PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
ESCOLA POLITÉCNICA E DE ARTES
GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO

UMA BIBLIOTECA PARA COMPUTAÇÃO INTERMITENTE

GABRIEL SANTOS MENEZES

Goiânia
2025

GABRIEL SANTOS MENEZES

UMA BIBLIOTECA PARA COMPUTAÇÃO INTERMITENTE

Trabalho de Conclusão de Curso apresentado à Escola Politécnica e de Artes, da Pontifícia Universidade de Goiás, como parte dos requisitos para obtenção do título de Bacharel em Engenharia de Computação.

Orientador (a):

Profª. Dr. Talles Marcelo Gonçalves de Andrade Barbosa

Banca examinadora:

Prof.

Prof.

GOIÂNIA

2025

GABRIEL SANTOS MENEZES

UMA BIBLIOTECA PARA COMPUTAÇÃO INTERMITENTE

Trabalho de Conclusão de Curso aprovado em sua forma final pela Escola Politécnica e de Artes, da Pontifícia Universidade Católica de Goiás, para obtenção do título de Bacharel em Engenharia de Computação, em:___ /____ / ____.

_____

Orientador (a): Dr. Talles Marcelo Gonçalves de Andrade Barbosa

_____

Prof.

_____

Prof.

RESUMO

A computação intermitente surge no contexto de sistemas embarcados alimentados por fontes de energia limitadas e variáveis, como aquelas baseadas em técnicas de *energy harvesting*, nas quais a execução de programas é interrompida por quedas recorrentes de energia e reinicializações do sistema. Esse cenário impõe desafios específicos, entre eles a perda de estado da aplicação, o risco de inconsistência de dados armazenados em memória não volátil e a fragilidade da comunicação serial frente a desligamentos frequentes. Este Trabalho de Conclusão de Curso tem como objetivo propor uma biblioteca de software, desenvolvida em linguagem C para o microcontrolador MSP430G2553, que concentre mecanismos básicos de suporte à computação intermitente em nível de aplicação. A hipótese do trabalho considera que reunir, em um único artefato de software, funções de monitoramento de energia, ajuste dinâmico da frequência de clock, checkpoint simples de estado mínimo em memória Flash e organização da comunicação UART integrada ao FreeRTOS pode facilitar o desenvolvimento de aplicações embarcadas sujeitas à intermitência do suprimento de energia. A metodologia adotada compreendeu levantamento bibliográfico sobre computação intermitente e trabalhos relacionados, análise do datasheet e do guia do usuário do MSP430G2553, definição de requisitos, projeto modular da biblioteca e implementação dos módulos *ic_power, clockcontrol, ic_checkpoint e uart_lowpower*. Protótipos de firmware foram desenvolvidos e executados em placa LaunchPad, permitindo observar o comportamento dos módulos em cenários de variação de tensão e reinicializações. A biblioteca organiza mecanismos discutidos na literatura, oferecendo uma base de software para experimentos e estudos futuros em computação intermitente com microcontroladores de baixo consumo.

Palavras-chave: Computação intermitente. Sistemas embarcados. MSP430G2553. Biblioteca de software. *Energy Harvesting*. *Checkpoint* de estado.

ABSTRACT

Intermittent computing arises in the context of embedded systems powered by limited and variable energy sources, such as those based on energy harvesting techniques, in which program execution is frequently interrupted by power losses and system reboots. This scenario introduces specific challenges, including application state loss, risks of inconsistency in non-volatile memory, and fragile serial communication under frequent power failures. This undergraduate thesis proposes a software library, developed in the C programming language for the MSP430G2553 microcontroller, aiming to organize basic support mechanisms for intermittent computing at the application level. The working hypothesis considers that concentrating, in a single software artifact, functions for energy monitoring, dynamic clock frequency adjustment, simple checkpointing of minimal application state in Flash memory, and UART communication management integrated with FreeRTOS can facilitate the development of embedded applications subject to intermittent power supply. The adopted methodology included a literature review on intermittent computing and related work, analysis of the MSP430G2553 datasheet and user's guide, requirement definition, modular design of the library, and implementation of the ic_power, clockcontrol, ic_checkpoint, and uart_lowpower modules. Firmware prototypes were developed and executed on a LaunchPad board, enabling observation of module behavior under controlled voltage variations and power interruptions. The obtained results indicate that the proposed library coherently organizes mechanisms discussed in the literature, providing a concrete software basis for experiments and further studies on intermittent computing using low-power microcontrollers.

Keywords: Intermittent computing. Embedded systems. MSP430G2553. Software library. Energy harvesting. State checkpointing.

SUMÁRIO

1.INTRODUÇÃO

Este capítulo apresenta o tema do trabalho, o problema considerado, a hipótese formulada e a justificativa para o desenvolvimento da biblioteca de software proposta.

Segundo Peter Marwedel (2011, p.13, tradução própria):

> Sistemas Embarcados podem ser definidos como sistemas de processamento de informação incorporados em produtos físicos, como automóveis, equipamentos de telecomunicações e dispositivos industriais, nos quais o computador é parte do produto e não o objetivo final em si.

A compreensão da computação intermitente, exige situar esse conceito em uma linha de evolução tecnológica. A evolução dos circuitos integrados ao longo das últimas décadas reduziu de forma contínua o tamanho físico dos transistores e, por consequência, o custo por função lógica implementada. Este processo permitiu a integração de milhões de transistores em um único chip e viabilizou o uso massivo de eletrônica digital em artefatos cotidianos, criando as condições para o surgimento de sistemas embarcados em larga escala.

Conforme discutido por Rabaey et al. (2003) e Weste et al. (2011), a adoção ampla da tecnologia CMOS (*Complementary Metal-Oxide-Semiconductor*, em português: Semicondutor Complementar de Óxido de Metal) tornou possível construir circuitos digitais com consumo reduzido. Em portas lógicas CMOS, a maior parte da energia consumida está associada à potência dinâmica, que é gasta ao carregar e descarregar capacitâncias internas durante as comutações de sinal (A potência dinâmica é apresentada formalmente na equação 1.0, na sessão 2.3). Esta característica motivou a introdução, em arquiteturas modernas, modos de baixo consumo e instruções específicas para desligar blocos internos que não estão em uso. Com base nela é possível controlar o consumo de energia controlando a frequência de operação.

Higa et al. (2023, adaptado) aborda que:

> A possibilidade de integrar, em um único chip, CPU, memórias e periféricos levou ao conceito de System-on-a-Chip (SoC; em português :Sistema em um chip). Um SoC é um circuito integrado que reúne no mesmo silício, núcleos de processamento no mesmo substrato, memórias e blocos funcionais, como, interfaces seriais, conversores analógico-digitais e controladores de comunicação sem fio.

Um tipo de SoC comumente encontrado em prateleiras ou disponíveis em comércio para uso geral são os Microcontroladores, com modelos no qual o foco está em baixo consumo, simplicidade de programação e integração de periféricos típicos de controle embarcado.

A redução de custo e consumo e a integração em SoCs viabilizaram, a proliferação de redes de sensores. “Uma rede de sensores é um sistema distribuído, normalmente interconectado a um meio de comunicação sem fio, em geral para uma aplicação de propósito específico” (CARVALHO, 2005, adaptado). Esses dispositivos geralmente são distribuídos em grande número e instalados em locais de difícil acesso, o que torna a substituição periódica de baterias uma tarefa cara e, em alguns contextos, inviável.

Este cenário levou à busca de alternativas às baterias tradicionais e à adoção de técnicas de *Energy Harvesting (Em português : Coleta de energia)*. Segundo Kazmierski et.al (2011,p.5), “*Energy Harvesting* é o processo de captar energia do ambiente” — proveniente, por exemplo, de luz solar, vibrações mecânicas, variações térmicas ou campos eletromagnéticos — e convertê-la em energia elétrica utilizável para alimentar dispositivos de baixa potência. Em vez de depender exclusivamente de baterias químicas, que se esgotam e precisam ser substituídas, dispositivos alimentados por coleta de energia obtêm energia do ambiente em que estão inseridos, armazenando-a, quando possível, em capacitores ou pequenas baterias recarregáveis.

Entretanto, a energia disponível por *Energy Harvesting* é, em geral, variável e limitada. A intensidade da luz muda com o horário e as condições climáticas; a vibração mecânica depende do movimento do usuário ou da máquina; a diferença de temperatura pode oscilar ao longo do dia. Como resultado, a tensão de alimentação dos circuitos eletrônicos sobe e desce conforme a energia é captada e consumida. Assim, surge a computação intermitente.

A Computação Intermitente descreve a execução de programas em sistemas alimentados de forma não contínua. Umesh e Mittal (2021) definem Computação Intermitente como “O cenário em que períodos de execução de programas são separados por reinicializações, em geral em sistemas alimentados por dispositivos de *Energy Harvesting*”. Quando a energia acumulada é suficiente para ligar o circuito, o dispositivo inicia a execução; ao longo do tempo, a energia disponível diminui e, ao atingir um limiar mínimo de tensão, a alimentação cai, provocando o desligamento e a perda do conteúdo da memória volátil. Quando a energia volta a subir, o dispositivo reinicia, normalmente a partir do início do programa.

Esse modo de operação cria um conjunto específico de desafios. Um primeiro desafio é a perda de estado: variáveis guardadas em memória RAM, que representam o progresso lógico de uma aplicação (como contadores, índices ou resultados parciais), são perdidas a cada desligamento. Um segundo desafio é o risco de inconsistência de dados, pois operações de escrita em memória não volátil ou em periféricos podem ser interrompidas no meio de uma atualização. Um terceiro desafio está na comunicação: interfaces seriais, como UART (*Universal Asynchronous Receiver/Transmitter,* em português Receptor*/Transmissor Assíncrono Universal*), também são resetadas, e dados em trânsito podem ser perdidos ou corrompidos. Por fim, há o desafio de ajustar

o consumo de energia ao longo do tempo, modulando a frequência de operação e utilizando modos de baixo consumo para aproveitar melhor a energia intermitente disponível.

No contexto supracitado, este trabalho aborda sistemas embarcados alimentados por fontes de energia potencialmente intermitentes, em que pequenos microcontroladores realizam tarefas de sensoriamento, processamento e comunicação em redes de sensores, utilizando Coleta de Energia como forma de suprimento de energia. Com o intuito de investigar como apoiar, em software, o monitoramento e o ajuste do consumo de energia, e a preservação das informações que caracterizam o estado lógico da aplicação, em aplicações que usam comunicação serial e estão sujeitas a reinicializações frequentes devido à intermitência do suprimento de energia.

A literatura sobre computação intermitente descreve diversas técnicas para monitorar energia, ajustar frequência de operação, implementar checkpoints de estado e lidar com periféricos sob alimentação intermitente. Em projetos reais, no entanto, essas técnicas precisam ser materializadas em código que acessa registradores de hardware, organiza chamadas de função e define onde e quando o contexto de execução será salvo. Sem uma biblioteca de apoio, cada desenvolvedor tende a reimplementar soluções semelhantes em cada novo firmware, acessando diretamente registradores de conversores A/D, controladores de *clock*, controladores de memória não volátil e módulos seriais.

Reunir estes mecanismos em uma biblioteca permite reduzir o retrabalho, documentar explicitamente as decisões de projeto relacionadas à computação intermitente e oferecer uma base de código a partir da qual novas aplicações possam ser construídas.

2.REVISÃO DA LITERATURA

Este capítulo apresenta o referencial teórico necessário para compreender o tema, descrevendo os principais desafios da computação intermitente e discutindo trabalhos relacionados que propõem mecanismos de gerenciamento de energia, preservação de estado e suporte a periféricos em sistemas com energia intermitente.

2.1. Desafios da Computação Intermitente

Umesh e Mittal (2021) organizam os desafios da computação intermitente em três grupos principais. O primeiro grupo diz respeito à preservação do progresso lógico e dos dados entre reinicializações. O segundo grupo trata da consistência de dados frente a interrupções: gravações em memória não volátil e operações de entrada e saída podem ser interrompidas no meio, o que exige mecanismos para evitar estados parcialmente aplicados. O terceiro grupo envolve a relação entre desempenho, consumo de energia e qualidade dos resultados, incluindo a configuração da frequência de clock do microcontrolador, o uso de modos de baixo consumo e a decisão de quando executar mais rapidamente ou mais lentamente, em função da energia disponível.

A pesquisa também classifica as soluções propostas na literatura em camadas de abstração: hardware, arquitetura e sistema operacional, compilador e linguagem de programação, e bibliotecas ou frameworks de aplicação. No nível de hardware, aparecem arquiteturas com memória não volátil integrada e circuitos específicos para detecção de quedas de energia. No nível de sistema, surgem mecanismos de checkpoint integrados ao sistema operacional. Em compiladores e linguagens, há trabalhos que inserem automaticamente pontos de salvamento de estado no código. O presente trabalho se insere na camada de software de aplicação, com a construção de uma biblioteca em linguagem C para o MSP430G2553, que reutiliza recursos existentes no hardware.

2.2 Trabalhos Relacionados

Três linhas de trabalho são particularmente relevantes para este estudo: modulação de desempenho em função da energia, preservação de estado em memória não volátil e suporte a periféricos sob energia intermitente.

No eixo de gerenciamento de energia e desempenho, Balsamo et al. (2016) apresentam o conceito de operação *power-neutral* (*power-neutral transient computing*), em que o sistema consome, em média, a mesma quantidade de energia que consegue colher, evitando tanto o esgotamento quanto o acúmulo excessivo de energia no armazenamento. Para alcançar tal operação, os autores propõem um mecanismo de modulação suave de desempenho, baseado em *dynamic frequency scaling* (DFS, em português: Ajuste Dinâmico da Frequência), isto é, ajuste dinâmico da frequência de clock

do processador em função da potência disponível. O trabalho mostra que ajustar a frequência de *clock* com base em informações de energia é uma forma efetiva de prolongar a operação útil de sistemas transientes.

No eixo de preservação de estado, Balsamo et al. (2015) apresentam o sistema Hibernus, voltado a microcontroladores alimentados por *energy harvesting*. O Hibernus monitora a tensão de alimentação e, ao se aproximar de um limiar de falha, salva o estado do sistema em memória não volátil e coloca o dispositivo em um modo de baixo consumo. Quando a alimentação é restabelecida, o estado é restaurado e a execução prossegue a partir do ponto salvo. Essa estratégia é reativa, pois o salvamento é disparado em resposta à queda de energia, e busca minimizar a sobrecarga de tempo e energia associada a cada ciclo de hibernação.

Maeng e Lucia (2019) abordam especificamente o problema de periféricos em sistemas com energia intermitente. Os autores mostram que técnicas de *checkpoint just-in-time* podem ser insuficientes quando periféricos, como interfaces de comunicação ou aceleradores, mantêm estado próprio e interagem com o ambiente externo de forma irreversível. Para esse cenário, apresentam o sistema Samoyed, que combina checkpoints just-in-time com operações atômicas de periféricos e *undo-logging*, isto é, o registro das operações de forma que seja possível desfazer seus efeitos em caso de reinicialização. O sistema também utiliza perfilamento de energia e escalonamento dinâmico de carga para coordenar o uso dos periféricos em função da energia disponível.

Em conjunto, esses trabalhos indicam que soluções de computação intermitente combinam: monitoramento de energia e modulação de desempenho, mecanismos de checkpoint em memória não volátil e tratamento explícito de periféricos e comunicação em presença de falhas de energia. O presente trabalho dialoga com essas contribuições ao concentrar, em uma biblioteca para o MSP430G2553, versões simplificadas desses mecanismos em nível de software de aplicação, sem alterar o hardware e sem depender de extensões específicas de compilador.

2.3. Tecnologia CMOS e Consumo De Energia

A maior parte dos microcontroladores utilizados em sistemas embarcados atuais, incluindo o MSP430G2553, é implementada em tecnologia CMOS (*Complementary Metal-Oxide-Semiconductor*). Em circuitos digitais CMOS, a maior parcela do consumo de energia está associada à potência dinâmica, isto é, à energia gasta para carregar e descarregar as capacitâncias dos nós internos a cada comutação de sinal. Essa potência dinâmica é usualmente modelada pela expressão descrita na Equação 1:

$$P_{dinâmico=} \propto C\,V^2 f \qquad Equação\ 1\ -\ Potência\ dinâmica$$

Em que C é a capacitância comutada, V é a tensão de alimentação, f é a frequência de *clock* e $\propto$ é o fator de atividade de chaveamento.

Essa relação mostra que o aumento da frequência ou da tensão eleva diretamente o consumo de potência e, consequentemente, a corrente drenada da fonte. Para reduzir esse consumo, projetistas de microcontroladores passaram a incluir, no conjunto de instruções e nos registradores de controle, mecanismos explícitos para desligar blocos internos, reduzir a frequência de *clock* e selecionar modos de baixo consumo. Microcontroladores de diferentes famílias, como PIC18, AVR e MSP430, passaram a disponibilizar instruções e registradores dedicados para controle fino de osciladores, *prescalers* e blocos periféricos.

A relação direta entre frequência e corrente de dreno explica por que ajustes de *clock* e modos de baixo consumo são componentes indispensáveis em soluções de computação intermitente. A existência, no MSP430G2553, de instruções e registradores específicos para gestão de consumo permite que uma biblioteca de software encapsule essas funcionalidades em uma API mais simples, reutilizável e adequada ao contexto de energia intermitente, sem exigir que o desenvolvedor manipule diretamente todos os detalhes de baixo nível.

3. UMA BIBLIOTECA PARA COMPUTAÇÃO INTERMITENTE UTILIZANDO O MSP430

Este capítulo descreve o desenvolvimento da biblioteca proposta, apresentando a metodologia adotada, as tecnologias utilizadas e os protótipos construídos. O objetivo é explicitar como o software foi organizado, quais decisões foram tomadas em cada etapa e de que forma os cenários montados permitem observar o funcionamento da biblioteca.

3.1 Metodologia de Desenvolvimento

O processo foi estruturado em etapas encadeadas, centrado na construção de um artefato de software — uma biblioteca em linguagem C para o MSP430G2553 — e na sua utilização em protótipos executados em placa de desenvolvimento.

Primeiro foi feito um levantamento bibliográfico sobre computação intermitente, com foco em definições, desafios e mecanismos recorrentes. Esse levantamento tomou como referência a pesquisa de Umesh e Mittal (2021), bem como os trabalhos de Balsamo et al. (2015; 2016) e Maeng e Lucia (2019), que tratam respectivamente de modulação de desempenho, hibernação baseada em memória não volátil e suporte a periféricos com checkpoints. A partir desses trabalhos, foram identificados como elementos centrais para o escopo deste projeto: monitoramento de energia, ajuste de frequência de *clock, checkpoint* simples de estado e tratamento da comunicação serial sob reinicializações.

O *datasheet* forneceu as principais características elétricas e funcionais do microcontrolador, incluindo faixas de tensão de alimentação, consumo em diferentes modos de operação, interfaces disponíveis e capacidade de memória. O *Guia de Usuário* da família MSP430 descreveu em detalhes o subsistema de clock (incluindo o oscilador controlado digitalmente, DCO, e as constantes de calibração), os modos de baixo consumo (*low-power modes*), a estrutura da memória Flash (incluindo segmentos de informação, memória), o conversor analógico-digital e o módulo USCI para UART. Essa análise permitiu relacionar os mecanismos discutidos na literatura com recursos concretos do MSP430G2553.

Em seguida foram levantados alguns dos requisitos da biblioteca. Com base no referencial teórico e na análise do hardware, foram definidos os seguintes requisitos principais:

- Disponibilizar uma API de monitoramento de energia capaz de estimar a tensão de alimentação a partir de leituras do ADC interno e classificá-la em faixas qualitativas, como “boa”, “de alerta” e “crítica”;
- Permitir a seleção, em tempo de execução, de perfis discretos de frequência de clock suportados pelas calibrações do DCO fornecidas pela Texas Instruments;

- Oferecer um mecanismo simples de checkpoint de um estado mínimo da aplicação em segmento de Info Flash, incluindo cabeçalho com informação de validade;
- Encapsular o uso da UART em torno de filas de transmissão e recepção, com função de *flush* para garantir o escoamento de dados antes de uma possível falha de energia.

Assim, o projeto dos módulos foi iniciado, dividindo o código em módulos relativamente independentes, cada um implementado em um par de arquivos .c e .h, com responsabilidades bem definidas:

- *ic_power*: responsável por inicializar o ADC interno para leitura de VCC, converter o valor de ADC em milivolts por meio de uma relação linear calibrada e classificar o nível de energia em faixas configuráveis;
- *clockcontrol*: responsável por selecionar, em tempo de execução, frequências de clock do DCO a partir das constantes de calibração do MSP430G2553, expondo funções de alto nível para troca de perfil de frequência;
- *ic_checkpoint*: responsável por definir o formato de um bloco de checkpoint, contendo cabeçalho e área de dados, e por implementar funções de apagamento do segmento de Info Flash escolhido, gravação do bloco e restauração para variáveis em RAM;
- *uart_lowpower*: responsável por encapsular a configuração da UART, associar interrupções a filas de transmissão e recepção e expor funções de envio de bytes e cadeias de caracteres, além de uma função de *flush* que bloqueia até que o buffer de transmissão esteja vazio.

Em seguida foi realizada a implementação incremental dos módulos em linguagem C, utilizando o Code Composer Studio como ambiente de desenvolvimento. Code Composer Studio é uma IDE (*Integrated Development Environment*) da Texas Instruments, que integra editor de código, compilador, ligador e depurador para microcontroladores como o MSP430. O módulo clockcontrol foi implementado primeiro, exercitando a troca de frequências do DCO e observando os efeitos em temporizadores e na UART. Em seguida, o módulo *ic_power* foi desenvolvido, configurando o ADC para leitura do canal associado à tensão de alimentação e conversão de contagens de ADC em milivolts.

Na sequência, o módulo *ic_checkpoint* foi implementado, com a escolha de um segmento de memória livre e a codificação das rotinas de apagamento e escrita, conforme o fluxo descrito no Guia de usuário para o controlador de memória Flash. O módulo *uart_lowpower* foi então integrado, encapsulando a configuração da USCI em modo UART, associando filas de transmissão e recepção e implementando uma função de *flush* para garantir o escoamento de dados antes de uma possível suspensão.

Por fim, foi feita a integração dos módulos com o sistema operacional de tempo real FreeRTOS e a construção de protótipos de firmware. FreeRTOS é um *Real-Time Operating System* voltado para microcontroladores, que fornece abstrações como tarefas, filas e temporizadores para organização de aplicações embarcadas. Nos protótipos, tarefas FreeRTOS foram usadas para estruturar a interação entre monitoramento de energia, modulação de *clock,* checkpoint de estado e comunicação serial.

3.2 Tecnologias Utilizadas

O desenvolvimento da biblioteca utilizou as seguintes tecnologias e ferramentas principais.

- Microcontrolador MSP430G2553 e placa LaunchPad: o MSP430G2553 foi o alvo de hardware, escolhido por suas características de baixo consumo, conjunto de periféricos e documentação disponível. A placa LaunchPad correspondente integra o microcontrolador, circuito de programação e depuração, LEDs e botões, permitindo gravação e depuração via USB.
- Code Composer Studio (CCS): o Code Composer Studio foi utilizado como ambiente de desenvolvimento para edição, compilação, ligação e depuração do código em C para o MSP430. A IDE oferece suporte específico à família MSP430, incluindo integração com o depurador de hardware e exemplos de projetos.
- FreeRTOS: O FreeRTOS é um sistema operacional de tempo real (Em inglês, *Real-Time Operating System*, RTOS) para microcontroladores, que fornece abstrações como tarefas, filas e temporizadores. O escalonador do FreeRTOS é responsável por realizar o escalonamento de tarefas, isto é, decidir qual tarefa deve usar a CPU em cada instante, e por executar trocas de contexto, salvando e restaurando registradores e o contador de programa de cada tarefa. Em ambientes de computação intermitente, esse mecanismo é importante porque organiza a aplicação em unidades de execução bem definidas e facilita a identificação do estado que precisa ser preservado entre reinicializações. Além disso, o FreeRTOS possui uma camada de *port* específica para cada arquitetura suportada; essa camada implementa os detalhes de troca de contexto para o MSP430 e para outros microcontroladores, permitindo que a mesma estrutura de tarefas seja recompilada para diferentes plataformas. A biblioteca proposta foi integrada a esse ambiente, de forma que as funções de monitoramento de energia, ajuste de clock, checkpoint e UART possam ser chamadas diretamente a partir de tarefas FreeRTOS.
- Terminal serial no computador hospedeiro: programas como PuTTY ou minicom foram utilizados como terminais seriais para enviar e receber dados pela UART do MSP430, por meio da interface USB da placa LaunchPad. O terminal permite visualizar textos enviados pela aplicação

e enviar comandos simples (por exemplo, caracteres) ao microcontrolador.

De acordo com o datasheet do MSP430G2553, o dispositivo foi projetado para operar com correntes da ordem de centenas de microampères em modo ativo, mesmo durante a execução contínua de instruções, desde que operando em frequências moderadas. Para uma frequência de 1 MHz e tensão de alimentação típica, o consumo em modo ativo situa-se na ordem de aproximadamente 230 a 330 microamperes, dependendo da configuração do *clock* e da tensão aplicada. Esse comportamento reflete decisões de projeto associadas à tecnologia CMOS e à arquitetura interna do dispositivo, que privilegiam baixo consumo dinâmico mesmo durante a execução normal do programa.

Em contraste, microcontroladores de uso geral amplamente difundidos no meio acadêmico e didático, como dispositivos da família AVR (por exemplo, ATmega48/328) e da família PIC18, apresentam correntes significativamente mais elevadas em modo ativo sob condições equivalentes de operação. Embora esses dispositivos também ofereçam modos de economia de energia e mecanismos de desligamento de periféricos, o consumo durante a execução contínua de instruções permanece tipicamente na faixa de miliamperes, o que impacta diretamente a viabilidade de aplicações alimentadas por fontes de energia intermitentes.

No contexto da computação intermitente, esse aspecto é particularmente relevante. Durante os períodos em que o sistema está energizado, a execução do código deve consumir o mínimo de energia possível para maximizar o trabalho útil realizado antes de uma eventual queda de tensão. Assim, um microcontrolador que drena corrente na ordem de microamperes em modo ativo permite explorar com maior eficiência fontes de energia limitadas e variáveis, como aquelas associadas a *energy harvesting*. Dessa forma, a escolha do MSP430G2553 está alinhada com o objetivo do trabalho de estudar e estruturar mecanismos de software voltados à computação intermitente, utilizando uma plataforma cujo comportamento energético favorece esse tipo de investigação.

O FreeRTOS também é um elemento importante na proposta. Um sistema operacional de tempo real (*Real-Time Operating System*- RTOS, em português Sistema Operacional em Tempo Real) é um software que oferece abstrações como tarefas, filas e temporizadores, além de um escalonador responsável por decidir qual tarefa executa em cada instante.

No FreeRTOS, o escalonador realiza trocas de contexto entre tarefas, isto é, salva o conjunto de registradores e o contador de programa da tarefa atual e restaura o contexto de outra tarefa, de forma cooperativa ou preemptiva, dependendo da configuração. Em computação intermitente, esse mecanismo é relevante porque organiza o código da aplicação em tarefas bem delimitadas, facilita o raciocínio sobre o estado que deve ser preservado em checkpoints e oferece uma camada de portabilidade: a mesma estrutura de

tarefas e filas pode ser recompilada para outros microcontroladores, desde que exista uma portabilidade adequado do FreeRTOS para a arquitetura. A biblioteca desenvolvida neste trabalho foi pensada para integrar-se a esse ambiente, encapsulando detalhes específicos do MSP430G2553 em funções podem ser chamadas a partir de tarefas FreeRTOS.

3.3 Apresentação dos Protótipos

Esta seção descreve os protótipos de firmware construídos para exercitar os módulos da biblioteca.

A figura 1 mostra apresenta o fluxo de início de execução da biblioteca proposta. Após a ocorrência de um reset, seja ele provocado pela energização inicial do sistema ou por uma queda abrupta de energia, o fluxo de execução da biblioteca é iniciado no estágio de inicialização. Nesse ponto, o firmware executa os procedimentos básicos de configuração do microcontrolador, assegurando que o sistema comece sua operação em um estado conhecido e controlado. Quando aplicável, o *watchdog timer* é desabilitado para evitar reinicializações indesejadas durante a configuração inicial.

Em seguida, o sistema configura o *clock* principal do microcontrolador em uma frequência considerada segura para a fase de inicialização. O oscilador controlado digitalmente (*Digital Controlled Oscillator* – DCO) é ajustado para uma frequência reduzida, tipicamente em torno de 1 MHz, com o objetivo de limitar o consumo de corrente durante os primeiros instantes de execução, nos quais a condição energética ainda não foi avaliada.

Após a configuração do *clock*, a interface de comunicação serial é inicializada. Nessa etapa, os pinos P1.1 e P1.2 são configurados para operar como entrada e saída da UART, respectivamente, e os registradores do módulo USCI_A0 são ajustados para o modo UART assíncrono. O *baud rate* é definido de forma coerente com a frequência configurada para o SMCLK, garantindo a correta temporização dos bits transmitidos e recebidos.

Concluída a inicialização da UART, o módulo de gerenciamento de energia é configurado. Para isso, o conversor analógico-digital ADC10 é inicializado e preparado para realizar leituras da tensão de alimentação do sistema. Com base nessas leituras, são carregados os limiares que definem as faixas energéticas utilizadas pela biblioteca, denominadas V_GOOD e V_WARN, que delimitam os estados de operação normal, alerta e crítico.

Na sequência, o módulo de checkpoint é inicializado. Essa etapa garante que a memória Flash de informação destinada ao salvamento de estado esteja corretamente configurada e protegida contra escritas indevidas, permanecendo inicialmente travada para operações de leitura.

Após a inicialização dos módulos, o fluxo verifica a existência de um checkpoint válido armazenado na memória não volátil. Essa verificação é

realizada por meio da leitura de um valor de controle previamente gravado, que indica se há um estado lógico consistente salvo.

Caso um checkpoint válido seja identificado, o sistema restaura para a memória RAM apenas o estado mínimo necessário para a continuidade da aplicação, como contadores, indicadores de progresso ou variáveis de controle. Caso não exista um checkpoint válido, a execução prossegue com o estado inicial padrão da aplicação, iniciando o processamento a partir do ponto inicial definido pelo firmware.

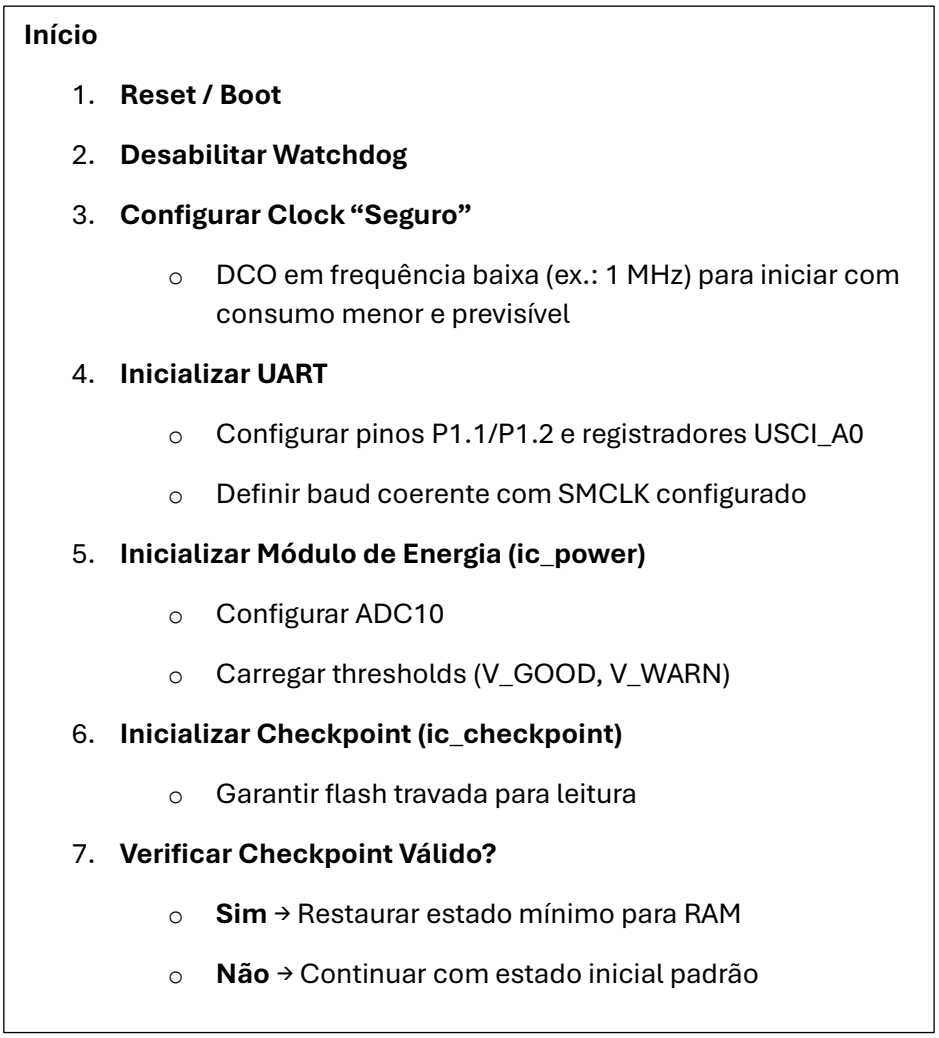

Figura 1: Fluxo de execução inicial da biblioteca

Após a conclusão da fase de inicialização, o sistema passa a operar em um laço de execução contínuo, que constitui o núcleo do funcionamento da biblioteca proposta. A figura 2 apresenta esse laço, responsável por monitorar continuamente a condição energética do sistema e por orientar as decisões de controle de desempenho, salvamento de estado e entrada em modos de baixo consumo.

A cada iteração do laço principal, a biblioteca realiza a amostragem da tensão de alimentação do microcontrolador. Essa amostragem é efetuada por meio da função ic_power_read_vcc(), que utiliza o conversor analógico-digital ADC10 para estimar o valor da tensão de alimentação, representado pela variável Vcc_mV. Esse valor expressa, em milivolts, a condição instantânea de energia disponível ao sistema.

Com base no valor amostrado, a biblioteca procede à classificação do nível energético por meio da função ic_power_classify (Vcc_mV). Essa função compara a tensão medida com limiares previamente definidos e associa o resultado a um estado energético, denominado PWR_STATE. Os estados possíveis são *GOOD*, quando a tensão é suficiente para operação normal; *WARN*, quando a tensão indica risco de queda iminente e exige ações de mitigação; e *CRITICAL*, quando a energia disponível é insuficiente para manter a execução segura do sistema.

A partir dessa classificação, o fluxo de execução segue para os blocos de decisão subsequentes do diagrama, nos quais são aplicadas as estratégias adequadas a cada faixa energética, caracterizando o comportamento adaptativo da biblioteca frente às variações da alimentação.

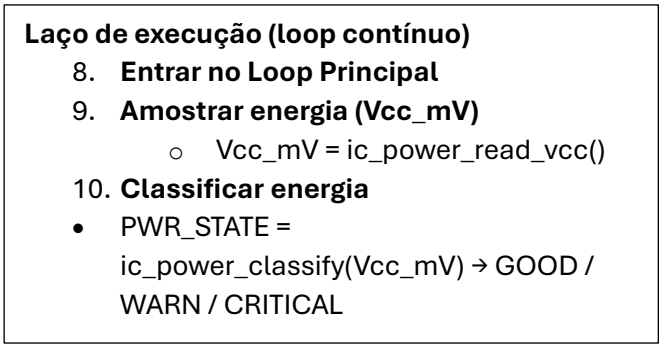

Figura 2: Fluxo de execução do laço principal

A figura 3 apresenta a tomada de decisão, onde após a classificação do nível energético, o fluxo de execução entra na etapa de tomada de decisão, que representa o núcleo do funcionamento da biblioteca no contexto da computação intermitente. Nessa etapa, as ações do sistema passam a ser condicionadas ao estado energético identificado, permitindo que o comportamento da aplicação seja adaptado dinamicamente à energia disponível.

Quando o estado energético é classificado como *GOOD*, entende-se que a tensão de alimentação é suficiente para uma operação estável e contínua. Nesse caso, a biblioteca ajusta o *clock* do sistema para um nível elevado, por exemplo 16 MHz, por meio da função clockcontrol_set (*HIGH*), privilegiando desempenho. Com o *clock* configurado, a lógica principal da aplicação é executada normalmente, incluindo processamento de dados, leitura de sensores e preparação de mensagens a serem transmitidas.

A comunicação serial ocorre de forma contínua, sendo os dados de transmissão enfileirados ou armazenados em buffers e enviados pela UART, enquanto os dados recebidos são capturados por interrupções e posteriormente consumidos pela aplicação. Ao final dessa etapa, o fluxo retorna à amostragem da energia, reiniciando o ciclo de monitoramento.

Caso o estado energético não seja *GOOD*, o fluxo avança para a verificação do estado *WARN*. A classificação *WARN* indica uma condição de pré-queda de energia, na qual ainda existe energia suficiente para executar ações corretivas, mas com risco de interrupção iminente. Nessa situação, a biblioteca reduz o *clock* do sistema para um valor inferior, por exemplo 1 MHz, com o objetivo de diminuir o consumo de corrente. Em seguida, é verificada a existência de dados pendentes de transmissão na UART. Caso existam dados a serem transmitidos, a função uart_flush_tx() é acionada com um tempo limite curto, permitindo que a transmissão seja concluída sem bloquear indefinidamente a execução. Independentemente do sucesso total do esvaziamento da fila, o fluxo prossegue para o salvamento do checkpoint mínimo.

Nesta etapa, a função ic_checkpoint_save() armazena em memória não volátil apenas o estado essencial da aplicação, como contadores, indicadores de progresso do algoritmo ou informações sobre o último pacote transmitido. Concluídas essas ações de mitigação, o fluxo retorna novamente à etapa de amostragem da energia.

Se o estado energético não for *WARN*, o sistema avalia a condição *CRITICAL*. A classificação *CRITICAL* indica que a energia disponível é insuficiente para manter a execução segura do sistema. Nesse cenário, a biblioteca prioriza a preservação do estado do sistema, desabilitando periféricos não essenciais e colocando o microcontrolador em um modo de baixo consumo. Tipicamente, é utilizado o modo LPM3, por meio da instrução __bis_SR_register (LPM3_bits | GIE), embora modos mais profundos possam ser adotados conforme a configuração do projeto.

Com a continuidade da queda de energia, ocorre a interrupção total da alimentação, resultando em um reset do microcontrolador. Quando a energia é restabelecida, o fluxo de execução reinicia a partir da etapa de *boot*, com a possibilidade de restauração do estado previamente salvo.

Em situações intermediárias ou transitórias, nas quais o estado energético não se enquadra claramente em nenhuma das faixas, o fluxo retorna à etapa de amostragem da energia, mantendo o ciclo de monitoramento ativo.

Tomada de decisão (núcleo da computação intermitente)

Decisão A — PWR_STATE == GOOD?

- **Sim (GOOD):**
    1. clockcontrol_set(HIGH) (ex.: 16 MHz)
    2. **Executar lógica da aplicação**
        - processamento
        - leitura de sensores
        - montagem de mensagens
    3. **Enviar/Receber UART**
        - TX: enfileirar/bufferizar bytes e transmitir via ISR/polling
        - RX: ISR armazena em buffer/fila e aplicação consome
    4. Voltar para **Amostrar energia** (passo 9)
- **Não:** vai para Decisão B

Decisão B — PWR_STATE == WARN?

- **Sim (WARN):** (pré-queda, mitigação)
    1. clockcontrol_set(LOW) (ex.: 1 MHz) para reduzir consumo
    2. **Existe TX pendente?** (fila/buffer não vazio)
        - **Sim:** uart_flush_tx(timeout_curto)
            - Se **flush conclui**, segue
            - Se **timeout**, segue mesmo assim (sem travar)
        - **Não:** segue
    3. **Salvar checkpoint mínimo**
        - ic_checkpoint_save(min_state)
        - (estado mínimo: contadores, etapa do algoritmo, último pacote enviado, etc.)
    4. Voltar para **Amostrar energia** (passo 9)
- **Não:** vai para Decisão C

Decisão C — PWR_STATE == CRITICAL?

- **Sim (CRITICAL):** (queda iminente)
    1. **Desabilitar periféricos não essenciais**
    2. **Entrar em Low Power Mode**
        - __bis_SR_register(LPM3_bits | GIE) (ou LPM4 conforme projeto)
    3. **Queda ocorre / Reset ocorre**
        - O fluxo retorna ao **Reset / Boot** (passo 1)
- **Não:** voltar para o passo 9 (caso raro/intermediário)

Figura 3: Fluxo de execução do laço principal

A figura 4 apresenta o protótipo utilizado para os testes da biblioteca. O *LaunchPad* MSP430G2553 utilizado projeto pode ser alimentado externamente por meio dos pinos de alimentação disponíveis em seus conectores laterais. A tensão de alimentação deve ser aplicada diretamente ao pino VCC da placa, enquanto o retorno deve ser conectado a um dos pinos GND.

A tensão utilizada deve respeitar a faixa de operação do microcontrolador MSP430G2553, tipicamente entre 1,8 V e 3,6 V, sendo comum o uso de 3,3 V em bancada. Para evitar conflitos elétricos, é necessário remover o jumper de alimentação que conecta a placa à tensão proveniente da interface USB, garantindo que apenas uma fonte de alimentação esteja ativa durante os testes.

A placa *LaunchPad* MSP430G2553 disponibiliza múltiplos pinos rotulados como VCC e GND, os quais pertencem ao mesmo domínio de alimentação elétrica e estão interligados internamente por planos de potência e terra. Dessa forma, a alimentação externa pode ser aplicada em qualquer um desses pontos sem alteração do funcionamento do microcontrolador. Para os experimentos realizados neste trabalho, optou-se pelo uso do conjunto de pinos VCC e GND localizado na região inferior da placa, por facilitar a conexão da fonte externa de alimentação. Optou-se por utilizar neste projeto uma alimentação intermitente, proveniente de dois minis painéis solares de 5 V, que transmitem a energia coletada para um circuito de estabilização que em seguida alimenta o Kit de desenvolvimento.

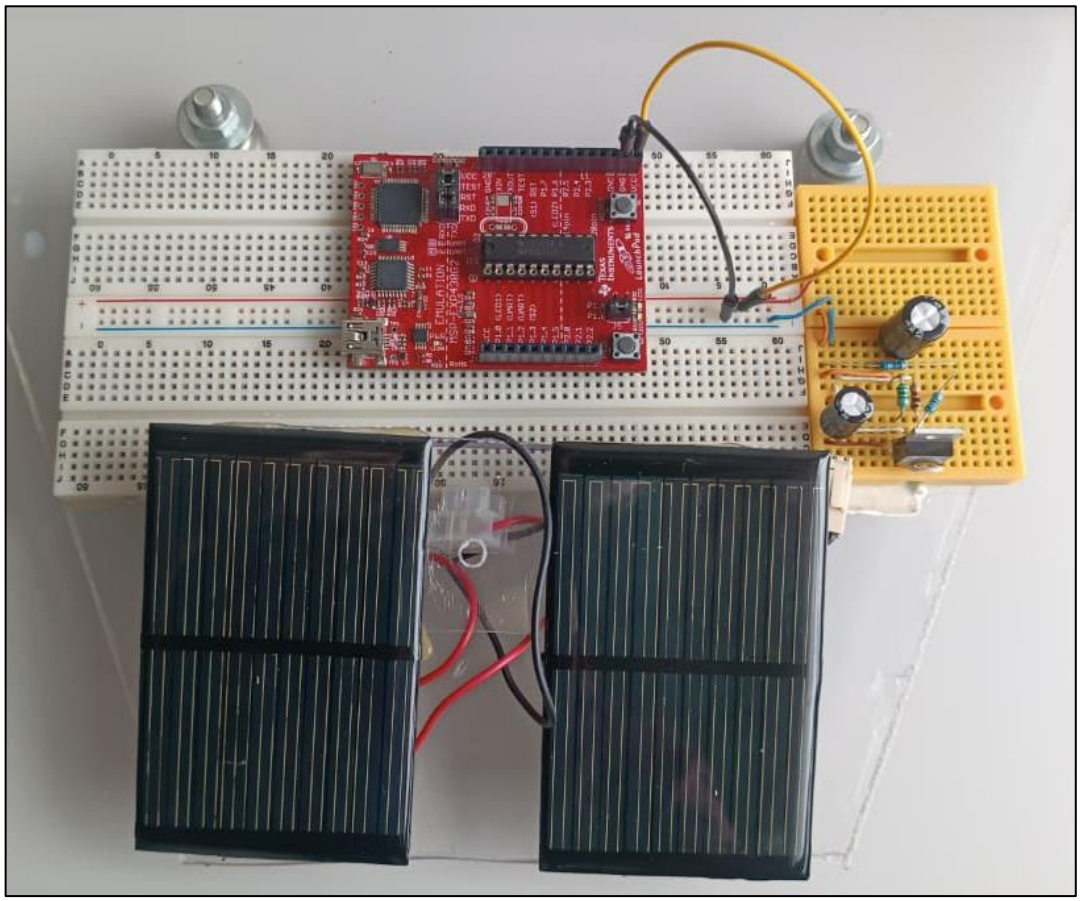

Figura 4: Protótipo utilizado para os testes

4. CONCLUSÃO

Este capítulo apresenta as conclusões do trabalho, discutindo a complexidade do desenvolvimento realizado, a utilidade potencial da biblioteca, os elementos de originalidade e as limitações do estudo, além de indicar propostas para trabalhos futuros.

O trabalho teve como objetivo propor uma biblioteca de software em linguagem C para o microcontrolador MSP430G2553, voltada ao tratamento organizado de alguns dos principais desafios da computação intermitente: monitoramento de energia, ajuste de desempenho via frequência de *clock,* preservação de um estado mínimo em memória não volátil e tratamento da comunicação UART em presença de reinicializações frequentes. A biblioteca foi organizada em quatro módulos — *ic_power, clockcontrol, ic_checkpoint e uart_lowpower* — e exercitada em protótipos de firmware executados em placa *LaunchPad.*

A complexidade do projeto decorre da necessidade de integrar conceitos de sistemas embarcados, computação intermitente e arquitetura específica do MSP430G2553. Foi preciso relacionar os mecanismos discutidos na literatura, como operação *power-neutral*, hibernação orientada por limiar de tensão e *checkpoints just-in-time* para periféricos, com os recursos concretos disponibilizados pela Texas Instruments, como DCO, modos de baixo consumo, ADC interno e segmentos de memória. Também foi necessário compreender o uso de um RTOS, como o FreeRTOS, para estruturar tarefas e filas em aplicações embarcadas.

No que se refere à utilidade, a biblioteca proposta oferece ao desenvolvedor de sistemas embarcados um ponto de partida concreto para experimentos com computação intermitente no MSP430G2553. Ao encapsular tarefas recorrentes de baixo nível — como leitura de VCC, seleção de perfis de frequência com base em calibrações do DCO, salvamento e restauração de estruturas reduzidas em Info Flash e gestão de filas de transmissão pela UART — a biblioteca busca reduzir o esforço necessário para configurar o hardware e permitir que novos projetos foquem na lógica específica da aplicação. Em ambientes acadêmicos, o artefato pode ser utilizado como apoio didático para demonstrar, em código, como mecanismos básicos de tolerância a falhas de energia podem ser implementados sobre um microcontrolador de uso corrente.

Quanto à originalidade, o trabalho não introduz novos conceitos teóricos em computação intermitente além dos já sistematizados. A contribuição do trabalho consiste em investigar e combinar, em um único artefato de software direcionado ao MSP430G2553, conceitos presentes na literatura — como monitoramento de energia por faixas, modulação de frequência de *clock,* checkpoint simples de estado e tratamento explícito de comunicação sob reinicializações —, explorando os recursos disponíveis no dispositivo sem modificar o hardware nem depender de extensões específicas de compilador.

Como propostas para trabalhos futuros, destacam-se a realização de uma avaliação quantitativa da biblioteca, comparando aplicações que utilizam os

módulos *ic_power, clockcontrol, ic_checkpoint* e *uart_lowpower* com aplicações que tratam os mesmos problemas de forma ad hoc, medindo métricas como número de instruções executadas entre reinicializações, energia aproveitada por ciclo e taxa de perda de dados. Uma investigação de uma independência de plataforma da biblioteca para outros microcontroladores da família MSP430, analisando as diferenças de memória, periféricos e calibrações de *clock* e adequando o código por meio de diretivas de compilação e camadas de abstração.

Em síntese, o trabalho apresentou uma biblioteca de software modular voltada à computação intermitente no MSP430G2553, e descreveu o processo de desenvolvimento desse artefato. A biblioteca fornece uma base concreta para que estudos posteriores aprofundem tanto a avaliação experimental quanto a extensão das funcionalidades.

5. REFERÊNCIAS

- BALSAMO, D. et al**. Graceful performance modulation for power-neutral transient computing systems**. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 35, n. 5, p. 738–749, 2016.
- BALSAMO, D. et al**. Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems**. *IEEE Embedded Systems Letters*, v. 7, n. 1, p. 15–18, 2015.
- CARVALHO,H. S. **Data fusion implementation in sensor networks Applied to health monitoring**. 2005.158 f. Tese (doutorado em Ciência da computação)-Departamento de Ciência da Computação ,Universidade Federal de Minas Gerais , Belo Horizonte.

  Disponível em: https://tecnoblog.net/responde/o-que-e-um-soc-system-on-a-chip/. Acesso em: 5 dezembro de 2025.
- FREERTOS. **FreeRTOS™** Real-time operating system for microcontrollers. Disponível em: FreeRTOS.org.
- HIGA,Paulo; MARQUES,Ana.O que é SoC e c omo funciona o conjunto de processadores do seu smartphone. **Tecnoblog**,2023. Entenda como funciona um system-on-a-chip.
- KAZMIERSKI, Tom J.; BEEBY, Steve. Energy harvesting systems. **Principles, Modeling and Applications**, 2011.
- MAENG, K.; LUCIA, B. **Supporting peripherals in intermittent systems with just-in-time checkpoints**. In: *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. New York: ACM, 2019. p. 1101–1116.
- MARWEDEL, P. **Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems and the Internet of Things**. Springer, 2011,
- RABAEY, J. M.; CHANDRAKASAN, A.; NIKOLIC, B. **Digital Integrated Circuits: A Design Perspective**. 2. ed. Prentice Hall, 2003. (capítulos iniciais de dissipação de potência em CMOS)
- UMESH, S.; MITTAL, S. **A survey of techniques for intermittent computing.** *Journal of Systems Architecture*, v. 112, p. 101859, 2021.
- WESTE, N. H. E.; HARRIS, **D. *CMOS VLSI Design: A Circuits and Systems Perspective.*** 4. ed. Addison-Wesley, 2011.

APÊNDICE A – CÓDIGO FONTE DA BIBLIOTECA

O código-fonte completo desenvolvido para este trabalho, assim como toda documentação, está disponível publicamente no repositório GitHub do autor, no seguinte endereço:

https://github.com/G-SantoSs/msp430-lowpower-freertos

ANEXO A – MANUAL DO FABRICANTE DE CADA COMPONENTE

Nesta seção são apresentados todos os datasheets (manuais) dos componentes utilizados no desenvolvimento do protótipo.

• MSP430: MSP430G2x53, Texas Instruments. Disponível em: https://www.ti.com/product/MSP430G2553. Acesso em: dezembro. 2025.

• Guia de Usuário para o MSP430: User’s guide, *MSP430x2xx Family User’s Guide*. Disponível em:

https://www.ti.com/product/MSP430G2553. Acesso em: dezembro.2025