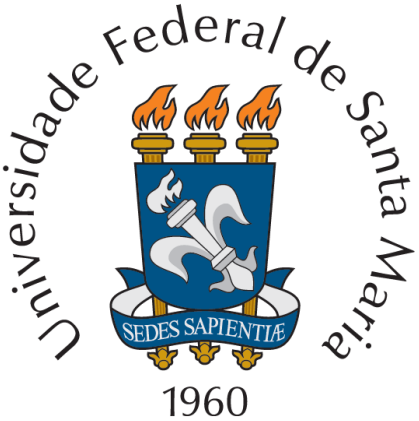
Relatório de Projeto TDD: FSM de Protocolo

Projeto de Sistemas Embarcados

João Vítor Sauzem Real

20 de agosto de 2025



Universidade Federal de Santa Maria
Departamento de Eletrônica e Computação

Professor: Carlos Henrique Barriquello

Sumário

1 Introdução

Este relatório apresenta o desenvolvimento de um módulo para tratamento de um protocolo de comunicação serial, implementado em linguagem C e utilizando a metodologia de Desenvolvimento Orientado a Testes (TDD). O objetivo foi criar uma implementação robusta e exaustivamente testada de uma Máquina de Estados Finitos (FSM) para garantir a confiabilidade na troca de dados.

O protocolo de comunicação segue o formato:

`STX | QTD | DADOS | CHK | ETX`

Onde cada campo é definido como:

- **STX**: Start of Text - Byte de início da transmissão (1 byte, valor `0x02`).
- **QTD**: Quantidade - Número de bytes no campo de dados (1 byte).
- **DADOS**: O conteúdo da transmissão (N bytes).
- **CHK**: Checksum - Verificação de integridade dos dados (1 byte, XOR de todos os bytes de dados).
- **ETX**: End of Text - Byte de fim da transmissão (1 byte, valor `0x03`).

2 Metodologia TDD

O desenvolvimento seguiu o ciclo **Red-Green-Refactor**:

1. **Red:** Escrever um teste que falha para uma nova funcionalidade desejada.
2. **Green:** Escrever o código mínimo necessário para fazer o teste passar.
3. **Refactor:** Melhorar o código existente sem alterar seu comportamento externo.

Este ciclo iterativo guiou a construção de toda a lógica da FSM, garantindo que cada funcionalidade e tratamento de erro fossem validados por um teste específico.

3 Desenvolvimento e Suíte de Testes

O processo TDD nos guiou a criar uma suíte de testes abrangente, que cobriu os cenários essenciais para as FSMs do transmissor e do receptor.

3.1 Testes do Transmissor (TX)

- **Inicialização:** Verificação do estado inicial da FSM.
- **Geração de Pacote Completo:** Validação da geração de um pacote padrão com dados.
- **Geração de Pacote Vazio:** Teste do caso de borda de um pacote sem campo de dados.

3.2 Testes do Receptor (RX)

- **Inicialização:** Verificação do estado inicial da FSM.
- **Pacote Válido:** Teste do caminho feliz, processando um pacote correto.
- **Robustez a Ruído:** Garantia de que a FSM ignora lixo eletrônico antes do início de um pacote.
- **Validação de Checksum:** Verificação da rejeição de pacotes com erro de integridade.
- **Tratamento de Erros de Enquadramento:** Testes para rejeição de pacotes malformados (sem ETX, com STX no meio) ou que excedam a capacidade do buffer.

4 Diagramas de Funcionamento

4.1 Digrama do Transmissor

O diagrama a seguir ilustra as transições de estado da FSM do transmissor, mostrando o processo de montagem da mensagem no padrão do protocolo de comunicação.

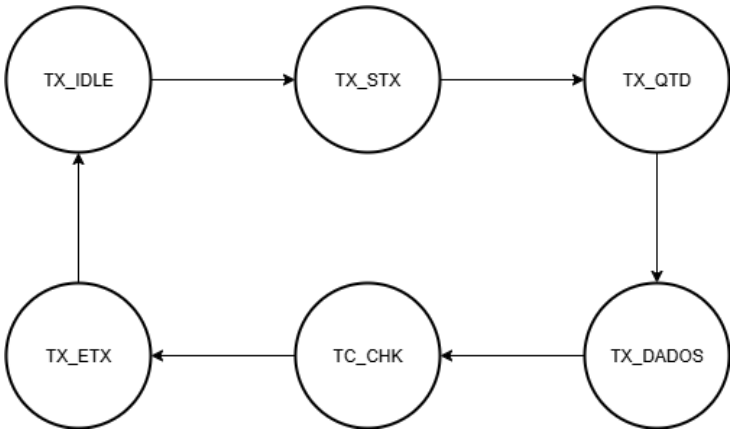


Figura 1: FSM - Transmissor.

4.2 Diagrama do Receptor

A FSM do receptor é mais complexa, pois precisa lidar com o caminho mais simples que o corre quando a mensagem é recebida corretamente, mas também, com todos cenários de erro possíveis.

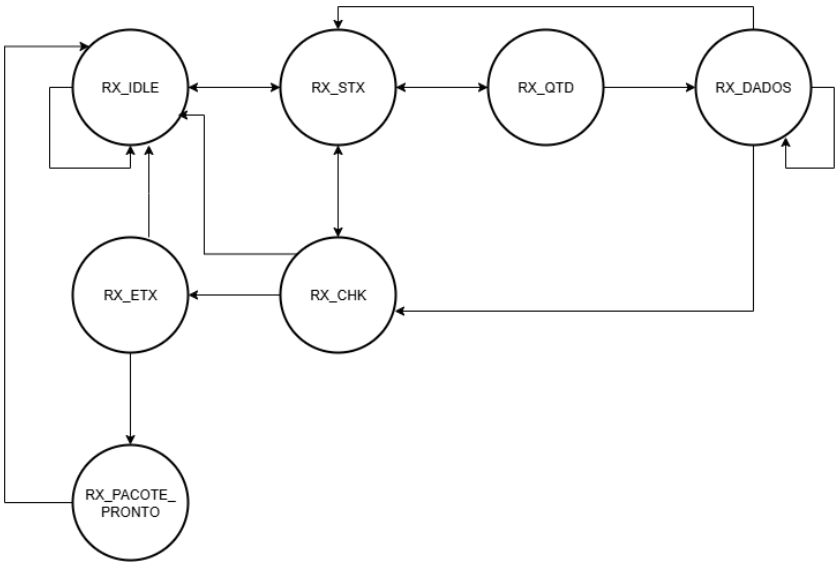


Figura 2: FSM - Receptor.

5 Código Fonte Final

5.1 protocolo.h

```c
#ifndef PROTOCOLO_H
#define PROTOCOLO_H

#include <stdint.h>
#include <stdbool.h>

#define STX_VALUE 0x02
#define ETX_VALUE 0x03
#define MAX_DADOS_SIZE 16

typedef enum {
    TX_IDLE, TX_STX, TX_QTD, TX_DADOS, TX_CHK, TX_ETX
} tx_state_t;

typedef struct {
    tx_state_t estado;
    uint8_t dados[MAX_DADOS_SIZE];
    uint8_t qtd_dados;
    uint8_t index_dados;
    uint8_t checksum;
} protocolo_tx_t;

void protocolo_tx_iniciar(protocolo_tx_t *tx, const uint8_t *
    dados_para_enviar, uint8_t qtd);
bool protocolo_tx_gerar_byte(protocolo_tx_t *tx, uint8_t *byte_gerado);

typedef enum {
    RX_IDLE, RX_STX, RX_QTD, RX_DADOS, RX_CHK, RX_ETX, RX_PACOTE_PRONTO
} rx_state_t;

typedef struct {
    rx_state_t estado;
    uint8_t dados[MAX_DADOS_SIZE];
    uint8_t qtd_dados;
    uint8_t index_dados;
    uint8_t checksum_calculado;
    uint8_t checksum_recebido;
    bool pacote_pronto;
} protocolo_rx_t;

void protocolo_rx_iniciar(protocolo_rx_t *rx);
void protocolo_rx_processar_byte(protocolo_rx_t *rx, uint8_t
    byte_recebido);
bool protocolo_rx_pacote_completo(protocolo_rx_t *rx);

#endif // PROTOCOLO_H
```

Listing 1: Interface pública do módulo de protocolo

5.2 protocolo.c

```c
#include "protocolo.h"
#include <string.h>
#include <stdio.h>

static uint8_t calcular_checksum(const uint8_t *dados, uint8_t qtd) {
    uint8_t chk = 0;
    for (uint8_t i = 0; i < qtd; i++) {
        chk ^= dados[i];
    }
    return chk;
}

void protocolo_tx_iniciar(protocolo_tx_t *tx, const uint8_t *
    dados_para_enviar, uint8_t qtd) {
    if (!tx || (!dados_para_enviar && qtd > 0) || qtd > MAX_DADOS_SIZE)
    {
        tx->estado = TX_IDLE;
        return;
    }
    tx->qtd_dados = qtd;
    if (qtd > 0) {
        memcpy(tx->dados, dados_para_enviar, qtd);
    }
    tx->index_dados = 0;
    tx->checksum = calcular_checksum(tx->dados, tx->qtd_dados);
    tx->estado = TX_STX;
}

bool protocolo_tx_gerar_byte(protocolo_tx_t *tx, uint8_t *byte_gerado) {
    if (tx->estado == TX_IDLE) return false;
    switch (tx->estado) {
        case TX_STX: *byte_gerado = STX_VALUE; tx->estado = TX_QTD;
    break;
        case TX_QTD: *byte_gerado = tx->qtd_dados; tx->estado = (tx->
    qtd_dados > 0) ? TX_DADOS : TX_CHK; break;
        case TX_DADOS:
            *byte_gerado = tx->dados[tx->index_dados++];
            if (tx->index_dados >= tx->qtd_dados) tx->estado = TX_CHK;
            break;
        case TX_CHK: *byte_gerado = tx->checksum; tx->estado = TX_ETX;
    break;
        case TX_ETX: *byte_gerado = ETX_VALUE; tx->estado = TX_IDLE;
    break;
        default: return false;
    }
    return true;
}

void protocolo_rx_iniciar(protocolo_rx_t *rx) {
    if (!rx) return;
    rx->estado = RX_IDLE;
    rx->index_dados = 0;
    rx->qtd_dados = 0;
    rx->checksum_calculado = 0;
    rx->pacote_pronto = false;
}

void protocolo_rx_processar_byte(protocolo_rx_t *rx, uint8_t
```

```
    byte_recebido) {
53      printf(">> Estado: %d | Byte: 0x%02X | Idx: %d | Qtd: %d | Chk: 0x
    %02X\n",
54             rx->estado, byte_recebido, rx->index_dados, rx->qtd_dados, rx
    ->checksum_calculado);
55
56      if (rx->estado == RX_PACOTE_PRONTO) return;
57
58      switch (rx->estado) {
59          case RX_IDLE:
60              if (byte_recebido == STX_VALUE) {
61                  protocolo_rx_iniciar(rx);
62                  rx->estado = RX_STX;
63              }
64              break;
65          case RX_STX:
66              rx->qtd_dados = byte_recebido;
67              if (rx->qtd_dados > MAX_DADOS_SIZE) protocolo_rx_iniciar(rx)
    ;
68              else rx->estado = (rx->qtd_dados > 0) ? RX_QTD : RX_CHK;
69              break;
70          case RX_QTD:
71          case RX_DADOS:
72              if (byte_recebido == STX_VALUE) {
73                  protocolo_rx_iniciar(rx);
74                  rx->estado = RX_STX;
75                  break;
76              }
77              rx->dados[rx->index_dados] = byte_recebido;
78              rx->checksum_calculado ^= byte_recebido;
79              rx->index_dados++;
80              if (rx->index_dados >= rx->qtd_dados) rx->estado = RX_CHK;
81              else rx->estado = RX_DADOS;
82              break;
83          case RX_CHK:
84              if (byte_recebido == STX_VALUE) {
85                  protocolo_rx_iniciar(rx);
86                  rx->estado = RX_STX;
87                  break;
88              }
89              rx->checksum_recebido = byte_recebido;
90              if (rx->checksum_recebido == rx->checksum_calculado) rx->
    estado = RX_ETX;
91              else protocolo_rx_iniciar(rx);
92              break;
93          case RX_ETX:
94              if (byte_recebido == ETX_VALUE) {
95                  rx->pacote_pronto = true;
96                  rx->estado = RX_PACOTE_PRONTO;
97              } else protocolo_rx_iniciar(rx);
98              break;
99          case RX_PACOTE_PRONTO: break;
100     }
101 }
102
103 bool protocolo_rx_pacote_completo(protocolo_rx_t *rx) {
104     return rx->pacote_pronto;
```

```
}
```

Listing 2: Implementação da FSM do protocolo

5.3 teste_protocolo.c

```c
#include <stdio.h>
#include <string.h>
#include "protocolo.h"

int testes_executados = 0;
#define verifica(mensagem, teste) do { if (!(teste)) return mensagem; } while (0)
#define EXECUTA_TESTE(teste) do { char *mensagem = teste(); testes_executados++; \
                                  if (mensagem) return mensagem; } while (0)

// ... (todos os testes foram implementados aqui) ...

static char* teste_rx_deve_processar_pacote_valido(void) {
    protocolo_rx_t rx;
    uint8_t dados_esperados[] = {0x11, 0x22};
    uint8_t checksum = 0x11 ^ 0x22;
    uint8_t pacote_valido[] = {0x02, 0x02, 0x11, 0x22, checksum, 0x03};
    protocolo_rx_iniciar(&rx);
    for (size_t i = 0; i < sizeof(pacote_valido); i++) {
        protocolo_rx_processar_byte(&rx, pacote_valido[i]);
    }
    verifica("RX deveria sinalizar pacote completo e valido", protocolo_rx_pacote_completo(&rx) == true);
    verifica("RX FSM deveria estar no estado PACOTE_PRONTO", rx.estado == RX_PACOTE_PRONTO);
    verifica("RX quantidade de dados recebida deve ser 2", rx.qtd_dados == 2);
    verifica("RX dados recebidos devem ser os esperados", memcmp(rx.dados, dados_esperados, 2) == 0);
    return 0;
}

// ... (demais testes aqui) ...

static char * executa_todos_os_testes(void) {
    EXECUTA_TESTE(teste_tx_deve_iniciar_em_idle);
    EXECUTA_TESTE(teste_tx_deve_gerar_pacote_completo_na_ordem);
    EXECUTA_TESTE(teste_tx_deve_gerar_pacote_sem_dados);
    EXECUTA_TESTE(teste_rx_deve_iniciar_em_idle);
    EXECUTA_TESTE(teste_rx_deve_processar_pacote_valido);
    EXECUTA_TESTE(teste_rx_deve_ignorar_lixo_antes_do_stx);
    EXECUTA_TESTE(teste_rx_deve_rejeitar_pacote_com_checksum_invalido);
    EXECUTA_TESTE(teste_rx_deve_processar_pacote_sem_dados);
    EXECUTA_TESTE(teste_rx_deve_rejeitar_pacote_com_qtd_maior_que_o_buffer);
    EXECUTA_TESTE(teste_rx_deve_resetar_se_novo_stx_chegar_no_meio);
    EXECUTA_TESTE(teste_rx_deve_rejeitar_pacote_sem_etx);
    return 0;
```

```
}

int main() {
    char *resultado = executa_todos_os_testes();
    if (resultado != 0) { printf("ERRO NO TESTE: %s\n", resultado); }
    else { printf("TODOS OS TESTES PASSARAM\n"); }
    printf("Testes executados: %d\n", testes_executados);
    return resultado != 0;
}
```

Listing 3: Código completo da suíte de testes (TDD)

6 Resultados e Conclusão

A execução da suíte de testes, após as devidas correções implementadas, confirmou o sucesso da implementação de ambas as FSMs. A saída do programa de testes foi:

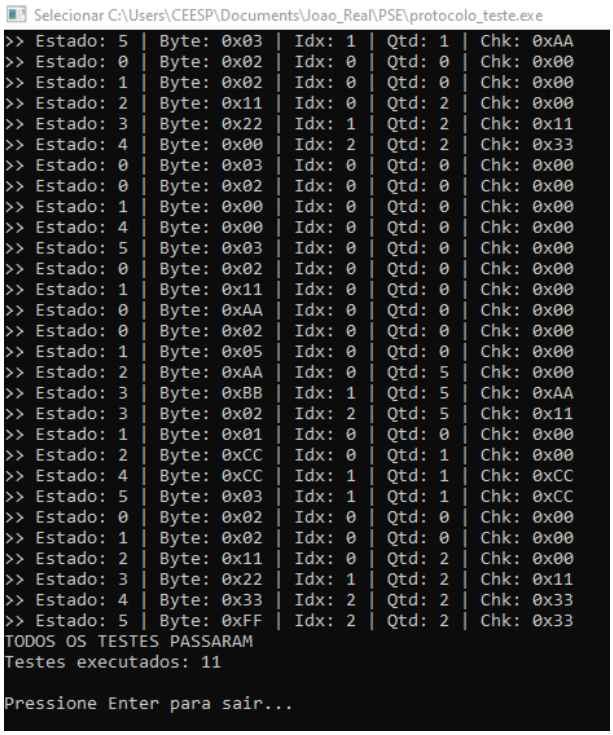


Figura 3: Saída no terminal após a bateria de testes.

A metodologia TDD provou ser extremamente eficaz, não apenas guiando o desenvolvimento para um design de código limpo e funcional, mas também sendo a ferramenta principal para diagnosticar e corrigir quaisquer tipos de erro. A suíte de testes resultante garante a corretude do módulo e serve como uma rede de segurança para futuras modificações.