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 simples 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:

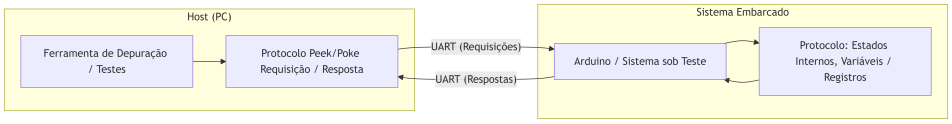
- Tempo real e determinismo
- Software embarcado, Software crítico
- Setores de alta criticidade
- Testes e certificação

3 Desenvolvimento

3.1 Protocolo

- Caraterísticas desejáveis
- Meios de transmissão
- "Stateful" vs "Stateless"
- Telemetria, Sincronização
- Considerações de extensibilidade

3.2 Arquitetura do Sistema



3.3 Especificação do comando peek

Examplo do Comando Peek:

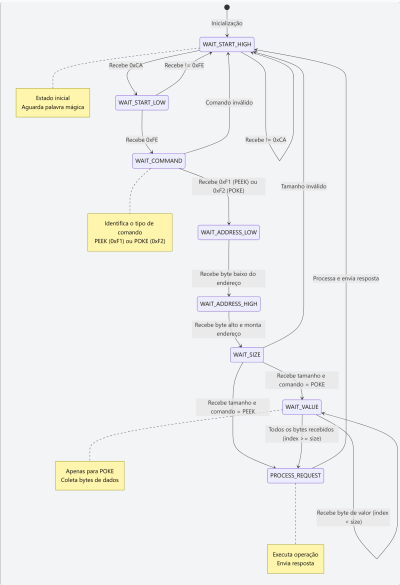
```
CA FE F1 04 01 02
```

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).

3.4 Máquina de Estados



3.5 Cliente Host – Desenvolvimento

- Ferramenta escrita em Python
- Inteface gráfica escrita em PySide (Qt)
- Escrita em poucas linhas de código (3 scripts .py)
- Usa `pyelftools`, resolve símbolos automaticamente a partir do arquivo elf/dwarf
- Comandos simples: `peek 'variavel'`, `poke 'variavel' = 42`

3.6 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 vs hard realtime
- Cenários e limitações
- Instrumentação de código ferramenta Host
- Instrumentação de código embarcado + buffer interno
- Instrumentação de código embarcado + Osciloscópio

Cenários testados

- Cenário 1: Teste de Latência
- Cenário 1: Teste de Estresse
- Cenário 1: Teste de Rajada (burst)
- Cenário 2: Teste com osciloscópio: a 10 Hz, 100 Hz e 1 kHz

4.2 Resultados – Teste de Latência

Métrica (Firmware)	Valor
Total de Amostras	99
Frame Jitter Médio (ms)	0.0022
Frame Jitter Mediana (ms)	0.0000
Desvio Padrão (ms)	0.0035
Tempo de Comando Médio (ms)	0.732
Gaps no Frame Counter	0
Gaps na Sequência de Comandos	0

Tabela 1: Dados de performance embarcada (teste de latência)

4.3 Resultados – Teste de Estresse

Métrica (Firmware)	**Valor**
Total de Medições	5.905
Taxa de Erro (%)	0.0
Frame Jitter Médio (ms)	0.0023
Frame Rate (fps)	99.0
Comando Process Time Médio (ms)	0.833
Gaps de Frame Counter	90
Gaps de Command Sequence	0

Tabela 2: Dados de performance embarcada (teste de estresse)

4.4 Teste de Rajada (burst)

Métrica (Firmware)	**Valor**
Total de Amostras	99
Frame Jitter Médio (ms)	0.0021
Frame Rate (fps)	99.0
Comando Process Time Médio (ms)	2.089
Gaps no Frame Counter	0
Gaps na Command Sequence	0

Tabela 3: Dados de performance embarcada (teste de rajada)

4.5 Teste com osciloscópio: a 10 Hz, 100 Hz e 1 kHz

Métrica	**10 Hz**	**100 Hz**	**1 kHz**
Testes de Tempo 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%
Testes de Frame Rate (PIN_FRAME_TOGGLE)			
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

Tabela 4: Síntese dos resultados dos testes com osciloscópio

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
- Implementação utilizando Serial (simples)

5.3 Trabalhos Futuros

- Portar para RTOS utilizando MCU com maior capacidade
- Adicionar monitramento/telemetria contínua
- Implementar controle de integridade (CRC)
- Introduzir uma camada de abstração de hardware
- Suporte a múltiplas plataformas embarcadas

Obrigado

Perguntas?

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