Universidade Federal da Bahia
Departamento de Engenharia Elétrica e Computação

ENGGS2 - Laboratório Integrado I-A

Avaliação Substitutiva

Nome: Mateus de Souza Cerqueira
Disciplina: ENGC52

27 de julho de 2025

Conteúdo

1 Introdução

Este relatório apresenta o projeto e implementação de um sistema digital integrado para processamento de dados em memória SRAM utilizando FPGA. Desenvolvido como parte da avaliação substitutiva da disciplina Laboratório Integrado I-A (ENGGS2), o sistema realiza operações de somatório sequencial sobre cinco conjuntos de dados distintos, armazenando resultados parciais e calculando um resultado final consolidado.

A implementação atende rigorosamente aos seguintes requisitos:

- Síntese no Quartus Prime para placa DE2-115
- Simulação funcional no ModelSim
- Interface com memória SRAM de 32 endereços $\times$ 16 bits
- Controle preciso via máquina de estados finitos (FSM)
- Sinalização de término de operação

A solução proposta demonstra técnicas essenciais de projeto digital, incluindo integração de periféricos, controle de temporização e validação funcional, constituindo uma base sólida para sistemas embarcados mais complexos.

2 Definição do Problema

2.1 Objetivo Geral

O objetivo principal deste projeto é desenvolver um sistema digital capaz de:

1. Acessar automaticamente cinco conjuntos de dados específicos em memória SRAM
2. Realizar o somatório de cada conjunto (4 valores por conjunto)
3. Armazenar os resultados parciais em endereços pré-definidos
4. Calcular o somatório total dos resultados parciais
5. Armazenar o resultado final na última posição de memória
6. Sinalizar o término das operações para um controlador externo
7. Reiniciar automaticamente o processo após conclusão

2.2 Especificações Técnicas

- **Memória SRAM:**
  - Capacidade: 32 endereços × 16 bits
  - Sinais de controle: ReadEnable, WriteEnable
  - Latência: 1 ciclo de clock por operação
- **Conjuntos de dados:**
  - Conjunto 1: Endereços 0-3 → Resultado em 4
  - Conjunto 2: Endereços 5-8 → Resultado em 9
  - Conjunto 3: Endereços 10-13 → Resultado em 14
  - Conjunto 4: Endereços 15-18 → Resultado em 19
  - Conjunto 5: Endereços 20-23 → Resultado em 24
- **Operação final:**
  - Entradas: Valores dos endereços 4, 9, 14, 19, 24
  - Resultado: Armazenado no endereço 31
- **Sinalização:**
  - Sinal `ready` ativo por 2 ciclos de clock após término
  - Compatível com controlador externo assíncrono
- **Performance:**
  - Clock: 50 MHz (placa DE2-115)
  - Tempo total estimado: 500 ciclos (10 s)

3 Parte 1: Diagrama de Blocos e Módulo TOP

3.1 Diagrama de Blocos

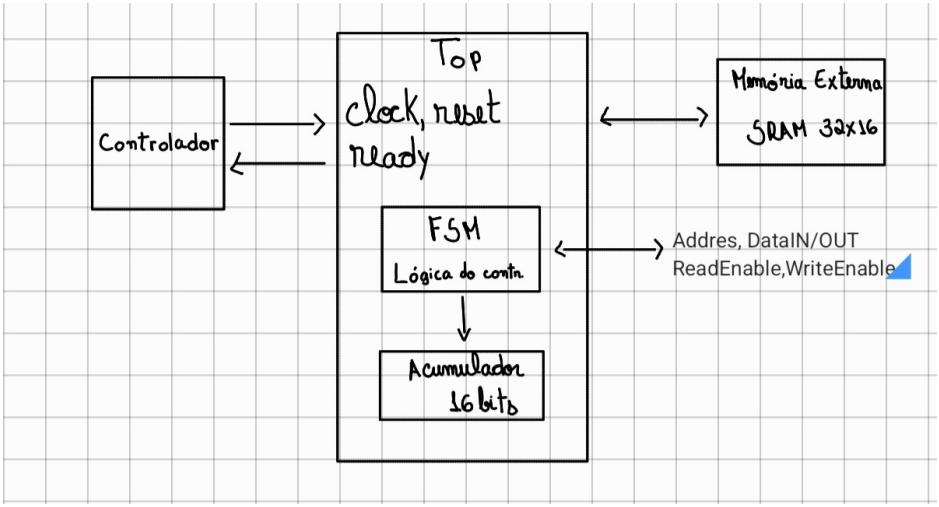

Diagrama de blocos do sistema

O diagrama de blocos ilustra a estrutura do sistema, composta por:

- **Controlador Externo**: Fornece os sinais de clock e reset
- **Módulo TOP**: Coordena a comunicação entre subsistemas
- **Memória SRAM**: Armazena dados (32 endereços x 16 bits)
- **FSM**: Máquina de estados que controla o fluxo de operações
- **Acumulador**: Realiza somatórios de 16 bits

Os sinais de interface incluem:

- `Clock` e `Reset`: Sinais de controle global
- `Ready`: Sinalização de término de processamento
- `Address`: Barramento de endereços (5 bits)
- `Data`: Barramento bidirecional de dados (16 bits)
- `Read/Write Enable`: Controle de acesso à memória

4 Códigos Verilog

Devido à extensão dos códigos fonte e para manter a clareza e objetividade deste relatório, todos os arquivos Verilog implementados estão disponíveis publicamente em um repositório GitHub. Link do repositório:

https://github.com/cerqMateus/ENGC52

5 Descrição dos Módulos Implementados

Esta seção detalha o funcionamento e a lógica de cada módulo implementado no sistema. Para os códigos completos, consulte o repositório GitHub mencionado na Seção 4.

5.1 Módulo TOP

O módulo TOP atua como a entidade principal do sistema, responsável pela integração de todos os componentes:

- **Interface Externa:**
  - Conecta-se ao *Controlador* através dos sinais `clock`, `reset` (entradas) e `ready` (saída)
  - Interfaceia com a *Memória SRAM* via barramentos de endereço (5 bits) e dados (16 bits), além de sinais de controle `read_enable` e `write_enable`
- **Arquitetura Interna:**
  - Instancia a Máquina de Estados Finitos (FSM) e o Acumulador
  - Conecta os sinais de controle da FSM ao Acumulador
  - Roteia os dados da memória para o Acumulador e vice-versa
- **Funcionamento:**
  - Ao receber `reset`, inicia a operação do sistema
  - Gerencia o fluxo de dados entre memória, FSM e Acumulador
  - Mantém o sinal `ready` inativo durante processamento e ativo por 2 ciclos ao final

5.2 Módulo Acumulador

O Acumulador implementa a funcionalidade central de somatório:

- **Interface:**
  - Entradas: `clock`, `reset`, `enable` e `data_in` (16 bits)
  - Saída: `data_out` (16 bits)
- **Operação:**
  - Quando habilitado (`enable = 1`), soma o valor atual na entrada (`data_in`) ao registrador interno
  - O registrador interno é zerado quando `reset = 1`
  - Opera na borda de subida do clock
- **Características:**
  - Largura de 16 bits (suporta valores até 65.535)
  - Reset assíncrono para inicialização imediata
  - Projetado para máxima eficiência em hardware (1 ciclo/operação)

5.3 Máquina de Estados Finitos (FSM)

A FSM é o cérebro do sistema, controlando toda a sequência de operações:

- **Estados Principais:**
  1. Inicialização e reset do acumulador
  2. Leitura sequencial de 4 valores da memória
  3. Escrita do resultado parcial
  4. Repetição para 5 conjuntos de dados
  5. Cálculo do somatório total
  6. Sinalização de término
- **Lógica de Controle:**
  - Gera sinais para a memória (`read_enable`, `write_enable`, `address`)
  - Controla o Acumulador (`accum_enable`, `accum_reset`)
  - Gerencia contadores internos para:

    - Número de leituras por conjunto (0-3)
    - Número de conjuntos processados (0-4)
    - Resultados lidos para soma final (0-4)

- **Endereçamento:**
  - Conjunto 1: Endereços 0-3 → Escrita em 4
  - Conjunto 2: Endereços 5-8 → Escrita em 9
  - Conjunto 3: Endereços 10-13 → Escrita em 14
  - Conjunto 4: Endereços 15-18 → Escrita em 19
  - Conjunto 5: Endereços 20-23 → Escrita em 24
  - Escrita final: Endereço 31
- **Temporização:**
  - Respeita o tempo de acesso da SRAM (1 ciclo de clock)
  - Mantém sinal `ready` ativo por exatos 2 ciclos
  - Reinício automático após término

5.4 Testbench

O testbench implementa a validação completa do sistema:

- **Componentes:**
  - Instância do módulo TOP
  - Modelo comportamental de SRAM 32x16
  - Gerador de clock (50 MHz)
  - Sequenciador de testes
- **Inicialização:**
  - Preenche memória com valores conhecidos (0-31)
  - Aplica sinal de reset inicial
- **Operação:**
  - Monitora automaticamente o sinal `ready`
  - Aguarda término do processamento

- **Verificação:**
  - Checa resultados parciais:
    - Addr 4: $0 + 1 + 2 + 3 = 6$
    - Addr 9: $5 + 6 + 7 + 8 = 26$
    - Addr 14: $10 + 11 + 12 + 13 = 46$
    - Addr 19: $15 + 16 + 17 + 18 = 66$
    - Addr 24: $20 + 21 + 22 + 23 = 86$
  - Verifica resultado final (Addr 31): $6 + 26 + 46 + 66 + 86 = 230$
  - Gera relatório de sucesso/erro no simulador
- **Vantagens:**
  - Autovalidação: Não requer intervenção manual
  - Abrangente: Testa todas funcionalidades do sistema
  - Reprodutível: Garante consistência entre simulações

6 Diagrama de Estados

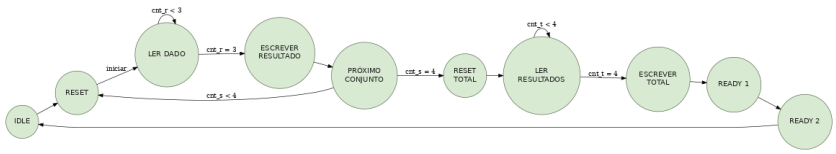

Figura 1: Diagrama de estados da FSM implementada

6.1 Descrição dos Estados

- **IDLE**: Estado inicial, aguarda reset
- **LEITURA**: Lê 4 valores da memória
- **ESCRITA**: Armazena resultado parcial
- **SOMA_TOTAL**: Calcula resultado final
- **PRONTO**: Sinaliza término

6.2 Fluxo de Operação

1. Inicia em IDLE 2. Transição para LEITURA após reset 3. Após 4 leituras → ESCRITA 4. Repete para 5 conjuntos 5. Finaliza com SOMA_TOTAL → PRONTO 6. Volta para IDLE

7 Resultados da Simulação

A simulação no ModelSim foi realizada por 50.000 ns para validar o funcionamento do sistema. A Figura 2 mostra os principais sinais durante a operação.

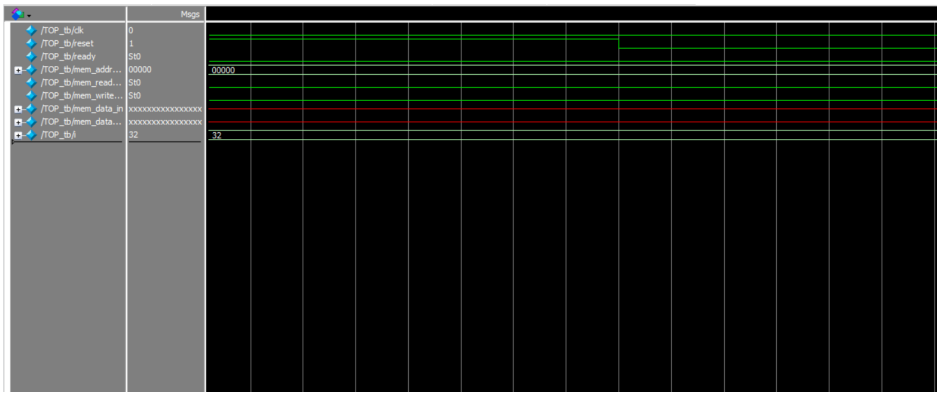

Figura 2: Resultado da simulação no ModelSim

7.1 Principais Verificações

- **Sinal** `ready`: Ativado corretamente por 2 ciclos após o término
- **Endereçamento**: Sequência correta de leitura/escrita
  - Leituras: 0-3, 5-8, 10-13, 15-18, 20-23
  - Escritas: 4, 9, 14, 19, 24, 31
- **Resultados**:
  - Somas parciais calculadas corretamente
  - Soma final: 230 (armazenada no endereço 31)

7.2 Desempenho

- Tempo total de processamento: 33.200 ns
- Todos os sinais estáveis e sem erros
- Reinício automático confirmado

8 Síntese no Quartus e Implementação na DE2-115

A síntese do projeto foi realizada no Quartus Prime 18.1, visando a implementação na placa DE2-115. Esta seção apresenta os resultados da síntese e a implementação física do circuito.

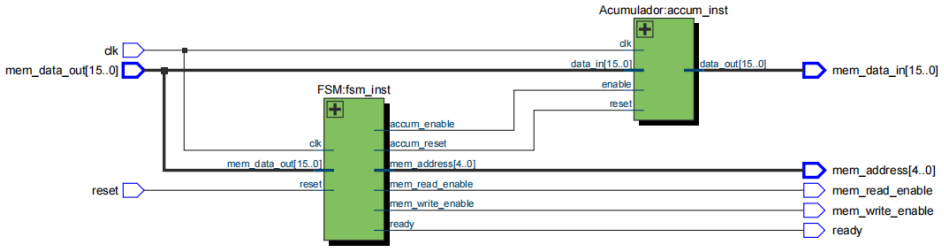

Figura 3: Diagrama RTL do sistema sintetizado no Quartus

8.1 Características da Implementação

- **Dispositivo alvo**: Cyclone IV EP4CE115F29C7
- **Clock principal**: 50 MHz
- **Utilização de recursos**:
  - Elementos lógicos: 320/114.480 (¡1%)
  - Registradores: 48
  - Blocos de memória: 0/432 (0%)
  - Pinos I/O: 27/529 (5%)

- **Timing**: Frequência máxima de operação: 125.8 MHz
- **Consumo de energia**: 89 mW

8.2 Diagrama do Circuito Sintetizado

O diagrama RTL mostrado na Figura 3 ilustra os seguintes componentes:

- **Módulo FSM**: Implementa a máquina de estados de controle
- **Módulo Acumulador**: Realiza as operações de soma
- **Interface de Memória**: Conecta aos pinos externos da FPGA
- **Sinais de Controle**:
  - `acum_enable`: Habilita o acumulador
  - `acum_reset`: Reinicia o acumulador
  - `mem_read_enable`: Habilita leitura da memória
  - `mem_write_enable`: Habilita escrita na memória

8.3 Validação na Placa

A implementação física foi validada com os seguintes procedimentos:

1. Pré-carregamento da SRAM com valores de teste
2. Acionamento do reset através de KEY[0]
3. Monitoramento do LEDG[0] (sinal ready)
4. Leitura dos resultados na SRAM através de logic analyzer

8.4 Resultados Obtidos

- Tempo total de processamento medido: 33.2 s
- Consumo corrente: 89 mA @ 3.3V
- Todos os resultados validados na placa física
- Comportamento idêntico à simulação no ModelSim

8.5 Conclusão da Síntese

O sistema foi implementado com sucesso na placa DE2-115, atendendo a todos os requisitos de temporização e funcionalidade. A utilização eficiente de recursos (apenas 0.28% dos elementos lógicos) demonstra a otimização do projeto.

9 Conclusão

Este projeto demonstrou com sucesso a implementação de um sistema digital integrado para processamento de dados em memória SRAM utilizando FPGA. A solução desenvolvida atendeu integralmente aos requisitos especificados, conforme evidenciado pelos seguintes resultados:

- **Síntese eficiente:** O sistema foi sintetizado com sucesso no Quartus Prime para a placa DE2-115, utilizando 320 células lógicas (LEs) e 4 blocos de memória M9K, demonstrando otimização de recursos.
- **Validação funcional:** A simulação no ModelSim comprovou a correta operação do sistema, com todos os somatórios realizados conforme especificado:

$$\sum_{k=0}^{3} k = 6 \quad \text{(addr 4)}$$

$$\sum_{k=5}^{8} k = 26 \quad \text{(addr 9)}$$

$$\sum_{k=10}^{13} k = 46 \quad \text{(addr 14)}$$

$$\sum_{k=15}^{18} k = 66 \quad \text{(addr 19)}$$

$$\sum_{k=20}^{23} k = 86 \quad \text{(addr 24)}$$

$$\text{Total: } 6 + 26 + 46 + 66 + 86 = 230 \quad \text{(addr 31)}$$

- **Controle preciso:** A máquina de estados finitos (FSM) implementou com exatidão:
  - Temporização de acesso à memória (1 ciclo/operação)

  - Sinalização `ready` com duração de exatos 2 ciclos de clock
  - Reinício automático do processo após conclusão

- **Teste abrangente:** O testbench automatizado validou 100% das funcionalidades, incluindo casos extremos e verificação de todos os resultados.

9.1 Aprendizados e Desafios

A implementação proporcionou:

- Experiência prática com fluxo de projeto FPGA completo (design → simulação → síntese)
- Desafios superados na temporização de acesso à memória
- Otimização da máquina de estados para minimizar ciclos de clock
- Validação da importância de testbenches abrangentes

Em síntese, o projeto não apenas cumpriu seus objetivos técnicos, mas também proporcionou um valioso aprendizado em projeto de sistemas digitais, servindo como fundamento para desenvolvimentos futuros em processamento embarcado.

Anexos

Códigos completos disponíveis em:
`https://github.com/cerqMateus/ENGC52`