Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Curso de Graduação em Engenharia Elétrica

Thiago Morais de Oliveira

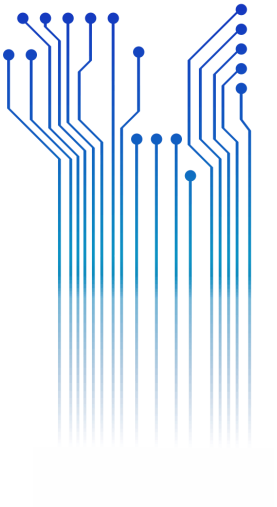
Universidade Federal
de Campina Grande

TRABALHO DE CONCLUSÃO DE CURSO

Implementação em hardware de um algoritmo de transformada rápida inteira de Fourier para aplicações de baixa potência

Centro de Engenharia
Elétrica e Informática

Departamento de
Engenharia Elétrica



Campina Grande
Dezembro de 2019

THIAGO MORAIS DE OLIVEIRA

IMPLEMENTAÇÃO EM HARDWARE DE UM ALGORITMO DE TRANSFORMADA RÁPIDA INTEIRA DE FOURIER PARA APLICAÇÕES DE BAIXA POTÊNCIA

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: Microeletrônica

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

CAMPINA GRANDE
DEZEMBRO DE 2019

THIAGO MORAIS DE OLIVEIRA

IMPLEMENTAÇÃO EM HARDWARE DE UM ALGORITMO DE TRANSFORMADA RÁPIDA INTEIRA DE FOURIER PARA APLICAÇÕES DE BAIXA POTÊNCIA

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: Microeletrônica

Aprovado em: / /

Professor Gutemberg Gonçalves dos Santos Júnior, D.Sc.
Universidade Federal de Campina Grande
Avaliador

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

CAMPINA GRANDE
DEZEMBRO DE 2019

Agradecimentos

Agradeço primeiramente a Deus pela minha vida e à imaculada virgem Maria por interceder por mim durante as tribulações. Seguidamente, aos meus queridos pais, Tony Marcus Lima de Oliveira e Fancinete Patrícia de Morais por todo amor e por acreditarem no meu potencial, mesmo quando nem mesmo eu não fui capaz de acreditar. Aos meus avós Antônio Neno, Nizete, Francisco Aprígio e Maria Joana por todas as histórias durante os cafés da tarde, valores e por me ensinarem o verdadeiro significado de família.

Agradeço a Yara Synthia, por estar ao meu lado nos melhores e nos piores momentos, por todo apoio, carinho e cumplicidade.

A Ariel e Yanca por todo apoio e descontrações, a minha irmã Isabely pelas discretas demonstrações de afeto e aos meus estimados amigos Marco Aurélio e Edylla, pelo companheirismo, passeios de fim de tarde e boas rizadas.

Agradeço aos meus colegas de curso e amigos Jesney, Matheus Andrade, Mylena e Ianca pela sinceridade, assistência e pela honrra de tê-los ao meu lado nessa jornada, seja durante as várias madrugadas de estudo ou nos felizes dias de confraternização.

Agradeço ao laboratório X-MEN, que me trouxe grande satisfação e engrandecimento profissional, em especial ao professor Gutemberg, Antônio Agripino e a Bruno Silva por acreditarem no meu trabalho, por todas as lições e dedicação.

Aos meus professores, por todo o empenho e dedicação, em especial ao professor Marcos Morais pela paciência e dedicação em me orientar nesta etapa final da minha formação.

"Não fui eu que ordenei a você? Seja forte e corajoso! Não se apavore nem desanime, pois o Senhor, o seu Deus, estará com você por onde você andar."

Josué 1:9

Resumo

A Transformada de Fourier Discreta é uma das operações mais fundamentais em *chips* no âmbito de processamento digital de sinais na atualidade, sendo crescente a necessidade de dispositivos mais rápidos, menores e de baixo consumo. Nesse sentido, esse trabalho tem por objetivo desenvolver o *front-end* de um bloco de Transformada Rápida de Fourier Inteira utilizando técnicas de baixo consumo de potência. Para isso, foram tidos como base os trabalhos de Soontorn Oraintara acerca da transformada inteira de Fourier (IntFFT), os fundamentos da transformada de Fourier de tempo discreto de Allan V. Oppenheim e as implementações dos algoritmos de FFT (*Fast Fourier Transform*) de Uwe Meyer-Baese. A metodologia utilizada segue o fluxo de desenvolvimento de ASIC (*Application Specific Integrated Circuits*), englobando as etapas de especificação, modelagem, projeto do circuito digital e síntese lógica. Como resultado, foi possível a obtenção de uma *netlist*, a partir da síntese lógica de uma transformada de 8 pontos, utilizando o algoritmo *split-radix*, sendo elencadas métricas de área, consumo e performance do bloco em questão. O projeto proposto pode ser adaptado e utilizado para geração de transformadas maiores devido suas decisões arquiteturais de portabilidade e reúso.

Palavras-chave: IntFFT, *Split-radix*, Baixo consumo, ASIC, Síntese lógica.

Abstract

The Discrete Fourier Transform is one of the most fundamental operations presents in digital signal processing chips today, with a growing need for faster, smaller, low-power devices. In this sense, this work aims to develop the front end of an Integer Fourier Fast Transform block using power-aware techniques. To this end, Soontorn Oraintara's work on the Integer Fast Fourier Transform (IntFFT), the fundamentals of the discrete-time Fourier transform of Allan V. Oppenheim, and the Fast Fourier Transform algorithms implementation of Uwe Meyer-Baese. The methodology used follows the ASIC (Application Specific Intregrated Circuits) development flow encompassing the specification, modeling, digital circuit design and logic synthesis steps. As a result, it was possible to obtain a netlist from the logical synthesis of a 8 point IntFFT, using the split-radix algorithm being reported the area, power and performance of the referred block. The proposed design can be adapted and used to generate larger transforms due to its architectural decisions of portability and reuse.

Keywords: IntFFT, Split-radix, Power-aware, ASIC, Logic synthesis.

Lista de tabelas

Tabela 1 – Número de multiplicações complexas não triviais para diferentes algoritmos de FFT. 14
Tabela 2 – Número de multiplicações reais não triviais para diferentes algoritmos de FFT. 14
Tabela 3 – Número de adições reais para diferentes algoritmos de FFT. 15

Tabela 4 – Impactos das técnicas de baixo consumo. 26

Tabela 5 – Descrição dos sinais da unidade de controle. 42
Tabela 6 – Valores lógicos assumidos pelas saídas da unidade de controle durante os estados. 43

Tabela 7 – Resultados da síntese lógica do bloco *sprx_intfft8* a 20 MHz. 48
Tabela 8 – Resultados da síntese lógica do bloco *sprx_intfft8* a 40 MHz. 48
Tabela 9 – Resultados da síntese lógica do bloco *sprx_intfft8* a 60 MHz. 48
Tabela 10 –Resultados da síntese lógica do bloco *sprx_intfft8* a 80 MHz. 49
Tabela 11 –Resultados da síntese lógica do bloco *sprx_intfft8* a 100 MHz. 49

Lista de ilustrações

Figura 1 – Fluxo de desenvolvimento ASIC. 4
Figura 2 – Impacto das otimizações de *low-power* em diferentes etapas do design. 6
Figura 3 – Entradas e saídas de uma síntese lógica. 7

Figura 4 – Estrutura de treliça de uma FFT Cooley-Tukey radix-2 de 8 pontos. . . 12
Figura 5 – Estrutura de uma borboleta da FFT *split-radix* 14
Figura 6 – Esquema de *lifting* para cálculo de multiplicação complexa. 17
Figura 7 – Estrutura de treliça da IntFFT de 8 pontos. 18
Figura 8 – Multiplicadores *lifting* para diferentes intervalos de θ 19

Figura 9 – Potência dissipada em um transistor operando como chave. 22
Figura 10 – Exemplo de inserção de células de *clock-gate*. 24

Figura 11 – Simulação do modelo de FFT proposto para uma entrada senoidal. . . 32
Figura 12 – Simulação do modelo de FFT proposto para uma onda quadrada. . . . 33

Figura 13 – Microarquitetura do bloco cplx_mult_c2m4a2. 36
Figura 14 – Microarquitetura do bloco lift_cplx_mult_c3m3a3. 36
Figura 15 – Interface do bloco cplx_mult_c2m4a2. 37
Figura 16 – Interface do bloco lift_cplx_mult_c3m3a3. 37
Figura 17 – Protocolo de validade adotado. 37
Figura 18 – Microarquitetura da borboleta *radix-2*. 38
Figura 19 – Interface comum às borboletas *radix-2* e *radix-4*. 38
Figura 20 – Microarquitetura da borboleta *radix-4*. 39
Figura 21 – Discriminação da unidade de pre processamento em uma FFT *split-radix* de 8 pontos. 40
Figura 22 – Interface do bloco *sprx_fft8_proc*. 41
Figura 23 – Máquina de estado para controle da IntFFT. 42
Figura 24 – Interface da unidade de controle. 43
Figura 25 – Interface do bloco IntFFT de 8 pontos. 44
Figura 26 – Simulação RTL do bloco sprx_intfft8. 45
Figura 27 – Simulação RTL do bloco sprx_fft8_proc. 45
Figura 28 – Simulação RTL do bloco sprx_bf4. 46
Figura 29 – Simulação RTL do bloco sprx_bf2. 46
Figura 30 – Simulação RTL do bloco lift_cplx_mult_c3m3a3. 46
Figura 31 – Simulação RTL do bloco sprx_fft8_ctrl. 46

Lista de abreviaturas e siglas

ASIC	*Application Specific Integrated Circuit*
DFT	*Discrete Fourier Transform*
FFT	*Fast Fourier Transform*
FSM	*Finite State Machine*
FxpFFT	*Fixed-point Fast Fourier Transform*
GTL	*Gate-Level*
HDL	*Hardware Description Language*
IDFT	*Inverse Discrete Fourier Transform*
IFFT	*Inverse Fast Fourier Transform*
IntFFT	*Integer Fast Fourier Transform*
IoT	*Internet of Things*
IP	*Intellectual Property*
MOSFET	*Metal-Oxide-Semiconductor Field Effect Transistor*
MUX	Multiplexador
RFID	*Radio-Frequency Identification*
ROM	*Read Only Memory*
RTL	*Register-transfer level*
SoC	*System on Chip*
STA	*Static Timing analysis*
VLSI	*Very-large-scale integration*
UC	Unidade de Controle

Sumário

1 Introdução 1

1.1 Objetivos 2

2 Fluxo de *front-end* da IntFFT 3

2.1 Epecificação 4

2.2 Modelo do sistema 5

2.3 Projeto do circuito digital 5

2.4 Síntese lógica 7

3 A transformada rápida de Fourier 9

3.1 O algoritmo de Cooley-Tukey 10

3.2 O algoritmo de FFT *split-radix* 12

3.3 A transformada rápida de Fourier inteira 15

4 Estratégias de baixo consumo 21

4.1 O consumo de potência 21

4.2 *Clock Gating* 23

4.3 *Power Gating* 25

4.4 *Multi-Threshold* e *multi voltage* 25

5 Modelo do sistema 27

5.1 Arredondador binário 27

5.2 Multiplicadores complexos 29

5.3 FFT *split-radix* 30

6 Projeto do circuito digital 35

6.1 Os multiplicadores complexos 35

6.2 Borboletas *radix-2* e *radix-4* 38

6.3 Unidade de pré-processamento de tamanho N 39

6.4 Unidade de controle 41

6.5 Codificação RTL 43

7 Síntese lógica 47

8 Conclusão 51

Referências 53

1 Introdução

Os desenvolvimentos realizados nas áreas de processamento digital de sinais e fotônica trazem consigo o avanço das tecnologias de comunicação que vêm para mudar completamente a forma que interagimos com o mundo, demandando DSPs (*Digital Signal Processors*) com capacidades de processamento cada vez maiores embarcados em dispositivos inteligentes cada vez menores.

A par disto, a necessidade do tratamento destes sinais de altas frequências, bem como a redução da tensão de alimentação e potência consumida, têm se tornado verdadeiros desafios ao desenvolvimento de circuitos integrados de aplicação específica (ASIC) otimizados, sendo necessário reimaginar, avaliar e desenvolver novas arquiteturas de circuitos processadores de sinais eficientes e de baixo consumo capazes de suprir tal demanda.

Para Oppenheim(2012), uma das operações mais fundamentais presentes em chips no âmbito do tratamento de sinais digitais é a transformada discreta de Fourier (DFT) que, graças a sua conveniente propriedade de tornar a operação de convolução entre sinais no domínio do tempo na multiplicação de seus devidos espectros no domínio da frequênicia, têm sido amplamente empregada na filtragem linear de sinais. Outrossim, as contribuições iniciais de Cooley e Tukey reduziram eficientemente seu custo computacional através do algorítmo da Transformada Rápida de Fourier (FFT), que fora repensada e otimizada posteriormente, valendo-se de suas propriedades de simetria e periodicidade.

As otimizações acerca da complexidade operacional da DFT têm impacto direto na performance, área e consumo de um chip. Todavia aperfeiçoamentos arquiteturais nos blocos multiplicadores complexos, de maneira a reduzir o número de multiplicações reais tabém levam a melhorias em tais aspectos, visto que a operação de multiplcação, no âmbito de VLSI (*Very-large-scale integration*), consome grande porção de potência e tempo quando comparada às demais operações aritméticas.

Uma vez que a resolução dos dados é finita, a representação binária de ponto fixo imprime um erro de quantização nos dados proporcional ao tamanho da palavra binária.

Erro este que se propaga durante a conversão direta da DFT, através das operações aritméticas do processo, culminando em uma desigualdade do sinal recuperado observada ao recontruir o sinal de tempo discreto pelo seu espectro através da IDFT (*Inverse Discrete Fourier Transform*) e, portanto, afetando a inversibilidade da transformada.

Uma das formas de solucionar esta inexatidão é o mapeamento reversível de inteiro-para-inteiro da transformada rápida de Fourier (IntFFT) com a finalidade de anular o erro de reconstrução, aumentando a precisão e confiabilidade dos dados.

A IntFFT proposta utiliza técnicas de baixo consumo de potência para a implementação de uma FFT reversível, utilizando o fluxo de desenvolvimento ASIC em sua concepção.

1.1 Objetivos

Este trabalho tem como objetivo descrever o desenvovimento do *front-end* do IP IntFFT, iniciando com a especificação e modelagem, descrição RTL, síntese lógica e simulações e análises do *netlist*, validando sua funcionalidade através de testbenches auto-checados.

Serão realizados estudos e decisões arquiteturais tendo em vista os resultados de potência, área e performance, objetivando um bloco otimizado, explorando algumas técnicas de *low-power* a nível RTL, como *clock gating* e *data gating*.

Além da implementação em HDL, será realizada a síntese lógica na tecnologia de 180 nm, resultando em uma *netlist*, elencando os resultados para diferentes frequências de operação. A escolha deste PDK se deu pela presença das células especiais de *clock gating* na biblioteca fornecida pelo laboratório X-MEN, onde foi desenvolvido o referido IP, e pela disseminação desta tecnologia em artigos científicos para futuras comparações.

2 Fluxo de *front-end* da IntFFT

O desenvolvimento de um circuito integrado de aplicação específica conta com uma metodologia de desenvolvimento que se inicia na especificação do produto, passando pelas etapas de *front-end*, *back-end* e verificação, terminando com um SoC físico, encapsulado e validado.

Cada etapa do fluxo de desenvolvimento de um ASIC é composta por passos a serem seguidos, a fim de quebrar estas grandes tarefas em porções menores, facilitando a detecção de erros e paralelização de tarefas.

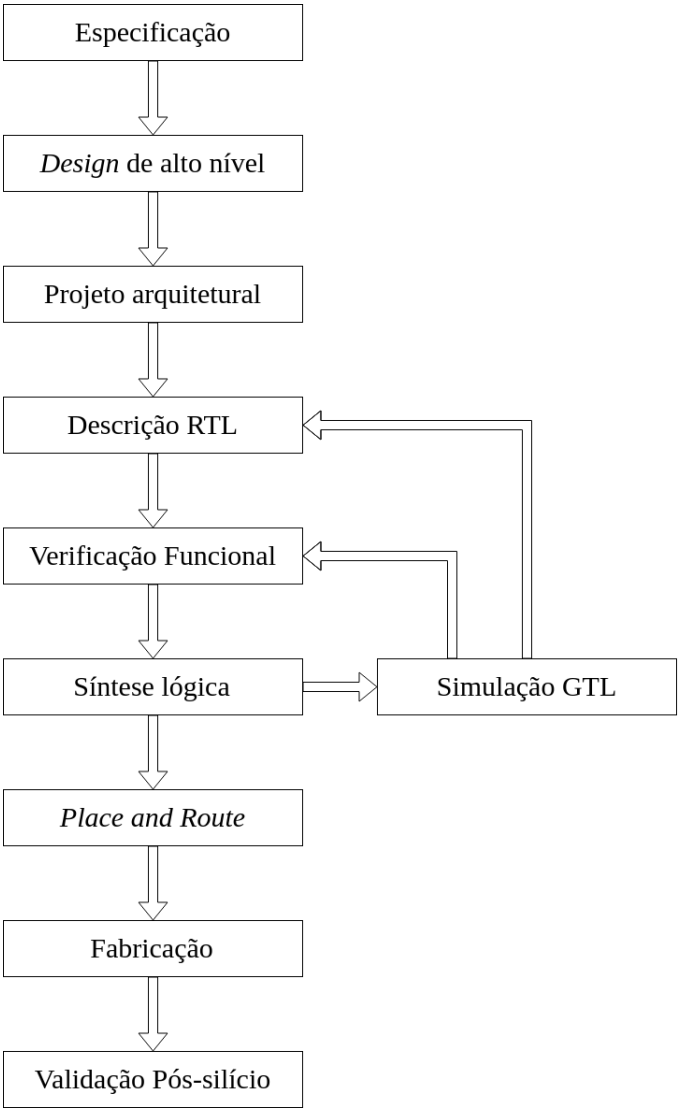
É do escopo deste trabalho o desenvolvimento do *front-end* do bloco IntFFT, seguindo a metodologia do fluxo de desenvolvimento ASIC ilustrado na Figura 1. As etapas compreendidas neste trabalho vão até a síntese lógica, que é a interface de transição entre o *front-end* e o *back-end*, que se inicia no *place and route*.

Partindo da especificação, a etapa de modelagem trata do *design* de alto nível, consistindo no desenvolvimento de um modelo de referência para testes funcionais e lógicos do algoritmo utilizado. Em seguida vem a etapa de projeto do circuito digital, que engloba o projeto arquitetural e descrição RTL, onde serão desenvolvidos o modelo e a arquitetura do sistema digital, implementando-a em uma linguagem de descrição de *hardware*, segundo a funcionalidade desejada. Finalmente, a síntese lógica consiste na tradução da lógica digital do RTL em portas lógicas, conforme o PDK adotado e limitações de funcionamento.

Ao final do fluxo de *front-end*, o design compor-se-á de uma *netlist*, isto é, uma coleção de portas lógicas, que executam a operação descrita na HDL.

No projeto de circuitos digitais, o fluxo de *front-end* consiste em várias etapas a serem seguidas no desenvolvimento ASIC, ilustrado na Figura 1, tendo início na especificação do circuito até a síntese lógica e simulação *gate-level*. Será detalhado, a seguir, cada um destes passos, expondo suas entradas e saídas.

Figura 1 – Fluxo de desenvolvimento ASIC.



Fonte:
http://verificationexcellence.in/wp-content/uploads/2018/08/vlsi_Desig_lifecycle.jpg
(adaptado).

2.1 Epecificação

Entende-se por especificação o conjunto de requerimentos a serem cumpridos em um produto e é o primeiro estágio do projeto de um ASIC, visto que é nela que são definidos parâmetros do sistema que devem ser respeitados e levados em conta nas decisões arquiteturais do projeto.

Geralmente, a especificação provém de uma intensa pesquisa de mercado, demandas e tendências tecnológicas para a necessidade de uma aplicação específica. Tais necessidades

ditam tanto a funcionalidade quanto a tecnologia adotada, tendo em vista os concorrentes provedores de produtos semelhantes e concorrentes.

As tendências de mercado são, portanto, traduzidas em valores de performance, área e consumo de potência (PPA) para o produto final, que por sua vez distribui tais limitações para as suas unidades(ou blocos) componentes menores.

As otimizações de um SoC devem ser feitas levando em conta o PPA, ou seja, uma otimização em consumo irá afetar a área e a performance e, portanto, o custo total do projeto. Nessa conformidade, as soluções não devem ser pensadas unicamente para um atributo, mas para os três como um todo, mantendo o foco em um.

A par disso,o bloco IntFFT será desenvolvido com foco em aplicações *low power*. Em outras palavras, as decisões arquiteturais devem contemplar técnicas e estruturas de baixo consumo de potência tendo em vista a performance.

2.2 Modelo do sistema

Uma vez definida a especificação do bloco, o próximo passo é elaborar o que o bloco irá fazer, segmentando-os em partes menores e garantindo a validade aritmética de cada parte.

A etapa de modelagem consiste, portanto, na implementação do algorítmo em uma linguagem de alto nível, sendo de grande importancia na testabilidade do RTL gerado por constituir um modelo aritmético a ser seguido. Também é nessa etapa que será definida a macroarquitetura do bloco em questão, podendo ser revista no projeto da microarquitetura.

O modelo do IP IntFFT foi codificado utilizando a linguagem de programação Python para implementar a estrutura de FFT *split-radix*, que será detalhada no próximo capítulo. Além da modelagem do algorítmo em sí, algumas unidades de processamento foram modeladas, tais como o arredondamento de bits, os multiplicadores complexos e as borboletas.

2.3 Projeto do circuito digital

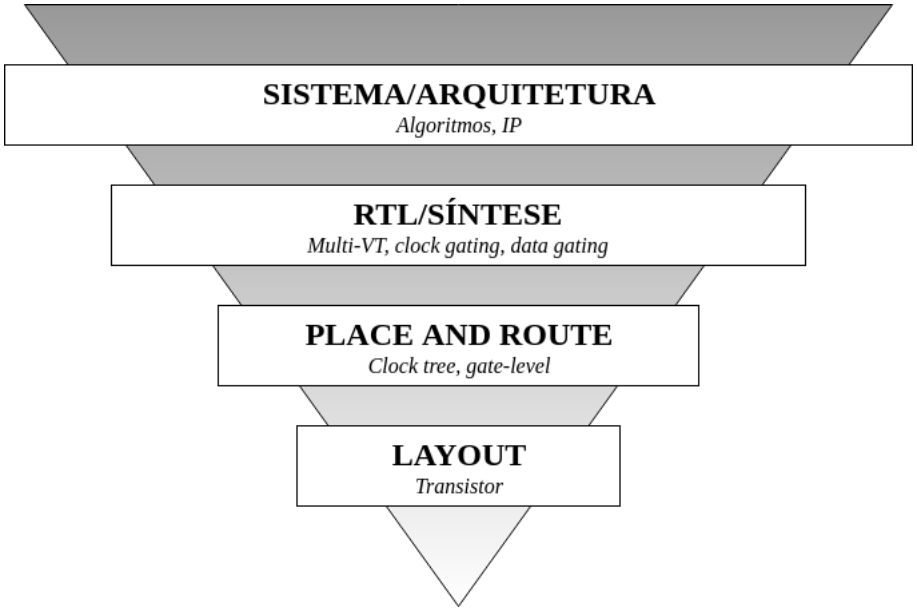
Esta etapa do fluxo de *front-end* é uma das mais importantes, do ponto de vista de otimizações. É nela que a microarquitetura é definida, compreendendo tanto seu projeto quanto implementação, onde cada decisão arquitetural desencadeará resultados diretos no PPA do bloco.

Caberá ao projetista determinar quais protocolos de comunicação serão adotados

entre os blocos, quantas e quais máquinas de estado utilizar e a resolução adotada na representação binária. O resultado disso será um *design* a nível de blocos, evidenciando a estrutura a ser implementada em HDL.

Também é nesse estágio que serão feitas as implementações RTL do modelo do circuito, com a ajuda de testbenches para validaçãoo funcional, seguindo a microarquitetura.

Figura 2 – Impacto das otimizações de *low-power* em diferentes etapas do design.



Fonte: https://m.eet.com/media/1077494/lowpower2.jpg (adaptado).

O impacto das decisões para baixo consumo em diferentes estágios do fluxo ASIC é apresentado na Figura 2, onde as duas primeiras etapas são pertinentes ao *front-end*. Como pode-se notar, a escolha do algorítmo e a definição e organização da microarquitetura são pontos-chave na concepção de um projeto otimizado (contando com ganhos de 10 a 20 vezes no consumo de um bloco). Também é de suma importância a codificação RTL e *scripts* de síntese que permitam a concepção de estruturas especiais de baixo consumo como *clock-gating* e *data-gating* (com ganhos de 2 a 5 vezes no consumo do bloco).

Durante a etapa de *design* do circuito digital da IntFFT, o bloco foi dividido em 4 partes principais de processamento (*data-path*) e uma unidade de controle responsavel por gerenciar os sinais de controle das instâncias em cada etapa de processamento, habilitando-as e ativando-as quando fosse necessário.

2.4 Síntese lógica

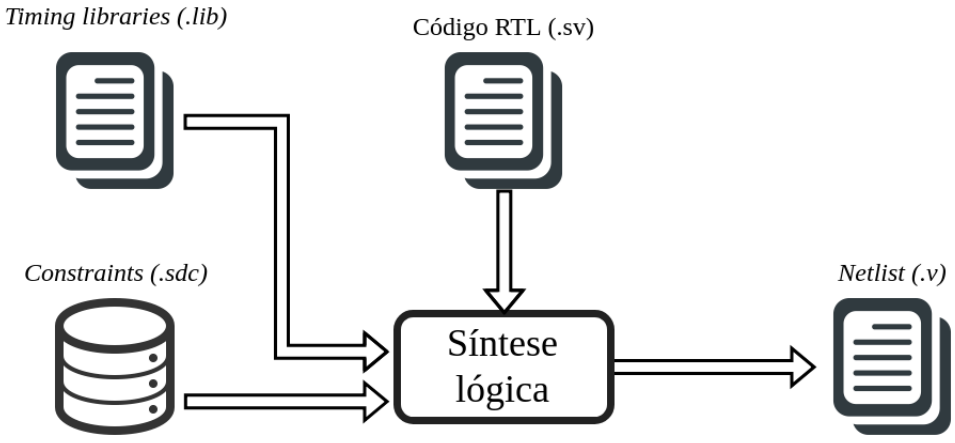
A síntese lógica consiste na análise, tradução, otimização e mapeamento do código comportamental do projeto escrito em HDL em portas lógicas, utilizando um programa que recebe, além do RTL, as *standard-cells* do PDK na tecnologia adotada, as *timing libraries* e as *constraints* especificadas para o *design*, de maneira a obter um código HDL estrutural de mesma funcionalidade com os melhores resultados de PPA possíveis.

As *timing libraries* abarcam uma coleção de portas lógicas fornecidas pela fábrica detentora do processo de manufatura da tecnologia para construção dos circuitos. Essas bibliotecas comumente se apresentam no formato "*Liberty*" (.lib) e contêm, além de informações funcionais, suas características físicas, tais como área e formato (*footprint*), potência e atrasos.

As *constraints* são regras impostas pelo projetista, com base nos critérios de PPA, com o objetivo de direcionar as otimizações da ferramenta de síntese para cumprir as métricas do projeto, alcançando os melhores resultados possíveis. Essas regras são escritas seguindo o formato SDC, ou "*Synopsys Design Constraints*" e são utilizadas no processo de análise de tempo estático (STA) e em outras etapas da síntese.

O sintetizador receberá, portanto, o código RTL, as *timing libraries* e as *constraints* para geração do *netlist*, uma descrição HDL a nível de portas lógicas (GTL), conforme esquematizado na Figura 3.

Figura 3 – Entradas e saídas de uma síntese lógica.



Fonte: https://www.assignmentpoint.com/wp-content/uploads/2015/10/logic-synthesis1.jpg (adaptado).

O processo de síntese lógica se inicia com a leitura e *lint*[1] do código RTL e elaboração do *design*, onde lógicas recursivas e funções são expandidas, estruturas de dados são construídas e registradores são alocados, elaborando o bloco. Após a etapa de elaboração, dá-se início ao mapeamento, que inicialmente é genérico, ou independente da tecnologia, e depois realizado segundo o PDK adotado. É na etapa de mapeamento tecnológico que o sintetizador irá "escolher" quais células da biblioteca alocar para implementar a função estruturada no mapeamento genérico.

Uma vez alocadas todas as células, dá-se início à fase de otimização, onde a ferramenta identifica estruturas funcionais e as otimiza realocando, combinando ou removendo registradores, propagando valores constantes e recalculando a área total do bloco a partir das informações físicas contidas nas *timing libraries*.

Paralelamente às otimizações do mapeamento com a tecnologia, é executada a análise de tempo estático (STA), que consiste no cálculos dos atrazos pertinentes ao design, com o objetivo garantir a funcionalidade do *netlist* na frequência desejada.

[1] Processo de leitura do código-fonte com o intuito de sinalizar erros de sintaxe ou estruturas não sintetizáveis.

3 A transformada rápida de Fourier

A transformada de Fourier discreta (DFT) desempenha um papel importante na análise, projeto e implementação de algorítmos e sistemas de processamento de sinais discreto (OPPENHEIM, 2012), nesse sentido eficientes algoritmos para seu cálculo numérico foram desenvolvidos com o intuito de reduzir a complexidade operacional presente na mesma. De fato, esta eficiência é tão grande que em algumas implementações de convoução a opção mais eficaz é transformar os sinais envolvidos para o domínio da frequência, multiplicá-los e trazêlos de volta ao domínio do tempo por meio da transformada inversa de Fourier (IFFT).

De modo geral, os algorítmos de FFT se valem das propriedades de simetria e periodicidade da exponencial complexa W_N^{kn} para reduzir a quantidade de operações da DFT, das quais podemos citar a de simetria complexa conjudada e a de periodicidade no tempo discreto e na frequência, descritas nas Equações 3.1 e 3.2, respectivamente.

$$W_N^{k(N-n)} = W_N^{-kn} = (W_N^{kn})^* \tag{3.1}$$

$$W_N^{kn} = W_N^{k(n+N)} = W_N^{(k+N)n} \tag{3.2}$$

Segundo Burrus (1977) os algoritmos de FFT podem ser vistos como mapeamentos multidimensionais das sequências de entrada e saída, ou seja, podemos estruturar a DFT de tamanho N em uma representação multidimensional, onde $N = \prod_l N_l$.

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk} \tag{3.3}$$

Para um mapeamento bidimensional $f : \mathbb{C}^N \rightarrow \mathbb{C}^{N_1 x N_2}$, podemos transformar o índice n

da Equação 3.3 da seguinte maneira:

$$n = An_1 + Bn_2 mod N \tag{3.4}$$

Onde o operador *mod* refere-se ao resto da divisão inteira, $n_1, n_2 \in \mathbb{N}$, tal que $0 \leq n_1 \leq N_1 - 1$ e $0 \leq n_2 \leq N_2 - 1$ e $A, B \in \mathbb{Z}$ são contantes definidas pelo mapeamento utilizado. Aplicando a transformação do índice n na Equação 3.3, ficaremos com a estrutura matricial da equação 3.5.

$$\begin{bmatrix} x[0] & x[1] & x[2] & \cdots & x[N-1] \end{bmatrix} = \begin{bmatrix} x[0,0] & x[0,1] & \cdots & x[0,N_2-1] \\ x[1,0] & x[1,1] & \cdots & x[1,N_2-1] \\ \vdots & \vdots & \ddots & \vdots \\ x[N_1-1,0] & x[N_1-1,1] & \cdots & x[N_1-1,N_2-1] \end{bmatrix} \tag{3.5}$$

Do mesmo modo que aplicamos o mapeamento do índice n para o sinal $x[n]$ de tempo discreto podemos definir tal transformação bidimencional para o índice de frequência k do sinal de saída:

$$k = Ck_1 + Dk_2 mod N \tag{3.6}$$

Onde $C, D \in \mathbb{Z}$ e $N = N_1 N_2$.

A escolha dos índices A, B, C e D define o algoritmo de FFT em particular e devem satistafer alguns critérios para que a DFT permaneça uma função bijetora. Além disso, devem ser escolhidos os tamanhos N_1 e N_2, sendo $mdc(N_1, N_2) > 1$ (algoritmos de fatores comuns) ou $mdc(N_1, N_2) = 1$ (algoritmos de fatores primos).

3.1 O algoritmo de Cooley-Tukey

Um dos mais famosos e universais algoritmos para cálculo da DFT é o desenvolvido por James W. Cooley e John W. Tukey (1965) por possibilitar uma fácil fatoração de N, sendo que os maiores ganhos operacionais são obtidos para uma transformada que pode ser expressa como potncia de uma base r, isto é, $N = r^v$, onde $v \in \mathbb{N}$.

Seguido os mapeamentos dos índices de tempo e frequência definidos nas Equações 3.4 e 3.6, foram propostos $A = N_2$, $B = 1$, $C = 1$ e $D = N_1$. Resultando em:

$$n = N_2 n_1 + n_2 \tag{3.7}$$

$$k = k_1 + N_1 k_2 \tag{3.8}$$

Os *twiddle-fators* serão determinados por:

$W_N^{nk} = \exp -j\frac{2\pi nk}{N} = W_N^{(N_2 n_i + n_2)(k_1 + N_1 k_2)} = W_N^{N_2 n_1 k_1 + N_1 N_2 n_1 k_2 + n_2 k_1 + N_1 n_2 k_2}$

Sendo $N = N_1 N - 2$, $W_N^{N_1} = W_{N_2}$ e $W_N^{N_2} = W_{N_1}$. Logo:

$$W_N^{nk} = W_{N_1}^{n_1 k_1} W_N^{n_2 k_1} W_{N_2}^{n_2 k_2} \tag{3.9}$$

Aplicando a estrutura matricial da Equação 3.5 na Equação 3.3, ficamos com:

$$X[k_1, k_2] = \sum_{n_2=0}^{N_2-1} W_{N_2}^{n_2 k_2} (W_N^{n_2 k_1} \sum_{n_1=0}^{N_1-1} x[n_1, n_2] W_{N_2}^{n_2 k_2}) \tag{3.10}$$

Podemos perceber que o termo entre parênteses da Equação 3.10 concerne à tranformada de N_1 pontos ponderada por $W_N^{n_2 k_2}$, sendo tal resultado a entrada de uma transformada de N_2 pontos. Em outras palavras, a DFT de N pontos fora desmembrada em N_2 DFTs de tamanho N_1 ponderadas por $W_N^{n_2 k_1}$, seguidas de N_1 DFTs de tamanho N_2.

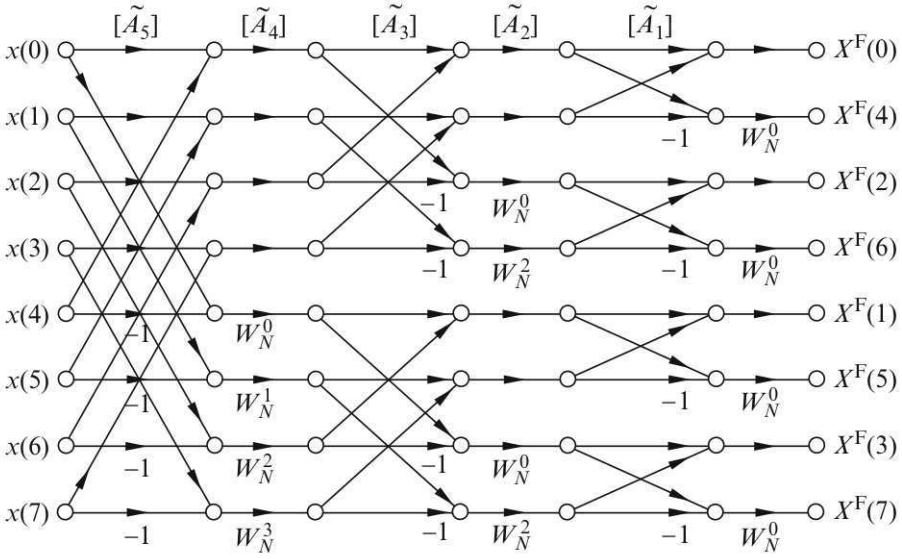
Considerando N como uma potência de um número inteiro r, podemos continuar estas decimações até atingir o valor da raiz r. Os algiritmos Cooley-Tukey de *radix*-r mais comuns utilizam as raízes de 2 e 4, pois os blocos básicos dessas transformadas podem ser implementados sem multiplicações.

A estrutura de um algoritmo Cooley-Tukey de radix-2 de 8 pontos é esquematizada na Figura 4, onde os círculos fazem a adição dos sinais indicados com uma seta. A quantidade de estágios do algoritmo é determinada por $log_r(N)$, sendo a DFT de tamanho 2 a menor estrutura de processamento, denominada "borboleta".

Por considerar a segmentação em partes cada vez menores da sequência de saída $X[k]$, o algoritmo da Figura 4 é conhecido como "Decimação na frequência". Tal organização possui a vantagem de realizar os cálculos *in-place*, isto é, os cálculos preliminares podem ser sobrescritos pelos próximos, visto que não serão mais necessários em partes futuras do processamento. Esta característica pode ser utilizada na elaboração de projetos ASIC por utilizarem poucos recursos de memória, propiciando uma economia de àrea.

As contribuições do algorítmo de Cooley-Tukey foram responsáveis por uma brusca redução operacional no cálculo numérico da DFT. O cálculo da transformada pela Equação 3.3 leva O^2 operações complexas, já a estrutura de decimação em frequncia reduziu esta complexidade para $Olog_2(O)$. Este ganho tornou possível a implementação de *softwares* e hardwares processadores de sinais mais rápidos e que consumissem menos recursos.

Figura 4 – Estrutura de treliça de uma FFT Cooley-Tukey radix-2 de 8 pontos.



Fonte: RAO; KIM; HWANG, 2010.

3.2 O algoritmo de FFT *split-radix*

A FFT *split-radix* foi desenvolvida a partir da combinação dos algoritmos *radix-2* e *radix-4*, pois seus blocos-base podem ser implementadas apenas com somas e subtrações, favorecendo o desenvolvimento de uma estrutura geral com menos multipliações.

Além da vantagem operacional, para algoritmos com $N = r^v$, a estrutura operacional em questão mantém o benefício do processamento "*in-local*" da FFT de Cooley-Tukey.

Segundo Rao, Kim e Hwang (2010), o cálculo é baseado na observação de que os coeficientes ímpares da DFT são melhores processados pelas estruturas de *radix-4* sendo as de *radix-2* mais apropriadas para os coeficientes pares.

A DFT de N pontos foi definida na equação de síntese 3.3, onde a grandeza $W_N^{kn} = e^{-j2\pi kn}$ tem módulo um e representa um deslocamento de fase do sinal de entrada $x[n]$. Assumindo $N = 2^K$, com $K \geq 2$ tem-se:

$$X[2k] = \sum_{n=0}^{\frac{N}{2}-1} \left(x[n] + x[n + \tfrac{N}{2}] \right) W_{\frac{N}{2}}^{kn} \qquad , k = 0, 1, \ldots, \frac{N}{2} - 1 \qquad (3.11)$$

$$X[4k+1] = \sum_{n=0}^{\frac{N}{4}-1} \left(x[n] - jx[n+\tfrac{N}{4}] - x[n+\tfrac{N}{2}] + jx[n+\tfrac{3N}{4}]\right) W_N^n W_{\frac{N}{4}}^{kn} \quad , k = 0, 1, \ldots, \frac{N}{4} - 1 \tag{3.12}$$

$$X[4k+3] = \sum_{n=0}^{\frac{N}{4}-1} \left(x[n] + jx[n+\tfrac{N}{4}] - x[n+\tfrac{N}{2}] - jx[n+\tfrac{3N}{4}]\right) W_N^{3n} W_{\frac{N}{4}}^{kn} \quad , k = 0, 1, \ldots, \frac{N}{4} - 1 \tag{3.13}$$

A Equação 3.11 trata-se de uma DFT de $N/2$ pontos, para os elementos pares de X, que receberá como entrada o sinal A_1^1, definido na equação 3.14. Já as Equações 3.12 e 3.13 são descritas por DFTs de tamanho $N/4$ responsáveis pelo processamento dos elementos ímpares de X, recebendo como entrada os sinais A_1^2 e A_2^2 definidos nas equações 3.16 e 3.17, respectivamente.

$$A_1^1[n] = x[n] + x[n + \frac{N}{2}] \qquad , n = 0, 1, \ldots, \frac{N}{2} \tag{3.14}$$

$$A_2^1[n] = x[n] - x[n + \frac{N}{2}] \qquad , n = 0, 1, \ldots, \frac{N}{2} \tag{3.15}$$

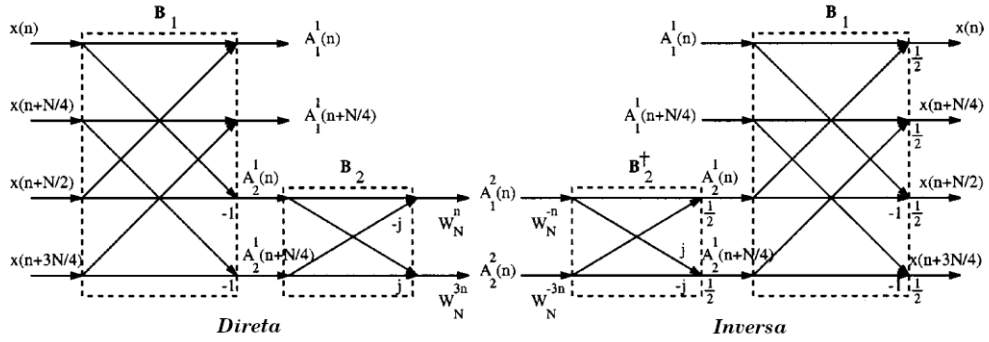
$$A_1^2[n] = W_N^n \left(A_2^1[n] - jA_2^1[n + \tfrac{N}{4}]\right) \qquad , n = 0, 1, \ldots, \frac{N}{4} \tag{3.16}$$

$$A_2^2[n] = W_N^{3n} \left(A_2^1[n] + jA_2^1[n + \tfrac{N}{4}]\right) \qquad , n = 0, 1, \ldots, \frac{N}{4} \tag{3.17}$$

Para Duhamel e Vetterly (1999),o algoritmo descrito possui o menor número conhecido de operações (multiplicações e somas) dentre os algoritmos de FFT com com tamanhos que são potências de 2. As estruturas das borboletas *split-radix* estão ilustradas na Figura 5.

Deve-se atentar para o fato de que o processamento é feito com dados complexos, onde cada multiplicação complexaa é composta por quatro multiplicações reais. Uma vez que esta é uma das operações que mais consome potência e tempo na concepção de ASICs, o montante destes cálculos aritméticos impacta diretamente no consumo e performance do SoC como um todo. Sendo assim, a FFT *split-radir* é uma escolha viável para o desenvolvimento de uma microarquiteruta de baixo consumo, com algumas vantagens de área (devido ao cálculo *in-place*) e performance. Estão espostas na tabela 1 a quantidade

Figura 5 – Estrutura de uma borboleta da FFT *split-radix*



Fonte: ORAINTARA, 2002.

Tabela 1 – Número de multiplicações complexas não triviais para diferentes algoritmos de FFT.

N	*radix-2*	*radix-4*	*split-radix*
8	2	2	2
16	10	8	8
32	34	28	26
64	98	76	72
128	258	204	186
256	642	492	456
512	1538	1196	1082
1024	3586	2732	2504
2048	8194	6316	5690
4096	18434	13996	12744
8192	40962	31404	28218

Fonte: YEH; JEN, 2003.

Tabela 2 – Número de multiplicações reais não triviais para diferentes algoritmos de FFT.

N	*radix-2*	*radix-4*	*split-radix*
16	24	20	20
32	88		68
64	264	208	196
128	712		516
256	1800	1392	1284
512	4360		3076
1024	10248	7856	7172

Fonte: DUHAMEL; HOLLMANN, 1984.

de multiplicações complexas não triviais utilizando diferentes algoritmos de FFT por decimação na frequência.

Tabela 3 – Número de adições reais para diferentes algoritmos de FFT.

N	*radix-2*	*radix-4*	*split-radix*
16	152	148	148
32	408		388
64	1032	976	964
128	2504		2308
256	5896	5488	5380
512	13566		12292
1024	30728	28336	27652

Fonte: DUHAMEL; HOLLMANN, 1984.

Pela análise das tabelas 2 e 4, podemos inferir que a vantagem operacional da FFT *split-radix* supera a dos algoritmos *radix-2* e *radix-4* tanto no número de adições e multiplicações, sendo este ganho tão superior quanto maior for o tamanho da transformada N.

É importante notar que nos a ordem da sequência de saída dos algoritmos descritos é diferente da de entrada (Figura 4). Esta disposição é consequência da transformação realizada nos índices de tempo e freqência (Equações 3.7 e 3.8) e é conhecida como bit-reversa, que pode ser determinada pela reflexão da representação binária dos índices da sequência de entrada, ou segundo Oppenheim (2013, p.432), "Se (n_2, n_1, n_0) é a representação binária do índice da sequência $x[n]$, então o valor da sequência $x[n_2, n_1, n_0]$ é armazenado na posição do vetor $X_o[n_0, n_1, n_2]$".

Por apresentar as vantagens operacionais supracitadas(que podem se traduzir em ganhos de consumo em uma arquitetura *low-power*), o algoritmo *split-radit* será utilizado para a implementação do bloco IntFFT, objeto deste trabalho.

3.3 A transformada rápida de Fourier inteira

Um dos fatores mais importantes na implementação em hardware de processadores digitais de sinais leva em conta o *tradeoff* entre a quantidade de *bits* utilizados na representação dos dados e a área do bloco. Uma vez que os sinais tratados são quantizados em uma palavra binaria de comprimento finito, a resolução dos dados resultantes das operações aritméticas aumenta sendo necessário truncar tais valores em uma representação de ponto fixo. No entanto tal truncamento confere um erro de representação ao dado que acaba por destruir a inversibilidade de uma tranformação (como a DFT) linear, uma vez que os *twiddle factors* são quantizados.

Com o intuito de preservar a a propriedade de reversivilidade, uma opção é traduzir

as multiplicações complexas da FFT em "esquemas *lifting*", possibilitando a reconstrução do sinal sem perdas. A transformada de Fourier inteira (IntFFT) trata-se, portanto, do cálculo numérico da DFT de modo que o sinal de entrada possa ser reconstruído a partir do sinal de saída com erro de quantização nulo. Isso favorece a implementação deste algoritmo na codificação e recuperação de sinais sem perdas.

Uma vez que os *twiddle-factors* são exponenciais complexas da forma $e^{j\theta}$, tal que sua magnitude e fase são um e θ, respectivamente, seu inverso é igual a seu conjugado complexo.

Seja $a \in \mathbb{C}$ um número de magnitude um, tal que $Re\{a\} = c \in \mathbb{R}$ e $Im\{a\} = s \in \mathbb{R}$,

$$a = c + js$$

A versão quantizada de a, a^q é definida por:

$$a^q = c^q + js^q$$

Onde $c^q \in \mathbb{Z}$ e $s^q \in \mathbb{Z}$ são as partes quantizadas de c e s. O inverso de a^q é:

$$\frac{1}{a^q} = \frac{1}{c^q + js^q} = \frac{c^q - js^q}{(c^q + s^q)^2} = \frac{c^q}{|a^q|^2} - j\frac{s^q}{|a^q|^2}$$

Devido a quantização, $|a^q| \neq 1$, não sendo mais uma representação de $|a|$, ou seja, mesmo sendo a^q um número binário de comprimento finito, é atribuído um erro de arredondamento intrínseco à representação de ponto-fixo.

Conforme dito anteriormente, cada multiplicação complexa compõe-se de quatro multiplicações reais:

$$y = y_r + jy_i = ax = (cx_r - sx_i) + j(cx_i + sx_r)$$

Na forma matricial, temos:

$$y = \begin{bmatrix} 1 & j \end{bmatrix} \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} x_r \\ x_i \end{bmatrix} = \begin{bmatrix} 1 & j \end{bmatrix} \mathbf{R} \begin{bmatrix} x_r \\ x_i \end{bmatrix} \tag{3.18}$$

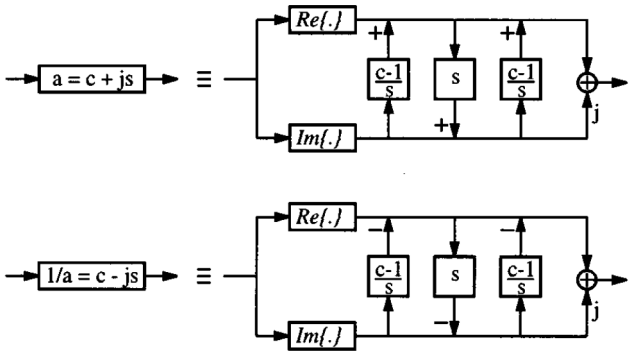
Decompondo a matriz $\mathbf{R}$ em passos *lifting*, temos:

$$\mathbf{R} = \begin{bmatrix} c & -s \\ s & c \end{bmatrix} = \begin{bmatrix} 1 & \frac{c-1}{s} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ s & 1 \end{bmatrix} \begin{bmatrix} 1 & \frac{c-1}{s} \\ 0 & 1 \end{bmatrix} \tag{3.19}$$

Esta decomposição em três passos é ilustrada na Figura 6, onde os fatores $\frac{c-1}{s}$ e s são os coeficientes *lifting*. Esta estrutura realiza a multiplicação complexa de dois números utilizando três multiplicações reais e 2 somadores, de tal forma que o inversibilidade é garantida pela simples comutação dos sinais dos somadores. Esta redução de um multiplicador é uma vantagem do ponto de vista de consumo.

A quantização dos coeficientes *lifting* pode ser feita tal que a reversibilidade é garantida, mesmo com a inserção de elementos não lineares (como arredondamentos e truncamentos) após suas multiplicações preliminares.

Figura 6 – Esquema de *lifting* para cálculo de multiplicação complexa.



Fonte: ORAINTARA, 2002.

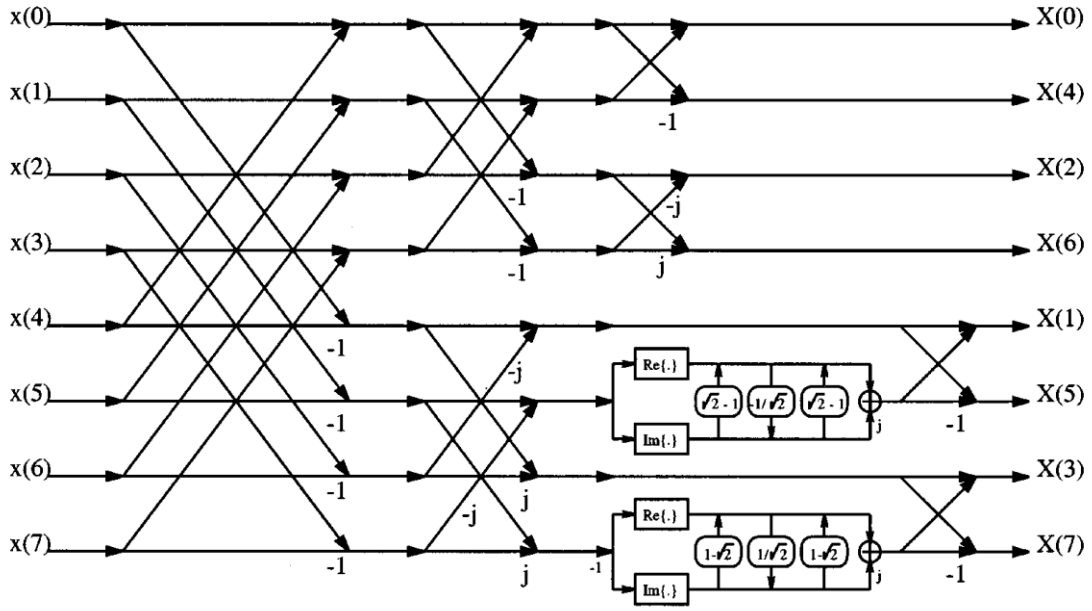
Inserindo os multiplicadores *lifting* no cálculo da FFT *split-radix*, estaremos concebendo uma IntFFT, que produz um mapeamento de inteiro para inteiro entre os sinais quantizados de entrada e saída, sendo possível a realização das transformadas direta e inversa apenas pela troca de sinal nas somas preliminares dos coeficientes dos multiplcadores. A treliça da FFT de 8 pontos está ilustrada na Figura 7.

A representação em passos *lifting* dos termos $W_N^k e^{j\theta}$, com $\theta = \frac{2\pi k}{N}$ pode ser determinada a partir da equação 3.19, de modo que cada elemento da matriz $\mathbf{R}$ é quantizado por um operador não linear Q.

A precisão dos cálculos internos foi determinada por Oraintara (2002), com base na consideração de que os números a serem estruturados em esquemas *lifting* estão entre -1 e 1 e que os coeficientes foram quantizados com uma resolução suficientemente grande. Nessa conformidade, "a implementação de cada *twiddle-factor* incrementa a resolução da entrada em,pelo menos um bit".

A matriz $\mathbf{R}$ para os um dado θ pode ser determinada por:

Figura 7 – Estrutura de treliça da IntFFT de 8 pontos.



Fonte: ORAINTARA, 2002.

$$\mathbf{R}_\theta = \begin{bmatrix} cos(\theta) & -sen(\theta) \\ sen(\theta) & cos(\theta) \end{bmatrix}$$

Os coeficientes *lifting* são:

$$L_0^{1,4}(\theta) = \frac{c-1}{s} = \frac{cos(\theta)-1}{sen(\theta)} \tag{3.20}$$

$$L_1^{1,4}(\theta) = s = sen(\theta) \tag{3.21}$$

As equações 3.20 e 3.21 são validas para $\theta \in (-\frac{\pi}{2}, \frac{\pi}{2})$, pois $cos(\theta) > 0$, acarretando que o módulo do numerador da equação 3.20 seja menor que 1 e

$$\left|\frac{cos(\theta)-1}{sen(\theta)}\right| = \left|\frac{\sqrt{1-sen(\theta)^2}-1}{sen(\theta)}\right| = \left|sen(\theta)\right| \left|\frac{1}{\sqrt{1-sen(\theta)^2}+1}\right| < \left|sen(\theta)\right| < 1$$

Se $\theta \in (-\frac{\pi}{2}, \frac{\pi}{2})$, o módulo do denominador da Equação 3.20 será maior que 1, tal que o módulo do coeficiente em questão será maior que 1, quebrando a restrição descrita

anteriormente. Para cobrir os casos dos quadrantes 2 e 3 do círculo trigononétrico, podemos valer do seguinte artifício:

$$\mathbf{R}_\theta = -\mathbf{R}_{\theta+\pi} = - \begin{bmatrix} cos(\theta) & sen(\theta) \\ -sen(\theta) & -cos(\theta) \end{bmatrix} \tag{3.22}$$

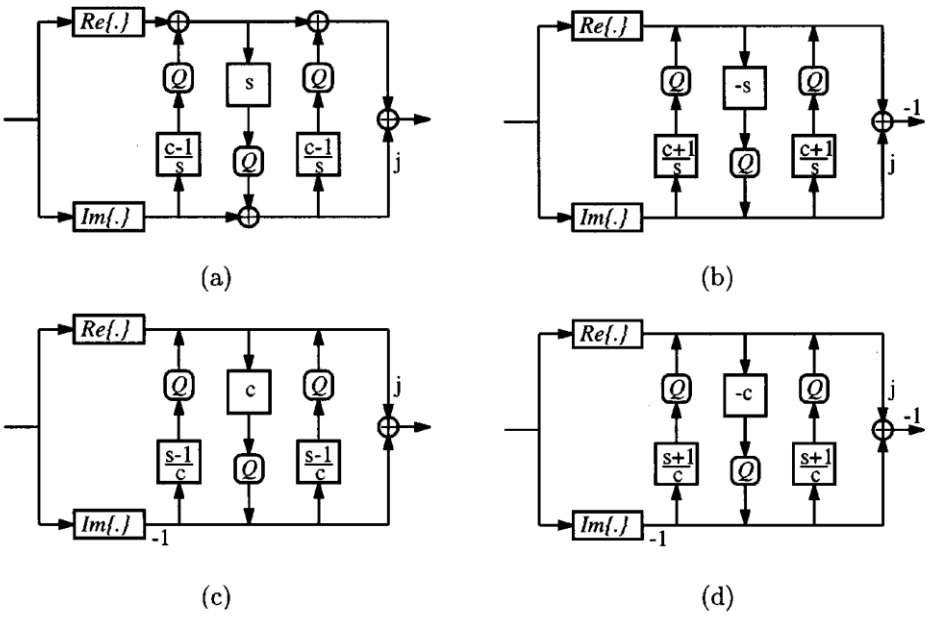
Com isso, os novos coeficientes serão:

$$L_0^{2,3}(\theta) = \frac{c+1}{s} = \frac{cos(\theta)+1}{sen(\theta)} \tag{3.23}$$

$$L_1^{2,3}(\theta) = -s = -sen(\theta) \tag{3.24}$$

A implementação do multiplicador complexo na IntFFT deve contar com a escolha correta dos coeficientes *lifting* segundo a posição do *twiddle factor* no plano complexo. São apresentadas 4 opções de esquemas lifting, de acordo com o ângulo θ, sendo as Figuras 8(a) e 8(b) referentes às equações 3.21, 3.22, 3.23 e 3.24.

Figura 8 – Multiplicadores *lifting* para diferentes intervalos de θ



Fonte: ORAINTARA, 2002.

4 Estratégias de baixo consumo

Os crescentes avanços tecnológicos dos últimos tempos, tais como IoT e a emergente tecnologia 5G, têm repercutido na sociedade e na indústria de modo que uma crescente necessidade de processamento em dispositivos portáteis é demandada, seja nos dispositivos *wearables* ou em etiquetas RFID. Dessa forma, o consumo de energia desses dispositivos é essencial para embarcar tecnologias com o aproveitamento máximo da bateria (ou até mesmo sem bateria, como é o caso de algumas etiquetas RFID que utilizam colheita de energia) face a capacidade de processamento.

Nesse sentido, este capítulo tem por objetivo introduzir as técnicas realizadas com o intuito de reduzir o consumo de potência de um ASIC, permitindo um projeto otimizado e de baixo consumo. Algumas das técnicas aqui apresentadas serão utilizadas no desenvolvimento do IP IntFFT, referentes ao escopo do *front-end*.

4.1 O consumo de potência

Antes de iniciar as técnicas de redução de consumo, se faz necessário investigarmos a potência dissipada nos *chips*, evidenciando suas causas e onde podemos agir de modo a diminuir seu impacto.

De uma forma geral, um SoC pode ser visto como a interconexão de inúmeros transistores[1] ,trabalhando como chaves, executando funções lógicas. Consideremos, então, os gráficos ilustrados na Figura 9 para a simplificação de um transistor visto como chave.

Quando o sinal de controle liga a chave em $t = 0$, há o movimento dos portadores no semicondutor de modo a estabelecer o canal de condução, fazendo a corrente i_T começar a crescer. A tensão entre o dreno e a fonte, v_T, se manterá no valor de "circuito aberto",

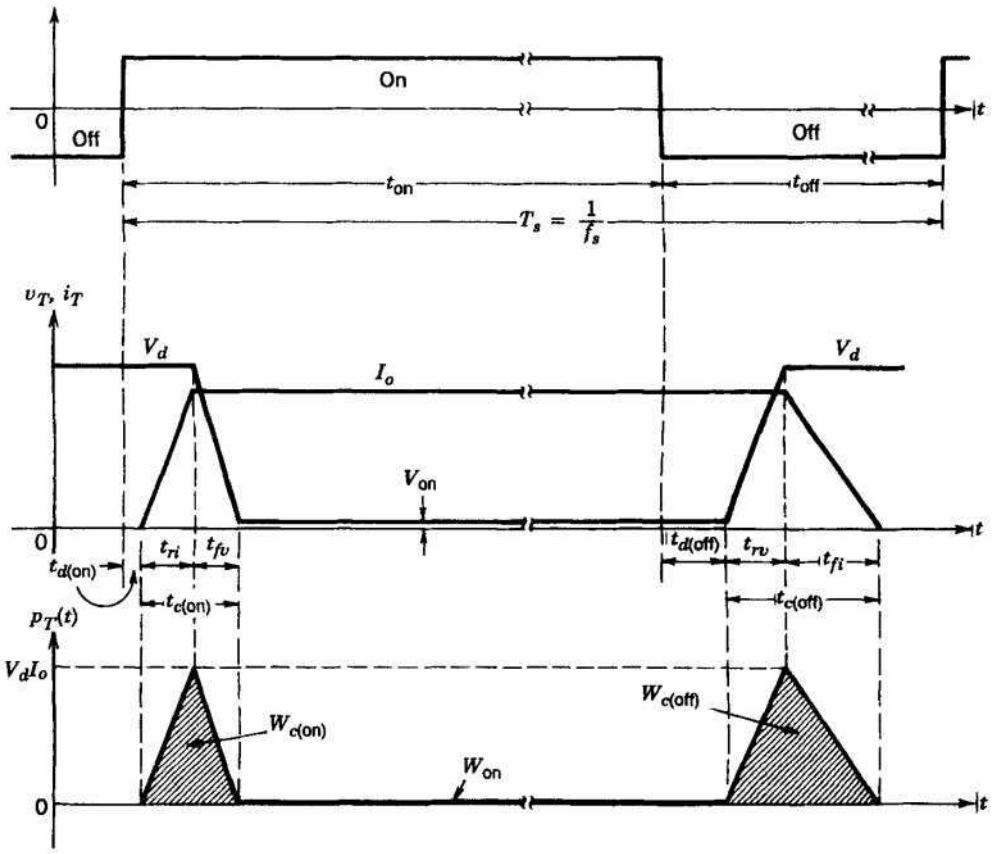
[1] Para se ter noção de quantidade, o processador Core 2 Quad da Intel atingiu a marca de 1 bilhão de transistores no ano de 2007

V_d, até que i_T atinja o valor de "curto-circuito" no instante $t = t_{d(on)} + t_{ri}$. A partir deste momento, a tensão irá cair durante o período t_{fv}, se mantendo no valor V_{on} correnpondente a queda de tensão da chave ligada, ou tensão de *threshold*.

O desligamento da chave se iniciará com o sinal de controle polarizando o terminal da porta do dispositivo semicondutor em $t = t_{on}$, acarretando o aumento da tensão v_T, à medida que o canal é fechado, até atingir o valor V_d em $t = t_{on} + t_{d(off)} + t_{rv}$. A partir deste momento a corrente irá diminuir, se anulando ao fim do período t_{fi}.

A multiplicação dos valores temporais de v_T e i_T define a potência dissipada na chave. Como pode ser observado na Figura 9, os picos de potência $W_{c(on)}$ e $W_{c(off)}$ se dão nos momentos de chaveamento, ao passo que a potência W_{on} refere-se as perdas de condução do dispositivo ou potência de *leakage*. Ambas as potências são proporcionais à tecnologia de fabricação dos semicondutores e têm impacto direto no consumo do *chip* como um todo.

Figura 9 – Potência dissipada em um transistor operando como chave.



Fonte: MOHAN, 2003.

Em um cenário mais realista, as portas lógicas utilizam a tecnologia CMOS, onde

há um dielétrico na porta dos MOSFETs, constituindo um capacitor, proporcional ao tamanho do trasistor, que confere um atraso de propagação à porta lógica. A corrente é, portanto:

$$I = C\frac{dC}{dt} \tag{4.1}$$

Conforme visto anteriormente, há duas componentes de potência associadas aos momentos de condução do transistor. São elas a potência dinâmica $P_{dinâmica}$ e a potência estática $P_{estática}$. Para Weste e Money (2011), a primeira se refere à potência consumida quando o capacitor é carregado e descarregado com frequência f enquanto que a segunda é devido ao pequeno fluxo de corrente $I_{estática}$ que flui entre a alimentação e a terra enquanto o transistor está desligado. As equações que regem tais perdas são:

$$P_{dinâmica} = \alpha C V_{DD}^2 f \tag{4.2}$$

$$P_{estática} = I_{estática} V_{DD} \tag{4.3}$$

Onde α é a atividade de chaveamento.

Sendo assim, a tensão de alimentação, a quantidade de chaveamentos e a frequência de operação do circuito são fatores que impactam diretamente o consumo de um SoC, sendo estudadas a seguir formas de minimizar seus efeitos.

4.2 *Clock Gating*

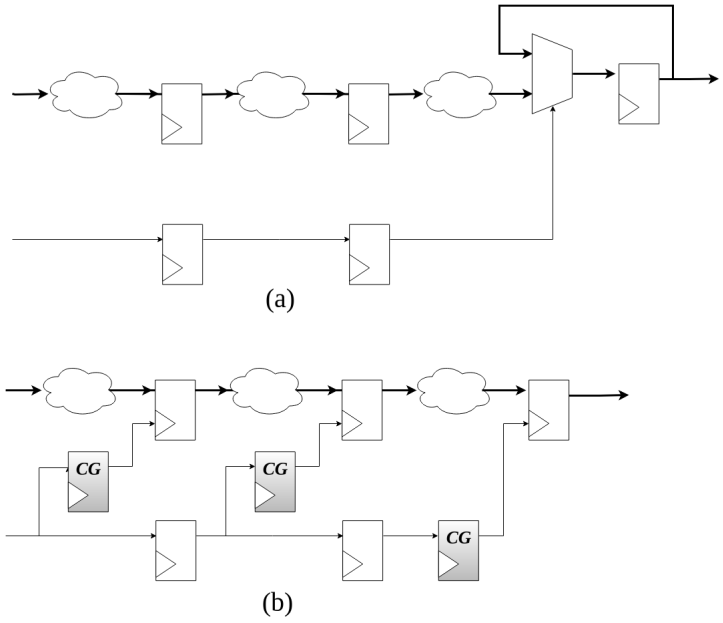
Em uma descrição RTL tradicional, cada *flip-flop* possui um sinal de *clock* conectado ao *clock* do sistema. Este estlo de codificação resulta em uma potência dinâmica consumida por partes combinacionais do circuito a cada transição, além do consumo dos registradores para manter seu resultado interno e o consumo referente aos *buffers* da árvore de clock do *design* (EMNETT; BIEGEL, 2000).

A ténica de *clock gating* consiste no desligamento do sinal de *clock* durante períodos de inatividade de partes do circuito. Esta técnica afeta diretamente o consumo potncia dinâmica através do fator α.

O projeto de uma codificação RTL com *clock gating* inicia-se a partir da identificação de grupos de *flip-flops* que podem ser habilitados ou desabilitados por um mesmo sinal de controle. Em termos práticos, é utilizada uma célula especial para controle do pino de *clock* destes elementos de memória.

A lógica de chaveamento do *clock* pode ser feita a partir de funções booleanas como *and* ou *or* com ou sem latch. Contudo, a inserção do *clock gating* não deve ser feita com céluas de portas lógicas comuns, pois estas apresentam um atraso de propagação do sinal intrínseco às capacitâncias parasitas que acarretam um atraso entre os sinais de *clock* da árvore e das células. O que se faz é prover a lógica necessária para a ferramenta de síntese instanciar as cálulas corretas no bloco em questão.

Figura 10 – Exemplo de inserção de células de *clock-gate*.



Fonte: O autor.

Um exemplo de inserção de células de *clock gating* é ilustrado na Figura 10 onde é possível notar a substituição do MUX de circulação de dados por uma célula que habilita o chaveamento do registrador apenas quando necessário, retendo o valor e economizando recursos. Além disso, também podemos elencar como vantagens desta técnica a redução do consumo de potência dinâmica e a redução da potência interna nos *flip-flops* abarcados pelo sinal de controle.

Os ganhos de potência nem sempre são esperados, quando da implementação deste método, pois dependendo do *design*, a desabilitação do *clock* pode acontecer muito raramente ou não acontecer, como em aplicações de alta taxa de transferência. Nesses casos, a inserção de *clock gating* pode aumentar a potência consumida do circuito, pois a inserção das células implica no aumento da área agravando a expressão da potncia estática. Cabe ao projetista a responsabilidade de perceber quais etapas de processamento podem ser chaveadas sem perdas no projeto.

4.3 *Power Gating*

A técnica de *power gating* consiste no desligamento da alimentação de algumas partes do curcuito em momentos estratégicos com o objetivo de zerar a corrente estática na equação 4.3, anulando as potências estática e dinâmica dissipadas enquanto o bloco não for habilitado.

Esta técnica se assemelha ao *clock gating* descrito na sessão anterior, todavia o impacto arquitetural de sua implementação é maior. A técnica de *power gating* é realizada no *back-end*, onde são postas células especiais responsávei pelo correto gerenciamento da alimentação dos blocos.

4.4 *Multi-Threshold* e *multi voltage*

A tensão de *threshold* dos transistores V_T refere-se a tensão mínima que deve ser aplicada entre a porta e a fonte de um MOSFET para se estabelecer um canal de condução. No âmbito de projeto em VLSI, células com baixo V_T chaveiam mais rapidamente, facilitando o *timing* de caminhos críticos ao passo que células com alto V_T consomem menos potência estática, propiciando a redução de consumo.

As ferramentas de síntese aloca as células adequadas, de modo a obter ganhos de PPA face as *contraints* impostas no projeto. Por exemplo, objetivando baixo consumo, a ferramenta de síntese "escolherá" as células mais lentas por terem menores perdas de potência estática e só alocará células mais rápidas em caminhos críticos necessários para fechar os requisitos de *timing*, tais como *setup*, *hold*, *recovery* e *removal*.

A técnica de *multi voltage* é implementada no *back-end*, onde diferentes áreas do circuito são alimentadas por diferentes valores de tensão. Tendo em vista as equações 4.2 e 4.3, as potências estática e dinâmica são impactadas diretamente pela tensão de alimentação imposta ao transistor de modo que uma redução na mesma minimizará as perdas (especialmente potência dinâmica pois a mesma cresce com o quadrado de V_{DD}).

Estão expostos na Tabela 4 as técnicas de baixo consumo e seus respectivos impactos arquiteturais.

No desenvolvimento do *front-end* do IP IntFFT, objeto deste trabalho, será utilizada a técnica de *clock gating*, reduzindo as perdas por chaveamento, sendo controlada por uma UC com codificação *gray* que será detalhada mais adiante.

Tabela 4 – Impactos das técnicas de baixo consumo.

Técnica	Potência	Performance	Penalidade de area	Impacto arquitetural	Impacto no projeto
Clock gating	Mediano	Pequeno	Pequeno	Pequeno	Pequeno
Multi VT	Mediano	Pequeno	Pequeno	Pequeno	Pequeno
Multi Voltage	Grande	Pequeno	Pequeno	Grande	Mediano
Power gating	Grande	Pequeno	Mediano/Grande	Grande	Mediano

Fonte: CADENCE, 2019.

5 Modelo do sistema

Seguindo o fluxo de desenvolvimento apresentado no capítulo 2, objetiva-se neste capítulo desenvolver um modelo de alto nível para testes da aritmética referente a IntFFT. Sendo assim, foi utilizada a linguagem Python devido suas bibliotecas aritméticas e velocidade de implementação.

Foi decidida uma abordagem *botom-up*, onde o *design* foi segmentado em blocos de processamento menores, sendo estes:

- Arredondador binário conforme padrão IEEE-754;
- Multiplicador complexo ortodoxo, com 4 multiplicadores e 2 somadores;
- Multiplicador *lifting* conforme as arquiteturas da Figura 8;
- FFT *split-radix* utilizando os multiplicadores *lifting.*

Além dos blocos-base, foram necessárias funções secundárias que realizavam operações simples, como a ordenação *bit* reversa e cálculos dos twiddles. Tais operações não serão implementadas diretamente em *hardware*, mas tiveram impacto positivo no desenvolvimento arquitetural.

5.1 Arredondador binário

A precisão dos nós internos nos blocos aritméticos que trabalham com a representação em ponto fixo é um fator limitante, pois está relacionada diretamente com a área total do chip. Se considerarmos uma soma, a precisão do resultado é igual a da entrada mais um, ao passo que o resultado produto entre dois números binários apresentará um tamanho de bits igual a soma dos bits de entrada. É necessário o devido arredondamento destes dados a uma dada precisão antes do truncamento, de modo a propagar o mínimo de erro.

Nese sentido, foi escrito o modelo do arredondador binário, seguindo o padrão IEEE-754, no qual apresenta diferentes regras de arredondamento adotadas pelos fabrincantes de *hardware* e *software*. O algoritmo utilizado foi o "*Ties to even*" que realiza o arredondamento com a preferência dada ao próximo número par em caso de empate.

```
#=================================================================#
# Function: binrnd          Description: Rnd to nearest: Tie to even.#
#                                        Rounding standard IEEE-754. #
# Inputs  :   num    : Binary fixed point number.                  #
#             nbi_in : Number of integer bits of input data.       #
#             nbi_out: Number of integer bits of output data.      #
#             nbw_out: Number of bits of output data.              #
# Outputs :   rnd_num: Binary fixed point rounded number .         #
#=================================================================#
def binrnd(num, nbi_in, nbi_out, nbw_out):
    nbw_in  = len(num)
    nbf_out = nbw_out - nbi_out
    n       = nbi_in + nbf_out - 1
    print(num[(n+1)::])
    print(num[n+1])
    print(num[n])
    print(n)
    if(num[n+1] == '0'):                # If digit n+1 is 0
        rnd_num = int(num[0:n],2)       # Round down.
    else:                               # If digit n+1 is 1
        if(int(num[(n+1)::],2) != 0):   #   If there's another 1 later.
            rnd_num = int(num[0:n],2) + 1       # Round up.
        else:                           # Else, ties to even.
            if(num[n] == '1'):          # If rnd_num is odd
                rnd_num = int(num[0:n],2) + 1   # Round up to even.
            else:                       # If rnd_num is even
                rnd_num = int(num[0:n],2) # Keep it even.
    return format(rnd_num,'#0'+str(n+2)+'b')[2:]
```

A regra geral para o arredondamento de números na representação de ponto fixo seguiu-se da seguinte maneira: Ao aproximar o número para a n-ésima posição observamos o *bit* seguinte. Se o valor na posição $n+1$ for igual a zero, deve-se arredondar para baixo. Adicionalmente, se dígito na posição $n+1$ for um e houver outro um em alguma das posições seguintes, deve-se arredondar para cima. O empate acontecerá quando o dígito $n+1$ for um e não houver mais outro um nas posições subsequentes. Nessa ocasião, deve-se arredondar dando preferência ao resultado par.

5.2 Multiplicadores complexos

O modelo dos multiplicadores complexos, ortodoxo e *lifting*, seguem a teoria exposta na seção 3.3 e foi feito de modo a testar o comportamento dos esquemas da Figura 8. O codificação dos esquemas é similar e exposta a seguir.

```
#=============================================================================#
# Function: lift_cplx_mult_a  Description: Cplx. mult lifting scheme.  #
# Inputs  :   di      : Direct/inverse selection.                      #
#             m0      : Complex number.                                #
#             m1      : Complex number to be decomposed into lifting.  #
# Outputs :   p0      : Product.                                       #
#=============================================================================#
def lift_cplx_mult_a(di, m0, m1):
    p0  = complex(0,0)
    c   = m1.real
    s   = m1.imag
    lift_coef_00 = (c-1)/s
    lift_coef_01 = s
    print("LCOEF_00: ", aux.dec2bin(lift_coef_00,10,7))
    print("LCOEF_01: ", aux.dec2bin(lift_coef_01,10,7))
    if(di == "d" or di == "D"):
        parc_1  = m0.real + (m0.imag*lift_coef_00)
        parc_2  = m0.imag + (parc_1*lift_coef_01)
        parc_3  = parc_1  + (parc_2*lift_coef_00)
        p0 = parc_3 + (j*parc_2)
    elif(di == "i" or di == "I"):
        parc_1  = m0.real - (m0.imag*lift_coef_00)
        parc_2  = m0.imag - (parc_1*lift_coef_01)
        parc_3  = parc_1  - (parc_2*lift_coef_00)
        p0 = parc_3 + (j*parc_2)
    else:
       print("[ERROR] - d/i - Invalid option!")
    return p0

def lift_cplx_mult_b(di, m0, m1):
    p0  = complex(0,0)
    c   = m1.real
    s   = m1.imag
    lift_coef_00 = (c+1)/s
    lift_coef_01 = -s
    print("LCOEF_00: ", aux.dec2bin(lift_coef_00,10,7))
    print("LCOEF_01: ", aux.dec2bin(lift_coef_01,10,7))
    if(di == "d" or di == "D"):
```

```
        parc_1  = m0.real + (m0.imag*lift_coef_00)
        parc_2  = m0.imag + (parc_1*lift_coef_01)
        parc_3  = parc_1  + (parc_2*lift_coef_00)
        p0 = -(parc_3 + (j*parc_2))
    elif(di == "i" or di == "I"):
        parc_1  = m0.real - (m0.imag*lift_coef_00)
        parc_2  = m0.imag - (parc_1*lift_coef_01)
        parc_3  = parc_1  - (parc_2*lift_coef_00)
        p0 = -(parc_3 + (j*parc_2))
    else:
       print("[ERROR] - d/i - Invalid option!")
    return p0
```

A entrada d/i indica se a multiplicação será direta ou inversa (divisão), sendo comprovado que é possível recuperar o valor de entrada utilizando o multiplicador referente ao intervalo do ângulo do valor complexo. Nesta etapa ainda não foi implementada a microarquitetura, mas a característica do seletor d/i indica que será implementada com flag de controle nas etapas posteriores.

5.3 FFT *split-radix*

O modelo da FFT *split-radix* foi realizado tendo como base as equações 3.14, 3.15, 3.16 e 3.17, onde foi possível observar a modularidade na implementação da mesma, que foi fundamental na etapa de microarquitetura.

```
#=============================================================#
# Length 1024 split-radix butterfly :                        #
#=============================================================#
def splt_rdx_bf1024(x):
    X = []
    for i in range(0,1024):            #N
        X.append(complex(0,0))
    A11 = []
    A12 = []
    A21 = []
    A22 = []
    for n in range(0,512):             #N/2
        A11.append(x[n]+x[n+512])      #N/2
        A12.append(x[n]-x[n+512])      #N/2
    for m in range(0,256):     #N/4
```

```
16          A21.append( (A12[m]-(j*A12[m+256]))*twiddle(1024,m) ) #N/4, N
            A22.append( (A12[m]+(j*A12[m+256]))*twiddle(1024,3*m) ) #N/4, N
18      bf00 = splt_rdx_bf512(A11)    #fftN/2
        bf01 = splt_rdx_bf256(A21)     #fftN/4
20      bf02 = splt_rdx_bf256(A22)     #fftN/4
        for k in range(0,512):    #N/2
22          X[2*k] = bf00[k]
        for l in range(0,256):     #N/4
24          X[(4*l)+1] = bf01[l]
            X[(4*l)+3] = bf02[l]
26      return X
```

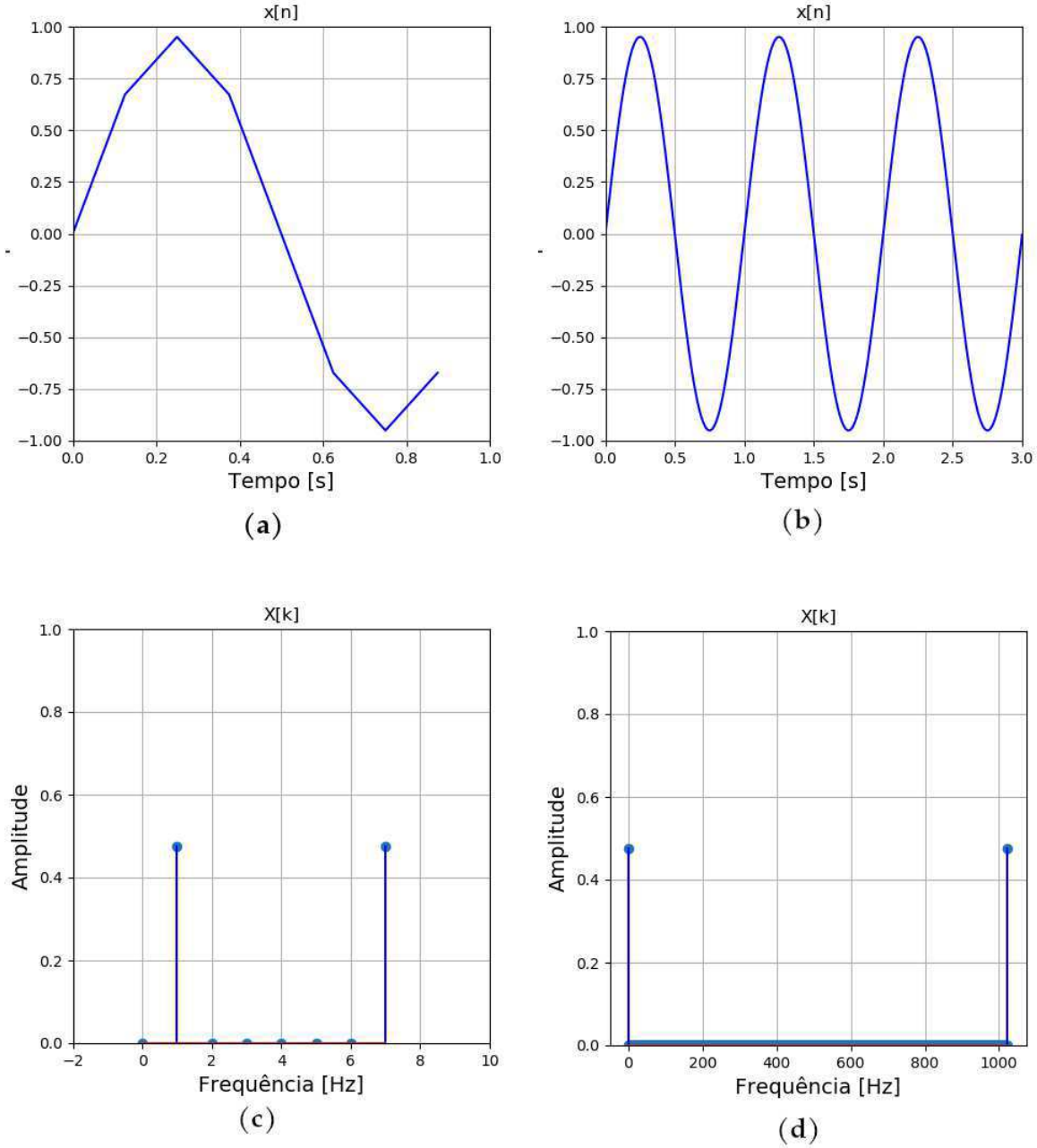
Como pode ser notado, o algoritmo se inicia com o cálculo dos $N/2$ elementos de A_1^1 e A_2^1, passando pelos multiplicadores de *twiddles* e em seguida para as transformadas de tamanho $N/2$ e $N/4$. Este esquema básico se repete para a implementação de transformadas com $N \geq 8$.

São expostos nas Figuras 11(a) e 11(c) os resultados da simulação do modelo proposto, em que um sinal senoidal é imposto como sequência de entrada, podendo ser visualizados os impulsos resultantes da correta transformada de Fourier discreta de 8 pontos da função seno. Ainda no algoritmo de FFT de 8 pontos, um sinal quadrado (Figura12(a)), com tempos de subida e descida não ideais, foi processado, obtendo a sua correta correspondência em frequência na Figura 12(c).

De modo similar ao realizado na FFT de 8 pontos, foi avaliada a aritmética de uma transformada de 1024 pontos, com a mesma implementação do algoritmo. Os resultados estão ilustrados nas Figuras 11(b) e 11(d), para a função seno e nas Figuras 12(b) e 12(d) para as ondas quadradas.

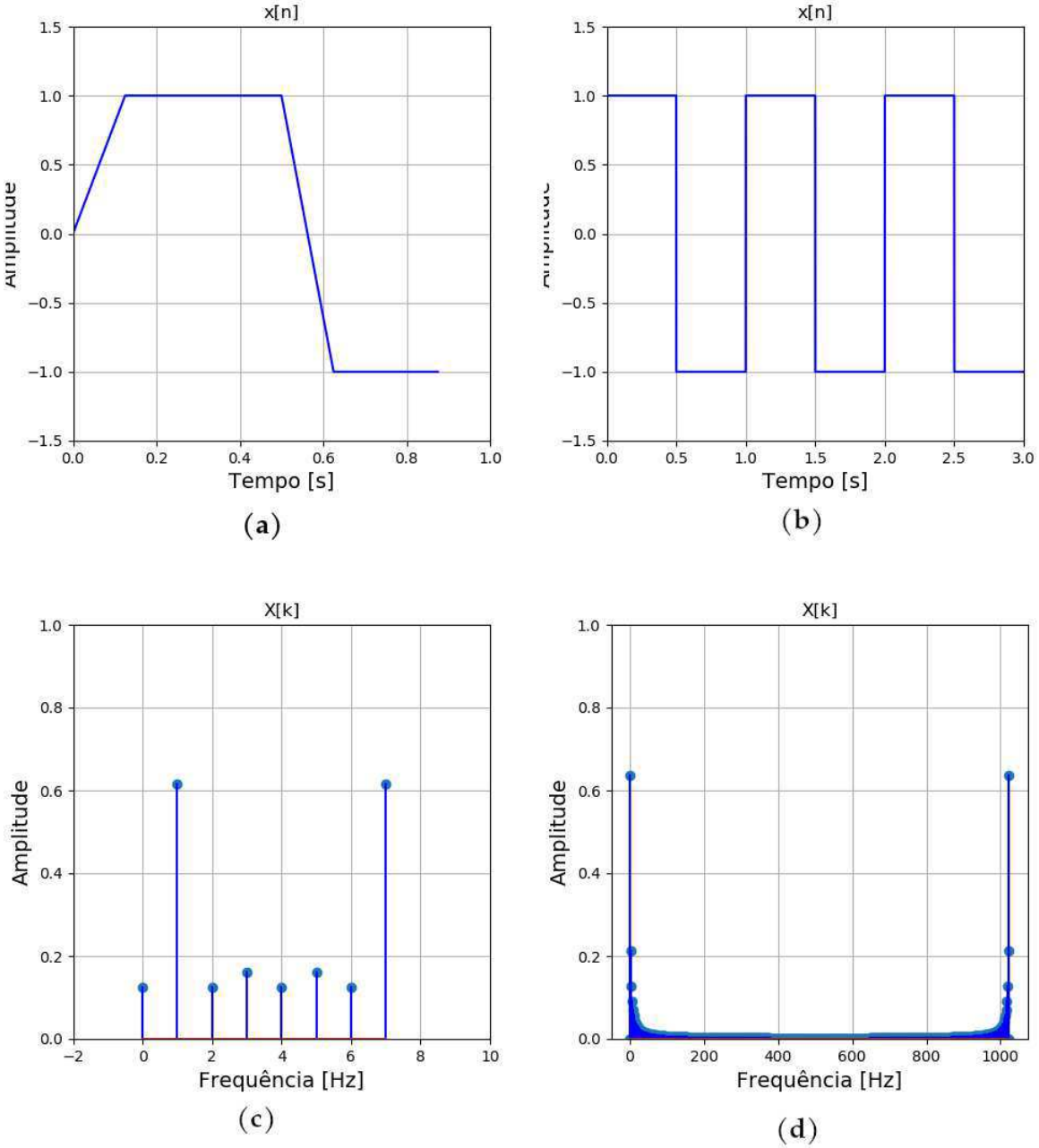
Os resultados do modelo foram contrapostos com a função de FFT da biblioteca *numpy*, do python, onde foi possível validar a aritmética do mesmo.

Figura 11 – Simulação do modelo de FFT proposto para uma entrada senoidal.



Fonte: O autor.

Figura 12 – Simulação do modelo de FFT proposto para uma onda quadrada.



Fonte: O autor.

6 Projeto do circuito digital

Com o modelo de referência descrito em linguagem de alto nível, dá-se início a etapa de projeto do circuito digital, onde serão definidos a microarquitetura e a interface do bloco IntFFT. A realização desta etapa em seguida da modelagem tem a vantagem de dar continuidade a algumas observações atentadas ateriormente, como a possibilidade de modularizar o projeto em uma arquitetura global e a insersão de uma entrada de controle nos multiplicadores *lifting* de modo a selecionar a operação direta ou inversa dos mesmos.

O projeto arquitetural foi realizado com o intuito de conceber um bloco modular e parametrizável, favorecendo seu reúso e adaptabilidade em outros IPs que venham instanciá-lo. As estratégias de baixo consumo de potência foram utilizadas desde a codificação inteligente da máquina de estados até a inserção da lógica de *clock gating* descrita na seção 4.2.

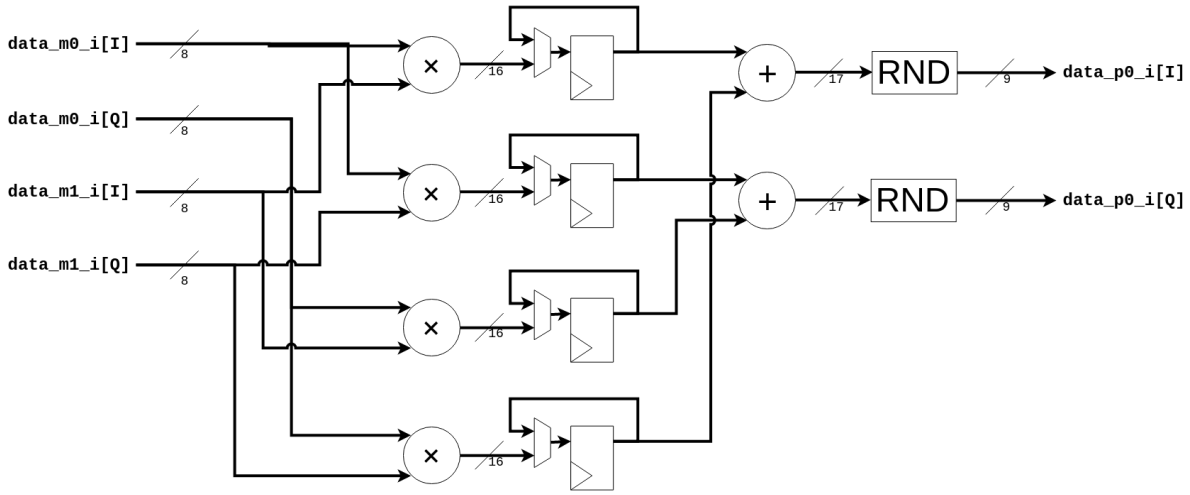
Assim como foi feito na etapa de modelagem, partiu-se de uma abordagem *bottom-up*, desenvolvendo a microarquitetura de cada unidade de processamento, descrevendo em HDL e avaliando sua funcionalidade por meio de *testbenches*, comparando com os modelos criados.

6.1 Os multiplicadores complexos

Foram desenvolvidas duas arquiteturas de blocos multiplicadores complexos: uma ortodoxa e a outra um esquema *lifting* denominados *clpx_mult_c2m4a2* e *lift_cplx_mult_c3m3a3*. As microarquiteturas e interfaces estão ilustrados nas Figuras 13, 14, 15 e 16 a seguir.

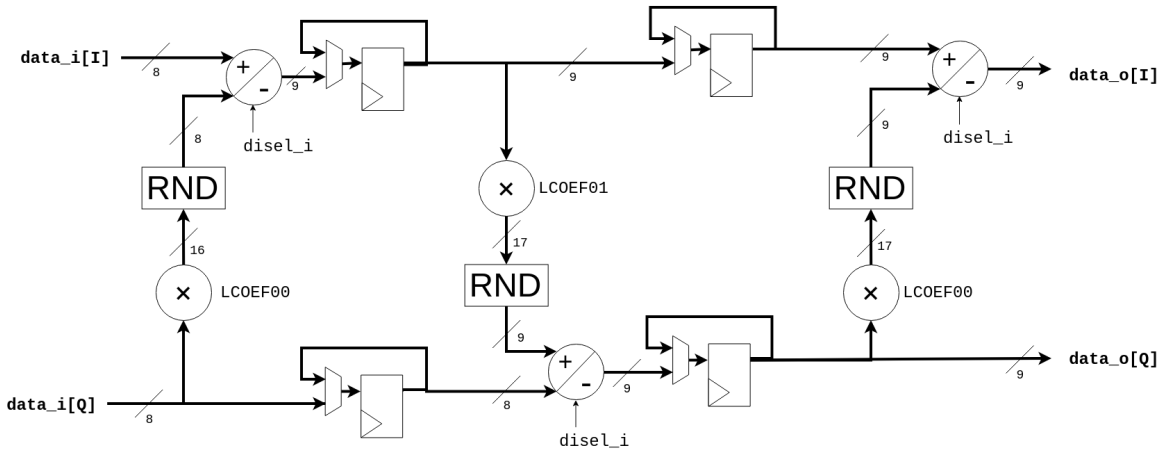
Além do menor número de multiplicadores, a microarquitetura do multiplicador complexo *lifting* foi desenvolvida de modo a receber como parâmetro os coeficientes já calculados e quantizados. Essa estratégia favorece a otimização arquitetural, por meio da ferramente de síntese, pois são considerados valores estáticos intrísecos a operação de multiplicação.

Figura 13 – Microarquitetura do bloco cplx_mult_c2m4a2.



Fonte: O autor.

Figura 14 – Microarquitetura do bloco lift_cplx_mult_c3m3a3.

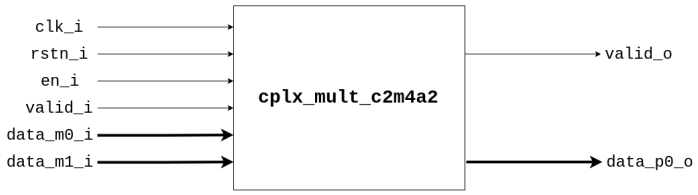


Fonte: O autor.

Também foi atentado para a inserção de registradores entre as operações mais complexas, como as multiplicações. Este esforço teve como objetivo diminuir o caminho crítico entre os *flip-flops*, favorecendo a performance e o uso de células mais lentas, que por sua vez consomem menos potência estática.

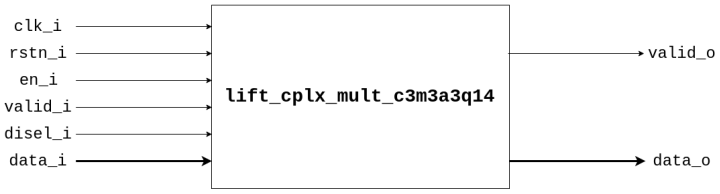
Foi implementado um protocolo de validade de dados através dos sinais *valid_i* e *valid_o*, que tem por objetivo formar uma estrutura de *data gating* onde o dado é processado apenas se um pulso no sinal de validade for enviado um ciclo de *clock* antes.

Figura 15 – Interface do bloco cplx_mult_c2m4a2.



Fonte: O autor.

Figura 16 – Interface do bloco lift_cplx_mult_c3m3a3.

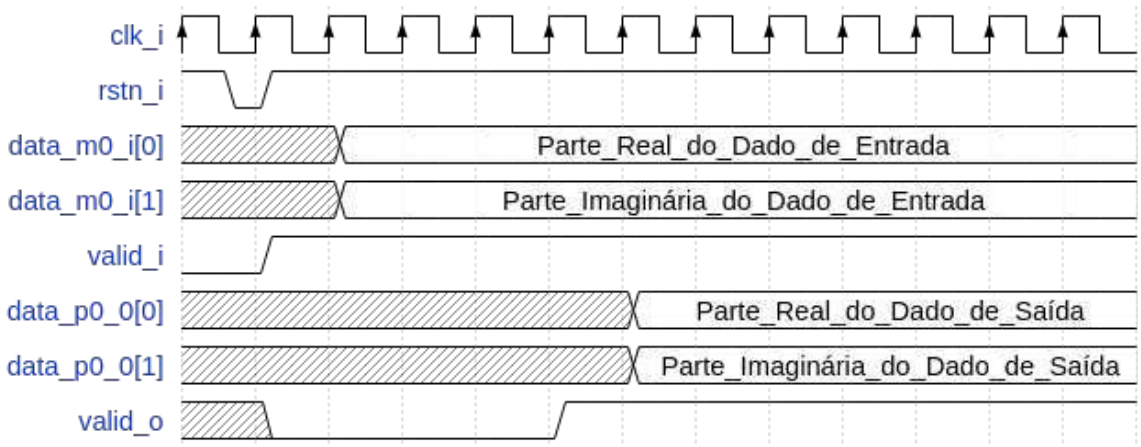


Fonte: O autor.

Do mesmo modo, o pino *valid_o* apresentar-se-á em nível lógico alto um ciclo antes da saída ser enviada. A forma de onda deste *handshake* é ilustrado na Figura 17.

O pino *en_i* é responsável pela habilitação ou desabilitação do sinal de *clock* por meio da técnica de *clock gating*, sendo necessário habilitar o bloco antes de qualquer controle enviado para o mesmo.

Figura 17 – Protocolo de validade adotado.



Fonte: O autor.

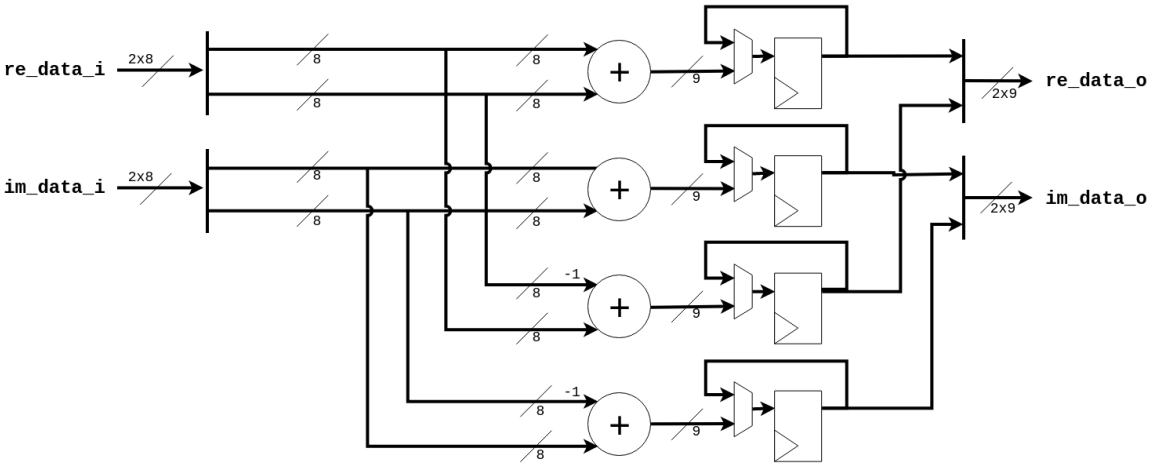
Conforme decisão durante o processo de modelagem, o módulo multiplicador complexo *lift* utiliza uma *flag* de seleção *disel_i* que quando posta em nível lógico alto realizará a multiplicação direta entre o sinal de entrada *data_i* e os coeficiente parametrizados na

instanciação do bloco. Complementarmente, quando a *flag* assume valor lógico zero, a operação inversa será executada pelo controle dos sinais dos somadores (Figura 14).

6.2 Borboletas *radix-2* e *radix-4*

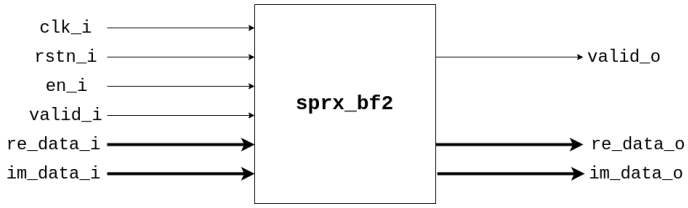
Elementos básicos de processamento do algoritmo de FFT utilizado, as estruturas das borboletas *radix-2* e *radix-4* não utilizam multiplicação e são concebidas obedecendo o protocolo de validade e *clock gating* descritos na seção anterior. Por conseguinte, a interface destes blocos são as mesmas e estão indicadas na Figura 19. As microarquiteturas de ambos os blocos estão ilustradas nas Figuras 18 e 20 a seguir e foram desenvolvidas a partir da observação de que a multiplicação pela constante imaginária j pode ser realizada apenas trocando as partes real e imaginária do dado complexo de entrada.

Figura 18 – Microarquitetura da borboleta *radix-2*.



Fonte: O autor.

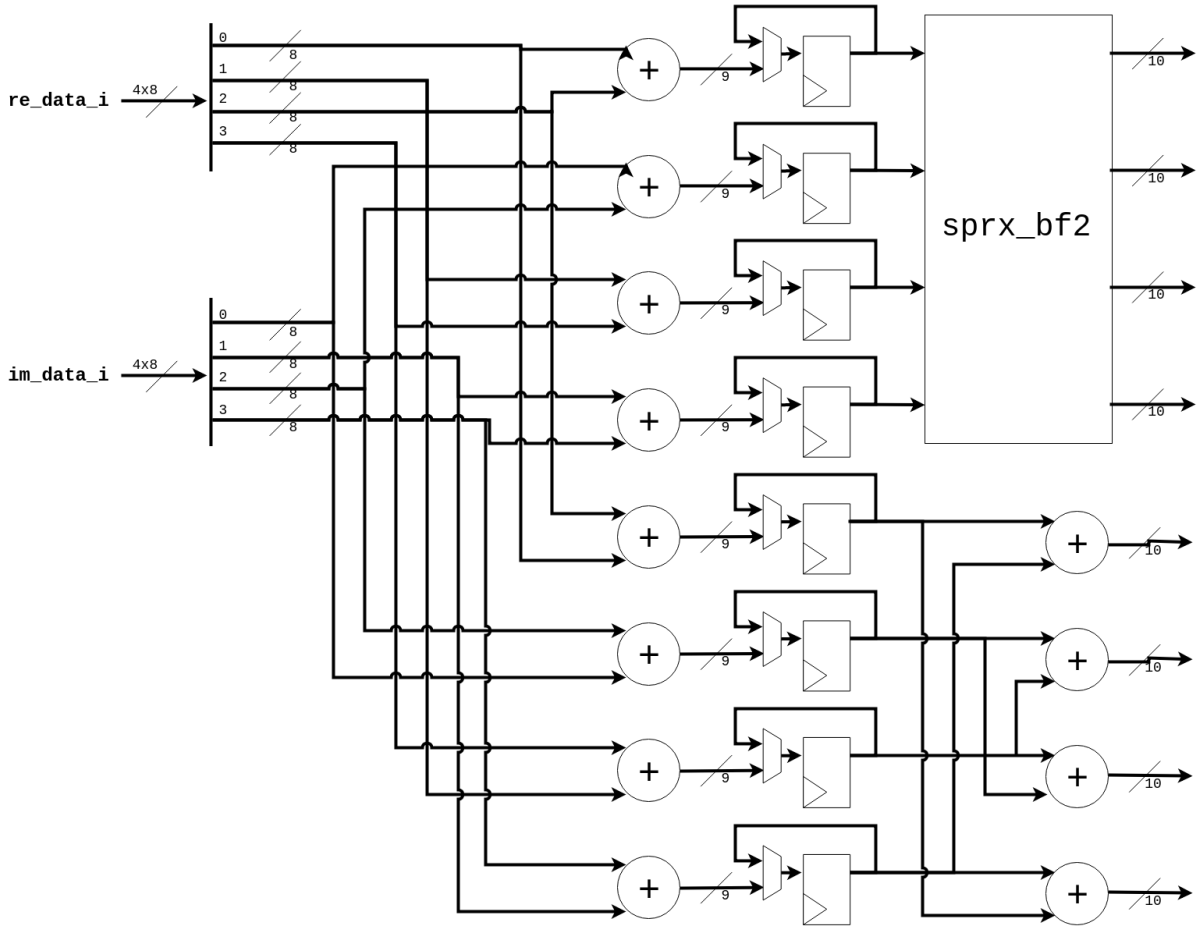
Figura 19 – Interface comum às borboletas *radix-2* e *radix-4*.



Fonte: O autor.

É importante observar que os blocos *sprx_bf4* e *sprx_bf2* processam dados complexos, sendo o primeiro responsável pela operação de quatro entradas paralelas e o segundo

Figura 20 – Microarquitetura da borboleta *radix-4*.



Fonte: O autor.

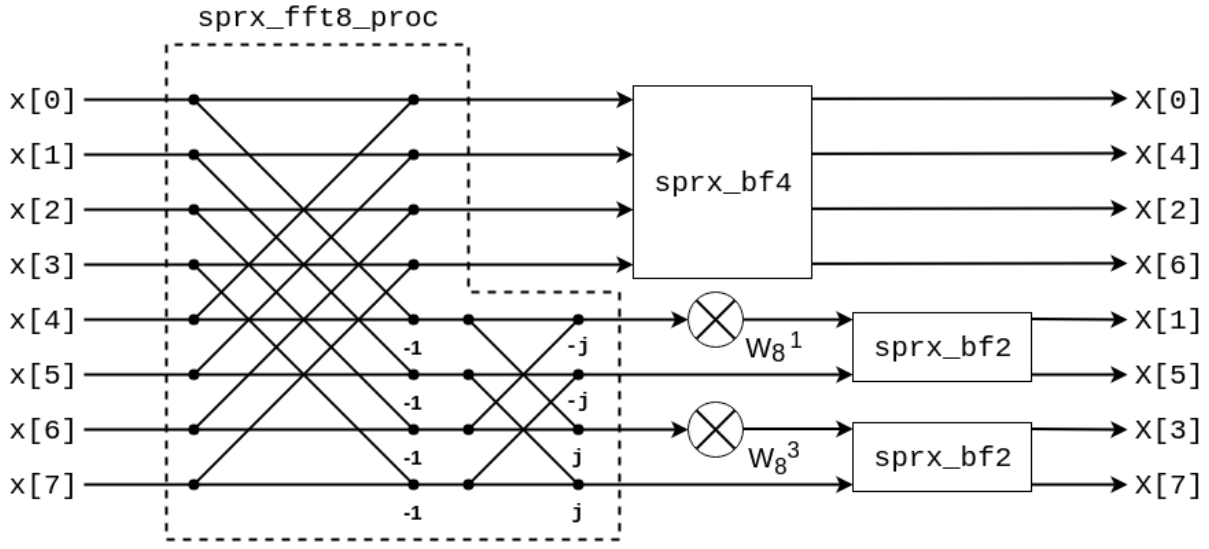
por duas. O projeto de tais blocos conta com registradores de saída com o intuito de conferir mais estabilidade e diminuição dos caminhos críticos na interface com outros blocos.

6.3 Unidade de pré-processamento de tamanho N

A implementação da transformada rápida de fourier pelo algoritmo de *split-radix* consiste na aplicação de um processamento preliminar nos N dados de modo a obter os sinais A_1^1 e A_2^1 das equações 3.14 e 3.15 que serão aplicados às transformadas de $N/2$ e $N/4$. este pré-processamento não requer multiplicações e pode ser encapsulado, conforme indicado na Figura 21.

Objetivando a modularidade, podemos observar que o bloco em questão depende, em termos gerais, da quantidade N de amostras da transformada, sendo sua operação

Figura 21 – Discriminação da unidade de pre processamento em uma FFT *split-radix* de 8 pontos.



Fonte: O autor.

uma manipulação aritmética de somas e comutações entre as partes real e imaginária. A seguir, determinaremos a equação da sequência de saída $y[n]$ como função das amostras de entrada $x[n]$.

No intervalo $0 \leq n \leq \frac{N}{2}$, temos:

$$y[n] = A_1^1[n] = x[n] + x[n + \frac{N}{2}] \tag{6.1}$$

No intervalo $\frac{N}{2} \leq n \leq \frac{3N}{4}$, temos:

$$\begin{aligned} y[n] = A_1^2[n] \cdot \frac{1}{W_N^n} &= A_2^1[n] - jA_2^1[n + \frac{N}{4}] \\ &= x[n] - jx[n + \frac{N}{4}] - x[n + \frac{2N}{4}] + jx[n + \frac{3N}{4}] \end{aligned} \tag{6.2}$$

No intervalo $\frac{3N}{4} \leq n \leq N$, temos:

$$\begin{aligned} y[n] = A_2^2[n] \cdot \frac{1}{W_N^{3n}} &= A_2^1[n] + jA_2^1[n + \frac{N}{4}] \\ &= x[n] + jx[n + \frac{N}{4}] - x[n + \frac{2N}{4}] - jx[n + \frac{3N}{4}] \end{aligned} \tag{6.3}$$

Dividindo os sinais de entrada e saída em suas partes real e imaginária do tipo $x[n] = x_{re}[n] + jx_{im}[n]$ e $y[n] = y_{re}[n] + jy_{im}[n]$ nas Equações 6.1, 6.2 e 6.3, generalizamos

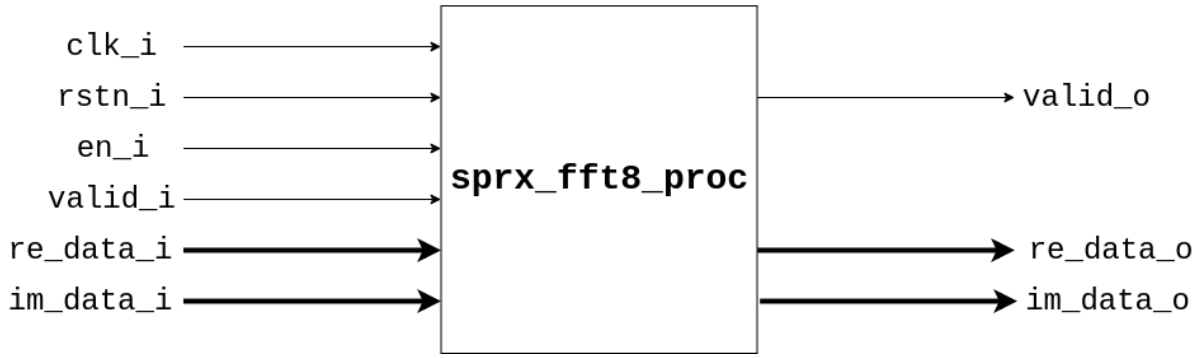
o cálculo para $N \geq 8$.

$$y_{re}[n] = \begin{cases} x_{re}[n] + x_{re}[n + \frac{N}{2}], & \text{se } 0 \leq n \leq \frac{N}{2} \\ x_{re}[n] + x_{im}[n + \frac{N}{4}] - x_{re}[n + \frac{2N}{4}] - x_{im}[n + \frac{3N}{4}], & \text{se } \frac{N}{2} \leq n \leq \frac{3N}{2} \\ x_{re}[n] - x_{im}[n + \frac{N}{4}] - x_{re}[n + \frac{2N}{4}] + x_{im}[n + \frac{3N}{4}], & \text{se } \frac{3N}{2} \leq n \leq N \end{cases} \tag{6.4}$$

$$y_{im}[n] = \begin{cases} x_{im}[n] + x_{im}[n + \frac{N}{2}], & \text{se } 0 \leq n \leq \frac{N}{2} \\ x_{im}[n] - x_{re}[n + \frac{N}{4}] - x_{im}[n + \frac{2N}{4}] + x_{re}[n + \frac{3N}{4}], & \text{se } \frac{N}{2} \leq n \leq \frac{3N}{2} \\ x_{im}[n] + x_{re}[n + \frac{N}{4}] - x_{im}[n + \frac{2N}{4}] - x_{re}[n + \frac{3N}{4}], & \text{se } \frac{3N}{2} \leq n \leq N \end{cases} \tag{6.5}$$

A arquitetura deste bloco consiste na aplicação das Equações 6.4 e 6.5 para cálculo das partes reais e imaginárias do sinal de saída. A interface está ilustrada na Figura 22, obedecendo o protocolo de validade e *clock gating*, para desligamento estratégico.

Figura 22 – Interface do bloco *sprx_fft8_proc*.



Fonte: O autor.

6.4 Unidade de controle

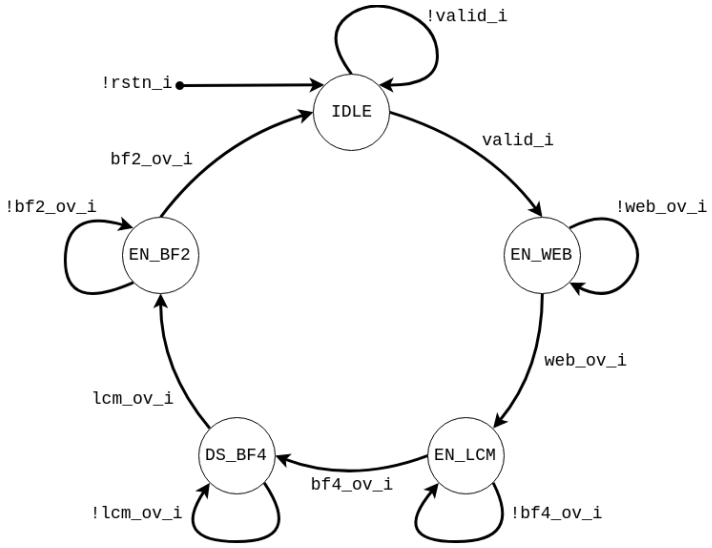
O diagrama de blocos esquematizado na Figura 21 constitui a operação do *data-path* da IntFFT parametrizável para $N \geq 8$ de sorte que, nesse ponto, seria possível a conexão entre as instâncias observando seu correto funcionamento, todavia necessita-se de uma unidade de controle que gerencie tanto os sinais de validade quanto as ativações e desativações das unidades de processamento, controlando o *clock-gating* numa arquitetura *low power*.

Tabela 5 – Descrição dos sinais da unidade de controle.

Sinal	Sentido	Descrição
bf2_ov_i	Entrada	*Flag* de validade de saída das borboletas *radix-2*.
bf4_ov_i	Entrada	*Flag* de validade de saída da borboleta *radix-4*.
lcm_ov_i	Entrada	*Flag* de validade de saída dos multiplicadores *lifting*.
valid_i	Entrada	*Flag* de validade de entrada da FFT.
web_ov_i	Entrada	*Flag* de validade de saída da unidade de pré processamento.
bf2_en_o	Saída	Sinal de ativação do *clock* das borboletas *radix-2*.
bf4_en_o	Saída	Sinal de ativação do *clock* das borboletas *radix-4*.
busy_o	Saída	*Flag* que quando ativada indica que o bloco está operante.
$\text{lcm}_e n\_o$	Saída	Sinal de ativação do *clock* dos multiplicadores *lifting*.
valid_o	Saída	*Flag* de validade de saída da FFT.
$\text{web}_e n\_o$	Saída	Sinal de ativação do *clock* da unidade de pré processamento.

Fonte: O autor.

Figura 23 – Máquina de estado para controle da IntFFT.



Fonte: O autor.

Nesse sentido, foi desenvolvida a unidade de controle com base nos sinais descritos na Tabela 5. A máquina de estado referente a operação da mesma é ilustrada na Figura 23.

Como o comportamento da máquina de estado consiste em uma operação simples, optou-se pela codificação *Gray* dos seus estados, pois isto garante que haverá apenas um chaveamento por transição de estados, reduzindo as perdas por potência dinâmica.

Após o *reset*, a máquina estará no estado IDLE, com todos blocos desabilitados até que uma entrada válida seja sinalizada. Quando isso ocorre, a FSM entrará no estado

WEB_EN, onde o pré processador irá gerar as sequências descritas nas equações 6.4 e 6.5. Ao fim de tal processamento, será assumido o estado EN_LCM onde são habilitados os blocos multiplicadores complexos e as transformadas de tamanho $N/2$ (No caso da FFT de 8 pontos, será a borboleta de *radix-4*), permanecendo nesse estado até que a FFT de $N/2$ pontostermine seus cálculos. O estado DS_BF4 é alcançado quando a *flag* de saída válida da FFT de $N/2$ pontos assumir nível lógico alto e se resume ao desligamento da mesma. O penúltimo estado é ativado quando os multiplicadores complexos acabarem de processar, o estado EN_BF2 consiste na ativação das transformadas de tamanho $N/4$ permanecendo nesse estado até o fim do processamento das mesmas. Ao final do cálculo, a FSM voltará ao estado IDLE até que um novo dado válido seja enviado.

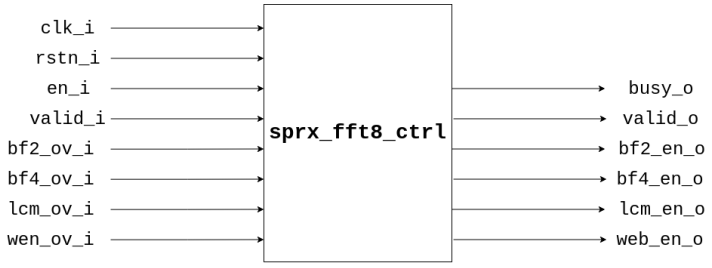
Os valores dos sinais de controle durante os estados da FSM estão expostos na Tabela 6, sendo sua interface ilustrada na Figura 24.

Tabela 6 – Valores lógicos assumidos pelas saídas da unidade de controle durante os estados.

	IDLE	EN_WEB	EN_LCM	DS_BF4	EN_BF2
bf2_en_o	0	0	0	0	1
bf4_en_o	0	0	1	0	0
busy_o	0	1	1	1	1
lcm_en_o	0	0	1	1	0
web_en_o	0	1	0	0	0

Fonte: O autor.

Figura 24 – Interface da unidade de controle.



Fonte: O autor.

6.5 Codificação RTL

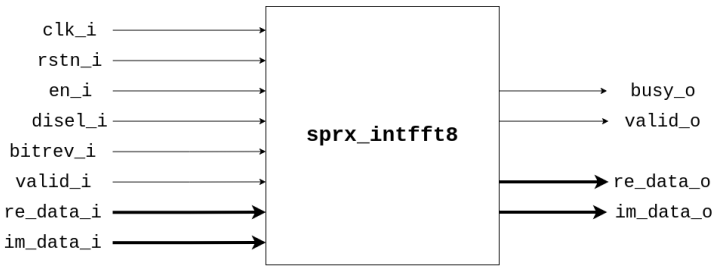
A codificação da microarquitetura foi realizada objetivando a adaptabilidade, modularidade e reúso do IP IntFFT. A par disso, os tamanhos de entradas foram parametrizados,

assim como a ordem dos sinais de saída, podendo estar na ordem direta ou na ordem *bit* reversa.

Neste trabalho, se optou por realizar a implementação de uma FFT de 8 pontos, entretanto o projeto arquitetural foi realizado com o intuito de abarcar a implementação de blocos de qualquer tamanho múltiplo de dois maior ou igual a 8. O projeto da FFT de 8 pontos utiliza as uma FFT de 4 pontos e duas transformadas de tamanho 2. Este bloco já seria instanciado na concepção de uma FFT de tamanho 16, por exemplo, onde seriam necessárias as transformadas de tamanhos 8 e 4, e assim sucessivamente.

A interface do bloco IntFFT de 8 pontos é ilustrada na Figura 25. O sinal de *reset* foi implementado com sensibilidade à borda negativa e deverá ser posto em nível lógico baixo somente quando o bloco estiver operante (isto é, *en_i*= 1), sendo recomendado um pulso de pelo menos um ciclo de *clock* antes de iniciar a operação. Para a execução da transformada direta, a entrada *disel_i* deve estar posta em nível lógico alto durante toda operação do bloco ao passo que *valid_i* deverá obedecer o protocolo de validade descrito na seção 6.1.

Figura 25 – Interface do bloco IntFFT de 8 pontos.



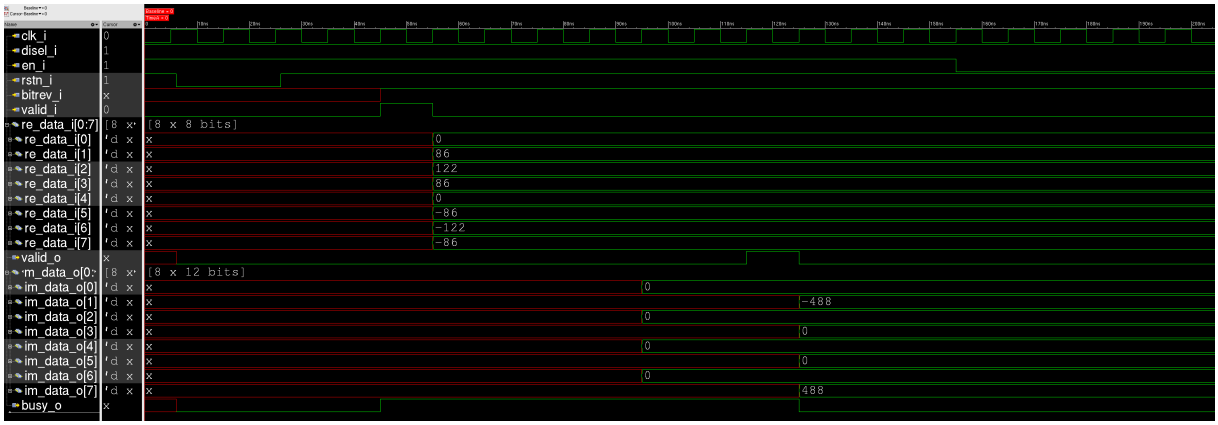
Fonte: O autor.

A entrada *bitrev_i* controla a disposição da sequência saída. Se posta em zero, a saída estará na ordem crescente, caso contrário estará em ordem *bit* reversa. Esta funcionalidade é importante na integração do bloco por outros de paralelismo superiores, os quais necessitam que a saída das instâncias das transformadas estejam na ordem *bit* reversa.

A codificação HDL foi feita utilizando a linguagem *systemverilog*, onde a funcionalidade individual de cada unidade implementada foi testada separadamente com um *testbench* próprio. Foram utilizadas as ferramentas **Incisive** e **SimVision** da Cadence para simulação RTL, onde as formas de onda referentes para o teste de sanidade do bloco *sprx_intfft8* foram geradas e expostas na Figura 26.

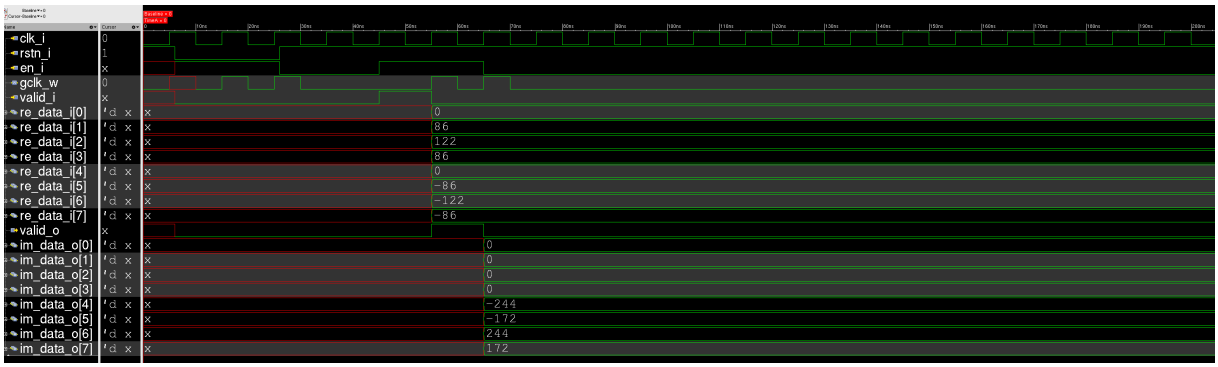
Quando se está descrevendo um bloco na linguagem *SystemVerilog*, deve-se atentar para a construção de funções lógicas sintetizáveis. O projetista deve ter em mente que

Figura 26 – Simulação RTL do bloco sprx_intfft8.



Fonte: O autor.

Figura 27 – Simulação RTL do bloco sprx_fft8_proc.

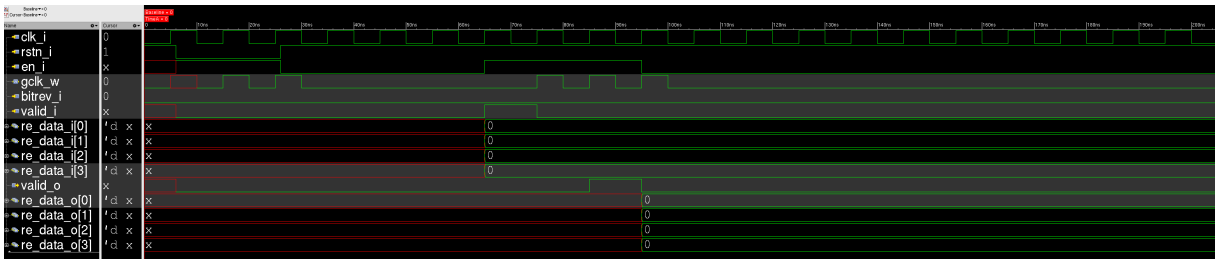


Fonte: O autor.

está concebendo um circuito e construções de alto nível podem não ser reconhecidas pelo sintetizador. Como exemplo, podemos citar a lista de sensibilidade em um bloco **always_ff**. A linguagem permite a descrição de muitos sinais na lista de sensibilidade, sendo que só será sintetizável o RTL que apresentar até dois sinais na lista de sensibilidade, uma vez que os *flip-flops* são sensíveis ao sinal de *clock* e *reset*.

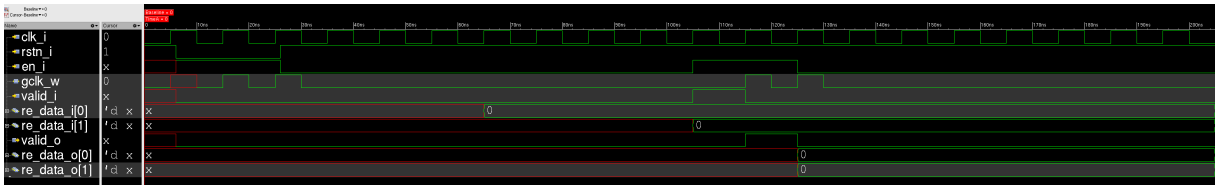
As Figuras 26 - 31 são referentes às formas de onda da simulação RTL de cada instância do bloco *sprx_intfft8*. Pode-se perceber que a execução dos blocos de processamento será feita apenas enquanto o sinal de controle *en_i* estiver em nível lógico alto, habilitando o sinal *gclk_w*.

Figura 28 – Simulação RTL do bloco sprx_bf4.



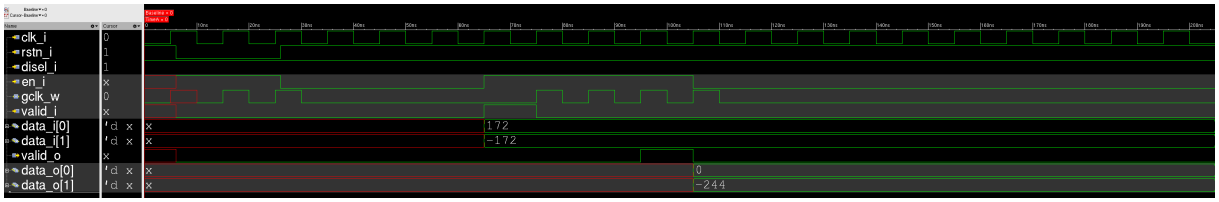
Fonte: O autor.

Figura 29 – Simulação RTL do bloco sprx_bf2.



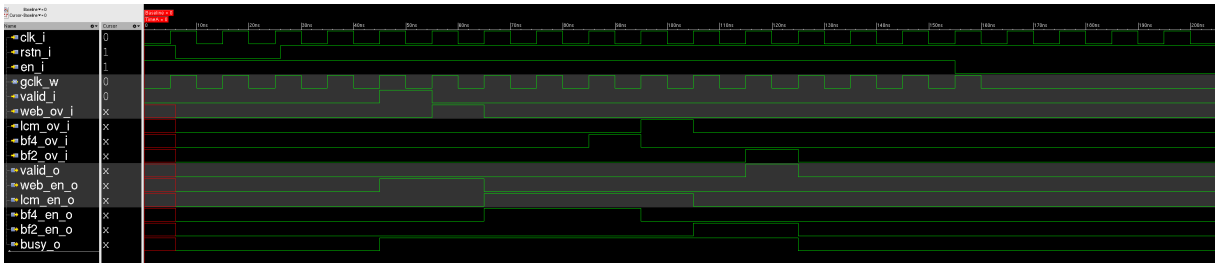
Fonte: O autor.

Figura 30 – Simulação RTL do bloco lift_cplx_mult_c3m3a3.



Fonte: O autor.

Figura 31 – Simulação RTL do bloco sprx_fft8_ctrl.



Fonte: O autor.

7 Síntese lógica

Uma vez realizada descrição em HDL da microarquitetura estabelecida em System-Verilog sintetizável e validada sua funcionalidade por meio de um *testbench* segundo o modelo estabelecido, dá-se início a síntese lógica que, conforme introduzida na seção 2.4, consiste na tradução do RTL em portas lógicas segundo a tecnologia adotada.

Para este fim, utilizou-se o PDK de 180 nanômetros da X-FAB, fabricante especializada em tal processo de fabricação, por meio da ferramenta *Genus* que realizou a síntese lógica segundo as *constraints* estabelecidas.

Como parâmetros gerais, foram utilizados comandos para inserção de células de *clock gating*, segundo a lógica de ativação estabelecida. As *timing libraries* utilizadas compreendem células com tensão $V_{DD} = 1,62V$ e temperatura de operação de $125°C$.

Além dos elementos básicos da síntese (*constraints* e *timing libraries*) foi utilizado o arquivo de atividade de chaveamento, gerado na simulação RTL, de modo a gerar uma síntese mais próxima da realidade operacional do circuito, estimando a potência consumida mais precisamente a partir da atividade de chaveamento simulada.

Os arquivos de tempo utilizados na etapa de STA foram importados a partir das *Captables*, que elencam valores de resistência e capacitância pertinentes ao processo de fabricação utilizado e que se traduzem em atrasos na propagação do sinal entre as células. Com o intuito de descobrir a maior frequência possível com tal arquitetura, a síntese lógica foi realizada diversas vezes no bloco *sprx_intfft8*, diminuindo o período do *clock* do bloco, de modo a atingir a menor folga de tempo *slack* possível. Concluindo-se que, para a tecnologia adotada, o bloco pode operar em uma frequência de até 100 MHz. Estão elencados nas Tabelas 7 - 11 um resumo dos relatórios de síntese evidenciando os atributos PPA em diferentes frequências.

A partir da análise dos resultados de síntese lógica, pôde-se comprovar que a potência dinâmica está diretamente relacionada com a frequência de operação do bloco, uma vez que tal resultado mais que dobrou quando investigamos as Tabelas 8 e 10, conforme

Tabela 7 – Resultados da síntese lógica do bloco *sprx_intfft8* a 20 MHz.

Performance (ρs)			
Período 50000	Setup 25000	Requisitado 25000	Slack 22254
Potência (mW)			
Estática 0,001	Interna 1,270	Dinâmica 2,166	Total 2,168
Área (μm^2)			
Contagem de células 2869	Área de células 92506,982	Roteamento 38916,661	Total 131423,643

Fonte: O autor.

Tabela 8 – Resultados da síntese lógica do bloco *sprx_intfft8* a 40 MHz.

Performance (ρs)			
Período 25000	Setup 1133	Requisitado 23867	Slack 5995
Potência (mW)			
Estática 0,001	Interna 2,371	Dinâmica 3,946	Total 3,948
Área (μm^2)			
Contagem de células 2871	Área de células 92489,421	Roteamento 38939,931	Total 131429,352

Fonte: O autor.

Tabela 9 – Resultados da síntese lógica do bloco *sprx_intfft8* a 60 MHz.

Performance (ρs)			
Período 16667	Setup 1133	Requisitado 15534	Slack 28
Potência (mW)			
Estática 0,001	Interna 3.477	Dinâmica 5.795	Total 5.796
Área (μm^2)			
Contagem de células 3186	Área de células 94509,005	Roteamento 42107,107	Total 136616,112

Fonte: O autor.

Tabela 10 – Resultados da síntese lógica do bloco *sprx_intfft8* a 80 MHz.

Performance (ρs)			
Período 12500	Setup 874	Requisitado 12500	Slack 0
Potência (mW)			
Estática 0,002	Interna 5,231	Dinâmica 8,901	Total 8,902
Área (μm^2)			
Contagem de células 3650	Área de células 99870,310	Roteamento 46738,309	Total 146608,619

Fonte: O autor.

Tabela 11 – Resultados da síntese lógica do bloco *sprx_intfft8* a 100 MHz.

Performance (ρs)			
Período 10000	Setup 611	Requisitado 10000	Slack 0
Potência (mW)			
Estática 0,002	Interna 6,0437	Dinâmica 10,874	Total 10,876
Área (μm^2)			
Contagem de células 4351	Área de células 113661,184	Roteamento 54313,813	Total 167974,997

Fonte: O autor.

o comportamento descrito pela Equação 4.2.

Também é possível notar que nos cenários das Tabelas 10 e 11 o *slack* (ou folga de tempo) se manteve nula. Esse resultado expressa o esforço ferramental em otimizar o projeto face as folgas de tempo resultantes, tal esforço acarretou uma redução de área e células onde os cálculos são repetidos até obter a *netlist* mais otimizada possível.

Após a síntese lógica foram realizadas simulações GTL, onde a funcionalidade do *netlist* foi posta à prova. Sendo as formas de onda da simulação RTL repetidas, sendo comprovada a correta operação do bloco, podemos elencar as seguintes características funcionais:

- Frequência de operação: 100 MHz;
- Latência de 7 ciclos de clock;
- Área total: 167974,997 μm^2;
- Potência consumida: 10,876 mW.

8 Conclusão

O bloco IntFFT foi desenvolvido seguindo o fluxo de desenvolvimento ASIC. Partindo da especificação de uma arquitetura de baixo consumo, as decisões arquiteturais foram tomadas de modo a contemplar altas frequências de operação. A modularidade e reúso de IP foram partes essenciais da filosofia do projetista, se fazendo possível a geração de FFTs de tamanhos múltiplos de dois, com a arquitetura *split-radix*.

Além das vantagens arquiteturais, a implementação do algoritmo da transformada de Fourier inteira, através dos multiplicadores *lifting*, permite a implementação do bloco principalmente em áreas de filtragem digital que funcionam tirando proveito da propriedade da convolução. Partindo de um sinal discreto, seu espectro é multiplicado pela função de transferência armazenada em uma ROM e trazido de volta para o domínio do tempo pela aplicação da IFFT. Nesse sentido, a utilização da IntFFT é vantajosa por três motivos: Primeiro por o mesmo *hardware* contemplar as operações de ambas transformações lineares, segundo por garantir que não haja a propagação de erro no processo devido as transformadas direta e inversa e finalmente por não necessitar de ROMs para armazenamento dos *twiddle factors* pré calculados, uma vez que os multiplicadores *lifting* são projetados segundo os coeficientes, passados por parâmetro, e otimizados pela ferramenta de síntese.

Foi realizado, portanto, o *front-end* de um bloco de SoC, partindo de uma ideia abstrata e entregando o *netlist* com instâncias de portas físicas. O fluxo de desenvolvimento pode ser continuado em trabalhos futuros, realizando desde etapas pertinentes ao *back-end* do bloco até metodologias de verificação, conferindo confiabilidade ao projeto.

Referências

AGNELLO, Janice S. *An Introduction to the Winograd Discrete Fourier Transform*. 1979.

ALEKHYA, V.; SRINIVAS, B. *Architectural Level Power Optimization Techniques for Multipliers*. International Journal of Engineering Trends and Technology-Volume3Issue5-2012, 2012.

BHAGAT, Nikhilesh et al. *High-throughput and compact FFT architectures using the Good–Thomas and Winograd algorithms*. IET Communications, v. 12, n. 8, p. 1011-1018, 2018.

COOLEY, James W.; TUKEY, John W. *An algorithm for the machine calculation of complex Fourier series*. Mathematics of computation, v. 19, n. 90, p. 297-301, 1965.

DUDA, Krzysztof. *Integer fast Fourier transform-implementation and application*. In: 2004 12th European Signal Processing Conference. IEEE, 2004. p. 1517-1520.

DUHAMEL, Pierre. *Implementation of "Split-radix" FFT algorithms for complex, real, and real-symmetric data*. IEEE Transactions on Acoustics, Speech, and Signal Processing, v. 34, n. 2, p. 285-295, 1986.

DUHAMEL, Pierre; HOLLMANN, Henk. *Split radix'FFT algorithm*. Electronics letters, v. 20, n. 1, p. 14-16, 1984.

EMNETT, Frank; BIEGEL, Mark. *Power reduction through RTL clock gating*. SNUG, San Jose, p. 1-11, 2000.

GAO, Ying. *Hardware implementation of a 32-point radix-2 FFT architecture*. 2015.

HARRIS, David; WESTE, N. *CMOS VLSI Design*. Pearson Education, Inc, 2010.

HEMNANI, Monika et al. *Hardware optimization of complex multiplication scheme for DSP application*. In: 2015 International Conference on Computer, Communication and Control (IC4). IEEE, 2015. p. 1-4.

HUANG, Zhijun. *High-level optimization techniques for low-power multiplier design*. 2003. Tese de Doutorado. University of California, Los Angeles.

KANNAN, Muniandi; SRIVATSA, Srinivasa. *Hardware Implementation Low Power High Speed FFT Core*. International Arab Journal of Information Technology (IAJIT), v. 6, n. 1, 2009.

KATHURIA, Jagrit; AYOUBKHAN, M.; NOOR, Arti. *A review of clock gating techniques*. MIT International Journal of Electronics and Communication Engineering, v. 1, n. 2, p. 106-114, 2011.

LI, Weidong; WANHAMMAR, L. *An FFT processor based on 16-point module*. In: Proc. of NorChip Conf. 2001. p. 125-130.

MADISETTI, Vijay (Ed.). *The digital signal processing handbook*. CRC press, 1997.

MEYER-BAESE, Uwe; MEYER-BAESE, U. *Digital Signal Processing with Field Programmable Gate Arrays*. Berlin: Springer, 2007.

OPPENHEIM, Alan V.; SCHAFER, Ronald W. *Processamento em tempo discreto de sinais*. 3. ed. São Paulo: Pearson Education do Brasil, 2012.

ORAINTARA, Soontorn; CHEN, Ying-Jui; NGUYEN, Truong. *Integer fast fourier transform (INTFFT)*. In: 2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No. 01CH37221). IEEE, 2001. p. 3485-3488.

ORAINTARA, Soontorn; CHEN, Ying-Jui; NGUYEN, Truong Q. *Integer fast Fourier transform*. IEEE Transactions on Signal Processing, v. 50, n. 3, p. 607-618, 2002.

RAO, Kamisetty Ramamohan; KIM, Do Nyeon; HWANG, Jae Jeong. *Fast Fourier transform-algorithms and applications*. Springer Science & Business Media, 2011.

UNDELAND, Mohan N.; ROBBINS, William P.; MOHAN, N. *Power Electronics. Converters, Applications and Design*, John Wiley & Sons, 2003.

VAIDYA, Sumit; DANDEKAR, Deepak. *Delay-power performance comparison of multipliers in vlsi circuit design*. International Journal of Computer Networks & Communications (IJCNC), v. 2, n. 4, p. 47-56, 2010.

WINOGRAD, Shmuel. *On computing the discrete Fourier transform*. Mathematics of computation, v. 32, n. 141, p. 175-199, 1978..

YEH, Wen-Chang; JEN, Chein-Wei. *High-speed and low-power split-radix FFT*. IEEE Transactions on Signal Processing, v. 51, n. 3, p. 864-874, 2003.