações médicas, em sistemas utilizados para disgnósticos por imagens como, por exemplo, ecocardiogramas.

Por meio desta técnica, um conjunto de sinais acústicos é adquirido a partir de transdutores ultrassônicos. Os sinais adquiridos são processados, levando-se em consideração as propriedades físicas do material analisado (tais como densidade, velocidade de propagação do som, entre outras). Além disso, considera-se também a dimensão do material a ser inspecionado e a resolução da imagem que pretende-se gerar.

A partir do processamento desses sinais, pode-se gerar imagens acústicas capazes de representar a localização espacial e a extensão de falhas contidas em objetos. Uma estrutura de aquisição da matriz de dados ultrassônicos por meio de um *array* de transdutores foi implementado por pesquisadores do *Instituto de Tecnologias Físicas y de la Información* Leonardo Torres Quevedo, e Laorden et al. (2012), devido ao grau de complexidade, propôs o processamento SAFT da matriz de sinais ultrassônicos por um sistema composto por processadores *multi-core* e Unidade de Processamento Gráfico (do inglês, *Graphics Processing Unit*) (GPU).

Logo, os dados utilizados para processamento em microcontroladores de baixo custo foram oriundos da estrutura de aquisição base já implementada, assim, ressalta-se que o objetivo deste trabalho não é apresentar uma estrutura nova de aquisição de imagens ultrassônicas e alternativas de baixo custo para sistemas de processamento complexo (como os mencionados anteriormente), nem tampouco desenvolver novos algoritmos para formação de imagens acústicas, mas sim analisar o desempenho e principais características de uma plataforma de processamento micro controlada implementada a partir de um processador de dois núcleos, validando análises de desempenho com abordagem de processamento para algoritmos mais complexos.

## 1.2 Objetivos

O objetivo geral deste trabalho é estudar técnicas para processamento paralelos de algoritmos utilizando microcontroladores com dois núcleos de processamento. Os objetivos específicos do trabalho são:

- Compreender o princípio de funcionamento, organização e arquitetura de um processador Espressif Xtensa *dual-core* de 240 MHz;
- Avaliar as características dos periféricos presentes no microcontrolador Espressif, modelo ESP32-WROOM-32;
- Compreender o processo de organização dos periféricos e memória presentes no microcontrolador a partir de um barramento de dados compartilhado.
- Estudar técnicas de programação para desenvolver algoritmos usando como base o sistema operacional de tempo real FreeRTOS;
- Comparar 3 casos: execução sequencial, paralelização com atribuição específica de tarefas a cada núcleo (tarefas pré-fixadas) e paralelização com o uso de um RTOS para alocação flexível de núcleos (tarefas pós-fixadas).
- Comparar como diferentes tipos de imagens são geradas em Unidade Lógica Aritmética (do inglês, *Arithmetic Logic Unit*) (ULA) de 32 e 64 bits, e como isso afeta a qualidade das imagens geradas por um algoritmo de geração de imagens ultrassônicas.

## 1.3 Revisão bibliográfica

Esta seção visa embasar o desenvolvimento da pesquisa científica a cerca de conhecimentos envolvendo implementações de algoritmos concorrentes em microcontroladores. Foram analisados trabalhos cujo objetivo era comparar o desempenho de algoritmos executados em microcontroladores de núcleo simples e com múltiplos núcleos, avaliando-se o tempo de execução e complexidade para implementação dos mesmos.

As características da tecnologia utilizada atualmente na fabricação de processadores fez com que a sua organização alcançasse seus limites, de modo que um avanço relevante e significativo seria possível com a interconexão de uma multiplicidade de processadores, permitindo uma solução cooperativa. De acordo com Peccerillo et al. (2022), uma das primeiras soluções adotadas para aprimorar o desempenho dos processadores foi a implementação do conceito conhecido como *"multi-core"*, envolvendo a coexistência de vários núcleos de processamento na mesma Unidade Central de Processamento (do inglês, *Central Processing Unit*) (CPU). A tecnologia atual utilizada na fabricação de processadores alcançou seus limites, impulsionando a necessidade de avanços significativos.

Segundo Gebali (2011), seria complexo e custoso melhorar o desempenho de um computador com arquiteturas que utilizam um único processador, mas problemas computacionais mais complexos poderiam ser solucionado utilizando estruturas de processamento paralelas com múltiplos processadores, desde que os algoritmos desenvolvidos pudessem ser paralelizados. Assim, a otimização do desempenho não se limitaria apenas ao *hardware*, mas também à programação paralela em todos os níveis.

Azevedo e Barros (2015) implementam um sistema paralelo distribuído para controlar um conversor CC-CC *buck* em veículos elétricos. A metodologia adotada consiste em utilizar um microcontrolador para monitorar a corrente e realizar o controle em tempo real através de um Controlador Proporcional Integral Derivativo (do inglês, *Proportional Integral Derivative Controller*) (PID). Foram utilizados vários microcontroladores, e os resultados obtidos pelo autor chegam a cerca de 10 vezes mais rápidos em testes com três microcontroladores quando comparados a implementação com um único microcontrolador.

Biondi et al. (2016) apresenta um estudo de caso que permite a abordagem do problema de migração de um sistema complexo de uma plataforma mono-nuclear para uma plataforma multi-nuclear, visando implementar um sistema de controle de posicionamento automotivo em arquitetura Arquitetura Aberta para Sistemas Automotivos (do inglês, *Automotive Open System Architecture*) (AUTOSAR). Ao longo do trabalho, foram analisados algoritmos de otimização linear, cujo custo computacional é relativamente alto, porém com simplificações nas formulações de prioridades os autores aceleraram a solução ótima utilizando o algoritmo para problema de alocação em uma plataforma de quatro núcleos.

De acordo com Floridia et al. (2018), a inevitável introdução de dispositivos *multi-cores* gera um problema no que tange aos testes, mesmo com ganhos de desempenho, sendo necessário o desenvolvimento de estratégias para testes em campo, com único núcleo e vários núcleos. A metodologia aborda uma forma de utilização de semáforos de *hardware* para controlar e reduzir o acesso a recursos compartilhados entre os diferentes núcleos, utilizando um microcontrolador *multi-core* fabricado pela *STMicroelectronics*. A exploração dos semáforos de *hardware* permitiu a implementação paralela de programas de testes entre diferentes núcleos, tendo como alvo o uso de técnicas Teste Automático Baseado em Software (do inglês, *Software-Based Self Test*) (SBST), levantando comparativo com técnicas *Built-In Self Test* (BIST)

A paralelização de algoritmos é amplamente abordada em aplicações que envolvem medicina instrumental, pois tamanho e consumo de energia são fatores importantes para equipamentos médicos móveis. Schonle et al. (2017) apresenta um sistema Sistema em um Chip (do inglês, *System on Chip*) (SOC), que combina um microcontrolador *quad-core* e uma unidade de processamento *multi-core* de baixa potência com circuitos analógicos e subsistemas de estimulação neural para sistemas de telemetria implantáveis, visando

compreender melhor os mecanismos neurofisiológicos. O autor apresenta uma proposta de desenvolvimento de um subsistema configurável, que possibilita o monitoramento de alguns sinais vitais, tais como frequência cardíaca, oxigenação arterial e frequência respiratória.

Jungklass e Berekovic (2018) defendem que o uso de mais núcleos de processamento tornam o acesso a recursos compartilhados de memória e periféricos mais custosos, ou seja, o acesso simultâneo à memória compartilhada pode aumentar os tempos de acessos e reduzir o desempenho dos núcleos individuais. O objetivo dos autores é mostrar como o aumento no número de núcleo influencia no desempenho do processador, encontrando solução para o acesso competitivo focando no gerenciamento de memória de microcontroladores *multi-core*.

Testes realizados por Jungklass e Berekovic (2018) com microcontroladores de três núcleos (Infineon, modelo AURIX TC277) e de seis núcleos (Infineon, modelo AURIX TC298) mostraram que, de forma geral, o desempenho do sistema aumenta com o aumento do número de núcleos. Entretanto, notou-se que o desempenho de cada núcleo sofre uma baixa devido ao acesso concorrente à memória, algo que segundo os autores, poderia ser minimizado com a utilização de algoritmos divididos em tarefas sem acesso concorrente à memória. Como proposta, os autores apresentam um algoritmo de distribuição automática de memória, que visa reduzir o número de acessos concorrentes a recursos compartilhados.

## 1.4 Estrutura do trabalho

Este trabalho é composto por cinco capítulos.

No Capítulo 1 apresenta-se uma introdução ao tema apresentado neste trabalho. São apresentadas as motivações e justificativas para desenvolvimento do mesmo, bem como os objetivos e uma revisão bibliográfica sobre o assunto proposto.

No Capítulo 2 apresenta-se uma revisão teórica, descrevendo os aspectos construtivos relacionados ao projeto do *hardware* do microcontrolador utilizado, bem como as principais características relacionadas à programação, ambiente de desenvolvimento e paralelização do algoritmo que será embarcado no sistema.

No Capítulo 3 descreve-se a metodologia de testes proposta para avaliar o desempenho do sistema, comparando os tempos de execução de um algoritmo SAFT, utilizado para gerar imagens acústicas, executado sequencialmente (considerando-se apenas um dos núcleos de processamento) e concorrente (usando os dois núcleos de processamento).

No Capítulo 4, apresentam-se os resultados obtidos ao longo do trabalho, com destaque para os tempos de execução dos algoritmos implementados. Nesse sentido, foram avaliados diferentes cenários, alterando-se não só a estratégia utilizada para processar os sinais mas também a resolução das imagens acústicas geradas.

No Capítulo 5, são apresentadas as considerações finais do trabalho, destacando-se os principais pontos observados ao longo dos ensaios realizados. Além disso, são apresentadas algumas propostas para desenvolvimento de trabalhos futuros.

# 2 Revisão teórica

Este capítulo apresenta um conjunto de conceitos teóricos e práticos que serão utilizados no decorrer do trabalho. Todo o estudo foi realizado utilizando-se como referência o kit de desenvolvimento ESP32-DevKitC V4, com microcontrolador ESP32-WROOM-32, fabricado pela empresa Espressif Systems. São descritos os principais pontos relacionados à arquitetura, organização de memória, periféricos e programação do microcontrolador em questão, apresentando também exemplos de algoritmos que podem ser utilizados para paralelizar tarefas e gerenciar os dispositivos periféricos.

## 2.1 Arquitetura do processador Xtensa LX6

O microcontrolador ESP-WROOM-32 possui um processador de dois núcleos, com duas CPU de arquitetura Xtensa, com barramento interno de 32 bits (Tensilica, Xtensa LX6). Estes processadores podem ser fornecidos a partir de uma configuração pré-definida pelo fabricante ou podem ser customizados por projetistas de circuitos integrados que desejam desenvolver sistemas SOC com características específicas para um determinado projeto.

Esse é um diferencial deste processador em relação a processadores com arquiteturas de núcleos simples tradicionais, que não costumam permitir aprimoramentos com facilidade. Essa característica costuma forçar os projetistas a resolver os problemas usando técnicas baseadas em ajustes de *software*, gerando soluções mais lentas e com maiores consumos de energia.

A solução apresentada pelo fabricante permite que os projetistas incorporem novas funcionalidades a um processador como, por exemplo, customização do *pipeline* adicionando mais dispositivos de entrada e saída (I/O) (TENSILICA, 2010).

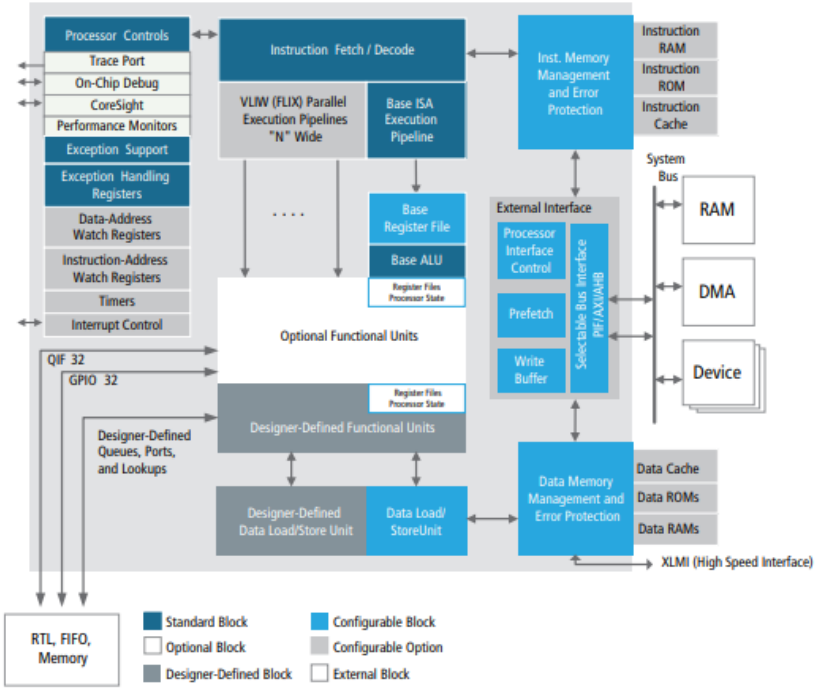
A arquitetura Xtensa Lx6 apresenta um conjunto compacto de instruções para projetos de sistemas embarcados. O sistema conta com uma ULA de 32 bits, com 64 registradores físicos de uso geral, 6 registradores de propósito especial e 80 instruções básicas, incluindo codificação de instruções (do inglês *Reduced Instruction Set Computer*) (RISC) (TENSILICA, INC, 2014).

O núcleo do processador é construído com base na arquitetura Xtensa ISA (*Instruction Set Architecture*), que opera instruções de 16 e 24 bits, executam operações lógicas e aritméticas de 32 bits. Conforme o fabricante, a opção pela utilização de instruções com menor número de bits possibilita uma redução do tamanho do programa desenvolvido e, consequentemente, uma redução dos custos relacionados a memórias Memória Somente de Leitura (do inglês, *Read-Only Memory*) (ROM), Flash ou Memória de Acesso Alea-

tório (do inglês, *Random Access Memory*) (RAM) necessárias para a implementação das aplicações (LEIBSON, 2006).

O conjunto de instruções executadas pelo núcleo Xtensa é resultado de pesquisas voltadas para o equilíbrio entre recursos de *hardware* e custo para atender às necessidades do mercado de processadores embarcados. A Figura 2 ilustra um diagrama de blocos do processador Xtensa, com todos os componentes que podem ser configuráveis.

Figura 2 – Arquitetura de processadores Xtensa LX6 representada a partir de uma diagrama de blocos.




Fonte: retirado de Tensilica, Inc (2014).

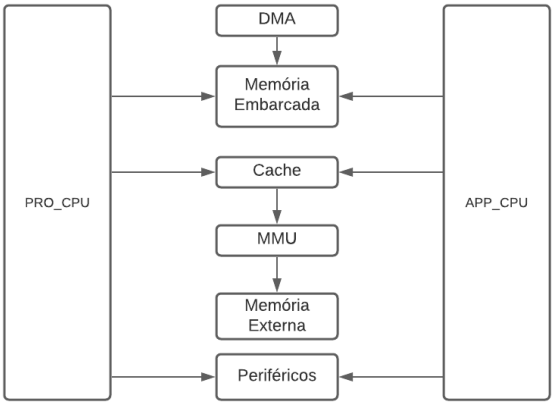
Caso a configuração do núcleo fosse pré-definida, todos os elementos do diagrama de blocos deveriam estar presentes em cada instancia do processador, resultando em um circuito eletrônico maior. Um núcleo com estas características certamente teria um grande poder computacional, porém apresentaria um maior custo, consumiria maior potência e não seria customizável para o desenvolvimento de conjuntos específicos de tarefas. Assim, pode-se dimensionar o processador baseado no Xtensa ISA que são ajustados aos perfis de desempenho requerido pelo *designer*.

No diagrama apresentado, os elementos configuráveis incluem interfaces para instruções locais e memória de dados, interfaces para caches de instrução e dados, barramento da interface principal do processador Interface de Protocolo (do inglês, *Protocol Interface*) (PIF), unidades de execução funcional predefinidas (como um multiplicador, multiplicador/acumulador), uma unidade de ponto flutuante Unidade de Ponto Flutuante (do inglês, *Floating Point Unit*) (FPU), uma unidade Processador de Sinais Digitais (do inglês, *Digital Signal Processor*) (DSP), uma unidade de carga/armazenamento, inter-

rupções e temporizadores, blocos de gerenciamento e proteção de memória, além de uma unidade para rastreamento e depuração (LEIBSON, 2006).

Pode-se então listar e separar as funcionalidades do processador, classificando-as em funções básicas, configuráveis e opcionais, conforme o diagrama apresentado na Figura 2. Na classificação de funções básicas, estão incluídos uma ULA de 32 bits, *pipeline* de 7 estágios e um conjunto básico de 80 instruções com tamanhos de 16 e 24 bits. Nas funções configuráveis, pode-se citar os registradores de uso geral e de uso específico. Finalmente, para as funções opcionais, destacam-se um multiplicador de 32 bits e um núcleo específico para processamento digital de sinais DSP. A Figura 3 ilustra o diagrama de blocos da estrutura do sistema utilizado pelo microcontrolador ESP32-WROOM-32.

Figura 3 – Diagrama esquemático com a organização dos processadores, dispositivos e periféricos presentes no microcontrolador ESP32.




Fonte: do autor.

É possível observar a presença de dois núcleos físicos que compõem o processador, chamados de unidade de protocolo (PRO_CPU) e unidade de aplicação (APP_CPU). Sempre que o microcontrolador é inicializado, o núcleo PRO_CPU é energizado e processa as instruções definidas pelo algoritmo armazenado na memória de programa. Devido a esta característica, este núcleo é chamado de núcleo padrão do processador.

O núcleo APP_CPU é chamado de núcleo secundário, e por padrão permanece desligado até que um comando de ativação proveniente do PRO_CPU seja recebido. Uma vez ativo, este núcleo começa a executar as instruções armazenadas nas posições de memória indicadas por seu contador de programa (GAY, 2020). É importante ressaltar que as funções de cada núcleo não são fixas, podendo ser permutadas segundo os interesses do programador.

O processamento dos algoritmos armazenados na memória de programa dos microcontroladores *multi-core* pode ser feito de forma assimétrica Multiprocessamento Assimétrico (do inglês, *Asymmetric Multiprocessing*) (AMP) ou simétrica Multiprocessamento Simétrico (do inglês, *Symmetric Multiprocessing*) (SMP). O ESP32-WROOM-32 possui

dois núcleos com as mesmas características (núcleos gêmeos) e trabalha de forma simétrica. Neste modo de operação, os processadores compartilham uma mesma memória, gerenciada por um sistema operacional de tempo real RTOS.

A fabricante Espressif disponibiliza uma versão de sistema operacional de tempo real que pode ser utilizada livremente. Esse sistema, denominado Sistema Operacional em Tempo Real Livre (do inglês, *Free Real-Time Operating System*) (FreeRTOS), foi inicialmente desenvolvido e implementado em sistemas de núcleo simples, sendo posteriormente adaptado para gerenciar microcontroladores com dois ou mais núcleos de processamento.

A partir do sistema FreeRTOS, o programador pode criar tarefas, defina os níveis de prioridade de execução das funções de forma arbitrária e atribuir cada atividade a um determinado núcleo. A partir dessa definição, o escalonador do sistema irá decidir qual tarefa será executada em função da ordem de uma fila de prioridades (GAY, 2020).

Vários periféricos do sistema podem acessar diretamente a memória compartilhada por Acesso Direto à Memória (do inglês, *Direct Memory Access*) (DMA), e cada CPU pode acessar diretamente a memória por meio dos barramentos de dados e instruções. A memória externa, que compõe o sistema, é mapeada pela Unidade de Gerenciamento de Memória (do inglês, *Memory Management Unit*) (MMU) e por alguns periféricos (ESPRESSIF SYSTEMS, 2023).

## 2.2 Memória

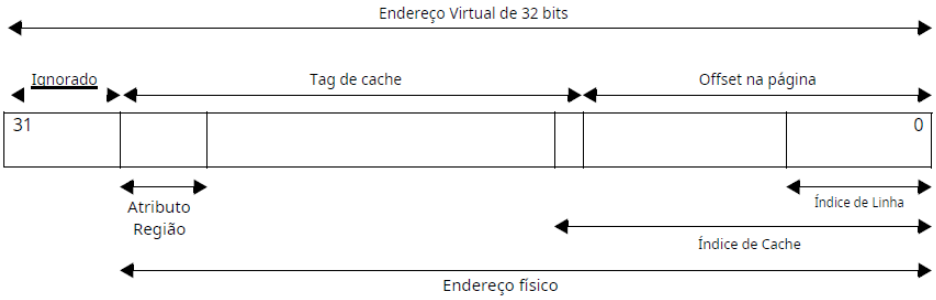
Os núcleos do processador Xtensa possuem portas de comunicação dedicadas para acesso à memória Interface de Memória Local Xtensa (do inglês, *Xtensa Local Memory Interface*) (XLMI). Cada porta XLMI possui um sinalizador (do inglês, *flag*), que indica se a memória associada está sendo usada por alguma lógica externa ao processador. Nesses casos, o processador não pode executar operações de busca, carregamento ou armazenamento envolvendo a memória selecionada. Esses *flags* podem ser usados para compartilhar uma mesma memória entre diferentes unidades de processamento, ou para permitir que a lógica externa carregue a memória local.

Os núcleos do processador Xtensa possuem uma interface de barramento principal PIF que suporta a carga, armazenamento e transações de busca de instruções e transações de entrada por um dispositivo externo ao processador. As operações de leitura e gravação pelo barramento PIF podem ser direcionados para as memórias locais de um processador para que os dados armazenados nessas memórias possam ser lidas ou modificadas por dispositivos externos (LEIBSON, 2006).

O sistema Xtensa ISA emprega uma arquitetura Harvard, implicando em uma separação física entre os barramentos de instrução e dados. Embora haja um compartilhamento do mesmo espaço de endereço, as memórias locais são divididas em memórias de instrução e de dados, com emprego também de memórias caches separadas.

Esses barramentos são capazes de gerenciar endereços de memória física e virtual de 32 bits, totalizando 4 GB de espaço de endereçamento para instruções e dados (LEIBSON, 2006). A Figura 4 apresenta um exemplo de estrutura de endereçamento utilizada pelo processador para acessar periféricos e memória.

Figura 4 – Estrutura de endereçamento de memória e periféricos utilizada por processadores Xtensa.

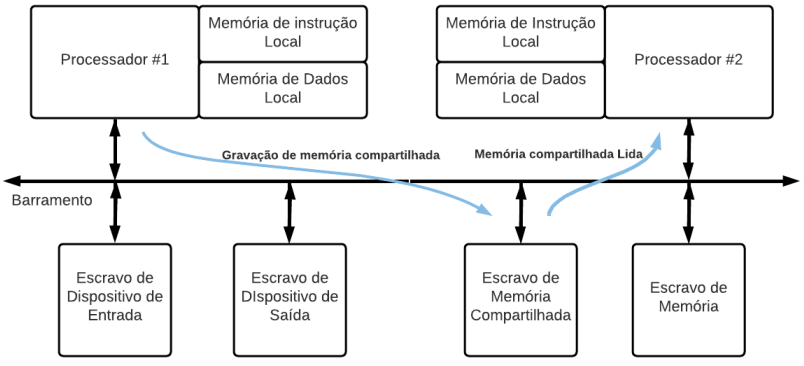



Fonte: adaptado de Tensilica (2010).

Destaca-se que a capacidade de armazenamento de memória (volátil, não-volátil e cache), faixas de endereço reservados para as memórias e a largura das interfaces de barramento são opções de configuração que podem ser definidas pelo projetista. As operações de armazenamento e busca de endereços que não estão alocados nas memórias locais (conforme definido na configuração do processador) são direcionadas para a interface de barramento PIF do processador Xtensa.

Existem várias topologias de sistemas que empregam memórias compartilhadas. Essa configuração costuma ser utilizada em casos onde deseja-se transferir muitos dados entre processadores do sistema. A Figura 5 apresenta um exemplo desse tipo de topologia, onde dois processadores compartilham uma memória por meio de um barramento *multimaster* comum, num arranjo utilizado por arquiteturas mais simples.

Figura 5 – Compartilhamento de memória entre dois processadores por meio de um barramento *multimaster*.

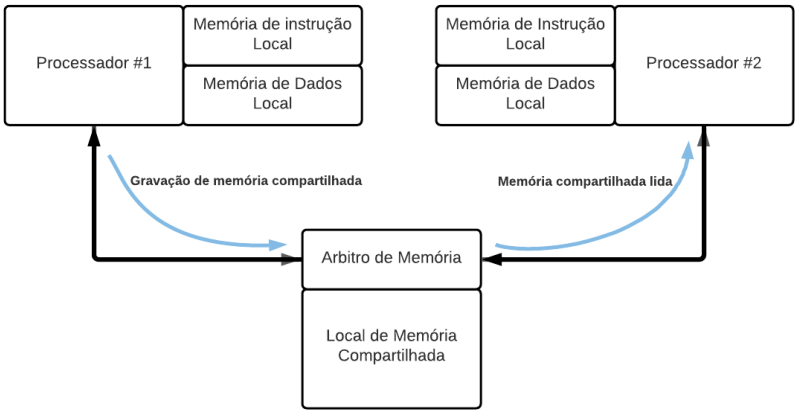



Fonte: adaptado de Leibson (2006).

Nesta topologia, a interface do processador Xtensa trabalha com memórias compartilhadas em um barramento *multimaster* comum. Esta abordagem apresenta alguns problemas de desempenho quando há altas taxas de transferência de dados nos dois processadores. Para contornar o problema, o barramento passa a ser acessado de forma cíclica por cada um dos processadores. Esse processo exige que os núcleos sinalizem os instantes em que assumiram o controle do barramento, evitando acessos indesejados do outro núcleo ou de algum periférico.

Outra topologia utilizada para compartilhamento de memória é mostrada na Figura 6. Nessa configuração, para diminuir a sobrecarga de um único barramento, cada processador se conecta à memória compartilhada por meio de um barramento local separado, e o controle de acesso à memória é feito por um bloco lógico.

Figura 6 – Sistema de compartilhamento de memória de porta dupla por meio de barramentos locais separados.



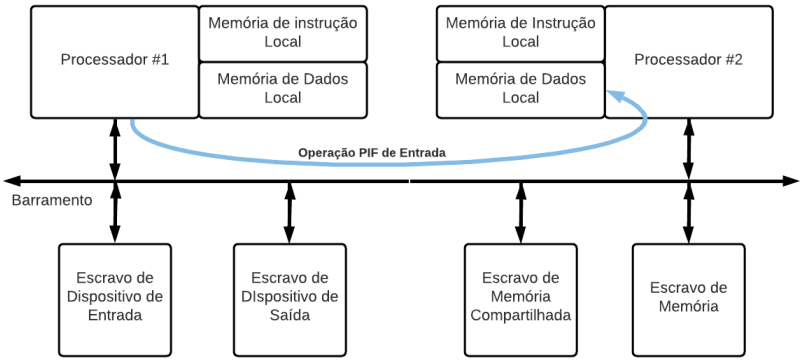

Fonte: adaptado de Leibson (2006)

Além das duas abordagens já mencionadas, os núcleos do processador Xtensa também podem acessar diretamente as memórias locais um do outro, por meio de um barramento PIF comum. A Figura 7 detalha como esse processo é realizado, ilustrando a utilização do recurso de entrada PIF do processador.

Fisicamente, este método é o mesmo que o ilustrado na Figura 5, porém apenas metade das transações de memórias são conduzidas pelo PIF, uma vez que a outra parte das transações ocorre no barramento de memória local não compartilhada. A vantagem de se trabalhar desta forma é que as transações podem ocorrer no barramento de memória local de cada núcleo simultaneamente, sem criar conflitos, otimizando sistemas que incorporam vários processadores.

Um dos atributos da arquitetura Harvard é a separação dos barramentos de dados e instruções. Nos processadores Xtensa, endereços abaixo de 0x4000_0000 são acessados pelo barramento de dados, ao passo que endereços no intervalo de 0x4000_0000 até 0x4FFF_FFFF são acessados utilizando o barramento de instrução. Endereços acima de

Figura 7 – Compartilhamento na memória local através do barramento PIF usando operações PIF de entrada




Fonte: adaptado de Leibson (2006)

0x5000_0000 são compartilhados, e podem ser acessados pelos barramentos de dados e instrução (ESPRESSIF SYSTEMS, 2023).

Cada núcleo de processamento pode acessar diretamente a memória embarcada por meio do barramento de dados e de instrução. A memória externa é mapeada pelo módulo MMU e periféricos.

A memória disponível no microcontrolador ESP32-WROOM-32 foi organizada da seguinte forma, como apresentado na Tabela 1.

Tabela 1 – Memórias disponívies no microcontrolador ESP32-WROOM-32.

| Memória | Tamanho | Função |
|---|---|---|
| ROM | 448 KB | Inicialização e funções principais |
| SRAM | 520 KB | Dados e instruções |
| Flash Embarcada | 4 MB | Firmware e dados |

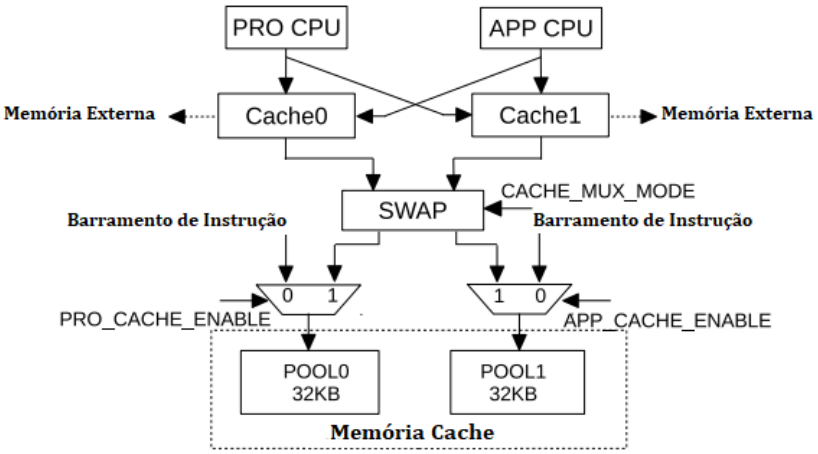
Fonte: Espressif Systems (2023).

Todas as unidades de memória embarcadas podem ser acessadas pelos dois núcleos de processamento (PRO_CPU e APP_CPU), com exceção da memória RTC FAST, que está localizada numa faixa de endereços que só pode ser acessada pelo núcleo PRO_CPU.

Cada núcleo de processamento possui 32 KB de memória cache, utilizados para endereçar as posições de memória das unidades de armazenamento externo, implementadas como SRAM 0. Bits de configuração diferentes devem ser utilizados para habilitar a memória cache de cada núcleo: para o núcleo PRO_CPU, utiliza-se o bit PRO_CACHE_ENABLE, e para o núcleo APP_CPU, utiliza-se o bit APP_CACHE_ENABLE. A Figura 8 apresenta um diagrama esquemático com as memórias cache mencionadas anteriormente.

O microcontrolador ESP32-WROOM-32 utiliza uma memória cache de conjunto associativo bidirecional. Dessa forma, quando a memória estiver sendo utilizada pelos

Figura 8 – Estrutura utilizada para conectar os dois núcleos de processamento aos bancos de memória cache.




Fonte: adaptado de Espressif Systems (2023).

núcleos PRO_CPU ou APP_CPU, o bit CACHE_MUX_MODE pode ser configurado para selecionar POOL0 ou POOL1 na SRAM interna 0 como memória cache. Quando PRO_CPU e APP_CPU estiverem utilizando a função Cache, POOL0 e POOL1 na SRAM0 interna serão usados simultaneamente como memória cache.

A Tabela 2 apresenta um resumo das opções descritas anteriormente (ESPRESSIF SYSTEMS, 2023). É possível observar que o bit CACHE_MUX_MODE não permite que os núcleos PRO_CPU e APP_CPU habilitem as funções de cache simultaneamente.

Tabela 2 – Modos de operação da memória cache disponível nos microcontroladores ESP32-WROOM-32.

| CACHE_MUX_MODE | POLO0 | POLO1 |
|---|---|---|
| 0 | PRO CPU | APP CPU |
| 1 | PRO CPU/APP CPU | - |
| 2 | - | PRO CPU/APP CPU |
| 3 | APP CPU | PRO CPU |

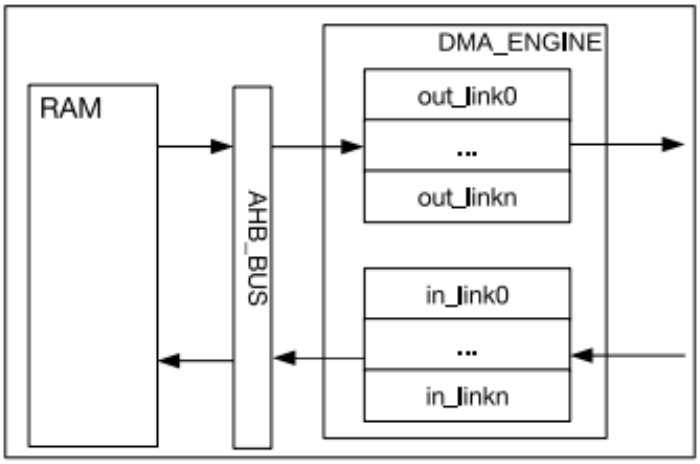
Fonte: adaptado de Espressif Systems (2023).

O microcontrolador possui ainda um controlador de DMA , usado para transferência de dados em alta velocidade entre periféricos e memória, e vice-versa. Esse método permite que os dados sejam transferidos de forma direta, sem a necessidade de participação dos processadores durante o processo.

No ESP32-WROOM-32, treze periféricos são capazes de transferir dados diretamente por DMA: UART0, UART1, UART2, SPI1, SPI2, SPI3 I2S0, I2S1, servo SDIO, HostSD/MMC, EMAC, BT e Wi-Fi (ESPRESSIF SYSTEMS, 2023). A Figura 9 apresenta a representação do controlador de DMA utilizado. Ressalta-se que o endereçamento

do controlador de DMA é feito utilizando o mesmo barramento de dados que as CPUs usam para realizar operações de leitura e escrita na memória RAM interna.

Figura 9 – Organização do módulo controlador de DMA (DMA_ENGINE).




Fonte: adaptado de Espressif Systems (2023).

No diagrama de blocos apresentado, o bloco RAM representa os bancos Memória de Acesso Aleatório Estática (do inglês, *Static Random-Access Memory*) (SRAM) internos disponíveis no microcontrolador. O controlador de DMA (DMA_ENGINE) pode ser utilizado tanto para transmitir quanto para receber dados. Em transmissão, os dados da RAM são enviados para um periférico, seguindo as definições configuradas por meio do registrador *out_link*. No modo recepção, os dados recebidos do periférico são armazenados nos endereços de memória RAM definidos conforme a configuração do registrador *in_link* (ESPRESSIF SYSTEMS, 2023).

## 2.3 Periféricos

O microcontrolador ESP32-WROOM-32 possui internamente vários dispositivos periféricos, capazes de gerenciar processos de comunicação, aquisição e temporização. Esses dispositivos podem ser organizados em quatro grupos principais: entradas/saídas digitais, entradas/saídas analógicas, comunicações e funções especiais.

O ESP32 possui módulos de comunicação Receptor/Transmissor Universal Assíncrono/Síncrono (*Universal Synchronous/Asynchronous Receiver/Transmitter*) (USART) que permitem a transmissão e recepção de dados de forma assíncrona. A UART é um protocolo amplamente utilizado para comunicação serial entre dispositivos. Ela consiste em um par de pinos, transmitindo dados de forma serial, um bit por vez, de maneira assíncrona.

Com a USART, é possível estabelecer comunicação serial com outros dispositivos, como sensores, módulos GPS, microcontroladores, entre outros. Oferecendo flexibilidade

para configurar velocidades de comunicação (*baud rate*), bits de dados, bits de parada e paridade, permitindo adaptar a comunicação segundo os requisitos do dispositivo conectado.

Finalmente, também fazem parte do sistema alguns periféricos especiais, como o módulo Contador de Pulsos (do inglês, *Pulse Counter*) (PCNT) e *timers*. Os *timers*, são essenciais para o gerenciamento de tempo, e podem ser utilizados para computar intervalos, controlar atrasos e realizar ações baseadas em tempo. Com o módulo *timer*, é possível criar temporizadores precisos para sincronizar eventos ou medir intervalos de tempo.

No contexto do processamento paralelo, os *timers* podem ser empregados para medir o tempo de execução de determinadas tarefas em cada núcleo do ESP32. Isso é crucial para o monitoramento do desempenho e para garantir que o processamento seja realizado em intervalos de tempo específicos. Todos os periféricos podem ser acessados pelos dois núcleos, mas como já mencionado, os acessos não podem ser concorrentes. Dessa forma, as tarefas devem ser organizadas de forma que o sistema FreeRTOS defina a utilização instantânea de apenas um periférico por núcleo, sem que seja necessário deixar um dos núcleos numa fila de espera.

## 2.4 Sistemas operacionais de tempo real

As aplicações práticas de um RTOS estão relacionadas à necessidade de controle de múltiplas tarefas em sistemas que trabalham com restrições de tempo. Dessa forma, sistemas RTOS tem por objetivo garantir que eventos sejam cumpridos, produzindo saídas/-respostas a partir de processamentos realizados por estímulos e/ou eventos, nas restrições temporais de um determinado sistema.

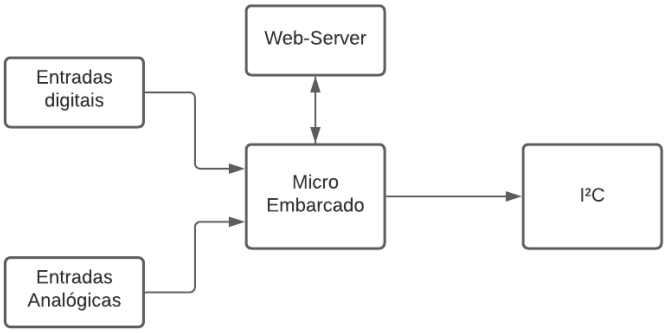
Um evento com restrição temporal é aquele que necessita de um tempo máximo para que se gere uma saída processada. Logo, um sistema operacional de tempo real deve reduzir riscos de problemas com restrições de tempo. A estratégia utilizada por sistemas RTOS para minimizar esse tipo de problema consiste na divisão de algoritmos complexos em elementos independentes, denominados tarefas (do inglês, *tasks*) (LEIBSON, 2006). A Figura 10 ilustra um exemplo de gerenciamento de múltiplas tarefas por um RTOS.

Cada tarefa possui um atributo de prioridade, que deve ser definido conforme a importância do trecho de código a ser executado. Aquisição de dados, monitoramento de sensores, comunicação e acionamentos de dispositivos são exemplos de tarefas e eventos que podem ser executados de forma eficiente com a divisão proporcionada pelo RTOS.

O desempenho do RTOS depende da maneira como a divisão das tarefas foi proposta e do modo com o problema a ser resolvido foi modelado. Essas definições afetam diretamente a confiabilidade e a qualidade do projeto.

Alguns fatores devem ser considerados quando se deseja analisar a viabilidade da

Figura 10 – Diagrama esquemático com um exemplo de controle de gerenciamento de múltiplas tarefas por um sistema RTOS.




Fonte: do autor.

utilização de um sistema RTOS num microcontrolador, tais como o tipo de CPU, RAM, ROM e SRAM. A escolha de CPU é um dos fatores mais importantes, pois microcontroladores mais limitados (seja em frequências ou números reduzidos de registradores) inviabilizam utilizar um RTOS.

A memória não-volátil deve conseguir armazenar o *Kernel* do sistema RTOS, responsável pela divisão e escalonamento de tarefas, além dos códigos da aplicação desenvolvida. Para aplicações mais básicas, estima-se que seja necessário um espaço de 4 kB à 200 kB para armazenar o *Kernel* de RTOS simples (GAY, 2020). Finalmente, as memórias voláteis devem conseguir armazenar os dados referentes à execução de tarefas efetuadas pelo RTOS.

Há diferentes tipos de RTOS disponíveis no mercado, com vantagens e desvantagens diversas. Em geral, a popularidade e o custo do RTOS são os parâmetros mais observados durante a escolha do sistema que será embarcado no microcontrolador. Como exemplos de RTOS, pode-se citar os sistemas Azure RTOS, QNX, MicroC/OS-II e o FreeRTOS.

O sistema FreeRTOS, como o nome sugere, é de livre distribuição e está disponível para utilização no microcontrolador ESP32-WROOM-32. Trata-se de uma classe de sistemas RTOS leve o suficiente para ser executada nesse tipo de microcontrolador (MAIER; SHARP; VAGAPOV, 2017), sendo utilizado para o desenvolvimento de diversas aplicações.

Para utilizar o sistema FreeRTOS, é necessário atribuir prioridades para as tarefas executadas. O gerenciamento dessas tarefas se torna mais simples se os subprogramas utilizados puderem ser executados de forma independente. A partir do sistema FreeRTOS, o programador pode definir a prioridade de execução de cada *task*, assim como o esforço de cada núcleo disponível para processamento (GAY, 2020).

As tarefas são escalonadas conforme a definição das prioridades de execução. No sistema FreeRTOS, uma tarefa executada com uma determinada prioridade será adi-

ada quando outra tarefa com maior prioridade estiver pronta para ser executada. Nessa situação, a tarefa com menor prioridade será colocada em espera e a tarefa com maior prioridade terá preferencia para execução. A tarefa colocada em espera permanecerá pronta para ser executada, aguardando autorização para ser retomada (GAY, 2020).

O sistema FreeRTOS padrão oferece algumas implementações de *heap* para integradores de plataforma. Essa implementação é fornecida pelo fabricante em função dos diferentes tipos de memória que deverão ser gerenciados (GAY, 2020). Os tipos de memórias gerenciados na plataforma do ESP32-WROOM-32 são: DRAM (RAM de dados), IRAM (RAM de instrução) e D/IRAM, podendo ser usada tanto para dados quanto para instruções.

## 2.5 Ambientes de desenvolvimento

Para programar o microcontrolador ESP32-WROOM-32, a fabricante Espressif disponibiliza, de forma gratuita, um conjunto de bibliotecas denominadas ESP-IDF (do inglês, *Espressif IoT Development Framework*). Após a instalação dessas bibliotecas, pode-se desenvolver diversos tipos de aplicações a partir de diferentes ambientes de desenvolvimento.

Dentre os ambientes de desenvolvimento que podem ser utilizados para programar o ESP32, destacam-se a Espressif-IDE e a Arduíno-IDE, frequentemente utilizadas em tutoriais e exemplos de programação disponíveis em livros, *websites* e cursos de aperfeiçoamento.

A Espressif-IDE é uma solução desenvolvida pela empresa fabricante do microcontrolador, que disponibiliza os arquivos de instalação do *software* a partir do repositório GitHub. A linguagem de programação oficial utilizada pelo fabricante é C/C++, e o ambiente de desenvolvimento apresenta uma estrutura mais completa, com mais recursos para depuração e suporte às bibliotecas oficiais fornecidas pela Espressif através do GitHub.

Como alternativa, pode-se utilizar o ambiente de desenvolvimento Arduino-IDE. A grande vantagem de se utilizar essa IDE é a possibilidade de se trabalhar com uma grande quantidade de bibliotecas para controle de dispositivos periféricos externos que podem ser conectados ao microcontrolador (MAIER; SHARP; VAGAPOV, 2017). Trata-se de uma plataforma mais simples e intuitiva, mas também mais limitada quando comparada à Espressif-IDE.

Neste trabalho, optou-se por desenvolver toda a programação a partir da Arduino-IDE. Entretanto, devido a uma série de limitações observadas, sugere-se que novos trabalhos sejam desenvolvidos a partir da Espressif-IDE.

A partir da Arduino-IDE, pode-se desenvolver diversos tipos de aplicações utilizando-se como base as bibliotecas ESP-IDF fornecidas pela Espressif.

Ao se utilizar o sistema FreeRTOS, por padrão serão executas as funções *setup()* e *loop()*, que representam as funções básicas utilizadas para programação em ambiente Arduino-IDE. As tarefas de suporte são implementadas na PRO_CPU (chamado nesta IDE de CPU 0), enquanto as tarefas de aplicativos são executadas na APP_CPU (chamada nesta IDE de CPU 1). A divisão das tarefas é feita desta forma para garantir o funcionamento de alguns serviços de comunicação, como Wifi e Bluetooth, por exemplo. Entretanto, é possível criar tarefas em qualquer uma das CPUs, desde que seja feita uma alocação prévia.

A Espressif disponibiliza uma API (do inglês, *Application Programming Interface*) para gerenciamento de tarefas, que permite uma atribuição fixa em função do *core* (afinidades pré-definidas). Para estes casos, a função *xTaskCreatPinnedToCore* deve ser utilizada para a criação de tarefas com afinidade definida. A estrutura a seguir apresenta um exemplo de utilização desta função:

```
xTaskCreatePinnedToCore(
pvTaskCode,     // Funcao a ser executada, ponteiro para entrada de tarefa
pcName,         // Nome descritivo da tarefa
usStackDepth,   // Tamanho da pilha de tarefas especificados com o
                // numero de bytes
pvParameters,   // Ponteiro de parametro para a tarefa criada
uxPriority,     // A prioridade que a tarefa devera ser executada
pvCreatedTask,  // Usado para passar de volta um identificador pelo
                // qual a tarefa criada pode ser referenciada.
xCoreID,        // Nucleo de fixacao da tarefa
);
```

Esta função utiliza um parâmetro de atribuição 0 para PRO_CPU e 1 para APP_CPU. Entretanto, a função também permite que uma tarefa seja executada na unidade de processamento que estiver livre no momento da alocação, sendo necessário utilizar no campo xCoreID o comando de "tksNO_AFFINITY".

Cada bloco de controle de tarefa armazena o campo xCoreID, identificando o núcleo de processamento que deverá ser utilizado para executar a função. Quando chamado por um dos núcleos, o escalonador/agendador de tarefas seleciona a tarefa e avalia a permissão para a tarefa ser executada no núcleo que a solicitou. Se o valor no parâmetro de xCoreID for "tskNO_AFFINITY", a tarefa criada não será fixada em nenhuma CPU, e o escalonador poderá alocá-la no núcleo que estiver disponível (ESPRESSIF SYSTEMS, 2021).

A prioridade de execução é definida a partir do nível de urgência da tarefa. É possível utilizar 25 níveis de prioridades, de 0 a 24, sendo nível 0 o de prioridade mínima.

Por definição, as funções do *setup()* e *loop()*, definidas na Arduíno-IDE, são iniciadas com níveis de prioridade 1.

## 2.6 Técnica de focalização por abertura sintética

A técnica de focalização por abertura sintética (SAFT) surgiu na época da segunda guerra mundial, e tinha como objetivo gerar imagens para sistemas compostos por radares. Essa técnica foi adaptada para sistemas ultrassônicos, que utilizam transdutores acústicos para emitir/receber ondas mecânicas que se propagam pelo meio em análise.

Essas análises podem ser feitas utilizando um transdutor mono elemento ou com um transdutores com múltiplos elementos (*array*). Ao se utilizar um transdutor mono elemento, o mesmo elemento será responsável por emitir e receber os sinais acústicos, operando no modo pulso-eco. Nesse caso, será necessário deslocar o transdutor sobre a peça para ser possível efetuar uma varredura completa.

Quando se utiliza um *array* de transdutores, pode-se emitir um sinal com um transdutor e receber novamente o sinal usando o mesmo transdutor ou usando um dos outros transdutores que compõe o *array*. Essa flexibilidade permite que sejam adquiridas utilizando várias combinações de transdutores diferentes, que podem ser utilizadas para gerar uma imagem com maior resolução.

Para geração de imagens através do algoritmo SAFT, utiliza-se um *array* com $N$ transdutores espaçados por uma distância $\lambda/2$, sendo $\lambda$ o comprimento de onda do sinal acústico. Em seguida, um sistema de excitação envia um sinal para cada transdutor, excitando apenas um elemento por vez. Após a emissão, o sinal de excitação é retirado e o mesmo transdutor que emitiu será utilizado para receber o sinal acústico que se propagou pelo meio analisado.

Devido à atenuação do sinal acústico ao se propagar pelo meio, é comum utilizar um amplificador em recepção para ajustar um ganho no sinal recebido pelo transdutor antes de realizar a aquisição dos dados. Em geral, utiliza-se um multiplexador analógico para selecionar os transdutores em cada etapa. Dessa forma, pode-se trabalhar com um sistema de excitação/recepção de um canal, reduzindo a complexidade do projeto do *hardware*. Como o processo deve ser repetido para todos os elementos que compõe o *array*, pode-se perceber que em sistemas multiplexados, quanto maior o número de transdutores do sistema, maior será o tempo para adquirir todos os sinais. A Figura 11 apresenta uma configuração de *array* de transdutores de N elementos, operando em modo pulso-eco.

O conjunto de sinais elétricos adquiridos representa os sinais acústicos em função do tempo. Um parâmetro importante para esta técnica de análise é a impedância acústica, definida como:

$$Z = \rho \cdot c, \tag{2.1}$$

Figura 11 – *Array* com $N$ elementos transdutores.


Fonte: adaptado de Laorden et al. (2012).

sendo $\rho$ a densidade do material (kg/m$^3$), $c$ a velocidade de propagação do som no meio (m/s) e $Z$ a impedância acústica (Rayls).

Nos casos em que uma onda mecânica que se propaga por um meio 1, com um determinado valor de impedância acústica, encontra um meio 2 com um valor de impedância acústica diferente, uma parte da energia mecânica será transmitida do meio 1 para o meio 2, e a parcela resultante será refletida do meio 2 para o meio 1. A relação entre a parcela do sinal refletida e transmitida pode ser obtida a partir do cálculo do coeficiente de reflexão, definido como:

$$R_{12} = \frac{Z_2 - Z_1}{Z_2 + Z_1}, \tag{2.2}$$

sendo $Z_1$ e $Z_2$ as impedâncias acústicas dos meios 1 e 2, e $R_{12}$ o coeficiente de reflexão do meio 2 para o meio 1.

Quanto maior a diferença entre as impedâncias acústicas dos meios, maior será a parcela refletida do sinal. Sendo assim, em modo pulso-eco, o sinal recebido pelo transdutor será aquele refletido pelo meio com impedância acústica diferente do meio analisado. Nessa condição, os sinais recebidos irão se propagar duas vezes pelo meio ao longo do processo de emissão e recepção.

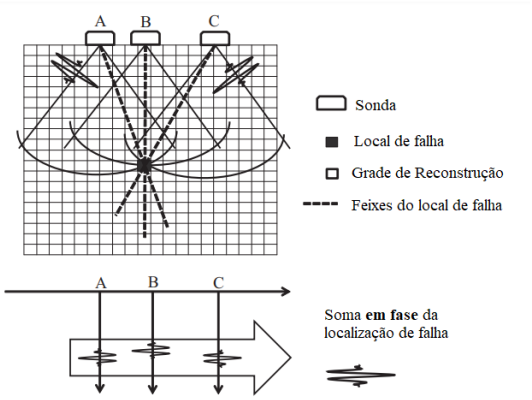
Considerando-se essas informações, e assumindo que a velocidade de propagação do som pelo meio em análise é conhecida, pode-se utilizar uma relação entre velocidade, distância e tempo para converter o tempo de propagação em distância.

A Figura 12 ilustra a representação de *array* de três elementos transdutores localizando um defeito posicionado no centro de uma peça. É possível notar que o defeito está localizado numa posição central da peça, e que o tempo de propagação dos sinais pelo meio muda conforme a posição dos transdutores.

Dessa forma, para ser possível gerar uma imagem capaz de identificar a posição do defeito, é necessário compensação da defasagem dos sinais utilizando uma técnica de focalização.

Definindo uma matriz $v(x, z)$, que representa as amplitudes (V) dos sinais recebidos

Figura 12 – *Array* de transdutores utilizados para identificação de um defeito num corpo de prova.




Fonte: adaptado de Guan, He e Rasselkorde (2015).

em função das distâncias $x$ e $z$ da peça analisada, pode-se compensar essas defasagens a partir da expressão

$$v(x,z) = \sum_{i=1}^{N} v_i(\tau_{ixz}), \tag{2.3}$$

sendo $N$ o número de elementos do *array*, $v_i$ os sinais adquiridos por cada elemento e $\tau_{ixz}$ o tempo de propagação do sinal pela amostra, definido como

$$\tau_{ixz} = 2\frac{|R|}{c}, \tag{2.4}$$

sendo $R$ a distância percorrida (m) e $c$ a velocidade de propagação (m/s).

Essa técnica de focalização é chamada de DAS (do inglês, *Delay-And-Sum*), e realiza a soma das amplitudes conforme a posição dos refletores e os elementos transdutores.

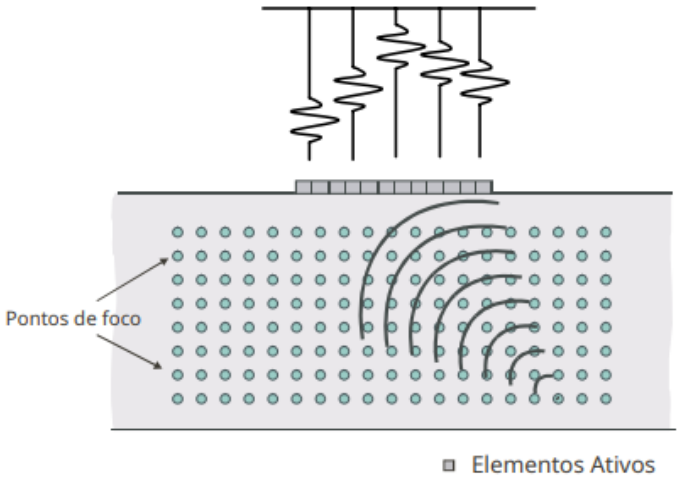
## 2.7 SAFT-TFM

O método de SAFT-TFM, também conhecido como *Full-Matrix* SA, foi desenvolvido para melhorar a qualidade das imagens obtidas a partir do método SAFT tradicional. Nessa configuração, a diferença é que os sinais emitidos por cada transdutor serão recebidos por todos os elementos do *array*, como pode ser visto na Figura 13.

Essa alteração aumenta o número de aquisição dos sinais recebidos, que passa de $N$ para $N^2$, aumentando também o tempo necessários para gerar as imagens. Nesses casos, a matriz de dados adquirida passa a ser chamada de matriz completa (HOLMES; DRINKWATER; WILCOX, 2005).

O processo de focalização nesses casos se torna um pouco mais complexo, pois será necessário calcular um número maior de defasagens antes somar as contribuições de cada sinal.

Figura 13 – *Array* de transdutores utilizados para caracterizar um determinado meio.




Fonte: adaptado de Holmes, Drinkwater e Wilcox (2005).

A nova expressão, que considera a combinação de transdutores emissores e receptores ($T$,$R$), além da posição dos transdutores em relação ao meio em análise.

Considerando-se um conjunto de sinais organizados como $matriz(T, data, R)$, sendo $matriz$ a representação de uma matriz de dados tridimensional, $T$ e $R$ os transdutores emissores e receptores, e $data$ o sinal adquirido a partir de cada combinação ($T$,$R$), representado no domínio do tempo, pode-se calcular a contribuição atribuída a cada ponto do *grid* ($x$,$z$) a partir da expressão

$$v(x,z) = \sum matriz\left(T, \frac{\sqrt{(tr_T - x)^2 + z^2} + \sqrt{(tr_R - x)^2 + z^2}}{c}, R\right), \tag{2.5}$$

sendo $tr_T$ e $tr_R$ as posições dos transdutores emissor e receptor em relação ao objeto que se deseja inspecionar e $c$ a velocidade de propagação do som pelo meio em análise.

Laorden et al. (2012) realizaram testes com um *array* de 128 elementos, gerando uma matriz completa com $128^2$ sinais. Os dados foram processados a partir de Unidade de Processamento Gráfico de Propósito Geral (do inglês, *General Purpose Graphics Processing Unit*) (GPGPU), e o foco do trabalho envolveu justamente a análise de desempenho desse tipo de algoritmo quando executado concorrentemente a partir dos diversos núcleos de processamento presentes numa placa gráfica. O algoritmo utilizado pelos autores para gerar as imagens foi organizado da seguinte forma:

1. Após a aquisição da matriz de sinais, aplica-se um filtro passa-faixas para minimizar a influência de ruído e remover o *bias* introduzido na etapa de aquisição de dados.

2. Realizam-se os cálculos dos tempos de atraso, que serão utilizados para somar as contribuições dos sinais adquiridos por cada transdutor, para cada ponto dos eixos $x$ e $z$ definidos para representar a imagem.

3. Na terceira etapa, realiza-se o somatório das contribuições de cada sinal, e utiliza-se a Transformada de Hilbert para detectar a envoltória do sinal resultante.

4. Na etapa final, apresenta-se a imagem gerada, usando funções gráficas.

Apesar de apresentar um custo computacional mais elevado, essa técnica apresenta uma melhor relação sinal/ruído quando comparada ao método SAFT tradicional, possibilitando a geração de imagens acústicas com melhor resolução.

## 2.8 Considerações parciais

Este capítulo apresentou uma revisão teórica acerca das características dos projetos de *hardware* e *software* do microcontrolador ESP32-WROOM-32. O estudo mostrou que a grande limitação deste sistema está relacionada à impossibilidade de se realizar acessos simultâneos à memória e periféricos pelos dois núcleos. Entretanto, verificou-se que essas limitações podem ser minimizadas dividindo adequadamente o algoritmo em tarefas, cuja execução passa a ser controlada pelo sistema FreeRTOS embarcado no microcontrolador. O maior desafio deste processo é paralelizar adequadamente o algoritmo que será executado, dividindo eficientemente as tarefas de modo a evitar filas e tempos de espera nos dois núcleos.

# 3 Metodologia

Este Capítulo apresenta a metodologia utilizada para avaliar o desempenho do microcontrolador ESP32-WROOM-32 ao executar tarefas simultâneas nos dois núcleos de processamento. Para avaliar a capacidade de processamento do microcontrolador, foram propostos ensaios utilizando um algoritmo de focalização por abertura sintética (SAFT), utilizado para geração de imagens ultrassônicas.

Foram realizas comparações entre os tempos necessários para executar o algoritmo a partir de três abordagens distintas: implementação sequencial, usando um único núcleo de processamento; implementação concorrente, usando dois núcleos de processamento responsáveis por executar tarefas pré-fixadas; e implementação concorrente, usando dois núcleos de processamento responsáveis por executar tarefas de forma alocadas de forma dinâmica por meio de um sistema operacional de tempo real.

Como já mencionado, o objetivo deste trabalho não é desenvolver um sistema capaz de processar os sinais num tempo menor do que o sistema apresentado por Laorden et al. (2011), mas sim avaliar o desempenho do microcontrolador utilizado para executar algoritmos complexos, com custo computacional elevado. As próximas seções apresentam o procedimento utilizado para a realização dos ensaios e caracterização da plataforma de processamento.

## 3.1 Dados utilizados para a realização dos ensaios

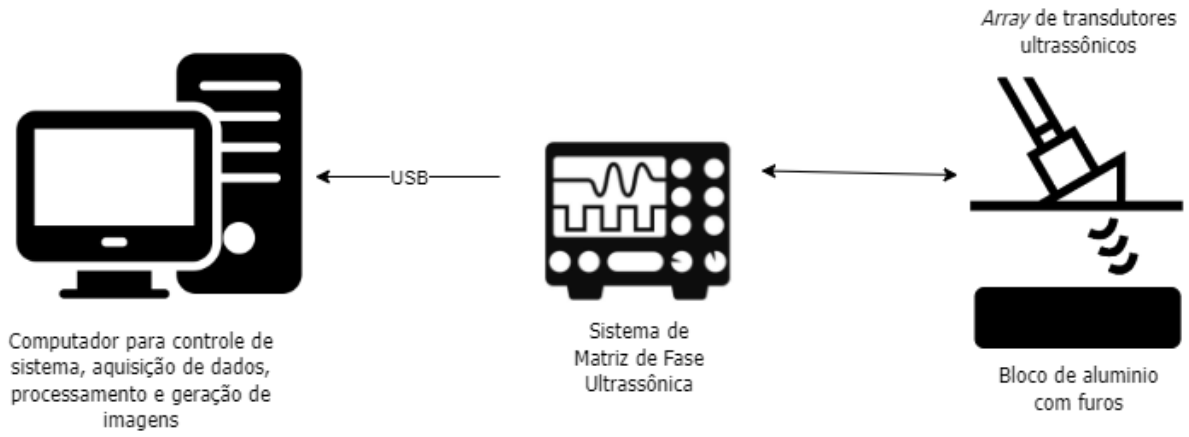
Para a realização dos ensaios propostos neste trabalho, foram utilizados sinais adquiridos por pesquisadores do *Instituto de Tecnologias Físicas y de la Información* Leonardo Torres Quevedo. Durante o experimento, foi realizada a inspeção de um bloco de alumínio (com dimensões de 100 mm $\times$ 100 mm $\times$ 100 mm), que apresenta uma série de furos de 0,5 mm de diâmetro.

A inspeção do bloco foi realizada utilizando um transdutor com 128 elementos, igualmente espaçados a uma distância de 0,6 mm de centro a centro. Um diagrama esquemático com o circuito de excitação e aquisição utilizado pode ser visto na Figura 14, e uma imagem do bloco de alumínio utilizado ao longo dos ensaios pode ser vista na Figura 15.

Os transdutores foram excitados com sinais pulsados, a uma frequência central de 5 MHz, e a aquisição dos sinais foi realizada a partir de uma placa de aquisição, com frequência de amostragem de 40 MHz. A Tabela 3 apresenta os dados utilizado para gerar as imagens acústicas por meio do algoritmo SAFT.

Para a realização do ensaio, define-se inicialmente um elemento como transdutor emissor. Na sequência, aplica-se um sinal de tensão pulsado nesse elemento, e adquirem-se

Figura 14 – Diagrama esquemático do circuito eletrônico utilizado ao longo do processo de aquisição dos sinais ultrassônicos utilizados neste trabalho.




Fonte: do autor.

Figura 15 – Peça de alumínio analisada por meio de ensaios não-destrutivos por ultrassom.

Fonte: retirado de Laorden et al. (2011).

os sinais recebidos por todos os transdutores do *array*, incluindo o transdutor utilizado para emitir o sinal. Esse procedimento deve ser repetido $N$ vezes, variando-se o elemento utilizado como emissor até que todos os elementos tenham sido utilizados.

No experimento realizado, o transdutor foi posicionado na parte superior do bloco, num ponto específico $y$. Dessa forma, pode-se gerar uma imagem acústica capaz de representar um corte transversal (dimensões $x$ e $z$) do bloco analisado.

Tabela 3 – Dados do ensaio com *array* de transdutores.

| Parâmetros | Valor | Unidade |
|---|---|---|
| Velocidade de Propagação | 6300 | m/s |
| Número de Elementos | 128 | - |
| Distancia entre Elementos | 0,6 | mm |
| Distância máxima para visualização | 95 | mm |
| Distância Mínima para visualização | 5 | mm |
| Frequência de excitação transdutores | 5 | MHz |
| Frequência de Amostragem | 40 | MHz |

Fonte: do autor.

Ao final, considerando-se que o *array* possui 128 elementos, foram adquiridos $128^2$ sinais. Para facilitar a manipulação desses dados, os sinais foram organizados na forma de uma matriz tridimensional, definida como $matriz(T, data, R)$, sendo $T$ e $R$ os transdutores emissores e receptores, e *data* os sinais adquiridos pelo sistema de aquisição representado no domínio do tempo. Ressalta-se que os sinais adquiridos foram filtrados, a fim de se reduzir os efeitos do ruído elétrico gerado ao longo do processo de aquisição.

## 3.2 Testes iniciais usando computador pessoal

Testes iniciais foram realizados a partir de um computador pessoal, onde foram implementados diferentes versões de algoritmos usando o *software Matrix Laboratory* (MATLAB). Nessa etapa, foram avaliados o tempo de execução de cada implementação e o espaço de memória necessário para armazenar a matriz completa de dados na memória RAM do computador.

Além disso, o MATLAB também foi usado para a realização de testes relacionados à otimização do algoritmo, buscando reduzir os tempos de execução do mesmo quando implementado sequencialmente, considerando um único núcleo de processamento.

O Algoritmo 1 apresenta um pseudocódigo com a representação de parte do código implementado no MATLAB.

Para a utilização do algoritmo, deve-se definir os valores iniciais e finais de $x$ e $z$ da imagem que se planeja gerar. Além disso, é necessário definir a resolução (em mm) da imagem que será gerada. De maneira análoga, deve-se definir os índices inicial e final dos transdutores que se planeja utilizar. A partir dessa definição, pode-se limitar o número de transmissores ou receptores, por exemplo. Esse processo pode ser usado para gerar imagens em menor tempo, mas destaca-se que a imagem gerada apresentará menor qualidade.

A posição distância entre os elementos que compõe o transdutor é utilizada para definir o vetor *ptr*, que representa a posição dos elementos em relação ao bloco inspecionado. Neste trabalho, considerou-se um transdutor localizado na posição central do

**Algoritmo 1** Pseudocódigo do algoritmo de focalização por abertura sintética (SAFT).

**for** ($z1 = z_{inicial}$; $z1 \leq z_{final}$; $z1 = z1 + resolucao$) **do**
  **for** ($x1 = x_{inicial}$; $x1 \leq x_{final}$; $x1 = x1 + resolucao$) **do**
    $aux = 0$;
    **for** ($R = R_{inicial}$; $R \leq R_{final}$; $R++$) **do**
      $T_r = \sqrt{(x(x1) - ptr(R))^2 + z^2}$
      **for** ($T = T_{inicial}$; $T \leq T_{final}$; $T++$) **do**
        $T_e = T_r + \sqrt{(x(x1) - ptr(T))^2 + z^2}$
        $k = round(T_e/resolucao)$
        $idx = k - idx_0 - 1$
        **if** ($idx \leq dimensao(data)$) **then**
          $aux = aux + matriz(R,idx,E)$
        **end if**
      **end for**
    **end for**
    $v(x,z) = aux$
  **end for**
**end for**

bloco. Com base nos pontos ($x$,$z$) definidos inicialmente, e na combinação ($T$,$R$) ajustada, calcula-se a distância propagada pela onda mecânica no bloco. Dividindo-se esse valor pela resolução ajustada, encontra-se um índice ($idx$), que representa um ponto do vetor *data*.

Repetindo-se essa análise para outras combinações de ($T$,$R$), encontram-se índices ($idx$) diferentes, dependendo das posições dos transdutores utilizados. Para gerar uma imagem acústica a partir desses sinais, utiliza-se uma variável auxiliar ($aux$) para acumular as contribuições obtidas a partir de cada combinação ($T$,$R$) utilizada.

Se o valor de índice calculado for superior ao número máximo de pontos do vetor *data*, deve-se descartar esse resultado. Nesse caso, o resultado indica que a combinação ($T$,$R$) escolhida não apresenta contribuição significativa para o ponto ($x$,$z$) analisado.

Ao longo desses ensaios, observou-se que a resolução da imagem (em mm), utilizada para controlar as variações das dimensões $x$ e $z$, influencia diretamente o tempo de processamento necessário para se obter o resultado.

Esse fato deve-se a um aumento da quantidade de cálculos que deverão ser realizados, conforme a equação (2.3), o tamanho da matriz resultante $v(x,z)$ depende do número de pontos definidos para os vetores $x$ e $z$. Além disso, cada ponto ($x$,$z$), será utilizado como referência para o cálculo da distância até um transdutor receptor. Essa distância será calculada a partir da raiz quadrada do somatório das distâncias entre ($x$,$z$) e ($tr(T)$,$tr(R)$) ao quadrado, representando um elevado custo computacional. Ressalta-se, entretanto, que a resolução lateral da imagem está diretamente relacionada à quantidade de pontos ($x$,$z$) considerados.

A relação entre resolução e tempo de processamento destaca a importância de

se considerar a resolução adequada ao realizar as análises, uma vez que um aumento na resolução pode representar um significativo aumento no tempo de processamento, inviabilizando ou aumentando excessivamente os custos de aplicações que devem ser executadas em tempo real.

## 3.3 Preparação dos dados para utilização no sistema embarcado

Após os testes iniciais realizados em MATLAB, foram iniciados os ensaios usando o sistema micro controlado. Para isso, foram inicialmente analisadas as especificações do microcontrolador, a fim de se definir a capacidade de armazenamento de memória (volátil e não-volátil) disponível.

O ESP32-WROOM-32 possui uma capacidade de armazenamento de memória limitada, fato que limita a quantidade de dados que podem ser manipulados e processados durante a realização dos ensaios. As restrições de espaço de armazenamento de memória é um fator limitante no que diz repeito ao aumento da resolução lateral da imagem a ser processada.

Segundo o fabricante, o microcontrolador possui 520 kB de memória SRAM, com apenas 320 kB utilizável. O fabricante ressalta que, devido a limitações técnicas, apenas 160 kB de memória SRAM podem ser utilizados para o desenvolvimento das aplicações. Possuindo ainda uma memória não-volátil (*flash*), com capacidade de armazenamento de 4 MB.

A matriz de dados utilizada ao longo do ensaio possui dimensão (128,1137,128), e todos os pontos foram armazenadas em formato *double precision*, ou seja, são necessários 64 bits (8 *bytes*) de espaço para armazenamento de cada elemento da matriz.

Com base nessas informações, o espaço necessário para armazenar a matriz completa de dados (em MB) pode ser calculado como:

$$\text{Espaço} = 128 * 1137 * 128 * 8/1024/1024 = 142{,}13 \text{ MB}. \tag{3.1}$$

Observa-se que o espaço necessário para armazenar a matriz completa é muito superior aos limites de armazenamento para o microcontrolador utilizado. Dessa forma, foi necessário recortar parte dos dados, utilizando apenas um grupo limitado de informações.

Para reduzir o volume de dados, optou-se por utilizar somente um transdutor emissor ($T = 64$) e 32 elementos receptores ($R = 64$ até $R = 96$). Além disso, a área de inspeção foi reduzida ($x = [-10, 10]$ mm e $z = [30{,}50]$ mm) e as variáveis com precisão *double* foram substituídas por variáveis com precisão *float*, que utilizam 32 bits (4 *bytes*) de espaço para armazenar cada elemento da matriz.

Após esses ajustes, obteve-se uma nova matriz de dados com dimensão (1,311,32), cujo espaço necessário para armazenamento (em kB) pode ser definido por:

$$\text{Espaço} = 311 * 32 * 4/1024 = 38{,}88 \text{ kB}. \tag{3.2}$$

A matriz simplificada resultante apresenta dimensões compatíveis com o espaço de memória disponível no microcontrolador utilizado. Destaca-se, entretanto, que a resolução lateral das imagens apresentadas na sequência apresentam limitações causadas pela redução do número de transdutores emissores e receptores. As limitações na capacidade de armazenamento de memória do microcontrolador tornam o processo de implementação de algoritmos mais complexos uma tarefa desafiadora.

Para os dados serem transferidos do microcomputador para o microcontrolador, deve-se utilizar algum meio de comunicação. Como mencionado, o ESP32-WROOM-32 possui diversas interfaces de comunicação diferentes, sendo elas físicas, como USART, SPI, $I_2C$, entre outras, ou sem fio, como Wi-Fi e Bluetooth, por exemplo.

Neste trabalho, foram inicialmente realizadas transferências de dados por meio de comunicação serial assíncrona. Esse processo, apesar de funcionar corretamente, demonstrou-se pouco eficiente, uma vez que os tempos necessários para que todo o pacote de dados fosse transferido eram muito elevados. Outra desvantagem observada ao se utilizar esse método é que todos os dados transferidos foram armazenados na memória SRAM, reduzindo significativamente a quantidade de espaço disponível para o armazenamento de dados auxiliares, necessário para a execução do algoritmo proposto.

Uma opção para se contornar esse tipo de problema seria transferir somente os dados necessários para a realização dos cálculos realizados em cada iteração do algoritmo. A desvantagem dessa abordagem é que a comunicação serial entre microcomputador e microcontrolador adicionaria um atraso de tempo considerável ao processo, afetando dessa forma a determinação correta dos tempos necessários para o processador executar as tarefas especificadas.

A fim de se minimizar esse tipo de problema, optou-se por carregar todos os pontos da matriz de dados simplificada a partir de um arquivo *header* (.h), compilado juntamente com o algoritmo que se deseja avaliar. Nesse caso, o compilador passa a armazenar a matriz simplificada na memória *flash* do microcontrolador, evitando que o espaço limitado de memória SRAM disponível seja ocupado por essas informações.

Ressalta-se que essa abordagem não poderia ser utilizada em aplicações de tempo real, uma vez que os dados deveriam ser transferidos imediatamente para os núcleos de processamento. Nesses casos, a utilização de métodos de transferência de dados mais rápidos do que a transferência de dados serial (como comunicação USB, por exemplo) seriam mais efetivos.

Para a realização dos testes, foram consideradas três resoluções (mm), gerando imagens com qualidades diferentes, definidas neste trabalho como baixa, média e alta. Com base nos valores de resolução escolhidos, foram calculados os espaços necessários para armazenar os dados da matriz na memória do microcontrolador. A Tabela 4 apresenta

uma comparação entre os espaços necessários para armazenamento de dados considerando os três casos.

Tabela 4 – Definição do espaço necessário para armazenamento do algoritmo e dos dados necessários para gerar imagens com diferentes resoluções a partir do sistema microcontrolado.

| Qualidade | Resolução (mm) | Matriz (Nx × Nz) | Espaço (kB) |
|---|---|---|---|
| Baixa | 0,392 | 51 × 51 | 10,16 |
| Média | 0,277 | 72 × 72 | 20,25 |
| Alta | 0,178 | 112 × 112 | 49,0 |

Fonte: do autor.

É importante mencionar que a imagem de alta qualidade ocupa praticamente a capacidade máxima de armazenamento do microcontrolador. Dessa forma, aumentar essa resolução pode resultar em uma sobrecarga da memória disponível (*overflow*).

Nos testes realizados, o processador parou de funcionar nos casos onde o *overflow* ocorreu. Nesse instante, o circuito de proteção *watchdog* reiniciou de maneira forçada o dispositivo, e uma mensagem de erro foi enviada por meio da Arduino-IDE.

Ao considerar somente um elemento emissor, grande parte das informações disponíveis na matriz de dados foram descartadas. Consequentemente, a resolução lateral das imagens resultantes foi comprometida, uma vez que um número menor de sinais foi utilizado para gerar as imagens.

Essa redução no número de pontos de dados (*pixels*) para adequação das dimensões da matriz de dados tem implicações diretas na precisão da localização dos defeitos na imagem final, como será descrito no próximo Capítulo. A estratégia adotada resultou em uma diminuição na densidade de informações adquiridas, prejudicando a capacidade de distinguir detalhes na imagem final.

## 3.4 Programação do sistema embarcado

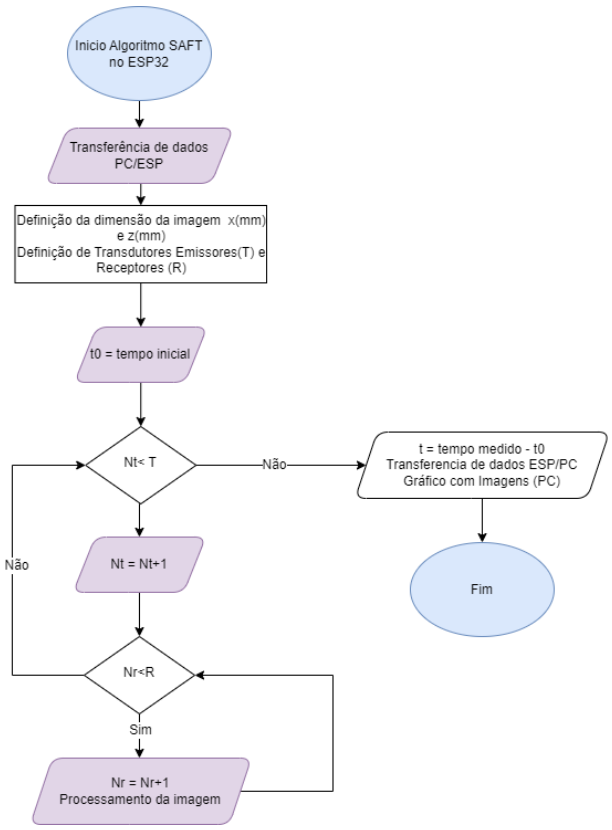
O algoritmo implementado no microcontrolador ESP32-WROOM-32 considerou a matriz de dados reduzida descrita na seção anterior. Ao todo, foram avaliados os desempenhos de três versões diferentes de algoritmos para o processamento das imagens. Os algoritmos utilizados podem ser acessados a partir do repositório do Github (FARIA, 2023).

Na primeira abordagem, avaliou-se o desempenho de um algoritmo sem paralelização, executado sequencialmente, por um único núcleo de processamento. Os resultados obtidos a partir dessa abordagem foram utilizados como referência para comparação do desempenho apresentado pelo sistema ao executar versões do algoritmo nos dois núcleos de processamento disponíveis no microcontrolador.

As comparações realizadas consideraram somente o tempo necessário para processar a matriz completa de dados. Ressalta-se que não foram considerados os tempos de aquisição e transferência de dados para o microcontrolador.

A Figura 16 apresenta o fluxograma do algoritmo utilizado para processar a matriz de dados sequencialmente, considerando somente um núcleo de processamento.

Figura 16 – Fluxograma do algoritmo utilizado para processar os dados sequencialmente, considerando apenas um núcleo de processamento.

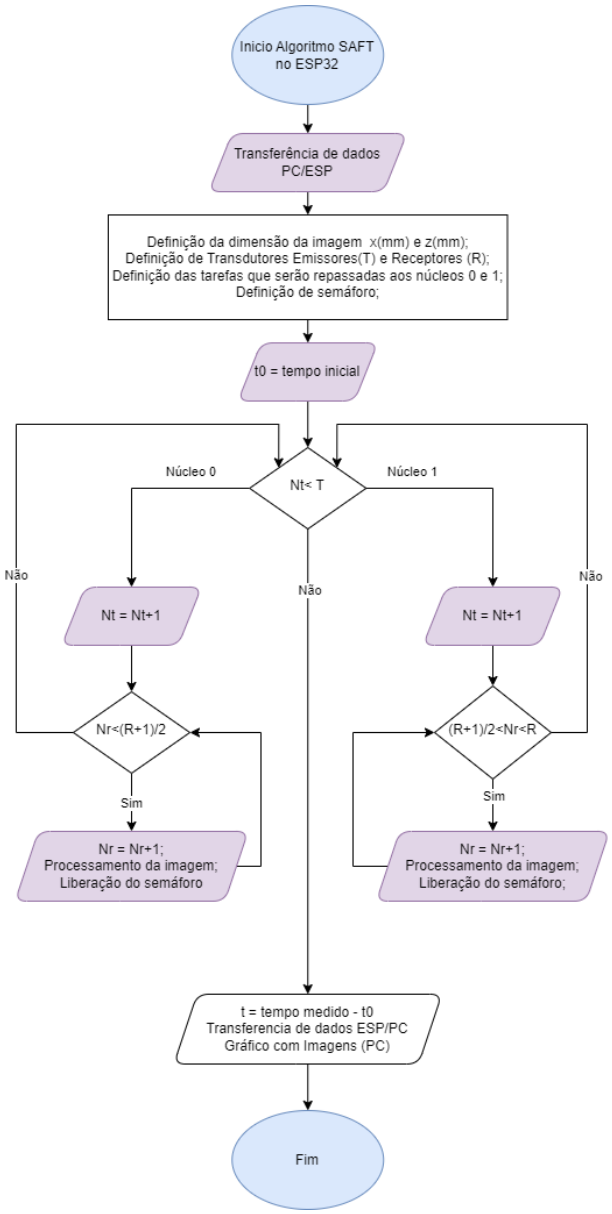



Fonte: do autor.

Em seguida, implementou-se uma versão do algoritmo concorrentemente, paralelizando as tarefas associadas aos transdutores receptores. Nesse modelo de paralelização, tarefas específicas foram atribuídas a cada um dos núcleos de processamento disponíveis no ESP32-WROOM-32, permitindo que os dois núcleos possam processar partes da imagem simultaneamente.

Como mencionado anteriormente, os núcleos de processamento não conseguem realizar acessos concorrentes às memórias SRAM e *flash*. Dessa forma, para garantir a integridade dos dados compartilhados entre os núcleos, foi necessário utilizar semáforos, gerenciados pelo sistema RTOS. Esses semáforos atuam como recursos de controle ao acesso a regiões críticas do algoritmo, onde dados compartilhados são manipulados por diferentes núcleos do processador. A Figura 17 apresenta um fluxograma que representa a maneira como esse algoritmo foi implementado no microcontrolador.

Figura 17 – Fluxograma do algoritmo utilizado para processar os dados paralelamente, com tarefas pré-fixadas para cada núcleo de processamento.



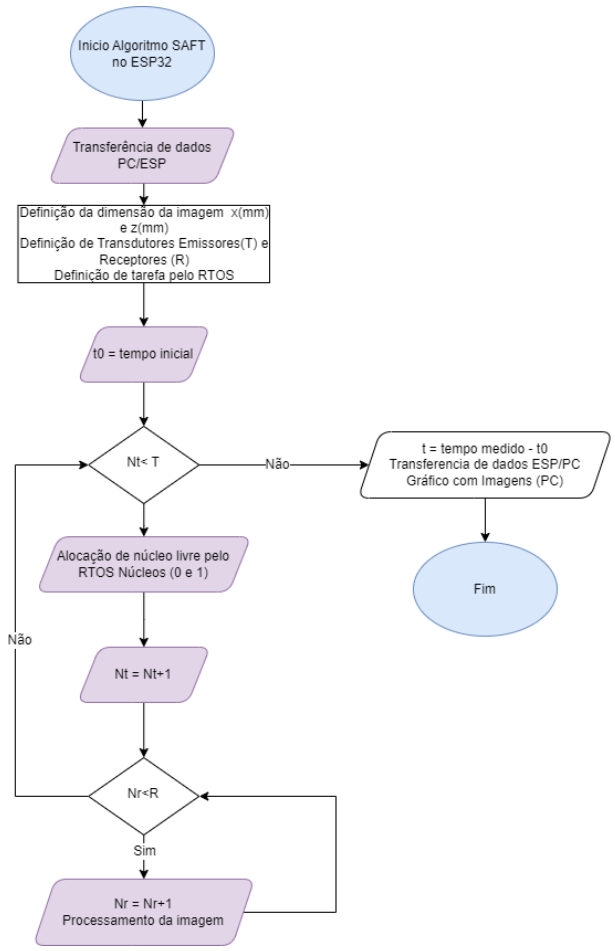

Fonte: do autor.

No início do programa, um semáforo foi criado para gerenciar o acesso aos dados compartilhados. Antes de um dos núcleos acessar uma região crítica do algoritmo ou um conjunto de variáveis compartilhadas, é necessário ocorrer uma liberação do semáforo. Essa liberação indica que o segundo núcleo não está acessando os recursos compartilhados, e permite que o primeiro núcleo possa prosseguir com o acesso a essas informações.

Após a conclusão das operações na região crítica, o núcleo libera o semáforo, permitindo que outros núcleos acessem os recursos compartilhados. Nota-se, portanto, que o semáforo é essencial para evitar que múltiplos núcleos modifiquem ou acessem dados compartilhados simultaneamente, mitigando possíveis danos ao funcionamento do sistema.

O último ensaio realizado consistiu na implementação de um algoritmo concorrente, capaz de alocar tarefas nos dois núcleos de processamento de forma dinâmica. Nessa configuração, a definição do núcleo responsável pelo processamento das tarefas passa a ser controlada pelo sistema RTOS. A Figura 18 apresenta o fluxograma do algoritmo implementado nesta condição.

Figura 18 – Fluxograma do algoritmo utilizado para processar os dados paralelamente, com alocação dinâmica das tarefas.




Fonte: do autor.

Nessa configuração, a paralelização do algoritmo foi realizada de forma dinâmica, permitindo uma distribuição flexível e otimizada das tarefas de processamento entre os núcleos do microcontrolador. Dessa forma, caso um dos núcleos termine a execução de uma determinada tarefa antes do outro, uma nova tarefa lhe será atribuída, evitando que o processador permaneça ocioso a espera da definição de novas atividades.

Após o processamento dos dados a partir das três estratégias mencionadas anteriormente, transferiu-se a matriz de dados resultante $v(x,z)$ para um computador pessoal por meio de comunicação serial assíncrona. Nesse processo, utilizou-se o *software* Putty para adquirir e armazenar os dados recebidos usando um arquivo de texto. O *software* MATLAB foi usado para carregar os dados armazenados nos arquivos de texto e plotar

as imagens resultantes.

Para determinação dos tempos de execução dos algoritmos, dois pinos de saída digital do micrococontrolador (GPIO's) foram conectados a um osciloscópio digital. Cada pino foi associado a um dos núcleos de processamento disponíveis no sistema embarcado.

Antes do início de qualquer tarefa de processamento, o nível lógico dos pinos foi definido como baixo, e um *delay* de inicialização de 1 s foi aplicado. Na sequência, antes da primeira linha a ser executada, define-se o nível lógico do pino associado ao núcleo em uso como alto. O pino em questão deve ser resetado após a execução da última linha de código do algoritmo.

O osciloscópio digital foi utilizado para medir os tempos entre as bordas de subida e descida geradas ao longo desse processo. Os tempos de subida e descida, definidos no *datasheet* do microcontrolador, foram subtraídos dos tempos medidos usando o osciloscópio. O procedimento foi repetido por 5 vezes, a fim de se estimar as incertezas experimentais associadas ao processo.

## 3.5 Considerações parciais

Neste Capítulo foram descritas as etapas de desenvolvimento utilizadas ao longo da execução deste trabalho. Inicialmente, foram descritos os sistemas de excitação e aquisição empregados durante os ensaios acústicos realizados para caracterizar, de forma não destrutiva, um bloco metálico com defeitos conhecidos.

Na sequência, foram apresentados a estrutura utilizada para organizar os dados adquiridos, bem como os algoritmos de processamento implementados no sistema embarcado. Foram descritas as etapas de preparação dos dados, indicando os critérios aplicados na seleção dos sinais enviados para a plataforma de processamento embarcado e o processo de transferência dessas informações para a memória não-volátil do microcontrolador.

Foram apresentadas três abordagens distintas para implementação dos algoritmos de focalização por abertura sintética. Na primeira, uma versão mais simples, as tarefas foram executadas sequencialmente, e um único núcleo de processamento foi utilizado. Na segunda, as tarefas foram executadas paralelamente por núcleos de processamento previamente definidos. Na terceira, as tarefas também foram executadas paralelamente, mas um sistema RTOS foi configurado para definir os núcleos de processamento responsáveis pela execução de cada tarefa.

Finalmente, um osciloscópio digital foi utilizado para medir os tempos de execução dos algoritmos implementados a partir das três abordagens mencionadas anteriormente, usando como referência sinais elétricos gerados por saídas digitais setadas e zeradas antes e após o início das atividades de processamento executadas pelos núcleos.

# 4 Resultados

Este Capítulo apresenta os principais resultados obtidos neste trabalho. Foram realizados ensaios variando-se as estratégias de implementação dos algoritmos e a resolução das imagens geradas, comparando-se o desempenho de cada implementação usando como referência os tempos de processamento para cada caso.

Os dados utilizados ao longo dos ensaios foram armazenados na memória *Flash* do microcontrolador durante a etapa de programação do mesmo. Os tempos necessários para processar as imagens foram medidos com o auxílio de um osciloscópio digital.

Para avaliar a qualidade das imagens geradas, a matriz de dados resultante foi transferida para um computador pessoal por meio de comunicação serial assíncrona. O *software* MATLAB foi utilizado para plotar todas as imagens apresentadas nas próximas seções.

## 4.1 Limitações do ambiente de desenvolvimento

Durante os testes realizados, foram observados diversos problemas e limitações associados ao uso da Arduino-IDE. O principal ponto a ser considerado diz respeito as bibliotecas desenvolvidas pela empresa Espressif, responsável pela fabricação do microcontrolador. Ao longo do desenvolvimento do trabalho, observou-se que nem todas as bibliotecas Espressif-IDF documentadas no site do fabricante estavam disponíveis para utilização na Arduino-IDE. Além disso, algumas atualizações disponíveis para bibliotecas Espressif-IDF, que poderiam ser utilizadas a partir da Espressif-IDE, ainda não haviam sido disponibilizadas para utilização a partir da Arduino-IDE.

Esses fatores podem ter influenciado os resultados que serão apresentados na sequência, uma vez que bibliotecas otimizadas para processamento digital de sinais poderiam minimizar os tempos associados a parte dos cálculos mais complexos, tais como os cálculos de raiz quadrada, por exemplo. Além disso, a configuração de tarefas e a alocação de núcleos para a paralelização foram mais restritivas na Arduino-IDE, afetando o desempenho das estratégias de paralelização com tarefas fixas utilizada neste trabalho.

A problemática apresentada pela limitação de ambiente Arduino-IDE poderia ser contornada com a utilização da plataforma de desenvolvimento Espressif-IDE, desenvolvida pela própria fabricante do microcontrolador. Uma vez que este trabalho já estava sendo desenvolvido no ambiente Arduino-IDE, essa alternativa acabou por não ser utilizada. Entretanto, ressalta-se que para trabalhos futuros, essa opção deve ser considerada.

## 4.2 Impacto da precisão das variáveis utilizadas para armazenamento de dados na imagem final

Como já mencionado, a variação da resolução em algoritmos de formação de imagens por ultrassom, como o SAFT, desempenha uma influência significativa na complexidade computacional e no número de operações executadas durante o processamento. Essa variação afeta diretamente a quantidade de pontos na imagem resultante e influência significativamente o desempenho do algoritmo.

Entretanto, deve-se considerar também a precisão das variáveis utilizadas para representar os dados adquiridos pelo sistema de aquisição. Os sinais da matriz de dados completa utilizada neste trabalho foram adquiridos utilizando-se variáveis do tipo *double*, contudo, para reduzir o espaço de armazenamento necessário, e adequar a precisão dos sinais à dimensão da ULA que compõe os processadores Xtensa, optou-se por reduzir a precisão das variáveis para 32 bits, convertendo a matriz *double* para uma nova matriz *float*.

Foram comparadas imagens obtidas em dois processadores distintos: um processador de alto desempenho, com ULA de 64 bits, e um processador Xtensa *dual-core* de 32 bits integrado ao microcontrolador. O objetivo dessa comparação é avaliar os efeitos das diferenças de precisão dos cálculos em ULA de 32 bits e ULA 64 bits.

Como mencionado, o objetivo desse tipo de análise por imagens é identificar defeitos sem a necessidade de danificar o meio de interesse. Nesse estudo, analisou-se um bloco de alumínio com um conjunto de furos, como representado pela Figura 19. A fim de reduzir o volume de dados que deveriam ser processados, limitou-se a área de interesse a um quadrado de 20 mm $\times$ 20 mm, como pode ser visto na imagem.

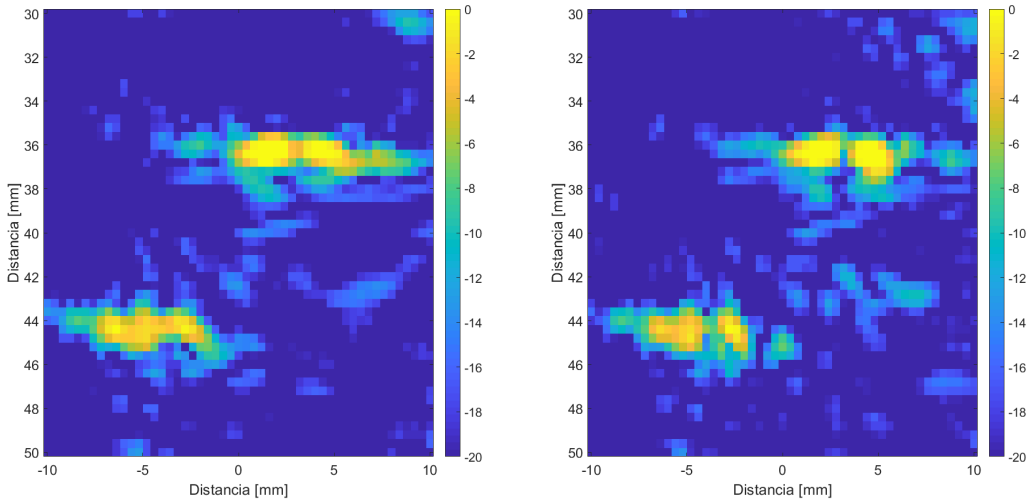
Figura 19 – Área de inspeção da peça de alumínio analisada a partir do ensaio não-destrutivo por ultrassom.


Fonte: retirado de Laorden et al. (2011).

A comparação considerou imagens com qualidade baixa (2601 *pixels*), média (5184 *pixels*) e alta (12544 *pixels*), representadas lado a lado nas figuras apresentadas na sequência. Para a comparação, foram consideras tanto a redução do número de *pixels* quanto a redução da precisão das variáveis utilizadas.

A Figura 20 apresenta o resultado dessa comparação, considerando-se imagens de baixa qualidade.

Figura 20 – Comparação entre os resultados obtidos ao processar imagens de baixa qualidade em processadores de 32 bits e 64 bits.




(a) Microcontrolador 32 bits. (b) Microprocessador 64 bits.

Fonte: do autor.

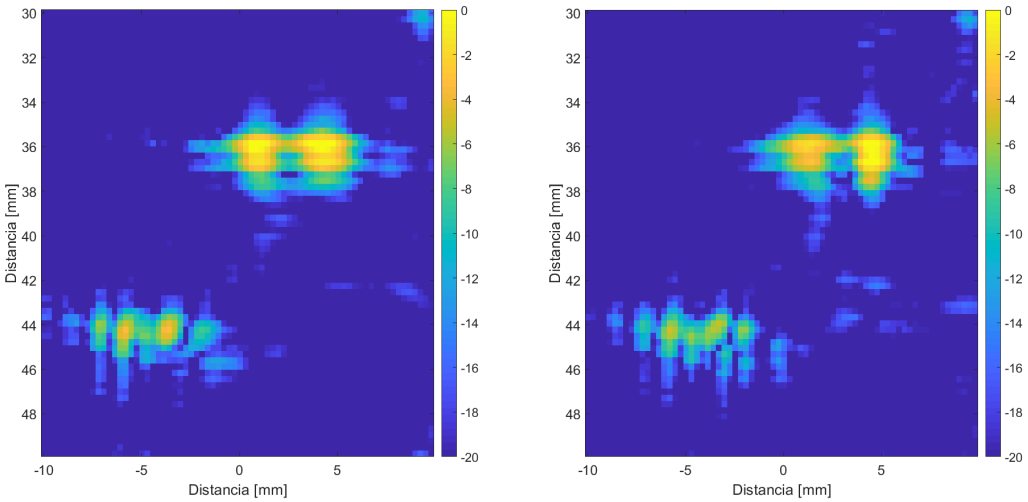
Por apresentar um número menor de pontos, a imagem de menor qualidade exige um número menor de operações, permitindo assim uma maior taxa de quadros por segundo, considerando uma aplicação em tempo real. Entretanto, como pode ser observado, a qualidade da imagem é comprometida, com falhas que dificultam a visualização dos furos da peça de alumínio.

Aumentando-se a qualidade para um nível médio, a quantidade de pontos na imagem será significativamente maior do que na resolução baixa. Isso resultará em um número maior de operações durante o processamento, pois cada ponto na imagem exigirá cálculos adicionais no algoritmo SAFT.

A Figura 21 apresenta as imagens geradas nessa condição. Nota-se que há uma pequena diferença entre as duas imagens, possivelmente causada pela redução da precisão dos dados. Esse efeito é menos perceptível do que o observado no caso anterior, pois a resolução da imagem exibida na figura é maior, garantindo dessa forma uma imagem com maior qualidade.

A Figura 22 apresenta uma comparação entre os resultados obtidos a partir do microcontrolador e do microprocessador considerando o caso de maior qualidade. Nessa

Figura 21 – Comparação entre os resultados obtidos ao processar imagens de qualidade intermediária em processadores de 32 bits e 64 bits.

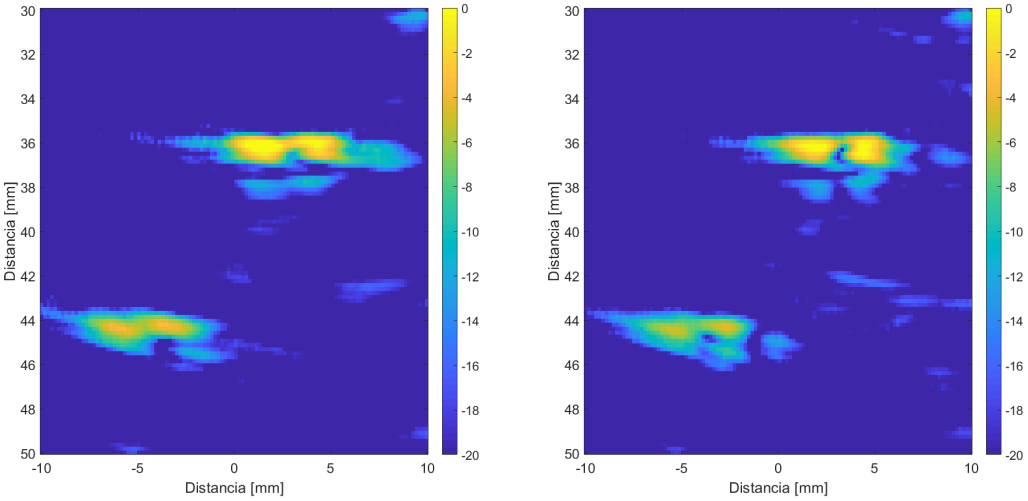



(a) Microcontrolador 32 bits. (b) Microprocessador 64 bits.

Fonte: do autor.

configuração, o número de pontos a serem processados aumenta de forma considerável, elevando significativamente o número de iterações necessárias para executar o algoritmo e, consequentemente, o tempo necessário para executar o algoritmo.

Figura 22 – Comparação entre os resultados obtidos ao processar imagens de alta qualidade em processadores de 32 bits e 64 bits.




(a) Microcontrolador 32 bits. (b) Microprocessador 64 bits.

Fonte: do autor.

Observa-se que mesmo usando uma resolução mais alta, não é possível identificar com precisão o diâmetro dos furos presentes no bloco inspecionado, cujos diâmetros são

de 0,5 mm. Essa limitação está associada ao baixo número de elementos piezelétrico ($T$,$R$) utilizados neste ensaio. Ressalta-se que essa limitação do número de sinais (e, consequentemente, redução do número de pontos da matriz) foi necessária para possibilitar que esses algoritmos pudessem ser implementados usando o sistema embarcado proposto neste trabalho, cujas especificações de *hardware* apresentam grandes limitações.

## 4.3 Comparação dos tempos de execução dos algoritmos implementados.

Após a análise comparativa entre as imagens obtidas a partir de um microprocessador de 64 bits, instalado em um computador pessoal, e um microprocessador de 32 bits, integrado a um microcontrolador, foram realizados ensaios comparando os tempos de execução dos algoritmos embarcados no sistema microcontrolador.

Nessa etapa, considerou-se inicialmente uma implementação sequencial, onde todos os dados foram processados pelo $CPU_0$ do processador. Na sequência, as tarefas foram divididas igualmente entre os dois processadores, restringindo-se a execução de cada parte do algoritmo a um núcleo de processamento específico. Finalmente, utilizando-se o sistema RTOS, implementou-se uma versão do algoritmo que possibilita a alocação de tarefas ao núcleo livre durante a solicitação realizada.

A Tabela 5 apresenta uma comparação entre os resultados obtidos para os três casos, considerando imagens com qualidades diferentes. A incerteza máxima para medição dos tempos de processamento é de ±5 ms.

Tabela 5 – Comparação entre os tempos de processamento necessários para executar os algoritmos propostos neste trabalho.

| **Qualidade** | **Execução sequencial** [**s**] | **Tarefas pré-fixadas** [**s**] | **Tarefas pós-fixadas** [**s**] | ***Pixels*** [**n**] |
|---|---|---|---|---|
| Baixa | 0,802 | 0,611 | 0,642 | 2601 |
| Média | 1,597 | 1,191 | 1,254 | 5184 |
| Alta | 3,866 | 2,892 | 3,060 | 12544 |

Fonte: do autor.

Conforme esperado, a execução das tarefas sequenciais, considerando apenas um dos núcleos de processamento, apresentou o pior resultado. Nesse caso, por não se utilizar todo o recurso disponível para processamento dos sinais, um mesmo núcleo acaba sendo sobrecarregado, tendo que executar um número maior de iterações.

Ao comparar os tempos necessários para a execução dos algoritmos implementados paralelamente, observa-se que o desempenho do algoritmo implementado a partir de tarefas pré-fixadas em função de um determinado núcleo de processamento apresentou

desempenho superior obtido utilizando-se a atribuição dinâmica das tarefas por meio de um sistema operacional de tempo real.

Nos dois casos, observou-se como fator limitante a impossibilidade de se realizar acessos concorrentes às memórias (SRAM e *flash*), impedindo, por exemplo, que dois núcleos realizem operações de leitura e escrita num mesmo banco de memória. A ausência desse tipo de recurso acaba restringindo o uso de um dos processadores em algumas situações, nos momentos em que o outro núcleo está realizando operações de leitura ou escrita na memória.

Essa limitação impede que algoritmos com essa característica sejam executados em paralelo de forma eficiente, e reduzem significativamente o desempenho da plataforma de processamento analisada.

Finalmente, considerando-se uma situação em que fosse necessário realizar uma caracterização por meio de imagens em tempo real, observa-se que sistemas baseados neste microcontrolador estariam limitados ao processamento de dados que poderiam ser usados para representar estruturas referentes a pequenas áreas, operando com uma frequência de atualização máxima de 1,6 Hz, considerando-se uma imagem com baixa qualidade (2601 pixels).

## 4.4 Considerações parciais

Este Capítulo apresentou os principais resultados obtidos ao longo do desenvolvimento deste trabalho. Foram realizadas comparações utilizando-se imagens de pequenas dimensões (20 mm× 20 mm), considerando três níveis diferentes de qualidade: baixa (2601 *pixels*), média (5184 *pixels*) e alta (12544 *pixels*).

Ao comparar as imagens geradas a partir de processadores com ULAs de 32 bits e 64 bits, observou-se que a redução da precisão das variáveis resulta em imagens com características ligeiramente diferentes, o que pode ser um problema para aplicações mais exigentes.

Verificou-se que as estratégias utilizadas para implementar os algoritmos afetam diretamente os tempos de execução dos mesmos, mas constatou-se que as limitações de acesso concorrente as memórias SRAM e *Flash* acaba limitando o desempenho do sistema, que em algumas situações permanece em modo de espera até que o semáforo libere o acesso de um determinado núcleo de processamento à memória.

Um ponto interessante observado ao longo dos ensaios é que o desempenho do sistema ao executar tarefas pré-fixadas foi superior ao desempenho obtido ao utilizar uma atribuição dinâmica de tarefas, por meio do RTOS. Essa diferença pode estar associada aos tempos necessários para o sistema identificar o núcleo de processamento livre para, então, alocar as tarefas que deverão ser executadas.

Os resultados mostram que, devido a esse tipo de limitação, no melhor dos casos (processamento executado por dois núcleos, com tarefas pré-fixadas), pode-se gerar imagens de baixa qualidade a uma taxa de atualização de 1,6 Hz. Ressalta-se que as bibliotecas utilizadas para programação dos algoritmos podem ter contribuído para essa redução de desempenho. Para trabalhos futuros, sugere-se que a Espressif-IDE seja utilizada em conjunto com as bibliotecas mais recentes fornecidas pelo fabricante do microcontrolador.

# 5 Considerações finais

Este trabalho apresentou a análise do desempenho de um microcontrolador de dois núcleos utilizado para processar tarefas com alto grau de complexidade computacional. Inicialmente, foram avaliadas as características da arquitetura do microcontrolador, tais como as especificações do processador e memórias, as características da ULA, e a possibilidade de acesso concorrente a memória e periféricos.

Para avaliar o desempenho do dispositivo, foram processadas imagens acústicas de pequenas dimensões (20 mm$\times$ 20 mm) considerando três níveis diferentes de qualidade: baixa (2601 *pixels*), média (5184 *pixels*) e alta (12544 *pixels*). Para o processamento dessas imagens, utilizou-se um algoritmo de focalização por abertura sintética (SAFT) implementado a partir de três abordagens diferentes, sendo uma sequencial e duas paralelas.

Um dos gargalos encontrados ao longo da execução deste trabalho envolveu a taxa de transferência de dados de um computador pessoal para o microcontrolador. A abordagem inicial, utilizando comunicação serial assíncrona, mostrou-se pouco eficiente, especialmente ao se utilizar o padrão ASCII para transferência de informações. Para próximos trabalhos, sugere-se que esse protocolo de transferência de dados seja substituído. Ressalta-se que o microcontrolador analisado possui um módulo de comunicação sem fio, que poderia ser utilizado para esse tipo de aplicação.

A abordagem sequencial usando um único núcleo de processamento apresentou os piores resultados. Esse comportamento era esperado, uma vez que apenas parte da capacidade do microprocessador foi utilizada ao longo da execução do algoritmo.

Ao comparar o desempenho dos algoritmos implementados de forma paralela, observou-se que o sistema apresentou um melhor desempenho quando a definição das tarefas a serem executadas é feita de forma prévia. Nessa configuração, houve um aumento de, aproximadamente, 5% no desempenho do sistema.

Um dos fatores responsáveis por gerar essa diferença entre os tempos de processamento é que, ao definir previamente o núcleo responsável por processar as tarefas, eliminam-se os tempos necessários para que o sistema operacional (RTOS) identifique o núcleo de processamento livre e atribua a ele uma tarefa específica.

A arquitetura do microcontrolador utilizado não possibilita que os núcleos de processamento realizem acesso concorrente a memória, impedindo nesse caso que os dois núcleos realizem operações de leitura e escrita simultaneamente. Ao longo do desenvolvimento deste trabalho, observou-se que essa restrição afetou diretamente o desempenho do sistema, fazendo com que em algumas situações, um dos núcleos de processamento permanecesse ocioso enquanto aguardava a liberação de um semáforo para acesso às memórias.

## 5.1 Propostas para trabalhos futuros

Como propostas para desenvolvimento de trabalhos futuros, sugere-se a utilização da Espressif-IDE como plataforma de desenvolvimento. Como mencionado, há várias bibliotecas compatíveis com essa IDE que não possuem uma versão equivalente que possa ser executada diretamente na Arduino-IDE, e que podem apresentar características diferentes das observadas ao longo do desenvolvimento deste trabalho.

Além disso, sugere-se a avaliação de outros microcontroladores, com processadores desenvolvidos por outros fabricantes. Como exemplo, pode-se citar os microcontroladores STM32H, desenvolvidos pela empresa STMicroelectronics, que utilizam processadores ARM cortex-M7 e apresentam uma arquitetura de *hardware* diferente da utilizada pelos microcontroladores da Espressif.

Finalmente, sugere-se a utilização de mini computadores (mini-pcs), utilizados para execução de aplicativos Android ou Linux embarcado. Esses dispositivos apresentam custos mais altos que microcontroladores convencionais, mas consideravelmente mais baixos do que computadores pessoais de uso geral. Como exemplo, pode-se citar o modelo Banana Pi M2 Zero, que possui um processador ARM-Cortex A7 com quatro núcleos de processamento e 512 MB de memória RAM.

# Referências

AZEVEDO, V. W. G.; BARROS, J. D. Distributed parallel computing with low cost microcontrollers for high performance electric vehicles. In: CONFERENCE ON DIGITAL SYSTEM DESIGN. *2015 Euromicro Conference on Digital System Design*. 2015. p. 104–110. Disponível em: <https://ieeexplore.ieee.org/document/7302257>. doi: 10.1109/DSD.2015.86.

BELL, D.; WOOD, G. *Multicore Programming Guide Communications Infrastructure and Voice/DSP Systems*. 2009. Disponível em: <https://e2echina.ti.com/cfs-file/___key/telligent-evolution-components-attachments/00-120-01-00-00-03-14-26/multicore-programming-guide.pdf>.

BIONDI, A. et al. Moving from single-core to multicore: Initial findings on a fuel injection case stud. In: SAE INTERNATIONAL. *SAE 2016 World Congress and Exhibition*. 2016. p. 7. Disponível em: <https://saemobilus.sae.org/content/2016-01-0017/>. doi: 10.4271/2016-01-0017.

ESPRESSIF SYSTEMS. *ESP-IDF FreeRTOS SMP Changes*. 2021. Disponível em: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/freertos-smp.html>.

ESPRESSIF SYSTEMS. *ESP32 Technical Reference Manual*. Shanghai, China, 2023. Rev. 5.0. Disponível em: <https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf>.

FARIA, L. de. *Processamento Paralelo no ESP32*. 2023. <https://github.com/lorranmarcos/Processamento-Paralelo-no-ESP32>. Último acesso: 20 de dezembro de 2023.

FLORIDIA, A. et al. Parallel software-based self-test suite for multi-core system-on-chip: Migration from single-core to multi-core automotive microcontrollers. In: IEEE. *2018 13th International Conference on Design Technology of Integrated Systems In Nanoscale Era (DTIS)*. 2018. p. 1–6. Disponível em: <https://ieeexplore.ieee.org/document/8368558>. doi: 10.1109/DTIS.2018.8368558.

GAY, W. *FreeRTOS for ESP32*. 1a edição. ed. London: Elektor, 2020. ISBN 978-1-907920-93-6.

GEBALI, F. *Algorithms And Parallel Computing*. 1a edição. ed. John Wiley e Sons, Inc, 2011. ISBN 78-0-470-90210-3. Disponível em: <https://aicitel.files.wordpress.com/2013/02/parallel-algorithms.pdf>.

GUAN, X.; HE, J.; RASSELKORDE, E. M. A time-domain synthetic aperture ultrasound imaging method for material flaw quantification with validations on small-scale artificial and natural flaws. *Ultrasonics*, v. 56, p. 487–496, 2015. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0041624X14002820>. doi: https://doi.org/10.1016/j.ultras.2014.09.018.

HARRINGTON, P.; NG, W. P. *Investigation of the speed-up of a dual microcontroller parallel processing system in the execution of a mathematical operation*. 2012. 25-26 p.

HENNESSY, J. L.; PATTERSON, D. A. *Computer architecture: a quantitative approach*. 4a edição. ed. Elsevier, 2011. Disponível em: <https://www.academia.edu/29853525/Arquitetura_de_Computadores_Uma_Abordagem_Quantitativa_David_A_Patterson_e_John_L_Hennessy>.

HOLMES, C.; DRINKWATER, B.; WILCOX, P. Post-processing of the full matrix of ultrasonic transmit-receive array data for non-destructive evaluation. *Ndt & E International*, v. 38, p. 701–711, 2005.

JUNGKLASS, M. S. P. et al. *MemOpt: Automated Memory Distribution forMulticore Microcontrollers with Hard Real-Time Requirements*. 2019. Disponível em: <https://ieeexplore.ieee.org/abstract/document/8906914>.

JUNGKLASS, P.; BEREKOVIC, M. Effects of concurrent access to embedded multicore microcontrollers with hard real-time demands. In: IEEE. *2018 IEEE 13th International Symposium on Industrial Embedded Systems (SIES)*. 2018. p. 1–9. Disponível em: <https://ieeexplore.ieee.org/document/8442079>. doi: 10.1109/SIES.2018.8442079.

LAORDEN, D. R. et al. Paralelización de los procesos de conformación de haz para la implementación del total focusing method. 2011. Disponível em: <http://hdl.handle.net/10261/37046>.

LAORDEN, D. R. et al. Paralelización de los procesos de conformación de haz para imagen ultrasónica con técnicas gpgpu. *Revista Iberoamericana de Automática e Informática industrial*, v. 9, n. 2, 2012.

LEIBSON, S. *Designing SOCs with Configured Cores: Unleashing the Tensilica Xtensa and Diamond Cores (Systems on Silicon)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006.

MAIER, A.; SHARP, A.; VAGAPOV, Y. Comparative analysis and practical implementation of the esp32 microcontroller module for the internet of things. In: IEEE. *2017 Internet Technologies and Applications (ITA)*. 2017. p. 143–148. Disponível em: <https://ieeexplore.ieee.org/document/8101926>. doi: 10.1109/ITECHA.2017.8101926.

PECCERILLO, B. et al. A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives. *Journal of Systems Architecture*, v. 129, p. 102561, 2022. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1383762122001138>. doi: https://doi.org/10.1016/j.sysarc.2022.102561.

SCHONLE, P. et al. A multi-sensor and parallel processing soc for wearable and implantable telemetry systems. In: IEEE. *ESSCIRC 2017 - 43rd IEEE European Solid State Circuits Conference*. 2017. p. 215–218. Disponível em: <https://ieeexplore.ieee.org/document/8094564>. doi: 10.1109/ESSCIRC.2017.8094564.

TENSILICA. *Xtensa Instruction Set Architecture (ISA)*. [S.l.], 2010. Disponível em: <https://0x04.net/~mwk/doc/xtensa.pdf>.

TENSILICA, INC. *Tensilica Datasheet - Xtensa LX6 Customizable DPU*. [S.l.], 2014. Disponível em: <https://mirrobo.ru/wp-content/uploads/2016/11/Cadence_Tensillica_Xtensa_LX6_ds.pdf>.

MINISTÉRIO DA EDUCAÇÃO
Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Colegiado do Curso de Engenharia Elétrica

## TERMO DE RESPONSABILIDADE

O texto do trabalho de conclusão de curso intitulado Estudo de técnicas para processamento paralelo de sinais usando microcontroladores *dual-core* é de minha inteira responsabilidade. Declaro que não há utilização indevida de texto, material fotográfico ou qualquer outro material pertencente a terceiros sem a devida citação ou consentimento dos referidos autores.

João Monlevade, 23 de fevereiro de 2024.

gov.br
Documento assinado digitalmente
LORRAN MARCOS DIAS DE FARIA
Data: 23/02/2024 09:48:05-0300
Verifique em https://validar.iti.gov.br

Lorran Marcos Dias de Faria