Pontifícia Universidade Católica
do Rio de Janeiro

Jerônimo Augusto Soares

Desenvolvimento de Sistema IoT sem Uso de Baterias

Projeto Final

Tese apresentada como requisito parcial para obtenção do grau de Bacharel pelo Programa de Graduação em Engenharia da Computação, do Departamento de Informática da PUC-Rio.

Orientador: Prof. Adriano Francisco Branco

Rio de Janeiro
Junho de 2025

Pontifícia Universidade Católica
do Rio de Janeiro

Jerônimo Augusto Soares

Desenvolvimento de Sistema IoT sem Uso de Baterias

Tese apresentada como requisito parcial para obtenção do grau de Bacharel pelo Programa de Graduação em Engenharia da Computação da PUC-Rio. Aprovada pela Comissão Examinadora abaixo:

Prof. Adriano Francisco Branco
Orientador
Departamento de Informática – PUC-Rio

Rio de Janeiro, 30 de Junho de 2025

Todos os direitos reservados. A reprodução, total ou parcial do trabalho, é proibida sem a autorização da universidade, do autor e do orientador.

Jerônimo Augusto Soares

Ficha Catalográfica

Soares, Jerônimo Augusto

Desenvolvimento de Sistema IoT sem Uso de Baterias / Jerônimo Augusto Soares; orientador: Adriano Francisco Branco. – 2025.

42 f: il. color. ; 30 cm

Projeto Final (graduação) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2025.

Inclui bibliografia

1. Internet das Coisas. 2. Energy Harvesting. 3. Gerenciamento de Energia. 4. Baixo Consumo de Energia. 5. Zephyr RTOS. 6. nRF52840. 7. IEEE 802.15.4. 8. Device Tree. I. Branco, Adriano. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Resumo

Soares, Jerônimo Augusto; Branco, Adriano. **Desenvolvimento de Sistema IoT sem Uso de Baterias**. Rio de Janeiro, 2025. 42p. Projeto Final – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Este trabalho detalha o desenvolvimento de um *firmware* de ultra-baixo consumo para um sistema IoT, visando a operação sem baterias com energia do ambiente (*Energy Harvesting*). Utilizando o SoC nRF52840 e o Zephyr RTOS, foi criada uma arquitetura orientada a eventos que gerencia recursos de alto consumo, como o *clock* de alta frequência, apenas quando necessário. O projeto superou a falta de documentação da placa ProMicro nRF52840 por meio da criação de uma configuração de *hardware* customizada. A validação experimental quantificou o impacto das ferramentas de *debug*, cuja desativação reduziu o consumo em repouso de 1,94 mA para 1,10 mA. Também foi identificado um custo energético fixo e elevado na inicialização, causado pelo *bootloader* da placa. Conclui-se que, embora a estratégia de *software* seja eficaz, o *hardware* de prototipagem limita o desempenho, tornando o projeto de uma PCB customizada um requisito para uma aplicação final viável.

Palavras-chave

Internet das Coisas; Energy Harvesting; Gerenciamento de Energia; Baixo Consumo de Energia; Zephyr RTOS; nRF52840; IEEE 802.15.4; Device Tree.

Abstract

Soares, Jerônimo Augusto; Branco, Adriano (Advisor). **Development of Batteryless IoT System**. Rio de Janeiro, 2025. 42p. Final Project – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

This work details the development of an ultra-low-power firmware for an IoT system, aiming for battery-less operation powered by ambient energy (Energy Harvesting). Using the nRF52840 SoC and the Zephyr RTOS, an event-driven architecture was developed that manages high-consumption resources, such as the high-frequency clock, on an on-demand basis. The project overcame the lack of documentation for the ProMicro nRF52840 board by creating a custom hardware configuration. Experimental validation quantified the impact of debugging tools, as disabling them reduced the idle current from 1.94 mA to 1.10 mA. A fixed and high energy cost at startup, caused by the board's bootloader, was also identified. It is concluded that although the software strategy is effective, the prototyping hardware limits performance, making the design of a custom PCB a requirement for a viable final application.

Keywords

Internet of Things (IoT); Energy Harvesting; Power Management; Low Power; Zephyr RTOS; nRF52840; IEEE 802.15.4; Device Tree.

Sumário

Lista de Abreviaturas

IoT – Internet of Things
EH – Energy Harvesting
SoC – System on a Chip
RTOS – Real-Time Operating System
PCB – Printed Circuit Board
CPU – Central Processing Unit
FPU – Floating-Point Unit
RAM – Random Access Memory
RTC – Real-Time Clock
ADC – Analog-to-Digital Converter
SPI – Serial Peripheral Interface
I2C – Inter-Integrated Circuit
UART – Universal Asynchronous Receiver-Transmitter
WPAN – Wireless Personal Area Network
PHY – Physical Layer
MAC – Medium Access Control
CSMA-CA – Carrier Sense Multiple Access with Collision Avoidance
BLE – Bluetooth Low Energy
ISM – Industrial, Scientific, and Medical
AES – Advanced Encryption Standard
ANT – Adaptive Network Topology
IDE – Integrated Development Environment
SDK – Software Development Kit
USB – Universal Serial Bus
JTAG – Joint Test Action Group
GPIO – General-Purpose Input/Output
DTS – Device Tree Source
HFCLK – High-Frequency Clock
PPK2 – Power Profiler Kit II
VSCode – Visual Studio Code

1
Introdução

1.1
A Ascensão da Internet das Coisas

A Internet das Coisas (IoT) revolucionou a forma como interagimos com o ambiente, com bilhões de dispositivos conectados em aplicações que vão desde automação residencial até monitoramento industrial. Com o advento da IoT, surgiram dispositivos capazes de responder a comandos e transmitir dados, auxiliando tanto em atividades cotidianas quanto em pesquisas.

O crescimento da Internet das Coisas (IoT) continua em ritmo acelerado, solidificando sua onipresença no cenário tecnológico global. De acordo com dados recentes da plataforma de pesquisa IoT Analytics [Sinha 2024], o número de dispositivos IoT conectados atingiu 16,6 bilhões ao final de 2023, um aumento de 15% em relação ao ano anterior. As projeções para o final de 2024 indicavam uma expansão contínua de 13%, elevando o total para 18,8 bilhões de dispositivos. Olhando para o futuro, a expectativa é que o ecossistema de IoT continue sua expansão massiva, com estimativas apontando para 40 bilhões de dispositivos conectados até 2030, como ilustrado na Figura 1.1.

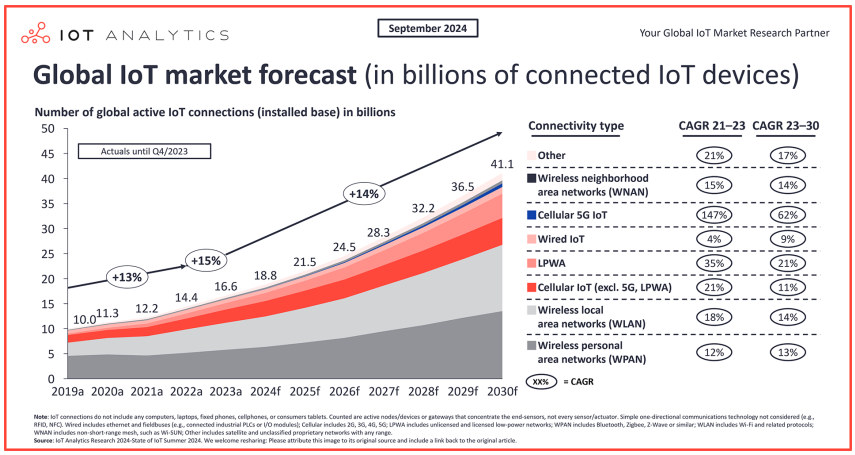

Figura 1.1: Previsão de crescimento do mercado de dispositivos IoT conectados globalmente. [Sinha 2024]

Em geral, esses dispositivos utilizam eletricidade para funcionar, obtida de fontes conectadas à rede elétrica por cabos ou de baterias e pilhas que necessitam de recarga ou substituição constante. Esse aumento no número de dispositivos destaca a urgência de se encontrar soluções energéticas sustentáveis para alimentá-los, e a dependência de fontes de energia tradicionais

representa um dos principais desafios para a expansão e sustentabilidade dos sistemas IoT.

1.2 O Problema da Dependência de Baterias

A proliferação massiva de dispositivos IoT, com projeções que apontam para dezenas de bilhões de unidades conectadas na próxima década [Sinha 2024], trouxe consigo um desafio fundamental: a alimentação energética. Embora a solução predominante continue sendo o uso de baterias, essa abordagem apresenta uma série de obstáculos que limitam o potencial da IoT [Chatterjee et al. 2023]. A necessidade de alimentar essa vasta rede de dispositivos é, de fato, considerada um dos maiores desafios para a concretização da visão da IoT [Jayakumar et al. 2014].

A dependência de baterias impõe severos desafios operacionais e financeiros. A substituição frequente em implementações de grande escala não é apenas cara, mas muitas vezes logisticamente inviável, especialmente em locais de difícil acesso [Jayakumar et al. 2014]. Pesquisas na área frequentemente relatam uma "experiência agitada com substituições frequentes de bateria" [Afanasov et al. 2020].

Além dos entraves operacionais, o impacto ambiental gerado pelo ciclo de vida das baterias representa uma crise iminente. O alto crescimento de dispositivos IoT levará a um descarte igualmente massivo de baterias, com estimativas de que 78 milhões de unidades serão descartadas diariamente até 2025 [EnABLES 2022]. O descarte inadequado representa um risco grave de contaminação, pois os metais pesados tóxicos presentes nas baterias podem poluir de forma irreparável o solo e os lençóis freáticos [18]. Agravando o problema, a taxa de reciclagem de materiais críticos é extremamente baixa; por exemplo, o lítio, componente chave nas baterias mais comuns para IoT, possui uma taxa de reciclagem ao fim da vida útil inferior a 1% [Statista 2020].

1.3 Motivação e Proposta de Solução

Diante do cenário de insustentabilidade e dos desafios logísticos impostos pelas baterias, a busca por fontes de energia alternativas tornou-se um campo de intensa pesquisa acadêmica e industrial [Ku et al. 2016]. A necessidade de sistemas IoT autônomos energeticamente é evidente em diversas aplicações, tais como:

- Monitoramento ambiental em áreas remotas;
- Sensores estruturais em pontes e edifícios;

– Agricultura de precisão;

– Acompanhamento médico hospitalar.

A metodologia de *Energy Harvesting* (EH), ou coleta de energia, surge como uma alternativa promissora para viabilizar essas aplicações. Essa abordagem consiste em utilizar um subsistema autônomo capaz de capturar energia de fontes ambientais — como luz solar, calor, vibrações ou radiofrequência (RF) — e convertê-la em energia elétrica [Chatterjee et al. 2023, Elahi et al. 2020]. A tecnologia EH é vista como uma "alternativa direta à operação alimentada por bateria", com o potencial de "estender a vida útil das implantações, bem como reduzir a substituição de baterias e os custos gerais de manutenção" [Afanasov et al. 2020, Bakar e Hester 2018].

Contudo, a ausência de uma fonte contínua de energia apresenta desafios significativos, destacando problemas como a computação intermitente, a consistência de dados e a necessidade de um gerenciamento de energia eficiente [Lucia et al. 2017].

Este trabalho se insere, portanto, no esforço de viabilizar sistemas IoT verdadeiramente autônomos e sustentáveis. A proposta de desenvolvimento de um sistema IoT sem o uso de bateria está alinhada com as linhas de pesquisa do laboratório GIST da PUC-Rio, especialmente na área de *Energy Harvesting*, onde este projeto foi realizado.

1.4 Objetivos

1.4.1 Objetivo Geral

Com esses desafios em mente, este projeto tem como objetivo explorar e avaliar possíveis aplicações do *Energy Harvesting*, desenvolvendo e validando um sistema IoT capaz de operar exclusivamente com energia coletada do ambiente.

1.4.2 Objetivos Específicos

1. Implementar *firmware* otimizado para ultra baixo consumo no microcontrolador nRF52840;

2. Desenvolver módulo de comunicação sem fio baseado no protocolo IEEE 802.15.4 com foco em eficiência energética;

3. Quantificar e validar o consumo energético do sistema em diferentes modos de operação;

4. Avaliar a PCB Pro Micro nRF52840 quanto à viabilidade como plataforma para aplicações com Energy Harvesting.

1.5 Desafios do Projeto

Durante o desenvolvimento deste trabalho, diversos desafios técnicos foram enfrentados, cuja superação representa contribuições significativas:

- **Documentação escassa:** A PCB ProMicro NRF52840 não possui suporte oficial, exigindo engenharia reversa;
- **Limitações de *debugging*:** Ferramentas de depuração impactam significativamente o consumo;
- **Gerenciamento de energia:** Complexidade na sincronização de periféricos para minimizar consumo.

1.6 Contribuições

Este trabalho apresenta as seguintes contribuições técnicas:

- **Port de Software para Hardware Não Suportado:** Realização do porte do Zephyr RTOS para a PCB ProMicro NRF52840, o que incluiu engenharia reversa para criar uma configuração de placa customizada e resolver conflitos com o bootloader existente;
- **Estratégia de dupla compilação:** Proposta e validação de uma metodologia com duas configurações de *build* (`debug` e `low-power`) para isolar o impacto energético das ferramentas de depuração e permitir medições precisas de consumo;
- **Quantificação de Sobrecargas Energéticas:** Análise e medição do custo energético de componentes de software auxiliares, como o *overhead* do *stack* USB de depuração e o consumo fixo de um *bootloader* comercial, demonstrando seu impacto crítico no orçamento de energia de sistemas de baixo consumo.

1.7 Organização do Documento

Este documento está estruturado em seis capítulos, que abordam desde a contextualização do problema até a análise dos resultados e as propostas para trabalhos futuros.

No **Capítulo 2**, é apresentada a fundamentação teórica que serve de base para este trabalho. São descritas as principais tecnologias utilizadas, incluindo as características do SoC nRF52840, os protocolos de comunicação sem fio de baixo consumo e a arquitetura do sistema operacional de tempo real Zephyr RTOS.

O **Capítulo 3** detalha a análise do problema e a arquitetura da solução proposta. São discutidas as diretrizes de projeto, as restrições técnicas enfrentadas, e é apresentada a arquitetura de *software* orientada a eventos, projetada para minimizar o consumo de energia.

Em seguida, o **Capítulo 4** descreve o processo de desenvolvimento e implementação do *firmware*. Este capítulo aborda desde a configuração do ambiente e a superação dos desafios de compatibilidade com a PCB ProMicro, até a implementação das estratégias de gerenciamento de energia e a metodologia de compilação dual para depuração e operação de baixo consumo.

O **Capítulo 5** apresenta a validação experimental e a análise dos resultados obtidos. Nele, são detalhadas a metodologia de medição de consumo, a análise comparativa do perfil energético da aplicação e a discussão sobre comportamentos não planejados, como o impacto do *bootloader* no consumo inicial.

Finalmente, o **Capítulo 6** consolida as conclusões do trabalho, realizando uma retrospectiva do projeto e destacando as contribuições técnicas. Ao final, são apresentadas sugestões e direções para trabalhos futuros, visando a expansão e o aprofundamento da pesquisa.

2
Fundamentação Teórica

Este capítulo apresenta a base teórica necessária para o entendimento do projeto, detalhando os três componentes centrais da solução: o *hardware*, representado pelo SoC *nRF52840*; os padrões de comunicação sem fio de baixo consumo, como o *IEEE 802.15.4*; e a plataforma de *software*, baseada no *Zephyr RTOS*.

2.1
Nordic nRF52840

O dispositivo escolhido para este projeto foi o SoC nRF52840 da Nordic Semiconductor. Dentre as características do nRF52840, destacam-se:

- CPU ARM Cortex-M4 de 64 MHz com FPU
- 1 MB de memória Flash
- 256 KB de RAM
- Suporte nativo para múltiplos protocolos: Bluetooth Low Energy, Bluetooth mesh, IEEE 802.15.4, Thread, Zigbee, ANT e protocolos proprietários 2.4 GHz
- Modos de baixo consumo com corrente < 5 µA
- Periféricos otimizados: RTC de baixo consumo, ADC de 12 bits, interfaces SPI/I2C/UART

A escolha deste SoC foi motivada principalmente pelo suporte a protocolos de transmissão de dados por rádio de baixo consumo e pela disponibilidade de modos de operação ultra-baixo consumo, tornando-o candidato viável para aplicações com *Energy Harvesting* [nRF52840 - Nordic Semiconductor].

2.2
Protocolos de Comunicação sem Fio

Nesta seção serão abordados alguns protocolos de comunicação suportados pelo SoC nRF52840.

2.2.1 IEEE 802.15.4

O IEEE 802.15 estabelece normas para redes de área pessoal sem fio (WPAN), desempenhando um papel importante na conectividade de dispositivos de baixo consumo de energia. Essas diretrizes garantem interoperabilidade e eficiência, especialmente no contexto de redes que operam com baixas taxas de transmissão de dados, como sensores sem fio, dispositivos de monitoramento e automação residencial [Melo 2017].

O padrão IEEE 802.15.4 foi projetado para redes de baixa taxa de transmissão de dados, como sensores sem fio, dispositivos de monitoramento e automação residencial. Ele define especificações para as camadas PHY (Physical Layer) e MAC (Medium Access Control). A camada PHY é responsável pela transmissão e recepção de dados através do meio físico. O modelo IEEE 802.15.4 opera em três bandas de frequência não licenciadas: 868 MHz (Europa), 915 MHz (América do Norte) e 2,4 GHz (uso mundial), sendo esta última a banda de operação do SoC nRF52840. Essas frequências permitem a comunicação sem fio em diferentes regiões geográficas.

A camada MAC controla o acesso à camada PHY, minimizando colisões e garantindo uma comunicação eficiente. Ela utiliza o protocolo CSMA-CA (Carrier Sense Multiple Access with Collision Avoidance) para evitar colisões entre transmissões. Além disso, a camada MAC é responsável pelo endereçamento dos dispositivos, incluindo o transmissor e o receptor.

O padrão IEEE 802.15.4 é base para as especificações Zigbee e Thread, suportadas pelo nRF52840.

2.2.2 Zigbee

O Zigbee é um protocolo de comunicação sem fio, com topologia de malha, que usa como base o IEEE 802.15.4 e é utilizado em aplicações IoT. Ele se destaca por sua capacidade de transmitir pequenos pacotes de dados com baixo consumo de energia [Zigbee | Complete IOT Solution - CSA-IOT].

2.2.3 Thread

O Thread é um protocolo de rede sem fio de baixo consumo de energia que também tem como base o padrão IEEE 802.15.4. Com topologia de malha, o Thread usa endereçamento IP, sendo compatível com IPv6, com acesso à nuvem e criptografia AES, além de garantir a ausência de ponto único de falha nos sistemas que o utilizam [OpenThread].

2.2.4 ANT

ANT (Adaptive Network Topology) é um protocolo de comunicação sem fio de ultra-baixa potência responsável por enviar informações de um dispositivo para outro de forma robusta e flexível, podendo operar com topologia ponto a ponto, estrela ou malha. Um dispositivo ANT pode ser configurado para passar longos períodos de suspensão, podendo acordar brevemente para se comunicar e retornar ao modo de suspensão [ANT / ANT+ Defined - THIS IS ANT].

2.2.5 Bluetooth Low Energy

O Bluetooth Low Energy (BLE) é uma variante do Bluetooth projetada para operação com baixo consumo de energia. Bastante presente em sensores, dispositivos vestíveis e rastreadores de *fitness*, o BLE utiliza a banda ISM de 2,4 GHz e 40 canais para transmissão. A comunicação por meio do BLE aceita as topologias ponto a ponto, estrela e malha [Bluetooth Technology Overview | Bluetooth® Technology Website].

2.2.6 2.4 GHz proprietary

O SoC nRF52840 possui suporte para o uso do protocolo proprietário da Nordic Semiconductor que opera em 2,4 GHz, podendo operar de forma simultânea com o Bluetooth Low Energy ou outros protocolos suportados [2.4 GHz proprietary - nordicsemi.com].

2.3 Zephyr RTOS

Para o desenvolvimento do *firmware*, optou-se pelo Zephyr RTOS, um sistema operacional de tempo real (RTOS) de código aberto, recomendado pela fabricante do nRF52840, mantido pela Linux Foundation, projetado para ser escalável, seguro e flexível [The Zephyr Project 2025]. Sua arquitetura modular permite que ele seja executado tanto em microcontroladores com recursos extremamente limitados, quanto em sistemas multi-core complexos [The Zephyr Project 2025]. A flexibilidade é um de seus pilares, com suporte nativo para mais de 750 placas de desenvolvimento e centenas de sensores, o que facilita a prototipagem e a portabilidade de aplicações entre diferentes *hardwares* [The Zephyr Project 2025].

Um dos focos principais do Zephyr é o gerenciamento eficiente de energia, um requisito crucial para dispositivos alimentados por fontes intermitentes. O objetivo do RTOS é utilizar a menor quantidade de energia possível, preservando a capacidade de resposta do sistema[The Zephyr Project 2025]. Isso é alcançado através de um *kernel tickless*, que minimiza o tempo de atividade da CPU, e de políticas de energia que permitem tanto ao sistema quanto aos periféricos entrarem em estados de baixo consumo de forma independente e configurável pelo desenvolvedor[The Zephyr Project 2025].

Este sistema é suportado pela Nordic Semiconductor por meio de uma extensão para o Visual Studio Code (VSCode), que auxilia no desenvolvimento com seus chips, possibilitando o uso de bibliotecas, a compilação de código e o carregamento de *firmware*.

3
Análise do Problema e Arquitetura Proposta

Com a base teórica estabelecida, este capítulo detalha a tradução dos objetivos do projeto em uma solução prática. Serão apresentadas as diretrizes e restrições que nortearam o desenvolvimento, os desafios técnicos de hardware e software enfrentados e, por fim, a arquitetura orientada a eventos proposta para atender ao requisito fundamental de ultra-baixo consumo energético.

3.1
Diretrizes de Projeto e Restrições Técnicas

A tradução dos objetivos gerais, definidos na seção 1.4, para uma implementação prática foi guiada por um conjunto de diretrizes de projeto e restrições técnicas que moldaram a arquitetura final do sistema.

A diretriz funcional primária era clara: o *firmware* deveria ser capaz de executar um ciclo de operação completo, consistindo em **ler um sensor periodicamente**, **processar essa informação** e **transmiti-la** usando um protocolo sem fio de baixo consumo, como o IEEE 802.15.4.

No entanto, a diretriz não-funcional mais crítica, que se sobrepôs a todas as outras, foi a **minimização extrema do consumo de energia**. Este princípio foi o critério dominante na escolha da arquitetura de *software*, na gestão dos clocks do sistema e na forma como os periféricos foram controlados. O objetivo do projeto não era apenas criar um sistema funcional, mas garantir que ele o fizesse gastando a menor quantidade de energia possível, maximizando o tempo em estados de sono profundo e minimizando a duração e a intensidade dos ciclos ativos.

O desenvolvimento foi conduzido sob restrições bem definidas que influenciaram diretamente o processo. A principal restrição de *hardware* foi o uso da PCB ProMicro nRF52840 [ICBbuy 2024], um componente com **documentação limitada**, o que exigiu esforços de engenharia reversa para viabilizar a compatibilidade com o *firmware*. Por fim, a restrição fundamental do projeto era a **não utilização de fontes de energia convencionais**, como pilhas ou baterias, o que direcionou todas as otimizações para um cenário de operação com *Energy Harvesting*.

3.2
Desafios Identificados

3.2.1 Limitações de *Hardware*

A PCB ProMicro NRF52840 (Figura 3.1) apresentou diversos desafios:

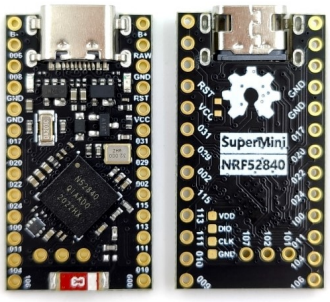

Figura 3.1: PCB ProMicro NRF52840.

- **Falta de suporte oficial:** A placa não está na lista de dispositivos suportados pela Nordic
- ***Bootloader* próprio:** Ocupa espaço significativo de memória (0x0000-0x26000)
- ***Debug* limitado:** Ausência de interface JTAG, apenas USB disponível
- ***Pinout* restrito:** Com acesso a 21 dos 48 pinos GPIO disponíveis no nRF52840, conforme mostrado na Figura 3.2

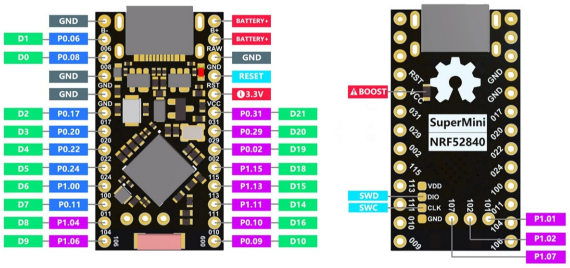

Figura 3.2: *Pinout* da PCB ProMicro NRF52840.

3.2.2 Restrições de Documentação

A documentação encontrada sobre a PCB ProMicro NRF52840 foi escassa:

- Exemplos existentes focados apenas em teclados sem fio, que são alimentados por pilhas ou baterias;
- Ausência de configuração para Zephyr RTOS;
- Necessidade de engenharia reversa para determinar configurações.

3.2.3 Incompatibilidades de *Software*

O desenvolvimento inicial revelou incompatibilidades significativas:

- *Firmware* compilado com configurações padrão não executava;
- Conflito entre endereçamento de memória do Zephyr e *bootloader*;
- Ferramentas de *debug* (USB/*logs*) impactavam drasticamente o consumo;
- *Bootloader* UF2 com comportamento não documentado.

3.3 Arquitetura Proposta

3.3.1 Visão Geral da Solução

A arquitetura desenvolvida segue o paradigma de computação orientada a eventos, essencial para minimizar o consumo energético. O ciclo de operação, ilustrado na Figura 3.3, consiste em:

1. ***Sleep*:** Estado de ultra baixo consumo;
2. ***Wake*:** Despertar por evento do RTC;
3. ***Measure*:** Leitura rápida do ADC;
4. ***Transmit*:** Envio dos dados via IEEE 802.15.4;
5. ***Return to Sleep*:** Retorno imediato ao modo de baixo consumo.

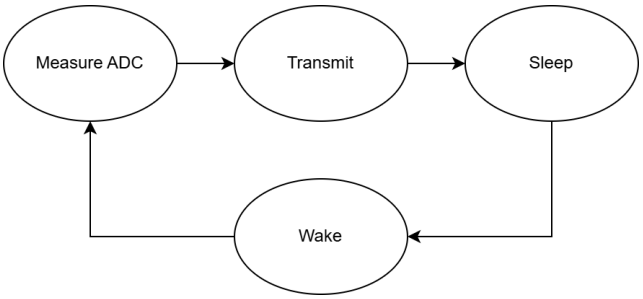

Figura 3.3: Ciclo de operação da aplicação.

3.3.2 Arquitetura de *Software*

A arquitetura de *software*, conforme a Figura 3.4 foi estruturada em módulos:

- **Gerenciador de Energia:** Controla estados de consumo do *clock* de alta frequência (HFCLK)
- ***Driver* ADC:** Interface otimizada para leituras rápidas;
- ***Stack* IEEE 802.15.4:** Implementação através do Zephyr;
- **Agendador RTC:** Desperta o sistema periodicamente;
- **Aplicação Principal:** Orquestra o fluxo de execução.

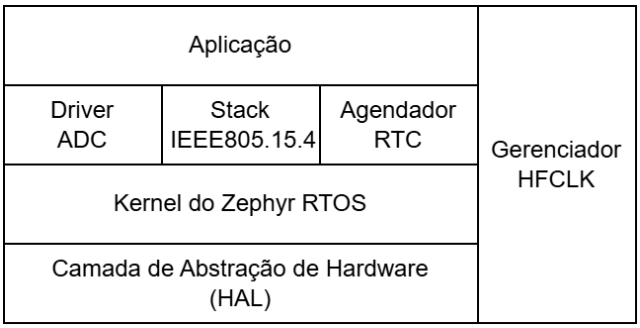

Figura 3.4: Diagrama de camadas de *software*.

4
Desenvolvimento e Implementação

Este capítulo descreve a materialização da arquitetura proposta, detalhando o processo de implementação do firmware. Serão abordadas as etapas práticas, desde a configuração do ambiente de desenvolvimento e a superação dos desafios de compatibilidade com a PCB ProMicro, até a codificação das rotinas de gerenciamento de energia e comunicação.

4.1
Configuração do Ambiente de Desenvolvimento

A primeira etapa do desenvolvimento prático consistiu na preparação do ambiente de trabalho e na configuração da plataforma de hardware para a execução do firmware.

4.1.1
Ferramentas e Tecnologias

O ambiente de desenvolvimento foi configurado com:

- **IDE:** Visual Studio Code com a extensão nRF Connect for VS Code
- **SDK:** nRF Connect SDK v3.0.2 (baseado em Zephyr RTOS)
- **Ferramentas auxiliares:**
  - Python *scripts* para conversão `.hex` → `.uf2`
  - Nordic Power Profiler Kit II para análise de consumo
  - Terminal serial para *debug* (quando aplicável)

A escolha do Zephyr RTOS foi fundamental por seu suporte nativo a:

- Gerenciamento de energia integrado;
- APIs de baixo nível para controle fino de *hardware*;
- Modelo de execução baseado em *threads* e *work queues*;
- Suporte oficial da Nordic Semiconductor.

4.1.2 Desafios na Configuração da PCB ProMicro

Para validar a funcionalidade da PCB ProMicro e testar sua operação, foi desenvolvido um código `blinky all GPIO`, cuja finalidade é acionar de maneira intermitente todos os pinos de entrada e saída da placa. Este procedimento visa verificar o correto funcionamento dos pinos e testar a compilação e gravação de um *firmware* no dispositivo.

4.1.2.1 Identificação do Problema e Descoberta da Causa Raiz

A primeira abordagem para compilar *firmware* para a ProMicro utilizou a configuração da placa nRF52840 DK. Embora a compilação fosse bem-sucedida, o *firmware* não executava na PCB. Para validar o *hardware*, testou-se com a Arduino IDE [Arduino - Home], que gerou *firmware* funcional após conversão para formato `.uf2`. Esta discrepância indicava um problema de configuração, não de *hardware*.

Após análise detalhada, identificou-se a origem do problema: a PCB ProMicro possui um *bootloader* pré-gravado em sua memória. A função primária deste *bootloader* é facilitar o desenvolvimento, permitindo que um novo firmware (no formato `.uf2`) seja gravado de forma extremamente simples via USB (arrastando um arquivo), um processo que é ativado ao conectar o pino de `RESET` ao `GND` duas vezes em menos de meio segundo.

É importante notar que uma alternativa seria utilizar um programador externo, como um J-Link, para gravar o firmware diretamente na memória do microcontrolador. No entanto, esse processo sobrescreveria o bootloader existente, eliminando permanentemente a conveniente funcionalidade de gravação via USB. Portanto, a decisão de engenharia foi trabalhar com o bootloader, em vez de removê-lo.

Apesar dessa grande conveniência, a presença deste *bootloader* impôs o desafio técnico central. Verificou-se que ele ocupa os endereços de memória de 0x0000 a 0x26000. O *firmware* gerado pelo Zephyr, por padrão, tentava ocupar esta mesma região, causando um conflito que impedia a execução [Winans 2020]. A solução, portanto, exigia a criação de uma configuração de *hardware* customizada para a placa.

4.1.2.2 Solução com *Device Tree* e Configuração Customizada

A customização de *hardware* no Zephyr é realizada através do sistema ***Device Tree***. Trata-se de uma estrutura de dados que descreve a configuração

do *hardware* disponível em um sistema alvo, como microcontroladores, pinos e periféricos. O principal objetivo do *Device Tree* é desacoplar o código da aplicação dos detalhes específicos do *hardware*, permitindo que o *software* se refira a nós lógicos (ex: `&led0`, `&adc`), enquanto o mapeamento para o *hardware* físico é definido nos arquivos de configuração.

Com base nesse conceito, e utilizando a configuração da placa `adafruit_itsybitsy_nrf52840` como referência, foi desenvolvido um arquivo de descrição de hardware `ProMicro_nrf52840.dts`, específico para a PCB em questão. O trecho a seguir demonstra como as partições de memória (partitions) foram ajustadas para coexistir com o *bootloader*, definindo que a partição da aplicação (`code_partition`) iniciasse no endereço 0x26000:

```
// Trecho do arquivo ProMicro_nrf52840.dts
&flash0 {
    partitions {
        compatible = "fixed-partitions";
        #address-cells = <1>;
        #size-cells = <1>;

        /* Reservado para o SoftDevice,
        embora não usado pelo bootloader UF2 */
        reserved_partition_0: partition@0 {
            label = "SoftDevice";
            reg = <0x00000000 DT_SIZE_K(152)>;
        };
        /* Início da aplicação em 0x26000 para evitar conflito
        com o bootloader */
        code_partition: partition@26000 {
            label = "Application";
            reg = <0x00026000 DT_SIZE_K(796)>;
        };

        storage_partition: partition@ed000 {
            label = "storage";
            reg = <0x000ed000 DT_SIZE_K(28)>;
        };

        boot_partition: partition@f4000 {
            label = "UF2";
            reg = <0x000f4000 DT_SIZE_K(48)>;
```

```
        };
    };
};
```

Com a configuração de base da placa resolvida, a customização para aplicações específicas, como a habilitação de periféricos, é realizada através de arquivos de ***overlay*** (`.overlay`). Eles permitem modificar ou estender a configuração do `.dts` sem alterá-lo diretamente. Para habilitar o conversor analógico-digital (ADC), por exemplo, pode-se criar o seguinte *overlay*:

```
/* Arquivo de overlay: ProMicro_nrf52840_nrf52840.overlay */
&adc {
    status = "okay";
    #address-cells = <1>;
    #size-cells = <0>;

    channel@0 {
        reg = <0>;
        zephyr,gain = "ADC_GAIN_1_6";
        zephyr,reference = "ADC_REF_INTERNAL";
        zephyr,acquisition-time = <ADC_ACQ_TIME_DEFAULT>;
        zephyr,input-positive = <NRF_SAADC_AIN0>;
        zephyr,resolution = <12>;
    };
};
```

Este *overlay* ativa o periférico ADC (alterando seu status para "okay") e configura um de seus canais, definindo parâmetros de aquisição e mapeando-o para um pino físico.

4.2 Implementação do *Firmware*

A implementação do *firmware* foi desenvolvida sobre o Zephyr RTOS, materializando a arquitetura orientada a eventos e de baixo consumo descrita anteriormente. O objetivo principal foi orquestrar os diferentes periféricos (RTC, ADC, Rádio) de forma a mantê-los em estado de baixo consumo na maior parte do tempo, ativando-os apenas quando estritamente necessário. A seguir, são detalhados os principais componentes de *software* desenvolvidos e as estratégias de implementação adotadas.

4.2.1 Orquestração de Tarefas e Fluxo de Execução

O fluxo de execução da aplicação é centrado em um modelo assíncrono, evitando laços de espera (*busy-waiting*) para maximizar o tempo em que o sistema permanece em modo de baixo consumo. O despertar do sistema é gerenciado pelo módulo `periodic_rtc_task`, uma abstração sobre o RTC de *hardware* que dispara um alarme em intervalos pré-configurados.

Para garantir que a execução de tarefas não ocorra no contexto da interrupção do RTC, o que é uma má prática em sistemas de tempo real, o callback do alarme não executa a lógica de medição e envio diretamente. Em vez disso, ele submete uma tarefa para a fila de trabalho do sistema (*system workqueue*) do Zephyr, através da função `k_work_submit()`. Esta abordagem delega a execução para uma thread de baixa prioridade, mantendo o sistema responsivo e evitando o bloqueio de interrupções críticas. A função de trabalho, `adc_and_send_work_handler`, encapsula todo o ciclo de operação ativo: ligar os recursos necessários, ler o ADC, transmitir os dados e, por fim, desligar os recursos para retornar ao estado de *sleep*.

4.2.2 Módulo de Abstração do Rádio (ieee_sender)

Para simplificar a comunicação e abstrair as complexidades do controle direto do periférico de rádio do microcontrolador, foi desenvolvido o módulo `ieee_sender`. Inspirado nos exemplos de teste de rádio da Nordic, este módulo expõe uma interface minimalista com as funções `ieee_sender_init()` e `ieee_sender_send()`, permitindo que a aplicação principal envie pacotes de dados sem precisar gerenciar registradores de *hardware* ou interrupções de rádio.

Internamente, o módulo configura o rádio para o modo IEEE 802.15.4 e utiliza os atalhos de *hardware* (*shortcuts*) do periférico para desabilitar o rádio automaticamente após a conclusão da transmissão, uma otimização fundamental para a economia de energia. A notificação de que um pacote foi enviado com sucesso é feita de forma assíncrona, através de uma função de callback, liberando a aplicação para aguardar a conclusão em um semáforo (`k_sem_take`) em vez de policiar o estado do *hardware*.

4.2.3 Abstrações para ADC e Tarefas Periódicas

Para completar a arquitetura do *firmware*, dois módulos de abstração foram utilizados para simplificar a interação com o *hardware* e o agendamento

de tarefas, mantendo o código da aplicação principal limpo e focado em sua lógica de orquestração.

O primeiro é a configuração para a leitura do conversor analógico-digital (ADC). Em vez de codificar pinos e configurações diretamente no código-fonte, a definição do canal do ADC a ser utilizado é feita através de uma entrada no arquivo de *overlay* do *Device Tree* (`ProMicro_nrf52840_nrf52840.overlay`). A aplicação utiliza a macro `ADC_DT_SPEC_GET_BY_IDX` fornecida pelo Zephyr para obter a configuração completa do canal em tempo de compilação. Isso torna o *firmware* mais portável e fácil de reconfigurar, pois a alteração do pino do sensor exige apenas uma modificação no *overlay*, sem necessidade de alterar o código C.

O segundo módulo, `periodic_rtc_task`, foi criado para encapsular a lógica de agendamento de tarefas periódicas. Em vez de interagir diretamente com a API de *counter* do Zephyr no `main.c`, este módulo oferece uma interface simplificada para iniciar um temporizador baseado no RTC de baixo consumo que dispara uma função de callback no intervalo desejado. Essa abstração não só organiza melhor o código, mas também facilita a reutilização dessa funcionalidade em outros projetos. Juntos, esses módulos, em conjunto com as estratégias de orquestração e gerenciamento de energia, formam um *firmware* robusto, modular e altamente otimizado para operação de baixíssimo consumo.

4.2.4 Gerenciamento de Energia *On-Demand*: O Controle do HFCLK

Para um dispositivo alimentado por fontes de energia escassas, a simples utilização de estados de baixo consumo (*sleep*) não é suficiente. É imperativo minimizar o consumo também durante os curtos períodos de atividade. A análise de consumo revelou que o *Clock* de Alta Frequência (HFCLK), necessário para a operação do rádio e do ADC, é o principal responsável pelo alto consumo de corrente durante o ciclo de trabalho. Manter este *clock* ativo desnecessariamente drenaria rapidamente a energia armazenada.

Para mitigar este problema, foi implementada uma estratégia de gerenciamento de energia explícita e sob demanda (*on-demand*). Em vez de habilitar o HFCLK na inicialização do sistema, ele permanece desligado por padrão. A ativação ocorre apenas no início da função de trabalho (`adc_and_send_work_handler`), através de uma chamada à função `hfclk_control_request()`. Esta função não só solicita a ativação do *clock*, mas também aguarda sua estabilização antes de permitir que a execução prossiga para a leitura do ADC e a transmissão de rádio.

Imediatamente após a conclusão da transmissão (ou na ocorrência de uma

falha), a última ação executada pela função de trabalho é desabilitar o HFCLK através da chamada `hfclk_control_release()`. O uso de um bloco de limpeza garante que o *clock* seja desligado mesmo em cenários de erro, evitando que o sistema permaneça em um estado de alto consumo. Esta abordagem garante que o componente de maior consumo do sistema esteja ativo apenas por algumas centenas de milissegundos a cada ciclo de 60 segundos, reduzindo drasticamente o consumo médio de energia.

4.2.5 Estratégias de Compilação para Depuração e Baixo Consumo

A depuração de *firmware* em sistemas de baixíssimo consumo apresenta um desafio inerente: as próprias ferramentas utilizadas para observar o sistema podem alterar drasticamente seu comportamento energético. A depuração é tipicamente realizada com o auxílio de ferramentas de *log* que enviam mensagens de status via comunicação serial, como a USB. Contudo, o impacto energético dessas ferramentas precisa ser compreendido e isolado.

Utilizando o Power Profiler Kit II (PPK2), detalhado na Seção 5.2, foi realizada uma análise do consumo da PCB ProMicro. Verificou-se que, com um *firmware* que habilitava a comunicação serial via USB para fins de depuração, o consumo de corrente em estado ocioso era próximo a 2 mA. Este valor, embora baixo para aplicações tradicionais, é excessivamente alto para um dispositivo que visa operar com energia coletada do ambiente. A investigação revelou que a comunicação USB ativa mantinha o *High-Frequency Clock* (HFCLK) sempre ligado, impedindo a validação de rotinas de economia de energia que dependiam de seu desligamento.

Diante dessa constatação, tornou-se evidente a necessidade de separar a compilação para depuração da compilação para operação final. A solução foi criar duas configurações de *build* distintas no Zephyr, que geram firmwares com características diferentes a partir do mesmo código-fonte:

1. Configuração de Depuração (`prj_debug.conf`): Habilita o *stack* USB, o console serial e o sistema de *logs* do Zephyr para permitir a observação detalhada do comportamento da aplicação.

```
# Habilita logs e console via USB
CONFIG_LOG=y
CONFIG_CONSOLE=y
CONFIG_USB_DEVICE_STACK=y
CONFIG_USB_CDC_ACM=y
CONFIG_UART_CONSOLE=y
```

2. Configuração de Baixo Consumo (`prj_low_power.conf`): Desativa completamente os subsistemas de *log*, console e USB, além de outros periféricos não utilizados, para garantir que a medição de energia reflita o comportamento real da aplicação em campo.

```
# Desativa logs, console e USB para baixo consumo
CONFIG_LOG=n
CONFIG_CONSOLE=n
CONFIG_UART_CONSOLE=n
CONFIG_USB_DEVICE_STACK=n

# Desativa periféricos não utilizados
CONFIG_I2C=n
CONFIG_SPI=n
CONFIG_BT=n
```

A adoção dessas configurações foi fundamental para a metodologia de teste do projeto. Com a versão de depuração, foi possível validar a lógica da aplicação (leitura de ADC, envio de rádio) de forma mais eficiente. Posteriormente, compilando com a versão de baixo consumo, o foco foi direcionado exclusivamente para a análise do perfil energético, garantindo que as otimizações de *software*, como o gerenciamento *on-demand* do HFCLK, surtissem o efeito desejado. Nessa etapa, a depuração recorria a métodos de menor impacto, como a sinalização visual por meio de um LED.

5
Validação Experimental e Análise de Resultados

Este capítulo apresenta os resultados quantitativos obtidos a partir da implementação do *firmware*, com foco principal na análise do consumo de energia, um fator crítico para a viabilidade de sistemas alimentados por *Energy Harvesting*. A seguir, será detalhada a metodologia de medição empregada, seguida pela apresentação e discussão dos dados coletados. A análise abrange não apenas o desempenho do sistema em diferentes configurações, mas também as descobertas e desafios não planejados que surgiram durante o processo de validação, como o impacto energético do *bootloader* da placa.

5.1
Metodologia de Medição de Consumo

Para validar de forma rigorosa as estratégias de baixo consumo implementadas no *firmware*, foi estabelecida uma metodologia de medição precisa, capaz de capturar tanto correntes de *sleep* na ordem de microampères (µA) quanto picos de transmissão na ordem de miliampères (mA).

5.1.1
Configuração Experimental

O principal instrumento utilizado para a análise energética foi o **Nordic Semiconductor Power Profiler Kit II (PPK2)**. Esta ferramenta foi escolhida por sua alta resolução e taxa de amostragem (até 100 kS/s), permitindo a caracterização detalhada do perfil de consumo do dispositivo ao longo do tempo.

A montagem experimental, ilustrada na Figura 5.1, consistiu em alimentar a PCB ProMicro diretamente através do PPK2, configurado para fornecer uma tensão estável de 3.3V, simulando uma fonte de energia típica de um sistema de *Energy Harvesting*. O PPK2, por sua vez, estava conectado a um computador para a visualização e gravação dos dados de consumo através do *software* nRF Connect for Desktop.

Todos os testes foram realizados para as duas configurações de compilação distintas, detalhadas no capítulo anterior:

– **`prj_debug.conf`:** Com o subsistema de *log* e a comunicação USB habilitados para depuração funcional.

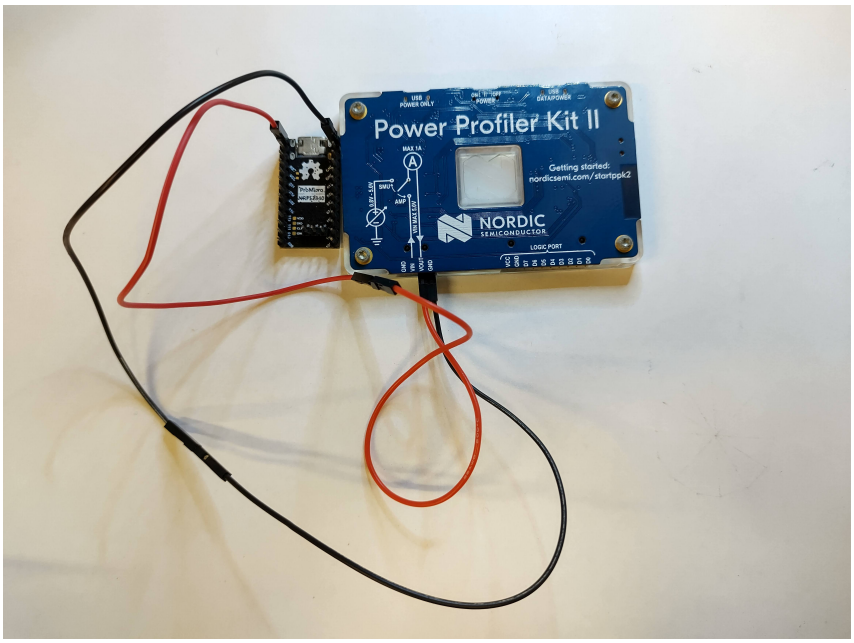

Figura 5.1: *Setup* experimental para medição de consumo, mostrando a PCB ProMicro conectada e alimentada pelo Power Profiler Kit II (PPK2).

– **`prj_low_power.conf`:** Com todas as funcionalidades de depuração desativadas para a medição do consumo de energia em um cenário mais próximo do real.

5.2 Análise de Consumo de Energia

5.2.1 Consumo em Estado Ocioso: O Impacto da Depuração

A primeira e mais impactante descoberta da análise foi a drástica diferença no consumo de corrente de base (em estado ocioso ou *sleep*) entre as duas configurações de compilação. Este resultado validou a necessidade da estratégia de dupla compilação descrita no capítulo anterior.

Na compilação com a configuração `prj_debug.conf`, que mantém o *stack* USB e os *logs* ativos, o consumo médio medido foi de **1,94 mA**. Em contraste, ao utilizar a configuração `prj_low_power.conf`, que desativa estes subsistemas, o consumo médio foi reduzido para **1,10 mA**, representando uma redução de aproximadamente 43%. As Figuras 5.2 e 5.3 ilustram essa diferença de forma clara.

Essa diferença substancial é atribuída diretamente à ativação do *stack* USB na versão de depuração. Esse componente não apenas consome energia por si mesmo, mas, de forma mais crítica, impede que o *Clock* de Alta Frequência (HFCLK) seja desabilitado, mantendo o sistema em um estado de consumo

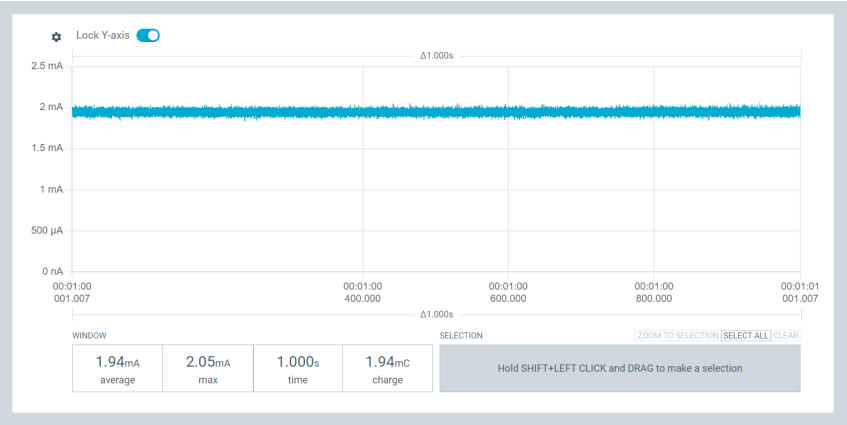

Figura 5.2: Modo *Debug*: consumo médio de 1,94 mA.

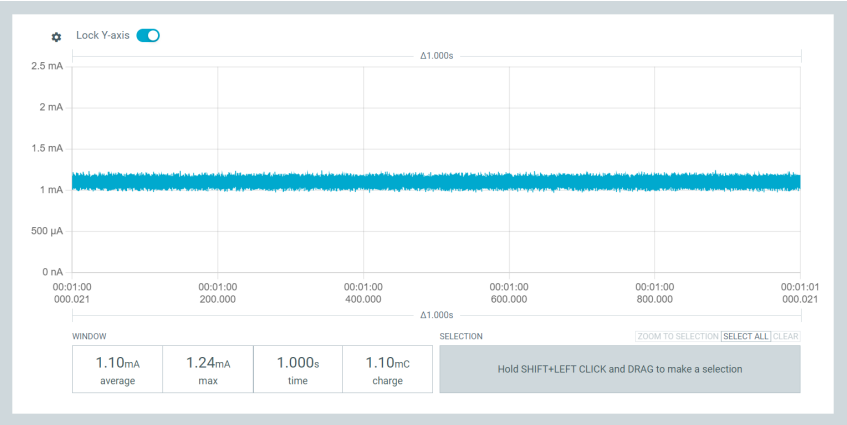

Figura 5.3: Modo *Low Power*: consumo médio de 1,10 mA.

consideravelmente mais alto, mesmo quando a aplicação principal está em espera.

5.2.2 Investigação da Corrente de Fuga Residual

Apesar da redução significativa, um consumo de 1,10 mA em estado de repouso ainda é excessivamente alto para uma aplicação que almeja operar com *Energy Harvesting*, estando ordens de magnitude acima do valor teórico de 3,16 µA especificado no *datasheet* do nRF52840 [Nordic Semiconductor 2023]. Isso motivou uma investigação sobre as possíveis causas para essa corrente de fuga residual.

A principal hipótese é o **consumo de componentes externos na PCB ProMicro**, que não foi projetada com foco em ultra baixo consumo e possui documentação limitada. É provável que componentes como reguladores de tensão ou outros circuitos passivos contribuam para um consumo "parasita" que não pode ser controlado pelo *firmware* do SoC.

Adicionalmente, explorou-se o potencial de otimizações de **software** mais

agressivas. Testes com o modo de desligamento total do sistema do Zephyr (`sys_poweroff`) resultaram em um consumo de base de aproximadamente **808 µA** (conforme a Figura 5.4), ainda alto, mas que demonstra haver margem para otimizações de *software* mais profundas na configuração dos modos de sono do sistema.

Figura 5.4: *System poweroff*: consumo médio de 808 µA.

5.2.3 Perfil de Consumo do Ciclo de Operação Ativo

A análise do ciclo ativo, quando o dispositivo desperta para ler o sensor e transmitir os dados, também revelou diferenças notáveis entre os dois modos, como pode ser visto nas Figuras 5.5 e 5.6.

No modo ***Debug*** (Figura 5.5), o pico de atividade dura aproximadamente 9,8 ms, com um consumo médio de 7,80 mA. Em contraste, no modo ***Low Power*** (Figura 5.6), o mesmo ciclo de trabalho é executado em apenas 1,2 ms, com um consumo médio de 4,65 mA. Essa otimização drástica no tempo e na corrente ocorre porque a versão de baixo consumo não possui o *overhead* de processamento e comunicação do sistema de *logs*, executando apenas as instruções essenciais.

5.3 Análise de Comportamentos Não Planejados

Além da análise dos ciclos de operação planejados, a medição detalhada do consumo revelou um comportamento não trivial e de grande impacto para o orçamento energético do sistema: um período de alto consumo que era incontrolável pela aplicação.

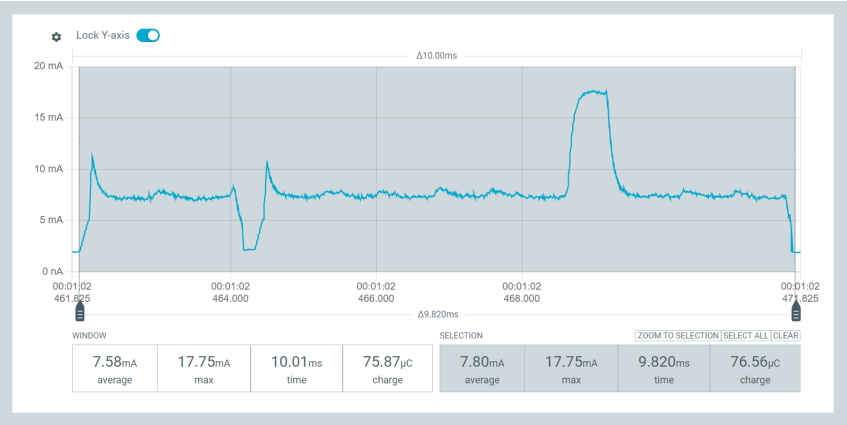

Figura 5.5: Ciclo ativo no modo *Debug*.

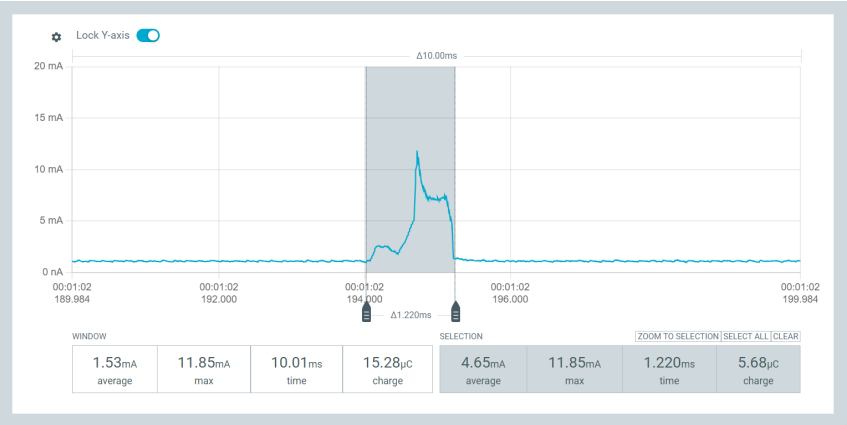

Figura 5.6: Ciclo ativo no modo *Low Power*.

5.3.1 O Custo Energético do *Bootloader*

Finalmente, a análise do momento de inicialização do dispositivo revelou um comportamento incontrolável pela aplicação. Foi verificado que, imediatamente após o dispositivo ser energizado, há um período de consumo elevado com duração total de aproximadamente **900 ms** antes que a primeira tarefa do *firmware* seja executada. Dentro deste intervalo, os primeiros **510 ms** apresentam um consumo de corrente particularmente intenso e oscilatório, como ilustra a Figura 5.7.

A fase de consumo mais intenso, correspondente aos primeiros 510 ms, foi atribuída ao *bootloader*, que executa suas rotinas antes de ceder o controle à aplicação. O intervalo de tempo subsequente, até que a primeira tarefa da aplicação seja executada em 900 ms, representa uma zona de transição. Este atraso adicional pode ser atribuído a uma combinação de fatores: o final do processo do bootloader e o tempo de inicialização do próprio kernel do Zephyr e seus drivers, que ocorre antes que o escalonador execute a primeira instrução da aplicação.

Figura 5.7: Perfil de consumo no primeiro segundo após a energização. A área selecionada mostra o período de atividade do *bootloader*, com duração de 510 ms e consumo médio de 5,13 mA.

Independentemente da causa exata para cada fase, a existência deste **"custo energético de inicialização"** total de 900 milissegundos é uma implicação crítica. Em um cenário de *Energy Harvesting* onde reinicializações por falha de energia podem ser frequentes, este custo fixo e incontrolável pela aplicação precisa ser rigorosamente contabilizado no orçamento energético, pois pode consumir uma porção significativa da energia coletada e impactar diretamente a frequência de operação do dispositivo.

6
Conclusão e Trabalhos Futuros

Este capítulo final consolida os resultados e aprendizados obtidos ao longo do desenvolvimento do projeto. Inicialmente, é apresentada uma retrospectiva do trabalho, recapitulando os desafios, as soluções implementadas e os resultados experimentais. Em seguida, são detalhadas as principais conclusões técnicas e de engenharia derivadas da análise. Por fim, são delineadas as possíveis direções para a continuidade e expansão do projeto em trabalhos futuros.

6.1
Retrospectiva do Projeto

O presente trabalho partiu do desafio fundamental de alimentar a crescente massa de dispositivos da Internet das Coisas (IoT) de forma sustentável, contornando as limitações operacionais e ambientais impostas pelas baterias. O objetivo central foi, portanto, projetar, implementar e validar um *firmware* de ultra-baixo consumo para uma plataforma de *hardware* real (a PCB ProMicro com o SoC nRF52840), visando a operação com fontes de energia intermitentes, como as de *Energy Harvesting*.

Durante o processo, desafios práticos de engenharia foram superados. Destaca-se a necessidade de realizar um trabalho de **engenharia reversa** na PCB ProMicro, que possuía documentação escassa, a fim de mapear o espaço de memória e identificar o endereço de início do *bootloader*. Este processo investigativo foi crucial para a criação de uma configuração de *hardware* customizada no *Device Tree*, que resolveu o conflito de inicialização. Adicionalmente, foi desenvolvida uma metodologia de dupla compilação para isolar o impacto energético das ferramentas de depuração e permitir uma medição precisa do consumo.

Foi desenvolvida sobre o Zephyr RTOS uma arquitetura de *software* orientada a eventos. A implementação se destacou pela criação de módulos de abstração, como o `ieee_sender` para a comunicação de rádio, e por uma estratégia de gerenciamento de energia explícita e *on-demand*, focada no controle do *Clock* de Alta Frequência (HFCLK).

A fase de validação experimental, utilizando o Power Profiler Kit II, quantificou o sucesso das estratégias de otimização, demonstrando uma redução drástica no consumo de energia entre os modos de depuração e de baixo consumo. Adicionalmente, a análise crítica dos resultados permitiu identificar e investigar as fontes de consumo residual e revelou o impacto energético

não trivial do *bootloader* da placa, uma descoberta crucial para o cálculo do orçamento energético total do sistema.

6.2 Conclusões e Atendimento aos Objetivos

A partir da implementação e da análise experimental detalhada, o presente trabalho alcançou os objetivos propostos, permitindo extrair conclusões técnicas e de engenharia que respondem diretamente às metas estabelecidas.

6.2.1 Implementação de Firmware Otimizado para Baixo Consumo

Em atendimento aos objetivos de **implementar um firmware otimizado** e **desenvolver um módulo de comunicação eficiente**, conclui-se que a arquitetura de software orientada a eventos é uma estratégia fundamental e bem-sucedida. A abordagem de manter o sistema em *sleep* e executar tarefas curtas em resposta a eventos (disparo do RTC), combinada com o gerenciamento explícito de recursos caros como o Clock de Alta Frequência (HFCLK), foi validada como uma técnica crucial para minimizar o consumo total de energia em dispositivos que operam com fontes intermitentes.

6.2.2 Quantificação do Consumo Energético e Suas Implicações

O objetivo de **quantificar e validar o consumo energético** do sistema levou a duas das mais significativas conclusões do trabalho:

1. **A configuração de compilação tem um impacto crítico no consumo.** Foi quantificado que a simples ativação do subsistema USB para depuração elevava o consumo de base em 85% (de 1,10 mA para 1,94 mA). Isso demonstra que a validação de consumo deve ser, obrigatoriamente, realizada com uma configuração "limpa", idêntica à da aplicação final, pois as ferramentas de *debug* podem mascarar completamente os resultados das otimizações.

2. **Existem custos energéticos fixos e não triviais no hardware de prototipagem.** A análise também quantificou um "custo de inicialização" de superior a 500 ms imposto pelo bootloader, um fator que não pertence à aplicação mas impacta diretamente seu orçamento energético.

6.2.3 Avaliação da Plataforma de Hardware (PCB ProMicro)

Finalmente, ao **avaliar a viabilidade da PCB ProMicro** para aplicações com *Energy Harvesting*, conclui-se que, embora seja uma excelente plataforma para o desenvolvimento e a depuração da lógica do firmware, suas limitações de hardware impõem um "piso" de consumo elevado. Fatores como componentes passivos na placa e a presença do bootloader tornam impossível atingir os valores teóricos de microampères (µA) especificados no datasheet do SoC.

A análise do bootloader, em particular, destaca um trade-off fundamental na prototipagem: uma alternativa para mitigar o seu custo energético seria utilizar um programador externo, como um J-Link, para sobrescrever completamente a memória do microcontrolador. Contudo, isso eliminaria a conveniente funcionalidade de gravação via USB, que foi essencial para a agilidade no desenvolvimento. Esta escolha reforça que, para um produto final que busque atingir os limites teóricos de consumo, o desenvolvimento de uma Placa de Circuito Impresso (PCB) customizada representa a evolução natural e mais eficaz do projeto.

6.3 Trabalhos Futuros

O presente trabalho estabelece uma base sólida para o desenvolvimento de dispositivos IoT autônomos, validando uma arquitetura de *firmware* e uma metodologia de análise de consumo. A seguir, são propostas algumas direções claras para a sua continuidade e evolução, abordando as limitações identificadas e expandindo o escopo do projeto:

1. **Implementação do Sistema Completo de *Hardware*:** A etapa mais crucial para a validação final do conceito é a integração do *hardware* que não foi escopo deste trabalho. Isso inclui:
   - **Módulo de *Energy Harvesting*:** Projetar ou selecionar um circuito de coleta de energia, composto por uma fonte (como paineis solares) e um circuito gerenciador de carga, para alimentar a PCB e validar a autonomia do sistema em condições reais de iluminação variável.
   - **Seleção do Sensor Final:** Integrar e validar um sensor de temperatura de ultra-baixo consumo, analisando seu ciclo de medição e o impacto real no orçamento energético total do dispositivo.

   - **Uso de Ferramentas de Depuração Avançada:** Adotar um programador/depurador externo, como o J-Link, para permitir a depuração a nível de registradores e a remoção completa do bootloader. Isso não só possibilitaria uma otimização mais fina do firmware, como também eliminaria o custo energético de inicialização, um ganho significativo para o sistema.

2. **Aprofundamento das Otimizações de *Firmware*:** Com base nos resultados obtidos, há espaço para otimizações adicionais no *software*:

   - **Exploração de Modos de Sono Profundo:** Investigar e aplicar configurações de gerenciamento de energia mais agressivas no Zephyr para tentar reduzir a corrente de fuga residual identificada, buscando se aproximar dos valores teóricos de consumo em *sleep* na casa dos microampères.
   - **Validação Funcional da Comunicação:** Realizar testes de campo para caracterizar o desempenho do rádio, como o alcance máximo e a taxa de sucesso de pacotes em diferentes ambientes, para validar a robustez da comunicação.

3. **Desenvolvimento de *Hardware* Customizado:** Conforme concluído, o *hardware* de prototipagem impõe um limite de consumo significativo. O passo mais avançado seria:

   - **Projeto de uma PCB Otimizada:** Desenvolver uma Placa de Circuito Impresso (PCB) customizada para a aplicação. Esta placa seria projetada com foco exclusivo em baixo consumo, utilizando apenas os componentes estritamente necessários, eliminando fontes de consumo parasita (como LEDs e reguladores ineficientes) e otimizando o *layout* para se aproximar dos limites teóricos de consumo do SoC.

7
Referências

[2.4 GHz proprietary - nordicsemi.com]2.4 GHz proprietary - nordicsemi.com. Available from Internet: <https://www.nordicsemi.com/Products/Wireless/2-4-GHz-proprietary>. Citado na página 16.

[Afanasov et al. 2020]AFANASOV, M. et al. Battery-less zero-maintenance embedded sensing at the mithræum of circus maximus. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*. Virtual Event Japan: ACM, 2020. p. 368–381. ISBN 978-1-4503-7590-0. Available from Internet: <https://dl.acm.org/doi/10.1145/3384419.3430722>. Citado 2 vezes nas páginas 10 e 11.

[ANT / ANT+ Defined - THIS IS ANT]ANT / ANT+ Defined - THIS IS ANT. Available from Internet: <https://www.thisisant.com/developer/ant-plus/ant-antplus-defined/>. Citado na página 16.

[Arduino - Home]ARDUINO - Home. Available from Internet: <https://www.arduino.cc/>. Citado na página 23.

[Bakar e Hester 2018]BAKAR, A.; HESTER, J. Making sense of intermittent energy harvesting. In *Proceedings of the 6th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems*. Shenzhen China: ACM, 2018. p. 32–37. ISBN 978-1-4503-6047-0. Available from Internet: <https://dl.acm.org/doi/10.1145/3279755.3279762>. Citado na página 11.

[Bluetooth Technology Overview | Bluetooth® Technology Website] BLUETOOTH Technology Overview | Bluetooth® Technology Website. Available from Internet: <https://www.bluetooth.com/learn-about-bluetooth/tech-overview/>. Citado na página 16.

[Chatterjee et al. 2023]CHATTERJEE, A. et al. Powering internet-of-things from ambient energy: a review. *Journal of Physics: Energy*, vol. 5, no. 2, p. 022001, fev. 2023. ISSN 2515-7655. Publisher: IOP Publishing. Available from Internet: <https://dx.doi.org/10.1088/2515-7655/acb5e6>. Citado 2 vezes nas páginas 10 e 11.

[Elahi et al. 2020]ELAHI, H. et al. Energy Harvesting towards Self-Powered IoT Devices. *Energies*, vol. 13, no. 21, p. 5528, out. 2020. ISSN 1996-1073.

Available from Internet: <https://www.mdpi.com/1996-1073/13/21/5528>. Citado na página 11.

[EnABLES 2022]EnABLES. *Up to 78 million batteries will be discarded daily by 2025, researchers warn*. 2022. Available from Internet: <https://cordis.europa.eu/article/id/430457-up-to-78-million-batteries-will-be-discarded-daily-by-2025-researchers-warn>. Citado na página 10.

[ICBbuy 2024]ICBBUY. *developmentboard:nrf52840 [ICBbuy]*. 2024. Available from Internet: <https://wiki.icbbuy.com/doku.php?id=developmentboard:nrf52840>. Citado na página 18.

[Jayakumar et al. 2014]JAYAKUMAR, H. et al. Powering the internet of things. In *Proceedings of the 2014 international symposium on Low power electronics and design*. La Jolla California USA: ACM, 2014. p. 375–380. ISBN 978-1-4503-2975-0. Available from Internet: <https://dl.acm.org/doi/10.1145/2627369.2631644>. Citado na página 10.

[Ku et al. 2016]KU, M.-L. et al. Advances in Energy Harvesting Communications: Past, Present, and Future Challenges. *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, p. 1384–1412, 2016. ISSN 1553-877X, 2373-745X. Available from Internet: <https://ieeexplore.ieee.org/document/7317504/>. Citado na página 10.

[Lucia et al. 2017]LUCIA, B. et al. Intermittent computing: Challenges and opportunities. August 2017. Citado na página 11.

[Melo 2017]MELO, P. *Padrão IEEE 802.15.4 - A base para as especificações Zigbee, WirelessHart, ISA100.11a e MiWi*. 2017. Available from Internet: <https://embarcados.com.br/padrao-ieee-802-15-4/>. Citado na página 15.

[Nordic Semiconductor 2023]Nordic Semiconductor. *nRF52840 Product Specification v1.11*. [S.l.], 2023. Acesso em: 30 jun. 2025. Available from Internet: <https://docs-be.nordicsemi.com/bundle/ps_nrf52840/attach/nRF52840_PS_v1.11.pdf>. Citado na página 32.

[nRF52840 - Nordic Semiconductor]nRF52840 - Nordic Semiconductor. Available from Internet: <https://www.nordicsemi.com/Products/nRF52840>. Citado na página 14.

[OpenThread]OPENTHREAD. Available from Internet: <https://openthread.io/>. Citado na página 15.

[18]RIBEIRO, J. G. R.; CHAGAS, N. S.; SANTOS, M. F. dos. *O Impacto causado ao meio ambiente pelo descarte incorreto de pilhas e baterias*. Tese (Graduação e Especialização) — Faculdade Una Pouso Alegre - Minas Gerais, jul. 2022. Available from Internet: <https://repositorio.animaeducacao.com.br/handle/ANIMA/24687>. Citado na página 10.

[Sinha 2024]SINHA, S. *State of IoT 2024: Number of connected IoT devices growing 13% to 18.8 billion globally*. 2024. Available from Internet: <https://iot-analytics.com/number-connected-iot-devices/>. Citado 2 vezes nas páginas 9 e 10.

[Statista 2020]STATISTA. *End of life recycling rates of battery metals worldwide, by type*. 2020. Available from Internet: <https://www.statista.com/statistics/1229936/end-of-life-battery-metals-recycling-rates-by-type/>. Citado na página 10.

[The Zephyr Project 2025]The Zephyr Project. *About The Zephyr Project*. 2025. Available from Internet: <https://www.zephyrproject.org/learn-about/>. Citado 2 vezes nas páginas 16 e 17.

[Winans 2020]WINANS, N. *Fixing the Mysterious Broken Bootloader*. 2020. Available from Internet: <https://zmk.dev/blog/2020/10/03/bootloader-fix>. Citado na página 23.

[Zigbee | Complete IOT Solution - CSA-IOT]ZIGBEE | Complete IOT Solution - CSA-IOT. Available from Internet: <https://csa-iot.org/all-solutions/zigbee/>. Citado na página 15.