UNIVERSIDADE DO VALE DO RIO DOS SINOS - UNISINOS
UNIDADE ACADÊMICA DE PESQUISA E PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA
NÍVEL MESTRADO PROFISSIONAL

GABRIEL ANTÔNIO JAEGER

DESENVOLVIMENTO E ANÁLISE COMPARATIVA DE SOLUÇÕES PARA TESTE FUNCIONAL DE MEMÓRIA DRAM EM TESTADOR AUTOMÁTICO INDUSTRIAL VERSUS TESTADOR MANUAL DE BANCADA

SÃO LEOPOLDO
2019

Gabriel Antônio Jaeger

DESENVOLVIMENTO E ANÁLISE COMPARATIVA DE SOLUÇÕES PARA TESTE FUNCIONAL DE MEMÓRIA DRAM EM TESTADOR AUTOMÁTICO INDUSTRIAL VERSUS TESTADOR MANUAL DE BANCADA

Dissertação apresentada como requisito parcial para obtenção do título de Mestre em Engenharia Elétrica, pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade do Vale do Rio dos Sinos - UNISINOS.

Orientador: Prof[a]. Dr[a]. Margrit Reni Krug

Coorientador: Me. Marcelo de Souza Moraes

São Leopoldo

2019

J22d Jaeger, Gabriel Antônio.
Desenvolvimento e análise comparativa de soluções para teste funcional de memória DRAM em testador automático industrial versus testador manual de bancada. – 2019.
109 f. : il. color. ; 30 cm.

Dissertação (mestrado) – Universidade do Vale do Rio dos Sinos, Programa de Pós-Graduação em Engenharia Elétrica, São Leopoldo, 2019.
"Orientador: Prof.ª Dr.ª Margrit Reni Krug ; Coorientador: Me. Marcelo de Souza Moraes."

1. Testes elétricos. 2. Memória - Testes. 3. Memórias SDRAM DDR4. 4. Cobertura de falhas. 5. Análise de falhas. I. Título.

CDU 621.3

Dados Internacionais de Catalogação na Publicação (CIP)
(Bibliotecária: Bruna Sant'Anna – CRB 10/2360)

RESUMO

A qualificação do processo produtivo poderá garantir a qualidade e a confiabilidade desejada dos produtos produzidos. Para isso, torna-se indispensável a compreensão do comportamento funcional dos componentes eletrônicos e das principais características de suas falhas. Por outro lado, o constante avanço tecnológico e o aumento na complexidade dos dispositivos agrega dificuldade na etapa de teste, pois com o aumento da densidade e da taxa de transmissão de dados das memórias, além das estruturas cada vez mais complexas, aumentam-se os desafios relacionados às estratégias de teste aplicadas, ao mesmo tempo que impõem a necessidade de um conhecimento aprofundado dos dispositivos desenvolvidos. Além disso, a constante evolução do mercado brasileiro no setor de semicondutores apresenta uma necessidade ainda maior do aperfeiçoamento dos processos e capacidade de testes de nível industrial, a fim de agregar valor, qualidade e gerar competitividade com o mercado internacional. Baseando-se neste contexto teve-se como objetivo deste trabalho o desenvolvimento e validação de um programa de teste elétrico funcional para memórias para utilização em um equipamento ATE (*Automatic Test Equipment*) industrial de alto desempenho, e posteriormente uma análise comparativa entre a solução de teste desenvolvida e uma existente em um testador de bancada (Turbocats TCE3200LP). Os dispositivos alvo deste trabalho foram as memórias SDRAM DDR4. A plataforma utilizada no projeto foi o ATE Magnum V da Teradyne, para o qual foi desenvolvido o programa de teste, que utilizou alguns dos principais algoritmos de teste de memória encontrados na literatura e adequados ao teste de memórias SDRAM. A validação funcional do desenvolvimento do teste, bem como a análise comparativa, foi realizada através da análise de aderência entre falhas detectadas pelo programa de teste elaborado no ATE e as falhas detectadas através dos algoritmos existente na plataforma TCE3200LP. O nível de aderência dos resultados entre a solução de teste desenvolvida e os testador de bancada foi de 47%. Entretanto, a aderência entre o teste no ATE e as amostras selecionadas foi de aproximadamente 80%, apresentando maior similaridade com a condição real de cada amostra se comparado ao resultado obtido pelo TCE2300LP. Em outras palavras, o índice de cobertura de falhas da solução de teste desenvolvida apresenta melhores resultados do que a solução do testador de bancada TCE3200LP, sendo uma alternativa interessante para utilização em análises de falhas e caracterização de defeitos.

Palavras-chaves: Memórias SDRAM DDR4, Teste Elétrico Funcional, Algoritmos de Teste, Cobertura de Falhas, Análise de Falhas.

ABSTRACT

The qualification of the manufacturing process can guarantee the desired quality and reliability of the products produced. For this, it is essential to understand the functional behavior of electronic components and the main characteristics of their failures. On the other hand, the constant technological advancement and the increase in the complexity of the devices adds difficulty in the test steps, because with the increase of memory density and data speed rate, besides the increasingly complex structures, the challenges related to the applied test strategies, while imposing the need for a thorough knowledge of the developed devices. In addition, the constant evolution of the Brazilian market in the semiconductor sector presents an even greater need for process improvement and industrial level testing capabilities, in order to add value, quality and generate competitiveness with the international market. Based on this context, the objective of this work is the development and validation of a functional electrical memory test program for use in a high performance industrial *Automatic Test Equipment* (ATE), and later a comparative analysis between the developed test solution and a benchtop tester (*Turbocats TCE3200LP*). The target devices of this work are the DDR4 SDRAM memories. The test platform used in the project was Teradyne's ATE Magnum V, for which one the test program was developed, which used some of the main memory testing algorithms found in the literature and suitable for SDRAM memory testing. The functional validation of the test development, as well as the comparative analysis, was performed through the results adherence analysis between failures detected by the ATE test program and the failures detected through the existing algorithms in the TCE3200LP platform. The adherence level of the results between the developed test solution and the bench testers was 47%. However, the adherence between the ATE test and the selected samples was approximately 80%, showing greater similarity with the actual condition of each sample compared to the result obtained by the TCE2300LP. In other words, the failure coverage index of the developed test solution yields better results than the TCE3200LP bench tester solution and is an interesting alternative for use in fault analysis and defect characterization.

Key-words: SDRAM DDR4 Memories, Functional Electrical Test, Test Algorithms, Fault Coverage, Failure Analysis.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AP	*Auto-Precharge*
ACT	*Activate Command* (Comando de Ativação da memória SDRAM)
ATE	*Automatic Test Equipment*
APG	*Automatic Pattern Generator* (Gerador Automático de Padrões)
BA	*Bank Address* (Endereço de Bancos)
BG	*Bank Group* (Grupos de Bancos)
BC	*Burst Chop*
BC (2)	*Complementary Bit Line*
BC4	*Burst Chop 4 Bits*
BIST	*Built-in Self Test*
BNDES	Banco Nacional de Desenvolvimento Econômico e Social
BL	*Bit Line*
BL (2)	*Burst Lenght*
BL8	*Burst Lenght 8 bits*
BT	*True Bit Line*
BT (2)	*Base Test*
CAS	*Column Adress Strobe*
CF	*Coupling Fault* (Falhas de Acoplamento)
CI	Circuito integrado
CL	*CAS Latency*
Cmd	*Command* (Comando)
CS	*Chip Select*
CK	*Clock*
CKE	*Clock Enable*

DLL	*Delay Lock Loops* (Fixador de atrasos de ciclos em memórias SDRAM)
DQ	*Data Query*
DQS	*Data Query Strobe*
DPS	*DUT Power Supply* (Fonte de Alimentação dos DUTs)
DM	*Data Mask*
dppm	Nível de defeitos por milhão
DUT	*Device Under Test* (Dispositivo em teste)
DRAM	*Dynamic Random-Access Memory* (Memória de Acesso Aleatório Dinâmica)
DDR	*Double Data Rate* (Taxa de Dados Dupla)
DDR4	*Double Data Rate 4th Generation* (Classe de memória DRAM de 4ª Geração)
DSA	*Device Specific Adapter* (Adaptador Específico para o Dispositivo)
ECR	*Error Catch RAM*
F	*Fail* (Reprovado)
FBGA	*Fine Pitch Ball Grid Array*
FFM	*Functional Fault Model* (Modelo de Falhas Funcionais)
FINEP	Financiadora Nacional de Estudos e Projetos
HV	*High Voltage* (Pinos de Alta Tensão)
IO	*Input/Output* (Entradas e Saídas)
JEDEC	*Joint Electronic Device Engineering Council*
Mbps	Mega-bits por segundo
MG5	Testador ATE Magnum V
MR	*Mode Register*
NPFS	*Neighborhood Pattern Sensitive Faults* (Falhas de sensibilização por padrões de vizinhança)
ODT	*On-Die Termination*
O/S	*Open / Short* (Circuito Aberto / Curto-circuito)

P	*Pass* (Aprovado)
PADIS	Programa de Apoio ao Desenvolvimento da Indústria de Semicondutores
PCB	*Printed Circuit Board* (Placa de Circuito Impresso)
PE	*Pin Electronics*
RAM	*Random Access Memory* (Memória de Acesso Aleatório)
RAS	*Row Adress Strobe*
ROM	*Read Only Memory* (Memória de Somente Leitura)
ROM	*Read Only Memory* (Memória de Somente Leitura)
RD	*Read* (Leitura)
RDC	*Read Data Line Complementary*
RDT	*Read Data Line True*
R/W	*Read /Write* (Operação de Leitura/Escrita)
SRAM	*Static Random Access Memory* (Memória Estática de Acesso Aleatório)
SC	*Stress Combination* (Combinação de Estresse)
SDR	*Single Data Rate* (Taxa de Dados Única)
SDRAM	*Synchronous Dynamic Random-Access Memory* (Memória Síncrona de Acesso Aleatório Dinâmica)
TDQS	*Termination Data Strobe*
TDR	*Time Domain Reflectometry* (Reflectometria no Domínio do Tempo)
TG	*Timing Generator* (Gerador de Tempos)
UI	*User Interface* (Interface do Usuário)
UPH	Unidades por Hora
VDD	*Power Supply* (Alimentação)
VLSI	*Very Large Scale Integrated Circuit*
VSS	*GND - Ground* (Aterramento)
WDC	*Write Data Line Complementary*

WDT	*Write Data Line True*
WL	*Word Line*
WR	*Write* (Escrita)

SUMÁRIO

1 INTRODUÇÃO

A necessidade de se testar é algo inerente a maioria dos processos produtivos existentes no mundo. O teste elétrico tem como função principal verificar se um determinado dispositivo ou sistema apresenta correto funcionamento baseado nas especificações do mesmo (BUSHNELL; AGRAWAL, 2002). Testes elétricos em componentes de memórias DRAM (*Dynamic Random Access Memory*) são necessários para a detecção e segregação de uma taxa de defeitos (usualmente expressa em defeitos por milhão - dppm) inerente ao processo produtivo destes dispositivos (GOER; NEEF, 2004). A demanda do mercado atual de semicondutores mostra cada vez mais uma tendência dos conceitos definidos pela Lei de Moore, mesmo que em uma proporção menor do que no início do século XXI, porém ainda com um aumento significativo de células por componente com o passar dos anos. Esta evolução apresentou em 2018 um crescimento em bits de 20% em relação ao ano anterior, e isto, por sua vez, acaba trazendo uma densidade cada vez maior nos componentes de memória, com bilhões de células por componente e trazendo assim, uma complexidade maior no que se diz respeito aos testes elétricos, os quais necessitam ser cada vez mais abrangentes e que tendem ser cada vez mais demorados (Solid State Technology, 2018).

Outro fator a ser destacado refere-se ao fato de que o teste representa uma fatia representativa no custo produtivo das memórias DRAM, que corresponde, em média, aproximadamente 40% do custo total de produção (AL-ARS, 2005). A geração de memórias DDR4 atua com taxa de transferência por IO de até 3200 Mbps (HYNIX, 2017), o que traz, aliada ao aumento da densidade, a necessidade de testes que possuam uma ampla cobertura de falhas, principalmente baseado em uma boa base de teste (algoritmos de boa cobertura) e uma combinação de estresse (*SC - Stress Combination*) baseada na aplicação (GOER; NEEF, 2004).

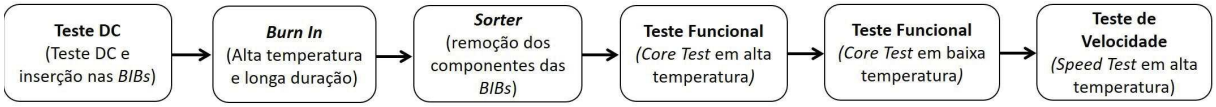
O crescimento exponencial na densidade de integração de componentes de memória, associada ao aumento da complexidade dos defeitos a serem detectados nestes componentes, faz com que as análises de falhas e os testes elétricos sejam cada vez mais significantes (AL-ARS, 2005). Além disso, a indústria de semicondutores vem em constante crescimento no Brasil, apresentando em 2018 um faturamento de aproximadamente 3,4 bilhões de reais (ABISEMI, 2019). Isto se deve principalmente aos incentivos fiscais fornecidos pelo Governo Federal através do PADIS (Programa de Apoio ao Desenvolvimento da Indústria de Semicondutores), que dentre alguns benefícios, apresenta isenções de até 100% nas alíquotas do imposto de renda (Governo Federal, 2007), para indústrias que encapsularem e testarem componentes eletrônicos no Brasil.

Contudo, mesmo com o avanço exponencial do número de transistores por componentes de memórias, o teste, por razões econômicas, não pode ter um crescimento no que diz respeito ao tempo de aplicação na mesma proporcionalidade. Pelo contrário, cada vez mais buscam-se

testes que possuam alta cobertura de falhas com menor tempo de execução possível. Porém, paralelamente a isto, o número de bits por chip está aumentando também, e mais do que isso, a sensibilidade a falhas está aumentando, uma vez que os mecanismos de falhas se tornam mais complexos. Por exemplo, testes para detectar falhas que são dependentes das células vizinhas são mais complexos e exigem uma execução de teste mais longos do que um teste funcional simples (JHA; GUPTA, 2003).

No que se refere as soluções para testes de memórias, as opções basicamente são: as mais robustas de uso industrial e as mais simplificadas para utilização em análises de laboratório. Os testadores de nível industrial são chamados de ATEs - *Automatic Test Equipment* que, por sua vez, possuem paralelismo elevado (testam inúmeros componentes simultaneamente) e opções amplas a nível de programação do teste. Testes de nível industrial (ou testes de produção) possuem geralmente um fluxo de teste com combinações de estresse, como exemplificado no fluxo de teste de memórias DRAM na Figura1, onde os ATEs são responsáveis pela realização dos testes funcionais e de velocidade.

Figura 1 – Exemplo de fluxo de teste de produção.



Fonte: Adaptado de HT Micron Semicondutores LTDA.

Dentre as principais soluções de teste de memórias existentes, pode-se exemplificar os testadores manuais de bancada, os ATEs e as aplicações customizadas. Na Figura 2 temos alguns exemplos de modelos disponíveis no mercado.

Figura 2 – Soluções de teste de memórias.

Fonte: Elaborado pelo autor

1.1 Justificativa

O teste das memórias SDRAM, as quais operam com altas frequências, alto número de bits, necessita de padrões de testes (algoritmos) cada vez mais complexos para que ocorra uma varredura de grande abrangência do componente e que garanta uma boa cobertura de falhas (JHA; GUPTA, 2003), o que torna o processo de teste ainda mais complexo, se comparado com outros dispositivos. Por exemplo, considerando a utilização de um teste tradicional, que realiza uma varredura completa de um componente com densidade n bits, onde $n = 2^N.B$, sendo que N representa o número de endereços e B a largura da palavra de dados (Tabela 1) (JHA; GUPTA, 2003).

Tabela 1 – Tempo de teste para diferentes densidades utilizando diferentes algoritmos para cobertura de todos os bits de uma memória RAM.

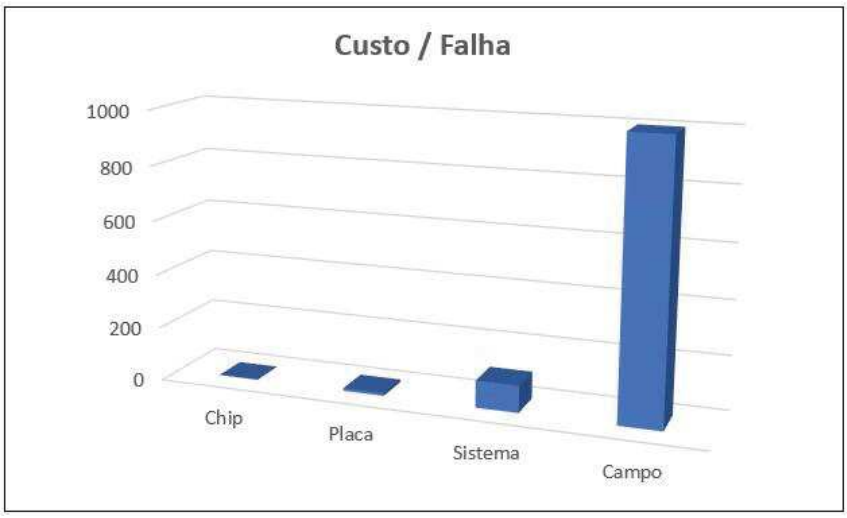
	Complexidade do Algoritmo			
n	n	$n \cdot \log_2 n$	$n^{3/2}$	n^2
1K	0.0001 s	0.001 s	0.0033 s	0.105 s
4K	0.0004 s	0.0049 s	0.026 s	1.7 s
16K	0.0016 s	0.023 s	0.21 s	27 s
64K	0.0066 s	0.1 s	1.7 s	410 s
256K	0.026 s	0.47 s	13 s	1.8 h
1M	0.105 s	2.1 s	107 s	30.5 h
4M	0.42 s	9.2 s	859 s	488 h
16M	1.68 s	40.3 s	1.9 h	0.89 anos
64M	6.71 s	174 s	15.3 h	14.3 anos
256M	26.8 s	752 s	122 h	228 anos
1G	107 s	3221 s	977 h	3655 anos

Fonte: Adaptado de Jha e Gupta (2003).

Com os dados apresentados na Tabela 1 torna-se possível observar o tempo de teste para diferentes memórias com densidades variadas e utilizando variados algoritmos de testes (com diferentes complexidades) para varredura total dos bits (JHA; GUPTA, 2003). Com o aumento da densidade e da complexidade do teste, é possível visualizar um aumento gradativo no tempo de teste, que em alguns casos pode exceder algumas dezenas de horas ou até mesmo anos. Outro fator que deve ser considerado, é a relação do custo por falha com o ponto de detecção. Uma falha tende a ser 10 vezes mais cara do que se detectada em um estágio anterior, ou seja, uma falha detectada no campo custa 10 vezes mais o valor de uma falha detectada a nível de sistema (KRUG, 2017), conforme exemplifica a Figura 3.

As soluções de teste no que diz respeito aos equipamentos utilizados (conforme exemplificado na Figura 2), pode em alguns casos ser um fator determinante para a qualidade e desempenho do teste. Se tratando de testadores de bancada e ATEs, pode-se destacar alguns pontos positivos e

Figura 3 – Custo da falha com relação ao ponto de detecção.



Fonte: Adaptado de Krug (2017).

negativos entre ambos, conforme ilustrado na Figura 4. Estes fatores podem, aliados ao tipo de aplicação do teste, influenciar diretamente na decisão de qual equipamento de teste escolher.

Figura 4 – Comparação entre ATE e Testador de Bancada.

ATE	Testador de Bancada
• Manuseio e classificação automatizada (handler); • Alto custo de aquisição; • Necessita aquisição do handler, além do ATE; • Alto paralelismo (número de dispositivos sob teste simultaneamente); • Possibilidade de programação customizada, de acordo com o produto a ser testado; • Complexidade para programação, devido linguagem de baixo nível específica do testador (*Pattern Test Language*); • Atende, geralmente, mais de uma geração/tipos de produtos; • Possibilita, na maioria das vezes, teste com alta velocidade (*high speed*).	• Manuseio e classificação manual; • Baixo custo de aquisição; • Não necessita da aquisição de um handler, podendo ser operado manualmente; • Baixo paralelismo (número de dispositivos sob teste simultaneamente); • Programação e algoritmos, na maioria das vezes, pré-estabelecidos e sem possibilidade de alteração; • Fácil configuração e seleção dos algoritmos; • Atende, geralmente, somente uma geração/tipo de produto; • Não possibilita, na maioria das vezes, teste com alta velocidade (*high speed*).

Fonte: Elaborado pelo autor.

Atualmente, o crescimento do mercado de semicondutores no Brasil e as boas perspectivas para o futuro proporcionados pelo PADIS - Programa de Apoio à Indústria de Semicondutores (Governo Federal, 2007) e a Lei da Informática, demandam cada vez mais de soluções de teste a nível industrial ou soluções similares que atendam esta crescente demanda. Isto, por sua vez, demanda conhecimento técnico e profissionais capacitados para desenvolvimento de testes em testadores para aplicações industriais, elaboração de programas de testes e principalmente

conhecimento de desenvolvimento de testes em ATEs (*Automatic Test Equipment*) de alta performance. Hoje no Brasil temos 19 empresas já beneficiadas pelo PADIS, além de aportes que foram feitos à ordem de R$ 1,4 bilhão pelo Banco Nacional de Desenvolvimento Econômico e Social (BNDES), pela Financiadora de Estudos e Projetos (FINEP) e pelo Programa CI-Brasil (ABISEMI, 2017). Tudo isso somado a cerca de 800 empresas habilitadas pela Lei da Informática, o que nos mostra um cenário de boa prospectiva tanto para a produção, como para consumo de componentes semicondutores no mercado interno brasileiro (ABISEMI, 2017).

1.2 Delimitação do tema ou problema

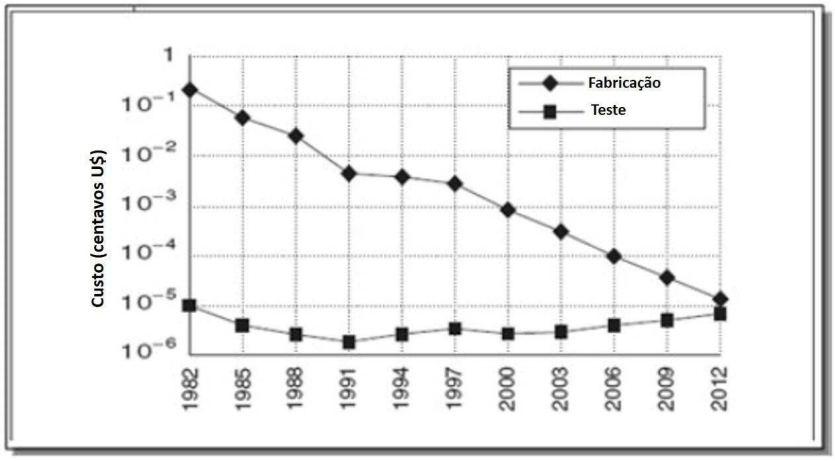
No contexto da situação atual e das perspectivas de crescimento da indústria nacional de semicondutores, foi possível identificar e/ou inferir os seguintes problemas, os quais direcionam o escopo e os objetivos deste trabalho:

- A presente escassez de profissionais capacitados para o desenvolvimento de soluções de teste de semicondutores em nível industrial no país;
- A dificuldade de transferir e aplicar na indústria nacional o conhecimento sobre teste de semicondutores já relativamente difundido no meio acadêmico;
- A complexidade para programação em linguagem de baixo nível específica para cada modelo de ATE;
- As barreiras impostas pela proteção da propriedade intelectual no que diz respeito à aquisição, geração e disseminação de conhecimento sobre tecnologias e teste de memórias SDRAM;
- A dificuldade para realização de análise de falhas estruturada em resultados de testes devido a restrição de conhecimento das rotinas e estratégias utilizadas nos programas de testes de nível industrial, e;
- O elevado custo do teste em relação ao custo de fabricação do componente, para o qual tem-se uma constante redução no custo de fabricação de um transistor que se opõe a um custo que se mantém inalterado para o teste de um transistor (Figura 5).

1.3 Delimitação do trabalho

Neste trabalho desenvolveu-se de um programa de teste funcional em temperatura ambiente para um componente encapsulado SDRAM DDR4 512Mx16b 96-FBGA em testador industrial ATE modelo Magnum V, do fabricante Teradyne. A validação funcional deste programa foi realizada através da análise de aderência de resultados com um testador de bancada modelo Turbocats TCE3200LP, do fabricante Triad. Realizou-se a implementação de algoritmos de testes

Figura 5 – Evolução do custo de fabricação de um transistor *vs* custo de teste de um transistor.



Fonte: Adaptado de Khoo (2014).

com aplicação típica em testes de memórias DRAM e posterior comparação de resultados para validação funcional dos mesmos. Após validação da solução de teste desenvolvida no ATE, realizou-se uma análise comparativa de desempenho e custos entre a solução desenvolvida e a solução existente do testador de bancada TC3200LP.

1.4 Objetivos

O objetivo geral desde projeto foi o desenvolvimento de uma solução para teste de DRAM utilizando um testador automático industrial.

1.4.1 Objetivos Específicos

Para alcançar o objetivo geral deste trabalho, os seguintes objetivos específicos foram necessários:

- Desenvolver, aprofundar e difundir o conhecimento no Brasil a respeito das tecnologias de memória SDRAM e sobre programação de ATEs de memória;
- Comparar a capacidade de detecção de falhas da solução de teste desenvolvida para ATE industrial com a solução de teste manual de bancada;
- Análise comparativa de desempenho e custo entre soluções de testes industriais e de bancada.

Através da definição do objetivo principal e objetivos específicos, tem-se o foco principal da execução deste trabalho e os resultados à serem apresentados.

2 REVISÃO BIBLIOGRÁFICA

Neste capítulo serão apresentados os principais tópicos de pesquisa abordados visando entendimento dos conceitos a serem aplicados na execução do trabalho e o aperfeiçoamento dos fundamentos teóricos necessários para o desenvolvimento do mesmo.

2.1 Memórias SDRAM

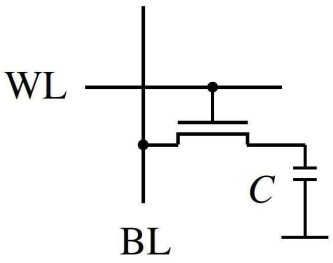
As memórias são dispositivos que tem, basicamente, a função de armazenar informações codificadas digitalmente. Estas informações podem representar números, letras, caracteres quaisquer, comandos de operações, endereços, etc. As informações são armazenadas nas memórias em células básicas denominadas “células de memória” e a cada uma delas está associado um “endereço de memória”, que é basicamente o código de acesso para estas informações, geralmente um conjunto de dígitos dos sistemas binário ou hexadecimal (RABAEY et al., 2002).

A classificação das memórias, entretanto, está diretamente relacionada com alguns fatores como: tamanho, tempo de acesso as informações armazenadas, padrão de acesso, aplicação e os requisitos do sistema (RABAEY et al., 2002) O que principalmente define o tipo de memória é a aplicação, a qual está relacionada com o tipo de armazenamento, podendo ser basicamente dividida em duas categorias: voláteis e não-voláteis. Com relação as memórias voláteis, a principal delas é a memória RAM. Dentre as memórias RAM existem duas variações: *Static* (estáticas) e *Dynamic* (dinâmicas) (BUSHNELL; AGRAWAL, 2002).

A diferença entre as RAMs estáticas e dinâmicas está, basicamente, na necessidade de *refresh* periódico dos dados nas células de memória. As estáticas utilizam geralmente um circuito *flip-flop* como célula básica de memória, o que acaba por armazenar o dado sem a necessidade de uma recuperação (*refresh*) a cada determinado período de tempo.

Com relação às memórias dinâmicas, pode-se destacar a DRAM (*Dynamic RAM*). As DRAMs utilizam um circuito simples, baseado principalmente na carga e descarga de capacitores (Figura 6). A sigla BL simboliza o *bit line*, ou seja, a informação a ser armazenada/lida e o WL é *word line*, que define em qual célula informação será armazenada. Conforme descreve WANG et al., os capacitores usados nas células de memória tendem a "perder"sua carga com o tempo. Desta maneira requerem uma renovação constante e periódica dos dados, a fim de não se perder a informação armazenada na memória. Este ciclo de renovação chama-se *refresh cycle*. Um controlador determina o tempo entre os ciclos de *refresh*, e um contador assegura que toda a matriz (todas as linhas) será submetida ao *refresh*. Logicamente isso significa que alguns ciclos do sistema são usados para a operação de *refresh*, e isto acarreta em um impacto negativo no desempenho.

Figura 6 – Célula de memória DRAM.

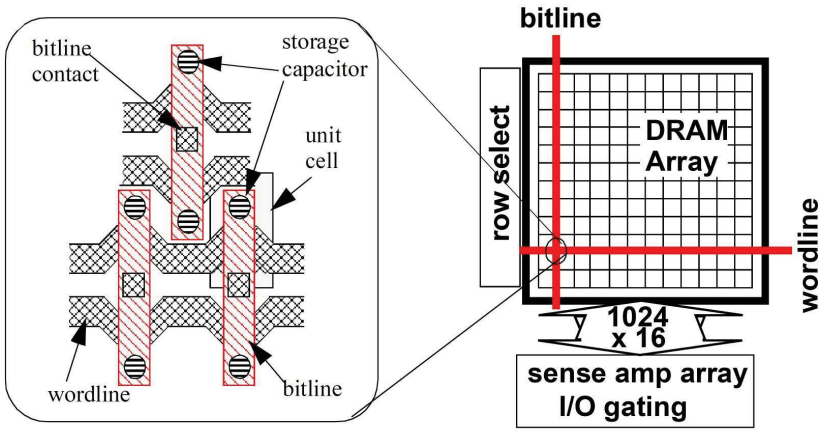


Fonte: Jha e Gupta (2003).

As memórias DRAM, que foram o objeto alvo deste trabalho, inicialmente em sua criação eram assíncronas e *single-bank* (banco único), uma vez que os processadores eram relativamente lentos. Mais recentemente, interfaces síncronas (SDRAM) foram desenvolvidas, apresentando uma série de vantagens.

Uma memória DRAM pode ser visualizada como um arranjo de células, como uma tabela ou planilha (Figura 7). Estas células são compostas de capacitores e contém um ou mais bits de dados, a depender da configuração do chip. Esta tabela é endereçada através de decodificadores de linha e coluna, que por sua vez recebem os sinais para sincronização do endereçamento, denominados sinais *Column Address Strobe* ($\overline{CAS}$) e *Row Address Strobe* ($\overline{RAS}$) (WANG et al., 2000).

Figura 7 – Vista *top-down* de um arranjo de células de uma memória DRAM.



Fonte: Wang et al. (2000).

Transistores de acesso denominados *sense amps* (amplificadores de sinal) são conectados a cada coluna de modo a possibilitar as operações de leitura e *refresh* do componente. Uma vez que as células são capacitores que se descarregam para cada operação de leitura, o *sense amp* precisa recuperar ou restaurar o dado ali armazenado antes do fim do ciclo de acesso (JHA; GUPTA, 2003).

Segundo RABAEY et al., a organização de uma memória está associada ao número de linhas x colunas, e ao número de bancos internos que possui, o que dará sua capacidade total de armazenamento (densidade) e a largura do barramento de dados (*data bus lenght*) (x4, x8, etc). O *data bus lenght* (Figura 8), como o próprio nome diz, representa quantos bits poderão ser lidos/escritos simultaneamente em um banco da memória. Portanto, a definição de uma memória pode se dar pela sua densidade x *data bus lenght*, como por exemplo, uma memória de 512 M x 8b, ou seja, 512 M de palavras de 8 bits, o que nos dá uma densidade de 512 x 8 = 4096 Mb (4 Gb).

2.1.1 Memórias DDR

No contexto das memórias SDRAM ocorreu uma evolução importante, a qual foi o desenvolvimento das memórias DDR. Apesar das otimizações ocorridas ao longo da história, os módulos de memória SDRAM continuavam realizando apenas uma transferência por ciclo, da forma mais simples possível. Depois de decorrido o longo ciclo inicial, as células de memória entregam uma leitura de dados por ciclo, que passa pelos *buffers* de saída e então é despachada através do barramento de dados. Todos os componentes trabalham na mesma frequência (WANG et al., 2000). As memórias DDR, entretanto, detêm de uma característica que as tornam capazes de realizar duas transferências por ciclo e serem quase duas vezes mais rápidas que as memórias SDRAM básicas, mesmo mantendo a mesma frequência de operação e a mesma tecnologia básica. Esta seria a justificativa do termo Double Data Rate (DDR), ou taxa de dados dupla. Com o lançamento das memórias DDR, as SDRAM passaram a ser chamadas de Single *Data Rate* ou SDR (David Tawei Wang, 2005).

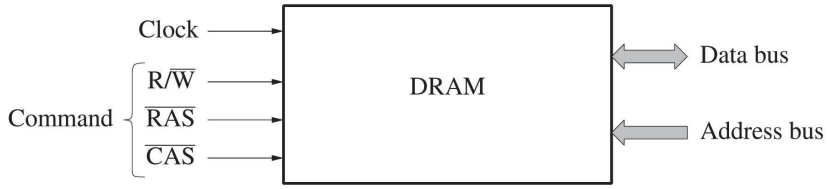
Com o avanço natural da tecnologia, teve-se o surgimento das memórias DDR2, DDR3 e DDR4. Como o próprio nome indica, é uma evolução da tão utilizada memória DDR. Entre as suas principais características estão o menor consumo de energia elétrica e menor custo de produção por bit, além de taxas de transferência de dados muito maiores. Basicamente o que ocorre é que a cada ciclo de evolução da DDR existe um avanço em dobro na taxa de dados (velocidade). O que antes na DDR ocorria em 2 transferências por ciclo, na DDR2 passa a ocorrer duas vezes mais rápido que na DDR, e assim sucessivamente (SEONG et al., 2010).

2.1.2 Arquitetura e Funcionamento

Em um visão bem simplificada de uma memória SDRAM, pode-se considerar um diagrama de blocos básico que apresenta suas principais entradas e saídas para o funcionamento (Figura 8). O *clock* é um dos sinais de entrada, que tem sinal síncrono com o restante do sistema, por se tratar de memórias SDRAM. Os pinos de entradas de endereçamento estão ilustrados como *Adress bus*, representado na Figura 8 por um único sinal. Outro sinal descrito é o $R/\overline{W}$, que indica a função a ser executada: *read* (leitura) ou *write* (escrita). Além disso, tem-se os pinos exemplificados por *Data bus*, ou seja, os I/Os (*inputs* e *outputs*) da memória. Isto mostra que

o acesso a dados não necessariamente ocorre bit a bit, mas sim podendo ocorrer o acesso a múltiplas células da memória em paralelo, aumentando, assim, a taxa de transferência. Este *bus* ou bloco de dados em paralelo é chamado de *word* (palavra) (AL-ARS, 2005).

Figura 8 – Diagrama de blocos da memória DRAM.



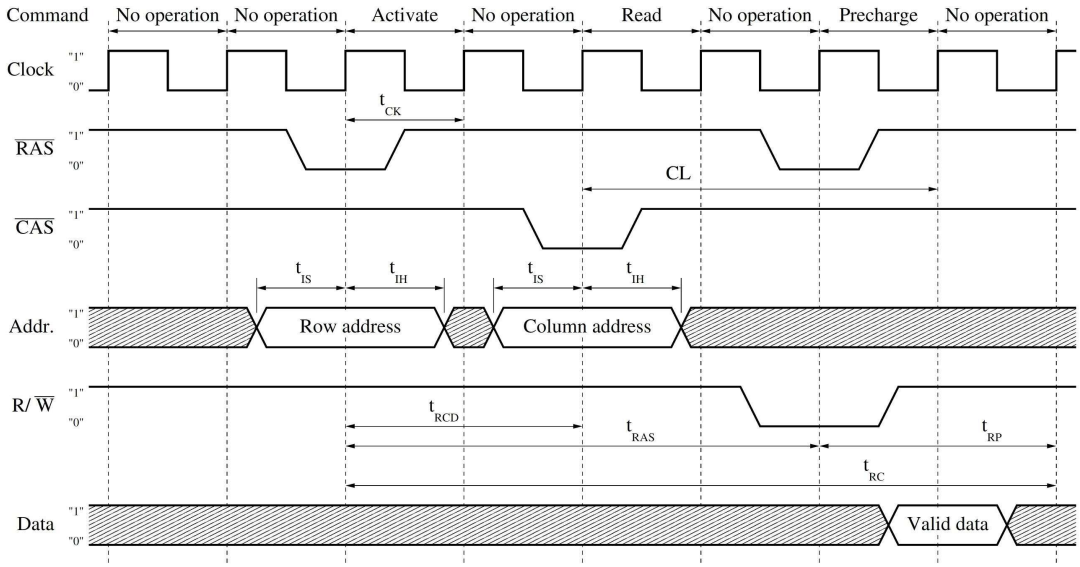
Fonte: Al-Ars (2005).

O sinal *Data bus* é exemplificado com uma flecha em ambos os sentidos, exatamente pela concepção de que os sinais podem tanto entrar, como sair. Isto porque utiliza-se uma estratégia de multiplexação, o qual pode-se utilizar os mesmos pinos tanto para sinais de entrada, como para sinais de saída. O mesmo ocorre também para o *Adress bus*, onde ocorre uma multiplexação por tempo, ou seja, os dados são enviados em duas partes. Os dados de linha (*row*) e coluna (*column*) são enviados separadamente, um após o outro nos mesmos pinos de endereço. Para controlar este envio, são utilizados dois sinais: $\overline{RAS}$ (linha) e $\overline{CAS}$ (coluna). Juntamente com o sinal de $R/\overline{W}$, pode-se considerar estes três sinais de *Command bus* no diagrama de blocos da memória DRAM (AL-ARS, 2005).

2.1.2.1 Operações e Diagrama de Tempo

Os diagramas de tempo ou *timing diagrams* são as representações dos sinais de entrada e saída da memória enquanto realizam alguma operação. Este diagrama representa a interdependência de tempo entre diferentes sinais da memória e define as condições de tempo requeridas para cada tipo de operação (AL-ARS, 2005). As operações, tanto de leitura, como de escrita em uma memória DRAM, dependem de uma sequência de comandos e respectivos tempos entre os mesmos. Na Figura 9, tem-se o exemplo de uma operação de leitura para uma memória de 512Mb/400MHz DDR DRAM usando um t_{CK} (período do ciclo de *clock*) de 10ns para leitura de *r0* e *r1*, para o qual 0 e 1 são os valores esperados na saída da operação de leitura, respectivamente. Os comandos descritos na parte superior da Figura 9 (*No operation - NOP*, *Active - ACT*, etc) são comandos internos da memória para cada operação de *clock*. Os dados armazenados na célula endereçada para leitura irão aparecer no *data bus* após um período que é chamado de CL ($\overline{CASLatency}$). Alguns outros tempos também são utilizados ($t\overline{RP}, t\overline{RC}, t\overline{RCD}$, etc) que tem sua funcionalidade descrita na Tabela 2.

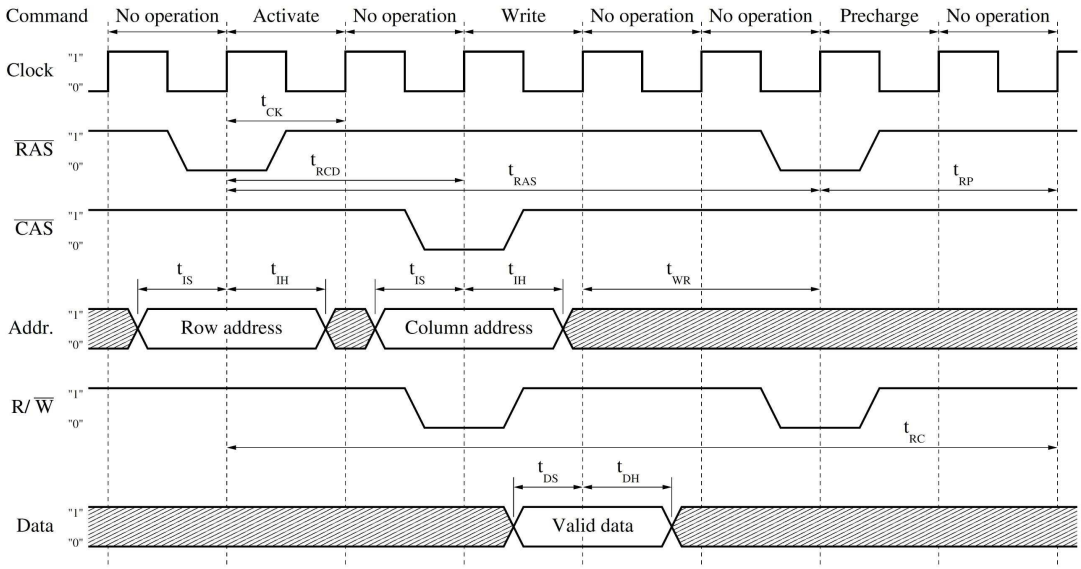
Figura 9 – Diagrama de tempo para operação de leitura em uma memória DRAM 512Mb, 400MHz, utilizando $t_{\mathrm{CK}} = 10ns$.



Fonte: (AL-ARS, 2005).

Já na Figura 10, tem-se o exemplo de uma operação de escrita na mesma memória. As entradas de escrita são *w0* e *w0*, que correspondem aos valores 0 e 1, respectivamente.

Figura 10 – Diagrama de tempo para operação de escrita em uma memória DRAM 512Mb, 400MHz, utilizando $t_{\mathrm{CK}} = 10ns$.



Fonte: Al-Ars (2005).

Os principais parâmetros de tempos para as operações de escrita também estão descritos na Tabela 2.

Tabela 2 – Parâmetros de tempos para uma memória DDR4 512x16 1600MHz.

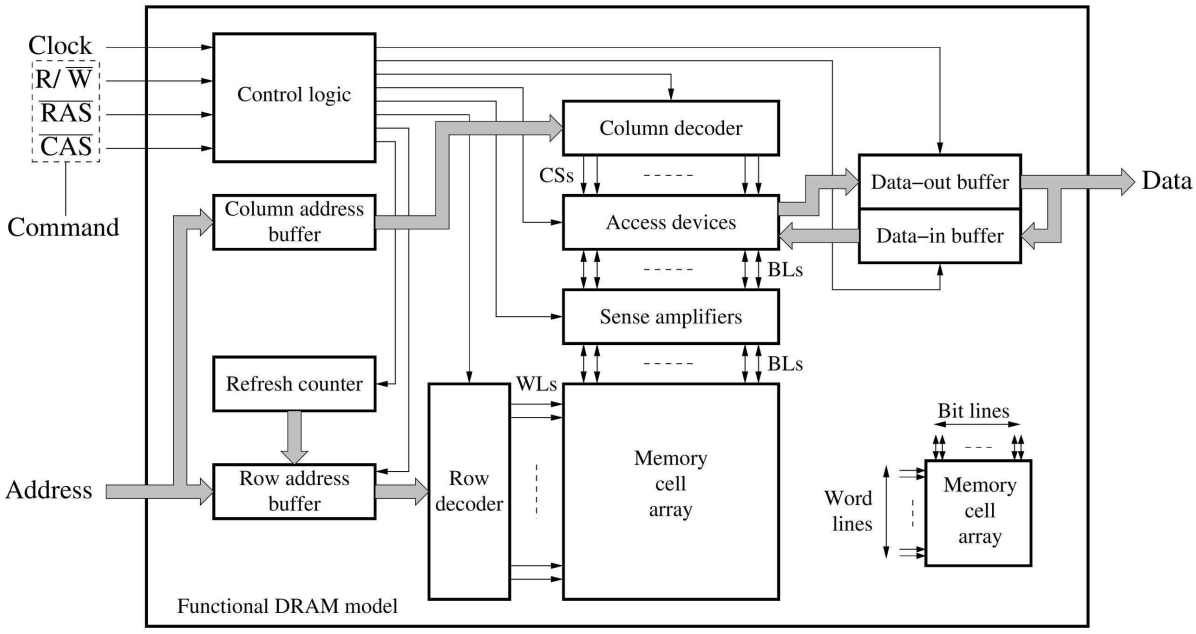
Parâmetro	**Descrição**	**Mínimo**	**Máximo**	**Unidade**
CL	*CAS Latency*	n = 11	-	n*CK
t_{CK}	*Clock cycle time*	1.25	-	ns
t_{DH}	*Data hold time*	0.18	-	tCK
t_{DS}	*Data setup time*	0.18	-	tCK
t_{IH}	*Input hold time*	140	-	ps
t_{IS}	*Input setup time*	115	-	ps
t_{RAS}	*Row address strobe time*	35	-	ns
t_{RC}	*Row cycle time*	48.75	-	ns
t_{RCD}	*Row-column delay time*	13.75	-	ns
t_{RP}	*Row precharge time*	13.75	-	ns
t_{WR}	*Write recovery time*	15	-	ns

Fonte: Hynix (2017).

2.1.2.2 Funcionamento da memória DRAM

Pode-se pensar na memória DRAM como um arranjo de blocos funcionais todos interconectados, cada um executando uma função específica dentro do sistema. Na Figura 11 pode-se ver os distintos blocos funcionais internamente na memória DRAM, que juntos operam para que a mesma funcione adequadamente (AL-ARS, 2005).

Figura 11 – Diagrama de blocos funcional simplificado de uma memória DRAM.

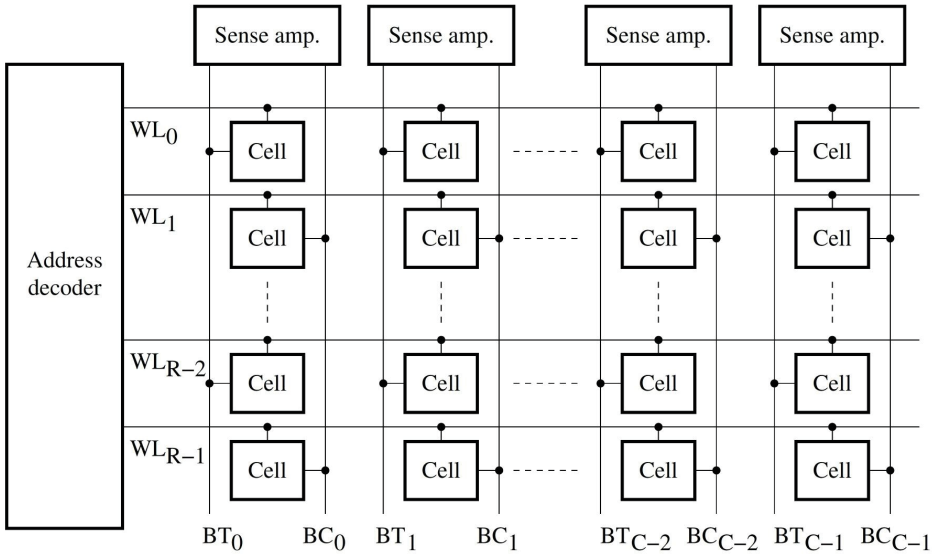


Fonte: Al-Ars (2005).

- ***Memory Cell Array*:** é o bloco mais importante da memória DRAM e ocupa em torno de 60% da área do componente. É nele que estão contidas as células de memória (Figura 6) devidamente organizadas em linhas e colunas. Este arranjo interno, que depende da

quantidade de linhas x colunas, é que vai definir a densidade da memória. Na Figura 12, tem-se o exemplo de um *memory cell array* exatamente como é internamente em uma memória DRAM (RABAEY et al., 2002). Os WL (*Word Line*) carregam o sinal de controle de acesso a célula de memória (sinais de endereçamento). Já os BL (*Bit Line*) contém a informação a ser armazenada. Neste caso, tem-se a informação que é fornecida por BT (*True Bit Line*) e BC (*Complementary Bit Line*), uma vez que utilizam os *sense amplifiers* de forma compartilhada (AL-ARS, 2005).

Figura 12 – Conexão de múltiplas células de memória em um *memory cell array*.



Fonte: Al-Ars (2005).

- ***Control Logic***: o controle lógico ou também chamado *timing generator* tem por função ativar e desativar os blocos funcionais desejados em determinados momentos. É nele que os sinais do *Command bus* são recebidos e gerados os sinais internos da memória para os demais blocos (AL-ARS, 2005).

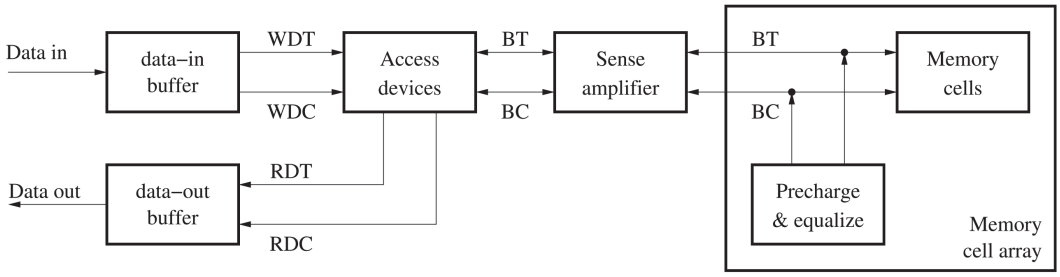
- ***Adress Decoders***: para poder acessar uma célula do *Memory cell array*, os endereços de linha e coluna precisam ser decodificados. A entrada do *Adress decorders* são os endereços das células e suas saídas serão os WL (para linhas) e BT e BC (para colunas) (AL-ARS, 2005).

- ***Outros blocos***: os *data-in buffers*, *data-out buffers* e os *adress buffers* são os *buffers* para armazenamento de dados e endereços temporários, tanto na entrada como na saída da memória. O *sense amplifier* é o que realiza a identificação do nível lógico na célula de memória. O último bloco funcional é chamado de *refresh counter* e como o nome diz, é responsável por controlar o *refresh*, ou seja, manter atualizados os níveis lógicos das células de memória (AL-ARS, 2005).

2.1.2.3 Arquitetura da memória DRAM

Circuitos elétricos de memórias são baseados nos caminhos dos sinais (*signal path*), o qual na memória DRAM estão divididos basicamente em três: *data path*, *adress path* e *control path*. Na Figura 13 tem-se exemplificado o *data path* de uma memória DRAM. Se uma operação de leitura ocorre, o dispositivo irá conectar BL (BT e BC) com RDT (*Read Data Line True*) e RDC (*Read Data Line Complementary*), que são os *drivers* do *data-out buffer*. Já caso ocorra uma operação de escrita, o dispositivo conecta BL (BT e BC) com WDT (*Write Data Line True*) e WDC (*Write Data Line Complementary*), que são os drivers do *data-in buffer* (AL-ARS, 2005).

Figura 13 – Diagrama de blocos de um *data path* de uma memória DRAM.



Fonte: Al-Ars (2005).

Como pode-se observar na Figura 13, cada bloco de circuito elétrico pode ser representando por uma estrutura eletrônica, ou seja, o *memory cells* possuem um determinado circuito elétrico, *sense amplifiers* outro, e assim consecutivamente (GOER; NEEF, 2004). Entrando no nível de *layout* do circuito integrado, por estar relacionado diretamente com projeto e características construtivas do CI, dificilmente será encontrada com detalhamento na literatura, principalmente devido a propriedade intelectual e confidencialidade industrial (AL-ARS, 2005).

O que acontecore na prática é que para otimizar o arranjo interno das células de memórias, o fabricante acaba adotando o que chama de endereço físico da célula de memória (*phisycal cell adress*) e o endereço lógico da célula de memória (*logical cell adress*). Isto significa que ao acessar logicamente uma região da memória, principalmente quando realiza-se o teste funcional, pode-se estar acessando uma região fisicamente diferente. Este processo é conhecido como *cell adress scrambling* e acaba por ser muitas vezes um dificultador para elaboração de testes com boa cobertura e que visam a detecção de falhas de vizinhança, por exemplo (GOER; NEEF, 2004).

2.2 Tipos de Falhas e Caracterização

Esta seção destina-se a descrever os principais tipos de falhas existentes nos dispositivos de memórias DRAM, bem como suas principais características. Como o objeto de estudo deste trabalho foram as falhas tipicamente funcionais, este tema por sua vez será abordado com mais

detalhamento dentro desta seção. Primeiramente é importante conceituar o que é uma falha, um erro e um defeito. Defeito refere-se a uma imperfeição física do componente. A falha, por sua vez, é a abstração física deste defeito na forma do funcionamento inadequado do sistema. O erro trata-se do comportamento inadequado do sistema, ou seja, apresentação de resposta diferente da resposta esperada (KRUG, 2017).

2.2.1 Tipos de Falhas

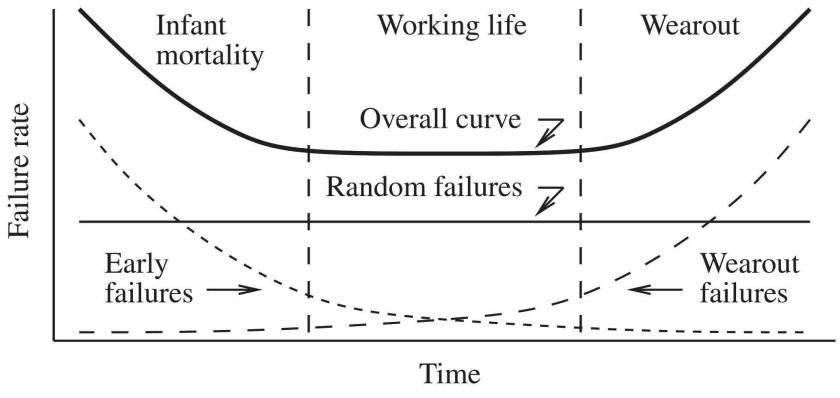
Uma falha pode se manifestar de duas formas, no que diz respeito a sua duração: falhas permanentes e falhas não-permanentes.

As **falhas permanentes** são falhas que afetam o comportamento funcional do componente permanentemente. Elas estão sempre relacionadas a algum mecanismo de falha físico, como por exemplo: falta de conexão entre substrato e *die*, trilhas defeituosas, curto entre conexões, etc (GOER; NEEF, 2004). As falhas permanentes podem ser modeladas, o que as difere das falhas não-permanentes.

Já as **falhas não-permanentes** podem ser classificadas em dois tipos: *intermitentes* e *transientes*. As falhas *transientes* são falhas que se apresentam decorrentes a variações ambientais, tais como: radiação, vibração, umidade, temperatura, etc. Após o retorno da condição ambiental para a condição nominal de operação a falha tende a desaparecer. Por outro lado, as falhas *intermitentes* não são relacionadas com as condições ambientais, podendo estar relacionadas com desgaste de componentes, perdas de conexões intermitentes, etc. As falhas *intermitentes* tendem com o passar do tempo a se tornarem falhas *permanentes* (JHA; GUPTA, 2003).

Outro comportamento clássico dos componentes eletrônicos são as falhas relacionadas à mortalidade infantil. Para descrever este fenômeno tem-se a Figura 14 a conhecida **curva da banheira**, ou do inglês *bathtub curve*, que demonstra o comportamento dos circuitos integrados no decorrer de sua vida útil (ASHOK, 2002).

Figura 14 – Curva da banheira.



Fonte: Jha e Gupta (2003).

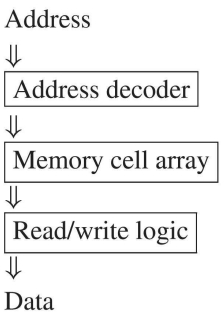
Como pode-se observar na Figura 14, no período inicial de existência do componente teve-se as falhas conhecidas por *mortalidade infantil*. Já após este período ocorre uma estabilização, onde tem-se somente as falhas aleatórias (*random failures*). Passado mais um período de tempo, ocorrem as falhas por desgaste ou fim da vida útil (*wearout failures*) (JHA; GUPTA, 2003).

Já no que diz respeito a natureza das falhas, pode-se classifica-las em três categorias: lógicas, paramétricas e de *delay*. As **falhas lógicas** alteram a função lógica de circuito ocasionando uma função lógica não desejada. As **falhas paramétricas** são as que alteram a magnitude dos parâmetros do circuito, causando a alteração de algum fator como resistência, capacitância, corrente elétrica, entre outros. Já as **falhas de *delay*** são as que apresentam alterações nos tempos de resposta do circuito, afetando o comportamento do dispositivo (WU; HUANG; WU, 1999).

2.2.2 Modelos de Falhas

Como o foco deste trabalho foi o desenvolvimento de um programa de testes funcional, neste capítulo serão abordados unicamente os modelos de falhas funcionais existentes. Para entendimento das falhas funcionais existentes, utiliza-se o que chama-se de modelo funcional reduzido, que é composto por três blocos: decodificador de endereços (*adress decoder*), arranjo de células (*cell array*) e escrita/leitura lógica (*read/write logic*), conforme a Figura 15 .

Figura 15 – Modelo funcional reduzido da memória DRAM.



Fonte: Jha e Gupta (2003).

Quando não existe o interesse do diagnóstico de falhas (localização do defeito), é possível modelar a memória usando o modelo funcional reduzido (Figura 15). Com este modelo, por exemplo, é possível mapear falhas funcionais descritas na Tabela 3, para um modelo reduzido conforme o contido na Tabela 4 (BUSHNELL; AGRAWAL, 2002).

Com o uso do modelo funcional reduzido, criou-se também o modelo reduzido de falhas funcionais (Tabela 4), com a apresentação do efeito da falha e não relacionado a especificamente o seu defeito (causa) (BUSHNELL; AGRAWAL, 2002).

A ideia da utilização de modelos de falhas se dá principalmente visando facilitar a execução do teste a nível industrial, uma vez que os defeitos reais possíveis em um circuito integrado são inúmeros e com a utilização do modelo de falhas é possível identificar melhor as estratégias

Tabela 3 – Falhas funcionais em memórias.

Falhas Funcionais
Cell stuck
Driver stuck
Read/Write line stuck
Chip-select line stuck
Data line stuck
Open circuit in data line
Short circuit between data lines
Crosstalk between data lines
Address line stuck
Open circuit in address line
Shorts between address line
Open circuit in decoder
Wrong address access
Multiple simultaneous address access
Cell can be set to 0 but not to 1 (or vice versa)
Pattern sensitive cell interation

Fonte: Bushnell e Agrawal (2002).

Tabela 4 – Modelo reduzido de falhas funcionais.

Notação	**Falha**
SAF	*Stuck-at Fault*
TF	*Transition Fault*
CF	*Coupling Fault*
NPSF	*Neighborhood Pattern Sensitive Fault*

Fonte: Bushnell e Agrawal (2002).

de teste (LIN; WU, 2007). Dentre os modelos de falhas, pode-se dividi-los em duas classes, o quais dizem respeito a sua detecção. Os modelos que apresentam as **falhas estáticas** são as quais podem ser sensibilizadas realizando-se no máximo uma operação. Já as chamadas **falhas dinâmicas** apresentam sensibilização somente com mais de uma operação.

2.3 Teste de Circuitos Integrados

Os testes de circuitos integrados podem ser classificados de acordo com a tecnologia para a qual foram designados, parâmetros que medem, propósito para qual os resultados de teste são utilizados e o método de aplicação (JHA; GUPTA, 2003).

Com relação a tecnologia do circuito a ser testado, pode-se classificar o teste em três tipos: analógicos, digitais ou *mixed-signal*. Já no que se refere aos parâmetros a serem mensurados, classifica-se em: testes lógicos ou testes elétricos (paramétricos e dinâmicos). Os testes conduzidos para reproduzir a aplicação do circuito a fim de identificar falhas são conhecidos por: testes funcionais e testes *in-circuit* (JHA; GUPTA, 2003).

2.3.1 Teste de Memórias DRAM

Neste capítulo será realizada uma abordagem aprofundada nas características principais de testes de memórias, que possuem e certa forma algumas particularidades com relação aos demais circuitos integrados. Basicamente o teste de memórias está dividido em três níveis: ***wafer level test***, ***component level test*** e ***board level test*** (AL-ARS, 2005). Dentro destes três níveis do processo produtivo de uma memória e aplicação até a chegada ao usuário final, ampliaremos o foco no que se refere ao teste de componente (*component level*). O teste de componente ocorre após o circuito de memória já encapsulado (*packaging*) e é composto tipicamente por dois tipos de testes: ***burn-in test*** e ***packaging test*** (AL-ARS, 2005). Um exemplo de fluxo de teste de componentes de memória está ilustrado na Figura 1, no capítulo 1, onde ocorrem testes de níveis paramétricos (Teste DC) previamente ao teste de *Burn-In*, e os testes funcionais e de velocidade (*Packaging Test*).

2.3.1.1 Teste Paramétrico

Durante este processo de teste é realizado previamente um teste DC estático, ou seja, um teste paramétrico, onde são verificados curto circuito, circuito aberto, fuga de corrente e sobrecorrente. O objetivo principal deste teste inicial é filtrar as falhas mais grosseiras, não havendo desperdício de tempo de teste em peças com falhas explícitas (DING, 2015). Neste teste, o testador DC compara os parâmetros, conforme os valores de referência (máximos e mínimos) definidos no programa de teste. Alguns testes paramétricos típicos são:

- **Teste de Continuidade**: o teste de continuidade, também conhecido como teste de contato ou teste curto-circuito e circuito aberto (*open/short*) verifica se as conexões apresentam característica de circuito aberto ou curto-circuito. Durante o teste, o dispositivo é submetido a uma corrente fixa conhecida, e então é medida a tensão entre pinos. Este nível de tensão é comparado com o nível de tensão esperado (especificação) e caso esteja nos níveis adequados o componente é aprovado. O teste de continuidade é dividido em nível de VCC e nível de VSS, uma vez difere do ponto de referência da aplicação da tensão, ou seja, ora com relação a VCC e ora com relação a VSS. O resultado é baseado na resposta em tensão existente no diodo de entrada dos pinos, que deve apresentar usualmente limite de 0,7V (DING, 2015).

- **Teste de Fuga de Corrente**: o teste de fuga de corrente ou teste de *leakage* é dividido em dois tipo: *leakage low* e *leakage high*. Neste teste mede-se uma corrente elétrica entre o pino alvo e VCC (*leakage high*) ou VSS (*leakage low*). Esta corrente deve se na ordem de nanoamperes, caso contrário, existe fuga de corrente elétrica (DING, 2015).

Após realizado o teste paramétrico, tipicamente executa-se no mesmo equipamento a inserção do componente aprovado na placa BIB (*Burn-In Board*) e inicia-se o teste de *Burn-in*.

2.3.1.2 Teste de Burn-In

O teste de Burn-in tem como objetivo principal eliminar falhas precoces. Através de uma combinação de estresse (alta temperatura de operação e altos níveis de tensão) ocorre a aceleração do tempo de vida do componente. Este processo de teste também costuma ser denominado de teste por envelhecimento precoce ou teste de mortalidade infantil, e é baseado no comportamento de vida dos circuitos integrados, conforme descreve a curva da banheira (Figura 14) (ASHOK, 2002).

Neste teste algumas rotinas de escrita são executadas nos componentes, visando principalmente o desgaste elétrico e estresse, a fim de originar falhas. Ao término do processo executa-se um teste funcional (algoritmos com escritas e leituras) a fim de eliminar previamente ao teste funcional alguns componentes que já apresentaram defeito (JHA; GUPTA, 2003).

2.3.1.3 Teste de Packaging

O objetivo principal do *packaging test* ou teste de encapsulamento é de validar a operação da memória de acordo com as suas especificações. Neste teste, uma série de testes são realizados, combinados as operações em alta e baixa temperatura (*stress combination*) a fim de certificar sua perfeita funcionalidade. Os processos de teste descrevem três estruturas principais para realização de um bom teste, além de três requisitos principais (AL-ARS, 2005).

- **Estruturas de Teste**:

  1. Sequência de operação: o teste precisa especificar uma sequência de operação na memória de escritas e leituras em uma ordem específica nas células de memória. Em memórias DRAM, uma grande porção de falhas são as chamadas falhas dinâmicas, que são falhas sensibilizadas que resultam de duas ou mais operações na memória.

  2. Padrões de dados (*data pattern*): o teste precisa também especificar um padrão de 0s e 1s a serem escritos e lidos nas células de memórias acessadas.

  3. Estresse: o teste de memória precisa incluir uma especificação de diferentes condições de operação ou estresse (como *timings*, temperatura e tensão). Este é um importante componente para que sejam testados as funcionalidades dentro do *range* de parâmetros definidos na especificação. Como as memórias DRAM possuem o armazenamento de informações através de capacitores, este é um componente que sensibiliza as correntes de fuga existentes pela injeção de altos níveis de tensão com temperaturas extremas.

- **Requisitos de Teste:**

  1. Detecção de falhas: o teste deve apresentar o resultado de falha quando realizando o teste de uma memória que contém falha. Este é o requisito básico para qualquer teste designado para detecção de específicos comportamentos de falhas.

2. Localização da falha: o teste deve estar apto a identificar a célula de memória (ou grupo de células) específico do local da falha. Este requisito também está associado a identificação e reparo de células (*fuse cells*).

3. Diagnóstico de falhas: o teste deve estar apto a auxiliar a identificar a causa raiz física do defeito através do comportamento da falha. Este requisito é associado a necessidade do *feedback* imediato ao processo produtivo após identificação de falhas potencialmente catastróficas.

O desenvolvimento de um programa de teste é realizado basicamente em duas etapas: primeiramente utilizando informações de especificação de projeto (teste baseado na especificação) e então numa segunda etapa, através da utilização do componente encapsulado realizam-se testes e novos mecanismos de falhas são descobertos e analisa-se o o desempenho do teste (*yield*), onde ocorre a necessidade do aprimoramento do programa de teste (*yield learning*). Outro importante ponto que ocorre nesta segunda etapa, que chamamos de teste de manufatura, é o *feedback* que ocorre dos retornos de cliente e servem para atualização do programas para cobertura de novas falhas, até então desconhecidas, ou redução do tempo de teste pela exclusão de rotinas no caso de falhas que não ocorrem (*Test Time Reduction*) (AL-ARS, 2005).

Com o que diz respeito a primeira etapa do desenvolvimento de um programa de testes, alguns itens podem ser denominados para que tenha-se um roteiro para o programa a ser desenvolvido. Segue abaixo uma sequência estabelecida para criação de um programa de testes em um testador do tipo ATE (JONE, 2013):

1. **Configuração das condições de teste**: definição dos tempos (*timings*), níveis e padrões de tensão (formas de ondas), configuração e status dos pinos;
2. **Sequência de teste**: definição da ordem de execução dos blocos de testes e configuração das classificações de resultados (*bins*);
3. **Algoritmos**: criação dos algoritmos (*patterns*) na linguagem do ATE e interface com a sequência de teste para que ocorra sua devida execução. Definição dos padrões de dados a serem executados por cada um dos algoritmos.

2.3.2 Algoritmos de Teste de Memórias DRAM

Conforme já comentado, a complexidade dos testes e dos componentes a serem testados se apresentam ainda mais agravados quando trata-se de memória SDRAM, que operam com altas frequências, alto número de bits, necessitam de padrões de testes (algoritmos) cada vez mais complexos para que ocorra uma varredura de grande abrangência do componente e que garanta uma boa cobertura de falhas (JHA; GUPTA, 2003).

Estes algoritmos vem sendo aperfeiçoados constantemente a fim de acompanhar da evolução dos dispositivos a serem testados. Muitos deles são algoritmos clássicos e estão disponibilizados

na literatura. Como o objetivo deste trabalho não será especificamente a criação de um algoritmo, serão apresentados aqui os algoritmos mais comuns e com boa aplicabilidade para teste em memórias DRAMs (BUSHNELL; AGRAWAL, 2002).

Uma boa estratégia de teste consiste na organização do que chamamos de BTs (*base tests*), associada a uma SC (*stress combination*) (GOER; NEEF, 2004), com foco somente na criação dos BTs. Uma sequência (set) de um BT consiste basicamente em cinco etapas: (GOER; NEEF, 2004)

1. Testes elétricos: esta classe consiste na realização de testes de contato, testes paramétricos DC e AC;

2. Testes *March*: *march test* são os algoritmos mais populares utilizados para detecção de falhas funcionais e falhas de endereçamento. Existem inúmeras variações dentro dos *march test*, de acordo com as particularidades existentes em cada padrão, como por exemplo *March A*, *March C-R*, *March U*, entre outros.

3. Testes *Base Cell*: esta classe de teste visa detecção de falhas de vizinhança, onde um existe a influência de uma *a-cell* e uma *v-cell* (ver Capítulo 2.2.1. Para esta detecção também utilizam-se algoritmos. Dentre alguns mais comuns podemos citar: *Butterfly*, *Galcol*, *Galrow*, *Checkerboard*, entre outros.

4. Testes repetitivos: testes repetitivos realizam múltiplas operações de leitura e escrita em uma única célula Neste caso o objetivo é detectar falhas unicamente de uma célula (*single-cell fails*. Exemplos de algoritmos de testes para testes repetitivos são: *HamRd*, *Hammer* e *HamWr*.

5. Testes *Pseudo-random*: classe de teste também muito utilizada em BIST (*Built-in Self Test*). Neste caso o resultado da resposta a determinados estímulos realizados na memórias são comparados a dados de referência (JHA; GUPTA, 2003). Alguns exemplos de algoritmos de *pseudo-random tests* são: *RARWRD* e *DADWRD*.

Para entendimento dos algoritmos de teste, alguns conceitos referente à notação e simbologia utilizadas precisam ser apresentadas para uma melhor contextualização. Na Tabela 5 estão descritas as principais notações e seus respectivos significados.

Tabela 5 – Notação utilizada em algoritmos.

Notação	Descrição
r	Operação de leitura
w	Operação de escrita
r0	Operação de leitura, esperando ler nível lógico 0
r1	Operação de leitura, esperando ler nível lógico 1
w0	Operação de escrita, esperando escrever nível lógico 0
w1	Operação de escrita, esperando escrever nível lógico 1
$\uparrow$	Atribuição de valor lógico 1 na célula, alterando de 0 para 1
$\downarrow$	Atribuição de valor lógico 0 na célula, alterando de 1 para 0
$\updownarrow$	Complementa o valor da célula
$\Uparrow$	Incrementa endereço
$\Downarrow$	Decrementa endereço
$\Updownarrow$	Acessa endereços em ordem crescente ou decrescente
$\longrightarrow$	Escrita de 0 em célula com nível 1
$\rightarrow$	Escrita de 1 em célula com nível 0
$\Rightarrow$	Escrita de valor x em célula com valor x
Del	Realiza um atraso de tempo (*delay*)

Fonte: Adaptado de Bushnell e Agrawal (2002).

Na Figura 16 tem-se alguns exemplos de descrições dos algoritmos *March* clássicos.

Figura 16 – Exemplo de descrições de algoritmos *March*.

Name	Ref.	algorithm
MATS	[NAI79]	{⎮(w0);⎮ (r0,w1);⎮(r1)}
MATS+	[ABA83]	{⎮(w0);⇑(r0,w1); ⇓(r1,w0)}
MATS++	[VAN91]	{⎮(w0);⇑(r0,w1);⇓(r1,w0,r0)}
March X	[VAN91]	{⎮(w0);⇑(r0,w1);⇓(r1,w0);⎮(r0)}
March C-	[MAR82]	{⎮(w0);⇑(r0,w1);⇑(r1,w0);⇓(r0,w1);⇓(r1,w0);⎮(r0)}
March A	[SUK81]	{⎮(w0);⇑(r0,w1,w0,w1);⇑(r1,w0,w1);⇓(r1,w0,w1,w0);⇓(r0,w1,w0)}
March Y	[VAN91]	{⎮(w0);⇑(r0,w1,r1);⇓(r1,w0,r0);⎮(r0)}
March B	[SUK81]	{⎮(w0);⇑(r0,w1,r1,w0,r0,w1);⇑(r1,w0,w1);⇓(r1,w0,w1,w0); ⇓(r0,w1,w0)}
March GS	[VAN93]	{⎮(w0);⇑(r0,w1,r1,w0,w1);⇑(r1,w0,r0,w1);⇓(r1,w0,w1,w0); ⇓(r0,w1,r1,w0);Del;⎮(r0,w1,r1);Del;⎮(r1,w0,r0)}
March M	[MIK96]	{⎮(w0);⇑(r0,w1,r1,w0);⎮(r0);⇑(r0,w1);⎮(r1);⇑(r1,w0,r0,w1,);⎮(r1);⇓(r1,w0)}
March LR	[VAN96]	{⎮(w0);⇓(r0,w1);⇑(r1,w0,r0,w1);⇑(r1,w0);⇑(r0,w1,r1,w0,);⇑(r0)}
March U	[VAN97]	{⎮(w0);⇑(r0,w1,w0,w1,r1);⇑(r1,w0,w1,w0,r0);⇓(r0,w1,w0,w1,r1);⇓ (r1,w0,w1,w0,r0; ⇓ (r0)}
March LA	[VAN99]	{⎮(w0);⇑(r0,w1,r1,w0);⇑(r0,w1); ⇓ (r1,w0,r0,w1); ⇓ (r1,w0)}
March SR	[VAN00]	{⇓ (w0);⇑(r0,w1,r1,w0);⇑(r0,r0););⇑(w1),⇓ (r1,w0,r0,w1); ⇓ (r1,r1)}
March SS	[HAM02]	{⎮(w0);⇑(r0,r0,w0,r0,w1);⇑(r1,r1,w1,r1,w0);⇓(r0,r0,w0,r0,w1);⇓ (r1,r1,w1,r1,w0; ⎮(r0)}

Fonte: Landrault (2010).

Na Tabela 6 é possível analisar a comparação entre o percentual de cobertura de falhas dos diversos algoritmos *March*. Neste comparativo, o algoritmo com maior cobertura de falhas é o

March SS, que cobre 100% dos modelos de falhas descritos (LANDRAULT, 2010).

Tabela 6 – Percentual de cobertura para falhas estáticas *single-cell* e *two-cell* para FFMs (*Functional Fault Models*) utilizando algoritmos *March*.

Algorithm	Single-Cell FFMs						Two-cell FFMs							Operation count
	SF	TF	WDF	RDF	DRDF	IRF	CFst	CFds	CFtr	CFwd	CFrd	CFdrd	Cfir	
MATS+	100	50	-	100	-	100	50	25	25	-	50	-	50	5.n
March B	100	100	-	100	-	100	75	62	50	-	50	-	50	17.n
March U	100	100	-	100	-	100	100	66	100	-	100	-	100	13.n
March C-	100	100	-	100	-	100	100	66	100	-	100	-	100	10.n
March LR	100	100	-	100	-	100	100	66	100	-	100	-	100	14.n
March SR	100	100	-	100	100	100	100	66	100	-	100	75	100	14.n
March SS	100	100	100	100	100	100	100	100	100	100	100	100	100	22.n

Fonte: Landrault (2010).

Na Tabela 7 tem-se o algoritmo com maior cobertura de falhas, o *March LA*, que abrange quase que todas falhas.

Tabela 7 – Percentual de cobertura para falhas dinâmicas para FFMs (*Functional Fault Models*) utilizando algoritmos *March*.

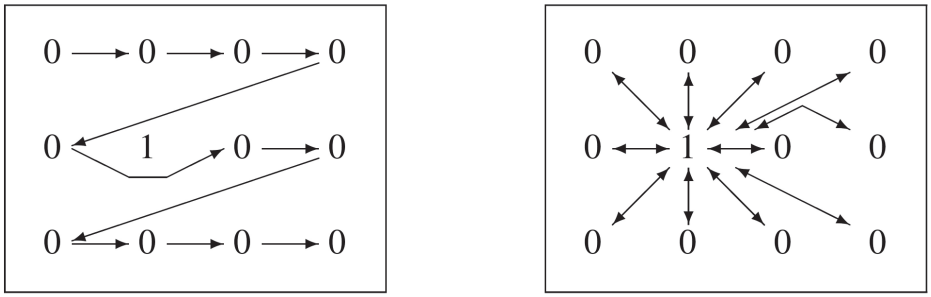
FFM	March Tests							
	MATS+ ($5n$)	March C- ($10n$)	March B ($17n$)	PMOVI ($13n$)	March U ($13n$)	March SR (14n)	March LA (22n)	March LR (14n)
dRDF	0%	0%	50%	50%	50%	50%	50%	50%
dDRDF	0%	0%	0%	50%	0%	0%	50%	0%
dIRF	0%	0%	50%	50%	50%	50%	50%	50%
dCFds$_{xw\overline{x}r\overline{x}}$	0%	0%	50%	87.5%	50%	50%	100%	50%
dCFds$_{xwxrx}$	0%	0%	0%	0%	0%	0%	0%	0%
dCFrd	0%	0%	25%	50%	25%	25%	50%	25%
dCFdrd	0%	0%	0%	37.5%	0%	0%	50%	0%
dCFir	0%	0%	25%	50%	25%	25%	50%	25%

Fonte: Hamdioui, Al-Ars e Goor (2002).

Além dos algoritmos *March*, tem-se os chamados algoritmos tradicionais, que possuem boa aplicação para os já mencionados testes *base-cell*, para o qual é necessário realizar a varredura da memória célula à célula. Pode-se citar como algoritmos clássicos: *Zero-one*, *Checkerboard*, *GALPAT* e *Walking 1/0*. A diferença principal entre eles encontra-se na orientação lógica da escrita e leitura das células da memória. Por exemplo, o *GALPAT* vem do termo *GALloping PATtern*, no qual a memória é preenchida com 0s (ou 1s), exceto na célula base, que contém o nível lógico oposto. Durante o teste a célula base se desloca através da memória, a fim de varrer todas as células. Já o *Walking 1/0* é semelhante, porém a leitura das células ocorre incluindo a

célula base, ao contrário do *GALPAT*. No primeiro são lidas as demais células e posteriormente ocorre a leitura da célula base (Figura 17) (JHA; GUPTA, 2003).

Figura 17 – Exemplos de operações de leitura para algoritmos *Walking 1/0* e *GALPAT* .



Walking 1/0 *GALPAT*

Fonte: Jha e Gupta (2003).

Além dos algoritmos tradicionais, outros algoritmos foram desenvolvidos visando especificamente identificar as chamadas falhas NPSF (ver Capítulo 2.2.1). Para cobertura das *Neighborhood Pattern Sensitive Faults*, que podem ser ANPSF, PNPSF ou SNPSF, algoritmos mais robustos tiveram de ser desenvolvidos. Isso porque, o tempo de teste se fossem realizadas as escritas e leituras para a abrangência de todas as possíveis interações seria impeditivamente grande. Além disso, existe outro agravante para cobertura de falhas NPFS, o qual está relacionado com a diferença entre o endereço lógico da célula de memória e o endereço físico (ver Capítulo 2.1.2). Técnicas de redução de escritas (*tilling method*) e de divisão da matriz de células em grupos (*two -group method*) foram criadas, surgindo assim algoritmos como TDANPSF1G, TLSNPSF1T, TLAPNPSF2T, entre outros.

2.3.2.1 Teste Word-Oriented

Quando trata-se de memórias, deve-se ter em mente que realizam-se operações em inúmeras células e que é necessário levar em consideração qual é a forma que estas células estão sendo acessadas. No caso da memória DRAM, o acesso basicamente ocorre em *data bus* (descrito no capítulo 2.1.2), ou seja, em pacotes de dados e não bit a bit. Por outro lado, a maioria dos algoritmos de teste são descritos visando o teste em uma única célula, o que neste caso demanda uma estratégia para o teste orientado por palavras de dados (*word-oriented test)* (GOOR; ABADIR; CARLIN, 2002).

Uma das estratégias utilizada refere-se a utilização de padrões de dados (*data patterns* ou *data backgrounds*) nas escritas dos algoritmos, além da inversão destes padrões de dados. Desta forma, obtém-se um nível de cobertura de falhas elevado, pois utilizam-se valores de dados em diferentes padrões, elevando a rigorosidade do teste. Na Tabela 8 é possível ver 10 padrões de dados de 16 bits para um teste de falhas de acoplamento (*CFs*), para o qual utiliza-se o padrão normal e o invertido.

Tabela 8 – Padrões de dados utilizados em testes *word-oriented*.

Normal			**Inverso**		
Padrão	Binário	Hexa	Padrão	Binário	Hexa
0	0000 0000 0000 0000	0x0000	1	1111 1111 1111 1111	0xFFFF
2	0101 0101 0101 0101	0x5555	3	1010 1010 1010 1010	0xAAAA
4	0011 0011 0011 0011	0x3333	5	1100 1100 1100 1100	0xCCCC
6	0000 1111 0000 1111	0x0F0F	7	1111 0000 1111 0000	0xF0F0
8	0000 0000 1111 1111	0x00FF	9	1111 1111 0000 0000	0xFF00

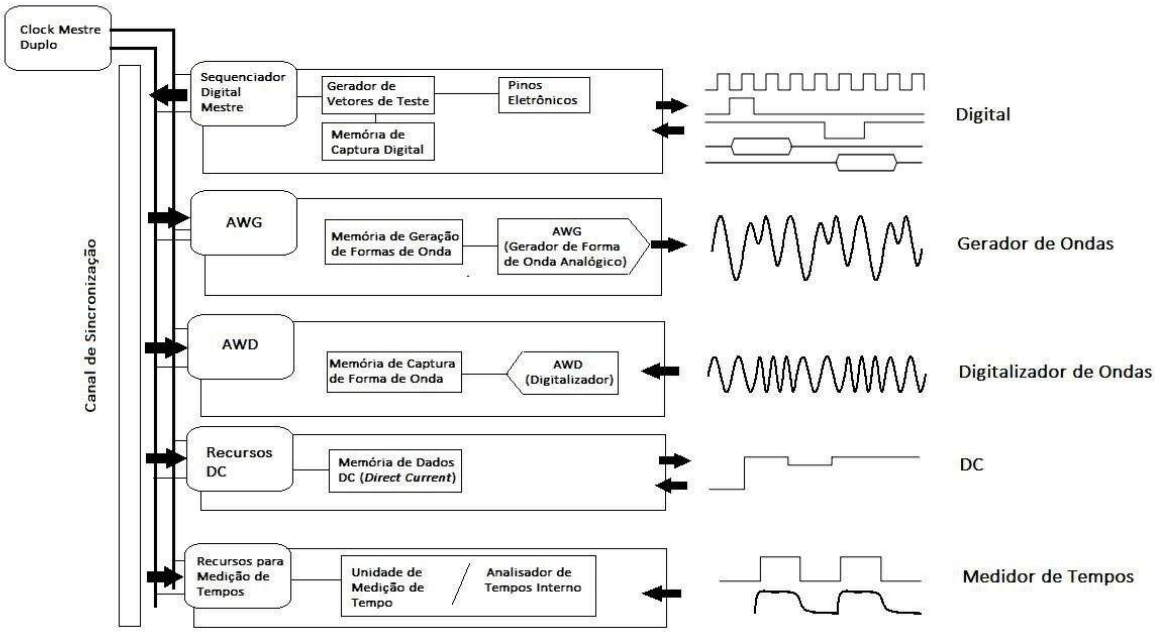
Fonte: Goor, Abadir e Carlin (2002).

2.3.3 ATE - Automatic Test Equipment

A fim de executar um teste com maior confiabilidade, velocidade, qualidade e principalmente alto volume de produção, foram desenvolvidos os dispositivos denominados de ATE (*Automatic Test Equipment*) (BUSHNELL; AGRAWAL, 2002). Um ATE possibilita que sejam testados mais de um dispositivo simultaneamente e aplicando *test patterns* (padrões de teste) pré-definidos. Os ATEs são equipamentos programáveis que possuem uma grande quantidade de alocação para DUTs (*Device Under Test*), ou seja, para que possam ser testados paralelamente vários componentes utilizando-se um mesmo padrão de teste. Estes dispositivos são a principal ferramenta utilizada para testes a nível industrial de circuitos integrados com alto volume de produção, dentre eles as memórias DRAM (BUSHNELL; AGRAWAL, 2002).

Os ATEs podem ser classificados em categorias, conforme a aplicação a ser testada, são elas: digitais, analógicos, *mixed-signals* (digital e analógico) e de memórias.

Figura 18 – Diagrama de blocos genérico para um ATE.



Fonte: Adaptado de Jone (2013).

Conforme pode-se ver na Figura 18, um ATE está dividido em 5 blocos principais: digital, gerador de ondas, digitalizador de ondas, DC e medidor de tempos. Cada um deles executa funções distintas em um ATE, porém estão sincronizados pelo que está representado na figura como canal de sincronização. Todos os blocos possuem também sincronização pelo sinal de *clock* que é único para ambos (JONE, 2013).

Além disto, uma estratégia utilizada na maioria dos testadores, refere-se a utilização de um equipamento de manipulação (*handler*) que realiza a preparação e classificação das peças testadas automaticamente, possibilitando assim o teste *multi-site*, ou seja, o teste em múltiplos DUTs simultaneamente, o que proporciona o alto paralelismo de teste dos ATEs.

2.3.4 Custo de Teste

Usualmente, já utilizam-se ATEs visando o aumento da produtividade, com foco na redução do tempo de teste (*test time reduction*) e aumento do paralelismo (DUTs em paralelo). Conforme anteriormente ilustrado na Figura 5, o custo para fabricação de um transistor apresenta uma tendência decrescente, diferentemente do custo de teste de um transistor, que apresenta-se estável a mais de 30 anos (KHOO, 2014). Portanto, o custo de adquisição de um testador ATE acaba sendo um dos principais responsáveis pelo custo do teste ser elevado. Obviamente, o custo final deste equipamento depende de suas configurações (digital, analógico, quantidade de canais), o que por sua vez depende da família de circuitos que se pretende testar. Conforme demonstra Lubaszewski2002TesteIntegrados, é possível mensurar o custo de teste conforme o exemplo a

seguir: Um ATE para aplicações de sinal misto, com capacidade de aplicar vetores de teste a uma taxa de 1 GHz e 1024 pinos de I/O foi comprado a um preço de US$ 4M (quatro milhões de dólares). Considerando os custos relacionados à depreciação, manutenção e operação, obteve-se o custo anual de operação do ATE: US$ 1,5M/ano. Se o testador for utilizado 24 horas por dia, 7 dias por semana, podemos calcular o custo do teste em função do tempo de teste:

Custo do Teste = $U\$1,5M/(24x365x3600) = 4,75cents/segundo$

Considerando um circuito integrado que leva 5 segundos para ser testado, o custo do teste deste componente individualmente é 23,75 centavos de dólar. No entanto, nem todos os circuitos passam no teste final de produção. Logo, o custo do teste deve ser diluído entre os circuitos que serão vendidos (os que passaram nos testes). Considerando para o exemplo um *yield* (aprovados) de 70% (ou seja, 30% dos circuitos não são aproveitados) o custo final do teste por unidade é 33,9 centavos de dólar.

Desta forma, espera-se que o tempo de teste seja sempre o menor possível, visando minimizar o seu custo, porém sem perder o nível de qualidade do teste. Uma alternativa para reduzir o tempo de teste é utilizar testadores com a capacidade de testar vários CIs em paralelo (paralelismo elevado), como usualmente já são hoje em dia os ATEs (LUBASZEWSKI; COTA; KRUG, 2002). Além do tempo de teste, o custo total de teste pode levar em consideração demais fatores, que podem ser classificados como custo fixo e custo variável, somando-se para compor o custo total. No custo fixo deve-se considerar os **custos de depreciação**, **custos de mão-de-obra direta** e **despesas gerais**, os quais podem compor (KHOO, 2014):

- **Custos de depreciação**: custo do testador e custo do manipulador (*handler*).
- **Custos de mão-de-obra**: salários dos operadores e dos técnicos de manutenção.
- **Despesas gerais**: custos de gerenciamento (salário dos gerentes, supervisores e engenheiros), das facilidades (eletricidade, ar comprimido, nitrogênio líquido), do espaço fabril, de manutenção e dos acessórios (DSA, soquetes, placas de calibração).

Além do custo fixo, têm-se o variável, o qual compõe **custo da reposição de peças e partes danificadas** do testador. Com estes fatores inclusos, o custo efetivo por componente testado acaba gerando valores ainda mais elevados do que aqueles calculados no exemplo anterior.

2.4 Trabalhos Similares e Estado da Arte

Durante a pesquisa realizada para a elaboração deste projeto encontrou-se trabalhos que se assemelham parcialmente ao objetivo deste. Entretanto, nenhum deles apresenta de forma objetiva o desenvolvimento de um programa de testes, mas sim, a análise de cobertura de falhas dos algoritmos utilizados, técnicas a serem utilizadas em algoritmos já existentes, análise de padrões de dados e técnicas de estresse durante a execução de testes de memória SDRAM.

Alguns autores, também apresentam estudos referentes ao custo de teste e as estratégias a serem utilizadas visando sua redução, o que por sua vez apresenta uma relação de similaridade com este projeto.

O artigo científico "*An Industrial Evaluation of DRAM Tests*"de GOER; NEEF, aborda de forma mais complexa uma análise de cobertura de falhas e variações de estresses (temperatura, tensão de operação, etc) com desenvolvimento de um programa de testes à nível industrial para memórias DRAM. O objetivo principal era a avaliação de diferentes algoritmos de testes e suas relações com diferentes SC (*stress combination*). Neste estudo GOER; NEEF utilizou 40 conhecidos algoritmos de teste para testar 1.896 diferentes componentes de memória DRAM 1M x 4b, além de aplicar 48 combinações de SC. O autor apresentou os resultados e os comparativos entre a cobertura de falhas dos algoritmos, o que foi de grande auxílio na etapa de seleção dos algoritmos de teste aplicados neste trabalho.

Outro trabalho similar realizado por RAMACHANDRAN, 2014, apresentou um artigo referente a "*DRAM testing using interleaving test algorithm*", no qual foram demonstradas estratégias de teste através da intercalação de diferentes algoritmos de testes. O foco principal foi a detecção de falhas paramétricas, principalmente por fuga de corrente, e foram desenvolvidos algoritmos intercalados para detecção destes tipos de falhas. Os testes utilizados neste caso são chamados de *retention tests*, que por sua vez se diferem um pouco dos testes funcionais. Em sua análise, o autor apresentou uma intercalação de algoritmos com cobertura de 35,2% para falhas *sub-threshold leakage current*, o que foi tratado como um resultado muito satisfatório (RAMACHANDRAN, 2014).

No artigo apresentado por GOOR; ABADIR; CARLIN, denominado *Minimal Test for Coupling Faults in Word-Oriented Memories*, foi realizada uma análise e apresentada uma solução que visava a redução do padrão de dados para teste de falhas do tipo CF (*coupling faults*). A semelhança entre este trabalho e o desenvolvido nesta dissertação refere-se principalmente pelo uso do método *word-oriented* e pela problemática relacionada ao tempo para execução de padrões de dados de teste, que visam uma maior cobertura de falhas. Os autores GOOR; ABADIR; CARLIN apresentaram uma solução com os detalhes dos algoritmos utilizados para detecção de falhas CF, no caso, os algoritmos MATS++ e *March C*. Após o estudo, apresentaram a solução com redução de tempo de teste entre 20% e 38%. Outro artigo, também elaborado pelo autor *A.J. van de Goor* apresentou uma proposta semelhante, porém com foco em conversão de algoritmos *March* de orientação a bit para orientação a palavra. O artigo chamado *Converting March Tests for Bit-Oriented Memories into Tests for Word-Oriented Memories* apresentou uma análise semelhante a do artigo anterior, com foco na alteração dos padrões de dados visando atingir o nível de cobertura de falhas existente, sem afetar a qualidade do teste (GOOR et al.,).

Já em uma linha de desenvolvimento de teste, têm-se a tese *DC Parametric Test and IDDQ Test Using Advantest T2000 ATE* do autor DING, que apresentou o desenvolvimento de um teste DC paramétrico e um teste de IDDQ utilizando um testador ATE modelo T2000 da Advantest. O

autor apresentou detalhadamente o desenvolvimento do programa de teste DC, que teve como aplicação o teste de componentes VLSI (*Very Large Scale Integrated Circuit*). Todas as definições dos modelos de falhas, dos parâmetros e dos algoritmos utilizados para a execução do teste foram apresentados, assim possuíndo grande similaridade com a esta dissertação. Com o viés de desenvolvimento, têm-se a dissertação de mestrado em Engenharia Elétrica do autor BONATTO, que apresentou o desenvolvimento de um IP de um controlador de memória SDRAM DDR com função de BIST (*Built-In Self-Test*) integrada, validando o correto funcionamento do controlador. Com o trabalho denominado *Núcleos de Interface de Memória DDR SDRAM para Sistemas-Em-Chip*, o autor utilizou toda fundamentação e embasamento teórico do funcionamento de memórias SDRAM, bem como testes de memórias e suas peculiaridades. Existe uma similaridade na sua análise de testes e nas problemáticas abordadas referente ao funcionamento da SDRAM (tempos, inicialização, *refresh*, etc), com relação as etapas existentes no decorrer deste projeto.

Por fim, em uma abordagem que apresenta o contexto de custo de teste, têm-se o artigo *Cost of Test Case Study for Multi-site Testing in Semiconductor Industry with Firm Theory* do autor KHOO. Este artigo apresenta elementos que sustentam o alto custo do teste de semicondutores, apresentando os fatores de contribuição e cálculos comprobatórios. Estes elementos corroboram com a esta dissertação, pois nela são elencados diversos cenários que podem demandar despesas no que se diz respeito ao teste de componentes. Contudo, KHOO apresentou uma solução de teste de multi-sites trazendo uma alternativa mais econômica para o problema abordado.

3 METODOLOGIA

Neste capítulo serão apresentados os materiais e ferramentas utilizados no desenvolvimento deste trabalho, bem como os métodos e procedimentos empregados para a execução do mesmo. De acordo com os conceitos teóricos fundamentados no capítulo 2, obtém-se os esclarecimentos necessários para o entendimento acerca da inicialização e funcionamento memórias DRAM, a linguagem de programação do testador Magnum V, algoritmos de testes, operação do testador TCE3200LP e demais temas pertinentes para a execução deste trabalho.

3.1 Pesquisa do tema

Para atingimento dos objetivos foi necessário o entendimento das ferramentas utilizadas no desenvolvimento de um programa de testes no equipamento ATE Magnum V, as quais são baseadas na linguagem de programação C++ e no *Visual Studio* da Microsoft, além da linguagem própria do testador para desenvolvimento dos algoritmos de teste (*test patterns*) em *APG Pattern Instruction*. Além dos conhecimentos referente a programação, teve-se como um fator determinante a necessidade do entendimento do componente testado, neste caso, memórias SDRAM DDR4. Para isto, utilizou-se o material relacionado na *Revisão Bibliográfica* (capítulo 2), além da documentação técnica deste componente, fornecida pela norma JEDEC (JEDEC Standard, 2012) e pelo *datasheet* do próprio fabricante (HYNIX, 2017). Nestes documentos é possível encontrar os protocolos para funcionamento das memórias SDRAM, que englobam os processos de inicialização, ativação e comandos de operação, além da definição dos tempos de ciclos e operações, que são necessários para inicialização e demais execuções da memória.

Entretanto, vale ressaltar a complexidade e limitação para aquisição de informações, principalmente relacionadas a estrutura e funcionamento do componente sob teste, que se dá devido as políticas de propriedade intelectual e segredos industriais. Por se tratar de um mercado altamente tecnológico e competitivo, a propriedade da informação e das técnicas de desenvolvimento acabam sendo altamente valiosas para a concorrência entre as empresas fabricantes destes componentes de memória.

3.2 Materiais e ferramentas

Os materiais e ferramentas utilizados foram previamente definidos, a fim de assegurar a completa execução do projeto. A seguir, serão descritos os materiais, ferramentas e equipamentos utilizados no decorrer do desenvolvimento deste trabalho.

3.2.1 Memória SDRAM DDR4

Memórias DDR são normatizadas pelo órgão internacional *Joint Electronic Device Engineering Council* (JEDEC), que passou a definir um padrão de estrutura para as memórias, especificações dos pinos, dimensões, protocolos de funcionamento, entre outras definições. A norma que atualmente regulamenta as especificações das memórias DDR SDRAM é a JESD79 e, especificamente as memórias DDR4 é a JESD79-4 (última revisão disponível de junho de 2017). Através da utilização norma, teve-se algumas definições que foram importantes para o andamento deste projeto. Outro material importante utilizado foi o *datasheet* do fabricante de memórias SK Hynix. Cada produto pode possuir características específicas, e o fato de utilizar o próprio *datasheet* da memória acabou por facilitar os estudos. O componente em análise foi o de *part number* H5AN8G6NAFR-xxC, que corresponde a uma memória SDRAM DDR4 8 Gb, sendo sua configuração interna de 512 M x 16b e frequência de operação podendo chegar até 3200 MHz. Possui encapsulamento FBGA com 96 pinos (esferas), o qual tem a configuração dos pinos da memória com vista de topo, ou seja, vista das esferas através do encapsulamento. Na Figura 19 tem-se a posição da matriz de pinos na memória, com um desenho de um encapsulamento FBGA (JEDEC Standard, 2012).

Figura 19 – Configuração de pinos da SDRAM DDR4 96FBGA (vista de topo).

	1	2	3	4	5	6	7	8	9	
A	VDDQ	VSSQ	DQU0				DQSU_c	VSSQ	VDDQ	A
B	VPP	VSS	VDD				DQSU_t	DQU1	VDD	B
C	VDDQ	DQU4	DQU2				DQU3	DQU5	VSSQ	C
D	VDD	VSSQ	DQU6				DQU7	VSSQ	VDDQ	D
E	VSS	DMU_n/ DBIU_n	VSSQ				DML_n/ DBIL_n	VSSQ	VSS	E
F	VSSQ	VDDQ	DQSL_c				DQL1	VDDQ	ZQ	F
G	VDDQ	DQL0	DQSL_t				VDD	VSS	VDDQ	G
H	VSSQ	DQL4	DQL2				DQL3	DQL5	VSSQ	H
J	VDD	VDDQ	DQL6				DQL7	VDDQ	VDD	J
K	VSS	CKE	ODT				CK_t	CK_c	VSS	K
L	VDD	WE_n A14	ACT_n				CS_n	RAS_n A16	VDD	L
M	VREFCA	BG0	A10/AP				A12 BC_n	CAS_n A15	VSS	M
N	VSS	BA0	A4				A3	BA1	TEN	N
P	RESET_n	A6	A0				A1	A5	ALERT_n	P
R	VDD	A8	A2				A9	A7	VPP	R
T	VSS	A11	PAR				NC	A13	VDD	T
	1	2	3	4	5	6	7	8	9	

Fonte: Hynix (2017).

Os pinos da memória podem possuir diferentes funções (entradas, saídas e *clock*). A seguir os principais pinos e as descrições de suas funcionalidades são descritos (HYNIX, 2017):

- **Clock:** CK e $\bar{C}K$ são as entradas diferenciais de clock. Todos endereços e controles de entradas de sinais são amostrados no cruzamento da borda de subida de CK e a borda de descida de $\bar{C}K$;

- **Clock Enable:** CKE em nível alto está ativo, e CKE em nível baixo desativa os sinais de clocks internos, buffers de entrada e drivers de saída. Mantendo CKE em nível baixo fornece operações de Precharge Power-Down e Self-Refresh (todos bancos inativos), ou Active Power-Down (linhas ativas em qualquer banco) CKE é assíncrono para saída do Self-Refresh. Depois de VREFCA e VREFDQ tornaram-se estáveis durante a alimentação e sequência de inicialização, eles devem ser mantidos durante todas as operações (incluindo Self-Refresh). CKE deve ser mantido em nível alto durante os acessos de leitura e escrita. Buffers de entrada, excluindo CK, $\bar{C}K$, ODT e CKE, são desabilitados durante o desligamento. Buffers de entrada, excluindo CKE, são desabilitados durante o Self-Refresh;

- **Chip Select:** todos comandos são mascarados quando $\bar{C}S$ é registrado em nível alto. $\bar{C}S$ estabelece a seleção do *Rank* externo em sistema com *Ranks* múltiplos. $\bar{C}S$ é considerado parte do código de comando;

- **On Die Termination:** ODT (em nível alto) permite a resistência de terminação interna para o DDR4 SDRAM. Quando ativado, ODT só é aplicada a cada DQ, DQS, $\bar{D}QS$ e DM/TDQS, $\bar{T}DQS$ (quando TDQS é habilitado através do Modo de Registo A11 = 1 em MR1) sinal para as configurações x8. Para configurações x16 o ODT é aplicado para cada sinal de $DQ, DQSU_c, DQSU_t, DQSL_t, DQSL_c, DMU_n, eDML_n$. O pino de ODT será ignorado se MR1 está programado para desabilitar RTT_NOM;

- **Comandos de Entrada:** RAS_n, CAS_neWE_n junto com CS_n definem o comando a ser digitado;

- **Máscara de Dados de Entrada**: DM é um sinal de máscara de entrada para os dados de escrita. Dados de entrada são mascarados quando DM é amostrado em nível alto coincidindo com que a entrada de dados durante um acesso de escrita. DM é amostrado em ambas as bordas de DQS;

- **Entradas de Endereçamento de Bancos**: BA0 - BA2 definem qual banco uma Ativação, Leitura, Escrita ou comando de Precharge está sendo aplicado. *Bank Adress* também determina se o mode register ou extender mode register será acessado durante a MRS *cycle*;

- **Entradas de Endereços**: Fornece os endereços de linha para comandos de ativação e de endereços de coluna para comandos de Leitura/Escrita para selecionar um local fora da matriz de memória no respectivo banco (A10/AP e A12/BC tem funções adicionais, ver

abaixo). Os endereços de entradas fornecem o op-code durante os comandos de Mode Register Set;

- **Auto-precharge**: A10 / AP é amostrado durante comandos de Read/Write para determinar se Autoprecharge pode ser realizado para acessar o banco somente após a operação de Read/Write. (Nível Alto: Autoprecharge; Nível Baixo: sem Precharge). A10 é amostrado durante um comando de Precharge para determinar se o Precharge deve ser aplicado para um banco (A10 nível baixo) ou todos bancos (A10 nível alto). Se somente um banco necessita de Precharged, o banco será seleciona pelos Bank Adresses (BA);

- **Burst Chop**: A12 / BC é amostrado durante os comandos de Read e Write para determinar se o burst chop (on-the-fly) será realizada. (Nível Alto, sem burst chop; Nível Baixo: burst chopped). Embora o banco ainda está ocupado e indisponível para outros comandos até oito vezes de transferência se passarem, um banco diferente já pode ser acessado;

- **Ativação do Reset Assíncrono**: Reset é ativado quando RESET está em nível baixo, e inativo quando RESET está em nível alto. RESET deve estar em nível alto durante operação normal. RESET é um sinal CMOS rail-to-rail com DC alto e baixo entre 80% e 20% de VDD, aprox. 1.20 V para DC nível alto e 0.30 V para DC nível baixo;

- **Data I/O**: Entrada e Saída de dados (bidirecional);

- **Data Strobe**: Saída com os dados de leitura, entrada com os dados de escrita. DDR4 SDRAM suporta somente data strobe diferencial e não tem suporte a single-ended;

- **Termination Data Strobe**: TDQS, $TDQS_n$ é aplicável somente somente para x8 DRAMs. x4/x16 DRAMs devem desabilitar a função TDQS através do mode register A11 = 0 no MR1.

O nome, tipo e breve descrição de cada um dos pinos da memória se dá conforme representado na Tabela 9. Além da configuração dos pinos, uma memória SDRAM possui internamente registradores de modo (*mode registers*) que devem ser configurados para atender a aplicação a qual a memória será utilizada. Estes registradores estão descritos no *datasheet* do componente (HYNIX, 2017) e também padronizados pela norma JEDEC (JEDEC Standard, 2012). A configuração dos registradores ocorre no momento da inicialização da memória e estes estão definidos para atender as especificações de testes utilizadas. Para isto, estudara-se cada um dos registradores de modo e as melhores opções para realização do teste na ATE Magnum V. No Apêndice A estão as configurações definidas para os sete registradores existentes na memória SDRAM DDR4 8Gb x16. A primeira tabela de cada registrador contém os valores em binário e hexadecimal dos bits de endereço que são responsáveis pela configuração de cada um dos *mode registers*. Já a segunda tabela contém a descrição dos valores definidos nas configurações.

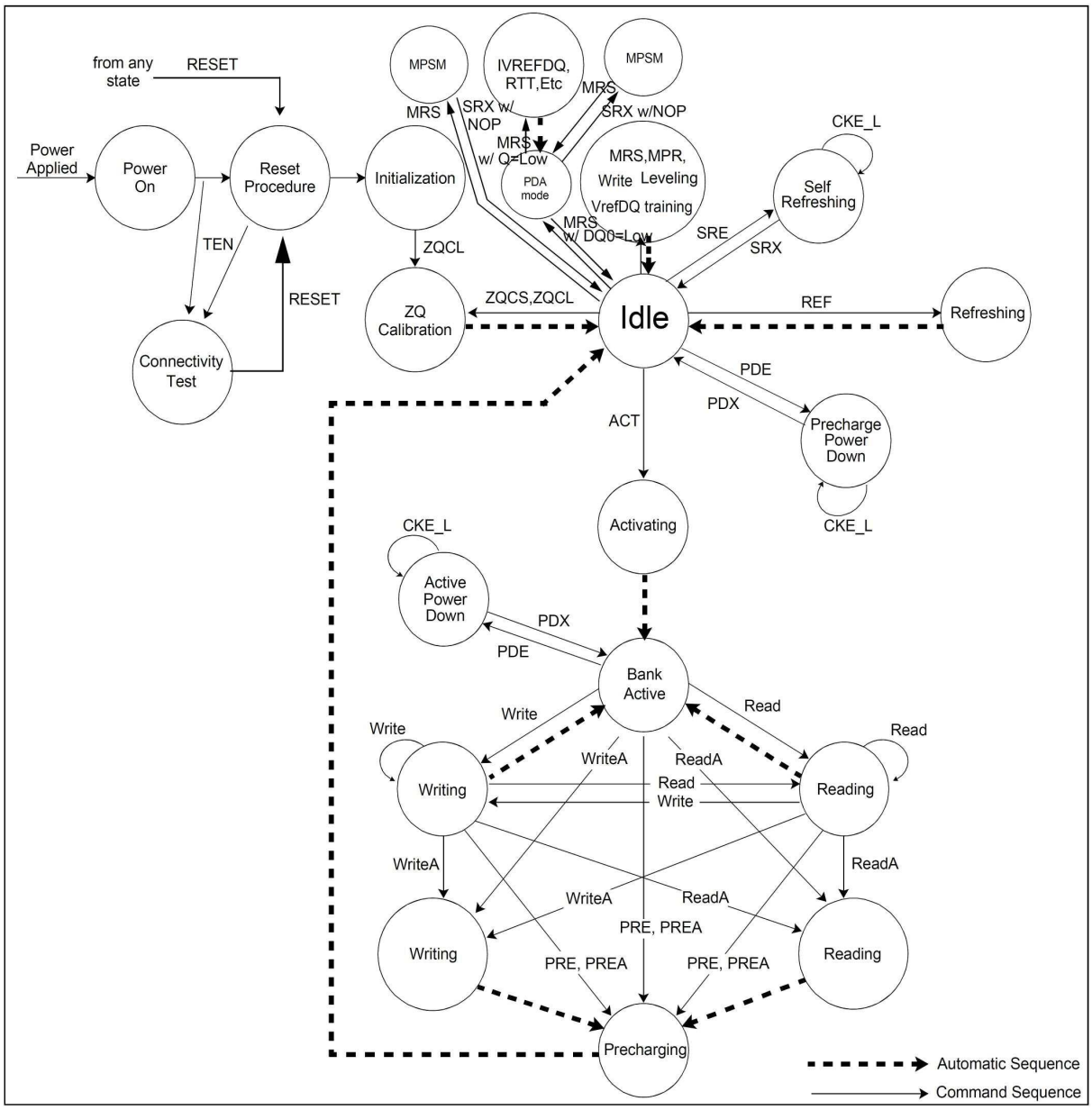
Tabela 9 – Descrição funcional dos pinos DDR4 512x16.

Nome	**Tipo**	**Descrição**
CK_t, CK_c	Entrada	*Clock*
CKE	Entrada	*Clock Enable*
CS_n	Entrada	*Chip Select*
$C0, C1, C2$	Entrada	*Chip ID*
ODT	Entrada	*On Die Termination*
ACT_n	Entrada	*Activation Command*
RAS_n, CAS_n, WE_n	Entrada	*Adress Commands*
$DM_n, DBI_n, TDQS_t$	Entrada/Saída	*Configuration Pins*
$BG0, BG1$	Entrada	*Bank Group Inputs*
$BA0, BA1$	Entrada	*Bank Adress Inputs*
$A0-A17$	Entrada	*Adress Inputs*
$A10/AP$	Entrada	*Auto-precharge*
$A12/BC_n$	Entrada	*Burst Chop*
$RESET_n$	Entrada	*Active Low Asynchronous Reset*
DQ	Entrada/Saída	*Data Input/Output*
DQS	Entrada/Saída	*Data Strobe Input/Output*
$TDQS$	Saída	*Termination Data Strobe*
PAR	Entrada	*Command and Adress Parity Input*
$ALERT_n$	Saídat	*Error Flag*
TEN	Entrada	*Connectivity Test Mode Enable*
NC	-	*No Connect*
V_{DDQ}	Alimentação	*DQ Power Supply*
V_{SSQ}	Alimentação	*DQ Ground*
V_{DD}	Alimentação	*Power Supply*
V_{SS}	Alimentação	*Ground*
V_{PP}	Alimentação	*Activation Power Supply*
V_{REFCA}	Alimentação	*Reference Voltage for CA*
ZQ	Alimentação	*Reference Pin for ZQ calibration*

Fonte: Adaptado de Hynix (2017).

Outro componente fundamental para entendimento da memória e realização de suas operações é o Diagrama de Estados Simplificado (Figura 20). Nele estão ilustrados os estados possíveis em que uma memória SDRAM DDR4 pode estar e principalmente em que estado ela deve estar para que ocorram determinadas operações. Basicamente, a partir do estado denominado *Idle* é que a memória está apta para realização de suas operações, ou seja, deve-se realizar alguns procedimentos previamente as operações para que a memória seja inicializada (ver capítulo 3.6).

Figura 20 – Diagrama de estados simplificado de uma memória SDRAM DDR4.



Fonte: JEDEC Standard (2012).

Além disto, quando refere-se aos comandos executados na memória, trata-se basicamente de uma série de combinações lógicas executadas pelos pinos de comando a fim de atenderem uma determinada condição na memória. Cada comando possui uma determinada função e, associado aos intervalos de tempos pré-determinados, realizam as operações da memória. Estes comandos também estão descritos na norma JEDEC e no *datasheet* do fabricante da memória, em uma tabela denominada *Tabela Verdade de Comandos* conforme ilustrado na Figura 21.

Figura 21 – Tabela verdade de comandos para memória SDRAM DDR4.

Function	Abbreviation	CKE		CS_n	ACT_n	RAS_n /A16	CAS_n /A15	WE_n/ A14	BG0-BG1	BA0-BA1	C2-C0	A12/ BC_n	A17, A13, A11	A10/ AP	A0-A9	NOTE
		Previous Cycle	Current Cycle													
Mode Register Set	MRS	H	H	L	H	L	L	L	BG	BA	V	OP Code				12
Refresh	REF	H	H	L	H	L	L	H	V	V	V	V	V	V	V	
Self Refresh Entry	SRE	H	L	L	H	L	L	H	V	V	V	V	V	V	V	7,9
Self Refresh Exit	SRX	L	H	H	X	X	X	X	X	X	X	X	X	X	X	7,8,9, 10
				L	H	H	H	H	V	V	V	V	V	V	V	
Single Bank Precharge	PRE	H	H	L	H	L	H	L	BG	BA	V	V	V	L	V	
Precharge all Banks	PREA	H	H	L	H	L	H	L	V	V	V	V	V	H	V	
RFU	RFU	H	H	L	H	L	H	H	RFU							
Bank Activate	ACT	H	H	L	L	Row Address(RA)			BG	BA	V	Row Address (RA)				
Write (Fixed BL8 or BC4)	WR	H	H	L	H	H	L	L	BG	BA	V	V	V	L	CA	
Write (BC4, on the Fly)	WRS4	H	H	L	H	H	L	L	BG	BA	V	L	V	L	CA	
Write (BL8, on the Fly)	WRS8	H	H	L	H	H	L	L	BG	BA	V	H	V	L	CA	
Write with Auto Precharge (Fixed BL8 or BC4)	WRA	H	H	L	H	H	L	L	BG	BA	V	V	V	H	CA	
Write with Auto Precharge (BC4, on the Fly)	WRAS4	H	H	L	H	H	L	L	BG	BA	V	L	V	H	CA	
Write with Auto Precharge (BL8, on the Fly)	WRAS8	H	H	L	H	H	L	L	BG	BA	V	H	V	H	CA	
Read (Fixed BL8 or BC4)	RD	H	H	L	H	H	L	H	BG	BA	V	V	V	L	CA	
Read (BC4, on the Fly)	RDS4	H	H	L	H	H	L	H	BG	BA	V	L	V	L	CA	
Read (BL8, on the Fly)	RDS8	H	H	L	H	H	L	H	BG	BA	V	H	V	L	CA	
Read with Auto Precharge (Fixed BL8 or BC4)	RDA	H	H	L	H	H	L	H	BG	BA	V	V	V	H	CA	
Read with Auto Precharge (BC4, on the Fly)	RDAS4	H	H	L	H	H	L	H	BG	BA	V	L	V	H	CA	
Read with Auto Precharge (BL8, on the Fly)	RDAS8	H	H	L	H	H	L	H	BG	BA	V	H	V	H	CA	
No Operation	NOP	H	H	L	H	H	H	H	V	V	V	V	V	V	V	10
Device Deselected	DES	H	H	H	X	X	X	X	X	X	X	X	X	X	X	
Power Down Entry	PDE	H	L	H	X	X	X	X	X	X	X	X	X	X	X	6
Power Down Exit	PDX	L	H	H	X	X	X	X	X	X	X	X	X	X	X	6
ZQ calibration Long	ZQCL	H	H	L	H	H	H	L	V	V	V	V	V	H	V	
ZQ calibration Short	ZQCS	H	H	L	H	H	H	L	V	V	V	V	V	L	V	

NOTE 1 All DDR4 SDRAM commands are defined by states of CS_n, ACT_n, RAS_n/A16, CAS_n/A15, WE_n/A14 and CKE at the rising edge of the clock. The MSB of BG, BA, RA and CA are device density and configuration dependant. When ACT_n = H; pins RAS_n/A16, CAS_n/A15, and WE_n/A14 are used as command pins RAS_n, CAS_n, and WE_n respectively. When ACT_n = L; pins RAS_n/A16, CAS_n/A15, and WE_n/A14 are used as address pins A16, A15, and A14 respectively
NOTE 2 RESET_n is Low enable command which will be used only for asynchronous reset so must be maintained HIGH during any function.
NOTE 3 Bank Group addresses (BG) and Bank addresses (BA) determine which bank within a bank group to be operated upon. For MRS commands the BG and BA selects the specific Mode Register location.
NOTE 4 "V" means "H or L (but a defined logic level)" and "X" means either "defined or undefined (like floating) logic level".
NOTE 5 Burst reads or writes cannot be terminated or interrupted and Fixed/on-the-Fly BL will be defined by MRS.
NOTE 6 The Power Down Mode does not perform any refresh operation.
NOTE 7 The state of ODT does not affect the states described in this table. The ODT function is not available during Self Refresh.
NOTE 8 Controller guarantees self refresh exit to be synchronous.
NOTE 9 VREF(VrefCA) must be maintained during Self Refresh operation. The first Write Leveling Activity may not occur earlier than TBD nCK after exit from Self Refresh.
NOTE 10 The No Operation command should be used in cases when the DDR4 SDRAM is in Gear Down Mode and Max Power Saving Mode Exit
NOTE 11 Refer to the CKE Truth Table for more detail with CKE transition.
NOTE 12 During a MRS command A17 is Reserved for Future Use and is device density and configuration dependent.

Fonte: JEDEC Standard (2012).

3.2.2 ATE Teradyne Magnum V

O desenvolvimento do programa de testes para teste de memórias utiliza um ATE modelo Magnum V do fabricante Teradyne. Este testador é dotado de *sites*, no qual cada um destes sites

possui um determinado número de canais que são direcionados aos soquetes para o teste de vários *DUTs* em paralelo. Estes soquetes são interligados ao testador por uma quantidade de canais que serão as linhas de acesso aos dispositivos a serem testados (MORAES; KONDO, 2017).

Figura 22 – ATE Magnum V - Teradyne .

Fonte: Moraes e Kondo (2017).

As especificações do equipamento indicam que a frequência máxima de operação é de 1600Mbps (*Max Data Rate*), ou seja, este será o limitante de velocidade para operações realizadas no programa de teste. As configurações de taxa de transferência, canais PE e canais DPS referentes a especificação exata do modelo do ATE que foi utilizado para a realização deste trabalho estão descritas na Tabela 10 .

Tabela 10 – Especificações Magnum V - Teradyne.

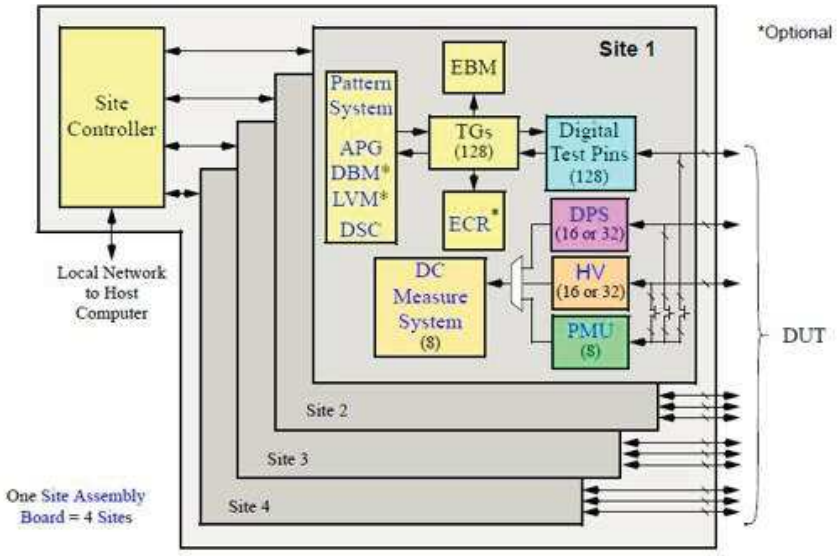
Item	**Configuração**
Assembly Boards	6 boards x 4 chassis, total 24 boards
Sites	4 sites x 24 boards, total 96 sites
Pin Eletronics	128 PE x 96 sites, total 12.288 PE
Dut Power Supply (DPS)	16 DPS x 96 sites, total 1.536 DPS
High Voltage Pin	16 HV x 96 sites, 1.536 HV
Max Clock Rate	800MHz
Max Data Rate	1600Mbps

Fonte: Nextest/Teradyne (2014b).

Dependendo do produto a ser testado é definida a quantidade máxima de DUTs que pode-se testar em cada um dos sites. Por exemplo, para o dispositivo eMMC com a configuração atual do equipamento *Magnum V* (conforme Tabela 10), pode-se ter 8 DUTs por site: 16 DPS (*DUT Power Supply*) por site divididos por dois VCCs (um VCC para a memória NAND e outro

VCCQ para o controlador), ou seja, máximo de 8 DUTs por site. Sendo assim, para teste de componentes eMMC no testador pode-se chegar a um paralelismo de 786 DUTs: 24 *Assembly Boards* = 96 sites = 8 DUTs por site. Cada *Site Assembly Board* é dotada de uma configuração de quatro sites, que contém internamente um sistema gerador de *patterns* (*APG - Automatic Pattern Generator*), sistema de medição DC (*DC Measure System*), fontes de alimentação dos DUTs (*DPS - DUT Power Supply*), pinos digitais (*PE - Pins Electronics*), pinos de alta tensão (*HV - High Voltage Pins*), uma unidade de medição paramétrica (*PMU - Parametric Measure Unit*), uma memória de armazenamento de falhas (*ECR - Error Catch RAM*), um gerador de sinais de tempo (*TG - Timming Generator*) e uma memória de armazenamento extendida (*EBM - Extended BUffer Memory*), além do *Site Controller* que realiza a comunicação com o computador central do testador *(Host Computer*). O diagrama de blocos da configuração de uma placa *Site Assembly Board* pode ser visto na Figura 23. Como é possível ver, cada *Site Assembly Board* é constituída

Figura 23 – Diagrama de blocos de uma *Site Assembly Board*.



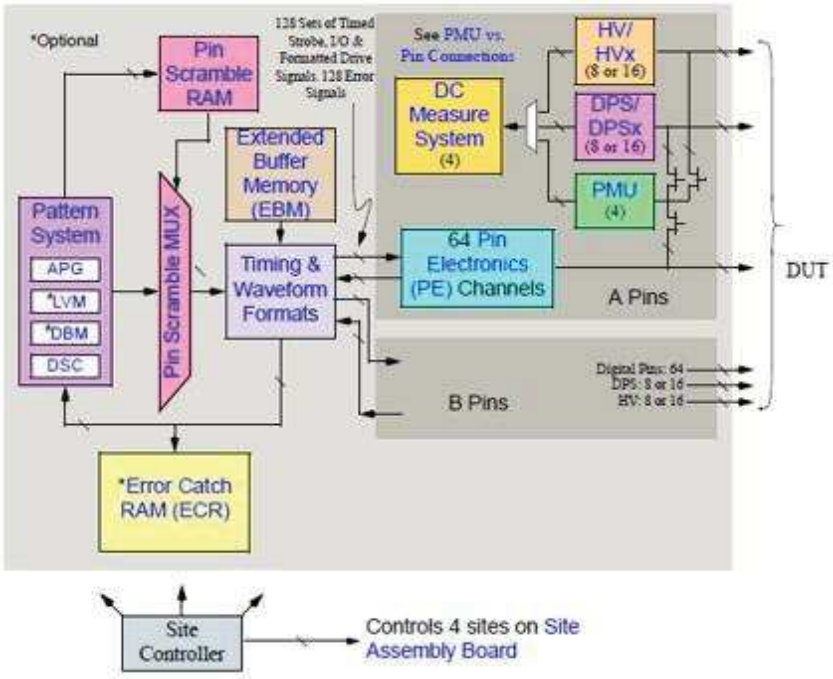
Fonte: Nextest/Teradyne (2014a)

por quatro sites. A configuração interna individualmente de cada site e as devidas conexões entre os blocos pode ser vista na Figura 24. Cada site é dotado de 64 canais (*PE - Pin Electronics Channels*), 16 fontes de alimentação de DUTs (*DPS - Dut Power Supply*), 16 pinos de alta tensão (*HV - High Voltage Pins*), além das unidades já citadas acima através da Figura 23.

3.2.2.1 Device Specific Adapter (DSA)

Para realização da interface entre o componente a ser testado e o testador utiliza-se um dispositivo denominado DSA (*Device Specific Adapter*). É nele que estão os soquetes de teste para conexão elétrica dos DUTs (*Device Under Test*). O DSA possui a quantidade de pinos e tamanho de acordo com o encapsulamento e quantidade de esferas do componente a ser

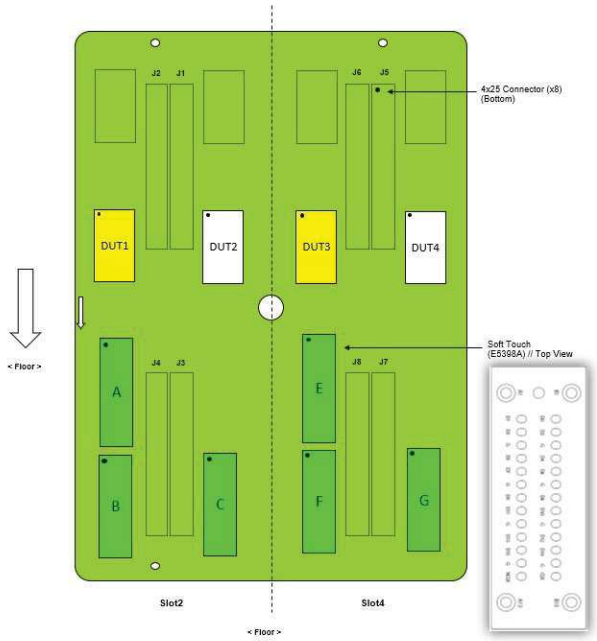
Figura 24 – Diagrama de blocos de um único site em uma *Site Assembly Board*.



Fonte: Nextest/Teradyne (2014a)

testado. O dispositivo DSA destinado para este uso neste projeto é, por sua vez, específico para componentes DDR4 78FBGA x8 ou DDR4 96FBGA x16, contendo soquetes destinados a estes modelos de memórias, bem como as conexões elétricas necessárias (trilhas e ligações com a *Site Assembly Board*). O DSA utilizado para esta aplicação fora projetado pela equipe do Laboratório de Testes do itt Chip (Unisinos) e fabricado pela empresa sul-coreana TSE. Na Figura 25 tem-se o layout da PCB do DSA utilizado na execução dos testes no testador Magnum V.

Figura 25 – *Layout* da PCB do DSA para memórias DDR4 x8/x16.



Fonte: Nextest/Teradyne (2014a)

O mesmo é dotado de duas posições de DUTs, sendo os DUTs 1 e 2 utilizados para componentes DDR4 x16 e os DUTs 3 e 4 para componentes DDR4 x8. Como o foco deste projeto são componentes DDR4 x16, somente serão utilizadas as posições 1 e 2. Para cada um dos DUTs 1 e 2 existem canais específicos do testador que estão conectados. Para isto, é fornecido pelo fabricante do DSA a relação entre o pino do componente (*Ball Name*), conforme mostrado na Figura 19 e na Tabela 9, o número da posição no soquete do DUT (*Ball Number*) e os canais utilizados no testador (*Channel*). Estas informações são utilizadas na configuração do *Pin Assignment* durante a elaboração do programa de testes, onde deve-se relacionar os pinos físicos do componentes com os respectivos canais utilizados no testador para conexão com os mesmos. No Apêndice B têm-se a relação dos pinos para o *Pin Assingment* do DSA utilizado para componentes DDR4 x16 com 96 pinos.

3.2.2.2 Time Domain Reflectometry (TDR)

Outra importante verificação a ser executada é a calibração TDR. Ela irá verificar os atrasos de tempo (*delays*) existente entre os níveis do testador e o DSA, e assim se auto-calibrar para a realização do teste baseado nos atrasos de tempo mensurados. Todo DSA apresenta atrasos, devido as características das propriedades físicas existentes (placa de circuito impresso, componentes eletrônicos, conectores, soquetes, etc). Sempre que um novo DSA é conectado ao testador se faz necessária a realização da calibração em nível de TDR, a fim de minimizar os erros durante a execução do teste. Estes atrasos irão determinar o quão preciso será o teste e a influência que poderá ocorrer para testes com frequências elevadas. Ou seja, quanto maior os

atrasos, maiores serão os danos para a execução de testes com altas frequências. Abaixo temos um trecho da calibração TDR realização para o DSA DDR4 x8/x16 utilizado neste projeto.

Figura 26 – Exemplo dos resultados da calibração TDR para o DSA DDR4 x8/x16.

```
Measure TDR + Save to NVM
Measuring TDR, please wait...
Programming TDR data into DUT board NVMs...
New NVM TDR Date: 09/09/2019 10:45,  Pin Qty: 128
Display TDR from NVM
Reading TDR data from DUT board NVMs...
TDR pin a_1 = Tester 3.4679 ns + Ext Portable  1.5484 ns = Total 5.0163 ns
TDR pin b_1 = Tester 3.6326 ns + Ext Portable  1.5321 ns = Total 5.1647 ns
TDR pin a_2 = Tester 3.4524 ns + Ext Portable  0.6238 ns = Total 4.0762 ns
TDR pin b_2 = Tester 3.6125 ns + Ext Portable  0.6131 ns = Total 4.2256 ns
TDR pin a_3 = Tester 3.3135 ns + Ext Portable  1.5317 ns = Total 4.8452 ns
```

Fonte: Fonte: Testador ATE Magnum V.

Neste exemplo, pode-se ver o atraso em nanosegundos mensurado no testador somado ao atraso externo, neste caso o DSA, totalizando um valor de atraso em nanosegundos para cada um dos pinos dos DUTs. Estes valores são medidos para todos os pinos do *Pin Assignment* (descritos no Apêndice B) e podem ser armazenados na memória não-volátil do próprio DSA (NVM) ou no disco (testador). Usualmente se utiliza a NVM, mantendo a informação dentro da placa do DSA.

3.2.2.3 Programação do testador Magnum V

Conforme mencionado anteriormente, a programação do ATE Magnum V é realizada em linguagem C++ utilizando o software *Microsoft Visual Studio* e toda operação e validação do programa de teste é realizada através do UI (*User Interface*) da Nextest (fabricante do testador). O programa desenvolvido no ambiente do *Microsoft Visual Studio* é escrito na linguagem C++ no ambiente global e na linguagem própria do testador *APG Pattern Language* no que se refere a escrita dos *patterns* que serão executados pelo testador. Basicamente, um programa de teste desenvolvido em um testador *Magnum V* possui uma estrutura que pode ser dividida quatro principais conjuntos e seus respectivos blocos internos, conforme ilustrado na Figura 27:

- Configurações globais (*Global Settings*): onde estão definidas as configurações básicas, as variáveis de uso global, configurações de comunicação com os sites do testador e as configurações de uso da APG. Contém os blocos do programa de teste: *Site_Begin_Block*, *Host_Begin_Block*, *TDR_Block* e demais blocos de configurações.

- Pinos, sequência de teste e *bins*: onde são configurados o *Pin Assignment* (informações no Apêndice B), a lista de pinos a serem utilizados, a sequencia de blocos de teste a ser executada e a definição dos *bins* (classificação dos resultados dos testes). Contém os blocos do programa de teste: *Pin_Assignment*, *Pin_List* e *Sequence_Table*.

- Blocos de Teste (*Test Blocks*): é onde ocorre a programação dos blocos de teste. Possui uma pré-configuração do bloco de teste, o bloco de teste com os setups de níveis de tensão, tempos, testes funcionais ou paramétricos e após o teste a classificação com o resultado de cada bloco. Neste bloco são executados os testes funcionais que chamam rotinas executadas em linguagem *APG Pattern Language* no conjunto dos testes funcionais. Contém os blocos do programa de teste: *Before_Setting_Block*, *Test_Block* e *After_Setting_Block*.

- Testes Funcionais (*Functional Tests*): onde se encontram as configurações dos níveis de tensão de alimentação, níveis de tensão dos I/Os, níveis de corrente, configuração dos tempos (tempos de ciclo e *strobes*), a assinatura das funções de cada um dos pinos (*Pin Scramble*) e as rotinas a serem executadas pela APG escritas no arquivo de *patterns* (*pattern.pat*). Contém os blocos do programa de teste: *DC_Level_Set*, *Timing_Set*, *Pin_Scramble* e *APG_Pattern*.

A programação os algoritmos em *APG Pattern Language* é baseada nas funções descritas no manual de programação do equipamento ((Nextest/Teradyne, 2014a)). Nos blocos de teste (*Test Blocks*) são descritas as rotinas que contém as execuções dos algoritmos executados no nível de *pattern*, ou seja, na ULA do testador. Todos os algoritmos devem ser descritos no nível de *pattern*, utilizando a linguagem própria do testador. APG (*Algoritm Pattern Generator*) é basicamente uma aplicação específica do processador necessária para gerar a sequência programável de endereços, dados e controle de sinais para funcionalidade de teste de um CI. A APG é programada utilizando um hardware específico em linguagem de baixo nível (*APG microInstructions*). O algoritmo codificado utilizando *APG microInstruction* é chamado de *pattern*. Os principais itens contidos em uma APG são:

- 3 geradores de endereços (X, Y, Z): 24 bits;

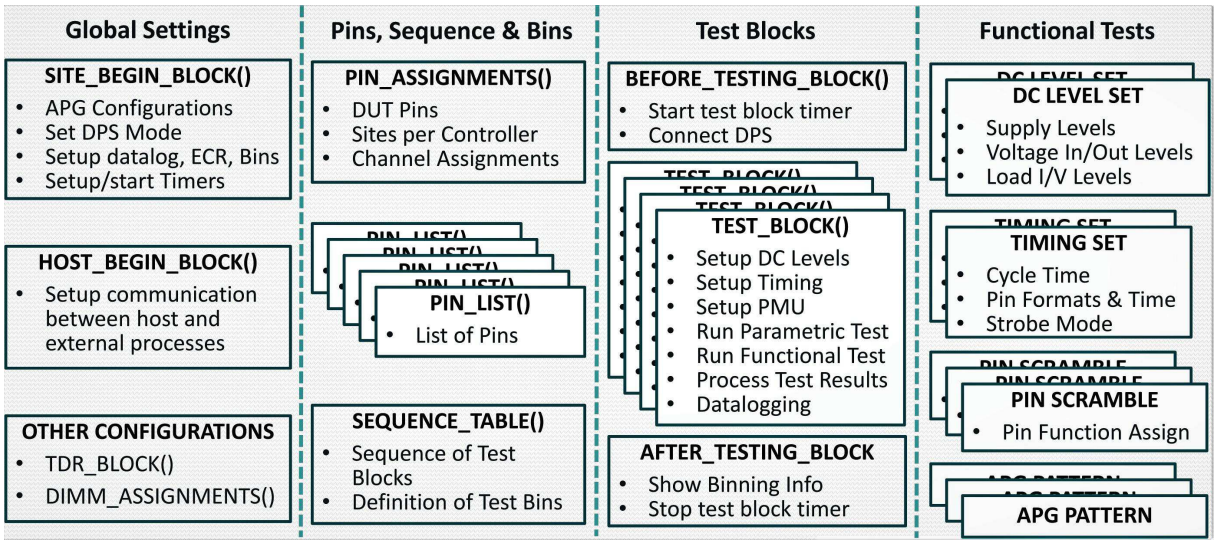
- Gerador de dados: 36 bits;

- Seletor de chips: 18 bits;

- Gerador de controle de DUT: 40 bits;

- Unidade de controle de execução: uRAM, MAR Engine, contadores, temporizadores.

Figura 27 – Estrutura de desenvolvimento do programa de testes.



Fonte: Moraes e Kondo (2017).

A configuração do APG é realizada no ambiente *site_begin.cpp*, onde são parametrizados os geradores de endereço (*Address Generators*, geradores de dados (*Data Generators* e os *Chip Selects*. É muito importante configurar os endereços para que atendam toda a densidade da memória (número de bits endereçáveis para endereços de linhas e colunas da memória).

Os geradores de endereço são denominados XALU (x), YALU (y) e ZALU (z) e são utilizados para realização das operações de endereços no nível de *pattern*. Já o gerador de dados DATGEN é utilizado para operações numéricas. O comando *pinfunc* é onde são definidos os padrão de tempos e o *Pin Scramble* a serem utilizado na instrução. A função *chips* define o seletor de chip a ser utilizado. Os seletores de chips são tradicionalmente usados quando realiza-se teste de componentes de memórias, para controle dos DUTs selecionados, funções de habilitar escrita, definição de pinos de escrita/leitura, etc.

Uma função em nível de *pattern* necessita basicamente de algumas condições iniciais, as instruções da função e um comando de término. A seguir na Figura 28 têm-se um modelo de uma função em nível de *pattern*:

Figura 28 – Exemplo de modelo de uma função escrita em nível de *pattern*.

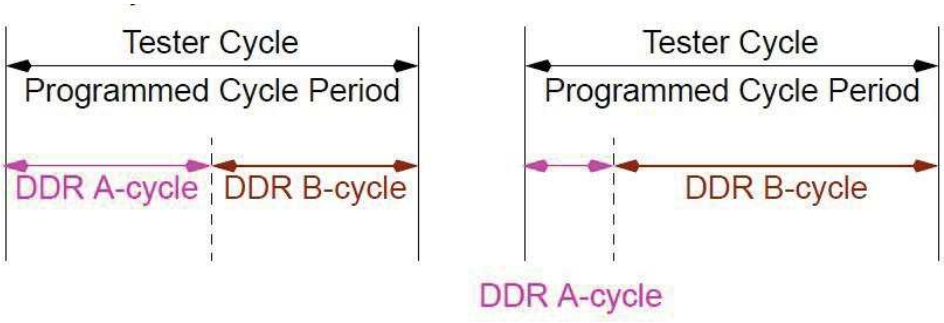
```
PATTERN( \textit{nome_do_pattern} )
    @{
       condições iniciais
       configuração dos contadores
    @}
    %instrução_nr1
    ...
    ...
    %instrução_nr2
    ...
    ...
    %   mar     done (fim do pattern)
```

Fonte: Testador ATE Magnum V.

3.2.2.4 Função DDR Mode

A fim de executar os testes utilizando o mesmo padrão de funcionamento de uma memória SDRAM DDR4, utiliza-se o modo DDR em parte do código desenvolvido no ATE. Isto faz com que tenha-se dois eventos de tempo para cada tempo de ciclo do testador, levando-se em consideração borda de subida e a borda de descida do *clock*. Para tanto, é necessária a configuração do *Pin Scramble* no modo DDR, os *timings* em modo DDR, além da descrição da função do *pattern*, em que se deve levar em consideração o modo DDR. Para configuração do ciclo em DDR, deve-se definir dentro do período total de um tempo de ciclo do testador, quanto irá corresponder ao *DDR-A-cycle* e quanto irá corresponder ao *DDR-B-cycle*, conforme mostra-se na Figura 29.

Figura 29 – Exemplos de configurações do tempo de ciclo para o modo DDR.

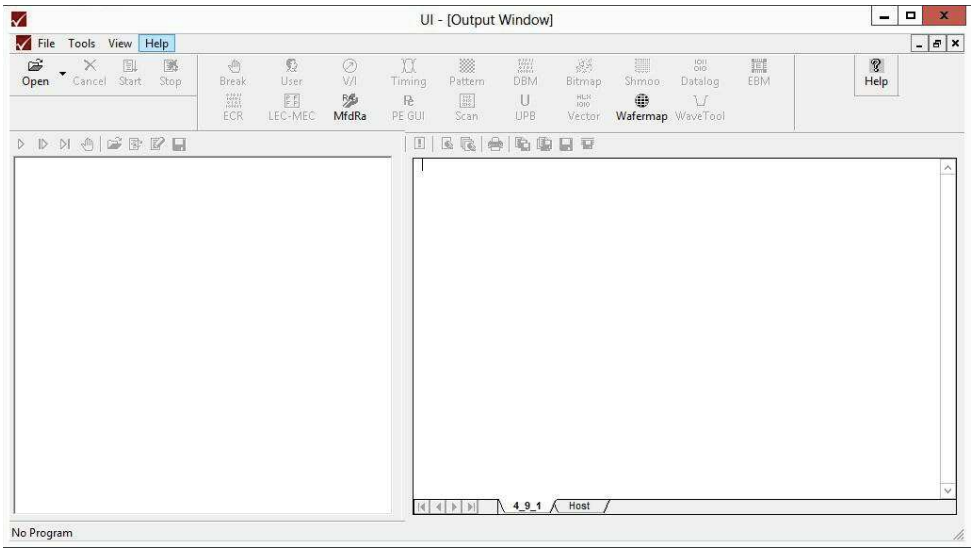


Fonte:Nextest/Teradyne (2014a).

3.2.2.5 Interface do ATE

Conforme descrito na seção 3.2.2.3, para a execução e validação do programa de testes, utiliza-se a interface de usuário *Nextest UI*, disponível no próprio ATE (Figura 30).

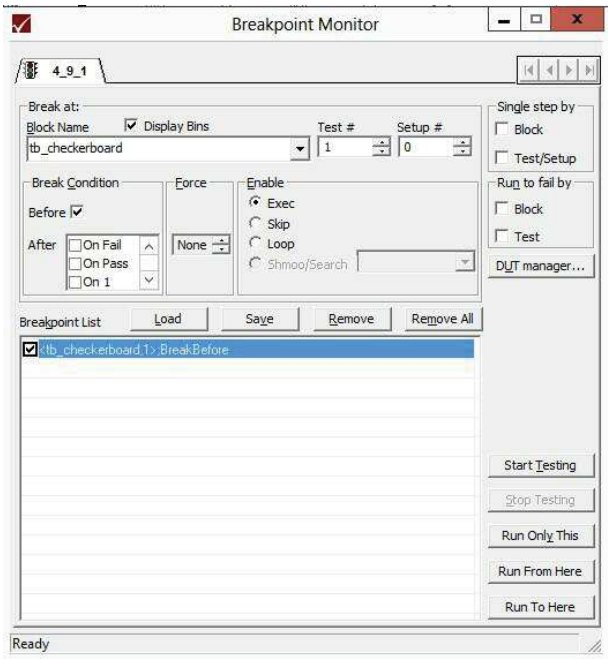
Figura 30 – Tela da interface do usuário (UI) no ATE Magnum V.



Fonte: Nextest/Teradyne (2014a).

Nesta ferramenta é possível verificar se existe alguma falha na execução do programa, bem como analisar os sinais que estão sendo originados pelo programa e as respostas da memória sob testes. As principais aplicações utilizadas dentro do UI são o *Wave Tool*, onde é possível visualizar os sinais gerados pelo programa de testes, analisar os sinais de cada um dos pinos e em determinado tempo de ciclo dentro da execução do teste. Outra importante ferramenta é o o *Break Point*, onde pode-se definir pontos específicos para que o teste seja pausado ou até mesmo definir pausas antes ou depois de falhas, em rotinas específicas dentro do teste, etc. Na Figura 31, têm-se um exemplo de um *break point* configurado para a primeira subrotina de teste dentro da execução do algoritmo *Checkerboard* e antes do início da execução da mesma. Outra opção também é o uso da ECR (*Error Catch RAM*), que são posições na memória do testador onde são armazenadas as informações dos endereços da memória sob teste que apresentaram erros. Assim, é possível analisar logicamente onde ocorreram estes erros, acessando a ECR pelo UI.

Figura 31 – Configuração de um *break point* no UI do ATE Magnum V.



Fonte: Nextest/Teradyne (2014a).

3.2.3 Testador TCE3200LP

Outro testador que foi utilizado no trabalho foi o Turbocats TCE3200LP da Triad. Este modelo é um testador de bancada e sua principal função no desenvolvimento deste trabalho foia validação do programa de teste desenvolvido no ATE Magnum V. A escolha deste modelo de testador deu-se por sem um testador de propriedade do Itt Chip (Unisinos) e por atender as especificações necessárias para teste de componentes SDRAM DDR4 x16.

Figura 32 – Testador TCE3200LP.

Fonte: (Turbocats, 2017).

Ao contrário de um ATE, no testador TCE3200LP não é possível o desenvolvimento de um programa de teste, somente selecionar os algoritmos e parâmetros a serem utilizados para teste. Com base nisto, foram relacionados os algoritmos (*patterns*) disponíveis no TCE3200LP a fim de serem os mesmos utilizados para desenvolvimento no programa de teste no ATE Magnum V, para que seja possível realizar uma posterior comparação entre os testes.

Tabela 11 – Algoritmos disponíveis no testador TCE3200LP.

Algoritmo	**Descrição**
01_AutoRefresh_t	Este algoritmo é usado para detectar as falhas de retenção da SDRAM durante o ciclo de auto-refresh: falhas de *sleeping sickness* e SAF *refresh line*.
02_SelfRefresh_t	Este algoritmo é usado para detectar as falhas de retenção da SDRAM durante o ciclo de auto-refresh: falhas de *sleeping sickness* e SAF *refresh line*.
06_CheckerBoard_t	Este é um teste simples para detectar curtos entre células de memória adjacentes, sob a condição que o decodificador de endereços funcione corretamente. 0s e 1s são escritos na memória que é dividida em dois grupos, formando o padrão de uma tabuleiro de xadrez.
07_Marching_t	Este algoritmo envolve a alteração de nível lógico de um endereço de memória e mantém este nível lógico enquanto se realiza a alteração de nível do endereço seguinte.
08_MarchA_t	Este teste é para detectar CFids, como também AF, SAFs, TFs e certos CFins.
14_MarchLA_t	Este algoritmo consiste em múltiplos elementos (R-W-W-W-R), visando detectar falhas dinâmicas.

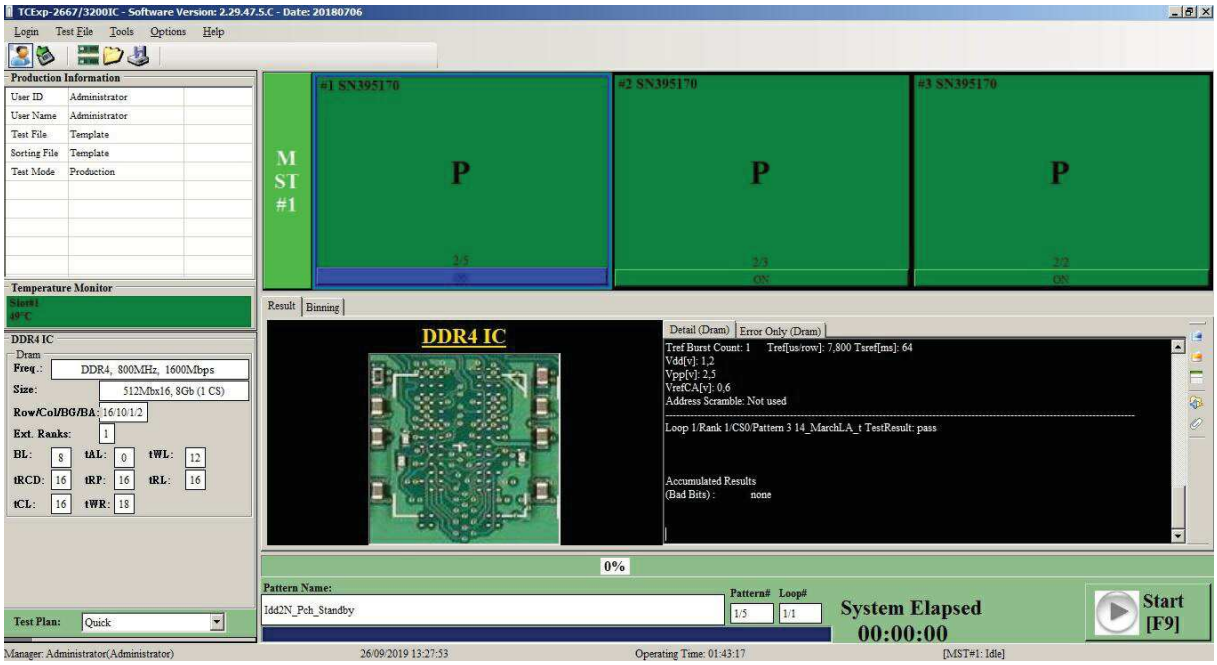
Fonte: Turbocats (2017).

Semelhante a estrutura de um ATE, é necessária a configuração e aquisição do soquete adequado ao dispositivo sob teste (DUT). Neste caso, utiliza-se soquetes DDR4 x16, que limitam o paralelismo do teste para 3 componentes simultaneamente (Figura 34).

Além disto, necessita-se da configuração de alguns parâmetros a fim de especificar o componente que será testado. Como pode-se ver na tela principal do software de teste do TCE3200LP (Figura 33), têm-se na parte superior os 3 soquetes com o status de teste atual, ao centro o resultado detalhado de cada teste, na parte inferior têm-se o progresso do teste em andamento e no lado esquerda da tela as configurações pré-definidas para o componente a ser testado (tempos, latências, frequência de operação, etc).

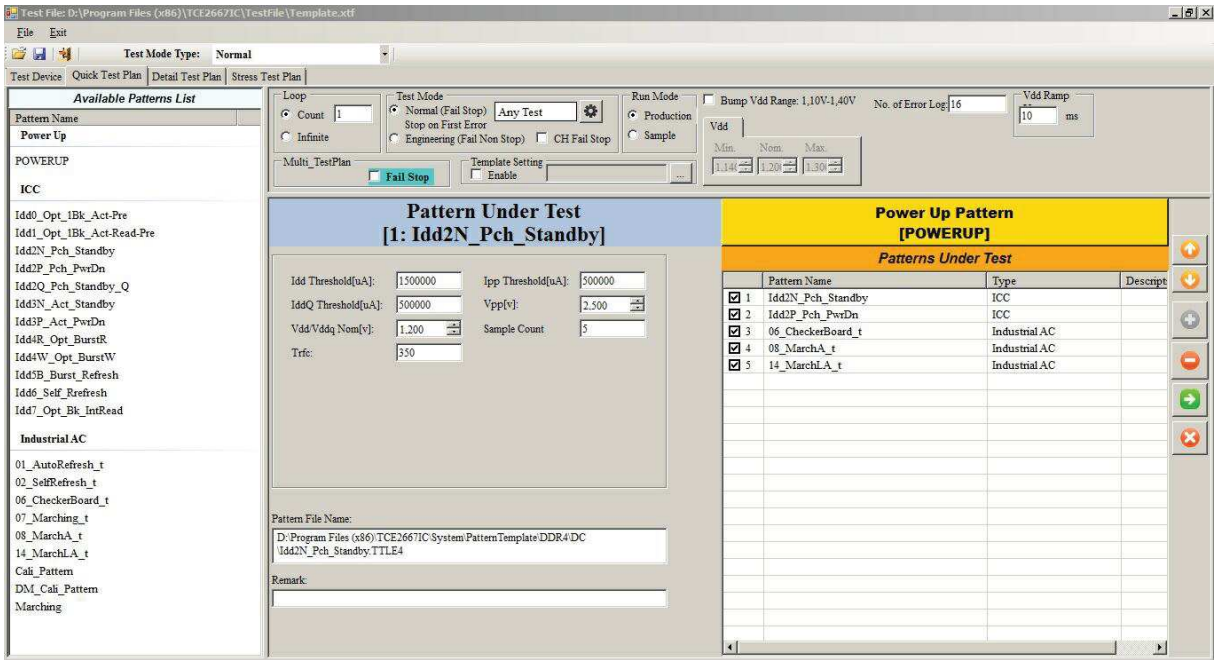
Para configuração dos *patterns* de teste a serem utilizados, utiliza-se a tela de *Edit Test Setup* e na opção *Quick Setup* é possível definir a sequência de teste a ser realizada e salvar este modelo para utilizar em momentos posteriores (Figura . Ao lado esquerdo da tela estão os algoritmos disponíveis para utilização. No centro estão as configurações atuais do algoritmo selecionado. Já no lado direito está a sequência de teste criada, que pode ser ajustada conforme necessidade do usuário.

Figura 33 – Tela principal de operação do testador TCE3200LP.



Fonte: Software Turbocats TCExp-2667/3200IC.

Figura 34 – Tela de setup de teste do testador TCE3200LP.



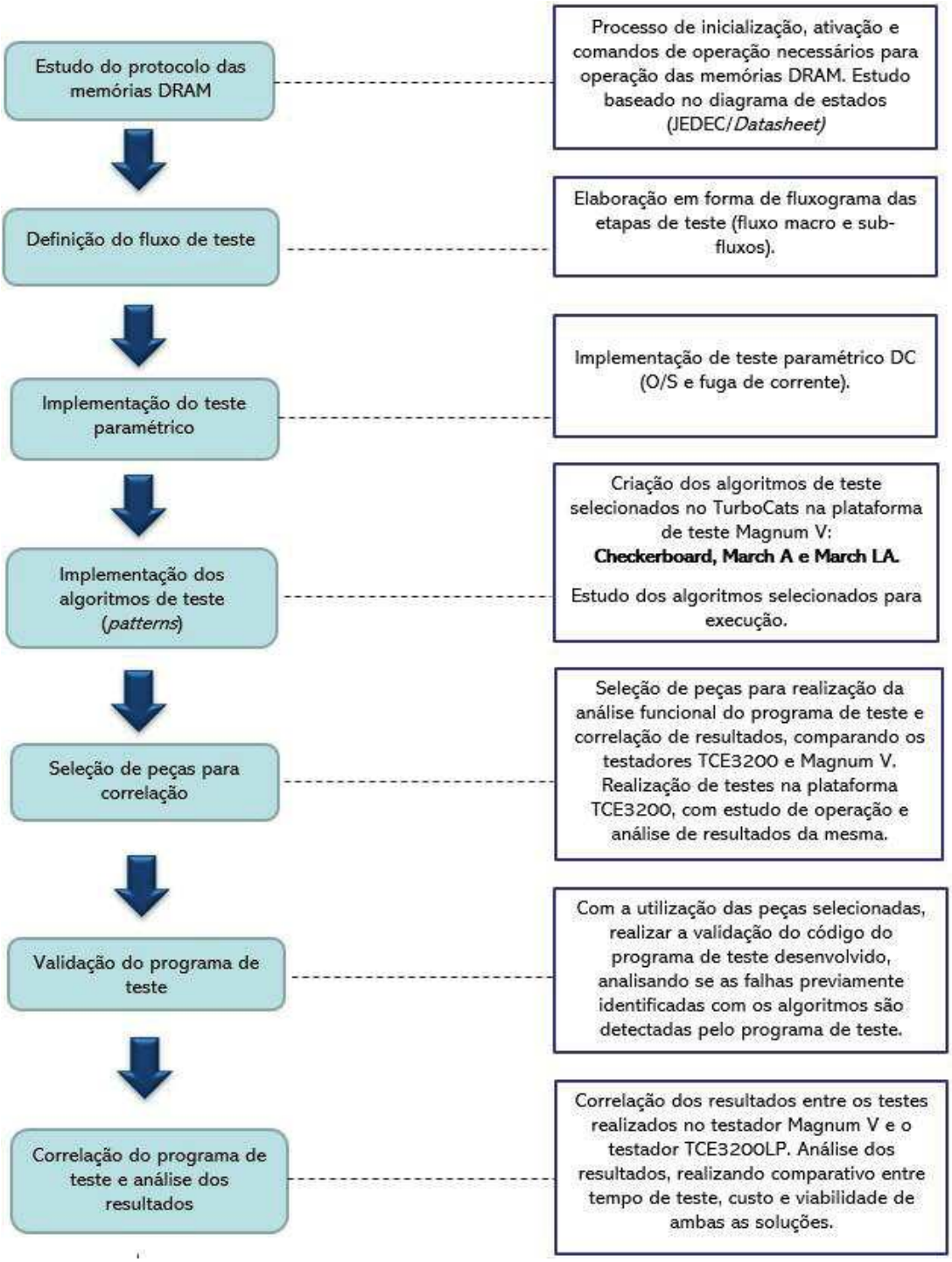
Fonte: Software Turbocats TCExp-2667/3200IC.

3.3 Fluxograma de desenvolvimento

O desenvolvimento do projeto é composto inicialmente pelas etapas de pesquisas e entendimento do protocolo das memórias SDRAM DDR4, bem como o conhecimento necessário

para desenvolvimento do programa de testes no ATE Magnum V. Este auxilia no embasamento para as demais etapas de execução, onde ocorre a definição do fluxo de teste e implementação efetiva do programa de testes no ATE. A fim de estruturar o desenvolvimento do projeto e definir um cronograma de execução, desenvolveu-se um fluxograma das atividades a serem realizadas, conforme Figura 35.

Figura 35 – Fluxograma de desenvolvimento do projeto.



Fonte: Elaborado pelo autor.

Com a definição do fluxograma das atividades, deu-se início a execução das mesmas, conforme descrito nos capítulos à seguir.

3.4 Definição dos algoritmos

Outra definição importante foram os algoritmos de testes implementados no programa de teste. Como a comparação dos resultados ocorreu utilizando o testador TCE3200LP, fez-se necessário que os algoritmos utilizados no programa desenvolvido também estivessem acessíveis no testador TCE3200LP, para possibilitar a análise de aderência dos resultados entre os dois testes. Com base na pesquisa realizada, pode-se relacionar os algoritmos existentes no TCE3200LP (ver Tabela 11). Além disso, foram estudados os níveis de cobertura de falhas para diferentes algoritmos e modelos de falhas (ver Tabelas 6 e 7).

Baseado nestes estudos, definiu-se que os algoritmos implementados no programa de testes foram: ***Checkerboard***, ***March A*** e ***March LA***. A escolha deu-se pela diferença entre falhas a serem detectadas por cada um dos algoritmos, uma vez que o *Checkerboard* visa detectar as chamadas falhas *single-cell* e alguns tipos de *NPSF*, o algoritmo *March A* detecta falhas funcionais estáticas e falhas de endereçamento e o algoritmo *March LA* que objetiva a detecção de falhas dinâmicas. Outros algoritmos complementares também puderam ser implementados a fim de melhorar a cobertura de falhas, entretanto não puderam ser validados com o TCE3200LP. Os algoritmos selecionados possuem um padrão de escritas e leituras que tiveram que ser obedecidos a fim de atender a estrutura do mesmo. Nos capítulos 3.8 e 3.9 têm-se os fluxogramas de testes e as sequencias de escritas/leituras de cada um dos algoritmos implementados.

3.5 Fluxograma e estrutura do programa de testes

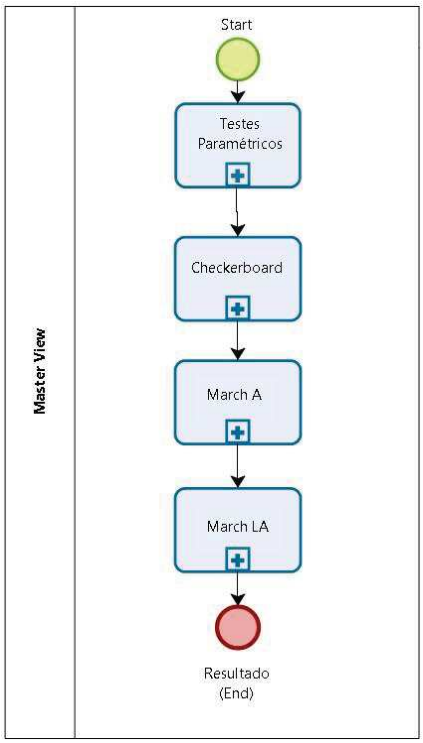
A fim de garantir a exequibilidade do projeto, criou-se previamente um modelo de fluxograma do programa de teste e definiu-se o método de estruturação do programa. Com isto, têm-se definidas as diretrizes a serem tomadas para as demais etapas do projeto.

3.5.1 Fluxograma do programa de testes

O fluxograma de teste contém as etapas criadas dentro do programa. O fluxograma foiconstituído de um fluxograma macro (Figura 36), com as etapas principais para a elaboração do programa de testes, e os seus subprocessos, com os algoritmos e detalhamento das rotinas contidas no fluxo macro.

Na Figura 36 pode-se identificar as etapas definidas em um plano macro para o desenvolvimento do programa de testes. Primeiramente, têm-se o desenvolvimento da rotina de *Testes Paramétricos*, a fim de detectar o estado elétrico do componente (teste DC) e se está parametricamente dentro das especificações nominais. O fluxograma do *Teste Paramétrico* e os parâmetros de teste estão detalhados na seção 2.3.1.1.

Figura 36 – Fluxograma de desenvolvimento do programa de testes.



Fonte: Elaborado pelo autor.

Após, ocorre a execução dos algoritmos de testes pré-estabelecidos no capítulo 3.4: *Checkerboard*, *March A* e *March LA*. Caso aprovado nos quatro testes realizados, o componente segue para próxima etapa de teste, no qual é realizado o algoritmo de teste *Checkerboard*. No subprocesso do algoritmo *Checkerboard*, é necessário realizar previamente a inicialização da memória antes da execução das escritas e leituras dos padrões de teste. Na inicialização trata-se de realizar os comandos necessários para deixar a memória no estado denominado *Idle* (ver Diagrama de Estados Simplificado, Figura 20), conforme descrito detalhadamente na seção 3.6. Após a memória inicializada, ocorre a execução do algoritmo *Checkerboard*, que está descrito na seção 3.8 deste relatório. Após o *Checkerboard*, executam-se os algoritmos *March A* e *March LA*, respectivamente. Ambos estão descritos no capítulo 3.9. Por fim, apresenta-se o resultado final do teste, com o respectivo *bin* de aprovação ou rejeição.

3.5.2 Estrutura do programa de testes

A estruturação do programa de testes ocorreu na definição do método utilizado para a realização das principais rotinas e as estratégias empregadas na escrita do código do programa. Pela necessidade da utilização do *DDR Mode* na programação do ATE Magnum V, utilizou-se dois modelos de tempos (*Timing Sets*): um em operação *Single Data Rate* (SDR) e outro em operação *Double Data Rate* (DDR). A partir disto, teve-se a necessidade da criação de *Pin Scrambles* específicos para os diferente modelos de tempos. A execução de um comando em

uma função no modo *pattern* do testador contém, na maioria das vezes, as seguintes instruções:

1. **Chip Select / Dutctrl:** são as duas funções utilizadas pelo APG para controle de sinais. É onde define-se o modelo de sinal a ser utilizado na instrução, com sua devida polaridade. Utiliza-se geralmente para os pinos que não correspondem a endereços ou entradas/saídas;
2. **Pinfunc:** com este comando são descritos o modelo de tempos (*timing_set*) e do *pin_scramble* a ser utilizado. É com ele que define-se os estados de driver (*on/off*) dos pinos através dos dados do *pin_scramble*;
3. **Datgen / (X/Y/Z)ALU** : é onde têm-se as informações de dados (*datgen*) ou endereços (*XALU, YALU, ZALU*) utilizados nas operações de escrita/leitura nas respectivas linhas/colunas;
4. **Count:** comando para descrever contadores que serão utilizados nas instrução em *pattern*.
5. **Udata:** utiliza-se para atribuir uma constante numérica para quase todas as variáveis;
6. **Mar:** é utilizado para programação do controlador da APG. Controla o fluxo e derivações do APG e gera sinais para os PE (*Pin Electronics*) dos DUTs.

Para melhor entendimento, na Figura 37 têm-se uma parte do código utilizado no programa de teste, na instrução do comando escrita (*write command*) no modo DDR. Nesta instrução, atribui-se primeiramente ao registrador XALU o valor definido em *xmain*. Logo após, atribui-se a YALU o valor definido em *ybase* somado ao valor de *dymain*. Atribui-se a *dutctrl* um valor que delimita a posição inicial do registrador. Com o omando *pinfunc* é atribuído o *timing_set 2* (*tset2*) e o *pin_scramble 6* (*ps6*). O comando *mar inc* libera a APG para a próxima instrução.

Figura 37 – Parte do código de um comando de escrita (*WR command*).

```
%WR_command:
xalu xmain,hold,dxmain,oxmain
yalu ymain,ybase,add,dymain,oymain
dutctrl 0x0001
pinfunc tset2, ps6
mar inc
```

Fonte: Elaborado pelo autor.

Logo abaixo, na Figura 38, têm-se o exemplo da configuração do registrador de modo MR5, realizado no modo SDR (*single data rate*). O comando *datgen* atribui o valor de *udatadr* e *sudata* no destino *dmain*. Habilita-se o *chip_select* já definidos *cs1pt* e *cs2pt*. Através do *pinfunc* atribui-se o *timing_set tset1* e o *ps3* (*Pin Scramble 3*). A mesma metodologia é utilizada para os demais *mode registers* da memória.

Figura 38 – Parte do código de um comando de configuração do registrador de modo MR5 (*Mode Register 5*).

```
%MR5_setting:
datgen udatadr, sudata, dmain, mainmain
udata 0x50400
chips cs1pt, cs2pt
pinfunc tset1, ps3
mar inc
```

Fonte: Elaborado pelo autor.

Pensando-se na execução dos comandos, que ocorrem a partir de níveis lógicos (0 e 1) em simultâneos pinos do componente, definira-se o método de programação utilizando os *Pin Scrambles*. Em resumo, cada *Pin Scramble* possui os níveis lógicos corretos nos determinados pinos da memória a fim de executar a sequencia necessária para a execução do comando (conforme descrito na *Tabela Verdade* da Figura 21. Para exemplificar, têm-se que para a execução do comando MRS (*Mode Register Set*) é necessária a condição lógica conforme a Figura 39, abaixo:

Figura 39 – Tabela verdade para execução do comando MRS (*Mode Register Set*).

Function	Abbreviation	CKE		CS_n	ACT_n	RAS_n /A16	CAS_n /A15	WE_n/ A14	BG0-BG1	BA0-BA1	C2-C0	A12/ BC_n	A17, A13, A11	A10/ AP	A0-A9
		Previous Cycle	Current Cycle												
Mode Register Set	MRS	H	H	L	H	L	L	L	BG	BA	V	OP Code			

Fonte: JEDEC Standard (2012).

Sabendo-se disto, cria-se então, um *Pin Scramble* que satisfaça as mesmas condições lógicas do comando. No código este é o *Pin Scramble* de número 3 (PS3), conforme exemplo contido no Apêndice C.

Além dos *Pin Scrambles*, outra parametrização importante está relacionada a organização da memória, que deve ser endereçada através dos registradores de endereços do testador. Como utiliza-se neste estudo uma memória DDR4 512Mbx16Mb, precisa-se de uma configuração que atenda 512Mb linhas x 16Mb colunas. Para o qual, têm-se o equivalente $2^{16} = 65.536$ bits em linhas e $2^{10} = 1024$ bits em colunas. Como uma memória SDRAM DDR4 x16 possui internamente 2 bancos de memórias que são endereçados através de BG0 (*Bank Group 0*) e de BA0/BA1 (*Bank Address 0 e 1*). Sendo assim, utiliza-se os endereços em X (XALU) do APG para configuração da organização de linhas, endereços de bancos e grupos de bancos e os endereços em Y (YALU) do APG para endereçamento das colunas. Na Tabela 12 está a organização em bits dos registradores de endereço da APG.

Tabela 12 – Configuração dos registradores de endereços na APG.

Endereço APG	**Endereço DRAM**	**Bits**	**Total Bits APG**
XALU	Linhas	16	19 (2^{19})
	BA0	1	
	BA1	1	
	BG0	1	
YALU	Colunas	10	10 (2^{10})

Fonte: Elaborado pelo autor.

Como pode-se observar, tem-se uma varredura de endereços em linhas (x) e colunas (y), que é na ordem de $[2^{16}(linhas) * 2^{10}(colunas)]x2(bancos)$, que totalizam 8Gb de células de memórias. Por isto, no XALU tem-se o registrador de linhas somado aos dois BA e um BG, totalizando 2^{19} posições de endereços, ou seja, 19 bits de endereços. Já no registrador YALU tem-se o registrador de colunas, contendo 2^{10} posições de endereços, ou seja, 10 bits de endereços.

3.6 Inicialização e definição dos tempos (*timings*)

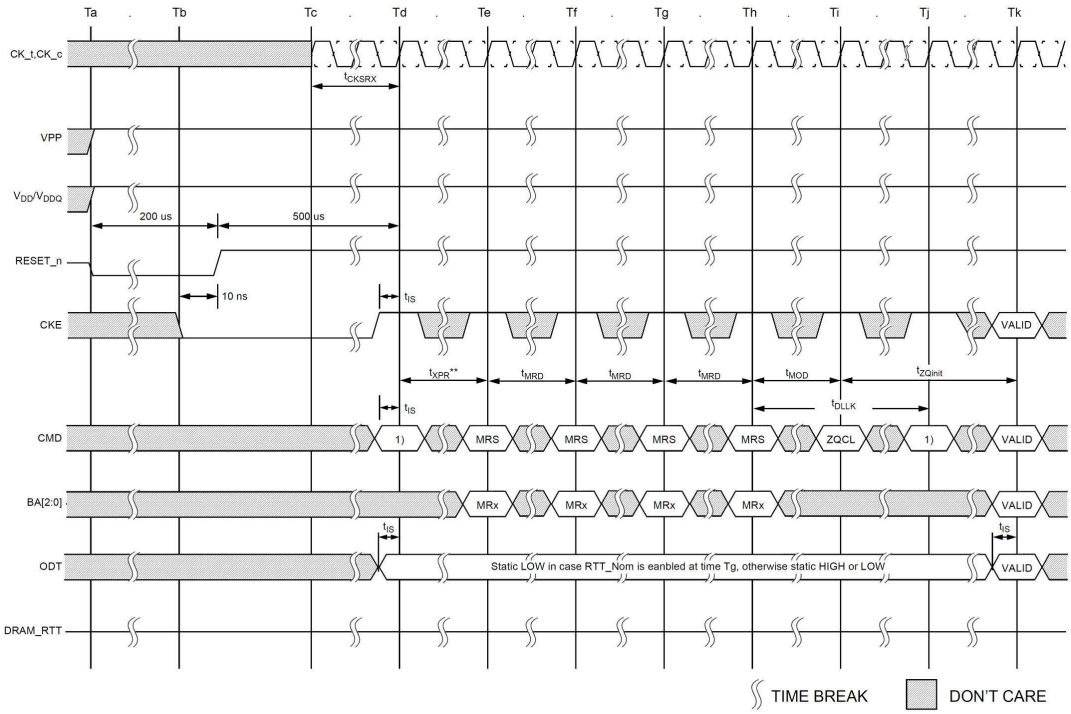
Para execução dos testes na memória DRAM é necessário garantir que a mesma esteja inicializada. Em outras palavras, é preciso executar alguns procedimentos padrões para que tenhamos a memória pronta para executar os comandos de escrita e leitura, que serão realizados nos testes funcionais (algoritmos). O único teste que não exige que a memória esteja inicializada é o teste paramétrico, pois não ocorre nenhuma operação funcional (escrita, leitura, comandos de *refresh*, ciclos de *clock*, etc). Outro ponto de importante definido foram os tempos (*timings*), os quais são utilizados tanto para o procedimento de inicialização, como para as operações de escrita/leitura, *refresh* e demais comandos a serem executados. Para definição, tanto do procedimento de inicialização, como dos tempos, utilizou-se o *datasheet* do componente (HYNIX, 2017) e a norma JEDEC para memórias SDRAM DDR4 (JEDEC Standard, 2012)..

3.6.1 Inicialização da memória SDRAM DDR4

A inicialização de uma memória depende basicamente do acesso ao estado *Idle* da mesma, utilizando os comandos necessários para tal. Para a execução dos algoritmos *Checkerboard*, *March A* e *March LA* é necessária a execução deste procedimento de inicialização. Conforme o Diagrama de Estados Simplificado da memória SDRAM DDR4 (Figura 20), evidenciado no referencial teórico desta dissertação, tem-se as etapas e comandos necessários para a inicialização, desde o momento da energização (*Power On*) até o estado *Idle* desejado. Para cada uma das etapas algumas sequências devem ser obedecidas, com a execução de alguns comandos em específico e respeitando tempos pré-determinados. Recomenda-se que no momento da inicialização, também a execução do comando RESET, a fim de garantir o correto funcionamento e para limpar

possíveis valores atribuídos aos *Mode Registers*. Na Figura 40 tem-se a sequência de comandos de inicialização e *reset* e os respectivos tempos que devem ser utilizados.

Figura 40 – Sequência para inicialização e *reset* após energização de uma memória SDRAM DDR4.



Fonte: JEDEC Standard (2012).

Logo após a inicialização é onde que realiza-se a configuração dos chamados *Mode Registers*, previamente descritos no capítulo 3.2.1 nas tabelas de número **??** ao número **??**. Para configuração dos registradores utilizam-se os códigos em hexadecimal definidos nas tabelas em linguagem de modo *pattern*, realizando o procedimento para todos os sete registradores da memória. Na Figura 41 exemplifica-se o código utilizado para a configuração de um dos registradores de modo, o MR0. Conforme a Tabela **??**, tem-se o código hexa 0x0304 e realiza-se os comandos conforme o exemplo a seguir:

Figura 41 – Exemplo de código para configuração do registrador de modo MR0.

```
%MR0_setting:
datgen udatadr, sudata, dmain, mainmain
udata 0x00304
chips cs1pt, cs2pt
pinfunc tset1, ps3
mar inc
```

Fonte: Elaborado pelo autor.

3.6.2 Tempos (*timings*)

Para obedecer os tempos definidos através da norma JEDEC e pelo fabricante da memória, definiram-se todos estes *timings* previamente utilizando-se variáveis globais na biblioteca "*protos.h*", o que favorece caso modificações sejam necessárias, assim visando correções de erros ou até mesmo alteração da frequência de teste (tempo de ciclo de operação). Inicialmente, com base na limitação do ATE em 800MHz, e opção do uso do modo DDR, optou-se pela operação em 1600Mbps, que é a menor frequência nominal possível para uma memória SDRAM DDR4. Para tal, definiu-se os tempos com uma base de tempo de ciclo de 1,25ns, que corresponde a uma frequência de 800MHz. Contudo, após a realização de algumas validações utilizando a ferramenta *Wave Tool* do ATE Magnum V, verificou-se uma distorção e um atraso (*delay*) significativo nos sinais. Isto provavelmente está relacionado com características construtivas do DSA que apresentam níveis razoavelmente elevados de atrasos já durante a execução da calibração TDR. Portanto, visando mitigar estes atrasos e distorções nas formas de onda, atuou-se aumentando o tempo do ciclo de *clock* do teste. Portanto, evidenciou-se ao elevar gradativamente o tempo de ciclo, que a partir de 2,5ns teve-se um resultado melhor nos sinais e os atrasos não apresentam mais impactos na comunicação entre o ATE e a memória sob testes. Sendo assim, fixou-se em 2,5ns o ciclo de testes, o que equivale a uma frequência de operação de 400MHz, ou seja, 800Mbps.

Os *timings* foram divididos em tempos de inicialização, que são executados em modo SDR (*Single Data Rate*), e tempos comuns para *Write*/*Read*, para *Write*, tempos de *Read* e de *Refresh*, que são executados em modo DDR (*Double Data Rate*). Na Tabela 13 estão os tempos inicialmente definidos para utilização no programa de teste. Para tanto, fez-se necessário um detalhamento de cada operação e seus respectivos tempos, que tiveram que ser respeitados, baseando-se no *datasheet* do componente (HYNIX, 2017) e no manual de operações normatizado pela JEDEC (JEDEC Standard, 2012).

Tabela 13 – Tabela de tempos (*timings*) utilizados no programa de teste.

Aplicação	Parâmetro	Descrição	Condição	Mín.	Máx.
Geral	tCK	Cycle Time (Tempo de Ciclo)		2.5ns	0.625ns
Inicialização	tRESET	Reset Time	RESETn = CKE = TEN = ODT = 0; CK = P	700us / tCK	-
	tCKE	Reset High até CKE High	RESETn = 1; CKE = TEN	500us / tCK	-
	tXPR	CKE High to MRS Cmd	= ODT = 0; CK = P	360ns / tCK	-
	tMRD	MRS Cmd até MRS Cmd	RESETn = CKE = 1; TEN	8nCK	-
	tMOD	MRS Cmd até non-MRS Cmd	= ODT = 0; CK = P; Cmd = DES	24nCK	-
	tZQinit	ZQCL Cmd até Primeiro Cmd válido	Cmd = DES	1024nCK	-
Read/Write	tRCD	ACT to READ/WRITE Cmd	Cmd = DES	10nCK	-
	tCCD_L	READ/WRITE Cmd to READ/WRITE Cmd	ZQCL Cmd to first	Cmd = DES	-
	tCCD_S	READ/WRITE Cmd to READ/WRITE Cmd	VALID Cmd	4nCK	-
	tRP	PRE Cmd até ACT Cmd	DLL off	10nCK	-
Write	CWL	CAS Write Latency (MR2[A5:A3])	DLL off	8nCK	-
	tWPRE	DQS Write Preamble (MR4[A12])	PRE Cmd até ACT Cmd	1nCK	-
	tWPST	DQS Write Postamble (MR4[A12])	Diferente BG (Bank Group)	1nCK	-
	tWTR_S	Última escrita (write) Dados até próxima leitura	Mesmo BG (Bank Group)	2nCK	-
	tWTR_L	Última escrita (write) Dados até próxima leitura		4nCK	-
	tWR	Útilma escrita Dados até PRE Cmd		15ns / tCK	-
Read	CL	CAS Latency (MR0[A6:A4;A2])	DLL off	9nCK	-
	tRPRE	DQS Read Preamble (MR4[A12])	Última escrita até	1nCK	-
	tRPST	DQS Read Postamble (MR4[A12])	próximo READ Cmd	1nCK	-
	tRTW	READ Cmd até WRITE Cmd	Última escrita até	7nCK	-
	tRTP	READ Cmd até PRE Cmd	PRE Cmd	4nCK	-
Refresh	Ref_BLCount	Conta bursts de R/W	*bursts* entre refreshes cmd	128	-
	tRFC	Tempo entre REF e ACT cmd		350ns / tCK	-
	Ref_AddCount	Conta endereços de R/W	endereços de colunas entre refreshes cmd	16	-

Fonte: Elaborado pelo autor.

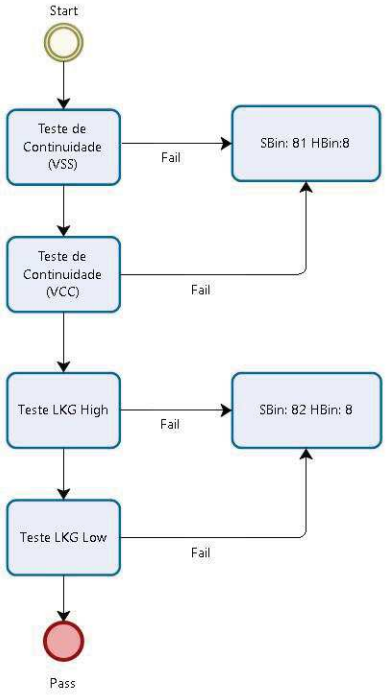
Como pode-se ver, todos tempos foram definidos com base no tempo de ciclo geral (parâmetro *Cycle Time*), o que facilitou a análise e alteração dos mesmos, caso fosse necessário, facilitando a correção de falhas durante o desenvolvimento do programa de testes.

Outro ponto importante é com relação a operação nominal da memória SDRAM DDR4, que apresenta um limite inferior de frequência de 1600Mbps. Sendo assim, se faz necessário operar desabilitando a opção DLL nos registradores de modo (MR1). Desabilitando o DLL, é possível definir *timings* fora de especificação, visando basicamente testabilidade em nível de células e núcleo, desprezando a velocidade de operação.

3.7 Teste Paramétrico

O primeiro teste realizado na memória, por convenção, é o teste paramétrico ou teste DC. Este teste visa detectar falhas grosseiras e evitar a realização do teste funcional em componentes que não possuem nenhuma funcionalidade. O *Teste Paramétrico*, é composto basicamente por dois diferentes tipos de testes estáticos: teste de continuidade e teste de fuga de corrente. Estes dois testes estão subdivididos em testes de continuidade em VSS, teste de continuidade VCC, teste de fuga de corrente *High* (nível de tensão alto) e teste de fuga de corrente *Low* (nível de tensão baixo). O fluxo de testes do teste paramétrico é definido conforme a Figura 42. Os modelos de execução dos testes paramétricos estão descritos no referencial teórico deste projeto (capítulo 2.3).

Figura 42 – Fluxograma do processo de Teste Paramétrico.



Fonte: Elaborado pelo autor.

Como os testes paramétricos não ocorrem no nível funcional da memória (não utilizam ciclos de *clock*, escritas/leituras), os mesmos são descritos sem a necessidade de utilizarmos o modo *pattern* do programa de teste (*APG Pattern Language*). Sua programação ocorre toda no ambiente dos *Test Blocks*, utilizando das *Pin Lists* pré-definidas para execução em todos níveis de pinos digitais e VREFCA. Na Figura 43,têm-se um exemplo de parte do código do programa de teste paramétrico. Neste exemplo está descrita a configuração para os testes de continuidade e a configuração e execução dos testes de continuidade VSS.

Figura 43 – Exemplo de parte do código de teste paramétrico.

```
TEST_BLOCK(tb_parametric ) {

test_block_header();

IntArray r1, r2, r3, r4, r5, r6, r7, r8; // Var to store results

// Continuity Tests Setup
power_down(); // Call function to set ref voltages to zero
vclamp(5 V, -2 V); // Set vclamp because this TB forces current
partime(10 MS); // Set parametric wait time
back_voltage(0 V); // Set background voltage to 0 V
back_voltage_enable(true); // Enable background voltage

// Continuity VSS Setup & Test
output("Running Continuity VSS"); // Test identification for datalog
ipar_force(-100 UA); // Set Force Value = -100uA
vpar_high(-200 MV); // Set high Limit  = -200mV
vpar_low(-1.0 V); // Set low limit   = -1.0V
partest( pass_nivl, pl_DIG ); // Execute Test on Digital Pins
parametric_datalog( pl_DIG ); // Output parametric results to UI site tab
int count = results_get( &r1 ); // Save test results
partest( pass_nivl, pl_VREF ); // Execute Test on Vref Pins
parametric_datalog( pl_VREF ); // Output parametric results to UI site tab
results_get( &r2 ); // Save test results
```

Fonte: Elaborado pelo autor.

A seguir, descreve-se a estrutura dos testes paramétricos implementados, com os respectivos níveis de corrente e tensão utilizados.

3.7.1 Teste de Continuidade (*VSS/VCC*)

Este teste verifica a existência de conexões no componente onde existam curto-circuitos ou circuitos abertos. Para realizá-lo, utiliza-se os parâmetros de tensão em nível de VCC (alto) e VSS (baixo) para detecção de curto-circuito e para detecção de circuito-aberto. O teste utiliza o conjunto dos pinos digitais da memória (conforme descrito na Tabela 9) para execução de um dos testes. Já o pino VREFCA é testado separadamente. Injeta-se uma corrente de -150μA para o teste VSS e 150μA para o teste VCC, medindo-se o nível de tensão nos pinos sob teste. Com base na especificações do *datasheet* ((HYNIX, 2017)), definira-se os limites de -1000mV a -200mV (para VSS) e 200mV a 1000mV (para VCC). Para cada pino relacionado, realiza-se a medição em nível de VSS e VCC. Caso o componente apresente resultado fora dos limites o mesmo é reprovado no teste.

3.7.2 Teste de Fuga de Corrente (*Current Leakage High/Low*)

Este teste verifica a existência de fuga de corrente entre pinos e conexões do componente. A fuga de corrente pode ser classificada em dois modos: fuga de corrente de alta tensão (*Leakage High*) e fuga de corrente de baixa tensão (*Leakage Low*). Da mesma forma que ocorre no teste de

continuidade, define-se os paramêtros para os limites, neste caso em corrente elétrica, e injeta-se uma tensão nos pinos sob teste. Com base na especificações do *datasheet* (HYNIX, 2017), definira-se os limites de -1000nA a 1000nA para ambos os testes. Para cada pino relacionado, realiza-se a medição primeiramente em nível alto, com a injeção de uma tensão de 1.26V diretamente no pino e verifica-se a fuga para o GND (*Current Leakage High*. Após, mede-se em nível baixo, com a injeção de uma tensão de 1.26V diretamente no GND e verifica-se a fuga para o pino sob teste. Caso o componente apresente resultado fora dos limites em qualquer um dos testes, o mesmo é reprovado.

3.8 Algoritmo Checkerboard

Conforme descrito no capítulo 3.6, previamente a execução dos algoritmos se faz necessária a inicialização da memória DRAM. Após a memória inicializada, executa-se o primeiro padrão do primeiro algoritmo, o *Checkerboard*, conforme descrito na Tabela 14. As notações utilizadas na Tabela 14 estão descritas no referencial teórico deste projeto, na Tabela 5.

Tabela 14 – Padrão de teste do algoritmo *Checkerboard*.

Algoritmo	Descrição
Checkerboard	⇑w1 em todas células ímpares ⇑w0 em todas células pares ⇑r1 em todas células ímpares ⇑r0 em todas células pares **repete-se a operação, invertendo w1 e w0*

Fonte: Adaptado de Landrault (2010).

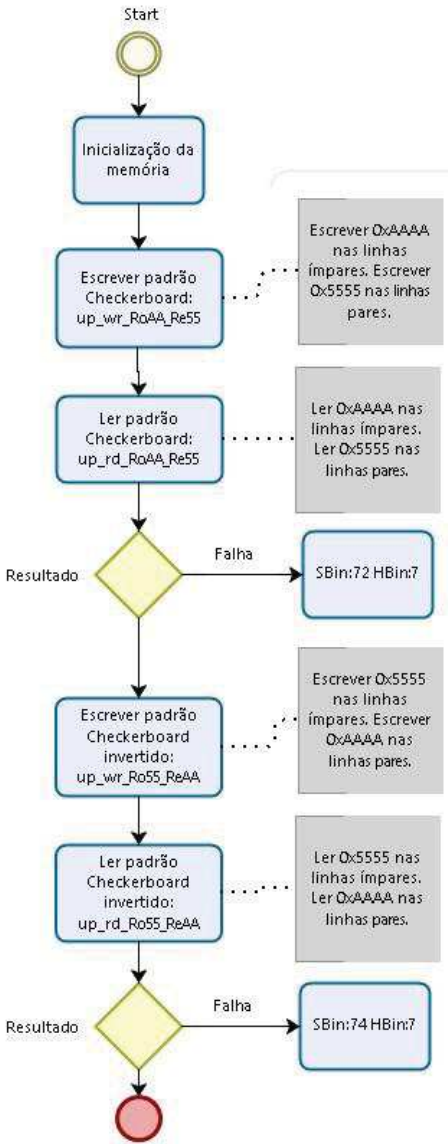
Entretanto, são escritos cinco padrões de dados hexadecimais diferentes, uma vez que é realizado o teste com orientação de palavras (*Test Word-Oriented*, descrito no capítulo 2.3.2.1) e são necessárias algumas combinações de padrões numéricas para realizar a varredura completa de todas linhas e colunas através dos comandos de escrita a serem utilizados, uma vez que a memória DRAM escreve através de comandos de *burst* (conforme descrito no referencial teórico, capítulo 2.3). A fim de executar os mesmos padrões em todas as células da memórias, primeiramente executa-se um padrão de teste nas linhas pares e o padrão inverso nas linhas ímpares. Logo após, executa-se o padrão executado na linha par na linha ímpar, e vice-versa. Continua-se a execução até a execução de todos os padrões referentes ao algoritmo nas linhas pares e ímpares, totalizando dez rotinas de escritas completas (Tabela 15). Na Figura 44 têm-se o fluxograma do algoritmo, com a exemplificação da aplicação com o padrão de dados 0x5555 / 0xAAAA.

Tabela 15 – Padrões de dados hexadecimais utilizados no algoritmo *Checkerboard*.

Padrão de Dados (Nr)	Linhas Pares	Linhas Ímpares
1	0x0000	0xFFFF
2	0xFFFF	0x0000
3	0x5555	0xAAAA
4	0xAAAA	0x5555
5	0x3333	0xCCCC
6	0xCCCC	0x3333
7	0x0F0F	0xF0F0
8	0xF0F0	0x0F0F
9	0x00FF	0xFF00
10	0xFF00	0x00FF

Fonte: Elaborado pelo autor.

Figura 44 – Fluxograma do processo do algoritmo de teste *Checkerboard*.



Fonte: Elaborado pelo autor.

Para execução do algoritmo *Checkerboard* se faz necessário uso do modo DDR, o que não acontece na inicialização e no teste paramétrico, uma vez que são executadas operações de escrita/leitura nos I/Os da memória. Basicamente, o algoritmo é estruturado através de laços (*loops*) onde ocorrem as escritas e leitura, os contadores de endereços que incrementam colunas e linhas, e os contadores de loops que controlam as execuções dos *refreshes*. As operações ocorrem em *bursts*, contudo, a cada comando de escrita e/ou leitura, a operação é realizada em 8 endereços da memória, uma vez que configuração do MR0 está setado em BL8 (*Burst Lenght = 8*). Além disso, para facilitar as repetidas operações, criam-se *subpatterns* com operações de escrita em BL8 e leitura em BL8, bem como com os comandos de *precharge* e *refresh*.

Como descrito nas Figuras 9 e 10, a execução dos comandos de escrita e leitura possui uma sequência de comandos a serem executados. A seguir, temos a primeira parte do código do algoritmo *Checkerboard* descrito no modo *pattern*. Nesta primeira parte do código, temos a inicialização dos contadores (*counts*) e dos registradores nas posições iniciais. O registrador *ymain* tem início 8 posições de endereço antes ao ponto zero, uma vez que no primeiro comando do *loop* este será somando ao contador de *bursts* de 8 BL colocando o mesmo na posição zero, ou seja, primeira coluna da primeira linha da memória. A maioria dos contadores possuem variáveis globais previamente declaradas no código (conforme descrito no capítulo 3.6.2) o que simplifica a inicialização dos mesmos. Na Figura 45 está demonstrada a inicialização realizada para o algoritmo *Checkerboard*.

Figura 45 – Código de inicialização do algoritmo *Checkerboard*.

```
//CHECKERBOARD START

PATTERN(p_checkerboard,double)
@{
xmain(0); //register x used to set the bank group, bank adress and row address
ymain(ymax()-7); //register y used to set the column address
ybase(8); //value to increment column address. BL = 8
count(1,tRCD-1); //tRCD counter
count(2,CWL-2); //WL counter -1 (last cycle in tWPRE instruction)
count(3,3); //Burst Lenght counter (BL8), 4 cycles (DDR)
count(4,Ref_BLCount-1); //Burst counter between refresh commands
count(5,CL-1); //CAS Latency = 10nCK
count(9,xmax()); //Count write/read operation for rows, banks and bank groups. xmax=2^19 - 1
count(22,tRP-1); //Precharge command cycle time
count(23,tRFC-1); //Time between REF command and ACT
@}
```

Fonte: Elaborado pelo autor.

Após a inicialização do *pattern* têm-se o laço de escrita, que conforme descrito pelo padrão do algoritmo (Tabela 14) primeiramente escreve-se 1 em todas células ímpares da memória e 0 em todas células pares. No Apêndice D têm-se o *loop* completo de escrita utilizado no algoritmo *Checkerboard*. Inicia-se com um comando de ACT para acesso a linha e um laço de *timing tRCD* até a execução do próximo comando. Após, realiza-se o comando de escrita

(*WR_command*) e inicia-se o *loop* de escrita com contador de 8BL, que realiza as escritas em 4 ciclos de clocks por utilizar o modo DDR. Assim que escritos os 8 bits, realiza-se a contagem do *refresh* decrementando o *count_4*. Como *tREFI*= $7,8\mu s$ (tempo entre *refreshes*) para DDR4 8Gb, estipulou-se a escrita de 128x8BL = 1024 bits para execução do *refresh*, que equivale a $8\mu s$. Sendo assim, executa-se duas vezes o *refresh* com o *subpattern*, para compensar o *tREFI* excedido, conforme indica a norma JEDEC. Este mesmo valor de 1024 bits também equivale a uma escrita completa de uma linha. Assim, o laço de contagem de linha *WR_row_counter* realiza o incremento da linha no registrador x e retorna ao início do *loop* realizando um comando de ACT para acessar a nova linha.

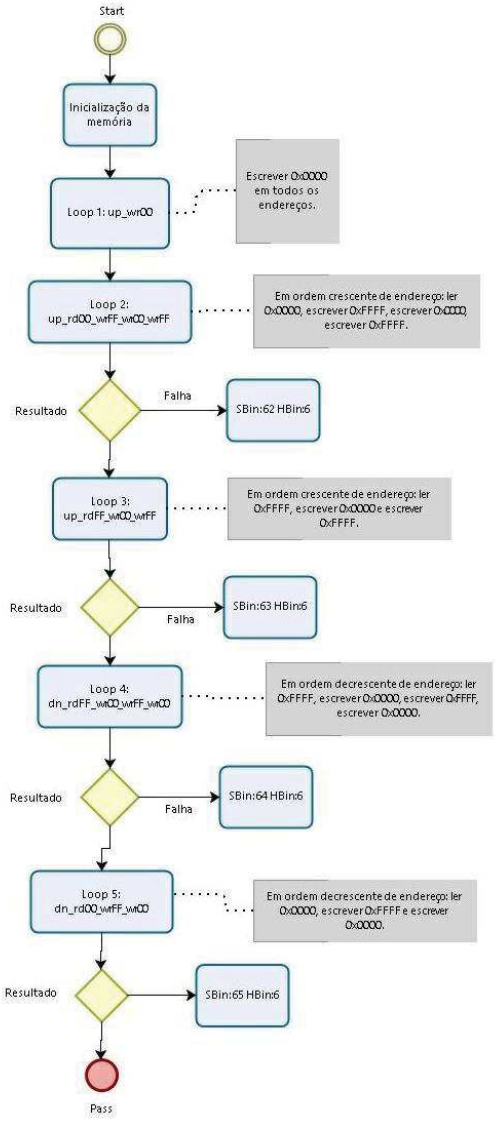
Após o término da escrita realiza-se a leitura, utilizando a mesma lógica da escrita, porém usando o dado como comparativo durante os *strobes* de leituras realizados a cada *burst*. Cada acesso ao *pattern* pelo *test_sequence* utiliza um padrão de dados (*data pattern*) diferente através dos *test_blocks*. Em outras palavras, para cada padrão de dados ocorre a execução de todo o algoritmo com a sequência de escritas e posteriormente a sequência de leituras. Ao se completar os 10 padrões nas linhas pares e ímpares é encerrado o algoritmo e inicia-se o algoritmo seguindo (*March A*).

3.9 Algoritmo March A e March LA

Da mesma forma do algoritmo *Checkerboard*, nos algoritmos *March A* e *March LA*, também ocorre primeiramente a inicialização da memória DRAM e, posteriormente, inicia-se o algoritmo com as escritas e leituras dos padrões definidos para cada um dos algoritmos. Ambos os algoritmos *March* utilizam rotinas de escritas que são escritas e lidas de forma ora em ordem crescente dos endereços e ora na ordem decrescente dos endereços, e com variação entre os padrões de dados escritos/lidos. Na Figura 46, é possível visualizar o fluxograma de teste do algoritmo *March A* e na Figura 47 o fluxograma do algoritmo *March LA*.

Diferentemente do algoritmo *Checkerboard*, os algoritmos *March* realizam o acesso de uma célula de memória por operação. Pode-se observar na Tabela 16 que a cada acesso a determinado endereço ocorrem operações de leitura e escrita, ou seja, realiza-se estas operações em um determinado endereço de linha x coluna e, só após finalizado, repete-se a mesma sequência no endereço subsequente.

Figura 46 – Fluxograma do processo do algoritmo de teste *March A*.



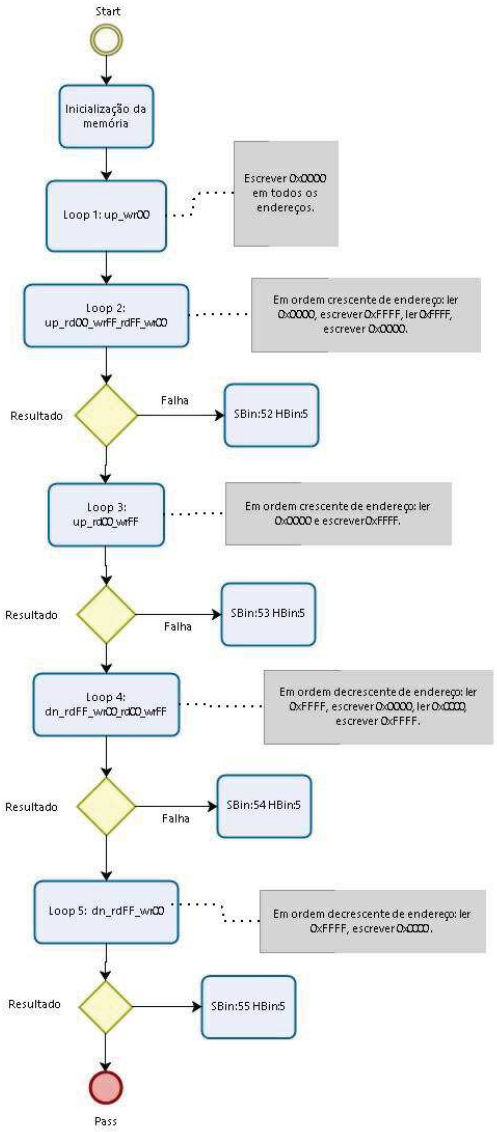
Fonte: Elaborado pelo autor.

Tabela 16 – Padrões dos algoritmos *March A* e *March LA*.

March A	⇑w0 em todas células ⇑(r0,w1,w0,w1) ⇑(r1,w0,w1) ⇓(r1,w0,w1,w0) ⇓(r0,w1,w0)
March LA	⇑w0 em todas células ⇑(r0,w1,r1,w0) ⇑(r0,w1) ⇓(r1,w0,r0,w1) ⇓(r1,w0)

Fonte: Adaptado de Landrault (2010).

Figura 47 – Fluxograma do processo do algoritmo de teste *March LA*.



Fonte: Elaborado pelo autor.

Por se tratar de um teste utilizando estratégias de escritas orientadas por palavras (*Test Word-Oriented*), da mesma forma que ocorre no *Checkerboard*, nos algoritmos *March A* e *March LA* são necessárias uma sequência de padrões de dados para atender de forma mais completa a funcionalidade do algoritmo. Porém, não têm-se a alternância entre linhas no momento da escrita (pares e ímpares), mas sim a execução dos padrões e seus respectivos valores inversos (Tabela 17).

Tabela 17 – Padrões hexadecimais utilizados nos algoritmos *March A* e *March LA*.

Padrão de Dados (Nr)	Valor	Valor Inverso
1	0x0000	0xFFFF
2	0x5555	0xAAAA
3	0x3333	0xCCCC
4	0x0F0F	0xF0F0
5	0x00FF	0xFF00

Fonte: Elaborado pelo autor.

Nos algoritmos *March*, devido a exigência de escrita e leitura em um único endereço, se faz necessário o uso de algumas técnicas para operação utilizando o *burst*. Como já apresentado anteriormente, a memória apresenta opções de operação de escrita somente com *burst* de 8 bits (BL8) e de 4 bits (BC4). Considerando que a memória está previamente configurada para BL8, utilizamos uma técnica de mascaramento dos dados (*Data Mask*) no momento tanto da escrita, como da leitura. Desta forma, mascaram-se as posições de memória que não serão escritas e/ou lidas para que ocorra a operação em um único endereço por vez. Esta técnica, aliada a escrita dos padrões de dados (Tabela 17) tornam os algoritmos executados adequados as regras pré-definidas na literatura.

Quanto ao código utilizado para os algoritmos, ambos utilizam os mesmos contadores e de forma muito semelhante ao descrito no algoritmo *Checkerboard*. A diferença principal está nos blocos de escritas entre as operações de *refresh* que agora são em relações a posições de endereços. Ou seja, como as operações demandam mais tempo, pois em um único endereço são realizadas escritas e leituras em sequência (diferentemente do *Checkerboard*), considera-se o tempo para execução do *refresh* a cada 8 *bursts*. Outra mudança significativa é que como a cada endereço múltiplas operações são realizadas, ora em ordem crescente, ora em ordem decrescente, utilizam-se *loops* para cada linha de operações do algoritmo, conforme mostra a Tabela 16.

No Apêndice E têm-se como exemplo o código utilizado na 5ª linha do padrão do algoritmo *March A*, denominada no código como L5 (*loop_5*. Em L5, realizam-se operações de forma decrescente, ou seja, dos endereços mais significativos para os menos significativos da memória. Como pode-se ver, utilizam-se sub rotinas que chamam os *subpatterns* para execução dos comandos que são utilizados com mais frequência: escritas de 1, escritas de 0, leituras de 1, leituras de 0, *precharge* e *refresh*. Para cada execução existe um *subpattern*, o que facilita a repetição destas operações.

Outro ponto que também diferencia estes algoritmos do *Checkerboard* é a necessidade do mascaramento das colunas na escrita/leitura por *burst*. Para tal, utiliza-se no registrador de dados (*dbase*) uma variável de 8 bits que inicia no bit mais significativo e que é deslocada para a direita (comando *shrdr*) a comando de escrita/leitura, alterando a posição da coluna que não será mascarada. Sendo o *count_12* = 8, sabe-se que a cada *burst lenght* completo o registrador *dbase* é zerado para recomeçar na primeira coluna do próximo *burst*. Desta forma, escreve-se ou lê-se

em um único endereço de coluna x linha a cada operação mesmo utilizando-se 8 BL.

3.10 Seleção das amostras

Visando realizar a validação funcional do programa foram necessárias peças com defeitos funcionais comprovados e peças sem defeitos. Estas peças foram coletadas e selecionadas de falhas de teste de produção da empresa HT Micron. Os componentes de memória, na ocasião, foram testados utilizando-se um ATE de nível industrial com programa de teste desenvolvido pelo fabricante da memória (SK Hynix, Coréia do Sul). Em um segundo momento, estas peças foram novamente testadas, porém utilizando do programa de testes desenvolvido neste projeto no ATE Magnum V. E finalmente, em um terceiro momento, estas peças foram classificadas através do uso do testador TCE3200LP.

Para obter-se um nível de amostragem confiável, foram coletadas 34 amostras, para as quais parte destas eram componentes funcionais e sem falhas, e outras apresentavam falhas no algoritmo *Checkerboard*, parte destas deveriam apresentar falha no algoritmo *March A* e outra parte possuíam falha no algoritmo *March LA*. Contudo, obteve-se um número total de 30 amostras coletadas de falhas de testes funcionais e 4 amostras aprovadas (sem falhas), e realizou-se, posteriormente, o teste no programa desenvolvido, a fim de classificar em quais dos algoritmos as mesmas iriam falhar, conforme descrito no capítulo 4 a seguir. Na Tabela 18, têm-se as amostras selecionadas com seu respectivo número e sua condição: PASS (aprovada) ou FAIL (reprovada).

Tabela 18 – Amostras coletadas para validação e análise comparativa do programa de testes.

Amostra Nr	**Condição**	**Amostra Nr**	**Condição**
1	PASS	18	FAIL
2	PASS	19	FAIL
3	FAIL	20	FAIL
4	FAIL	21	FAIL
5	FAIL	22	FAIL
6	FAIL	23	FAIL
7	FAIL	24	FAIL
8	FAIL	25	FAIL
9	FAIL	26	FAIL
10	FAIL	27	FAIL
11	FAIL	28	FAIL
12	FAIL	29	FAIL
13	FAIL	30	FAIL
14	FAIL	31	PASS
15	FAIL	32	PASS
16	FAIL	33	FAIL
17	FAIL	34	FAIL

Fonte: Elaborado pelo autor.

Tendo-se coletadas as amostras e o programa de teste implementado, inicia-se as etapas de

validação e nível de aderência do programa de testes, conforme descrito no capítulo à seguir.

4 RESULTADOS OBTIDOS

Neste capítulo serão apresentados os resultados obtidos através da validação do programa de testes, análise comparativa do programa e resultados com os demais testadores, bem como as análises destes resultados obtidos.

4.1 Validação do programa de testes

Com a primeira revisão do programa de testes finalizada, iniciaram-se as execuções de testes utilizando as amostras coletadas. Em um primeiro momento, realizaram-se os testes com a utilização do programa de testes e componentes aprovados, a fim de validar a funcionalidade do programa. A primeira validação é do teste paramétrico, que não apresenta falhas utilizando peças previamente aprovadas.

Após ajustes em alguns parâmetros de tempos e correções de *bugs*, executa-se parcialmente o programa, delimitado até a etapa de inicialização da memória DRAM. Neste momento, valida-se a inicialização utilizando um comando simples de escrita 0x0101 e lê-se a resposta da memória após o sinal de DQS (*Data Query Strobe*) deste mesmo sinal.

Com isto, executou-se o algoritmo *Checkerboard*. O algoritmo apresenta ao todo 10 padrões de testes (Tabela 14, uma vez que executa os mesmos nas linhas pares e ímpares. A execução do *Checkerboard* também, apresentou resultado satisfatório, aprovando os dois componentes bons sob testes. Os sinais analisados no *Wave Tool* comportou-se de acordo com o esperado, com as escritas e leituras dos *bursts* ocorrendo de forma correta. No Apêndice F têm-se a saída de resultados na tela principal do software UI referente aos dois últimos padrões de testes do algoritmo *Checkerboard*. Neste resultado são apresentados os status de cada um dos DUTs (*Pass* ou *Fail*), o número do teste dentro do algoritmo (que está relacionado ao padrão de dados utilizado), o nome do teste, o nome do *pattern* e o status de *pass* (P) ou *fail* (F) de cada um dos 16 pinos de I/O (DQs).

Com o *Checkerboard* finalizado, realizou-se a execução do algoritmo *March A* e, após validado, a verificação do algoritmo *March LA*. No Apêndice G têm-se ambos os algoritmos com seus resultados de execução, ilustrando-se o resultado dos últimos dois padrões de testes do *March A* e os dois primeiros do *March LA*, todos aprovados conforme esperado. Os algoritmos *March* executaram a metade dos padrões de dados do algoritmo *Checkerboard*, conforme Tabela 17, ou seja, somente cinco testes. Mensurou-se o tempo de teste de cada um dos algoritmos e o tempo total de execução, conforme ilustrado na Tabela 19.

Tabela 19 – Tempos de teste de cada algoritmo no programa desenvolvido.

Algoritmo	**Tempo de Teste**
Paramétrico	5.61s
Checkerboard	106.80s
March A	3290.50s
March LA	2914.71
Total	6317.65s

Fonte: Elaborado pelo autor.

Observa-se que o tempo dos algoritmos *March* são extremamente longos, devido principalmente a singularidade de escrita/leitura em um endereço de memória. Algumas modificações foram estudadas visando a redução deste tempo, porém detectou-se objeções para a execução das mesmas.

1. **Utilização do *Burst Chop***: uma alternativa é a mudança no método de escritas e leituras através do *burst lenght* (BL) de 8 bits para o *burst chop* (BC) de 4 bits. Com esta alteração visa-se principalmente realizar operações com menor tempo de execução com relação ao BL8 já utilizado, facilitando o mascaramento e endereçamento. A principal objeção para utilizar este método refere-se à complexidade para execução da operação com 4 bits utilizando os registradores existentes no testador. Tentou-se realizar algumas modificações, porém devido a limitação do tempo para o término do projeto não foi exequível esta atualização.

2. **Aumento da frequência**: a execução do programa de testes está sendo realizada com uma frequência de 400MHz, ou seja, metade da frequência fornecida pelo ATE. Esta redução se dá principalmente por algumas limitações estruturais relacionadas ao DSA utilizado e os atrasos existentes. Estes por sua vez apresentam sinais de má qualidade com a execução na frequência nominal, além de atrasos que impactam nas operações de escrita e leitura. Neste caso, a principal objeção se apresenta nos atrasos detectados pela calibração TDR do DSA e dos seus impactos nos sinais em alta frequência.

3. **Alteração do padrão do algoritmo**: uma outra opção é a realização do teste com as operações sendo realizadas em BL8 e sem o mascaramento. Ou seja, realizar operações em 8 endereços de memória, ao invés de somente um, conforme prescreve o padrão do algoritmo. Desta forma, utilizaria-se uma metodologia similar a utilizada no algoritmo *Checkerboard*, tendo-se resultados de tempo muito similares. Entretanto, as objeções neste caso encontram-se na descaracterização do algoritmo na sua forma original e no risco da redução de cobertura de falhas. Recomenda-se uma análise detalhada visando aderência e cobertura de falhas nesta ocasião.

4. **Adiamento de *refreshes***: a memória DRAM DDR4, baseada na norma JEDEC, possibilita o adiamento de *refreshes* e posterior execução. Desta maneira, executaria-se uma linha

inteira de endereços com a execução do *refresh* somente ao término, evitando mais de um *refresh* e uma mesma linha de endereço. A objeção encontrada com isto seria que o ganho em tempo não seria tão significativo, não viabilizando a alteração.

5. **Utilizar somente um padrão de dados**: por fim, uma opção também existente seria da redução dos padrões de dados, utilizando-se ao invés de cinco, somente um ou dois que apresentem os melhores resultados. Para tal, espera-se uma redução em tempo de 5 vezes o tempo executado atualmente. Entretanto, da mesmo forma do item 3, existe uma descaracterização do algoritmo na sua forma original e um risco da redução de cobertura de falhas. Recomenda-se também uma análise detalhada visando aderência e cobertura de falhas nesta ocasião.

Portanto, visando executar os testes de nível de aderência entre as soluções de teste, optou-se por manter a versão final do programa conforme a previamente validada, sem alterações visando redução de tempo e com foco principal na detecção de falhas e cobertura.

4.2 Aderência do programa de testes

Conforme apresentado nos objetivos deste projeto e previamente definido na metodologia (Figura 35), após o término do desenvolvimento do programa de testes e sua validação realizou-se a o teste comparativo para avaliar o nível de aderência do programa desenvolvido no ATE Magnum V com o testador de bancadas TCE3200LP. Desta forma, o teste utilizou as amostras coletadas (descrito no capítulo 3.10). Primeiramente, executou-se os testes nos componentes no ATE Magnum V com o programa desenvolvido e posteriormente realizou-se os testes no TCE3200LP. Após, obteve-se os resultados de aderência entre os dois testadores, conforme demonstrado na Tabela 20. Três itens foram avaliados na comparação: resultado, algoritmos e pinos. Ambos são descritos na Tabela para cada amostra testada e apresentado seu resultado na coluna *Aderência*, com a descrição de OK ou NOK (não OK).

Tabela 20 – Aderência dos resultados do ATE Magnum 5 e do testador TCE3200LP.

Nr da Amostra	Resultado MG5	Algoritmo	Falha (pinos)	Resultado TCE3200LP	Falha (pinos)	Aderência
1	PASS	-	-	PASS		Resultado: OK
2	PASS	-	-	PASS		Resultado: OK
3	FAIL	tb_checkerboard	(DQL0)FFFFFFFFFFFFFFFF(DQU7)	FAIL	(DQL0)FFFFFFFFPPPPPFPPP(DQU7)	Resultado: OK
		tb_marchA				Algoritmos: OK
		tb_marchLA				Pinos: NOK
4	FAIL	tb_checkerboard	(DQL0)FPPPPPPPPPPPPPP(DQU7)	FAIL	(DQL0)FPPPPPPPPPPPPPP(DQU7)	Resultado: OK
		tb_marchA	(DQL0)FPPPPPPPPPPPPPP(DQU7)		(DQL0)FPPPPPPPPPPPPPP(DQU7)	Algoritmos: OK
		tb_marchLA	(DQL0)FPPPPPPPPPPPPPP(DQU7)		(DQL0)FPPPPPPPPPPPPPP(DQU7)	Pinos: OK
5	FAIL	tb_checkerboard	(DQL0)PFFFFFFFPPPPPPPP(DQU7)	FAIL	(DQL0)FFFFFFFFPPPPPPPP(DQU7)	Resultado: OK
		tb_marchA	(DQL0)PFFFFFFFPPPPPPPP(DQU7)		(DQL0)FFFFFFFFPPPPPPPP(DQU7)	Algoritmos: OK
		tb_marchLA	(DQL0)PFFFFFFFPPPPPPPP(DQU7)		(DQL0)FFFFFFFFPPPPPPPP(DQU7)	Pinos: OK
6	FAIL	tb_checkerboard	(DQL0)FFFFFFFFFFFFFFFF(DQU7)	PASS		Resultado: NOK
		tb_marchA	(DQL0)FFFFFFFFFFFFFFFF(DQU7)			
		tb_marchLA	(DQL0)FFFFFFFFFFFFFFFF(DQU7)			
7	PASS	-	-	PASS	-	Resultado: OK
8	FAIL	tb_parametric - LKG High	TEN (1.030uA)	PASS	-	Resultado: NOK
9	PASS	-	-	FAIL		Resultado: NOK
10	PASS	-	-	PASS		Resultado: OK
11	FAIL	tb_checkerboard	(DQL0)PFFFFFFFFFFFFFFF(DQU7)	PASS		Resultado: NOK
		tb_marchA	(DQL0)PPPPPPFPPPFFPFFF(DQU7)			
		tb_marchLA	(DQL0)PFPFFFFPPFPFFFFF(DQU7)			
12	FAIL	tb_checkerboard	(DQL0)FFFFFFFFFFFFFFFF(DQU7)	PASS		Resultado: NOK
		tb_marchA	(DQL0)PFFFPPFFFPFPFFFF(DQU7)			
		tb_marchLA	(DQL0)FFFFFFFFFFFFFFFF(DQU7)			
13	FAIL	tb_parametric - LKG High	TEN (1.024uA)	PASS		Resultado: NOK
14	FAIL	tb_checkerboard	(DQL0)FFFFFFFFFFFFFFFF(DQU7)	PASS	-	Resultado: NOK
		tb_marchA	(DQL0)FFFFFFFFFFFFFFFF(DQU7)			
		tb_marchLA	(DQL0)FFFFFFFFFFFFFFFF(DQU7)			
15	FAIL	tb_checkerboard	(DQL0)FFFFFFFFFFFFFFFF(DQU7)	FAIL	-	Resultado: OK
		tb_marchA	(DQL0)FFFFFFFFFFFFFFFF(DQU7)		-	Algoritmos: NOK
		tb_marchLA	(DQL0)FFFFFFFFFFFFFFFF(DQU7)		MarchLA	Pinos: NOK
16	PASS	-	-	PASS		Resultado: OK
17	FAIL	tb_checkerboard	(DQL0)FFFFFFFFFFFFFFFF(DQU7)	FAIL	Power Short Detected	Resultado: OK
		tb_marchA	(DQL0)FFFFFFFFFFFFFFFF(DQU7)			Algoritmos: NOK
		tb_marchLA	(DQL0)FFFFFFFFFFFFFFFF(DQU7)			Pinos: NOK
18	FAIL	tb_parametric - LKG High	DQU1 (+-6.168uA)	FAIL	tb_parametric - All Short (test stopped)	Resultado: OK
		tb_checkerboard	(DQL0)PPPPPPPPPFPPPPPP(DQU7)			Algoritmos: N/A
		tb_marchA	(DQL0)PPPPPPPPPFPPPPPP(DQU7)			Pinos: N/A
		tb_marchLA	(DQL0)PPPPPPPPPFPPPPPP(DQU7)			
19	FAIL	tb_checkerboard	(DQL0)FFFFFFFFFFFFFFFF(DQU7)	PASS	-	Resultado: NOK
		tb_marchA	(DQL0)FFFFFFFFFFFFFFFF(DQU7)			
		tb_marchLA	(DQL0)FFFFFFFFFFFFFFFF(DQU7)			
20	FAIL	tb_parametric - LKG High	TEN (1.011uA)	PASS	-	Resultado: NOK
21	FAIL	tb_checkerboard	(DQL0)PPPPPPPPPPPPFPP(DQU7)	PASS	-	Resultado: NOK
22	FAIL	tb_parametric - LKG High	TEN (1.035uA)	PASS	-	Resultado: NOK
23	PASS	-	-	PASS	-	Resultado: OK
24	PASS	-	-	PASS	-	Resultado: OK
25	FAIL	tb_parametric - LKG High	TEN (1.035uA)	FAIL	MarchLA	Resultado: OK Algoritmos: NOK Pinos: NOK
26	FAIL	tb_parametric	TEN (1.011uA)	PASS	-	Resultado: NOK
		tb_checkerboard	(DQL0)FFFFFFFFFFFFFFFF(DQU7)			
		tb_marchA	(DQL0)PFPFPPFFFFFFFFFP(DQU7)			
		tb_marchLA	(DQL0)PFPFPPFFFFFFFFFP(DQU7)			
27	FAIL	tb_parametric - LKG High	TEN (1.013uA)	PASS	-	Resultado: NOK
28	FAIL	tb_checkerboard	(DQL0)FFFFFFFFFFFFFFFF(DQU7)	PASS	-	Resultado: NOK
		tb_marchA	(DQL0)FFFFFFFFFFFFFFFF(DQU7)			
		tb_marchLA	(DQL0)FFFFFFFFFFFFFFFF(DQU7)			
29	FAIL	tb_checkerboard	(DQL0)FFFFFFFFFFFFFFFF(DQU7)	PASS	-	Resultado: NOK
		tb_marchA	(DQL0)FFFFFFFFFFFFFFFF(DQU7)			
		tb_marchLA	(DQL0)FFFFFFFFFFFFFFFF(DQU7)			
30	FAIL	tb_parametric - LKG High	TEN (1.002uA)	PASS	-	Resultado: NOK
31	PASS	-	-	PASS	-	Resultado: OK
32	PASS	-	-	PASS	-	Resultado: OK
33	FAIL	tb_parametric - LKG High	TEN (1.013uA)	PASS	-	Resultado: NOK
34	PASS	-	-	PASS	-	Resultado: OK

Como pode-se ver, o nível de aderência entre os testes não foi elevado. Obteve-se um resultado de **47,06%** de aderência de resultados, sendo que destes resultados, **8,82%** tiveram aderência de algoritmos e **5,88%** tiveram aderência de pinos.

Além disso, realizou-se a análise de aderência entre a condição inicial da amostra (Tabela 18) e os resultados obtidos no ATE Magnum V e no TCE3200LP. Na Tabela 21 a seguir estão descritos os resultados de cada um dos testes realizados nas amostras e o percentual de aderência do teste desenvolvido (*Resultado MG5*) e do TCE3200LP com relação a condição inicial da amostra.

Tabela 21 – Aderência entre condição inicial das amostras e resultados dos testes.

Amostra Nr	**Condição Inicial**	**Resultado MG5**	**Resultado TCE3200LP**
1	PASS	PASS	PASS
2	PASS	PASS	PASS
3	FAIL	FAIL	FAIL
4	FAIL	FAIL	FAIL
5	FAIL	FAIL	FAIL
6	FAIL	FAIL	PASS
7	FAIL	PASS	PASS
8	FAIL	FAIL	PASS
9	FAIL	PASS	FAIL
10	FAIL	PASS	PASS
11	FAIL	FAIL	PASS
12	FAIL	FAIL	PASS
13	FAIL	FAIL	PASS
14	FAIL	FAIL	PASS
15	FAIL	FAIL	FAIL
16	FAIL	PASS	PASS
17	FAIL	FAIL	FAIL
18	FAIL	FAIL	FAIL
19	FAIL	FAIL	PASS
20	FAIL	FAIL	PASS
21	FAIL	FAIL	PASS
22	FAIL	FAIL	PASS
23	FAIL	PASS	PASS
24	FAIL	PASS	PASS
25	FAIL	FAIL	FAIL
26	FAIL	FAIL	PASS
27	FAIL	FAIL	PASS
28	FAIL	FAIL	PASS
29	FAIL	FAIL	PASS
30	FAIL	FAIL	PASS
31	PASS	PASS	PASS
32	PASS	PASS	PASS
33	FAIL	FAIL	PASS
34	FAIL	PASS	PASS
Aderência %		***79,41%***	***35,29%***

Fonte: Elaborado pelo autor.

Além da comparação do nível de aderência entre os resultados de teste, realizou-se também uma comparação entre os tempos de execução de ambos os testes. O tempo total de teste mensurado no *Turbocats* foi de 113s e o tempo total de teste do programa desenvolvido no ATE é de 6318s (conforme Tabela 19, ou seja, 56 vezes mais lento do que o teste no testador de bancada.

4.3 Análise de Custo e Produtividade

Uma análise comparativa dos custos das aplicações também foi realizada, a fim de contextualizar economicamente cada uma delas. Primeiramente, gerou-se um levantamento de cada um

dos equipamentos (Tabela 22). Para o ATE Magnum V é necessário considerar os acessórios complementares que compõem todo o testador (*handler*, kit de conversão - *COK* e DSA).

Tabela 22 – Comparativo entre valores dos testadores.

Equipamento	**Item**	**Valor (US$)**
Magnum V	Magnum	2.800.000,00
	Handler	550.000,00
	CoK	100.000,00
	DSA	100.000,00
	Total	3.550.000,00
Turbocats	TCE3200LP	36.000,00

Fonte: Elaborado pelo autor.

Em uma relação direta, pode-se estimar que com o valor de aquisição de um ATE Magnum V completo pode-se adquirir aproximadamente 100 testadores Turbocats TCE3200LP. Esta relação foi utilizada para a análise a seguir, onde de uma forma suscinta são comparados **Tempo de Teste**, **Qualidade do Teste** e **Produtividade**, a fim de avaliar as soluções de uma forma diversificada. Na Tabela 23, considera-se também uma análise estimativa, que chamamos de *Magnum V Otimizado*. Esta análise estimativa considera uma otimização da solução desenvolvida (atual), considerando basicamente três modificações descritas anteriormente no capítulo 4.1: aumento da frequência de operação para 1600 Mbps, utilização do BC4 ao invés do BL8 e seleção de somente um padrão de dados (*data pattern*) ao invés de cinco. Com estas otimizações, observa-se alguns retrospectos muito positivos com relação ao tempo de teste, o que acaba dando ainda mais viabilidade ao programa de teste desenvolvido.

Tabela 23 – Tabela comparativa entre soluções existentes e otimizadas.

Item	**Parâmetro**	**Turbocats**	**Magnum V Desenvolvido**	**Magnum V Otimizado (Estimado)**
Tempo de Teste	Paramétrico (s)	20	5,6	5,6
	Checkerboard (s)	14	106,8	10,7
	March A (s)	38	3290,5	329,1
	March LA (s)	41	2914,7	291,5
	Total (s)	113	6317,6	636,8
Qualidade do Teste	Aderência com teste do fabricante	35,29%	79,41%	79,41%
Produtividade	Paralelismo por testador (DUTs)	3	192	384
	Paralelismo Equivalente	300	192	384
	UPH Máximo	9557	109	2170

Fonte: Elaborado pelo autor.

** Considerando-se 100 testadores TCE3200LP com 3 DUTs cada.*

Conforme ilustrado na Tabela 23, com relação a tempo de teste, a solução mais rápida ainda fica sendo o testador de bancada. Porém, a estimativa da solução desenvolvida otimizada apresenta uma redução de quase dez vezes em relação à original. No que diz respeito a aderência com o teste do fabricante, que seria a classificação da qualidade do teste, pode-se estimar que a solução otimizada terá o mesmo resultado da solução atual, uma vez que será reduzida a quantidade de padrões de teste, podendo reduzir a cobertura, mas por outro lado a frequência de teste será aumentada, podendo aumentar a detecção de falhas de retenção e latência. Já na análise de produtividade, utilizou-se a regra de equivalência pela análise dos custos de cada um dos equipamentos (Tabela 22). Já para a solução otimizada, estipulou-se um paralelismo máximo, com o compartilhamento de canais otimizado, chegando em 384 DUTs por ciclo de teste. Nessa comparação, ainda temos o testador de bancada apresentando melhor resultado: 9557 unidades testadas por hora. Entretanto, não fora considerada nesta análise o tempo e custo operacional: tempo de manipulação manual das peças, mão de obra de operadores, espaço físico necessário, entre outros fatores não considerados.

4.4 Análise dos Resultados

Conforme demonstrado no capítulo anterior, a aderência entre os resultados do teste desenvolvido no ATE e do testador de bancada não foi satisfatória. Todavia, obteve-se um bom nível de **aderência de resultados de aproximadamente 80% entre o teste desenvolvido e a condição conhecida das amostras**. Isso demonstra principalmente que com relação a cobertura e qualidade do teste, o programa de testes desenvolvido no ATE é muito mais eficaz do que o testador TCE3200LP da *Turbocats*.

Por outro lado, não pode-se descartar a vasta diferença entre os tempos de execução de ambos os testes. Realizando um cálculo simples, considerando o paralelismo de 2 DUTs do ATE e 3 DUTs para o TCE, pode-se dizer que para cada 2 componentes testados no ATE executa-se aproximadamente os testes de 160 componentes no testador de bancada.

Entretanto, uma análise importante a ser considerada é das possibilidades de otimização ainda existentes na solução de teste desenvolvida. Com um aumento do paralelismo do ATE, sem modificação do programa de teste, podemos testar um máximo de 192 componentes por ciclo de teste. Com esta expansão, tem-se uma reversão no cenário, no qual para cada teste executado de 6318s (aproximadamente 2hrs) no ATE obtém-se 192 componentes testados, contra os mesmos 160 componentes do testador de bancada. Obviamente, a aquisição de mais DSAs demanda recursos financeiros, o que por sua vez poderia ser investido também na aquisição de testadores de bancada para uma operação em grande escala. Com a compra de um único testador de bancada é possível dobrar este número de componentes para 320, o que já ultrapassaria novamente os resultados do ATE com capacidade máxima de DSAs. Conforme na análise comparativa da Tabela 23, com uma estimativa da solução otimizada podemos chegar em um UPH de 2.170 peças. Este número por sua vez, ainda fica aproximadamente 4,5 vezes menor do que uma

solução equivalente considerada no testador de bancada, utilizando-se de um número maior de testadores (100 testadores TCE3200LP).

Em resumo, visando baixo custo com uma solução menos robusta, pode-se dizer que o TCE3200LP pode apresentar melhor desempenho produtivo com relação a solução atual desenvolvida no ATE Magnum V. Por outro lado, a confiabilidade do teste está muito mais elevada nos resultados do ATE, que apresenta um nível de qualidade superior e toda cadeia subsequente que isto pode apresentar: menor retorno de campo, fidelização com o cliente, redução nos custos de retrabalho, etc.

Contudo, sabe-se que ainda existem possibilidades significativas para redução de tempo de teste, conforme descritas na seção 4.1. Para isto, deve-se levar em conta análises visando a qualidade do teste e manutenção e/ou melhoria do nível de aderência com os testes do ATE de uso da produção da HT Micron.

5 CONCLUSÃO

Primeiramente, é possível concluir que a execução do projeto apresentou resultados muito positivos, tendo em vista que o seu objetivo principal, o desenvolvimento de uma solução para teste de DRAM utilizando um testador automático industrial, foi atingido. Através da elaboração de um programa de testes funcional em um testador ATE modelo Magnum V da Teradyne para testes de memórias SDRAM DDR4, obteve-se como resultado final a implementação de um teste paramétrico e de três algoritmos de testes funcionais: *Checkerboard*, *March A* e *March LA*. A solução de testes desenvolvida apresentou resultados muito bons, principalmente no que diz respeito a aderência de resultados com as amostras analisadas, uma vez que utilizaram-se poucos algoritmos de testes funcionais e obteve-se um nível de aderência de aproximadamente 80%. Com isto, os demais objetivos específicos relacionados a difusão das tecnologias referentes a testes de memórias e análises comparativas quanto a cobertura de falhas e desempenho também foram concluídos.

A implementação de um teste funcional para componentes de memória SDRAM DDR4 destaca-se principalmente pela sua complexidade, tanto pela aquisição de conhecimento das funcionalidades e protocolos do dispositivo a ser testado, bem como para a programação e escrita do código utilizando a linguagem própria do ATE. As limitações técnicas e de especificações, reflexo da propriedade intelectual, reflete em um desafio ainda maior no que se diz respeito a execução de testes em componentes de memórias. Além disto, alia-se uma poderosa ferramenta que é o ATE, que pode atingir níveis de testes de excelente qualidade, mas que, por outro lado, possui uma enorme complexidade para ser programado e utilizado na sua melhor forma.

Já no aspecto que analisa a viabilidade da aplicação deste teste a nível industrial, ainda pode-se dizer que se faz necessária uma reestruturação do programa com foco na redução do tempo de testes. Conforme descrito com uma das principais justificativas deste projeto, o tempo de teste é um dos contribuintes principais para o alto custo de teste de componentes eletrônicos, entre eles as memórias. Sendo assim, tendo um tempo de teste elevadíssimo, temos por consequência um custo de teste muito alto, o que por sua vez acaba inviabilizando o uso do mesmo para uma aplicação a nível de teste de produção ou teste industrial. Já considerando uma solução otimizada, que apresenta algumas modificações a nível de programa de teste e algumas aquisições de equipamentos, os resultados já se apresentam um pouco mais competitivos, podendo-se estimar um UPH de 2.170 peças. Entretanto, uma solução de teste de bancada equivalente no que se refere a investimentos com relação ao ATE, poderia chegar a um UPH estimado de 9.557 componentes.

Por outro lado, a superioridade no que diz respeito a qualidade e cobertura de teste comparando-se com o TCE3200LP nos mostra que este programa de testes pode ser uma boa ferramenta para uso de testes de engenharia, análise de falhas, desenvolvimento de novos projetos, entre outras

possibilidades que não demandem de um alto volume de componentes para teste e desprezem o tempo de teste elevado.

Por fim, é possível afirmar que o conhecimento adquirido, tanto com relação ao funcionamento e especificações técnicas das memórias DRAM, como com relação a programação do ATE Magnum V, foram de grande importância para o crescimento profissional e acadêmico, além das oportunidades deixadas para continuidade e novas perspectivas através deste modelo agora desenvolvido. Observa-se possibilidades de análises detalhadas de falhas, visando caracterização de defeitos e assinaturas de falhas. Além do mais, existem oportunidades de melhoria do programa de testes, com a otimização dos tempos e do *burst*, visando redução do tempo de teste. Outra melhoria também existente é relacionada a cobertura de falhas, com possibilidade de inclusão de algoritmos e demais testes, visando o aumento da correlação com a solução de teste industrial hoje já existente na empresa onde realizou-se este estudo de caso.

REFERÊNCIAS

ABISEMI. Oportunidades e desafios marcam o crescimento do setor de semicondutores brasileiro. p. 1, 2017. Disponível em: <http://www.abisemi.org.br/abisemi/noticia/41/oportunidades-e-desafios-marcam-o-crescimento-do-setor-de-semicondutores-brasileiro>. Citado na página 18.

ABISEMI. ABISEMI DEBATE OPORTUNIDADES E DESAFIOS PARA O SETOR DE SEMICONDUTORES BRASILEIRO. 2019. Disponível em: <http://www.abisemi.org.br/abisemi/noticia/77/abisemi-debate-oportunidades-e-desafios-para-o-setor-de-semicondutores-brasileiro>. Citado na página 14.

AL-ARS, Z. *DRAM fault analysis and test generation*. [s.n.], 2005. ISBN 9789090196121. Disponível em: <http://www.narcis.nl/publication/RecordID/oai:tudelft.nl:uuid:93137061-e3ef-418e-aa7a-b7539f70af50>. Citado 9 vezes nas páginas 14, 23, 24, 25, 26, 27, 31, 32 e 33.

ASHOK, K. S. *Semiconductor Memories Technology, Testing, and Reliability*. [S.l.]: Wiley-IEEE Press, 2002. v. 133. 8 p. ISSN 0018-9464. ISBN 9780470546406. Citado 2 vezes nas páginas 28 e 32.

BONATTO, A. C. *Núcleos de Interface de Memória DDR SDRAM para Sistemas-Em-Chip*. Tese (Doutorado) — UFRGS, 2009. Citado na página 42.

BUSHNELL, M. L.; AGRAWAL, V. D. *Essential of Electronic Testing for Digital, Memory & Mixed-Signal VLSI Circuits*. Rutgers University; Bell Labs, Lucent Technologies: KLUWER ACADEMIC PUBLISHERS, 2002. 713 p. ISBN 0792386868. Citado 7 vezes nas páginas 14, 20, 29, 30, 34, 35 e 38.

David Tawei Wang. Modern DRAM Memory Systems: Performance Analysis And A High Performance, Power-Constrained Dram Scheduling Algorithm. 2005. Disponível em: <http://www.ece.umd.edu/~blj/papers/thesis-PhD-wang--DRAM.pdf>. Citado na página 22.

DING, J. *DC Parametric Test and IDDQ Test Using Advantest T2000 ATE*. Tese (Doutorado), 2015. Citado 2 vezes nas páginas 31 e 41.

GOER, A. J. V. D.; NEEF, J. D. Industrial evaluation of DRAM tests. *Proceedings -Design, Automation and Test in Europe, DATE*, Memory Tes, p. 623–630, 2004. ISSN 15301591. Citado 5 vezes nas páginas 14, 27, 28, 34 e 41.

GOOR, A. J. V. d.; ABADIR, M. S.; CARLIN, A. Minimal test for coupling faults in word-oriented memories. *Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition*, 2002. Citado 3 vezes nas páginas 37, 38 e 41.

GOOR, A. J. V. D. et al. Converting March Tests for Bit-Oriented Memories into Tests for Word-Oriented Memories. Citado na página 41.

Governo Federal. *Lei 11.484 - PADIS*. Casa Civil, 2007. Disponível em: <http://www.planalto.gov.br/ccivil_03/_ato2007-2010/2007/lei/l11484.htm>. Citado 2 vezes nas páginas 14 e 17.

HAMDIOUI, S.; AL-ARS, Z.; GOOR, A. J. V. D. Testing static and dynamic faults in random access memories. *Proceedings of the IEEE VLSI Test Symposium*, v. 2002-Janua, p. 395–400, 2002. Citado na página 36.

HYNIX, S. 8Gb DDR4 SDRAM Datasheet. 2017. Citado 11 vezes nas páginas 14, 25, 43, 44, 46, 47, 67, 69, 72, 73 e 101.

JEDEC Standard. JESD79-4 - DDR4 SDRAM. n. September, 2012. Citado 9 vezes nas páginas 43, 44, 46, 48, 49, 66, 67, 68 e 69.

JHA, N. K.; GUPTA, S. Testing of digital systems. v. 69, n. 10, p. 1321–1333, 2003. Citado 10 vezes nas páginas 15, 16, 21, 28, 29, 30, 32, 33, 34 e 37.

JONE, D. W.-B. Automatic Test Equipment - DRAM IC category. *Department of ECECS, University of Cincinnati*, 2013. Citado 2 vezes nas páginas 33 e 39.

KHOO, V. C. Cost of Test Case Study for Multi-site Testing in Semiconductor Industry with Firm Theory. *International Journal of Business and Management Invention*, v. 3, n. 4, p. 14–27, 2014. Citado 4 vezes nas páginas 19, 39, 40 e 42.

KRUG, M. R. Teste de Dispositivos. 2017. Citado 3 vezes nas páginas 16, 17 e 28.

LANDRAULT, C. *Memory Testing*. Ehitajate tee 5, 19086 Tallinn, Estonia, 2010. 72 p. Disponível em: <http://ati.ttu.ee/en>. Citado 4 vezes nas páginas 35, 36, 73 e 77.

LIN, K.-j.; WU, C.-w. Functional Testing of Content-Addressable Memories. 2007. Citado na página 30.

LUBASZEWSKI, M.; COTA, F.; KRUG, M. R. Teste e Projeto Visando o Teste de Circuitos e Sistemas Integrados. *Concepção de Circuitos Integrados*, p. 167–189, 2002. Citado na página 40.

MORAES, M.; KONDO, E. *Magnum V Memory Tester : Programming Basics*. 2017. Citado 2 vezes nas páginas 50 e 56.

Nextest/Teradyne. Magnum V Programmer ' s Manual. 2014. Citado 7 vezes nas páginas 51, 52, 53, 55, 57, 58 e 59.

Nextest/Teradyne. Magnum V Test System Specifications. p. 1–14, 2014. Citado na página 50.

RABAEY, J. M. et al. *Digital Integrated Circuits - A Design Perspective*. [S.l.: s.n.], 2002. ISBN 0814403352. Citado 3 vezes nas páginas 20, 22 e 26.

RAMACHANDRAN, A. DRAM TESTING USING INTERLEAVING TEST ALGORITHM. v. 2, n. 6, p. 199–205, 2014. Citado na página 41.

SEONG, N. H. et al. SAFER: Stuck-at-fault error recovery for memories. *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, p. 115–124, 2010. ISSN 10724451. Citado na página 22.

Solid State Technology. *Are the major DRAM suppliers stunting DRAM demand?* 2018. https://electroiq.com/2018/03/are-the-major-dram-s p. Disponível em: <https://electroiq.com/2018/03/are-the-major-dram-suppliers-stunting-dram-demand/>. Citado na página 14.

Turbocats. TCE3200LP - User Manual. p. 101, 2017. ISSN 0028-0836. Citado 2 vezes nas páginas 59 e 60.

WANG, C.-w. et al. A Built-In Self-Test and Self-Diagnosis Scheme for Embedded SRAM. p. 45–50, 2000. Citado 3 vezes nas páginas 20, 21 e 22.

WU, C.-F.; HUANG, C.-T.; WU, C.-W. RAMSES: a fast memory fault simulator. *Defect and Fault Tolerance in VLSI Systems, 1999. DFT '99. International Symposium on*, p. 165–173, 1999. ISSN 10636722. Citado na página 29.

Apêndices

APÊNDICE A – TABELAS DE CONFIGURAÇÃO DOS *MODE REGISTERS* DA MEMÓRIA DDR4 X16 96FBGA

Mode Register MR0:

Configuração *Mode Register* MR0.

Endereço	A17	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Binário	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0
Hexadecimal	0x0304														

Fonte: Elaborado pelo autor.

Descrição das configurações no *Mode Register* MR0.

Bits	**Valor (Binário)**	**Descrição**
A11:A9	001	Write Recovery (WR) = 12 cycles / Read to Precharge (RTP) = 6 cycles
A8	1	DLL Reset = Sim
A7	0	Test Mode (TM) = Normal
A6:A4,A2	0001	CAS Latency = 10
A3	0	Read Burst Type = Sequential
A1:A0	00	Burst Lenght = 8 (fixed)

Fonte: Elaborado pelo autor.

Mode Register MR1:

Configuração *Mode Register* MR1.

Endereço	A17	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Binário	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
Hexadecimal	0x0100														

Fonte: Elaborado pelo autor.

Descrição das configurações no *Mode Register* MR1.

Bits	**Valor (Binário)**	**Descrição**
A12	0	Qoff = output buffers enabled
A11	0	TDQS = disabled
A10:A8	001	$\text{RTT}_{NOM} = RZQ/4$
A7	0	Write Leveling = disabled
A6:A5	00	RFU
A4:A3	00	Additive Latency = disabled
A2:A1	00	Output Driver Impedance Control = RZQ/7
A0	0	DLL = Disabled

Fonte: Elaborado pelo autor.

Mode Register MR2:

Configuração *Mode Register* MR2.

Endereço	A17	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Binário	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Hexadecimal	0x0000														

Fonte: Elaborado pelo autor.

Descrição das configurações no *Mode Register* MR2.

Bits	**Valor (Binário)**	**Descrição**
A12	0	Write CRC = disabled
A11	0	RFU
A10:A9	00	$\text{RTT}_WR = DynamicODTOff$
A8	0	RFU
A7:A6	00	Low Power Auto Self-Refresh (LP ASR) = Manual Mode (Normal Temp. Range)
A5:A3	000	CAS Write Latency (CWL) = 9 (Speed 1600Gbps)
A2:A0	00	RFU

Fonte: Elaborado pelo autor.

Mode Register MR3:

Configuração *Mode Register* MR3.

Endereço	A17	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Binário	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Hexadecimal	0x0000														

Fonte: Elaborado pelo autor.

Descrição das configurações no *Mode Register* MR3.

Bits	**Valor (Binário)**	**Descrição**
A12:A11	00	MPR Read Format = Serial
A10:A9	00	WCL when CRC and DM are both enabled = 4nCK (1600)
A8:A6	000	Fine Granularity Refresh Mode = Normal (Fixed 1x)
A5	0	Temperature Sensor Readout = disabled
A4	0	Per DRAM Addressability = disabled
A3	0	Geardown Mode = 1/2 rate
A2	0	MPR Operation = Normal
A1:A0	00	MPR Page Selection = Page 0

Fonte: Elaborado pelo autor.

Mode Register MR4:

Configuração *Mode Register* MR4.

Endereço	A17	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Binário	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Hexadecimal	0x0000														

Fonte: Elaborado pelo autor.

Descrição das configurações no *Mode Register* MR4.

Bits	**Valor (Binário)**	**Descrição**
A13	0	hPPR = disabled
A12	0	Write Preamble = 1nCK
A11	0	Read Preamble = 1nCK
A10	0	Read Preamble Training Mode = disabled
A9	0	Self Refresh Abort = disabled
A8:A6	000	CAL (CS to CMD/ADDR Latency Mode) = disabled
A5	0	sPPR = disabled
A4	0	Internal Vref Monitor = disabled
A3	0	Temperature Controlled Refresh Mode = disabled
A2	0	Temperature Controlled Refresh Range = normal
A1	0	Maximum Power Down Mode = disabled
A0	0	RFU

Fonte: Elaborado pelo autor.

Mode Register MR5:

Configuração *Mode Register* MR5.

Endereço	A17	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Binário	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
Hexadecimal	0x0400														

Fonte: Elaborado pelo autor.

Descrição das configurações no *Mode Register* MR5.

Bits	**Valor (Binário)**	**Descrição**
A12	0	Read DBI = disable
A11	0	Write DBI = disable
A10	1	Data Mask = enable
A9	0	CA parity Persistent Error = disabled
A8:A6	000	$\mathrm{RTT}_{PARK} = disabled$
A5	0	ODT Input Buffer during Power Down Mode = activated
A4	0	C/A Parity Error Status = clear
A3	0	CRC Error Clear = clear
A2:A0	000	C/A Parity Latency Mode = disabled

Fonte: Elaborado pelo autor.

Mode Register MR6:

Configuração *Mode Register* MR6.

Endereço	A17	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Binário	0	0	0	0	1	0	0	0	0	0	0	1	1	1	0
Hexadecimal	0x040E														

Fonte: Elaborado pelo autor.

Descrição das configurações no *Mode Register* MR6.

Bits	**Valor (Binário)**	**Descrição**
A12:A10	001	$\text{tCCD}_L min = 5nCK; tDLLK min = 597nCK(1600e1866)$
A9:A8	00	RFU
A7	0	VrefDQ Training = disabled
A6	0	VrefDQ Training Range = Range 1
A5:A0	001110	VrefDQ Training Value = 69,10%

Fonte: Elaborado pelo autor.

APÊNDICE B – TABELA DO *PIN ASSIGNMENT* DO DSA DDR4 X16 96FBGA

Nome do Pino	Posição do Pino	Canal DUT1	Canal DUT2
A0	P3	a_5	a_6
A1	P7	b_5	b_6
A2	R3	a_7	b_8
A3	N7	b_7	b_8
A4	N3	a_13	a_14
A5	P8	b_13	b_14
A6	P2	a_15	a_16
A7	R8	b_15	b_16
A8	R2	a_19	a_20
A9	R7	b_19	b_20
A10/AP	M3	a_21	b_22
A11	T2	b_21	b_22
A12/BC_n	M7	a_23	a_24
A13	T8	b_23	b_24
A14/WE	L2	a_25	a_26
A15/CAS_n	M8	b_25	b_26
A16/RAS_n	M3	a_27	b_28
CK_t	K7	b_27	b_28
CK_c	K8	a_29	a_30
CKE	K2	b_29	b_30
CS0_n	L7	a_31	a_32
ACT_n	L3	b_31	b_32
BA0	N2	a_37	a_38
BA1	N8	b_37	b_38
BG0	M2	a_39	a_40
BG1	M9	a_53	a_24
ODT	K3	b_39	b_40
PARITY	T3	a_45	a_46
RESET_n	P1	b_45	b_46
DQL0	G2	a_1	a_2
DQL1	F7	b_1	b_2
DQL2	H3	a_3	a_4

DQU3	C7	b_35	b_36
DQU4	C2	a_41	a_42
DQU5	C8	b_41	b_42
DQU6	D3	a_43	a_44
DQU7	D7	a_43	b_44
DQSL_t	G3	a_17	a_18
DQSL_c	F3	b_17	b_18
DQSU_t	B7	a_49	a_50
DQSU_c	A7	b_49	b_50
DML_n/DBI_t	E7	a_47	a_48
DMU_n/DBI_t	E2	b_47	b_48
VREFCA	M1	a_hv1	b_hv1
DQL3	H7	a_1	a_2
DQL4	H2	a_9	a_10
DQL5	H8	b_9	b_10
DQL6	J3	a_11	a_12
DQL7	J7	b_11	b_12
DQU0	A3	a_33	a_34
DQU1	B8	b_33	b_34
DQU2	B8	b_33	b_34
ALERT	P9	a_51	a_52
TEN	N9	b_51	b_52
ZQ	F9	240ohm no GND	240ohm no GND
VDD	*	a_dps1a	b_dps1a
VDDQ	*	a_dps3a	b_dps3a
VPP	B1, R9	a_dps4a	b_dps4a
VSS/Q	*	GND	GND

Fonte: Adaptado de Hynix (2017).

Pinos de alimentação:

* *VDD: D1,J1,L1,R1,B3,G7,B9,J9,L9,T9 (10 pinos)*
* *VDDQ: A1,C1,G1,F2,J2,F8,J8,A9,D9,G9 (10 pinos)*
* *VSS: B2, E1,E9,G8,K1,K9,M9,N1,T1 (9 pinos)*
* *VSSQ: A2,A8,C9,D2,D8,E3,E8,F1,H1,H9 (10 pinos)*

APÊNDICE C – EXEMPLO DE CÓDIGO DE *PIN SCRAMBLE* PARA O COMANDO MRS

```
SCRAMBLE_MAP( PS3 ) //MRS (Mode Register Set) command
{
SCRAMBLE( CKt,t_cs1 )
SCRAMBLE( CKc,t_cs2 )
SCRAMBLE( RESETn,t_drive_high )
SCRAMBLE( CKE,t_drive_high )
SCRAMBLE( CSn,t_drive_low )
SCRAMBLE( ACTn,t_drive_high )
SCRAMBLE( RASn_A16,t_drive_low )
SCRAMBLE( CASn_A15,t_drive_low )
SCRAMBLE( WEn_A14,t_drive_low )
SCRAMBLE( ODT,t_drive_low )
SCRAMBLE( BG0,t_d18 )
SCRAMBLE( BA0,t_d17 )
SCRAMBLE( BA1,t_d16 )
SCRAMBLE( A13,t_drive_low )
SCRAMBLE( A12_BCn,t_d14 )
SCRAMBLE( A11,t_d11 )
SCRAMBLE( A10_AP,t_d10 )
SCRAMBLE( A9,t_d9 )
SCRAMBLE( A8,t_d8 )
SCRAMBLE( A7,t_d7 )
SCRAMBLE( A6,t_d6 )
SCRAMBLE( A5,t_d5 )
SCRAMBLE( A4,t_d4 )
SCRAMBLE( A3,t_d3 )
SCRAMBLE( A2,t_d2 )
SCRAMBLE( A1,t_d1 )
SCRAMBLE( A0,t_d0 )
SCRAMBLE(   PARITY,t_drive_low )
SCRAMBLE(   DQL0,t_tri_state )
SCRAMBLE(   DQL1,t_tri_state )
SCRAMBLE(   DQL2,t_tri_state )
SCRAMBLE(   DQL3,t_tri_state )
SCRAMBLE(   DQL4,t_tri_state )
SCRAMBLE(   DQL5,t_tri_state )
SCRAMBLE(   DQL6,t_tri_state )
SCRAMBLE(   DQL7,t_tri_state )
SCRAMBLE(   DQU0,t_tri_state )
SCRAMBLE(   DQU1,t_tri_state )
SCRAMBLE(   DQU2,t_tri_state )
SCRAMBLE(   DQU3,t_tri_state )
SCRAMBLE(   DQU4,t_tri_state )
SCRAMBLE(   DQU5,t_tri_state )
SCRAMBLE(   DQU6,t_tri_state )
SCRAMBLE(   DQU7,t_tri_state )
SCRAMBLE(   DQSLt,t_tri_state )
SCRAMBLE(   DQSLc,t_tri_state )
SCRAMBLE(   DQSUt,t_tri_state )
SCRAMBLE(   DQSUc,t_tri_state )
SCRAMBLE(   DMLn_DBILn,t_tri_state )
SCRAMBLE(   DMUn_DBIUn,t_tri_state )
```

```
SCRAMBLE(   ALERTn,t_tri_state )
SCRAMBLE(   TEN,t_drive_low )
}
```

Fonte: Elaborado pelo autor.

APÊNDICE D – CÓDIGO DO *LOOP* DE ESCRITA DO ALGORITMO *CHECKERBOARD*

```
//Write Loop Start

%WR_act_command:
xalu xmain,hold,dxmain,oxmain
dutctrl 0x0001
pinfunc tset2, ps4
mar inc

%WR_tRCD_loop:
dutctrl 0x0001
pinfunc tset2, ps5
count count1,decr,aon
mar cjmpnz, WR_tRCD_loop

//Burst Column Write between Refreshes Loop Start

%WR_command:
xalu xmain,hold,dxmain,oxmain
yalu ymain,ybase,add,dymain,oymain
dutctrl 0x0001
pinfunc tset2, ps6
mar inc

%WR_WL_loop:
dutctrl 0x0001
chips cs6f, cs7f
pinfunc tset2, ps7, adhiz
count count2,decr,aon
mar cjmpnz, WR_WL_loop

%WR_tWPRE:
dutctrl 0x0001
chips cs6f, cs7f
pinfunc tset2, ps7, adhiz
mar inc

%WR_8BL_loop:
dutctrl 0x0001
chips cs6f, cs7f
datgen bckfen,mainbase
pinfunc tset2, ps7
count count3, decr, aon
mar cjmpnz, WR_8BL_loop

%WR_REF_counter:
dutctrl 0x0001
pinfunc tset2, ps5
count count4,decr, aon
mar cjmpnz,WR_command

//Burst Column Write between Refreshes Loop End
```

```
%WR_sub_pre_ref_1:
dutctrl 0x0001
pinfunc tset2, ps5
mar gosub, s_precharge_xmain_refresh

%WR_sub_pre_ref_2:
dutctrl 0x0001
pinfunc tset2, ps5
mar gosub, s_precharge_xmain_refresh

%WR_row_counter:
xalu xmain,increment,con,dxmain,oxmain
dutctrl 0x0001
pinfunc tset2, ps5
count count9,decr, aon
mar cjmpnz,WR_act_command

//Write Loop End
```

Fonte: Elaborado pelo autor.

APÊNDICE E – CÓDIGO DO 5º *LOOP* (L5) DE ESCRITAS/LEITURAS DO ALGORITMO *MARCH A*

```
//L5 Loop Start: dn_rd0_wr1_wr0

%L5_act_command:
xalu xmain,hold,dxmain,oxmain
dutctrl 0x0001
pinfunc tset2, ps4
mar inc

%L5_tRCD_loop:
dutctrl 0x0001
pinfunc tset2, ps5
count count1,decr,aon
mar cjmpnz, L5_tRCD_loop

%L5_reset_dbase:
datgen sudata,udatadr,dbase
udata 0x002000000
dutctrl 0x0001
pinfunc tset2, ps5
mar inc

%L5_sub_rd0_1:
dutctrl 0x0001
pinfunc tset2, ps5
mar gosub, s_rd0_dmain_xmain_ymain

%L5_sub_wr1_1:
dutctrl 0x0001
pinfunc tset2, ps5
mar gosub, s_wr1_dmain_dbase_xmain_ymain

%L5_sub_wr0_1:
dutctrl 0x0001
pinfunc tset2, ps5
mar gosub, s_wr0_dmain_dbase_xmain_ymain

%L5_dbase_reset_counter:
yalu ymain,decrement,bon,dymain
datgen sdbase,shrdr,dbase
dutctrl 0x0001
pinfunc tset2, ps5
count count12,decr,aon
mar cjmpnz,L5_sub_rd0_1

%L5_REF_counter:
dutctrl 0x0001
pinfunc tset2, ps5
count count11,decr,aon
mar cjmpnz,L5_reset_dbase

%L5_sub_pre_ref:
dutctrl 0x0001
```

```
pinfunc tset2, ps5
mar gosub, s_precharge_xmain_refresh

%L5_column_row_counter:
xalu xmain,decrement,bmeqmax,dxmain,cfromy
dutctrl 0x0001
pinfunc tset2, ps5
count count10,decr, aon
mar cjmpnz,L5_act_command

//L5 Loop End: dn_rd0_wr1_wr0
```

Fonte: Elaborado pelo autor.

APÊNDICE F – PARTE DOS RESULTADOS DE TESTE DO ALGORITMO *CHECKERBOARD*

```
+-------------------------------------------------------------------------------------------------------+
 Pass/Fail   Test    Test                  Pattern               DUT #    Per-Pin
 Status      Number  Name                  Name                           Status
+-------------------------------------------------------------------------------------------------------+
 PASS        18      tb_checkerboard       p_checkerboard        1        (DQL0)PPPPPPPPPPPPPPPP(DQU7)
 PASS        18      tb_checkerboard       p_checkerboard        2        (DQL0)PPPPPPPPPPPPPPPP(DQU7)

DUT 1 (PASS):
DUT 2 (PASS):


+-------------------------------------------------------------------------------------------------------+
 Pass/Fail   Test    Test                  Pattern               DUT #    Per-Pin
 Status      Number  Name                  Name                           Status
+-------------------------------------------------------------------------------------------------------+
 PASS        20      tb_checkerboard       p_checkerboard        1        (DQL0)PPPPPPPPPPPPPPPP(DQU7)
 PASS        20      tb_checkerboard       p_checkerboard        2        (DQL0)PPPPPPPPPPPPPPPP(DQU7)

DUT 1 (PASS):
DUT 2 (PASS):

TestBlock = tb_checkerboard Complete
tb_checkerboard:  Time = 106.804055 Sec
```

Fonte: Elaborado pelo autor.

APÊNDICE G – PARTE DOS RESULTADOS DE TESTE DOS ALGORITMOS *MARCH A* E *MARCH LA*

```
+-------------------------------------------------------------------------------------------------------------+
 Pass/Fail   Test    Test                  Pattern               DUT #      Per-Pin
 Status      Number  Name                  Name                             Status
+-------------------------------------------------------------------------------------------------------------+
 PASS        8       tb_marchA             p_marchA              1          (DQL0)PPPPPPPPPPPPPPPP(DQU7)
 PASS        8       tb_marchA             p_marchA              2          (DQL0)PPPPPPPPPPPPPPPP(DQU7)

DUT 1 (PASS):
DUT 2 (PASS):


+-------------------------------------------------------------------------------------------------------------+
 Pass/Fail   Test    Test                  Pattern               DUT #      Per-Pin
 Status      Number  Name                  Name                             Status
+-------------------------------------------------------------------------------------------------------------+
 PASS        10      tb_marchA             p_marchA              1         (DQL0)PPPPPPPPPPPPPPPP(DQU7)
 PASS        10      tb_marchA             p_marchA              2         (DQL0)PPPPPPPPPPPPPPPP(DQU7)

DUT 1 (PASS):
DUT 2 (PASS):

TestBlock = tb_marchA       Complete
tb_marchA:  Time = 3290.501642 Sec
Running TestBlock = tb_marchLA

+-------------------------------------------------------------------------------------------------------------+
 Pass/Fail   Test    Test                  Pattern               DUT #      Per-Pin
 Status      Number  Name                  Name                             Status
+-------------------------------------------------------------------------------------------------------------+
 PASS        2       tb_marchLA            p_marchLA             1         (DQL0)PPPPPPPPPPPPPPPP(DQU7)
 PASS        2       tb_marchLA            p_marchLA             2         (DQL0)PPPPPPPPPPPPPPPP(DQU7)

DUT 1 (PASS):
DUT 2 (PASS):


+-------------------------------------------------------------------------------------------------------------+
 Pass/Fail   Test    Test                  Pattern               DUT #     Per-Pin
 Status      Number  Name                  Name                            Status
+-------------------------------------------------------------------------------------------------------------+
 PASS        4       tb_marchLA            p_marchLA             1         (DQL0)PPPPPPPPPPPPPPPP(DQU7)
 PASS        4       tb_marchLA            p_marchLA             2         (DQL0)PPPPPPPPPPPPPPPP(DQU7)

DUT 1 (PASS):
DUT 2 (PASS):
```

Fonte: Elaborado pelo autor.