Universidade Tecnológica Federal do Paraná

Curso de Engenharia Eletrônica

Felipe Silva

Patrick Grochewski

Tulio Sanches Fogagnoli

Relatório Final - Sistemas Operacionais

Curitiba

2025

Felipe Silva
Patrick Grochewski
Tulio Sanches Fogagnoli

Relatório Final - Sistemas Operacionais

Trabalho escrito para registrar as evidências do projeto realizado na disciplina ELF66(Sistemas Operacionais).

Universidade Tecnológica Federal do Paraná

Curso de Engenharia Eletrônica

Curitiba

2025

Sumário

INTRODUÇÃO

Este projeto descreve o desenvolvimento de um sistema embarcado para a geração e controle de um sinal de modulação por largura de pulso (PWM). Utilizando um microcontrolador STM32 Cortex-M4 e o sistema operacional de tempo real FreeRTOS, a solução permite a configuração remota de parâmetros como frequência e duty cycle via PC. O sistema implementa também um mecanismo de feedback, realizando a leitura do próprio sinal gerado através de um conversor analógico-digital (ADC) para monitoramento e controle em malha fechada.

OBJETIVO GERAL

O desenvolvimento de um sistema embarcado para gerar e controlar uma onda PWM, permitindo que seus parâmetros, como frequência e duty cycle, sejam configurados por um computador (PC).

Além disso, o sistema deve ser capaz de realizar a leitura do próprio sinal PWM que ele gera (após uma etapa de filtragem), utilizando uma entrada ADC para fornecer um feedback do sinal que foi produzido.

Objetivos Principais

- Desenvolvimento de Sistema Embarcado: Criar um sistema para gerar e controlar uma onda PWM (Modulação por Largura de Pulso).
- Configuração via PC: Permitir que os parâmetros da onda (como frequência e duty cycle) sejam configurados através de um computador.
- Feedback do Sinal: O próprio sistema fará a leitura do sinal PWM que ele gerou.
- Leitura com ADC: Essa leitura será feita utilizando uma entrada ADC (Conversor Analógico-Digital) após o sinal passar por uma filtragem.

Objetivos Específicos

- Plataforma: Implementar o sistema utilizando um microcontrolador STM32 Cortex-M4 e o sistema operacional de tempo real FreeRTOS.
- Multitarefa: Gerenciar no mínimo 4 threads com comunicação e sincronização entre elas (usando filas e semáforos).
- Comunicação: Comunicar com o PC via Sockets TCP/IP para realizar a configuração e a visualização dos dados.
- Controle: Controlar um atuador por meio do sinal PWM gerado.
- Leitura de Feedback: Ler o sinal PWM filtrado usando um canal ADC do STM32.
- Envio de Dados: Enviar o valor lido e interpretado do PWM de volta para o PC.

1 Desenvolvimento

O desenvolvimento do projeto iniciou-se com a escolha dos componentes principais que iriam compor o projeto. Escolhemos qual hardware usaríamos, desde os conversores até o microcomputador principal, os botões, encoders, caixa e conectores.

1.1 HARDWARE

Os componentes de hardware se comunicam para gerar o sinal, conectar-se à rede e realizar a leitura de feedback, conforme o diagrama da Figura 1. Os componentes principais incluem:

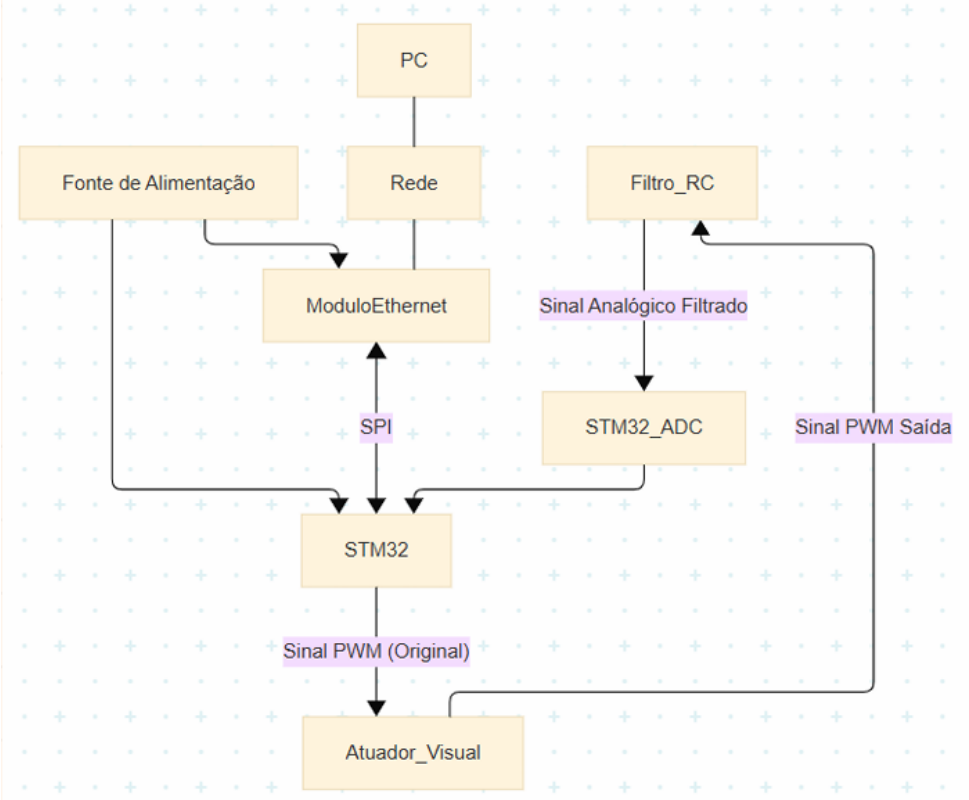

Figura 1 – Arquitetura do Hardware

- Microcontrolador STM32F4xx: O componente central que executa toda a lógica embarcada.
- Módulo Ethernet SPI: Um módulo como o W5500 é utilizado para estabelecer a comunicação em rede. Ele se conecta ao microcontrolador via interface SPI.

- Filtro RC Passa-Baixa: Essencial para o loop de feedback, converte o sinal PWM para uma tensão analógica que pode ser lida pelo ADC.
- Atuador Visual: Um componente, como um LED, usado para fornecer uma representação visual do sinal PWM gerado.
- PC: Funciona como a interface para o usuário, permitindo a configuração do sistema e a visualização dos dados de feedback.
- Fonte de Alimentação: Fornece energia para a placa de desenvolvimento e os módulos conectados.

O MCU escolhido é o STM32F407VET6, que possui um núcleo ARM Cortex-M4 com Unidade de Ponto Flutuante (FPU) e instruções DSP. Seus periféricos integrados, como Timers com capacidade de geração PWM, múltiplos canais ADC de 12 bits e interface SPI, são essenciais para o projeto.

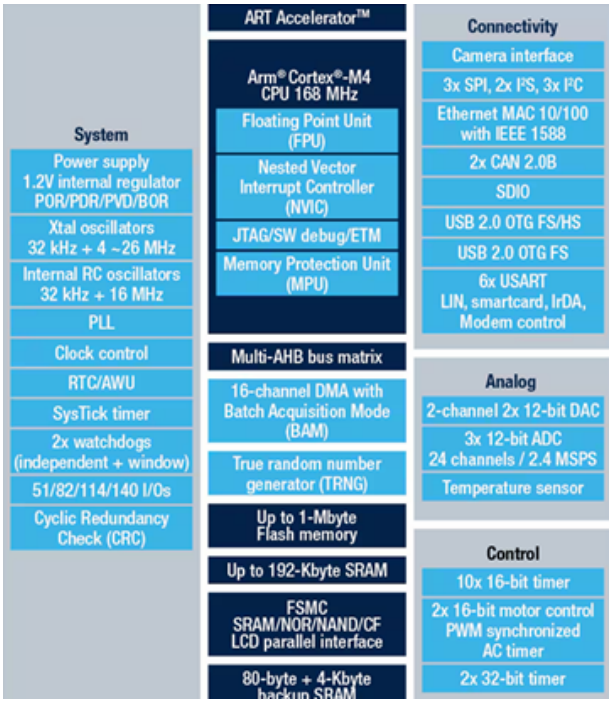

Figura 2 – Arquitetura STM32

Para que o sinal digital PWM possa ser lido por um canal ADC, que espera uma tensão analógica, é necessário um filtro. Foi utilizado um filtro RC passa-baixa, composto por um resistor e um capacitor. Este filtro atenua as altas frequências do sinal pulsado, resultando em uma tensão DC média que é proporcional ao duty cycle do PWM. A saída do PWM é conectada ao filtro, e a tensão sobre o capacitor é conectada à entrada do ADC.

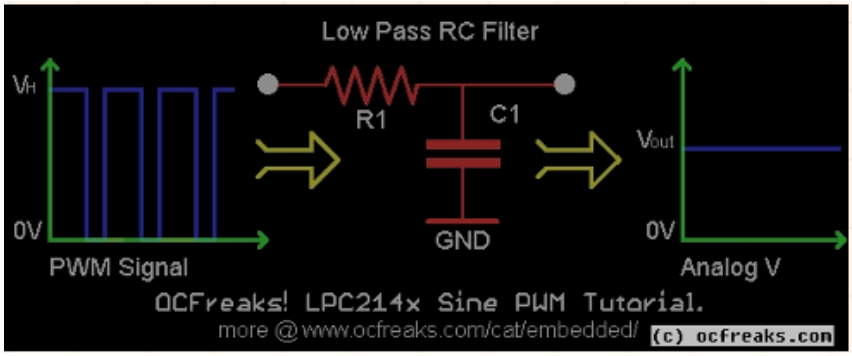

Figura 3 – Filtro RC

Módulo de Comunicação: A comunicação com o PC é feita por um módulo conversor SPI-Ethernet, como o W5500, que possui uma pilha TCP/IP embarcada, aliviando o processamento do MCU principal.

Atuador PWM: O sinal PWM é gerado pelos Timers do STM32 e é utilizado para controlar um atuador, como o brilho de um LED para visualização. Este mesmo sinal serve como entrada para o loop de feedback.

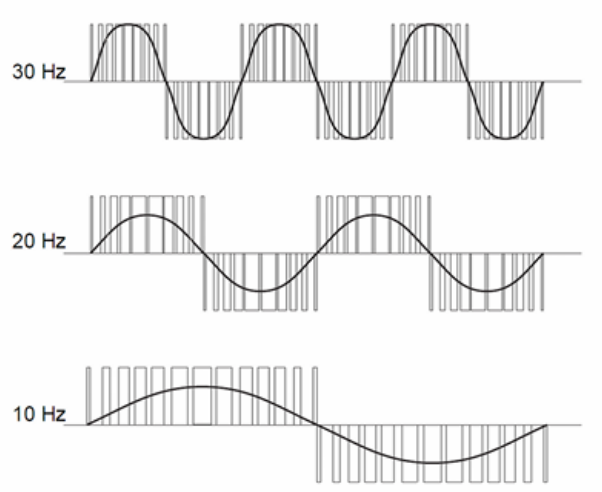

Figura 4 – PWM

Figura 5 – Módulo Ethernet W5500

1.2 SOFTWARE

A arquitetura de software, executada sob o Kernel do FreeRTOS , é baseada em múltiplas threads que gerenciam as diferentes funcionalidades do sistema. O projeto exige o gerenciamento de no mínimo 4 threads com comunicação e sincronização. As threads principais são:

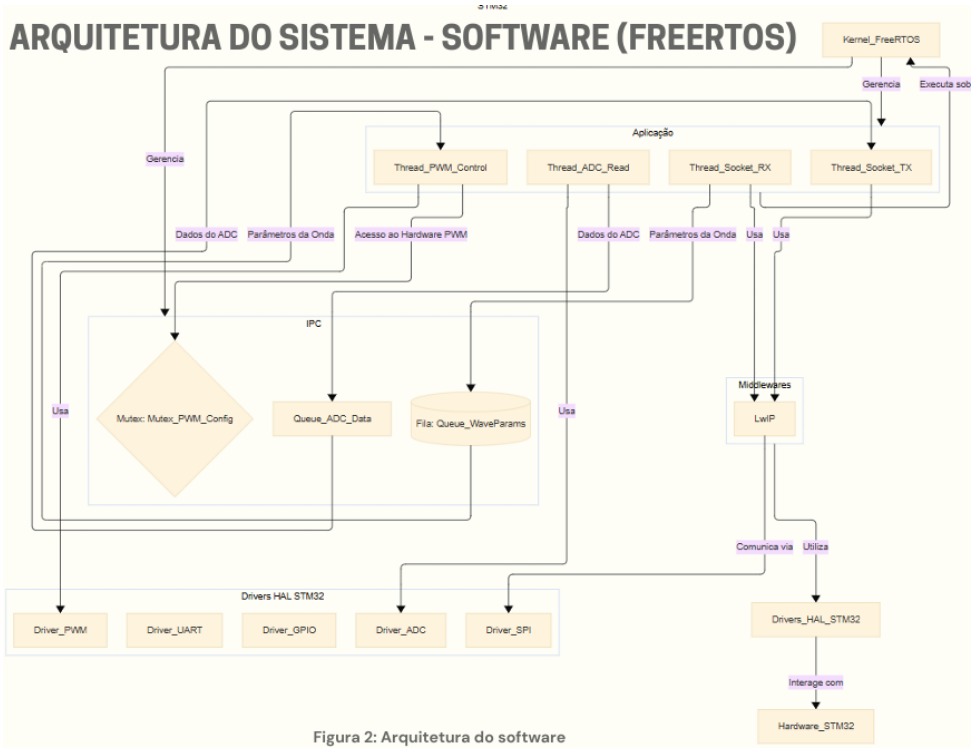

Figura 6 – Arquitetura Software

- ThreadSocketRX: Responsável por receber comandos de configuração do PWM enviados pelo PC..
- ThreadPWMControl: Recebe os parâmetros da onda e configura o hardware do timer PWM do microcontrolador.
- ThreadADCRead: Realiza a leitura periódica do sinal PWM que foi filtrado, utilizando o periférico ADC.
- ThreadSocketTX: Envia o valor do PWM lido (feedback) e outras informações de status de volta para o PC.

A comunicação entre as threads (IPC) é realizada com filas e mutex para garantir a integridade dos dados e o acesso seguro aos recursos de hardware.

1.2.1 Threads e Sincronização

O sistema operacional FreeRTOS gerencia as tarefas, a comunicação entre processos (IPC) e a sincronização. As quatro threads principais têm responsabilidades e prioridades distintas:

1 - socketreceivetask (Prioridade Média): Aguarda a conexão do PC e recebe os parâmetros de configuração da onda (frequência, duty cycle). Envia esses parâmetros para uma fila chamada QueueWaveParams.

2 - pwmcontroltask (Prioridade Alta): Aguarda a chegada de novos parâmetros na QueueWaveParams. Utiliza um Mutex (MutexPWMConfig) para garantir acesso exclusivo e seguro aos registradores do hardware PWM durante a reconfiguração.

3 - adcreadtask (Prioridade Baixa-Média): Periodicamente, lê o valor do canal ADC conectado à saída do filtro RC. Envia o valor digital lido para a fila QueuePWMFeedback-Data.

4 - socketsendtask (Prioridade Média): Aguarda dados na QueuePWMFeedbackData e envia o valor do feedback do PWM de volta para o PC, além de outras mensagens de status.

1.2.2 Comunicação Inter-Processos (IPC) e TCP/IP

Filas de Mensagens QueueWaveParams: Transfere os parâmetros da onda da thread de recepção para a thread de controle do PWM.

QueuePWMFeedbackData: Transfere o valor do PWM lido da thread do ADC para a thread de envio para o PC.

Mutex:

MutexPWMConfig: Protege o acesso concorrente aos recursos de hardware do PWM, evitando condições de corrida.

Pilha TCP/IP: Utiliza-se a pilha LwIP (integrada com o STM32CubeMX) ou a pilha do próprio módulo W5500. A API de Sockets é usada para criar um servidor no STM32, que aguarda conexões e realiza a troca de dados com o cliente no PC.

1.2.3 Fluxo de Operação

- Inicialização: O sistema inicializa os clocks, periféricos (GPIOs, ADC, PWM, SPI), o FreeRTOS com suas tarefas e IPCs, e a camada de rede, colocando o socket do servidor em modo de escuta (listen).
- Conexão: Uma aplicação cliente no PC se conecta ao servidor no STM32.

- Configuração: O usuário define os parâmetros do PWM no PC, que os envia para o STM32 via socket.
- Processamento: A socketreceivetask recebe os dados e os repassa para a pwmcontroltask via fila, que então ajusta o sinal PWM.
- Feedback: A adcreadtask lê continuamente o valor filtrado do PWM e o envia para a socketsendtask via outra fila.
- Visualização: A socketsendtask envia o valor de feedback de volta para o PC, onde ele pode ser exibido para o usuário.
- Ciclo Contínuo: Os passos de 3 a 6 se repetem, permitindo uma interação contínua e o monitoramento em tempo real.

2 Conclusão

O projeto propõe com sucesso um sistema embarcado robusto para a geração, controle remoto e leitura de feedback de um sinal PWM. A arquitetura atende aos requisitos definidos ao empregar um microcontrolador STM32 Cortex-M4, o sistema operacional FreeRTOS, comunicação via Sockets Ethernet e um loop de feedback com ADC e filtro RC. O uso de uma arquitetura multitarefa com quatro threads principais e mecanismos de IPC (filas e mutex) garante a modularidade, responsividade e escalabilidade do sistema.

3 Referências

1.Cortex-M4 - Arm Developer, acessado em maio 27,2025

2.STMicroelectronics STM32L4 32-Bit Low-Power Microcontrollers +FPU - Mouser, acessado em maio 27, 2025,

3.Working with STM32 and SPI (Serial Peripheral Interface): Send a byte - EmbeddedExpertIO, acessado em maio 27, 2025, https://blog.embeddedexpert.io/?p=466
4.FreeRTOS kernel fundamentals - AWS Documentation, acessado em maio 27, 2025, https://docs.aws.amazon.com/freertos/latest/userguide/dev-guide-freertos-kernel.html

5.The Architecture of FreeRTOS (reproduced from [80]). | Download Scientific Diagram, acessado em maio 27, 2025, https://www.researchgate.net/figure/The-Architecture-of-FreeRTOS-reproduced-from-80fig7328031606

6.The Design of FreeRTOS-Plus-TCP: Part 1 - The Big Picture, acessado em maio 27, 2025, https://www.freertos.org/Community/Blogs/2024/the-design-of-freertos-plus-tcp-part1-the-big-picture

7.Using the SPI interface on STM32 devices – VisualGDB Tutorials, acessado em maio 27, 2025, https://visualgdb.com/tutorials/arm/stm32/spi/

8.Princípios básicos do kernel do FreeRTOS - FreeRTOS, acessado em maio 27, 2025, https://docs.aws.amazon.com/ptbr/freertos/latest/userguide/dev-guide-freertos-kernel.html

9.FreeRTOS - Guia do usuário - Amazon.com, acessado em maio 27, 2025,

10.LwIP + FreeRTOS not working on STM32F746 - STMicroelectronics Community, acessado em maio 27, 2025, https://community.st.com/t5/stm32cubemx-mcus/lwip-freertos-not-working-on-stm32f746/td-p/28684