Universidade Federal de Campina Grande

Centro de Engenharia Elétrica e Informática

Curso de Graduação em Engenharia Elétrica

THIAGO FERREIRA DE PAIVA LEITE

FLUXO DE PROJETO DE CIRCUITOS INTEGRADOS EM TECNOLOGIA FD-SOI 28NM

Campina Grande, Paraíba
Agosto de 2015

THIAGO FERREIRA DE PAIVA LEITE

FLUXO DE PROJETO DE CIRCUITOS INTEGRADOS EM TECNOLOGIA FD-SOI 28NM

Trabalho de Conclusão de Curso 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:

Professor Marcos Ricardo Alcântara Morais, D. Sc.

Campina Grande, Paraíba
Agosto de 2015

THIAGO FERREIRA DE PAIVA LEITE

FLUXO DE PROJETO DE CIRCUITOS INTEGRADOS EM TECNOLOGIA FD-SOI 28NM

Trabalho de Conclusão de curso 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

Aprovado em / /

Prof. Marcos Ricardo Alcântara Morais, D.Sc, UFCG
Orientador

Professor Avaliador
Componente da Banca

AGRADECIMENTOS

Agradeço aos meus pais, Selma Maria e Gilson Leite, que sempre me incentivaram a estudar e puderam me oferecer tudo que sempre precisei.

À minha irmã, Maíra Leite, e minha tia, Ana Maria, que sempre estiveram presentes, me aconselhando e apoiando nos momentos mais difíceis.

Aos meus amigos, que ajudaram a transformar a jornada da graduação em momentos agradáveis, os quais serão lembrados com bastante afeição.

Ao laboratório TIMA de Grenoble, França, por ter disponibilizado toda a infraestrutura necessária à realização deste trabalho. Agradeço aos meus colegas Leonel Guimarães, Otto Rolloff, Raphael Vieira, Jean Simatic e Tugdual Pelleter, pelo companheirismo durante a execução deste trabalho de conclusão de curso e à Rodrigo Possamai pelos conselhos e orientação na execução do mesmo.

Por fim, mas não menos importante, gostaria de agradecer à orientação do professor Marcos Morais, a qual foi de grande ajuda no decorrer deste estudo.

“Mar meu Deus do Céu”

Laís Leal.

RESUMO

A fabricação eficaz de circuitos integrados digitais em larga escala depende, basicamente, da tecnologia de fabricação utilizada para produzi-los e do estabelecimento de uma sequência sistemática de etapas de pré-fabricação, referidas comumente como fluxo de projetos de circuitos integrados. Neste trabalho propõe-se o desenvolvimento de um fluxo de projeto de circuitos integrados digitais em tecnologia FD-SOI 28nm. O circuito de um banco de filtros FIR passou por todas as etapas do fluxo de projeto proposto, tendo seu layout concebido. Em seguido o layout de um microprocessador código aberto foi criado. Por fim são apresentadas algumas formas de integração de IP cores ao projeto de circuitos integrados.

Palavras-chave: Circuitos Integrados, FD-SOI, IP cores.

ABSTRACT

The effective manufacture of digital integrated circuits depends on the manufacturing technology used to produce them and the establishment of a systematic sequence of prefabrication steps, commonly referred to as design flow of integrated circuit. In this work, the development of a design flow for digital integrated circuits using the FD-SOI 28nm technology is proposed. The layout of a set of FIR filters was produced, beginning on the hardware description in VHDL up to the final stages of the proposed design flow, for the validation purposes. Subsequently, the layout of an open source microprocessor was created using the same design flow. Finally, some forms of IP cores integration to the target IC is presented.

Keywords: Integrated Circuits, FD-SOI, IP cores.

LISTA DE ILUSTRAÇÕES

Figura 1. Corte transversal de transistores em tecnologia SOI parcialmente depletado (a) e FD-SOI (b). (MENTOR GRAPHICS, 2008).....14
Figura 2. Seção transversal de transistor em tecnologia tipo bulk, à esquerda, e em tecnologia FD-SOI, à direita. (THINGS2DO, 2013).....15
Figura 3. Transistores FD-SOI com óxido enterrado de 10nm (a) e 25nm (b). (LIU et al., 2011).....18
Figura 4. Transistores FD-SOI com poço convencional, acima, poço invertido, abaixo, e suas respectivas margens de tensão de polarização do BP, à direita. (FLATRESSE, 2013) 19
Figura 5. Corte transversal de um transistor FD-SOI (a). Variação do comprimento do canal em função do polybiasing (b) e (c). (FLATRESSE, 2013).....21
Figura 6. Fluxo clássico de projetos de circuitos integrados, adaptado à tecnologia FD-SOI.....23
Figura 7. Diagrama de blocos dos scripts que compõem fluxo de projeto de CIs proposto. 24
Figura 8. Estrutura do ambiente de testes simplificado proposto. 26
Figura 9. Função da ferramenta de síntese no fluxo de projeto de CIs clássico (SYNOPSYS, 2010). 28
Figura 10. Etapas da concepção do layout no fluxo de projeto de circuitos integrados proposto. 30
Figura 11. Diagrama de blocos do banco de filtros FIR projetado. 36
Figura 12. Simulação do ambiente de testes criado para a verificação do comportamento do banco de filtros FIR.....37
Figura 13. Formas de onda obtidas com a simulação comportamental da descrição do banco de filtros FIR (ampliação). 38
Figura 14. Módulo top do banco de filtros FIR, com entradas e saídas, gerado pelo Design Compiler.....39
Figura 15. Esquemático do banco de filtros FIR, gerado pela ferramenta de síntese.....39
Figura 16. Portas lógicas que compõem os blocos FIR. 40
Figura 17. *Pads* do banco de filtros FIR. 41
Figura 18. Layout do banco de filtros FIR após adição de células de preenchimento. 41
Figura 19. Layout do banco de filtros FIR após adição das linhas de alimentação (VDD e VSS). 42
Figura 20. Layout do banco de filtros FIR com células de preenchimento para evitar efeito de proximidade e *well taps* para evitar *latchup*.....43
Figura 21. Layout do banco de filtros FIR após etapa de *placement*. 43
Figura 22. Layout do banco de filtros FIR após síntese da árvore de relógio.....44
Figura 23. Layout do banco de filtros FIR após adição de células de preenchimento padrão. 45
Figura 24. Layout final do banco de filtros FIR. 46
Figura 25. Ampliação do layout final do banco de filtros FIR. 46
Figura 26. Diagrama de blocos da arquitetura geral do microprocessador STORMcore (NOLTING, 2012) 48
Figura 27. Ambiente de teste do microprocessador STORMcore. 50
Figura 28. Layout do microprocessador STORMcore após criação das linhas de alimentação, VDD e VSS. 51
Figura 29. Layout do microprocessador STORMcore após finalização da etapa de *floorplan*.....52
Figura 30. Layout do microprocessador STORMcore após *placement*.....53
Figura 31. Layout do microprocessador STORMcore após síntese da árvore de relógio.....53
Figura 32. Layout final do microprocessador STORMcore. 54

LISTA DE TABELAS

Tabela 1. *Polybiasing*, comprimentos de projeto e comprimentos efetivos disponíveis para biblioteca de células padrão da tecnologia FD-SOI 28nm.....22

LISTA DE ABREVIATURAS E SIGLAS

CI – Circuito Integrado

IP core - *Intellectual Property core*

FD-SOI – *Fully Depleted Silicon on Insulator*

EDA – *Electronic Design Automation*

CAD – *Computer Aided Design*

SoC – *System on Chip*

FET – *Field Effect Transistor*

UTBB – *Ultra Thin Body and BOX*

BOX – *Buried Oxide*

RDF – *Random Dopant Fluctuation*

BB – *Body Biasing*

PVT – *Process, Voltage, Temperature*

HDL – *Hardware Description Level*

VHDL – VHSIC Hardware Description Language

VHSIC – *Very High Speed Integrated Circuit*

P&R – *Place and Route*

VIP – *Verification IP*

RTL – *Register Transfer Level*

SUMÁRIO

1 Introdução 12
2 A Tecnologia FD-SOI 14
2.1.1 Óxido enterrado 16
2.1.2 Tensão de Limiar (V_{th}) 17
2.1.3 Polybiasing 20
3 Fluxo de Projeto de Circuitos Integrados em Tecnologia FD-SOI 28nm 23
3.1 Descrição do Hardware e Verificação Funcional da Arquitetura 25
3.2 Síntese Lógica e Simulação Pós-síntese 27
3.3 Concepção do Layout e Simulação Pós-Layout 29
4 Formas de execução do fluxo de projeto de CIs proposto 35
4.1 Projeto sem integração de IP de terceiros 35
4.1.1 O circuito projetado 36
4.1.2 Simulação Comportamental 37
4.1.3 Síntese Lógica 38
4.1.4 Concepção do Layout 40
4.2 Integração de IP de terceiros com código fonte 47
4.2.1 O microprocessador STORMcore 47
4.2.2 Síntese Logica 50
4.2.3 Concepção do Layout 51
4.3 Integração de IPs Sintetizáveis sem Código Fonte 54
4.4 *Hard* IPs 55
4.5 IPs de Verificação 55
5 Conclusões 57
Bibliografia 59
APÊNDICE A – Código Fonte do Módulo FIR do Banco de Filtros FIR 61
APÊNDICE B – Código Fonte do Módulo TOP_FIR do Banco de Filtros FIR 63
APÊNDICE C – Código Fonte do Ambiente de Testes do Banco de Filtros FIR 65

1 INTRODUÇÃO

O desenvolvimento crescente da eletrônica tem possibilitado a redução das dimensões dos circuitos à escalas cada vez menores. Circuitos que ocupavam salas, a alguns anos atrás, hoje cabem na palma da mão. Essa evolução foi possível graças ao surgimento e evolução das técnicas de fabricação dos transistores.

Contudo, a continuação do ritmo de redução nas dimensões dos transistores tem sido cada vez mais difícil. A redução da espessura do óxido de porta torna mais intenso o fenômeno de tunelamento, provocando, assim, aumentos exponenciais na corrente de fuga, podendo causar falhas no dielétrico (HARON, 2008).

Outros limitantes da miniaturização dos transistores, que podem ser citados, são os efeitos negativos da redução do comprimento do canal (em inglês, *short channel effects*); redução na mobilidade de portadores e tunelamento banda a banda, causados pelo aumento da dopagem do canal, a fim controlar os efeitos de canal curto (FRANK, 2001); e limitações físicas dos materiais (HARON, 2008).

Para combater esses efeitos negativos, novas tecnologias de fabricação de circuitos integrados têm sido desenvolvidas. A tecnologia *Ultra Thin Body & BOX Fully-Depleted Silicon On Insulator* (UTBB FD-SOI), que será abordada no capítulo seguinte, é apontada pela fabricante ST Microelectronics (2014a, 2014b) como melhor opção para dar-se continuidade à tendência redução dos nodos das tecnologias de fabricação, uma vez que tecnologias concorrentes possuem custo de fabricação maiores (JONES, 2012).

A fabricação eficiente de circuitos integrados digitais não depende somente da tecnologia utilizada para fabricá-los. As etapas previas à fabricação, desde a especificação do circuito que se deseja projetar, até a forma de organização das camadas de metal e silício que constituirão o circuito, também tem papel fundamental na obtenção de circuitos integrados eficientes.

Tornar sistemática essa sequência de etapas de pré-fabricação, referidas comumente como fluxo de projetos de circuitos integrados, é condição essencial à produção de CIs em larga escala. Ferramentas CAD comerciais são utilizadas em cada uma das etapas do fluxo de projeto de circuitos integrados com essa finalidade.

Em projetos de CIs complexos, é comum reutilizar partes de circuitos previamente projetados, incorporando-os ao circuito que se deseja projetar em alguma das etapas do fluxo de projeto. Esses blocos reutilizáveis, que implementam lógicas específicas, são denominados IP cores, do inglês *Intelectual Property cores*, e estão cada vez mais presentes nos projetos de circuitos integrados digitais. Em alguns casos, os IPs são adquiridos de terceiros, podendo ser adquiridos diretamente da indústria que fabricará os CIs ou de outros fornecedores de IPs.

Nesse contexto, este trabalho tem como objetivo propor um fluxo de projeto de circuitos integrados digitais, baseado nos fluxos clássicos, porém, levando em consideração as nuances da tecnologia FD-SOI 28nm. São mostrados exemplos de execução do fluxo proposto, bem como alguns casos de integração de IP *cores*.

Este documento está organizado da seguinte forma: na seção 2 o embasamento teórico necessário à compreensão da tecnologia FD-SOI é apresentado. A seção 3 descreverá o fluxo de projeto de circuitos integrados digitais proposto e a seção subsequente mostrará exemplos de circuitos que passaram pelas etapas do fluxo proposto. Por fim, a seção 5 tratará das conclusões deste trabalho. Os códigos fonte utilizados na descrição de hardware dos circuitos da seção 4 encontram-se nos apêndices deste documento.

2 A TECNOLOGIA FD-SOI

A tecnologia SOI (*Silicon On Insulator*) de fabricação de semicondutores, industrializada desde o final do século XX, usa um substrato silício-isolante-silício (uma camada de óxido de silício isolante é inserida entre duas redes monocristalinas de átomos de silício) ao invés do tradicional substrato de silício dos circuitos integrados convencionais, comumente chamados tipo bulk (CELLER; CRISTOLOVENEAUNU, 2003).

A tecnologia FDSOI (*Fully-Depleted Silicon On Insulator*), por sua vez, faz uso de wafers SOI aperfeiçoados a fim de criar transistores de efeito de campo (FETs) que possuam canais de condução completamente depletados. Ser completamente depletado significa ter a camada de silício sobre isolante com espessura igual à profundidade da região de depleção (SINGH; SAXENA; RASTOGI, 2011).

A Figura 1 e Figura 2 comparam a tecnologia SOI com FD-SOI e FD-SOI com a tradicional tecnologia tipo bulk, respectivamente.

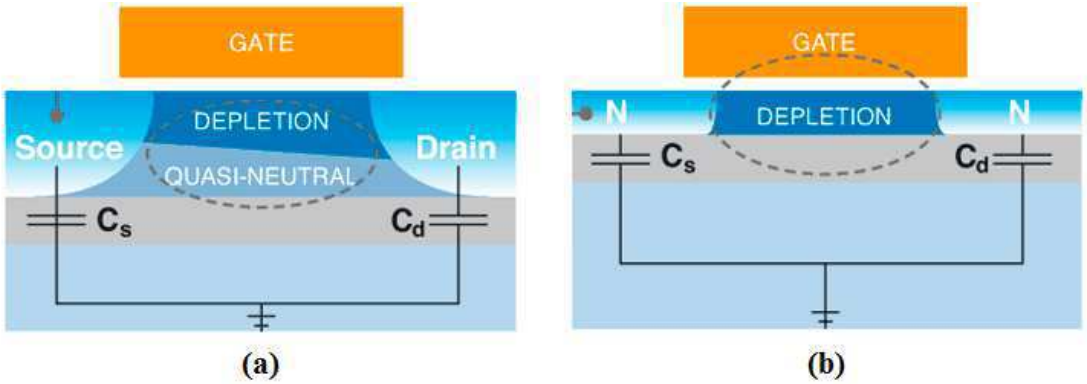

Figura 1. Corte transversal de transistores em tecnologia SOI parcialmente depletado (a) e FD-SOI (b). (MENTOR GRAPHICS, 2008)

Na Figura 1 pode-se identificar o canal de condução formado nos dois transistores (SOI a esquerda e FD-SOI a direita) na camada superior de silício – região em azul escuro localizada entre a fonte (*source*) e o dreno (*drain*).

Na Figura 1a) observa-se um transistor com canal parcialmente depletado, ou seja, neste há condução apenas em parte do silício localizado entre a fonte e o dreno. Em outras palavras, o campo elétrico formado na região do canal não é suficientemente forte para

que haja uma inversão completa do tipo de portadores majoritário nesta área. Tal fato pode ser explicado pela forma como se dá a fabricação de semicondutores SOI. Nos primórdios do processo de fabricação dessa tecnologia, não era possível obter-se espessuras de camadas de cristais de silício e de óxido enterrado suficientemente finas para a formação de uma região de depleção completa.

Com o desenvolvimento de um novo processo de fabricação, chamado de *SmartCut*, finas espessuras de silício (entre 5 e 50nm) são factíveis (HARS, 2012). Isso permite a formação de um campo elétrico mais forte em toda a região inter-óxidos, o que, por sua vez, permite a obtenção de canais de transistor completamente depletados, o que deu origem a tecnologia FD-SOI.

A Figura 1b) exemplifica a tecnologia supracitada. Nela pode-se observar que o canal de condução ocupa toda a região inter-óxidos (região entre a fonte e o dreno), ou seja, obtém-se uma depleção completa dessa região, daí o nome *Fully-Depleted*.

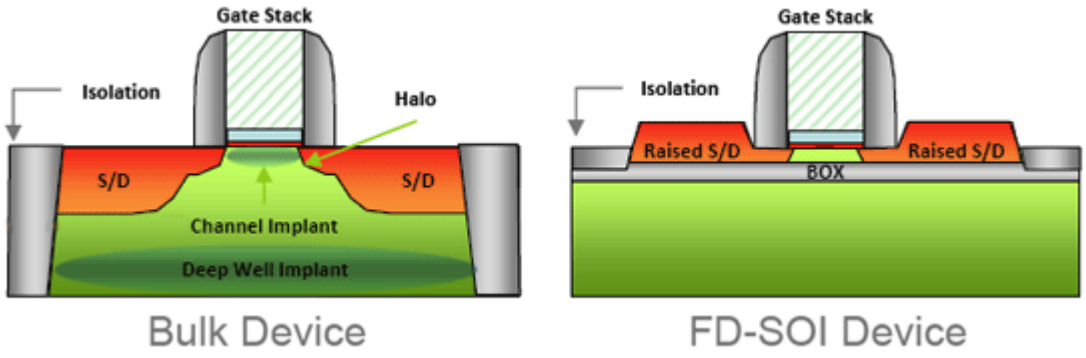

Figura 2. Seção transversal de transistor em tecnologia tipo bulk, à esquerda, e em tecnologia FD-SOI, à direita. (THINGS2DO, 2013)

Na Figura 2 observam-se as diferenças entre a convencional tecnologia tipo bulk e a tecnologia FD-SOI. As diferenças são basicamente a inserção de uma camada fina de óxido enterrado (BOX) no transistor FD-SOI; no transistor tipo bulk a camada mais profunda do substrato passa por um processo de dopagem (*Deep Well Implante*); e as camadas de silício que formar a fonte/dreno são elevadas nos transistores produzidos em tecnologia FD-SOI.

2.1.1 ÓXIDO ENTERRADO

Na tecnologia FD-SOI 28nm, um transistor possui um canal de espessura máxima de aproximadamente 7nm. O canal do transistor e o substrato são separados por uma pequena camada de óxido enterrado (do inglês, *Buried Oxide*), cuja espessura é de aproximadamente 25nm. Essa camada recebe o nome, em alguns trabalhos da literatura científica, de *Ultra Thin Body & Buried Oxide* (UTBB).

Em alguns trabalhos da literatura especializada, o substrato na tecnologia FD-SOI, região abaixo da camada de óxido enterrado, é também chamado de Back-Plane (BP), ou, em português, plano de fundo (HAMON, BEIGNE, 2003). Desta forma, é possível diferenciá-lo do substrato de transistores fabricados em tecnologia bulk, nos quais o substrato é denominado de *body*. A presença de uma camada isolante entre o canal e o plano de fundo permite fazer 3 diferentes constatações:

- Há uma certa dificuldade na circulação de corrente elétrica entre o plano de fundo e a Fonte/Dreno, devido à presença do óxido enterrado (CRISTOLOVENEAUNU, LI, 1995; CELLER, CRISTOLOVENEAUNU, 2003.);
- Não há necessidade de dopagem da camada de silício onde se forma o canal do transistor, o que reduz o fenômeno de *Random Dopant Fluctuation* (RDF), variação da concentração de impurezas dopantes no silício (HAMON, BEIGNE, 2013; ST MICROELECTRONICS 2013; NAZAROV et al., 2011). O ajuste da tensão de limiar, em contrapartida, é controlado pelo metal da porta (NAZAROV et al., 2011);
- O tipo de impureza dopante no plano de fundo não precisa ser oposto ao utilizado na dopagem do silício das regiões que formam a fonte e o dreno. Pode-se obter, portanto, uma inversão do estilo de dopagem usado em FETs do tipo bulk. (HAMON, BEIGNE, 2003; JACQUET 2013).

Como consequência direta da primeira observação feita, há uma redução na corrente de fuga entre fonte/dreno e o plano de fundo. Desta maneira, é possível dizer que o consumo do dispositivo é menor, devido à menor quantidade de perdas por correntes de fuga entre substrato, canal, Fonte e Dreno.

Randon-Dopant Fluctuations são as variações na concentração de impurezas implantadas, intrínsecas ao processo de fabricação dos transistores. Esse fenômeno é apontado como um dos principais fatores que provocam variações da tensão de limiar de um transistor para outro. Como a quantidade total de átomos que constituem o canal de um transistor vem tornando-se cada vez menor, à medida que novas tecnologias vem sendo desenvolvidas, uma pequena variação na concentração de impurezas, entre um transistor e outro, tem um impacto significativo na performance do dispositivo.

Como o silício da região que forma o canal dos transistores FD-SOI não é dopado, efeitos de RDF são eliminados (HAMON, BEIGNE, 2003).

A respeito da inversão de dopagem comentada na terceira observação, pode-se afirmar que tal capacidade possibilita mudanças na amplitude da tensão de alimentação dos transistores FD-SOI. Esta tensão influencia a Tensão de Limiar, a qual será tratada com maiores detalhes na subseção seguinte.

2.1.2 TENSÃO DE LIMIAR (V_{TH})

A Tensão de Limiar, do inglês, *Threshold Voltage*, pode ser definida como a menor diferença de potencial elétrico entre fonte e dreno necessária para a formação de um canal de condução entre eles. O fator de substrato (*body-effect coeficient*), por sua vez, é uma medida do grau de influência da polarização do substrato do transistor (*Body Biasing*, BB) na sua Tensão de Limiar (RABAEY, 2003).

Para compreender melhor tal influência e sua relação com outras variáveis, pode-se observar a equação que define a Tensão de Limiar, obtida a partir do modelo de Shichman-Hodges:

$$V_T = V_{T0} + \gamma(\sqrt{|-2\emptyset_F + V_{SB}|} - \sqrt{|-2\emptyset_F|}) \quad (1)$$

Na qual:

V_T é a Tensão de Limiar;

V_{SB} é a Diferença de Potencial entre o Contato de Substrato (*Body Contact*) e o Contato da Fonte (*Source Contact*);

V_{T0} é Tensão de Limiar para $V_{SB} = 0$. Normalmente dependente do processo de fabricação;

$\emptyset_F$ é o Potencial de Fermi;

γ é o chamado coeficiente de corpo, fator de substrato ou *body-effect coefficient*.

Observar-se que, para transistores em tecnologia convencional, a tensão de limiar é positiva para transistores MOS do tipo N (NMOS) e negativas para transistores do tipo P (PMOS) (RABAEY, 2003). Observa-se também que quanto maior for o fator de substrato (γ), maior será a influência da tensão de substrato na Tensão de Limiar.

Um estudo de Cattaneo (2009) sobre o comportamento de transistores em tecnologia SOI mostrou que o efeito de corpo das tecnologias SOI é menor do que em outras tecnologias devido principalmente à espessura do óxido enterrado. Porém, transistores em tecnologia SOI com óxido enterrado ultrafino possuem efeito de corpo mais expressivo do que outros tipos de transistores de tecnologias SOI precedentes (HAMON; BEIGNE, 2003; OHTOU; SARAYA; HIRAMOTO, 2008; NOEL et al., 2011).

O impacto da espessura do óxido enterrado fica evidente no estudo comparativo, feito por Liu et al. (2011), entre o efeito de corpo para BOXs com 10 e 25nm, Figura 3 (a) e Figura 3 (b), respectivamente.

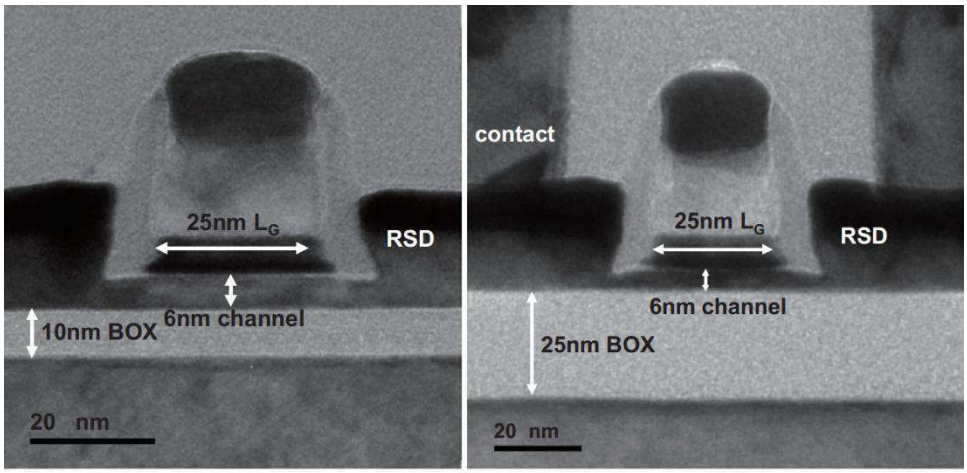

Figura 3. Transistores FD-SOI com óxido enterrado de 10nm (a) e 25nm (b). (LIU et al., 2011)

As medições de tensão de limiar obtidas no referido estudo foram:

- 120 mV/V, para um BOX de 10nm;
- 60 mV/V, para um BOX de 25nm.

Analisando-se o comportamento de um transistor com relação à tensão de limiar, pode-se dizer que quanto menor for a magnitude desta, mais fácil será levar o transistor a um estado de condução, visto que tensões de porta menores provocarão a formação de

um canal de condução. Com isso, pode-se afirmar que nesse caso os transistores tornam-se mais rápidos. Porém, uma tensão de limiar baixa resulta em mais perdas por corrente de fuga.

Por outro lado, o aumento da tensão de limiar faz com que a barreira de potencial a ser vencida para formação de um canal de condução no transistor seja maior que no caso anterior. Sendo assim, torna-se mais difícil que o transistor entre em estado de condução, o que acaba por reduzir as perdas por corrente de fuga. Todavia, neste caso percebe-se que os transistores se tornam mais lentos.

Vale salientar que a inversão do BP, comentada na subseção anterior, permite uma configuração antes impossível em tecnologias MOS anteriores. Um canal com mesmo tipo de dopante que a fonte e o dreno seria um curto circuito e não um transistor. Essa inversão do BP é chamada de *Flip-Well* (FW), ou Poço Invertido. Um comparativo entre esse tipo de BP e o convencional, tal como na tecnologia bulk, é apresentado na Figura 4. Nessa figura também pode-se observar as margens de polarização para a tecnologia FD-SOI 28nm.

Existe ainda um terceiro tipo de poço, o *Single-Well*. Nos transistores com esse tipo de BP a mesma impureza dopante é usada para formar os substratos dos transistores PMOS e NMOS. Com um poço único, aumentar a tensão de limiar para um tipo de transistor, reduziria a tensão de limiar para o seu complementar. Esta configuração de BP não será detalhada neste trabalho.

Observa-se do lado direito da Figura 4 as amplitudes das tensões de polarização dos substratos, passíveis de serem aplicadas a cada tipo de BP. Essas amplitudes são limitadas pela corrente de polarização direta e reversa da junção PN, formada entre os substratos dos transistores PMOS e NMOS. Esses limites de tensão de polarização sã válidos para a tecnologia FD-SOI 28nm.

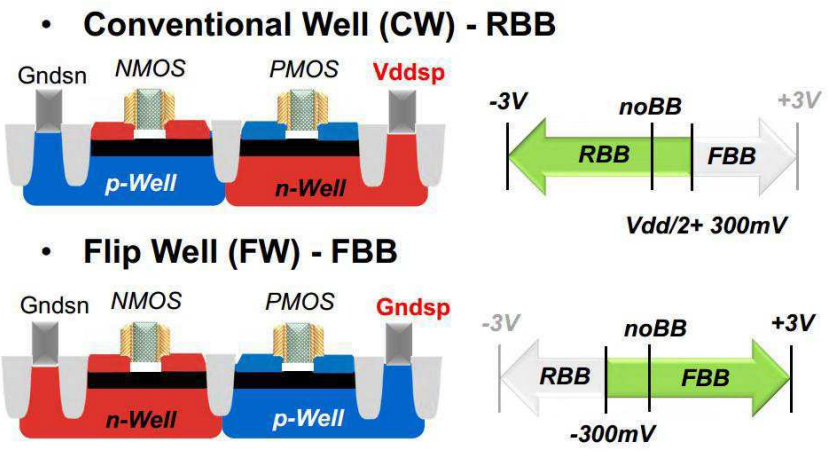

Figura 4. Transistores FD-SOI com poço convencional, acima, poço invertido, abaixo, e suas respectivas margens de tensão de polarização do BP, à direita. (FLATRESSE, 2013)

Vale salientar que o esquema de polarização do BP pode ser manipulado para tornar o circuito mais lento ou mais rápido, de acordo com a necessidade e com a limitação da configuração de poço escolhida (convencional ou invertida).

Com um esquema de polarização FBB (*Foward Body Biasing*), há uma redução na tensão de limiar, o que torna os transistores mais rápidos, a custo de um aumento na corrente de fuga, conforme comentado anteriormente. Adotar esse tipo de polarização em transistores com substrato convencional é permitido até cerca de 300 mV, conforme estudos feitos por Flatresse e Jacquet (2013). Contudo, conforme visto na Figura 4, esse esquema de polarização é mais adequado para transistores com substrato invertido, dado que nesse caso há uma faixa de tensão maior que polariza inversamente a junção PN formada entre os poços dos transistores PMOS e NMOS.

Analogamente com esquema de polarização RBB (*Reverse Body Biasing*), há um aumento da tensão de limiar, o que torna os transistores mais lentos, porém com menor corrente de fuga. Esse esquema de polarização é mais adequado para transistores com substrato convencional, pois nesse caso há uma faixa de tensão maior que polariza inversamente a junção PN formada entre os poços dos transistores PMOS e NMOS, conforme visto na Figura 4.

É possível, também, implementar esquemas de polarização que hora deixem o circuito mais rápido, para obter-se um melhor desempenho enquanto o circuito estiver ativo, hora deixem o circuito mais lento, com a finalidade de reduzir as perdas de energia, enquanto o circuito não estiver sendo utilizado. Tal estratégia é sugerida pela ST Microelectronics e apontada como uma das vantagens da tecnologia FD-SOI.

2.1.3 POLYBIASING

Uma outra característica peculiar da tecnologia FD-SOI é o *polybiasing*. Essa técnica consiste em alterar, em tempo de projeto, o comprimento do canal do transistor, ou seja, o poli silício de porta. A Figura 5 mostra como o polybiasing altera a constituição física dos transistores.

Na Figura 5b) observa-se um exemplo de vista do transistor da Figura 5a), obtida com auxílio de uma ferramenta CAD, na qual não há polybiasing, ou seja, o transistor retratado tem comprimento mínimo de canal. As regiões em verde são a fonte e o dreno e a parte em vermelho representa o poli silício de porta. Na Figura 5c), é mostrado como é feita a variação do comprimento do canal do transistor, em relação a seu tamanho

original retratado na Figura 5b), para que se obtenha o efeito de polybiasing desejado. Observa-se que o desenho do comprimento de porta maior é feito apenas na região do canal do transistor.

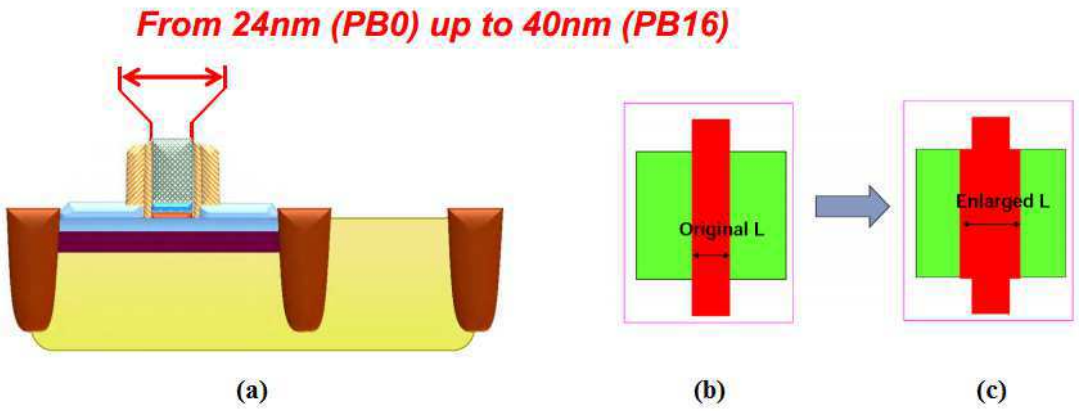

Figura 5. Corte transversal de um transistor FD-SOI (a). Variação do comprimento do canal em função do polybiasing (b) e (c). (FLATRESSE, 2013)

A variação no comprimento do canal resulta na alteração da Tensão de Limiar. A seguinte analise pode ser feita: quanto maior for o comprimento do canal, maior é o valor desta tensão, e menores as perdas por corrente de fuga *sub-threshold*. Alterar o comprimento do canal de um transistor faz com que este se comporte como um transistor fabricado em tecnologias com comprimentos de canal maiores. Sendo assim, uma das vantagens do polybiasing que pode ser apontada é que tecnologias com nodos menores geralmente apresentam melhor qualidade de fabricação que as tecnologias de nodos maiores.

Outra vantagem que pode ser apontada é que a variação dos valores de V_{th} com o uso de *polybiasing* permite a fabricação de circuitos compatíveis com arquiteturas multi-V_{th}. Além disso, conforme escolhas de projeto, pode-se usar diferentes valores de polybiasing para obter-se desempenhos e consumos diferentes.

O design kit da tecnologia FD-SOI 28nm provido pela ST Microelectronics apresenta uma biblioteca *stardard cells* com 4 opções de comprimento de canal, conforme apresentado na Tabela 1. Cada um representando, portanto, uma opção de polybiasing que varia de 0 a 16nm (P0 à P16). Seus comprimentos efetivos, no entanto, variam de 24 a 40nm.

Tabela 1. *Polybiasing*, comprimentos de projeto e comprimentos efetivos disponíveis para biblioteca de células padrão da tecnologia FD-SOI 28nm.

Polybiasing	**Comprimento de Projeto (nm)**	**Comprimento Efetivo (nm)**
P0	30	24
P4	34	28
P10	40	34
P16	46	40

Nota-se que, para a tecnologia de 28nm, o canal do transistor possui um comprimento mínimo de 30nm. Para transistores com esse comprimento de canal, o comprimento efetivo – que é a distância real entre os terminais de Fonte e Dreno é de 24nm. A diferença entre as duas distâncias deve-se à difusão dos dopantes por baixo do óxido de porta.

A existência de diferentes bases de medição para o comprimento de projeto (30nm mínimo para a tecnologia FD-SOI 28nm), o comprimento de referência da tecnologia (28nm) e o comprimento efetivo (24nm para a tecnologia FD-SOI 28nm) são resultado de decisões comerciais.

3 FLUXO DE PROJETO DE CIRCUITOS INTEGRADOS EM TECNOLOGIA FD-SOI 28NM

Propõe-se neste capítulo um fluxo de projetos de circuitos integrados, que segue o fluxo clássico, mas levando em consideração as nuances da tecnologia FD-SOI 28nm. O fluxo proposto inicia-se com a caracterização de circuitos em linguagem de descrição de hardware (HDL), a partir das especificações do projeto, e finaliza-se com a obtenção de um layout para o circuito. Ferramentas EDA comerciais são utilizadas para síntese lógica e concepção do layout.

A Figura 6 apresenta um diagrama de blocos do fluxo proposto. Primeiramente, partindo-se das especificações do cliente, uma arquitetura inicial, em nível de blocos, é projetada. Em seguida cada um dos blocos que constituem tal arquitetura é descrito, em linguagem VHDL e nível RTL.

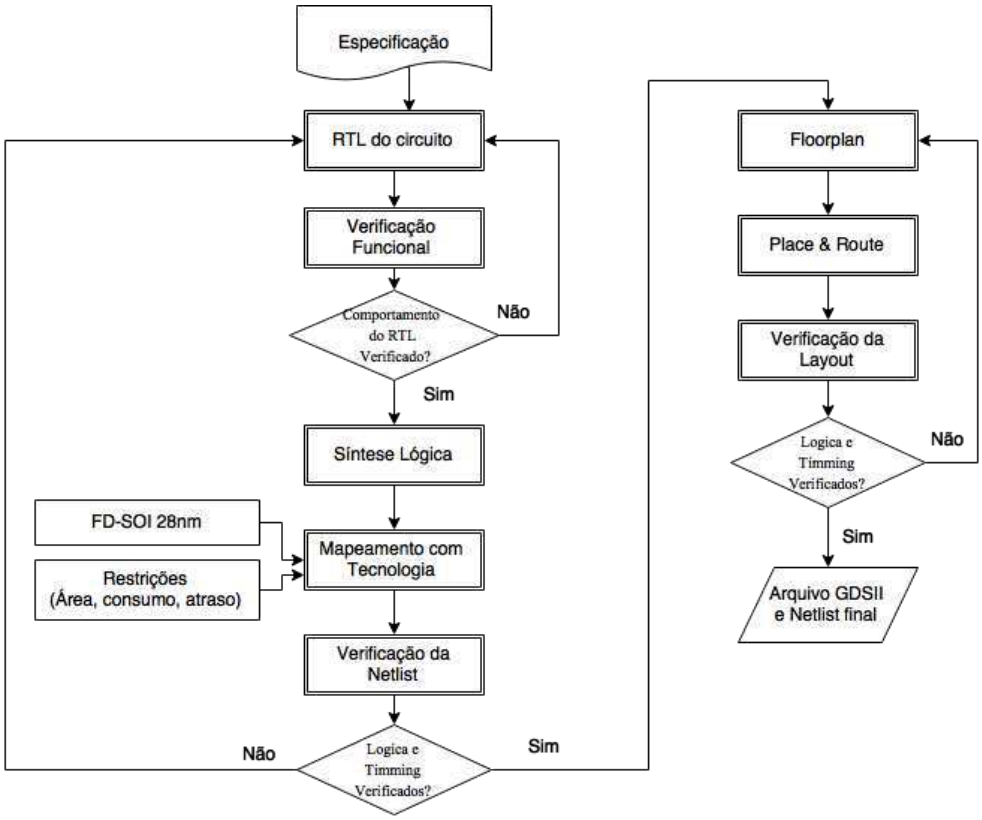

Figura 6. Fluxo clássico de projetos de circuitos integrados, adaptado à tecnologia FD-SOI

Cada bloco passa, então, individualmente por um processo simplificado de verificação funcional. Após verificação individual, a arquitetura completa também passa por um processo de verificação similar ao que foi aplicado à cada bloco. Segue-se então com a síntese lógica da arquitetura, bem como as etapas de simulação pós-síntese, concepção do layout, do inglês *place and route* (P&R), e finalmente simulação pós-layout. Cada uma das etapas do fluxo desenvolvido será descrita com detalhes nas subseções desse capítulo.

A execução das etapas do fluxo de projeto de CIs proposto, mostrado na Figura 6, é regida por uma série de códigos escritos em *shell script* e TCL, responsáveis pela configuração e controle adequado das ferramentas EDA que são utilizadas. Uma ilustração, em diagrama de blocos, da organização desses scripts e inter-relação entre eles pode ser vista na Figura 7. Nessa figura, os scripts apontados com setas tracejadas representam scripts de configuração. O script design_flow.sh corresponde ao "top" do fluxo de projeto desenvolvido e é nele que está a chamada, direta ou indiretamente, para todos os outros scripts do fluxo.

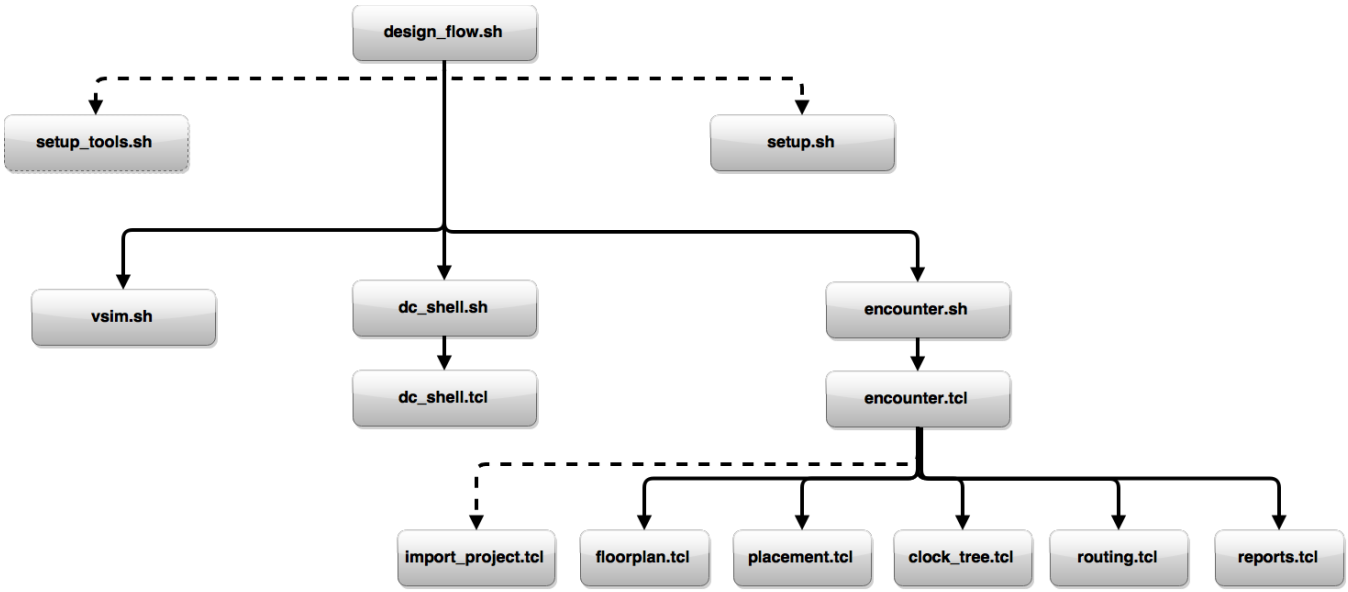

Figura 7. Diagrama de blocos dos scripts que compõem fluxo de projeto de CIs proposto.

Adotou-se um estilo de nomeação de arquivos no qual todos os scripts que tem seu nome iniciado com a palavra "setup" tem como única finalidade a configuração de variáveis de ambiente necessárias à configuração dos softwares que serão utilizados e criação de variáveis que tornam o fluxo de projeto facilmente configurável.

As subseções seguintes desse capítulo descreverão com maiores detalhes cada uma das etapas do fluxo de projeto proposto. A subseção 1 exporá maiores detalhes da arquitetura dos *testbenchs* usados para verificar os blocos da arquitetura desejada, bem como a arquitetura como um todo. A subseção 2 mostrará como funcionam os scripts responsáveis pela síntese lógica, bem como as simulações pós síntese que foram feitas. A subseção 3, por sua vez, explica como foi feito o layout da arquitetura e como os scripts funcionam em conjunto para atingir o que se espera do layout.

3.1 DESCRIÇÃO DO HARDWARE E VERIFICAÇÃO FUNCIONAL DA ARQUITETURA

A primeira etapa do fluxo de CIs proposta consiste em projetar uma arquitetura, a partir das especificações do projeto, composta por blocos com funcionalidades distintas. A finalidade da concepção de tal arquitetura dividida em blocos é facilitar o processo de verificação da arquitetura (aplicação do princípio da divisão e conquista) e simplificar a divisão das tarefas entre os de designers, uma vez que cada pessoa ao grupo ficaria responsável por um dos blocos em cada etapa do projeto.

Uma vez definido cada bloco que comporá a arquitetura do circuito que se deseja projetar, cada bloco é então descrito em linguagem VHDL e nível RTL (do inglês *Register Transfer Level*). O módulo top da arquitetura em questão passa a ser, então, simplesmente um conjunto de instâncias dos blocos que o compõe.

Para verificar se o comportamento do hardware descrito está de acordo com a especificação, um ambiente de testes, em inglês *testbench*, é utilizado. Neste, uma série de estímulos servem como entrada para o circuito projetado e suas saídas são verificadas a fim de detectar possíveis erros na descrição do *hardware*. Existem metodologias de verificação funcional (como UVM e OVM) e apesar de extremamente recomendadas não foram usadas no fluxo proposto neste trabalho.

Propõe-se uma estrutura de ambiente simplificada, conforme observado no diagrama de blocos da Figura 8. Nesta estrutura o DUT, do inglês *Design Under Test*, ou, em português, arquitetura sob teste, representa o *hardware* que se deseja verificar; O modelo de referência é um componente cujo comportamento é igual ao que se espera do

DUT. Tem como finalidade gerar saídas corretas para as entradas que são introduzidas ao DUT, para que se possa, de fato, testar se o este funciona de acordo com a especificação.

O bloco “gerador de estímulos”, como o próprio nome sugere, é o componente do *testbench* responsável por gerar as entradas que serão usadas para verificar o DUT. Conforme observa-se na Figura 8, os estímulos gerados são entradas do DUT e do modelo de referência, simultaneamente.

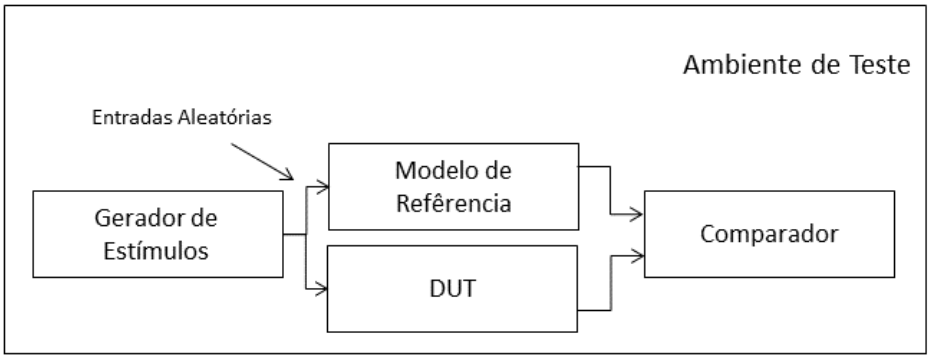

Figura 8. Estrutura do ambiente de testes simplificado proposto.

O comparador, por sua vez, tem como tarefa comparar as saídas do modelo de referência com as saídas do DUT, indicando as saídas que são diferentes entre esses dois blocos. Utilizando esta estrutura de ambiente de testes, tem-se de forma automática o número de saídas erradas, sem que seja necessário olhar as formas de onda geradas por completo.

Para que os arquivos descritos em HDL e o *testbenchs* sejam simulados, o script vsim.sh é executado. O software utilizado nesta etapa é o ModelSim, da Mentor.

Os comandos do ModelSim vcom e vopt são utilizados para compilar e otimizar os arquivos em VHDL (tanto os RTLs quanto os testbenchs), respectivamente. Formas de onda são então geradas pelo software, e podem ser visualizadas na interface gráfica do ModelSim. As formas de onda são uma importante ferramenta na depuração de erros, tanto nas descrições em nível RTL do circuito quanto no próprio testbench.

3.2 SÍNTESE LÓGICA E SIMULAÇÃO PÓS-SÍNTESE

Com a finalização do processo de verificação funcional da arquitetura desejada, tem-se um código HDL que descreve o hardware desejado que, até então, do ponto de vista comportamental, funciona de acordo com o especificado.

A próxima etapa do fluxo de projeto de circuitos integrados proposto é a síntese lógica. Segundo o manual da ferramenta de síntese Design Compiler (2011), síntese lógica é o processo de conversão de uma arquitetura, descrita em linguagem de descrição de hardware como Verilog ou VHDL, em um conjunto otimizado de portas lógicas padrão específicas de uma tecnologia. Neste trabalho, o processo de síntese lógica mapeia um hardware descrito em VHDL em um conjunto de portas lógicas padrão da tecnologia FD-SOI 28nm. O software Design Compiler, da Synopsys, foi utilizado para tal finalidade.

Um procedimento típico que a EDA supracitada utiliza para realização da síntese lógica constitui-se das seguintes etapas:

- As descrições de hardware são traduzidas em componentes extraídos da biblioteca de tecnologia genérica (GTECH), que contém portas lógicas básicas e flip-flops independentes de qualquer tecnologia;
- O conjunto de portas lógicas genéricas obtido é então otimizado e mapeado à biblioteca de uma tecnologia específica, conhecida como biblioteca alvo. Tal processo de mapeamento é baseado em especificações de tempo, consumo e área, feitas pelo projetista;
- Após otimização, segue-se o procedimento de síntese de teste. Esta etapa opcional verifica, através de lógica de teste inserida pelo projetista, se netlist obtida é testável, tornando possível a resolução precoce de eventuais problemas de teste;

O resultado obtido é um conjunto de portas lógicas otimizado bem como as conexões entre elas. A Figura 9 mostra uma visão simplificada do papel da ferramenta de síntese lógica no fluxo de projetos de circuitos integrados clássico.

O *Design Compiler* possui duas interfaces para executar síntese lógica, a saber: a interface em linha de comando (*dc_shell*) e a interface gráfica (*Design Vision*). A interface gráfica foi usada neste trabalho apenas para obtenção do esquemático pós-

síntese lógica. Todo o procedimento de configuração do *Design Compiler* e execução da síntese lógica foi feito utilizando-se a interface *dc_shell*, dirigida pelos scripts dc_shell.sh e dc_shell.tcl.

O script dc_shell.sh configura algumas variáveis de ambiente necessária ao correto funcionamento do outro script e invoca a ferramenta de síntese. Quando esta é invocada, automaticamente executa os comandos presentes no arquivo synopsys_dc.setup. Este arquivo foi previamente configurado para que o Design Compiler mapeie os arquivos HDL em portas lógicas padrão da tecnologia FD-SOI 28nm. A biblioteca dessa tecnologia permite escolher células padrão com plano de fundo convencional ou invertido, células padrão com ou sem *polybiasing* e com duas possibilidades de altura, características descritas na fundamentação teórica deste trabalho.

Outras configurações secundárias também são feitas nesse arquivo, como os caminhos dos arquivos da biblioteca de timing da tecnologia, diretivas acerca do formato do arquivo de saída, algumas condições iniciais de mapeamento, entre outras configurações.

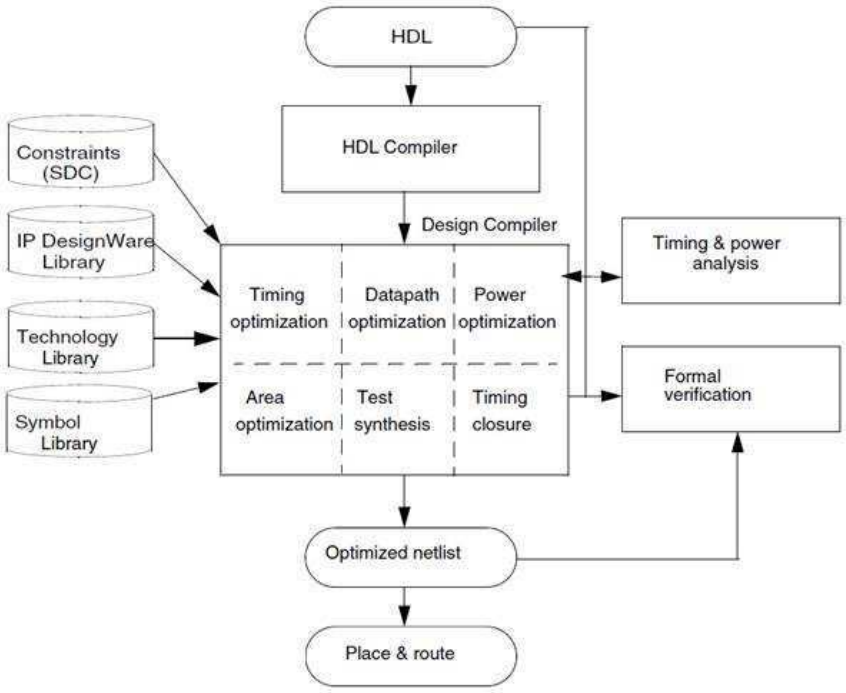

Figura 9. Função da ferramenta de síntese no fluxo de projeto de CIs clássico (SYNOPSYS, 2010).

É no script dc_shell.tcl, contudo, que estão os principais comandos que fazem com que a ferramenta de síntese lógica opere de forma a atingir os objetivos desejados.

Nesse script, as duas estratégias de compilação foram implementadas *Top down* e *Bottom up*. Essa última executa primeiro o mapeamento em portas lógicas dos blocos mais internos hierarquicamente e vai mapeando os demais elementos mais externos em seguida. Esse estilo de compilação é particularmente interessante para grandes arquiteturas, ou quando se tem uma restrição de memória. Cabe ao projetista escolher um estilo ou outro antes de rodar o script dc_shell.tcl.

Comandos de report criam relatórios com informações sobre a área total ocupada pela netlist, consumo de energia, informações sobre o sinal de relógio, entre outros relatórios. É aconselhável fazer uma análise detalhada desses relatórios a fim de saber se a *netlit* concebida atende as especificações de consumo, área, etc.

O comando write_sdc cria um arquivo em formato SDC, que contém informações de unidades de tempo, atrasos e informações a respeito do sinal de relógio desejado, além de conter informações a respeito das restrições impostas à síntese (atraso na entrada e saída, máximo *clock skew*, entre outros). Tal arquivo será lido posteriormente pela ferramenta de concepção de layout.

Por fim, o comando write_sdf é executado, gerando um arquivo em formato SDF, o qual contem informações a respeito dos atrasos nas portas lógicas e interconexões da biblioteca alvo. Esse arquivo é usado no procedimento de *back annotation*, que incorpora às formas de onda simuladas no ambiente de teste informações de atraso.

A fim de executar-se simulação pós-síntese com *back annotation*, o script vsim.sh é executado. O comando sdfcom compila o arquivo SDF gerado para que o procedimento de *back annotation* seja executado em seguida. A visualização das formas de onda contendo os atrasos relativos às portas lógicas e interconexões da tecnologia alvo é uma importante ferramenta para depuração de erros, principalmente os causados por violação de tempo (*hold time* e *setup time*).

3.3 CONCEPÇÃO DO LAYOUT E SIMULAÇÃO PÓS-LAYOUT

Seguindo o fluxo clássico de concepção de ASICs, após obtenção de um conjunto de portas lógicas interconectadas (netlist) mapeadas à uma tecnologia específica (nesse caso FD-SOI 28nm), segue-se o processo de concepção do layout. Segundo Clein (2000), concepção de um layout é o processo de criação de uma representação física precisa de

uma *netlist* que esteja em conformidade com as restrições impostas pelo processo de fabricação, pelo fluxo de projeto e pelos requisitos de desempenho.

A ferramenta CAD SoC Encounter da Cadence foi utilizada nesta etapa do fluxo proposto. Com ela é possível converter uma netlist (arquivo descrito em HDL) em um arquivo GDSII, um arquivo em formato binário que representa hierarquicamente formas geométricas e outras informações pertinentes ao layout. Vale salienta que o referido formato é atualmente um padrão para troca de informações de layouts de circuitos integrados com a indústria.

A Figura 10 mostra, em forma de diagrama de blocos, as etapas necessárias à execução do *placement* e *routing* com êxito.

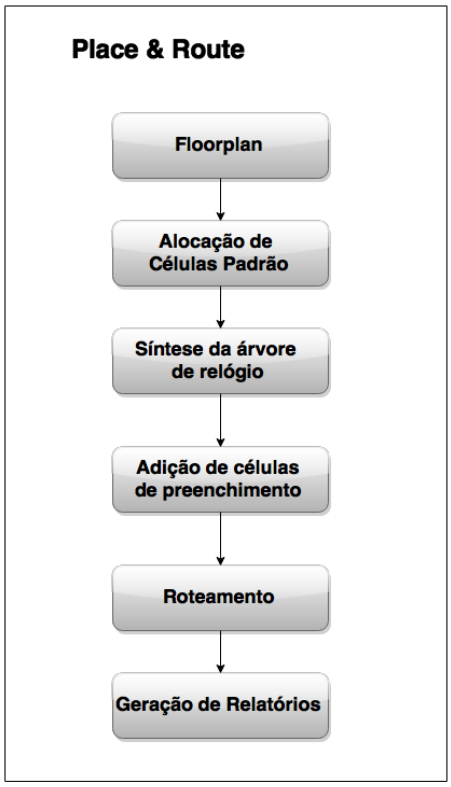

Figura 10. Etapas da concepção do layout no fluxo de projeto de circuitos integrados proposto.

Primeiramente deve-se configurar alguns parâmetros e variáveis necessários para que a ferramenta proceda como esperado, tais como o caminho da netlist obtida na etapa de síntese lógica, os arquivos da tecnologia alvo a serem utilizados, definição das bibliotecas a serem utilizadas para extração de informações de timing e capacitâncias parasitas, definição dos *constraints* (em comum com a etapa de síntese lógica), entre outras configurações.

É importante comentar que a medida que as tecnologias de fabricação de CIs vai atingindo nodos inferiores a 90nm, fenômenos como o efeito *crosstalk* e inversão de temperatura tem tornado a análise de timing cada vez mais complexa e distante da clássica análise de melhor caso e pior caso (MOLINA, 2012).

Analisando-se especificamente nodos de 28nm, as diversas variações PVT em nível de portas lógicas e interconexões causaram o aumento do número de casos passíveis de serem analisados (*corner cases*) (MOLINA, 2012). Sendo assim, torna-se importante adotar uma abordagem *Multi Mode Multi Corner* (MMMC), que torna a análise de *timing* mais precisa. Apesar das bibliotecas da tecnologia FD-SOI 28nm providas pela ST Microelectronics permitirem essa forma de análise de *timing*, tal abordagem não foi adotada neste trabalho por simplicidade, mas será mencionada na seção de trabalhos futuros.

No fluxo de projeto proposto, os scripts import_project.tcl, design.global e conf2init.viewDefinitio.tcl executam as mencionadas configurações iniciais. Vale ressaltar que é durante a escolha dos arquivos da tecnologia, dentre os disponíveis na biblioteca provida pela ST Microelectronics, que é possível selecionar células com as características desejadas da tecnologia FD-SOI, tal como baixa tensão de limiar, polybiasing ou inversão ou não do *back plane*.

Feitas as configurações iniciais, segue-se a etapa de *floorplaning*, que consiste em especificar uma tentativa inicial de posicionamento das *standard* cells quem compõe a arquitetura que passou pelo processo de síntese lógica. Nesta etapa são definidas linhas, na área delimitada que o hardware ocupará, nas quais as portas lógicas padrão serão alocadas. O script floorplan.tcl é executado a fim de realizar tal tarefa.

É também no estágio de *floorplaning* que são definias as margens do hardware (entre a área onde serão alocadas as *standard cells* e os limites do chip) e a densidade de ocupação da área do chip. A densidade especificada pode ser iterativamente incrementada à medida que se observa que a ferramenta de *place* e *route* conseguir alocar a arquitetura proposta sem violações.

Ainda na etapa de *floorplaning* são definidas as linhas de alimentação do chip (VDD e GND) em termos de parâmetros como largura, espaçamento entra elas e distância ao centro do chip. O *floorplaning* acaba com a especificação de como as linhas de potência serão conectadas a cada linha previamente definida.

O próximo passo, tal como especificado na Figura 10, é o *placement*. Seu objetivo é alocar as células padrão da tecnologia FD-SOI nas linhas pré-definidas na fase de

floorplaning. A ferramenta procederá de forma a alcançar a densidade pré-estabelecida, as configurações de placement definidas, as condições de *timing*, entre outras condições que podem ser definidas.

No fluxo proposto, o script placement.tcl executa os comandos que conduzem uma alocação das células padrão de acordo com o esperado. A ferramenta automaticamente interliga as *standard cells* após o placement. Esse processo automático de interconexão chama-se *pre-routing* e um algoritmo mais avançado fará uma interconexão mais adequada em um momento posterior.

Uma vez feita a alocação das células padrão, a próxima etapa a ser executada é a síntese da árvore de *clock* (*clock tree synthesis*). Essa etapa é crucial para o projeto de CIs pois o clock máximo com o qual um determinado hardware é capaz de operar corretamente depende do *clock skew*, que é consequência direta da forma como a ferramenta vai interligar as entradas de *clock* dos blocos sequenciais. Além disso o *clock* representa uma parte significativa do consumo total dos circuitos síncronos e as linhas que carregam o sinal de *clock* introduzem ruído que pode causar interferência no funcionamento de algumas partes do hardware, o que pode, por vezes, exigir uma blindagem da árvore de *clock*.

O script clock_tree.tcl é executado com a finalidade de gerar a árvore de clock. Nesse script especificam-se alguns parâmetros, dentre os quais o que especifica a que um espaço extra deve ser adicionado entre as linhas de *clock* e o que define que a ferramenta poderá redimensionar inversores e buffers, para obter resultados otimizados. Também são especificados buffers, conforme recomendação da ST Microelectronics, a serem utilizado na síntese de árvores de relógio equilibradas e com o menor valor possível de *clock skew*, dependendo do tipo de células padrão a serem utilizadas (baixa tensão de limiar ou *polybiasing*, por exemplo).

A ferramenta de concepção de layout utilizada automatiza o procedimento de síntese da árvore de relógio, levando em consideração as configurações citadas no parágrafo anterior. As restrições contidas no arquivo SDC gerado pela ferramenta de síntese lógica também são levadas em consideração. Automaticamente após a síntese das linhas de *clock*, relatórios de *timing* e *skew* são elaborados pela ferramenta.

Uma vez que as células padrão da tecnologia já foram alocadas nas linhas predeterminadas, as linhas de VDD e GND já foram criadas e o procedimento de elaboração da árvore de *clock* foi finalizado, segue-se com o próximo passo necessário a criação do layout final que é o procedimento de adição de células de preenchimento, em

inglês *filler cells*. Esse passo do processo de concepção do layout consiste em preencher os espaços vazios existentes entre as *standard cells* objetivando a conexão adequada das linhas de VDD e GND nas linhas predefinidas no *floorplaning*, evitar problemas de planificação das camadas de metal e tornar o layout manufaturável.

No caso da tecnologia FD-SOI, as células de preenchimento são também importantes no estabelecimento de esquemas de polarização adequados (RBB ou FBB, conforme especificado no capítulo anterior deste trabalho).

As células de preenchimento, adequadas à cada esquema de polarização, são especificadas na documentação específica das bibliotecas da tecnologia FD-SOI 28nm, fornecidas pela ST Microelectronics. O script filler_cells.tcl executa os comandos necessários para a adição das filler cells adequadas ao layout, conforme sugerido na documentação da biblioteca da tecnologia FD-SOI 28nm.

Em seguida as células padrão são conectadas conforme especificado na *netlist* gerada pela ferramenta de síntese. Essa etapa denomina-se roteamento, em inglês *routing*.

Inicialmente o script routing.tcl configura alguns parâmetros de otimização, como por exemplo a densidade máxima para o roteamento, esforço de otimização, esforço de redução de consumo de energia estática e dinâmica. Feito isto a ferramenta de roteamento da Cadence, nanoRoute, automaticamente conecta as entradas de cada célula padrão de acordo com as premissas pré-estabelecidas na fase de configuração.

Nesse momento faz-se uma análise pós-layout de timing seguida de otimização de roteamento com a finalidade de corrigir quaisquer violações de timing que possam ter ocorrido após o primeiro roteamento e adição de filler cells. Este tipo de otimização denomina-se *in-place routing optimization*. Nessa etapa de otimização a ferramenta de concepção de layout pode efetuar redimensionamento de buffers e inversores, bem como refazer algumas interconexões entre as células padrão para corrigir eventuais violações de timing que possam ter ocorrido.

As etapas que se seguem não mais modificam o layout obtido até então, mas são importantes para obtenção de relatórios pertinentes à analises de desempenho do layout obtido e geração de arquivos necessários para simulação pós-layout.

O script design_check.tcl executa primeiramente uma análise DRC, para certificar-se de que não há nenhuma violação de regras de design. Qualquer violação de espaçamento ou demais regras de design específica da tecnologia alvo, que eventualmente tenha persistido às otimizações, é então reportada e pode ser consultada no relatório de violação DRC criado por esse script.

Em seguida uma verificação de conectividade detecta violações como fios não conectados, pinos não conectados, roteamento parcial e linhas não roteadas, gerando um relatório de violações. Além disso, uma análise detalhada é executada com o objetivo de detectar omissão ou inconsistência de dados presentes nas bibliotecas de tecnologia ou no design obtido, gerando assim um relatório contendo as inconsistências encontradas. A ferramenta de concepção de layout faz análises em vários níveis: Entradas/Saídas, netlist, bibliotecas físicas e de timing, nas etapas de *floorplan* e de *placement*.

As inconsistências e/ou violações encontradas nos relatórios supracitados podem ser corrigidas manualmente, caso seja possível, ou então através de correções ou mudanças nas restrições de placement e/ou routing.

Em seguida o script extraction.tcl extrai os parâmetros RC do layout obtido para fins de uma última análise de timing. O script reports.tcl executa um comando que faz com que um resumo de todas as análises feitas seja criado e por fim o script netlist.tcl gera o arquivo GDSII correspondente ao layout obtido, a netlist final, em verilog e um arquivo em formato SDF.

A última etapa do fluxo proposto neste trabalho consiste em utilizar a netlist e o arquivo SDF obtidos para realizar uma simulação comportamental pós-layout. O processo de back annotation com o arquivo SDF possibilita a visualização de possíveis erros no comportamento do hardware que podem ocorrer em função de atrasos. Vale salientar que caso algum erro comportamental seja detectado nesse nível, é recomendado mudança nos *constraints* das etapas de *place* e *route* a fim de obter-se um novo layout que esteja em conformidade com as especificações comportamentais da arquitetura em questão.

4 FORMAS DE EXECUÇÃO DO FLUXO DE PROJETO DE CIS PROPOSTO

Para o projeto completo de um chip ou IP core, por vezes é necessário integrar ao que se está projetando circuitos já prontos, a fim de reduzir a complexidade do projeto, atingir-se um maior desempenho ou menor consumo de energia, por exemplo. Dependendo da complexidade do circuito que se deseja projetar, algumas opções de integração estão disponíveis.

Neste capítulo serão discutidas cinco formas distintas de desenvolvimento de chips/IP cores. Um circuito pode ser completamente desenvolvido e sintetizado pela equipe de projeto de CIs, ou então partes específicas podem ser integradas de diversas formas ao projeto. Em alguns casos há acesso ao código fonte que descreve o hardware que está sendo integrado ao chip, que possibilita, neste caso, que algumas alterações sejam feitas no hardware antes das fases de síntese e layout.

Por outro lado, algumas vezes não se dispões de códigos fontes, não sendo possível fazer-se alterações no hardware que se deseja integrar.

Duas entre as cinco formas de projeto de chips/IP cores, comentadas nesse capítulo, serão exemplificadas. As demais serão apenas comentadas, para que haja uma compreensão de como o fluxo de projeto de CIs proposto no capítulo anterior seria modificado em função da integração desejada.

4.1 PROJETO SEM INTEGRAÇÃO DE IP DE TERCEIROS

Nessa modalidade de projeto de chips as etapas de descrição do hardware, síntese lógica e concepção do layout são completamente feitas por um desenvolvedor ou grupo de desenvolvedores, sem que haja nenhuma integração com IPs de terceiros.

Essa forma de projeto de chips/IP foi exemplificada com a concepção do layout de um banco de filtros FIR, descrito a seguir. Esse projeto passou por todas as etapas descritas no capítulo anterior, desde a descrição do hardware até a concepção do layout.

4.1.1 O CIRCUITO PROJETADO

Um banco de filtros FIR passou por todas as etapas descritas pelo fluxo de projetos proposto no capítulo anterior, a fim de exemplificar sua funcionalidade. O circuito proposto é composto por um conjunto de 16 filtros FIR de ordem 12. Um filtro FIR (*Finite Impulse Response*) é um tipo de filtro cuja resposta ao impulso tem duração finita, ou seja, sua saída vai para zero após um em tempo finito.

Esse circuito foi escolhido para exemplificar o fluxo de projeto de CIs sem integração de IPs de terceiros pois trata-se de um circuito relativamente simples, porém com ampla aplicação, principalmente na área de processamento digital de sinais.

A Figura 11 mostra um esquema, em diagrama de blocos, do circuito descrito em VHDL.

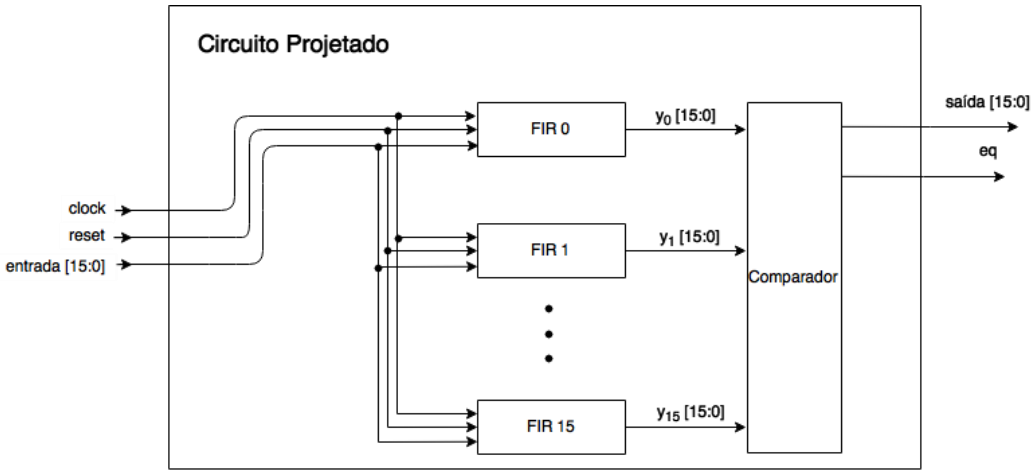

Figura 11. Diagrama de blocos do banco de filtros FIR projetado.

As saídas dos blocos FIR da Figura 11 são calculadas pela seguinte expressão:

$$y_N[n] = \sum_{i=0}^{12} b_i * x[n-1] \tag{1}$$

Na qual $y_N[n]$ representa a saída enésimo FIR no instante de tempo n; x[n-1] representa a entrada do filtro no instante n-1 e b_i representa i-ésimo coeficiente do filtro. No circuito projetado, todos os filtros têm coeficientes iguais.

Como as entradas de todos os filtros FIR que fazem parte do circuito projetado são iguais e os coeficientes dos filtros também são iguais, as saídas dos mesmos devem ser iguais, caso o circuito tenha sido corretamente descrito. A fim de facilitar a verificação comportamental, o módulo “Comparador” foi adicionado ao hardware projetado e tem como finalidade comparar as saídas dos filtros duas a duas. Se todas as saídas dos filtros FIR forem iguais, a saída “eq” do comparador permanece em nível lógico 0. Caso contrário essa saída permanecerá em nível lógico 1.

4.1.2 SIMULAÇÃO COMPORTAMENTAL

Primeiramente, descreveu-se o circuito da Figura 11 em nível comportamental utilizando linguagem de descrição de hardware VHDL (os códigos encontram-se em anexo). Em seguida, descreveu-se um ambiente de testes para a verificação comportamental do circuito.

A estrutura do *testbench* utilizado conta com um vetor de entradas, com as quais calculou-se previamente as saídas corretas em cada instante de tempo, utilizando-se a equação 1. Obteve-se, então, as formas de onda mostradas na Figura 12 e na Figura 13, geradas pelo software ModelSim.

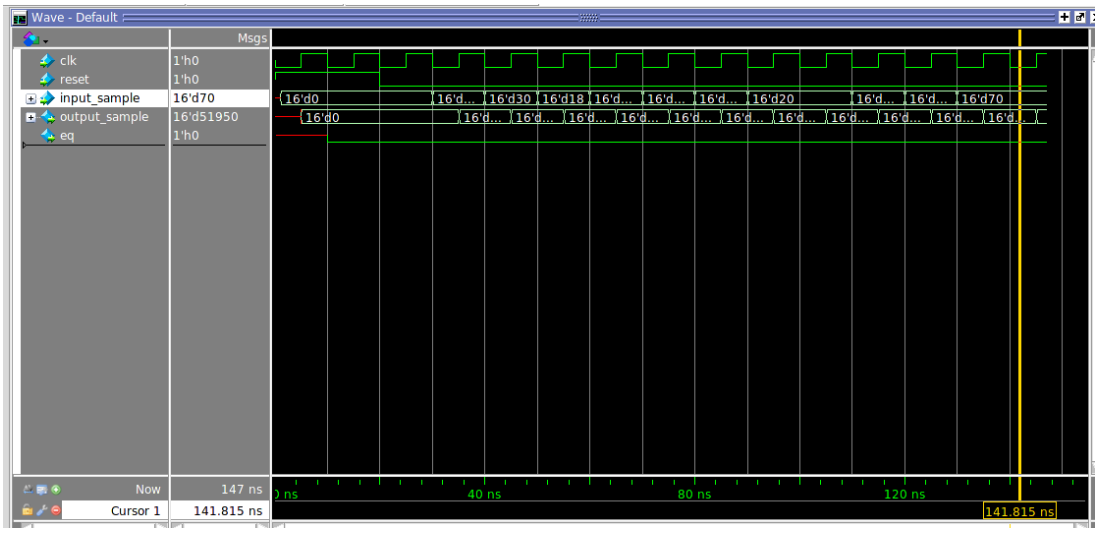

Figura 12. Simulação do ambiente de testes criado para a verificação do comportamento do banco de filtros FIR.

Observa-se que o sinal “eq” permanece em nível lógico 0 durante toda a simulação, o que significa que todos os filtros FIR estão produzindo as mesmas saídas durante todo o tempo simulado. Os valores de saída são atualizados a cada subida do sinal

de relógio, e verificou-se que estes condizem com o esperando, ou seja, o comportamento do circuito corresponde ao esperado, e deve-se passar à próxima etapa de síntese lógica.

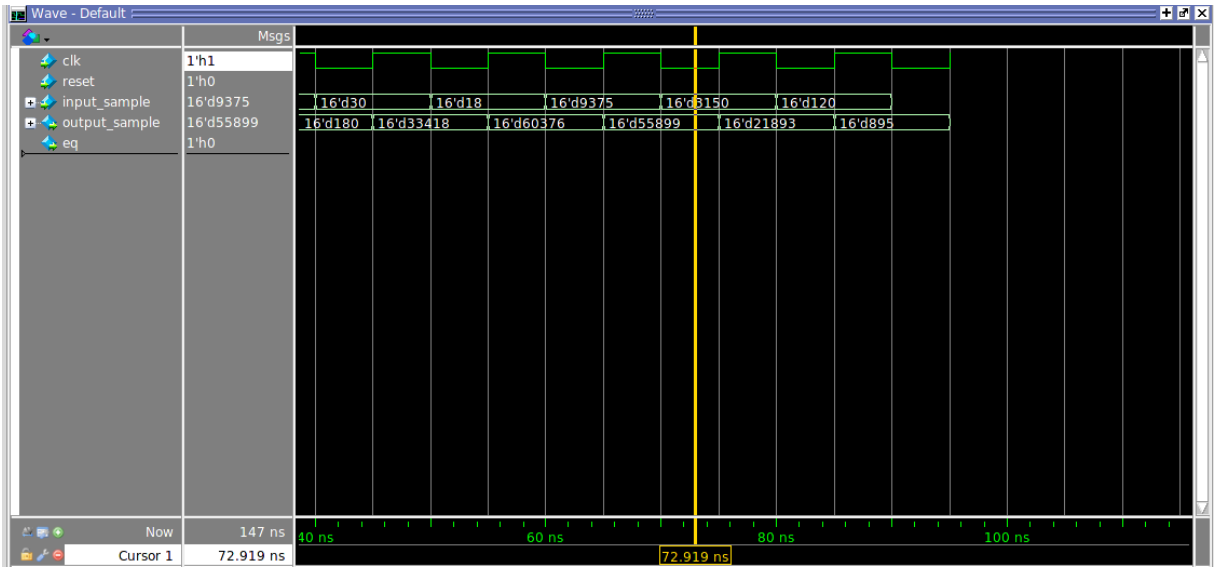

Figura 13. Formas de onda obtidas com a simulação comportamental da descrição do banco de filtros FIR (ampliação).

4.1.3 SÍNTESE LÓGICA

Para realização da síntese lógica, é necessário selecionar o tipo de células padrão FD-SOI que se deseja utilizar, em termos de tipo de plano de fundo dos transistores e *polybiasing* desejado, conforme explicado no capítulo de fundamentação teórica.

Para o circuito dos filtros FIR projetado, foram utilizadas células padrão com plano de fundo convencional e sem *polybiasing*. Além disso, para as simulações de atrasos, foram utilizadas informações disponíveis na biblioteca padrão considerando-se uma alimentação dos transistores de 1V e temperatura de 125ºC.

As restrições impostas foram frequência do *clock* 200MHz, *clock skew* máximo de 50ps e atraso na saída de 500ps. A ferramenta de síntese utilizada foi o Design Compiler, no qual é possível observar graficamente a netlist obtida, a qual pode ser vista na Figura 14, Figura 15 e Figura 16. Em termos de desempenho, a netlist gerada possui uma área total de 118437,5µm^2 e um consumo de energia total estimado de 12,5mW para uma frequência de *clock* de 200MHz.

A Figura 14 mostra uma visão geral do circuito desenvolvido, em termos de suas entradas e saídas. A Figura 15, por sua vez, mostra um diagrama esquemático no qual são mostrados os blocos que compõe o circuito projetado, bem como as interligações entre

eles. Já a Figura 16, por outro lado, mostra algumas das portas lógicas que compõe o circuito sintetizado do bloco FIR do hardware projetado.

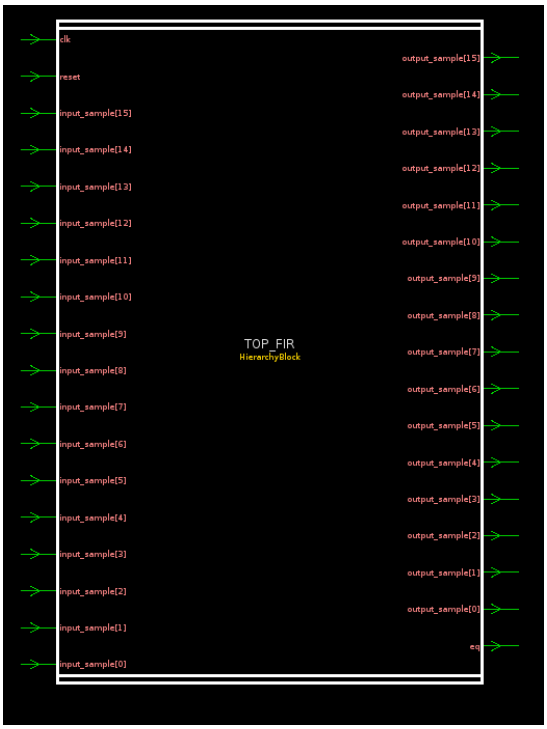

Figura 14. Módulo top do banco de filtros FIR, com entradas e saídas, gerado pelo Design Compiler.

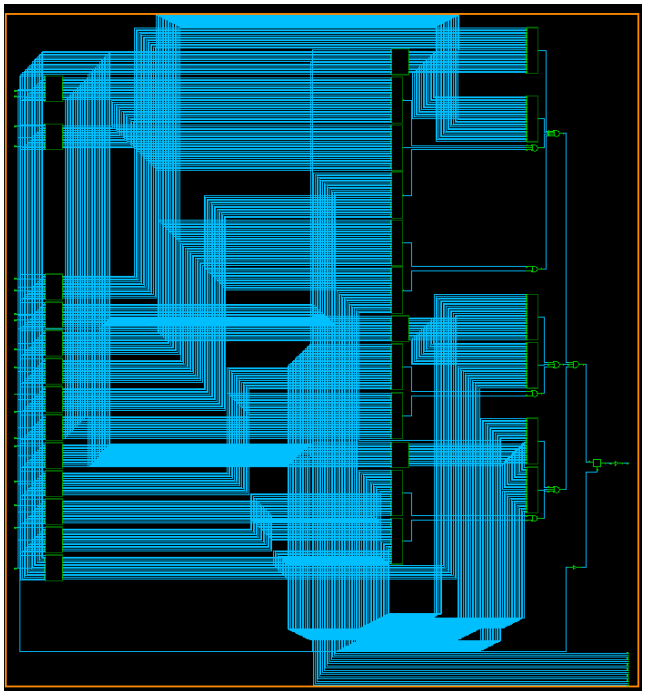

Figura 15. Esquemático do banco de filtros FIR, gerado pela ferramenta de síntese.

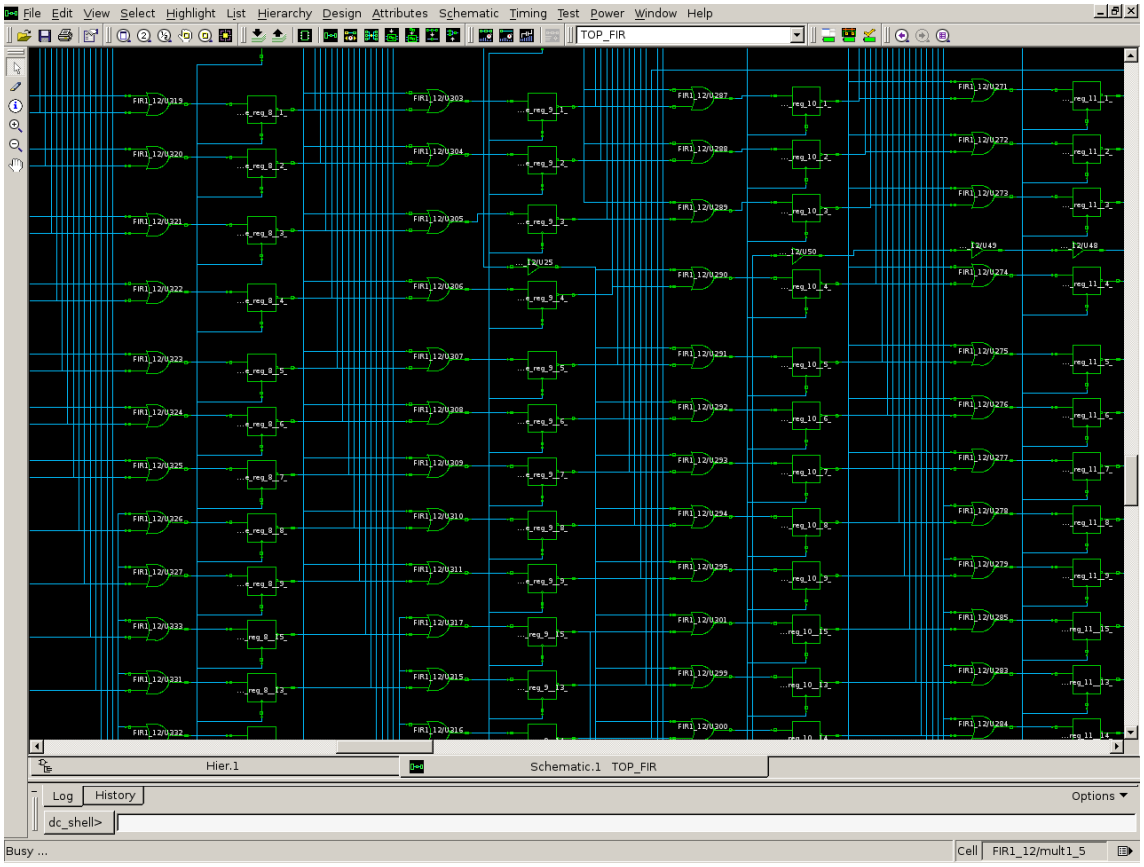

Figura 16. Portas lógicas que compõem os blocos FIR.

4.1.4 CONCEPÇÃO DO LAYOUT

Uma vez conseguida a *netlist* do circuito do banco de filtros FIR, segue-se o fluxo de projeto com as etapas de concepção do layout. As células padrão que compõem a *netlist* gerada serão, então, alocadas e interconectadas. Além da fase de organização das células padrão e interconexão das mesmas, outras etapas devem ser executadas a fim de conseguir-se um layout passível de ser fabricado.

Primeiramente, o circuito projetado teve seus *pads* de entrada/saída alocados e espaçados uns dos outros. Os *pads* são estruturas metálicas que conectam o chip aos pinos do encapsulamento. A Figura 17 mostra o layout do banco de filtros projetado, apenas com os *pads* e a delimitação da área que conterá as células padrão que compõe o circuito. Os *pads* foram escolhidos de acordo com sugestões do CMP (*Circuits Multi-Projects*), empresa que dá suporte à usuários da tecnologia FD-SOI 28nm.

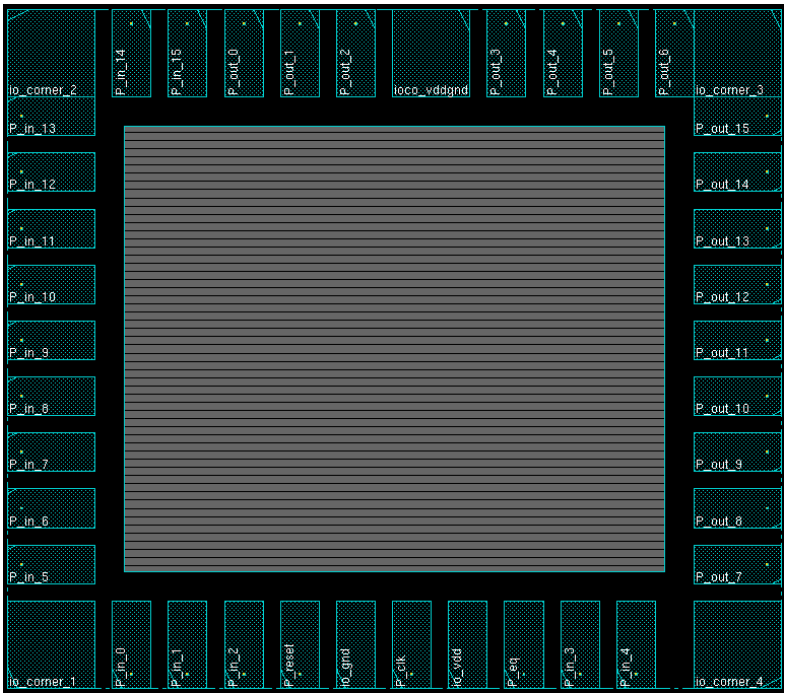

Figura 17. *Pads* do banco de filtros FIR.

Em seguida, células de preenchimento, do inglês *filler cells*, são adicionadas para preencher os espaços vazios entre os *pads*, garantindo um plano de fundo uniforme entre os *pads*. Essas células padrão também foram escolhidas de acordo com sugestões do CMP. A Figura 18 mostra o layout após adição das células de preenchimento.

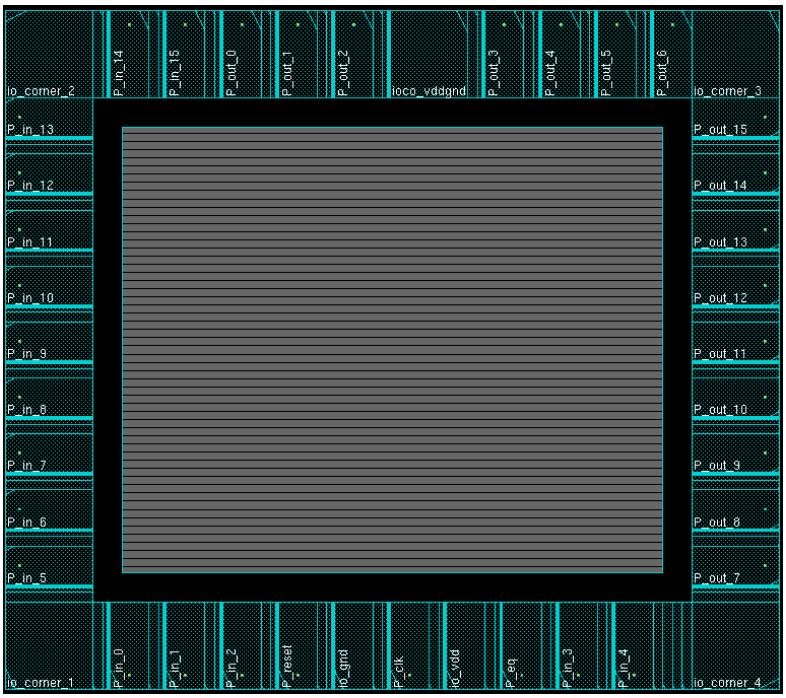

Figura 18. Layout do banco de filtros FIR após adição de células de preenchimento.

Posteriormente, segue-se a etapa conhecida como *floorplan*. Nesta, são definidas as linhas e colunas onde serão alocadas as células padrão que compõem o banco de filtros FIR. É também nesta etapa que são definidas as linhas de alimentação (VDD e VSS) que alimentarão as células padrão. A Figura 19 mostra o layout do circuito projetado após adição das linhas de alimentação.

As linhas de alimentação principais circundam a área delimitada que conterá as células padrão. Além da alimentação principal, duas faixas secundárias de VDD e VSS cortam a parte central do layout, a fim de melhorar a distribuição de energia elétrica entre as células padrão do circuito.

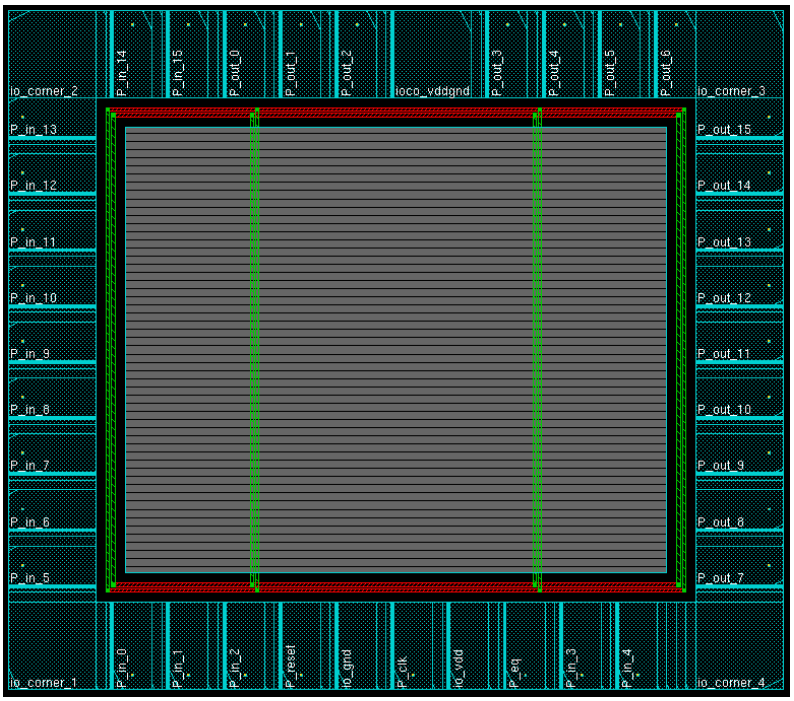

Figura 19. Layout do banco de filtros FIR após adição das linhas de alimentação (VDD e VSS).

Para evitar circulação de correntes elétricas indesejadas causadas por efeito de proximidade (do inglês, *proximity effect*), o CMP recomenda adicionar um tipo especial de célula de preenchimento às bordas superior e inferior da área destinada à alocação de células padrão. Para finaliza a etapa de *floorplan*, *well taps* são adicionados à área central do chip, espaçados uns dos outros uniformemente, conforme regra de projeto (*design rule*) definida pelo fabricante da tecnologia FD-SOI 28nm, para evitar *latchup*.

A Figura 20 mostra o layout do banco de filtros FIR após finalizada a etapa de *floorplan*.

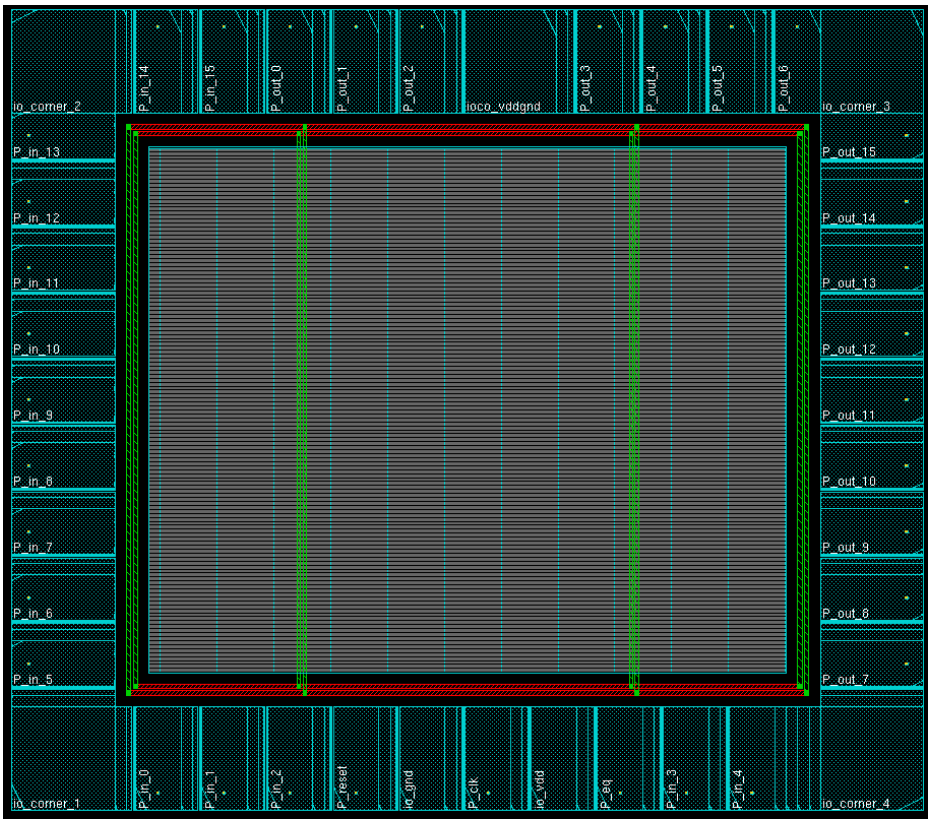

Figura 20. Layout do banco de filtros FIR com células de preenchimento para evitar efeito de proximidade e *well taps* para evitar *latchup*.

Uma vez finalizada a etapa de floorplan, as células padrão que compõem a netlist gerada são então alocadas. Essa etapa recebe o nome de placement, conforme descrito no capítulo anterior. A Figura 21 mostra o layout do banco de filtros ao fim dessa etapa.

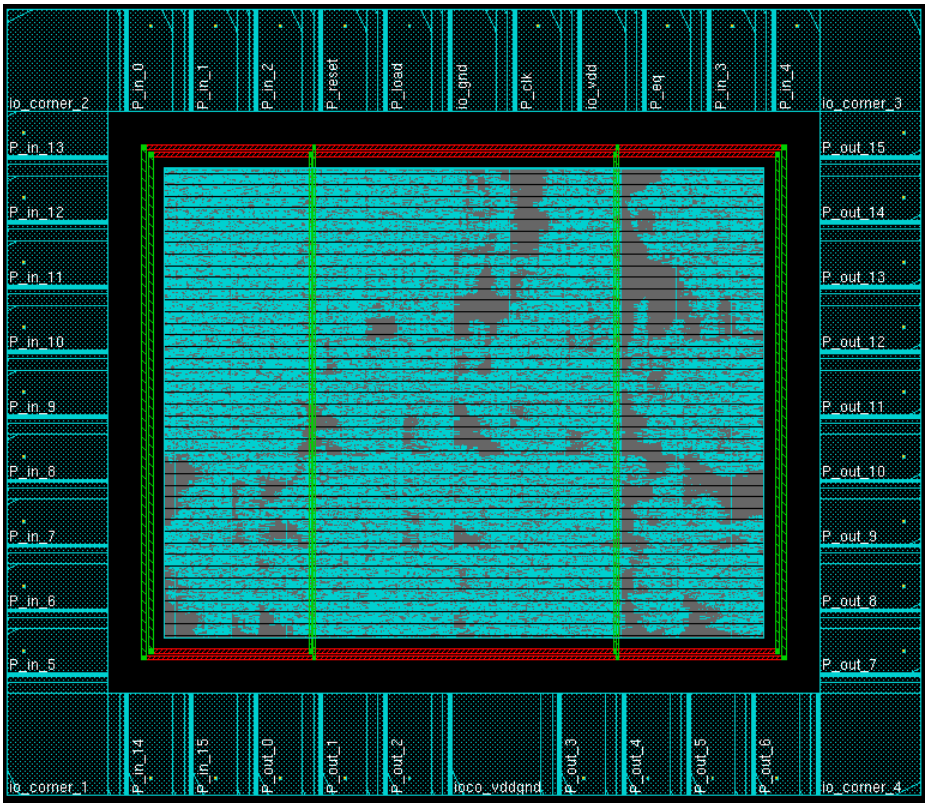

Figura 21. Layout do banco de filtros FIR após etapa de *placement*.

A árvore de relógio, do inglês *clock tree*, é então projetada de forma a atender às restrições imposta: máximo *clock skew*, *jitter* máximo, atrasos nas entradas e saídas. Para tanto, a ferramenta de layout pode alterar a posição de algumas células padrão ou adicionar buffers e inversores. A Figura 22 mostra como ficou o layout do circuito projetado após síntese da árvore de relógio.

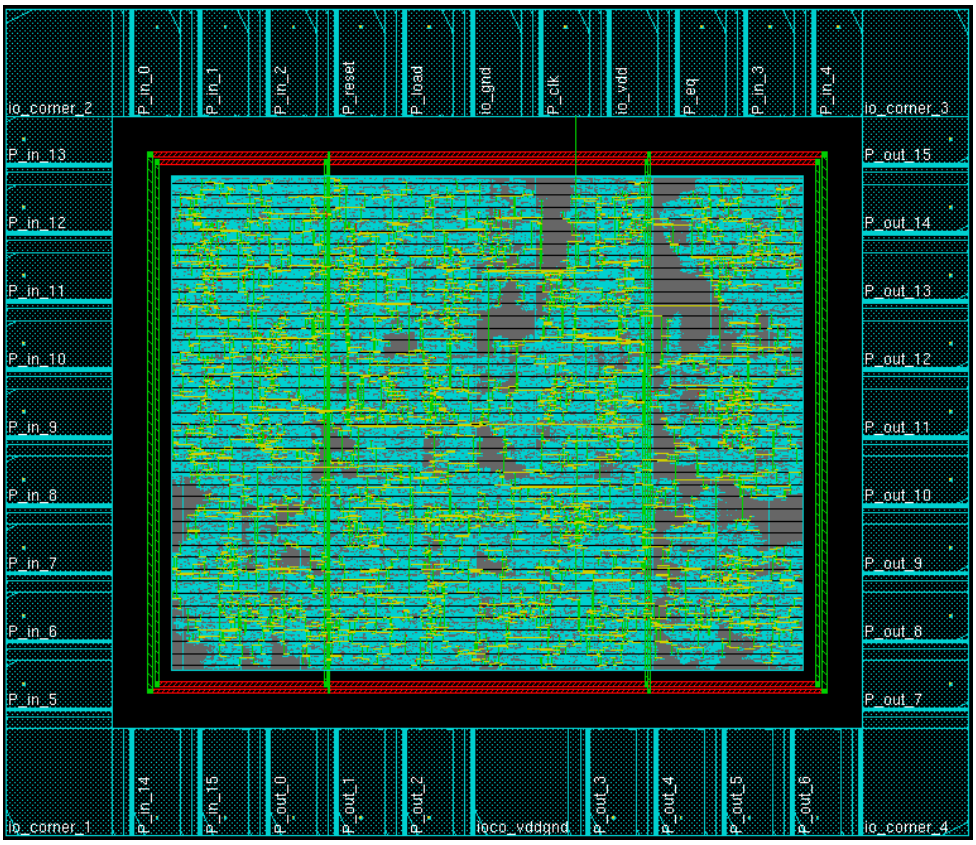

Figura 22. Layout do banco de filtros FIR após síntese da árvore de relógio.

Uma vez que as células padrão que compõe a netlist estão devidamente posicionadas e a árvore de relógio foi sintetizada, os espaços vazios entre as portas lógicas padrão devem ser preenchidos com células de preenchimento padrão, para fazer com que o plano de fundo seja contínuo e com isso a fabricação do circuito integrado mais simples, entre outros benefícios. A Figura 23 mostra o layout do circuito projetado após adição das células de preenchimento. Percebe-se que as partes em cinza da Figura 22, que representam espaços não ocupados por células padrão da netlist, foram preenchidos e são mostrados em azul na Figura 23.

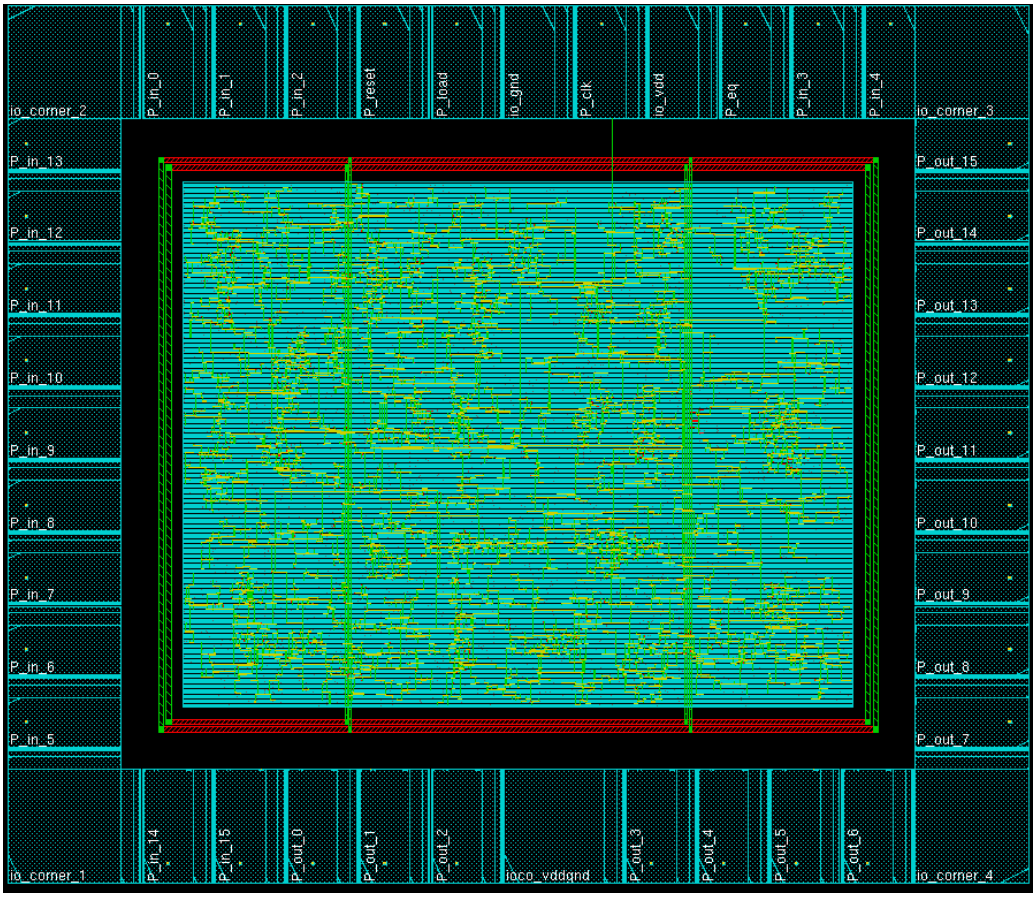

Figura 23. Layout do banco de filtros FIR após adição de células de preenchimento padrão.

Por fim, as células padrão são alimentadas com VDD e VSS e interconectadas conforme especificado pela *netlist*. O layout final obtido pode ser visto na Figura 24 e na figura 18. Uma vez que se tem o layout final, relatórios de violações são gerados a fim de verificar-se a existências de eventuais erros de conexão ou violações de alguma regra de projeto (violação de DRC). Não sendo encontradas violações, são gerados os arquivos necessários à simulação pós layout (um arquivo em *verilog* e um arquivo em formato sdf) e um arquivo GDSII que será usado para fabricação do chip.

A Figura 25 é uma ampliação da Figura 24, apenas para ilustrar melhor o resultado do roteamento e alimentação das células padrão.

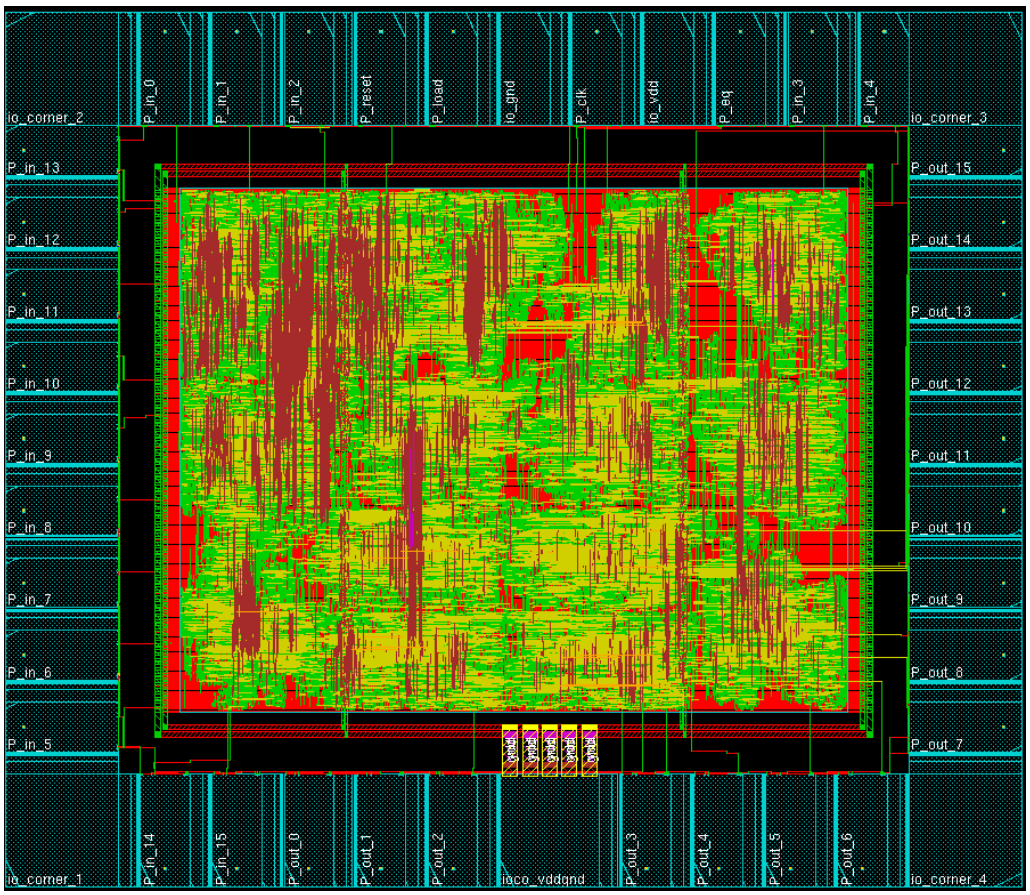

Figura 24. Layout final do banco de filtros FIR.

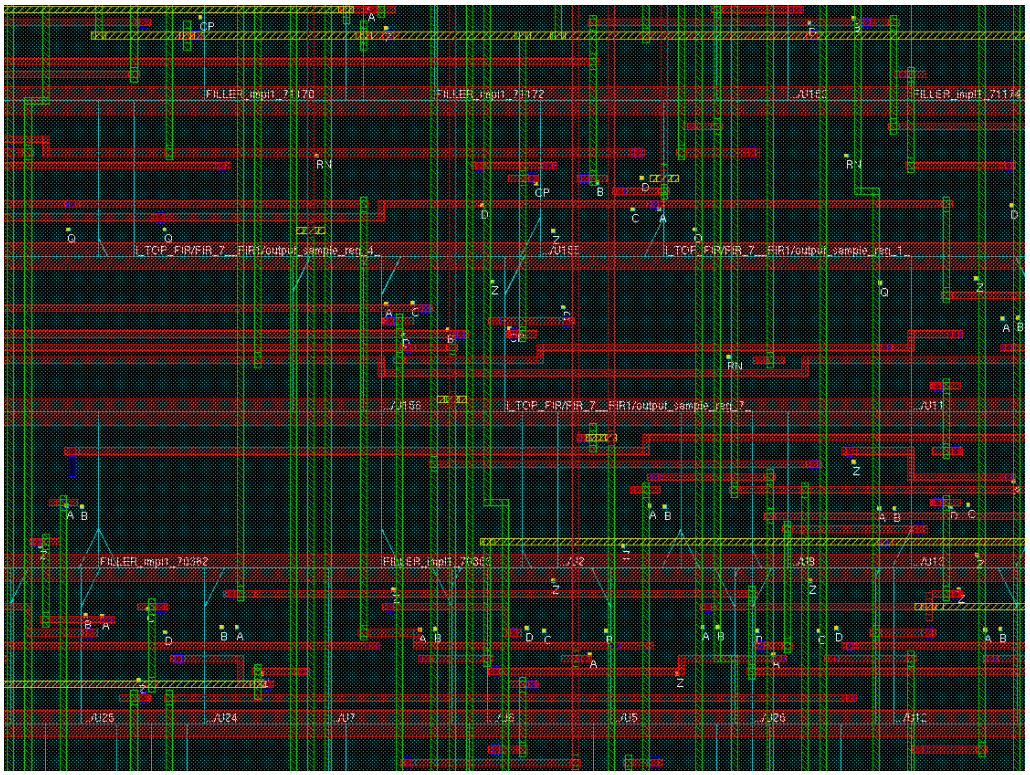

Figura 25. Ampliação do layout final do banco de filtros FIR.

4.2 INTEGRAÇÃO DE IP DE TERCEIROS COM CÓDIGO FONTE

Nessa modalidade de projeto de chips, códigos prontos contendo descrição de circuitos são utilizados como módulo top ou sub módulo do sistema que se deseja projetar. Vários tipos de IP cores tem seu código fonte disponível para download em comunidades como a "OpenCores.org", que disponibiliza hardwares código aberto.

É possível fazer alterações no hardware descrito, uma vez que se tem acesso ao código fonte. Quando o código pronto é utilizado como módulo top, as etapas de descrição de hardware e verificação funcional já estão feitas, contudo deve-se fazer síntese lógica e concepção do layout tal como descrito no capítulo anterior.

Essa forma de projeto de chips/IPs é exemplificada neste trabalho com a concepção do layout de um microprocessador de código aberto STORMcore, conseguido da biblioteca de hardwares da comunidade OpenCores.org. Neste exemplo, o microprocessador passou apenas pelas etapas de síntese lógica e concepção do layout concepção do layout.

4.2.1 O MICROPROCESSADOR STORMCORE

As características do microprocessador STORMcore são:

- Processador RISC com instruções de 32 bits;
- Opcode e funções compatíveis com a família de microprocessadores de 32-bits ARMv2;
- Execução das instruções em 8 estágios de pipeline;
- 7 modos de operação diferentes;
- Coprocessador interno;
- Memórias cache de dados e de instruções configuráveis;
- Barramento wishbone de 32 bits com pipeline;

Um diagrama de blocos da arquitetura desse microprocessador pode ser visto na Figura 26.

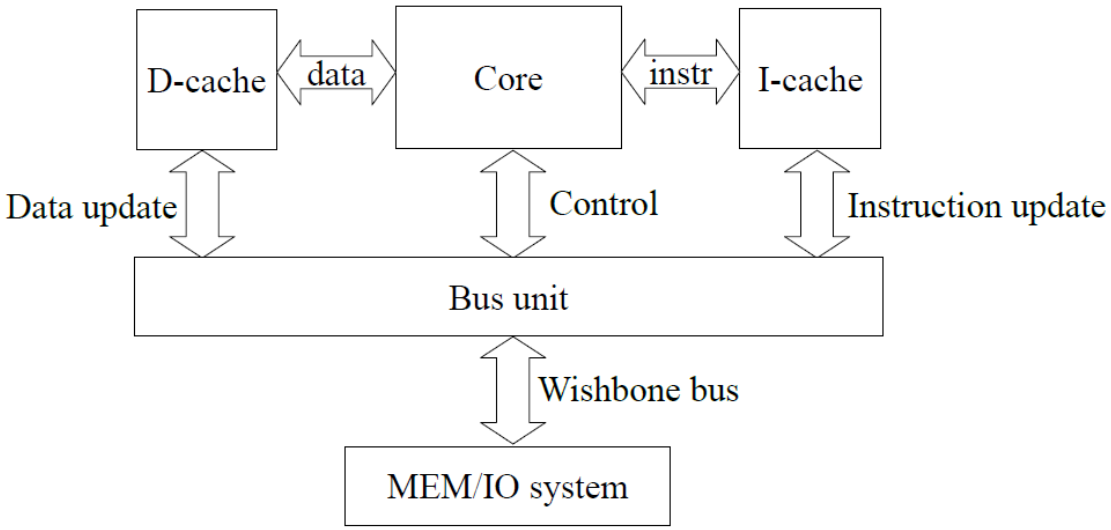

Figura 26. Diagrama de blocos da arquitetura geral do microprocessador STORMcore (NOLTING, 2012)

O bloco "Core" e composto pelos seguintes sub-blocos, com suas respectivas funções:

- ULA – Executa todas as operações lógicas e aritméticas necessárias, exceto o incremento do registrador contador de programa (PC). Além disso controla o acesso de dados ao coprocessador.
- Barrel_shifter – Executa operação de shift do segundo operador a ser usado pelo módulo ULA.
- Flow_ctrl – Módulo responsável pela geração dos sinais de controle que coordenam o funcionamento dos módulos em todos os estágios do pipeline. Sinais de controle especiais que resolvem conflitos de *pipeline* e *branch* também são gerados nesse módulo.
- Load_store_unit – Gera os endereços e sinais de controle necessários ao acesso da memória cache de dados.
- Mc_sys – Módulo que contém o incrementador de PC, os registradores de status atual e anterior, o gerenciador de interrupções, o sistema de gerenciamento de branch e o coprocessador interno.
- Multiply_unit – Multiplica dois operandos de 32 bits. Tem como saída os 32 bits menos significativos resultantes de tal operação.
- Opcode_decoder – Decodifica as instruções, gerando sinais de controle e os endereços dos operandos a serem utilizadas pelo módulo ULA.

- Operand_unit – Carrega os valores dos operandos a serem utilizados pela ULA, sejam eles provenientes do banco de registradores ou de um valor imediato. O detector de conflitos de dados causados por pipeline e o sistema de aceleração de operadores estão localizados nesse módulo.
- Reg_file – Banco de registradores do processador. Contem 32 registradores ao todo, a serem utilizados a depender do modo de operação do microprocessador.
- Wb_unit – Executa operação de *write-back* no banco de registradores.

Além do bloco "Core", o microprocessador STORMcore também é formado por dois blocos de memória cache, uma de dados (D-Cache) e uma de instruções (I-Cache), e pelo módulo "Bus unit", um barramento wishbone que coordena o acesso à memória de dados e de instruções.

O barramento wishbone é um tipo de barramento de comunicação, código aberto, criado para uniformizar a comunicação entre IPs de um circuito integrado. Vários IPs de código aberto usam esse tipo de barramento. Uma análise detalhada desse barramento foge ao escopo deste trabalho, mas pode ser encontrada no manual de referência Wishbone (2010).

Os oito estágios de pipeline do microprocessador STORMcore são:

- Acesso à instrução (IA) – O PC é atualizado, indicando o endereço da próxima instrução que será executada.
- Busca à instrução (IF) – A memória cache de instrução recebe uma solicitação de nova instrução e fornece a instrução correspondente.
- Decodificação da instrução (ID) – A instrução é decodificada no módulo "Opcode_decoder".
- Busca aos operandos (OF) – Os registradores necessários são carregados do banco de registradores. A aceleração de operadores acontece nesse estágio, caso necessário, para resolver conflitos de dados (*data hazards*).
- Multiplicação / shift (MS) – Uma multiplicação ou shift pode é feita nesse estágio, se necessário.
- Execução (EX) – Operações aritméticas e lógicas são executadas na ULA. A verificação da condição de execução da instrução e acesso aos dados do coprocessador também são feitos nesse estágio.

- Acesso a memória (MA) – Os acessos a memória de dados são feitos nesse estágio do pipeline. Para tento, o endereço a ser acessado e os sinais de controle necessários são enviados ao bloco D-Cache.
- Atualização dos registradores (WB) – os dados são lidos da memória cache de dados nesse estágio. O dado lido será escrito no respectivo endereço do banco de registradores na próxima borda de subida do sinal de relógio.

Um ambiente de teste simples foi disponibilizado para efetuar simulações comportamentais no microprocessador STORMcore. O ambiente disponibilizado segue o diagrama de blocos da Figura 27. Nele, o módulo "Internal Memory" guarda as instruções de um programa que calcula os 30 primeiros termos da sequência de Fibonacci. As instruções são lidas e executadas no módulo "STORM Core". As saídas do programa são mostradas na porta de saída do módulo "IO Controller"

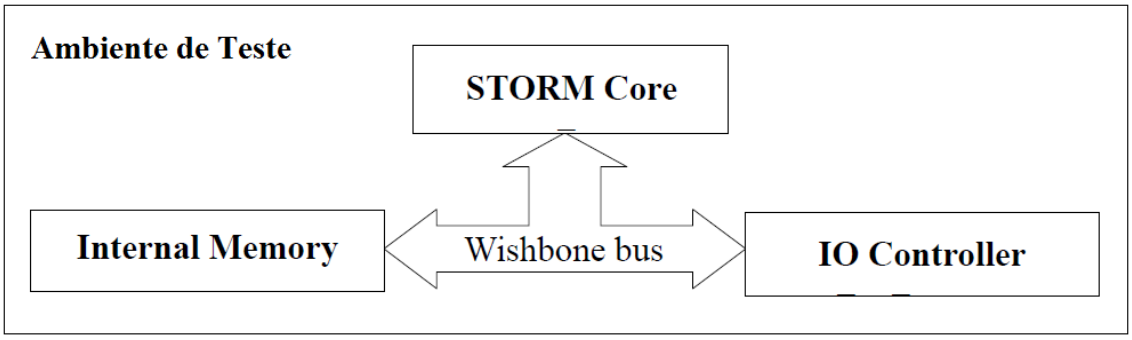

Figura 27. Ambiente de teste do microprocessador STORMcore.

4.2.2 SÍNTESE LOGICA

O microprocessador STORMcore foi sintetizado com células padrão com plano de fundo convencional e sem *polybiasing*, tal qual utilizado na síntese do banco de filtros FIR. Para as simulações de atrasos, foram utilizadas informações disponíveis na biblioteca padrão considerando-se uma alimentação dos transistores de 1V e temperatura de 125ºC.

As restrições impostas foram frequência do sinal de relógio de 100MHz, *clock skew* máximo de 50ps e atraso na saída de 500ps. A ferramenta de síntese utilizada foi o Design Compiler, da Synopsys. Em termos de desempenho, a netlist gerada possui uma

área total de 31403,7μm^2 e um consumo de energia total estimado de 9,1mW para uma frequência de sinal de relógio de 100MHz.

A simulação pós síntese apresentou problemas de violação de *setup time* e *hold time*, os quais podem ser provenientes de problemas no ambiente de teste ou na *netlist* gerada pela ferramenta de síntese. Devido a limitações de tempo e devido à complexidade do microprocessador STORMcore, os problemas encontrados não foram resolvidos.

4.2.3 CONCEPÇÃO DO LAYOUT

O mesmo procedimento utilizado para a concepção do layout do banco de filtros FIR foi seguido para o microprocessador STORMcore, mas para este layout não foram adicionados *pads*.

Sendo assim, o primeiro passo executado foi a criação das linhas de alimentação, VDD e VSS, e a delimitação das linhas que conterão as células padrão. A Figura 28 mostra o layout após essa etapa.

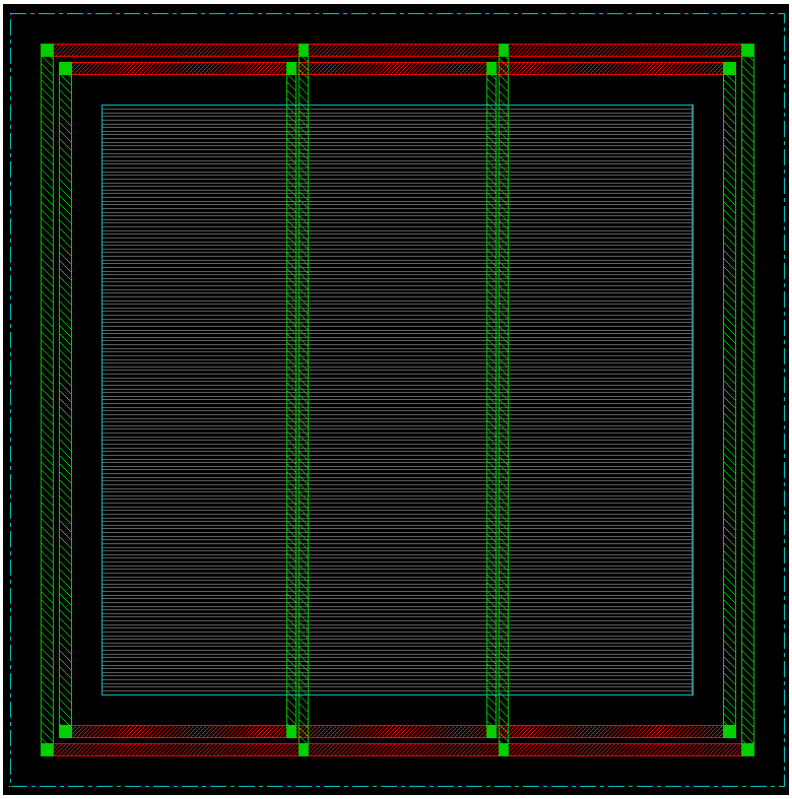

Figura 28. Layout do microprocessador STORMcore após criação das linhas de alimentação, VDD e VSS.

Em seguida, as células de preenchimento e *well taps* são adicionados para evitar efeitos de proximidade e *latchup*, respectivamente. A Figura 29 mostra o layout após tal etapa.

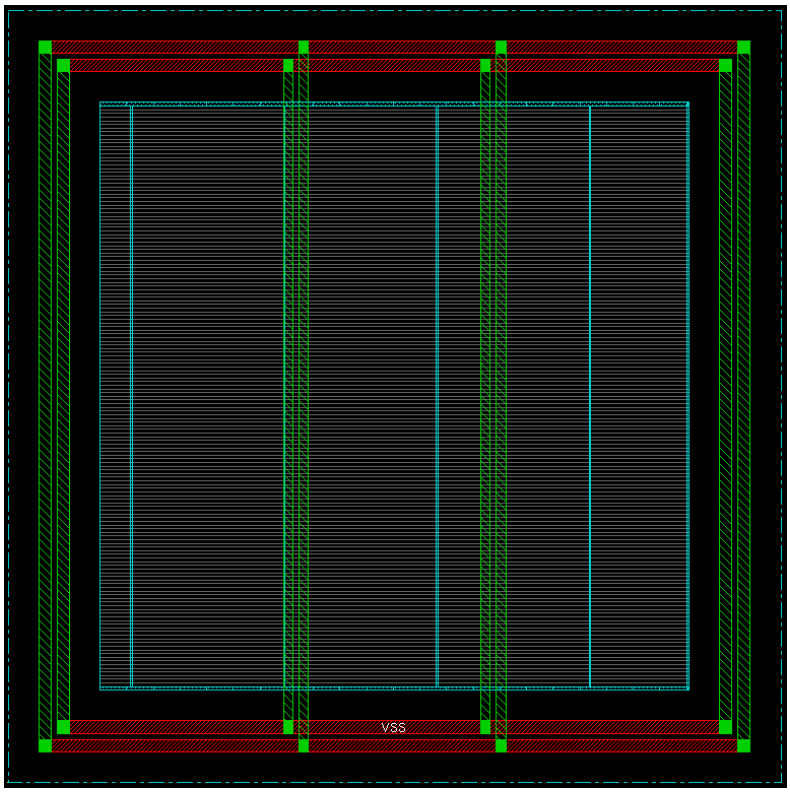

Figura 29. Layout do microprocessador STORMcore após finalização da etapa de *floorplan*.

Uma vez finalizada a etapa de *floorplan*, inicia-se a etapa de *placement*, ou seja, alocação das células padrão. A Figura 30 mostra o layout do microprocessador após essa etapa. Percebe-se que, em relação ao banco de filtros FIR, o layout do microprocessador STORMcore após *placement* apresenta menos espaços vazios entre as células padrão, ou seja, esse layout é mais denso que aquele.

Em seguida, a árvore de relógio é sintetizada, tal como feito para o circuito do banco de filtros. O resultado dessa etapa pode ser visto na Figura 31, as trilhas em amarelo representam as linhas que levarão o sinal de relógio às células padrão sequenciais.

Por fim, as células padrão são interligadas, conforme especificado pela *netlist* gerada na síntese lógica. A Figura 32 mostra o layout final obtido. A última etapa do fluxo de projeto proposto consiste em verificar violações de DRC, as quais não foram encontradas nesse layout, e erros de conectividade. Como as entradas e saídas do microprocessador não foram conectadas a nenhum *pad*, erros de linhas flutuantes aparecem no relatório de conectividade.

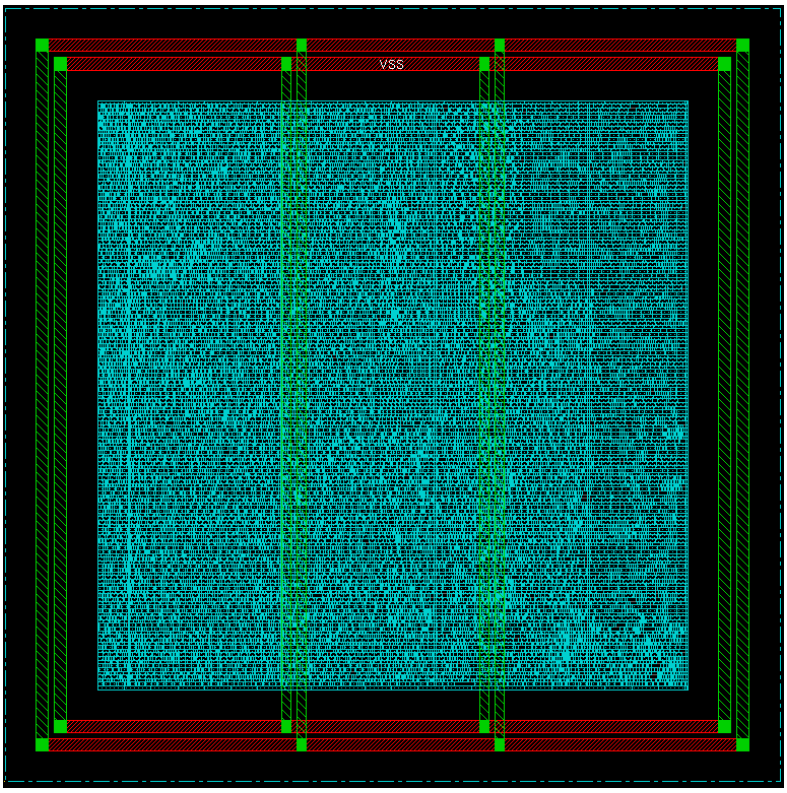

Figura 30. Layout do microprocessador STORMcore após *placement*.

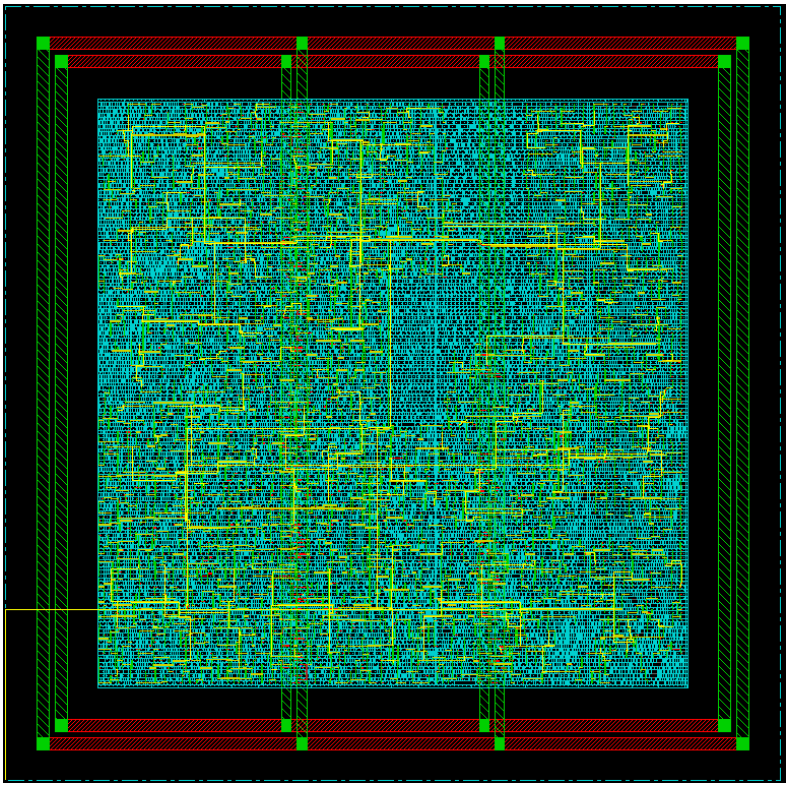

Figura 31. Layout do microprocessador STORMcore após síntese da árvore de relógio.

O layout final possui 38253 células padrão, ocupando uma área total de 36922µm^2. O *clock skew* máximo reportado foi de 24.6ps. Simulações pós layout foram feitas com os arquivos *sdf* e *verilog* obtidos a partir do layout final, porém erros foram encontrados na simulação e não foram resolvidos por questões de tempo e complexidade do circuito.

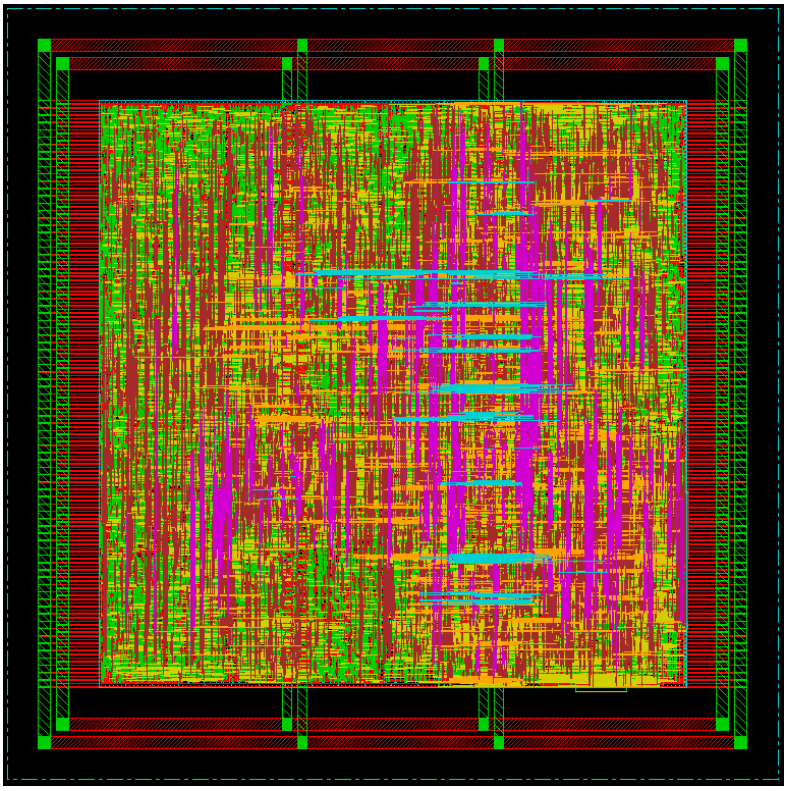

Figura 32. Layout final do microprocessador STORMcore.

4.3 INTEGRAÇÃO DE IPS SINTETIZÁVEIS SEM CÓDIGO FONTE

Além das formas de projeto de chips/IPs descritas anteriormente, por vezes, pode haver a integração, ao circuito que se deseja projetar, de IPs que não permitem visualização do código HDL que o descreve.

Em alguns casos, os IPs são disponibilizados como um conjunto de portas lógicas genéricas, as quais são posteriormente mapeadas à tecnologia de interesse, na fase de síntese lógica. Essa forma de distribuição de IPs dá aos fornecedores certa proteção, uma vez que é difícil aplicar engenharia reversa à *netlists*.

Em outros casos, o código fonte do IP, que se deseja integrar ao projeto de circuito integrado, é criptografado. A descrição de hardware criptografada é sintetizável, porém

não pode ser modificada. Nos dois casos citados, o circuito que se deseja projetar, contendo o IP sem código fonte, deve passar pelas etapas de síntese, para que sejam mapeadas à tecnologia alvo, e concepção do layout.

4.4 *HARD* IPS

IPs disponíveis em nível de *layout* são conhecidos como *hard* IPs. Esse nome deve-se principalmente ao fato destes serem dificilmente alterados ou reutilizados com outras tecnologias, uma vez que um determinado *layout* tem que obedecer à regras de projeto, em inglês, *Design Rules*, específicas a cada tecnologia.

Diferentemente dos casos citados nas subseções 4.3 e 4.2, nos quais é necessário efetuar síntese lógica e seguir os passos de concepção do layout com a *netlist* gerada, hard IPs são integrados diretamente na etapa de concepção do layout, precisando, apenas, ser roteados ao circuito que se deseja projetar.

São muito usados na indústria, como, por exemplo, IPs que implementam memórias integradas, circuitos analógicos e mistos (conversores A/D e D/A) e barramentos. Essa classe de IPs tem performance e área mais previsíveis que as previamente citadas, uma vez que *hard* IPs são representados em nível mais baixo.

4.5 IPS DE VERIFICAÇÃO

IPs de verificação, do inglês VIP, são módulos reutilizáveis utilizados para fins de verificação. Tipicamente consistem em modelos funcionais de barramentos, geradores de tráfego e monitores de protocolo. Esses IPs são integrados ao circuito que se deseja projetar, a fim de acelerar o processo de desenvolvimento de ambientes de teste. Assim, IPs de verificação permitem aos engenheiros de verificação focar na tarefa de verificar o circuito em si, ao invés de gastar uma quantidade excessiva de tempo na criação de ambientes de verificação complexos.

A verificação funcional é uma etapa crítica do processo de desenvolvimento de chips digitais. Essa é amplamente reconhecida como sendo o maior gargalo no fluxo de desenvolvimento de circuitos integrados. Acredita-se que até 70% do tempo e dos

recursos disponíveis ao projeto de circuitos integrados digitais são gastos na verificação funcional. Neste contexto, a redução do tempo e dos recursos gastos na etapa de verificação funcional, com uso de IPs de verificação, pode representar uma melhoria significativa de um fluxo de projeto de circuitos integrados.

5 CONCLUSÕES

Neste trabalho foi proposto um fluxo de projeto de circuitos integrados digitais, com base nos fluxos clássicos de projeto, mas explorando as características da tecnologia FD-SOI 28nm. O fluxo proposto foi testado com dois exemplos de circuitos, sendo, um deles, um IP cores código aberto.

O fluxo de projeto de CIs proposto mostrou-se válido, uma vez que se conseguiu produzir o layout de dois circuitos de forma satisfatória. O uso de scripts de configuração, que definem variáveis de ambiente genéricas, como o script setup.sh, por exemplo, mostrou-se bastante útil, uma vez que alterações mínimas nos demais scripts do fluxo, são suficientes para projetar layouts de diferentes circuitos.

A forma como estão organizadas as bibliotecas da tecnologia FD-SOI 28nm permite, aos projetistas, explorar as características e melhorias propostas por essa tecnologia, com poucas alterações ao fluxo de projeto proposto. Uma análise do layout do circuito do banco de filtros FIR, feito com outros tipos de células padrão disponíveis (com *polybiasing* e com o substrato invertido) é recomendada e ajudaria na escolha do melhor tipo de transistor a ser utilizado, para determinada aplicação. Contudo, tal análise não foi feita neste trabalho, por limitações de tempo.

Analisar os circuitos sob pontos de operação de tensão e temperaturas diferentes também é possível, com poucas alterações aos scripts do fluxo de projeto proposto, e ajudam a dar uma ideia da tensão ideal de alimentação dos transistores, para uma determinada aplicação.

As ferramentas de síntese lógica e de concepção de layout utilizadas permitem definir áreas específicas do circuito com tensões de alimentação diferentes. Esse tipo de projeto, com domínios de tensão, pode ajudar a reduzir as perdas, com a redução da tensão de alimentação em regiões do circuito cuja performance não é um fator essencial, pode também melhorar o desempenho de outras partes do circuito, com o aumento da tensão de alimentação dos transistores. Contudo, esse tipo de abordagem não foi adotado neste trabalho, por limitações de tempo. Recomenda-se o uso de domínios de tensão, principalmente em projetos nos quais consumo de energia e/ou performance sejam fatores críticos.

O desenvolvimento do banco de filtros FIR foi essencial para a validação do fluxo de projeto de CIs proposto, pois todas as etapas necessárias à concepção do layout, desde a descrição do hardware em VHDL, até a concepção do layout final, incluindo a alocação de *pads*, foram executadas com sucesso. Uma vez concluído o layout, conseguiu-se fazer uma análise de desempenho satisfatório, a partir dos relatórios gerados pela ferramenta de concepção do layout.

No que concerne o circuito do microprocessador STORMcore, apesar dos erros nas simulações pós-síntese e pós-layout, o projeto serviu para dar uma ideia de como são os IP cores código aberto e como seu barramento seria utilizado para conectá-lo a outros IPs.

Por completude, outras formas de integração de IP cores ao projeto de circuitos integrados digitais e sua adaptação ao fluxo de projeto de CIs proposto foram apresentadas (*Hard* IPs, IPs de verificação e IP sem código fonte), mas, por limitações de tempo, não foram apresentados exemplos dessas formas de inserção de IP cores.

BIBLIOGRAFIA

D. J. Frank et. aI. Device Scaling Limits of Si MOSFETs and Their Application Dependencies. Proc. IEEE IntI. Workshop on Memory Technology, Design and Testing, vol. 89, pp. 259-288,2001.

N. Z. HARON; S. HAMDIOUI. Why CMOS scaling coming to an END. Design and Test Workshop IDT 2008. 3rd International, p. 98 a 103, 2008.

Cattaneo, M. Efeito de Corpo em transistores SOI de porta dupla vertical. Dissertação de Mestrado. Centro Universitário da FEI, São Bernardo do Campo, 2009.

CELLER, G. K.; CRISTOLOVENEAUNU, S. Frontiers of silicon-on-insulator. Journal of Applied Physics Volume 93, p. 4955 a 4978. American Institute of Physics, [S. l.], 2003.

Cristoloveanu, S.; Balestra, F. Technologie silicium sur isolant (SOI). Les Sélections : Dossier Techniques de l'ingénieur e2380. Éditions T. I., 2013.

D. CLEIN. CMOS IC Layout, Concepts, Methodologies and Tools. Newnes, 2000.

CRISTOLOVENEAU, S.; LI, S. S. Electrical characterization of silicon-on-insulator materials and devices. Springer, Norwell, 1995

R. MOLINA. How to Close Timing with Hundreds of Multi-Mode/Multi-Corner Views. 6/12/2012. Disponível em <http://www.eejournal.com/archives/articles/20121206-cadence/>. Acesso em: 07/2015.

Faynot, O.; Vandooren, A.; Ritzenthaler, R.; Poiroux, T.; Lolivier, J.; Jahan, C.; Barraud, S.; Ernst, T.; Adrieu, F.; Casse, M.; Giffard, B; Deleonibus, S. Advanced SOI MOSFETs: structures and devices physics. Silicon-on-Insulator Technology and Devices XII, p.: 1 a 10. The Electrochemical Society, Inc, [S. l.], 2005.

F. FLATRESS. UTBB-FDSOI Design & Migration Methodology. ST Microelectronics. [S. l.], 2013. Disponível em: <http://cmp.imag.fr/documents/doc/UTBB-FDSOI%20Design%20and%20Migration%20Methodology_.pdf>. Acessado em: 07/2015.

Jacquet, D. Architectural choices & design-implementation methodologies for exploiting extended FD-SOI DVFS & body-bias capabilities. SOI technology Summit, VLSI Symposium, Shanghai, 2013

JONES, H. Economic impact of the technology choices at 28nm/20nm. International Business Strategies. Los Gatos, 2012. Disponível em: http://www.soiconsortium.org/pdf/Economic_Impact_of_the_Technology_Choices_at_28nm_20nm.pdf . Acessado em 05/2015.

Mentor Graphics. Fully Depleted (FD) vs Partially Depleted (PD) SOI. Advanced Substrate News. SOITEC, [S. l.], 2008. Disponível em:

<http://www.advancedsubstratenews.com/2008/05/fully-depleted-fd-vs-partially-depleted-pd-soi>. Acessado em: 05/2015.

MOORE, G. E. Cramming More Components onto Integrated Circuits. Proceedings of the IEEE, vol. 86, n 1. IEEE, [S. l.], 1998.

Rabaey, J. M.; Chandrakasan, A.; Nikolic, B. Digital Integrated Circuits, A design Perspective. 2nd Ed. Prentice Hall, 2003.

SINGH, R. K.;SAXENA, A.; RASTOGI, M.. Silicon on Insulator Technology Review. International Journal of Engineering Sciences & Emerging Technologies, 2011. Volume 1, Issue 1, pp: 1-16

SKOTNICKI, T. Competitive SOC with UTBB SOI. SOI Conf. , Phoenix A., 2011

ST MICROELECTRONICS. Learn more about FD-SOI. [S. l.], 2014 Disponível em: <http://www.st.com/web/en/about_st/learn_fd-soi.html>. Acesso em: 05/2015. [2014a]

ST MICROELECTRONICS. An introduction to FD-SOI. St Online Media, [S. l.], 2013 Disponível em: <http://www.youtube.com/watch?v=uvV7jcpQ7UY> Acesso em: 04/2015.

ST MICROELECTRONICS. Continuing Moore's law. [S. l.], 2014. Disponível em: <http://www.st.com/web/en/about_st/moore_law.html>. Acesso em: 05/2015. [2014b]

APÊNDICE A – CÓDIGO FONTE DO MÓDULO FIR DO BANCO DE FILTROS FIR

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use ieee.numeric_std.all;

entity FIR is
 generic (N: integer := 13);
 port
  (
    clk             : in std_logic;
    reset                   : in std_logic;
    input_sample        : in signed (15 downto 0);
    output_sample       : out signed (15 downto 0)
    );

end FIR;

architecture A of FIR is

component multiplier
          port (input_a : in signed (15 downto 0); input_b : in signed (15 downto 0); output : out
signed (31 downto 0));
end component;

type table is array (0 to N-1) of signed (15 downto 0);
type table1 is array (0 to N-1) of signed (31 downto 0);
signal holderBefore      : table; --signed values
signal toAdd      : table1; --signed values
signal coefficients :   table; --signed value
signal j : std_logic_vector (3 downto 0);

begin

 M1 : for I in 0 to N-1 generate
    mult1 : multiplier port map
     (input_a => coefficients(I), input_b => holderBefore(I), output =>  toAdd(I));
  end generate M1;

          coefficients(0) <= to_signed(6375,16);
          coefficients(1) <= to_signed(1,16);
          coefficients(2) <= to_signed(-3656,16);
          coefficients(3) <= to_signed(3,16);
          coefficients(4) <= to_signed(4171,16);
          coefficients(5) <= to_signed(4,16);
```

```
        coefficients(6) <= to_signed(28404,16);
        coefficients(7) <= to_signed(4,16);
        coefficients(8) <= to_signed(4171,16);
        coefficients(9) <= to_signed(3,16);
        coefficients(10) <= to_signed(-3656,16);
        coefficients(11) <= to_signed(1,16);
        coefficients(12) <= to_signed(6375,16);


 calculate: process (clk, reset)
 begin

  if (CLK = '1' and CLK'event) then
                if(reset = '1') then
                        holderBefore <= (others => (others => '0'));
                        j <= "0000";
                        output_sample <= (others => '0');
                else
                                for i in N-1 downto 1 loop
                        holderBefore(i)    <= holderBefore(i-1);
                        end loop;
        holderBefore(0)    <= input_sample;
        output_sample <= input_sample + toAdd(0)(15 downto 0) + toAdd(1)(15 downto 0) +
                 toAdd(2)(15 downto 0) + toAdd(3)(15 downto 0) + toAdd(4)(15 downto 0) +
toAdd(5)(15 downto 0) +
                 toAdd(6)(15 downto 0) + toAdd(7)(15 downto 0) + toAdd(8)(15 downto 0) +
toAdd(9)(15 downto 0) +
                 toAdd(10)(15 downto 0) + toAdd(11)(15 downto 0) + toAdd(12)(15 downto 0);

                end if;
          end if;

  end process calculate;

end A;
```

APÊNDICE B – CÓDIGO FONTE DO MÓDULO TOP_FIR DO BANCO DE FILTROS FIR

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use ieee.numeric_std.all;

entity TOP_FIR is
 generic (NF: integer := 16);
 port
  (
    clk             : in std_logic;
    reset           : in std_logic;
    input_sample    : in signed (15 downto 0);
    output_sample   : out signed (15 downto 0);
                    eq                  : out std_logic
    );
end TOP_FIR;

architecture A of TOP_FIR is

component FIR is
 generic (N: integer);
 port
  (
    clk             : in std_logic;
    reset           : in std_logic;
    input_sample    : in signed (15 downto 0);
    output_sample   : out signed (15 downto 0)
    );
end component FIR;

component eqc is
 port
  (
    inputa          : in signed (15 downto 0);
    inputb          : in signed (15 downto 0);
                    eq                  : out std_logic
    );
end component eqc;

type table is array (0 to NF-1) of signed (15 downto 0);
signal out1      : table; --signed values
signal eq1 : std_logic_vector (NF-2 downto 0); --signed value
```

```
begin

 F1 : for a in 0 to NF-1 generate
    FIR1 : FIR
                              generic map (N => 13)
                              port map
       (clk, reset, input_sample, out1(a));
  end generate F1;

 output_sample <= out1(0);

 E1 : for ab in 0 to NF-2 generate
    EQC1 : EQC port map
      (out1(ab), out1(ab+1), eq1(ab));
  end generate E1;

 calculate: process (clk, reset)
 begin

  if (CLK = '0' and CLK'event) then
                              if (eq1 = "000000000000000") then
                                        eq <= '0';
                              else
                                        eq <= '1';
                              end if;
           end if;
 end process calculate;


end A;
```

APÊNDICE C – CÓDIGO FONTE DO AMBIENTE DE TESTES DO BANCO DE FILTROS FIR

```
library IEEE ;
use IEEE.std_logic_1164.ALL ;
use ieee.numeric_std.all;

library FIR_library;
use FIR_library.all;

entity test_bench_TOP_FIR is
end test_bench_TOP_FIR ;

architecture test of test_bench_TOP_FIR is

component TOP_FIR
--generic (NF: integer);
 port
  (
    clk             : in std_logic;
    reset                   : in std_logic;
    input_sample        : in signed (15 downto 0);
    output_sample       : out signed (15 downto 0);
                    eq                          : out std_logic
    );

end component;

signal clk_s                :std_logic := '0';
signal reset_s              :std_logic;
signal in_s                 :signed (15 downto 0);
signal out_s                :signed (15 downto 0);
signal eq_s       :std_logic;


begin

TOP_FIR1 : TOP_FIR
          generic map (NF => 16)
          port map (clk => clk_s, reset => reset_s, input_sample => in_s, output_sample =>
out_s, eq => eq_s);

clk_s   <= not (clk_s) after 5 ns;
reset_s <= '1', '0' after 20 ns;

TEST1 : process
```

```
begin
        wait for 1 ns;
        in_s <= (to_signed(0,16));

        wait for 19 ns;

        wait for 2 ns;
        in_s <= (to_signed(0,16));

        wait until clk_s'event and clk_s='0';
        in_s <= (to_signed(180,16));

        wait until clk_s'event and clk_s='0';
        in_s <= (to_signed(30,16));

        wait until clk_s'event and clk_s='0';
        in_s <= (to_signed(18,16));

        wait until clk_s'event and clk_s='0';
        in_s <= (to_signed(9375,16));

        wait until clk_s'event and clk_s='0';
        in_s <= (to_signed(3150,16));

        wait until clk_s'event and clk_s='0';
        in_s <= (to_signed(120,16));

        wait until clk_s'event and clk_s='0';
        in_s <= (to_signed(20,16));

        wait until clk_s'event and clk_s='0';
        in_s <= (to_signed(20,16));

        wait until clk_s'event and clk_s='0';
        in_s <= (to_signed(650,16));

        wait until clk_s'event and clk_s='0';
        in_s <= (to_signed(7000,16));

        wait until clk_s'event and clk_s='0';
        in_s <= (to_signed(70,16));

        wait until clk_s'event and clk_s='0';
        in_s <= (to_signed(70,16));

        wait for 7 ns;
        assert false report "END OF SIMULATION." severity failure;

end process TEST1;
end architecture;
```