Centro de Engenharia Elétrica e Informática
Departamento de Engenharia Elétrica

Kelvin Dantas Vale

Relatório de Estágio Supervisionado
Laboratório de Excelência em Microeletrônica do Nordeste - XMEM

Campina Grande
2021

Kelvin Dantas Vale

Relatório de Estágio Supervisionado
Laboratório de Excelência em Microeletrônica do Nordeste - XMEM

Relatório de Estágio Supervisionado submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Eletrônica

Orientador:
Marcos Ricardo de Alcântara Morais, D. Sc.

Campina Grande
2021

Kelvin Dantas Vale

Relatório de Estágio Supervisionado
Laboratório de Excelência em Microeletrônica do Nordeste - XMEM

Relatório de Estágio Supervisionado submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Aprovado em / /

Marcos Ricardo de Alcântara Morais, D. Sc.
UFCG

Gutemberg Gonçalves dos Santos Júnior, D. Sc
Professor Convidado
UFCG

Campina Grande
2021

Dedico este trabalho à todos que me acompanharam e me ajudaram até aqui. Até mesmo um único dia diferente na minha história poderia alterar tudo até o dia de hoje.

AGRADECIMENTOS

Agradeço primeiramente à Deus pelo dom da vida e pela contante iluminação deste proveniente.

Agradeço aos meus pais. Minha mãe por, independente das dificuldades na criação de uma criança com deficit de atenção, nunca me deixou desacreditar de mim mesmo e sempre me provar que eu era capaz de alcançar além dos céus. Meu pai, por me apresentar ao incrível mundo da matemática e da física de forma tão lúdica e pelas incontáveis e memoráveis discussões nesta área desde minha infância. Se não fosse pela incrível união de quem vocês são eu jamais chegaria onde estou.

Agradeço à minha irmã casula, que mostrou durante anos que com organização e dedicação pode-se aprender qualquer coisa. O mundo é seu! É só se dedicar o suficiente.

Agradeço ao restante da minha família por nunca terem duvidado de mim e sempre me mostrado que a vida pode ser vivida de um modo mais simples e fácil.

Agradeço a todos do Laboratório XMEN pela incrível oportunidade de trabalhar com vocês e aprender tanto todos os dias.

E, por fim, agradeço à minha amada esposa por ter me aguentado durante todo esse processo de estágio e TCC, que demandou tanto tempo e atenção. E por ter mantido minha cabeça no lugar quando eu mesmo já estava perdido.

RESUMO

Neste relatório consta a descrição das atividades desenvolvidas durante o período de estágio no Laboratório de Excelência em Microeletrônica do Nordeste. No decorrer deste, será explanado brevemente sobre a metodologia de verificação UVM e aplicado o processo de verificação funcional visando a conclusão dos objetivos determinados.

Palavras-chave: Desenvolvimento de *hardware*, Verificação funcional, *Hardware Design*, Verificação, UVM, Microeletrônica, Relatório de Estágio.

ABSTRACT

In this report there is a description of the ectivities developed during the training period at Laboratório de Excelência em Microeletrônica do Nordeste. In the text, it will be briefly explained about UVM methodology and aplied it, during the process, aiming the conclusion of the objectives.

Keywords: Hardware Development, Functional Verification, Hardware Design, Verification, UVM, Microelectronics, Training Report.

LISTA DE ILUSTRAÇÕES

Figura 1 – Arquitetura XMCPROC 12
Figura 2 – Diagrama de sinais XCRC 13
Figura 3 – Diagrama de sinais XAES 14
Figura 4 – Diagrama de sinais XBUS 14
Figura 5 – Diagrama de sinais SPI 15
Figura 6 – Fluxograma dos processos XSPIS 18
Figura 7 – Arquitetura tradicional UVM 20
Figura 8 – Arquitetura UVM para verificação do XSPIS 24
Figura 9 – Resultado do comando acima 25
Figura 10 – Resultado do comando acima 26

LISTA DE ABREVIATURAS E SIGLAS

UFCG	Universidade Federal de Campina Grande
DEE	Departamento de Engenharia Elétrica
UVM	*Universal Verification Methodology*
CRC	*Cyclic Redundancy Check*
TLM	*Transaction Level Modeling*

SUMÁRIO

Lista de ilustrações 7

1 INTRODUÇÃO 10
1.1 Objetivos 10
1.2 Organização do Relatório 11

2 EMBASAMENTO TEÓRICO 12
2.1 XSPIS - Interfaces e controles 12
2.2 XSPIS - Registradores e padrão de entrada de dados 16
2.3 UVM 19

3 ATIVIDADES DESENVOLVIDAS 21
3.1 XSPIS *Agent* 21
3.2 XCRC *Agent* 22
3.3 XAES *Agent* 23
3.4 XBUS *Agent* 23
3.5 Modelo 23
3.6 Arquitetura do ambiente 23
3.7 Resultado da verificação 24

4 ERROS SOLUCIONADOS 25

5 CONSIDERAÇÕES FINAIS 28

1 INTRODUÇÃO

Com o investimento em P&D na área de microeletrônica, foi criado o Laboratório de Excelência em Microeletrônica do Nordeste (XMEN) na UFCG, campus Campina Grande. Este, fundado em 2016, visava a introdução e capacitação de alunos na área de microeletrônica, treinando os mesmos não só no desenvolvimento do projeto em si, como também na participação de um fluxo de desenvolvimento. Tais características fazem do laboratório um excelente intermediário entre a vida acadêmica e o mundo profissional, tendo em vista que, paralelamente, é constantemente desenvolvido pesquisas, como estudantes, e há uma cadeia de comando, prazos e cobranças, tal como um engenheiro.

Atualmente, devido ao constante crescimento e sucesso do XMEN, neste não há apenas a capacitação de seus membros como objetivo, mas também a participação ativa em desenvolvimento de projetos completos para empresas contratantes. Inicialmente composto por alunos, hoje o laboratório possui, em seu corpo de trabalho, graduandos, mestrandos, doutorandos e engenheiros, para garantir o constante aperfeiçoamento teórico e permanência no crescente mercado da microeletrônica.

1.1 OBJETIVOS

Este relatório objetiva constar o relato do estágio do autor no Laboratório XMEN. Este foi realizado na área de verificação funcional do fluxo de desenvolvimento do local no qual estagiou e decorreu do dia 08 de fevereiro de 2021 ao dia 10 de maio de 2021, totalizando uma carga horária de 394 horas. Seguem abaixo os objetivos específicos.

- Fazer revisão bibliográfica visando aprofundamento teórico em verificação na metodologia UVM;
- Compreender o funcionamento do bloco, desenvolvido pelo laboratório XMEN, XSPIS;
- Estudo e familiarização com o fluxo de referência do XMEN para projeto lógico dos circuitos digitais e estratégias de verificação;
- Verificação dos circuitos digitais seguindo a linguagem de descrição de hardware de referência usado na empresa (SystemVerilog) em conjunção com a metodologia de verificação UVM;

1.2 ORGANIZAÇÃO DO RELATÓRIO

No Capítulo 2 será discorrido sobre as características do bloco e a teoria de verificação implementada. No capitulo 3 serão apresentadas as atividades desenvolvidas ao longo do estágio. No capitulo 4, será descrito a solução de um dos problemas que atrasou a execução do projeto. Por fim, no capítulo 5, serão apresentadas as considerações finais.

2 EMBASAMENTO TEÓRICO

O XSPIS é um topo que possui como base um bloco SPI e, a este, há uma lógica associada para que o mesmo seja capaz de controlar os demais blocos do chip o qual está inserido. O chip, o XMCPROC, possui os seguintes blocos: XCRC, responsável pelo cálculo de CRC; XSPIS, responsável por comunicar o bloco com o meio esterno via protocolo SPI e controlar os demais do chip; XAES, responsável por codificar e decodificar dados com o protocolo AES; XMEMC, responsável por controlar diretamente a memória interna do bloco. Segue o diagrama abaixo.

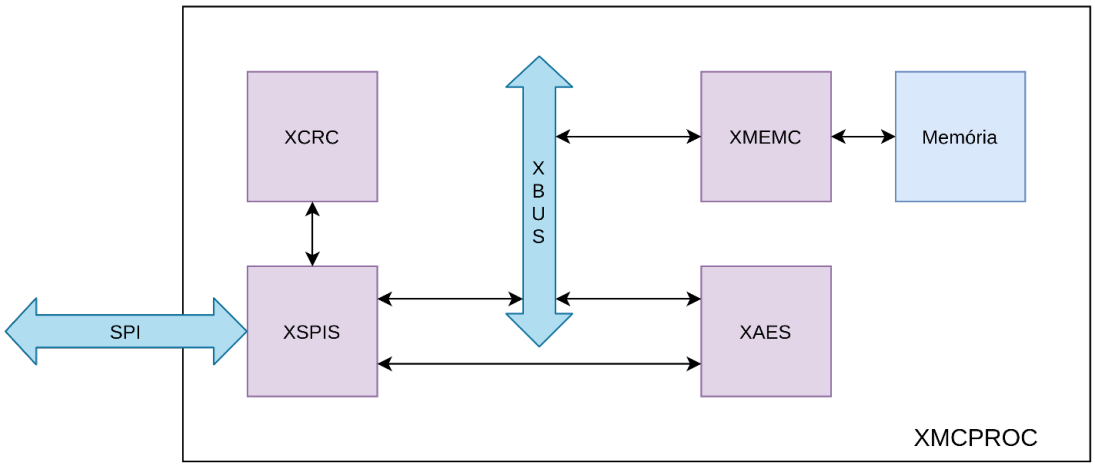


Figura 1 – Arquitetura XMCPROC

O XMCPROC provê, ao usuário, acesso à um bloco de memória com codificação de dados. No chip, a informação a ser escrita é decodificada e armazenada, bem como a informação a ser lida é internamente codificada antes do envio. Sendo assim, para verificar o XSPIS, deve-se ter conhecimento do modo como o mesmo configura e utiliza os demais blocos.

2.1 XSPIS - INTERFACES E CONTROLES

Nesta seção será explanado separadamente como os blocos possuem suas informações escritas, lidas e seu modo de operação configurado.

Observação

· Ao final do nome dos sinais que serão apresentados, há indicadores para informar se o mesmo é entrada, '_i', ou saída, '_o', do bloco. Estes, são referentes ao XSPIS, ou seja, indicarão se os sinais são entradas ou saídas do bloco a ser verificado.

Controle do XCRC

O XCRC é um bloco de cálculo com latência de um *clock* cuja função é a determinação do CRC (*Cyclic Redundancy Check*) dos dados inseridos. Este possui uma interface de comunicação simples que está ilustrada abaixo.

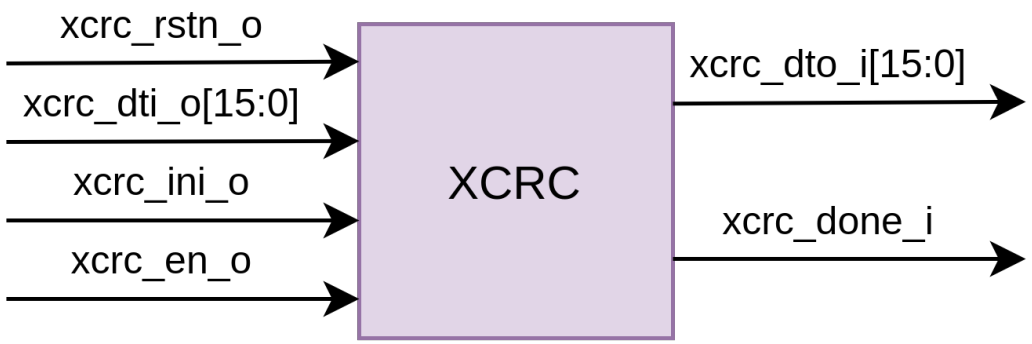


Figura 2 – Diagrama de sinais XCRC

Este irá começar o cálculo após um *reset*, ou seja, após o sinal *xcrc_rstn_o* variar do nível lógico 0 ao 1, e quando os sinais *xcrc_en_o* e *xcrc_ini_o* estiverem em nível lógico 1. Ao cumprir esta condição, pode-se enviar quantas transações forem necessárias, de 16 em 16 bits, pelo *xcrc_dti_o[15:0]* até completar o tamanho da palavra desejada. Para que uma nova operação de cálculo seja iniciada, deve-se efetuar novamente o processo de *reset*.

Controle do XAES

O XAES é um bloco cuja função é a aplicação do algoritmo AES para encriptar o valor que lhe é fornecido. Sua interface de comunicação com o XSPIS está ilustrada abaixo.

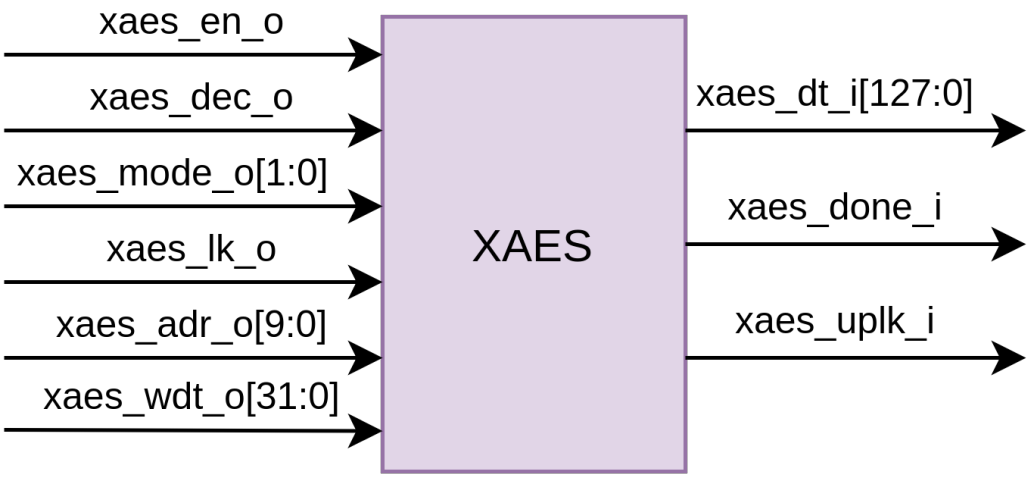


Figura 3 – Diagrama de sinais XAES

A partir dos sinais de entrada do XAES, é possível configurar, se este está ativo, *xaes_en_o*; se o mesmo estará atuando no modo de encriptação ou no modo de decriptação, *xaes_dec_o*; o modo de encriptação que será aplicado, *xaes_mode_o[1:0]*; se o bloco estará em um estado de *lock key*, *xaes_lk_o*; e, por meio dos sinais *xaes_wdt_o[31:0]* e *xaes_adr_o[9:0]*, escrever em seus registradores internos os 320 bits a serem aplicados o algoritmo.

Controle por meio do barramento XBUS

O acesso ao XMEMC ocorre unicamente por meio do barramento XBUS, sendo assim, será analisado a interface entre o este e o XSPIS. O diagrama do barramento com os sinais está ilustrado abaixo.

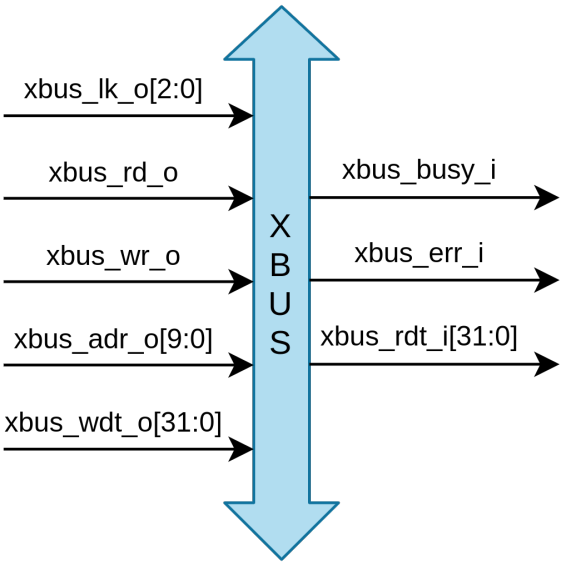


Figura 4 – Diagrama de sinais XBUS

Por meio dos sinais de entrada do XBUS, é informado qual o tipo de operação será efetuado, *xbus_rd_o* e *xbus_wr_o*, leitura e escrita, respectivamente; qual o *lock* da memória, *xbus_lk_o[2:0]*; e, por meio dos sinais *xbus_wdt_o[31:0]* e *xbus_adr_o[9:0]*, enviar o dado e o endereço a ser escrito na memória.

Sinais protocolo SPI

Como já relatado, além de controlar os blocos acima descritos, o XSPIS também deve se comunicar com o meio externo ao chip em um protocolo SPI. Sendo assim, segue abaixo o diagrama dos sinais usados na implementação deste protocolo.

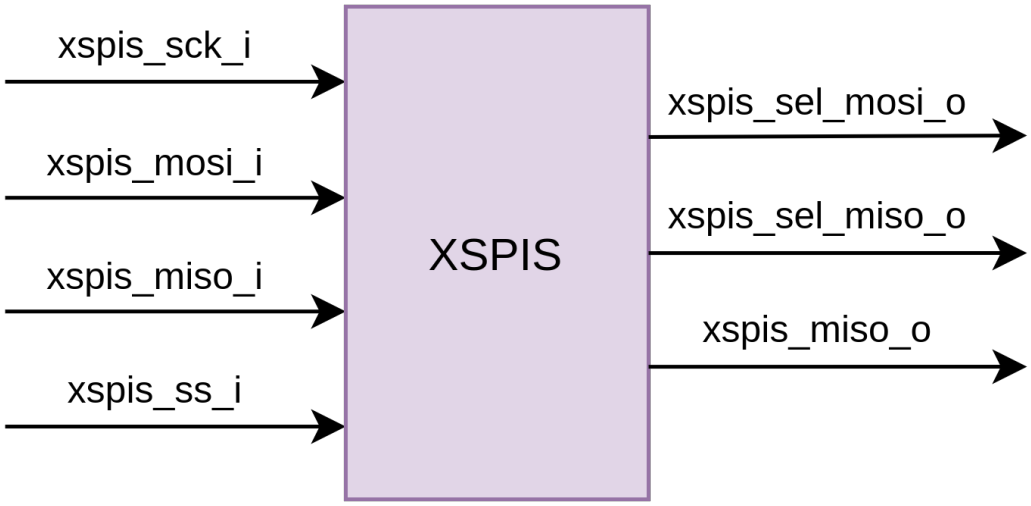


Figura 5 – Diagrama de sinais SPI

Especificação dos sinais

· *xspis_sck_i* - *Clock* da comunicação síncrona SPI.

· *xspis_mosi_i* - Bit enviado do mestre ao escravo. Sendo este bloco o escravo da comunicação.

· *xspis_miso_i* - Bit enviado do escravo ao mestre. Sendo este bloco o mestre da comunicação.

· *xspis_ss_i* - Sinal de seleção de escravo. Caso esteja no nível lógico 0, o bloco irá iniciar a comunicação com o mestre.

· *xspis_sel_mosi_o* - Se em nível lógico 1, o sinal *xspis_mosi_i* atuará como saída de dados.

· *xspis_sel_miso_o* - Se em nível lógico 1, o sinal *xspis_mosi_i* atuará como entrada de dados.

· *xspis_miso_o* - Bit enviado do escravo ao mestre. Sendo este bloco o escravo da comunicação.

2.2 XSPIS - REGISTRADORES E PADRÃO DE ENTRADA DE DADOS

No XSPIS há dois registradores que são possíveis serem acessados via SPI, estes estão localizados nos endereços 10'h3FE e 10'h3FF, sendo, respectivamente, os registradores de controle e status. No registrador de controle o usuário pode configurar as seguintes variáveis:

- procen, bit 31: Inicia o processo do XAES, quando em nível lógico 1.
- encdec, bit 30: Configura se o processo do XAES é de encriptação, nível lógico 0, ou de decrepitação, nível lógico 1.
- mode, bits 29-28: Determina o modo no qual o XAES irá operar.
- lk_key, bit 27: Chave que bloqueia parte da memória.
- crcen, bit 26: Quando em nível lógico 1, habilita o XCRC.
- spilen, bits 23-22: Determina o tamanho da palavra (Não utilizado).
- spi_mode, bits 21-20: Escolhe entre a comunicação por 3 fios, valor 2'b01, ou por modo *single*, valor 2'b00.

Ao ler o registrador de *status*, o usuário obterá as seguintes informações.

- procrsp, bit 31: Indica que o core está ativo quando em nível lógico 1.
- lk_err, bit 30: Indica que há áreas bloqueadas na memória quando em nível lógico 1.
- crc_err, bit 29: Indica que houve uma divergência entre o crc enviado e o calculado pelo XCRC quando em nível lógico 1.

Observações

· Os bits não especificados acima foram reservados para uso futuro.
· O registrador de controle possui permissão de escrita e leitura.
· O registrador de *status* possui permissão de leitura, porém não possui permissão de escrita.

Na comunicação serial via protocolo SPI, teremos que os bits a serem recebidos são interpretados conforme a ordem de chegada. Portanto, segundo a documentação do bloco, a seguinte disposição de bits foi determinada.

- 1^o bit: Caso o mestre envie um bit de nível lógico zero, indica que a operação em andamento é de leitura; caso seja enviado com nível lógico um, indica que é uma operação de escrita.
- 2^o ao 11^o bit: Neste campo, foi determinado que seria enviado o endereço cuja operação determinada anteriormente irá acessar.
- 12^o ao 13^o bit: Este foi denominado sensor. A partir deste valor, o bloco irá agir conforme o diagrama abaixo.
- 14^o ao 16^o bit: Este campo está reservado para uso futuro.
- 17^o ao 48^o bit: Neste será enviado o dado a ser escrito, ou recebido o dado lido.
- 49^o ao 64^o bit: Campo reservado ao código CRC. Será considerado apenas se, no registrador de controle do XSPIS, o CRC estiver configurado de forma ativa.

Segue o fluxograma dos possíveis processos, tendo como parâmetro os primeiros 16 bits enviados.

Fonte: Thiago da Silva Cruz, designer do bloco.

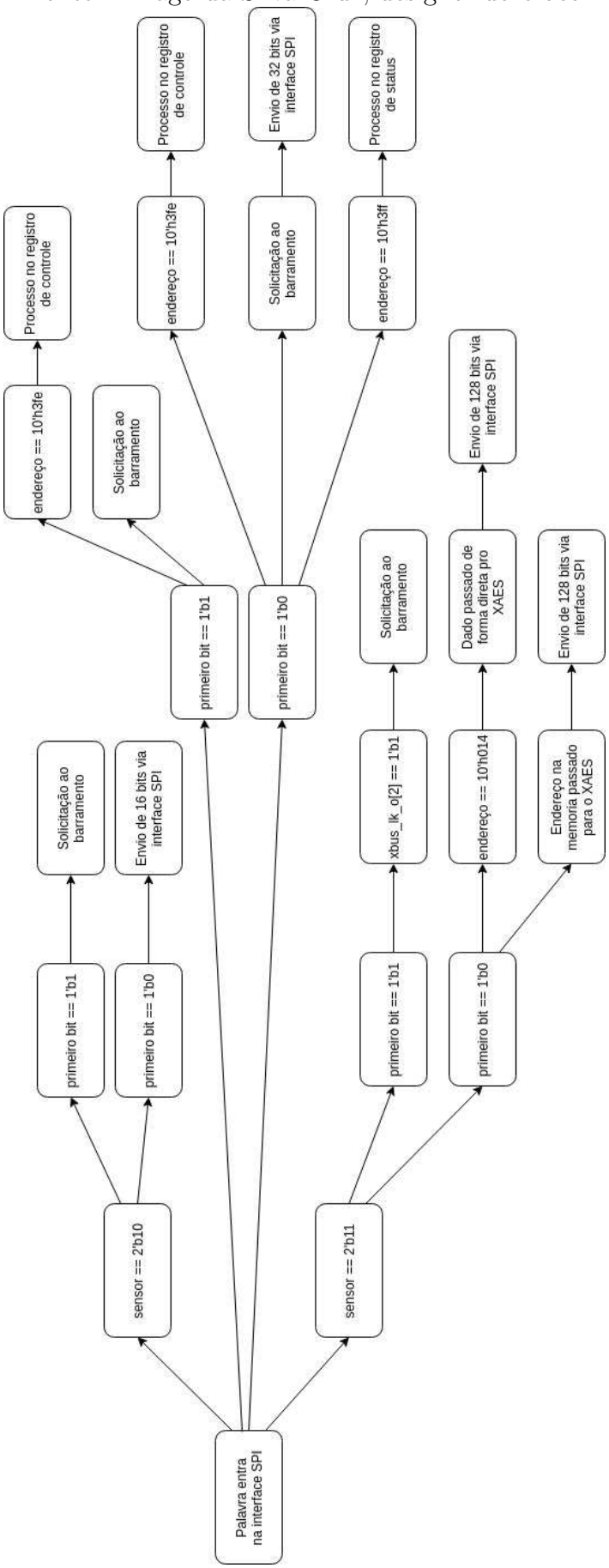


Figura 6 – Fluxograma dos processos XSPIS

2.3 UVM

No fluxo de trabalho, para estimular as entradas do bloco e observar suas saídas, é utilizada, no laboratório XMEN, a metodologia UVM (*Universal Verification Methodology*). Esta, teve sua funcionalidade desenvolvida tendo como base a comunicação entre seus componentes em nível de transação, abstração TLM (*Transaction Level Modeling*). Para viabilizar o uso desta metodologia, foi criada uma biblioteca de classes padrões a serem instanciadas e configuradas, além de arquiteturas de ambiente propostos para a verificação.

Em um ambiente de verificação padrão, há a instância de blocos com objetivos e funcionalidades específicas. Dentre os que a biblioteca fornece, os mais comumente usados são:

Monitor, cujo objetivo é monitorar as interfaces que estão ligadas ao DUT (*Design Under Test*) e enviar transações com as informações apropriadas para o Comparador, Cobertura e Refmod;

Driver, se comunica diretamente com o bloco por meio das interfaces, sendo este responsável por enviar as informações respeitando os protocolos das interfaces;

Sequencer, a partir de uma sequencia, monta as transações com os dados a serem enviados ao DUT e as encaminha ao *driver*;

Agent, encapsula e configura os blocos sob sua instância, sendo estes o *driver*, o monitor e o *sequencer*;

Comparator, recebe transações em duas de suas portas já estabelecidas em sua classe base e as compara campo a campo para determinar se estas são idênticas ou não;

Coverage, neste, são estabelecidas condições para que a verificação seja dada por completa;

Refmod, recebe os mesmos dados de entrada que o DUT e envia sua resposta ao comparador, sendo esta usada como referência de comparação pelo bloco;

Scoreboard, encapsula e configuraa os blocos, *comparator*, *coverage* e *refmod*;

Enviroment, neste é instanciado todos os blocos que serão usados na verificação.

Segue abaixo uma imagem que ilustra a disposição dos blocos acima descritos em uma arquitetura simples de um ambiente de verificação em UVM.

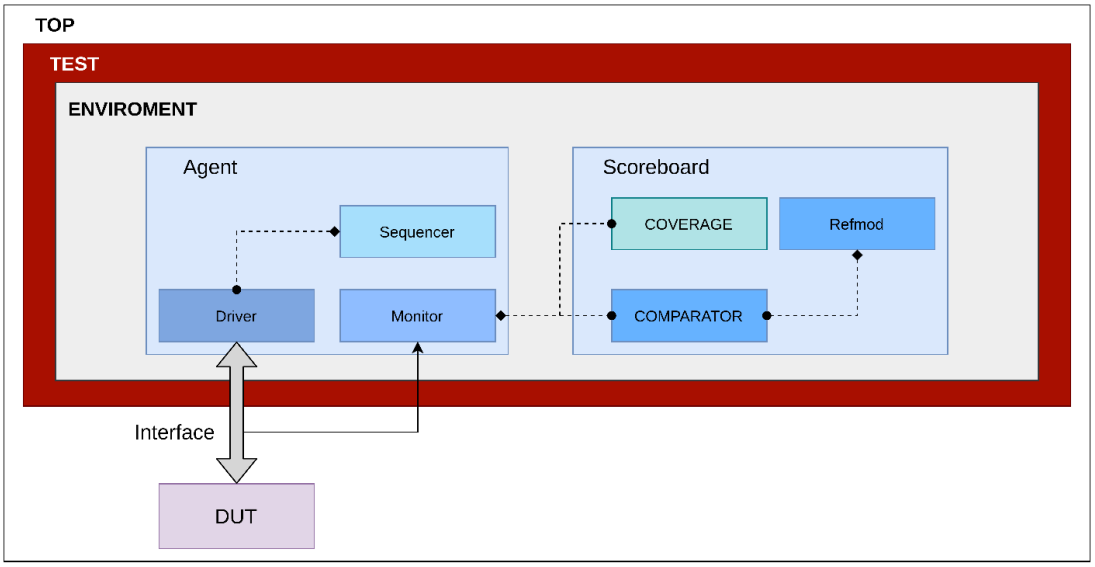


Figura 7 – Arquitetura tradicional UVM

No esquema acima ilustrado, os símbolos abaixo são:

♦ → Porta de saída da transação.

• → Porta de entrada da transação.

Observação

· A depender da complexidade do DUT (*Design Under Test*) a ser verificado, este ambiente pode ser modificado e adaptado para melhor aproveitamento da metodologia e suas ferramentas.

3 ATIVIDADES DESENVOLVIDAS

Tendo em vista que o XSPIS possui 4 interfaces de comunicação, foram criados 4 *agents*, cada qual com seus blocos internos, *driver*, *monitor* e *sequencer*, adaptados à uma comunicação específica.

3.1 XSPIS *AGENT*

Para a interface SPI, foi criado um *agent* que simula um mestre SPI. Neste, visando respeitar a sequência de bits determinado na documentação, foi desenvolvido, na sequência, duas *tasks* que padronizam a geração de transações para a escrita e leitura respeitando o protocolo. Segue abaixo o trecho do código citado acima.

Código da padronização da escrita e leitura do SPI

```
task write(transaction_type tr, logic[9:0] addr, logic[1:0] sensor, logic[31:0] wdt, logic[15:0] crc, bit crc_in);

  tr = transaction_type::type_id::create("tr");
    `uvm_do_with(tr, {tr.xspis_mosi_i=={1'b1,addr[9:0],sensor[1:0],3'b000}; tr.xspis_ss_i==1'b0;})
    //Envia solicitação de escrita com o endereço

  tr = transaction_type::type_id::create("tr");
    `uvm_do_with(tr, {tr.xspis_mosi_i=={wdt[31:16]}; tr.xspis_ss_i==1'b0;}) //Envia a primeira parte do dado
  tr = transaction_type::type_id::create("tr");
    `uvm_do_with(tr, {tr.xspis_mosi_i=={wdt[15:0]}; tr.xspis_ss_i==1'b0;}) //Envia a segunda parte do dado

  if (crc_in==1'b1) begin
    tr = transaction_type::type_id::create("tr");
      `uvm_do_with(tr, {tr.xspis_mosi_i=={crc[15:0]}; tr.xspis_ss_i==1'b0;}) //Envia o crc

    uvm_config_db#(logic[15:0])::set  (
      .cntxt(uvm_root::get()),
      .inst_name("*"),
      .field_name("crc_val"),
      .value(crc)
    );
  end
```

```
23
24      tr = transaction_type::type_id::create("tr");
25        `uvm_do_with(tr, {tr.xspis_ss_i==1'b1;})
26    endtask
27
28    task read(transaction_type tr, logic[9:0] addr, logic[1:0] sensor, logic
       [15:0] crc);
29
30      tr = transaction_type::type_id::create("tr");
31        `uvm_do_with(tr, {tr.xspis_mosi_i=={1'b0,addr[9:0],sensor[1:0],3'b000
       }; tr.xspis_ss_i==1'b0;})
32        //Envia solicitação de escrita com o endereço
33
34      tr = transaction_type::type_id::create("tr");
35        `uvm_do_with(tr, {tr.xspis_mosi_i=={16'h0}; tr.xspis_ss_i==1'b0;}) //
       Envia DUMMY PARTE 1
36      tr = transaction_type::type_id::create("tr");
37        `uvm_do_with(tr, {tr.xspis_mosi_i=={16'h0}; tr.xspis_ss_i==1'b0;}) //
       Envia DUMMY PARTE 2
38
39      tr = transaction_type::type_id::create("tr");
40        `uvm_do_with(tr, {tr.xspis_mosi_i=={16'h0}; tr.xspis_ss_i==1'b0;}) //
       Envia o crc
41
42      tr = transaction_type::type_id::create("tr");
43        `uvm_do_with(tr, {tr.xspis_ss_i==1'b1;})
44    endtask
```

Ao executar estas funções, é garantido que os bits enviados estão no mesmo tamanho e sequência especificado pela documentação. Além disto, tendo em vista que a verificação não visa garantir a funcionalidade dos outros blocos do topo, na função '*write()*' é enviado ao *driver* da interface referente ao XCRC o valor crc enviado ao XSPIS.

3.2 XCRC *AGENT*

Neste, foi desenvolvido um *agent* sem sequenciador, pois o objetivo do mesmo é apenas simular os *handshakes* do XCRC e enviar ao XSPIS um valor referente ao cálculo do crc. Para que a execução do ambiente de verificação não se tornasse tão pesada, foi definido um sistema no qual, como falado anteriormente, o *agent* do XSPIS envia o valor passado pela interface SPI para o *driver* do XCRC. Portanto, dependendo das condições determinadas no teste, o *driver* irá enviar o mesmo crc que o passado ao XSPIS, ou forçar um erro para ver como o mesmo irá reagir.

3.3 XAES *AGENT*

Neste, será simulado a interação do XAES com os sinais de controle provenientes do XSPIS. Quanto ao valor que deveria ser resultado do algorítimo AES, a cada transação é enviado um valor aleatório, pois o valor em si não muda em nada a funcionalidade do bloco em análise.

3.4 XBUS *AGENT*

Para o *agent* do barramento, foi definido um bando de registradores que pode ser acessado pela interface, além de neste ser simulado o comportamento dos sinais tanto do XBUS quanto dos outros blocos por este acessado. Com esta abstração, pode-se gerar uma comunicação bem mais rápida e eficiente com o XSPIS, além do que, com o banco de registradores interno ao ambiente, torna mais fácil a comparação dos valores lidos pela interface serial.

3.5 MODELO

A fins de gerar uma referência de comparação, foi desenvolvido um modelo em python, no qual o fluxograma apresentado na fundamentação teórica é implementado em alto nível. Neste modelo é definida uma classe cujas características se assemelham ao DUT e, por meio de funções passadas ao Systemverilog via DPI-C, o *refmod* envia ao modelo os mesmos valores que o RTL recebe.

3.6 ARQUITETURA DO AMBIENTE

Ao apresentar as especificidades dos blocos acima, teremos abaixo ilustrada a arquitetura desenvolvida para esta verificação.

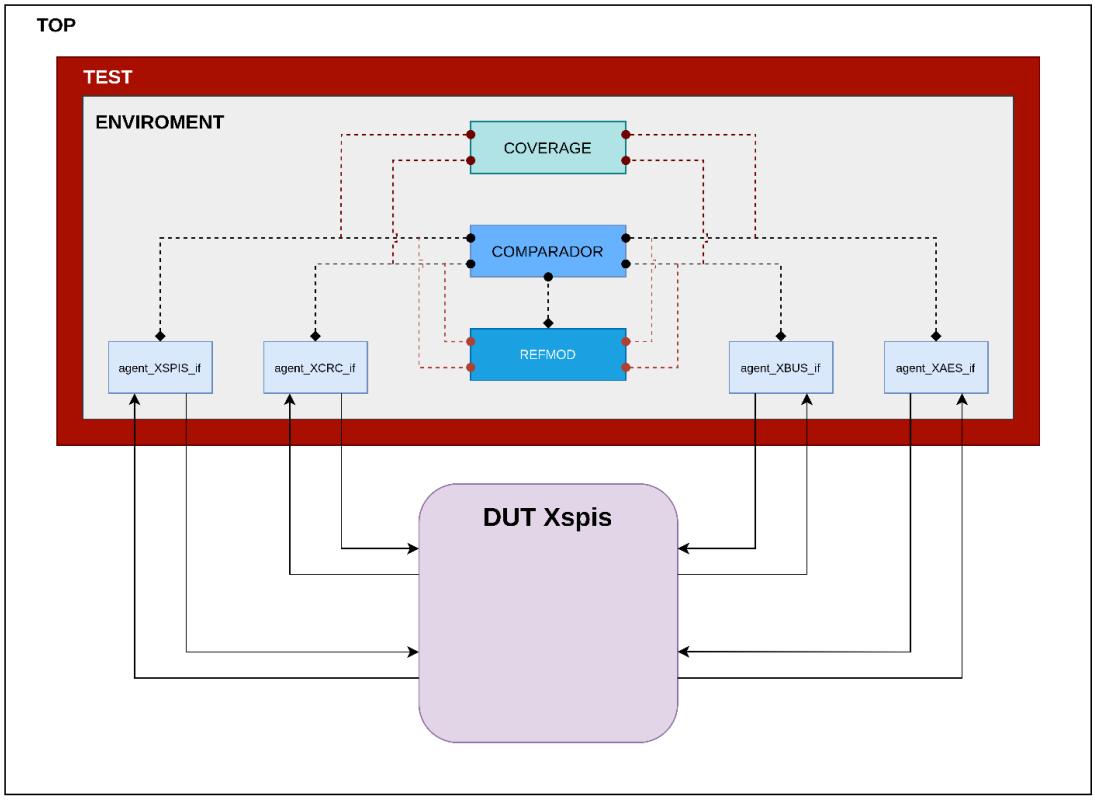


Figura 8 – Arquitetura UVM para verificação do XSPIS

3.7 RESULTADO DA VERIFICAÇÃO

Como resultado, obteve-se a não conclusão da verificação do bloco, cujo interrompimento se deu antes da etapa de análise de cobertura. Tal ocorreu devido a fatores externos acrescidos do fato de que a complexidade da verificação do XSPIS superou o esperado, acarretando em um menor tempo de produção e uma maior demanda deste para a execução da tarefa.

Tais fatores externos foram, o mal funcionamento do computador inicialmente utilizado, este parou de funcionar; e o fato de que a versão do python nos outros computadores do laboratório serem diferentes da versão do computador usado para desenvolver o modelo, gerando divergência de bibliotecas entre a ferramenta e o código desenvolvido. Sendo assim, foi necessário a migração do trabalho para outro e dias de pesquisa para desenvolver a solução para o compilador.

Portanto, a verificação do bloco, no ponto em que foi parada, gera relatórios inconclusivos quanto a confiabilidade da funcionalidade do design. Para continuidade deste trabalho, seria necessário a correção de alguns erros durante a compilação do ambiente e relatar ao engenheiro responsável pela correção dos *Bugs* as divergências do funcionamento do DUT em relação à especificação do mesmo.

4 ERROS SOLUCIONADOS

Durante o desenvolvimento do projeto, houveram alguns erros de compilação. Dentre estes, o mais complexos de ser solucionado foi durante a execução do python. Sendo este o escrito abaixo:

ModuleNotFoundError: No module named 'encodings'

O *encodings* é um dos módulos indispensáveis da biblioteca python, desta maneira, independente da versão utilizada, o modulo em questão estará presente. Após pesquisas e consultas aos colegas de trabalho, determinou-se que o erro ocorre quando uma versão do python tenta executar uma biblioteca compartilhada gerada por outra versão. Ou seja, a biblioteca compartilhada 'teste.so' foi gerada por uma versão de python diferente da utilizada na ferramenta XCELIUM da CADENCE.

Para solucionar o problema, foram executados os seguintes procedimentos:

> *OBS: Alguns comandos serão possíveis apenas nos computadores do XMEN, tendo em vista que são provenientes de scripts padrões do laboratório, porém, os passos estão detalhados permitindo a reprodução em outras máquinas.*

Primeiro passo: Carregar as ferramentas da CADENCE:

Comando (*script* XMEN):

```
1 cds
```

Segundo passo: Verificar se o python utilizado pertence às ferramentas da CADENCE:

Comando:

```
1 which python3
```

Figura 9 – Resultado do comando acima

```
[kelvin.vale@Incineroar ~]$ which python3
/usr/local/mentor/calibre/aoj_cal_2019.1_29.17/bin/python3
```

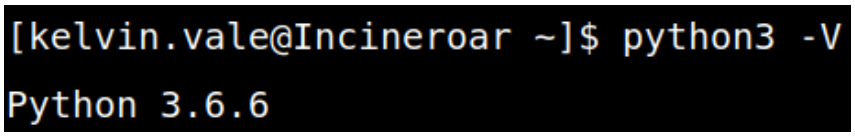
Fonte: Próprio autor

Terceiro passo: Verificar a versão do python3 utilizado nas ferramentas da CADENCE:

Comando:

```
python3 –V
```

Figura 10 – Resultado do comando acima



Fonte: Próprio autor

Tendo em vista que se sabe a versão do python utilizada pela ferramenta, instala-se localmente a mesma versão para ser utilizada na compilação da biblioteca compartilhada. Para tal, teremos:

Quarto passo: Alterar o diretório de trabalho do terminal para o que se deseja instalar a nova versão do python:

Comando:

```
cd  (Caminho para o local que se deseja instalar)
```

Quinto passo: Utilizando a permissão "sudo", baixar o pacote da versão desejada:

Comando:

```
sudo wget https://www.python.org/ftp/python/3.x.x/Python-3.x.x.
    tgz
```

OBS: No caso da ferramenta utilizada, teremos 3.6.6.

Sexto passo: Utilizando a permissão "sudo", descompactar o pacote baixado:

Comando:

```
sudo tar xzf Python-3.x.x.tgz
```

Sétmo passo: Alterar o diretório de trabalho do terminal para a pasta descompactada:

Comando:

```
cd Python-3.x.x/
```

Oitavo passo: Executar a configuração presente no diretório utilizando as *tags* apresentadas no código abaixo:

Comando:

```
./configure --enable-optimization --enable-shared --prefix= (
    Caminho que se deseja instalar a nova biblioteca)
```

–enable-optimization : Habilita a optimização durante a instalação.
–enable-shared : Necessário para executar a instalação executando as bibliotecas de modo a viabilizar a biblioteca compartilhada.
–prefix : Define o diretório no qual será instalado a nova biblioteca.

Nono passo: Preparar a instalação:

Comando:

```
make
```

Ou,

```
make -j8
```

A opção '-j8' executa o *make* com 8 núcleos dedicados ao processamento, acarretando em um processamento mais rápido.

Décimo passo: Instalar:

Comando:

```
make install
```

Ao concluir o décimo passo, a versão do python poderá ser incluída no Makefile como o compilador das funções do modelo em alto nível.

5 CONSIDERAÇÕES FINAIS

Neste documento foram apresentadas todas as atividades desenvolvidas pelo aluno no Laboratório de Microeletrônica do Nordeste, durante o período de vigência do estágio, bem como suas dificuldades e soluções de problemas. Tal período foi de suma importância para o aprimoramento dos conhecimentos na área de microeletrônica, além do treinamento para o mercado de trabalho no qual está para ser absorvido, tendo em vista que o mesmo foi inserido em um fluxo completo de desenvolvimento de projeto, além de ampliar sua rede de contatos por meio do convívio diário com outros engenheiros da mesma área.

Além do desenvolvimento pessoal e profissional, notou-se, durante o desenvolvimento do projeto, a importância das disciplinas estudadas durante a graduação, tanto para desenvolvimento de raciocínio lógico, quanto para gerar uma base de conhecimento para o aprimoramento profissional do estagiário. Dentre as disciplinas, destacaram-se, Introdução à programação e Técnicas de programação, para o aprendizado da lógica de programação, bem como a programação orientada a objeto; Circuitos Lógicos, Arquitetura de Sistemas Digitais e seus respectivos laboratórios, para o aprendizado não só do Systemverilog, como também da base teórica da microeletrônica e lógica computacional.