Niago Moreira Nobre Leite

Relatório Final de Estágio
NXP Semiconductors

Campina Grande, Brasil

Julho de 2020

Niago Moreira Nobre Leite

Relatório Final de Estágio
NXP Semiconductors

Relatório final de estágio integrado submetido à Coordenação de Curso de Graduação de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Engenharia Elétrica.

Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática - CEEI
Departamento de Engenharia Elétrica - DEE

Aluno
Niago Moreira Nobre Leite

Professor Orientador
Gutemberg Gonçalves dos Santos Júnior, DSc.

Campina Grande, Brasil
Julho de 2020

Agradecimentos

A completude desse estágio emoldura um momento de intenso aprendizado nesta caminhada que traço. Diversas pessoas me deram forças para que eu lutasse diariamente contra minhas próprias barreiras internas antes de enfrentar de fato as externas.

Agradeço com bastante carinho e nomeio as seguintes presenças na NXP que me ensinaram muito não só dentro dos campos técnicos, mas dentro de uma ética de trabalho duro, de dedicação individual e coletiva, e de uma trajetória que me foi referência em diversos aspectos: Carol, Hugo e Vitor, além de um agradecimento geral a todo o brilhante time de MTR, que funciona como um conjunto de engrenagens impecavelmente encaixadas.

Por fim, agradeço aos professores Eanes, Edmar, Gutemberg e Marcos Morais que me apresentaram e/ou me prepararam, direta ou indiretamente, para mais um caminho que pude percorrer dentro da engenharia.

Lista de ilustrações

Sumário

1 Introdução

Este relatório final abrange as atividades de estágio desenvolvidas entre 10 de julho de 2019 e 30 de junho de 2020 na sede brasileira da NXP Semiconductors.

A microeletrônica é uma das grandes máquinas motoras responsáveis pelos avanços tecnológicos realizados pela humanidade. Ainda que para a maioria das pessoas, os portentos da microeletrônica sejam percebidos apenas diante de objetos como computadores pessoais e dispositivos portáteis, esta área participa de muitas outras categorias de produtos e aparelhos que utilizem da eletrônica em algum aspecto. Isto posto, a indústria da microeletrônica é responsável por girar um parcela considerável da economia mundial e é composta de diversas empresas que atuam em seus numerosos segmentos, como a automobilística, a infraestrutura de sistemas de informação e as comunicações digitais.

Sistemas embarcados incluem-se como alguns dos agentes mais significativos dentro de produtos que envolvem a microeletrônica. Computadores dedicados à execução de tarefas específicas e capazes de serem produzidos aos milhões de unidades a baixo custo, são comumente encontrado na forma de *chips* encapsulados como na Figura 1. Os circuitos eletrônicos integrados que compõem *chips* como este podem conter sistemas inteiros com memórias, unidades de processamento e periféricos, e assim recebem o nome de Sistemas em Chip (*Systems on Chip*, SoCs).

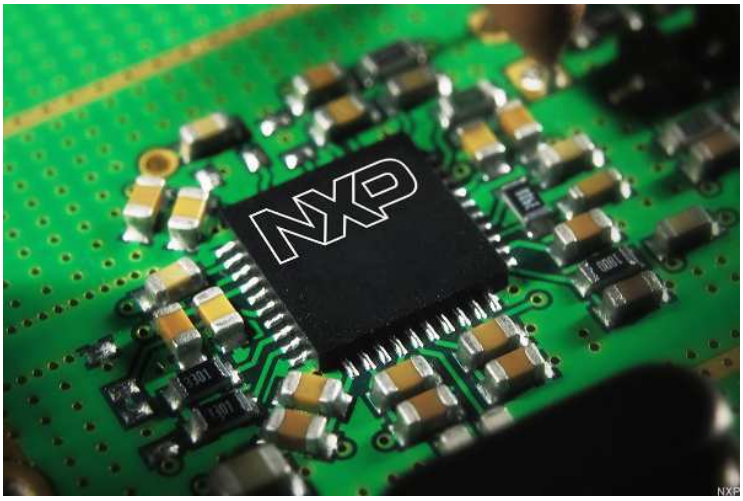


Figura 1 – Exemplo de *chip* encapsulado integrado a um circuito maior.

1.1 Sobre a NXP Semiconductors

O projeto de Sistemas em Chip movimenta bilhões de dólares anualmente e dada sua enorme diversidade para uma extensa gama de aplicações, esses elementos ubíquos acabam por constituir e dirigir um ramo inteiro de concepção, projeto e fabricação para atender às demandas mundiais.

A NXP Semiconductors (ou simplesmente NXP) é uma empresa de semicondutores dos Países Baixos e insere-se nesse contexto, atuando em diversos ramos da indústria de microeletrônica, sendo líder em soluções automobilísticas, de autenticação segura e comunicações digitais, e provendo serviços e produtos para empresas como Apple, Bosch, Huawei, Nokia Networks, Panasonic e Samsung. Tem escritórios em trinta e cinco países, com 11200 engenheiros em trinta e três deles. Assim como Qualcomm, AMD, Marvell, Analog Devices, Intel, Infineon e Microchip, a NXP atua desenvolvendo dispositivos semicondutores digitais, analógicos e de sinais mistos. Suas *business lines* incluem automobilística, computação dedicada, segurança de informação, radiofrequência e aplicações multimídia.

A NXP é derivada da subdivisão de semicondutores da Philips, desmembrada em 2006. Desde então, realizou a aquisição da Freescale Semiconductor, em 2015, e em 2018, passou por um processo de aquisição pela Qualcomm, mas que ao final não foi concluído. A sede brasileira da NXP chama-se *Brazil Semiconductor Technology Center* (BSTC), e traça história desde quando sob direção da Motorola no final da década de 1990. A partir de 2004, o BSTC esteve sob direção da Freescale, e depois, da NXP. Atualmente, o BSTC conta com mais de 130 funcionários, em sua maioria engenheiros, e atua em projetos colaborativos internacionais, bem como projetos desenvolvidos inteiramente no Brasil, desde microcontroladores, dispositivos para processamento de sinais, a gerenciamento de energia e automotivos.

O presente trabalho segue com as seções: Objetivos do Estágio (Seção 2); Fundamentação Teórica (Seção 3); Atividades Desempenhadas (Seção 4) e Considerações Finais (Seção 5).

2 Objetivos do Estágio

O termo de compromisso de estágio compreende três fases: treinamento em metodologias e ferramentas de projeto; conhecimento e treinamento nos módulos incumbidos e participação no desenvolvimento dos sistemas do time de trabalho. Os objetivos gerais do estágio são:

- Desenvolvimento de padrões de testes para realizar a verificação dos blocos lógicos devidos;
- Integrar-se ao Grupo MTR (*Memory Test and Repair*) de modo a atuar em projetos novos e existentes, adquirindo elasticidade para a compreensão de cronogramas já iniciados e capacidade para previsões e estimativas temporais de tarefas futuras.

Fase I: Treinamentos em ferramentas e metodologias de projeto

- Ferramenta de controle de versões;
- Software utilizado para gerenciamento de Releases de Código, Bug Tracking e Requisição de Mudanças de Projeto, entre outros;
- Ferramenta utilizada para gerenciar ambiente de trabalho para dados em controle de versões;
- Sistema de execução e balanceamento de jobs em batch;
- Metodologia interna para atividades relativas a design. Inclui definição de diretrizes de codificação e padrões de barramento, entre outros;
- Padrão interno para definição de representação hierárquica de blocos de design, em diretórios;
- Ferramenta para gerenciamento de design e automação de ferramentas de EDA;
- SVA (*System Verilog Assertions*) - Linguagem para definição de assertivas;
- UVM (*Universal verification Methodology*) - Biblioteca de classes e metodologia de verificação;
- Ferramentas de simulação e depuração de projetos.

Fase II: Conhecimento e Treinamento nos diversos módulos que compõem o Sistema MTR (*Memory Test and Repair*)

Fase III: Participação no desenvolvimento de ambiente de verificação de um módulo do sistema MTR

- Implementação de monitor de barramento serial;
- Implementação de driver de barramento serial;
- Implementação de uma interface de programação para abstração de acesso a um dos módulos do sistema MTR;
- Implementação de um modelo de referência de um dos módulos do sistema MTR e de um sistema para comparação de dados;
- Implementação de testes e assertivas utilizando SVA (*System Verilog Assertions*).

Contudo, devido a mudanças de prioridade e cronograma no desenvolvimento de alguns projetos, a Fase III compreendeu adicionalmente, atividades de desenvolvimento e suporte em ambientes de verificação já maduros para outros módulos desenvolvidos pelo Grupo MTR.

3 Fundamentação Teórica

3.1 Fluxo da Microeletrônica

Dentro da microeletrônica, que utiliza a tecnologia de semicondutores (dentre os mais famosos, o elemento silício) para construção de circuitos eletrônicos integrados, é necessário que exista organização tal para lidar com a complexidade de seus processos produtivos. Estes processos envolvem dezenas de áreas da engenharia e milhares de especialistas, e possibilitam a construção de vários dos sistemas de engenharia que movimentam o mundo contemporâneo.

O chamado Fluxo da Microeletrônica é o termo ou jargão que se refere a todo o processo de execução de um projeto desses sistemas eletrônicos, de forma robusta, econômica e em prazos que garantam *Time to Market*. De modo geral, respeitando as divisões e alocações internas de cada empresa do ramo, desde a concepção à manufatura o fluxo segue uma lógica comum que tem se provado economicamente viável até então.

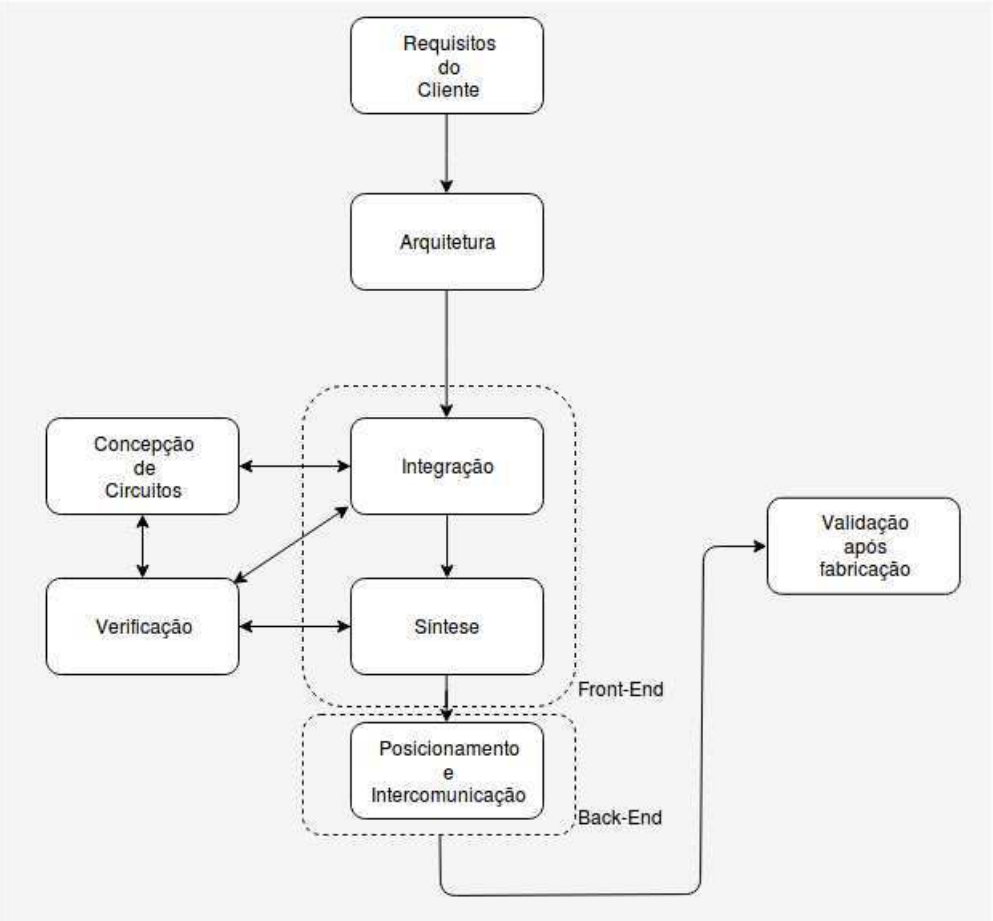


Figura 2 – Fluxo da microeletrônica.

Especificações de arquitetura são obtidas a partir de conversas com clientes ou baseadas em tendências de mercado, observando momentos de integração futura do produto a outros sistemas comercialmente disponíveis, como é o caso de processadores de uso geral.

Nesse contexto, arquitetura significa o conjunto de especificações técnicas e funcionais do sistema, como consumo de energia, frequência de operação, área, e presença de recursos como antenas, moduladores, módulos *bluetooth*, capacidade de processamento gráfico e unidades de aplicação.

As especificações são discutidas mais de perto com times de *design* de modo a decidir protocolos, sub-blocos e a organização funcional em diferentes níveis de granularidade. Os blocos podem ser desenvolvidos para um sistema exclusivo ou podem ter funcionalidade genérica de tal sorte a serem aproveitados em mais de um projeto. Normalmente, *design* e verificação atuam concomitantemente, de forma recursiva e interativa, e o ambiente de verificação evolui conforme novas versões do *design* vão sendo liberadas, corrigindo *bugs*, sofrendo modificações arquiteturais ou adicionando novos recursos.

Times de *design* descreverão os blocos-base, que podem ser exclusivos do *chip*, mas também podem ser reutilizados ou reaproveitados de projetos anteriores. Tais blocos, os IPs (*Semiconductor Intellectual Property Cores*), serão verificados para que seja garantida sua adequação funcional e estrutural ao serem integrados para elaborar o sistema final, e uma verificação a nível de sistema também será realizada. O processo de síntese converte esse modelo a nível de transistores, que serão posteriormente posicionados adequadamente com uso de bibliotecas de modelos físicos para gerar um leiaute do chip. Em seguida, essa planta-baixa é enviada a uma fábrica para manufatura em escala. Após gravado em silício, o *chip* retorna para etapas de testes e validação utilizando estímulos físicos, antes de finalmente seguir para comercialização.

3.2 O Grupo MTR

Dentre os grupos de desenvolvimento globais que cooperam dentro da NXP Semiconductors para concretizar os diversos produtos em seu portfólio, figura o Grupo *Memory Test and Repair* (doravante referido por MTR), que tem como principal objetivo o desenvolvimento de módulos para fornecer uma solução altamente configurável para teste e reparo das memórias embarcadas nos *chips*.

A necessidade de teste e reparo de memórias em SoCs decorre de seu notável aumento em número com o passar dos anos, bem como sua inserção cada vez maior em sistemas críticos, como de segurança ou subsistemas de automóveis. Naturalmente, esses fatores impactam no rendimento (*yield*) dos SoCs – se a qualidade estrutural e funcional dessas memórias não é assegurada após fabricação, o nível de descarte de pastilhas de silício não pode ser contido adequadamente e portanto, há um impacto econômico em larga escala que afeta retroativamente as áreas responsáveis por conceber o produto na empresa.

Existem diversas maneiras de melhorar o rendimento na produção de *chips*, dentre elas, o uso de técnicas de *Design for Testing* (DFT), que facilitam o testes de produção em silício, adicionam redundâncias adequadas à etapa de *design* e aumentam a garantia de ausência de problemas de fabricação nos produtos, que podem acarretar em problemas funcionais. Uma das técnicas de DFT para testes de memórias embarcadas é o uso de estruturas *Memory Built-In Self-Test* (MBIST), que encapsulam uma lógica que pode ser integrada juntamente às memórias e assegura seu teste, além de evitar que o acesso seja feito a partir do nível topo do *chip*. Um exemplo de arquitetura básica para acessar uma memória sob teste com o MBIST pode ser visualizada na Figura 3.

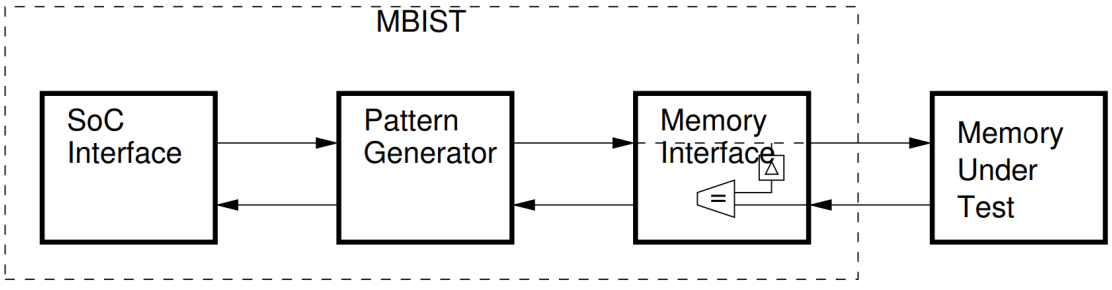


Figura 3 – Diagrama de blocos para arquitetura básica de MBIST.

Fonte: Silveira, Qureshi e Zeli (2018).

O Grupo MTR tem parte de seus integrantes no Brasil e parte nos Estados Unidos, e seus clientes são outros grupos internos da NXP, que desenvolvem seus respectivos módulos e sistemas para comercialização ou para integração com sistemas de outros grupos. Esses grupos então contatam o Grupo MTR com dados relevantes de projeto (por exemplo, especificações das memórias embarcadas, como tamanho, tecnologia, e complexidade do tipo de reparo, bem como as configurações desejadas do sistema MTR encomendado), que irá por sua vez desenvolver e verificar um ou mais módulos talhados especificamente para aquele sistema. Esse processo pode levar algumas iterações até sua versão final, pois frequentemente a quantidade, o posicionamento e a especificação das memórias de um *chip* são alterados entre o momento de primeiro contato com o Grupo MTR e a finalização da etapa de *front-end* do SoC.

3.2.1 O Sistema MTR

O Sistema MTR desenvolvido pelo grupo é constantemente remodelado para prover suporte a novas tecnologias e a sistemas mais complexos com um número de memórias crescente, bem como para adicionar novos recursos ao passo que se diminui área e tempo de teste final. Por causa disso, eventualmente as soluções do MTR passam por rearquiteturas, e a seguir serão explorados alguns detalhes da arquitetura mais recentemente publicada, em 2019, por integrantes do próprio grupo.

O MTR fornece uma solução customizável e flexível de acordo com as restrições de área e tempo de teste para um dado SoC, e permite o teste e reparo de diversos tipos de memórias RAM e ROM. O teste de uma memória consiste no uso de algoritmos de acesso, podendo ser tanto padrões da indústria quanto proprietárias desenvolvidas pelo próprio grupo, com o objetivo de identificar falhas de fabricação. O reparo automático subsequente usa de uma redundância estrutural para que essas falhas sejam contornadas apropriadamente.

Para que o reparo seja possível, as memórias contém células adicionais redundantes (*spare cells*) que podem ser utilizadas em substituição a células detectadas como defeituosas durante o teste. A depender da arquitetura e tecnologia de fabricação, as *spare cells* podem substituir *bits* individuais, colunas ou linhas inteiras das memórias. As técnicas de cálculo do tipo de reparo e sua aplicação para cada arquitetura de memória variam e podem ser executadas durante o teste ou após sua completude, porém um conceito geral válido é o que as células defeituosas têm seu endereço ignorado e as operações a elas dirigidas são redirecionadas para as células-reserva.

De modo geral, um teste de memória é realizado através de uma sequência de teste ou algoritmo de marcha (*march algorithm*), desenvolvido com o objetivo de identificar defeitos como *bits* não sendo escritos, *bits* presos (*stuck-at*), *bits* trocados entre posições adjacentes e falhas nos detectores de endereços.

Um algoritmo é composto por elementos (*march elements*, MEs), que por sua vez são conjuntos de operações ou fases (*march phases*, MPs) relativas a uma direção de percurso de endereços – depreende-se, pois, que o controle de um algoritmo pode ser modelado através de máquinas de estado finitas (*finite state machines*, FSMs) implementadas no RTL. As *march phases* representam etapas de operação lógica para as memórias.

Embora existam memórias de arquiteturas alternativas que apresentam operações lógicas intrínsecas às suas células, como ANDs e ORs, com a finalidade de proporcionar computação *in-memory* (Nair; Münch; Tahoori, 2020), serão aqui abordadas apenas as operações-base de escrita e leitura.

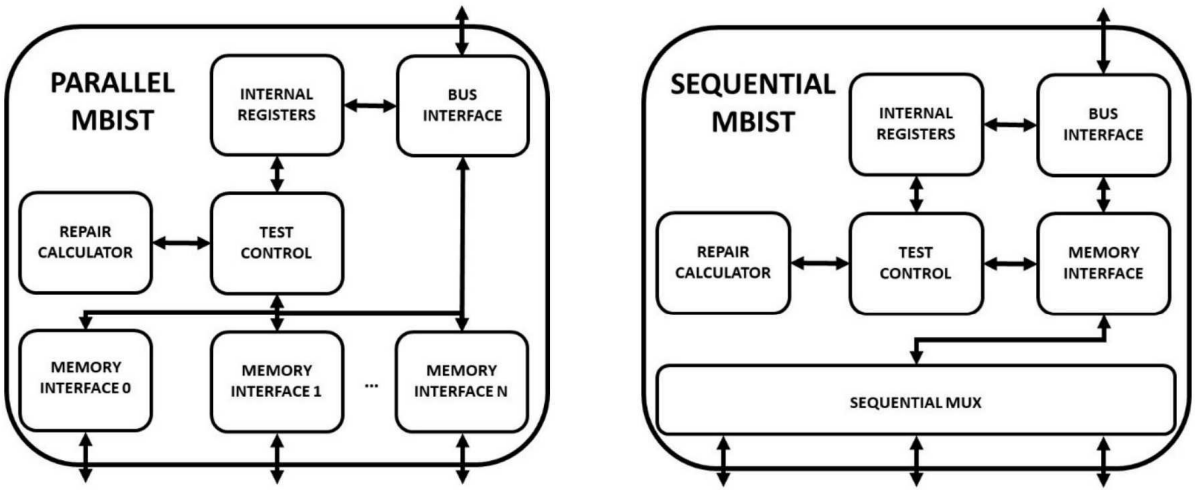
Cada ME possui uma direção, que diz respeito à sequência de endereços a serem percorridos (incrementando do menor ao maior, *up* (⇑); decrementando, do maior ao menor, *down* (⇓)), e uma sequência de operações (MPs) associada. Por exemplo, seja o algoritmo dado pela Expressão 1.

$$\Uparrow (W0); \quad \Downarrow (R0, W1, R1); \quad \Uparrow (R1) \tag{1}$$

A Expressão 1 denota um algoritmo de três MEs (o primeiro e o último com uma MP, e o segundo com três MPs), e pode ser traduzido da seguinte maneira:

1. Percorrendo do menor ao maior índice, escreva 0 em todos endereços da memória;
2. Percorrendo do maior ao menor índice, leia um valor esperado de 0, escreva 1, e leia um valor esperado de 1, em todos os endereços da memória;
3. Percorrendo do menor ao maior índice, leia um valor esperado de 1 em todos os endereços da memória.

Existem vários "sabores" (ou *flavors*) de MBIST providos pelo MTR para necessidades específicas de teste. É possível, por exemplo, testar memórias em paralelo (simultaneamente) ou de forma sequencial, conforme as Figuras 4a e 4b. Podem ainda ser usadas estruturas MBIST distintas para testar grupos de RAM e grupos de ROM, ou usar uma mesma estrutura para ambos, ou ainda, estruturas com protocolos mais simples de acesso para sistemas menos complexos, mas que ainda precisam de uma solução integrada de teste.



(a) (b)

Figura 4 – Diagrama de blocos para MBIST paralelo (a) e sequencial (b).

Fonte: Zeli, Silveira e Qureshi (2019).

No caso de MBIST paralelo, o tempo de teste cumulativo é igual ao pior caso, isto é, a memória com maior número de palavras. Para a arquitetura MBIST sequencial, que utiliza um multiplexador para selecionar individualmente as memórias em ordem de teste, o tempo de teste é proporcional à soma dos tempos de teste por memória. Naturalmente, para alguns casos, a arquitetura paralela pode se mostrar superior por condensar os tempos de teste, mas por apresentar maior área, um módulo sequencial pode ser sugerido para obter menor área, com pouco efeito negativo no tempo total de teste.

Salienta-se rapidamente que as características aqui tratadas para as memórias serão apenas aquelas relevantes do ponto de vista de abstração lógica, isto é, os endereços indexados, e o tamanho de dado (*data width*, DW). A Figura 5 ilustra uma dada matriz de células de memória, com atenção às linhas de endereços (vertical) e de dados (horizontal), bem como aos blocos lógicos decodificadores periféricos.

O custo de teste de memórias pode ainda ser otimizado concatenando memórias físicas para criar *gangs*, ou memórias lógicas (Miyazaki; Yoneda; Fujiwara, 2006), que são vistas pelo MBIST como uma única interface de memória. Permite-se assim que se evite o teste sequencial de todas as memórias físicas, mas sim de interfaces virtuais que englobam várias memórias. Além disso, é possível diminuir o excesso de área do módulo MBIST, já que ele é mais eficientemente compartilhado entre as memórias, incluindo suas informações de reparo. Essa associação pode ser feita apenas por conveniência de teste, ou por denotar uma associação funcional específica entre memórias, proposta no projeto.

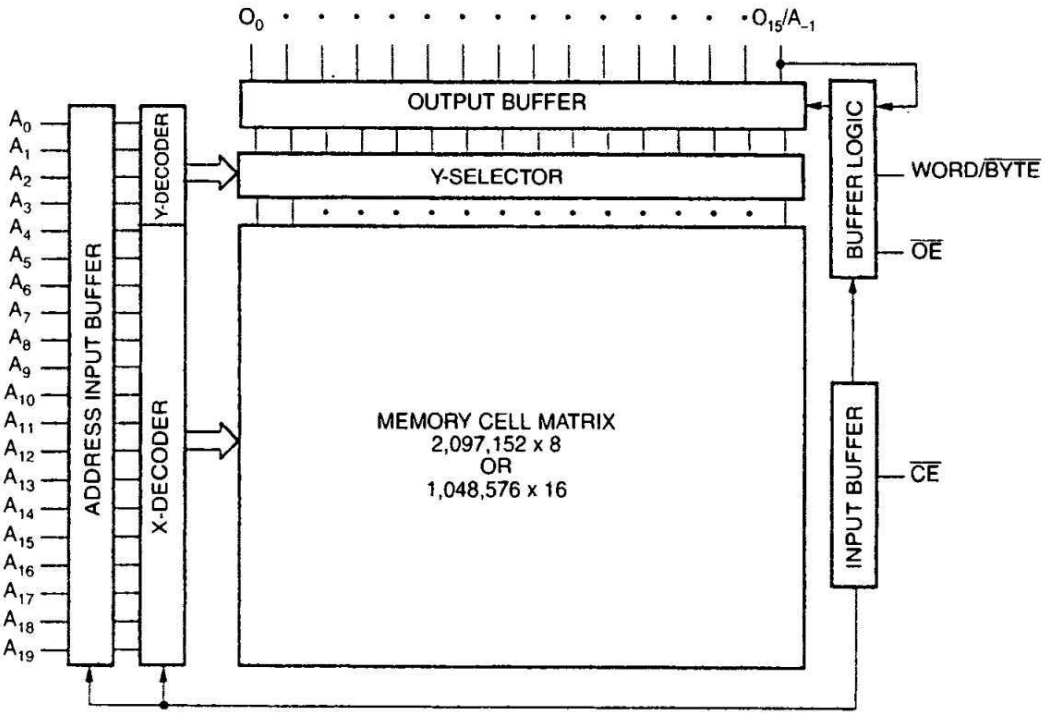


Figura 5 – Diagrama de célula de memória simples.

Fonte: Sharma e Council (1997).

Memórias de geometrias diferentes podem ser agrupadas de diversas formas, mas consideremos os casos de memórias de mesma geometria e alinhadas em endereços correspondentes a potências de 2, para evitar tratar as exceções de *dead space* (posições dentro da memória lógica que não correspondem a qualquer dado físico), também simplificando a lógica de decodificação de endereços por parte do MBIST.

Da Figura 5, é possível traçar as possibilidades imediatas de concatenação **vertical** (de modo a gerar uma memória lógica resultante com maior número de palavras) e **horizontal** (aumentando o tamanho do dado), representadas na Figura 6.

A Figura 7 ilustra um exemplo de fluxo de teste executado pelo MBIST para dadas interfaces de memória a serem selecionadas, fluxo este implementado de forma hierárquica com várias FSMs aninhadas, o que permite maleabilidade no uso de recursos e evita o desperdício de lógica e área ao usar uma única FSM para todas as possíveis combinações.

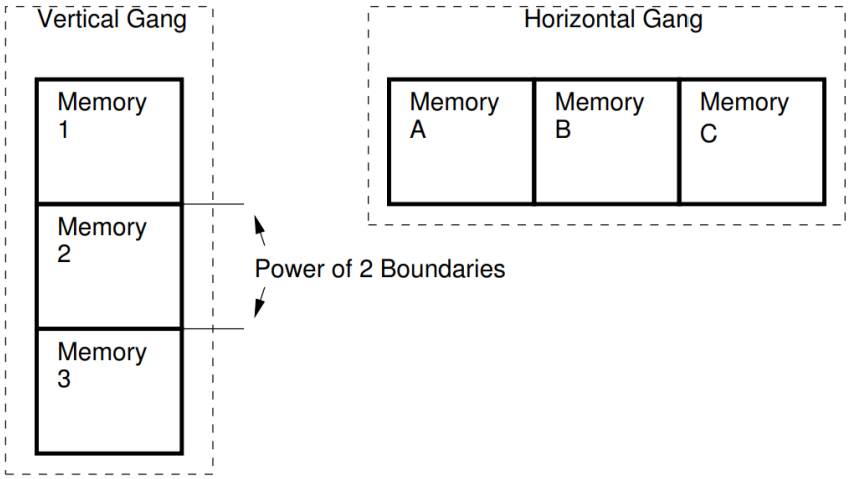


Figura 6 – Exemplos de concatenações de memórias (*ganging*).

Fonte: Silveira, Qureshi e Zeli (2018).

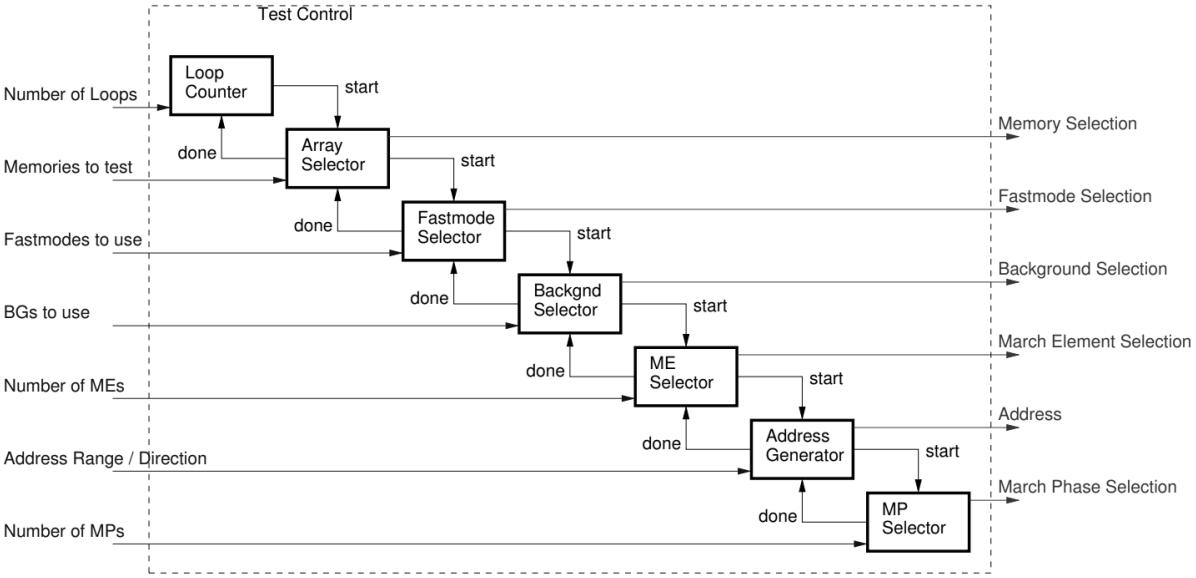


Figura 7 – Fluxo de teste realizado pelo módulo MBIST.

Fonte: Silveira, Qureshi e Zeli (2018).

A estrutura representada na Figura 7 é totalmente flexível e pode ser configurada conforme as necessidades do projeto; várias estruturas internas podem ser ativadas ou desativadas através de escritas a registro, por exemplo. Na mesma Figura, *Background* corresponde a um dado com quantidade determinada de *bits* a ser escrito em cada endereço (por exemplo, 16'bFFFF, o que preencheria as posições com *bits* 1); *Fastmodes* são a maneira que blocos de endereços são considerados para as operações de memória, por exemplo, selecionando colunas inteiras, linhas inteiras, ou blocos bidimensionais específicos.

4 Atividades Desempenhadas

As atividades desempenhadas estão descritas dentro de duas categorias: capacitação e desenvolvimento/inserção em projetos.

4.1 Treinamentos em ferramentas e metodologias de projeto

Esta etapa condensou-se basicamente no treinamento das ferramentas proprietárias de empresa, bem como no aprendizado dos conceitos gerais do fluxo de microeletrônica que viessem a ser relevantes para a execução das atividades previstas. O fato de que o estágio começou fora do período comum de entrada de estagiários na empresa (normalmente em janeiro) implicou em não haver treinamentos formais com aulas, de modo que muitos dos conceitos foram obtidos à medida em que se faziam necessários. Os treinamentos foram feitos estudando o material adequado passado, sob orientação do supervisor, que direcionava o estudo e tirava dúvidas gerais.

O sistema operacional utilizado nos computadores da empresa é Windows, no entanto a maior parte do trabalho é realizada em ambiente Linux, que roda em uma máquina virtual acessada de forma remota e hospedada externamente. Não houve treinamento para o uso de Linux, pois já eram suficientemente conhecidas as interações e comandos necessários a nível de usuário. Séries de apostilas internas para ferramentas proprietárias foram estudadas, juntamente com laboratórios de exercícios para ferramentas de controle de versões, gerenciamento de dados de produto e de ambiente de trabalho, uso do servidor de recursos computacionais, diretrizes de padronização de código e diretórios etc.

Todas essas ferramentas são desenvolvidas parcial ou integralmente pelo setor de *Design Enablement* da empresa. Para conhecimento de algumas metodologias e recursos de verificação, os manuais de usuário das bibliotecas UVM e para SVA foram usados, os quais são públicos.

4.2 Atividades de Verificação Funcional

As atividades de verificação previstas foram a nível de IP, isto é, de subsistemas e blocos lógicos isolados, e não a nível de SoC. A verificação a nível de SoC usualmente foca em checar, testar e estressar conexões, interrupções e protocolos de comunicação entre diferentes IPs, ao passo que se assume que os blocos lógicos individuais que compõem o SoC foram verificados e funcionam como previsto. A verificação a nível de IP, por outro lado, atua sob uma ótica de arquitetura e comportamento operacional de fato, de modo que segue um raciocínio de elaboração de testes de exaustão usando modelos funcionais de referência e verificação de entradas e saídas para garantir que o IP esteja agindo de acordo com sua especificação e possa ser integrado a um sistema maior.

4.2.1 Ambiente de Verificação de Controlador de MBIST

De início, um bloco que havia saído recentemente da primeira iteração inicial de *design* foi alocado para a verificação. Esse bloco constituía de um controlador para um dos *flavors* do módulo MBIST, para sistemas com uma necessidade de acesso específica. O módulo possui um acesso serial, a ser integrado a uma interface JTAG no nível do SoC junto a alguns pinos de comando do bloco, bem como registros internos para configuração e leitura dos módulos MBIST a ele conectados. A Figura 8 mostra, de maneira simplificada, como o módulo pode ser integrado a um sistema com a variante MBIST adequada.

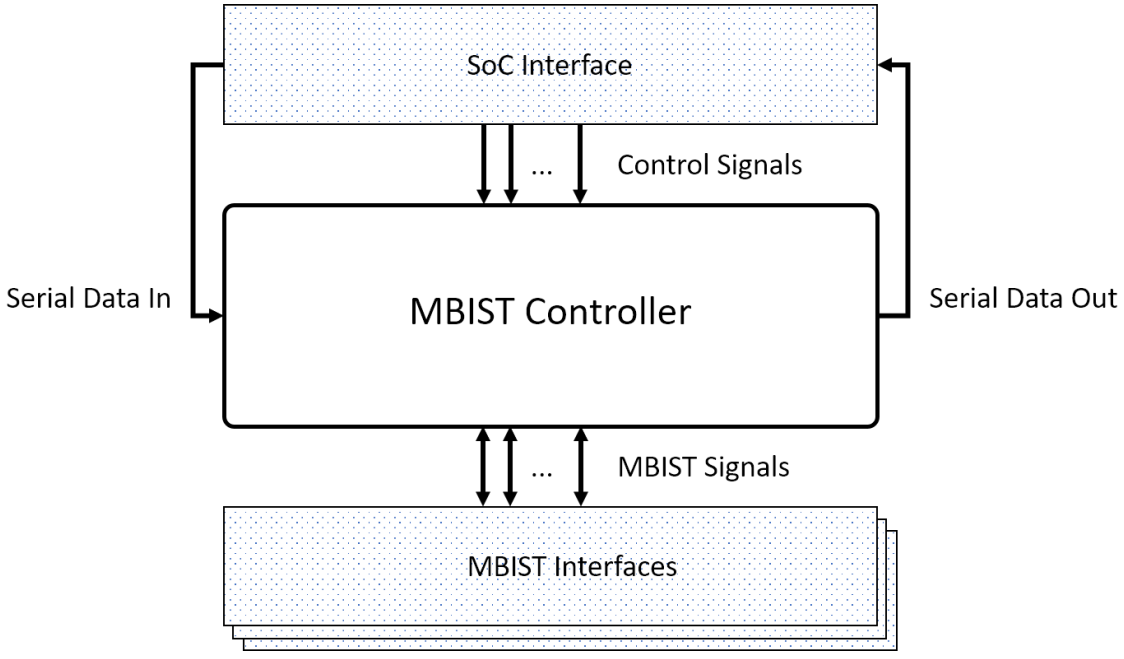


Figura 8 – Exemplo de integração do controlador de MBIST em sistema.

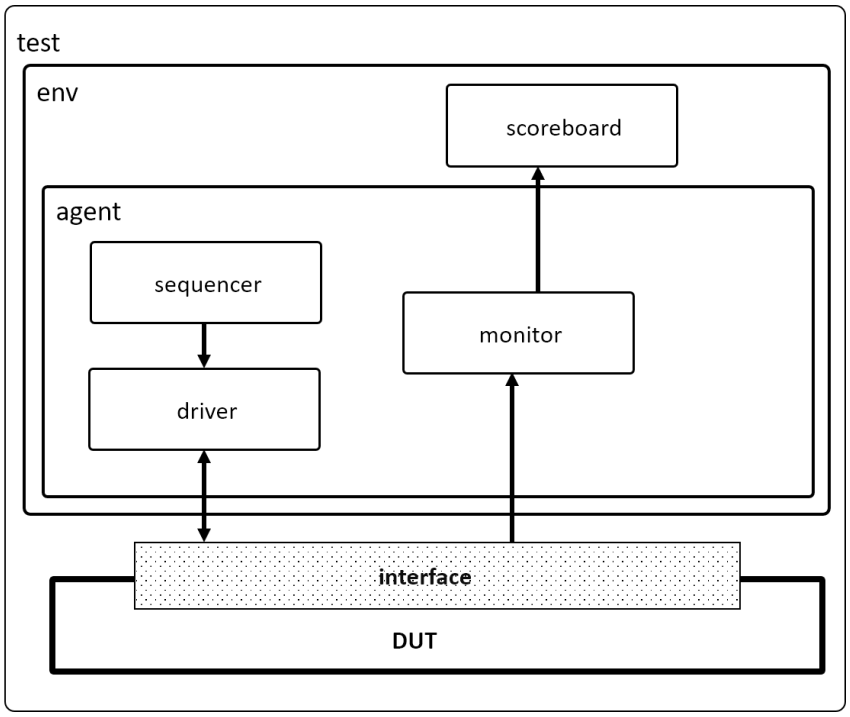
Fonte: O autor.

De início, um plano de verificação foi elaborado, identificando as características que seriam abordadas durante a verificação e estruturando de modo geral o escopo preliminar dos testes a serem desenvolvidos. Em seguida, um ambiente completo de verificação baseado em UVM teve de ser construído do zero, o que expôs a oportunidade de subtrair as várias dúvidas que apareceram durante a implementação, aprofundar o aprendizado sobre recursos e limitações da linguagem SystemVerilog e das bibliotecas UVM, e exercitar o desenvolvimento de um ambiente de verificação de forma integral. As Figuras 9a e 9b mostram, respectivamente, um ambiente generalista UVM com seus componentes, e uma versão simplificada do ambiente baseado em UVM para estímulo e observação do bloco lógico referido.

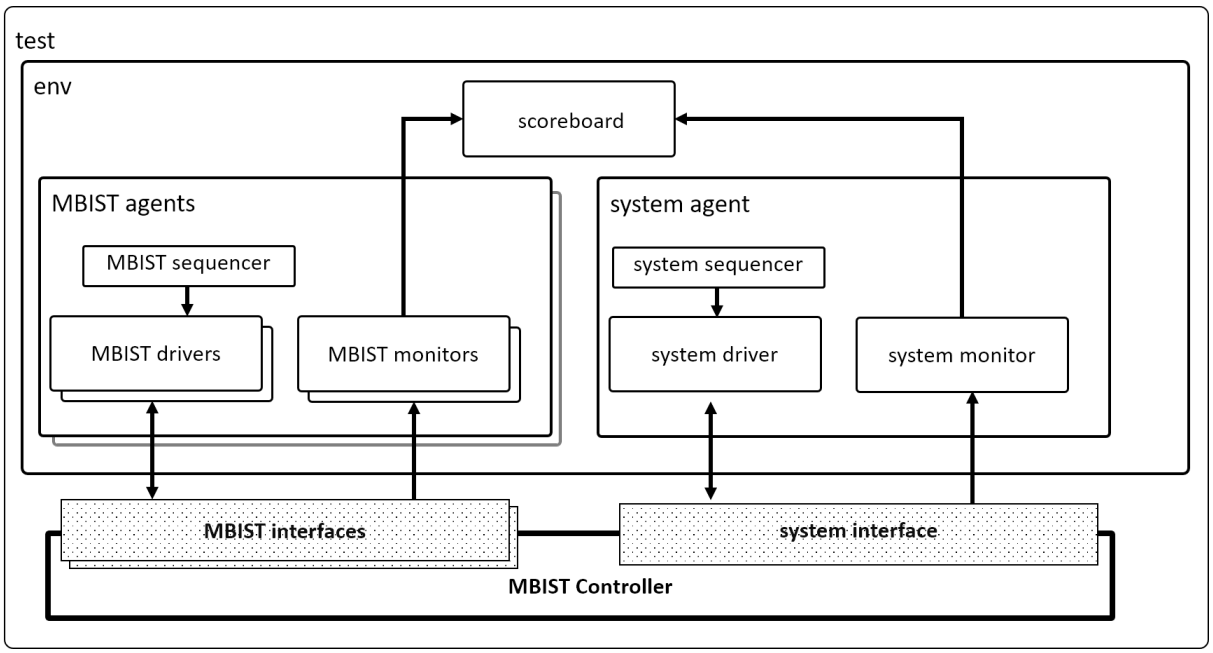
Comumente, um ambiente UVM usa conceitos da Programação Orientada a Objetos, como classes, herança, encapsulamento e polimorfismo, para se organizar em componentes (construídos uma vez em fase inicial da simulação) e objetos (criados e destruídos dinamicamente ao longo da simulação). Esses elementos são dispostos hierarquicamente de modo a prover estímulos aleatórios ou específicos ao Dispositivo Sob Teste (*Device Under Test*, DUT), assim como monitorá-lo. Essa sistemática promove automatização, reuso, escalabilidade e abstração funcional do DUT, e costuma ser eficaz no agrupamento de seus dados comportamentais para comparação automática ou manual nos testes desenvolvidos para cobrir as características desejadas.

Na Figura 9a, a interface provém e permite uma conexão entre classes e sinais físicos no RTL. A classe ambiente (*env*, de *environment*) instancia alguns componentes de verificação, dentre eles um ou mais agentes (*agents*) incumbidos de encapsular outros componentes que interagem entre si e são responsáveis por atividades específicas: um sequenciador (*sequencer*) gera e arbitra as transações desejadas a um *driver*, que as traduz em sinais de estímulo ao DUT, comunicando-se com ele através das interfaces; ainda, um monitor é responsável por observar as saídas do DUT e enviá-las para comparação no *scoreboard* com modelos de referência previamente estabelecidos. No caso do ambiente do controlador de MBIST (Figura 9b), essas interações são arranjadas através de instâncias que agem independentemente, tanto para a integração do bloco a nível SoC quanto a nível MBIST.

Foi possível desenvolver, além da infraestrutura de verificação do bloco lógico mencionado, um teste inicial de estímulo, que gerava entradas seriais aleatórias enquanto se monitoravam alguns pontos de acesso ao módulo de forma simultânea. A atividade de verificação deste bloco foi pausada neste momento, sendo o próximo passo planejado o desenvolvimento de modelos de referência para comparação dos dados monitorados e de um componente *scoreboard*, para conferência das transações geradas de forma aleatória ou específica no ambiente de verificação.



(a)



(b)

Figura 9 – Diagramas de ambiente de verificação UVM generalista (a) e para o controlador de MBIST (b).

Fonte: O autor.

4.2.2 Ambiente de Verificação de MBIST

Com mudanças no time a nível organizacional e também de cronograma, algumas atividades foram realocadas, de modo que o estagiário passou a atuar em um ambiente já em desenvolvimento contínuo há anos, e significativamente maior que o que estava sendo trabalhado e compreendido até então; o ambiente aludido é representado em alto-nível e de maneira simplificada na Figura 10. Vários desafios novos surgiram nessa etapa, e reuniões de *ramp-up*, bem como orientações ao longo do estágio, fizeram-se necessárias.

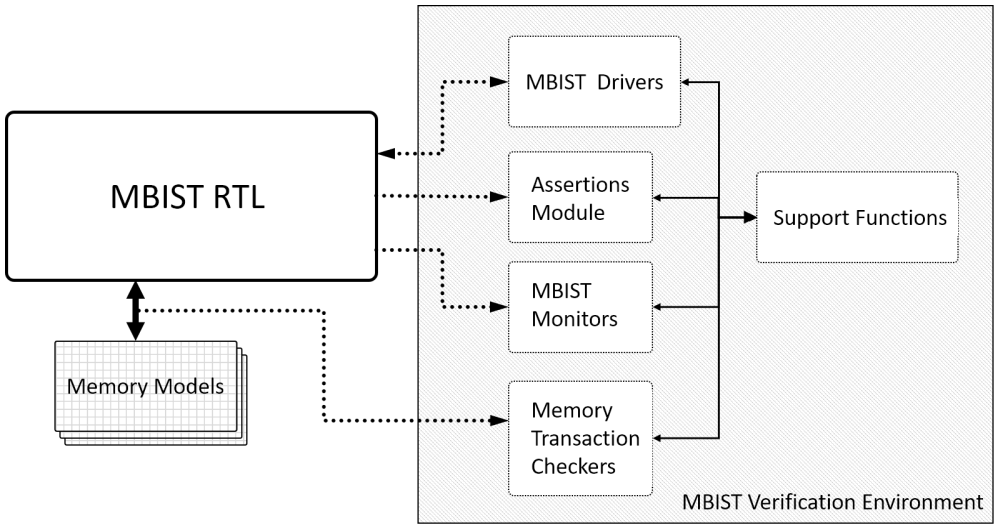


Figura 10 – Diagrama de ambiente de verificação para o módulo MBIST.

Fonte: O autor.

O ambiente exibido na Figura 10 mostra alguns componentes de verificação que interagem com o MBIST. Os *drivers* são encarregados de configurar pinos e registros para funções e modos de operação desejados de acordo com os testes desenvolvidos. Os monitores, o módulo de assertivas (*assertions module*) e os aferidores de transações de memória (*memory transaction checkers*) averiguam a integridade das características do DUT, do ponto de vista do seu comportamento interno, dos seus protocolos de comunicação e da fidedignidade das operações de teste e reparo por ele executadas. Todos os componentes e módulos interagem no ambiente com uma gama de funções de suporte que atuam, dentre outras frentes, no gerenciamento do fluxo de execução dos algoritmos de teste e contadores envolvidos, e na configuração e detecção de particularidades funcionais e/ou estruturais a depender do *flavor* de MBIST utilizado.

As tarefas que decorreram tiveram em seu foco, além da adição de suporte a novos recursos e simplificação no suporte a estruturas que foram reduzidas, o desenvolvimento de novos padrões de teste para alguns blocos que estavam em processo de re-arquitetura. Exemplos de mudanças incluem: modificações estruturais e funcionais no mapa de registros do módulo MBIST e o desenvolvimento de novo protocolo de comunicação entre o MBIST e os modelos de memória.

Os padrões de verificação desenvolvidos habitualmente envolvem o uso da Camada de Abstração de Registros (*Register Abstraction Layer*, RAL) do UVM, que permite um acesso rápido e inteligível de registros internos do RTL para sua leitura e escrita. Uma estrutura generalista de teste pode envolver uma etapa de configurações iniciais do módulo MBIST para o modo de operação desejado, seguida de operações sequenciais, iteradas ou não em laços de execução, escritas e leituras a registros, e estabelecimento de condições de aviso e de erro diversas. Além do controle de fluxo e da sinalização de falhas próprios dos padrões, o ambiente inteiro de verificação atua concomitantemente nas simulações para fiscalizar os efeitos colaterais que decorram durante sua execução.

O RTL dos sistemas ou subsistemas que utilizam MTR são gerados automaticamente a partir de planilhas de configuração que são percorridas por um *parser*. Sempre que um padrão novo é desenvolvido ou que são encontradas (por meio de padrões já existentes) falhas no RTL em virtude de modificações realizadas, há inspeção de formas de onda, em que os membros responsáveis pela verificação do MTR organizam os sinais relevantes nas aplicações utilizadas e observam o comportamento do DUT para chegar a uma conclusão e reportar ao restante da equipe.

Semanalmente, são rodadas séries massivas de testes de regressão, que utilizam os padrões mantidos pelos membros de verificação, a fim de acompanhar o desenvolvimento do RTL pelos *designers*. As falhas que venham a aparecer podem ser endereçadas por todo o grupo, a depender de sua natureza, mas são em sua maioria tratadas pelos verificadores. Estes irão conversar com os membros encarregados pelo *design*, no caso de abrir notificações de *bug* ou de aperfeiçoamento, ou adicionarão ao ambiente de verificação o suporte devido àquelas situações particulares de falha.

Por conta dos fatores anteriormente tratados e também por albergar de forma altamente flexível projetos de SoCs de complexidades diversas, o ambiente de verificação utilizado para a maior parte dos *flavors* MBIST é repleto de estruturas e funções que interagem de forma bastante intricada. Trabalhar no desenvolvimento de um ambiente tão misto foi enriquecedor, uma vez que lidar com um código de extensão e configurabilidade tamanhas demandava larga atenção a detalhes e uma metodologia bastante analítica de trabalho.

5 Considerações Finais

O período de estágio na NXP Semiconductors agregou uma massa de conhecimento inteiramente nova, além de ter solidificado vários conceitos adquiridos durante as disciplinas cursadas no curso de graduação em Engenharia Elétrica da Universidade Federal de Campinas Grande anteriormente. Matérias como Circuitos Lógicos, Arquitetura de Sistemas Digitais e Arquiteturas Avançadas para Computação deram base fundamental para compreender mais rapidamente alguns dos desafios propostos e saber procurar um caminho razoável para resolvê-los. Claro, os obstáculos em um ambiente empresarial são distintos daqueles em um ambiente acadêmico, mas um preparo devido prova-se de grande auxílio para vencê-los.

Foram constantemente exercitadas habilidades de comunicação, o gerenciamento de tempo e de estresse, e também a coordenação de atividades individuais dentro de um contexto de trabalho em equipe, já que todos esses fatores mostram-se indispensáveis para um bom rendimento em um ambiente colaborativo e internacional para projetos de alta complexidade como os encaminhados para o Grupo MTR. A interação com membros fora do Brasil é constante, e as informações dentro do time necessitam ter transparência e fluidez para que os cronogramas se mantenham sob controle, as tarefas sejam executadas apropriadamente e não haja retrabalho. O fato que o Grupo MTR, com pouco mais de uma dezena de integrantes ao redor do mundo inteiro, em sua maior parte brasileiros, consegue servir com segurança dezenas de projetos anualmente através de praticamente todas as *business lines* da NXP, fornecendo subsistemas ou sistemas completos para o teste e reparo de memórias embarcadas, prova por si o resultado da dedicação e da comunicação de seus membros. Um foco constante em otimização, automatização e criatividade, permitindo a cada membro a proposição de soluções inovadoras são só algumas das qualidades que ajudam a desafiar a expectativa de produção de um time desse porte, mas que continua a agregar valor e reconhecimento em toda a empresa.

Uma série de aprendizados foram fomentados e desenvolvidos nesse ano de estágio. Foram varridos desde o ofício técnico de verificação funcional, com *debug*, manutenção de ambientes de teste e desenvolvimento de novos recursos que acompanhem, realimentem e inspirem o *design* dos módulos concebidos pelo grupo; à discussão aberta a respeito de erros e acertos do time e observação de formalidades e dificuldades na comunicação com clientes e na providência de suporte a suas adversidades. Uma atenção contínua era mantida nas extensas comunicações virtuais a fim de poder argumentar em cima delas durante as reuniões pertinentes que as endereçassem, e todo trabalho era documentado diariamente para que pudesse ser referenciado futuramente em discussões.

Justificaram-se, pois, os longos períodos de capacitação para que as técnicas envolvidas no trabalho com a microeletrônica fossem adequadamente absorvidas e a rotina de atividades acompanhasse esse ritmo. Portanto, ressaltam-se algumas dificuldades estruturais na execução e validação desse estágio na interface estudante/Universidade, como a limitação de duração do estágio a seis meses corridos, ainda que não prevista na Lei de Estágio Nº 11.788 de 25 de setembro de 2008, apesar do período de um ano previsto para inserção adequada do estudante no ambiente de trabalho para desenvolvimento de tecnologia de ponta. A NXP trabalha na concepção de produtos que precisam ter graus elevados de confiança, como nos meios automobilístico e de segurança, e portanto precisa-se de garantia de sua qualidade estabelecendo uma metodologia de trabalho e um conjunto de conhecimentos que permita tal solidez.

Bibliografia

Miyazaki, M.; Yoneda, T.; Fujiwara, H. A memory grouping method for sharing memory bist logic. In: *Asia and South Pacific Conference on Design Automation, 2006.* [S.l.: s.n.], 2006.

Nair, S. M.; Münch, C.; Tahoori, M. B. Defect characterization and test generation for spintronic-based compute-in-memory. In: *2020 IEEE European Test Symposium (ETS).* [S.l.: s.n.], 2020.

SHARMA, A.; COUNCIL, I. S.-S. C. *Semiconductor Memories: Technology, Testing, and Reliability.* [S.l.]: IEEE Press, 1997. ISBN 9780780311145.

Silveira, R.; Qureshi, Q.; Zeli, R. Flexible architecture of memory bists. In: *2018 IEEE 19th Latin-American Test Symposium (LATS).* [S.l.: s.n.], 2018.

Zeli, R.; Silveira, R.; Qureshi, Q. Soc memory test optimization using nxp mtr solutions. In: *2019 IEEE Latin American Test Symposium (LATS).* [S.l.: s.n.], 2019.