UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

PROJETO DE ESTRUTURAS DE ARMAZENAMENTO DIGITAL EM UM SoC PARA CONTROLE DE IRRIGAÇÃO

GILMAR SILVA BESERRA

ORIENTADOR: ADSON FERREIRA DA ROCHA

CO-ORIENTADOR: JOSÉ CAMARGO DA COSTA

DISSERTAÇÃO DE MESTRADO

PUBLICAÇÃO: 196/2004

BRASÍLIA/DF: Agosto/2004

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

PROJETO DE ESTRUTURAS DE ARMAZENAMENTO DIGITAL EM UM SOC PARA CONTROLE DE IRRIGAÇÃO

Gilmar Silva Beserra

DISSERTAÇÃO DE MESTRADO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA COMO PARTE DOS REQUISITOS NECEESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE

_____
Adson Ferreira da Rocha, Doutor (UnB)
(Orientador)

_____
José Camargo da Costa, Doutor (UnB)
(Co-orientador, Examinador Interno)

_____
Ricardo P. Jacobi, Doutor (UnB)
(Examinador Interno)

_____
João Navarro Soares Júnior, Doutor (USP)
(Examinador Externo)

FICHA CATALOGRÁFICA

BESERRA, GILMAR SILVA
Projeto de Estruturas de Armazenamento Digital em um SoC para Controle de Irrigação [Distrito Federal] 2004.
Dissertação de Mestrado – Universidade de Brasília. Faculdade de Tecnologia. Departamento de Engenharia Elétrica.
1. Projeto VLSI 2. Processadores RISC
3. Modelamento de circuito 4. Testabilidade de sistemas
I. ENE/FT/UnB II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

BESERRA, G. S. (2004). Projeto de Estruturas de Armazenamento Digital em um SoC para Controle de Irrigação. Dissertação de Mestrado, Publicação 196/2004
Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 170 p.

CESSÃO DE DIREITOS

NOME DO AUTOR: Gilmar Silva Beserra
TÍTULO DA DISSERTAÇÃO DE MESTRADO:Projeto de Estruturas de Armazenamento Digital em um SoC para Controle de Irrigação.
GRAU / ANO: Mestre / 2004

_____

Gilmar Silva Beserra

A Deus, que conforta os nossos corações e ilumina os nossos caminhos.
Aos meus pais, familiares, amigos, professores e colegas, cujo apoio foi indispensável para a conclusão de mais uma etapa importante da minha vida.

"*A pior coisa que pode acontecer na vida de uma pessoa não é quando seu projeto não dá certo, seu plano de ação não funciona, ou quando a viagem termina no lugar errado. O pior é não começar. Esse é o maior naufrágio*."
Amyr Klink

AGRADECIMENTOS

Agradeço a todos que contribuíram para a realização deste trabalho:

A Deus, pela vida, pela força, pelos dons de amar, aprender e criar.

Ao meu pai Gilvan Alves Beserra, pelo exemplo de conduta profissional e por sempre ter me incentivado a aperfeiçoar os meus conhecimentos e a buscar novas conquistas, e à minha mãe Maria Conceição Silva Beserra, pelo exemplo de vida e por ter me ensinado a ter sempre bom ânimo, saber perdoar e ter solicitude constante, paciência, generosidade, modéstia e esperança. Aos meus irmãos Liandra, Éder e Iuri, e aos demais familiares, pelo apoio incondicional.

À minha tia Ires, cuja ajuda e hospitalidade me tornaram possível freqüentar a universidade.

Aos professores Adson Ferreira da Rocha, José Camargo da Costa e Leonardo R. A. X. de Menezes, pela confiança, compreensão, amizade, pelos exemplos de dedicação e competência e por terem aceitado me orientar neste trabalho. Aos professores João Navarro Soares Júnior e Ricardo Jacobi, pelas sugestões oferecidas para a melhoria do projeto.

À colega Janaína Domingues Costa, pela ajuda e incentivo, e principalmente pelo exemplo de competência e perseverança. Ao colega Genival Araújo, pela amizade, pela ajuda e pela dedicação na realização do projeto. Aos colegas Leandro S. Assunção, Rafael Soares, Pablo Vogel, João Carlos Marra, Wilson H. Veneziano, Carlos A. Tenório Carvalho Jr., Wellington A. Amaral, Maxwell D. B. de Mello, Renato Cardoso e Ryan Rangel, pelo apoio, colaboração e amizade.

Aos funcionários da pós-graduação do ENE e aos técnicos do Grupo de Apoio Técnico do ENE, pela amizade e colaboração.

Às amigas Kharyn Leigh, Maggie Elliot, Vala Negahdauri e Arezoo Mansori, pelo carinho, confiança e pela força que me deram nos momentos mais difíceis.

À amiga Neda Sadeghiani, pelo grande coração, pela amizade, incentivo, confiança, carinho e pela ajuda prestada para a apresentação desse trabalho.

Ao CNPq, pelo apoio financeiro.

RESUMO

Neste trabalho foram projetados e implementados uma memória ROM de 256 bits, uma memória RAM de 128 bits e um banco com 16 registradores de 16 bits, em tecnologia CMOS 0.35 μm, como veículos de validação preliminar de uma arquitetura contendo 2kB de ROM e 8 kB de RAM. Tais estruturas integram um Sistema em *Chip* (SoC) para comunicação sem fio em um sistema de controle de irrigação. Foram desenvolvidos os projetos arquitetural, elétrico e físico das unidades anteriormente citadas utilizando técnicas de projeto orientado à testabilidade. Esses módulos foram projetados e simulados utilizando ferramentas do CADENCE e atenderam às especificações previamente definidas. Após validadas, as estruturas foram enviadas para fabricação.

ABSTRACT

A 256-bit ROM, a 128-bit RAM, and a bank of sixteen 16-bit registers were implemented in a 0.35 μm CMOS technology. The ROM and RAM memory capacity will be expanded to 2kBytes and 8 kBytes in order to integrate a System on Chip (SoC) for irrigation control on crops. An architecture that integrates and expands the memory according to a 16-bit RISC microprocessor datapath was also proposed. Design for Testability (DFT) techniques were also used. After simulation and validation with the CADENCE framework, the circuits were sent to fabrication.

SUMÁRIO

1 INTRODUÇÃO 1

2 REVISÃO BIBLIOGRÁFICA 4

2.1 ROM 4
2.1.1 ROM MOS 4
2.1.2 ROM Programáveis por Máscara 5
2.1.3 ROM Programáveis (PROM e EPROM) 6
2.1.4 Alternativas para Implementação de uma ROM 6
2.2 RAM 9
2.2.1 SRAM x DRAM 12
2.2.1.1 A célula de memória dinâmica 13
2.2.1.2 A célula de memória estática 13
2.2.2 Circuitos periféricos da memória RAM estática 17
2.2.2.1 Amplificador sensor e pré-carga 17
2.2.2.2 Decodificador de linha 26
2.2.2.3 Decodificador de coluna 27
2.2.2.4 Circuito de escrita 29
2.3 PROJETO VOLTADO PARA A TESTABILIDADE 30
2.4 TESTE EM MEMÓRIAS 31
2.4.1 Modelos de falhas em SRAM 32
2.4.1.1 Falhas no decodificador de endereço 32
2.4.1.2 Falhas na matriz de células 33
2.4.1.3 Falhas na lógica de escrita/leitura 34
2.4.2 Tipos de teste 34
2.4.2.1 March test 34
2.4.2.2 BIST (*Built-in Self-Testing*) 37

3 METODOLOGIA DE PROJETO 40

3.1 INTRODUÇÃO 40
3.2 PROCESSO DE PROJETO 41
3.3 METODOLOGIA ADOTADA 42

4 DESCRIÇÃO DO PROCESSADOR RISC-16 44

4.1 INTRODUÇÃO 44
4.2 ESPECIFICAÇÃO 44
4.3 INSTRUÇÕES 44
4.4 ULA 46
4.5 MEMÓRIAS 47
4.6 REGISTRADORES 48
4.6.1 Registradores mapeados em memória 48
4.7 BARRAMENTO DE DADOS E CONTROLE 50
4.9 - ESTRUTURA DE I/O 53

5 PROJETO DA ROM (*READ ONLY MEMORY*) 55

5.1 ESPECIFICAÇÃO DA ROM 55
5.2 PROJETO ELÉTRICO DA ROM 55
5.3 ESTRUTURAS DE TESTE DA ROM 60
5.4 SIMULAÇÃO 62

6 PROJETO DA SRAM (*STATIC RANDOM-ACCESS MEMORY*) 63

6.1 INTRODUÇÃO 63
6.2 ESPECIFICAÇÃO 63
6.3 ARQUITETURA DA SRAM 64
6.4 CÓDIGO VHDL 65
6.5 BLOCOS DA SRAM 66
6.5.1 Célula 66
6.5.2 Amplificador sensor 73
6.5.3 Circuito de escrita 77

6.5.4 Pré-carga e circuito auxiliar de pré-carga ... 78
6.5.5 Decodificador de linha ... 81
6.6 SIMULAÇÕES ... 82
6.7 TESTE DA SRAM ... 85

7 PROJETO DO BANCO DE REGISTRADORES ... 87

7.1 INTRODUÇÃO ... 87
7.2 ESTRUTURA DE UMA CÉLULA DE UM REGISTRADOR ... 87
7.3 REGISTRADOR DE 16 BITS ... 89
7.4 BANCO DE REGISTRADORES ... 91
7.4.1 Descrição funcional ... 91
7.4.2 Simulação do código VHDL ... 92
7.4.3 Projeto elétrico ... 93
7.4.4 Estruturas de teste ... 95

8 INTEGRAÇÃO DAS MEMÓRIAS ... 97

8.1 INTRODUÇÃO ... 97
8.2 ESPECIFICAÇÃO ... 97
8.3 PROJETO ELÉTRICO ... 98
8.3.1 ROM 2 kB ... 98
8.3.2 RAM 8 kB e registradores ... 100
8.3.3 Decodificadores de linha e coluna ... 102
8.3.3.1 Decodificador de coluna ... 102
8.3.3.2 Decodificador de linha ... 103
8.3.4 Erro de endereçamento na memória ... 104
8.4 VALIDAÇÃO DA ESTRUTURA ... 105

9 DISCUSSÃO DOS RESULTADOS ... 107

9.1 O *CHIP* DE TESTE ... 107
9.2 SIMULAÇÕES FINAIS ... 110
9.2.1 ROM ... 110
9.2.2 SRAM ... 113
9.2.3 Banco de Registradores ... 115

10 CONCLUSÃO ... 119

11 REFERÊNCIAS BIBLIOGRÁFICAS ... 120

APÊNDICE A – VETORES DE TESTE ... 122

A1 – MEMÓRIA ROM ... 122
A2 – MEMÓRIA RAM ... 123
A3 – BANCO DE REGISTRADORES ... 126

APÊNDICE B – CÓDIGOS VHDL ... 129

B1 - MEMÓRIA ... 129
B2 – BANCO DE REGISTRADORES ... 133

APÊNDICE C – LAYOUTS ... 137

C.1 – MATRIZ DA ROM ... 137
C.2 – MATRIZ DA ROM COM DECODIFICADOR DE LINHA ... 137
C.3 – ROM FINAL COM ESTRUTURAS DE TESTE ... 138
C.4 – CÉLULA 6-T DA MEMÓRIA SRAM ... 138
C.5 – PRÉ-CARGA CONTROLADA DA SRAM ... 138
C.6 – CIRCUITO DE ESCRITA DA SRAM ... 139
C.7 – AMPLIFICADOR SENSOR DE CORRENTE COM ESTÁGIO DE SAÍDA ... 139
C.8 – MATRIZ SRAM 8X16 BITS ... 139
C.9 – MATRIZ SRAM COM DECODIFICADOR E CONVERSORES ... 140
C.10 – CÉLULA DE UM REGISTRADOR ... 140
C.11 – REGISTRADOR DE 16 BITS ... 140
C.12 – CONVERSOR SERIAL-PARALELO DE 4 BITS ... 141
C.13 – CONVERSOR SERIAL-PARALELO DE 16 BITS ... 141

C.14 – BANCO DE REGISTRADORES 142
C.15 – BANCO DE REGISTRADORES COM CONVERSORES E TESTE..... 142
APÊNDICE D – ROTINA DE INICIALIZAÇÃO DA ROM..... 143
APÊNDICE E – MODELO DE TRANSISTOR UTILIZADO 150

LISTA DE TABELAS

TABELA 4.1 – INSTRUÇÕES [COSTA, 2004A] 45
TABELA 4.2 – REGISTRADORES DO BANCO [COSTA, 2004A] 48
TABELA 4.3 - CAMPOS DE FFF3 [COSTA, 2004A] 49
TABELA 4.4 - CAMPOS DE FFF1 [COSTA, 2004A] 49
TABELA 4.5 – ENDEREÇOS DE I/O [COSTA, 2004A] 53
TABELA 5.1 - PROGRAMA ARMAZENADO NA ROM 56
TABELA 6.1 - SINAIS DA SRAM 65
TABELA 6.2 – PARÂMETROS PARA A TECNOLOGIA 0,35 UM [AMS, 2001] 70
TABELA 6.3 - PARÂMETROS PARA A TECNOLOGIA 0,35 UM [AMS, 2001] 72
TABELA 8.1 - PROPOSTA DE ENDEREÇOS DOS REGISTRADORES MAPEADOS EM MEMÓRIA 101
TABELA 9.1 - PINOS DE ENTRADA E SAÍDA DO MÓDULO 109

LISTA DE FIGURAS

FIG. 1.1 - SISTEMA DE IRRIGAÇÃO COM NÓS, ESTAÇÕES DE CAMPO E ESTAÇÃO BASE [COSTA, 2003 – ALTERADA] 2
FIG. 1.2 - DETALHES DE UM NÓ DO SISTEMA DE IRRIGAÇÃO 2
FIG. 2.1 - TRECHO DE UMA ROM MOS 5
FIG. 2.2 - ARRANJO NOR [WESTE, 1993] 7
FIG. 2.3 - ROM DINÂMICA [WESTE, 1993] 8
FIG. 2.4 - CIRCUITO ROM COM ECONOMIA DE POTÊNCIA [WESTE, 1993] 8
FIG. 2.5 - ARQUITETURA GENÉRICA DE UMA MEMÓRIA RAM [SEDRA, 2000] 9
FIG. 2.6 - ARQUITETURA DE UM *CHIP* DE MEMÓRIA [WESTE, 1993 - ALTERADA] 11
FIG. 2.7 - CÉLULA DE RAM COM CIRCUITOS PERIFÉRICOS [WESTE, 1993 – ALTERADA] 11
FIG. 2.8 - ARQUITETURA DE UMA RAM DE 64 KB [HODGES, 2003 – ALTERADA] 12
FIG. 2.9 - CÉLULA DE UM TRANSISTOR [SEDRA, 2000] 13
FIG. 2.10 - CÉLULA 6-T [WESTE, 1993], [SEDRA, 2000], [CHANDRAKASAN, 2001], [HODGES, 2003] 14
FIG. 2.11 - CÉLULA COM RESISTORES [WESTE, 1993] 15
FIG. 2.12 - CÉLULA SBLSRWA [WANG, 2000] 16
FIG. 2.13 - CÉLULA SRAM COM 4 TRANSISTORES [JOUBERT, 1999] 17
FIG. 2.14 - AMPLIFICADOR SENSOR [SEDRA, 2000] 18
FIG. 2.15 - OPERAÇÃO DO AMPLIFICADOR SENSOR [SEDRA, 2000] 19
FIG. 2.16 - PRÉ-CARGA COM VDD [WESTE, 1993] 20
FIG. 2.17 – FORMAS DE ONDA PARA PRÉ-CARGA COM VDD [WESTE, 1993] 21
FIG. 2.18 - CIRCUITO BÁSICO DE LEITURA [HODGES, 2003] 22
FIG. 2.19 - AMPLIFICADOR SENSOR DIFERENCIAL [HODGES, 2003] 23
FIG. 2.20 - DESCARGA NA SAÍDA [HODGES, 2003] 24
FIG. 2.21 - CARGA NA SAÍDA [HODGES, 2003] 24
FIG. 2.22 - AMPLIFICADOR SENSOR DE CORRENTE [MAHESHWARI, 2000] 25
FIG. 2.23 - ESTÁGIO DE SAÍDA DO AMPLIFICADOR SENSOR [SEEVINCK, 1990] 26
FIG. 2.24 - DECODIFICADOR NOR [SEDRA, 2000] 27
FIG. 2.25 - DECODIFICADOR DE COLUNA [SEDRA, 2000] 28
FIG. 2.26 - DECODIFICADOR EM ÁRVORE [SEDRA, 2000] 28
FIG. 2.27 - CIRCUITO DE ESCRITA COM TRANSISTORES N [WESTE, 1993] 29
FIG. 2.28 - OPERAÇÃO DO CIRCUITO DURANTE A ESCRITA [WESTE, 1993] 30
FIG. 2.29 - CIRCUITO DE ESCRITA COM PORTAS DE TRANSMISSÃO [WESTE, 1993] 30
FIG. 2.30 - MODELO FUNCIONAL REDUZIDO DE UM CHIP SRAM [GOOR, 1993] 32
FIG. 2.31 - DIAGRAMA DE BLOCOS DA LÓGICA BIST GENÉRICA PARA TESTE DE RAMS [FRANKLIN, 1990]. 38
FIG. 3.1 - DIAGRAMA DE DOMÍNIOS E NÍVEIS DE ABSTRAÇÃO [WESTE, 1993] 40
FIG. 3.2 – METODOLOGIA DE PROJETO 41
FIG. 4.1 – TIPOS DE INSTRUÇÕES [COSTA, 2004A] 44
FIG. 4.2 – ULA [BENÍCIO, 2002 - ALTERADA] 46
FIG. 4.3 – MEMÓRIAS 47
FIG. 4.4 – BANCO DE REGISTRADORES [COSTA, 2004A - ALTERADA] 48
FIG. 4.5 - FORMATOS DOS REGISTRADORES REGINT E INT CAUSA [COSTA, 2004A] 49
FIG. 4.6 - FORMATO DO REGISTRADOR REGSTATUS [COSTA, 2004A] 49
FIG. 4.7 – CAMINHO DE DADOS, LINHAS DE CONTROLE E INTERRUPÇÃO [COSTA, 2004A] 51
FIG. 4.8 – MÁQUINA DE ESTADOS FINITOS DA UNIDADE DE CONTROLE [COSTA, 2004A] 52
FIG. 4.9 – REGISTRADORES DE I/O [COSTA, 2004A] 54
FIG. 5.1 – MATRIZ DA ROM COM 16 PALAVRAS DE 16 BITS 57
FIG. 5.2 – TRECHO DO DECODIFICADOR DE LINHA 58
FIG. 5.3 - SIMULAÇÃO DO DECODIFICADOR DE 4 BITS 59
FIG. 5.4 - ROM 59
FIG. 5.5 – SIMULAÇÃO DO CONVERSOR PARALELO-SERIAL DE 4 BITS [COSTA, 2004A] 60
FIG. 5.6 – ESTRUTURAS DE TESTE DA ROM 61
FIG. 5.7 – LEITURA DA PALAVRA ARMAZENADA NA POSIÇÃO 0000 62
FIG. 6.1 – DIAGRAMA DA SRAM 63
FIG. 6.2 – ARQUITETURA DA SRAM 64
FIG. 6.3 - FORMAS DE ONDA DA SIMULAÇÃO DO CÓDIGO VHDL DA MEMÓRIA 66
FIG. 6.4 – CÉLULA 6-T 67
FIG. 6.5 - CÉLULA 6-T E VTC DOS INVERSORES 1 E 2 [HODGES, 2003] 67
FIG. 6.6 - OPERAÇÃO DE LEITURA NA CÉLULA [HODGES, 2003 – ALTERADA] 69

FIG. 6.7 - OPERAÇÃO DE ESCRITA NA CÉLULA [HODGES, 2003 – ALTERADA] 71
FIG. 6.8 – VALORES DE W 72
FIG. 6.9 – AMPLIFICADOR SENSOR DE CORRENTE COM SEÇÃO DE SAÍDA COM VALORES DE W (L = 0,4 UM) 73
FIG. 6.10 - ESTRUTURA PARA SIMULAÇÃO DO AMPLIFICADOR SENSOR DE CORRENTE 74
FIG. 6.11 – ESTRUTURA PARA SIMULAÇÃO DA LEITURA DE 1 BIT 75
FIG. 6.12 – SIMULAÇÃO DA LEITURA DO BIT 1 76
FIG. 6.13 – SIMULAÇÃO DA LEITURA DO BIT 0 77
FIG. 6.14 – CIRCUITO DE ESCRITA COM OS VALORES DE W (L = 0,4 UM) 78
FIG. 6.15 – PRÉ-CARGA COM VALORES DE W (L = 0,4 UM) 79
FIG. 6.16 - CIRCUITO AUXILIAR DE CONTROLE 80
FIG. 6.17 - FUNÇÃO DO DECODIFICADOR DE LINHA [HODGES, 2003 – ALTERADA] 81
FIG. 6.18 - SIMULAÇÃO DO DECODIFICADOR DE LINHA 82
FIG. 6.19 - CÉLULA COM CIRCUITO DE ESCRITA E AMPLIFICADOR SENSOR 83
FIG. 6.20 – SIMULAÇÃO DA CÉLULA DA FIGURA 6.19 84
FIG. 6.21 – SIMULAÇÃO DA SRAM SEM ESTRUTURAS DE TESTE 85
FIG. 6.22 - ESQUEMÁTICO DA SRAM COM ESTRUTURAS DE TESTE 86
FIG. 6.23 - CÉLULA RAM COM *PADS* INTERNOS 86
FIG. 7.1– CÉLULA DOS REGISTRADORES [BENÍCIO, 2002] 87
FIG. 7.2 - VALORES DE W PARA OS TRANSISTORES DA CÉLULA 88
FIG. 7.3 - SIMULAÇÃO DE UMA OPERAÇÃO DE ESCRITA E LEITURA EM UM REGISTRADOR 89
FIG. 7.4 - REGISTRADOR DE 16 BITS 89
FIG. 7.5 - SIMULAÇÃO DO REGISTRADOR DE 16 BITS 90
FIG. 7.6 - DIAGRAMA DO BANCO DE REGISTRADORES 91
FIG. 7.7 - SIMULAÇÃO DO BANCO DE REGISTRADORES 93
FIG. 7.8 - ESQUEMÁTICO DO BANCO DE REGISTRADORES 94
FIG. 7.9 - SIMULAÇÃO DO BANCO 95
FIG. 7.10 – ESTRUTURA DE TESTE DO BANCO DE REGISTRADORES 96
FIG. 7.11 - REGISTRADOR DE 16 BITS COM *PADS* INTERNOS 96
FIG. 8.1- ENDEREÇAMENTO DAS MEMÓRIAS 98
FIG. 8.2 - ESQUEMÁTICO DA ROM 2 KB 99
FIG. 8.3 – *BUFFER TRI-STATE* 99
FIG. 8.4 - ESQUEMÁTICO DA RAM 8 KB 100
FIG. 8.5 – REGISTRADORES NA RAM 101
FIG. 8.6 – ALTERNATIVA PARA IMPLEMENTAÇÃO DA NAND6 [HODGES, 2003] 102
FIG. 8.7 – DECODIFICADOR DE COLUNA (6 BITS) [HODGES, 2003] 103
FIG. 8.8 - ALTERNATIVA PARA IMPLEMENTAÇÃO DA NAND7 [HODGES, 2003 – ALTERADA] 103
FIG. 8.9 - CIRCUITO PARA GERAÇÃO DO SINAL DE ERRO DE ENDEREÇAMENTO 104
FIG. 8.10 – ESTRUTURA PARA TESTE PRELIMINAR DOS BLOCOS DA MEMÓRIA 105
FIG. 9.1 - *LAYOUT* DO *CHIP* DE TESTE (ÁREA TOTAL DE 25MM^2) 107
FIG. 9.2 - ESQUEMÁTICO DO MÓDULO BANCO_ROM_RAM 108
FIG. 9.3 - SIMULAÇÃO FINAL DA ROM PARA *TESTE_CONV* = 0 110
FIG. 9.4 - SIMULAÇÃO FINAL DA ROM PARA *TESTE_CONV* = 1 111
FIG. 9.5 - TEMPO DE ACESSO DA ROM 112
FIG. 9.6 – *LAYOUT* FINAL DA ROM (331,9µM X 233,8µM) 112
FIG. 9.7 - SIMULAÇÃO FINAL DA RAM COM ESTRUTURAS DE TESTE 113
FIG. 9.8 - TEMPO DE ACESSO DA RAM 114
FIG. 9.9 - *LAYOUT* FINAL (274,4µM X 521,3µM) 115
FIG. 9.10 - SIMULAÇÃO FINAL DO BANCO DE REGISTRADORES COM *TESTE_CELL* = 0 116
FIG. 9.11 - SIMULAÇÃO FINAL DO BANCO DE REGISTRADORES COM *TESTE_CELL* = 1 117
FIG. 9.12 - TEMPO DE ACESSO DO BANCO DE REGISTRADORES 118
FIG. 9.13 - *LAYOUT* FINAL DO BANCO DE REGISTRADORES (878,9µM X 880,8µM) 118

LISTA DE SÍMBOLOS, NOMENCLATURAS E ABREVIAÇÕES

BIOS: *Basic Input Output System*

BIST: *Built-In Self-Testing*

CAD: *Computer Aided Design*

CF: *Coupling Fault*

C. I.: Circuito integrado

CISC: *Complex Intruction Set Computer*

Chip: Dispositivo eletrônico composto por circuito integrado encapsulado

CMOS: *Complementary Metal Oxide Semiconductor*

EEPROM: *Electrically-Erasable* PROM

EPROM: *Erasable* PROM

DFT: *Design For Testability*

DL: *Data Line*

DRC: *Design-Rule Check*

EMBRAPA: Empresa Brasileira de Pesquisa Agropecuária

E/S: Entrada/Saída

Flag: Sinalização do processador para um evento ocorrido ou resultado esperado

FPGA: *Field Programmable Gate Array*

DRAM: *Dynamic* RAM

Layout: Conjunto das máscaras que definem as camadas do circuito integrado durante o processo de fabricação

LVS: *Layout versus Schematic*

MAMB: *Multiple Arrays Multiple Bits*

MASB: *Multiple Arrays Single Bit*

MCT: Ministério da Ciência e Tecnologia

PLA: *Programmable Logic Arrays*

PROM: *Programmable* ROM

RAM: *Random Access Memory*

RF: Radio Freqüência

RISC: *Reduced Instruction Set Computer*

ROM: *Read Only Memory*

RTL: *Register-Transfer Level*

SA: *Stuck At*

SAMB: *Single Array Multiple Bits*

SASB: *Single Array Single Bit*

SoC: *System on Chip*

SBLSRWA: *Single Bit-Line Read-and-Write Access*

SCMN: Sistemas em *Chip*, Microssistemas e Nanoeletrônica

SRAM: *Static* RAM

TJB: Transistor Bipolar de Junção

UFPE: Universidade Federal de Pernambuco

UFRGS: Universidade Federal do Rio Grande do Sul

UFRJ: Universidade Federal do Rio de Janeiro

UFSC: Universidade Federal de Santa Catarina

ULA: Unidade Lógica Aritmérica

UnB: Universidade de Brasília

Unicamp: Universidade Estadual de Campinas

USP: Universidade de São Paulo

VHDL: VHSIC *Hardware Description Language*

VHSIC: *Very High Speed Integrated Circuit*

VTC: *Voltage Transfer Characteristic*

1 INTRODUÇÃO

O constante avanço tecnológico dos semicondutores e o aumento do mercado de dispositivos eletrônicos tem impulsionado o desenvolvimento de sistemas computacionais em um único circuito integrado (C.I.). Tais sistemas são tipicamente compostos de milhões de transistores que englobam *hardware* digital e analógico e são conhecidos como SoC's (*Systems on Chip*). O projeto desse tipo de sistema é algo complexo, uma vez que envolve questões como portabilidade, limite de consumo de potência, desempenho, confiabilidade e interferência eletromagnética, entre outras. Apesar da dificuldade inerente ao desenvolvimento de tais circuitos, esse tipo de projeto permite a implementação de novos sistemas e a formação de profissionais capacitados nessa área.

Com essa motivação, foi criado o Instituto do Milênio, que constitui um grande esforço por parte de pesquisadores e da comunidade de microeletrônica e áreas afins para o avanço em projetos de circuitos integrados (CIs). Dentre os objetivos do Instituto do Milênio, podem ser citados a formação de recursos humanos (alunos de iniciação científica, mestres, doutores, pós-doutores e outros), o domínio de processos de fabricação de CI's CMOS, contribuições em algoritmos para ferramentas de CAD avançadas e a experiência em realização de projetos de CI's nas aplicações de RF. Entre os projetos inseridos nesse programa está o "Sistemas em chip, Microssistemas e Nanoeletrônica" (SCMN), que é financiado pelo Ministério da Ciência e Tecnologia (MCT).

Neste projeto, um sistema de comunicação sem fio e de processamento de dados em um único *chip* está sendo desenvolvido através de uma parceria entre UnB, USP, UFSC, UFPE, UFRJ, Unicamp, UFRGS e EMBRAPA. O objetivo de tal trabalho é integrar um sistema de irrigação que controla o volume de água utilizado mantendo a umidade do solo em um nível ótimo. A necessidade hídrica das culturas é determinada por meio da medição da umidade do solo e de dados meteorológicos.

O sistema de irrigação é composto por uma estação-base, estações de campo, e por nós. A estação-base, única na propriedade rural, é a interface entre o usuário e as estações de campo, e concentra as informações oriundas das mesmas. A informação é enviada pelas estações de campo para a estação-base por um *link* sem fio (vide Figura 1.1), e então é disponibilizada para o usuário, que pode programar o comportamento do sistema após a análise dos dados recebidos.

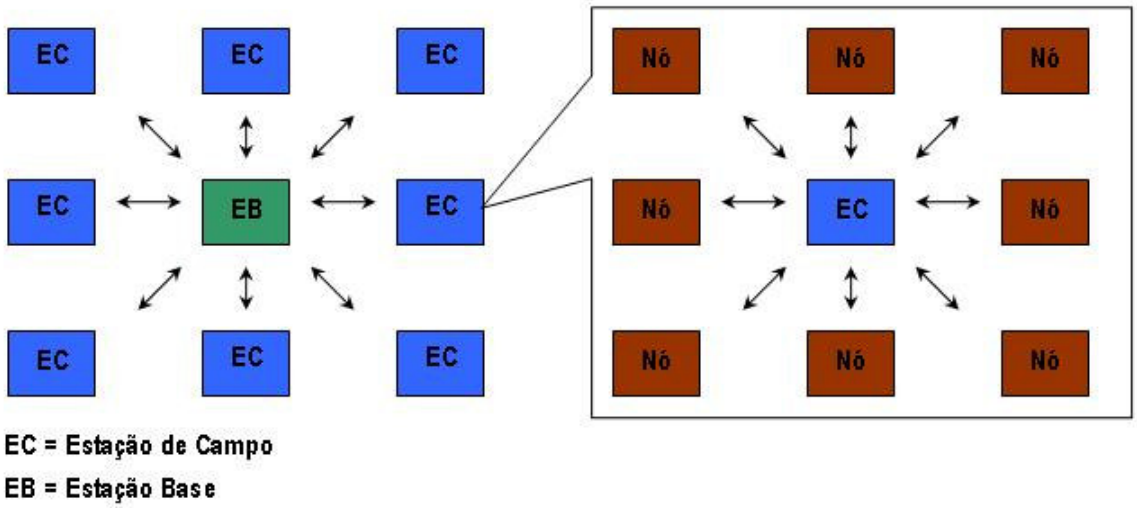


Fig. 1.1 - Sistema de irrigação com nós, estações de campo e estação base [COSTA, 2003 – alterada]

Os nós monitoram a umidade do solo por meio de um sensor de pressão. Eles têm uma área de cobertura de 100 hectares e são compostos basicamente de um SoC CMOS 0,35 μm (microprocessador RISC, oscilador, interfaces serial e A/D e um transceptor de RF operando entre 915 e 927,75 MHz e consumindo 1 mW), sensor de umidade de solo (tensiômetro), antena, coletor solar, fonte de alimentação, bateria, atuador eletromecânico e programas computacionais. Eles mandam as informações coletadas pelo sensor para as estações de campo por um *link* sem fio. A Figura 1.2 mostra o arranjo físico do nó em detalhe.

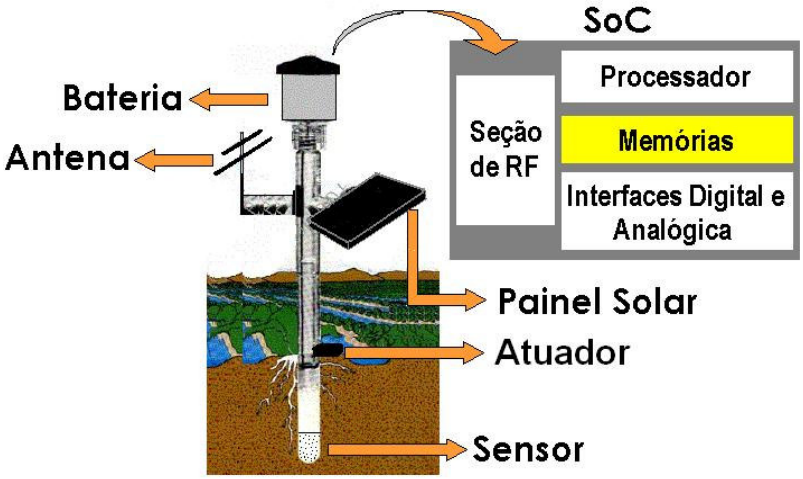


Fig. 1.2 - Detalhes de um nó do sistema de irrigação

O sistema que se encontra na placa mostrada no esquema da Figura 1.2 é composto

por um microprocessador, memórias RAM e ROM, interfaces analógica e digital, e um transceptor RF. A arquitetura do microprocessador é a RISC de 16 bits, com *clock* de 10 MHz, alimentação de 3.3 V e consumo de potência de 100 mW. Ele possui uma unidade lógico-aritmética (ULA) operando em ponto fixo, 16 instruções e 16 registradores de 16 bits.

A contribuição da presente dissertação está em projetar e implementar os registradores e as memórias (RAM e ROM) que compõem o SoC em questão. Foi definida inicialmente uma estrutura RAM básica de 256 linhas por 256 colunas, que utiliza células 6T e amplificadores diferenciais como sensores. Foi definida também, a princípio, a necessidade da utilização de uma memória ROM entre 256 a 512 bits para armazenamento de rotinas de teste. O banco de registradores definido possui 16 registradores de 16 bits [BENÍCIO, 2002]. Algumas especificações iniciais foram alteradas à medida em que foi feito um estudo sobre as topologias disponíveis para realização dos circuitos, conforme será descrito neste trabalho. Todo o projeto é voltado para a testabilidade.

No Capítulo 2 foi feita uma revisão bibliográfica mostrando diferentes estruturas de memórias, e apresentando noções de testabilidade de circuitos integrados. O Capítulo 3 trata da metodologia que foi adotada durante a realização do projeto. O funcionamento do microprocessador é brevemente descrito no Capítulo 4, que abrange também as alterações que foram recentemente propostas em [COSTA, 2004a]. Os projetos da ROM, RAM e do banco de registradores são detalhados nos Capítulos 5, 6 e 7, respectivamente. O Capítulo 8 apresenta uma proposta para a integração das memórias, e uma estrutura menor como veículo de validação. Os resultados de todo o trabalho são discutidos no Capítulo 9, e finalmente as conclusões finais são apresentadas no Capítulo 10.

2 REVISÃO BIBLIOGRÁFICA

Sistemas digitais requerem a capacidade de armazenar e restaurar grandes quantidades de informação em altas velocidades. Memórias são circuitos ou sistemas que armazenam informações digitais [HODGES, 2003]. Neste capítulo, serão classificados diversos tipos de memórias, com ênfase no projeto de memórias ROM e RAM estáticas (SRAM). Serão também abordadas células de registradores, e metodologias de teste em circuitos integrados.

2.1 ROM

Uma ROM (*Read Only Memory*) é classificada como memória porque tem um conjunto de endereços que podem ser lidos, e cujos valores não podem ser alterados, sendo fixados em geral no momento em que a mesma é fabricada. Existem também ROM programáveis (PROM), que podem ser programadas eletronicamente, quando o projetista tiver valores para gravar. Existem ainda PROM cujo conteúdo pode ser apagado, as chamadas EPROM. O processo de apagar uma EPROM é muito lento, sendo preciso submetê-la à ação de raios ultravioletas.

A memória de apenas leitura é usada em uma grande variedade de aplicações de sistemas digitais. Uma aplicação muito popular é encontrada em sistemas microprocessados, nos quais ela é usada para armazenar as instruções do programa da BIOS. A ROM é particularmente adequada para tal aplicação porque não é volátil, isto é, ela retém seu conteúdo quando a fonte de alimentação é desligada [SEDRA, 2000].

Uma ROM pode ser vista como um circuito lógico combinacional em que a entrada é o conjunto de bits de endereçamento e a saída, o conjunto de bits de dados recuperados a partir da localização endereçada [SEDRA, 2000]. Ela é totalmente decodificável: contém uma palavra para cada possível combinação das entradas. O número de entradas endereçáveis determina o número de bits de endereço: se a ROM tem 2^n entradas endereçáveis, então devem existir n bits de endereço [PATTERSON, 2000].

2.1.1 ROM MOS

A ROM MOS consiste em uma matriz de MOSFET tipo enriquecimento cujas portas são conectadas às linhas de palavras, cujas fontes estão aterradas e cujos drenos estão

conectados às linhas de bits (Figura 2.1). Cada linha de bit está conectada à fonte de alimentação via um transistor de carga PMOS, da mesma forma que uma lógica pseudo-NMOS. Há um transistor NMOS em uma célula particular se essa célula estiver armazenando um "0"; se a célula estiver armazenando um "1", não há esse transistor. O decodificador de linhas seleciona uma das palavras pelo aumento da tensão na linha da palavra correspondente. Os transistores das células conectados a essa linha de palavra conduzirão, puxando a tensão das linhas de bits (nas quais os transistores na linha selecionada estão conectados) para o nível lógico zero. As linhas de bits que estão conectadas às células (da palavra selecionada) sem transistores (isto é, as que estão armazenando os dígitos "1") permanecerão com a tensão da fonte de alimentação (nível lógico um) por causa da ação dos transistores de carga PMOS (*pull-up*). Desse modo, os bits da palavra endereçada podem ser lidos [SEDRA, 2000].

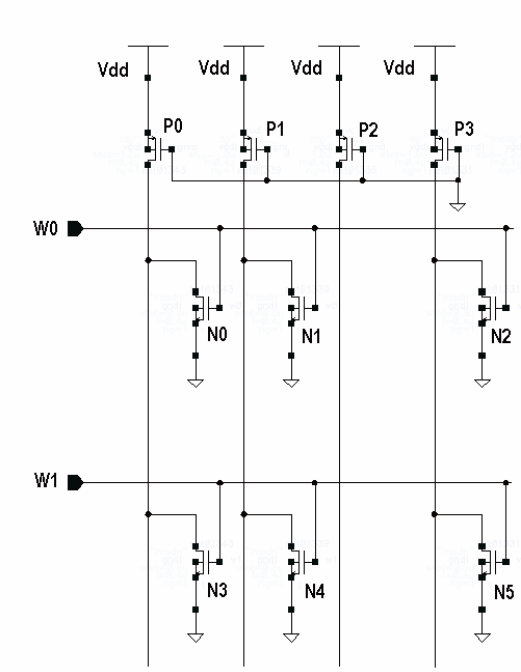


Fig. 2.1 - Trecho de uma ROM MOS

2.1.2 ROM PROGRAMÁVEIS POR MÁSCARA

A fim de se evitar que se faça um projeto dedicado para cada ROM encomendada, as ROM são fabricadas usando um processo conhecido como programação por máscara. Os circuitos integrados são fabricados em uma lâmina de silício usando uma seqüência de etapas de processamento. Uma das etapas finais do processo de fabricação consiste em depositar uma camada de alumínio em toda a superfície da lâmina, e depois remover, por

corrosão, o alumínio das regiões indesejadas utilizando uma máscara. Dessa forma, somente restará o alumínio onde devem existir interconexões. Essa última etapa pode ser usada para programar (isto é, armazenar o traçado desejado) uma ROM. Por exemplo, se a ROM for feita de transistores MOS, os mesmos são incluídos em todas as localizações dos bits, mas somente as portas daqueles transistores nos quais os zeros serão armazenados serão conectadas às linhas de palavras. Esse traçado é determinado na máscara, que é produzida de acordo com as especificações do usuário. As vantagens econômicas do processo de programação são claras, já que todas as ROM são fabricadas de modo similar. A diferenciação do produto ocorre apenas nas etapas finais de fabricação.

2.1.3 ROM PROGRAMÁVEIS (PROM E EPROM)

As PROM são ROM que podem ser programadas pelo usuário, mas apenas uma vez. Um arranjo típico empregado nas PROM com TBJ (transistor bipolar de junção) envolve o uso de fusíveis de silício policristalino para conectar o emissor de cada TBJ à coluna de dígitos correspondentes. Dependendo do conteúdo desejado de uma célula da ROM, o fusível pode ser deixado intacto ou pode ser queimado usando uma corrente alta. O processo de programação é irreversível.

Uma ROM programável e apagável, ou EPROM, pode ser apagada e reprogramada quantas vezes o usuário desejar. A programação é feita com a aplicação de uma tensão alta na porta selecionada, e o apagamento, com luz ultravioleta. Contudo, deve ser observado que o processo de apagamento e reprogramação é demorado e é idealizado para ser executado com pouca freqüência.

As ROM programáveis mais versáteis são as PROM eletricamente apagáveis (*Electrically-Erasable* PROM – EEPROM), que não têm necessidade da utilização de iluminação ultravioleta [SEDRA, 2000].

2.1.4 ALTERNATIVAS PARA IMPLEMENTAÇÃO DE UMA ROM

A ROM pode ser implementada com apenas um transistor por bit armazenado. Trata-se de uma estrutura de memória estática, na qual o estado é mantido indefinidamente – mesmo sem a fonte de alimentação. Um *array* de uma ROM é usualmente implementado com um arranjo NOR, conforme mostra a Figura 2.2.

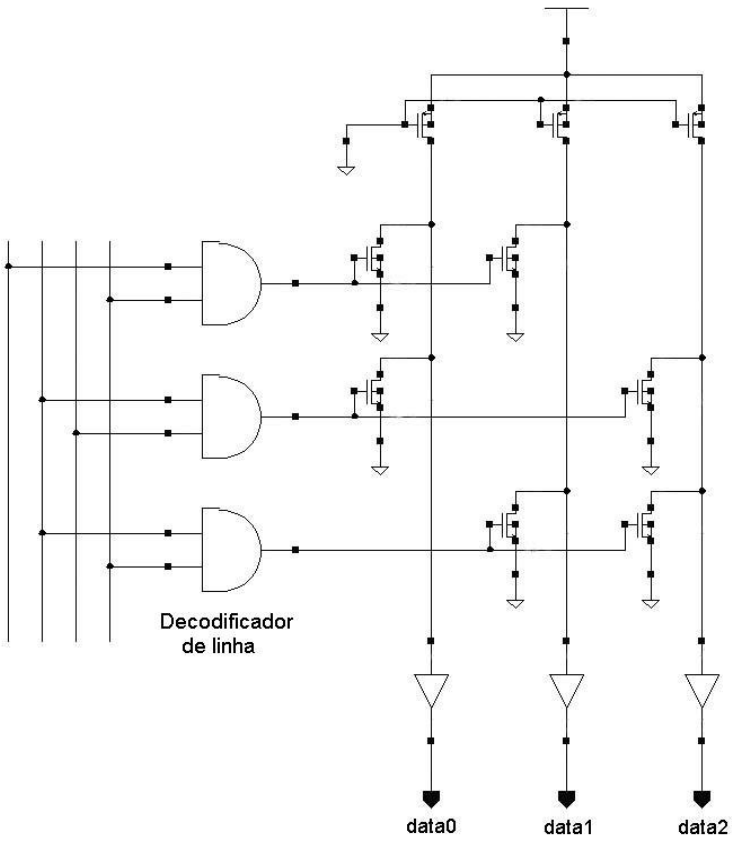


Fig. 2.2 - Arranjo NOR [WESTE, 1993]

Pode-se notar que esse arranjo é semelhante ao mostrado na Figura 2.1, apenas incluindo os decodificadores de linha e os inversores nas saídas.

Uma alternativa CMOS dinâmica pode ser vista na Figura 2.3. Neste caso, as linhas de palavras são forçadas para o nível lógico baixo enquanto as linhas de bit são pré-carregadas. Isso assegura que não haja corrente DC. Depois que os *pull-ups* são desligados, uma das linhas de palavras é ativada. A temporização para assegurar essa seqüência de eventos deve ser cuidadosamente projetada e simulada, o que torna o seu projeto mais complexo.

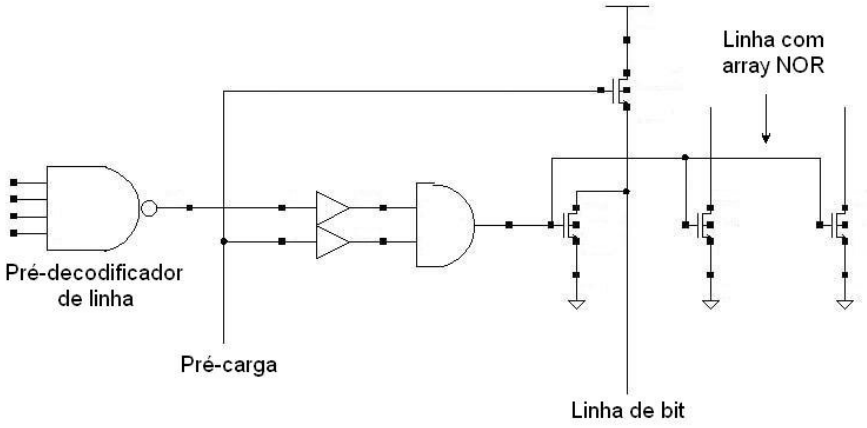


Fig. 2.3 - ROM dinâmica [WESTE, 1993]

A ROM pseudo-NMOS é um projeto mais simples, e não requer temporização. A dissipação de potência DC pode ser significativamente reduzida ligando-se os *pull-ups* de acordo com a decodificação de coluna. A Figura 2.4 mostra um exemplo no qual somente uma das quatro linhas de bit é acionada por vez [WESTE, 1993].

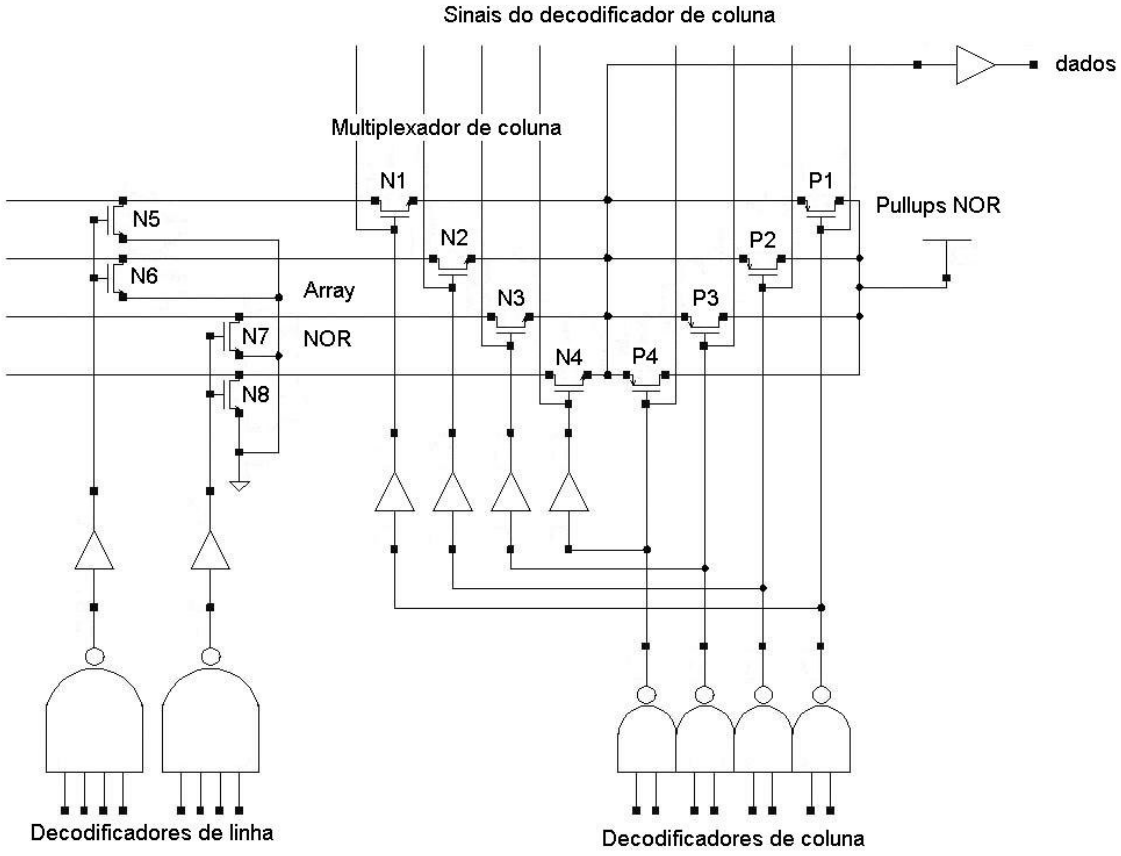


Fig. 2.4 - Circuito ROM com economia de potência [WESTE, 1993]

Apesar da vantagem da economia de potência, a complexidade dessa estrutura aumentaria muito para um barramento de saída de dados de 16 bits, que é o requerido para o projeto.

2.2 RAM

A memória de acesso aleatório (*Random-Access Memory* - RAM) é do tipo em que o tempo necessário para armazenagem (escrita) e para recuperação (leitura) da informação independe da localização física (dentro da memória) onde a mesma é armazenada.

A maior parte da pastilha de uma memória consiste de células nas quais os bits são armazenados. Cada célula de memória é capaz de armazenar um único bit. A Figura 2.5 ilustra a arquitetura de uma memória descrita em [SEDRA, 2000]. A matriz de células tem 2^M linhas e 2^N colunas, o que significa uma capacidade total de 2^{M+N} bits. Cada célula está conectada a uma das 2^M linhas de palavra, e em uma das 2^N linhas de bit. Uma célula é selecionada ativando-se sua linha de palavra e sua linha de bit. As memórias são capazes de selecionar 4, 8, 16, 32 ou 64 colunas de uma linha simultaneamente, dependendo da aplicação [HODGES, 2003].

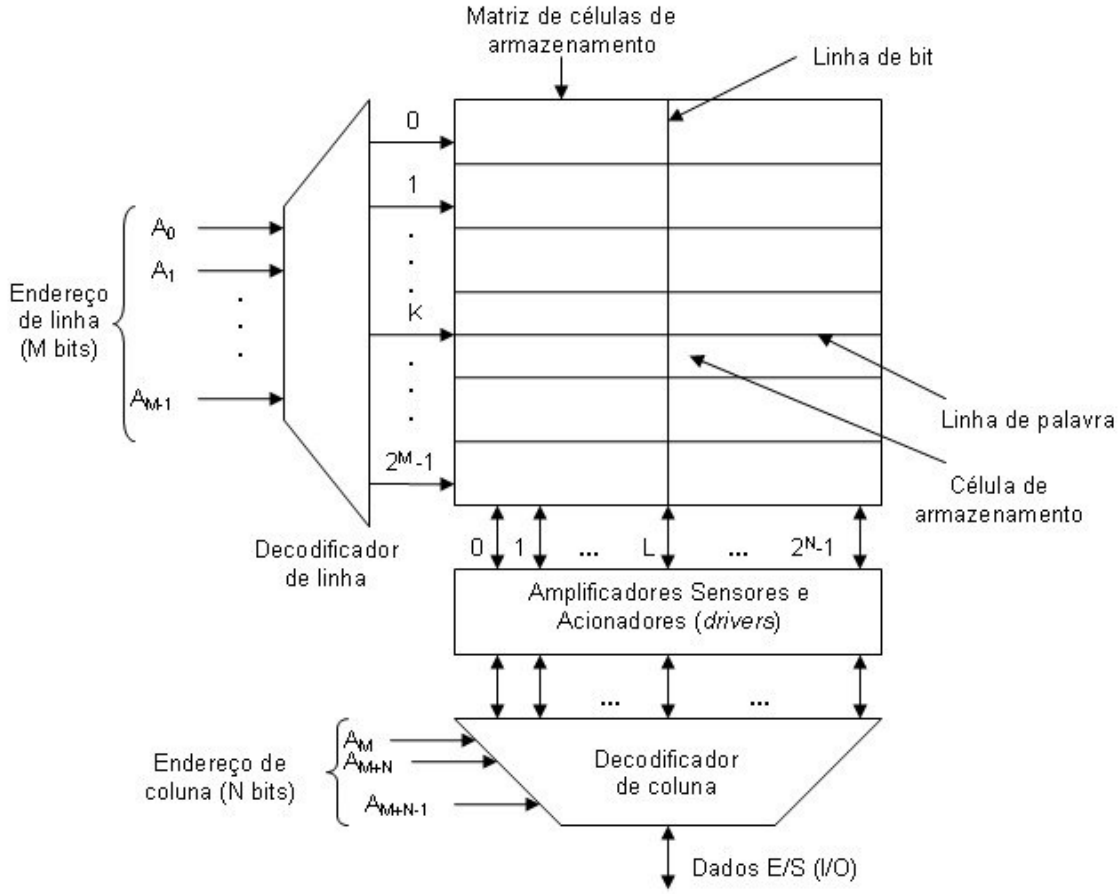


Fig. 2.5 - Arquitetura genérica de uma memória RAM [SEDRA, 2000]

A ação de ativar uma das 2^M linhas de palavra é realizada pelo decodificador de linhas, que é um circuito lógico combinatório que aumenta a tensão da linha de palavra cujo endereço de M bits é aplicado à entrada do decodificador. Os bits de endereço são representados por $A_0A_1 \ldots A_{M-1}$.

Quando a k-ésima linha de palavra estiver ativada para uma operação de leitura, por exemplo, todas as 2^N células na linha k fornecerão seus conteúdos para suas respectivas linhas de bit. Portanto, se a célula na coluna L estiver armazenando um "1", a tensão da linha de bit número L aumentará de um pequeno valor. A razão da tensão de leitura da saída ser pequena é que a célula é pequena, uma vez que há um número muito grande de células. Essa tensão é aplicada a um amplificador sensor que, por sua vez, tem uma excursão de sinal digital ampla em sua saída (nesse caso, de 0 a Vdd). Esse sinal, junto com os sinais de saída das outras células da linha selecionada, é encaminhado ao decodificador de coluna, onde o sinal da coluna cujo endereço de N bits está aplicado na entrada do decodificador faz com que o sinal de saída apareça na linha de dados de entrada/saída (I/O).

Uma operação de escrita ocorre de maneira semelhante: o bit de dado a ser armazenado é aplicado na linha I/O. A célula em que o bit de dado será armazenado é selecionada por meio de seu endereço de coluna. O amplificador sensor da coluna selecionada age como um reforçador para escrever o sinal aplicado na célula selecionada.

O tempo de acesso à memória é o tempo entre a iniciação de uma operação de leitura e o surgimento do dado na saída. O tempo de ciclo de memória é o tempo mínimo permitido entre duas operações consecutivas da memória. A memória MOS tem tempos de acesso e de ciclo na faixa de alguns ns a algumas centenas de ns [SEDRA, 2000].

A arquitetura genérica mostrada anteriormente possui uma particularidade: os dados de entrada e saída possuem um único barramento, o que não se aplica ao caminho de dados do processador para o qual a memória será projetada. Uma arquitetura típica, proposta por [WESTE, 1993], com barramentos separados para dados de entrada e de saída, está mostrada na Figura 2.6. O *array* no centro consiste de 2^n por 2^m bits de armazenagem (na verdade, 2^{n-k} por 2^{m+k}). Um decodificador de linha endereça uma palavra de 2^m bits dentre 2^{n-k} palavras. O decodificador de coluna endereça 2^k de 2^m bits da linha acessada. Este decodificador de coluna acessa um multiplexador, que irá direcionar os dados endereçados para e a partir das interfaces para o mundo externo.

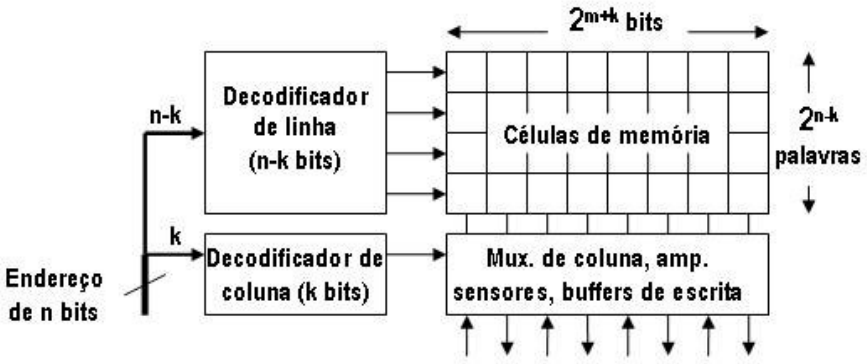


Fig. 2.6 - Arquitetura de um *chip* de memória [WESTE, 1993 - alterada]

A Figura 2.7 mostra uma linha e uma coluna da arquitetura da Figura 2.6, juntamente com os circuitos requeridos para o funcionamento da mesma. O decodificador de linha selecionará uma dentre n-k linhas, e pode ser implementado com portas AND. O circuito de condicionamento da linha de bit, a célula RAM, os amplificadores sensores, os multiplexadores de coluna e os *buffers* de escrita formam um circuito que provê uma leitura e uma escrita seguras na célula. As linhas de bit normalmente são sinais complementares. O decodificador de coluna é similar ao de linha, mas seleciona uma dentre k colunas, onde k é normalmente menor do que n, e aciona um multiplexador (ao invés de um seletor). Freqüentemente, o decodificador de coluna pode ser misturado ao multiplexador de coluna.

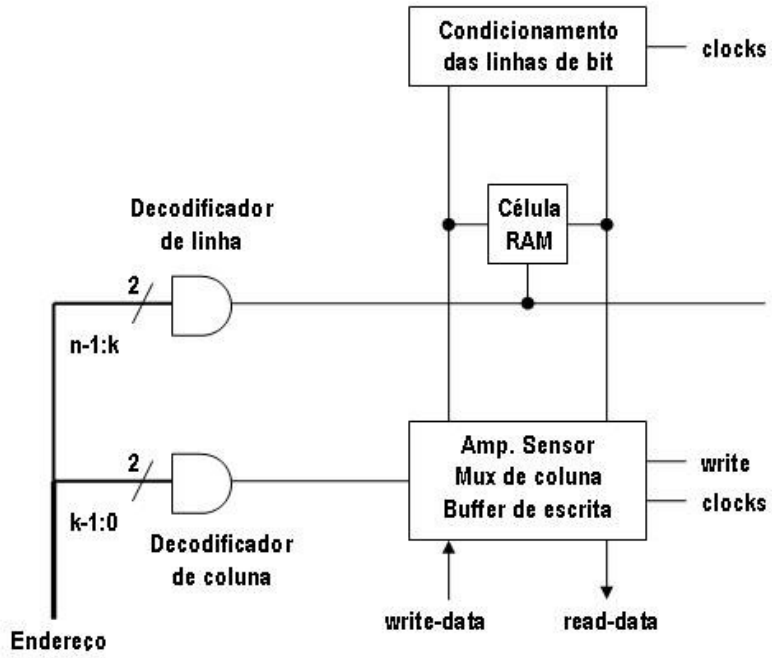


Fig. 2.7 - Célula de RAM com circuitos periféricos [WESTE, 1993 – alterada]

Existem variações dos circuitos apresentados, de acordo com as especificações de densidade (capacidade em bits), velocidade e margem de ruído.

Como exemplo, a Figura 2.8 mostra a arquitetura de uma RAM de 64 kb. Neste caso, a matriz possui 65536 células, e n = m = 8 ($2^{m+n} = 2^8$ x 2^8). A memória usa um endereçamento de 16 bits para acessar um único bit, que será mostrado na saída [HODGES, 2003].

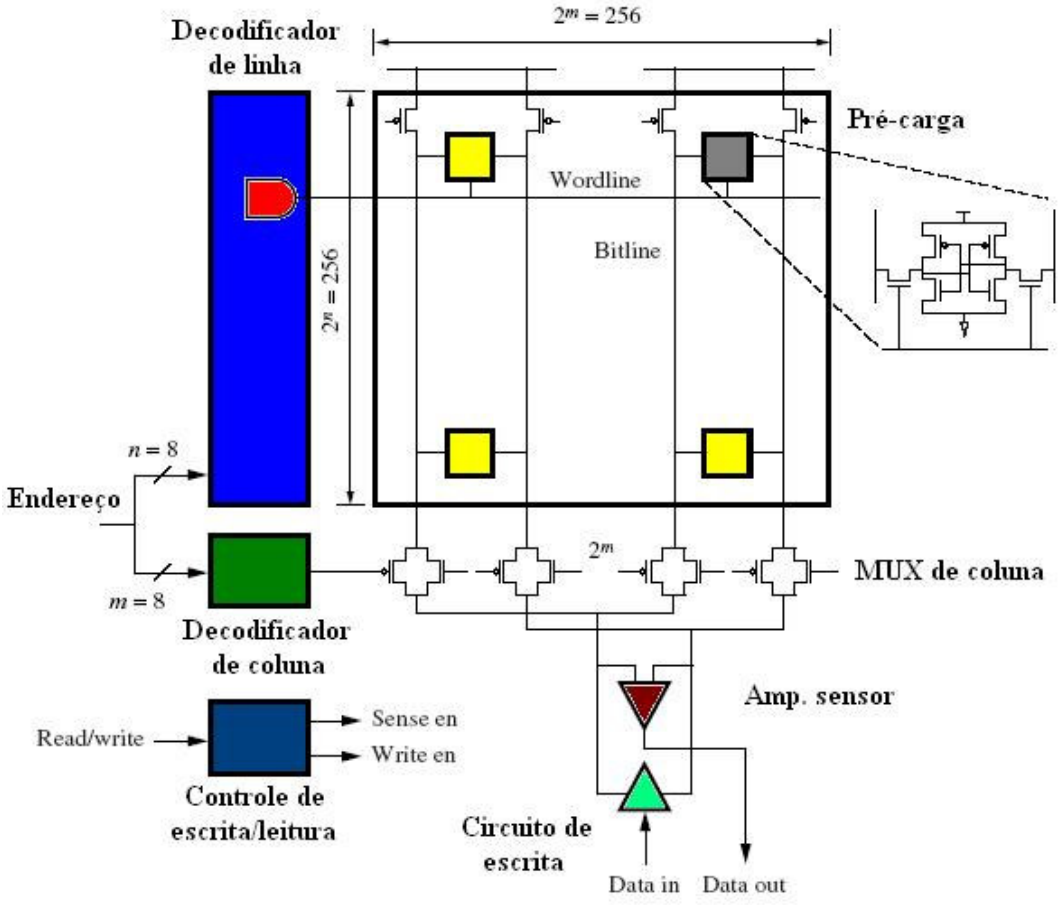


Fig. 2.8 - Arquitetura de uma RAM de 64 kb [HODGES, 2003 – alterada]

2.2.1 SRAM X DRAM

As RAM estáticas (*Static* RAM – SRAM) utilizam *latches* ou *flip-flops* como células básicas para armazenar a informação. As RAM dinâmicas (*Dynamic* RAM – DRAM), por outro lado, armazenam os dados binários em capacitores. Apesar de nesse caso haver uma redução na área da célula, os circuitos de escrita e leitura são mais elaborados. Além disso, enquanto as RAM estáticas retêm seus dados enquanto a fonte de alimentação está ligada, as RAM dinâmicas precisam regenerar periodicamente os dados armazenados nos capacitores. Essa operação é conhecida como *refresh*, e é necessária porque os capacitores

se descarregam lentamente em virtude das correntes de fuga, que sempre existem [SEDRA, 2000].

As SRAM são mais fáceis de serem projetadas, e menos problemáticas do que as DRAM. Além disso, elas tendem a ser mais rápidas, apesar de maiores, do que as DRAM [WESTE, 1993].

2.2.1.1 A célula de memória dinâmica

A célula que se tornou o padrão industrial, conhecida como célula de um transistor (vide Figura 2.9), consiste de um único transistor MOSFET canal n, conhecido como transistor de acesso (a*ccess transistor*), e de um capacitor de armazenamento Cs. A porta do transistor é conectada à linha de palavra e sua fonte (dreno) é conectada à linha de bit. Neste caso, ao contrário das SRAMs, apenas uma linha de bit é utilizada.

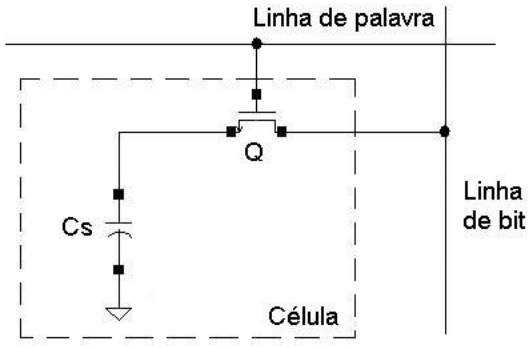


Fig. 2.9 - Célula de um transistor [SEDRA, 2000]

O bit de informação é armazenado como carga no capacitor *Cs*. Quando a célula armazena um "1", o capacitor está carregado com Vdd-Vt. Quando um "0" está armazenado, o capacitor é descarregado até alcançar a tensão zero. Devido aos efeitos de fuga, a carga do capacitor vai se reduzindo, de forma que a célula precisa ser regenerada periodicamente pela operação de restauração. Durante essa operação, o conteúdo da célula é lido e o dado é reescrito, restaurando, assim, a tensão do capacitor a seu valor original [SEDRA, 2000].

2.2.1.2 A célula de memória estática

O circuito convencional (vide Figura 2.10) consiste em um *flip-flop* formado por dois inversores com conexões cruzadas e dois transistores de acesso (N1 e N2). Eles conduzem

quando a linha de palavra é selecionada, conectando assim o *flip-flop* às linhas de bit Bb e B, que são complementares entre si. A seguir, são detalhadas as operações de leitura e escrita nessa célula [SEDRA, 2000].

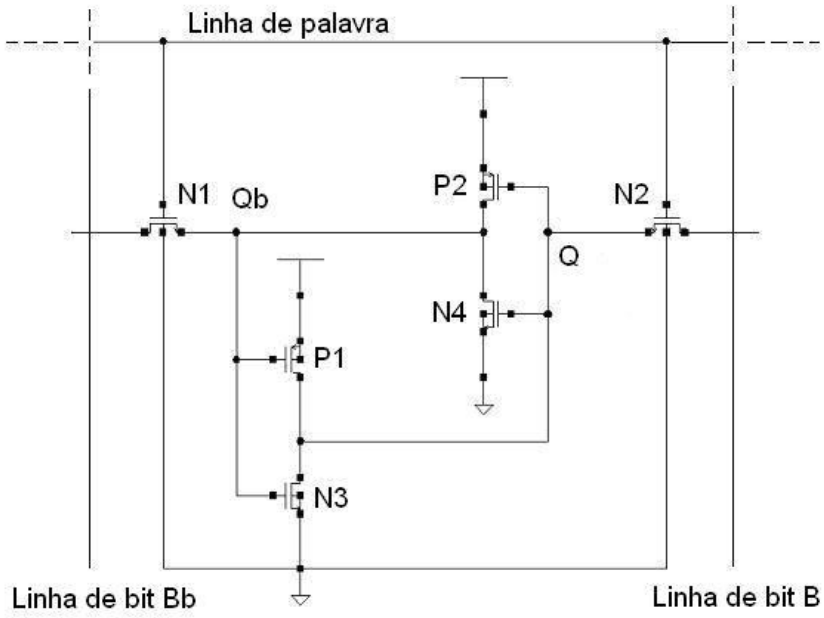


Fig. 2.10 - Célula 6-T [WESTE, 1993], [SEDRA, 2000], [CHANDRAKASAN, 2001], [HODGES, 2003]

Na operação de leitura, assumindo-se que a célula está armazenando um "1", Q estará em Vdd e Qb em 0 V. Antes do início da operação de leitura, as colunas B e Bb são pré-carregadas a uma tensão intermediária entre Vdd e 0V (usualmente Vdd/2). Quando a linha de palavra é selecionada, N1 e N2 são ligados e flui corrente de Vdd através de P2 e N2 para a linha B, carregando a capacitância da linha B (C_B). No outro lado do circuito, flui corrente através da linha Bb através de N1 e N3 para o terra, descarregando a capacitância C_{Bb}. Dessa forma, durante a operação de leitura de um "1", a tensão sobre C_B aumentará e a tensão sobre C_{Bb} diminuirá, o que faz surgir uma tensão diferencial entre as linhas B e Bb para que o amplificador sensor possa detectar o dado presente na célula. É importante observar que a célula deve ser projetada de tal forma que as mudanças de tensão em Q e Qb sejam suficientemente pequenas para que o *flip-flop* não mude de estado durante a leitura.

Na operação de escrita, assumindo-se que a célula esteja armazenando um "1" e que se deseja escrever um "0", a tensão da linha B deve ser abaixada para 0V e a da linha Bb, aumentada para Vdd. A célula deve ser selecionada colocando-se a linha de palavra em Vdd. Nesta operação, as capacitâncias das linhas B e Bb são carregadas (ou descarregadas)

com relativa rapidez pelo circuito de acionamento. O resultado final é que o tempo de atraso dessa operação é dominado pelo atraso da linha de palavra.

Os transistores p podem ser substituídos por resistores de poli-silício de valores altos, conforme mostra a Figura 2.11, se o processo suportar essa opção. O valor do resistor tem que ser escolhido de tal forma que evite a fuga de qualquer valor armazenado na RAM durante a operação de escrita. Geralmente os valores estão na faixa de 100 a 1000 Megaohms [WESTE, 1993]. Apesar de potência e da área serem menores, isso acontece às custas do aumento da complexidade do processo para se utilizar os resistores [HODGES, 2003].

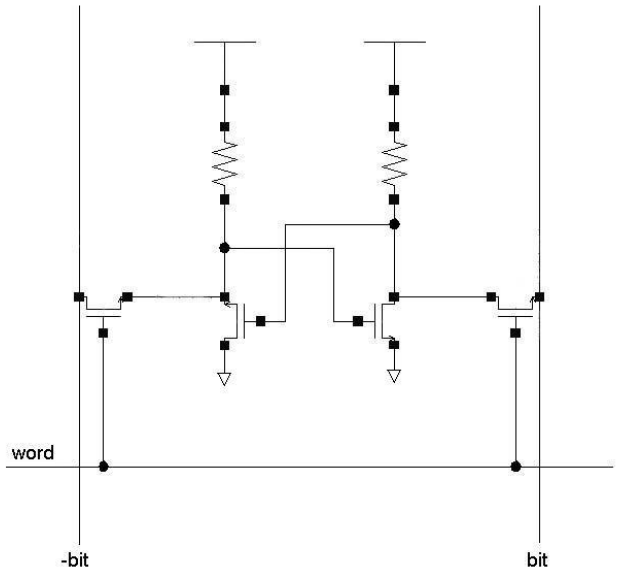


Fig. 2.11 - Célula com resistores [WESTE, 1993]

[WANG, 2000] apresenta uma alternativa para a célula 6-T, que consiste em uma célula com uma única linha de bit, e na qual a leitura e a escrita são simultâneas. Trata-se da célula SBLSRWA (*Single Bit-Line Read-and-Write Access*). Conforme mostra a Figura 2.12, a fonte do dispositivo N3 está conectada à linha de escrita WWL (*write word line*) ao invés de no terra, conforme é feito na célula convencional. O lado esquerdo da célula SBLSRWA é conectado à linha de bit WBL (*write bit line*) através do transistor N1, que é controlado pela linha WWL. O lado direito da célula é conectado à linha de leitura RBL (*read bit line*) através do transistor de passagem N2, que é controlado pela linha de palavra RWL (*read word line*). Com essa configuração, a leitura e escrita simultâneas podem ser facilitadas. Durante a operação de escrita de um "1" lógico na célula, e supondo-se que a mesma armazena inicialmente "0", a tensão no nó n1 é igual a zero. Enquanto na célula 6-

T convencional o transistor N3 era ativado e descarregava o nó n1, nessa célula isso não acontece, já que a fonte de N3 está conectada à linha WWL e, nesse caso, onde se quer escrever "1" na célula, está em nível alto, usualmente Vdd. Como resultado, a tensão no nó n1 pode facilmente alcançar Vdd-Vtn. Conseqüentemente, N4 é ativado e P2 é desativado. Assim, o nó n2 muda para 0V, o que faz com que a tensão no nó n1 aumente para Vdd.

A célula descrita acima foi implementada em [WANG, 2000] em tecnologia 0.25 μm CMOS, com Vdd de 1V. Apesar da vantagem em relação à célula 6-T de não haver o risco de mudança de estado da célula durante a operação de leitura, o caminho de dados do processador não requer que sejam feitas uma leitura e escrita simultâneas.

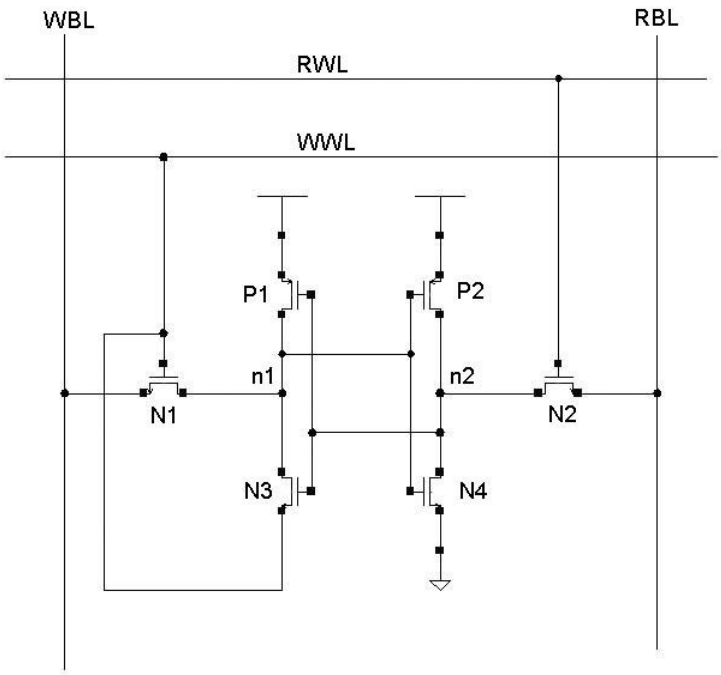


Fig. 2.12 - Célula SBLSRWA [WANG, 2000]

Uma célula de 4 transistores foi proposta em [JOUBERT, 1999] para reduzir a área do *chip* e, conseqüentemente, aumentar a capacidade de armazenamento por área e reduzir o custo. A função de armazenamento da célula 6-T está no par de inversores cruzados. Os transistores de acesso não são necessários para essa função, mas apenas para fornecer um canal para ler e escrever o dado. Se os mesmos puderem ser suprimidos, e o par de inversores puder ser acessado de outra forma, a área será reduzida.

A Figura 2.13 mostra uma célula com 4 transistores. Para acessá-la, deve-se variar as tensões das fontes de cada transistor seletivamente. Sendo assim, a operação de leitura é realizada retirando-se a tensão da fonte de um dos transistores, e monitorando-se a corrente

no inversor oposto, ou seja, no inversor que não contém o transistor cuja tensão de fonte foi modificada. A operação de escrita é feita reduzindo-se efetivamente a tensão de alimentação da célula, causando uma redução da tensão de limiar dos inversores.

Essa célula foi implementada em [JOUBERT, 1999] em tecnologia 1.2 μm CMOS, com uma tensão de alimentação igual a 5 V. Apesar de ocupar uma área menor do que a 6-T, a implementação dos circuitos para acesso à célula seria muito complexa.

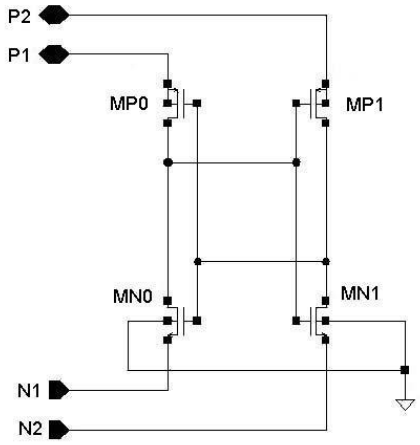


Fig. 2.13 - Célula SRAM com 4 transistores [JOUBERT, 1999]

2.2.2 CIRCUITOS PERIFÉRICOS DA MEMÓRIA RAM ESTÁTICA

2.2.2.1 Amplificador sensor e pré-carga

Um tipo de amplificador sensor diferencial [SEDRA, 2000] que pode ser usado diretamente em SRAM que utilizam células com linhas B e Bb, cuja arquitetura é similar à da Figura 2.5, está mostrado na Figura 2.14. Deve-se assumir que a célula de memória cuja saída deve ser amplificada gera uma tensão diferencial de saída entre as linhas B e Bb. Esse sinal será aplicado aos terminais de entrada do amplificador sensor que, por sua vez, responde gerando um sinal com chaveamento lógico pleno (0 a Vdd) em seus terminais de saída. Nota-se que os terminais de saída e de entrada são os mesmos.

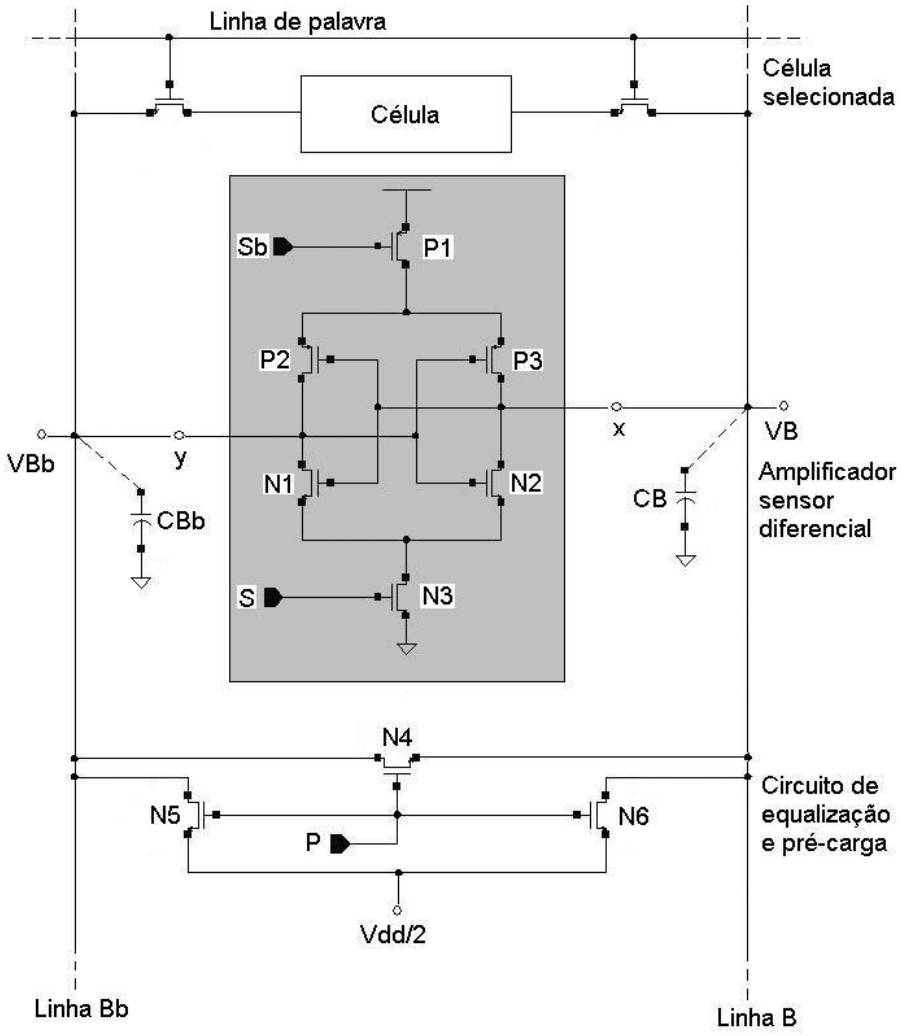


Fig. 2.14 - Amplificador sensor [SEDRA, 2000]

O circuito da Figura 2.14 é um *latch* formado pelo acoplamento de dois inversores CMOS, um formado pelos transistores N1 e P2, e outro, pelos transistores N2 e P3. Os transistores N3 e P1 atuam como chaves que conectam o amplificador sensor ao terra e ao Vdd somente quando a ação de sensoriamento é necessária (sinal $S = 1$). Se esse não for o caso, S é zero e o amplificador sensor é desligado, o que conserva energia, já que usualmente há um amplificador sensor por coluna, resultando em milhares de amplificadores por pastilha. Os terminais x e y são tanto a entrada quando a saída do amplificador, e são conectados às linhas B e Bb.

Pode-se observar ainda na Figura 2.14 os circuitos de pré-carga e equalização. Quando P vai para o nível alto antes da operação de leitura, todos os três transistores conduzem. Enquanto N5 e N8 pré-carregam as linhas B e Bb em Vdd/2, o transistor N4 acelera esse processo equalizando as tensões em ambas as linhas. Essa equalização é crítica

para a operação do amplificador, já que qualquer variação de tensão entre as linhas de bit pode resultar em uma interpretação errônea em relação ao sinal de entrada. A seqüência de eventos durante a operação de leitura é detalhada a seguir:

1. Os circuitos de pré-carga e de equalização são ativados fazendo-se P = 1. As tensões das linhas de bit são equalizadas em Vdd/2. Então, o sinal P =0 e as linhas de bit ficam flutuantes durante um breve intervalo.
2. A linha de palavra é ativada, e a célula é conectada às linhas de bit. Uma tensão surge entre B e Bb. Se a célula estiver armazenando um "1", a tensão de B é maior do que a de Bb. Caso contrário, a tensão de Bb é maior do que a de B, e a célula está armazenando um "0".
3. O amplificador sensor é acionado através do sinal de controle *S*, que o conecta ao terra e ao Vdd. Como, inicialmente, os terminais de entrada dos inversores estão em Vdd/2, os mesmos estarão operando em sua região de transição, ou seja, inicialmente o *latch* estará operando em seu ponto de equilíbrio instável. Assim, dependendo do sinal entre os terminais de entrada, o *latch* se deslocará rapidamente para um de seus dois pontos de equilíbrio estável (vide Figura 2.15).

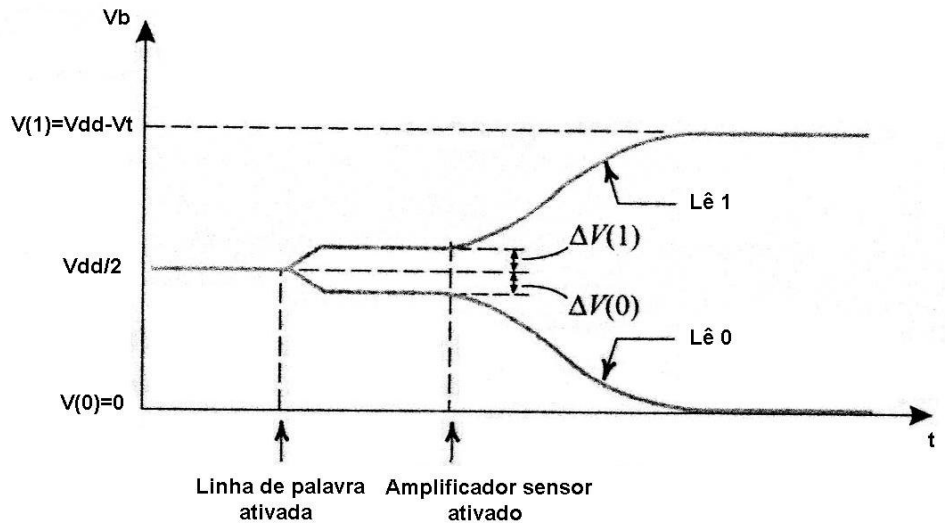


Fig. 2.15 - Operação do amplificador sensor [SEDRA, 2000]

De acordo com [WESTE, 1993], um método de ler uma célula RAM seria pré-carregar as linhas de bit e então habilitar o decodificador, selecionando a linha de palavra. Para um dado par de linhas de bit, a célula RAM irá abaixar o nível de tensão de uma delas, dependendo do dado armazenado na mesma. Pode-se usar um circuito *pull up* com

um transistor p para pré-carregar cada linha. Neste exemplo, um amplificador sensor simples seria composto por um inversor (vide Figura 2.16). Apesar de simples, essa implementação é lenta, já que o tempo necessário para se ter uma resposta seria o tempo de um transistor *pull-down* e um transistor de acesso da célula alcançarem o limiar do inversor. As formas de onda para essa implementação estão mostradas na Figura 2.17. Para otimizar a velocidade, pode-se ajustar o limiar do inversor para acima de Vdd/2, mas com uma margem de ruído aceitável. Alternativamente, pode-se pré-carregar as linhas de bit com transistores n, ou seja, as linhas de bit serão pré-carregadas para Vdd – Vt. Isso pode melhorar muito a velocidade de acesso à célula e reduzir a dissipação de potência, já que as linhas não alcançam o valor da tensão de alimentação [WESTE, 1993].

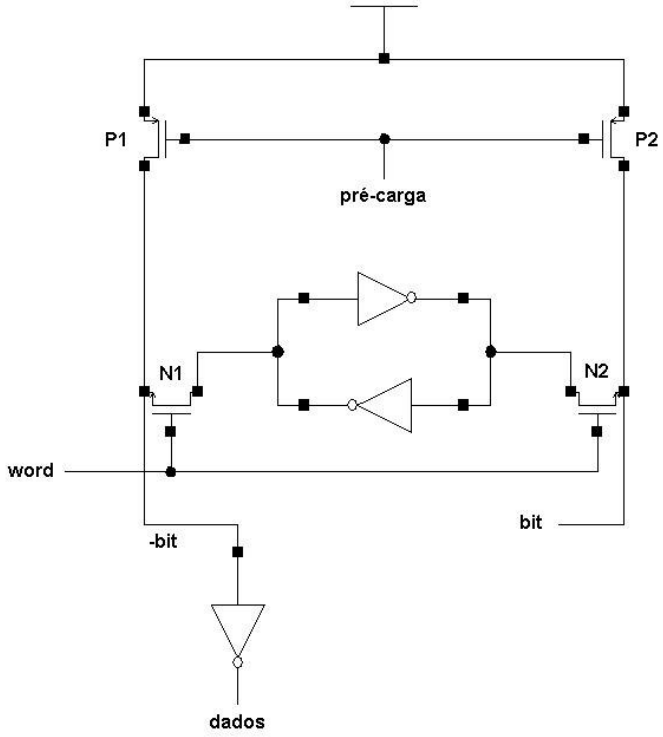


Fig. 2.16 - Pré-carga com Vdd [WESTE, 1993]

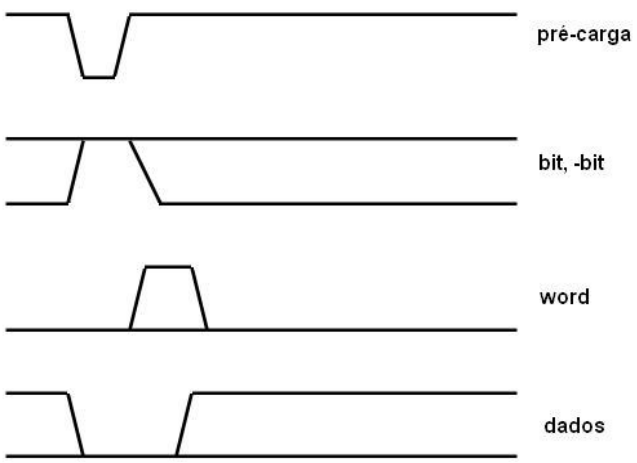


Fig. 2.17 – Formas de onda para pré-carga com Vdd [WESTE, 1993]

O principal aspecto do ciclo de leitura de uma RAM com pré-carga é a temporização entre a decodificação do endereço, o pulso de pré-carga, e a habilitação do decodificador de linha. Se a linha de palavra for selecionada antes do término do pulso da pré-carga, a célula poderá acessar ambas as linhas de bit, que estão em nível alto, e pode conseqüentemente mudar o seu estado. Se o endereço muda antes do término do ciclo de pré-carga, mais de uma linha de palavra será selecionada e mais de uma célula poderá mudar as linhas de bit para nível lógico baixo, causando uma leitura errônea de dados [HODGES, 2003].

A Figura 2.18 mostra uma versão simplificada de um circuito de leitura apresentado em [HODGES, 2003]. O circuito de pré-carga, neste caso, é composto por dois transistores PMOS, e as tensões das linhas de bit vão para Vdd quando os transistores M7 e M8 são ativados por meio do sinal pc. Então, os sinais de endereço, dados e *clock* são aplicados. O endereço habilita uma determinada linha de palavra e uma determinada coluna. Usualmente a habilitação da coluna é feita simultaneamente à habilitação do amplificador sensor (vide Figura 2.19), que é usado para gerar saídas em níveis lógicos alto e baixo a partir da diferença de tensão entre as linhas de bit. O circuito de pré-carga deve ser consistente com o amplificador sensor, caso contrário o mesmo poderá não funcionar adequadamente.

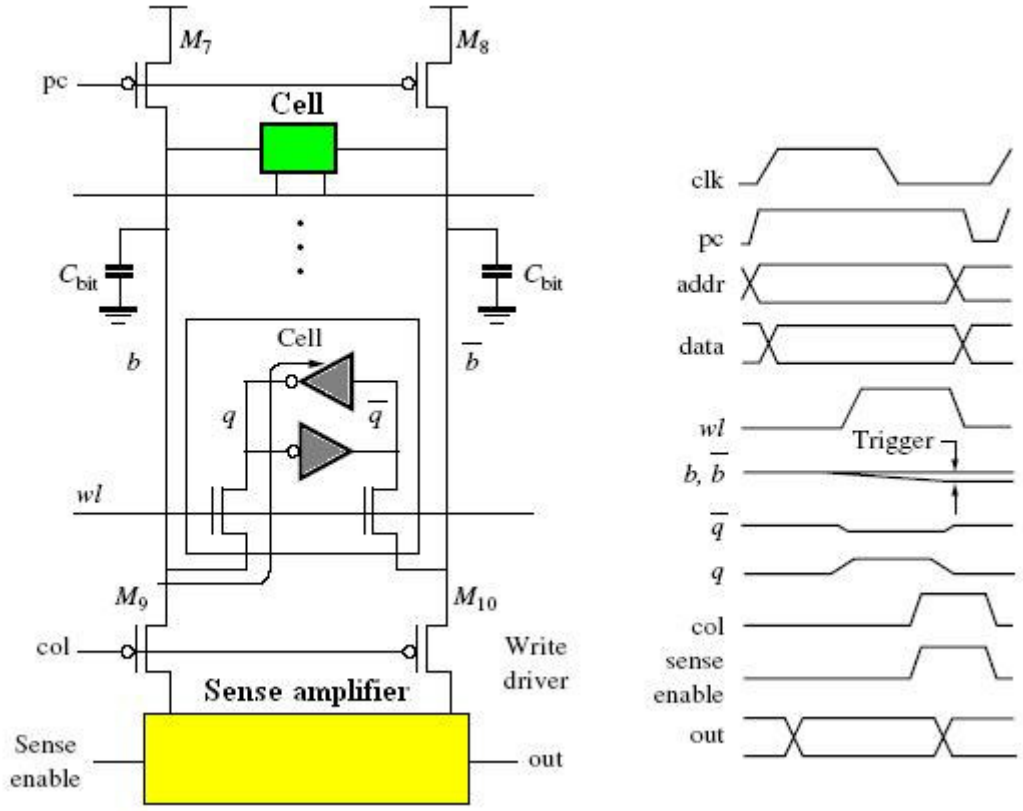


Fig. 2.18 - Circuito básico de leitura [HODGES, 2003]

Pode-se usar, neste caso, um amplificador diferencial CMOS para melhorar a imunidade ao ruído e a velocidade na leitura. Como a variação de tensão entre as linhas de bit é limitada pelas capacitâncias das mesmas, qualquer ruído pode causar uma leitura errônea. O amplificador sensor mostrado na Figura 2.19 atenua o ruído em modo comum e amplifica os sinais em modo diferencial. A operação detalhada desse circuito envolve o conceito de projeto de circuitos analógicos, mas como o seu uso nesse caso é para aplicações de grandes sinais, suas propriedades podem ser estudadas do ponto de vista digital [HODGES, 2003].

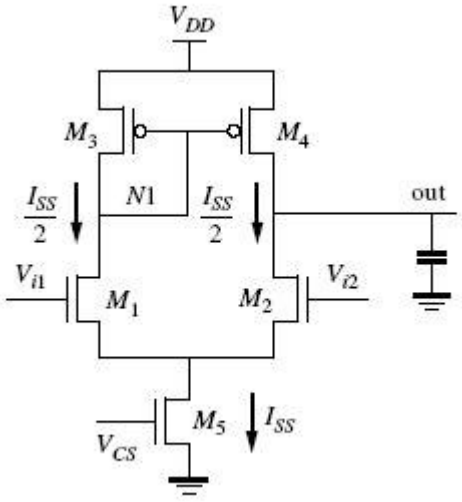


Fig. 2.19 - Amplificador sensor diferencial [HODGES, 2003]

O circuito da Figura 2.19 pode ser dividido em três componentes: o espelho de corrente, o amplificador em modo comum e a fonte de corrente de polarização. Todos os transistores estão incialmente na região de saturação, de forma que o ganho é alto. Eles também possuem grandes valores de L, de maneira a se melhorar a linearidade. Os transistores M3 e M4 atuam para prover uma mesma corrente para ambos os lados do circuito, ou seja, a corrente fluindo em M3 é espelhada em M4 (I3 ≈ I4). O transistor M5 fornece a corrente de polarização Iss, que, por sua vez, depende da tensão Vcs. No estado permanente, a corrente que flui em ambas as colunas do amplificador deve ser igual a Iss/2.

Os dois transistores de entrada M1 e M2 formam um par diferencial acoplado à fonte. As duas tensões de entrada, V_{i1} e V_{i2}, estão conectadas às colunas do amplificador. A polarização do circuito deve ser feita cuidadosamente para permitir que o nó de saída tenha uma excursão suficiente.

Após polarizado, o amplificador funciona conforme é descrito a seguir. Inicialmente, a corrente de polarização em ambas as colunas é igual e ambas as entradas são iguais a Vdd – Vtn [HODGES, 2003]. Quando a tensão em uma entrada diminui, a corrente da coluna correspondente também diminui. Ao mesmo tempo, a corrente na outra coluna aumenta para manter o valor total de Iss em M5.

Assumindo-se que a tensão na entrada em M1 diminuiu para abaixo de Vdd – Vtn na quantidade necessária para desativá-lo (Figura 2.20), a corrente em M3 e M4 será zero. Entretanto, como a corrente em M5 é Iss, segue que essa corrente deve estar descarregando

a capacitância de saída através de M2. Logo, a tensão de saída é rapidamente forçada para zero.

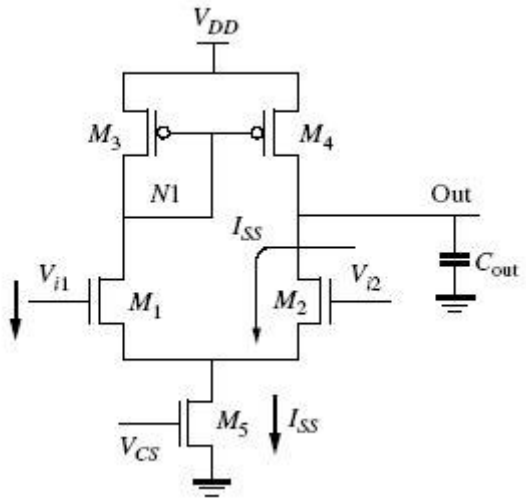


Fig. 2.20 - Descarga na saída [HODGES, 2003]

Se a tensão de entrada V_{i2} diminuir (Figura 2.21), o transistor M2 é desativado, e todas as correntes fluirão através de M1, M3 e M5. A corrente de M3 é espelhada em M4. Como a corrente em M2 é zero, Iss flui para a saída, carregando-a para Vdd.

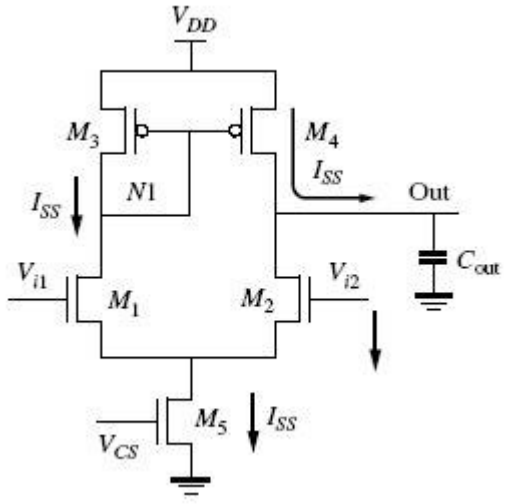


Fig. 2.21 - Carga na saída [HODGES, 2003]

Deve existir um compromisso entre velocidade e dissipação de potência, e ambas são determinadas pela corrente Iss [HODGES, 2003].

[SEEVINCK, 1990] propõe um amplificador sensor que consiste de 4 transistores

PMOS, de dimensões iguais, em uma configuração cruzada (vide Figura 2.22). Ele é selecionado aterrando-se o nó sel. Quando isso acontece, correntes fluem através dos transistores pelas cargas das linhas de bit. Os drenos de T3 e T4 estão conectados às linhas de dados DL e DLb. As cargas das linhas de bit são de baixa resistência para assegurar que, durante o acesso à leitura, as linhas de bit estão sempre próximas de Vdd.

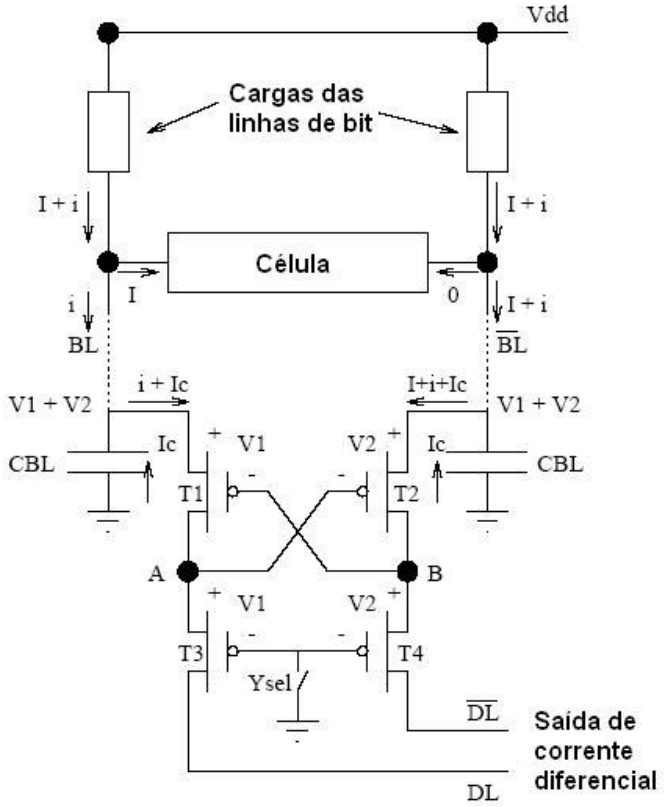


Fig. 2.22 - Amplificador sensor de corrente [MAHESHWARI, 2000]

O circuito opera da seguinte forma: supõe-se que a célula é acessada, e que a corrente I flui para dentro da célula, conforme mostra a Figura 2.22. A tensão Vgs de T1 será igual à de T3, já que suas correntes são iguais e ambos estão na saturação. Essa tensão está representada por V1. O mesmo ocorre com T2 e T4. As suas tensões Vgs são representadas por V2. Segue que, desde que Ysel esteja aterrado, as tensões de ambas as linhas serão iguais a V1+V2. Logo, os potenciais das linhas de bit serão iguais independentemente da distribuição de corrente. Isso significa que existe um curto-circuito virtual entre as mesmas. Como as tensões serão iguais, as correntes nas cargas das linhas de bit também serão iguais. Como a célula consome a corrente I, segue que passa mais corrente na perna direita do amplificador do que na esquerda. De fato, a diferença entre

essas correntes é I, a corrente consumida pela célula. As correntes nos drenos de T3 e T4 são passadas para as linhas de dados. A corrente diferencial nas linhas de dados é portanto igual à corrente na célula. Logo, obtém-se um sensor de corrente [SEEVINCK, 1990].

É necessária a inclusão de uma seção de saída para converter a corrente diferencial em tensão. Este circuito está mostrado na Figura 2.23. As linhas de dados têm uma carga comum para Vss que consiste de dois transistores n (N1 e N3). Devido ao efeito de corpo, os transistores T3 e T4 permanecerão na saturação, conforme é requerido. A corrente na saída do amplificador é espelhada através de dois espelhos de corrente de canal n (N1, N2 e N3, N4) e um espelho de corrente de canal p (P1 e P2). O inversor na saída amplifica o sinal [SEEVINCK, 1990].

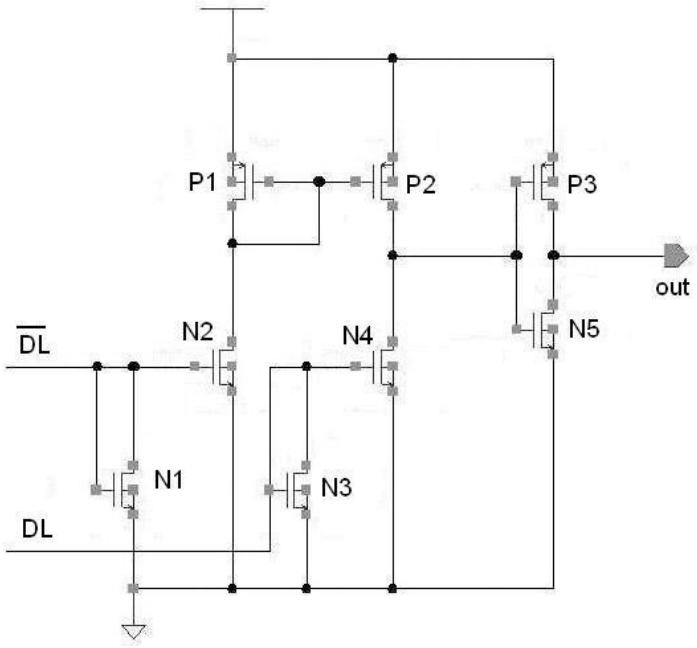


Fig. 2.23 - Estágio de saída do amplificador sensor [SEEVINCK, 1990]

2.2.2.2 Decodificador de linha

Este circuito é necessário para selecionar uma das linhas de palavra em resposta a um endereçamento de entrada. Para realizar essa função, pode-se utilizar portas NOR. A estrutura matricial da Figura 2.24 mostra um decodificador proposto por [SEDRA, 2000] para o circuito de memória da Figura 2.5. Cada linha possui um transistor p associado que é ativado antes do processo de decodificação usando o sinal de pré-carga P. Durante a pré-carga, todas as linhas de palavra são puxadas para Vdd. Supõe-se que nesse momento os bits de entrada de endereçamento ainda não foram aplicados e todas as entradas estão em

nível baixo. A operação de decodificação se inicia quando os bits de endereçamento e seus complementos são aplicados. Os transistores NMOS são posicionados de tal forma que as linhas de palavra não selecionadas sejam descarregadas. Para qualquer combinação de entrada, apenas uma linha de palavra não será descarregada e, portanto, sua tensão permanece alta.

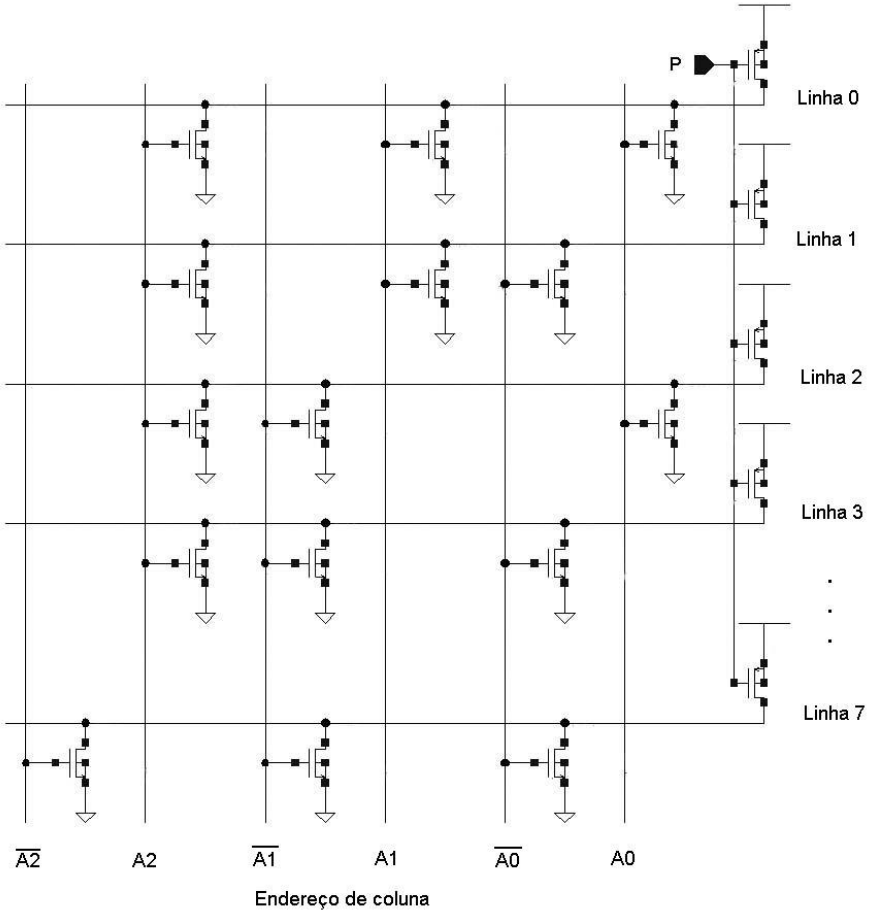


Fig. 2.24 - Decodificador NOR [SEDRA, 2000]

2.2.2.3 Decodificador de coluna

[SEDRA, 2000] propôs um decodificador para a estrutura da Figura 2.5, cuja função é conectar uma das 2^N linhas de bits à linha de dados de E/S da memória. Dessa forma, ele pode ser implementado como um multiplexador usando lógica de transistores de passagem (Figura 2.25).

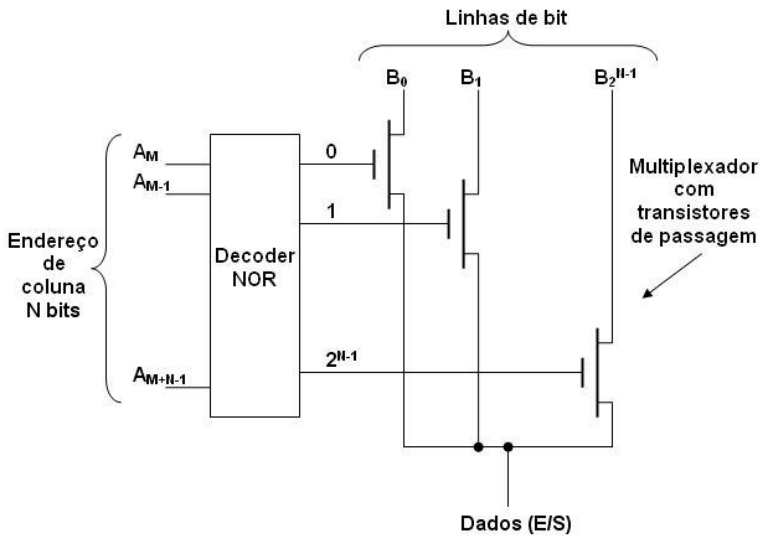


Fig. 2.25 - Decodificador de coluna [SEDRA, 2000]

Cada linha de bit é conectada à linha de dados de E/S por meio de um transistor NMOS. As portas dos transistores de passagem são controladas por 2^N linhas, uma das quais é selecionada por um decodificador NOR similar ao da Figura 2.24.

Outra estrutura alternativa, conhecida como decodificador em árvore (Figura 2.26), tem uma estrutura simples de transistores de passagem, e usa um número menor de transistores. Entretanto, como um número significativo de transistores pode estar no caminho do sinal, a resistência das linhas de bit aumenta e a velocidade diminui [SEDRA, 2000].

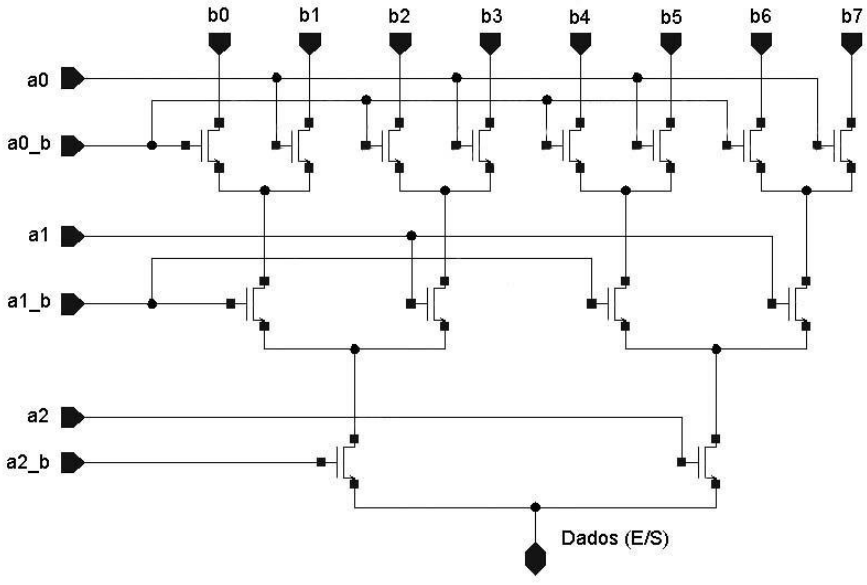


Fig. 2.26 - Decodificador em árvore [SEDRA, 2000]

2.2.2.4 Circuito de escrita

De acordo com [WESTE, 1993], o objetivo da operação de escrita da RAM é aplicar tensões na célula de tal forma que a mesma muda o seu estado, condição tal que é totalmente indesejada na operação de leitura. A Figura 2.27 mostra um circuito simples para escrita com transistores n de passagem. Neste circuito, os transistores N1 e N2 são habilitados para introduzir o dado e o seu complemento diretamente nas linhas de bit. A linha de palavra é então selecionada, e uma das linhas de bit vai para o nível lógico zero e a outra, para Vdd. A Figura 2.28 mostra uma operação de escrita de um "1" lógico quando a célula estava armazenando um zero. O nó –Cell deve ser descarregado para abaixo do valor de limar do inversor da célula RAM, enquanto que o nó Cell deve ser carregado para um valor acima do mesmo. No caso anterior, os transistores N_D, N1 e N3 têm que descarregar Pbit para abaixo da tensão de limiar. Além disso, N5 deve ser descarregado por N1 e Nd. Na outra linha de bit, P_D, N2 e N4 devem carregar Nbit com um valor tão alto quanto possível.

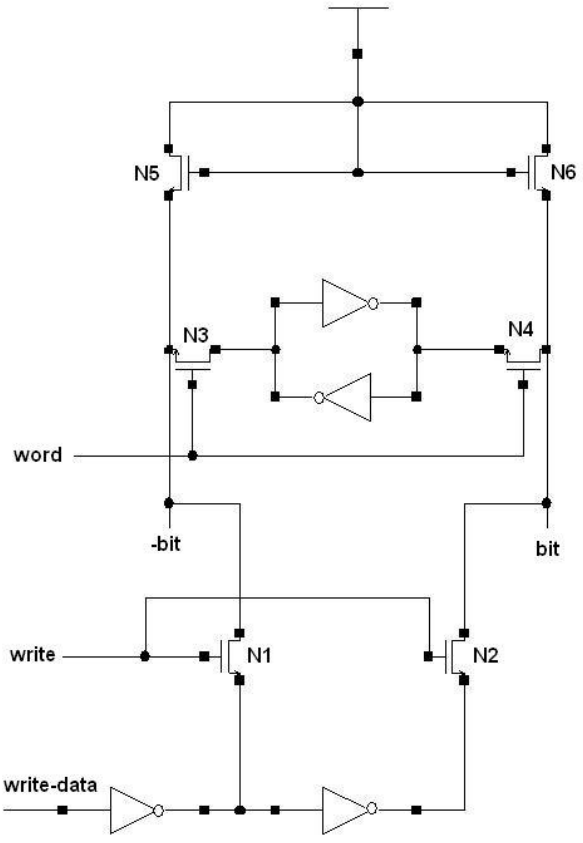


Fig. 2.27 - Circuito de escrita com transistores n [WESTE, 1993]

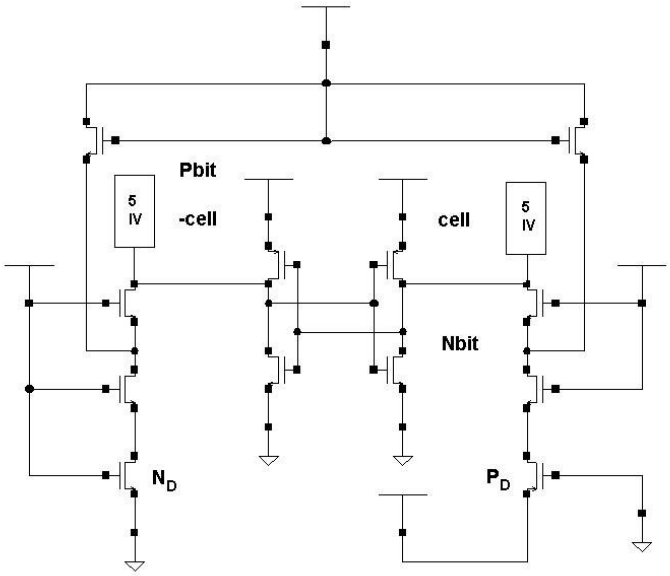


Fig. 2.28 - Operação do circuito durante a escrita [WESTE, 1993]

Para melhorar a operação de escrita, é necessário usar transistores complementares de acesso, conforme mostra a Figura 2.29. Como os transistores n não transmitem bem o "1" lógico, e os transistores p não transmitem bem o "0" lógico, ligando-se ambos em paralelo tanto o "0" quanto o "1" lógicos são transmitidos de maneira satisfatória.

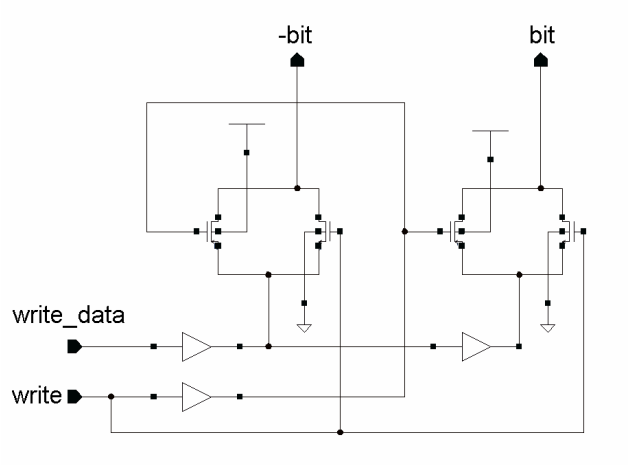


Fig. 2.29 - Circuito de escrita com portas de transmissão [WESTE, 1993]

2.3 PROJETO VOLTADO PARA A TESTABILIDADE

A testabilidade deve ser atendida através de mecanismos integrados no sistema que permitam a facilidade de acesso a pontos de teste internos, através dos quais se possa aplicar o teste e avaliar a resposta deste no próprio *chip*.

O teste exaustivo considera a geração e a aplicação de todos os possíveis padrões de entrada do circuito. Embora este tipo de teste seja trivial e assegure a maior cobertura de falhas possível, esta abordagem só é viável para circuitos combinacionais simples. Por exemplo, para testar exaustivamente um circuito combinacional com 8 entradas, seriam necessários 256 padrões de teste. Porém, para circuitos seqüenciais complexos não é indicada a aplicação deste tipo de teste, dado o grande número de estados do circuito. No teste pseudo-exaustivo, divide-se o circuito em módulos. Estes subcircuitos são exaustivamente testados, sem que seja necessário testar exaustivamente o circuito completo [COSTA, 2004a].

As técnicas utilizadas no projeto orientado à testabilidade permitem o aumento da controlabilidade e da observabilidade, com pequenos acréscimos no *hardware* e na quantidade de pinos de entrada e saída. Dentre elas, existem as técnicas *ad hoc,* que são medidas tomadas para melhorar a testabilidade durante a fase de projeto de forma não estruturada e não sistematizada. Elas são aplicadas em um dado circuito sem a preocupação de sua utilização como regra geral. Como exemplo, pode-se citar o particionamento de circuitos e a introdução de pontos de observação.

Particionamento: A técnica baseia-se na divisão do circuito a ser testado em módulos de menor complexidade, simplificando, assim, o teste dos módulos individuais e do circuito como um todo. Uma forma de realizar o isolamento entre os blocos é através do uso de portas lógicas.

Pontos de Teste: Em circuitos que apresentam baixa observabilidade e/ou controlabilidade, pode-se utilizar o artifício de adicionar pontos de acesso, tanto para entrada de dados quanto para leitura dos mesmos. A utilização de um ponto de teste como entrada primária do circuito aumenta sua controlabilidade, enquanto que a utilização deste como saída aumenta sua observabilidade. Esta técnica utilizada em conjunto com a de particionamento permite o controle e a observação dos blocos individualmente, sendo importante para a caracterização dos blocos que formam o circuito projetado [COSTA, 2004a].

2.4 TESTE EM MEMÓRIAS

O teste em memórias é cada vez mais importante por causa da grande densidade dos *chips* de memória atuais e também porque algoritmos antigos demoram muito para serem completados [GOOR, 1993]. No início, os procedimentos de teste foram desenvolvidos como técnicas *ad hoc*. A cobertura de falhas desses procedimentos de teste era

freqüentemente limitada e indeterminada. Por isso, foram introduzidos modelos de falhas, como *stuck-at*, falhas em decodificadores, falhas de acoplamento, etc. Os defeitos no *layout* da memória são modelados como falhas, e o comportamento elétrico de cada defeito é analisado e classificado [GOOR, 1993].

2.4.1 MODELOS DE FALHAS EM SRAM

Conforme foi visto no item 2.2, a arquitetura de uma RAM consiste em muitos blocos. Apesar de cada bloco desempenhar um papel individual, em certos blocos as falhas são as mesmas. De acordo com [GOOR, 1993], para o propósito de modelamento, a estrutura da memória pode ser simplificada para o modelo funcional mostrado na Figura 2.30, que inclui o decodificador de endereço, a matriz de células e a lógica de leitura/escrita.

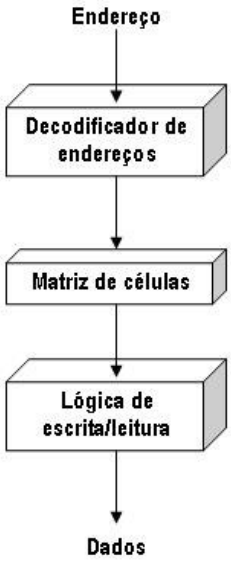


Fig. 2.30 - Modelo funcional reduzido de um chip SRAM [GOOR, 1993]

2.4.1.1 Falhas no decodificador de endereço

As falhas que podem ocorrer no decodificador são:

- Para um dado endereço, nenhuma posição é acessada;
- Uma determinada posição não está acessível;
- Para um dado endereço, múltiplas posições são acessadas;
- Uma mesma posição é acessada com múltiplos endereços.

2.4.1.2 Falhas na matriz de células

Muitas falhas diferentes podem acontecer na matriz. Elas podem ser classificadas como falhas que envolvem somente uma única célula (tais como *stuck-at*, *stuck-open*, transição e falha na retenção de dados) e falhas onde uma célula ou um grupo de células interfere no comportamento de outra célula. Esse último tipo é conhecido como falha de acoplamento (CF – *Coupling Fault*).

Na falha *stuck-at*, o valor lógico de uma célula ou linha é sempre 0 (SA0) ou sempre 1 (SA1). Uma falha *stuck-open* significa que a célula não pode ser acessada, talvez porque a linha de palavra se encontra em aberto. Na falha de transição, a célula não muda o estado durante uma transição de 0 para 1 ou de 1 para 0. A falha de retenção acontece quando a célula perde o valor armazenado com o passar do tempo. Isso pode ser causado por um dispositivo *pull-up* defeituoso dentro da célula, já que correntes de fuga podem descarregar o nó onde o mesmo se encontra, causando uma perda de informação.

Exitem vários tipos de falhas de acoplamento. Um exemplo é o de inversão, que envolve duas células *i* e *j*. Nesse caso, durante uma operação de escrita na célula *i*, ocorre uma interferência da célula *j* e o conteúdo de *i* é invertido. Deve-se notar que, para um dado par de células, o acoplamento pode ser mútuo (ou seja, a célula *j* inverte o valor da célula *i*, e vice-versa). Existem também as *idempotent coupling faults*, que também envolvem duas células *i* e *j*. Nesse caso, quando ocorre uma transição na operação de escrita em uma célula *j*, o conteúdo da célula *i* é forçado para um valor fixo (0 ou 1). Sendo assim, existem 4 tipos: na transição de 0 para 1 na célula *j*, a célula *i* é forçada para 0 (1º tipo) ou para 1 (2º tipo), e na transição de 1 para 0 na célula *j*, a célula *i* é forçada para 0 (3º tipo) ou para 1 (4º tipo). Outra falha de acoplamento é a de estado. Ela é similar à anterior, apenas não ocorre devido a uma transição na outra célula, mas devido ao estado da mesma, ou seja, se a célula *j* está em um dado estado y, a célula acoplada *i* é forçada para um certo valor x.

Finalmente, existem as *linked faults*, que afetam a mesma célula e envolvem duas ou mais falhas de acoplamento do mesmo tipo. Elas também podem ocorrer entre falhas de tipos diferentes.

Os testes podem detectar a presença de falhas funcionais, mas exigem tempo e dinheiro. Dessa forma, devem ser escolhidos apenas testes que irão detectar as falhas mais prováveis. A probabilidade de uma falha em particular depende de fatores como tecnologia utilizada, projeto do circuito, *layout*, fabricação, etc.

2.4.1.3 Falhas na lógica de escrita/leitura

As falhas nesses blocos englobam as falhas que podem ocorrer nas estruturas do circuito de escrita do dado na célula, e do circuito de leitura (amplificador sensor).

2.4.2 TIPOS DE TESTE

Ainda de acordo com [GOOR, 1993], para detectar todas as falhas funcionais em um *chip*, deve-se testar o decodificador, a matriz de células e a lógica de escrita/leitura. Os testes para falhas na matriz de células da memória detectarão as mesmas falhas na lógica de escrita/leitura, o que significa que não são necessários testes separados para essas estruturas. Além disso, as falhas podem apenas ser detectadas, mas não localizadas.

Muitos tipos de teste foram propostos. A escolha do teste geralmente está relacionada ao tempo de teste e à simplicidade dos algoritmos. Serão descritos dois tipos de teste: *march test* e BIST (*Built-In Self-Testing*).

2.4.2.1 March test

O *march test* consiste em uma seqüência de operações aplicadas em cada célula da memória, antes de passar para a célula seguinte. Uma operação consiste em escrever um "0" em uma célula (w0), escrever um "1" (w1), ler um "0" (r0) e ler um "1" (r1). Após todas as operações do march test terem sido aplicadas em uma célula, deve-se aplicá-las a outra célula. O endereço da próxima célula é determinado pela ordem, que pode ser ascendente (↑: do endereço 0 até o último) ou descendente (↓: do último endereço até o 0). O símbolo ↑↓ significa que a ordem não é importante.

Há varios tipos de *march tests,* sendo que cada um é otimizado para um conjunto particular de falhas funcionais. Os três tipos mais importantes são: MATS+, March C- e March B [GOOR, 1993].

MATS+: A equação 2.1 mostra o algoritmo MAT+. Ele requer 5.n operações (onde n é o número de endereços) e detecta todas as falhas na decodificação de endereços. Além disso, as falhas *stuck-at* também são detectadas, pois os valores 0 e 1 são lidos em todas as células.

$$\{\uparrow\downarrow(w0)\uparrow(r0,w1)\downarrow(r1,w0)\} \qquad (2.1)$$
$$M0 \quad M1 \quad M2$$

O elemento M0 executa operações de escrita de 0 (w0) em todos os endereços, em uma ordem qualquer. O elemento M1 executa operações de leitura do 0 (r0), e em seguida operações de escrita do 1 (w1), partindo do endereço 0 até o endereço n-1. Finalmente, o elemento M2 executa operações de leitura do 1 (r1) seguidas de operações de escrita do 0 (w0), partindo do endereço n-1 até o endereço 0.

March C-: A equação 2.2 mostra o algoritmo March-, que requer 10.n operações. Ele detecta as todas falhas na decodificação de endereços, e todas as falhas *stuck-at* (por exemplo, M1 detecta falhas SA1 e M2 detecta falhas SA0). É possível também detectar todas as falhas de de transição (por exemplo, M1 seguido de M2 detecta as transições de 0 para 1, e M2 seguido de M3 detecta as transições de 1 para 0). Os dois elementos seguintes detectam as falhas de acoplamento entre células que causam inversão dos valores armazenados, bem como *idempotent coupling faults*.

$$\{\uparrow\downarrow(w0);\uparrow(r0,w1);\ \uparrow(r1,w0);\downarrow(r0,w1);\downarrow(r1,w0)\uparrow\downarrow(r0)\} \qquad (2.2)$$
$$M0 \quad M1 \quad M2 \quad M3 \quad M4 \quad M5$$

O March C- detecta igualmente as falhas de acoplamento de estado, pois duas células *i* e *j* são levadas a todos os 4 estados [(0,0), (0,1), (1,0), (1,1)], e em cada estado as células são lidas. Por exemplo, o estado (1,0) ocorre após o estado (0,0) através da operação w1 na célula *i* em M1, e então a operação r0 verifica o valor da célula *j*. O estado (1,0) pode vir também após o estado (1,1) através da operação w0 na célula *j* em M4, e então a operação r1 de M4 verifica o valor da célula *j*.

March B: A equação 2.3 mostra esse algoritmo, que requer 17.n operações, e detecta todas as falhas de decodificação de endereços, *stuck-at*, de transição (inclusive em conjunto com falhas de acoplamento), falhas de acoplamento de inversão (isoladas e algumas em conjunto com *idempotent coupling faults*), e também *idempotent coupling faults* [GOOR, 1993].

$$\{\uparrow\downarrow(w0);\uparrow(r0,w1,r1,w0,w1);\ \uparrow(r1,w0,w1);\downarrow(r1,w0,w1,w0);\downarrow(r0,w1,w0)\} \qquad (2.3)$$
$$M0 \quad M1 \quad M2 \quad M3 \quad M4$$

***Testes para falhas* stuck open**: Nesse caso, a célula está inacessível porque a linha de palavra está aberta. Para detectar essa falha, o *march test* tem que verificar se um 0 e um

1 podem ser lidos em todas as células, ou seja, deve haver um elemento onde os valores 0 e 1 são lidos em uma célula, e outro elemento, possivelmente o mesmo, no qual os valores 1 e 0 são lidos na célula. O elemento M1 do March B é um exemplo onde isso ocorre.

Testes para falhas de retenção de dados: Qualquer tipo de *march test* pode ser alterado para verificar esse tipo de falha também. Essa detecção requer que uma célula de memória seja trazida para um dos seus estados lógicos. Um certo tempo deve passar (as correntes de fuga devem descarregar o nó aberto da célula da SRAM), e então o conteúdo é verificado. Esse teste deve ser repetido armazenando-se o valor lógico inverso na célula, de maneira que se possa testar a falha associada à conexão aberta no outro nó da célula. A quantidade de tempo a se esperar depende da quantidade de carga armazenada nos capacitores do nó e da corrente de fuga. Resultados empíricos mostram que o tempo de espera (*delay time*) de 100 ms é adequado para algumas células SRAM estudadas [GOOR, 1993].

March G: Trata-se de um teste extendido do March B para incluir as falhas de retenção de dados e *stuck open*. Ele consiste de 7 elementos e dois *delays*, requer um tempo de teste igual a 23.n+2.Del, e é uma boa alternativa quando se pode tolerar tempos maiores de teste poque cobre todas as falhas. A equação 2.4 mostra a equação utilizada para esse teste, e a equação 2.5 mostra uma versão alternativa na qual as duas operações extras de leitura no elemento M1 (que foram incluídas no M1 do March B para detectar falhas de transição juntamente com falhas de acoplamento) são distribuídas em M2 e M4 para deixar o teste mais simétrico.

{↑↓(w0);↑(r0,w1,r1,w0,r0,w1); ↑(r1,w0,w1);↓(r1,w0,w1,w0);↓(r0,w1,w0),Del;↑↓(r0,w1,r1);Del;
M0 M1 M2 M3 M4 M5
↑↓(r1,w0,r0)} (2.4)
M6

{↑↓(w0);↑(r0,w1,r1,w0,w1);↑(r1,w0,r0,w1);↓(r1,w0,w1,w0);↓(r0,w1,r1,w0),Del;↑↓(r0,w1,r1);
M0 M1 M2 M3 M4 M5
Del;↑↓(r1,w0,r0)} (2.5)
M6

2.4.2.2 BIST (*Built-in Self-Testing*)

Conforme é descrito em [FRANKLIN, 1990], a principal característica do BIST é que o mesmo aumenta a funcionalidade da lógica do circuito para que ele possa se testar. Esse conceito foi proposto primeiramente para circuitos combinacionais, e mais tarde para teste de estruturas regulares, como memórias RAM, ROM e PLA (*Programmable Logic Arrays*).

Os projetistas de *chips* de memória geralmente usam regras agressivas de projeto para maximizar o número de células em um *chip* e minimizar o tempo de acesso. Isso impõe certas limitações no projeto da lógica BIST. Em geral, essa lógica tenta minimizar:

- a área ocupada pelo *hardware* BIST
- a penalidade exercida na performance da operação normal da memória
- o número adicional de pinos requeridos
- a disparidade entre a velocidade de funcionamento e a do teste
- o tempo de teste.

A lógica BIST pode ser conceitualmente dividida em quatro partes: lógica de controle, lógica de geração de endereços, lógica de geração de dados e verificação da resposta, e lógica *test-trigger*. A Figura 2.31 mostra uma organização BIST genérica para teste de RAM.

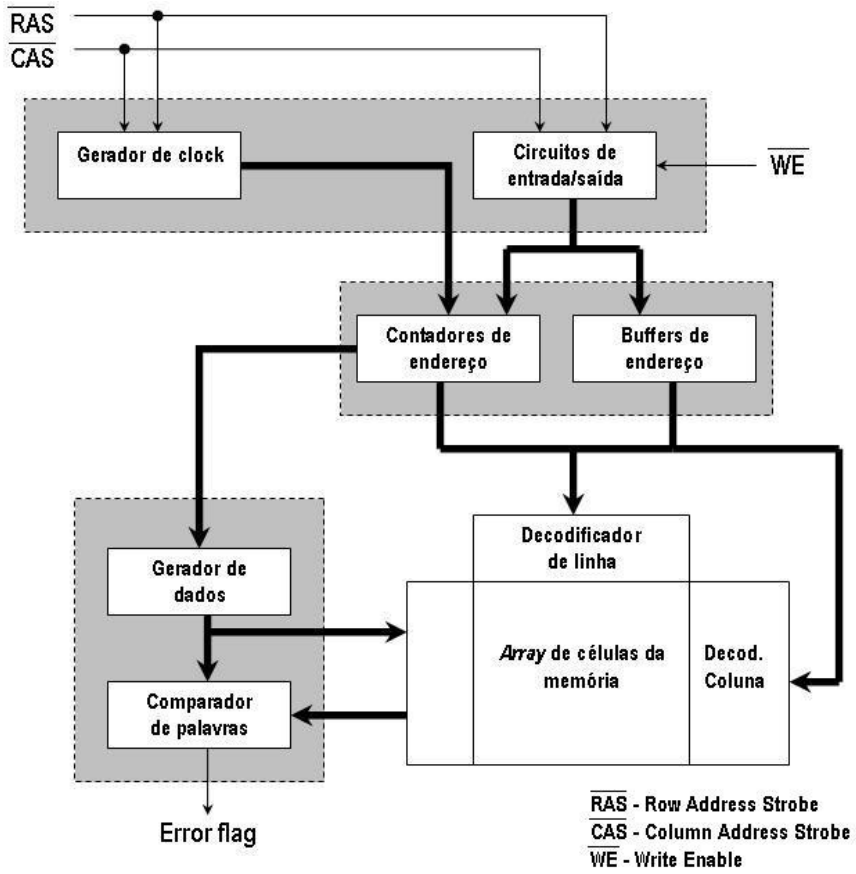


Fig. 2.31 - Diagrama de blocos da lógica BIST genérica para teste de RAMs [FRANKLIN, 1990]

Lógica de controle: A lógica de controle inicia e pára o teste, e supervisiona o fluxo de controle do algoritmo de teste. Pode ser implementada usando lógica aleatória ou *microcode*. A lógica aleatória oferece maior velocidade e tem sido tradicionalmente usada pelos projetistas [FRANKLIN, 1990].

Lógica de geração de endereços: Quase todos os algoritmos de teste requerem que os endereços sejam gerados de maneira razoavelmente uniforme. A lógica de controle também pode ser usada para gerar os endereços, mas é melhor deixar essa tarefa para uma unidade em separado. Para a maior parte dos algoritmos, a geração de endereços pode ser feita usando registradores de deslocamento com realimentação linear, registradores ou contadores, com uma intervenção ocasional da lógica de controle[FRANKLIN, 1990].

Lógica de geração de dados e verificação da resposta: A unidade de geração de dados produz padrões de teste a serem escritos nas células. Podem ser usadas diferentes estratégias para uma dada arquitetura de teste. Registradores de deslocamento com realimentação linear ou contadores podem gerar os dados. Para arquiteturas SASB (*Single*

Array Single Bit), a correção dos valores lidos pode ser feita pela comparação dos mesmos com os valores esperados, ou por análise de assinatura. Para as arquiteturas SAMB (*Single Array Multiple Bits*), MASB (*Multiple Arrays Single Bit*) e MAMB (*Multiple Arrays Multiple Bits*), outros métodos de detecção de falhas – além da comparação com os valores esperados – são comparação dos valores lidos de múltiplos bit, leituras AND ou leituras OR. Nas arquiteturas MASB e MAMB, outro método conveniente de verificação é comparar saídas de bits simetricamente localizados nos vetores de teste [FRANKLIN, 1990].

***Lógica* test-trigger**: Todas as RAM com BIST possuem um modo normal, no qual a lógica BIST está inativa, e um ou mais modos de teste nos quais a lógica BIST está ativa. Os modos de teste podem ser acessados usando sobretensões, pinos extras ou seqüências únicas de temporização com entradas como *Chip Enable*, *Write Enable*, *Row Address Strobe*, e *Column Address Strobe* [FRANKLIN, 1990].

3 METODOLOGIA DE PROJETO

3.1 INTRODUÇÃO

As etapas do projeto de um circuito integrado podem ser descritas em três domínios: comportamental, estrutural e físico. Em cada um desses domínios há um número de opções que podem ser escolhidas para resolver um determinado problema. Os níveis hierárquicos de abstração incluem:

- Nível funcional ou arquitetural
- *Register-transfer level* (RTL)
- Nível lógico
- Nível de circuito

A relação entre esses domínios e níveis pode ser observada na Figura 3.1. Neste diagrama, os três domínios são representados pelas três linhas radiais, ao longo das quais os tipos de objetos de cada domínio são enumerados. Os círculos representam níveis similares de abstração. Os níveis e objetos podem diferir um pouco, dependendo da metodologia. Pode-se observar que o circuito pode ser representado de várias maneiras, que vão desde um ponto específico (como o transistor, por exemplo) até um nível abstrato (linguagem de programação).

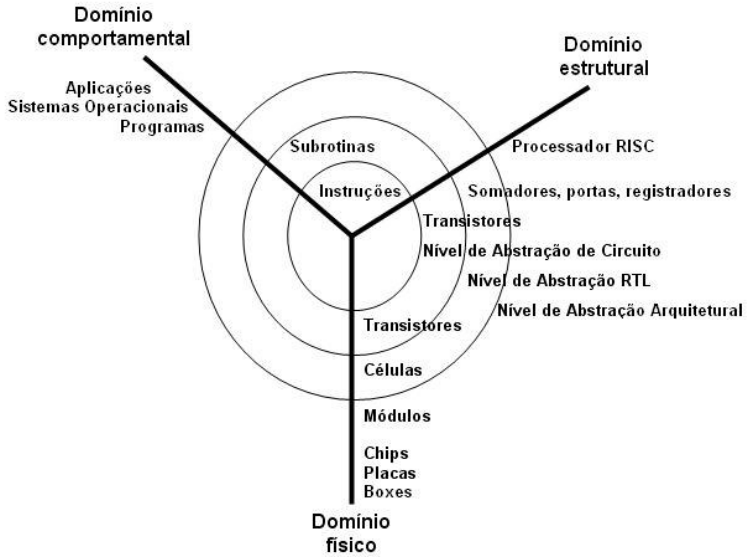


Fig. 3.1 - Diagrama de domínios e níveis de abstração [WESTE, 1993]

Dessa forma, o projetista tem várias opções na hora de iniciar um projeto. Geralmente ele começa com uma especificação funcional e avança para o nível RTL, a seguir para o nível lógico, depois para o nível estrutural e, finalmente, para o nível de *layout*. Existem várias ferramentas computacionais para auxiliá-lo nesse sentido, conforme será descrito posteriormente.

3.2 PROCESSO DE PROJETO

Devido à sua complexidade, o processo de projeto de circuitos integrados é dividido em etapas. O fluxograma da Figura 3.2 mostra as etapas do mesmo, que serão brevemente descritas a seguir.

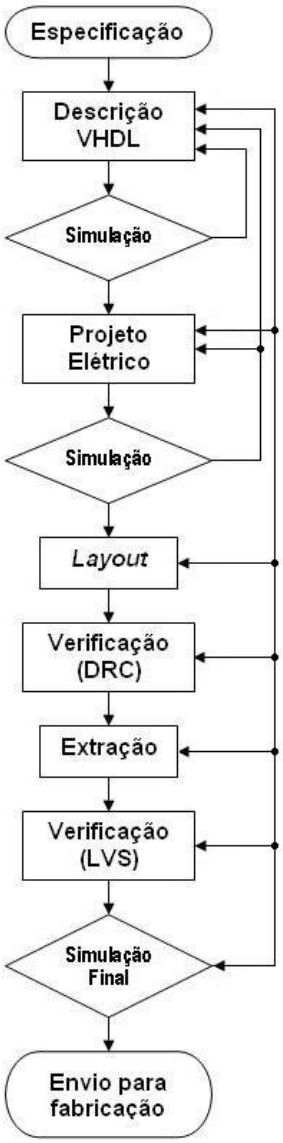


Fig. 3.2 – Metodologia de projeto

- ***Especificação***: trata-se da definição dos objetivos, funções a serem executadas pelo circuito, sinais de entrada e saída, velocidade, consumo e as condições de operação. Inclui o estudo das topologias adotadas no projeto.
- ***Projeto comportamental/funcional (descrição VHDL)***: nesta etapa, é feita a descrição funcional de um sistema de acordo com as especificações. Isso pode ser feito com a elaboração de códigos em linguagem de descrição de *hardware*, como VHDL. Esses códigos, após validados por meio de simulações, podem então ser carregados em dispositivos lógicos programáveis, como o FPGA, para verificar o comportamento do circuito em *hardware*.
- ***Projeto elétrico***: esta etapa consiste na elaboração do esquema elétrico das estruturas previamente especificadas, utilizando elementos básicos como transistores, capacitores, diodos, resistores, etc. Nesta etapa também são realizadas simulações, verificações e testes para garantir que o circuito se comportará conforme foi especificado. São definidos também vetores de teste para a validação do *chip* após a fabricação do mesmo.
- ***Layout***: nesta etapa, os elementos do circuito são desenhados de acordo com as regras definidas pela tecnologia escolhida. Após o desenho do circuito, é feita uma verificação das regras de projeto (DRC – *Design Rule Check*), para verificar se não houve erros de espaçamento, espessura, sobreposição das camadas, etc. É feita então uma avaliação do efeito das capacitâncias parasitárias no funcionamento do circuito.
- ***Fabricação***: por último, os arquivos de *layouts* são enviados para fabricação.

3.3 METODOLOGIA ADOTADA

Foi escolhida a tecnologia CMOS devido às vantagens oferecidas pela mesma, dentre elas seu baixo custo e baixo consumo de potência. Em particular, foi utilizada a tecnologia 0.35 μm CMOS da AMS.

Foi utilizado o estilo *full custom* (aproveitando elementos de células digitais básicas fornecidas pela *foundry*), isto é, o projeto é totalmente dedicado e foi feito desde a especificação até as máscaras. A passagem de um nível para o outro só se deu após a completa validação do nível anterior. Apesar de mais demorado, esse estilo oferece vantagens como menor área, já que os *layouts* são feitos manualmente, e treinamento de projetistas.

O projeto é hierárquico, e foi adotado um misto das metodologias *top-down* e *bottom-up*, já que passou-se de um nível mais abstrato (descrição VHDL) para o projeto físico (*layout*), e a realização dos circuitos e de seus *layouts* partiu de células mais simples para células mais complexas. Foram utilizadas também técnicas de projeto orientado à testabilidade (DFT – *Design For Testability*).

Primeiramente, foi feita uma especificação das estruturas de armazenamento digital necessárias ao projeto. Essa especificação foi feita em um trabalho anterior [BENÍCIO, 2002], e posteriormente alterada após o estudo das topologias de circuito para realização das memórias. O passo seguinte foi a descrição funcional do circuito em VHDL (VHSIC *Hardware Description Language*), seguida da simulação do código elaborado, para verificar o correto comportamento das estruturas previstas juntamente com o microprocessador completo. Uma descrição completa dessa etapa pode ser encontrada em [COSTA, 2004a]. Após essa validação, foi realizado o projeto elétrico das estruturas especificadas. Tendo em vista que o projeto lógico da memória só poderia ser realizado utilizando-se *flip-flops*, e que essas estruturas ocupam muito espaço no *chip*, foi decidida a realização do projeto elétrico após a descrição VHDL, utilizando-se uma estrutura menor de célula de armazenamento. Em seguida, os circuitos foram simulados e os *layouts* foram desenhados manualmente. Foi feita, então, uma verificação das regras de projeto (DRC), de acordo com a tecnologia escolhida. As etapas seguintes consistiram da extração dos circuitos e da comparação das *netlists* obtidas na extração com as obtidas a partir dos esquemáticos (LVS). Finalmente, foram feitas simulações finais a partir dos circuitos extraídos, para avaliar se as capacitâncias parasitas devido à geometria dos *layouts* não afetaram o correto funcionamento das estruturas projetadas, e as mesmas foram enviadas para fabricação.

Para a descrição VHDL, foi utilizado o *software* Foundation 4.1i, da Xilinx. Essa ferramenta permite a elaboração, síntese, simulação, implementação e programação de códigos VHDL em dispositivos FPGA. A placa disponível para teste foi a XCV600-240. A memória e os registradores foram simulados e implementados em FPGA juntamente com todas as demais estruturas do processador. O projeto elétrico foi feito e simulado utilizando-se respectivamente as ferramentas Composer e SpectreS, disponível no pacote CADENCE, e os *layouts* foram realizados na ferramenta Virtuoso do mesmo pacote.

4 DESCRIÇÃO DO PROCESSADOR RISC-16

4.1 INTRODUÇÃO

Este capítulo se propõe a descrever, de forma geral, o processador para o qual as memórias e o banco de registradores foram projetados. São abordadas, de forma resumida, a sua arquitetura, as especificações gerais, o conjunto de instruções e demais estruturas. Esse processador foi inicialmente proposto por [BENÍCIO, 2002], mas foram feitas algumas alterações recentemente propostas no seu caminho de dados em [COSTA, 2004a], de forma que somente o processador descrito neste último trabalho será abordado a seguir.

4.2 ESPECIFICAÇÃO

O processador possui arquitetura RISC de 16 bits. A tensão de alimentação foi definida em 3,3 V, e o *clock*, em 10 MHz. O processador conta também com uma unidade lógica e aritmética (ULA) de 16 bits, operando em ponto fixo, e uma unidade de memória SRAM de 8 KBytes. Essa arquitetura utiliza três operandos em registradores de endereçamento, e para isso é requerido um banco de registradores. Foi definido um banco com 16 unidades. A tecnologia a ser utilizada é a CMOS 0,35 um [COSTA, 2004a].

4.3 INSTRUÇÕES

O conjunto de instruções proposto pode ser dividido em três tipos:

R – operação entre três registradores

I – operação entre dois registradores, envolvendo uma constante de 4 bits

J – operação entre um registrador e uma constante de 8 bits.

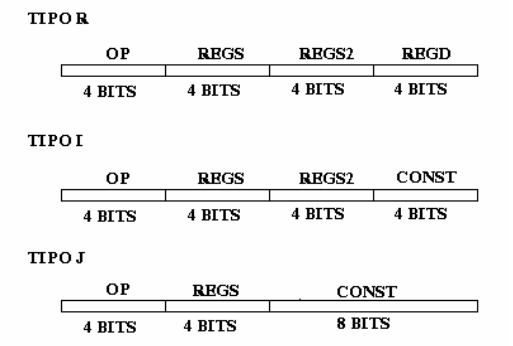


Fig. 4.1 – Tipos de instruções [COSTA, 2004a]

Os campos das instruções mostrados na Figura 4.1 estão especificados abaixo:

Operação (OP): codifica a instrução

Registrador fonte (REGS/REGS2): codificam os registradores fonte

Registrador destino (REGD): codifica o registrador destino

Constante (CONST): identifica um valor constante de 4 ou 8 bits

O conjunto de instruções envolve as seguintes classes: lógica/aritmética, transferência e desvio. Foram selecionadas as 16 instruções consideradas mais significativas em termos de desempenho para o sistema. As demais foram deixadas como pseudoinstruções. A Tabela 4.1 mostra o conjunto de instruções definido para o primeiro protótipo.

Tabela 4.1 – Instruções [COSTA, 2004a]

COD.	CAT.	INST.	EXEMPLO	SIGNIFICADO	TIPO
0010	Aritmética	Add	Add $s1,$s2,$s3	Adiciona $s2 a $s3 e armazena em $s1	R
0011		Sub	Sub $s1,$s2,$s3	Subtrai $s3 a $s2 e armazena em $s1	R
1000		Addi	Addi $s1,100	Adiciona $s1 a constante e armazena em $s1	J
1001		Shift	Sft $s1,8	Desloca $s1 da constante e armazena em $s1. Se o valor da constante for negativo, desloca à esquerda.	J
0100	Lógica	And	And $s1,$s2,$s3	AND *booleano* bit a bit entre $s2 e $s3 e armazena em $s1	R
0101		Or	Or $s1,$s2,$s3	Or *booleano* bit a bit entre $s2 e $s3 e armazena em $s1	R
1010		Not	Not $s1	NOT *booleano* bit a bit de $s1 e armazena em $s1	J
0110		Xor	Xor $s1,$s2,$s3	XOR *booleano* bit a bit de $s2 e $s3 e armazena em $s1	R
0111		Slt	Slt $s1,$s2,$s3	Torna $s1 = 1 se $s2 < $s3 senão $s1 = 0	R
0000	Transferência	Lw	Lw $s1,$s2,$s3	Carrega palavra armazenada no endereço $s2 deslocado de $s3 e salva em $s1	R
0001		Sw	Sw $s1,$s2,$s3	Carrega palavra armazenada em $s1 e salva no endereço $s2 deslocado de $s3	R
1011		Lui	Lui $s1,100	Carrega a constante nos oito bits mais significativos de $s1	J
1100	Desvio Condic.	Beq	Beq $s1,$s2,5	Se $s1 = $s2 desvia programa para $pc + CONST	I
1101		Blt	Blt $s1,$s2,5	Se $s1 < $s2 desvia programa para $pc + CONST	I
1110	Desvio Incondic.	J	J $s1,100	Desvia para o endereço $s1 deslocado da constante	J
1111		Jal	Jal $s1,100	Desvia para o endereço $s1 deslocado da constante salvando origem em $ra	J

4.4 ULA

A ULA é o dispositivo que realiza operações aritméticas e lógicas no processador. Ela possui duas portas de entrada para os operandos de 16 bits, uma porta de saída para o resultado da operação e cinco terminais de controle para selecionar a função a ser executada. O somador é do tipo *Carry Lookahead*. As operações que podem ser realizadas são soma, subtração, AND, OR, XOR, NOT, LUI (carregamento dos 8 bits mais significativos), deslocamentos à direita e à esquerda e comparação entre as duas entradas [COSTA, 2004a]. Devido à simplicidade do sistema, a ULA opera em ponto fixo. De uma maneira geral, ela pode ser representada através do diagrama da Figura 4.2.

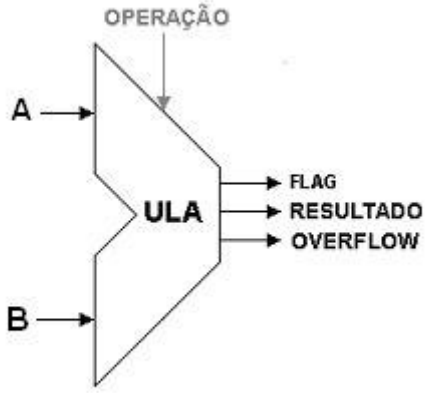


Fig. 4.2 – ULA [BENÍCIO, 2002 - alterada]

Para facilitar a programação e permitir uma redução no comprimento dos programas, são também disponibilizadas mais quatro saídas nessa unidade: Z, N, C e Caux. As funções são descritas abaixo [COSTA, 2004a]:

- Z → Indica que o resultado da operação executada pela ULA é nulo.
- N→ Indica que o resultado da operação executada pela ULA é negativo.
- C→ Indica que o resultado da operação executada pela ULA gerou *carry*.
- Caux → Indica que houve carry no meio da palavra. (houve carry no primeiro byte do resultado da operação da ULA).

A posição de memória RAM FFF1 foi reservada para armazenar esses valores, que são atualizados no final das instruções lógico-aritméticas.

4.5 MEMÓRIAS

Para atender às especificações definidas inicialmente e, ao mesmo tempo, aos requisitos do projeto, tais como baixo consumo de potência, área relativamente pequena e desempenho dinâmico adequado, foi definida a utilização de uma memória estática de leitura/escrita (SRAM), que armazenará as instruções e dados necessários para o funcionamento do sistema. A capacidade de armazenamento foi especificada através das instruções básicas do processador e das funções mínimas de processamento necessárias para um funcionamento adequado do transceptor. Sendo assim, escolheu-se uma SRAM de 8kBytes, com endereçamento de palavras de 16 bits, na qual serão endereçados 12 registradores para as interfaces do sistema. Será utilizada uma memória ROM para armazenamento das rotinas de teste do sistema em *chip* [COSTA, 2004a]. Neste protótipo, decidiu-se que uma ROM de 2 kBytes seria suficiente para armazenar tais rotinas. Sendo assim, serão necessários 13 bits de endereçamento para acessar ambas as memórias. A Figura 4.3 mostra o diagrama com a distribuição das memórias.

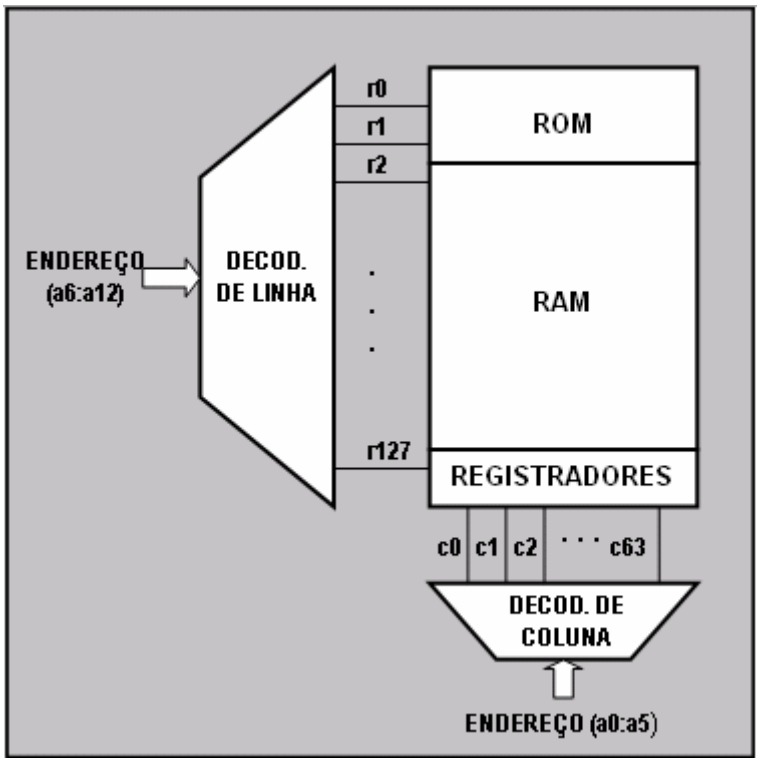


Fig. 4.3 – Memórias

4.6 REGISTRADORES

Conforme descrito no item 4.2, o banco possui 16 registradores de 16 bits. A Figura 4.4 mostra o banco de registradores utilizado no caminho de dados do processador. Os 16 registradores de 16 bits estão descritos na Tabela 4.2.[COSTA, 2004a].

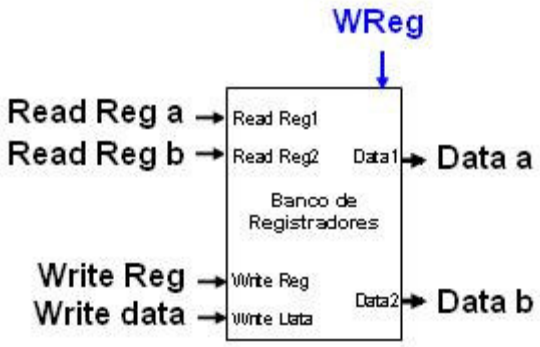


Fig. 4.4 – Banco de registradores [COSTA, 2004a - alterada]

Tabela 4.2 – Registradores do banco [COSTA, 2004a]

CÓDIGO	SÍMBOLO	FUNÇÃO	OBS
0000	$zero	Constante zero	Constante 0 de 16bits
0001	$t0	Temporários	Registradores Auxiliares
0010	$t1		
0011	$t2		
0100	$a0	Argumento	Argumentos para operações aritméticas e procedimentos
0101	$a1		
0110	$a2		
0111	$s0	Salvos	Armazena valores durante chamadas de procedimento
1000	$s1		
1001	$s2		
1010	$s3		
1011	$s4		
1100	$gp	Apontador Global	Aponta para as variáveis globais no interior da pilha
1101	$sp	Apontador Pilha	Aponta para o topo da pilha
1110	$pc	Contador de Programa	Aponta para a próxima instrução
1111	$ra	Endereço de Retorno	Aponta para o endereço de retorno de uma subrotina

4.6.1 REGISTRADORES MAPEADOS EM MEMÓRIA

Algumas posições de memória foram reservadas para o uso de registradores. A posição FFF2 é chamada de RegInt e armazena o endereço da instrução que estava sendo executada quando a interrupção/exceção foi requisitada, enquanto que a posição FFF3 é

chamada de IntCausa e guarda seu tipo. A Figura 4.5 apresenta seus formatos, e a Tabela 4.3 apresenta a descrição dos campos [COSTA, 2004a].

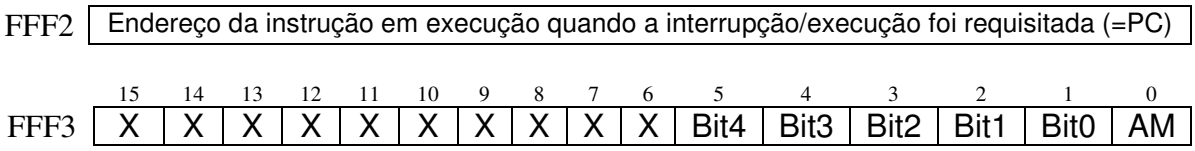


Fig. 4.5 - Formatos dos registradores RegInt e Int Causa [COSTA, 2004a]

Tabela 4.3 - Campos de FFF3 [COSTA, 2004a]

Campo	Significado
AM (AcInt_mem)	Indica que a interrupção começou a ser tratada
Bit0	Interrupção requisitada pelo Conversor A/D
Bit1	Interrupção requisitada pela Interface RF
Bit2	Interrupção requisitada pela Interface Serial
Bit3	Exceção causada pela ocorrência de um *overflow*
Bit4	Exceção causada pela ocorrência de erro de endereçamento de memória

O registrador RegStatus armazena as *flags* do sistema. Esse registrador é atualizado ao final das instruções lógico-aritméticas e foi mapeado na posição FFF1 de memória RAM. A Figura 4.6 apresenta seu formato, e a Tabela 4.4 mostra os campos utilizados.

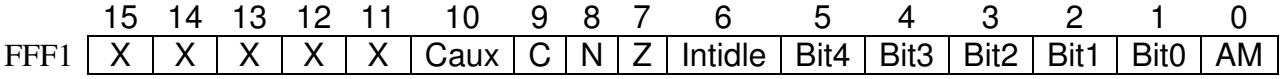


Fig. 4.6 - Formato do registrador RegStatus [COSTA, 2004a]

Tabela 4.4 - Campos de FFF1 [COSTA, 2004a]

Campo	Significado
AM (AcInt_mem)	Indica que a interrupção começou a ser tratada
Bit0	Interrupção requisitada pelo Conversor A/D
Bit1	Interrupção requisitada pela Interface RF
Bit2	Interrupção requisitada pela Interface Serial
Bit3	Exceção causada pela ocorrência de um *overflow*
Bit4	Exceção causada pela ocorrência de erro de endereçamento de memória
IntIdle	Indica que o sistema está pronto para receber novas interrupções
Z	Indica que o resultado da operação executada pela ULA é nulo
N	Indica que o resultado da operação executada pela ULA é negativo
C	Indica que o resultado da operação executada pela ULA gerou *carry*
Caux	Indica que houve *carry* no meio da palavra. (houve carry no primeiro byte do resultado da operação da ULA)

4.7 BARRAMENTO DE DADOS E CONTROLE

A Figura 4.7 apresenta o esquema do *datapath* do processador, juntamente com suas linhas de controle e sistema de controle de interrupções [COSTA, 2004a].

Os primeiros passos a serem executados são comuns a todas as instruções, e consistem em:

- Inicialização: neste estado, as interrupções encontram-se desabilitadas para que o programador possa fazer o *set up* das interfaces. O processador só executa esta etapa uma única vez, quando o sistema é iniciado.
- Pré-carga: este estado é necessário para pré-carregar as linhas de bit da memória RAM antes de uma operação de leitura.
- O valor armazenado no *Program Counter* (PC), registrador interno que guarda o endereço da próxima instrução, é passado para a memória (Endereço), onde estão armazenados os códigos do programa a ser executado. O código correspondente à posição lida é então apresentado no barramento de saída da memória (*Read Data*). Neste mesmo passo, o PC é incrementado de uma unidade, e o novo valor é salvo também no banco de registradores (registrador $pc).
- É feita a leitura de dados a partir de um ou dois registradores do banco, dependendo de como os campos da instrução lida na memória estão preenchidos. São feitos também o cálculo do *offset* e a decodificação da instrução.

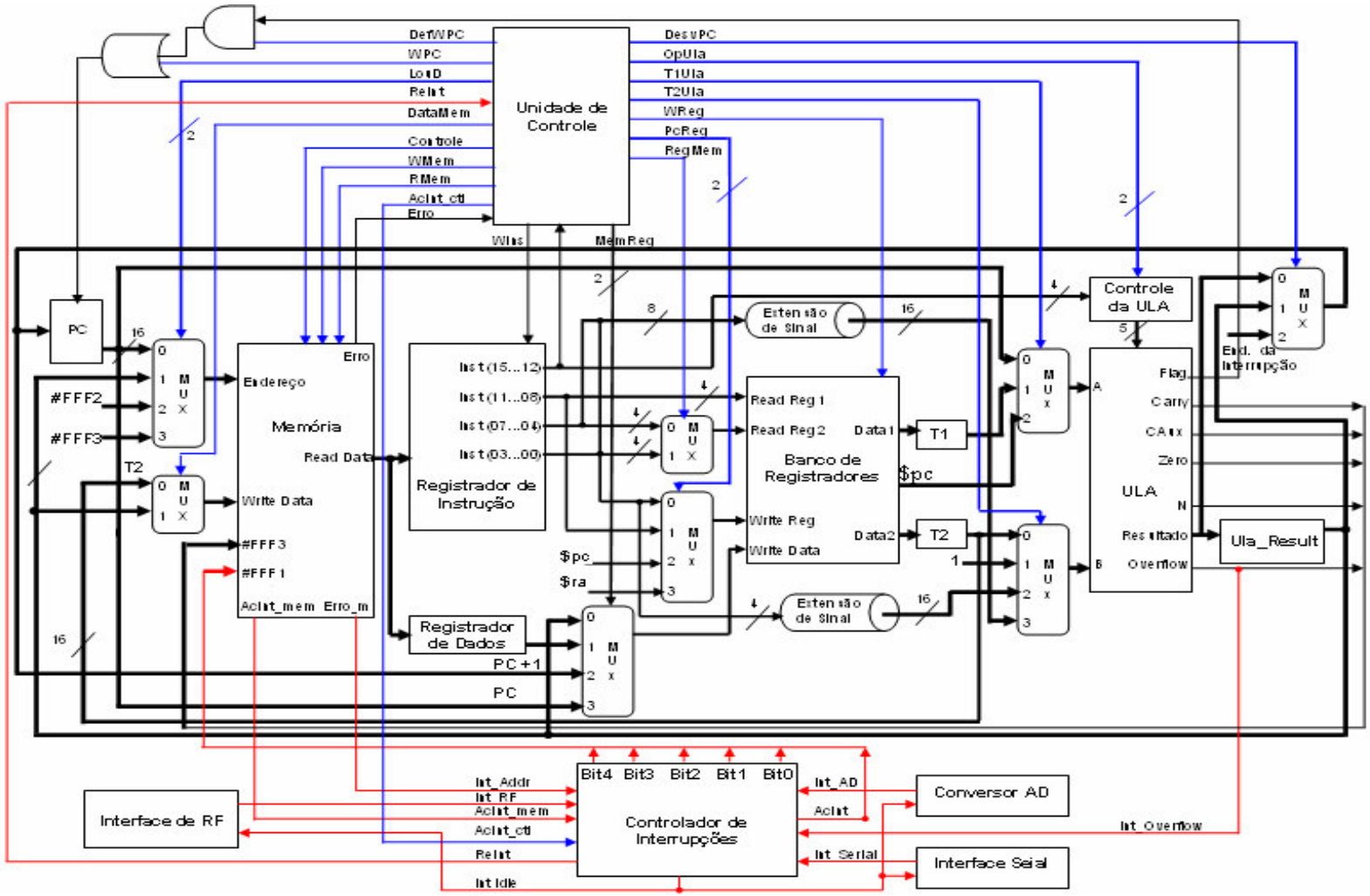


Fig. 4.7 – Caminho de dados, linhas de controle e interrupção [COSTA, 2004a]

As ações seguintes variam de acordo com a classe de instrução, conforme pode ser observado na máquina de estados finitos (Figura 4.8) que compõe a unidade de controle do processador. Sendo assim, para completar a execução das instruções, cada classe atua de modo distinto. As instruções de referência à memória farão uma operação de escrita ou leitura do dado. As instruções lógicas/aritméticas escreverão o dado calculado pela ULA em um registrador. Por fim, as instruções de desvio condicional poderão, em função de uma comparação feita pela ULA, modificar o endereço da próxima instrução a ser executada.

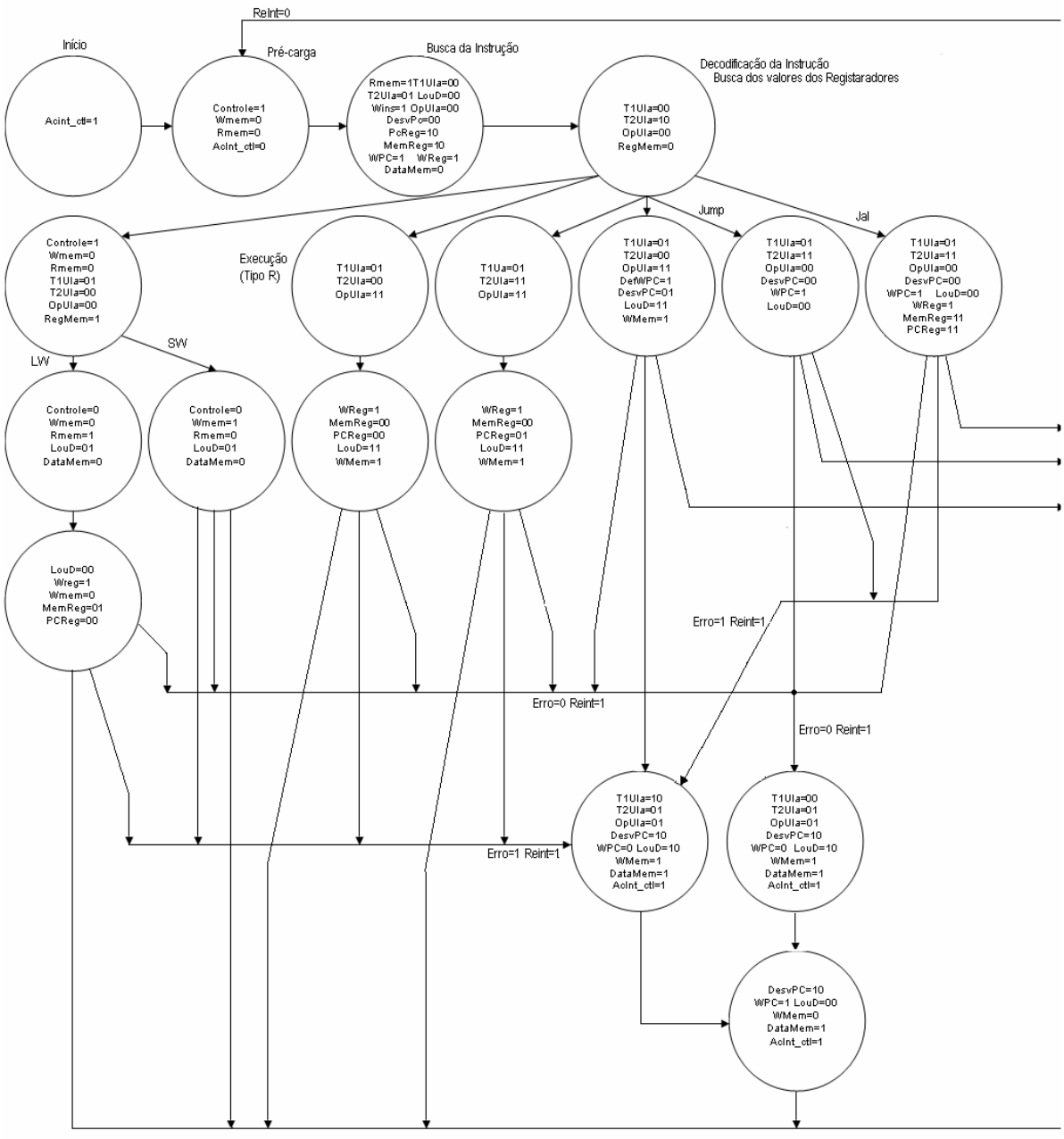


Fig. 4.8 – Máquina de estados finitos da unidade de controle [COSTA, 2004a]

4.8 – INTERRUPÇÃO

O sistema de processamento comporta três tipos de interrupção para interfaces de comunicação, e duas sinalizações de erro, conforme é mostrado na Figura 4.7 e descrito abaixo:

- recepção de dado pela unidade de RF
- recepção de dado pela porta serial
- recepção de dado pela interface A/D
- erro de *overflow* (ULA)
- erro de endereçamento (memória)

Quando ocorre uma das interrupções ou um dos erros acima, o processador executa os seguintes passos: verificação da ocorrência de interrupção ao final de cada ciclo de instrução, armazenamento do endereço da instrução em execução e do tipo de interrupção gerada nos registradores RegInt e IntCausa, que estão endereçados nas posições $FFF2_H$ e $FFF3_H$, respectivamente, e por fim, a transferência da execução do programa para um endereço de memória predeterminado, onde uma instrução de desvio encaminhará a execução para a rotina de tratamento da interrupção [COSTA, 2004a].

4.9 - ESTRUTURA DE I/O

Foram reservados 12 registradores de 16 bits para comunicação com as interfaces da unidade de RF, comunicação serial e conversor A/D. Eles são mapeados através de endereços da memória SRAM. As posições da memória utilizadas para este fim estão descritas na Tabela 4.5.

Tabela 4.5 – Endereços de I/O [COSTA, 2004a]

Conteúdo	Unid. de RF	Unid. de Com Serial	Interface A/D
Setup	#FFFFh #FFFEh	#FFFAh	#FFECh
Status	#FFFDh	#FFEFh	#FFEBh
Dados(Transmissão)	#FFFCh	#FFEEh	-
Dados(Recepção)	#FFFBh	#FFEDh	#FFEAh

Um sinal de interrupção é gerado na recepção de dados, ou seja, quando o processador recebe dados de uma das interfaces citadas. Durante o tratamento desses dados, o processador bloqueia a recepção de novos pedidos. Após o tratamento, as interrupções são novamente ativadas, de maneira que o processador possa receber novos dados.

Os registradores de *Setup*, cujo formato está representado na Figura 4.9, mostram a disposição dos parâmetros para o controle da transmissão e recepção de dados.

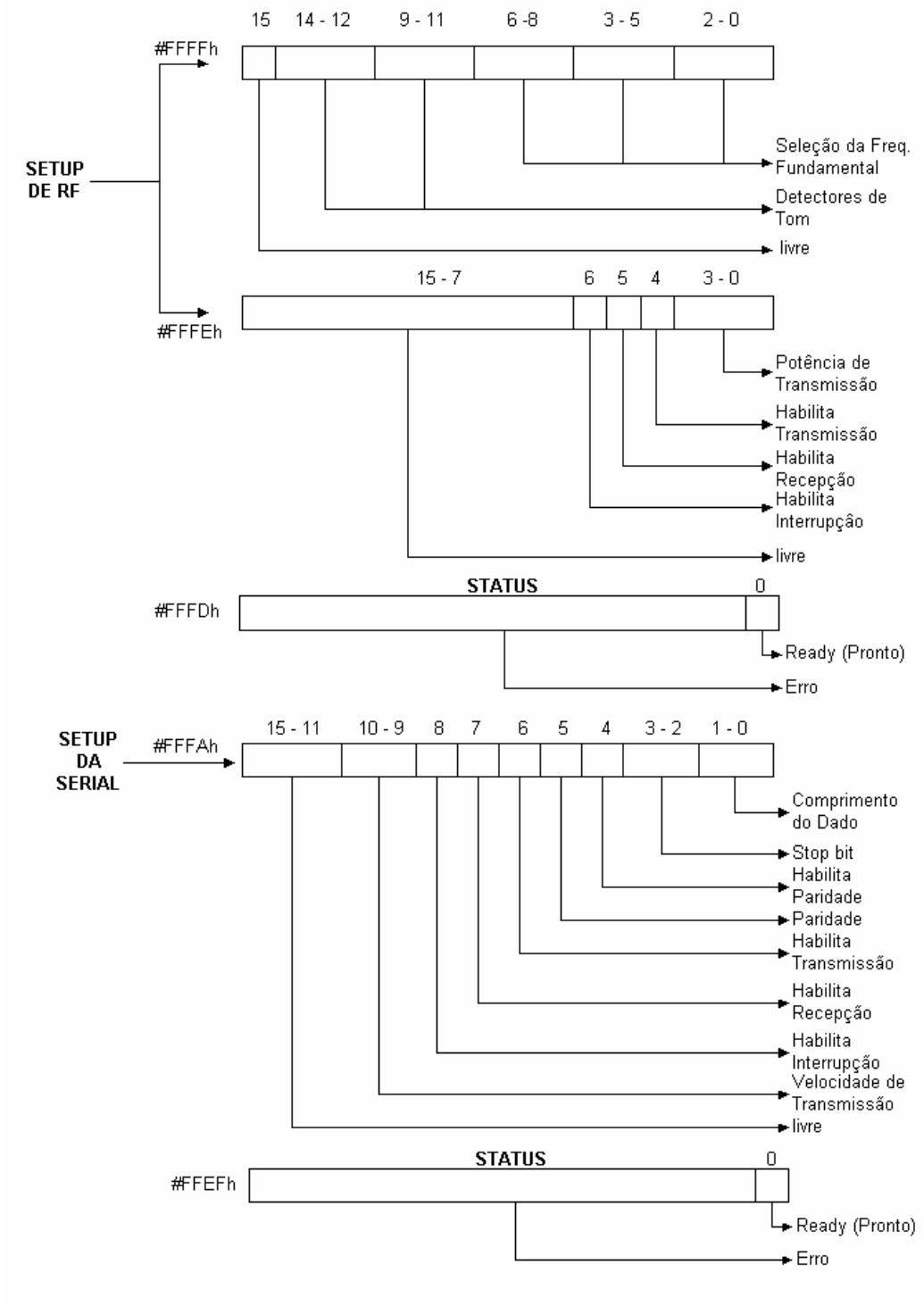


Fig. 4.9 – Registradores de I/O [COSTA, 2004a]

5 PROJETO DA ROM (*READ ONLY MEMORY*)

5.1 ESPECIFICAÇÃO DA ROM

Foi definida para o uso do processador uma memória ROM de 2 kBytes, na qual serão armazenadas as rotinas de inicialização do sistema, que deverão tratar do gerenciamento da estrutura de *hardware* de forma a habilitá-lo de acordo com a comunicação serial. Uma proposta para a rotina de inicialização encontra-se no Apêndice D.

Dentre as opções de memórias ROM descritas no Capítulo 2, escolheu-se uma ROM MOS, cuja arquitetura está mostrada na Figura 2.2, devido à sua simplicidade e à tecnologia adotada na fabricação, visto que PROM requerem métodos mais sofisticados que não estão disponíveis para a realização desse projeto.

Foi realizada para fins de teste uma memória ROM de 256 bits. Como as palavras são de 16 bits, a mesma possui 16 linhas e 4 bits de endereço (2^4x16 bits = 256 bits). Foram adicionadas também algumas estruturas de teste para verificar o correto funcionamento das partes do circuito.

5.2 PROJETO ELÉTRICO DA ROM

Foi escolhido, a título de exemplo, o programa descrito abaixo (desenvolvido na linguagem *assembly* criada para o RISC-16) para ser armazenado na ROM projetada. A Tabela 5.1 mostra a decodificação do código do programa em binário. As duas últimas linhas foram preenchidas com 0000_H e 1111_H, respectivamente.

```
2700        add $s0, $zero, $zero
2800        add $s1, $zero, $zero
E803        j $s1, 3
2100        add $t0, $zero, $zero
8101        addi $s0, 1
2800        add $s1, $zero, $zero
2900        add $s2, $zero, $zero
891F        addi $s2, 31
28B0        add $s1, $int, $zero
4889        and $s1, $s1, $s2
C814        beq $s1, $t0, 4
EF00        j $ra, 0
2800        add $s1, $zero, $zero
E800        j $s1, 0
end
```

Tabela 5.1 - Programa armazenado na ROM

Endereço	Instrução	Código binário
0000	2700	0010 0111 0000 0000
0001	2800	0010 1000 0000 0000
0010	E803	1110 1000 0000 0011
0011	2100	0010 0001 0000 0000
0100	8101	1000 0001 0000 0001
0101	2800	0010 1000 0000 0000
0110	2900	0010 1001 0000 0000
0111	891F	1000 1001 0001 1111
1000	28B0	0010 1000 1011 0000
1001	4889	0100 1000 1000 1001
1010	C814	1100 1000 0001 0100
1011	EF00	1110 1111 0000 0000
1100	2800	0010 1000 0000 0000
1101	E800	1110 1000 0000 0000
1110	0000	0000 0000 0000 0000
1111	1111	1111 1111 1111 1111

A matriz final está representada na Figura 5.1. Os transistores utilizados na matriz possuem as dimensões mínimas: $(W/L)_n$ = 0.5um/0.4um e $(W/L)_p$ = 1.2um/0.4um. Essas dimensões foram escolhidas visando à menor ocupação possível de área. Foram adicionados dois inversores em cada saída para melhorar a qualidade do sinal. As dimensões dos transistores dos inversores também são as mínimas. Poderia ter sido mantido apenas um inversor por saída, mas como dessa forma a saída ficaria invertida, preferiu-se manter a convenção de lógica usual explicada no Capítulo 2, ou seja, a célula armazena um “0” lógico quando há um transistor na mesma, e um “1” lógico quando não há um transistor na mesma.

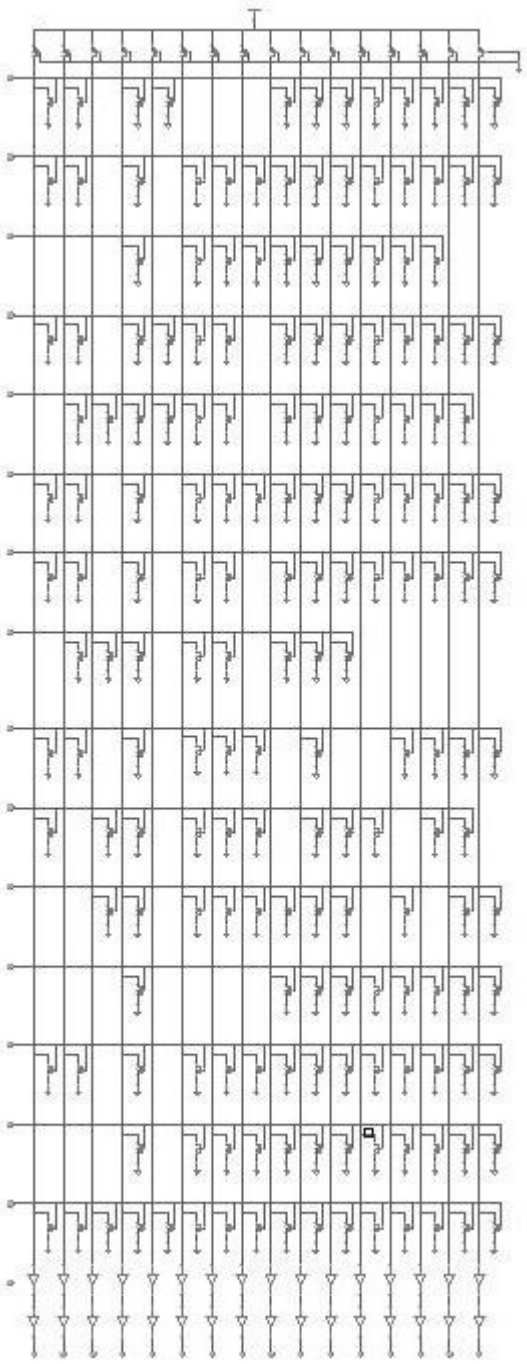


Fig. 5.1 – Matriz da ROM com 16 palavras de 16 bits

Nesse caso, como a memória só possui 256 bits, não há necessidade de se utilizar uma decodificação bi-dimensional incluindo-se um decodificador de coluna. Só há necessidade de um decodificador de linha, com 4 bits de entrada (a0, a1, a2 e a3), que correspondem aos bits de endereço da memória, e 16 bits de saída (s0 a s15). A função do decodificador é selecionar uma das 16 palavras, de acordo com o endereço dado na entrada. O decodificador escolhido foi feito utilizando-se portas NOR, com uma estrutura

similar à que foi explicada no Capítulo 2, item 2.2.2.2. A diferença é que o decodificador NOR da Figura 2.24 utiliza um sinal de pré-carga P, visto que essa estrutura estava prevista para a memória RAM. Na memória ROM, as portas dos transistores p foram aterradas, de forma que eles estão sempre conduzindo (vide estrutura da Figura 2.2). A palavra W0 será selecionada quando o endereço for 0000, a palavra W1, quando o endereço for 0001, e assim por diante. Logo,

$$W0 = /a3./a2./a1./a0 = /(a3+a2+a1+a0)$$
$$W1 = /a3./a2./a1.a0 = /(a3+a2+a1+/a0)$$
$$W2 = /a3./a2.a1./a0 = /(a3+a2+/a1+a0)$$
$$W3 = /a3./a2.a1.a0 = /(a3+a2+/a1+/a0)$$
$$...$$
$$W15 = a3.a2.a1.a0 = /(/a3+/a2+/a1+/a0)$$

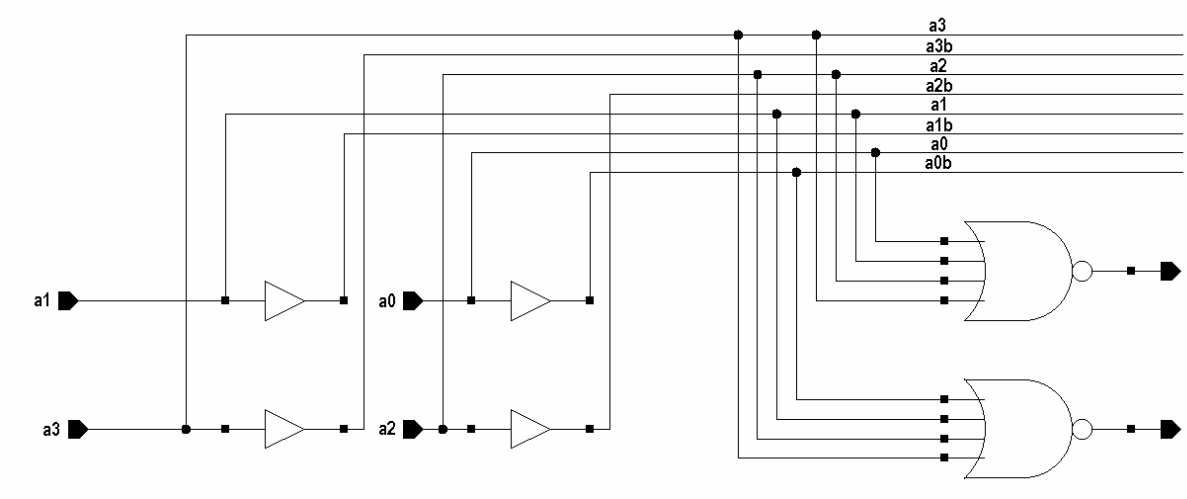


Fig. 5.2 – Trecho do decodificador de linha

A Figura 5.2 mostra um trecho do esquemático representando as portas lógicas do decodificador, que também foi implementado com transistores com dimensões mínimas. As formas de onda da simulação feita com essa estrutura estão mostradas na Figura 5.3.

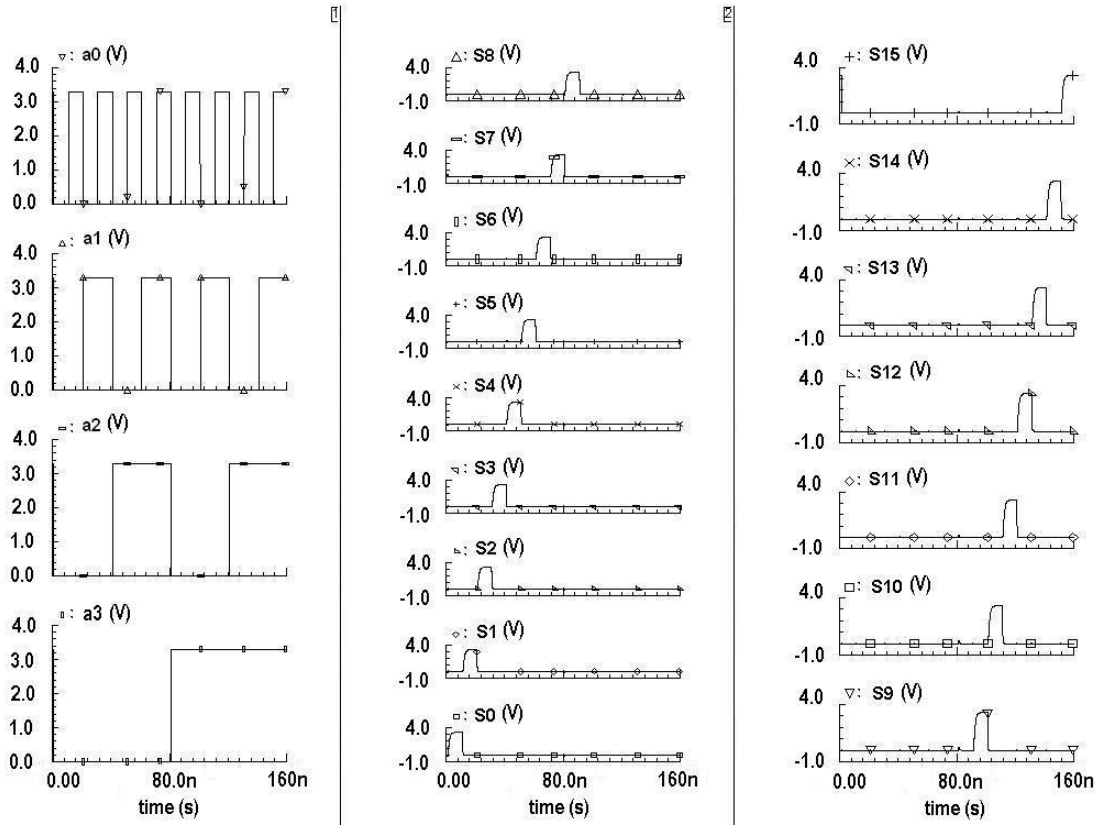


Fig. 5.3 - Simulação do decodificador de 4 bits

O próximo passo foi juntar o decodificador com a matriz, de maneira que os valores especificados na Tabela 5.1 pudessem ser acessados a partir de seus endereços. A Figura 5.4 mostra o diagrama final da matriz ROM com o decodificador e o conversor paralelo-serial.

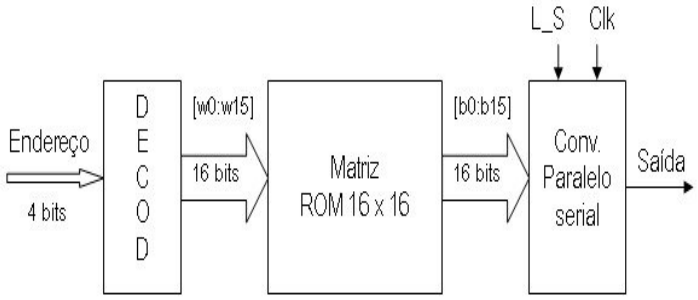


Fig. 5.4 - ROM

Para se reduzir o número de pinos, foi acrescentado um conversor paralelo-serial de 16 bits, que já se encontrava disponível na biblioteca de células do projeto do SoC, na

saída da matriz. Seu funcionamento é descrito a seguir. Quando L_S =1, os dados presentes nas entradas do conversor são carregados nos *flip-flops* que o compõem. Quando L_S=0, os dados de entrada são colocados na saída do conversor a cada período de *clock*. Para que o primeiro valor, que corresponde ao bit menos significativo, apareça na saída, não é necessário que L_S seja zero. Basta que o valor já esteja carregado (ou seja, que L_S tenha sido 1) e que o *clock* esteja alto. A Figura 5.5 ilustra o funcionamento dessa estrutura. A saída Out foi obtida em resposta a uma entrada 1011.

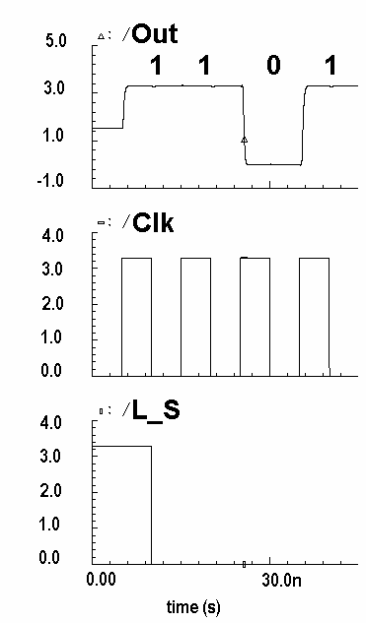


Fig. 5.5 – Simulação do conversor paralelo-serial de 4 bits [COSTA, 2004a]

5.3 ESTRUTURAS DE TESTE DA ROM

Conforme foi explicado no Capítulo 2, existem vários tipos de teste que podem ser realizados. Geralmente memórias ROM são testadas utilizando-se técnicas específicas. Neste caso, devido à simplicidade do circuito e também ao tempo limitado disponível para elaboração do mesmo, foi escolhida a técnica de particionamento do circuito, de modo que se possa observar as saídas tanto do decodificador quanto da matriz em si. Foi elaborado também um teste para o conversor paralelo-serial de 16 bits utilizado na saída.

Foram acrescentados 16 multiplexadores 2:1 nas entradas do conversor paralelo-serial. Quando o sinal de seleção teste_conv é igual a zero, as saídas dos multiplexadores serão as entradas 0, que correspondem aos bits de saída da ROM. Quando teste_conv é igual a um, as saídas dos multiplexadores serão as entradas 1, que são uma combinação dos

bits de endereço da ROM. Essa combinação foi feita de maneira a se facilitar as interconexões no *layout*, e pode ser vista na Figura 5.6. Sendo assim, nas entradas do multiplexador que está conectado ao bit mais significativo do conversor foram ligados o bit mais significativo da saída da ROM, b15, e o bit de endereço a2. No multiplexador conectado ao segundo bit mais significativo do conversor, foram ligados a saída b14 da ROM, e o bit de endereço a1. Nas entradas do multiplexador conectado ao terceiro bit mais significativo do conversor foram ligados a saída b13 da ROM e novamente o bit de endereço a2, e assim por diante. Para os 7 bits menos significativos do conversor, foram alternadas com as saídas da ROM os bits de endereço a0 e a1, nas entradas dos multiplexadores. Dessa forma, será possível testar o conversor paralelo-serial na saída da ROM utilizando os bits de endereço da memória, que serão pinos de entrada no *chip*.

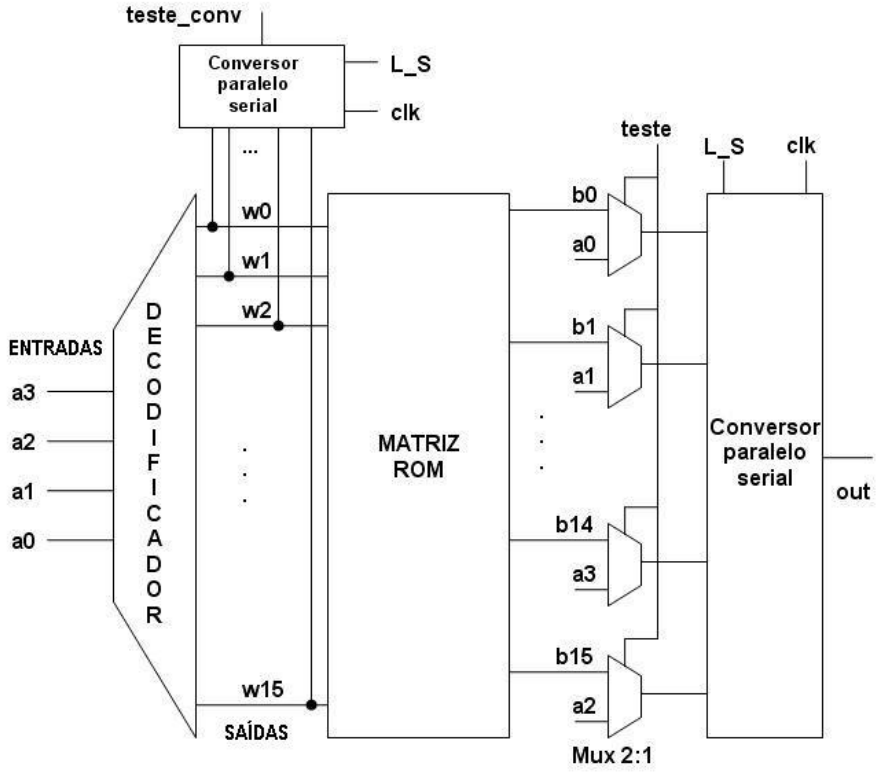


Fig. 5.6 – Estruturas de teste da ROM

Para testar o decodificador, foi inserido também um conversor paralelo-serial de 16 bits. Dessa forma, as saídas do decodificador podem ser verificadas serialmente na saída do conversor (teste_decoder).

Os vetores de teste estão listados no Apêndice A.1.

5.4 SIMULAÇÃO

Foram feitas simulações de leitura em todas as posições. Como exemplo, a palavra armazenada na primeira posição (0000) da ROM foi 0010 0111 0000 0000. A Figura 5.7 mostra o resultado da simulação de uma leitura para $a_0=a_1=a_2=a_3=0$, cujo resultado pode ser visto na saída do conversor paralelo-serial, ainda sem a utilização dos multiplexadores descritos anteriormente.

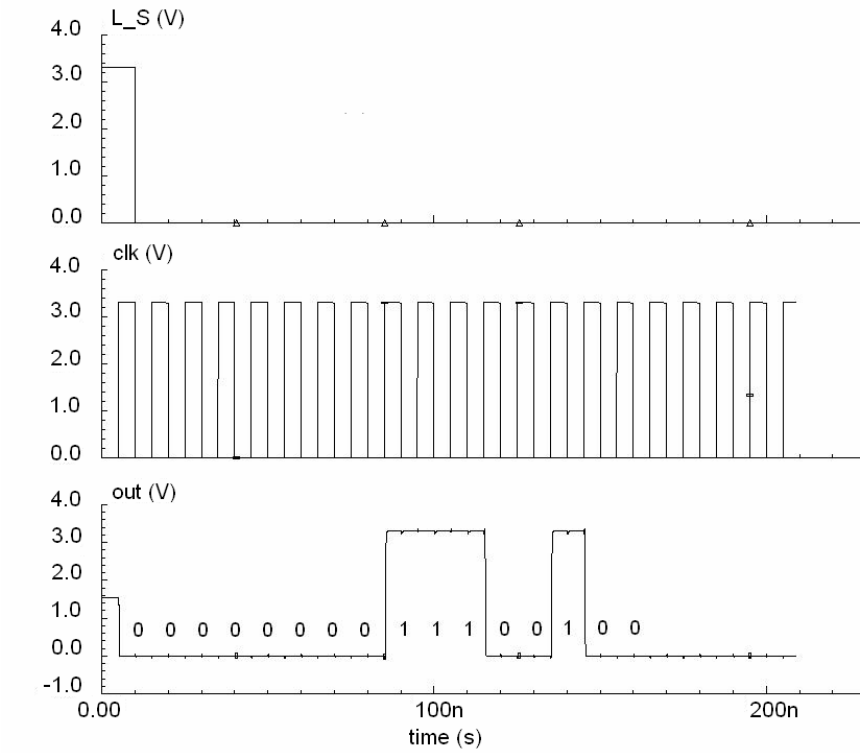


Fig. 5.7 – Leitura da palavra armazenada na posição 0000

Dessa forma, observa-se o correto funcionamento da estrutura, apenas ressaltando-se que o resultado na saída *out* do conversor é lido do bit menos significativo para o mais significativo, conforme foi explicado no item anterior.

A simulação final incluindo os multiplexadores e a estrutura de teste do decodificador está mostrada no Capítulo 9.

6 PROJETO DA SRAM (*STATIC RANDOM-ACCESS MEMORY*)

6.1 INTRODUÇÃO

A RAM proposta por [BENÍCIO, 2002] possui capacidade de 8 kBytes, conforme foi exposto no Capítulo 4. Foi escolhida uma RAM estática, pois, conforme foi visto no Capítulo 2, a DRAM exige operações de *refresh* devido às correntes de fuga. A SRAM ocupa maior área do que a DRAM, que usa capacitores como elementos de armazenagem, mas tende a ser mais rápida do que a mesma. Isso ocorre porque as SRAM armazenam o dado e o seu complemento, e essa armazenagem diferencial reduz significativamente o tempo de leitura, que usualmente é maior do que o tempo de escrita e, portanto, é responsável pelo tempo de acesso das SRAM [MOHAN, 2003].

O presente capítulo trata do projeto da célula utilizada pela SRAM, bem como dos circuitos auxiliares para operações de escrita e leitura na mesma.

6.2 ESPECIFICAÇÃO

Da maneira similar à memória ROM, foi implementada uma SRAM de 128 bits como veículo de validação preliminar da estrutura final, sendo 3 bits de endereçamento para 16 linhas de palavras de 16 bits (2^3 x 16 bits = 128 bits).

A Figura 6.1 mostra o diagrama da SRAM e seus sinais de controle, entrada e saída.

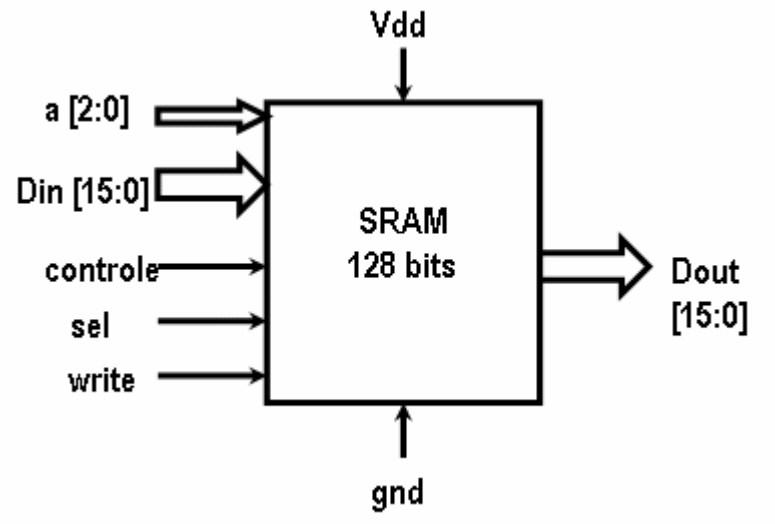


Fig. 6.1 – Diagrama da SRAM

O sinal controle serve para ativar a pré-carga (*controle* = 1) antes de uma operação de leitura, e para desativá-la após a mesma. O sinal *sel* habilita o amplificador sensor quando está em nível lógico baixo, e o sinal *write* habilita a operação de escrita (*write* = '1').

6.3 ARQUITETURA DA SRAM

A matriz consiste em células nas quais os bits são armazenados em grupos de 16. Cada célula é um circuito eletrônico capaz de armazenar um bit. Conforme foi descrito no Capítulo 2, uma matriz de células com 2^M linhas e 2^N colunas possui uma capacidade total de armazenagem de 2^{M+N}. Cada célula está conectada em uma das 2^M linhas, conhecidas como linhas de palavras, e em uma das 2^N colunas, conhecidas como linhas de bit. Uma célula particular é escolhida ativando-se sua linha de palavra e sua linha de bit [SEDRA, 2000]. Neste caso em particular, M = 3 e N = 4. A Figura 6.2 mostra a arquitetura da SRAM projetada. O endereço é de 3 bits, e os dados de entrada e saída (Din e Dout, respectivamente) possuem 16 bits.

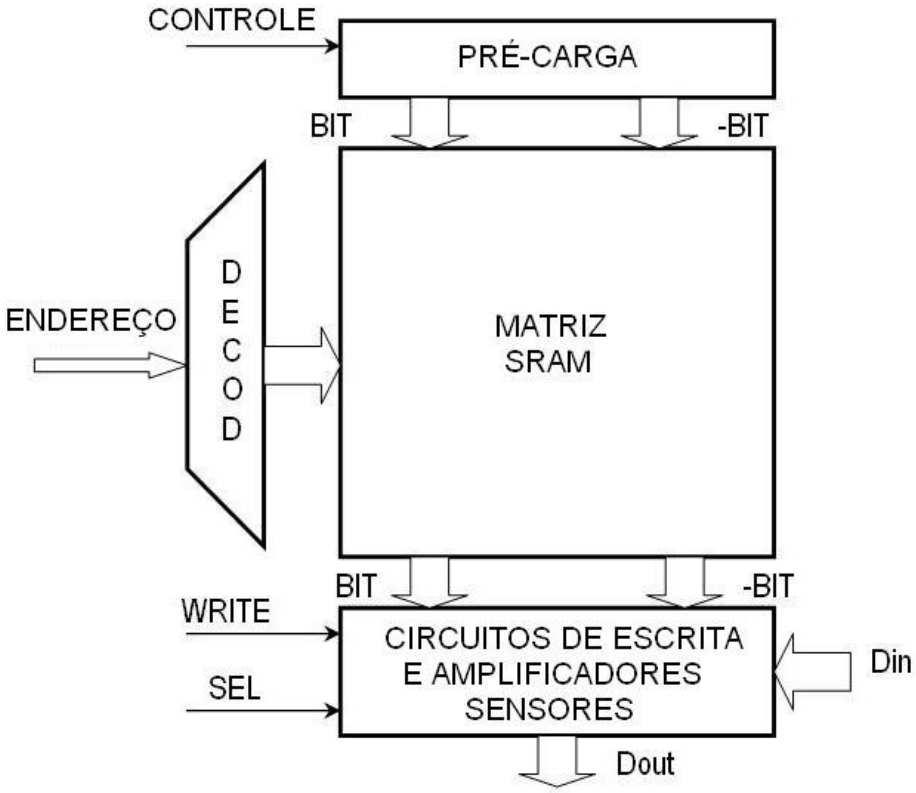


Fig. 6.2 – Arquitetura da SRAM

O decodificador de linha ativa uma das palavras. Trata-se de um circuito lógico combinatório que seleciona a palavra de acordo com o endereço correspondente na sua entrada. A matriz é formada por células 6-T (6 transistores). A pré-carga alimenta as linhas *bit* e *–bit* antes de uma operação de leitura. Ela é ativada quando *controle* = 1. Quando *write* = 1 e *sel* = 1, o circuito de escrita armazena o valor de *Din* na palavra selecionada pelo decodificador. Quando *write* = 0 e *sel* = 0, os amplificadores sensores são habilitados e, a partir da diferença de corrente entre as linhas de bit, mostram 0 ou 1 na saída, dependendo do valor armazenado na célula. Os estados (*write* = 0, *sel* = 1) e (*write* = 1, *sel* = 0) não estão definidos.

A Tabela 6.1 mostra a correspondência entre os sinais da SRAM mostrada na Figura 6.2 e os sinais da unidade de controle do processador (vide Figura 4.7).

Tabela 6.1 - Sinais da SRAM

Sinal da SRAM	Sinal da Unidade de controle
controle	controle
Endereço [a2:a0]	Endereço
Din	WriteData
Dout	ReadData
sel	Rmem\
write	WMem

6.4 CÓDIGO VHDL

Foi elaborado um código em linguagem de descrição de *hardware* (neste caso, VHDL) para verificação funcional da memória inserida no caminho de dados do processador RISC 16 bits.

A memória é a área destinada ao armazenamento de informações, ou seja, é o local onde são mantidos os programas que estão sendo executados, e que contém também os dados necessários à execução desses programas. A memória projetada em VHDL é composta por um *array* de 50 palavras. O tamanho da memória foi reduzido unicamente para diminuir o tempo de síntese do projeto. A estrutura utilizada na simulação possui 5 sinais de entrada (*rst*, *RMem*, *WMem*, *endereco* e *WriteData*) e 1 de saída (*ReadData*). O *rst* é utilizado para inicializar a memória. Quando *RMem* está ativo, o conteúdo do endereço especificado é passado para saída *ReadData,* e quando *WMem* =1 o conteúdo do endereço especificado é substituído pelo dado na entrada *WriteData*.

A Figura 6.3 mostra as formas de onda do código simulado no *software* Foundation 4.1i. O código completo está listado no Apêndice B.1.

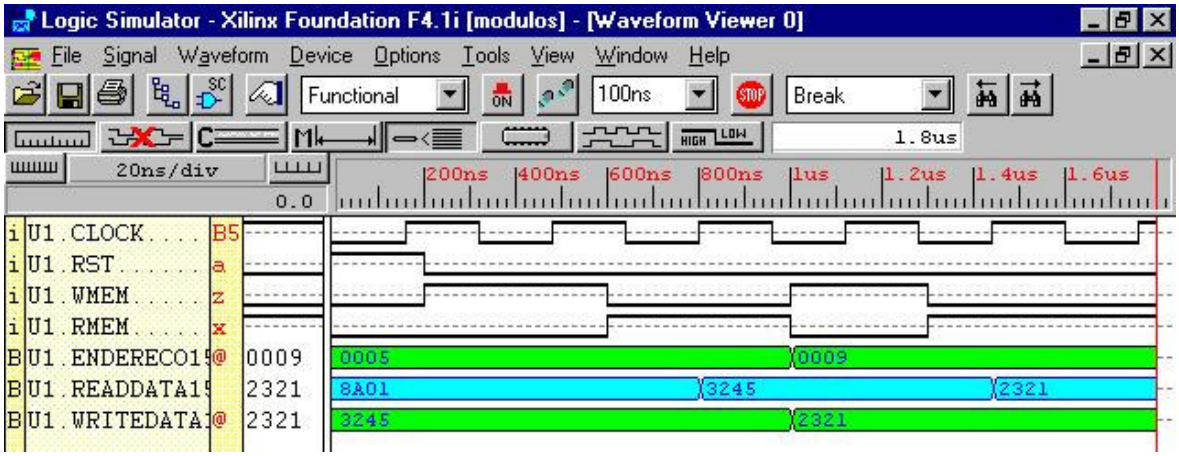


Fig. 6.3 - Formas de onda da simulação do código VHDL da memória

Nesta simulação, o sinal *rst* estava inicialmente ativado, e foi feita a leitura da primeira posição de memória, onde estava armazenada a instrução $8A01_H$ (vide código da memória no Apêndice B1). Pode-se observar que, para que esta instrução apareça no barramento de saída *ReadData* quando *rst* = 1, não é necessário que ReadMem seja igual a 1. Em seguida, foi feita uma escrita (*WMem* = 1, *RMem* = 0) do dado 3245_H (*WriteData*) na posição 0005_H (*endereco*) e uma leitura (*WMem* = 0, *RMem* = 1) na mesma posição, para verificar se a escrita foi feita. Em seguida, foi feita uma escrita do dado 2321_H na posição 0009_H e uma leitura nesta mesma posição. Observa-se que o funcionamento da memória é adequado às especificações requeridas pelo processador.

6.5 BLOCOS DA SRAM

A seguir, são descritos os blocos que compõem a memória SRAM. Como esta última é um circuito de funcionamento complexo e foi adotada a metodologia hierárquica, os blocos foram projetados e testados separadamente antes de serem integrados ao circuito final.

6.5.1 CÉLULA

A célula escolhida foi a tradicional 6-T [SEDRA, 2000], [WESTE, 1993], [CHANDRAKASAN, 2001], [HODGES, 2003](Figura 6.4), que consiste em um par cruzado de inversores conectado por dois transistores de acesso (transistores de passagem)

às linhas *bit* e *–bit*. Esses transistores conduzem quando a linha de palavra é selecionada. Para escrever na célula, o dado é colocado na linha bit, e seu complemento, na linha –bit. Internamente, a célula armazena o dado em um lado e seu complemento no outro. Uma operação de leitura inicia-se com um pré-carregamento das linhas *bit* e *–bit*. A linha de palavra *W* é então acionada, e uma das linhas *bit* ou *–bit* será descarregada por um dos transistores *pull-down* da célula.

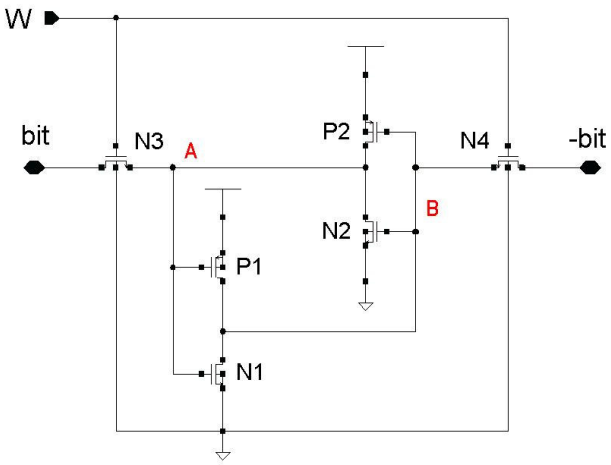


Fig. 6.4 – Célula 6-T

A VTC (*Voltage Transfer Characteristic*) de dois inversores em uma configuração cruzada está mostrada na Figura 6.5. Essa curva deve ser levada em consideração para operações de escrita e leitura na célula. Na configuração cruzada, os valores armazenados são representados pelos dois estados estáveis na VTC. A célula mantém o seu estado corrente até que um dos seus nós internos ultrapasse a tensão de limiar Vs. Quando isso ocorre, a célula muda o seu estado. Logo, durante uma operação de leitura, deve-se evitar essa mudança, enquanto que em uma operação de escrita essa mudança deve ser forçada.

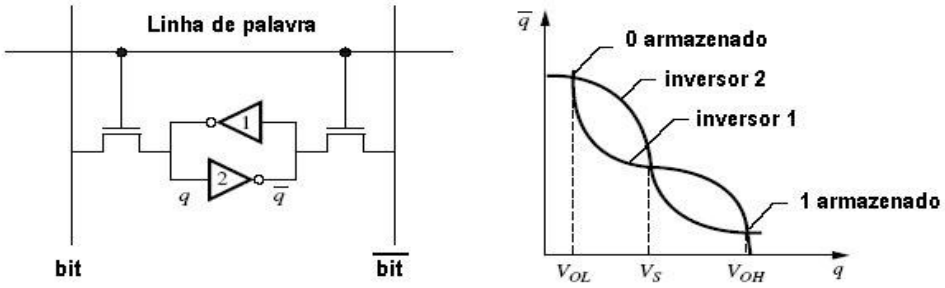


Fig. 6.5 - Célula 6-T e VTC dos inversores 1 e 2 [HODGES, 2003]

Em uma operação de leitura, apenas um dos lados da célula "puxa" corrente. Como resultado, aparece uma pequena tensão diferencial (cuja polaridade depende do dado armazenado na célula) entre as linhas de bit, que possuem uma capacitância grande associada ao número de células conectadas às mesmas. Essa capacitância se deve principalmente a capacitâncias de fonte/dreno, mas também possui componentes associados à capacitância do fio e aos contatos dreno/fonte. Tipicamente, um contato é compartilhado entre duas células. Sendo assim, a capacitância de uma linha de bit, C_{bit}, pode ser obtida utilizando a Eq. 6.1 [HODGES, 2003]:

$$C_{bit} = (cap.\ fonte/dreno + cap.\ fio + cap.\ contato)\ x\ número\ de\ células\ na\ coluna \quad (6.1)$$

O projeto da célula envolve a seleção das dimensões dos seis transistores mostrados na Figura 6.4 para garantir as operações de leitura e escrita. Como a célula é simétrica, somente três transistores precisam ser especificados: N1, P1 e N3 ou N2, P2 e N4. O objetivo é selecionar o tamanho que minimize a área, tenha boa performance, boa estabilidade na leitura e escrita, e boa imunidade ao ruído [HODGES, 2003].

Operação de leitura: Supondo que a célula está armazenando inicialmente "0", o nó *A* está com zero, devido a N2, e o nó *B* está com "1", devido a P1. Então:

1. A operação de leitura começa com as duas linhas de bit (*bit* e *–bit*) pré-carregadas com Vdd-Vtn. O endereço é decodificado e a linha de palavra é selecionada, de forma que N3 e N4 conduzem.

2. A conexão em série de N3 e N2 (que estão conduzindo) começa a "puxar" corrente I_{cell} da linha *bit*, descarregando a capacitância C_{bit}. Como o nó B está em Vdd e P1 está conduzindo, pouca corrente flui através de N4 e a linha *-bit* permanece em Vdd.

3. Após um tempo suficiente (tipicamente da ordem de ns) para permitir que uma diferença de corrente apareça entre *bit* e *–bit*, o amplificador sensor é ativado, e o valor que está armazenado na célula é lido.

Observações: A velocidade de leitura está relacionada ao tempo necessário para o amplificador ler o valor correto. N3 e N2 formam um divisor resistivo entre *bit* e o nó A. Logo, a tensão em A irá aumentar em resposta à ativação da linha de palavra, o que pode ser problemático se N1 começar a conduzir, descarregando o nó B e, conseqüentemente, a linha -*bit*, que deve permanecer em Vdd durante a operação de leitura. O inversor formado por P2 e N2 poderia começar a amplificar a descarga do nó B e alterar o "1" armazenado para "0". Deve-se achar um compromisso entre o limite do aumento de tensão em A, velocidade de leitura e área dos dispositivos [CHANDRAKASAN, 2001], [HODGES, 2003].

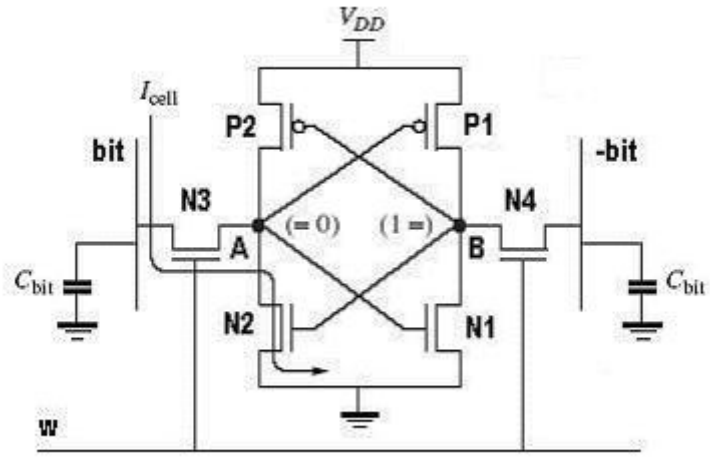


Fig. 6.6 - Operação de leitura na célula [HODGES, 2003 – alterada]

Os valores limite dos tamanhos dos transistores podem ser estimados resolvendo-se as equações simplificadas de corrente Ids em N3 e N2 para a variação máxima de tensão permitida no nó A. Como tensões entre zero e 1,3V caracterizam o nível lógico zero, a tensão em A deve ser menor ou igual a 1,3V. Pode-se assumir que a corrente em N3 é igual à corrente em N2. Dessa forma, ignorando-se o efeito de corpo [RABAEY, 2003]:

$$k_{n,N3}\left[(Vdd-\Delta V-Vtn)Vdsatn-\frac{Vdsatn^2}{2}\right]=k_{n,N2}\left[(Vdd-Vtn)\Delta V-\frac{\Delta V^2}{2}\right] \quad (6.2)$$

onde:

$k_{n,N3}$ = parâmetro de transcondutância do processo (N3)

Vdd = tensão de alimentação

ΔV = variação de tensão no nó A

Vtn = tensão de limiar do transistor n

Vdsatn = tensão de saturação do transistor n

$k_{n,N2}$ = parâmetro de transcondutância do processo (N2)

A tensão ΔV é afetada pelos tamanhos de N3 e N2. À medida em que a tensão no nó A aumenta, o valor de Vgs para N3 diminui e, portanto, menos carga é removida da linha bit, retardando a leitura. Para criar uma célula estável, é necessário ajustar os tamanhos de N3 e N2. O primeiro passo é definir a razão entre o *pull-down* e o transistor de passagem (CR – *cell ratio*) [RABAEY, 2003]:

$$CR = \frac{W_{N2} / L_{N2}}{W_{N3} / L_{N3}} \qquad (6.3)$$

Logo, a partir da Eq. 6.2 [RABAEY, 2003]:

$$\Delta V = \frac{Vdsatn + CR(Vdd - Vtn) - \sqrt{Vdsatn^2(1+CR) + CR^2(Vdd - Vtn)^2}}{CR} \qquad (6.4)$$

Substituindo-se os valores mostrados na Tabela 6.2, obtém-se que CR deve ser maior ou igual a 10.

Tabela 6.2 – Parâmetros para a tecnologia 0,35 um [AMS, 2001]

Parâmetro	*ΔV*	*Vdd*	*Vtn*	*Vdsatn*
Valor	1,3V	3,3V	0,5V	2,8V

Operação de escrita: Essa operação começa com um circuito externo carregando dados complementares nas linhas *bit* e *–bit*.

1. O decodificador de endereço então seleciona a linha de palavra, de forma que os dois transistores de acesso conduzam. Supondo-se novamente que a célula está armazenando "0", pretende-se escrever "1" (nó *A* vai para "1") na mesma. A linha de bit que irá para "0" (linha *-bit*) também forma um divisor de tensão com P1 e N4.

2. Para escrever na célula, a porta de passagem deve conduzir mais do que o dispositivo PMOS. Isso permitirá que o nó B vá para um valor baixo o suficiente para que o par inversor P2 e N2 comece a amplificar o novo dado. À medida em que esse inversor começa a amplificar o nível zero do nó B, o *pull-down* do nó A é cortado, P2 começa a conduzir e o nó A vai para Vdd. O par inversor P1 e N1 é ativado, forçando o nó *A* para zero [CHANDRAKASAN, 2001].

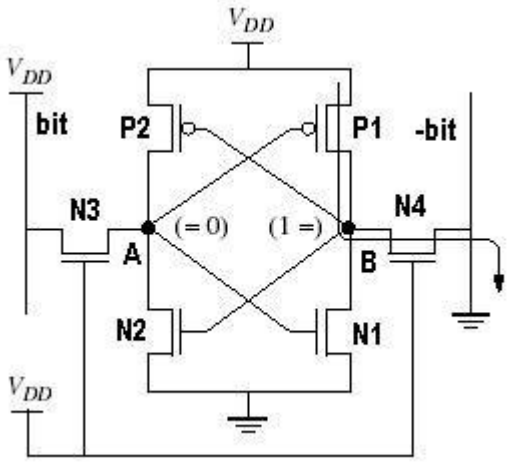


Fig. 6.7 - Operação de escrita na célula [HODGES, 2003 – alterada]

Observa-se que o novo dado (“0”) foi forçado através da linha –bit e do transistor N4. Uma escrita confiável pode ser assegurada se a tensão de B for menor do que a tensão de limiar do inversor formado por P2 e N2. As condições podem ser obtidas novamente pelas equações simplificadas de corrente [RABAEY, 2003]:

$$k_{n,N4}\left[(Vdd - Vtn)V_B - \frac{{V_B}^2}{2}\right] = k_{p,P1}\left[(Vdd - |Vtp|)Vdsatp - \frac{{Vdsatp}^2}{2}\right] \qquad (6.5)$$

onde:

$k_{n,N4}$ = parâmetro de transcondutância do processo (N4)

Vdd = tensão de alimentação

Vtn = tensão de limiar do transistor n

V_B = tensão no nó B

$K_{p,P1}$ = parâmetro de transcondutância do processo (P1)

Vtp = tensão de limiar do transistor p

Vdsatp = tensão de saturação do transistor p

Assumindo-se que o *pull-up ratio* (PR) é dado pela Eq. 6.6, a tensão V_B é dada pela Eq. 6.7 [RABAEY, 2003].

$$PR = \frac{W_{P1} / L_{P1}}{W_{N4} / L_{N4}} \qquad (6.6)$$

$$V_B = Vdd - Vtn - \sqrt{(Vdd - Vtn)^2 - 2\frac{\mu_p}{\mu_n} PR\left[(Vdd - |Vtp|)Vdsatp - \frac{Vdsatp^2}{2}\right]} \qquad (6.7)$$

Utilizando-se os parâmetros listados na Tabela 6.3, obtém-se que PR deve ser menor ou igual a 4.

Tabela 6.3 - Parâmetros para a tecnologia 0,35 um [AMS, 2001]

Parâmetro	*Vtp*	*Vdd*	*Vtn*	*Vdsatp*	μ_p/μ_n
Valor	-0,6V	3,3V	0,5V	2,8V	0,5

Sendo assim, obedecendo-se às condições impostas pela operação de escrita e de leitura, optou-se por manter os valores de L mínimos (0,4 um), e variar apenas os valores de W, que estão mostrados na Figura 6.8.

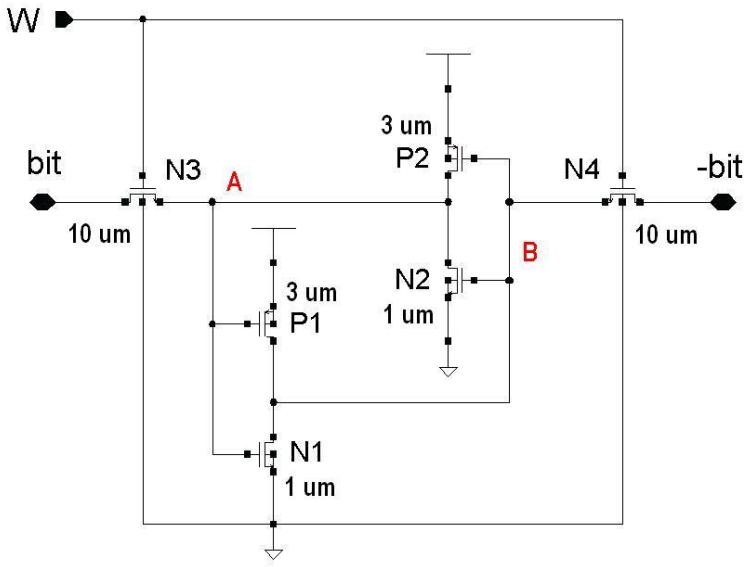


Fig. 6.8 – Valores de W

6.5.2 AMPLIFICADOR SENSOR

O amplificador sensor escolhido (Figura 6.9) consiste de 4 transistores PMOS, de dimensões iguais, numa configuração cruzada. Conforme foi explicado no Capítulo 2, ele é selecionado aterrando-se o nó *sel*. Como a célula está conectada ao amplificador sensor através das linhas de bit, quando a mesma for selecionada pela linha de palavra, correntes irão fluir através dos transistores pelas cargas das linhas de bit. Os drenos de P3 e P4 estão conectados às linhas de dados (DL e –DL), que estão próximas do nível terra. Isso significa que esses transistores operam na saturação. As cargas das linhas de bit são de baixa resistência para assegurar que, durante o acesso à leitura, as linhas de bit estão sempre próximas de Vdd.

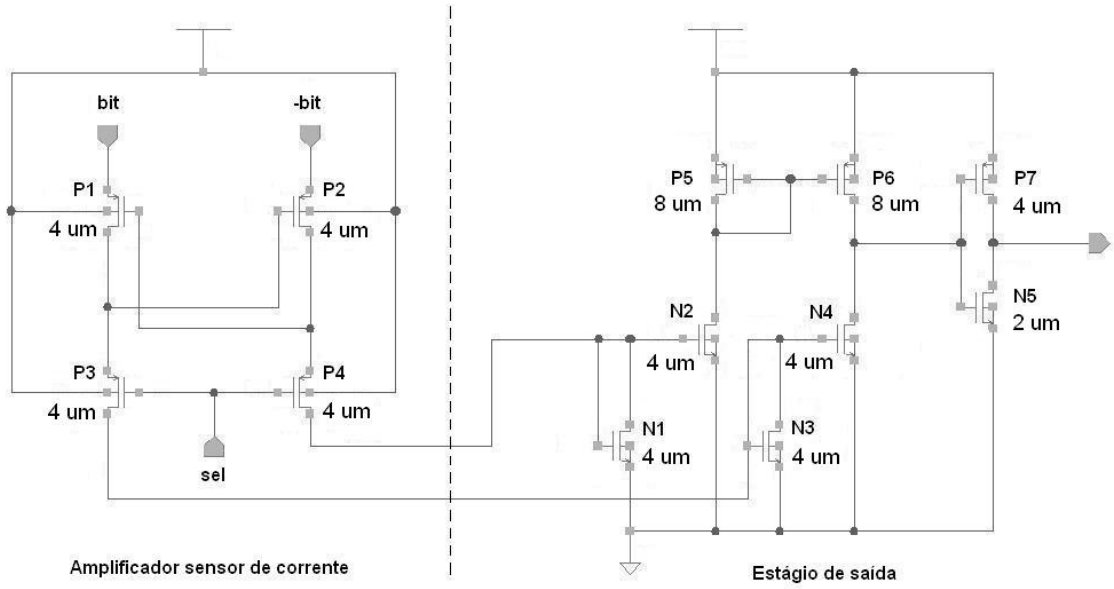


Fig. 6.9 – Amplificador sensor de corrente com seção de saída com valores de W (L = 0,4 um)

O circuito opera da seguinte forma: supondo que a célula é acessada, flui uma corrente *I* para dentro da célula, dependendo do dado que está armazenado na mesma. A tensão Vgs de P1 será igual à de P3, já que suas correntes são iguais e ambos estão na saturação. Esta tensão será representada por V1. O mesmo ocorre com P2 e P4. as suas tensões Vgs são representadas por V2. Segue que, desde que *sel* esteja aterrado, as tensões de ambas as linhas de bit serão iguais a V1+V2. Logo, os potenciais das linhas de bit serão iguais, independentemente da distribuição de corrente. Isso significa que existe um curto-circuito virtual entre as linhas de bit. Como as tensões serão iguais, as correntes nas cargas das linhas de bit também serão iguais. Como a célula consome a corrente *I*, segue que

passa mais corrente em uma das pernas do amplificador do que na outra, de acordo com o dado armazenado na célula. De fato, a diferença entre essas correntes é *I*, a corrente da célula. As correntes nos drenos de P3 e P4 são passadas para as linhas de dados. A corrente diferencial nas linhas de dados é, portanto, igual à corrente na célula. Logo, tem-se um sensor de corrente [SEEVINCK, 1990].

Para a leitura do dado na célula, foi acrescentado um estágio que converte a corrente diferencial entre as saídas do amplificador (DL e –DL) em tensão. Este amplificador consome menos corrente e é mais rápido do que o amplificador diferencial convencionalmente usado [SEEVINCK, 1990].

O teste do amplificador utilizou a configuração mostrada na Figura 6.10.

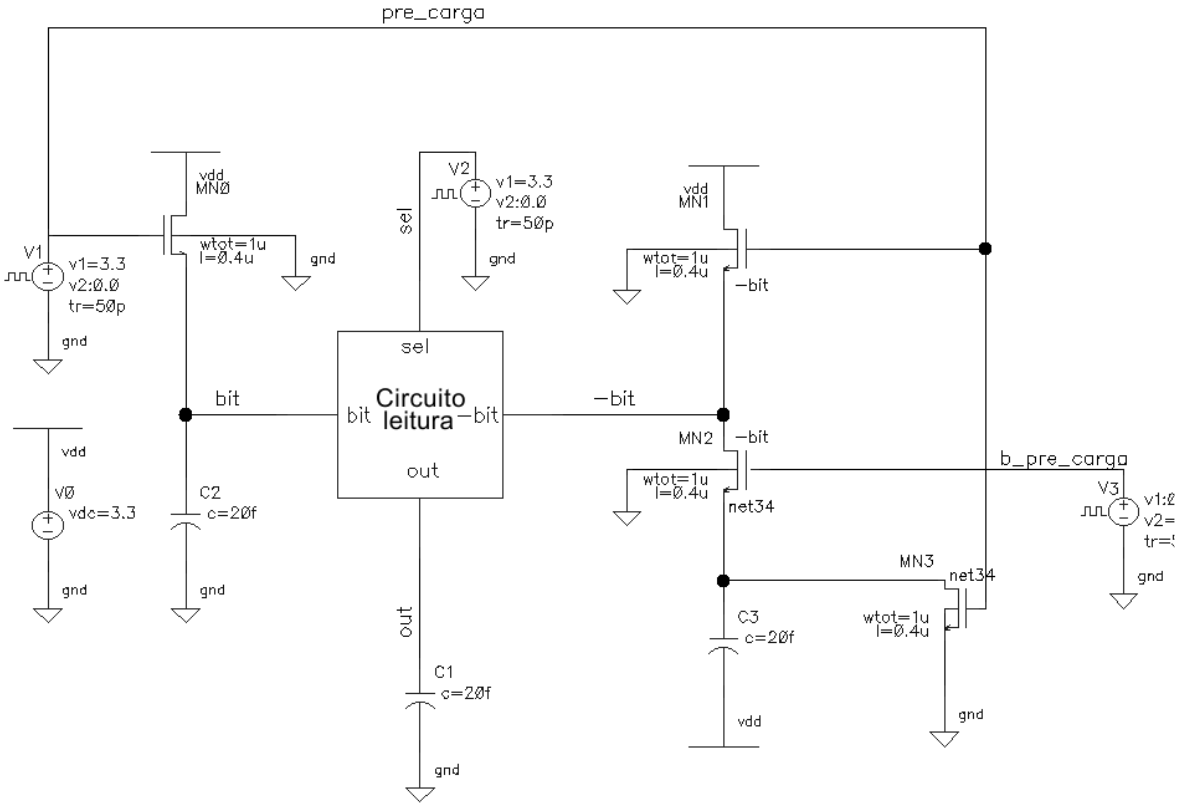


Fig. 6.10 - Estrutura para simulação do amplificador sensor de corrente

A fonte de tensão V1, juntamente com os transistores MN1 e MN0, faz a pré-carga das linhas de bit, que é necessária antes de cada operação de leitura. Os capacitores C2 e C3 simulam a célula de armazenamento, conforme será visto. O transistor MN3 tem a função de "polarizar" o capacitor *C3* com uma tensão negativa enquanto é feita a pré-carga

das linhas de bit. O transistor MN2, controlado pela fonte V3, funciona como chave, conectando o capacitor C3 à linha –bit depois que é feita a pré-carga.

O teste funciona da seguinte forma: quando a fonte V1 estiver com tensão V_{DD}, os transistores MN0 e MN1 conduzem e aplicam uma tensão V_{DD}-V_t no capacitor C2 e nas entradas *bit* e *–bit* do circuito de leitura. MN3 também estará conduzindo e carregando o capacitor C3 com tensão –(V_{DD}-V_t). A fonte V3, cuja tensão é simétrica à de V1, estará aplicando 0V em MN2, fazendo com que este transistor não conduza. O circuito de leitura não está habilitado, pois V2 aplica V_{DD} na entrada *sel*. Nesta configuração, é feita a simulação da pré-carga das linhas de bit.

Quando a tensão de V1 cai para 0V, os transistores MN0, MN1 e MN3 param de conduzir. Ao mesmo tempo a tensão em V3 sobe para V_{DD} levando MN2 à condução. Nesta configuração, os capacitores C2 e C3, carregados com tensões opostas, estarão conectados às linhas *bit* e *–bit,* respectivamente. Portanto, assim que a fonte V2 ativar o sinal de seleção (*sel* = 0), as tensões armazenadas nos capacitores irão produzir correntes entrando e saindo do circuito de leitura, gerando uma diferença de correntes nas linhas de bit, conforme mostra a Figura 6.11.

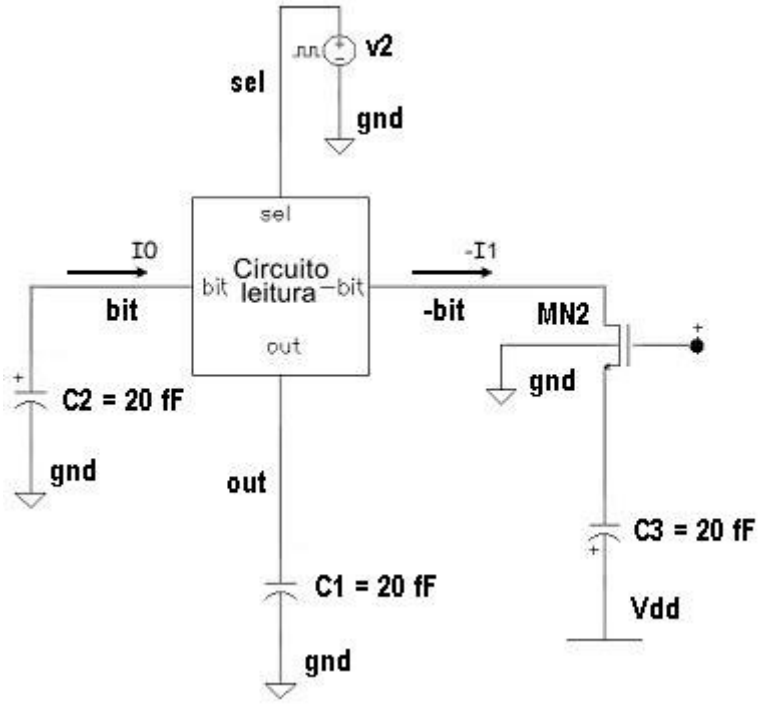


Fig. 6.11 – Estrutura para simulação da leitura de 1 bit

A duração dessas correntes é suficiente para o amplificador sensor do circuito de leitura detectar a diferença entre as mesmas e convertê-la em tensão no estágio de saída.

Na Figura 6.12, observa-se a simulação da leitura do bit 1. A configuração de circuito é a da Figura 6.10. Inicialmente, a saída *out* pode estar indefinida, mas assim que o sinal de seleção é ativado, o circuito de leitura entra em funcionamento, descarregando os capacitores C1 e C2, gerando as correntes I0=I0/bit e –I1=I0/-bit. O sinal *b_pre_carga* habilita o transistor MN2, que conecta C3 à linha *–bit*. O resultado é a mudança do estado da saída que fornece a informação desejada.

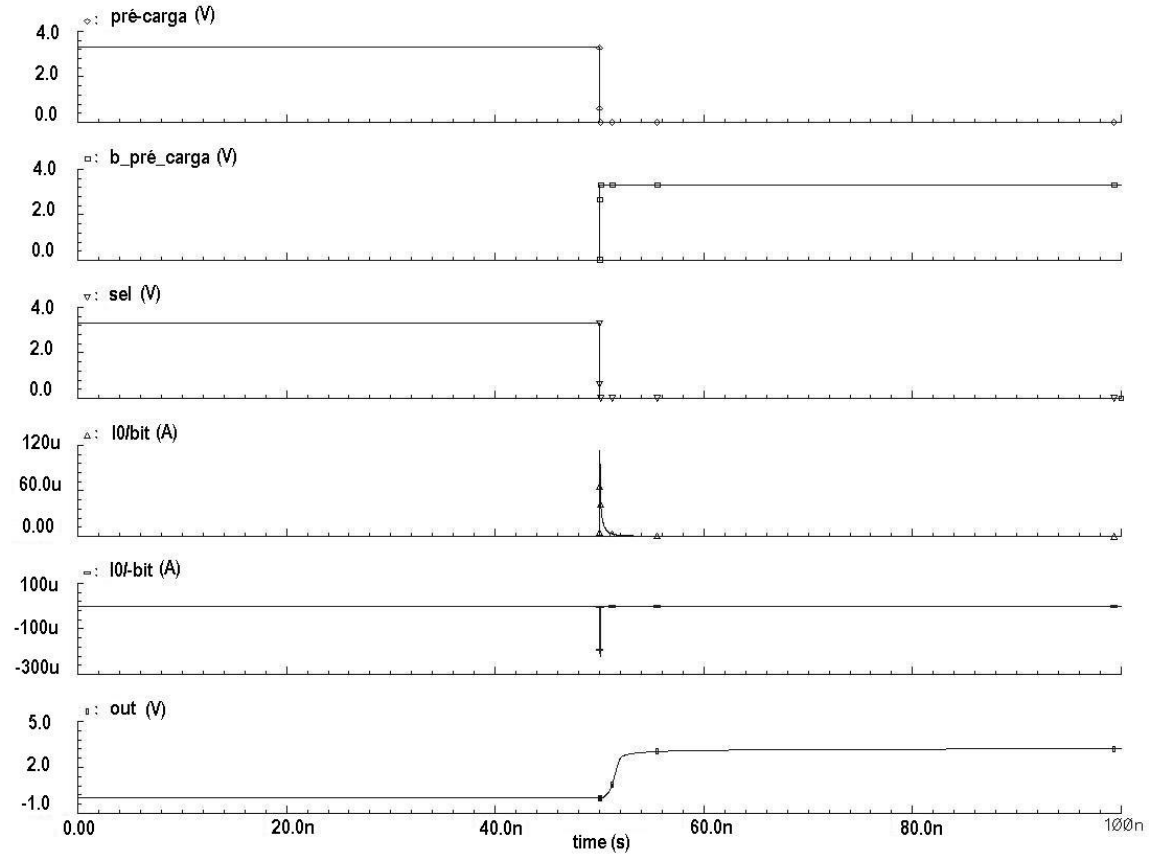


Fig. 6.12 – Simulação da leitura do bit 1

A leitura do bit 0 é feita de forma análoga, bastando trocar as conexões das linhas *bit* e *–bit* entre si. Dessa forma, serão invertidos os sinais das correntes, assim como o sinal da diferença entre elas, resultando na leitura do bit 0. Esta situação é vista na Figura 6.13, onde a saída permanece em zero após a habilitação do sinal de seleção. Há uma variação na tensão de saída, iniciada no instante da transição dos sinais. Entretanto, ela não ultrapassa o nível de leitura do bit 0.

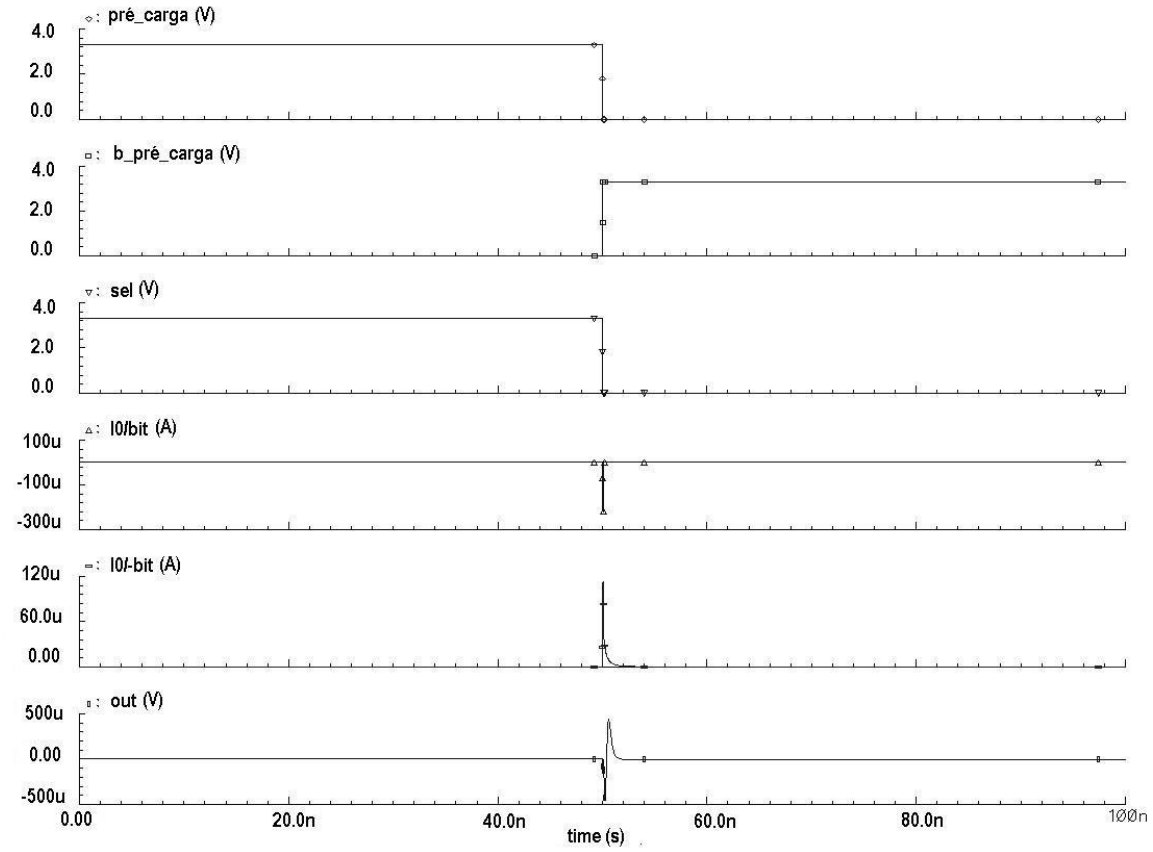


Fig. 6.13 – Simulação da leitura do bit 0

6.5.3 CIRCUITO DE ESCRITA

O objetivo de uma operação de escrita é aplicar tensões na célula RAM de forma que ocorra uma mudança de estado na mesma. A Figura 6.14 mostra o circuito utilizado, onde os transistores são habilitados para permitir que o dado e seu complemento sejam carregados nas linhas de bit. Este circuito utiliza portas de transmissão complementares para melhorar o desempenho da operação de escrita [WESTE, 1993].

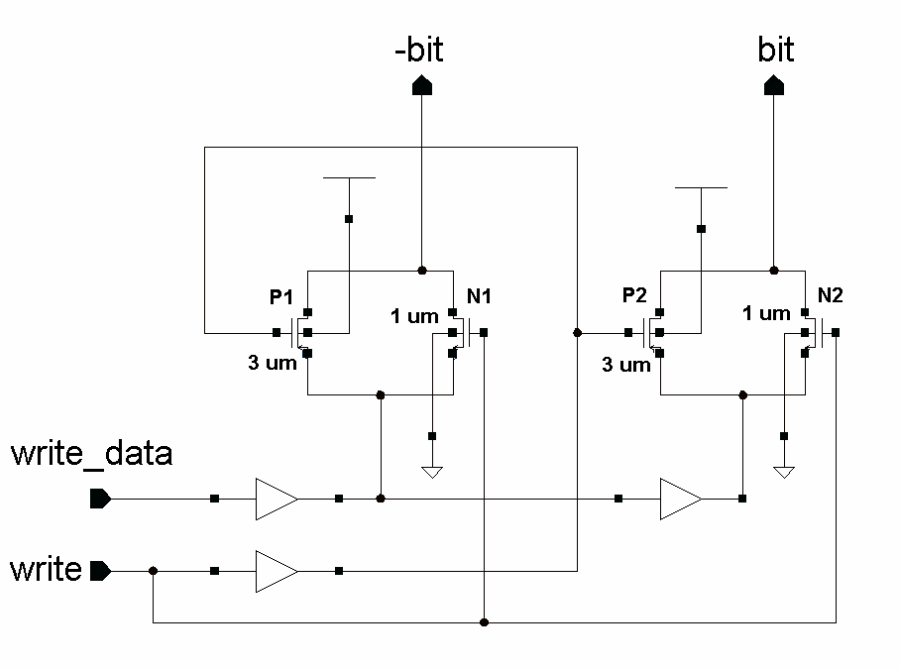


Fig. 6.14 – Circuito de escrita com os valores de W (L = 0,4 um)

A entrada *write_data* corresponde ao dado que será escrito na célula de memória, e *write* é o sinal de ativação do circuito de escrita. O funcionamento do circuito de escrita será descrito a seguir.

Inicialmente, *write* está em nível lógico 0 (0V). Nessa condição, os transistores P1, P2, N1 e N2 estão cortados, fazendo com que as saídas *bit* e *–bit* estejam em alta impedância. Quando o processador necessitar escrever um bit na memória, ele ativa o sinal de escrita (*WMem*), aumentando a tensão em *write* até V_{DD}. O dado a ser escrito deve estar presente em *write_data*. Os pares de transistores P1, N1 e P2, N2 funcionam como portas de transmissão. O bit que chega a *write_data* é reforçado pelo inversor e encaminhado aos dois conjuntos de porta de transmissão. Estas portas transferem para as linhas *bit* e *–bit* o sinal de *write_data* e o seu complemento, respectivamente. Após ter sido feita a gravação do bit, a habilitação da célula é removida e a linha de palavra é levada para nível baixo, desabilitando o circuito de escrita.

6.5.4 PRÉ-CARGA E CIRCUITO AUXILIAR DE PRÉ-CARGA

O circuito de pré-carga (Figura 6.15) carrega as linhas de bit com uma tensão próxima de Vdd antes de uma operação de leitura. Nas outras etapas de funcionamento, a pré-carga permanece desabilitada (*controle* = 0) [CHANDRAKASAN, 2001].

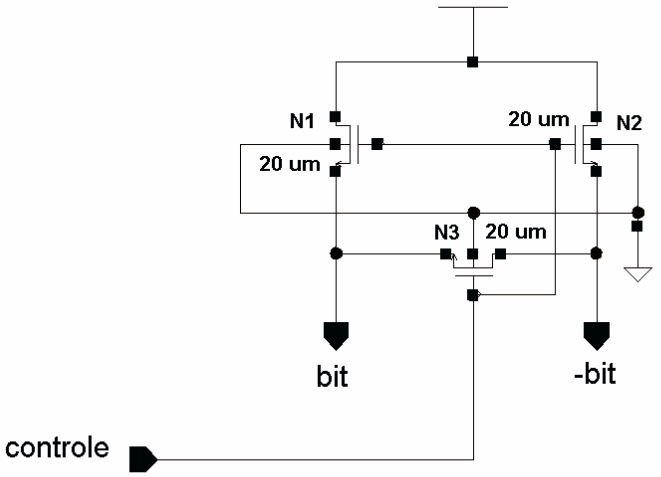


Fig. 6.15 – Pré-carga com valores de W (L = 0,4 um)

O circuito de pré-carga é necessário para carregar as capacitâncias parasitárias nas linhas de bit (dados) antes de um ciclo de leitura.

Os transistores N_1, N_2 e N_3 são NMOS. O sinal de controle é comandado pelo microprocessador, e sempre que é ativado com uma tensão próxima de V_{DD}, os transistores funcionarão como chaves que levam as linhas bit e –bit para um potencial aproximadamente igual a (V_{DD}-V_T) carregando as capacitâncias dessas linhas. O transistor N_3 acelera esse processo e garante que as tensões das linhas serão praticamente iguais. Estes transistores precisam ter grandes valores de "W", de modo a poderem responder rapidamente e fornecer a corrente de carga necessária às capacitâncias das linhas de bit. Essa corrente de carga é dada pela relação $i = C_p \frac{dV}{dt}$, onde C_P é a capacitância parasitária da linha causada pelos outros elementos da célula de memória (circuito de leitura, circuito de escrita, célula de armazenamento, etc), além das próprias linhas de conexão.

Pode-se observar que os transistores N_1 e N_2 operam na saturação, pois V_G=V_D=V_{DD}, logo V_{GS}=V_{DS}, onde V_S é a tensão na linha de bit. Portanto, V_{DS}>V_{GS}-V_t, o que implica em operação na região de saturação do transistor.

Foi necessária a inclusão de um circuito auxiliar de pré-carga. A função desse circuito é desligar o sinal de endereçamento de linha durante uma escrita e leitura sucessivas em um mesmo endereço. O que ocorre é que, após a operação de escrita, a linha de palavra permanecerá selecionada se a próxima operação naquele endereço for uma leitura. Como antes de cada leitura é feita uma operação de pré-carregamento das linhas de

bit, pode ocorrer uma mudança no dado armazenado pela célula, se a linha de palavra estiver habilitada. O circuito auxiliar de controle de pré-carga atua justamente neste intervalo, bloqueando a linha de palavra durante a operação de pré-carga das linhas de bit. A Figura 6.16 mostra o esquemático desse circuito.

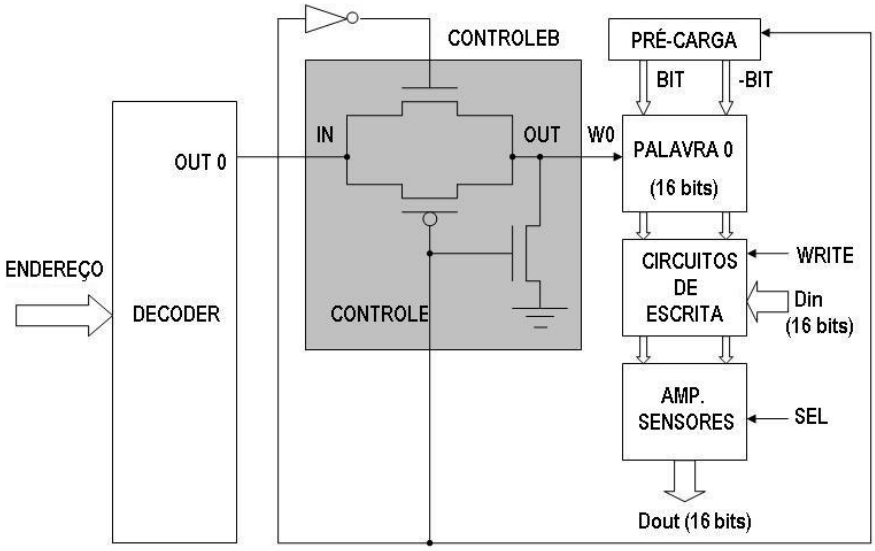


Fig. 6.16 - Circuito auxiliar de controle

No circuito da Figura 6.12, a entrada *in* recebe a saída do decodificador de linha, enquanto *out* é ligado à linha de palavra correspondente. Os terminais *cont* e *cont_b* recebem o sinal de controle e de seu complemento, respectivamente. Durante uma operação de escrita, o sinal *controle* está desabilitado, portanto *cont* está em 0V e *cont_b* em V_{DD}. Nestas condições, o sinal que está em *in* chega livremente até *out* e aciona a linha de palavra, selecionando as células para escrita. Quando o sinal *controle* é ativado com V_{DD} antes da leitura, os transistores da chave CMOS estarão cortados e bloquearão o sinal de palavra, que permanece na entrada *in*. O transistor n cuja fonte está aterrada passa a conduzir, levando o potencial na linha de palavra para 0V. Os transistores de passagem das células deixarão de conduzir, não permitindo a entrada das tensões das linhas de bit, de forma que a pré-carga das linhas pode ser feita sem alterar o estado das células. Com a desativação do sinal *controle,* como na operação de escrita, os transistores da porta de transmissão voltam a conduzir, e o decodificador volta a ter acesso às linhas de palavras das células.

6.5.5 DECODIFICADOR DE LINHA

Conforme foi explicado no Capítulo 2, a função do decodificador de linha é selecionar uma das 8 palavras da matriz SRAM de acordo com os 3 bits de endereço em sua entrada (Figura 6.17). Dessa forma, todas as células conectadas a essa linha de palavra são acessadas para leitura ou escrita. A linha de palavra tem uma capacitância grande, C_{word}, que deve ser carregada pelo decodificador, e que engloba duas capacitâncias de porta e a capacitância do fio por célula [HODGES, 2003]. A capacitância da linha de palavra pode ser encontrada pela Equação 6.8.

$$C_{word} = (2 \text{ x cap. porta} + \text{cap. fio}) \text{ x número de células na linha} \qquad (6.8)$$

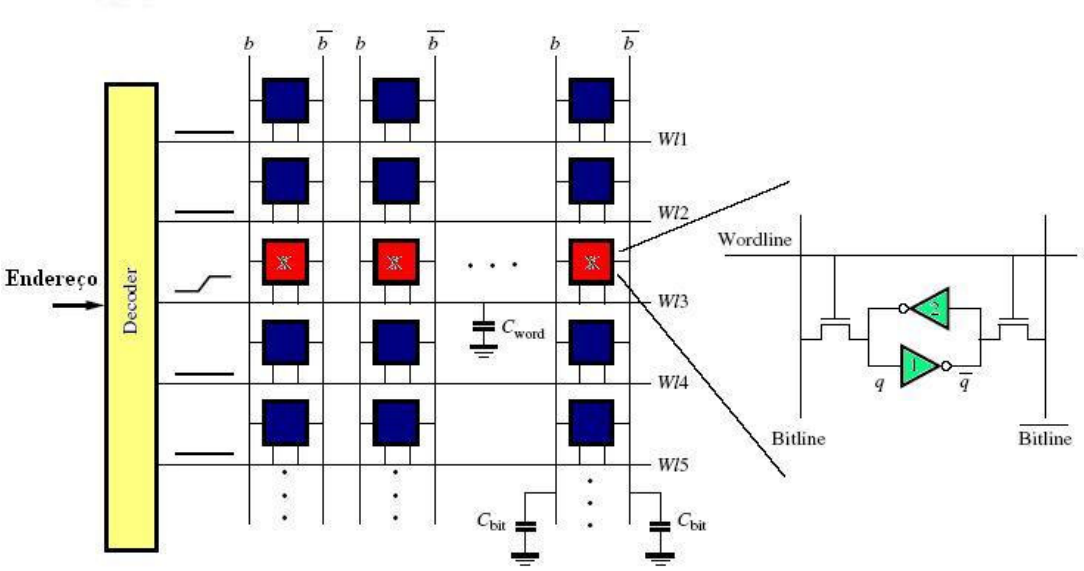


Fig. 6.17 - Função do decodificador de linha [HODGES, 2003 – alterada]

Da mesma forma que a ROM, a RAM possui poucas posições, o que dispensa a utilização de um decodificador de coluna. Sendo assim, foi escolhido um decodificador similar ao da ROM, mas com capacidade de endereçar apenas 8 bits. Trata-se, portanto, de um circuito combinacional implementado com portas NOR, cujo projeto foi análogo ao mostrado no item 5.2.

A Figura 6.18 mostra as forma de onda obtidas na simulação do decodificador de 3 bits.

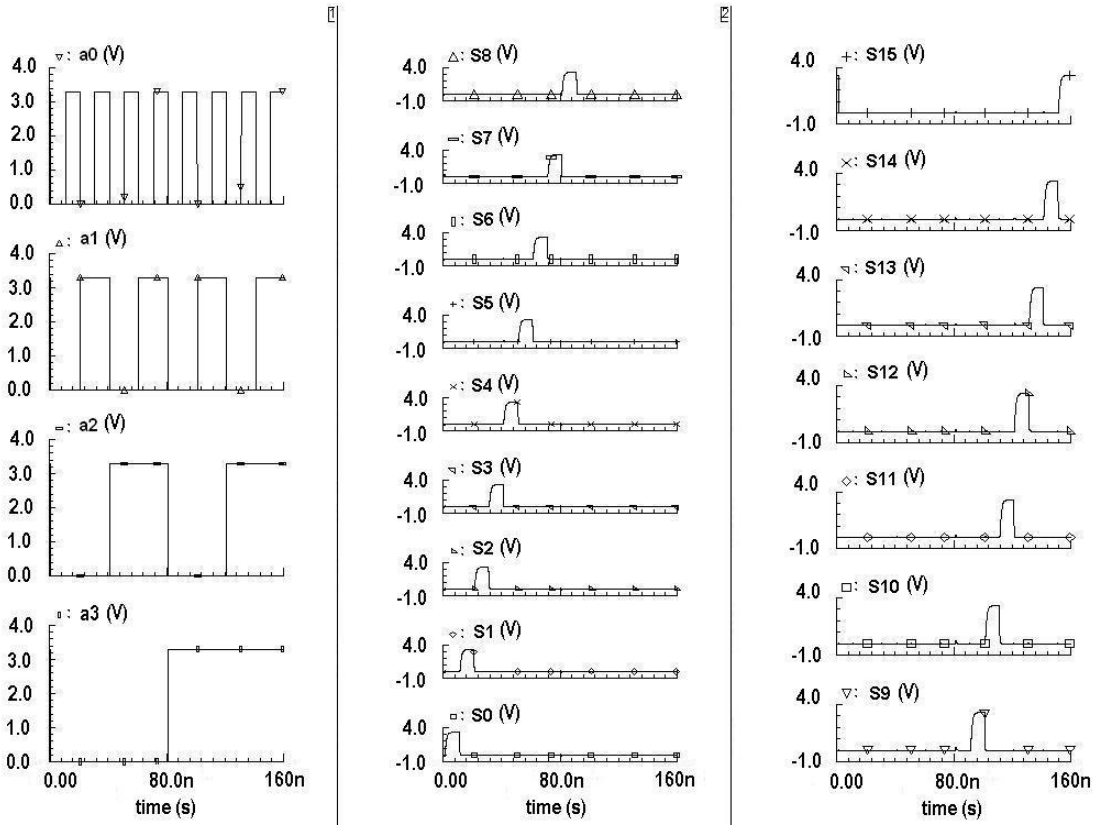


Fig. 6.18 - Simulação do decodificador de linha

6.6 SIMULAÇÕES

Foram feitas simulações para operações de escrita e de leitura. A princípio, essas operações foram executadas em uma única célula. A Figura 6.19 mostra o esquemático correspondente, e a Figura 6.20, as formas de onda obtidas.

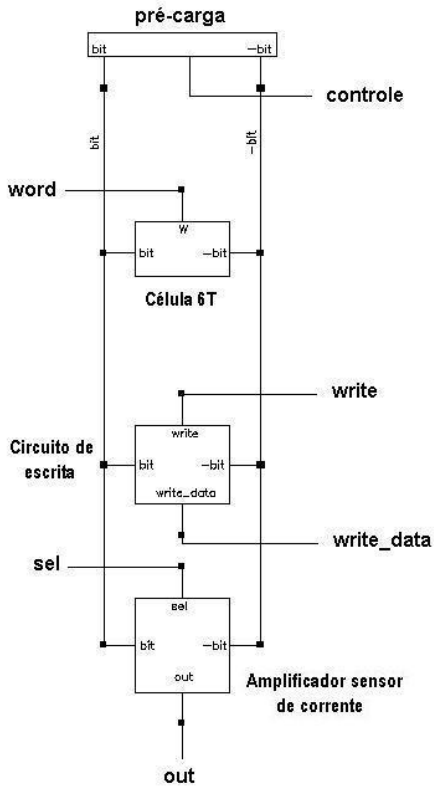


Fig. 6.19 - Célula com circuito de escrita e amplificador sensor

Na simulação, é feita inicialmente uma operação de escrita na célula. O dado de entrada, "1", é em seguida lido duas vezes. Observa-se que é necessária uma operação de pré-carga antes de cada leitura. São feitas, então, a escrita e leitura do dado "0". Pode-se notar que a tensão nos nós A e B (vide Figura 6.4) são complementares e mudam de acordo com o dado escrito na célula. Quando se escreve um "1", o nó A vai para Vdd e o nó B, para zero. Quando se escreve "0", a situação se inverte. Da mesma forma, as tensões nas linhas de bit também são complementares (*bit* vai para Vdd quando o dado é '1' e *–bit* vai para zero, e ambas são invertidas quando o dado é "0"). Entretanto, é importante ressaltar que as mesmas possuem tensões iguais durante a pré-carga.

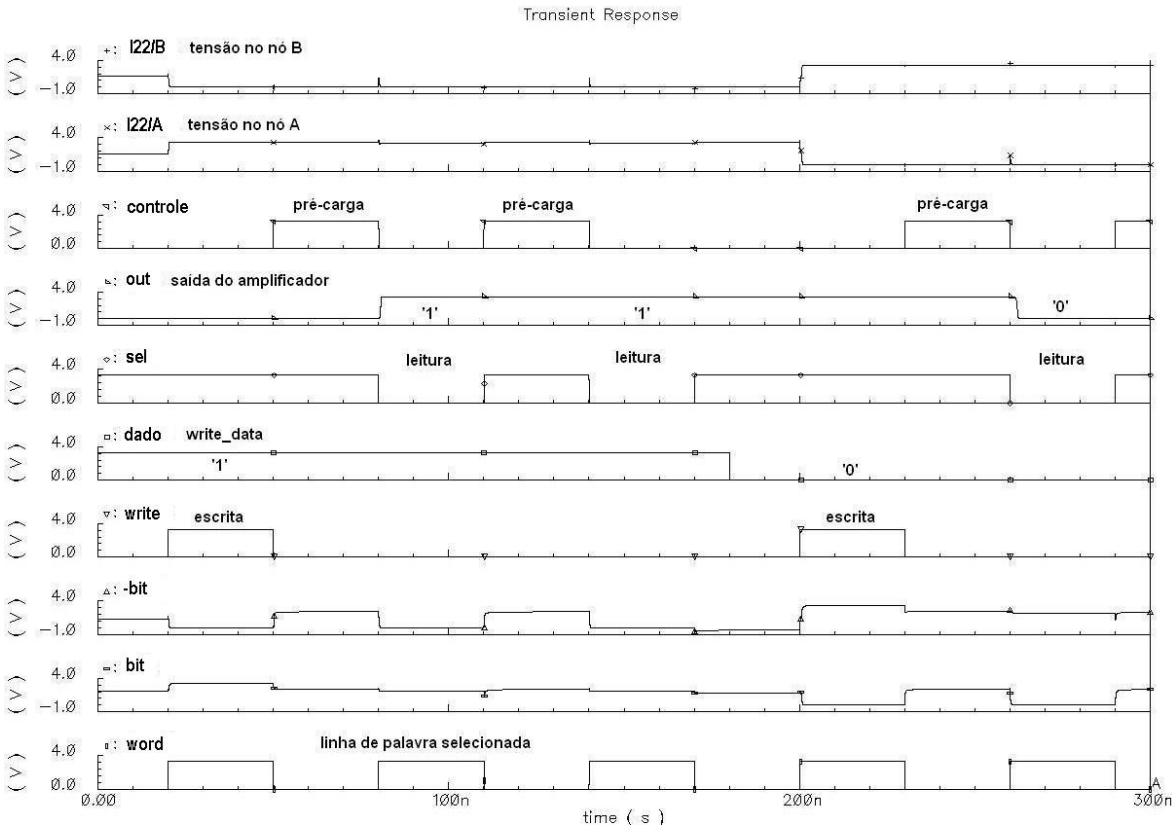


Fig. 6.20 – Simulação da célula da Figura 6.19

Em seguida, simulou-se a matriz de 128 células. Neste exemplo, foi escrito o dado 0000 0000 1111 1111 nas posições 000, 001 e 100. Foi feita uma leitura em cada uma dessas posições. A combinação do dado, bem como das posições a serem escritas e lidas, é apenas uma dentre várias que podem ser feitas para testar o correto funcionamento da estrutura. A simulação da Figura 6.21 foi realizada ainda sem as estruturas de teste.

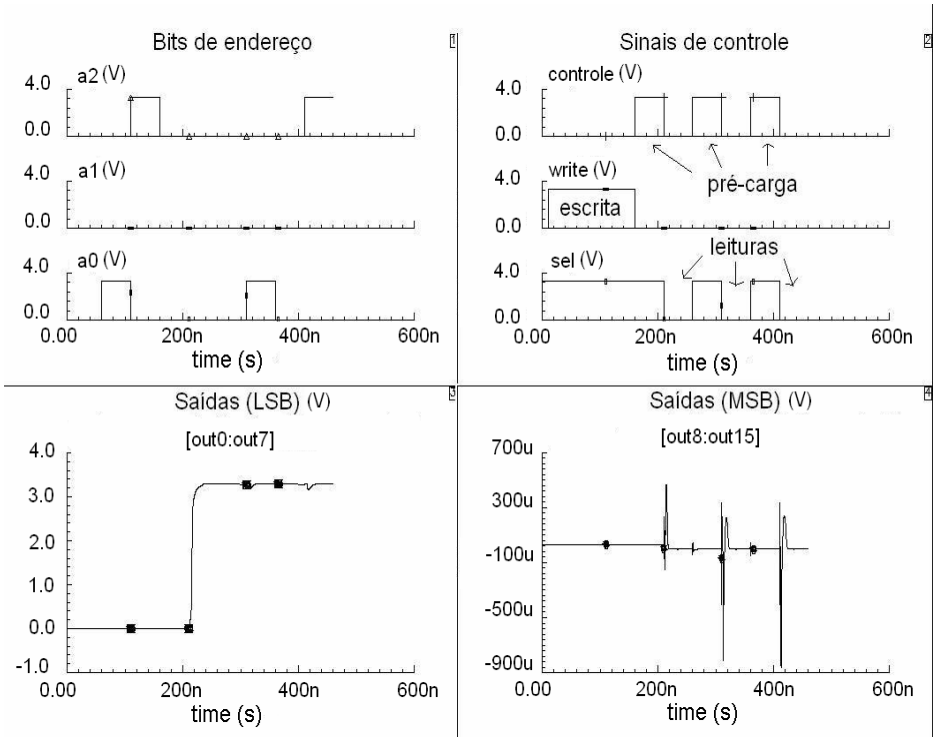


Fig. 6.21 – Simulação da SRAM sem estruturas de teste

6.7 TESTE DA SRAM

Para diminuir o número de pinos no *chip*, foi adicionado um conversor paralelo-serial de 16 bits na saída da matriz SRAM. Esse conversor está sendo testado na ROM, conforme foi descrito no capítulo anterior. Um dos tipos de teste em memórias é a seqüência de operações de escrita e leitura em todas as células (*march test*). Sendo assim, foram elaborados alguns vetores de teste para a SRAM projetada. A obtenção dos vetores e as tabelas com os valores escolhidos se encontram no Apêndice A.2.

A Figura 6.22 mostra o esquemático da SRAM final com estruturas de teste. O pino *Out_dec* é a saída do conversor paralelo-serial de 8 bits que foi ligado na saída do decodificador de 3 bits. Isso significa que, de maneira análoga ao teste do decodificador de 4 bits da ROM, as saídas do decodificador da RAM podem ser diretamente obtidas e verificadas na saída desse conversor.

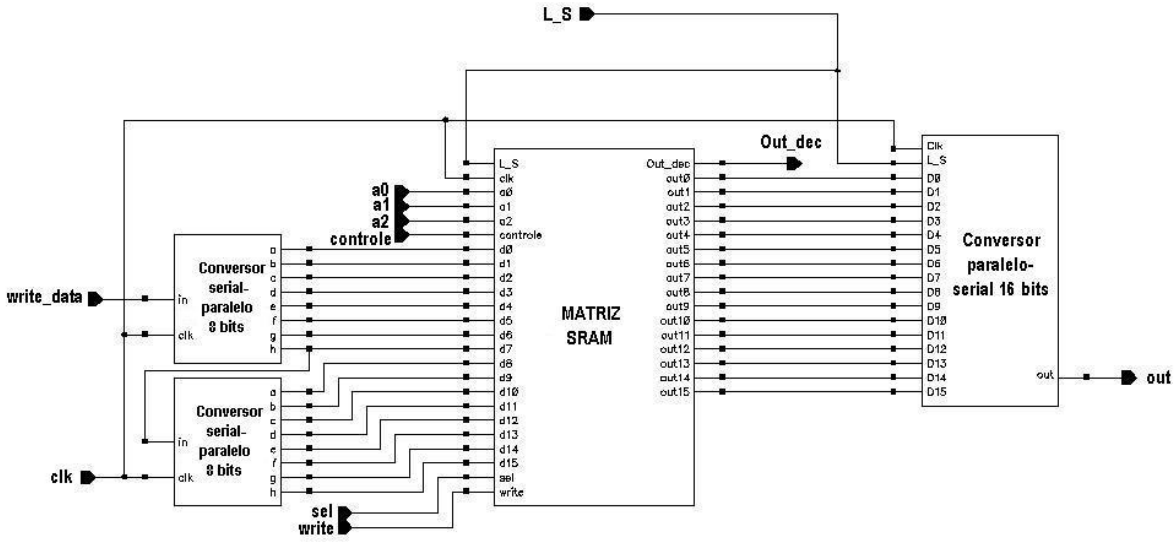


Fig. 6.22 - Esquemático da SRAM com estruturas de teste

Foi enviada também no *chip* uma única célula RAM completa, como a mostrada na Figura 6.19,cujos sinais podem ser controlados e observados por meio de *pads* internos (Figura 6.23). Essa estrutura de teste visa verificar se a célula está funcionando corretamente.

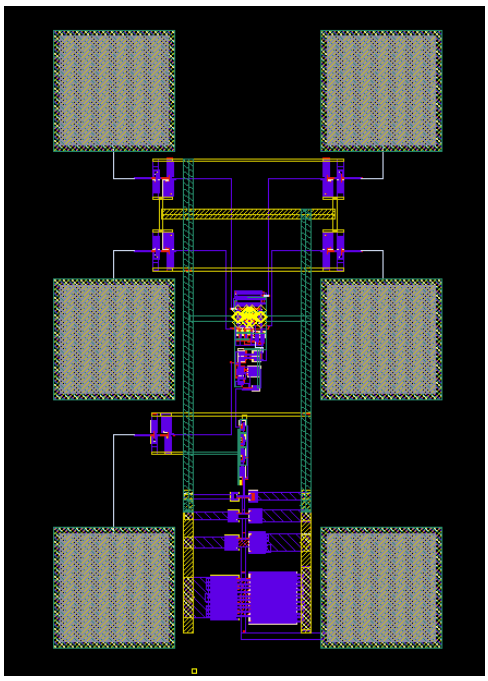


Fig. 6.23 - Célula RAM com *pads* internos

A simulação da SRAM com estruturas de teste encontra-se detalhada no Capítulo 9.

7 PROJETO DO BANCO DE REGISTRADORES

7.1 INTRODUÇÃO

Registradores são geralmente memórias RAM com múltiplas portas de entrada e saída. Foi previsto um banco com 16 registradores de 16 bits. Este capítulo apresenta a descrição dos registradores e do banco que foram projetados para o microprocessador RISC-16.

7.2 ESTRUTURA DE UMA CÉLULA DE UM REGISTRADOR

A Figura 7.1 mostra a estrutura da célula utilizada nos registradores. O circuito possui uma porta de escrita e duas de leitura. A porta de escrita apresenta uma estrutura simplificada onde o transistor de passagem (N1) é usado para drenar a fraca realimentação provida pelo inversor (N3, P3). O limiar de transição do inversor de armazenagem (N2, P2) é polarizado próximo a Vss através do aumento da largura de N2 com relação a P2, a fim de auxiliar a operação de escrita. Os inversores da célula alimentam um *buffer* (N4, P4), que, por sua vez, alimenta as linhas de leitura através dos transistores de passagem (N5, N6). Para cada porta de escrita acrescentada, um transistor e um decodificador são necessários. Uma vantagem dessa estrutura é que, independentemente da carga na saída do *buffer* formado pelos inversores P4 e N4, o estado da célula de armazenamento não muda [WESTE, 1993].

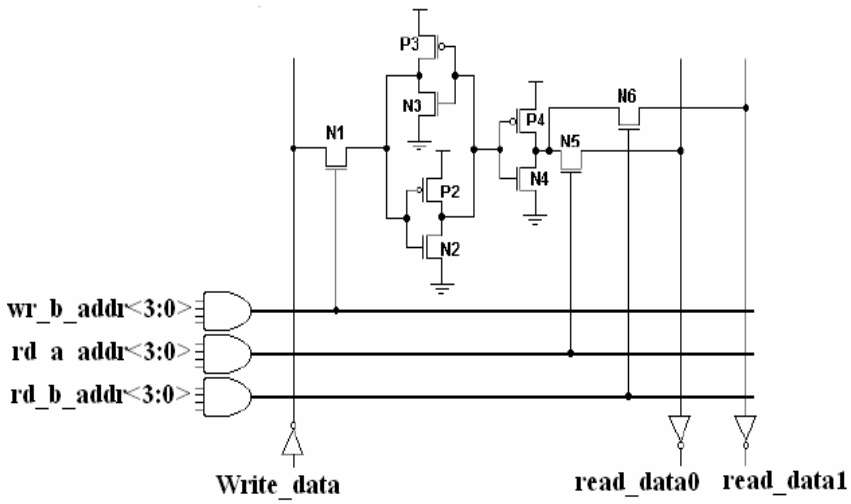


Fig. 7.1– Célula dos registradores [BENÍCIO, 2002]

Os valores de W dos transistores utilizadas na célula estão mostrados na Figura 7.2. Esses valores foram adaptados de [BENÍCIO, 2002], onde L= 0,8 um. Os valores na Figura 7.2 foram utilizados para L = 0,4 um.

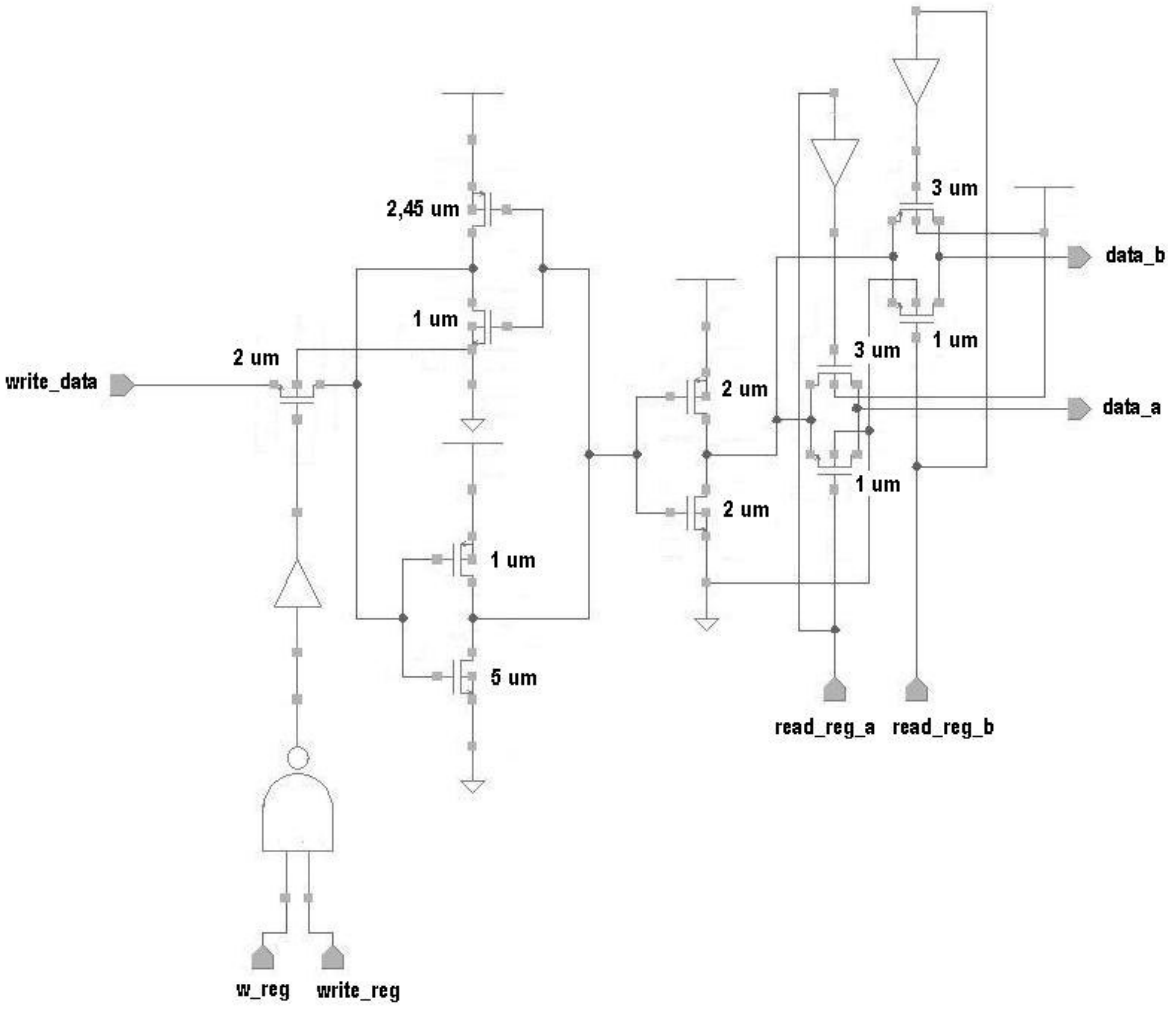


Fig. 7.2 - Valores de W para os transistores da célula

Tendo em vista a necessidade de interligar vários registradores entre si, foi adicionado um transistor p em cada uma das portas de saída da célula, ou seja, o acesso ao dado na operação de leitura se deu através de portas de transmissão CMOS, ao invés de transistores de passagem n, conforme foi originalmente proposto. A simulação dessa estrutura pode ser vista na Figura 7.3.

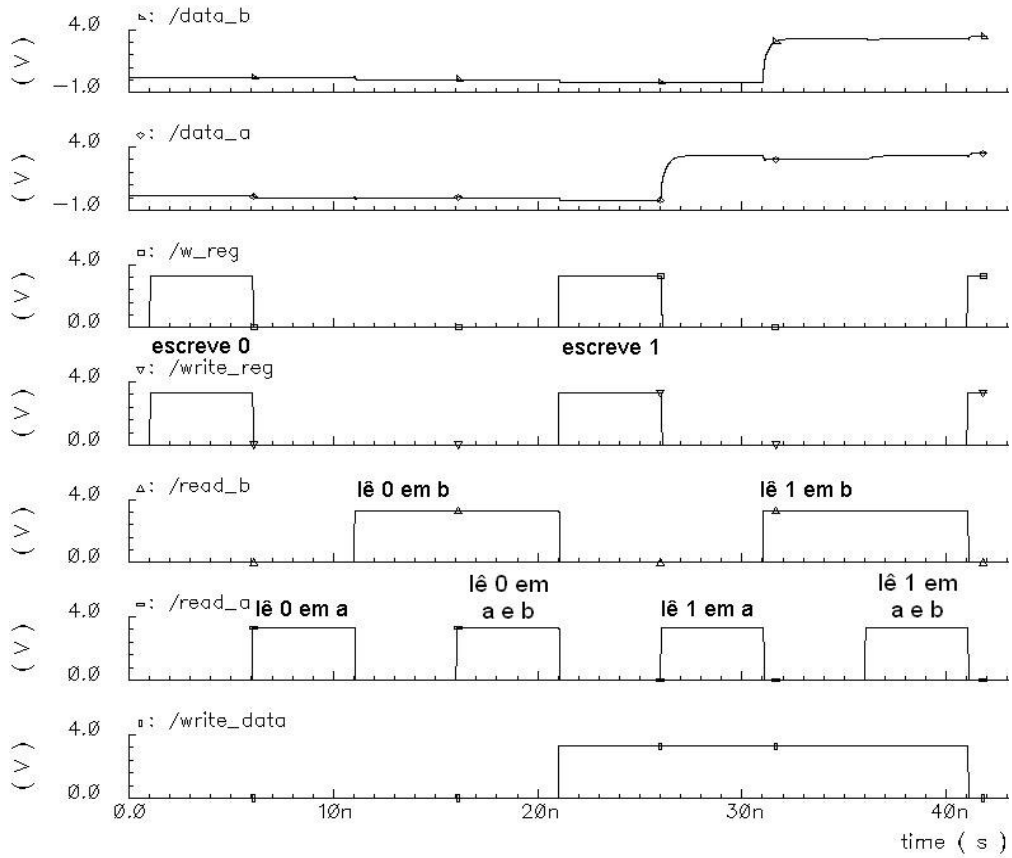


Fig. 7.3 - Simulação de uma operação de escrita e leitura em um registrador

Pode-se observar que os dados escritos podem ser lidos tanto em cada porta, separadamente, quanto em ambas as portas, simultaneamente.

7.3 REGISTRADOR DE 16 BITS

O registrador de 16 bits foi obtido juntando-se 16 registradores de um bit. Os sinais de controle *read_reg_a*, *read_reg_b*, *write_reg* e *w_reg* foram interligados entre si, conforme pode ser observado na Figura 7.4.

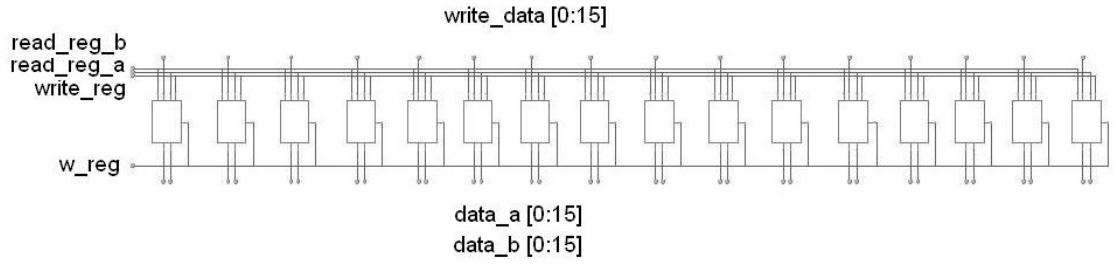


Fig. 7.4 - Registrador de 16 bits

A simulação da Figura 7.5 mostra as operações de escrita e leitura através das duas portas em um registrador de 16 bits. Os dados de entrada foram introduzidos bit a bit nas entradas *write_data* de cada registrador.

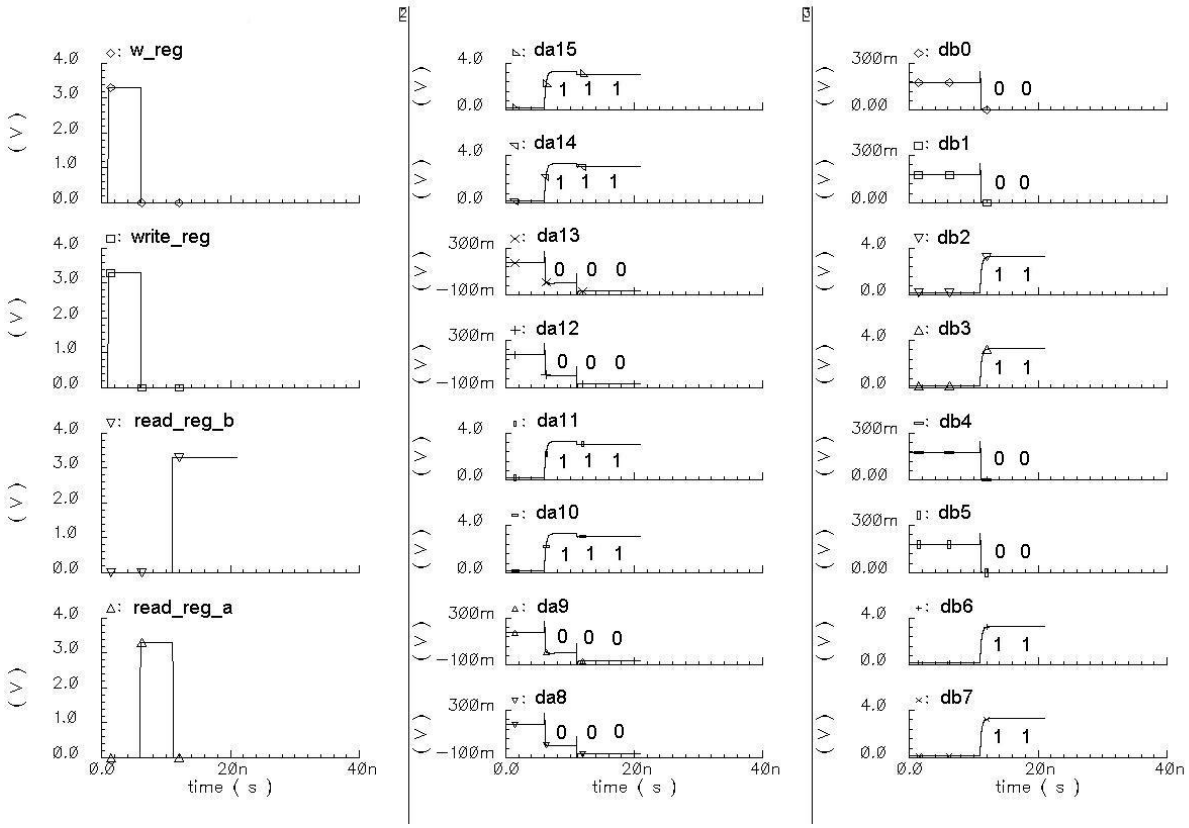


Fig. 7.5 - Simulação do registrador de 16 bits

Conforme pode ser observado na Fig. 7.5, primeiramente é feita uma escrita (*write_reg* = *w_reg* = 1) do dado 1100 1100 1100 1100. Em seguida, o dado é lido pela porta a (*read_reg_a* = 1), depois pela porta b (*read_reg_b* = 1), e, finalmente, por ambas as portas (*read_reg_a* = *read_reg_b* = 1).

7.4 BANCO DE REGISTRADORES

7.4.1 DESCRIÇÃO FUNCIONAL

O banco é formado por 16 registradores de 16 bits. Seu diagrama pode ser ilustrado pela Figura 7.6.

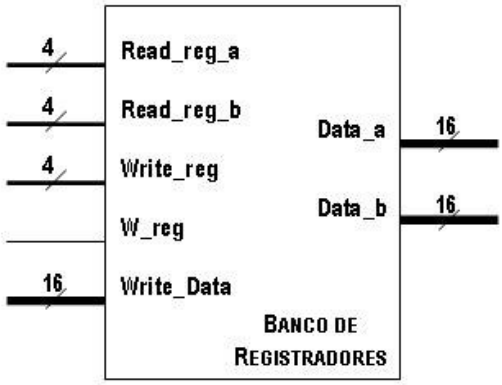


Fig. 7.6 - Diagrama do banco de registradores

Para integrar os 16 registradores entre si, foi necessária a inclusão de três decodificadores de 4 bits, para as entradas *write_reg*, *read_reg_a* e *read_reg_b*. O banco opera da seguinte forma:

Operação de escrita: Escolhe-se um dos 16 registradores para efetuar a escrita selecionando-se o endereço do mesmo na entrada do decodificador em *write_reg*. Além disso, o sinal *w_reg* deve estar em nível lógico alto para que a escrita possa ser realizada. Isso é feito pela unidade de controle do microprocessador, que atribuirá os valores dos sinais de acordo com a máquina de estados vista na Figura 4.8. Os dados a serem escritos no registrador selecionado em *write_reg* são carregados na entrada *write_data*. Conforme é mostrado no caminho de dados do processador (vide Figura 4.7), existe um multiplexador nessa entrada que definirá se os dados virão do registrador da ULA (*ULA_result*), do registrador de dados, ou da entrada ou saída do registrador PC.

Operação de leitura: O registrador a ser lido pode ser selecionado pelos decodificadores em *read_reg_a* ou *read_reg_b*, caso se queira que os dados estejam

disponíveis em *data_a* ou *data_b*, respectivamente. Isso é feito pelo registrador de instruções, no qual os campos do código da instrução que está sendo executada são separados. Os bits de 11 a 08 são ligados diretamente em *read_reg_a*. Um multiplexador na entrada *read_reg_b* define qual o campo da instrução será ligado nessa entrada (bits de 07 a 04 ou de 03 a 00), dependendo da condição do sinal *RegMem*. Ao contrário da escrita, na leitura o sinal *w_reg* deve estar em nível lógico baixo. Desde que isso aconteça, o valor na entrada *write_reg* é indiferente. Com isso, os dados são lidos e apresentados em uma ou em ambas as portas de saída *data_a* ou *data_b*, dependendo de onde a leitura foi solicitada.

7.4.2 SIMULAÇÃO DO CÓDIGO VHDL

Um banco de registradores nada mais é que um conjunto de registradores em que cada um deles pode ser lido ou escrito simplesmente especificando o número do registrador no banco. Nesse projeto, ele é composto por 16 registradores de 16 bits e possui um sinal de *reset* (*reset_A*), 5 entradas: *ReadRegA*, *ReadRegB*, *WriteReg*, *WriteData* e *WReg* e duas saídas: *DataA* e *DataB*. O sinal *reset_a* é utilizado para iniciar todos os registradores com zero. *ReadRegA* e *ReadRegB* são entradas que especificam o número dos registradores a serem lidos. Para escrever uma palavra no banco de registradores são necessárias duas entradas: uma para especificar o número do registrador a ser escrito (*WriteReg*) e outra para fornecer os dados a serem escritos no registrador (*WriteData*). O banco sempre coloca na saída quaisquer registradores que apareçam nas entradas *ReadRegA* e *ReadRegB*. Entretanto, a escrita é controlada pelo sinal, *WReg*, que deve estar ativo para que a escrita ocorra na transição do *clock*. Foram ainda acrescentados ao banco de registradores 16 saídas para permitir a visualização de seu conteúdo durante a simulação (*rzero, rum, rdois, rtres, rquatro, rcinco, rseis, rsete, roito, rnove, ra, rb, rc, rd, re, rf*).

A Figura 7.7 mostra a simulação obtida com o código explicado acima. O código completo está listado no Apêndice B.2.

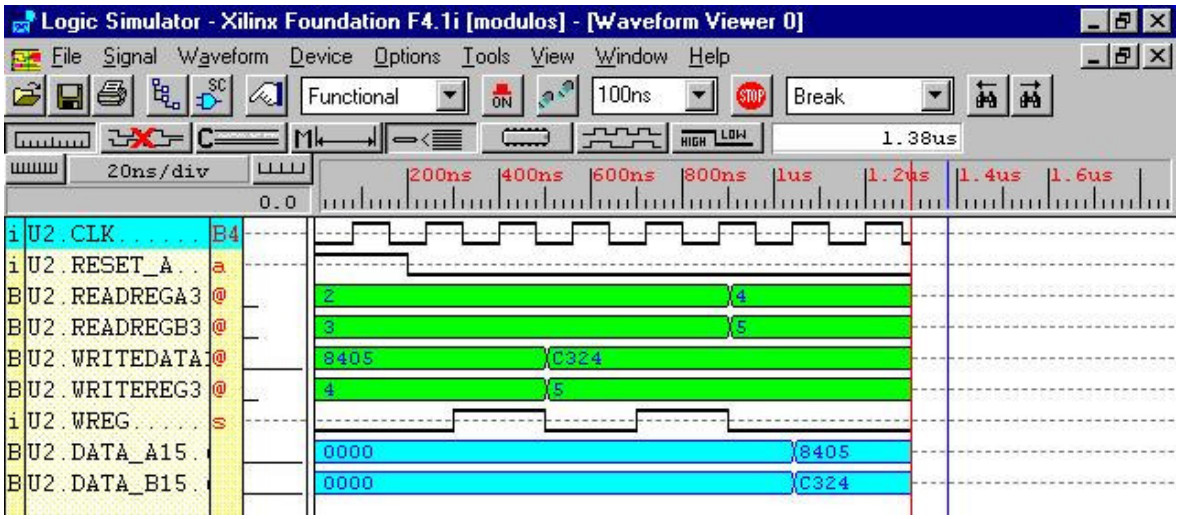


Fig. 7.7 - Simulação do banco de registradores

7.4.3 PROJETO ELÉTRICO

A Figura 7.8 mostra o esquemático final do banco de registradores. Pode-se observar que as saídas de todos os registradores são ligadas em dois barramentos comuns: data_a e data_b, daí a necessidade de se mudar as portas de acesso à leitura da célula por portas de transmissão CMOS, ao invés de transistores n de passagem, como estava originalmente previsto.

A simulação do circuito da Figura 7.8 pode ser observada na Figura 7.9. Foi feita primeiramente uma escrita no registrador 1111 (*write_reg* = 1111). Para simplificação, é mostrado apenas o bit mais significativo de *write_data* (*write_data[15]*). O bit menos significativo de *write_data* (*write_data[0]*) é o complemento de *write_data[15]*. Os outros bits de *write_data* são iguais a zero. Após a primeira escrita, é feita uma leitura em ambas as portas a e b do registrador 1111 (*read_reg_a* = *read_reg_b* = 1111). Observa-se que os bits mais e menos significativos de *data_a* são complementares entre sim (*data_a[15]* = 1 e *data_a[0]* = 0), e que o mesmo acontece com a porta b, de acordo com o esperado. Em seguida, inverte-se o *write_data[15]* (e, conseqüentemente, seu complemento, *write_data[0]*), e é feita uma escrita no registrador 0000 (*write_reg* = 0000). Por fim, é feita uma leitura no registrador 0000 em ambas as portas (*read_reg_a* = *read_reg_b* = 0000), e observa-se que, desta vez, *data_a[15]* = *data_b[15]* = 0 e *data_a[0]* = *data_b[0]* = 1.

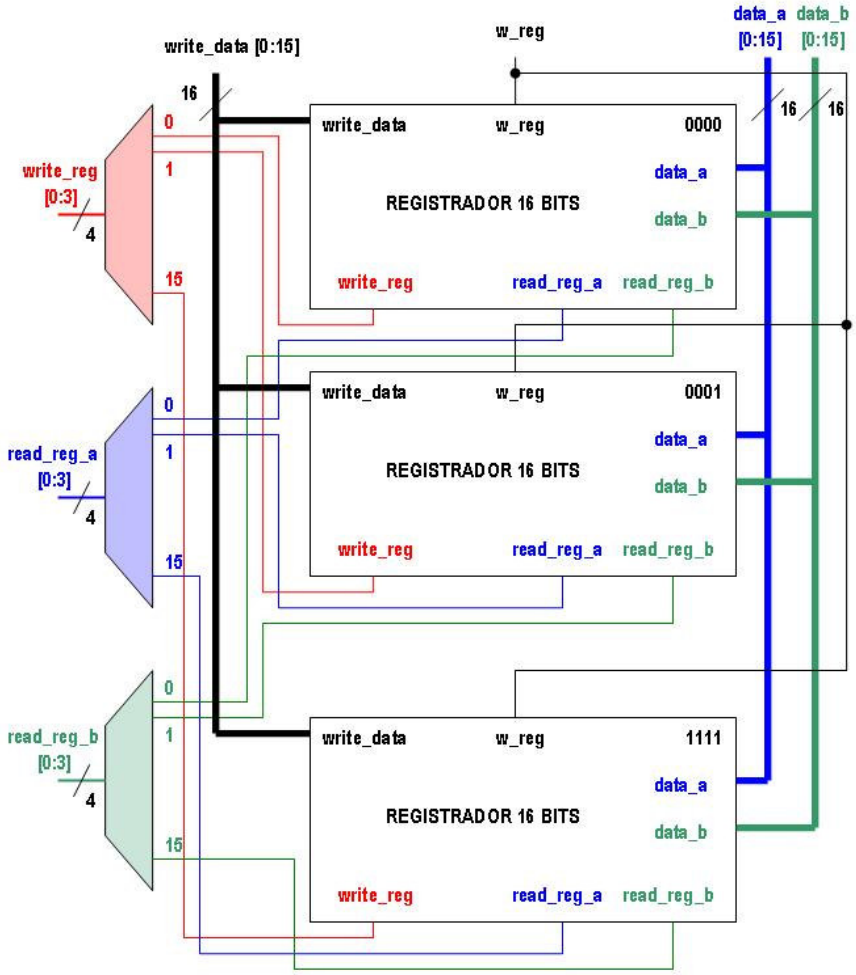


Fig. 7.8 - Esquemático do banco de registradores

Observações: Neste caso, foi realizada uma operação de escrita no registrador 0000 visando a uma maior simplicidade nas configurações das fontes durante a simulação. No caminho de dados do processador RISC-16, não é possível efetuar uma escrita nesse registrador, cujo valor está fixado em 0000_H. Para evitar que sejam realizadas operações de escrita nesse registrador, uma sugestão é, ao invés das células de registrador convencionais, utilizar portas de transmissão CMOS, controladas pela saída 0 do decodificador, que passariam o valor zero para o barramento *write_data* quando selecionadas.

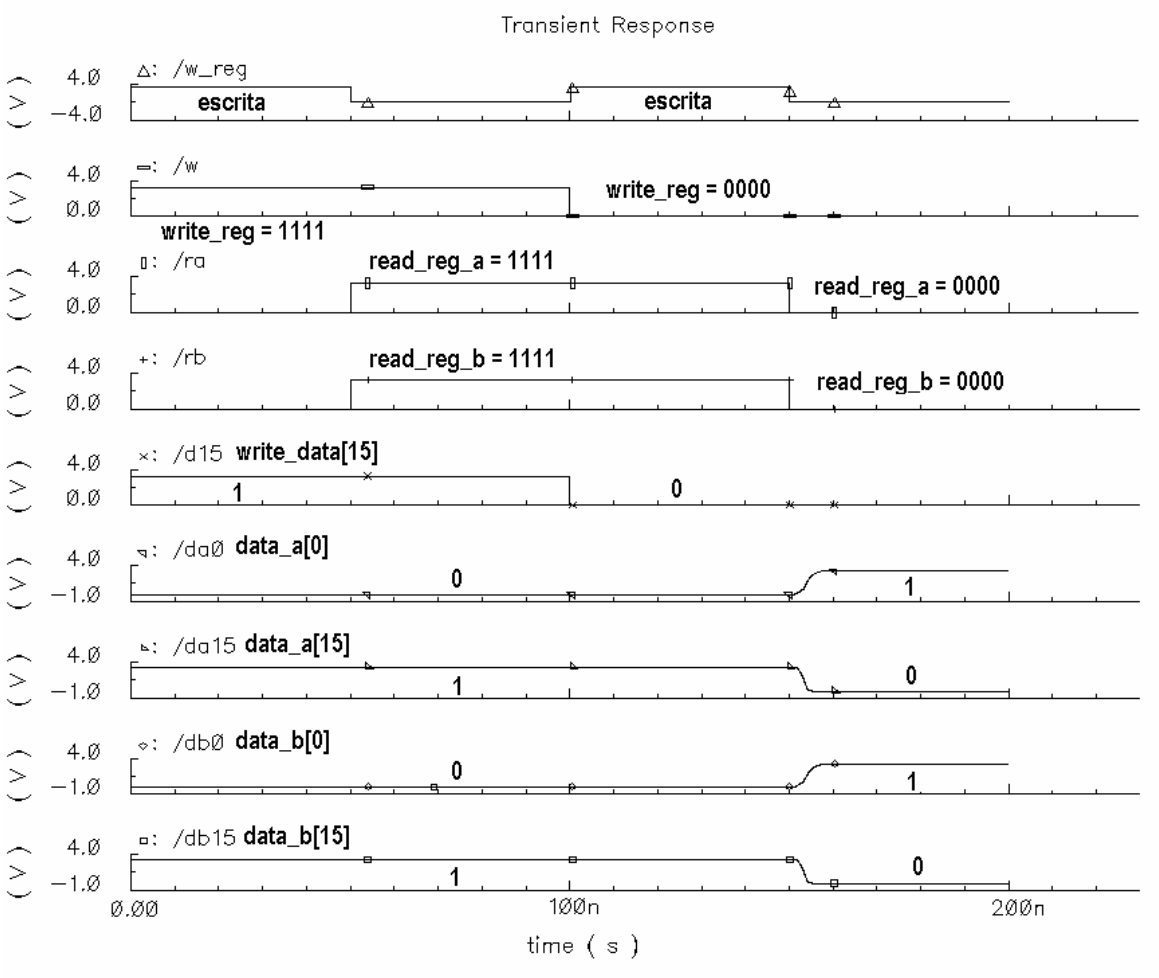


Fig. 7.9 - Simulação do banco

7.4.4 ESTRUTURAS DE TESTE

O banco de registradores projetado funciona em dois modos: o normal e o de teste. Foram adicionados conversores serial-paralelo e paralelo-serial para reduzir o número de pinos, dois multiplexadores nas saídas dos conversores conectados a *data_a* e *data_b*, e uma célula isolada de um registrador (vide Figura 7.10). Quando o sinal *teste_cell* é igual a zero, o banco funciona no modo normal, e as saídas *data_a* e *data_b* são obtidas serialmente através dos multiplexadores. Caso *teste_cell* seja igual a um, obtém-se as saídas *data_a* e *data_b* da célula isolada que foi acrescentada, de modo a se verificar se a mesma está funcionando corretamente. As simulações finais são detalhadas no Capítulo 9, e os vetores de teste estão mostrados no Apêndice A.3.

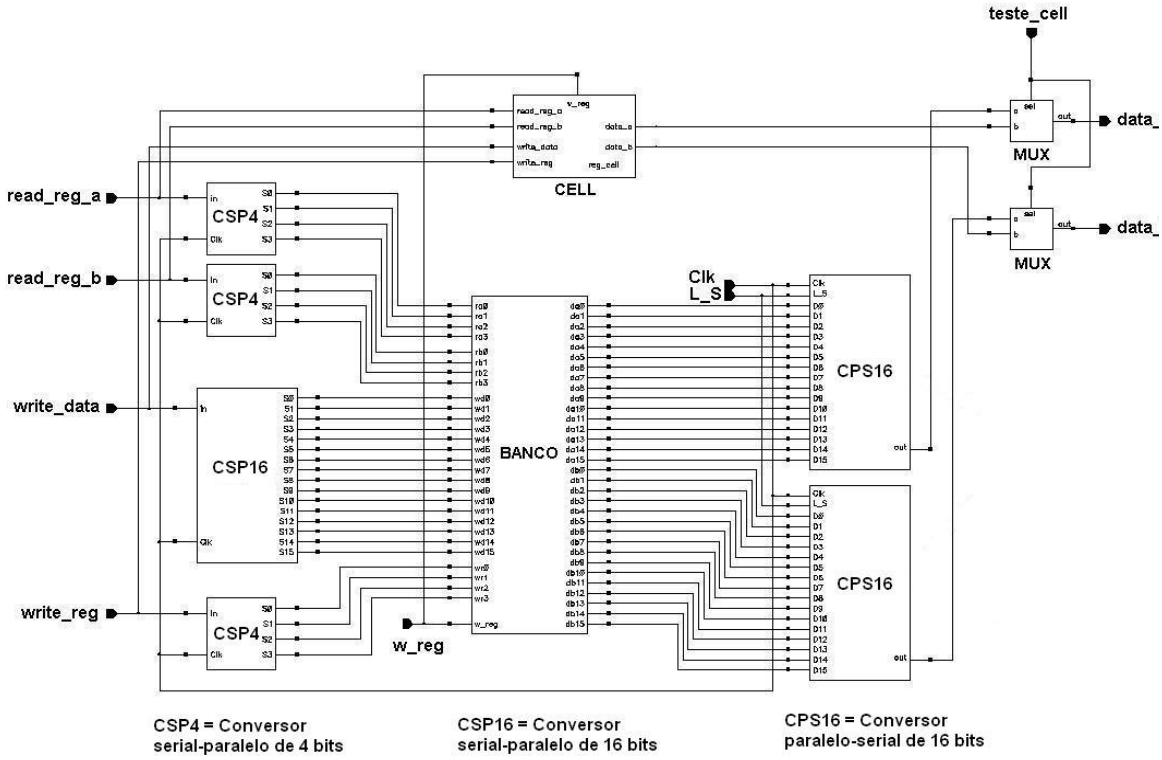


Fig. 7.10 – Estrutura de teste do banco de registradores

Foi enviado no *chip* um registrador de 16 bits com conversores paralelo-serial de 16 bits em suas saídas *data_a* e *data_b*, um conversor serial-paralelo de 16 bits na entrada *write_data*, e com seus sinais conectados a *pads* internos (Figura 7.11) para testar o funcionamento dessa estrutura, que será utilizada em outras partes do processador.

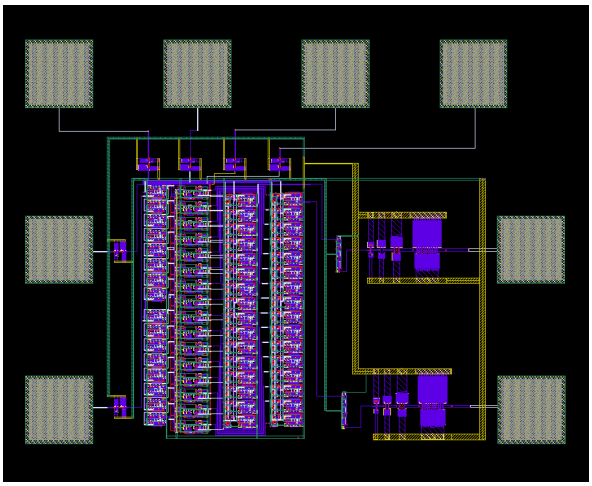


Fig. 7.11 - Registrador de 16 bits com *pads* internos

8 INTEGRAÇÃO DAS MEMÓRIAS

8.1 INTRODUÇÃO

As memórias ROM e RAM projetadas nos capítulos 5 e 6 constituem apenas blocos isolados para teste das estruturas escolhidas para compor as memórias de capacidade maior a serem utilizadas no SoC. Neste capítulo, será feita uma proposta de integração das memórias de maneira que as mesmas possam satisfazer as especificações do caminho de dados e serem corretamente utilizadas no mesmo.

8.2 ESPECIFICAÇÃO

Para o correto funcionamento do processador, foi decidida a integração das memórias de maneira que as mesmas pudessem ser endereçadas utilizando os mesmos decodificadores, ou seja, o mesmo barramento de endereços será utilizado tanto para a ROM quanto para a RAM. Sendo assim, as primeiras posições serão ocupadas pela ROM, de maneira que, na inicialização do processador, quando PC = 0000, o mesmo tenha acesso à rotina de inicialização que estará armazenada lá. Os endereços restantes correspondem à RAM, sendo que algumas posições da mesma estão reservadas para a comunicação com as interfaces do sistema e para o armazenamento de algumas *flags* necessárias ao programador. Nessas posições, serão utilizadas células de registradores ao invés de células 6-T, de maneira a se facilitar a operação de escrita, visto que o barramento de dados estará diretamente ligado à célula, ao invés de na entrada *write_data*, como acontece na escrita das posições convencionais.

Dessa forma, serão utilizados 13 bits de endereçamento para acesso à ROM e à RAM. A Figura 8.1 mostra a estrutura de endereçamento proposta, juntamente com os endereços reservados para a ROM, RAM e registradores.

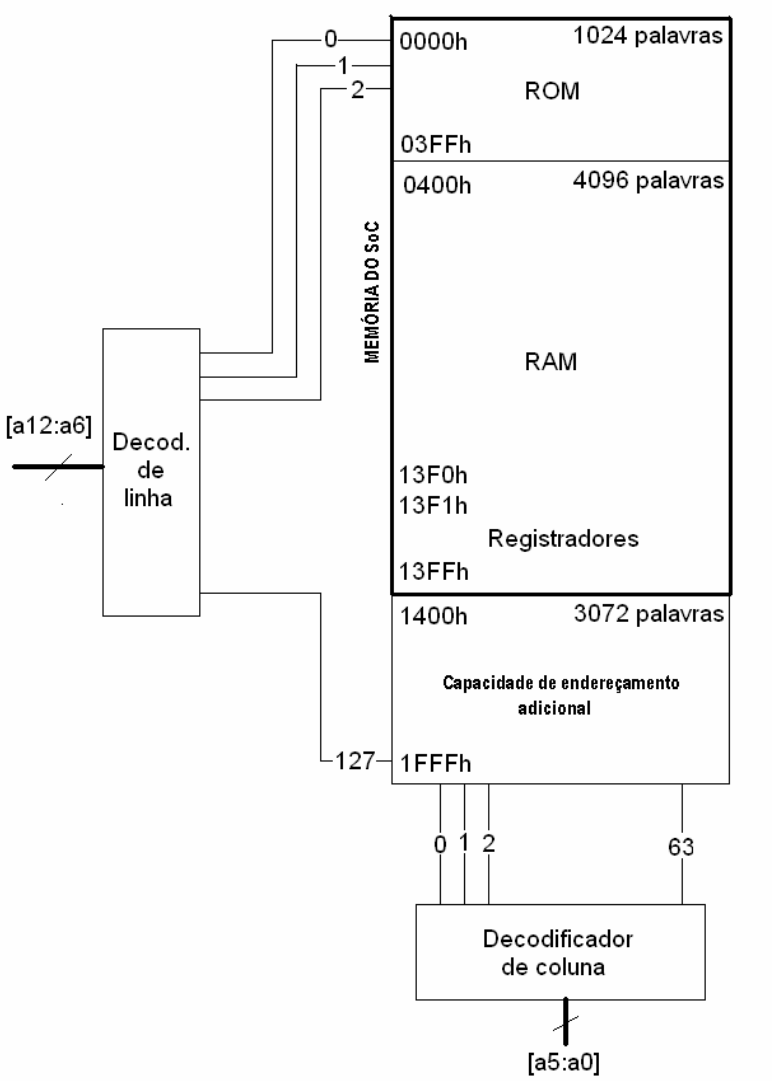


Fig. 8.1- Endereçamento das memórias

8.3 PROJETO ELÉTRICO

8.3.1 ROM 2 KB

A ROM tem como base a mesma estrutura utilizada no Capítulo 5, onde foi mostrado o projeto de uma ROM de 256 bits. Será necessário agora expandir a capacidade da mesma, visto que lá serão armazenadas as rotinas de inicialização do sistema. De acordo com a especificação inicial, a capacidade de 2kB é suficiente para tal propósito. Uma sugestão para a rotina de *boot* do sistema pode ser encontrada no Apêndice D.

O primeiro passo para se armazenar tal rotina na ROM é a decodificação da mesma para binário. Dessa forma, tais dados são armazenados na ROM de forma análoga à que foi mostrada no Capítulo 5, ou seja, onde há um transistor NMOS, significa que há um zero armazenado, e onde não há tal transistor, há um 1 armazenado.

Como tal rotina ainda não é definitiva, e como há 1024 palavras disponíveis na ROM de 2kB, as posições do esquemático implementado estão armazenando 0000_H até que todo o conteúdo da ROM a ser enviada para fabricação seja definido. As palavras foram divididas em 16 linhas, sendo que cada linha possui 64 colunas, e cada coluna comporta uma palavra de 16 bits. A Figura 8.2 mostra a disposição descrita acima.

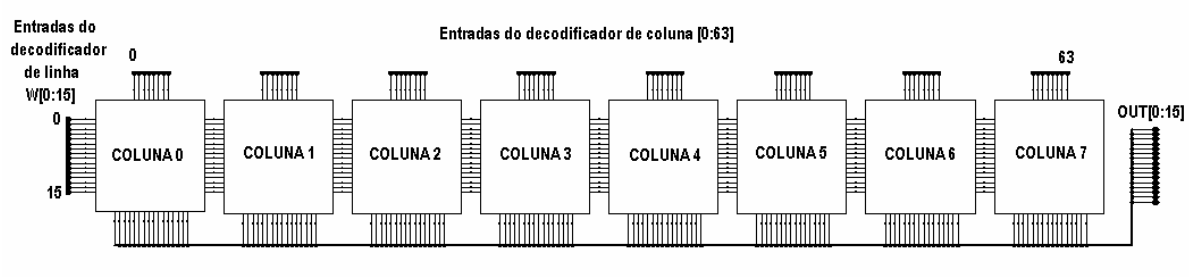


Fig. 8.2 - Esquemático da ROM 2 kB

Como o barramento de saída da memória será comum, foram adicionados *buffers tri-state* na saída da ROM, de maneira que o circuito da mesma fique isolado durante operações com a RAM ou com os registradores. A Figura 8.3 mostra o esquemático do *buffer* escolhido.

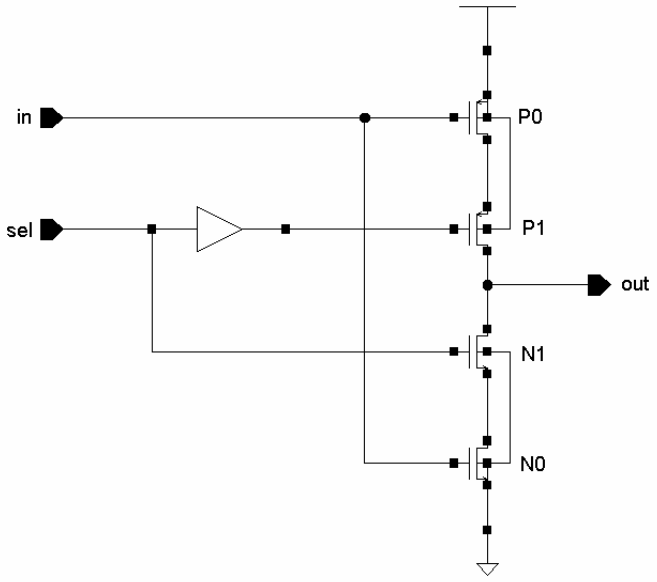


Fig. 8.3 – *Buffer tri-state*

8.3.2 RAM 8 KB E REGISTRADORES

As posições seguintes serão referentes à memória RAM, cuja estrutura é similar à apresentada no Capítulo 6. As células foram dispostas em 64 linhas, sendo que cada linha, da mesma forma que na ROM, possui 64 colunas, sendo cada coluna com uma palavra de 16 bits. A Figura 8.4 mostra a disposição descrita acima.

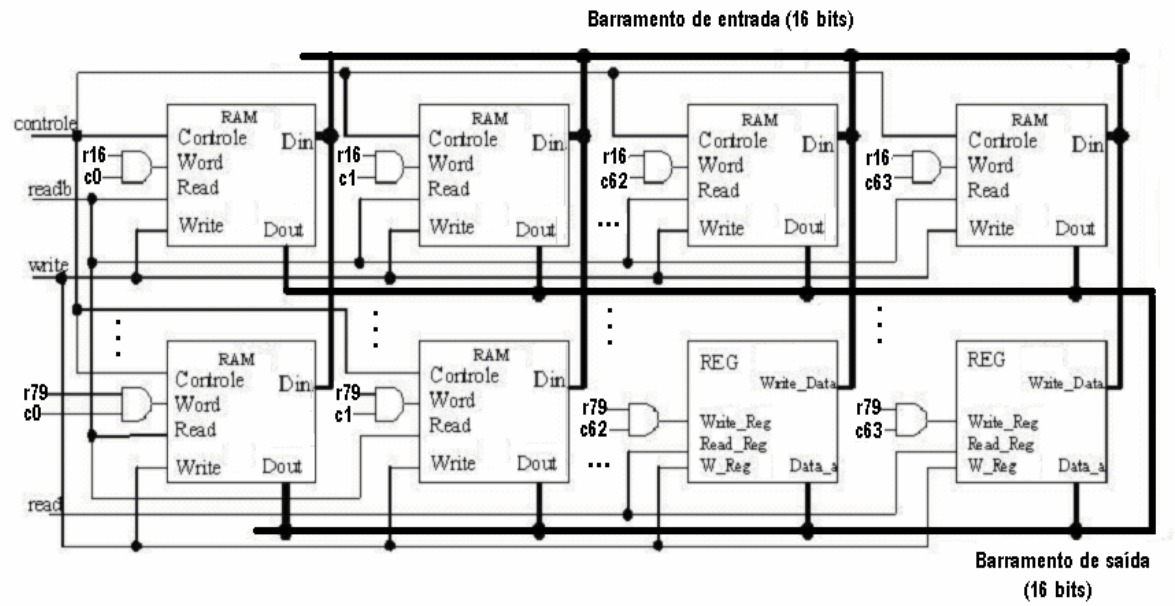


Fig. 8.4 - Esquemático da RAM 8 kB

No esquemático da Figura 8.4, a saída *write_data* dos registradores está conectada ao barramento de entrada. Entretanto, na integração do SoC, as saídas de cada registrador serão diretamente conectadas às interfaces e aos outros locais de origem dos dados que devem ser armazenados, como, por exemplo, às *flags* da ULA para o registrador RegStatus (13FF).

Com o propósito de isolar o circuito da RAM durante as operações feitas na ROM, foram adicionadas portas de transmissão na saída da RAM.

Foram incluídos 15 registradores cujo endereçamento será feito pela memória. Tais registradores estão mapeados nas últimas posições da memória RAM, e seus endereços estão especificados na Tabela 8.1, assim como suas descrições. Esta tabela apresenta uma proposta de endereçamento dos registradores baseada na que foi feita na Tabela 4.5. Apenas foram acrescentados os registradores RegInt, IntCausa e RegStatus, e, como o barramento de endereço da memória possui apenas 13 bits, os endereços dos registradores de I/O foram alterados, já que os endereços antigos só poderiam ser acessados se o

barramento de endereço da memória tivesse 16 bits.

Tabela 8.1 - Proposta de endereços dos registradores mapeados em memória

Endereço(H)	Registrador	Descrição
13F1	Interface A/D	Dados (recepção)
13F2	Interface A/D	*Status*
13F3	Interface A/D	*Setup*
13F4	Interface serial	Dados (recepção)
13F5	Interface serial	Dados (transmissão)
13F6	Interface serial	*Status*
13F7	Interface serial	*Setup*
13F8	Interface RF	Dados (recepção)
13F9	Interface RF	Dados (transmissão)
13FA	Interface RF	*Status*
13FB	Interface RF	*Setup*
13FC	Interface RF	*Setup*
13FD	RegInt	Guarda o endereço da instrução que estava sendo executada quando a interrupção/ exceção foi requisitada
13FE	IntCausa	Guarda o tipo de interrupção
13FF	RegStatus	Armazena as *flags* do sistema

Conforme descrito no Capítulo 7, a célula de um registrador possui uma porta de escrita e duas de leitura. Os registradores que serão mapeados em memória foram adaptados, de forma que será necessária apenas uma porta de leitura. A maneira como os mesmos serão ligados na memória RAM está ilustrada na Figura 8.5, onde Rn e Cn são as saídas dos decodificadores correspondentes à linha n e à coluna n, respectivamente, e Din corresponde ao local de origem dos dados a serem armazenados no registrador em questão.

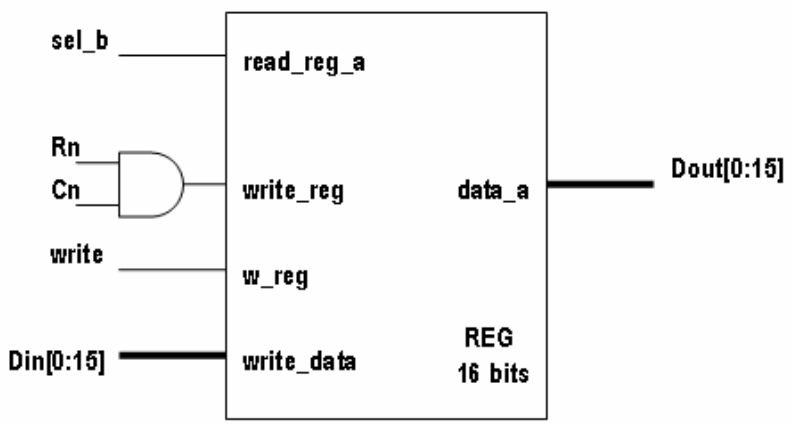


Fig. 8.5 – Registradores na RAM

8.3.3 DECODIFICADORES DE LINHA E COLUNA

Será necessário endereçar 1024 palavras da ROM e 4096 palavras da RAM, isto é, 5120 palavras de 16 bits. Dessa forma, torna-se necessário utilizar uma disposição bidimensional das posições da memória, ou seja, a mesma será dividida em linhas e colunas. Esta seção descreve uma alternativa para os decodificadores de linha e de coluna.

8.3.3.1 Decodificador de coluna

Conforme pode ser visto na Figura 8.1, uma das possibilidades é utilizar um decodificador de coluna de 6 bits. Uma alternativa é proposta em [HODGES, 2003]utilizando portas NAND. Entretanto, seria necessário utilizar portas NAND com 6 entradas. Sabe-se que portas com mais de 3 ou 4 entradas causam grandes atrasos. Sendo assim, é preferível utilizar portas lógicas em cascata. Isso pode ser feito utilizando dois estágios: o pré-decodificador, que gera os sinais intermediários, e o estágio final do decodificador.

Uma porta NAND de 6 entradas pode ser implementada utilizando 3 portas NAND de duas entradas, e 1 porta NAND de 3 entradas, como mostra a Figura 8.6.

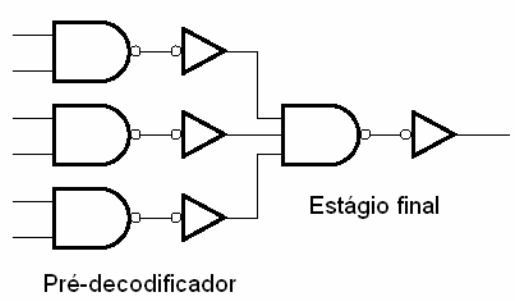


Fig. 8.6 – Alternativa para implementação da NAND6 [HODGES, 2003]

Sendo assim, um decodificador de 6 bits pode ser implementado da seguinte forma: no pré-decodificador, são gerados 12 sinais intermediários a partir dos bits de endereço e seus complementos (*A0A1, A0A1b, A0bA1, A0bA1b, A2A3, A2A3b*, etc.). Esses sinais podem agora ser usados pelo estágio final para gerar as 64 saídas requeridas utilizando combinações NAND de 3 entradas com inversores. Cada saída do pré-decodificador aciona 16 portas NAND (ou seja, 64 NAND x 3 entradas cada/12 saídas intermediárias). A Figura 8.7 mostra o esquemático do decodificador.

8.3.3.2 Decodificador de linha

O decodificador de linha visto na Figura 8.1 é de 7 bits, e pode ser implementado de maneira análoga à do decodificador de coluna. A diferença é que a porta NAND de 7 entradas pode ser implementada conforme mostra a Figura 8.8.

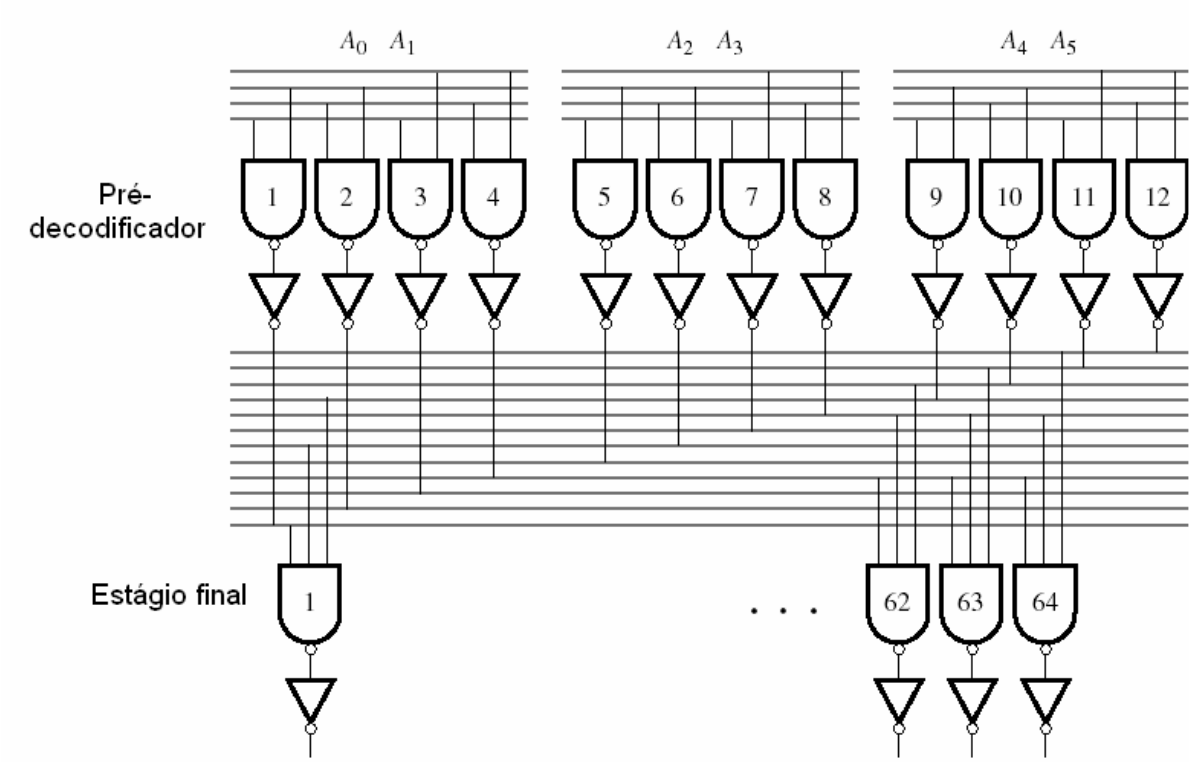


Fig. 8.7 – Decodificador de coluna (6 bits) [HODGES, 2003]

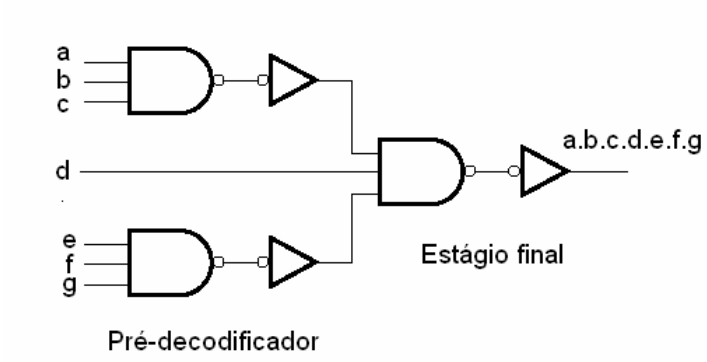


Fig. 8.8 - Alternativa para implementação da NAND7 [HODGES, 2003 – alterada]

8.3.4 ERRO DE ENDEREÇAMENTO NA MEMÓRIA

Na Figura 8.1 observa-se que, como o endereçamento é feito com 13 bits, existem 3072 palavras de 16 bits que poderiam ser endereçadas, mas não serão utilizadas. Sendo assim, cada vez que uma dessas posições não utilizadas for selecionada pelo barramento de endereços, isso caracterizará um erro de endereçamento na memória.

Uma estrutura que detecta tal erro pode ser implementada com portas XOR, conforme será descrito a seguir.

O último endereço válido é $13FF_H$, ou seja, os endereços inválidos são de 1400_H a $1FFF_H$. Em binário:

0001 0011 1111 1111
0001 0100 0000 0000
0001 0100 0000 0001
0001 0100 0000 0010
...
1111 1111 1111 1111

Pode-se, então, utilizar os bits b15, b14, b13, b11 e b10 nas portas XOR para gerar o sinal de erro. Se qualquer um desses sinais for 1, então *erro = 1*. A Figura 8.9 mostra o circuito para geração desse sinal.

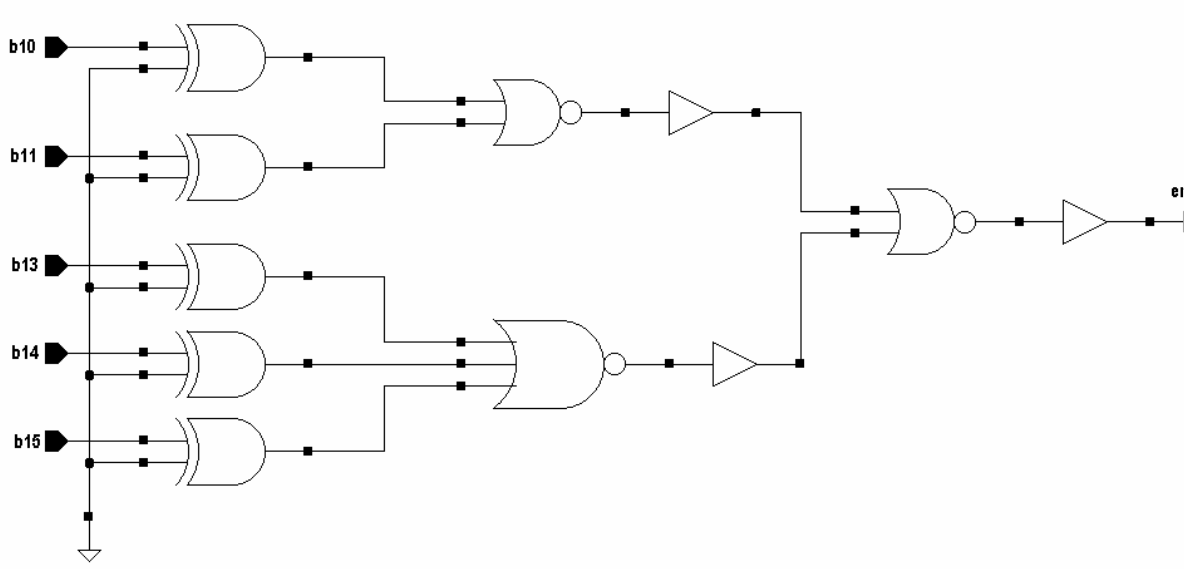


Fig. 8.9 - Circuito para geração do sinal de erro de endereçamento

8.4 VALIDAÇÃO DA ESTRUTURA

Foi mostrado que a unidade de memória a ser utilizada pelo processador será composta por células da ROM, RAM e de registradores. Para testar essa estrutura, está sendo simulada no SpectreS do CADENCE uma memória completa, totalizando 192 bits. São utilizados 4 bits de endereçamento, e dois decodificadores (linha e coluna) para acessar as células. As palavras armazenadas em cada posição são de 16 bits. Os 4 bits de endereçamento são suficientes para acessar até 2^4 posições, o que totalizaria 256 bits. Entretanto, escolheu-se armazenar apenas 192 bits, e deixar as 4 posições restantes vazias, para simular a estrutura que detectará o erro de endereçamento. A Figura 8.10 mostra o esquemático completo da estrutura em questão. Esta etapa ainda se encontra em andamento.

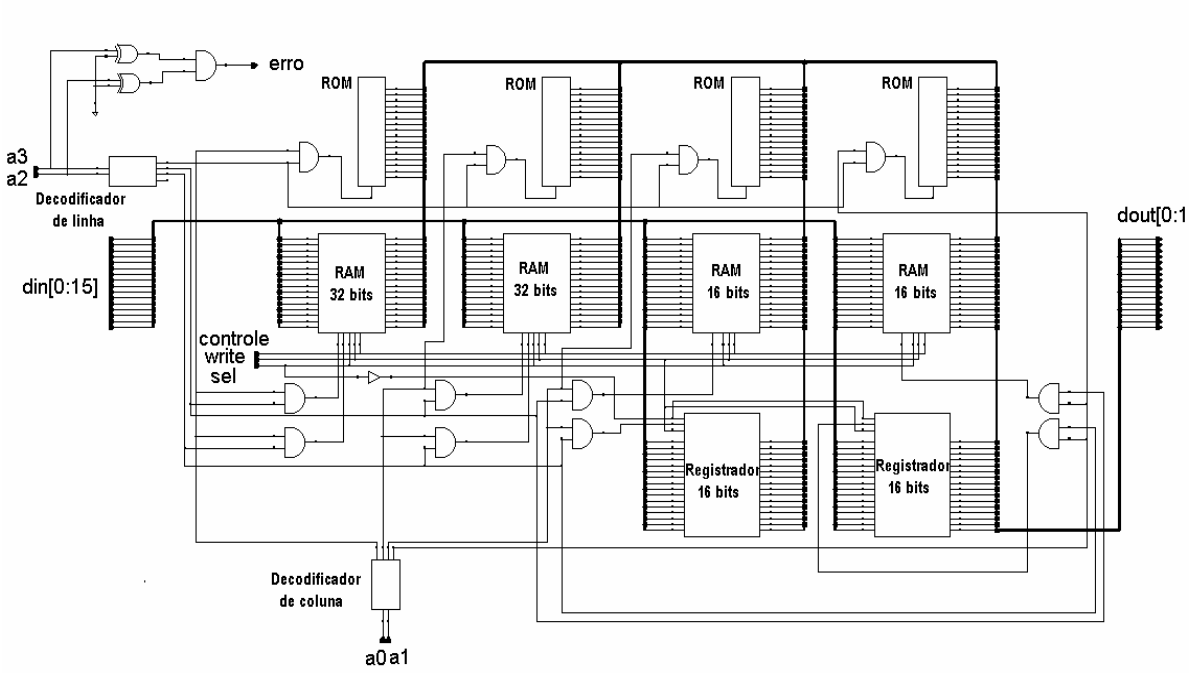


Fig. 8.10 – Estrutura para teste preliminar dos blocos da memória

Observa-se que as saídas dos decodificadores de linha e de coluna são conectadas às células por meio de portas AND, e que o circuito que gera o sinal de erro utiliza portas XOR, da mesma forma que o circuito proposto no item 8.3.4. Observa-se ainda que o barramento de saída é comum, daí a necessidade de se isolar as saídas da ROM e da RAM por meio de *buffers tri-state* e portas de transmissão, respectivamente.

Existem outras alternativas para essa implementação, e deve ser feito um estudo

posterior para estabelecer um compromisso entre área ocupada e velocidade, o que pode resultar em uma estrutura diferente. Por exemplo, nesse caso, há 16 amplificadores sensores para cada coluna, o que torna o circuito muito grande. Uma alternativa diferente seria utilizar apenas 16 amplificadores sensores e selecionar as colunas utilizando portas NMOS ou de transmissão em cada linha de bit.

9 DISCUSSÃO DOS RESULTADOS

Neste capítulo são apresentados e analisados os resultados obtidos durante o projeto das estruturas de armazenamento que integram o SoC do sistema de irrigação. O *chip* de teste implementado para validar e caracterizar as estruturas descritas nos Capítulos 5, 6 e 7 e as simulações realizadas no decorrer de seu desenvolvimento são também discutidos.

9.1 O *CHIP* DE TESTE

O *chip* de teste enviado para fabricação foi realizado em tecnologia CMOS 0,35µm com quatro níveis de metal. Algumas das estruturas digitais e analógicas que compõem o sistema de irrigação fazem parte do mesmo. O objetivo dessa etapa foi verificar o funcionamento individual das partes antes da integração das mesmas na versão final do sistema, que será posteriormente enviada. A Figura 9.1 apresenta seu *layout* e as áreas ocupadas por cada módulo.

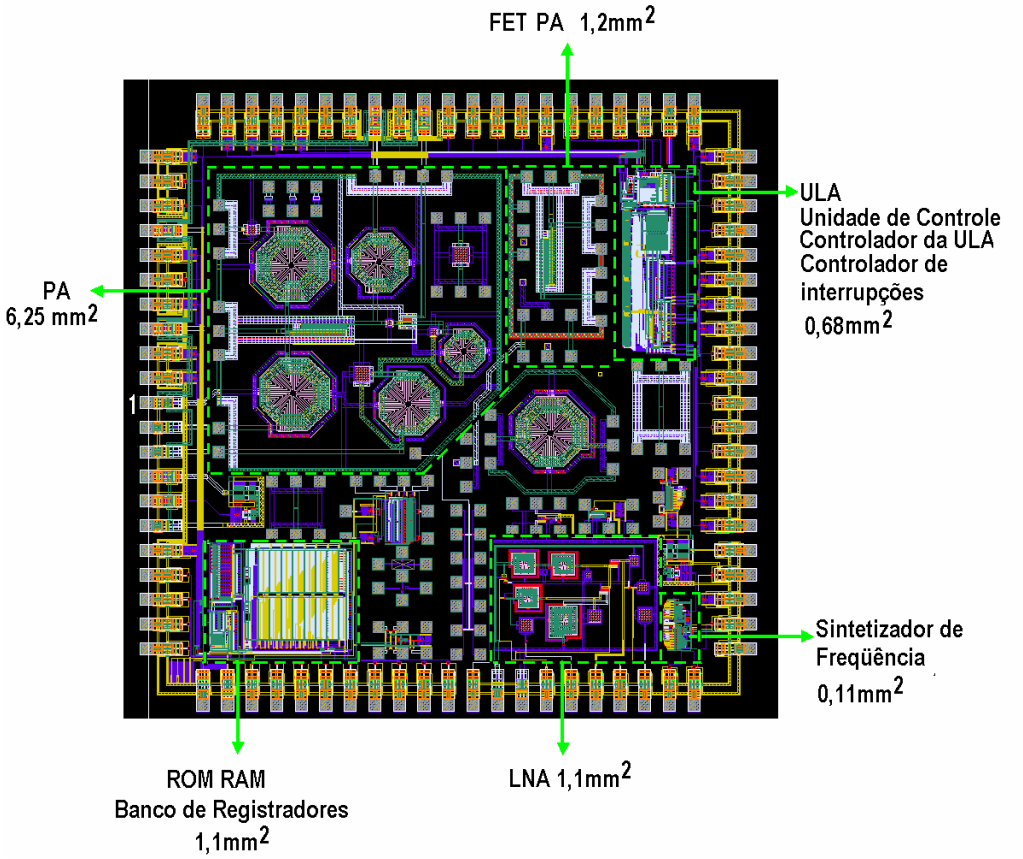


Fig. 9.1 - *Layout* do *chip* de teste (área total de 25mm^2)

Conforme pode ser observado na Figura 9.1, foram incluídos no *chip* o banco de registradores e as memórias ROM e RAM. De acordo com o que foi exposto nos Capítulos 5 e 6, as memórias foram desenhadas com uma capacidade menor para que se pudesse testar as suas estruturas. A capacidade de armazenamento extremamente reduzida foi escolhida também devido a limitações de área no *chip*. Essas estruturas foram reunidas em um bloco denominado BANCO_ROM_RAM e ocupam uma área de 1,1mm^2, sendo que apoximadamente 0,14mm^2 são destinados às estruturas de teste. Apesar de serem responsáveis por 12,73% da área, essas estruturas melhoram a observabilidade e a controlabilidade do circuito sem aumentar significativamente o número de pinos de teste.

A Figura 9.2 apresenta o esquemático do módulo que integra as estruturas acima listadas.

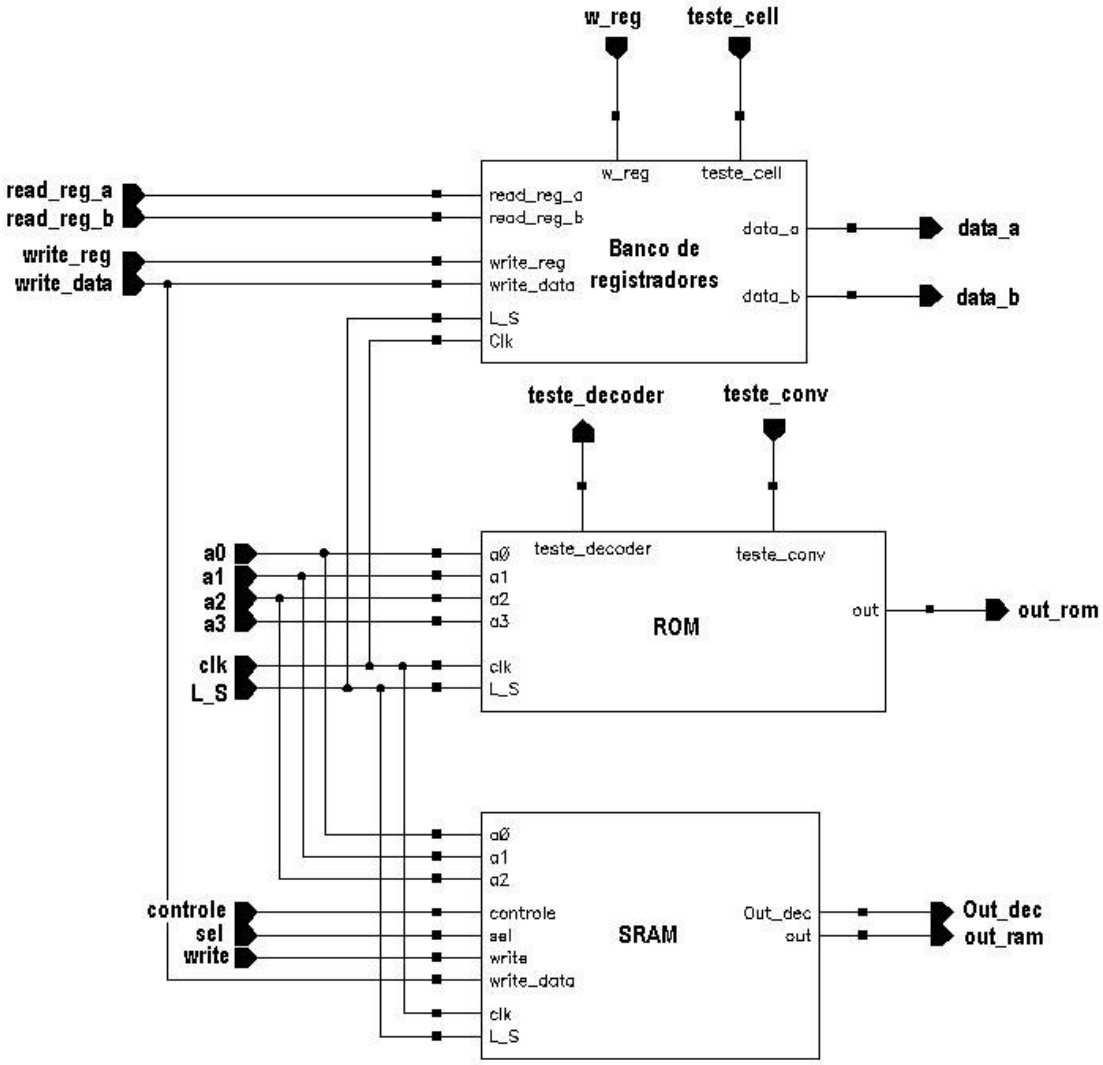


Fig. 9.2 - Esquemático do módulo BANCO_ROM_RAM

Foram utilizados 16 pinos de entrada e 6 de saída para caracterizar o módulo da Figura 9.2. Suas descrições são apresentadas na Tabela 9.1.

Tabela 9.1 - Pinos de entrada e saída do módulo

PINO	DIREÇÃO	DESCRIÇÃO
read_reg_a	Entrada	Endereço de 4 bits do registrador cujo conteúdo será lido e apresentado na saída *data_a*
read_reg_b	Entrada	Endereço de 4 bits do registrador cujo conteúdo será lido e apresentado na saída *data_b*
write_reg	Entrada	Endereço de 4 bits do registrador no qual o dado em *write_data* será escrito
write_data	Entrada	Dado de 16 bits de entrada do banco de registradores e da memória
w_reg	Entrada	Sinal que habilita a escrita no banco de registradores quando igual a 1
teste_cell	Entrada	Seleção do multiplexador que permitirá que as saídas *data_a* e *data_b* de uma única célula do registrador sejam apresentadas nas saídas dos conversores paralelo-serial
data_a	Saída	Saída a de 16 bits do banco de registradores
data_b	Saída	Saída b de 16 bits do banco de registradores
out_rom	Saída	Saída de 16 bits na qual são apresentados os dados lidos da ROM (*teste_conv* = 0) ou uma combinação dos bits de endereço (*teste_conv* = 1)
Clk	Entrada	Sinal de *clock*. Comum para todas as estruturas
L_S	Entrada	Controla o carregamento (*Load*) e o deslocamento (*Shift*) dos dados colocados na entrada dos conversores paralelo-serial.
a3	Entrada	Bit mais significativo de endereçamento da ROM
a2	Entrada	Segundo bit mais significativo de endereçamento da ROM, e bit mais significativo de endereçamento da RAM
a1	Entrada	Terceiro bit mais significativo de endereçamento da ROM, e segundo bit mais significativo de endereçamento da RAM
a0	Entrada	Bit menos significativo da ROM e da RAM
teste_decoder	Saída	Saída serial de 16 bits do decodificador de linha da ROM. Testa o correto funcionamento do decodificador de 4 $\rightarrow$16 bits
teste_conv	Entrada	Sinal de seleção dos multiplexadores que ora deixam passar a saída da ROM (*teste_conv* = 0), ora uma combinação dos bits de endereço da ROM (*teste_conv* = 1). Testa o conversor paralelo-serial
controle	Entrada	Sinal de pré-carga da memória RAM. Deve ser igual a 1 durante um ciclo de *clock* antes da operação de leitura
sel	Entrada	Sinal que habilita os amplificadores sensores quando igual a 0
write	Entrada	Sinal que habilita a operação de escrita na memória RAM quando igual a 1
Out_dec	Saída	Saída serial dos 8 bits de saída do decodificador de linha da RAM
out_ram	Saída	Saída serial de 16 bits da memória RAM

9.2 SIMULAÇÕES FINAIS

9.2.1 ROM

Conforme foi explicado no Capítulo 5, a ROM funciona em dois modos. No modo normal, o sinal de seleção dos multiplexadores *teste_conv* é igual a zero e, portanto, a saída *out* mostra os dados lidos na matriz da ROM. No modo teste, a saída *out* mostra uma combinação dos bits de endereço, de maneira a se poder verificar se o conversor paralelo-serial de 16 bits funciona de acordo com o esperado. A Figura 9.3 mostra a simulação final da ROM com *teste_conv* = 0.

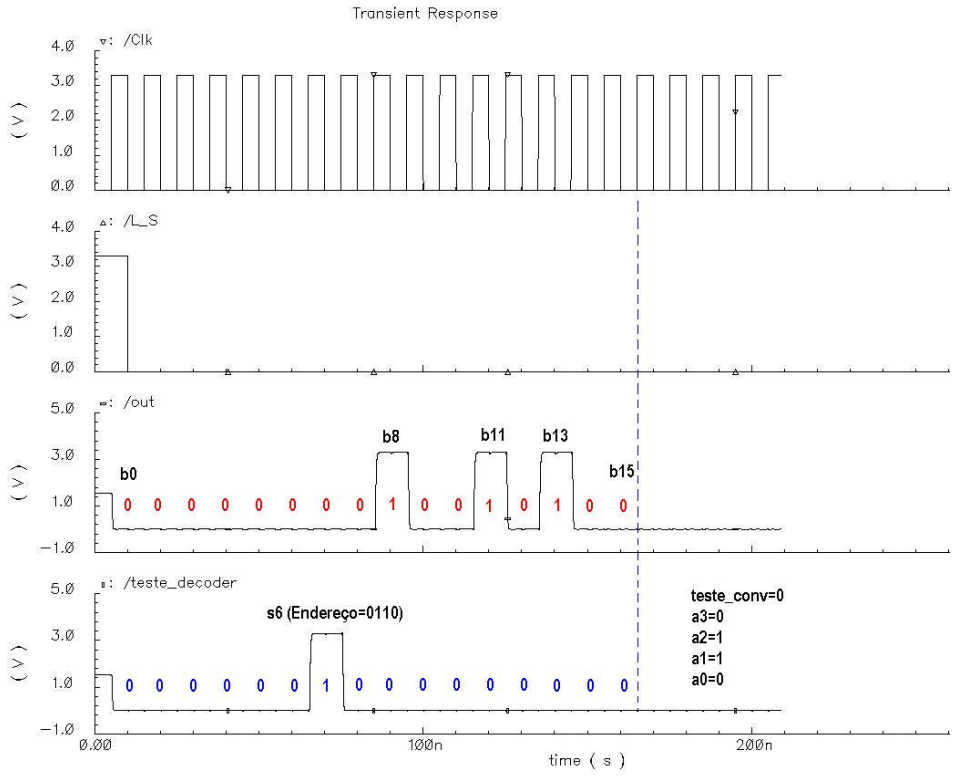


Fig. 9.3 - Simulação final da ROM para *teste_conv* = 0

Neste caso, foi feita uma leitura do dado armazenado na posição 0110 da ROM. De acordo com a Tabela 5.1, o dado é 0010 1001 0000 0000 (2900_H). Observa-se que o mesmo é mostrado na saída *out,* enquanto que a saída *teste_decoder*, que é a saída do conversor paralelo-serial no qual as 16 saídas do decodificador de endereços estão conectadas, mostra o endereço selecionado.

A Figura 9.4 mostra a simulação final para *teste_conv* = 1. Neste caso, observa-se na saída *out* uma combinação dos bits de endereço (0110), de acordo com a disposição

detalhada no item 5.3 e ilustrada na Figura 5.6. A saída *teste_decoder* é idêntica à da Figura 9.3.

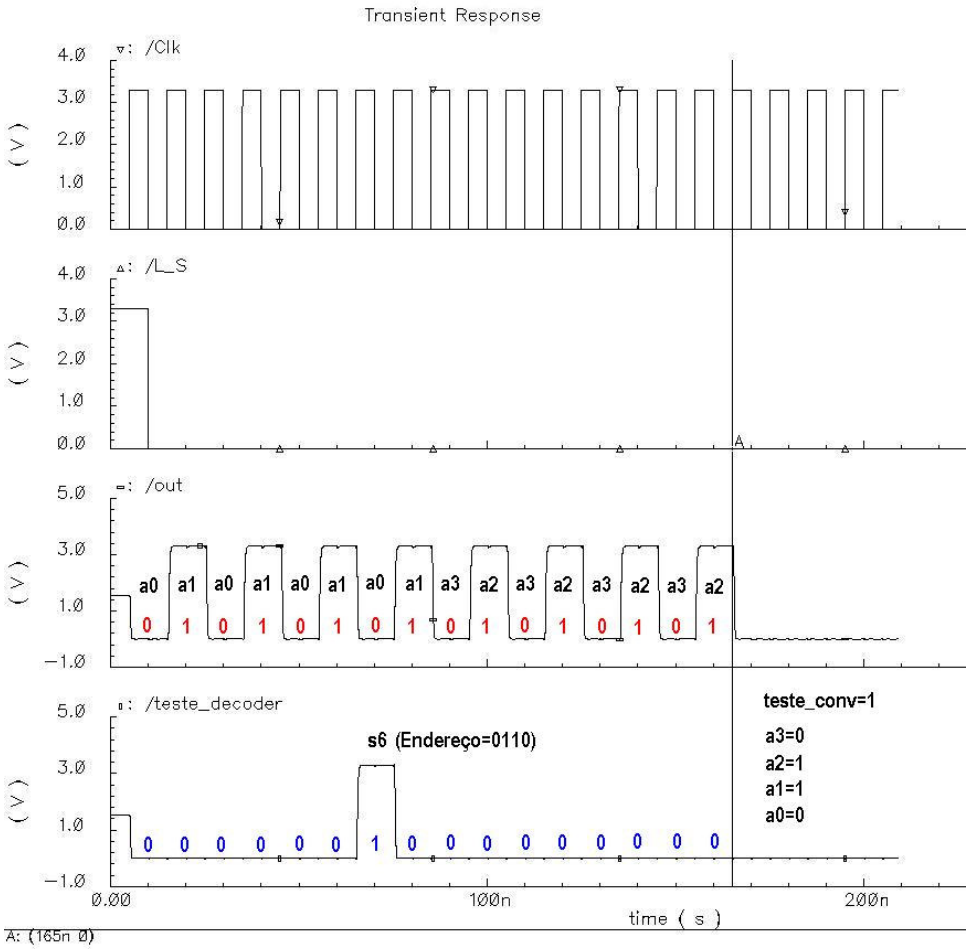


Fig. 9.4 - Simulação final da ROM para *teste_conv* = 1

O tempo de leitura observado nas simulações sem a estrutura de teste foi, no pior caso, de aproximadamente 3,8.ns (vide Figura 9.5). A potência dissipada pela ROM no pior caso, ou seja, quando está sendo efetuada uma leitura em uma posição onde todos os bits são 0, foi estimada em aproximadamente 12,8 mW (valor rms). Isso ocorre porque, neste caso, há um transistor NMOS conduzindo para cada bit 0 armazenado. Como foi priorizada a área ocupada no *chip*, as dimensões W e L dos transistores são as mínimas. Como a potência dissipada está muito alta nesse caso, sugere-se que na continuação desse trabalho a dimensão L dos transistores seja aumentada, de maneira que a resistência seja maior e, conseqüentemente, a corrente seja menor.

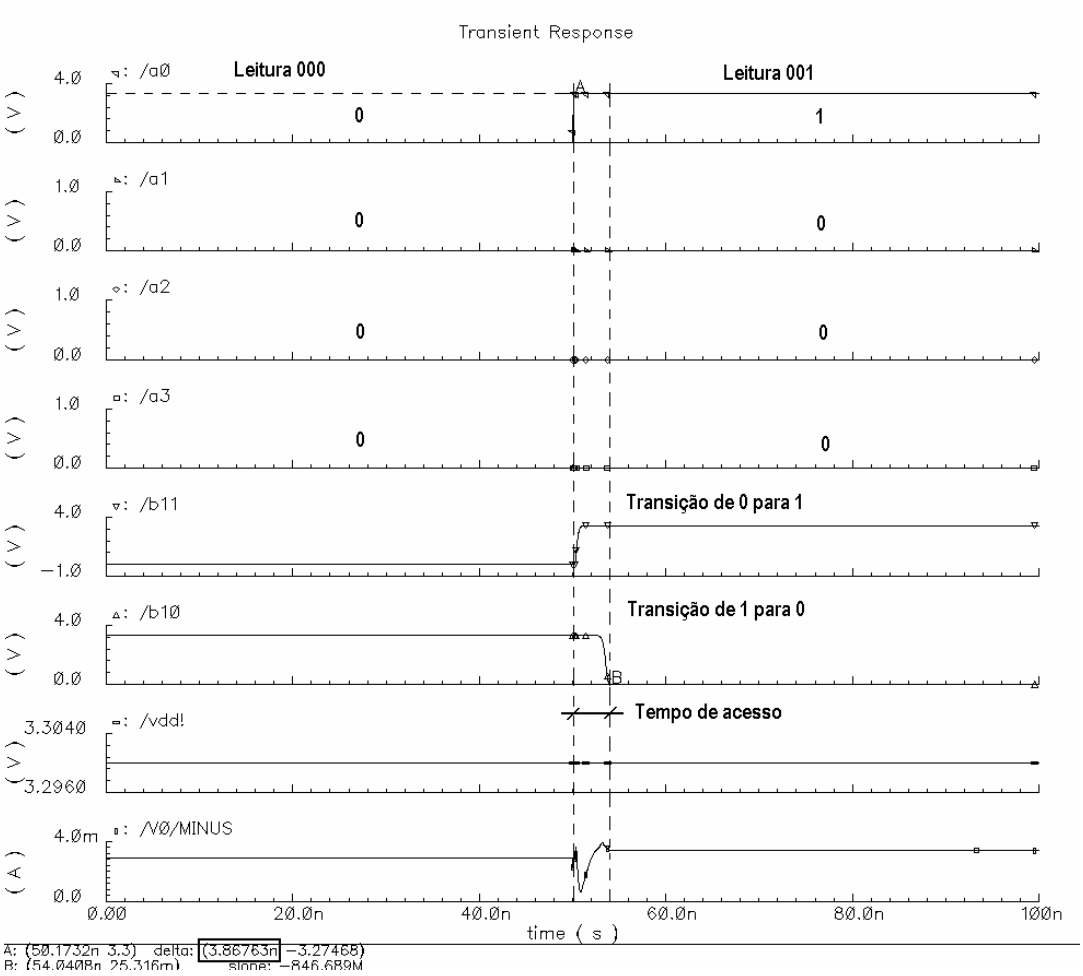


Fig. 9.5 - Tempo de acesso da ROM

A Figura 9.6 mostra o *layout* final da ROM, incluindo a estrutura de teste. A área total ocupada foi de aproximadamente 0,078 mm^2.

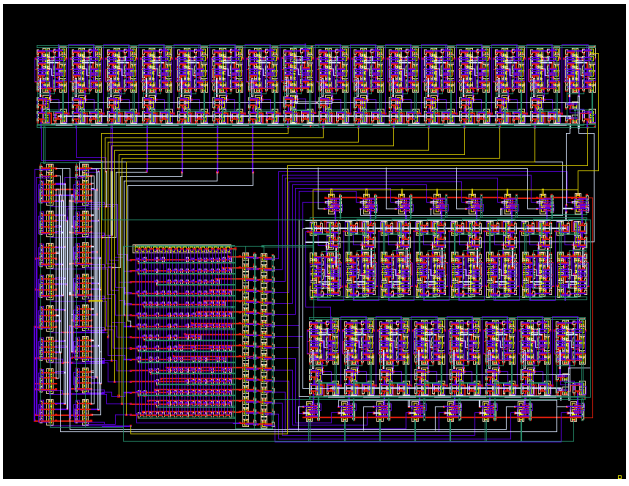


Fig. 9.6 – *Layout* final da ROM (331,9µm x 233,8µm)

9.2.2 SRAM

Foram feitas várias simulações com a estrutura final da memória RAM, ou seja, com os conversores paralelo-serial e serial paralelo. A Figura 9.7 mostra uma delas, na qual é feita uma operação de escrita do dado 0101 0101 0101 0101 em todas as oito posições da RAM, e em seguida é feita uma leitura desse dado na posição 000. As escritas nas outras posições foram feitas para se adicionar um tempo entre a escrita e leitura do dado, a fim de se testar se o mesmo está sendo retido nas células.

Observa-se que, para escrever o dado, é necessário esperar 16 ciclos de *clock*, para que o mesmo possa se deslocar pelo conversor serial-paralelo. Da mesma forma, o dado na saída só pode ser completamente observado após 16 ciclos de *clock* depois que o sinal L_S foi ativado. É importante notar que o primeiro dado já é mostrado ainda com L_S = 1, na borda de subida do próximo pulso de *clock*.

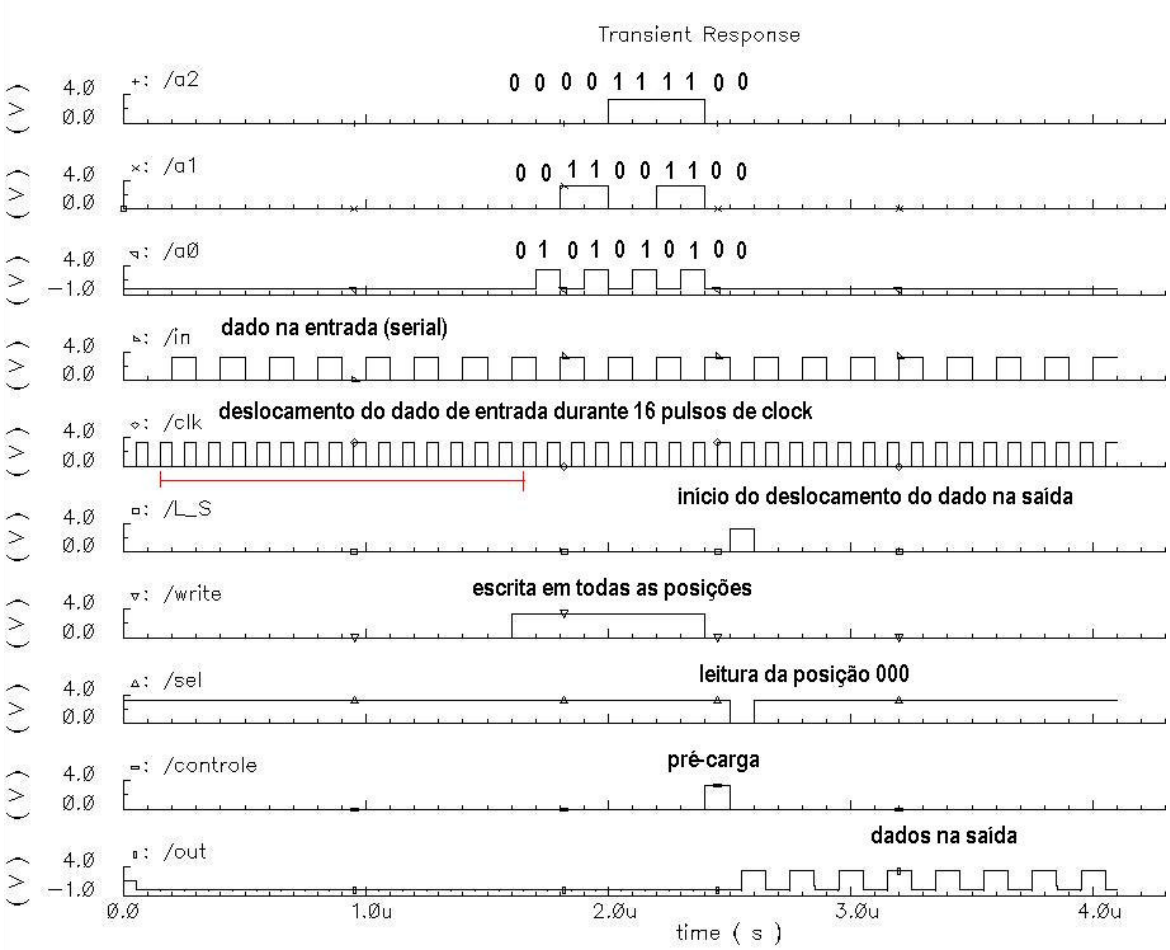


Fig. 9.7 - Simulação final da RAM com estruturas de teste

O tempo de acesso observado nas simulações sem estruturas de teste foi de aproximadamente 28 ns, no caso da transição de 0 para 1, e de 10 ns, no caso da transição de 1 para 0, conforme mostra a simulação da Figura 9.8. Nesse caso, foram feitas três escritas e três leituras nas posições 000, 111 e 000. A potência dissipada nessas operações foi estimada em 32,48 mW (valor rms), sendo que o valor médio foi de 12,68 mW. Observa-se que estes valores estão altos, sendo esse mais um aspecto do projeto a ser melhorado nas versões futuras.

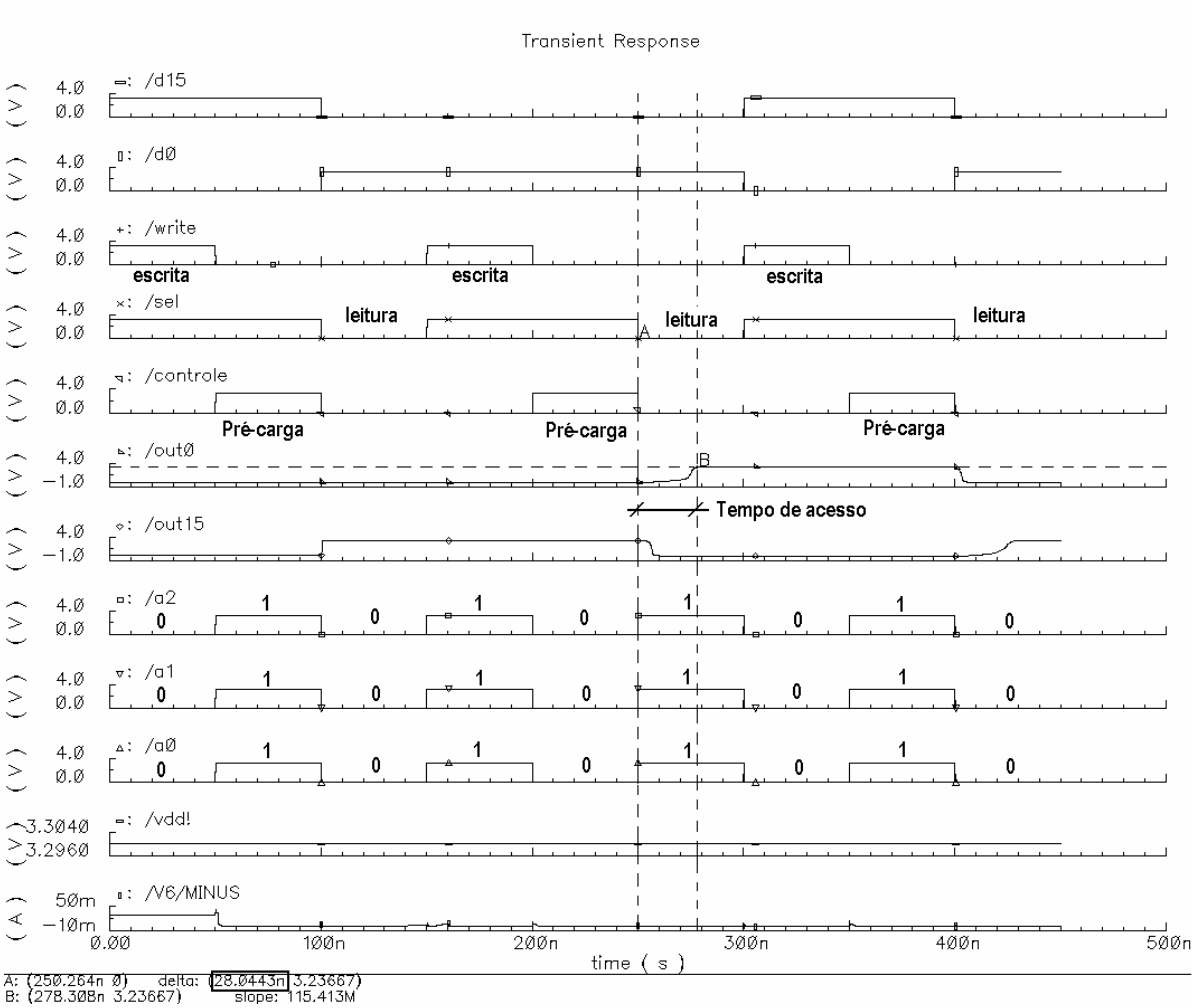


Fig. 9.8 - Tempo de acesso da RAM

O *layout* completo da SRAM implementada está mostrado na Figura 9.9. A área final ocupada no *chip* foi de 0,14 mm^2.

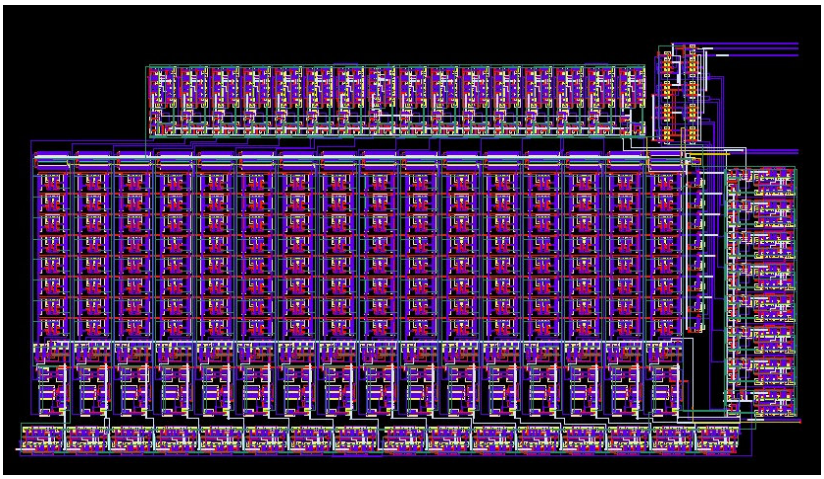


Fig. 9.9 - *Layout* final (274,4µm x 521,3µm)

9.2.3 BANCO DE REGISTRADORES

Da mesma forma que para a memória RAM, foram feitas simulações de escrita e leitura em registradores do banco. Conforme foi visto no Capítulo 7, foram incluídas estruturas de teste no banco, de maneira que o mesmo funciona em dois modos. No modo normal, *teste_cell* = 0 e as saídas *data_a* e *data_b* apresentam a saída do banco de registradores. A Figura 9.10 mostra as formas de onda obtidas em uma simulação com *teste_cell* = 0 e na qual foi realizada uma escrita do dado 0000 0000 1111 1111 no registrador 1100, e uma posterior leitura em ambas as portas do mesmo registrador.

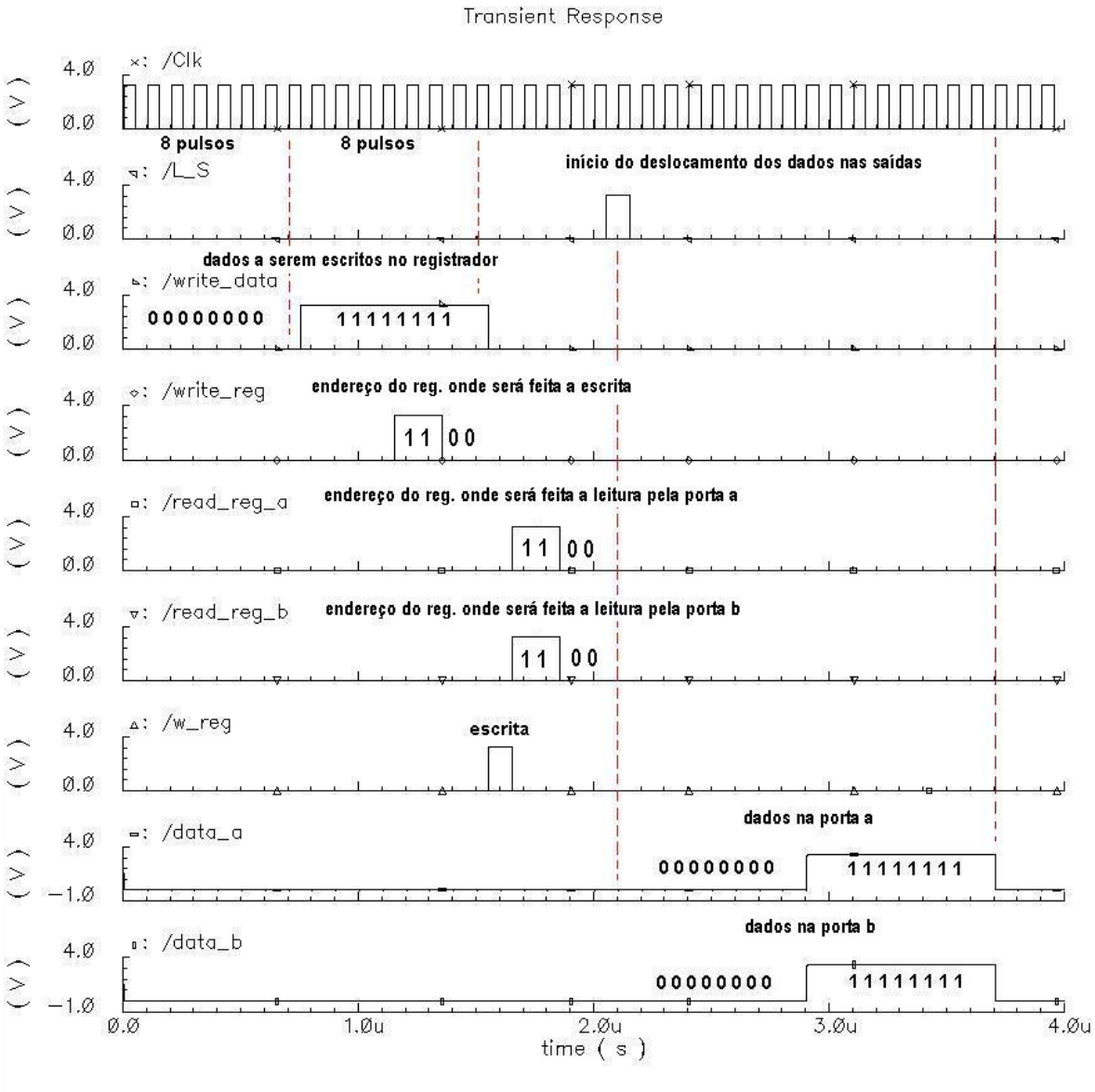


Fig. 9.10 - Simulação final do banco de registradores com *teste_cell* = 0

Da mesma forma que na RAM, é necessário esperar 16 pulsos de *clock* antes de escrever o dado porque há um conversor serial-paralelo de 16 bits na entrada *write_data*. É necessário também carregar o endereço dos registradores 4 pulsos de *clock* antes da operação ser executada porque há conversores serial-paralelo de 4 bits nas entradas *read_reg_a*, *read_reg_b* e *write_reg*.

No modo de teste, *teste_cell* = 1, e as saídas *data_a* e *data_b* mostram as saídas de uma única célula de um registrador, conforme pode ser observado na simulação da Figura 9.11. Neste caso, de maneira análoga à que foi feita na simulação de uma única célula isolada de um registrador no Capítulo 7, foi feita a escrita do dado 0, seguida da leitura do dado na porta a, na porta b, e em ambas as portas simultaneamente. Este procedimento foi seguido também para o dado 1.

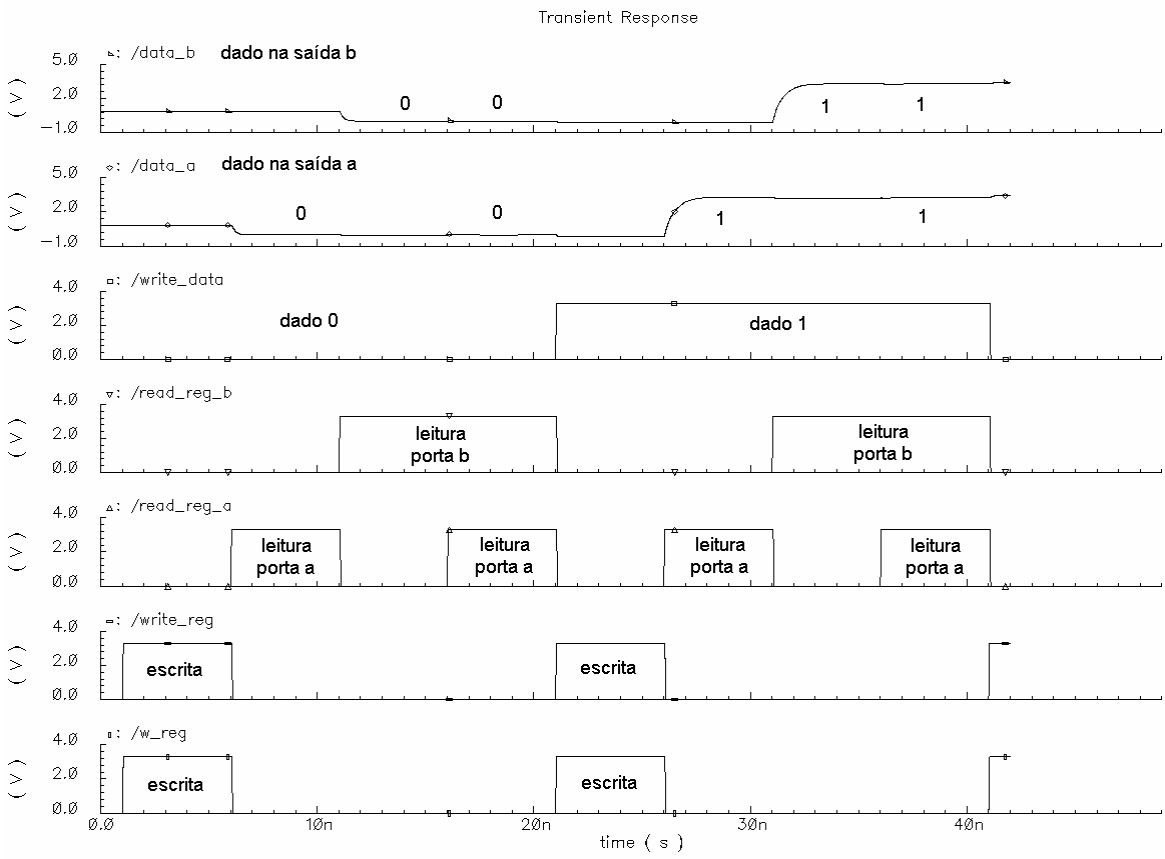


Fig. 9.11 - Simulação final do banco de registradores com *teste_cell* = 1

O tempo de acesso observado nas simulações foi de aproximadamente 7 ns, conforme é mostrado na Figura 9.12. Nessa simulação foram feitas duas operações de escrita e uma de leitura, sem as estruturas de teste, e a potência foi estimada em 2,9 mW (valor rms), sendo que o valor médio foi de 265 uW.

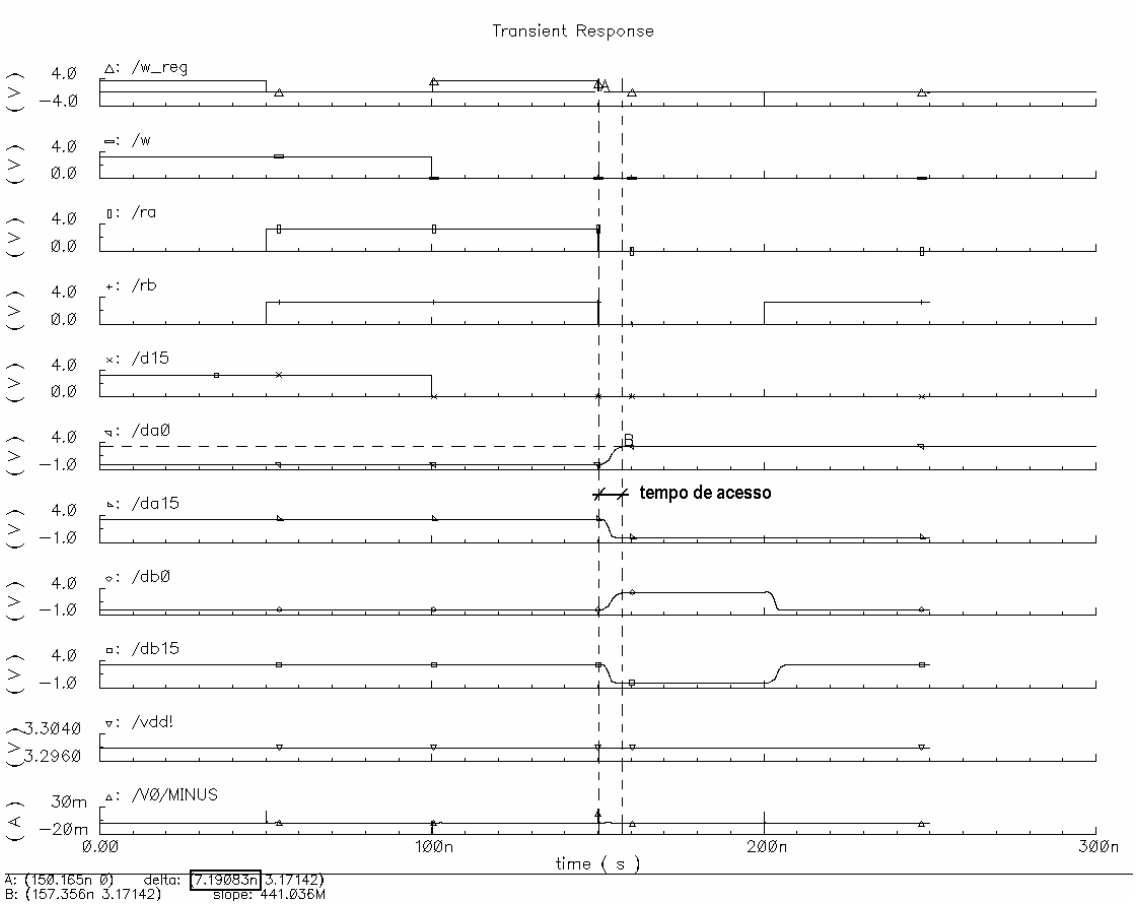


Fig. 9.12 - Tempo de acesso do banco de registradores

A Figura 9.13 apresenta o *layout* do banco de registradores. A área final ocupada foi de 0,78 mm^2.

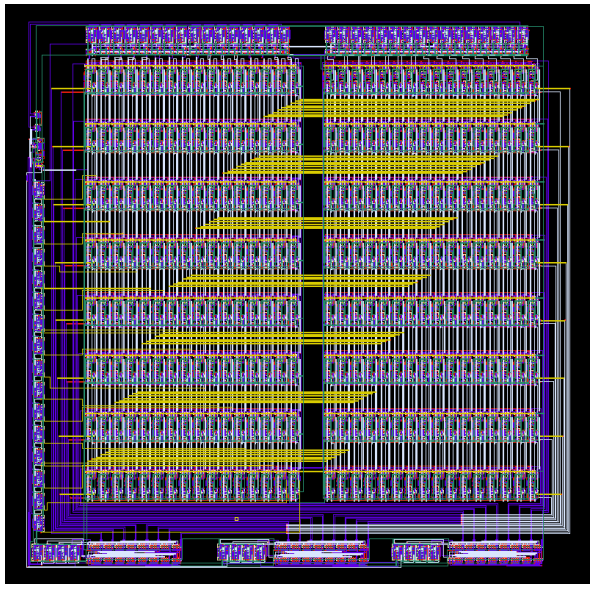


Fig. 9.13 - *Layout* final do banco de registradores (878,9µm x 880,8µm)

10 CONCLUSÃO

Neste trabalho foram desenvolvidos um banco de registradores, uma memória ROM de 256 bits e uma RAM de 128 bits. As memórias foram implementadas tendo em vista a validação dos circuitos escolhidos para a realização da estrutura de armazenamento que será utilizada em um SoC. Este último se destina à agricultura de precisão e tem como objetivo propiciar uma melhor gestão do uso da água. Foram utilizadas técnicas de projeto orientado à testabilidade. Os módulos foram projetados e simulados utilizando ferramentas do CADENCE.

Foram propostas também estruturas para expansão das memórias ROM e RAM, bem como para integração das mesmas e inclusão de registradores que serão mapeados em memória. Trabalhos futuros incluem a validação da memória completa por meio de simulações e elaboração de testes, como o BIST, para a verificação do funcionamento da memória após a mesma ter sido integrada ao processador RISC.

Um *chip* de teste, no qual foram incluídos os módulos implementados juntamente com outras estruturas isoladas do SoC, foi fabricado em tecnologia 0,35 um e estará sendo testado em breve. A próxima etapa é a realização de testes que permitirão a análise do seu funcionamento e a verificação de desempenho. Esses resultados serão de suma importância para a integração desses módulos e realização do processador como um todo.

11 REFERÊNCIAS BIBLIOGRÁFICAS

[AMS, 2001] AMS 0.35 UM CMOS PROCESS PARAMETERS – Revision # 2, 2001.

[BENÍCIO, 2002] BENÍCIO, G. M. "Projeto de Microprocessador RISC 16-Bit Para Sistema de Comunicação Sem Fio em Chip", (dissertação de mestrado em engenharia elétrica), Universidade de Brasília, Brasília, 2002.

[CHANDRAKASAN, 2001] CHANDRAKASAN, A. *et al*.(editors), "Design of High-Performance Microprocessor Circuits", IEEE Press, 2001.

[COSTA, 2004a] COSTA, J.D. "Implementação de um Processador RISC 16-Bits CMOS num Sistema em Chip", (dissertação de mestrado em engenharia elétrica), Universidade de Brasília, Brasília, 2004.

[COSTA, 2004b] COSTA, J.D., BESERRA, G.S., ARAÚJO, G., MARRA, J.C., ROCHA, A. F., COSTA, J. C. "Projeto de Estruturas de um Processador RISC para aplicação em um SoC de controle de irrigação", X Workshop IBERCHIP, 2004.

[COSTA, 2003] COSTA, J.D. *et al*. "Módulo I.P. de um processador para aplicações embarcadas sem fio", IX Workshop IBERCHIP, 2003.

[FRANKLIN, 1990] FRANKLIN, M., SALUJA, K.K. "Built-in Self-Testing of Random-Access Memories", IEEE, 1990.

[GOOR, 1993] GOOR, A.J. van de, "Using March Tests to Test SRAMs", IEEE Design & Test of Computers, 1993.

[HODGES, 2003] HODGES, D.A., JACKSON, H.G., SALEH, R. "Analysis and Design of Digital Integrated Circuits", 3rd ed., McGraw-Hill, 2003.

[JOUBERT, 1999] JOUBERT, T. H., SEEVINCK, E., PLESSIS, M. du, "A Four Transistor CMOS SRAM Cell", IEEE, 1999.

[MAHESHWARI, 2000] MAHESHWARI, A. "Current-Sensing Methods for Global Interconnects in Very Deep Sub-micron (VDSM) CMOS" (dissertação de mestrado em Engenharia Elétrica e da Computação), Graduate School of the University of Massachusetts Amherst, Nov. 2000.

[MOHAN, 2003] MOHAN, N., SAMBANDAN, S. "Design of 4kb Asynchronous SRAM in 0.18 μm CMOS Technology", Project Report, Course E&CE 637, Design of VLSI MOS Integrated Circuits, Department of Electrical and Computer Engineering, University of Waterloo, 2003.

[PATTERSON, 2000] PATTERSON, D. A., HENNESSY, J. L., "Organização e projeto de computadores – A interface hardware/software", 2ª ed., LTC, 2000.

[RABAEY, 2003] RABAEY, J. M., CHANDRAKASAN, A., NIKOLIC, B., "Digital Integrated Circuits – A Design Perspective", 2nd ed., Prentice Hall, 2003, Chapter 12 in http://bwrc.eecs.berkeley.edu/IcBook/

[SEDRA, 2000] SEDRA, A. S., SMITH, K. C., "Microeletrônica", 2ª ed., Makron Books, 2000.

[SEEVINCK, 1990] SEEVINCK, E. "A current sense-amplifier for fast CMOS SRAMs", Symposium on VLSI Circuits, IEEE, 1990.

[WANG, 2000] WANG, B. T., KUO, J. B., "A Novel Two Port 6T CMOS SRAM Cell Structure for Low Voltage VLSI with Single Bit Lines Simultaneous Read and Write Capability", IEEE International Symposium on Circuits and Systems, May 26-31, 2000.

[WESTE, 1993] WESTE, N., ESHRAGHIAN, K., "Principles of CMOS VLSI Design – A Systems Perpective", 2nd ed., Addison-Wesley, 1993.

APÊNDICE A – VETORES DE TESTE

A1 – MEMÓRIA ROM

Os valores na Tabela A.1 correspondem aos valores esperados na saída quando aplicadas as entradas especificadas. Deve-se ressaltar que tais valores somente estarão disponíveis nas saídas *teste_decoder* e *out* após terem sido carregados nas entradas dos conversores paralelo-serial (L_S = 1) e deslocados (L_S = 0) durante 16 ciclos de clock.

Tabela A. 1 – Vetores de teste da ROM

Entradas					Saídas	
teste conv	**a3**	**a2**	**a1**	**a0**	**teste_decoder**	**out**
0	0	0	0	0	1000 0000 0000 0000	0000 0000 1110 0100
0	0	0	0	1	0100 0000 0000 0000	0000 0000 0001 0100
0	0	0	1	0	0010 0000 0000 0000	1100 0000 0001 0111
0	0	0	1	1	0001 0000 0000 0000	0000 0000 1000 0100
0	0	1	0	0	0000 1000 0000 0000	1000 0000 1000 0001
0	0	1	0	1	0000 0100 0000 0000	0000 0000 0001 0100
0	0	1	1	0	0000 0010 0000 0000	0000 0000 1001 0100
0	0	1	1	1	0000 0001 0000 0000	1111 1000 1001 0001
0	1	0	0	0	0000 0000 1000 0000	0000 1101 0001 0100
0	1	0	0	1	0000 0000 0100 0000	1001 0001 0001 0010
0	1	0	1	0	0000 0000 0010 0000	0010 1000 0001 0011
0	1	0	1	1	0000 0000 0001 0000	0000 0000 1111 0111
0	1	1	0	0	0000 0000 0000 1000	0000 0000 0001 0100
0	1	1	0	1	0000 0000 0000 0100	0000 0000 0001 0111
0	1	1	1	0	0000 0000 0000 0010	0000 0000 0000 0000
0	1	1	1	1	0000 0000 0000 0001	1111 1111 1111 1111
1	0	0	0	0	1000 0000 0000 0000	0000 0000 0000 0000
1	0	0	0	1	0100 0000 0000 0000	1010 1010 0000 0000
1	0	0	1	0	0010 0000 0000 0000	0101 0101 0000 0000
1	0	0	1	1	0001 0000 0000 0000	1111 1111 0000 0000
1	0	1	0	0	0000 1000 0000 0000	0000 0000 0101 0101
1	0	1	0	1	0000 0100 0000 0000	1010 1010 0101 0101
1	0	1	1	0	0000 0010 0000 0000	0101 0101 0101 0101
1	0	1	1	1	0000 0001 0000 0000	1111 1111 0101 0101
1	1	0	0	0	0000 0000 1000 0000	0000 0000 1010 1010
1	1	0	0	1	0000 0000 0100 0000	1010 1010 1010 1010
1	1	0	1	0	0000 0000 0010 0000	0101 0101 1010 1010
1	1	0	1	1	0000 0000 0001 0000	1111 1111 1010 1010
1	1	1	0	0	0000 0000 0000 1000	0000 0000 1111 1111
1	1	1	0	1	0000 0000 0000 0100	1010 1010 1111 1111
1	1	1	1	0	0000 0000 0000 0010	0101 0101 1111 1111
1	1	1	1	1	0000 0000 0000 0001	1111 1111 1111 1111

A2 – MEMÓRIA RAM

Teste do decodificador: da mesma forma que na ROM, os dados na saída Out_dec também só estarão disponíveis após serem carregados na entrada do conversor paralelo-serial de 8 bits (L_S = 1) e deslocados durante 8 ciclos de clock (L_S = 0). A Tabela A.2 mostra os vetores de teste.

Tabela A. 2 - Vetores de teste do decodificador

Entradas			**Saída**
a2	**a1**	**a0**	**Out_dec**
0	0	0	1000 0000
0	0	1	0100 0000
0	1	0	0010 0000
0	1	1	0001 0000
1	0	0	0000 1000
1	0	1	0000 0100
1	1	0	0000 0010
1	1	1	0000 0001

Teste da matriz SRAM: Como existem inúmeras seqüências de operações que podem ser realizadas, foi escolhida como exemplo a seqüência correspondente à do algoritmo March C-, conforme é mostrado na Tabela A.3. A equação correspondente é a 2.2, reescrita a seguir:

$$\{\uparrow\downarrow(w0);\uparrow(r0,w1);\ \uparrow(r1,w0);\downarrow(r0,w1);\downarrow(r1,w0)\uparrow\downarrow(r0)\}$$

Durante a escrita, deve-se esperar 16 ciclos de clock antes de fazer *write* = 1, pois há um conversor serial-paralelo de 16 bits nessa entrada. Deve-se observar que, para que os valores sejam observados nas saídas durante a leitura, é necessário que o sinal *controle* = 1 um ciclo de clock antes de fazer *sel* = 0. Além disso, como há um conversor paralelo-serial de 16 bits na saída da matriz, é necessário seguir o mesmo procedimento especificado para obter as saídas da ROM. É importante ressaltar que as operações listadas na Tabela A.3 devem ser executadas na seqüência em que aparecem.

Tabela A. 3 - Vetores de teste para a RAM

Entradas							Saída
write_data(H)	**a2**	**a1**	**a0**	**sel**	**write**	**controle**	**out (H)**
0000	0	0	0	1	1	0	----
0000	0	0	1	1	1	0	----
0000	0	1	0	1	1	0	----
0000	0	1	1	1	1	0	----
0000	1	0	0	1	1	0	----
0000	1	0	1	1	1	0	----
0000	1	1	0	1	1	0	----
0000	1	1	1	1	1	0	----
X	0	0	0	1	0	1	----
X	0	0	0	0	0	0	0000
X	0	0	1	1	0	1	0000
X	0	0	1	0	0	0	0000
X	0	1	0	1	0	1	0000
X	0	1	0	0	0	0	0000
X	0	1	1	1	0	1	0000
X	0	1	1	0	0	0	0000
X	1	0	0	1	0	1	0000
X	1	0	0	0	0	0	0000
X	1	0	1	1	0	1	0000
X	1	0	1	0	0	0	0000
X	1	1	0	1	0	1	0000
X	1	1	0	0	0	0	0000
X	1	1	1	1	0	1	0000
X	1	1	1	0	0	0	0000
1111	0	0	0	1	1	0	----
1111	0	0	1	1	1	0	----
1111	0	1	0	1	1	0	----
1111	0	1	1	1	1	0	----
1111	1	0	0	1	1	0	----
1111	1	0	1	1	1	0	----
1111	1	1	0	1	1	0	----
1111	1	1	1	1	1	0	----
X	0	0	0	1	0	1	----
X	0	0	0	0	0	0	1111
X	0	0	1	1	0	1	1111
X	0	0	1	0	0	0	1111
X	0	1	0	1	0	1	1111
X	0	1	0	0	0	0	1111
X	0	1	1	1	0	1	1111
X	0	1	1	0	0	0	1111
X	1	0	0	1	0	1	1111
X	1	0	0	0	0	0	1111
X	1	0	1	1	0	1	1111

X	1	0	1	0	0	0	1111
X	1	1	0	1	0	1	1111
X	1	1	0	0	0	0	1111
X	1	1	1	1	0	1	1111
X	1	1	1	0	0	0	1111
0000	0	0	0	1	1	0	1111
0000	0	0	1	1	1	0	1111
0000	0	1	0	1	1	0	1111
0000	0	1	1	1	1	0	1111
0000	1	0	0	1	1	0	1111
0000	1	0	1	1	1	0	1111
0000	1	1	0	1	1	0	1111
0000	1	1	1	1	1	0	1111
X	1	1	1	1	0	1	1111
X	1	1	1	0	0	0	0000
X	1	1	0	1	0	1	0000
X	1	1	0	0	0	0	0000
X	1	0	1	1	0	1	0000
X	1	0	1	0	0	0	0000
X	1	0	0	1	0	1	0000
X	1	0	0	0	0	0	0000
X	0	1	1	1	0	1	0000
X	0	1	1	0	0	0	0000
X	0	1	0	1	0	1	0000
X	0	1	0	0	0	0	0000
X	0	0	1	1	0	1	0000
X	0	0	1	0	0	0	0000
X	0	0	0	1	0	1	0000
X	0	0	0	0	0	0	0000
1111	1	1	1	1	1	0	0000
1111	1	1	0	1	1	0	0000
1111	1	0	1	1	1	0	0000
1111	1	0	0	1	1	0	0000
1111	0	1	1	1	1	0	0000
1111	0	1	0	1	1	0	0000
1111	0	0	1	1	1	0	0000
1111	0	0	0	1	1	0	0000
X	1	1	1	1	0	1	0000
X	1	1	1	0	0	0	1111
X	1	1	0	1	0	1	1111
X	1	1	0	0	0	0	1111
X	1	0	1	1	0	1	1111
X	1	0	1	0	0	0	1111
X	1	0	0	1	0	1	1111
X	1	0	0	0	0	0	1111
X	0	1	1	1	0	1	1111
X	0	1	1	0	0	0	1111
X	0	1	0	1	0	1	1111

X	0	1	0	0	0	0	1111
X	0	0	1	1	0	1	1111
X	0	0	1	0	0	0	1111
X	0	0	0	1	0	1	1111
X	0	0	0	0	0	0	1111
0000	1	1	1	1	1	0	1111
0000	1	1	0	1	1	0	1111
0000	1	0	1	1	1	0	1111
0000	1	0	0	1	1	0	1111
0000	0	1	1	1	1	0	1111
0000	0	1	0	1	1	0	1111
0000	0	0	1	1	1	0	1111
0000	0	0	0	1	1	0	1111
X	0	0	0	1	0	1	1111
X	0	0	0	0	0	0	0000
X	0	0	1	1	0	1	0000
X	0	0	1	0	0	0	0000
X	0	1	0	1	0	1	0000
X	0	1	0	0	0	0	0000
X	0	1	1	1	0	1	0000
X	0	1	1	0	0	0	0000
X	1	0	0	1	0	1	0000
X	1	0	0	0	0	0	0000
X	1	0	1	1	0	1	0000
X	1	0	1	0	0	0	0000
X	1	1	0	1	0	1	0000
X	1	1	0	0	0	0	0000
X	1	1	1	1	0	1	0000
X	1	1	1	0	0	0	0000

As células hachuradas correspondem à ocorrência de operação de escrita ou leitura, ressaltando-se que esta última ocorre quando sel = 0. Os dados na saída que não se encontram hachurados não correspondem a operações de leitura, mas significam que o dado anterior está sendo mantido ao se seguir a seqüência de operações descrita na Tabela A.3.

A3 – BANCO DE REGISTRADORES

Para testar o banco de registradores, escolheu-se a escrita de um dado em todos os 16 registradores e a posterior escrita. Dessa forma, para *write_data* = 1111 1111 0000 0000, foram elaboradas as tabelas a seguir.

***teste_cell* = 0**: Neste caso, as saídas *data-a* e *data-b* correspondem à saída do banco de registradores. Como existem conversores serial-paralelo de 4 bits nas entradas *read_reg_a*, *read_reg_b* e *write_reg*, é necessário esperar 4 ciclos de clock antes de

habilitar a escrita fazendo o sinal *w_reg* = 1. Há um conversor paralelo-serial em cada saída, e dessa forma é necessário seguir para *data*_a e *data*_b o mesmo procedimento utilizado para verificar as saídas da ROM e da RAM.

Tabela A. 4 - Vetores de teste do banco de registradores

Entradas				**Saídas**	
write_reg	**w_reg**	**read_reg_a**	**read_reg_b**	**data_a**	**data_b**
0000	1	XXXX	XXXX	---- ---- ---- ----	---- ---- ---- ----
0001	1	XXXX	XXXX	---- ---- ---- ----	---- ---- ---- ----
0010	1	XXXX	XXXX	---- ---- ---- ----	---- ---- ---- ----
0011	1	XXXX	XXXX	---- ---- ---- ----	---- ---- ---- ----
0100	1	XXXX	XXXX	---- ---- ---- ----	---- ---- ---- ----
0101	1	XXXX	XXXX	---- ---- ---- ----	---- ---- ---- ----
0110	1	XXXX	XXXX	---- ---- ---- ----	---- ---- ---- ----
0111	1	XXXX	XXXX	---- ---- ---- ----	---- ---- ---- ----
1000	1	XXXX	XXXX	---- ---- ---- ----	---- ---- ---- ----
1001	1	XXXX	XXXX	---- ---- ---- ----	---- ---- ---- ----
1010	1	XXXX	XXXX	---- ---- ---- ----	---- ---- ---- ----
1011	1	XXXX	XXXX	---- ---- ---- ----	---- ---- ---- ----
1100	1	XXXX	XXXX	---- ---- ---- ----	---- ---- ---- ----
1101	1	XXXX	XXXX	---- ---- ---- ----	---- ---- ---- ----
1110	1	XXXX	XXXX	---- ---- ---- ----	---- ---- ---- ----
1111	1	XXXX	XXXX	---- ---- ---- ----	---- ---- ---- ----
XXXX	0	0000	0000	0000 0000 1111 1111	0000 0000 1111 1111
XXXX	0	0001	0001	0000 0000 1111 1111	0000 0000 1111 1111
XXXX	0	0010	0010	0000 0000 1111 1111	0000 0000 1111 1111
XXXX	0	0011	0011	0000 0000 1111 1111	0000 0000 1111 1111
XXXX	0	0100	0100	0000 0000 1111 1111	0000 0000 1111 1111
XXXX	0	0101	0101	0000 0000 1111 1111	0000 0000 1111 1111
XXXX	0	0110	0110	0000 0000 1111 1111	0000 0000 1111 1111
XXXX	0	0111	0111	0000 0000 1111 1111	0000 0000 1111 1111
XXXX	0	1000	1000	0000 0000 1111 1111	0000 0000 1111 1111
XXXX	0	1001	1001	0000 0000 1111 1111	0000 0000 1111 1111
XXXX	0	1010	1010	0000 0000 1111 1111	0000 0000 1111 1111
XXXX	0	1011	1011	0000 0000 1111 1111	0000 0000 1111 1111
XXXX	0	1100	1100	0000 0000 1111 1111	0000 0000 1111 1111
XXXX	0	1101	1101	0000 0000 1111 1111	0000 0000 1111 1111
XXXX	0	1110	1110	0000 0000 1111 1111	0000 0000 1111 1111
XXXX	0	1111	1111	0000 0000 1111 1111	0000 0000 1111 1111

teste_cell = 1: Neste caso, está sendo testada apenas uma célula de um registrador. As entradas e saídas são de um bit. A Tabela A.5 descreve os vetores de teste para essa possibilidade.

Tabela A. 5 - Vetores de teste de um registrador

Entradas					Saídas	
write_reg	**w_reg**	**read_reg_a**	**read_reg_b**	**write_data**	**data-a**	**data_b**
1	1	X	X	0	----	----
X	0	1	0	X	0	X
X	0	0	1	X	0	0
X	0	1	1	X	0	0
1	1	X	X	1	----	----
X	0	1	0	X	1	0
X	0	0	1	X	1	1
X	0	1	1	X	1	1

APÊNDICE B – CÓDIGOS VHDL

B1 - MEMÓRIA

```
LIBRARY          ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY memory IS PORT (
                 clock:          in STD_LOGIC;
                 rst:            in STD_LOGIC;
                 RMem:           in STD_LOGIC;
                 WMem:           in STD_LOGIC;
                 endereco:       in STD_LOGIC_VECTOR(15 downto 0);
                 WriteData:      in STD_LOGIC_VECTOR(15 downto 0);
                 AcInt:          out STD_LOGIC;
                 RegInt:         out STD_LOGIC_VECTOR(15 downto 0);
                 addr:           out STD_LOGIC_VECTOR(15 downto 0);
                 Bit_zero:       in STD_LOGIC;
                 Bit_um:         in STD_LOGIC;
                 Bit_dois   :    in STD_LOGIC;
                 Bit_tres:       inout STD_LOGIC;
                 Bit_quatro:     inout STD_LOGIC;
                 UlaResult:      in STD_LOGIC_VECTOR(15 downto 0);
                 IntIdle:        in STD_LOGIC;
                 zero:           in STD_LOGIC;
                 N:              in STD_LOGIC;
                 Carry:          in STD_LOGIC;
                 Caux:           in STD_LOGIC;
                 RegStatus:      out STD_LOGIC_VECTOR(15 downto 0);
                 erro, BitAddr:  out STD_LOGIC;
                 erro_mem:       out STD_LOGIC;-- sinal encaminhado ao controlador de
interrupções para indicar erro de endereçamento
                 ReadData:       inout STD_LOGIC_VECTOR(15 downto 0) -- dado de saída
);
END MEMORY;

ARCHITECTURE arch_memory OF memory IS

  TYPE ram_type IS ARRAY (0 to 100) of STD_LOGIC_VECTOR(15 downto 0);
  SIGNAL tmp_ram: ram_type;
  SIGNAL x,y: STD_LOGIC_VECTOR (15 downto 0);
  SIGNAL z,w,k: STD_LOGIC;

 BEGIN
 x(0) <= zero;
 y(0) <=WriteData(0);
 x(1) <= N;                    y(1) <= Bit_zero;
 x(2) <= Carry;                y(2) <= Bit_um;
 x(3) <= Caux;                 y(3) <= Bit_dois;
 x(4) <= Bit_tres;             y(4) <= Bit_tres;
 x(5) <= '0';                  y(5) <= Bit_quatro;
 x(6) <= '0';
 x(7) <= '0';
 x(8) <= '0';
 x(9) <= '0';
```

```
 x(10)<= ‘0’;
 x(11) <='0'; x(12) <='0'; x(13) <='0'; x(14)<='0';x(15)<='0';

Int_erro: PROCESS(clock,endereco)
BEGIN
    IF ReadData(15)='1'and ReadData (14)='1'and ReadData(13)='1'and UlaResult>x"0064"THEN
            k <='1';
    ELSE
             k <='0';
    END IF;
END PROCESS;

            write: PROCESS(clock, rst, RMem, endereco, WriteData)
            BEGIN
                        IF rst='1' THEN
                                    tmp_ram <= (-----------------------------------------------------------
--A memória é inicializada com um programa para testar o funcionamento do microprocessador.
-- Funciona como a ROM do processador

                                                0 => x"8a01",               -- addi $SA,01      Faz SA
=0001
                                                1 => x"8b64",               -- addi $SB,64      Armazena
em
--$SB o endereço do registrador de Interrupções (RegInt).
                                                2 => x"0b0c",               -- lw $SC,$SB,$S0Armazena
no
 --Registrador $SC o conteúdo de RegInt
                                                3 => x"5cac",               -- or $SC,$SC,$SA
            Faz com que o --bit menos significativo de $SC seja 1
                                                4 => x"1b0c",               -- sw $SC,$SB,$S0
            Faz com que --AcInt seja 1. ([0064]<=XXX1)
                                                5 => x"8a63",               -- addi $SA.63      $SA=0064
                                                6 => x"10a0",               --sw $S0,$SA,$S0
[0064]<=0000
                                                7 => x"8505",               -- addi $S5,05      $S5=0005
                                                8 => x"8305",               -- addi $S3,05      $S3=0005
                                                9 => x"8203",               -- addi  $S2,03     $S2=0003
                                                10 => x"2324",              -- add   $S4,$S3,$S2
$S4=0008
                                                11 => x"3321",              -- sub   $S1,$S3,$S2
$S1=0002
                                                12 => x"9203",              -- sft   $S2,03     $S2=0018
                                                13 => x"94fe",              -- sft   $S4,fe     $S4=0002
                                                14 => x"4145",              -- and  $S5,$S1,$S4
$S5=0002
                                                15 => x"5346",              -- or    $S6,$S3,$S4  $S6=0007
                                                16 => x"a300",              -- not  $S3          $S3=FFFA
                                                17 => x"6467",              -- xor  $S7,$S4,$S6  $S7=0005
                                                18 => x"7568",              -- slt   $S8,$S5,$S6  $S8=0001
                                                19 => x"7658",              -- slt   $S8,$S6,$S5  $S8=0000
                                                20 => x"0579",              -- lw   $S9,$S5,$S7
$S9<=[0007]
                                                21 => x"1143",              -- sw  $S3,$S1,$S4
[0004]=FFFA
                                                22 => x"b849",              -- lui  $S8,2f      $S8=4900
                                                23 => x"c143",              -- beq $S1,$S4,03  Desvia para
27
                                                24 => x"4532",              -- and $S2,$S5,$S3
                                                25 => x"0000",
                                                26 => x"0000",
```

```
                                27 => x"c184",          -- beq  $S1,$S8,04Não desvia
                                28 => x"d572",          -- blt  $S5,$S7,02  S5<S7
Desvia
                                29 => x"0000",
                                30 => x"0000",
                                31 => x"d752",          -- blt  $S7,$S5,02   Não desvia
                                32 => x"e023",          -- j    $S0,1c      Salta para
35
                                33 => x"0000",
                                34 => x"0000",
                                35 => x"f027",          -- jal  $S1,1e      Salta para
39
                                36 => x"0000",
                                37 => x"0000",
                                38 => x"0000",
                                39 => x"e0c8",          -- j $S0, C8        Erro de
Endereçamento. (Vai para rotina de tratamento de interrupção)
                                40 => x"3222",          -- sub $S2,$S2,$S2  $S2=0000
                                41 => x"8a02",          -- addi $SA,02
                                42 => x"881f",          -- addi $S8,1F      $S8=491F
                                43 => x"b241",          -- lui $S2,41       $S2=4100
                                44 => x"821f",          -- addi $S2, 1F     $S2=411F
                                45 => x"228a",          -- add $SA,$S2,$S8  Overflow
(Vai para rotina de tratamento de interrupção).
                                46 => x"82c8",          -- addi $S2,C8      $S2=41E7
                                47 => x"0021",          -- lw $S1,$S2,$S0 Erro de
Endereçamento. (Vai para rotina de tratamento de interrupção)
                                48 => x"1201",          -- sw S1,$S2,$S0  Erro de
Endereçamento. (Vai para rotina de tratamento de interrupção)
                                49 => x"f0d0",          -- jal $S0,D0       Erro de
Endereçamento. (Vai para rotina de tratamento de interrupção)
                                50 => x"8a07",          -- addi $SA,07
                                51 => x"e007",          -- j $S0,07         Volta para
posição de memória 0007. (Instrução 8505)
-------------------------------------------------------------------------------------------------------------------------------
-- Rotina de tratamento de Interrupção

                                60 => x"3aaa",          -- sub $SA,$SA,$SA
$SA=0000
                                61 => x"8a01",          -- addi $SA,01      $SA=0001
                                62 => x"3bbb",          -- sub $SB,$SB,$SB
$SB=0000
                                63 => x"8b64",          -- addi $SB,64      $SB=0064
Armazena em $SB o endereço do registrador mapeado em memória RegInt
                                64 => x"0b0c",          -- lw $SC,$SB,$S0Armazena
em $SC o conteúdo de RegInt
                                65 => x"5cac",          -- or $SC,$SA,$SC
$SC=XXX1
                                66 => x"1b0c",          -- sw $SC,$SB,$S0  Faz AcInt
= 1
                                67 => x"3ccc",          -- sub $SC,$SC,$SC
$SC=0000
                                68 => x"8c63",          -- addi $SC,63      $SC=0063
Armazena em $SC a posição de memória correspondente ao Registrador de Endereço
                                69 => x"0c0c",          -- lw $SC,$S0,$Sc -- carrega o
registrador $SC com endereço onde ocorreu a interrupção
                                70 => x"3aaa",          -- sub $SA,$SA,$SA
$SA=0000
                                71 => x"8a64",          -- addi $SA,64      $SA=0064
                                72 => x"10a0",          -- sw $S0,$S0,$SA
```

```
habilita novas interrupções
                                        73 => x"ec01",                  -- j $SC,01             retorna
para instrução onde a interrupção foi requisitada +1
-------------------------------------------------------------------------------------------------------------------------
                                        OTHERS => "0000000000000000");

                        ELSE
                                IF endereco > x"0064" THEN  w<='1';
                        ELSE     w<='0';
                                IF (clock'event and clock = '1') THEN
                                        IF (WMem ='1' AND RMem = '0')  THEN -- O conteudo do
endereco especificado eh substituido
                                                IF endereco = x"0064" THEN
                                                        tmp_ram (100) <= y;
                                                        RegInt <=y;
                                                        AcInt <= y(0);          -- pelo dado em
WriteData se ele for diferente do endereco 50 (reg de status)
                                                        BitAddr <=y(5);
                                                ELSIF endereco = x"0063" THEN
                                                        tmp_ram(99)<=WriteData;
                                                        addr<=WriteData;
                                                ELSIF endereco = x"0062" THEN
                                                     tmp_ram(98)<=WriteData;
                                                     RegStatus<= x;
                                                 ELSE
                                                   tmp_ram(conv_integer(endereco)) <= WriteData;
                                                 END IF;
                                        END IF;
                                END IF;
                        END IF;
                END IF;
END PROCESS;

   read: PROCESS(clock, rst, WMem, endereco,tmp_ram)
                BEGIN
                        IF rst='1' THEN
                                ReadData <=tmp_ram(0);-- Comeca a ler a memoria do endereco 0
                        ELSE
                          IF endereco > x"0064" THEN
                           z<='1';
                          ELSE
                              z<='0';
                                IF (clock'event AND clock = '1') THEN
                                        IF (RMem ='1' and WMem ='0') THEN
-- O conteúdo do endereço especificado e colocado na saída Read Data.
                                                ReadData <=
tmp_ram(conv_integer(endereco));
                                        END IF;
                                END IF;
                        END IF;
                  END IF;
                END PROCESS;
erro<=k;
erro_mem <= z or w or k;

END arch_memory
```

B2 – BANCO DE REGISTRADORES

```
----------------------------------------------------------------
-- Banco de registradores
-- 16 registradores de 16 bits
----------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY reg_file IS

 PORT (WriteData:                    IN  STD_LOGIC_VECTOR(15  downto  0); -- armazena o
dado a ser escrito no reg selecionado
      ReadRegA, ReadRegB, WriteReg:        IN STD_LOGIC_VECTOR(3 downto 0);
      clk,WReg,  reset_A:                  IN STD_LOGIC;
      --Bit_zero:IN STD_LOGIC;
      --Bit_um:IN STD_LOGIC;
      --Bit_dois:IN STD_LOGIC;
      --Bit_tres:IN STD_LOGIC;
      --Bit_quatro:IN STD_LOGIC;
      --LouD:IN STD_LOGIC_VECTOR(1 downto 0);
      --AcInt: out STD_LOGIC;
      --AcInt_in: in STD_LOGIC;
      rzero,rum, rdois,rtres,rquatro,rcinco:    OUT STD_LOGIC_VECTOR(15 downto 0);
      rseis,rsete,roito,rnove,ra,rb,rc,rd,re,rf:    OUT   STD_LOGIC_VECTOR(15   downto   0);  --
essas saidas foram inseridas para facilitar a visualizacao dos registradores
      Data_A, Data_B :                  OUT STD_LOGIC_VECTOR(15 downto 0));

END reg_file;

ARCHITECTURE structural OF reg_file IS

TYPE reg_16 is ARRAY(0 to 15) of STD_LOGIC_VECTOR(15 downto 0);
SIGNAL R : reg_16;
SIGNAL x,y: STD_LOGIC_VECTOR (15 downto 0);
--signal regtres:STD_LOGIC_VECTOR(15 downto 0);


BEGIN  --  structural
                        --x(1) <=Bit_zero;
                        --x(2) <=Bit_um;
                        --x(3) <=Bit_dois;
                        --x(4) <=Bit_tres;
                        --x(5) <=Bit_quatro;
                        --x(0) <=WriteData(0);


 decoder: PROCESS(clk, WReg, WriteReg, WriteData)
  BEGIN
 if rising_edge (clk) then
            if reset_A ='1' then
       -- for i in 0 to 8 loop
            --          R(i) <= "0000000000000010";
       -- end loop;
            R(0) <=  "0000000000000000";
            R(1) <=  "0000000000000000";
            R(2) <=  "0000000000000000";
```

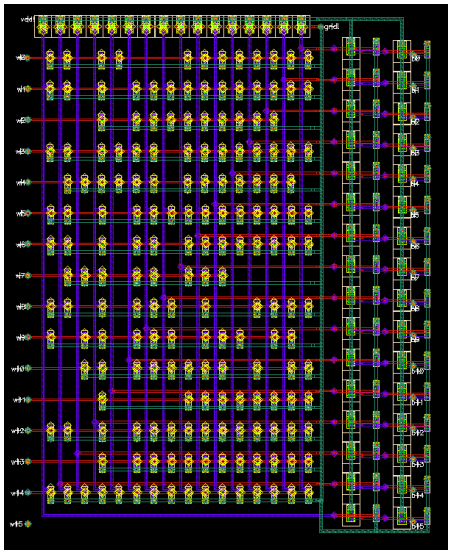
```
            R(3) <=  "0000000000000000";
            R(4) <=  "0000000000000000";
            R(5) <=  "0000000000000000";
            R(6) <=  "0000000000000000";
            R(7) <=  "0000000000000000";
            R(8) <=  "0000000000000000";
            R(9) <=  "0000000000000000";
            R(10) <=  "0000000000000000";
            R(11) <=  "0000000000000000";
            R(12) <=  "0000000000000000";
            R(13) <=  "0000000000000000";
            R(14) <=  "0000000000000000";
            R(15) <=  "0000000000000000";
        elsif  WReg = '1' then              -- Se WReg = 1 a escrita esta habilitada. O registrador
selecionado por WriteReg
                                            -- recebe o valor presente em Write Data
                    --if LouD = "10" then


      case WriteReg is

                    when "0000" => R(0)  <= WriteData;
                    when "0001" => R(1)  <= WriteData;
                    when "0010" => R(2)  <= WriteData;
                    when "0011" => R(3)  <= WriteData;
                    when "0100" => R(4)  <= WriteData;
                    when "0101" => R(5)  <= WriteData;
                    when "0110" => R(6)  <= WriteData;
                    when "0111" => R(7)  <= WriteData;
                    when "1000" => R(8)  <= WriteData;
                    when "1001" => R(9)  <= WriteData;
                    when "1010" => R(10) <= WriteData;
                    when "1011" => R(11) <= WriteData;
                    when "1100" => R(12) <= WriteData;
                    when "1101" => R(13) <= WriteData;
                    when "1110" => R(14) <= WriteData;
                    when "1111" => R(15) <= WriteData;
                    when others => null;
            end case;

end if;
end if;
END PROCESS;

PROCESS (clk, ReadRegA, R)
  BEGIN
   if rising_edge (clk) then
            case ReadRegA is                -- A saida Data1 recebe o conteudo do registrador
selecionado
                                            -- pelos bits 11 a 8 da instrucao (primeiro
operando)
                    when "0000" => Data_A <=R(0);
                    when "0001" => Data_A <=R(1);
                    when "0010" => Data_A <=R(2);
                    when "0011" => Data_A <=R(3);
                    when "0100" => Data_A <=R(4);
                    when "0101" => Data_A <=R(5);
                    when "0110" => Data_A <=R(6);
                    when "0111" => Data_A <=R(7);
                    when "1000" => Data_A <=R(8);
```

```
                        when "1001" => Data_A <=R(9);
                        when "1010" => Data_A <=R(10);
                        when "1011" => Data_A <=R(11);
                        when "1100" => Data_A <=R(12);
                        when "1101" => Data_A <=R(13);
                        when "1110" => Data_A <=R(14);
                        when "1111" => Data_A <=R(15);
                        when others => Data_A <="0000000000000000";
            end case;
end if;
END PROCESS ;

 PROCESS(clk, readRegB, R)
 BEGIN
            if rising_edge (clk) then
       case ReadRegB is -- A saida Data2 recebe o conteudo do registrador selecionado -
                                                    --  pelos  bits  7  a  4  da  instrucao  (segundo
operando)
             when "0000" => Data_B <=R(0);
             when "0001" => Data_B  <=R(1);
             when "0010" => Data_B  <=R(2);
             when "0011" => Data_B  <=R(3);
             when "0100" => Data_B  <=R(4);
             when "0101" => Data_B <=R(5);
             when "0110" => Data_B  <=R(6);
             when "0111" => Data_B  <=R(7);
             when "1000" => Data_B  <=R(8);
             when "1001" => Data_B  <=R(9);
             when "1010" => Data_B  <=R(10);
             when "1011" => Data_B  <=R(11);
             when "1100" => Data_B  <=R(12);
             when "1101" => Data_B  <=R(13);
             when "1110" => Data_B  <=R(14);
             when "1111" => Data_B  <=R(15);
                        when others => Data_B  <="0000000000000000";
            end case;
end if;
END PROCESS ;
--y<=R(13);
--AcInt<=y(0);

--regtres<=R(3);
 rzero<=R(0);
 rum<= R(1);
 rdois<=R(2);
 rtres<= R(3);
 rquatro<= R(4);
 rcinco<= R(5);
 rseis<=R(6);
 rsete<=R(7);
 roito<=R(8);
 rnove<=R(9);
 ra<=r(10);
 rb<= R(11);
 rc<=R(12);
 rd<=R(13);
 re<=R(14 );
 rf<=r(15);
```

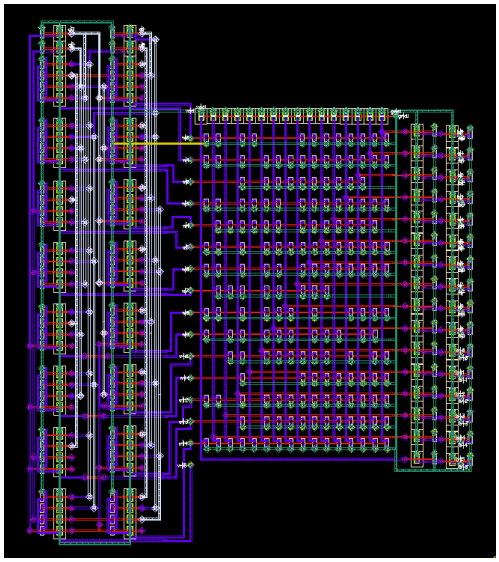
```
-- rzero<=regtres(0);
-- rum<= regtres(1);
-- rdois<=regtres(2);
-- rtres<= regtres(3);
-- rquatro<= regtres(4);
-- rcinco<= regtres(5);
-- rseis<=regtres(6);
-- rsete<=regtres(7);
-- roito<=regtres(8);
-- rnove<=regtres(9);
-- ra<=regtres(10);
-- rb<= regtres(11);
-- rc<=regtres(12);
-- rd<=regtres(13);
--re<=regtres(14 );
--rf<=regtres(15);
END structural;
```

APÊNDICE C – LAYOUTS

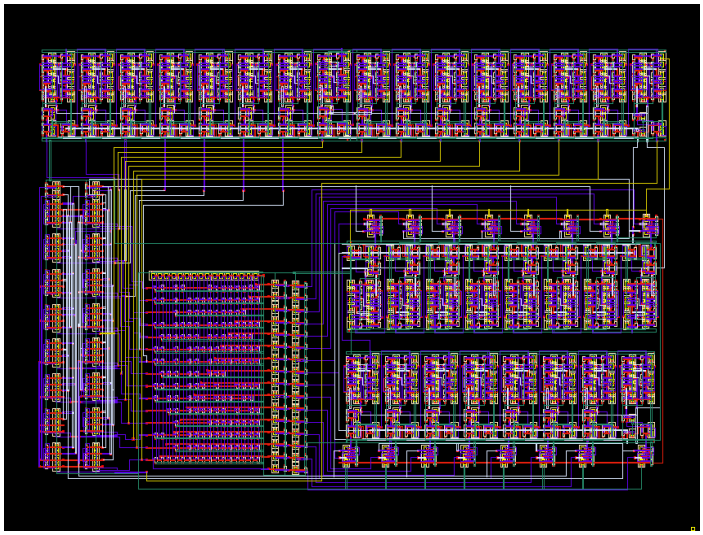
C.1 – MATRIZ DA ROM



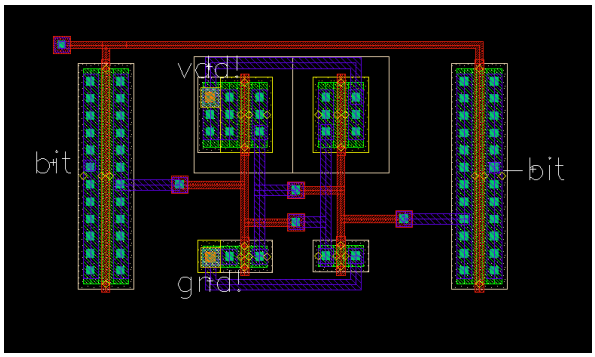
C.2 – MATRIZ DA ROM COM DECODIFICADOR DE LINHA



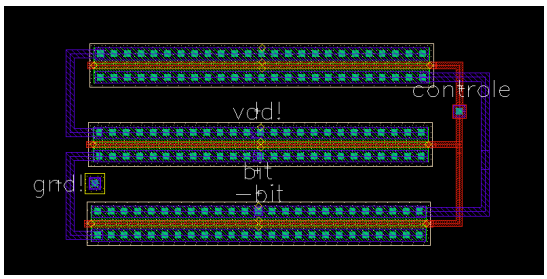
C.3 – ROM FINAL COM ESTRUTURAS DE TESTE



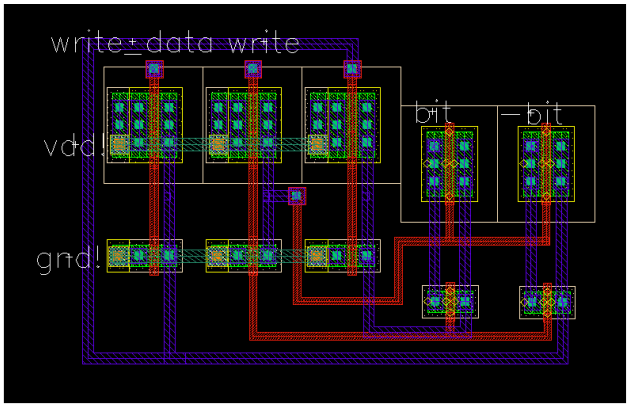
C.4 – CÉLULA 6-T DA MEMÓRIA SRAM



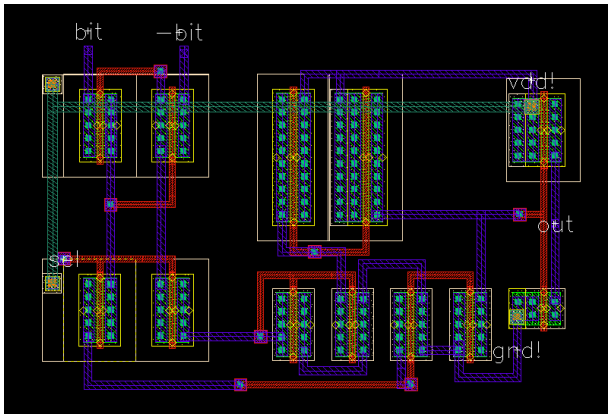
C.5 – PRÉ-CARGA CONTROLADA DA SRAM



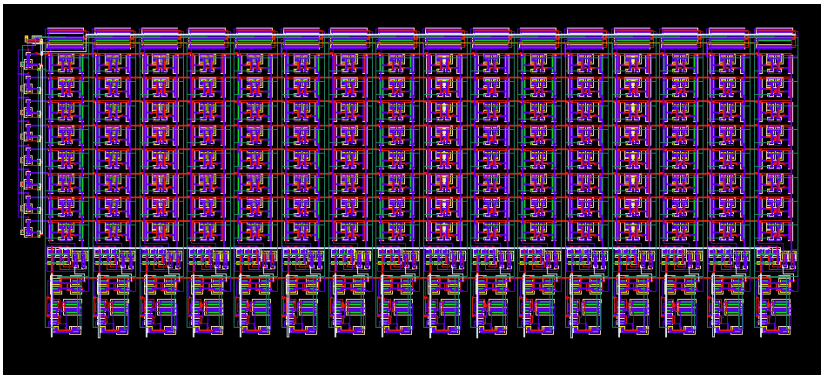
C.6 – CIRCUITO DE ESCRITA DA SRAM



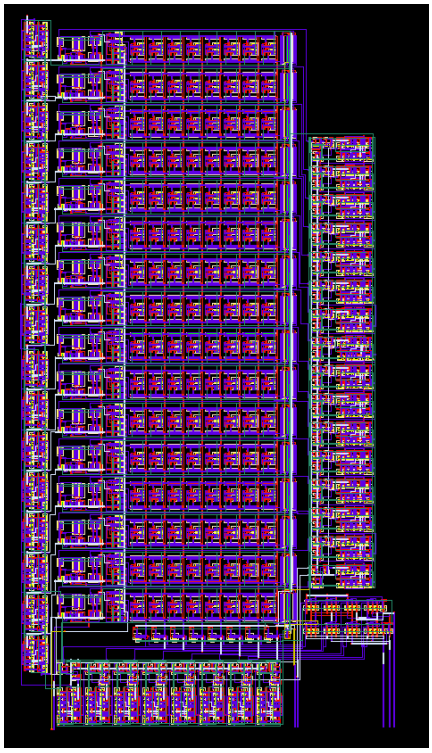
C.7 – AMPLIFICADOR SENSOR DE CORRENTE COM ESTÁGIO DE SAÍDA



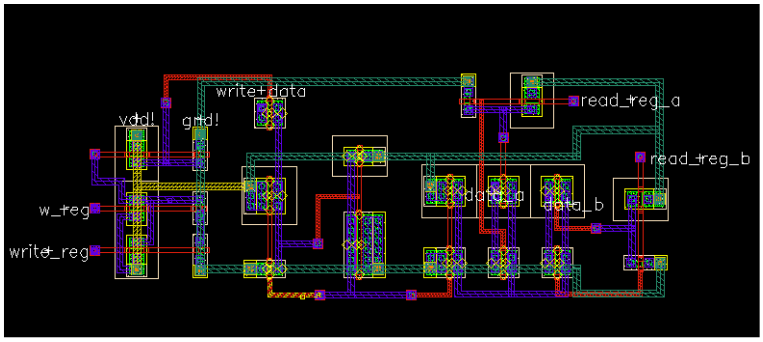
C.8 – MATRIZ SRAM 8X16 BITS



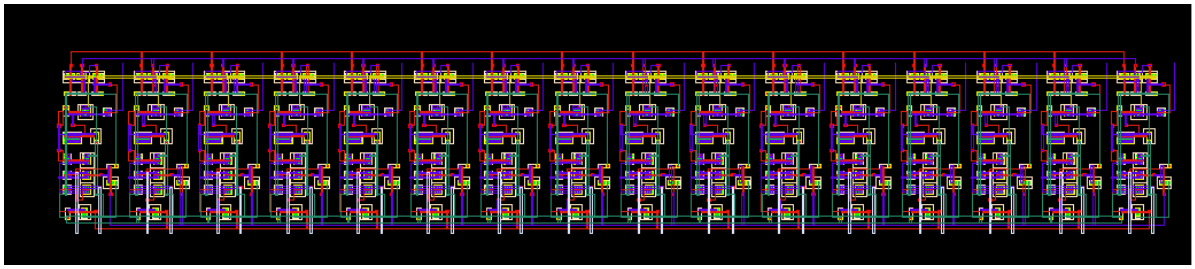
C.9 – MATRIZ SRAM COM DECODIFICADOR E CONVERSORES



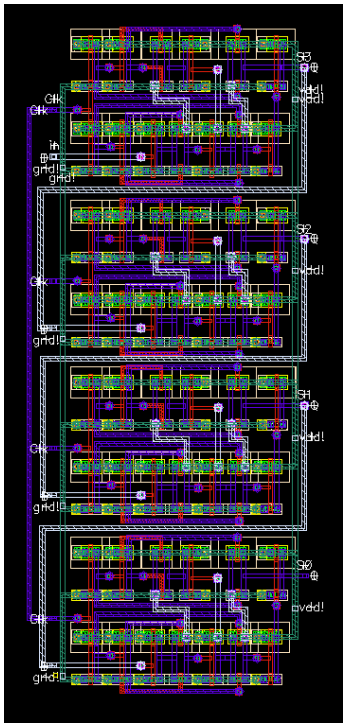
C.10 – CÉLULA DE UM REGISTRADOR



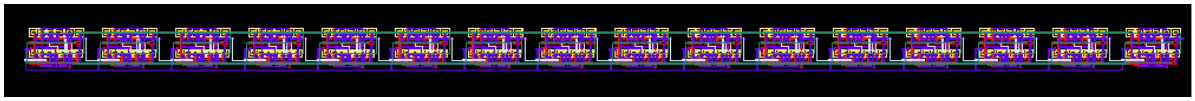
C.11 – REGISTRADOR DE 16 BITS



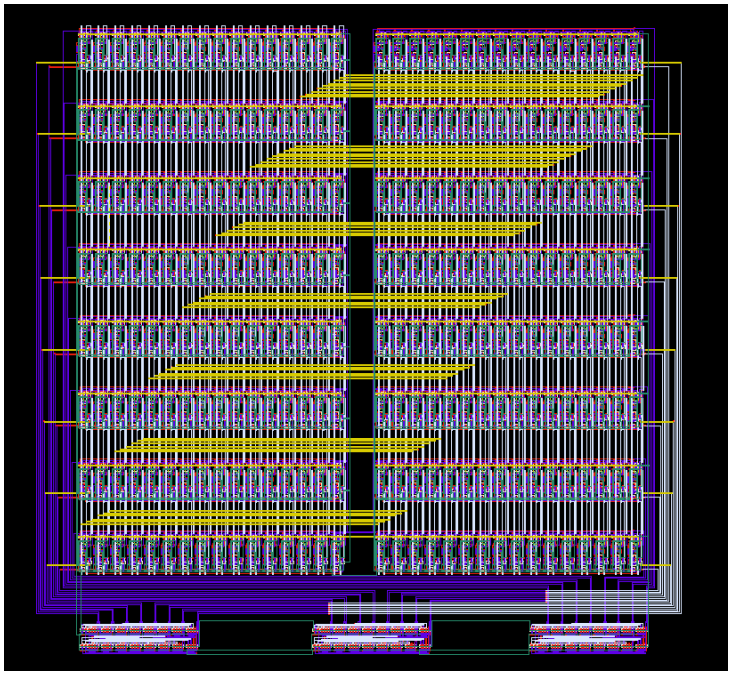
C.12 – CONVERSOR SERIAL-PARALELO DE 4 BITS



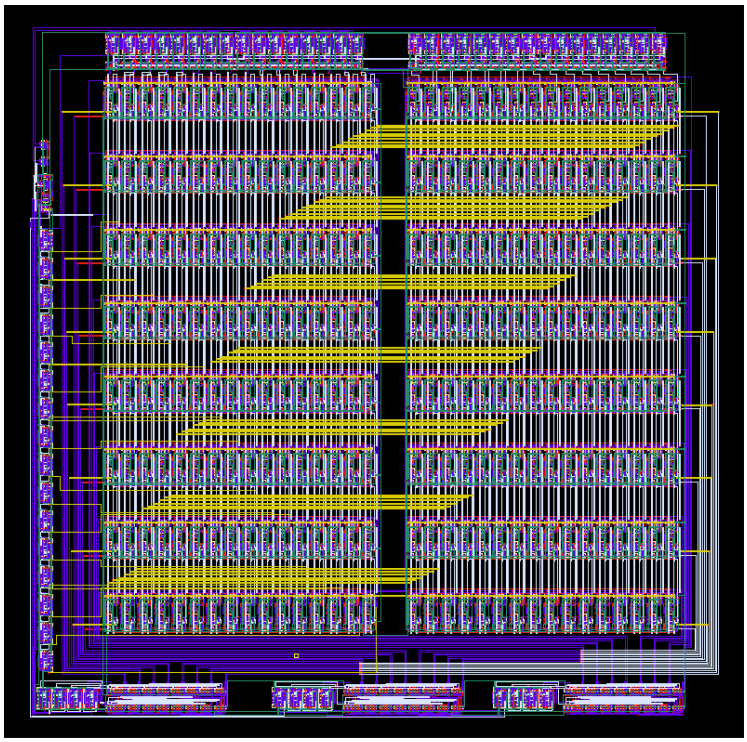
C.13 – CONVERSOR SERIAL-PARALELO DE 16 BITS



C.14 – BANCO DE REGISTRADORES



C.15 – BANCO DE REGISTRADORES COM CONVERSORES E TESTE



APÊNDICE D – ROTINA DE INICIALIZAÇÃO DA ROM

Projeto SCI
Software de Boot Serial 4
Alaercio Londe da Silva

1. Introdução

A interface serial comunica-se com o processador por meio de uma UART e uma memória ligada diretamente à interface e ao processador. É através desta posição de memória que a interface disponibiliza ao processador os dados para que possam ser armazenados em outra posição na mesma memória.

Pelas especificações, verifica-se que o processo de aquisição de dados deve funcionar da seguinte forma:

1- A interface disponibiliza o dado a ser processado no registrador mapeado para este fim.
2- É disparada uma interrupção ao processador, que pára suas atividades, e captura o dado.
3- O dado é enviado à memória por meio de operação de escrita.

As rotinas de inicialização deverão tratar do gerenciamento da estrutura de hardware de forma a habilitá-lo de acordo com a comunicação serial. Assim podemos formalizar um diagrama para o código:

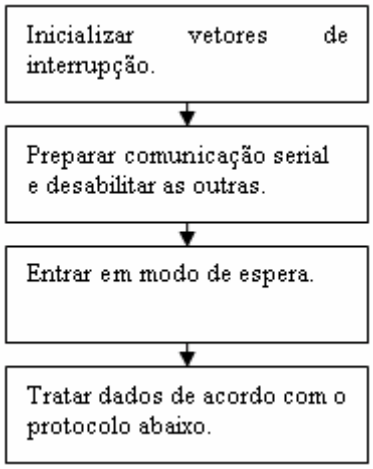


Figura 1: Visão Geral

2. Protocolo de comunicação

Um protocolo de comunicação deve ser assumido na porta serial de modo a prover a recepção e transmissão por esse canal.

Um modelo pode ser visualizado da seguinte forma:

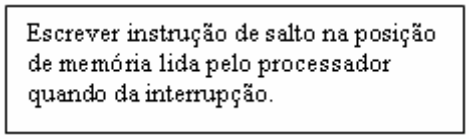
Código (16 bits)	Nº de Bytes (16bits)	Posição de memória de gravação (16bits)	Dados (N bits)

Figura 2: Protocolo

Código: Define que tipos de dados estão sendo enviados. Define uma palavra especial para reiniciar boot.
Nº de Bytes: Quantidade de Bytes dos campos: Dados, código e Nº de Bytes
XYXYXYXY---Programa.
XXYYXXYY---Constantes de operação, dados de erro, etc..
Posição de memória de gravação: posição de gravação definida pelo programador
Dados: Conteúdo do campo de dados.

3. Implementação por blocos

Partimos agora para um aprofundamento progressivo do projeto do software que vai fazer o controle do boot serial. Para isso começaremos com o primeiro bloco da **Figura 1**, inicializar vetores de interrupção, que é tratar a interrupção inicial vinda do hardware.



Este procedimento se torna importantíssimo, pois as interrupções serão ativas na memória RAM nos seus primeiros endereços. Logo teremos que guardar a instrução em tais posições de memória que realiza o salto para a posição onde se encontra a rotina que se quer executar. Este procedimento torna as interrupções flexíveis porque temos o controle das posições onde poderemos guardar os programas.

Agora teremos que escrever a rotina para comunicação serial direcionada pelo salto anterior, que começa pela verificação da disponibilidade da porta.

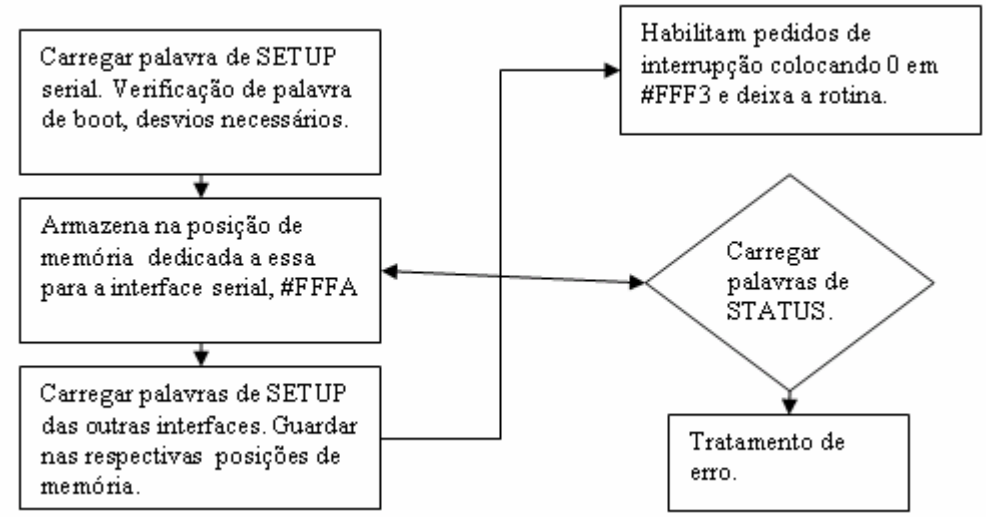


Figura 3: Primeira Rotina

Neste caso a palavra de STATUS é de muito importante, pois é através dela que a interface mostra sua capacidade de comunicação. Em caso positivo o Bit menos significativo conterá o valor zero (0) e caso negativo, incapacidade de comunicação, o valor será um (1). Nesta possibilidade aparecerá o código do erro nos outros bits da palavra de STATUS.

A palavra de SETUP serial contém, dentre outras funcionalidades, os bits 6 e 7 que habilitam transmissão e recepção respectivamente. Logo essa palavra tem que ser cuidadosamente confeccionada e armazenada na posição #FFFAh de modo a termos o funcionamento preciso da interface serial. Assim como nas outras interfaces teremos que desabilitar, neste caso, as interrupções geradas por tais interfaces.

Toda vez que uma interrupção é acionada o hardware usa a posição #FFF3 para desabilitar novos pedidos. Ele coloca o valor zero (1) no bit menos significativo para tal operação, assim toda vez que for deseja uma interrupção deve-se setar este bit para um (0).

Agora vamos entrar na fase de espera da interrupção serial para iniciarmos a comunicação.

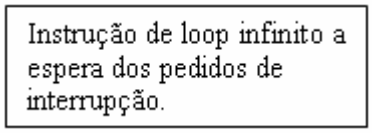


Figura 4: Modo de Espera

Neste caso o qual estamos trabalhando para um programa para a memória ROM e fazendo apenas comunicação serial, mas estas idéias serão expandidas para RF e ∑Δ.

Esta próxima rotina, então, deve ser capaz de adquirir os dados que estão chegando pela interface serial de acordo como protocolo de comunicação estabelecido. Não esquecendo que o nosso processador trabalha com dezesseis bits enquanto a interface com oito.

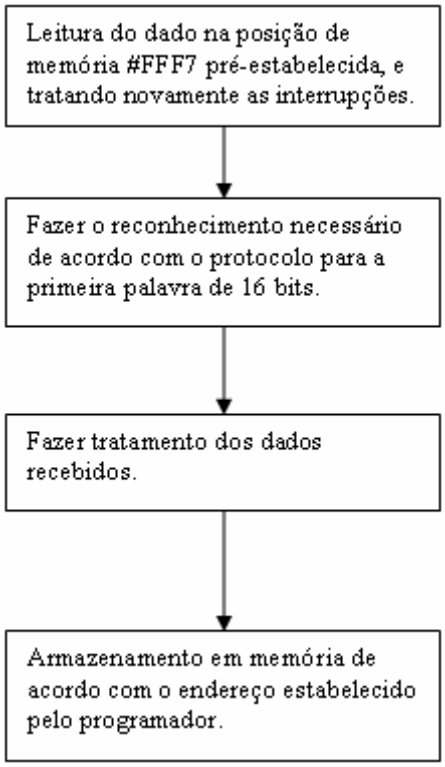


Figura 5: Dados Serial

Quando um byte for armazenado na posição de memória dedicado à interface serial uma interrupção será acionada indicando a sua chegada.

O próximo passo é fazer a leitura deste Byte (16 bits) sabendo que ele está gravado no endereço #FFF7h. De acordo com o protocolo de comunicação esta primeira palavra é o número de bytes do frame.

4. Implementação do Software

```
INI       ORG     $0000
SaltM     EQU     b11101110          código fictício que representa a instrução
SaltN     EQU     b00000001          Jal,$ra,0 que será armazenada  para o salto da int.
VAR1      EQU     3
VAR2      EQU     5
VAR3      EQU     7
COD       EQU     255                        codigo de reboot
Inicio    Add     $s0,$zero,$zero            s0 =0
          Addi    $t0,1
          Lui     $t1,$FF
          Addi    $t1,$F3                    t1 com $FFF3
          Sw      $t0,$t1,$zero              desabilitar interrupção geral $FFF3=1

          Add     $t1,$zero,$zero            entre outra coisa HABILITA INT. SERIAL
          Addi    $t1,b10101010              SETUP mais significativo
          Shift   $t1,-8                     t1 pega SETUP serial fictício
          Addi    $t1,b10101010              SETUP menos significativo
          Lui     $t2,$FF
          Addi    $t2,$FA                    t1 com FFFAh
          Sw      $t1,$t2,$zero              armazena SETUP serial

          Add     $t1,$zero,$zero            entre outra coisa DESABILITA INT. RF
          Addi    $t1,b11110000              SETUP mais significativo
          Shift   $t1,-8                     t1 pega SETUP RF fictício
          Addi    $t1,b11110000              SETUP menos significativo
          Lui     $t2,$FF
          Addi    $t2,$FF                    t1 com FFFFh
          Sw      $t1,$t2,$zero              armazena  primeira SETUP  RF

          Add     $t1,$zero,$zero            entre outra coisa DESABILITA INT. RF
          Addi    $t1,b11110001              SETUP mais significativo
          Shift   $t1,-8                     t1 pega SETUP RF fictício
          Addi    $t1,b11110001              SETUP menos significativo
          Lui     $t2,$FF
          Addi    $t2,$FE                    t1 com FFFEh
          Sw      $t1,$t2,$zero              armazena  segunda SETUP RF

          Add     $t1,$zero,$zero            entre outra coisa DESABILITA INT. SENSOR
          Addi    $t1,b00001111              SETUP mais significativo
          Shift   $t1,-8                     t1 pega SETUP Sensor fictício
          Addi    $t1,b00001111              SETUP menos significativo
          Lui     $t2,$FF
          Addi    $t2,$F6                    t1 com FFF6h
          Sw      $t1,$t2,$zero              armazena SETUP sensor

          Lui     $t1,$FF                    Verifica posição de STATUS
          Addi    $t1,$F9                    t1 com posição FFF9h
          Lw      $t2,$t1,$zero              t2 com  STATUS
          Addi    $s0,b00000001              s0 igual a 1
          And     $s1,$t2,$s0                verifica o STATUS serial
          Beq     $s1,$zero,LeSe             se s1 for zero interface pronta,STATUS =0, salto
          J       Inicio

LeSe      Add     $t1,$zero,$zero
          Addi    $t1,SaltM                  Salto mais significativo
          Shift   $t1,-8                     t1 pega Salto
          Addi    $t1,SaltN                  Salto menos significativo
          Add     $t0,$zero,$zero
          Addi    $t0,1024                   Salvamento de const. que possui instrução de salto no
endereço de int.
          Sw      $t1,$t0,$zero              Addi só suporta constantes de 8 bits, é necessário que o
montador faça o salto.

          Add     $s0,$zero,$zero
```

```
        Lui     $t1,$FF
        Addi    $t1,$F3             t1 com $FFF3
        Sw      $s0,$t1,$zero       habilitar interrupção geral $FFF3=0
LInf    Beq     $s0,$s0,0           loop infinito

RecInt  Add     $t0,$zero,$zero
        Addi    $t0,1
        Lui     $t1,$FF
        Addi    $t1,$F3             t1 com $FFF3
        Sw      $t0,$t1,$zero       desabilitar interrupção geral $FFF3=1

        Add     $t0,$zero,$zero      t0 para depois da interrupção
        Add     $t1,$zero,$zero
        Addi    $t0,$t0,b00000100   Reconhecimento interrupção RF
        Lui     $t1,$FF
        Addi    $t1,$F3
        Lw      $t1,$t1,$zero
        And     $t0,$t1,$t0
        Bne     $t0,$zero,SaltRF

        Add     $t0,$zero,$zero      t0 para depois da interrupção
        Add     $t1,$zero,$zero
        Addi    $t0,$t0,b0001000    Reconhecimento interrupção Serial
        Lui     $t1,$FF
        Addi    $t1,$F3
        Lw      $t1,$t1,$zero
        And     $t0,$t1,$t0
        Bne     $t0,$zero,SaltSE

        Add     $t0,$zero,$zero      t0 para depois da interrupção
        Add     $t1,$zero,$zero
        Addi    $t0,$t0,b00000010   Reconhecimento interrupção Sigma- Delta
        Lui     $t1,$FF
        Addi    $t1,$F3
        Lw      $t1,$t1,$zero
        And     $t0,$t1,$t0
        Bne     $t0,$zero,SaltSD

        Add     $t0,$zero,$zero      t0 para depois da interrupção
        Add     $t1,$zero,$zero
        Addi    $t0,$t0,b00010000   Reconhecimento interrupção Overflow
        Lui     $t1,$FF
        Addi    $t1,$F3
        Lw      $t1,$t1,$zero
        And     $t0,$t1,$t0
        Bne     $t0,$zero,SaltOverF

        Add     $t0,$zero,$zero      t0 para depois da interrupção
        Add     $t1,$zero,$zero
        Addi    $t0,$t0,b00100000   Reconhecimento interrupção endereçamento
        Lui     $t1,$FF
        Addi    $t1,$F3
        Lw      $t1,$t1,$zero
        And     $t0,$t1,$t0
        Bne     $t0,$zero,SaltEnd


SaltSE  Add     $t1,$zero,$zero
        Add     $t0,$zero,$zero
        Addi    $a0,b00000001       monitora o numero de palavras
        Lui     $a1,$FF
        Addi    $a1,$F7             a1 com FFF7h
        Lw      $s1,$a1,$zero       carrega  palavra serial
        Addi    $t0,b00000002

        Addi    $s2,VAR1            s2 igual a 3
        Blt     $s2,$t0,6           se t0 for maior que tres salta
```

```
            Addi    $t1,COD
            Beq     $t1,$s1,Inicio          Reboot
            Lui     $a2,$FF
            Addi    $a2,$F3
            Sw      $s0,$a2,$zero           interrupção habilitada
            J       LInf

            Addi    $s2,VAR2                s2 igual a 5
            Blt     $s2,$t0,5               se t0 for maior que cinco salta

            Sw      $s1,$s3,$zero           guarda palavra em s3, já pode vir a terceira palavra,numero de
bytes
            Lui     $a2,$FF
            Addi    $a2,$F3
            Sw      $s0,$a2,$zero           interrupção habilitada
            J       LInf

            Addi    $s2,VAR3                s2 igual a 7
            Blt     $s2,$t0,Salva           se t0 for maior que sete salta

            Sw      $s1,$a3,$zero           guarda palavra em a3, já pode vir a quarta palavra,posição de
memória
            Lui     $a2,$FF
            Addi    $a2,$F3
            Sw      $s0,$a2,$zero           interrupção habilitada
            J       LInf

Salva       Sw      $s1,$a3,$zero           começa a salvar na posição de memória desejada
            Addi    $a3,b00000001           s1 está com o segundo byte,incrementa memória
            Beq     $s3,$a0,Fim             verifica último Byte
            Lui     $a2,$FF
            Addi    $a2,$F3
            Sw      $s0,$a2,$zero           interrupção habilitada
            J       LInf
Fim         J       $a3,0                   salta para a memória Ram, lá teremos o código dos dado
enviados.
```

5. Detalhamento do Software

Este software será gravado na memória ROM do chip onde fará uma inicializarção do sistema dando ênfase à porta serial para a comunicação com o exterior, com o intuito de se efetuar os primeiros testes no processador. Novas Versões deste Software serão desenvolvidas principalmente para a interface RF bem como o melhoramento intensivo desse será apresentado.

Na primeira linha determinamos que a posição inicial da memória onde as instruções serão gravadas é zero. Da segunda à quarta linha definimos as constantes SaltM e SaltN que são na verdade,juntas, uma única constante de dezesseis bits. Esta constante possui o código a ser gravado na posição de memória ativada pela interrupção de forma a fazer o salto para a rotina de reconhecimento de interrupção a ser executada.Uma rotina de verificação é feita para o descobrimento da interface que gerou a interrupção usando para isso as informações da posição de memória $FFF3.

A partir de “Início” até “LeSe” no programa acima foi tratada as palavras de configuração do SETUP de todas as interfaces, habilitando a interface serial e desabilitando todas as outras. Desabilitar as interrupções de um modo geral foi a primeira coisa a ser feita através do endereço FFF3h, habilitando o seu bit menos significativo para um. E no final Testamos a disponibilidade de comunicação da serial através do registrador de STATUS, Se pronta passa para a fase de recepção caso contrário configura-se novamente.

De “LeSe” Até “LInf” São feitos os preparativos de salto quando do advento da interrupção. Escrevendo no endereço 1024 decimal,primeira posição da memória RAM, a instrução de salto. Esta instrução deve saltar para o endereço armazenado em $ra, registrador de retorno, somado de uma unidade,operação do hardware, para executar a rotina requerida. Assim o requerimento de interrupção pode ser acionado, entrando-se em um loop de espera.

De “RecInt” até “SaltSE” esta representada a rotina de reconhecimento de todas as interrupções.

De “SaltSE” até “Salva” é feita a leitura do primeiro byte (16 bits) que chegou pela serial e análise referente ao protocolo para a o reconhecimento do byte de reboot, já que esta rotina faz parte da interrupção.A leitura do segundo byte (16 bits) que chegou pela serial referente ao protocolo para a o número de bytes do frame. A leitura do terceiro byte (16 bits) que chegou pela serial referente ao protocolo para a posição de memória escolhida pelo programador. Aciona-se novamente a interrupção que foi desabilitada pelo hardware a cada nova palavra. A partir de então todos os bytes que chegam são de dados e são direcionados para a memória.

Por fim, um salto é dado para a posição de memória RAM inicial, fazendo o tratamento dos dados que chegaram.

APÊNDICE E – MODELO DE TRANSISTOR UTILIZADO

A ferramenta do CADENCE na qual as estruturas projetadas neste trabalho foram simuladas foi o Spectre, que utiliza o modelo BSIM3v3 para os transistores MOS. Os parâmetros desse modelo, bem como todas as suas características, podem ser encontrados na *homepage*:

http://www-device.eecs.berkeley.edu/~bsim3/

Nas Tabelas E.1 a E.5 são listados e descritos alguns dos parâmetros utilizados neste modelo, bem como os valores padronizados. Esses dados foram obtidos no Manual BSIM3v3.2.2 (UC Berkeley, 1999). Os valores dos parâmetros utilizados pelo Spectre para a tecnologia 0.35 um CMOS foram fornecidos pela AMS, e podem ser encontrados na *homepage*:

http://asic.austriamicrosystems.com

Tabela E. 1 - Parâmetros DC (BSIM3v3.2.2 Manual – UC Berkeley, 1999)

Symbols used in equation	Symbols used in SPICE	Description	Default	Unit	Note
Vth0	vth0	Threshold voltage @Vbs=0 for Large L.	0.7 (NMOS) -0.7 (PMOS)	V	nI-1
VFB	vfb	Flat-band voltage	Calculated	V	nI-1
K1	k1	First order body effect coefficient	0.5	$V^{1/2}$	nI-2
K2	k2	Second order body effect coefficient	0.0	none	nI-2
K3	k3	Narrow width coefficient	80.0	none	
K3b	k3b	Body effect coefficient of k3	0.0	1/V	
W0	w0	Narrow width parameter	2.5e-6	m	
Nlx	nlx	Lateral non-uniform doping parameter	1.74e-7	m	
Vbm	vbm	Maximum applied body bias in Vth calculation	-3.0	V	
Dvt0	dvt0	first coefficient of short-channel effect on Vth	2.2	none	
Dvt1	dvt1	Second coefficient of short-channel effect on Vth	0.53	none	
Dvt2	dvt2	Body-bias coefficient of short-channel effect on Vth	-0.032	1/V	
Dvt0w	dvt0w	First coefficient of narrow width effect on Vth for small channel length	0	1/m	
Dvt1w	dvtw1	Second coefficient of narrow width effect on Vth for small channel length	5.3e6	1/m	
Dvt2w	dvt2w	Body-bias coefficient of narrow width effect for small channel length	-0.032	1/V	
μ0	u0	Mobility at Temp = Tnom NMOSFET PMOSFET	 670.0 250.0	cm^2/Vs	
Ua	ua	First-order mobility degradation coefficient	2.25E-9	m/V	
Ub	ub	Second-order mobility degradation coefficient	5.87E-19	$(m/V)^2$	
Uc	uc	Body-effect of mobility degradation coefficient	mobMod =1, 2: -4.65e-11 mobMod =3: -0.046	m/V^2 1/V	

Symbols used in equation	Symbols used in SPICE	Description	Default	Unit	Note
vsat	vsat	Saturation velocity at Temp − Tnom	8.0E4	m/sec	
A0	a0	Bulk charge effect coefficient for channel length	1.0	none	
Ags	ags	gate bias coefficient of Abulk	0.0	1/V	
B0	b0	Bulk charge effect coefficient for channel width	0.0	m	
B1	b1	Bulk charge effect width offset	0.0	m	
Keta	keta	Body-bias coefficient of bulk charge effect	-0.047	1/V	
A1	a1	First non-saturation effect parameter	0.0	1/V	
A2	a2	Second non-saturation factor	1.0	none	
Rdsw	rdsw	Parasitic resistance per unit width	0.0	Ω-μm^{Wr}	
Prwb	prwb	Body effect coefficient of Rdsw	0	$V^{-1/2}$	
Prwg	prwg	Gate bias effect coefficient of Rdsw	0	1/V	
Wr	wr	Width Offset from Weff for Rds calculation	1.0	none	
Wint	wint	Width offset fitting parameter from I-V without bias	0.0	m	
Lint	lint	Length offset fitting parameter from I-V without bias	0.0	m	
dWg	dwg	Coefficient of Weff's gate dependence	0.0	m/V	
dWb	dwb	Coefficient of Weff's substrate body bias dependence	0.0	$m/V^{1/2}$	
Voff	voff	Offset voltage in the subthreshold region at large W and L	-0.08	V	
Nfactor	nfactor	Subthreshold swing factor	1.0	none	
Eta0	eta0	DIBL coefficient in subthreshold region	0.08	none	
Etab	etab	Body-bias coefficient for the subthreshold DIBL effect	-0.07	1/V	
Dsub	dsub	DIBL coefficient exponent in subthreshold region	drout	none	
Cit	cit	Interface trap capacitance	0.0	F/m^2	
Cdsc	cdsc	Drain/Source to channel coupling capacitance	2.4E-4	F/m^2	
Cdscb	cdscb	Body-bias sensitivity of Cdsc	0.0	F/Vm^2	
Cdscd	cdscd	Drain-bias sensitivity of Cdsc	0.0	F/Vm^2	
Pclm	pclm	Channel length modulation parameter	1.3	none	
Pdiblc1	pdiblc1	First output resistance DIBL effect correction parameter	0.39	none	

Symbols used in equation	Symbols used in SPICE	Description	Default	Unit	Note
Pdiblc2	pdiblc2	Second output resistance DIBL effect correction parameter	0.0086	none	
Pdiblcb	pdiblcb	Body effect coefficient of DIBL correction parameters	0	1/V	
Drout	drout	L dependence coefficient of the DIBL correction parameter in Rout	0.56	none	
Pscbe1	pscbe1	First substrate current body-effect parameter	4.24E8	V/m	
Pscbe2	pscbe2	Second substrate current body-effect parameter	1.0E-5	m/V	nI-3
Pvag	pvag	Gate dependence of Early voltage	0.0	none	
δ	delta	Effective Vds parameter	0.01	V	
Ngate	ngate	poly gate doping concentration	0	cm^{-3}	
$\alpha 0$	alpha0	The first parameter of impact ionization current	0	m/V	
$\alpha 1$	alpha1	Isub parameter for length scaling	0.0	1/V	
$\beta 0$	beta0	The second parameter of impact ionization current	30	V	
Rsh	rsh	Source drain sheet resistance in ohm per square	0.0	Ω/ square	
Js0sw	jssw	Side wall saturation current density	0.0	A/m	
Js0	js	Source drain junction saturation current per unit area	1.0E-4	A/m^2	
ijth	ijth	Diode limiting current	0.1	A	nI-3

Tabela E. 2 - Parâmetros C-V (BSIM3v3.2.2 Manual – UC Berkeley, 1999)

Symbols used in equation	Symbols used in SPICE	Description	Default	Unit	Note
Xpart	xpart	Charge partitioning flag	0.0	none	
CGS0	cgso	Non LDD region source-gate overlap capacitance per channel length	calculated	F/m	nC-1
CGD0	cgdo	Non LDD region drain-gate overlap capacitance per channel length	calculated	F/m	nC-2
CGB0	cgbo	Gate bulk overlap capacitance per unit channel length	0.0	F/m	
Cj	cj	Bottom junction capacitance per unit area at zero bias	5.0e-4	F/m^2	

Symbols used in equation	Symbols used in SPICE	Description	Default	Unit	Note
Mj	mj	Bottom junction capacitance grating coefficient	0.5		
Mjsw	mjsw	Source/Drain side wall junction capacitance grading coefficient	0.33	none	
Cjsw	cjsw	Source/Drain side wall junction capacitance per unit area	5.E-10	F/m	
Cjswg	cjswg	Source/drain gate side wall junction capacitance grading coefficient	Cjsw	F/m	
Mjswg	mjswg	Source/drain gate side wall junction capacitance grading coefficient	Mjsw	none	
Pbsw	pbsw	Source/drain side wall junction built-in potential	1.0	V	
Pb	pb	Bottom built-in potential	1.0	V	
Pbswg	pbswg	Source/Drain gate side wall junction built-in potential	Pbsw	V	
CGS1	cgsl	Light doped source-gate region overlap capacitance	0.0	F/m	
CGD1	cgdl	Light doped drain-gate region overlap capacitance	0.0	F/m	
CKAPPA	ckappa	Coefficient for lightly doped region overlap capacitance Fringing field capacitance	0.6	V	
Cf	cf	fringing field capacitance	calculated	F/m	nC-3
CLC	clc	Constant term for the short channel model	0.1E-6	m	
CLE	cle	Exponential term for the short channel model	0.6	none	
DLC	dlc	Length offset fitting parameter from C-V	lint	m	
DWC	dwc	Width offset fitting parameter from C-V	wint	m	
Vfbcv	vfbcv	Flat-band voltage parameter (for capMod=0 only)	-1	V	
noff	noff	CV parameter in Vgsteff,CV for weak to strong inversion	1.0	none	nC-4
voffcv	voffcv	CV parameter in Vgsteff,CV for week to strong inversion	0.0	V	nC-4
acde	acde	Exponential coefficient for charge thickness in capMod=3 for accumulation and depletion regions	1.0	m/V	nC-4
moin	moin	Coefficient for the gate-bias dependent surface potential	15.0	none	nC-4

Tabela E. 3 - Parâmetros de temperatura (BSIM3v3.2.2 Manual – UC Berkeley, 1999)

Symbols used in equation	Symbols used in SPICE	Description	Default	Unit	Note
Tnom	tnom	Temperature at which parameters are extracted	27	°C	
μte	ute	Mobility temperature exponent	-1.5	none	
Kt1	kt1	Temperature coefficient for threshold voltage	-0.11	V	
Kt1l	kt1l	Channel length dependence of the temperature coefficient for threshold voltage	0.0	Vm	
Kt2	kt2	Body-bias coefficient of Vth temperature effect	0.022	none	
Ua1	ua1	Temperature coefficient for Ua	4.31E-9	m/V	
Ub1	ub1	Temperature coefficient for Ub	-7.61E-18	$(m/V)^2$	
Uc1	uc1	Temperature coefficient for Uc	mobMod=1, 2: -5.6E-11 mobMod=3: -0.056	m/V^2 1/V	
At	at	Temperature coefficient for saturation velocity	3.3E4	m/sec	
Prt	prt	Temperature coefficient for Rdsw	0.0	Ω-μm	
At	at	Temperature coefficient for saturation velocity	3.3E4	m/sec	
nj	nj	Emission coefficient of junction	1.0	none	
XTI	xti	Junction current temperature exponent coefficient	3.0	none	
tpb	tpb	Temperature coefficient of Pb	0.0	V/K	
tpbsw	tpbsw	Temperature coefficient of Pbsw	0.0	V/K	
tpbswg	tpbswg	Temperature coefficient of Pbswg	0.0	V/K	
tcj	tcj	Temperature coefficient of Cj	0.0	1/K	
tcjsw	tcjsw	Temperature coefficient of Cjsw	0.0	1/K	
tcjswg	tcjswg	Temperature coefficient of Cjswg	0.0	1/K	

Tabela E. 4 - Parâmetros de processo (BSIM3v3.2.2 Manual – UC Berkeley, 1999)

Symbols used in equation	Symbols used in SPICE	Description	Default	Unit	Note
Tox	tox	Gate oxide thickness	1.5e-8	m	
Toxm	toxm	Tox at which parameters are extracted	Tox	m	nl-3
Xj	xj	Junction Depth	1.5e-7	m	
$\gamma 1$	gamma1	Body-effect coefficient near the surface	calcu-lated	$V^{1/2}$	nl-5
$\gamma 2$	gamma2	Body-effect coefficient in the bulk	calcu-lated	$V^{1/2}$	nl-6
Nch	nch	Channel doping concentration	1.7e17	$1/cm^3$	nl-4
Nsub	nsub	Substrate doping concentration	6e16	$1/cm^3$	
Vbx	vbx	Vbs at which the depletion region width equals xt	calcu-lated	V	nl-7
Xt	xt	Doping depth	1.55e-7	m	

Tabela E. 5 - Faixa de parâmetros geométricos (BSIM3v3.2.2 Manual – UC Berkeley, 1999)

Symbols used in equation	Symbols used in SPICE	Description	Default	Unit	Note
Lmin	lmin	Minimum channel length	0.0	m	
Lmax	lmax	Maximum channel length	1.0	m	
Wmin	wmin	Minimum channel width	0.0	m	
Wmax	wmax	Maximum channel width	1.0	m	
binUnit	binunit	Bin unit scale selector	1.0	none	