UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA

GUSTAVO GONÇALVES BORGES

PORTABILIDADE DO CONTIKI-NG PARA ARDUINO UNO COM COMUNICAÇÃO SEM FIO VIA NRF24L01+

RECIFE
2025

GUSTAVO GONÇALVES BORGES

PORTABILIDADE DO CONTIKI-NG PARA ARDUINO UNO COM COMUNICAÇÃO SEM FIO VIA NRF24L01+

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de bacharel em Ciência da Computação.

Orientador(a): Prof. Dr. Renato Mariz de Moraes

Coorientador(a): Prof. Dr. Paulo Filipe Cândido Barbosa

Recife
2025

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Borges, Gustavo Gonçalves.
Portabilidade do Contiki-NG para Arduino Uno com comunicação sem fio via nRF24L01+ / Gustavo Gonçalves Borges. - Recife, 2025.
30 p. : il., tab.

Orientador(a): Renato Mariz de Moraes
Cooorientador(a): Paulo Filipe Cândido Barbosa
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Ciências da Computação - Bacharelado, 2025.
Inclui referências.

1. Sistemas embarcados. 2. Redes ad hoc. 3. Redes de sensores. I. Moraes, Renato Mariz de. (Orientação). II. Barbosa, Paulo Filipe Cândido. (Coorientação). IV. Título.

000 CDD (22.ed.)

GUSTAVO GONÇALVES BORGES

PORTABILIDADE DO CONTIKI-NG PARA ARDUINO UNO COM COMUNICAÇÃO SEM FIO VIA NRF24L01+

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de bacharel em Ciência da Computação.

Aprovado em: 20/08/2025

BANCA EXAMINADORA

Prof. Dr. Renato Mariz de Moraes (Orientador)
Universidade Federal de Pernambuco

Prof. Dr. Adriano Augusto de Moraes Sarmento (Examinador Interno)
Universidade Federal de Pernambuco

Portabilidade do Contiki-NG para Arduino Uno com Comunicação sem Fio via nRF24L01+

Gustavo G. Borges, Paulo F. C. Barbosa, Renato M. de Moraes

[1] Centro de Informática – Universidade Federal de Pernambuco (UFPE)
contato@cin.ufpe.br – 50.740-560 – Recife – PE – Brazil

ggb@cin.ufpe.br, renatomdm@cin.ufpe.br, pfcb@cin.ufpe.br

Abstract. *Efficient communication between embedded devices is a fundamental requirement for applications in Wireless Sensor Networks (WSNs). For this purpose, this work presents the development and implementation of the portability of the Contiki-NG operating system to the Arduino Uno, using the nRF24L01+ wireless transceiver module. The methodology describes the porting of Contiki-NG to the Arduino Uno platform along with the development of specific drivers for the nRF24L01+ radio. The results achieved demonstrate that this portability was successful and serves as a basis for the development of low-cost sensor networks with greater energy autonomy and applicability in scenarios such as environmental monitoring, agriculture, and distributed automation.*

Resumo. *A comunicação eficiente entre dispositivos embarcados é um requisito fundamental para aplicações em redes de sensores sem fio (WSN). Para este fim, este trabalho apresenta o desenvolvimento e implementação de portabilidade do sistema operacional Contiki-NG para o Arduino Uno utilizando o módulo transceptor sem fio nRF24L01+. A metodologia descreve a portabilidade do Contiki-NG para a plataforma Arduino Uno com o desenvolvimento de drivers específicos para o rádio nRF24L01+. Os resultados alcançados mostram que foi possível realizar a portabilidade e serve de base para o desenvolvimento de redes de sensores de baixo custo, com maior autonomia energética e aplicabilidade em cenários como monitoramento ambiental, agricultura e automação distribuída.*

1. Introdução

Nos últimos anos, a crescente demanda por soluções eficientes em Redes de Sensores Sem Fio (RSSF ou WSNs, do inglês *Wireless Sensor Netowrks*) tem impulsionado avanços significativos na área de sistemas embarcados e comunicação de baixo consumo energético. Essas redes são fundamentais em aplicações como monitoramento ambiental, agricultura de precisão, automação industrial e cidades inteligentes, onde a capacidade de coletar, processar e transmitir dados de forma eficiente é essencial.

No contexto da Internet das Coisas (IoT, do inglês *Internet of Things*), surgem desafios relacionados à limitação de recursos computacionais, capacidade energética e alcance dos dispositivos. Nesse cenário, o uso de sistemas operacionais especializados, como o Contiki-NG, torna-se uma alternativa viável para permitir que dispositivos de hardware restrito operem em redes eficientes, escaláveis e interoperáveis. O Contiki-NG

oferece suporte a protocolos relevantes para redes IoT, incluindo IEEE 802.15.4, 6LoWPAN, RPL e LoRa, além de fornecer uma pilha de rede otimizada para dispositivos com recursos limitados.

Embora o Contiki-NG seja tradicionalmente aplicado em plataformas como Tmote Sky, Zolertia Z1 e outras placas específicas para IoT, este trabalho propõe sua portabilidade e utilização na plataforma Arduino Uno, combinada com o módulo de comunicação nRF24L01+, com o objetivo de viabilizar redes de sensores de baixo custo e baixo consumo energético. Mais ainda, devido ao fácil acesso a esse conjunto de hardware no mercado e seu baixo custo, permite seu uso em larga escala. A comunicação entre dispositivos Arduino sob o Contiki-NG representa um desafio técnico, dado que a plataforma não possui suporte nativo no sistema operacional, o que exige o desenvolvimento de drivers e adaptações específicas.

O restante deste trabalho está organizado da seguinte forma. A Seção 2 apresenta os trabalhos relacionados. A Seção 3 fornece a fundamentação teórica trazendo as principais definições e conceitos importantes para o devido entendimento do estudo aqui realizado. A Seção 4 descreve a metodologia empregada e desenvolvimento da proposta. A Seção 5 detalha os resultados alcançados. Finalmente, a Seção 6 conclui o trabalho e discute possíveis desdobramentos futuros.

2. Trabalhos Relacionados

2.1. Repositório: /PIlin/contiki-arduino

O principal trabalho que serviu de base para este projeto foi o repositório `PIlin/contiki-arduino`[1], cujo diretório raiz está ilustrado na Figura 1, que apresenta uma estrutura inicial para a execução do sistema operacional Contiki-NG em plataformas baseadas na arquitetura AVR, como o Arduino Uno.

A escolha deste repositório foi motivada por três preocupações centrais. Em primeiro lugar, buscou-se uma base que seguisse, tanto quanto possível, a organização modular recomendada na documentação oficial do Contiki-NG, com a devida separação entre os diretórios `/cpu`, `/platform` e `/dev`, o que facilita a manutenção, a reutilização e a futura expansão do sistema. Em segundo lugar, considerou-se a necessidade de utilizar um projeto relativamente recente. Embora, segundo [Oikonomou et al. 2022], mais de 2.000 publicações revisadas por pares utilizem a expressão de busca "Contiki AND OS" no banco de dados Scopus, evidenciando sua ampla adoção em pesquisas acadêmicas sobre redes de sensores sem fio, muitos dos repositórios relacionados ao uso do Contiki com Arduino não são atualizados há mais de 15 anos, o que compromete sua compatibilidade com as versões atuais do sistema e das ferramentas de compilação. Por fim, optou-se pelo repositório `PIlin` devido à sua presença ligeiramente mais significativa na comunidade online, com menções em fóruns como o do Arduino e maior visibilidade em mecanismos de busca.

O repositório adotado reorganiza parcialmente a estrutura tradicional do Contiki-NG, movendo os diretórios `/cpu` e `/platform` diretamente para a raiz do projeto. Além disso, os arquivos que originalmente ficariam centralizados na pasta `/dev`, responsáveis por drivers genéricos ou compartilhados entre plataformas, foram distribuídos e in-

[1]Disponível em: `https://github.com/PIlin/contiki-arduino`

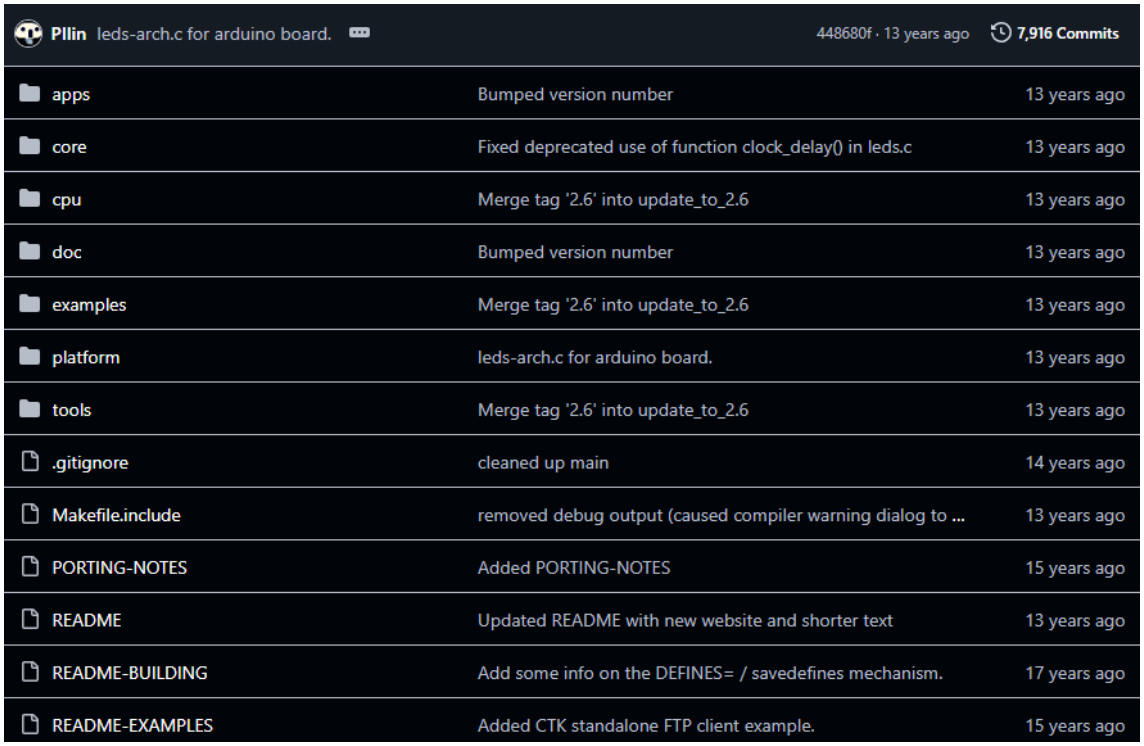


Figura 1. Diretório raiz do `/PIlin/contiki-arduino`

corporados diretamente nos subdiretórios específicos de cada arquitetura dentro de `/cpu`. Essa descentralização altera a modularidade típica do Contiki-NG e torna mais difícil a reutilização de código entre diferentes arquiteturas, mas ao mesmo tempo simplifica a organização em projetos com foco exclusivo em uma única família de microcontroladores, como a arquitetura AVR no caso deste trabalho.

Ele também inclui o arquivo `PORTING-NOTES`, com orientações para adaptação a placas da família Arduino, sugestões para suporte a novos microcontroladores AVR e observações sobre as limitações de cada chip. Apesar de fornecer suporte básico à execução do sistema no Arduino Uno, o repositório não oferece drivers completos para rádios externos nem implementações funcionais para a camada de comunicação SPI ou suporte ao módulo nRF24L01+, o que exigiu, neste trabalho, o desenvolvimento de componentes adicionais para possibilitar a comunicação sem fio. O documento aborda, por exemplo, a necessidade de uma estrutura genérica e flexível para facilitar a inclusão de novos microcontroladores AVR, bem como sugestões para a criação de cabeçalhos específicos por chip.

2.2. InsideGadgets: Exemplos prático com nRF24L01+

Além da base do projeto `PIlin/contiki-arduino`, foram consultadas referências importantes relacionadas à implementação prática com o módulo nRF24L01+, especialmente os recursos publicados pelo site InsideGadgets e seu repositório no GitHub.

InsideGadgets: "Using the nRF24L01 Wireless Module": Um post técnico de 2012 no site InsideGadgets apresenta um guia detalhado de integração do nRF24L01+ com

microcontroladores AVR, como o ATmega328P [InsideGadgets 2012]. O autor aborda a configuração via SPI, o mapeamento dos pinos CE, CSN, MOSI, MISO e IRQ, além de explicações sobre bits de status como `MAX_RT` e `RX_DR`. Esse material foi fundamental para validar os procedimentos de leitura e escrita em registradores e o tratamento de interrupções no driver desenvolvido.

Repositório `insidegadgets/nRF24L01` no GitHub: Esse repositório disponibiliza diversos exemplos em linguagem C para a plataforma AVR, com códigos de leitura e escrita de registradores, configuração dos modos PTX e PRX, uso das filas FIFO e diagnóstico via registrador `STATUS` [InsideGadgets 2015]. As implementações serviram como referência prática para estruturar o driver compatível com o Contiki-NG, além de confirmar estratégias para inicialização, configuração e transmissão manual via SPI.

Essas fontes foram importantes para:

- Validar a lógica de configuração do barramento SPI, incluindo mapas de pinos e sequências de inicialização;
- Confirmar a manipulação correta dos registradores `STATUS`, `CONFIG`, `EN_AA`, `SETUP_RETR`, entre outros;
- Compreender os modos de operação e o tratamento de erros como retransmissões máximas excedidas (`MAX_RT`).

Essas referências complementaram a documentação técnica oficial (*datasheet*) e serviram como base prática para a validação do driver implementado neste trabalho.

3. Fundamentação Teórica

Esta seção apresenta os conceitos teóricos fundamentais para a compreensão do trabalho. Inicia-se com uma visão geral sobre a Internet das Coisas (IoT) e as redes de sensores sem fio (WSNs), abordando os principais tipos de comunicação e plataformas de hardware utilizadas nesse contexto. Em seguida, são discutidas as características dos sistemas operacionais embarcados, com ênfase no Contiki-NG, sua arquitetura modular e os aspectos relacionados à sua portabilidade. Por fim, são detalhadas as especificidades do microcontrolador ATmega328P e do módulo transceptor nRF24L01+, cuja integração constitui o núcleo experimental deste projeto.

3.1. Redes de Internet das Coisas

A Internet das Coisas (IoT) refere-se à interligação de dispositivos físicos à internet, permitindo que objetos comuniquem dados de forma automatizada. Essa tecnologia está presente em diversos setores, como cidades inteligentes, indústria e saúde, proporcionando maior eficiência e automação de processos [Al-Fuqaha et al. 2015].

As aplicações da IoT abrangem desde o monitoramento ambiental e controle de tráfego até sistemas de iluminação e energia inteligentes. Esses sistemas conectados geram grandes volumes de dados que precisam ser processados e analisados em tempo real, exigindo arquiteturas distribuídas e escaláveis. A integração da IoT com tecnologias emergentes, como computação em nuvem e redes 5G, tem potencializado ainda mais sua adoção em ambientes urbanos, industriais e domésticos [Zanella et al. 2014].

3.1.1. Comunicação sem Fio

A comunicação sem fio permite a transmissão de dados sem a utilização de cabos, normalmente por meio de ondas de rádio. Apesar da flexibilidade, esse tipo de comunicação está sujeito a problemas como interferência, atenuação e colisões de pacotes, exigindo técnicas robustas de controle [Palattella et al. 2016].

Em sistemas de IoT, a escolha da tecnologia de comunicação sem fio impacta diretamente o desempenho da rede, incluindo alcance, taxa de transmissão e consumo energético. Tecnologias como Wi-Fi, ZigBee, LoRa e 5G são empregadas em diferentes contextos, cada uma com suas vantagens e limitações. Além disso, aspectos como eficiência espectral, robustez do enlace e escalabilidade da rede são fatores críticos na escolha da solução mais adequada [Khan et al. 2019].

3.1.2. Redes Sem Fio

As redes sem fio podem ser classificadas em dois grandes grupos: redes com infraestrutura, que utilizam pontos de acesso fixos para gerenciar a comunicação, e redes sem infraestrutura, também chamadas de redes *ad hoc*, nas quais os dispositivos se comunicam diretamente e organizam-se dinamicamente.

Redes com Infraestrutura: Esse tipo de rede utiliza estações base ou pontos de acesso que centralizam a comunicação. São comuns em sistemas de telefonia celular e redes Wi-Fi, oferecendo boa taxa de transmissão, mas com custo elevado de implantação.

Redes sem Infraestrutura: Já as redes *ad hoc* operam sem pontos centralizadores. Os dispositivos se comunicam de forma colaborativa, formando caminhos dinâmicos entre os nós. Tais redes são ideais para ambientes com difícil acesso ou cenários emergenciais, devido à flexibilidade e baixo custo.

3.1.3. Redes de Sensores Sem Fio

As Redes de Sensores Sem Fio (WSNs) são um tipo específico de rede *ad hoc*, compostas por sensores de baixo consumo energético. Essas redes são utilizadas em aplicações como monitoramento ambiental, rastreamento de ativos e automação predial, muitas vezes operando em locais remotos e com autonomia limitada [Rawat et al. 2014].

Com o crescimento da IoT, surgem novos desafios relacionados à latência, escalabilidade e gestão de energia em WSNs. Soluções como computação em névoa (*fog computing*) têm sido exploradas para aproximar o processamento dos dados da borda da rede, reduzindo a sobrecarga em servidores centrais e melhorando a capacidade de resposta em aplicações sensíveis ao tempo, como sistemas de vigilância e cidades inteligentes [Aazam et al. 2018].

3.1.4. Plataformas IoT

A comunicação entre dispositivos em redes IoT é organizada por uma arquitetura em camadas, listadas a seguir, e implementadas em plataformas de hardware de baixo custo com limitação de recursos (CPU, memória, energia, comunicação).

- **Camada física:** Responsável pela conversão de dados em sinais transmitidos pelo meio.
- **Camada de enlace:** Gerencia o acesso ao meio e identifica os dispositivos.
- **Camada de rede:** Responsável pelo roteamento dos dados entre os nós.
- **Camada de aplicação:** Fornece os serviços finais, como visualização ou controle remoto.

3.2. Plataformas de Hardware para IoT e WSNs

As redes de sensores sem fio (WSNs) e aplicações de Internet das Coisas (IoT) exigem plataformas de hardware com baixo consumo energético, capacidade de processamento embarcado e interfaces de comunicação eficientes. Diversos dispositivos foram projetados especificamente para esse propósito, integrando sensores, rádios e microcontroladores em um único módulo.

A seguir, são descritas algumas das principais plataformas amplamente utilizadas em pesquisa e desenvolvimento de WSNs, incluindo seus recursos técnicos e a compatibilidade com sistemas operacionais embarcados.

Tmote Sky: O Tmote Sky é uma plataforma embarcada baseada no microcontrolador MSP430 da Texas Instruments, com rádio CC2420 compatível com o padrão IEEE 802.15.4. Possui sensores integrados de temperatura e luminosidade, sendo ideal para aplicações ambientais. Essa placa é compatível com sistemas operacionais como Contiki, Contiki-NG, TinyOS e RIOT [Moteiv Corporation 2006b, Texas Instruments 2018, Texas Instruments 2007].

Zolertia Z1: A Zolertia Z1 também utiliza o MSP430 e oferece melhorias de integração em relação ao Tmote Sky, com uma construção mais robusta e conectores de expansão. Suporta rádios IEEE 802.15.4 e é utilizada em cenários reais de IoT. A Z1 é compatível com sistemas como Contiki-NG, RIOT OS e OpenWSN [Zolertia 2015, Texas Instruments 2018, Texas Instruments 2007].

Micaz: Micaz é uma plataforma clássica baseada no microcontrolador Atmega128L e rádio CC2420. Amplamente utilizada em pesquisas acadêmicas, ela suporta os sistemas operacionais TinyOS e Contiki, sendo possível simular seu comportamento com precisão no ambiente Cooja [Crossbow Technology 2006, Texas Instruments 2007].

TelosB: A TelosB é similar ao Tmote Sky, baseada no MSP430 e rádio CC2420. Foi projetada para fácil uso em laboratórios e ensino, sendo uma das plataformas mais usadas em estudos com Cooja. É compatível com Contiki, Contiki-NG, TinyOS e RIOT [Moteiv Corporation 2006a, Texas Instruments 2018, Texas Instruments 2007].

Arduino Uno: Apesar de não ser originalmente projetado para redes de sensores, o Arduino Uno (baseado no microcontrolador ATmega328P) é amplamente utilizado em prototipagem e ensino. Sua compatibilidade com sistemas operacionais embarcados é limitada, mas há esforços para portá-lo para sistemas como Contiki-NG e RIOT, exigindo adaptações específicas. O Arduino Uno foi o hardware escolhido neste trabalho devido ao seu baixo custo, baixo consumo e facilidade de obtenção, permitindo seu uso em larga escala [Microchip Technology 2016].

Simulação no Cooja: O simulador Cooja, desenvolvido para o sistema Contiki, permite a simulação de diversas dessas plataformas com alta fidelidade. Ele oferece suporte a Tmote Sky, TelosB, Micaz e outras, possibilitando o desenvolvimento e validação de redes de sensores mesmo sem acesso ao hardware físico. Algumas simulações incluem modelagem de consumo energético e interferências no canal de comunicação [Cooja 2017].

A diversidade de plataformas suportadas pelo Contiki-NG demonstra sua flexibilidade e aplicabilidade em cenários diversos, desde experimentos acadêmicos até aplicações comerciais. No contexto deste trabalho, a escolha do Arduino Uno representa um desafio adicional, já que não há suporte nativo, exigindo o desenvolvimento de uma camada de portabilidade e integração com periféricos como o módulo nRF24L01+.

3.3. Sistemas Operacionais e o Contiki-NG

O desenvolvimento de aplicações para Redes de Sensores Sem Fio (WSNs) requer sistemas operacionais capazes de operar em dispositivos com recursos extremamente limitados, como memória reduzida, consumo energético restrito e baixa capacidade de processamento. Para atender a esses requisitos, surgiram sistemas operacionais especialmente projetados para ambientes embarcados e de IoT.

Entre os principais sistemas operacionais utilizados em WSNs, destacam-se:

- **TinyOS:** Um dos primeiros sistemas para sensores, baseado em uma arquitetura orientada a eventos e escrito em NesC. Foi amplamente utilizado em pesquisas, mas apresenta maior complexidade na programação [Levis et al. 2003].
- **RIOT OS:** Sistema modular e multitarefa com suporte a programação em C, compatível com uma ampla gama de microcontroladores e protocolos IoT. É uma alternativa moderna com suporte a multithreading preemptivo [Baccelli et al. 2018].
- **FreeRTOS:** Voltado para sistemas embarcados em geral, com foco em aplicações industriais. Suporta tarefas em tempo real, mas seu uso em WSNs energicamente restritas é menos comum.

O Contiki-NG tem sido continuamente aprimorado por uma comunidade ativa de desenvolvedores e pesquisadores, consolidando-se como uma plataforma robusta e versátil para o desenvolvimento de aplicações em IoT. O sistema adota boas práticas de engenharia de software, com foco em modularidade, testabilidade e suporte a dispositivos modernos. Entre os avanços recentes, destacam-se a introdução de melhorias na pilha de rede, ampliação do suporte a novos microcontroladores e uma abordagem orientada a testes automatizados, fatores que contribuem para maior confiabilidade e escalabilidade das soluções baseadas nessa plataforma [Oikonomou et al. 2022].

Apesar dessas opções, o sistema **Contiki-NG** se destaca por seu equilíbrio entre eficiência, flexibilidade e suporte a protocolos modernos voltados para IoT. Seu ecossistema, compatibilidade com simulação em alto nível e arquitetura modular o tornam especialmente atrativo para desenvolvimento e pesquisa em redes de sensores sem fio.

3.3.1. Contiki-NG

Contiki-NG (*Next Generation*) é um sistema operacional de código aberto voltado para dispositivos embarcados com recursos limitados, como memória e energia. Ele é amplamente utilizado em redes de sensores sem fio, sistemas de automação e aplicações da Internet das Coisas (IoT). O sistema é uma evolução do Contiki original, trazendo melhorias em modularidade, suporte a protocolos modernos e compatibilidade com diversas plataformas de hardware.

Seu núcleo é projetado para ser leve e eficiente, com um modelo de execução cooperativo baseado em processos protothread, que simplifica o gerenciamento de tarefas concorrentes sem o uso intensivo de pilhas. Isso permite que o Contiki-NG funcione em microcontroladores com poucos *kilobytes* de RAM e memória *flash*, como o MSP430, ARM Cortex-M e AVR, sendo especialmente adequado para plataformas de sensores.

Contiki-NG fornece uma pilha completa de protocolos para comunicação em redes de sensores, incluindo suporte nativo a IPv6, 6LoWPAN, RPL e CoAP. Esses protocolos permitem que dispositivos embarcados se comuniquem de forma eficiente em redes mesh, mesmo em condições adversas de conectividade.

Além disso, o sistema inclui ferramentas para simulação e depuração, como o Cooja, que possibilita simular topologias de rede complexas e testar o comportamento de aplicações antes da implantação real. Essa capacidade de simulação é um dos diferenciais do Contiki-NG, permitindo acelerar o desenvolvimento de soluções confiáveis em WSN.

Arquitetura Modular A estrutura do Contiki-NG é altamente modular e organizada hierarquicamente, com ênfase na separação entre camadas de hardware, drivers e aplicações. Essa arquitetura é centralizada no diretório `/arch` (ver Figura 2), que contém os componentes de portabilidade, organizados da seguinte forma:

- `/arch/cpu`: contém os drivers específicos para diferentes microcontroladores (MCUs), incluindo temporizadores, UART, SPI, watchdog, entre outros;
- `/arch/dev`: agrupa drivers de dispositivos genéricos ou compartilhados entre plataformas;
- `/arch/platform`: abriga os arquivos de configuração e drivers específicos de cada placa, como LEDs, sensores, rádio e pinagem física.

Essa organização permite o reuso de drivers entre placas que compartilham o mesmo microcontrolador e facilita a inclusão de novos dispositivos no ecossistema do sistema operacional.

Graças a essa separação de responsabilidades, o Contiki-NG permite uma portabilidade mais estruturada e flexível, com regras claras para suporte à CPU (nível de microcontrolador) e suporte à plataforma (placa física), conforme abordado na próxima

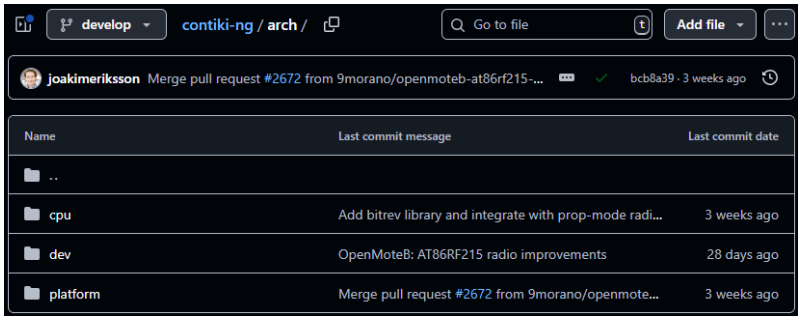


Figura 2. Organização modular do diretório `/arch` no Contiki-NG.

seção. É possível ajustar o protocolo MAC, configurar camadas de rede e até portar o sistema para novas plataformas de hardware, como foi feito neste trabalho com a plataforma Arduino Uno e o módulo transceptor nRF24L01+.

3.3.2. Sistema de *Build* do Contiki-NG

O sistema de build do Contiki-NG permite compilar imagens completas do sistema para diversas plataformas de hardware, integrando o código da aplicação e do sistema operacional em um único firmware.

A compilação é realizada a partir de um Makefile simples, que define o projeto e inclui o Makefile principal do Contiki-NG. Por padrão, o build é feito para a plataforma `native`, que gera um executável para o sistema de desenvolvimento. Para compilar para outras plataformas, utiliza-se a variável `TARGET`, informando o hardware desejado, como por exemplo:

```
make TARGET=arduino
```

Além disso, quando a plataforma suporta diferentes placas, pode-se especificar a variável `BOARD` para definir a variante exata, como em:

```
make TARGET=zoul BOARD=remote-reva
```

Essa flexibilidade permite construir firmwares específicos para diferentes configurações de hardware a partir do mesmo código-fonte.

Os arquivos gerados são organizados em uma estrutura de diretórios dentro da pasta `build/`, separando os resultados por plataforma, placa e configurações específicas. Essa organização facilita a manutenção de múltiplas versões, como para testes ou implantação.

O sistema oferece comandos para limpeza dos arquivos gerados (`clean` e `distclean`) e diversas opções úteis para gerenciamento e inspeção, como listar plataformas suportadas, visualizar configurações e acessar a saída serial do dispositivo.

A arquitetura do sistema de build é composta por uma hierarquia de Makefiles que inclui regras gerais, específicas para a plataforma (`TARGET`), para a arquitetura do processador (`CPU`) e para módulos opcionais. Essa estrutura modular permite flexibilidade e facilita a portabilidade do Contiki-NG para diferentes hardwares.

3.3.3. Portabilidade do Contiki-NG para Novas Plataformas

O sistema operacional Contiki-NG foi projetado com uma arquitetura modular, voltada para dispositivos embarcados e aplicações em Internet das Coisas (IoT). Sua flexibilidade permite que seja portado para diversas plataformas de hardware, desde que respeitadas as etapas específicas de adaptação definidas em sua documentação oficial.

A portabilidade de um sistema operacional consiste na adaptação de seu núcleo, drivers e interfaces para funcionar corretamente em uma nova arquitetura de microcontrolador e plataforma física. No caso do Contiki-NG, essa tarefa é bem mais simples devido à uma clara divisão entre os módulos de CPU, plataforma, periféricos e rádios, organizados em diretórios como `arch/cpu/`, `arch/platform/` e `dev/`.

O processo de portabilidade pode ser dividido, conceitualmente, em duas etapas principais:

- **Suporte à CPU (microcontrolador):** envolve a criação de arquivos que definem o comportamento do processador central, incluindo o temporizador principal (`clock.c`), registradores de interrupção, UART, watch dog, e outras funções de baixo nível. Também é necessário configurar o sistema de build com um `Makefile` específico, listando os arquivos-fonte que devem ser compilados para aquele processador.
- **Suporte à plataforma (placa física):** refere-se à implementação de drivers para componentes fora do chip principal, como LEDs, botões, sensores, rádios externos e interfaces SPI/I2C. Essa camada também exige sua própria configuração via `Makefile` e arquivos como `platform.c` e `contiki-conf.h`.

A integração com o sistema de compilação do Contiki-NG se baseia em três variáveis principais: `TARGET`, que define a plataforma; `BOARD`, que especifica variações de hardware; e `MODULES`, que adiciona funcionalidades extras. Essas variáveis orientam o sistema de build sobre quais fontes e configurações utilizar ao compilar um firmware.

Entre os componentes essenciais que devem ser implementados ou adaptados durante a portabilidade, destacam-se:

- **Temporizadores:** implementações dos módulos `clock` e `rtimer`, fundamentais para o agendamento de eventos e funcionamento do kernel.
- **GPIO e UART:** drivers de entrada e saída, incluindo suporte a `printf` e comunicação serial.
- **SPI:** interface crítica para comunicação com rádios externos e sensores.
- **Rádio transceptor:** caso a plataforma não possua rádio integrado, é necessário desenvolver suporte específico para o módulo externo (por exemplo, nRF24L01+).

Essa modularidade permite que partes do código sejam reutilizadas entre diferentes placas com a mesma CPU, facilitando o desenvolvimento de variantes de plataforma com componentes distintos. Por fim, a documentação recomenda criar exemplos funcionais e testes de compilação para validar a nova plataforma, garantindo compatibilidade com o ecossistema Contiki-NG.

3.4. Microcontroladores e a Plataforma AVR

Microcontroladores são componentes fundamentais em sistemas embarcados, especialmente em aplicações de IoT e WSNs. Eles integram em um único chip um processador, memória e periféricos, permitindo o controle local de sensores e atuadores com consumo energético reduzido.

Entre as diversas arquiteturas de microcontroladores, a família AVR, desenvolvida pela Atmel (atualmente Microchip), se destaca por seu uso em plataformas de prototipagem como o Arduino. Esses microcontroladores de 8 bits combinam simplicidade, baixo consumo e custo reduzido, sendo ideais para aplicações acadêmicas e projetos de prototipagem rápida.

3.4.1. Arduino Uno e o Microcontrolador ATmega328P

O Arduino Uno é uma das plataformas embarcadas mais utilizadas no mundo educacional e de pequenos projetos. Ele utiliza o microcontrolador ATmega328P, da família AVR, que oferece recursos suficientes para aplicações básicas em IoT e automação.

As principais características do ATmega328P incluem [Microchip Technology 2016]:

- CPU RISC de 8 bits operando a 16 MHz;
- 32 KB de memória flash para armazenamento de código;
- 2 KB de SRAM para dados em tempo de execução;
- 1 KB de EEPROM para armazenamento não volátil;
- Interfaces de comunicação como SPI, I2C e UART;
- Periféricos integrados para temporização, PWM e conversão analógico-digital (ADC).

Apesar das limitações em termos de memória e capacidade de processamento, o ATmega328P continua sendo amplamente utilizado em pesquisas e aplicações educacionais, principalmente devido à sua simplicidade, baixo custo e ao extenso ecossistema de suporte disponível [Gadekar et al. 2021]. No entanto, sua integração com sistemas operacionais embarcados como o Contiki-NG requer adaptações específicas, sobretudo no que diz respeito ao gerenciamento de memória e à comunicação SPI com dispositivos externos, como transceptores sem fio [Oikonomou et al. 2022].

Cabe destacar que a escolha do Arduino Uno como plataforma para este trabalho está diretamente relacionada à sua ampla acessibilidade e ao desafio técnico envolvido na portabilidade do Contiki-NG para um microcontrolador com recursos tão limitados. Essa portabilidade exige otimizações tanto no sistema operacional quanto na camada de abstração de hardware, possibilitando a integração com módulos como o nRF24L01+ e viabilizando a construção de redes de sensores baseadas em tecnologias abertas e de baixo custo.

3.4.2. Interface SPI no Arduino Uno

A comunicação SPI (Serial Peripheral Interface) é uma das interfaces mais importantes em sistemas embarcados, permitindo a comunicação síncrona de alta velocidade entre

microcontroladores e periféricos externos, como sensores, memórias e transceptores de rádio.

No Arduino Uno, a SPI é implementada por hardware no microcontrolador ATmega328P, utilizando os pinos físicos presentes na Tabela 1.

Tabela 1. Mapeamento dos pinos SPI no Arduino Uno

Nome Lógico	Função SPI	Porta AVR	Bit	Pino Arduino Uno
MOSI	Master Out Slave In	PORTB	PB3	D11
MISO	Master In Slave Out	PORTB	PB4	D12
SCK	Clock	PORTB	PB5	D13
CSN	Chip Select Not	PORTB	PB2	D10

3.5. Módulo Transceptor nRF24L01+

O nRF24L01+ é um transceptor de rádio frequência (RF) de baixo consumo energético e curto alcance, operando na faixa de 2,4 GHz ISM (do inglês *Industrial, Scientific and Medical*). Desenvolvido pela Nordic Semiconductor, o módulo é amplamente utilizado em aplicações embarcadas que exigem comunicação sem fio ponto-a-ponto ou em redes multi-hop com baixa taxa de transmissão.

O transceptor suporta taxas de transmissão de até 2 Mbps e opera com tensões entre 1,9 V e 3,6 V. Possui uma interface de comunicação via SPI (do inglês *Serial Peripheral Interface*), permitindo fácil integração com microcontroladores de baixo custo, como o ATmega328P [Nordic 2007].

Entre suas principais características, destacam-se:

- Modulação GFSK (*Gaussian Frequency Shift Keying*);
- Até 6 canais de recepção simultânea (*pipes*), com endereçamento individual;
- Suporte a reconhecimento automático de pacotes (ACK) e retransmissão automática (*Auto ACK*);
- *Buffers* de transmissão e recepção com FIFO de 3 pacotes;
- Modo de economia de energia com *wake-up* via pino de interrupção (IRQ).

O protocolo proprietário *Enhanced ShockBurst*™ permite uma comunicação confiável com baixo *overhead* de controle, é responsável o empacotamento, envio, verificação e resposta automática de pacotes, sem intervenção direta do microcontrolador. Isso reduz significativamente o consumo energético em aplicações sensíveis, como Redes de Sensores Sem Fio (WSN).

3.5.1. Comandos SPI e Registradores Internos

O módulo é controlado por comandos SPI que permitem acessar seus registradores internos e buffers de dados. Os comandos são compostos por um byte de instrução e, em alguns casos, bytes adicionais para leitura ou escrita [Nordic 2007]. Exemplos de comandos incluem:

- `R_REGISTER`: lê o conteúdo de um registrador (0x00 a 0x1D);

- `W_REGISTER`: escreve em um registrador;
- `R_RX_PAYLOAD` e `W_TX_PAYLOAD`: leitura e escrita dos buffers de recepção e transmissão;
- `FLUSH_TX` e `FLUSH_RX`: limpeza das filas FIFO.

A tabela de registradores do nRF24L01+ é extensa, sendo cada registrador associado a aspectos como configuração do rádio, endereçamento, controle de retransmissão, status de pacotes e outros. Entre os mais utilizados no desenvolvimento de drivers estão:

- `CONFIG` (0x00): habilita o rádio, ativa modo PRX ou PTX e configura interrupções;
- `STATUS` (0x07): indica eventos como sucesso de transmissão (`TX_DS`), recepção de pacote (`RX_DR`) e erro de retransmissão (`MAX_RT`);
- `RF_CH` (0x05): define o canal de operação;
- `RX_ADDR_P0-P5` e `TX_ADDR`: definem os endereços de comunicação;
- `EN_AA`, `SETUP_RETR`, `FEATURE`: controlam recursos como ACKs automáticos, número de retransmissões e payload dinâmico.

3.5.2. Modos de Operação: PTX, PRX e *Enhanced ShockBurst*™

O nRF24L01+ opera em dois modos principais:

- **PTX (*Primary Transmitter*):** envia pacotes de dados e aguarda um ACK automático do receptor, fluxograma na Figura 3.
- **PRX (*Primary Receiver*):** permanece escutando o canal e responde com ACKs ao receber pacotes válidos, fluxograma na Figura 4.

Ambos os modos utilizam o protocolo proprietário *Enhanced ShockBurst*™, que automatiza a criação de pacotes, verificação de integridade por CRC, gerenciamento de ACKs e retransmissões sem a necessidade de controle direto pelo microcontrolador. Esse mecanismo contribui para reduzir o consumo energético, tempo de processamento e simplificar a lógica de software.

Essas funcionalidades são configuradas por meio de registradores específicos:

- `EN_AA` (0x01): habilita ou desabilita o *Auto ACK* para cada pipe de recepção;
- `SETUP_RETR` (0x04): define o número máximo de retransmissões automáticas e o intervalo entre tentativas;
- `FEATURE` (0x1D): ativa funcionalidades avançadas como *payload* dinâmico, ACKs com *payload* e o comando `W_ACK_PAYLOAD`;
- `DYNPD` (0x1C): habilita *payloads* de tamanho variável para cada pipe;
- `CONFIG` (0x00): ativa o modo PRX ou PTX e configura interrupções relacionadas à comunicação.

O uso adequado desses registradores é essencial para aproveitar os recursos do *Enhanced ShockBurst*™, garantindo uma comunicação eficiente, confiável e de baixo consumo energético, características fundamentais para aplicações em Redes de Sensores Sem Fio (WSN).

Fluxos típicos de operação incluem:

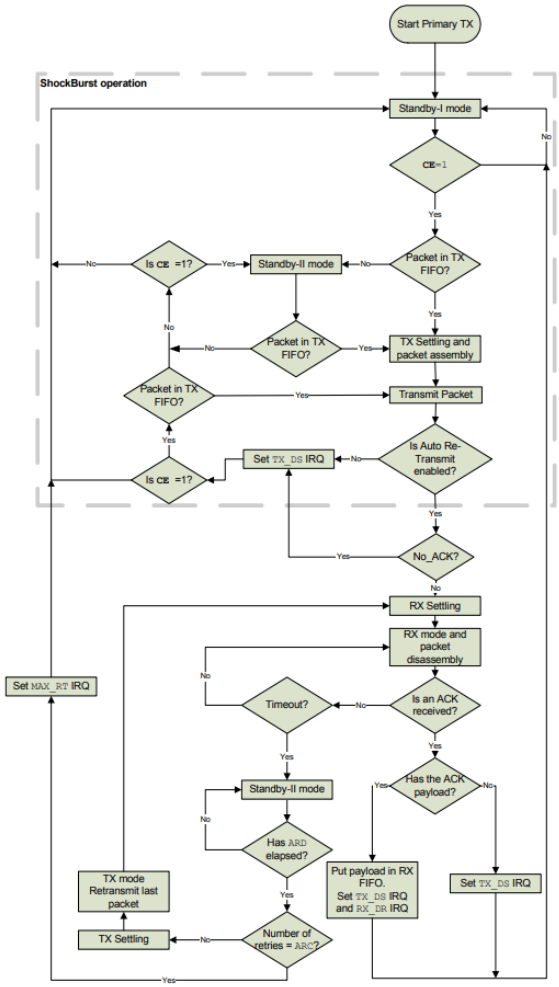


Figura 3. Fluxograma de funcionamento do modo PTX do nRF24L01+ [Nordic 2007].

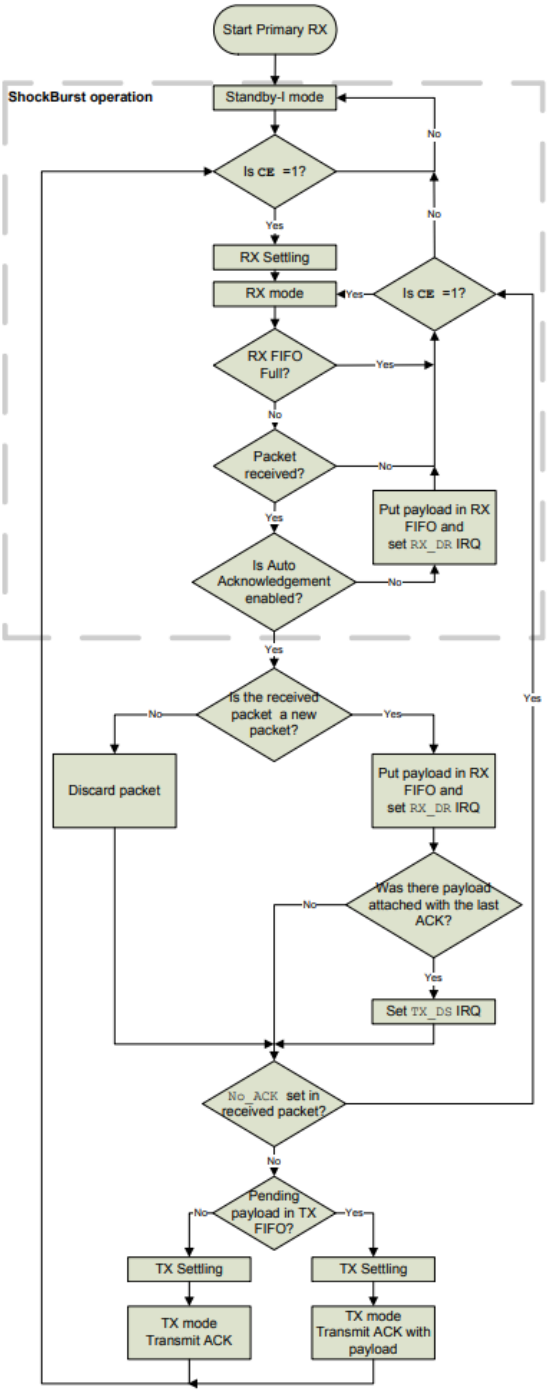


Figura 4. Fluxograma de funcionamento do modo PRX do nRF24L01+ [Nordic 2007].

- Escrita no FIFO de transmissão (TX) e pulso no pino `CE` para iniciar o envio (modo PTX);
- Espera por resposta via pino `IRQ` e verificação do registrador `STATUS`;
- No modo PRX, a recepção é contínua e os dados recebidos são armazenados no FIFO de recepção (RX), prontos para leitura.

3.5.3. Aplicação neste Trabalho

No contexto deste trabalho, o módulo nRF24L01+ é utilizado como meio de comunicação sem fio entre nós baseados no microcontrolador ATmega328P, operando sob o sistema operacional Contiki-NG. Sua integração exige o desenvolvimento ou adaptação de um driver de comunicação SPI e controle de registradores internos, de modo a configurar corretamente modos de operação, endereços de recepção, canais, potência de transmissão, entre outros.

A escolha do nRF24L01+ deve-se à sua ampla documentação, disponibilidade no mercado, baixo custo e suporte a funcionalidades fundamentais para WSN, como retransmissão automática e gerenciamento de múltiplos destinos.

4. Metodologia e Desenvolvimento

4.1. Metodologia

Como mostrado no fluxograma da Figura 5, as etapas foram estruturadas de forma sequencial, com foco na adaptação do sistema operacional Contiki-NG para o microcontrolador Arduino Uno (ATmega328P) e na integração do módulo transceptor nRF24L01+ para comunicação sem fio. As atividades foram organizadas com base em três eixos principais: portabilidade do sistema, implementação de drivers e testes práticos de comunicação.

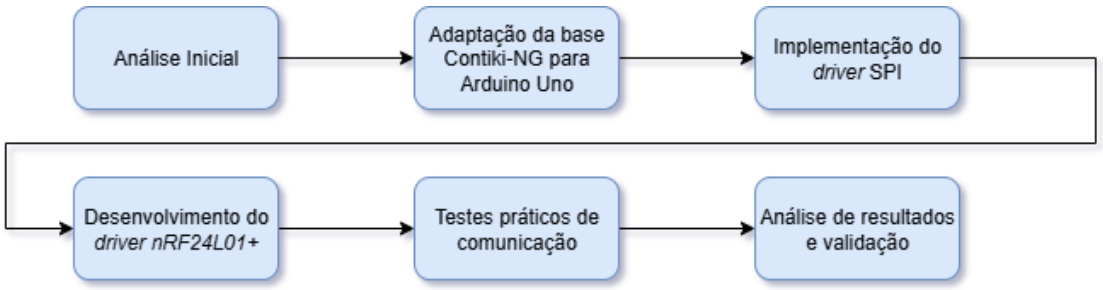


Figura 5. Fluxograma da metodologia

Inicialmente, foi realizada uma análise dos requisitos da arquitetura AVR e das limitações do Arduino Uno em relação ao suporte nativo do Contiki-NG. A partir dessa análise e de pesquisas por trabalhos relacionados ao tema, optou-se por utilizar como base o repositório `PIlin/contiki-arduino`, que já disponibilizava uma estrutura inicial de portabilidade. Essa base foi adaptada e expandida para permitir o uso do rádio nRF24L01+ e a comunicação SPI.

Na segunda etapa, foram feitas adaptações do *driver* SPI compatível com o Contiki-NG e com a arquitetura do Arduino Uno. Foram configurados os pinos físicos

da interface SPI e implementadas funções de transferência de dados utilizando os registradores `SPCR` (SPI *Control Register*), `SPSR` (SPI *Status Register*) e `SPDR` (SPI *Data Register*). Essa camada foi essencial para garantir a comunicação confiável entre o microcontrolador e o rádio.

Em seguida, foi desenvolvido o *driver* do nRF24L01+, com funcionalidades de leitura e escrita de registradores, configuração dos modos PTX (transmissor) e PRX (receptor) e manipulação da fila de transmissão (TX FIFO). Foram também implementadas funções de controle baseadas no registrador STATUS do nRF24L01+ e nas funcionalidades do *Enhanced ShockBurst*.

Como etapa final, foram realizados testes práticos para verificar a comunicação SPI com o módulo nRF24L01+ e seus registradores, o envio de pacotes e a recepção de confirmações (ACKs). Os testes foram conduzidos com um e dois dispositivos, e os resultados foram analisados com base nos valores dos registradores e na saída serial do Arduino.

Os resultados esperados incluem a validação do *driver* SPI, a comunicação bem-sucedida com o rádio e a troca de mensagens entre dois dispositivos Arduino Uno. A metodologia foi pensada para ser incremental, de forma a garantir o funcionamento isolado de cada etapa antes da integração total. Além disso, o trabalho foi conduzido com base em documentação técnica oficial (*datasheet* do nRF24L01+, *datasheet* do Atmega328P, documentação do Contiki-NG) e boas práticas de desenvolvimento embarcado.

4.2. Ambiente de Desenvolvimento

A plataforma utilizada para a portabilidade foi o Arduino Uno, baseado no microcontrolador ATmega328P. O ambiente de desenvolvimento foi configurado em um sistema operacional Linux Mint 22.1 Cinnamon, instalado em *dual boot* com Windows 11, visando garantir maior compatibilidade com as ferramentas de compilação e *upload* utilizadas para microcontroladores da arquitetura AVR.

O projeto foi hospedado em um fork próprio do repositório original `contiki-arduino`, disponível em `github.com/ggborges/contiki-arduino`, onde foram adicionadas e adaptadas as modificações necessárias para a execução do Contiki-NG na plataforma escolhida.

As ferramentas utilizadas para o desenvolvimento e testes foram as seguintes:

- **avr-gcc**: compilador da família GCC específico para microcontroladores AVR. Realiza a conversão do código-fonte em C para código de máquina executável pelo ATmega328P.
- **avr-objcopy**: integrante do conjunto de binutils para AVR, é utilizado para gerar arquivos no formato `.hex` a partir de arquivos intermediários (`.elf`, `.arduino`) criados pelo compilador.
- **avrdude**: ferramenta responsável por realizar o upload do arquivo `.hex` para o Arduino via porta USB, utilizando o protocolo apropriado para gravação em dispositivos AVR.
- **screen**, **minicom** e **ttylog**: programas utilizados para monitoramento da saída serial do Arduino, permitindo visualizar mensagens `printf` e depurar o comportamento do sistema em tempo real.

Todos os pacotes foram instalados por meio do gerenciador de pacotes `apt`, com os seguintes comandos principais:

```
sudo apt update
sudo apt install avr-gcc binutils-avr avrdude screen minicom
pip install ttylog
```

4.3. Arquitetura do Projeto

Diferentemente da estrutura oficial do Contiki-NG, o repositório reorganiza parcialmente os diretórios: as pastas `cpu/` e `platform/` foram movidas para a raiz do projeto, e os arquivos originalmente localizados em `arch/dev/` foram incorporados diretamente nas pastas de cada arquitetura dentro de `cpu/`. Essa modificação visa simplificar o processo de build e tornar o suporte a novas placas mais direto, embora se afaste da proposta modular documentada no projeto principal.

Durante o desenvolvimento, os principais diretórios utilizados foram:

- `/cpu/avr`: responsável pelos drivers comuns aos microcontroladores da arquitetura AVR. Contém arquivos como `clock.c`, `watchdog.c`, `uart.c`, `spi.c` e outros periféricos essenciais.
- `/platform/arduino`: abriga os arquivos de configuração específicos da placa Arduino Uno, como a pinagem e integração com o sistema.
- `/examples`: utilizado para testes de comunicação, com programas personalizados baseados nos exemplos clássicos do Contiki-NG.

Neste trabalho, foram utilizados principalmente os diretórios `/cpu`, `/platform` e `/examples`, partindo da base existente no repositório e ampliando-a com a criação de drivers e testes específicos para o microcontrolador ATmega328P e o módulo nRF24L01+.

O diretório `/cpu/avr` (Figura 6) contém os drivers voltados à arquitetura AVR como um todo, incluindo a subpasta `/avr/dev`, que reúne arquivos compartilhados entre diferentes microcontroladores dessa família. Foi nessa pasta que foi desenvolvido o driver específico para o módulo transceptor `nRF24L01+`, com a criação dos arquivos `nrf24l01_atmega328p.c` e `nrf24l01_atmega328p.h`, localizados em `/cpu/avr/dev`, desenvolvido especificamente para o microcontrolador alvo e adaptado ao modelo de comunicação esperado pelo Contiki-NG. Além disso, alterações foram realizadas no arquivo `spi.c`, responsável pela comunicação SPI, com o objetivo de adaptar sua operação ao módulo nRF24L01+ e à pinagem do Arduino Uno.

A organização final do projeto buscou manter a lógica modular do Contiki-NG sempre que possível, mesmo com a estrutura adaptada do repositório base. A separação entre camadas de CPU, plataforma e exemplos foi respeitada, e os novos drivers foram implementados de forma isolada, visando facilitar a manutenção futura e a expansão para outros microcontroladores AVR compatíveis.

A pasta `/examples` fornece uma série de aplicações de demonstração incluídas no Contiki-NG. Entre elas estão programas como `hello-world.c`, além da subpasta `/rime`, que contém implementações para testes de comunicação, como os exemplos `example-unicast.c` e `example-broadcast.c` (Figura 7).

Name	Last commit message
..	
dev	Merge tag '2.6' into update_to_2.6
radio	ATmega128RFA1 bug fix: no interrupt pin used (embedded radio).
Makefile.avr	Reorder includes to prefer platform routines. Remove redundant RPL de...
avr.c	* Generic AVR stuff.
avrdef.h	fixed defs and to compile for atmega32 also - has no TCNT3
bootloader.c	Add stk500 platform and changes suggested by Daniel Willmann
bootloader.h	Merge CDC-ECM, RNG, DFU bootloader, watchdog, settings manager, energ...
cc2420_spi.c	updated to new cc2420 spi naming
leds-arch.c	Provide dummy implementations for the leds and minileds module in order
minileds.c	Provide dummy implementations for the leds and minileds module in order
mtarch.c	Add missing mtarch_stop function
mtarch.h	AVR specific implementation of contiki multi-threating architecture
rtimer-arch.c	Add clock_delay_us and clock_set_seconds to clock.h. Modify clock_wai...
rtimer-arch.h	Add F_CPU=0x800000 option with phase lock to external 32768 Hz crystal
settings.c	Merge CDC-ECM, RNG, DFU bootloader, watchdog, settings manager, energ...
settings.h	Fix compiler warning
slip_uart1.c	Converted u8_t to uint8_t and u16_t to uint16_t in the cpu directory.
spi.c	Remove unused SPI initializer flag from AVR. Comment out in the msp-4...
watchdog.c	Add stk500 platform and changes suggested by Daniel Willmann

Figura 6. Organização prática dos arquivos do projeto para a arquitetura AVR.

Durante o projeto, alguns desses exemplos não puderam ser utilizados diretamente devido a limitações de memória do Arduino Uno, conforme discutido anteriormente. Ainda assim, essa pasta serviu como referência para a criação de exemplos personalizados voltados à verificação e teste do funcionamento da pilha de comunicação com o rádio nRF24L01+.

4.4. Desenvolvimento da Portabilidade para o Arduino Uno

O processo de portabilidade foi realizado para a plataforma Arduino Uno, baseada no microcontrolador ATmega328P. Essa escolha foi motivada pela sua ampla disponibilidade, documentação extensa e custo reduzido, apesar de suas limitações de memória e processamento.

4.4.1. Adaptação do Sistema de *Build*

Para possibilitar a compilação de aplicações Contiki para o Arduino Uno, foram necessárias modificações e verificações nos arquivos de *build*:

- Criação ou modificação de `Makefile.arduino` com a definição das variáveis `CONTIKI_SOURCEFILES` e flags específicas do AVR;
- Validação da geração correta dos arquivos `.hex` e sua transferência via `avrdude`.
- Validação do uso do comando `make TARGET=arduino ARDUINO_MODEL=Uno`, conforme estrutura já definida no repositório base;

Para compilar aplicações destinadas ao Arduino Uno, foi adotada a abordagem do projeto `contiki-arduino` [Ilin 2007], que utiliza a variável `ARDUINO_MODEL` no

Figura 7. Exemplos incluídos na pasta `/examples/rime` do Contiki-NG.

`Makefile` da aplicação para especificar o modelo da placa. Essa solução foge do padrão do sistema de *build* do Contiki-NG, que normalmente emprega as variáveis `TARGET` e `BOARD` para definir a plataforma e o modelo. Ainda assim, essa adaptação local permitiu compilar corretamente os exemplos com os arquivos-fonte específicos para o rádio e a interface SPI.

4.4.2. Implementação de *Drivers*

Durante a portabilidade, a principal modificação feita nos *drivers* existentes foi no arquivo `spi.c`, localizado em `/cpu/avr/dev`, que trata da comunicação SPI. As alterações incluíram:

- Definição explícita dos pinos utilizados para MOSI, MISO, SCK e CSN;
- Configuração da velocidade da comunicação SPI conforme as limitações do nRF24L01+;
- Implementação da função `spi_transfer()` para envio e recepção de dados via SPI.

Além disso, foi criado o arquivo correspondente `spi.h`, contendo as declarações das funções utilizadas no driver.

Outro componente desenvolvido foi o driver específico para o módulo transceptor sem fio nRF24L01+, composto pelos arquivos `nrf24l01_atmega328p.c` e `nrf24l01_atmega328p.h`. Esse driver foi implementado do zero, com base na do-

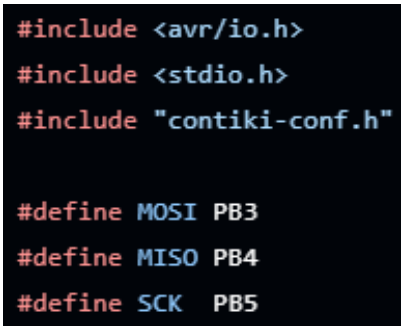


Figura 8. Mapeamento dos pinos utilizados na comunicação SPI no arquivo `spi.c`.

cumentação técnica do dispositivo e considerando a arquitetura modular do Contiki-NG e seu desenvolvimento será apresentado na próxima seção.

Demais arquivos como `clock.c`, `rtimer-arch.c`, `watchdog.c`, `uart.c` e `debug.c` já estavam presentes no repositório e não foram modificados neste projeto.

4.5. Desenvolvimento do *Driver* para o nRF24L01+

A implementação da comunicação sem fio entre dispositivos Arduino Uno utilizando o sistema operacional Contiki-NG exigiu a adaptação de componentes-chave, em especial o suporte à comunicação SPI e a implementação de um *driver* dedicado ao módulo transceptor nRF24L01+.

Foram realizadas as seguintes modificações iniciais:

- Criação de um *driver* para o módulo nRF24L01+, o arquivo `nrf24l01_atmega328p.c`,com comunicação via SPI;
- Ajustes no arquivo `spi.c` e criação do arquivo `spi.h`;
- Desenvolvimento do arquivo de teste `test-nrf.c` para primeiras validações da comunicação com o módulo transceptor.

4.5.1. Adaptação do *Driver* SPI

O driver SPI `spi.c`, localizado em `/cpu/avr`, foi modificado para:

- Configurar os pinos de comunicação SPI (D11, D12, D13), como mostrado na Figura 8;
- Ativar o modo mestre e habilitar o SPI (registradores `SPCR` e `SPSR`);
- Criar a função `spi_transfer()`, que envia um *byte* pelo MOSI e simultaneamente lê um *byte* pelo MISO;
- Ajustar a velocidade do clock SPI para `F_CPU/2` com o bit `SPI2X`.

Como mostrado na Figura 9, a função `spi_init()` configura os pinos de controle e dados para a comunicação SPI no modo mestre, definindo `MOSI`, `SCK` e `CSN` como saídas, e `MISO` como entrada. Habilita o periférico SPI, seleciona o modo mestre e ajusta a taxa de clock. A função `spi_transfer()` realiza a transmissão síncrona de um byte e aguarda a conclusão através do bit `SPIF` antes de retornar o valor recebido no barramento.

Os principais registradores manipulados no driver SPI foram:

```c
void
spi_init(void)
{
  /* Initalize ports for communication with SPI units. */
  /* CSN=SS and SCK must be output when master! */
  DDRB  |= BV(MOSI) | BV(SCK) | BV(CSN);
  DDRB &= ~BV(MISO); // MISO como entrada
  PORTB |= BV(MOSI) | BV(SCK);

  /* Enables SPI, selects "master", clock rate FCK / 4, and SPI mode 0 */
  SPCR = BV(SPE) | BV(MSTR);       // SPI Enable + Master mode (SPR1=0, SPR0=0)
  SPSR &= ~BV(SPI2X);              // SPI2X = 0 → não dobra a velocidade
}

uint8_t spi_transfer(uint8_t data) {
  SPDR = data;
  while (!(SPSR & BV(SPIF)));
  return SPDR;
}
```

Figura 9. Trecho de código em C da inicialização e comunicação da interface SPI.

- `DDRB`: define os pinos como entrada ou saída (ex: `DDRB |= (1 « PB3)`);
- `PORTB`: define o nível lógico dos pinos (*HIGH* ou *LOW*);
- `SPCR`: ativa o SPI, define o modo mestre;
- `SPSR`: ativa o bit `SPI2X` para duplicar a velocidade;
- `SPDR`: registrador de dados transmitidos e recebidos via SPI.

4.5.2. *Driver* do nRF24L01+

O driver para o módulo nRF24L01+ foi implementado nos arquivos:

- `/cpu/avr/dev/nrf24l01_atmega328p.c`
- `/cpu/avr/dev/nrf24l01_atmega328p.h`

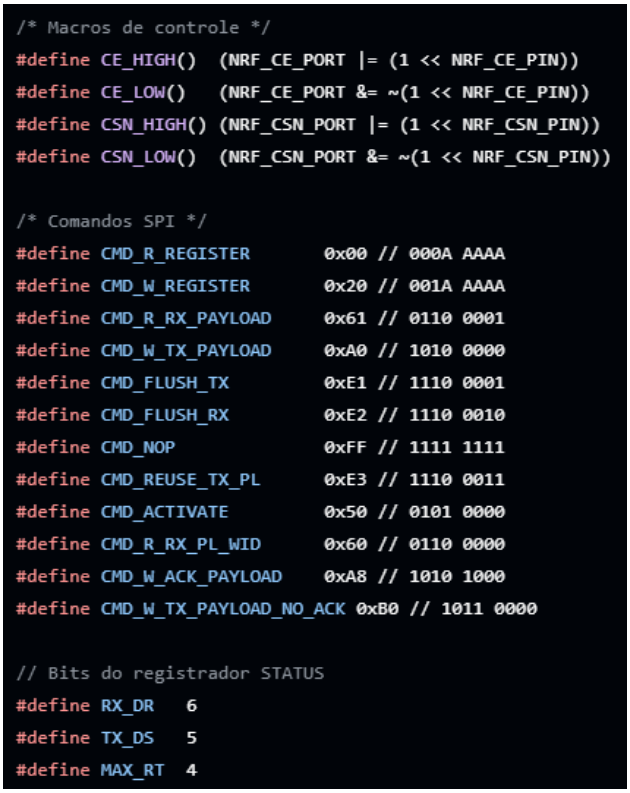
As funcionalidades inicialmente implementadas incluem:

- Leitura e escrita de registradores do nRF24L01+;
- Função para leitura do registrador STATUS (endereço 0x07);
- Inicialização do SPI e do modo de operação do módulo (modo transmissor).

A Figura 10 apresenta a definição das macros de controle e dos comandos SPI utilizados para comunicação com o módulo nRF24L01+. As macros `CE_HIGH()` e `CE_LOW()` controlam o pino de habilitação do rádio, enquanto `CSN_HIGH()` e `CSN_LOW()` gerenciam o pino de seleção do chip. Já os comandos SPI definidos permitem operações como leitura e escrita de registradores (`CMD_R_REGISTER`, `CMD_W_REGISTER`), manipulação da fila de transmissão (`CMD_FLUSH_TX`, `CMD_W_TX_PAYLOAD`) e controle de funcionalidades adicionais do transceptor, conforme especificado no datasheet.

O registrador STATUS (*datasheet*, p. 58) informa eventos como:

- Bit 6 - `RX_DR`: dado recebido;
- Bit 5 - `TX_DS`: transmissão bem-sucedida;
- Bit 4 - `MAX_RT`: máximo de retransmissões atingido.



```
/* Macros de controle */
#define CE_HIGH()  (NRF_CE_PORT |= (1 << NRF_CE_PIN))
#define CE_LOW()   (NRF_CE_PORT &= ~(1 << NRF_CE_PIN))
#define CSN_HIGH() (NRF_CSN_PORT |= (1 << NRF_CSN_PIN))
#define CSN_LOW()  (NRF_CSN_PORT &= ~(1 << NRF_CSN_PIN))

/* Comandos SPI */
#define CMD_R_REGISTER       0x00 // 000A AAAA
#define CMD_W_REGISTER       0x20 // 001A AAAA
#define CMD_R_RX_PAYLOAD     0x61 // 0110 0001
#define CMD_W_TX_PAYLOAD     0xA0 // 1010 0000
#define CMD_FLUSH_TX         0xE1 // 1110 0001
#define CMD_FLUSH_RX         0xE2 // 1110 0010
#define CMD_NOP              0xFF // 1111 1111
#define CMD_REUSE_TX_PL      0xE3 // 1110 0011
#define CMD_ACTIVATE         0x50 // 0101 0000
#define CMD_R_RX_PL_WID      0x60 // 0110 0000
#define CMD_W_ACK_PAYLOAD    0xA8 // 1010 1000
#define CMD_W_TX_PAYLOAD_NO_ACK 0xB0 // 1011 0000

// Bits do registrador STATUS
#define RX_DR   6
#define TX_DS   5
#define MAX_RT  4
```

Figura 10. Macros de controle, comandos SPI e bits do registrador `STATUS` para o `nRF24L01+`.

4.5.3. Validação e Testes

Os testes iniciais tiveram como objetivo validar a comunicação SPI entre o Arduino Uno e o módulo nRF24L01+, além de verificar se os registradores do rádio podiam ser acessados corretamente. As principais etapas realizadas foram:

Leitura Simples do registrador STATUS Um exemplo simples e inicial foi criado em `/examples/test-nrf24/test-nrf.c` contendo:

- Inicialização do SPI e do nRF24L01+;
- Impressão repetida do valor do registrador STATUS a cada 5 segundos.

Durante os testes, foram observados os seguintes resultados:

- `0x10`: bit `MAX_RT` ativado ⇒ tentativa de envio falhou (sem ACK);
- `0x0E`: RX FIFO vazia ⇒ nada recebido.

Esses resultados indicam que o Arduino está transmitindo, mas como apenas um dispositivo estava conectado, não houve ACK. A resposta do registrador STATUS demonstra que a comunicação SPI foi corretamente estabelecida.

Leitura e Escrita no Registrador CONFIG Para validar a comunicação SPI, foi realizado um teste de escrita e leitura no registrador `CONFIG` (endereço `0x00`), responsável por configurar o modo de operação do rádio.

```c
void nrf_write_register(uint8_t reg, const uint8_t value) {
  CSN_LOW(); // Ativa CSN
  spi_transfer(CMD_W_REGISTER | reg); // Comando W_REGISTER
  spi_transfer(value); // Envia o valor do registrador
  CSN_HIGH(); // Desativa CSN
}

void nrf_write_register_bytes(uint8_t reg, const uint8_t *data, uint8_t len) {
  CSN_LOW(); // Ativa CSN
  spi_transfer(CMD_W_REGISTER | reg); // Comando W_REGISTER
  for (uint8_t i = 0; i < len; i++) {
    spi_transfer(data[i]);
  }
  CSN_HIGH(); // Desativa CSN
}

uint8_t nrf_read_register(uint8_t reg) {
    uint8_t value;
    // "Every new command must be started by a high to low transition CSN."  (NRF24L01+ Datasheet, pg 50)
    CSN_LOW(); // Ativa CSN
    _delay_us(10); // Espera um pouco para garantir que CSN esteja baixa

    spi_transfer(CMD_R_REGISTER | reg); // Comando de leitura
    value = spi_transfer(CMD_NOP); // Lê o valor do registrador
    printf("READ: reg=0x%02X value=0x%02X\n", reg, value);

    CSN_HIGH(); // Desativa CSN
    return value;
}
```

Figura 11. Funções de leitura e escrita de registradores do `nRF24L01+` via SPI.

- Um valor de controle, como `0x0B`, foi escrito no registrador, configurando o rádio com CRC de 1 byte, modo receptor e módulo ativado.
- Em seguida, o valor foi lido usando a função `nrf24_read_register()` e comparado com o valor escrito.
- A leitura correta confirmou que a escrita e leitura via SPI estavam funcionando.

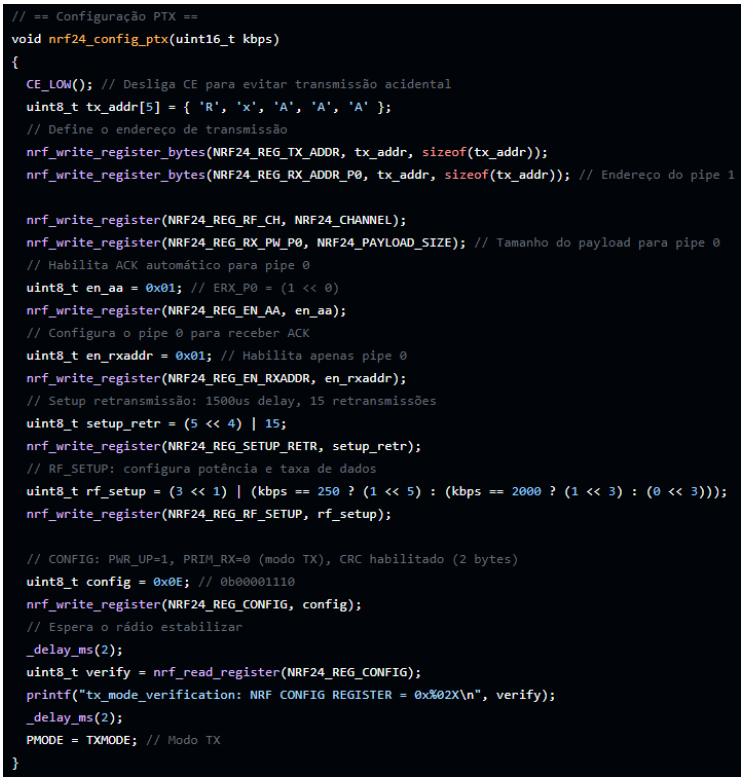
A Figura 11 ilustra as funções responsáveis pela leitura e escrita de registradores no módulo `nRF24L01+` por meio do barramento SPI. As funções `nrf_write_register()` e `nrf_write_register_bytes()` enviam dados de um único byte para registradores ou múltiplos bytes, respectivamente, enquanto `nrf_read_register()` realiza a leitura do conteúdo de um registrador. Em todas elas, o pino `CSN` é controlado para iniciar e encerrar a comunicação, conforme o protocolo descrito no *datasheet* do transceptor.

Implementação Parcial de PTX e PRX Com base na documentação oficial do nRF24L01+, foram criadas duas funções no *driver*:

- `setup_ptx()`: configura o módulo como transmissor (PTX), define o endereço de destino, canal de comunicação e ativa a retransmissão com ACK (Figura 12);
- `setup_prx()`: configura o módulo como receptor (PRX), define o pipe de recepção, ativa o modo de escuta e prepara o rádio para receber dados.

Ambas as funções ajustam os registradores do rádio, como `CONFIG`, `EN_RXADDR`, `EN_AA`, `SETUP_RETR`, `RF_CH`, entre outros.

A Figura 13 apresenta as funções `nrf_write_payload()` e `nrf_read_payload()`, responsáveis, respectivamente, pelo envio e recepção do *payload* de dados no módulo `nRF24L01+` via SPI. Essas funções controlam o sinal `CSN`, limitam o tamanho do *payload* conforme especificado no *datasheet* e utilizam comandos



```
// == Configuração PTX ==
void nrf24_config_ptx(uint16_t kbps)
{
  CE_LOW(); // Desliga CE para evitar transmissão acidental
  uint8_t tx_addr[5] = { 'R', 'x', 'A', 'A', 'A' };
  // Define o endereço de transmissão
  nrf_write_register_bytes(NRF24_REG_TX_ADDR, tx_addr, sizeof(tx_addr));
  nrf_write_register_bytes(NRF24_REG_RX_ADDR_P0, tx_addr, sizeof(tx_addr)); // Endereço do pipe 1

  nrf_write_register(NRF24_REG_RF_CH, NRF24_CHANNEL);
  nrf_write_register(NRF24_REG_RX_PW_P0, NRF24_PAYLOAD_SIZE); // Tamanho do payload para pipe 0
  // Habilita ACK automático para pipe 0
  uint8_t en_aa = 0x01; // ERX_P0 = (1 << 0)
  nrf_write_register(NRF24_REG_EN_AA, en_aa);
  // Configura o pipe 0 para receber ACK
  uint8_t en_rxaddr = 0x01; // Habilita apenas pipe 0
  nrf_write_register(NRF24_REG_EN_RXADDR, en_rxaddr);
  // Setup retransmissão: 1500us delay, 15 retransmissões
  uint8_t setup_retr = (5 << 4) | 15;
  nrf_write_register(NRF24_REG_SETUP_RETR, setup_retr);
  // RF_SETUP: configura potência e taxa de dados
  uint8_t rf_setup = (3 << 1) | (kbps == 250 ? (1 << 5) : (kbps == 2000 ? (1 << 3) : (0 << 3)));
  nrf_write_register(NRF24_REG_RF_SETUP, rf_setup);

  // CONFIG: PWR_UP=1, PRIM_RX=0 (modo TX), CRC habilitado (2 bytes)
  uint8_t config = 0x0E; // 0b00001110
  nrf_write_register(NRF24_REG_CONFIG, config);
  // Espera o rádio estabilizar
  _delay_ms(2);
  uint8_t verify = nrf_read_register(NRF24_REG_CONFIG);
  printf("tx_mode_verification: NRF CONFIG REGISTER = 0x%02X\n", verify);
  _delay_ms(2);
  PMODE = TXMODE; // Modo TX
}
```

Figura 12. Configuração do modo `PTX` do módulo nRF24L01+.

dedicados para escrita (`CMD_W_TX_PAYLOAD`) e leitura (`CMD_R_RX_PAYLOAD`). No contexto do sistema desenvolvido, as funções de alto nível `nrf24_send()` e `nrf24_receive()`, que também foram implementadas neste trabalho, utilizam internamente essas rotinas para realizar a comunicação entre os módulos transceptores.

Após tentar realizar uma transmissão com apenas um módulo conectado, o valor lido foi `0x10`, indicando que o bit `MAX_RT` foi ativado, ou seja, o número máximo de tentativas de retransmissão foi atingido sem receber ACK. Esses resultados confirmam que a transmissão foi tentada e que o rádio respondeu com estados coerentes, mesmo sem sucesso no envio, o que era esperado dado que o módulo receptor ainda não estava funcionando plenamente.

5. Resultados e Discussão

5.1. Resultado de escrita e leitura no registrador `CONFIG`

A Figura 14 apresenta o resultado do teste de escrita e leitura do registrador `CONFIG` (0x00) do módulo nRF24L01+, realizado via comunicação SPI. O valor lido inicialmente foi `0x0F` (0000 1111), que corresponde à configuração inicial do dispositivo, uma vez que, para este teste, foi inicializado no modo receptor primário (PRX). Esse valor indica que o módulo está com CRC habilitado (bit 3 = 1), utilizando CRC de 2 bytes (bit 2 = 1), está ligado (bit 1 = 1) e configurado para receber dados (bit 0 = 1).

Durante os testes, valores como `0x7F` (0111 1111) e `0x2A` (0010 1010) foram escritos e lidos corretamente. Por exemplo, o valor `0x7F` indica que as máscaras de interrupção (bits 6, 5 e 4) estão todas ativadas, enquanto o CRC está habilitado e o módulo

```c
void nrf_write_payload(const uint8_t *data, uint8_t len) {
  if (len > NRF24_PAYLOAD_SIZE) len = NRF24_PAYLOAD_SIZE; // Limita o tamanho do payload

  CSN_LOW(); // Ativa CSN
  _delay_us(10); // Espera um pouco para garantir que CSN esteja baixa
  spi_transfer(CMD_W_TX_PAYLOAD); // Comando W_TX_PAYLOAD
  for (uint8_t i = 0; i < len; i++) {
    spi_transfer(data[i]);
  }
  printf("Wrote %d bytes to payload\n", len);
  CSN_HIGH(); // Desativa CSN
}

uint8_t nrf_read_payload(uint8_t *data, uint8_t len) {
  if (len > NRF24_PAYLOAD_SIZE) len = NRF24_PAYLOAD_SIZE; // Limita o tamanho do payload

  CSN_LOW(); // Ativa CSN
  _delay_us(10); // Espera um pouco para garantir que CSN esteja baixa
  spi_transfer(CMD_R_RX_PAYLOAD); // Comando R_RX_PAYLOAD
  for (uint8_t i = 0; i < len; i++) {
    data[i] = spi_transfer(CMD_NOP);
  }
  printf("Read %d bytes from payload\n", len);
  CSN_HIGH(); // Desativa CSN
  nrf_clear_interrupts(); // Limpa interrupções
  return len;
}
```

Figura 13. Funções de escrita e leitura de *payload* no `nRF24L01+` via SPI.

está em modo PRX. Já o valor `0x2A` configura o módulo com as máscaras de interrupção apenas parcialmente ativadas e com o módulo em modo transmissor (bit 0 = 0).

5.2. Resultados Práticos

A portabilidade foi bem-sucedida para aplicações básicas, o que demonstra que o Contiki-NG pode operar no Arduino Uno com suporte ao rádio nRF24L01+. A arquitetura modular e o sistema de build permitiram uma adaptação progressiva, embora com limitações evidentes em relação a desempenho e capacidade de memória.

Essa portabilidade representa um avanço significativo em termos de acessibilidade e democratização de tecnologias para redes de sensores sem fio [Merino Calero 2019]. Sua realização demonstra a viabilidade de empregar plataformas de baixo custo e amplamente disponíveis, como o Arduino Uno, em experimentos com sistemas embarcados voltados para a Internet das Coisas, mesmo sob a limitação de recursos e utilizando sistemas operacionais com suporte à comunicação em rede.

Além disso, a adaptação do Contiki-NG para essa arquitetura abre caminho para seu uso em contextos educacionais e experimentais, reduzindo barreiras de entrada para estudantes, pesquisadores e desenvolvedores independentes [Martins Gomes and Baunach 2021]. Essa iniciativa também viabiliza a construção de redes de sensores baseadas em tecnologias abertas, reprodutíveis e de baixo custo, alinhando-se aos princípios de desenvolvimento sustentável e inclusivo na área de sistemas distribuídos.

5.3. Desafios Enfrentados

Apesar da transmissão ter sido corretamente iniciada pelo PTX, ainda não foi possível confirmar a recepção do pacote pelo PRX. Isso pode estar relacionado à sincronização

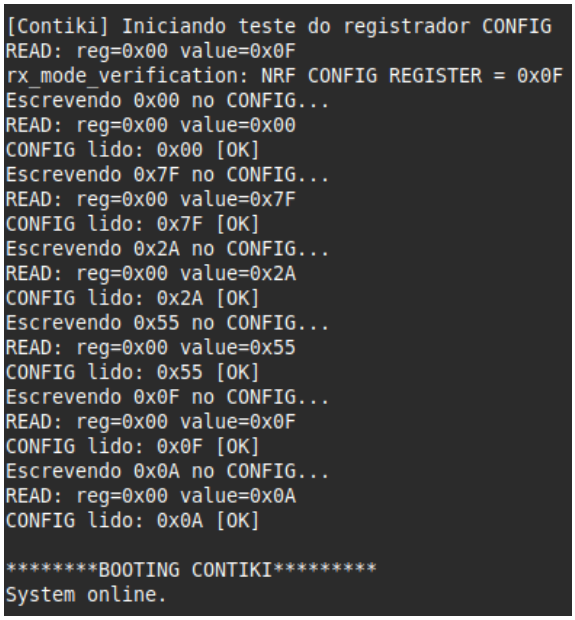


Figura 14. Resultado do teste de escrita e leitura do registrador `CONFIG` via interface SPI.

dos endereços, canal, temporização de CE, ou à ausência de tratamento adequado do sinal IRQ no receptor.

Durante o processo, alguns desafios foram observados:

- Limitações de memória RAM e Flash no ATmega328P para compilar exemplos como o `unicast` da pasta `examples/rime`.
- Dificuldade na leitura estável do registrador STATUS do nRF24L01+ durante tentativas de recepção e transições IRQ;
- Adaptação da comunicação SPI às particularidades do Contiki-NG e do rádio.

6. Conclusão e Trabalhos Futuros

O driver implementado para o módulo nRF24L01+ demonstra-se funcional nas operações fundamentais de comunicação SPI, leitura e escrita de registradores, além da transmissão básica com *payload*. Foram realizados testes com leitura consistente do registrador `STATUS`, escrita e verificação do registrador `CONFIG`, além de transmissões utilizando a fila TX do rádio (`TX FIFO`) através do comando `W_TX_PAYLOAD`. Esses testes confirmam que a base da comunicação entre o microcontrolador ATmega328P e o transceptor sem fio está devidamente estabelecida.

Apesar dos avanços, este trabalho ainda não possui suporte completo à recepção, o que limita a operação bidirecional esperada em aplicações reais de redes de sensores sem fio. Nesse sentido, diversos aprimoramentos são propostos como trabalhos futuros:

- Diagnosticar e corrigir a falha atual na recepção de pacotes no modo PRX;
  - Implementar o tratamento apropriado dos sinais IRQ do nRF24L01+ por meio de interrupções externas no Arduino Uno;
- Validar o fluxo completo de envio com confirmação automática (ACK) e recepção do *payload* no receptor;

- Criar camadas de abstração compatíveis com os protocolos nativos do Contiki-NG, facilitando a integração do módulo como dispositivo de rede;
- Investigar e aplicar técnicas de Inteligência Artificial (IA) e Aprendizado de Máquina (ML) para otimização do consumo energético do rádio, especialmente em cenários de *duty cycling* e decisões autônomas de ativação do transceptor.

O processo de portabilidade do Contiki-NG para o Arduino Uno, principalmente a integração com o rádio nRF24L01+, demonstrou-se um desafio complexo, mas também uma oportunidade especial de aprendizado. A necessidade de entender profundamente a arquitetura do sistema operacional, os detalhes da interface SPI e o funcionamento do rádio possibilitou um desenvolvimento significativo em diversas áreas de conhecimento. Essa experiência reforçou a importância da paciência e da análise detalhada de documentação técnica para projetos embarcados.

Um dos pontos mais interessantes foi a modularidade do Contiki-NG, que possibilitou a adaptação incremental do sistema para uma plataforma com recursos muito limitados como o Arduino Uno. Além disso, a experiência prática com o rádio nRF24L01+ evidenciou a complexidade dos protocolos de comunicação em baixo nível.

Embora os resultados iniciais tenham sido promissores, ainda há muito a avançar, especialmente no tratamento da recepção e na otimização do uso de memória. Além do aprofundamento da implementação do driver do rádio e ampliar a estabilidade da comunicação. Este trabalho mostrou na prática como diferentes camadas tecnológicas interagem para viabilizar soluções IoT acessíveis e eficientes.

Referências

Aazam, M., Zeadally, S., and Harras, K. A. (2018). Fog computing architecture, evaluation, and future research directions. *IEEE Communications Magazine*, 56(5):46–52.

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., and Ayyash, M. (2015). Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, 17(4):2347–2376.

Baccelli, E., Hahm, O., Günes, M., Wählisch, M., and Schmidt, T. C. (2018). Riot: An open source operating system for low-end embedded devices in the iot. *IEEE Internet of Things Journal*, 5(6):4428–4440.

Cooja, C.-O. (2017). Cooja simulator. `https://github.com/contiki-os/cooja`. Acesso em: 04 ago. 2025.

Crossbow Technology (2006). Micaz wireless measurement system datasheet. `https://courses.ece.ubc.ca/494/files/MICAz_Datasheet.pdf`. Acesso em: 04 ago. 2025.

Gadekar, S., Kolpe, G., Rutuja, G., Fatate, V., Bhosale, B., Chate, S., and Lad, A. (2021). Arduino uno-atmega328 p microcontroller based smart systems. In *Proceedings of the 3rd International Conference on Communication & Information Processing (ICCIP)*.

Ilin, P. (2007). contiki-arduino - port of contiki for arduino boards. `https://github.com/PIlin/contiki-arduino`. Acesso em: jul. 2025.

InsideGadgets (2015). Repository with nrf24l01 examples for avr. `https://github.com/insidegadgets/nRF24L01`. Acesso em: ago. 2025.

InsideGadgets, A. (2012). Using the nrf24l01 wireless module. https://www.insidegadgets.com/2012/08/22/using-the-nrf24l01-wireless-module/. Acesso em: ago. 2025.

Khan, L. U., Yaqoob, I., Tran, N. H., Kazmi, S. M. H., Dang, T. T. A., and Hong, C. S. (2019). Wireless technologies for smart energy grid: Recent advances and future challenges. *IEEE Communications Surveys & Tutorials*, 21(1):998–1021.

Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., Gay, D., Hill, J., Welsh, M., Brewer, E., and Culler, D. (2003). Tinyos: An operating system for sensor networks. *Ambient Intelligence*, pages 115–148.

Martins Gomes, R. and Baunach, M. (2021). A study on the portability of iot operating systems. In *Tagungsband des FG-BS Frühjahrstreffens 2021*, pages 10–18420. Gesellschaft für Informatik eV.

Merino Calero, P. (2019). *Validación Hardware de plataforma para IoT en el Edge y porting de Contiki-NG*. PhD thesis, Industriales.

Microchip Technology (2016). Atmega328p 8-bit avr microcontroller datasheet. https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf. Acesso em: 04 ago. 2025.

Moteiv Corporation (2006a). Telosb mote platform datasheet. https://arcb.csc.ncsu.edu/~mueller/rt/rt11/readings/projects/g4/datasheet.pdf. Acesso em: 04 ago. 2025.

Moteiv Corporation (2006b). Tmote sky datasheet. http://www.crew-project.eu/sites/default/files/tmote-sky-datasheet.pdf. Acesso em: 04 ago. 2025.

Nordic (2007). nrf24l01+ product specification v1.0. https://cdn.sparkfun.com/assets/3/d/8/5/1/nRF24L01P_Product_Specification_1_0.pdf. Acesso em: 04 ago. 2025.

Oikonomou, G., Duquennoy, S., Elsts, A., Eriksson, J., Tanaka, Y., and Tsiftes, N. (2022). The contiki-ng open source operating system for next generation iot devices. *SoftwareX*, 18:101089.

Palattella, M. R., Accettura, N., Vilajosana, X., Watteyne, T., Grieco, L. A., Boggia, G., and Dohler, M. (2016). Internet of things in the 5g era: Enablers, architecture, and business models. *IEEE Journal on Selected Areas in Communications*, 34(3):510–527.

Rawat, P., Singh, K. D., Chaouchi, H., and Bonnin, J. M. (2014). Wireless sensor networks: A survey on recent developments and potential synergies. *The Journal of Supercomputing*, 68:1–48.

Texas Instruments (2007). Cc2420: 2.4 ghz ieee 802.15.4 / zigbee-ready rf transceiver. https://www.ti.com/lit/ds/symlink/cc2420.pdf. Acesso em: 04 ago. 2025.

Texas Instruments (2018). Msp430g2553 mixed signal microcontroller datasheet. https://www.ti.com/lit/ds/symlink/msp430g2553.pdf. Acesso em: 04 ago. 2025.

Zanella, A., Bui, N., Castellani, A., Vangelista, L., and Zorzi, M. (2014). Internet of things for smart cities. *IEEE Internet of Things Journal*, 1(1):22–32.

Zolertia (2015). Zolertia z1 datasheet revision c. `https://github.com/Zolertia/Resources/blob/master/Z1/Hardware/Revision%20C/Datasheets/Zolertia%20Z1%20datasheet%20Revision%20C.pdf`. Acesso em: 04 ago. 2025.