Depuração de Sistemas em Tempo Real:

Uma Abordagem de Instrumentação de Código para Testes de Sistemas Críticos

Sandro Fadiga

Orientador: Prof. Dr. Glauco Augusto de Paula Caurin

2025

Escola de Engenharia de São Carlos Universidade de São Paulo - USP

1. Introdução

Introdução

- Contextualização
- Objetivos
- Justificativa
- Desenvolvimento
- Testes e Resultados
- Conclusão

2. Fundamentação Teórica

2.1 Contextualização

Figura 1: Foto de um típico ambiente de ensaios integrados (Iron Bird).

2.2 Objetivos

Objetivo Geral

Desenvolver e validar um protocolo leve de depuração (peek/poke/telemetria) para sistemas embarcados críticos com mínima intrusão temporal.

Objetivos Específicos

- Especificar protocolo binário eficiente
- Implementar no microcontrolador e host
- Avaliar impacto temporal em diferentes taxas

2.3 Justificativa

Tópicos abordados:

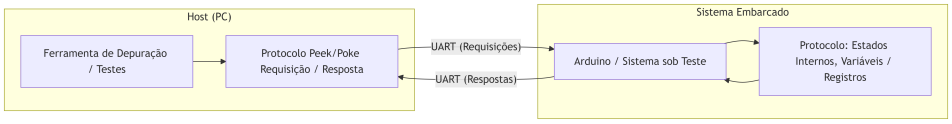
- Software embarcado crítico
- Tempo real e determinismo
- Setores de alta criticidade
- Testes e certificação (DO-178C, ISO 26262, IEC 61508/62304)

3 Desenvolvimento

3.1 Protocolo

- Arquitetura do Sistema
- Especificação do protocolo
- Máquina de Estados
- Cliente Host – Desenvolvimento
- Considerações de extensibilidade

3.2 Arquitetura do Sistema



3.2 Especificação do protocolo

Examplo do Comando Peek:

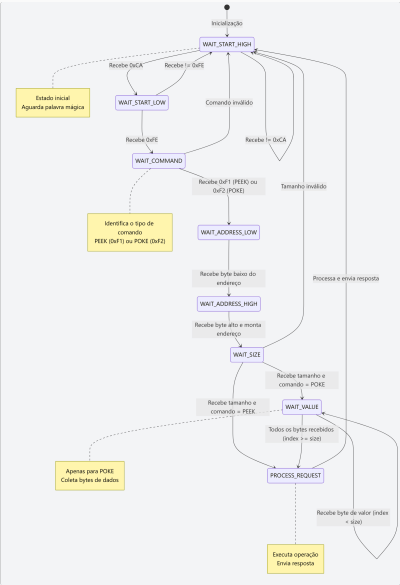
```
CA FE F1 04 01 02
```

Decomposição:

- `CA FE`: Palavras mágicas
- `F1`: Comando PEEK
- `04`: Byte baixo do endereço (LSB)
- `01`: Byte alto do endereço (MSB)
- `02`: Tamanho (2 bytes)

Nota: o endereço `0x0104` é transmitido como `04 01` (little-endian).

3.3 Máquina de Estados



3.4 Cliente Host – Desenvolvimento

- Ferramenta escrita em Python/Qt
- Usa `pyelftools` ou `libdwarf`
- Resolve símbolos automaticamente a partir do arquivo elf/dwarf
- Comandos simples: `peek 'variavel'`, `poke 'variavel' = 42`

3.5 Considerações de extensibilidade

- Checagem de erros
- Monitoramento/telemetria contínua
- Extensão do protocolo (novos comandos)
- Segurança (prevenção do uso indesejado)
- Uso em ensaios em vôo

4. Testes e Resultados

4.1 Metodologia e Testes

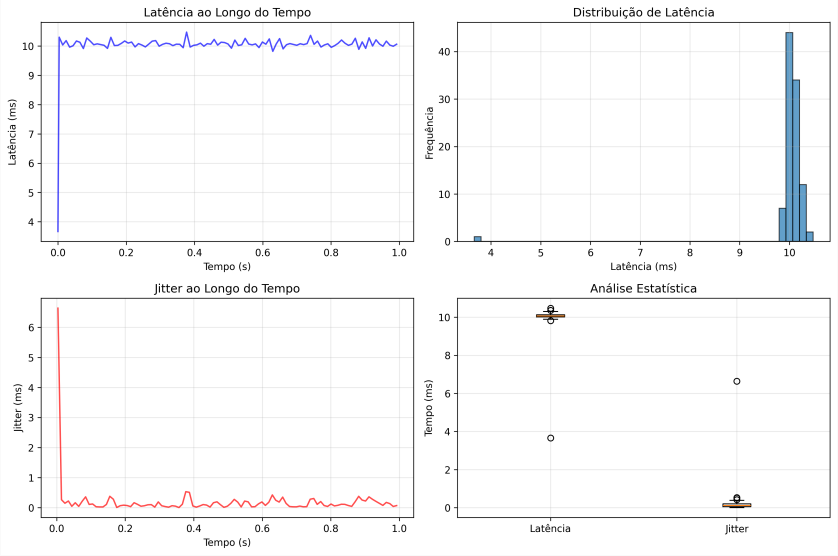
Hardware

- Plataforma: Arduino UNO e "frame locking"
- Instrumentação de código ferramenta Host
- Instrumentação de código embarcado
- Osciloscópio FNIRSI DSO-153

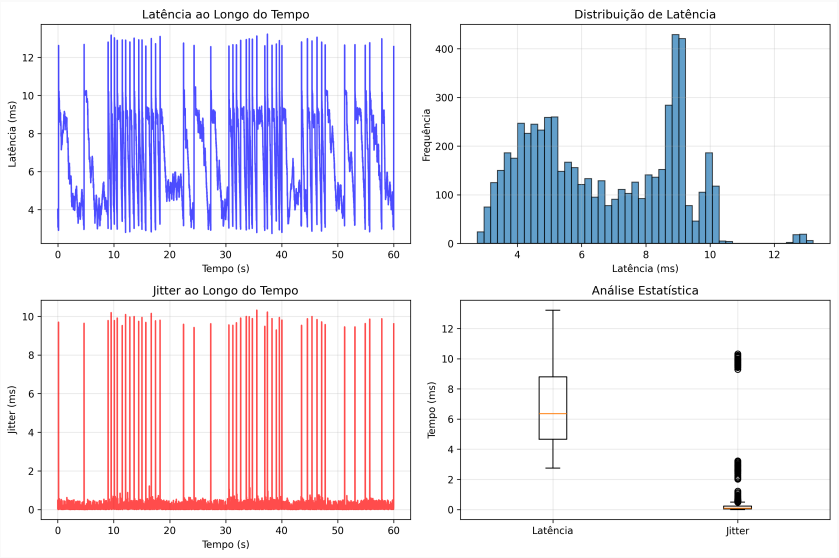
Cenários testados

- Teste de Latência
- Teste de estresse
- Teste de Rajada (burst)
- Telemetria contínua a 10 Hz, 100 Hz e 1 kHz

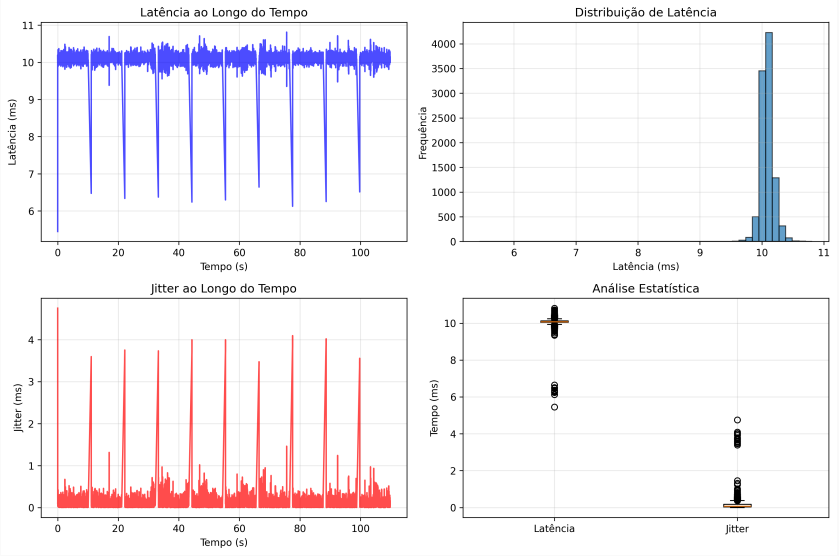
4.2 Resultados – Teste de Latência



4.3 Resultados – Teste de estresse



4.4 Teste de Rajada (burst)



4.5 Telemetria contínua a 10 Hz, 100 Hz e 1 kHz

Tabela 1: Síntese dos resultados dos testes com osciloscópio

Métrica	**10 Hz**	**100 Hz**	**1 kHz**
Frequência de detecção dos pulsos de comando (PIN_BUSY)			
Frequência Medida	14.81 Hz	99.13 Hz	99.13 Hz
Período	43.34 ms	10.09 ms	10.09 ms
Largura Pulso	720 µs	732 µs	730 µs
Duty Cycle	1.76%	7.26%	7.24%
Medição de frame rate			
Frequência	49.55 Hz	49.55 Hz	49.70 Hz
Período Total	20.18 ms	20.18 ms	20.12 ms
Período Loop	10.09 ms	10.09 ms	10.06 ms
Duty Cycle	50.02%	50.00%	50.00%
Jitter perceptível no loop	Não observado	Não observado	Não observado

5. Conclusão

5.1 Conclusão

Objetivos atingidos

✓Protocolo leve especificado e implementado
✓Funciona em hardware real (Arduino UNO)
✓Overhead temporal quantificado e baixo até 100 Hz
✓Ferramenta host com protocolo funcional

5.2 Limitações Observadas

- Ausência de monitoramento/telemetria contínua
- Sem mecanismos de integridade
- Sem travas de segurança ou validação de dados
- Dependência de UART (taxa máxima 115200 bps)

5.3 Trabalhos Futuros

- Portar para RTOS utilizando MCU com maior capacidade
- Adicionar monitramento/telemetria contínua
- Implementar CRC32 + ACK/NACK
- Suporte a múltiplos alvos simultâneos (ferramenta host)

Obrigado

Perguntas?

Código-Fonte disponível em:
`https://github.com/sfadiga/destra`