UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

Jorge Wichrowski Krieger de Mello

Arquitetura de Hardware para Transformada Rápida de Fourier Aplicada ao Tratamento de Sinais do Sistema Nervoso

Trabalho de conclusão apresentado como
requisito parcial para a obtenção de
grau de Engenheiro de Computação

Prof. Dr. Sergio Bampi
Orientador

Porto Alegre, dezembro de 2012

AGRADECIMENTOS

Quero agradecer a Deus que permitiu que eu chegasse bem até aqui para poder finalizar este texto. Também pela oportunidade de fazer um trabalho com boas possibilidades de continuação, que pode fazer a diferença para as pessoas.

Agradeço também a minha mãe Elaine Wichrowski, pai Jorge Krieger de Mello, irmã Bruna, minha noiva Patricia Lacerda, meus cachorros Toli e Tilo, por fazerem parte da minha família durante minha vida, por me apoiarem nos momentos difíceis e fáceis, por me darem condições de seguir em frente fazendo com isso a Vontade de Deus. Agradeço também a minha vó Roma por me ensinar o básico do aprendizado, vô Antonio, vó Helga, vô Jorge por estarem comigo em todos os momentos. Demais familiares também foram importantes para mim. Obrigado a todos.

Agradeço aos amigos de todos os lugares, Noid (Rodrigo Cecin) por sua falta de paciência, Mestre Colega Guilherme Odin por sua companhia nos incansáveis estudos da faculdade, ao Pedro bolsista do laboratório 215, mas um grande companheiro de estudos. Demais amigos importantes pela companhia, aqui são muitos, vou citar alguns, Inocêncio Lenda, Carolina Puginna, Mezzomo, Everson, Jupará, Alemas, Alegria, Raul, Cauê, Eskel e muitos outros.

Obrigado aos colegas do laboratório 215, em destaque, Kleber Stangherlin que me deu todo o suporte das ferramentas da Cadence, maior especialista da UFRGS nelas. Ao Leonardo Soares que junto com Kleber me ajudaram na pesquisa de partes dessa monografia, bem como na troca de muitas ideias. Aos demais do laboratório Cristiano Thiele, Cláudio Diniz, Eduarda, Cauane, Vizzotto, Daniel, Felipe, a outros que aparecem lá esporadicamente.

Agradeço aos professores que foram referências para mim neste curso, ao professor Sergio Bampi (Concepção I e Concepção II) que me acolheu como bolsista IC e me orientou neste trabalho, ao professor Ribas (Sistemas Digitais), Ramon (Análise de Circuitos I), Alceu Heinke (Análise de Circuitos II), Elisabeta Gallicchio (Equações Diferencias e Matemática Aplicada), André Reis (Cad para Sistemas Digitais), Dante Barone (Robótica II), Marcelo Walter (Computação Gráfica). Ao professor Susin o qual não conheço pessoalmente, mas conheço suas ideias e as considero boas.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

CI - Circuito Integrado

CMOS - *Complementary Metal–Oxide–Semiconductor*

FPGA - *Field Programmable Gate Array*

FFT - *Fast Fourier Transform*

NMOS (N-type metal-oxide-semiconductor),.

PI - Módulo de Propriedade Intelectual

PMOS *(P-type metal-oxide-semiconductor)*,

RTL - *Register Transfer Level*

UFRGS - Universidade Federal do Rio Grande do Sul

VCD - *Value Change Dump*

VHDL - *Very High Speed Integrated Circuit Hardware Description Language*

VT - Tensão de Limiar do Transistor MOS (*Threshold Voltage*)

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

O trabalho faz um estudo dos sinais nervosos provenientes da medula espinhal. Determinar as características desses sinais é importante para entender seu mecanismo de ação e a forma de processá-los. O processamento destes sinais foi considerado para se obter o seu espectro no domínio frequência, haja vista as vantagens do uso deste em relação ao domínio do tempo. A transformada para os espectros de frequência deve ser realizada com uso de algoritmos rápidos que processem as amostras do sinal. A descrição de um algoritmo de Fast Fourier Transform em VHDL (Very High Speed Integrated Circuit Hardware Description Language) foi a primeira parte prática do trabalho. A validação deste hardware, sua síntese lógica e mapeamento para portas lógicas CMOS foram os passos seguintes. Neste trabalho foram aplicadas as células de portas lógicas MOS desenvolvidas com regras de 65nm no nível de layout pelo próprio autor e que foram caracterizadas no grupo de pesquisa para operação em tensões de alimentação ultra-baixas. A característica principal do trabalho foi que o hardware CMOS da Transformada Rápida foi sintetizado para operar com baixíssimo consumo de potência, pela operação da lógica em regime near-threshold, com alimentação baixa, reduzida até a 450mV nominal. O alvo de frequência de operação utilizado na síntese foi baixo, da ordem de 1 MHz, para economizar energia. Os resultados da síntese lógica apresentados nesta monografia são discutidos em termos de compromisso tensão de alimentação e potência do circuito da transformada rápida de Fourier. O comportamento do circuito com variações de temperatura, operando a tensões baixas, também foi analisado. A operação em tensão reduzida de 0.45V de alimentação foi demonstrada e analisada ao final do trabalho.

Palavras-chave: *FFT, VHDL, Transformada Rápida de Fourier, Medula Espinhal, Lógica CMOS, Near-Threshold, Síntese Lógica, Circuito Integrado.*

ABSTRACT

The work is a study of nerve signals from the spinal cord. To determine the characteristics of these signals is important in order to understand their mechanisms of action and how to process them. The digital processing of signals was considered in order to obtain their frequency domain spectrum, given the advantages of using this domain with respect to the time domain. The conversion to the frequency spectra must be performed using fast algorithms. The description of a Fast Fourier Transform algorithm in VHDL (Very High Speed Integrated Circuit Hardware Description Language) was the first part of this practical design work. The validation of this hardware and its logic synthesis and mapping to CMOS logic gates were the following steps. In this design static MOS logic gates developed by the author at the layout level with 65nm CMOS design rules were used. They were characterized in our research group to operate at ultra-low supply voltages. The main design characteristics of this work is that the Fast Fourier Transform CMOS hardware was synthesyzed to operate at a very low power consumption level, through the near-threshold logic operation for ultra low Vdd down to 450mV. The target operating frequency for the design in low, around 1 MHz, to save switching energy. The logic synthesis results presented in this work are discussed in terms of the compromise between supply voltage, area and power consumed by the FFT CMOS circuit. The temperature behavior at low Vdd is also analyzed for the circuit. The operation down to 0.45V supplies was demonstrated and analyzed at the end of the monography.

Keywords: *FFT, VHDL, Fast Fourier Transform, Spinal Cord, Near-Threshold CMOS Logic, Logic Synthesis, Integrated Circuit.*

1 INTRODUÇÃO

Este trabalho de conclusão versa sobre a aplicação da FFT (*Fast Fourier Transform*) a sinais obtidos para fins de tratamentos médicos. O primeiro passo é o estudo do algoritmo da FFT otimizado para execução em hardware. A seguir será descrito um circuito integrado em VHDL (Very *High Speed Integrated Circuit Hardware Description Language*) dedicado ao cálculo da FFT. Será feita uma análise desse projeto para sua execução em baixa potência, visto que este é um requisito de projetos da área médica, pois instrumentos médicos que precisam ser implantados necessitam de grande autonomia de funcionamento. A área de aplicação especificamente é o tratamento de sinais da medula espinhal, para definição das características dos sinais que serão processados na arquitetura dedicada.

O circuito desenvolvido nesse estudo é parte dos módulos de PI (Módulo de Propriedade Intelectual) necessários para um sistema eletrônico para assistir na solução para resolver o problema da paralisia por rompimento da medula espinhal que acomete algumas pessoas. No contexto de um trabalho final multidisciplinar será especificado um esboço do projeto onde a FFT está inserida. Para isso será feita uma rápida apresentação da ideia.

1.1 Objetivos

Esta monografia objetiva contribuir com a linha de pesquisa voltada para circuitos CMOS digitais operando em regime de dissipação de potência ultra-baixa. O trabalho colabora com o desenvolvimento futuro de uma aplicação que vise fazer uma eletrônica para minimizar os efeitos nocivos do tipo de lesão em espinha, processando sinais oriundos de células nervosas espinhais. A parte prática da Transformada de Fourier também permitirá a geração de dados de pesquisa para auxiliar futuros trabalhos em outras áreas.

A monografia tem como objetivo de desenvolver trabalhos em hardware e software, como é a combinação mais importante na formação apresentada no curso de Engenharia de Computação.

1.2 Estrutura

Podemos dividir este trabalho em cinco partes distintas:

Primeiramente o estudo do problema em nível médico e biológico, soluções disponíveis, análise do sinal a ser amostrado. Segundo, o estudo da Transformada de Fourier e como será desenvolvido um projeto levando em

conta os objetivos a serem alcançados. Também nesta etapa foram traçados os requisitos da FFT, necessários para tratar os sinais nervosos. Na terceira etapa foram desenvolvidos os layouts das células lógicas CMOS para se poderem gerar as bibliotecas necessárias para o projeto em VHDL da transformada rápida. Numa quarta etapa foi desenvolvido o RTL (*Register Transfer Level*) do hardware da FFT. Este RTL terá então sua funcionalidade testada por simulação lógica e também pelos seus resultados quando executado em uma placa que contém um dispositivo FPGA (*Field Programmable Gate Array*). Por fim foi realizada a síntese lógica desse circuito, utilizando as bibliotecas geradas para diferentes condições de operação (temperatura e tensão, sendo esta reduzida para valores *near-threshold*) as quais foram criadas a partir das células geradas na terceira etapa. Neste trabalho foi possível comprovar a operação do módulo de PI da FFT em baixa frequência e com dissipação muito baixa de potência. A monografia documenta os passos deste trabalho e apresenta os resultados e conclusões do projeto deste módulo de PI..

No **capítulo 2** serão detalhados os resultados do estudo acerca do sistema nervoso, coluna vertebral, medula espinhal, impulso nervoso e sinapse. Será feito um aprendizado do comportamento dos sinais do sistema nervoso.

No **capítulo 3** é descrito o projeto de um módulo de PI que visa auxiliar o problema descrito no capitulo anterior. Neste contexto é feita a descrição do módulo da FFT que desenvolveremos para processamento dos sinais de origem nervosa.

No **capítulo 4** é tratada a Transformada de Fourier e o algoritmo para cálculo rápido dos coeficientes.

No **capítulo 5** feito é discutido o layout das células CMOS, as quais serviram para gerar as bibliotecas em diferentes pontos de operação de alimentação e temperatura.

No **capítulo 6** serão dados detalhes de como foi projetado o código RTL em VHDL da Transformada Rápida de Fourier.

O **capítulo 7** trata da síntese lógica, dos resultados e conclusões obtidas acerca desses resultados.

2 ESTUDO DO PROBLEMA

2.1 Sistema Nervoso

O sistema nervoso é composto de duas partes o sistema nervoso central e o sistema nervoso periférico:

O sistema nervoso central é composto pelo encéfalo (figura 2.1) e pela medula espinhal que são responsáveis por processar informações. A medula espinhal também conduz informações até o cérebro, mas ela pode agir por conta própria, isto ocorre quando em movimentos já treinados e repetitivos como caminhar, correr e também em reação a dor, ou calor.

O sistema nervoso periférico é composto por nervos, gânglios e terminações nervosas que são responsáveis pela condução das informações. Este trabalho irá focar mais no sistema nervoso central em especial a medula espinhal, mas agora falaremos um pouco do encéfalo. O encéfalo é composto por três partes distintas o Cérebro, Cerebelo e o Tallo Encefálico.

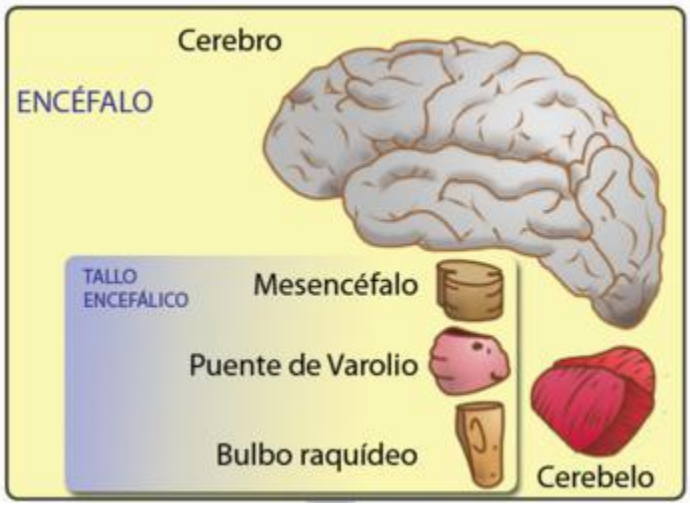


Figura 2.1: Encéfalo (http://www.cerebronosso.bio.br)

O Cerebelo é o responsável pelos movimentos, equilíbrio e aprendizagem motora.

O cérebro está dividido em quatro principais áreas o chamado Lóbulo Frontal, Lóbulo Parietal, Lóbulo Temporal e Lóbulo Occipital como mostra a figura 2.2. Deste o lóbulo Frontal foi o último a se desenvolver na escala

evolutiva e é nele que funções como pensamento e decisões a serem tomadas são realizadas. O lóbulo Parietal é o responsável por processar as sensações (dor, calor, tato). O lóbulo Occipital pela visão e o lóbulo Temporal pelas informações auditivas.

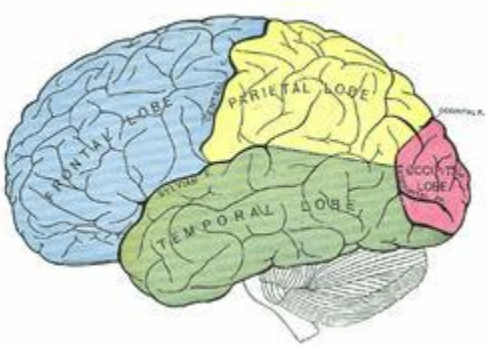


Figura 2.2: Áreas do Cérebro (http://www.cerebronosso.bio.br)

O sistema nervoso é constituído por neurônios e por células glias (grande maioria). As células glias são apenas para isolar,proteger,sustentar e nutrir os neurônios.

2.2 Neurônio

Um neurônio, também conhecido como uma célula de neurônio ou nervo é uma célula excitável eletricamente que processa e transmite informação através de sinais elétricos e químicos. Um sinal químico ocorre através de uma sinapse, uma ligação com outras células especializadas. Neurônios podem ligar-se uns aos outros para formar redes neurais. Há cerca de 86 bilhões de neurônios no sistema nervoso humano.

O neurônio é formado pelos Dendritos (terminal de recepção) e pelo Axônio (terminal de transmissão) como pode ser visto na figura 2.3.

A Bainha de Mielina é responsável pela rápida transmissão do impulso nervoso, entre uma bainha e outra fica o Nodo de Ranvier, local aonde esta localizado os canais de Sódio (Na+) e Potássio(K+) que realizam a troca destes componentes entre o meio intracelular e o meio extracelular. A destruição da bainha de mielina é responsável por severas doenças entre elas a esclerose múltipla.

Os neurônios podem ser classificados em:

Neurônios receptores ou sensitivos (aferentes): são os que recebem estímulos sensoriais e conduzem o impulso nervoso ao sistema nervoso central.

Neurônios motores ou efetuadores (eferentes): transmitem os impulsos motores (respostas ao estímulo).

Neurônios associativos ou interneurônios: estabelecem ligações entre os neurônios receptores e os neurônios motores.

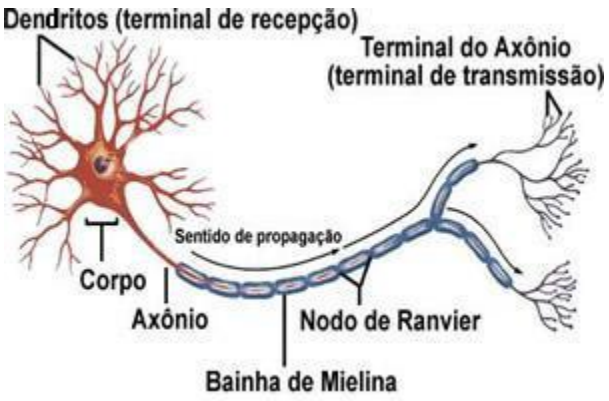


Figura 2.3: Neurônio

(http://www.infoescola.com/sistema-nervoso/neuronios/)

2.3 Impulso Nervoso

Impulso nervoso é o nome dado aos sinais que os nervos transmitem, e apresenta o mecanismo de funcionamento descrito a seguir:

Na membrana da célula nervosa há a chamada **Bomba de Sódio (Na+) e Potássio (K+)** que constantemente fica bombeando Sódio (Na+) para fora da célula. Como sai muito mais Na+ do que entra de K+ a célula nervosa em repouso se torna eletronegativa em seu interior.

Há também o **Canal de Sódio (Na)** que permanece fechado em repouso.

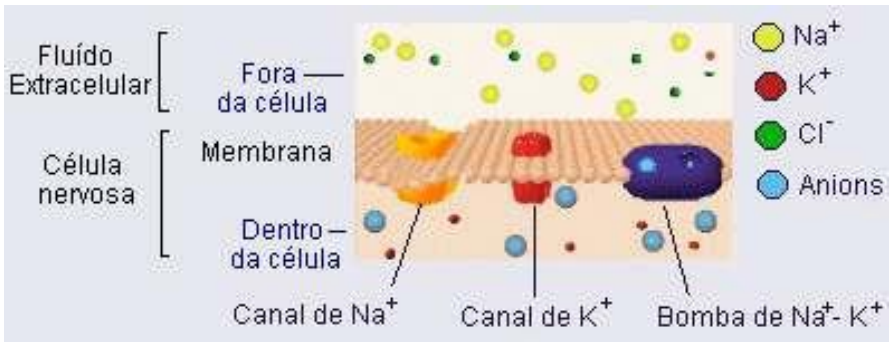


Figura 2.4: Bomba de Sódio e Potássio
(www.epub.org.br/cm/n10/fundamentos/animation.html)

O **impulso nervoso** se dá da seguinte maneira, o meio intracelular é rico em K (Potássio) e o meio extracelular é rico em Na (Sódio). O meio intracelular em estado de repouso apresenta -70mV de potencial elétrico, ao receber um pequeno estímulo o Na entra para dentro da célula a medida que um pouco de Potássio(K) sai, tornando o meio intracelular positivo aproximadamente +30mV. Em seguida o Sódio(Na) é jogado para fora de novo enquanto o Potássio(K) volta para dentro, esta inversão é transmitida ao longo do **axônio** e é chamada de **onda de polarização**, a qual esta ilustrada na figura 2.5.

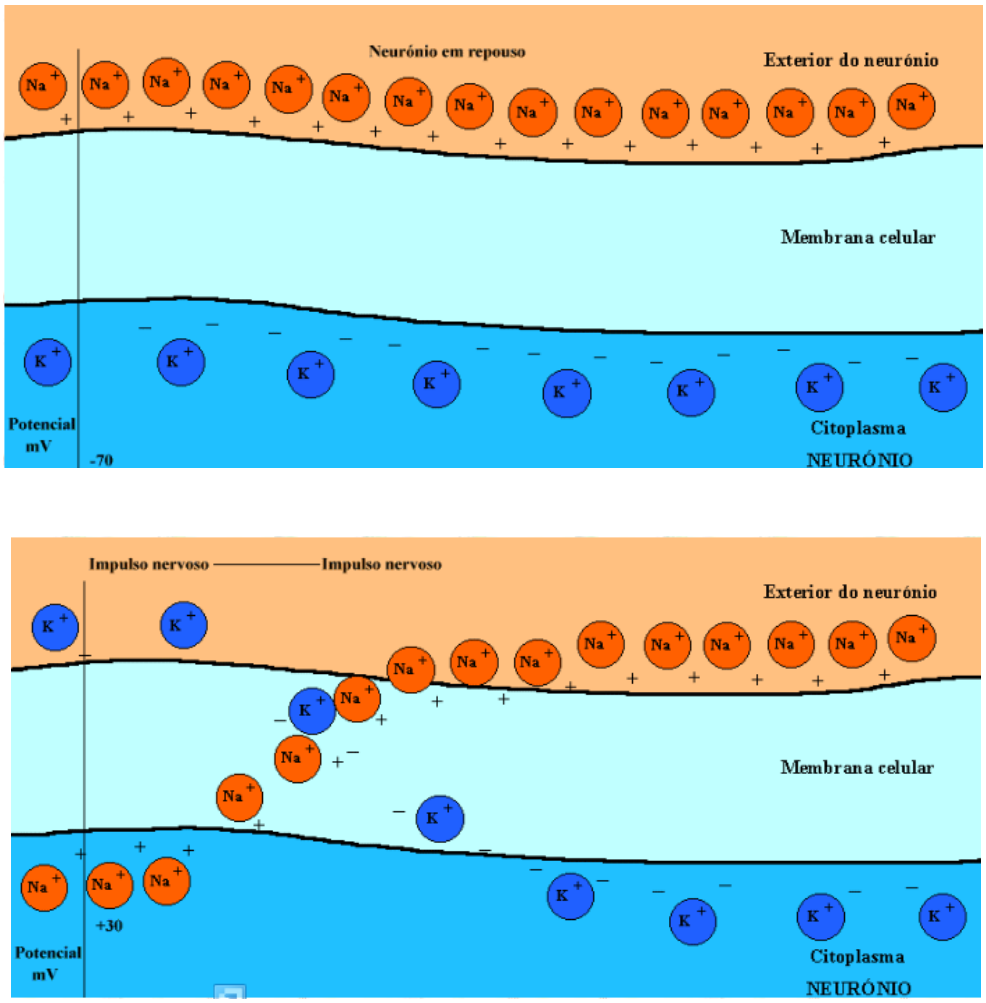


Figura 2.5: Onda de Polarização

(ANA LUISA MIRANDA VILELA. Anatomia e Fisiologia Humanas. http://www.afh.bio.br/nervoso/nervoso4.asp#SNP.)

Impulsos nervosos ocorrem em média a cada 5ms, o que significa uma frequência média de 200Hz.

2.4 Sinapse

A **sinapse química** ocorre quando não há contato físico entre os neurônios, ou seja, na grande maioria deles. Entre eles há uma fenda sináptica de 20 a 50nm.

A **energia elétrica** contida no neurônio é convertida para energia química (são liberados **neurotransmissores**) e depois para elétrica no neurônio seguinte. Esta é sinapse mais comum e que ocorre com mais frequência, sua vantagem que ela é unidirecional, mas também é mais lenta. Na medula espinhal a sinapse elétrica é predominante, podemos considerar ela como foco de estudos a serem realizados.

2.5 Medula Espinhal

A **medula espinhal** possui aproximadamente 40 cm de comprimento. Possui um sistema descendente que leva informação do cérebro aos músculos e possuí um sistema ascendente que leva informações sensoriais do corpo ao cérebro. Essa comunicação é realizada pelo **sistema periférico** que é formado pelos **31 pares de nervos raquidianos** que saem da **medula espinhal**.

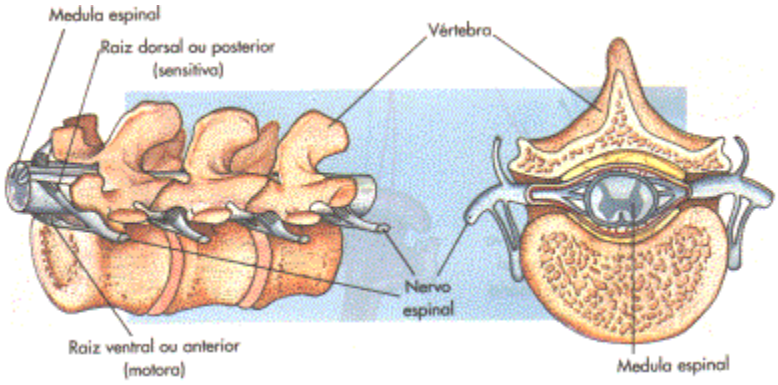


Figura 2.6: Coluna Vertebral

(http://www.afh.bio.br/nervoso/nervoso4.asp#SNP)

O corpo dos neurônios fica no centro da medula ou massa cinzenta, já o axônio fica na massa branca.

Dado interessante é que se a lesão da medula atingir apenas a parte cinzenta, poderá ficar restrita a uma determinada porção do corpo sem afetar os membros mais abaixo. Agora a lesão que ocorre na massa branca paralisa tudo que estiver abaixo dela. A lesão em si vem acompanhada de outros problemas que acabam piorando a recuperação do paciente, por isso há necessidade de tecnologia avançada e médicos capacitados para reduzir as chances de sequelas. Na figura 2.7 podemos ver um corte horizontal na medula espinhal aonde são mostradas suas partes.

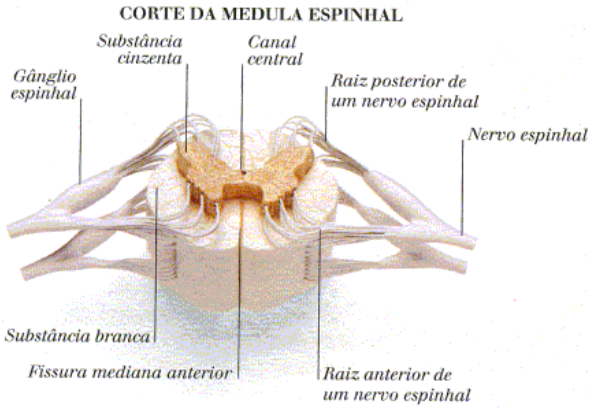


Figura 2.7:Corte Horizontal na Medula

(ANA LUISA MIRANDA VILELA. Anatomia e Fisiologia Humanas. http://www.afh.bio.br/nervoso/nervoso4.asp#SNP.)

Abaixo na figura 2.8, é mostrado os 31 pares de nervos raquidianos em uma espinha humana:

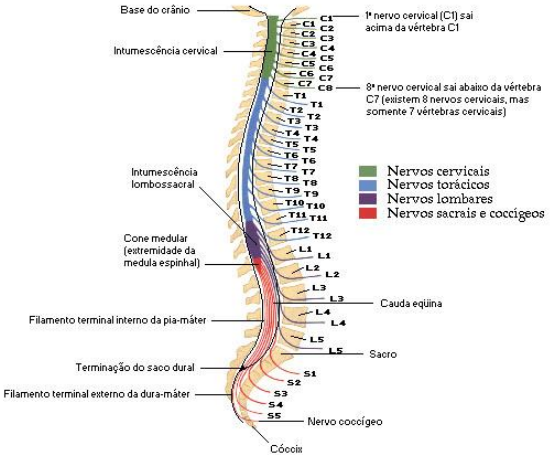


Figura 2.8: Coluna Vertebral 31 pares de Nervos Raquidianos (ANA LUISA MIRANDA)

2.6 Trabalhos Nesta Área

Há hoje muitos estudos nesta área para se tentar curar pacientes com paralisia. Existem abordagens totalmente voltadas para uso de estruturas biológicas como células tronco. Há abordagens puramente da área de engenharia, neste caso é necessário que o paciente possua alguma mobilidade para controlar a estrutura que o fará caminhar. Militares dos Estados Unidos da América têm desenvolvido estes equipamentos, mas com objetivo de se aumentar a força dos combatentes para transportar cargas que vão muito além do seu peso. No entanto as abordagens que atualmente tem mais resultado são modelos híbridos, que combinam uso de técnicas biológicas e mecânicas. Dois casos muito interessantes sobre o assunto são os trabalhos do Professor Courtine (COURTINE Grégoire. http://courtine-lab.epfl.ch/team) e o do Professor Nicolelis (NICOLELIS Miguel. Mind Out of Body. 2011.).

O trabalho desenvolvido pelo Professor Courtine no projeto NEUWalk combina o uso de soluções químicas semelhantes a de uma família de neurotransmissores que inclui a norepinefrina, a serotonina e a dopamina. Estas substâncias químicas substituem os neurotransmissores normalmente liberados pelos processos neuronais de indivíduos saudáveis. Após esta aplicação os indivíduos estão prontos para serem estimulados eletricamente com eletrodos implantados na superfície do canal espinhal, conhecida como espaço epidural. Assim um movimento involuntário de andar realizado pela própria medula espinhal é produzido. Após diversas seções de treinamento os pesquisadores notaram que o número de fibras nervosas no local havia quadruplicado. Foi possível concluir que a técnica utilizada aumenta a neuroplasticidade da medula espinhal. A neuroplasticidade é a capacidade das células nervosas se recuperarem após sofrerem algum tipo de lesão, capacidade que sabidamente é baixa para a medula espinhal. Desta maneira o projeto mostrou resultados muito bons, neste momento os pesquisadores estão se preparando para iniciar os testes em seres-

humanos.

Outro trabalho com destaque nesta área é realizado pelo pesquisador brasileiro o Professor Nicolelis. Sua ideia de solução é mais mecânica, mas possui uma parte biológica para controle do dispositivo. O projeto em questão é de um exoesqueleto, uma espécie de armadura que a pessoa veste para poder se locomover. Este exoesqueleto realiza os movimentos pelo individuo, mas quem controla é ele mesmo. Este controle é feito a partir do processamento de sinais adquiridos de regiões específicas do cérebro. Neste caso esses sinais precisam ser coletados e processados. Este projeto está em fase de desenvolvimento tendo uma expectativa para se ter resultados satisfatórios em 2 anos.

3 PROJETO PARA MODELO DE SINAIS DE IMPULSOS NERVOSOS

Muitas abordagens podem ser usadas quando queremos solucionar um determinado problema. Neste caso queremos que músculos que não mais recebem sinais de comandos passem a receber. Para isso é possível imaginar a seguinte solução, transferir o sinal elétrico imediatamente antes da lesão para região imediatamente após a lesão, tentando assim, recuperar a maior parte possível de movimentos. Como falado no capítulo anterior esses sinais apresentam baixa intensidade, fica claro a necessidade de amplifica-los. Nesta ação são utilizados amplificadores, dispositivos ativos formados por transistores. Esses dispositivos interferem eletricamente nos sensores, logo é necessária a filtragem dessa interferência antes da propagação do sinal. Essa operação é na maioria dos casos feita com mais eficiência no domínio frequência. O domínio frequência possui um espectro com a intensidade das diversas frequências presentes naquele sinal e pode ser alcançado pela aplicação da Transformada de Fourier.

3.1 Transformada de Fourier e o Modelo de Hodgkin-Huxley

Para converter o sinal do domínio tempo para o domínio frequência será utilizada a Transformada de Fourier que pode ser aplicada em tempo real devido ao uso de métodos rápidos para obtenção do espectro, para definir os requisitos da Transformada de Fourier Rápida é necessária uma análise do sinal. O nosso sinal nervoso apresenta pulsos que duram um intervalo entre 2ms e 7ms com média de 4,5ms. Esta característica esta de acordo com os tempos citados na literatura [3], 1,5ms para alcançar o pico de voltagem e 3ms para voltar ao valor do estado de repouso. O inverso deste período é aproximadamente 220Hz. Na figura 3.1 é mostrado o sinal de um modelo aproximado (modelo de **Hodgkin-Huxley**) para as ações de potenciais elétricos em neurônios.

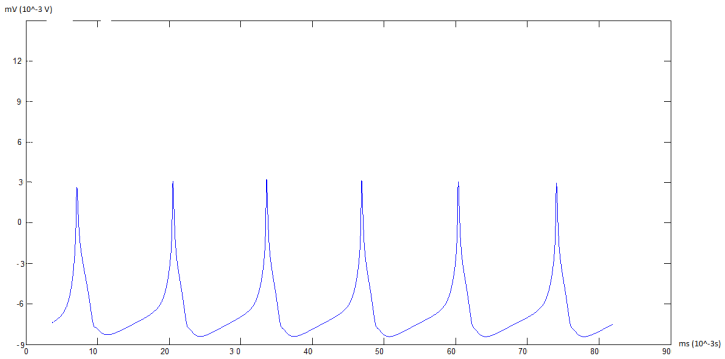


Figura 3.1: Forma de onda dos impulsos nervosos.

Se usarmos a definição do teorema de **Nyquist**, que diz que para um sinal ser posteriormente reconstruído com o mínimo de perda de informação devemos amostrá-lo com o dobro da frequência máxima que predomina no seu espectro, chega-se à conclusão de que necessitamos amostrar este sinal com uma taxa de aproximadamente 440Hz, o qual será arredondado para cima para 500Hz, de modo a incluir uma margem de segurança na função de amostragem do sinal.

Na figura 3.2 é possível ver esse efeito analisando a magnitude das componentes espectrais (no domínio das frequências) do modelo de **Hodgkin-Huxley** para descrever ações de potenciais em neurônios. O espectro de módulo da amplitude mostrado na figura 3.2 foi calculado com o software **MatLab**.

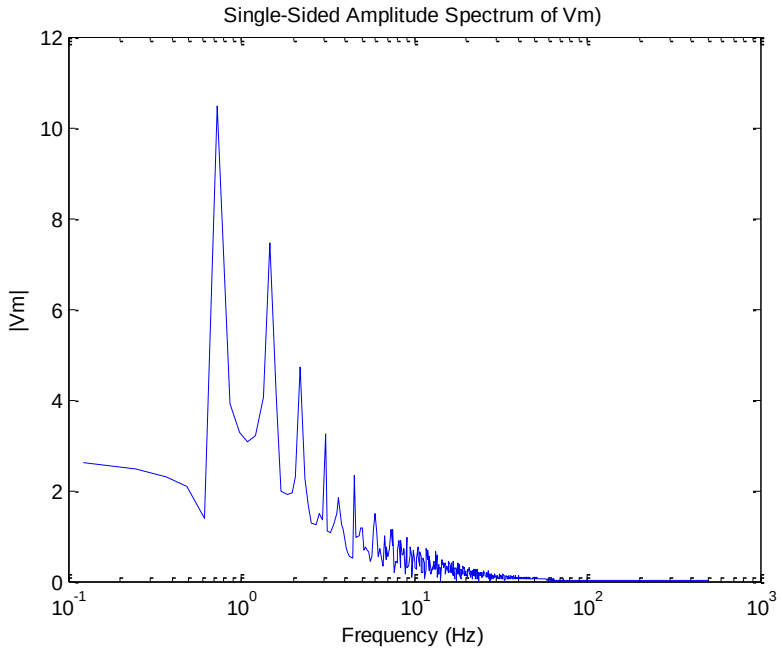


Figura 3.2: Efeito da transformada sobre uma janela de 8192 pontos

3.2 Operando Circuitos CMOS a *Near-Threshold*

Pode-se observar que estamos trabalhando com frequências bem baixas para o padrão dos circuitos atuais. Seria possível concluir por mera dedução lógica que não haverá problemas para atender os requisitos de desempenho que estes circuitos provavelmente conseguirão operar a baixas potências. Pois quanto mais baixa a frequência do relógio (*clock)*, menor é a potência dinâmica dissipada na carga/descarga de um capacitor, conforme:

$$P = C\, f\, V^2$$

A frequência é baixa, logo a potência dissipada será baixa. Como o

requisito de frequência é baixo, podemos operar com tensões menores o que nos permite reduzir ainda mais a potência. Nesse trabalho a principal tarefa no projeto da FFT em hardware é operar a tensões near-threshold (próximo ao valor da tensão limiar de condução de um transistor).

Esta técnica promete aumentar significativamente a eficiência energética, diminuindo a tensão de alimentação. Microprocessadores completos podem consumir tão pouco quanto 2mW. Para se ter uma ideia, por exemplo, isso pode garantir que eles funcionem apenas com energia fornecida por um painel solar.

Atualmente a escala de transistores tem sido realmente um fator preponderante que segue Lei de Moore sobre a densidade de escala do transistor em relação aos avanços na tecnologia de processo de semicondutores. Esta tem contribuído com maior desempenho e menor consumo de energia, mas cabe dizer que a principal variável para a diminuição do consumo de energia foi a redução da tensão de operação. A potência dinâmica de um chip é proporcional ao quadrado da tensão, de modo que apenas 10% de redução da tensão traduzem-se em 19% a menos de corrente ativa.

Por quase 30 anos, a tensão de alimentação foi reduzida, chegando a 1V por volta de 2005. Nesse ponto, a tensão nominal de operação de CIs (Circuitos Integrados) CMOS parou de cair em função dos limites impostos pelo material de silício. Transistores modernos usam uma variedade de materiais, para além de silício, mas se a tensão de alimentação é muito abaixo 1V, os transistores apresentam fraco desempenho, e com isso, aumenta a vulnerabilidade a erros.

A tensão diminuiu modestamente desde 2005 e está em torno de 0.8-0.9V, com alguns chips móveis que operam em modo de baixa dissipação a tensões em torno de 0.65V. Isto foi conseguido através de técnicas cuidadosas de design que abordam os muitos problemas da operação de baixa tensão. Os chips modernos incorporam bilhões de transistores, e efeitos puramente estatísticos ditam variações sobre alguns que são mais lentos ou mais rápidos do que outros. Os fabricantes de semicondutores devem produzir grandes volumes de chips que funcionam corretamente ao longo de muitos anos em uma grande variedade de condições. Condições de funcionamento dinâmicas são um problema bem conhecido e tornar-se pior em baixas tensões. Grandes picos ou quedas na corrente podem facilmente causar erros, de natureza semelhante aos que atingem as redes de energia sobrecarregadas. Além disso, poucos *flips* (viradas de bits) causados por partículas alfa podem corromper os dados armazenados em circuitos de memória e matrizes de grandes dimensões.

O ponto onde um transistor começa tornar-se ligado para começar a conduzir uma pequena corrente é descrita como a tensão de limiar de condução, que é de cerca de 0.2-0.3V em tecnologias de processos modernos, embora transistores não alcancem a saturação total atual até cerca de 0,8V em inversão forte, e saturam quando operam na região sublimiar a apenas 1000 mV à temperatura ambiente. Nossa pesquisa assim como outras pesquisas da área é tentar mostrar que a tensão de alimentação mais eficiente de energia é apenas um pouco maior do que o limiar. Ter uma tensão de alimentação que é

substancialmente maior do que o limiar é uma conveniência, mas causa um aumento do custo da energia. O grande desafio é a variabilidade e garantir a funcionalidade correta a altos rendimentos (*yield*) na fabricação dos chips em grandes quantidades.

O objetivo é permitir que as tensões de alimentação extremamente baixas sejam utilizadas de modo que circuitos permaneçam extremamente robustos, tolerante a falhas e resistentes contra os erros. Enquanto isso soa bastante desafiador, é eminentemente factível. Os chips modernos são em grande parte compostos de lógica digital e memória, juntamente com porções analógicas e de I/O. Em um nível alto, CPUs e GPUs lembram grandes ilhas de memória para armazenamento de dados (*caches*) e lógica de computação (núcleos). Mesmo lógica como uma unidade de ponto flutuante contém elementos de memória, mas a lógica é amplamente dominada pela lógica combinatória e de fiação, em vez de elementos de armazenamento de dados.

Geralmente, a memória é muito mais difícil para escalar tensão do que a lógica. Em tensões operacionais mais baixas, estruturas como células SRAM são menos estáveis e tornam-se muito vulneráveis a erros de leitura ou gravação de dados. Em contraste, o impacto de baixa tensão na lógica tende a ser um aumento de atraso, o que reduz a frequência. Embora menor desempenho seja indesejável, é também muito mais tolerável do que a corrupção de dados.

3.2.1 Nichos de mercado para Aplicações Near-Threshold

É preciso mostrar que near-threshold também pode ter aplicações em ambientes de mercado:

3.2.1.1 - Servidores

O crescimento exponencial da Internet tem produzido um aumento dramático na demanda por computadores estilo servidor em bases instaladas de servidores que devem exceder 40 milhões. O crescimento do servidor é acompanhado por um crescimento igualmente rápido na demanda de energia para potência deles. O centro de dados que serve páginas web fornece uma oportunidade perfeita para o near-threshold. A carga de trabalho é um fluxo de pedidos independentes para renderizar páginas web que podem ser naturalmente executados em paralelo. A página HTML é buscada a partir da memória, submetidas a operações relativamente simples, e retornada para a memória sem exigir extensos dados compartilhados. Para alcançar este objetivo, núcleos operando a near-threshold podem ser utilizados para se obter o rendimento muito elevado com a eficiência energética sem precedentes.

3.2.1.2 – Em Computadores Pessoais

Os futuros dispositivos devem combinar um elevado nível de capacidade de processamento, juntamente com as capacidades de processamento de sinal integrados em um fator de forma muito menor do que os laptops de hoje. Hoje se tem a expectativa que a vida da bateria dure dias, enquanto que os requisitos de funcionalidade sejam ao extremo. No espaço da plataforma de computadores pessoais, núcleos podem executar a grande variação de desempenho. A tensão e a frequência dos núcleos e seus

periféricos de apoio podem ser dinamicamente alterada em tempo real para responder aos constrangimentos de desempenho e de consumo de energia. Esta tensão dinâmica pode ser aproveitada para permitir adaptação aos circuitos near-threshold, no espaço de computação pessoal. O método de escalonamento pode ser conduzido por comandos do sistema operacional e/ou sensores distribuídos. Explorando as variações de fase nas cargas de trabalho, técnicas de detecção de fase eficientes precisam ser estabelecidas para processadores multicores de vários segmentos para permitir que os sistemas de gestão de energia atuem sem comprometer significativamente o desempenho.

3.2.1.3 – Em Redes de Sensores

Com os avanços em circuitos e projetos de sensores, difusos baseados em sensores de sistemas, o incremento para milhares de nós, esta rapidamente se tornando uma possibilidade. Um nó único de um sensor consiste tipicamente de uma unidade de processamento de dados e de armazenamento, e fora do chip de comunicação, elementos dos sensores, e uma fonte de energia. Eles se comunicam muitas vezes por rede sem fio e tem aplicações potenciais em uma ampla gama de domínios industriais, de automação predial para implantes de segurança patrimonial. A versatilidade de um sensor está diretamente relacionada com o seu tempo de vida, e a possibilidade de recarregar a sua energia através das próprias ondas eletromagnéticas usadas na comunicação, conseguindo com isso, em tese, funcionamento sem fim, ou pelo menos até o fim de sua vida útil. Isso só pode ser uma realidade com um mínimo gasto energético.

3.3 Apresentação Geral da Aplicação ao monitoramento do sistema nervoso medular

Por fim, para apresentar melhor este projeto foi desenvolvido pelo autor um vídeo que se encontra disponível em *www.youtube.com/watch?v=ZDleVX4qQWc* . Esse vídeo traz uma ideia melhor de como o projeto está inserido em seu contexto. Nele é mostrado uma proposta de amostragem de sinais em medulas, que pode se enquadrar em aplicações futuras para este trabalho.

4 TRANSFORMADA DE FOURIER

4.1 Transformada de Fourier

Esta parte do trabalho tem como objetivo entender e explicar o funcionamento de um módulo bastante usado atualmente para processamento de sinais: a Transformada de Fourier.

A Transformada de Fourier tem como objetivo tomar um sinal com representação no domínio tempo e representá-lo no domínio frequência. Isto é feito separando todas as frequências que aparecem no sinal no domínio tempo e medindo suas intensidades. Após isso tem-se um espectro que contém pulsos maiores nas frequências mais representativas no domínio tempo.

Esse tipo de transformação é muito útil, pois torna mais rápidas algumas operações matemáticas como a convolução (passagem de um sinal pelo outro). Este tipo de operação fica extremamente rápida, pois o que seria uma soma de produtos se torna apenas uma multiplicação. Isto nos permite obter a resposta a um determinado impulso sobre o sinal de maneira eficaz. Também é facilitada a operação de filtragem do sinal, ou seja, obter apenas faixas de valores de interesse.

Existem quatro tipos de transformações de Fourier distintas. Em todas, o princípio matemático é o mesmo, o que muda é o tipo original do sinal e com isso difere o modo como realizamos as transformadas, elas são: a Série de Fourier de Tempo Contínuo, a Série de Fourier de Tempo Discreto ou Transformada de Fourier Discreta (FTD), a Transformada de Fourier de Tempo Contínuo, a Transformada de Fourier de Tempo Discreto. Este trabalho tratará da Transformada de Fourier Discreta (FTD), que é provavelmente a mais usada atualmente, pois ela trata de sinais periódicos e discretos, ou seja, para computadores que lidam apenas com sinais digitais o fato do sinal ser discreto é um pré-requisito. Como a maioria dos sinais do nosso meio são contínuos, o que se faz é amostrar e converter para o domínio digital o valor de cada amostra para que o computador possa aplicar a transformada.

Tendo em vista a grande utilização da Transformada de Fourier Discreta foram desenvolvidos métodos para ela que reduzem o tempo de cálculo em diversas vezes. Estes métodos tem basicamente o mesmo princípio reorganizar os dados amostrados de modo a eliminar as redundâncias de cálculos matemáticos. Quando utilizamos esses algoritmos falamos que estamos realizando a Transformada Rápida de Fourier.

Vamos analisar os sinais para os quais estamos focando este trabalho. Para tanto é necessário compreender as propriedades básicas dos pares de

Fourier e neste contexto entender como um sinal no domínio tempo será representado no domínio frequência.

Domínio Tempo	Domínio Frequência
Contínuo	Não Periódico
Discreto	Periódico
Periódico	Discreto
Não Periódico	Contínuo

Tabela 4.1: Tipos de representação em Diferentes Domínios

Vamos agora analisar o nosso sinal para decidir qual a melhor representação em Fourier para ele. Estamos falando de um sinal analógico com frequência máxima de 250Hz que deverá ser amostrado com uma frequência de corte de pelo menos 500Hz, como resultado da amostragem temos um sinal discreto. Este sinal será assumido como periódico durante a realização de um atividade repetitiva, como andar e correr, por exemplo. Então nossa representação escolhida como já mencionada anteriormente é FTD (Transformada de Fourier Discreta).

4.2 A FFT - *Fast Fourier Transform*

Quando falamos da Transformada Rápida de Fourier estamos falando de algoritmos para o cálculo da transformada que reduz sua complexidade, reorganizando as entradas de forma a evitar a realização de cálculos redundantes.

Esta reorganização só é possível fixando um N (tamanho de amostras) de modo a poder separar os cálculos em blocos que podem ser subdivididos em blocos de mesmo tamanho até se ter blocos do tamanho mínimo, por isso normalmente este tamanho é base de 2 (2,4,8,16,32,64...).

Existem diversos algoritmos com diversas vantagens. Neste trabalho vamos tratar do mais conhecido o Radix-2 ou algoritmo de Cooley-Tukey. O algoritmo dividi as amostras de índices pares e impares. O resultado é novamente dividido de novo em dois grupos. Isto é feito até que se restem apenas conjuntos de dois elementos.

O N deve ser base 2, isto por motivos já explicados de que temos que subdividir as amostras em partes iguais até blocos de tamanho mínimo que contém as amostras aos pares. O aumento do desempenho em relação ao método original se torna mais significativo a medida que o N aumenta.

Transformada de Fourier = N^2

Transformada Rápida de Fourier = $N \log_2(N)$

Veremos como é deduzido o algoritmo da FFT partindo da fórmula para o cálculo da Transformada de Fourier. (Hankin. 2001) Primeiramente renomeamos o termo exponencial.

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{\frac{-j2\pi nk}{N}} = \sum_{n=0}^{N-1} x(n)\, W_N^{nk}$$

Dividiremos a Transformada em uma parte dos coeficientes pares e outra dos coeficientes ímpares.

$$X(k) = \sum_{n=0}^{N/2-1} x(2n) W_N^{2nk} + \sum_{n=0}^{N/2-1} x(2n+1) W_N^{(2n+1)k}$$

$$X(k) = \sum_{n=0}^{N/2-1} x(2n) W_N^{2nk} + W_N^k \sum_{n=0}^{N/2-1} x(2n+1) W_N^{2nk}$$

Termo W sofre uma pequena manipulação:

$$W_N^2 = \left(e^{\frac{-j2\pi}{N}} \right)^2 = \left(e^{\frac{-j2\pi \times 2}{N}} \right) = e^{\frac{-j2\pi}{N/2}} = W_{\frac{N}{2}}$$

Reescrevemos a equação X(k) de forma completa:

$$X(k) = \sum_{n=0}^{N/2-1} x(2n) W_{\frac{N}{2}}^{nk} + W_N^k \sum_{n=0}^{N/2-1} x(2n+1) W_{\frac{N}{2}}^{nk}, \quad \text{for } k=0,1,2,\ldots,N-1$$

Esta última pode ser dividida em duas partes que formam a borboleta da FFT - Radix 2.

$$X(k+\frac{N}{2}) = \sum_{n=0}^{N/2-1} x(2n) W_{\frac{N}{2}}^{n\left(k+\frac{N}{2}\right)} + W_N^k \sum_{n=0}^{N/2-1} x(2n+1) W_{\frac{N}{2}}^{n\left(k+\frac{N}{2}\right)},$$

$$X(k+\frac{N}{2}) = \sum_{n=0}^{N/2-1} x(2n) W_{\frac{N}{2}}^{nk} - W_N^k \sum_{n=0}^{N/2-1} x(2n+1) W_{\frac{N}{2}}^{nk}$$

$$P' = P + W_N^k Q$$

$$Q' = P - W_N^k Q$$

4.3 A Transformada Rápida de Fourier de 4 pontos

Considere-se a FFT de 4 pontos, e a equação da transformada de Fourier de tempo discreto:

$$X[k] = \sum_{n=0}^{N-1} x[n].e^{-\frac{j2\pi kn}{N}}$$

Supondo-se 4 pontos de amostra, então n=0 até n=3:

$$X(0) = \sum_{n=0}^{3} x[n].e^{-\frac{j2\pi 0n}{4}} = x(0) + x(1) + x(2) + x(3)$$

$$X(1) = \sum_{n=0}^{3} x[n].e^{-\frac{j2\pi 1n}{4}} = x[0] + x[1] * e^{-\frac{j2\pi 1}{4}} + x[2] * e^{-\frac{j2\pi 2}{4}} + x[3] * e^{-\frac{j2\pi 3}{4}}$$

$$X(2) = \sum_{n=0}^{3} x[n].e^{-\frac{j2\pi 2n}{4}} = x[0] + x[1] * e^{-\frac{j2\pi 2}{4}} + x[2] * e^{-\frac{j2\pi 4}{4}} + x[3] * e^{-\frac{j2\pi 6}{4}}$$

$$X(3) = \sum_{n=0}^{3} x[n].e^{-\frac{j2\pi 3n}{4}} = x[0] + x[1] * e^{-\frac{j2\pi 3}{4}} + x[2] * e^{-\frac{j2\pi 6}{4}} + x[3] * e^{-\frac{j2\pi 9}{4}}$$

Os resultados acima serão reescritos na forma matricial:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & e^{-\frac{j2\pi 1}{4}} & e^{-\frac{j2\pi 2}{4}} & e^{-\frac{j2\pi 3}{4}} \\ 1 & e^{-\frac{j2\pi 2}{4}} & e^{-\frac{j2\pi 4}{4}} & e^{-\frac{j2\pi 6}{4}} \\ 1 & e^{-\frac{j2\pi 3}{4}} & e^{-\frac{j2\pi 6}{4}} & e^{-\frac{j2\pi 9}{4}} \end{bmatrix} * \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ x[3] \end{bmatrix}$$

Simplificando os valores da matriz de coeficiente complexos, obtém-se:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} * \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ x[3] \end{bmatrix}$$

Partindo dessa Matriz deduziremos um dos algoritmos da Transformada Rápida de Fourier e que será utilizado neste trabalho, o algoritmo de Cooley-Tukey também chamado de Radix-2 ou Decimação no Tempo. Neste algoritmo a Transformada de Fourier de N pontos é sucessivamente dividida em duas

partes de N/2 pontos.

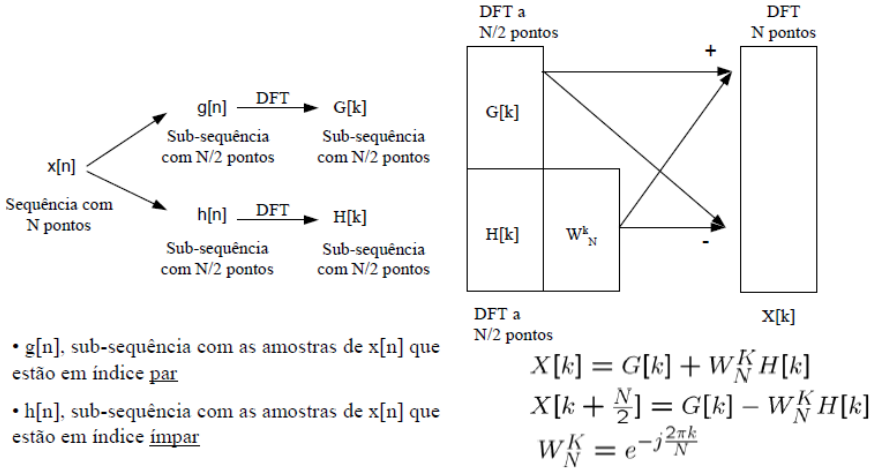


Figura 4.1: Radix-2 (STD. Inverno 2007)

Esta divisão é possível pois separamos a parte par da parte impar isso pode ser visto através do *bit-reversal que* separa a sequência nas duas subsequências com as amostras de índice par e ímpar, designadas por g[n] e h[n], respectivamente.

Domínio Tempo	Domínio Frequência
00	00
01	10
10	01
11	11

Tabela 4.2: Bit-Reversal 4 pontos

Resolveremos a multiplicação de matrizes mostrada anteriormente:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} * \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} a+b+c+d \\ a-c-j(b-d) \\ a+c-(b+d) \\ a-c+j(b-d) \end{bmatrix}$$

Ao aplicar a FFT temos o mesmo que está representado na matriz acima:

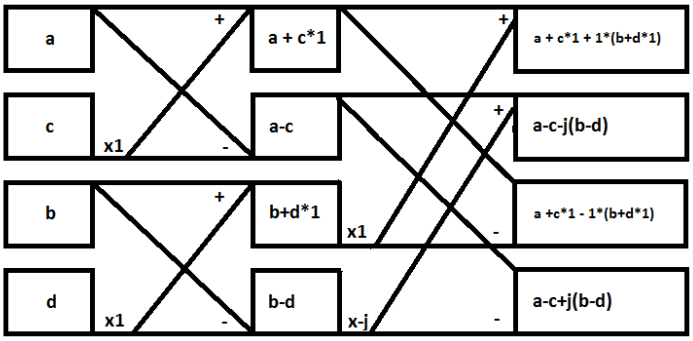


Figura 4.2: Algoritmo FFT 4 pontos

Pode-se observar que, ao invés da aplicação direta da transformada de Fourier que demanda 16 operações, a FFT reorganiza os cálculos de forma que se tenha apenas 8 operações. Conforme já foi falado acima esse desempenho aumenta com o aumento do N:

N	N^2 (DFT)	$N.\log_2 N$ (FFT)	Vantagem
2	4	2	2
4	16	8	2
8	64	24	2,67
16	256	64	4
32	1024	160	6,4
64	4096	384	10,67
128	16384	896	18,29
256	65536	2048	32
512	262144	4068	56,89
1024	1048576	10240	102,4
2048	4194304	22528	186,18
4096	16777216	49512	341,33
8192	671088964	106496	630,15

Tabela 4.3: Complexidade FFT (UFPE. 2010.)

Vamos agora ver um exemplo simples com números para que não restem dúvidas de como é realizado o algoritmo da Transformada Rápida de Fourier. Este teste considera uma FFT de apenas 4 pontos, logo sua aplicação é bastante simplificada:

$$x[n] = \{3, 1, 2, -1\}$$

$$g[n] = \{3, 2\}$$

$$h[n] = \{1, -1\}$$

$$X[k] = \{5, 1 - j2, 5, 1 + j2\}$$

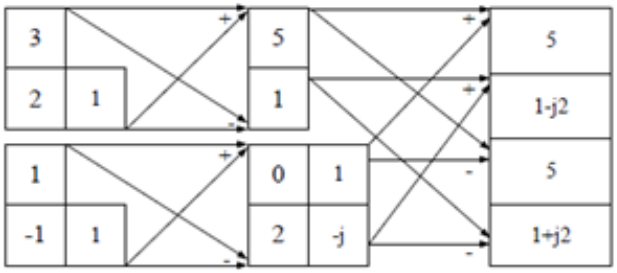


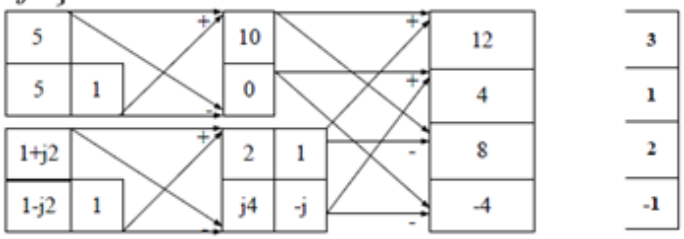
Figura 4.3: Exemplo FFT (STD. Inverno 2007)

A transformada de Fourier inversa pode ser obtida da seguinte maneira:

$$X^*[k] = \{5, 1 + j2, 5, 1 - j2\}$$

$$G[k] = \{5, 5\}$$

$$H[k] = \{1 + j2, 1 - j2\}$$



$$x[n] = \{3, 1, 2, -1\}$$

Figura 4.4: Exemplo FFT Inversa (STD. Inverno 2007)

4.4 A Transformada Rápida de Fourier de 8 pontos

A Transformada Rápida de Fourier de 8 pontos será estudada à priori e descrita neste trabalho, nossa janela tem 8 amostras do nosso sinal, cada amostra possui tamanho de 8 bits.

Abaixo segue um diagrama do algoritmo da FFT de 8 pontos, não iremos deduzi-lo, pois já fizemos para a FFT de 4 pontos, mas cabe aqui resaltar que o procedimento é o mesmo, usa-se o bit-reversal para determinar a organização dos cálculos, usa-se o calculo cruzado “borboleta”, o que muda é que temos um estágio a mais e o fator *Twiddle* difere para o último estágio.

Domínio Tempo	Domínio Frequência
000	000
001	100
010	010
011	110
100	001

101	101
110	011
111	111

Tabela 4.4: *Bit-Reversal* 8 pontos

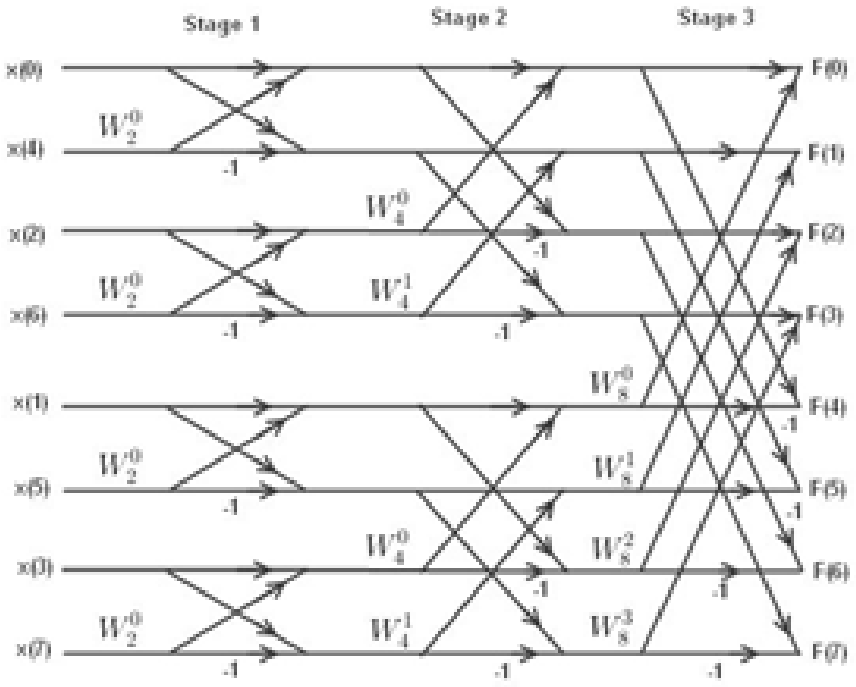


Figura 4.5: Algoritmo FFT 8 pontos (ALWAYSLEARN. – A DFT and FFT Tutorial. http://www.alwayslearn.com.)

O fator *twiddle* para o algoritmo da Transformada Rápida de Fourier, é qualquer um dos coeficientes trigonométricas constantes que são multiplicados pelos dados no decurso do algoritmo.

Mais especificamente, o fator *twiddle* se refere à raiz da unidade de constantes multiplicativas complexas nas operações borboleta do algoritmo FFT *Cooley-Tukey*, usado para combinar recursivamente a menor Transformada de Fourier Discreta possível. Embora este seja o significado mais comum e utilizado neste trabalho, o termo também pode ser usado para qualquer multiplicativo constante de dados independente num FFT.

Cada estágio da FFT possuí o seu fator *twiddle*, o primeiro para N igual a 2 é 1 e -1, por isso muitas vezes aparece omitido como constante, para N igual 4 já surgem os números complexos, a partir de N igual a 8 começam a aparecer também os números em ponto flutuante.

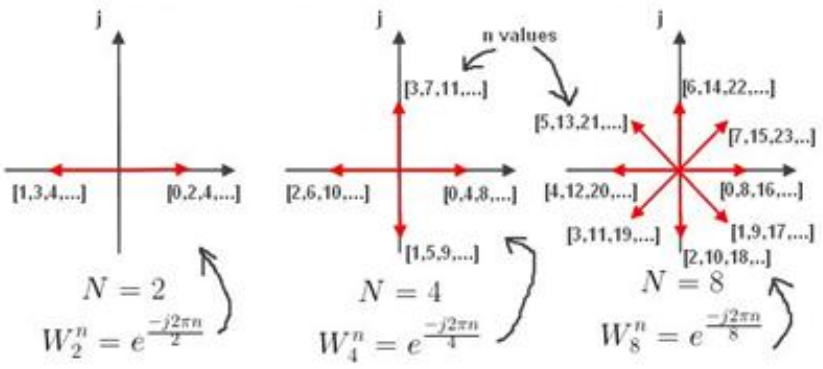


Figura 4.6: Fator *Twiddle* para diferentes N (ALWAYSLEARN. – A DFT and FFT Tutorial. http://www.alwayslearn.com.)

5 CÉLULAS LÓGICAS BÁSICAS PARA OPERAÇÃO EM *NEAR-THRESHOLD*

Como parte importante deste trabalho foram desenvolvidos layouts de células lógicas necessárias para gerar um arquivo *SPICE* com as capacitâncias parasitas. Este arquivo é utilizado pela ferramenta da *Cadence*™, o *ELC*™, para gerar bibliotecas em diferentes condições de operação. Essas bibliotecas possuem uma descrição dos parâmetros de funcionamento das instâncias para que a ferramenta RTL-Compiler™ realize a síntese lógica. O *RTL-Compiler*™ é o responsável pela síntese lógica do hardware digital, que será simulado posteriormente e para o qual se faz a avaliação de atrasos e de potência dissipada. Todos resultados desse trabalho foram feitos utilizando apenas o conjunto mínimo de células. Isto impede de se ter uma variabilidade de células para que a ferramenta faça otimizações requeridas.

Os layouts essenciais e de tamanho mínimo para se gerar as bibliotecas com várias tensões são: *NAND*, *AND*, *NOR*, *OR*, Registrador/Reset , Inversor.

A tecnologia CMOS utilizada na biblioteca de células desenvolvidas é 65nm da IBM e o ambiente de desenvolvimento das células digitais é o *Virtuoso*™.

5.1 Inversor

Podemos ver na figura 5.1 o negativo da imagem do inversor feito no Virtuoso™. O transistor *PMOS (P-type metal-oxide-semiconductor)*, que quando o inversor esta em nível lógico 0 conduz corrente de *VDD* para a saída, possui 1,5 vezes o tamanho do transistor *NMOS (N-type metal-oxide-semiconductor)*. Isto é feito, pois a resistência equivalente do transistor P é aproximadamente 1,5 vezes maior que do transistor N, então para que o tempo de resposta seja o mesmo do transistor N é feito aumento do tamanho do transistor P. Com isso o tempo de reposta da célula se torna similar em transições de subida e descida.

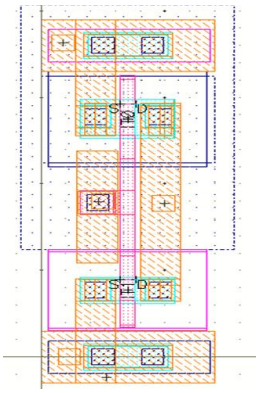


Figura 5.1: Inversor

5.2 *And* 2X1

Tanto para a porta *AND* da figura 5.2 e a *NAND* da figura 5.3 foi utilizada a largura de canal de 1,5 vezes para o transistor P em relação ao transistor N.

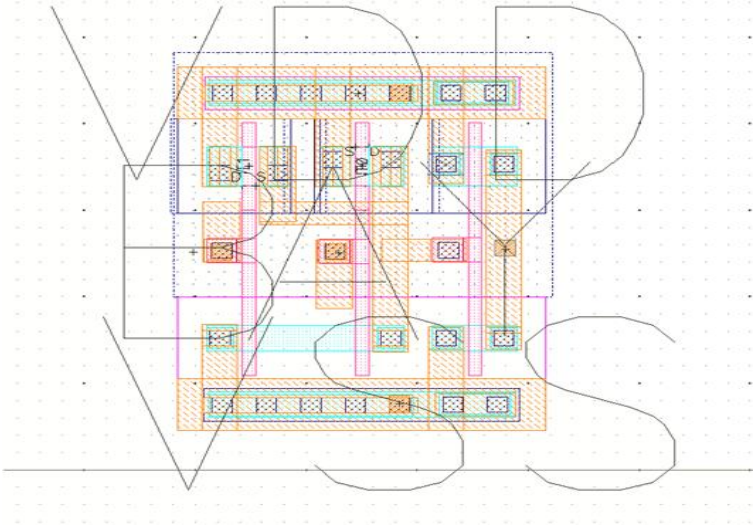


Figura 5.2: *AND* 2X1

5.3 *Nand* 2X1

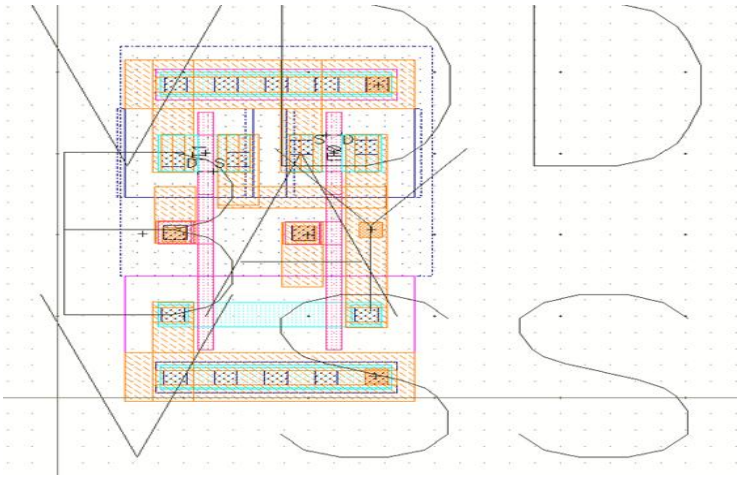


Figura 5.3: *NAND* 2X1

5.4 *Or* 2X1

Tanto para porta *OR* 2X1(figura 5.4) quanto para a porta *NOR* 2x1 (figura 5.5) o tamanho dos transistores foi redefinido em função de uma característica de pior caso aonde a soma da resistência equivalente dos resistores PMOS em série representam até 3 vezes a resistência de um transistor NMOS. Com base neste fato o tamanho do transistor PMOS passou a ser 3 vezes o tamanho o transistor NMOS. Como este tamanho extrapolaria a altura pré-definida para estas células de 1.8micra foi utilizado a técnica de *"folding".* O *folding* permite o aumento do tamanho do transistor duplicando a sua região, mas sem aumentar a altura da célula. O transistor tem a sua região ativa paralelizada (posta lado a lado), assim você tem a seguinte configuração *Dreno*/*Source*/*Dreno*, para um *folding* de com 2 *fingers*.

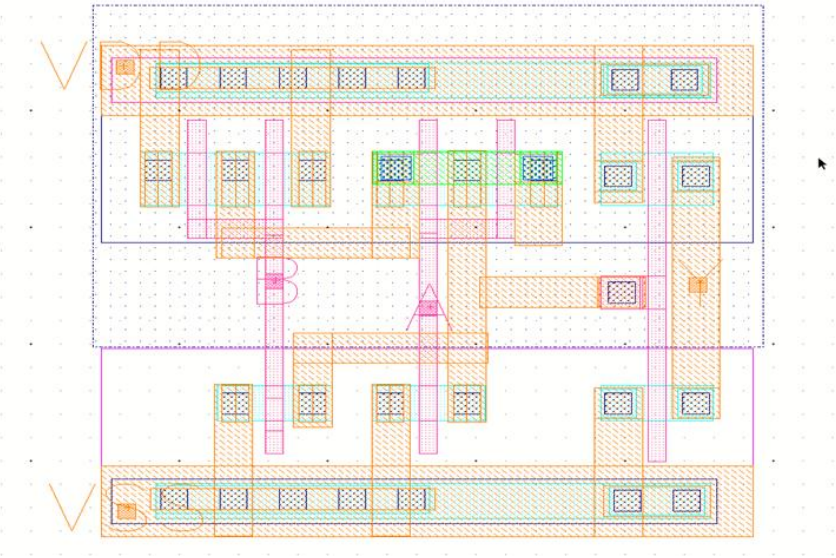


Figura 5.4: OR 2X1

5.5 Nor 2X1

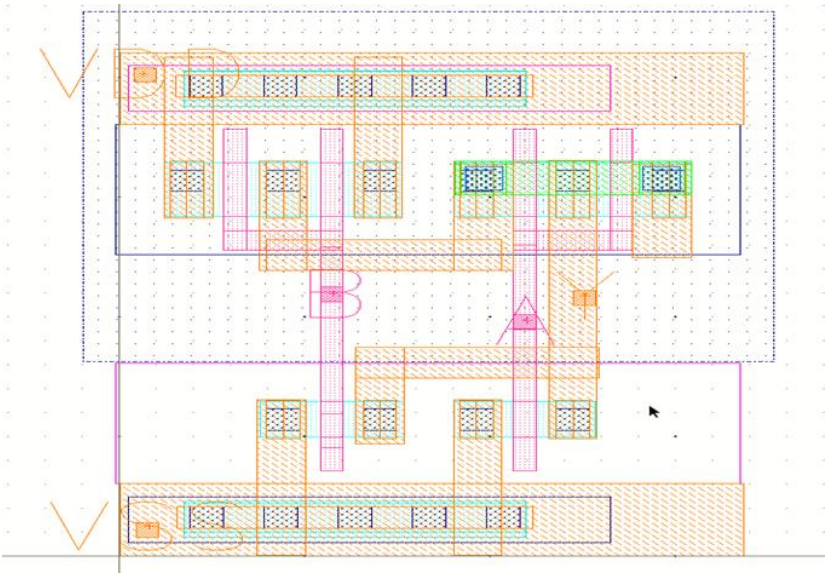


Figura 5.5: NOR 2X1

5.6 Registrador com Reset

O registrador com reset nada mais é que um registrador sem reset que ao invés de ter um inversor ligado no outro, em cada LATCH, tem uma porta NOR ligada em um inversor, em cada LATCH.

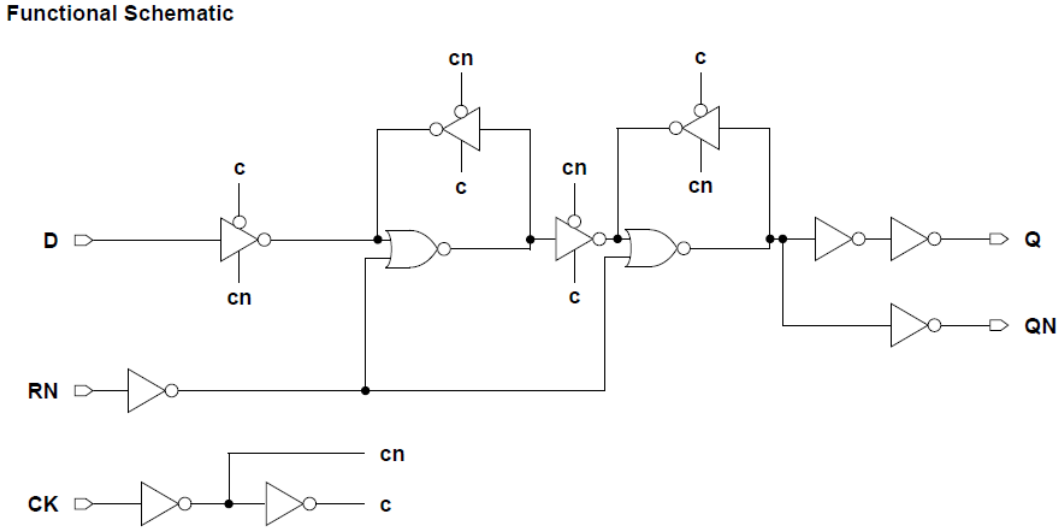


Figura 5.6: Registrador Esquemático

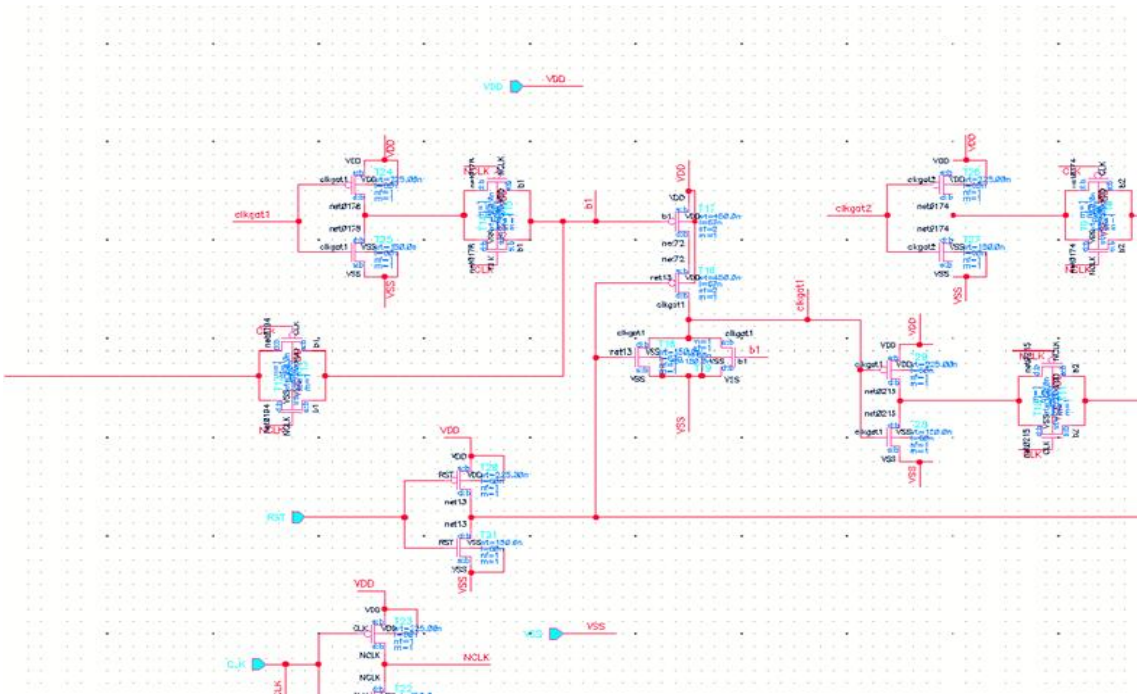


Figura 5.7: Registrador Esquemático 1

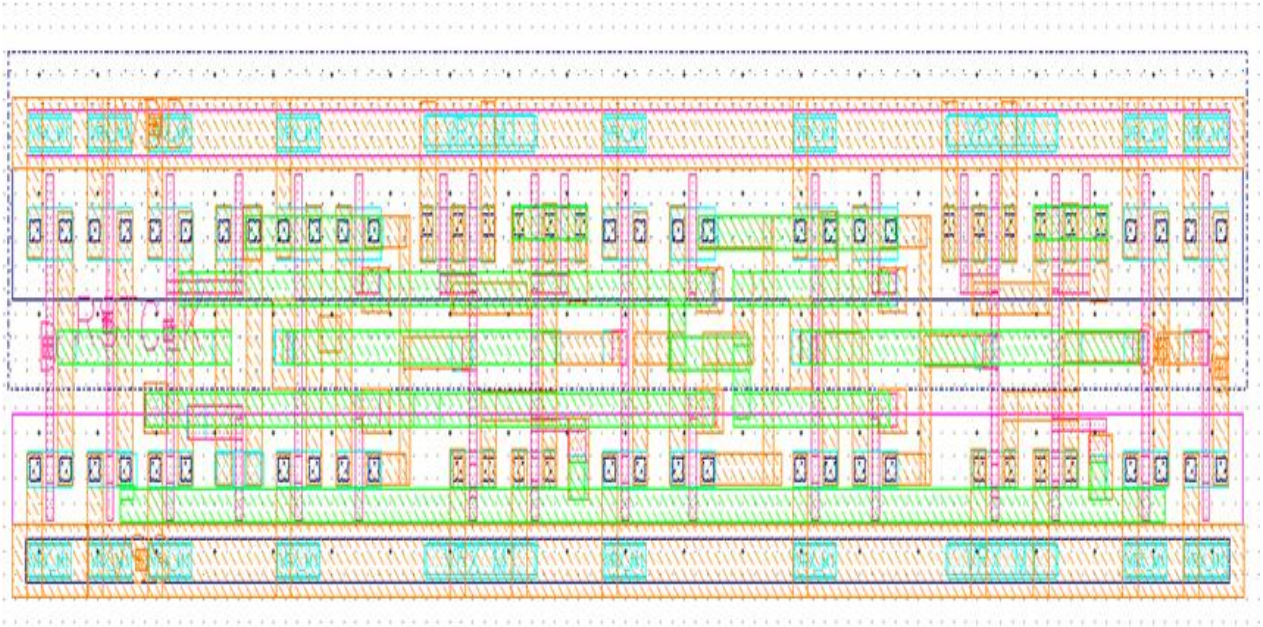


Figura 5.8: Registrador Layout

6 CÓDIGO RTL

O entendimento do algoritmo da FFT é extremamente importante para que se possa descrever um hardware capaz de implementá-la. Esse entendimento foi feito no capítulo 4 dessa monografia. Baseado neste entendimento foram criados dois casos de teste para verificar se o hardware descrito tem a funcionalidade esperada. Um teste com valores escolhidos ao acaso (randomicamente) e outro que explora os limites dos valores entrada. Para verificar o funcionamento do hardware foi feita uma simulação funcional no *ModelSim*™ e no *SimVision*™ foi realizada a simulação lógica após realizar a síntese lógica. Como trabalho adicional o código RTL foi mapeado para um dispositivo FPGA da *Xilinx*™ o qual terá suas saídas validadas usando o módulo do *ChipScope*™ para observar os sinais no interior do FPGA.

6.1 Casos de Teste

Abaixo na figura 6.1 e na figura 6.2 a esquematização do algoritmo da FFT de 8 pontos e os valores dos dois casos para teste:

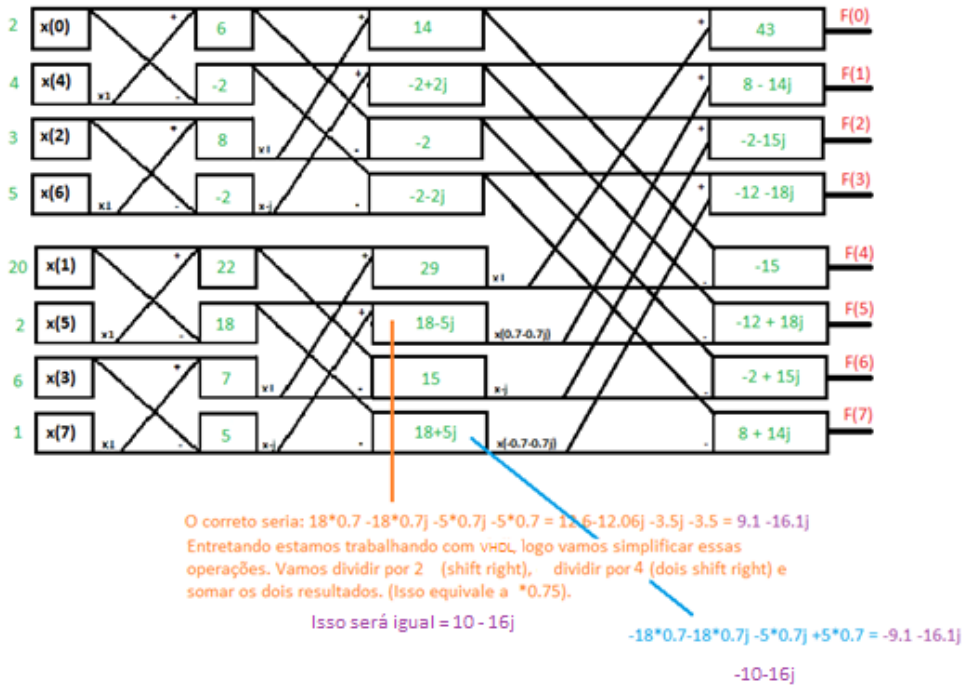


Figura 6.1: Primeiro caso para teste FFT

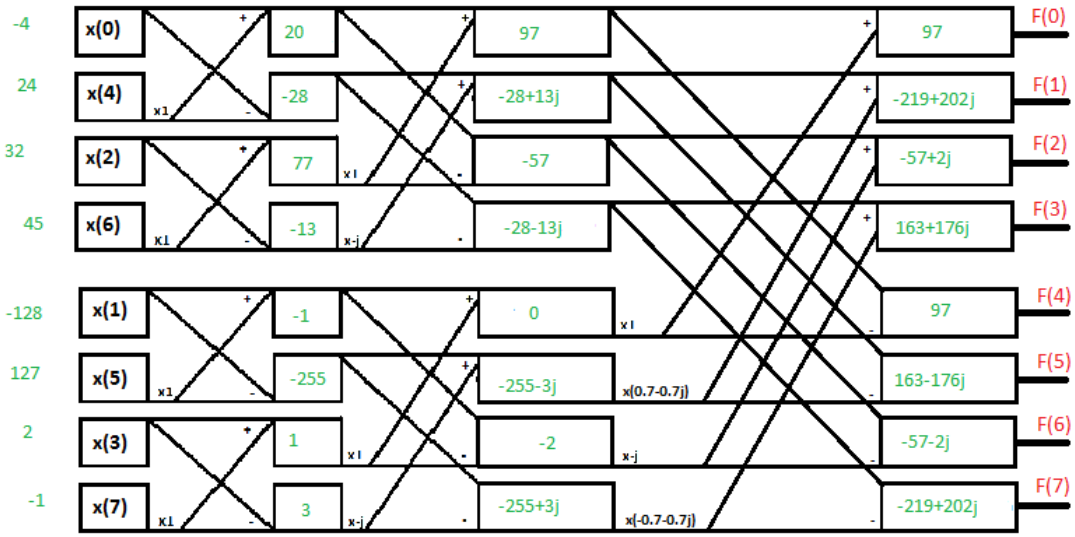


Figura 6.2: Segundo caso para teste FFT

Ainda para este trabalho foi criado um *testbench* com 500 valores aleatórios de entrada com a finalidade de se obter uma estimativa de chaveamento das portas lógicas do circuito, para obter-se uma estimativa da potência. Este *testbench* foi criado utilizando um programa em C que gera valores aleatórios, este programa se encontra no Anexo C desta monografia**.** Gerar esta estimativa de potência é importante para que se possa obter valores de potência mais próximos de uma real situação de uso desta FFT, aonde seus valores de entrada não serão constantes, pelo contrário, apresentarão altas taxas de variação em função das atividade neuronais.

6.2 Passo a Passo do RTL

Foi desenvolvido para este trabalho um circuito integrado descrito em VHDL que executa a FFT de 8 pontos. Ele demora apenas 5 ciclos de clock para produzir os resultados, lembrando que mínimo teórico seria 3 em função da dependência dos dados nos 3 estágios. A menos que se use um circuito puramente combinacional com alto custo de complexidade. Vamos agora dar todos os detalhes de como foi descrito o circuito integrado da FFT de 8 pontos:

Na figura 6.3 podemos ver a declaração das bibliotecas necessárias. Os sinais de entrada e saída do circuito, os dados de entrada tem tamanho de 8 bits e estão representados em complemento de 2. A saída tem tamanho 16 bits embora para a FFT de 8 pontos uma saída de 11 bits já seria suficiente para representar uma soma de valores máximos de entrada "127", mas já pensando numa expansão futura dessa FFT, lembrando que a escolha de 8 pontos se deu pela simplicidade a fim de que se pudesse aprender melhor este algoritmo, a saída foi fixada em 16 bits. Outro detalhe é que entram 8 sinais e saem 16 sinais, isto porque a Transformada de Fourier produz números complexos. Também é possível verificar sinais auxiliares declarados como *"signal"*. Alguns sinais auxiliares são mapeados para os pinos de saída outros são apenas sinais de controle.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;


entity fftsn is
port  (
      clk , reset : in std_logic;
      x0, x1, x2, x3, x4, x5, x6, x7 : in std_logic_vector(7 downto 0);
      F0_R, F0_I, F1_R, F1_I, F2_R, F2_I, F3_R, F3_I, F4_R, F4_I, F5_R, F5_I, F6_R, F6_I, F7_R, F7_I : out std_logic_vector(15 downto 0);
         pronto: out std_logic);
end fftsn;

architecture behavior of fftsn is

   signal f0r, f0i, f1r, f1i, f2r, f2i, f3r, f3i, f4r, f4i, f5r, f5i,  f6r, f6i, f7r, f7i: std_logic_vector(15 downto 0);
   signal ctrl_inv: std_logic_vector(3 downto 0);

   type state is (state0, state1, state2, state3, state4);
   signal atual_estado : state;
```

Figura 6.3: Entradas e Saídas VHDL FFT

Na figura 6.4 é feito o mapeando dos sinais de interesse do nosso hardware para as saídas.

```
begin
F0_R <= f0r;
F0_I <= f0i;

F1_R <= f1r;
F1_I <= f1i;

F2_R <= f2r;
F2_I <= f2i;

F3_R <= f3r;
F3_I <= f3i;

F4_R <= f4r;
F4_I <= f4i;

F5_R <= f5r;
F5_I <= f5i;

F6_R <= f6r;
F6_I <= f6i;

F7_R <= f7r;
F7_I <= f7i;
```

Figura 6.4: Mapeamento dos Sinais Imaginários e Reais da FFT (em VHDL)

Na figura 6.5 é mostrado o reset do sistema aonde os sinais do hardware são zerados.

```
process(clk)
    begin
    if reset = '1' then
            f0r <= "0000000000000000";
            f0i <= "0000000000000000";
            f1r <= "0000000000000000";
            f1i <= "0000000000000000";
            f2r <= "0000000000000000";
            f2i <= "0000000000000000";
            f3r <= "0000000000000000";
            f3i <= "0000000000000000";
            f4r <= "0000000000000000";
            f4i <= "0000000000000000";
            f5r <= "0000000000000000";
            f5i <= "0000000000000000";
            f6r <= "0000000000000000";
            f6i <= "0000000000000000";
            f7r <= "0000000000000000";
            f7i <= "0000000000000000";

            ctrl_inv(0) <= '0';
            ctrl_inv(1) <= '0';
            ctrl_inv(2) <= '0';
            ctrl_inv(3) <= '0';

            pronto <= '0';
            atual_estado <= state0;
    elsif (clk'event and clk = '1') then
```

Figura 6.5: Reset VHDL FFT

No primeiro estado da máquina de estados é realizado o reset dos sinais de controle para a FFT. É nele que é feita a extensão dos bits entrada para 16. Essa extensão se dá verificando se o número original é negativo, neste caso se entende com "11111111", ou se for positivo se estende com "00000000". Abaixo podemos ver na figura 6.6 a extensão de dois dos oito sinais de entrada:

```
case atual_estado is
  when  state0 =>

        pronto <= '0';
        ctrl_inv(0) <= '0';
        ctrl_inv(1) <= '0';
        ctrl_inv(2) <= '0';
        ctrl_inv(3) <= '0';

        if (x0(7) = '1') then
           f0r <= "11111111" & x0;
           f0i <= "0000000000000000";
        else
           f0r <= "00000000" & x0;
           f0i <= "0000000000000000";
        end if;

        if(x1(7) = '1') then
           f1r <= "11111111" & x1;
           f1i <= "0000000000000000";
        else
           f1r <= "00000000" & x1;
           f1i <= "0000000000000000";
        end if;
```

Figura 6.6: Estado 1 VHDL FFT

No segundo estado são realizadas as operações do primeiro estágio da FFT, que consistem em somas e subtrações cruzadas, conforme mostra a figura 6.7 abaixo:

```
when state1 =>
        f0r <= f0r + f4r;
    f1r <= f0r - f4r;

    f2r <= f2r + f6r;
    f3r <= f2r - f6r;

    f4r <= f1r + f5r;
    f5r <= f1r - f5r;

    f6r <= f3r + f7r;
    f7r <= f3r - f7r;

    atual_estado <= state2;
```

Figura 6.7: Estado 2 VHDL FFT

No terceiro estado são realizadas as operações do segundo estágio da FFT, que consistem em somas e subtrações parecidas com as do primeiro estágio, mas é onde pela primeira vez surgem valores complexos para serem armazenados (figura 6.8).

```
when state2 =>
  f0r <= f0r + f2r;

  f1r <= f1r;
  f1i <= "0000000000000000" - f3r;

  f2r <= f0r - f2r;

  f3r <= f1r;
  f3i <= f3r;

  f4r <= f4r + f6r;

  f5r <= f5r;
  f5i <= "0000000000000000" - f7r;

  f6r <= f4r - f6r;

  f7r <= f5r;
  f7i <= f7r;

  atual_estado <= state3;
```

Figura 6.8: Estado 3 VHDL FFT

A figura 6.9 abaixo mostra o quarto estado da FFT, onde para os coeficientes de Fourier F0, F4, F6 e F2 são calculados os seus valores finais; para os demais coeficientes é realizado o teste se o número armazenado é negativo para então fazer a sua inversão de sinal, pois o próximo estado usa uma aproximação para calcular a multiplicação por 0.7 e esta aproximação só pode ser aplicada a números positivos. Então neste caso é setado um sinal de controle, para que no próximo estado após calculada a multiplicação por 0.7 este resultado seja invertido.

```
when state3 =>

  f0r <= f0r + f4r;
  f4r <= f0r - f4r;

  f2r <= f2r;
  f2i <= "0000000000000000" - f6r;

  f6r <= f2r;
  f6i <= "0000000000000000" + f6r;

  if(f5r(15) = '1') then
    ctrl_inv(0) <= '1';
    f5r <= "0000000000000000" - f5r;
  end if;

  if(f5i(15) = '1') then
    ctrl_inv(1) <= '1';
    f5i <= "0000000000000000" - f5i;
  end if;

  if(f7r(15) = '1') then
    ctrl_inv(2) <= '1';
    f7r <= "0000000000000000" - f7r;
  end if;

  if(f7i(15) = '1') then
    ctrl_inv(3) <= '1';
    f7i <= "0000000000000000" - f7i;
  end if;
  atual_estado <= state4;
```

Figura 6.9: Estado 4 VHDL FFT

Na próxima figura 6.10 vemos o final do processamento da FFT, é mostrado o quinto estado do cálculo aonde é feita as multiplicações por 0,7. É necessário resaltar que os cálculos neste trabalho estão representados números de ponto fixo. Ou seja, aqueles resultados que tiverem números em ponto flutuante serão arredondados ou truncados. Também foi feita uma aproximação para aumentar o desempenho que consiste na operação de multiplicar por 0,7 presente no terceiro estágio da FFT de 8 pontos, ao invés disso, dividiremos por 2 e depois dividiremos por 4 e somaremos os dois resultados, equivale a multiplicar por 0,75, mas como estamos tratando de números inteiros com o arredondamento ou truncamento dos resultados podemos na verdade estar obtendo o mesmo valor que multiplicando por 0,7.

Essa aproximação garante mais velocidade e menos potência, pois essas operações são apenas deslocamento de bits para direita com números positivos. Se o valor de entrada nesse estágio da FFT era negativo então será necessário inverter o resultado final. A figura 6.10 mostra exatamente isso, as várias possibilidades para o cálculo dos coeficientes F1 e F5.

```
when state4 =>

  pronto <= '1';

  if(ctrl_inv(1) = '1' and ctrl_inv(0) = '1') then
     f1r <= f1r - (('0'&f5r(15 downto 1)) + ("00"&f5r(15 downto 2)))  - (('0'&f5i(15 downto 1)) + ("00"&f5i(15 downto 2)));
        f1i <= f1i +  (('0'&f5r(15 downto 1)) + ("00"&f5r(15 downto 2))) - (('0'&f5i(15 downto 1)) + ("00"&f5i(15 downto 2)));
  elsif(ctrl_inv(1) = '1') then
     f1r <= f1r +  (('0'&f5r(15 downto 1)) + ("00"&f5r(15 downto 2))) - (('0'&f5i(15 downto 1)) + ("00"&f5i(15 downto 2)));
        f1i <= f1i -  (('0'&f5r(15 downto 1)) + ("00"&f5r(15 downto 2))) - (('0'&f5i(15 downto 1)) + ("00"&f5i(15 downto 2)));
  elsif(ctrl_inv(0) = '1') then
     f1r <= f1r -  (('0'&f5r(15 downto 1)) + ("00"&f5r(15 downto 2))) + (('0'&f5i(15 downto 1)) + ("00"&f5i(15 downto 2)));
     f1i <= f1i +  (('0'&f5r(15 downto 1)) + ("00"&f5r(15 downto 2))) + (('0'&f5i(15 downto 1)) + ("00"&f5i(15 downto 2)));
  else
     f1r <= f1r +  (('0'&f5r(15 downto 1)) + ("00"&f5r(15 downto 2))) + (('0'&f5i(15 downto 1)) + ("00"&f5i(15 downto 2)));
     f1i <= f1i -  (('0'&f5r(15 downto 1)) + ("00"&f5r(15 downto 2))) + (('0'&f5i(15 downto 1)) + ("00"&f5i(15 downto 2)));
  end if;



  if(ctrl_inv(1) = '1' and ctrl_inv(0) = '1') then
     f5r <= f1r + (('0'&f5r(15 downto 1)) + ("00"&f5r(15 downto 2)))  + (('0'&f5i(15 downto 1)) + ("00"&f5i(15 downto 2)));
     f5i <= f1i -  (('0'&f5r(15 downto 1)) + ("00"&f5r(15 downto 2))) + (('0'&f5i(15 downto 1)) + ("00"&f5i(15 downto 2)));
  elsif(ctrl_inv(1) = '1') then
     f5r <= f1r -  (('0'&f5r(15 downto 1)) + ("00"&f5r(15 downto 2))) + (('0'&f5i(15 downto 1)) + ("00"&f5i(15 downto 2)));
     f5i <= f1i +  (('0'&f5r(15 downto 1)) + ("00"&f5r(15 downto 2))) + (('0'&f5i(15 downto 1)) + ("00"&f5i(15 downto 2)));
  elsif(ctrl_inv(0) = '1') then
     f5r <= f1r +  (('0'&f5r(15 downto 1)) + ("00"&f5r(15 downto 2))) - (('0'&f5i(15 downto 1)) + ("00"&f5i(15 downto 2)));
     f5i <= f1i -  (('0'&f5r(15 downto 1)) + ("00"&f5r(15 downto 2))) - (('0'&f5i(15 downto 1)) + ("00"&f5i(15 downto 2)));
  else
     f5r <= f1r -  (('0'&f5r(15 downto 1)) + ("00"&f5r(15 downto 2))) - (('0'&f5i(15 downto 1)) + ("00"&f5i(15 downto 2)));
     f5i <= f1i +  (('0'&f5r(15 downto 1)) + ("00"&f5r(15 downto 2))) - (('0'&f5i(15 downto 1)) + ("00"&f5i(15 downto 2)));
  end if;
```

Figura 6.10: Estado 5 VHDL FFT

6.3 Simulações e Verificações

As simulações e validações deste RTL foram um dos pontos bem explorados do trabalho. Porque além da análise de potência de um determinado circuito integrado, é buscado explorar o aprendizado de um módulo de processamento de sinais. Para esta finalidade é preciso entender o circuito e descrever o comportamento dele corretamente. Por isso usamos todo ferramental disponível e técnicas que foram aprendidas durante o curso de Engenharia de Computação. Uma dessas técnicas para validação foi o uso de FPGA, além do ModelSim™ para simulação funcional, duas ferramentas que não eram necessárias para que se chegasse aos mesmos resultados finais,

mas que são de suma importância para validação daquilo que esta se propondo fazer.

6.3.1 Simulação Funcional ModelSim™

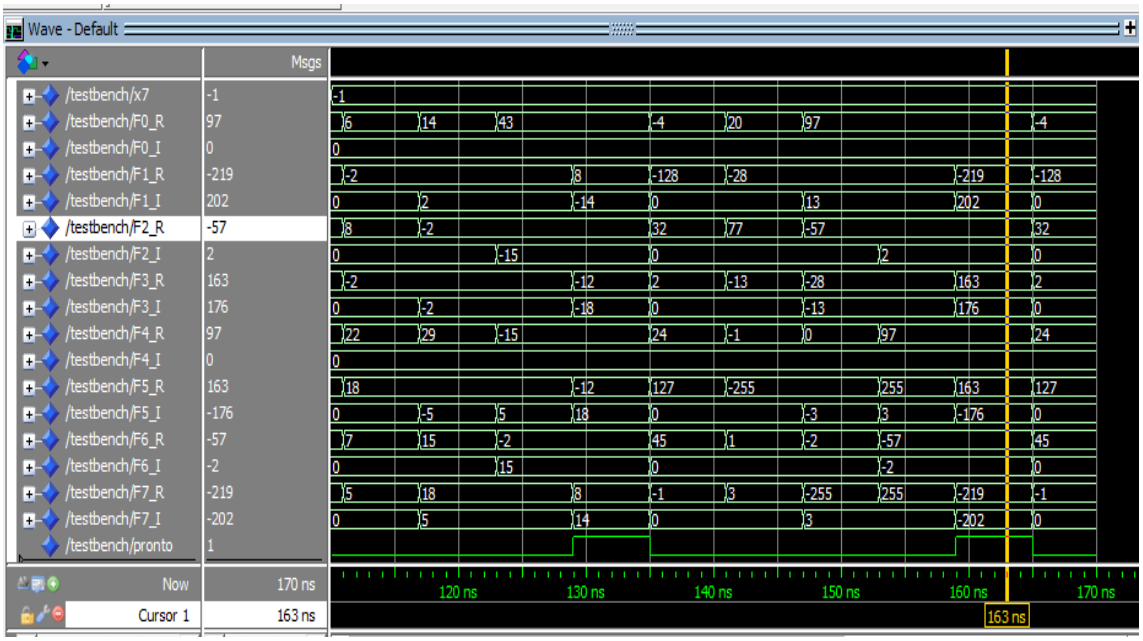


Figura 6.11: Simulação ModelSim™

6.3.2 Simulação Pré-Layout SimVision™

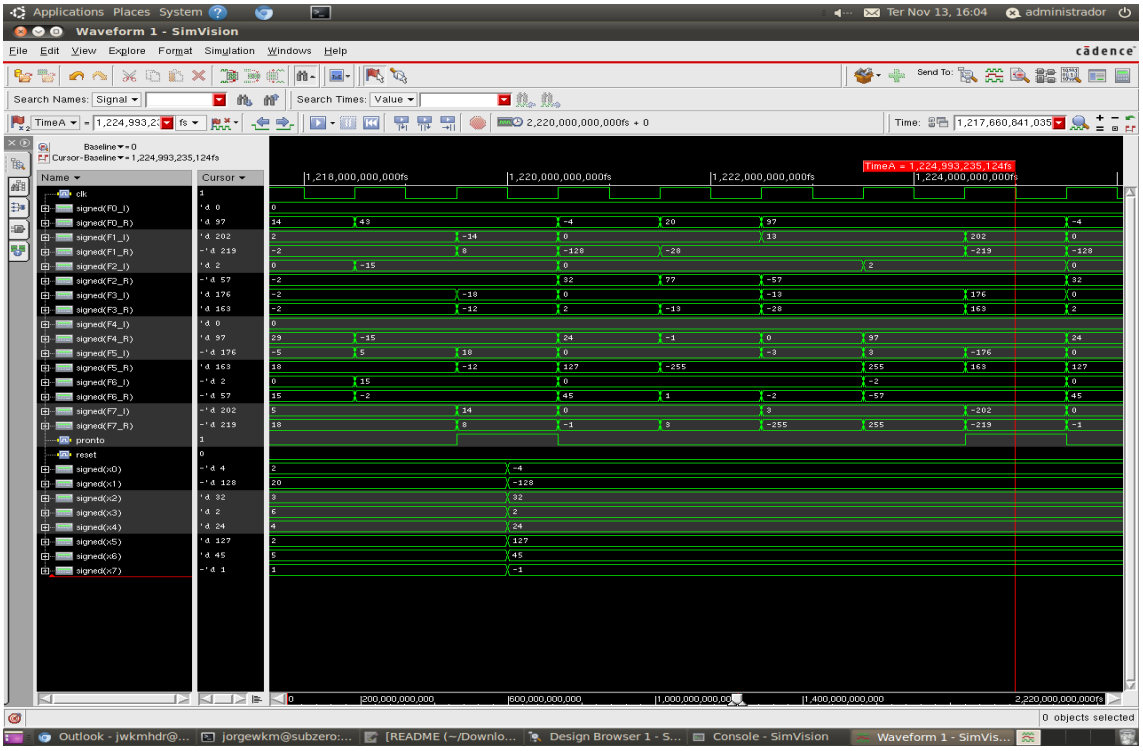


Figura 6.12: Simulação Lógica Pré-Layout.

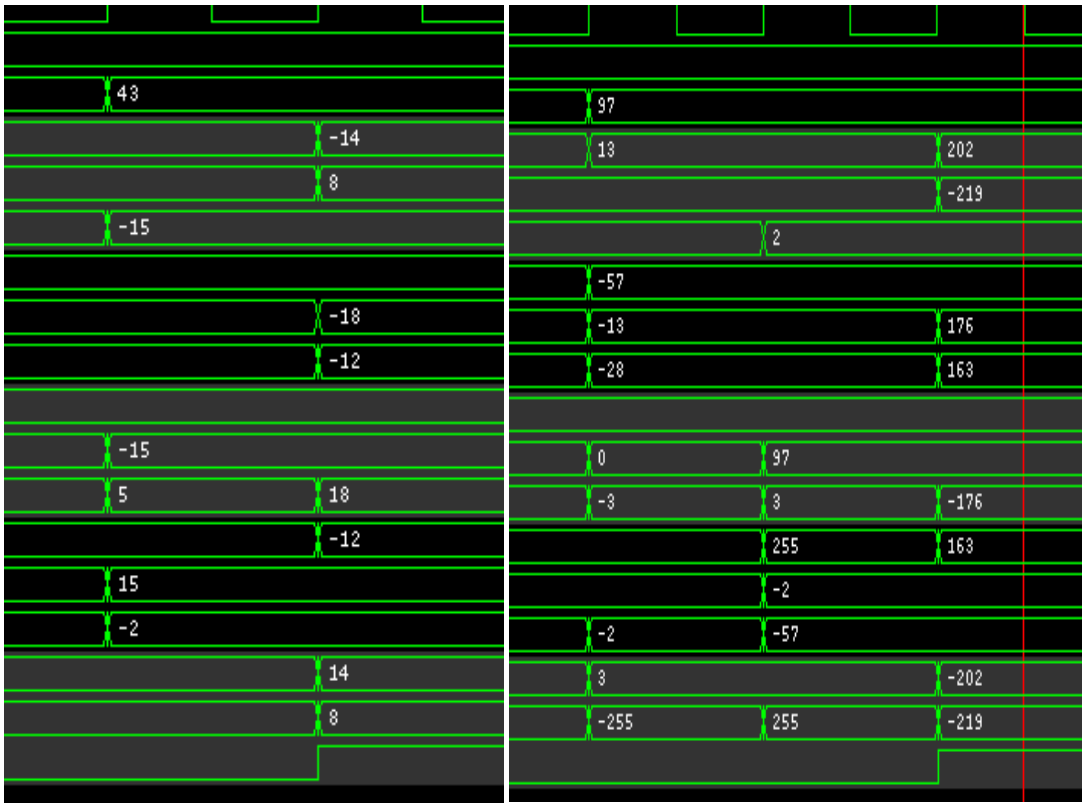


Figura 6.13: Simulação Lógica 1

6.3.3 Simulação 500 Valores Aleatórios

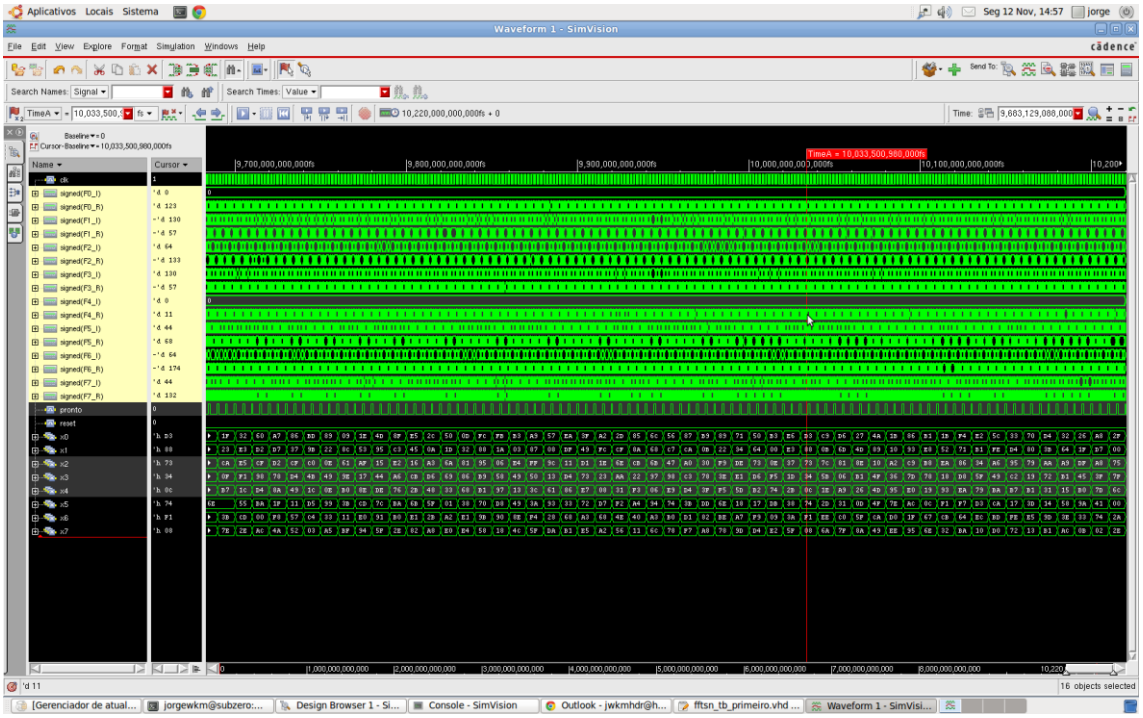


Figura 6.14: Simulação Lógica 500 Valores Aleatórios

6.3.4 Sinais no FPGA usando *ChipScope*™

O VHDL da FFT ainda foi testado e validado no hardware, numa placa de FPGA da *Xilinx*™ usando o módulo do *ChipScope*™ para observar os sinais no interior do FPGA.

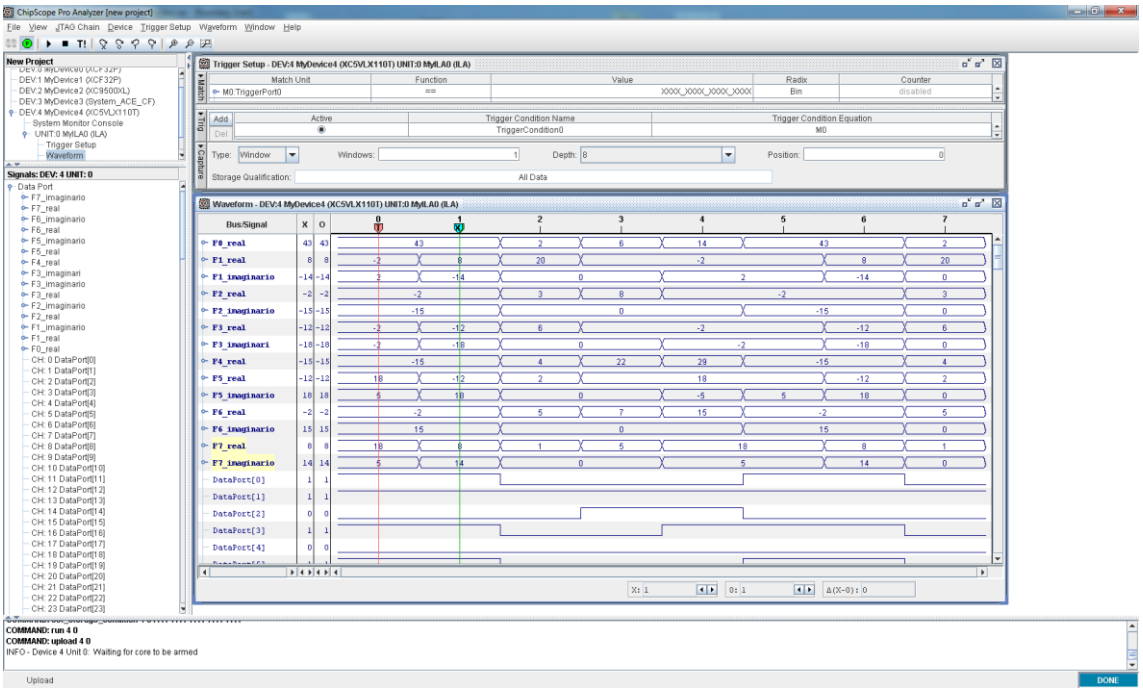


Figura 6.15: Resultado *ChipScope*™

7 RESULTADOS

Os resultados a seguir foram obtidos fazendo-se sucessivas sínteses, em diferentes condições, porém com a restrição de frequência mínima de operação do circuito da FFT em 1MHz.

A fim de estimar a potência desses circuitos foram utilizadas duas abordagens, uma delas foi a partir dos resultados gerados pelo *RTL-compiler*™, o responsável por fazer a síntese lógica do hardware. Outra abordagem foi rodar um arquivo de *testbench*, gerando um arquivo VCD que possui o registro da atividade de chaveamento de todas as portas lógicas do circuito, com ele se pode fazer uma estimativa muito mais real da potência consumida pelo circuito. Esta estimativa encima do arquivo de VCD também é realisada pelo *RTL-compiler*™, mas neste caso o valor para chaveamento de cada uma das portas varia de acordo com a sua taxa de variação no *testbench*. Isto pode ter grande impacto no resultado final de potência, por exemplo se você tem um somador de 16 bits que apenas realiza somas com os 8 bits menos significativos a potência estimada pelo *RTL-compiler*™ vai ser muito maior que para o real, pois o *RTL-compiler*™ considera uma atividade de chaveamento de 50% para todas as portas, e como mencionado no exemplo metade das portas desse circuito nunca mudam seu valor. Por este motivo uma estimativa pelo VCD produz geralmente resultados menores, muitas vezes, pode atingir até três ordens de grandeza.

Para os resultados obtidos a partir de arquivos VCD foi considerado um *testbench* com valores de entrada para o hardware da Transformada Rápida de Fourier fixos e outro com 500 valores de entrada aleatórios. Em ambos, os resultados foram muito semelhantes, embora teoricamente poder-se-ia esperar que *testbench* de valores aleatórios deveria produzir uma maior atividade de chaveamento das portas, gerando portanto um gasto maior de potência. Em trabalhos futuros esta questão do método de estimativa de potência deve ser melhor analisada. Uma possível justificativa para esses resultados seria a característica recursiva ou acumulativa do algoritmo da FFT.

Nos resultados que seguem foram feitas simulações para quatro diferentes temperaturas, sendo as tensões de alimentação consideradas: 0,45V e 1,20V. Além disso, foi simulada a operação do circuito para a temperatura de 25 graus Celsius, com os pontos de operação da alimentação em 0,46V, 0,48V, 0,54V, 0,75V, 0,9V. Espera-se com isso analisar de forma mais rigorosa a menor tensão de *near-threshold* e a tensão nominal obtendo maior quantidade detalhes das mesmas. Bem como verificar como cresce a curva de potência com aumento da tensão de alimentação. Baseando-se nos primeiros resultados da tensão nominal e da tensão near-threshold foi possível estimar a variação da potência com a temperatura nessas tensões intermediárias.

7.1 Comportamento da Potência com a Temperatura avaliada pelo *RTL-compiler*™

	-40°Celsius	0°Celsius	25°Celsius	80°Celsius
0,45V	76,8µW	68,2µW	62,5µW	47,5µW
1,2V	243,4µW	239.8µW	249,4µW	255,4µW

Tabela 7.1: Potência X Temperatura (*RTL-compiler*™)

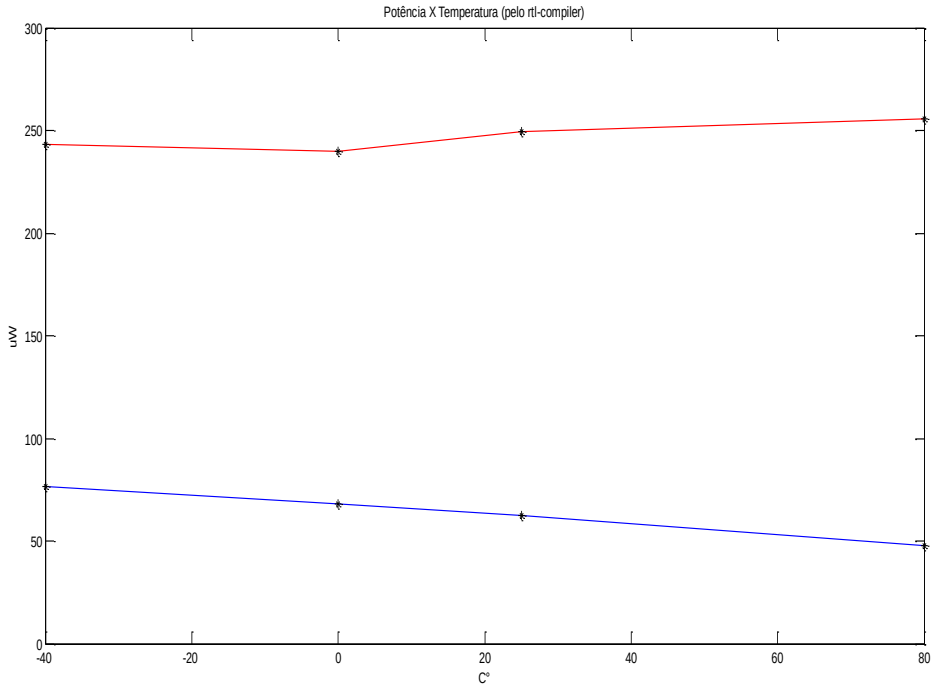


Figura 7.1: Potência X Temperatura (*RTL-compiler*™)

A figura 7.1 mostra os resultados de potência estimados pela ferramenta RTL-Compiler™, considerando o método mais simples de avaliação de potência, o método estatístico em que fixamos a atividade de chaveamento de todas as entradas de sinal em 0,5 (50% de atividade de chaveamento). Podemos observar nesses resultados a economia de potência, quase 5 vezes menor para uma tensão que é 0,37 vezes a tensão de alimentação nominal.

7.2 Comportamento da Potência com a Temperatura usando VCD

	-40°Celsius	0°Celsius	25°Celsius	80°Celsius
0,45V	7,6µW *	5,5µW	5,0µW	6,4µW
1,2V	26,6µW	27,0µW	28,5µW	33,8µW

Tabela 7.2: Potência X Temperatura (VCD)

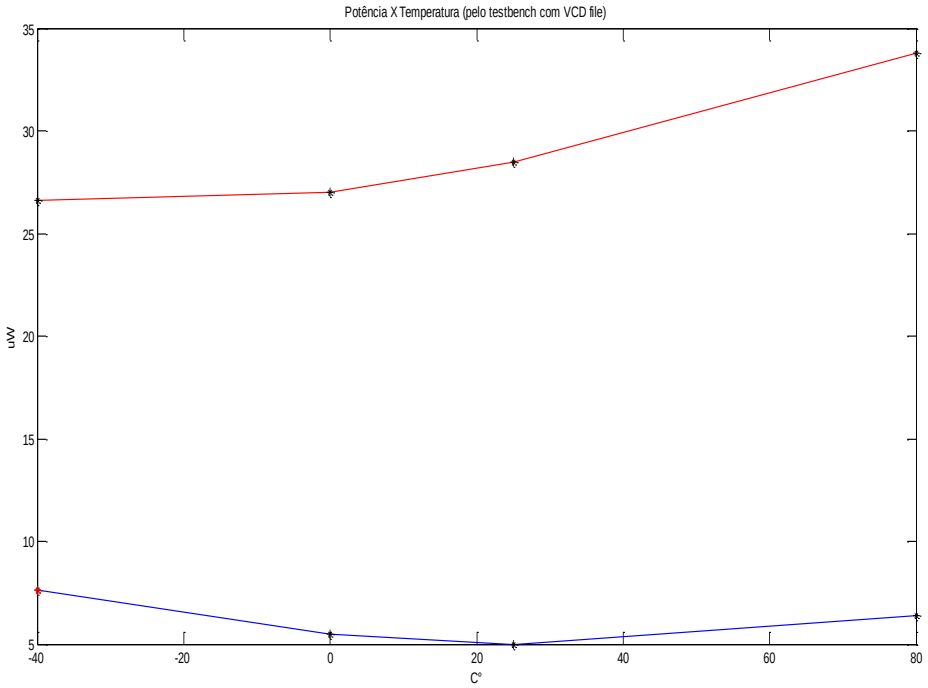


Figura 7.2: Potência X Temperatura (VCD)

A figura 7.2 mostra os resultados de potência estimados utilizando um arquivo VCD que descreve o padrão de chaveamento das portas do circuito em função da execução de um *testbench*. Podemos observar nesses resultados que, operando na região de *Near-threshold,* a diminuição de temperatura tem um efeito negativo no desempenho, fazendo com que o gasto de potência seja maior.

Houve violações nas restrições de *timing* a 1 MHz para a temperatura de operação de -40°Celsius. Logo, não se pode verificar o chaveamento das portas a partir do *testbench*, e este valor foi estimado baseado no padrão dos demais resultados. O comportamento a baixas temperaturas foi estimado para avaliar as características de operação a tensões baixas, sabendo-se que nesta aplicação de CIs para sistemas biomédicos próximos ao paciente, a temperatura de operação mínima deve ser em torno de zero graus Celsius.

7.3 Comportamento da Potência com o aumento da Tensão de Alimentação (Vdd)

	0,45V	0,46V	0,48V	0,54V	0,75V	0,9V	1,2V
Alfa = 0,5	62,5µW	62,1µW	47,2µW	61,0µW	93,5µW	132,2µW	249,4µW
Estim. Testbench	5,0µW	5,0µW	6,4µW	7,5µW	10,2µW	15,4µW	28,5µW

Tabela 7.3: Potência X Tensão

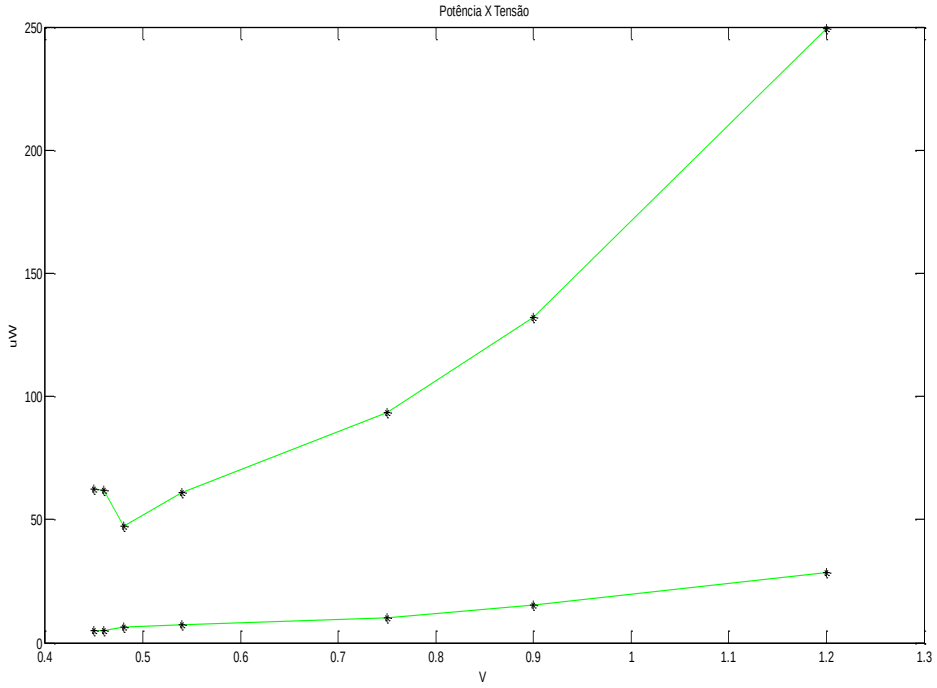


Figura 7.3: Potência X Tensão

Vemos na figura 7.3 o crescimento do consumo de potência com o aumento da tensão de alimentação. Na aproximação de estimativa utilizando atividade 0,5 (*alpha*=0,5), o *RTL-compiler*™ sobre-estima a dissipação de potência, como mostra a figura 7.3 e a Tabela 7.3. Nesta estimativa o padrão de crescimento se aproxima mais de uma curva quadrática. Mas para a potência estimada a partir do chaveamento das portas simulado em função de um *testbench*, a curva quadrática se aproxima de uma variação linear. Como sabemos que a componente de dissipação causada pela corrente de curto-circuito das portas lógicas é linear com a tensão Vdd, os termos lineares da equação são dominantes. A continuar aumentando a temperatura, esta curva se tornaria mais próxima de uma quadrática. As duas metodologias de estimativa de potência resultam bastante diferentes, sendo que a avaliação com os arquivos .VCD mostram dissipação muito mais baixa – da ordem de 5 micro-Watts para tensão de 0,45 V.

7.4 Decremento das Instâncias de Células com o aumento da Tensão

0,45V	0,46V	0,48V	0,54V	0,75V	0,9V	1,2V
18081	17609	18222	17728	9519	9227	9381

Tabela 7.4: Instâncias de Células X Tensão

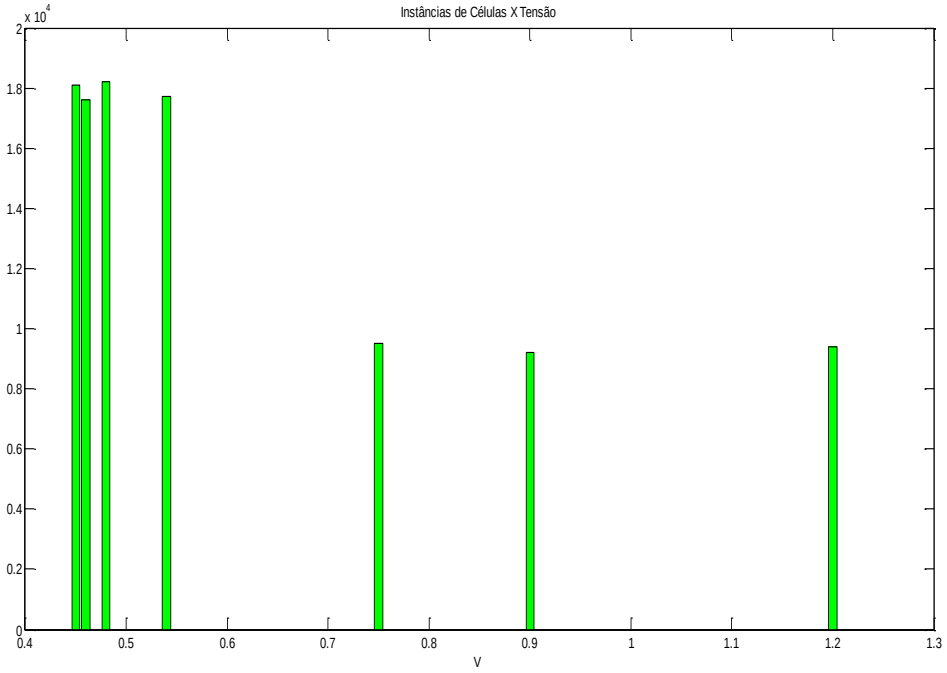


Figura 7.4: Instâncias de Células X Tensão

Atender as restrições de timing em tensões maiores é mais fácil, pois os transistores operam em inversão forte, conduzindo mais corrente elétrica, principalmente se tiverem ultrapassado VT (Tensão de *Threshold*) o que gera circuitos menores após a síntese lógica.

Entretanto deve-se considerar que para este trabalho a biblioteca inclui somente um *strength* de cada célula, dificultando métodos de *sizing*. Bibliotecas com mais variedades de *strengths* de células dariam ao *RTL-compiler*™ mais opções para corrigir as violações de timing nos circuitos com menores tensões de alimentação, utilizando assim ao final do processo de síntese lógica uma menor quantidade de células.

7.5 Aumento do *Timing Slack* com o aumento da Tensão

0,45V	0,46V	0,48V	0,54V	0,75V	0,9V	1,2V
187,0ns	219,8ns	235,5ns	274,4ns	294,2ns	296,5ns	297,9ns

Tabela 7.5: *Timing Slack* X Tensão

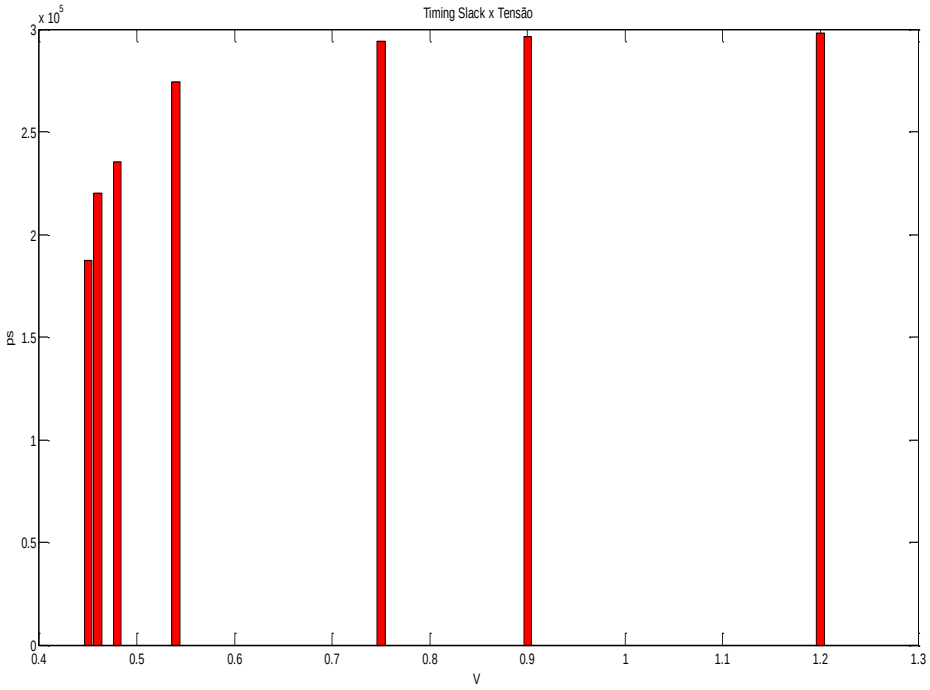


Figura 7.5: *Timing Slack* X Tensão

Como falado no item 7.4 tensões mais altas geram mais folga no alcanço das restrições de *timing*. Assim os requisitos de desempenho podem ser atendidos, pois esses circuitos tem um tempo de resposta maior.

7.6 Variação do número de Instâncias de Células com a Temperatura (Tensão 0,45V)

	-40°Celsius	0°Celsius	25°Celsius	80°Celsius
0,45V	16861	17650	18081	20011

Tabela 7.6: Instâncias de Células X Temperatura 0,45V

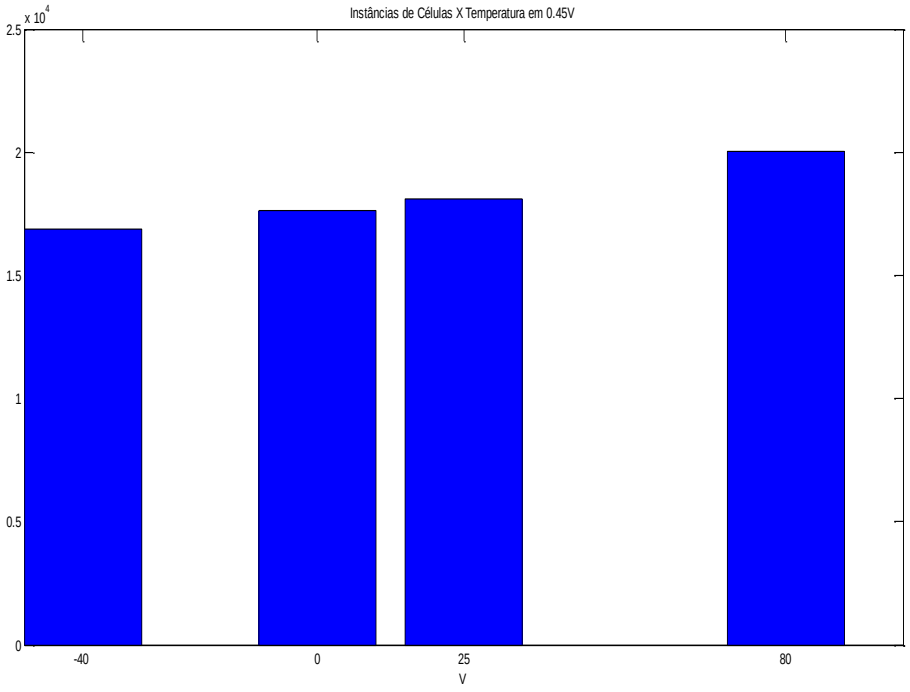


Figura 7.6: Instâncias de Células X Temperatura 0,45V

Com relação a esses resultados deve ser dito que para a temperatura de -40°Celsius o resultado do número de células não é final, visto que não foi alcançado os objetivos de timing pelo *RTL-compiler*™.

7.7 Decremento do *Timing Slack* com decremento da Temperatura (Tensão 0,45V)

	-40°Celsius	0°Celsius	25°Celsius	80°Celsius
0,45V	-1094,0ns	1,5ns	187,0ns	242,1ns

Tabela 7.7: *Timing Slack* X Temperatura 0,45V

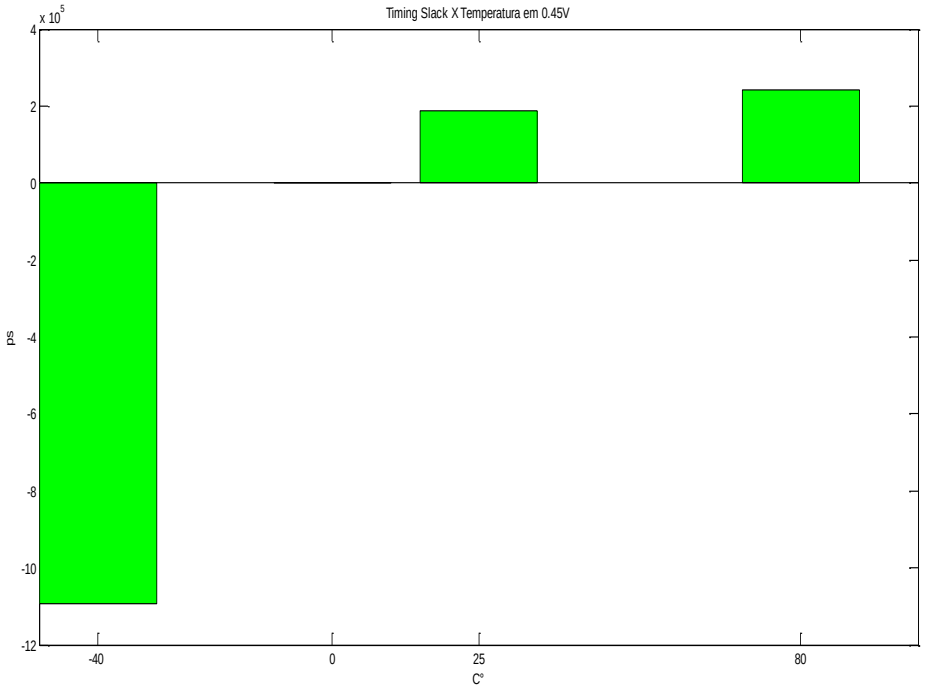


Figura 7.7: *Timing Slack* X Temperatura 0,45V

Na figura 7.7 podemos ver os resultados de obtenção de *timing* para circuitos operando a 0,45V. Nota-se que a diminuição da temperatura piora o desempenho de circuitos operando na região de *near-threshold*, o que provoca aumento da potência. Isso ocorre porque o VT aumenta com a diminuição da temperatura, logo o Transistor MOS opera com menor *gate overdrive*.

A altura das colunas no gráfico também nos permite concluir que a diminuição de temperatura tem forte impacto em 0,45V. Podemos ver isso pelo grande timing slack negativo operando a -40°Celsius.

7.8 Variação do número de Instâncias de Células com a Temperatura (Tensão 1,20V)

	-40°Celsius	0°Celsius	25°Celsius	80°Celsius
1,2V	9165	9433	9381	9390

Tabela 7.8: Instâncias de Células X Temperatura 1,20V

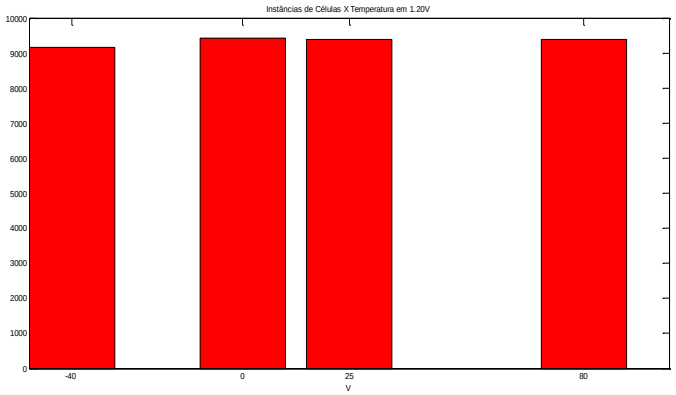


Figura 7.8: Instâncias de Células X Temperatura 1,20V

Para tensões nominais a diminuição da temperatura aumenta o desempenho, gastando menos potência, além de gerar circuitos menores.

7.9 Variação do *Timing Slack* com a Temperatura (Tensão 1,20V)

	-40°Celsius	0°Celsius	25°Celsius	80°Celsius
1,2V	297,8ns	297,9ns	297,9ns	297,9ns

Tabela 7.9: *Timing Slack* X Temperatura 1,20V

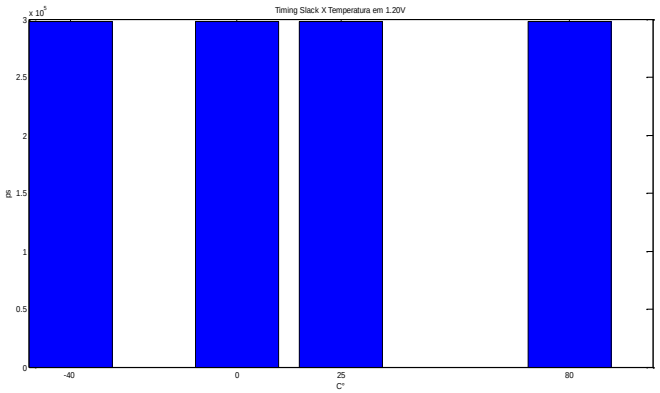


Figura 7.9: *Timing Slack* X Temperatura 1,20V

7.10 Variação do *Timing Slack* com a Temperatura (Tensões 0,45V e 1,2V)

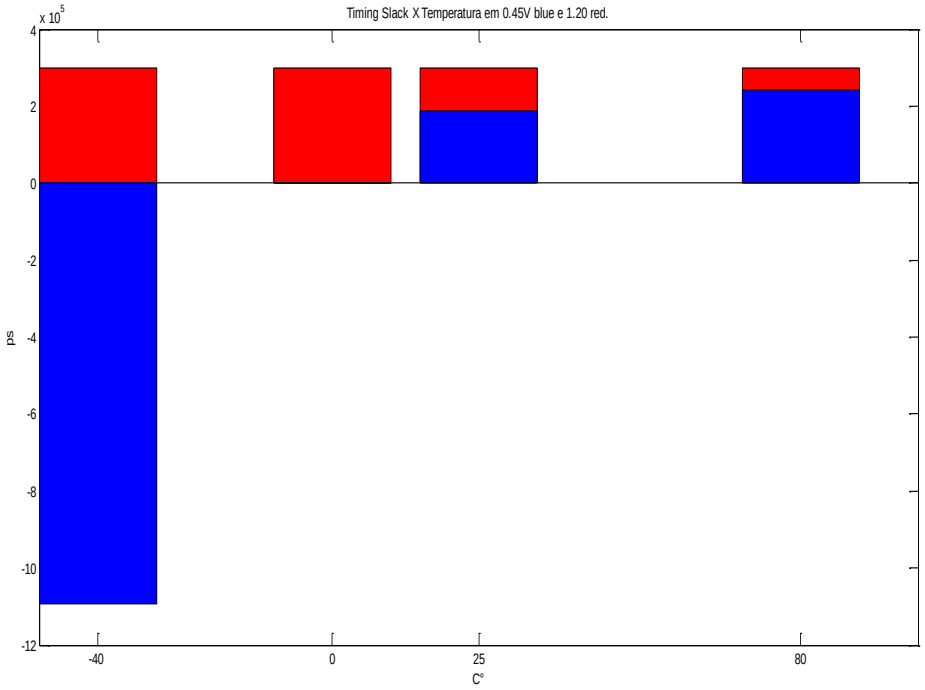


Figura 7.10: *Timing Slack* X Temperatura 1,20V e 0,45V

Na figura 7.10 vemos uma combinação dos gráficos das figuras 7.9 e 7.7. Ela mostra claramente que a diminuição da temperatura não afeta a obtenção de requisitos de desempenho em circuitos operando a tensões nominais, pelo contrário, ajuda. Já para circuitos operando em *near-threshold* essa variação produz resultados negativos, o que é atribuído ao aumento do valor da tensão de limiar de condução dos transistores à medida que a temperatura decresce. Este aumento do VT afeta negativamente os atrasos observados nas portas lógicas às temperaturas mais baixas.

7.11 Análise Final

Na figura 7.11 podemos ver o comportamento da corrente em função da tensão para um transistor em duas temperaturas diferentes. Note que esta figura não representa valores para a tecnologia 65nm empregada neste trabalho, é apenas uma imagem ilustrativa, que serve para demonstrar o comportamento com a temperatura dos transistores MOSFET discretos, em tecnologias antigas. É possível perceber que para baixas tensões o transistor com maior temperatura apresenta maior corrente. Isto ocorre porque o VT (Tensão de *Threshold*) é dependente da temperatura e aumenta com a diminuição dela. Se já estamos operando a uma tensão abaixo da tensão de funcionamento do transistor (VT), estaremos com a diminuição da temperatura operando numa diferença de tensão ainda maior em relação ao VT. Isto explica porque a baixas temperaturas um circuito operando a *near-threshold* apresenta um desempenho pior que a temperaturas mais elevadas. Quando se trata da operação em condições de inversão forte, aumentando-se a tensão de alimentação, a diminuição da temperatura aumenta a mobilidade dos portadores de carga na camada de inversão no transistor de silício. Operando acima de VT, o incremento da corrente é verificado pelo aumento da mobilidade e também da tensão de alimentação. Donde se conclui que a diminuição da temperatura traz benefícios de desempenho e de diminuição de potência para circuitos em tensões de alimentação nominais.

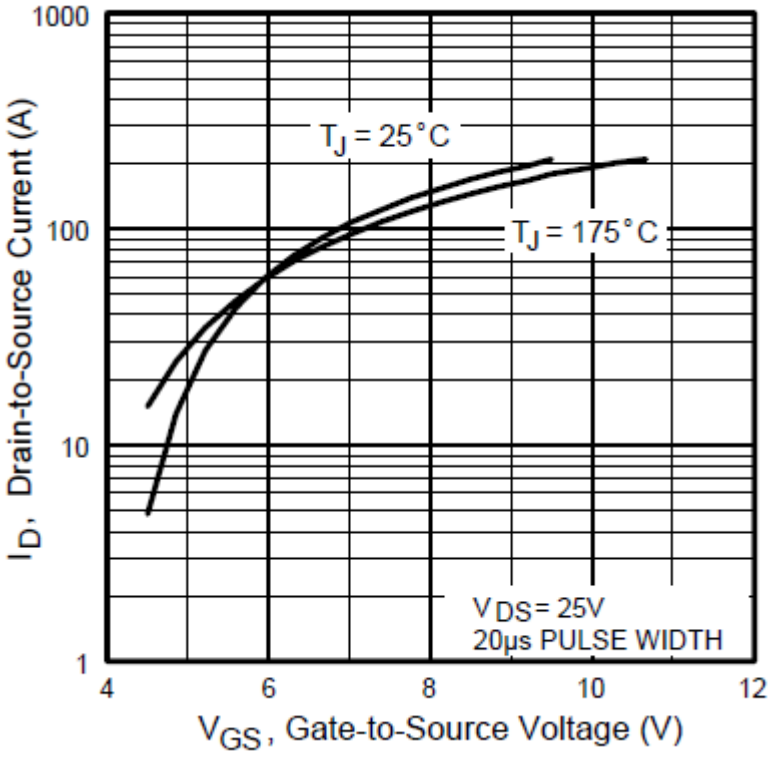


Figura 7.11: Corrente de dreno de MOSFET discreto vs. Tensão VGS (CLUBE DO HARDWARE.Resistência Gate Mosfet. http://forum.clubedohardware.com.br/resistencia-gate-mosfet)

8 CONCLUSÃO

A Transformada Rápida de Fourier foi descrita em VHDL para implementar o algoritmo Radix-2 que é o mais conhecido. Não exploramos outros algoritmos (Radix-4,Radix-8,Danielson-Lanczos) que apresentam pequenas melhoras dependendo do caso, mas a ideia é sempre a mesma : dividir para evitar cálculos redundantes. A FFT apresentou os resultados esperados.

A proposta englobou um desenvolvimento tanto de software (para avaliação do algoritmo da FFT aplicado a sinais biológicos) quanto de hardware, aplicando conhecimentos do tratamento de sinais para solucionar um problema da área médica. O trabalho de iniciação científica realizado na UFRGS está incluso nesta monografia, e como atividade adicional foi escrito um artigo para a Conferência latino-americana LASCAS 2013 da IEEE, no qual foram utilizados métodos de síntese similares ao descrito para FFT, inclusive a mesma biblioteca de células, para implementar um filtro *notch* (de supressão de banda estreita). Cópia do artigo se encontra nos anexos e é parte das referências bibliográficas deste trabalho.

Em aplicações que requerem grande autonomia de bateria, onde desempenho não é o diferencial, trabalhar na região de near-threshold representa uma grande vantagem, pois esses circuitos devem consumir menos potência, ou seja, devemos gastar o mínimo de energia necessário para realizar a tarefa.

Durante este trabalho foram realizadas diversas simulações, das quais se podem tirar conclusões importantes. Uma delas referente ao melhor ponto de operação dos circuitos *near-threshold* a determinadas temperaturas. Constatou-se que os circuitos operam melhor em faixas altas de temperatura, entre 30° e 40°. Exatamente a faixa de temperatura a que ficará submetido um circuito implantado em um ser humano, pois um hardware operando a baixas frequências terá sua temperatura determinada pelo meio e não pelo aquecimento de seus componentes. Essa constatação é extremamente importante principalmente para os projetistas saberem que as bibliotecas construídas para serem rápidas em determinados pontos de operação em tensões nominais provavelmente serão não tão ajustadas para a operação em tensões ultra-baixas, como as *near-threshold*.

Por fim é importante ressaltar que esta monografia obteve resultados bastante interessantes contribuindo para a área. O projeto do PI da FFT permitiu exercitar conhecimentos que já existiam, mas que possivelmente poderiam ser esquecidos por serem pouco tratados na era atual de circuitos operando no máximo desempenho possível. Para uma geração nova de

circuitos implantados de ultra-baixa dissipação e funcionando a *near-threshold*, esses conhecimentos não podem ser esquecidos. Este trabalho contribuiu com resultados úteis a quem tiver que projetar circuitos integrados para operação em regime de potências e tensões ultra-baixas.

REFERÊNCIAS

[1] ALWAYSLEARN. A DFT and FFT Tutorial. http://www.alwayslearn.com.

[2] BEAR, Mark F. Neurociências - Desvendando o Sistema Nervoso. 3ª Ed

[3] BLAHUT, Richard E. Fast algorithms for digital signal processing. 1985.

[4] BUZSÁKI G. Rhythms of the brain, Oxford University Press: New York,2006.

[5] CALHOUN, B.H.; WANG, A.; CHANDRAKASAN, A, “Modeling and sizing for minimum energy operation in subthreshold circuits”, in IEEE Journal of Solid-State Circuits, Vol. 40, 2005, pp.1778 - 1786.

[6] CEBL, CSU EEG Brain-computer interface Lab, http://www.cs.colostate.edu/eeg/eegSoftware.html, Accessed in 2012.

[7] COURTINE Grégoire. http://courtine-lab.epfl.ch/team

[8] HANKIN, Simon. Sinais e sistemas. 2001.

[9] IZHIKEVICH, Eugene. Neuron Models on FPGA. 2003

[10] KAUL, H. “Near-threshold voltage (NTV) design – opportunities and challenges”, in 49th Annual Design Automation Conference, 2012, pp. 1153 - 1158.

[11] LUISA ANA. Anatomia e Fisiologia Humanas. http://www.afh.bio.br/nervoso/nervoso4.asp#SNP.

[12] MARKOVIC, D. “Ultralow-power design in near-threshold region”, in Proceedings of the IEEE, Vol. 98, 2010, pp. 237 - 252.

[13] NICOLELIS Miguel. Mind Out of Body. 2011.

[14] NEURO INFORMATICS 2012. http://www.neuroinformatics2012.org. 2012.

[15] PEDRONI, Volnei A. Circuit design and simulation with VHDL . 2004.

[16] SCHLINDWEIN, Fernando S. DFT / FFT, spectral estimation, some proofs and windows. 2012.

[17] SOARES, Leonardo; STANGHERLIN, Kleber; MELLO, Jorge de; BAMPI, Sergio. “61 pJ/sample Near-Threshold Notch Filter with Pole-Radius Variation”. Submetido ao Congresso IEEE LASCAS 2013.

[18] WANG, A.; CHANDRAKASAN, A. “A 180-mV subthreshold FFT processor using a minimum energy design methodology”, in IEEE Journal of Solid-State Circuits, Vol. 40, 2005, pp.310 - 319.

[19] WICHROWSKI, Jorge. www.youtube.com/watch?v=ZDleVX4qQWc .2012.

ANEXO A <Código da FFT em VHDL>

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;


entity fftsn is
port  (
       clk , reset : in std_logic;
       x0, x1, x2, x3, x4, x5, x6, x7 : in std_logic_vector(7 downto
0);
       F0_R, F0_I, F1_R, F1_I, F2_R, F2_I, F3_R, F3_I, F4_R, F4_I,
F5_R, F5_I, F6_R, F6_I, F7_R, F7_I : out std_logic_vector(15 downto
0);
             pronto: out std_logic);
end fftsn;


architecture behavior of fftsn is


 signal f0r, f0i, f1r, f1i, f2r, f2i, f3r, f3i, f4r, f4i, f5r, f5i,
f6r, f6i, f7r, f7i: std_logic_vector(15 downto 0);
 signal ctrl_inv: std_logic_vector(3 downto 0);


 type state is (state0, state1, state2, state3, state4);
       signal atual_estado : state;


  begin
  F0_R <= f0r;
  F0_I <= f0i;


  F1_R <= f1r;
  F1_I <= f1i;


  F2_R <= f2r;
  F2_I <= f2i;


  F3_R <= f3r;
  F3_I <= f3i;


  F4_R <= f4r;
  F4_I <= f4i;
```

```
 F5_R <= f5r;
 F5_I <= f5i;

 F6_R <= f6r;
 F6_I <= f6i;

 F7_R <= f7r;
 F7_I <= f7i;

process(clk)
      begin
      if reset = '1' then
               f0r <= "0000000000000000";
               f0i <= "0000000000000000";
               f1r <= "0000000000000000";
               f1i <= "0000000000000000";
               f2r <= "0000000000000000";
               f2i <= "0000000000000000";
               f3r <= "0000000000000000";
               f3i <= "0000000000000000";
               f4r <= "0000000000000000";
               f4i <= "0000000000000000";
               f5r <= "0000000000000000";
               f5i <= "0000000000000000";
               f6r <= "0000000000000000";
               f6i <= "0000000000000000";
               f7r <= "0000000000000000";
               f7i <= "0000000000000000";

               ctrl_inv(0) <= '0';
               ctrl_inv(1) <= '0';
               ctrl_inv(2) <= '0';
               ctrl_inv(3) <= '0';

               pronto <= '0';
               atual_estado <= state0;
      elsif (clk'event and clk = '1') then


             case atual_estado is
```

```
when  state0 =>

        pronto <= '0';
      ctrl_inv(0) <= '0';
        ctrl_inv(1) <= '0';
        ctrl_inv(2) <= '0';
        ctrl_inv(3) <= '0';

        if (x0(7) = '1') then
             f0r <= "11111111" & x0;
             f0i <= "0000000000000000";
      else
             f0r <= "00000000" & x0;
             f0i <= "0000000000000000";
      end if;

      if(x1(7) = '1') then
             f1r <= "11111111" & x1;
             f1i <= "0000000000000000";
      else
             f1r <= "00000000" & x1;
             f1i <= "0000000000000000";
      end if;

      if(x2(7) = '1') then
             f2r <= "11111111" & x2;
             f2i <= "0000000000000000";
      else
             f2r <= "00000000" & x2;
             f2i <= "0000000000000000";
      end if;

      if(x3(7) = '1') then
             f3r <= "11111111" & x3;
             f3i <= "0000000000000000";
      else
             f3r <= "00000000" & x3;
             f3i <= "0000000000000000";
      end if;
```

```
  if(x4(7) = '1') then
    f4r <= "11111111" & x4;
    f4i <= "0000000000000000";
  else
    f4r <= "00000000" & x4;
    f4i <= "0000000000000000";
  end if;

  if(x5(7) = '1') then
    f5r <= "11111111" & x5;
    f5i <= "0000000000000000";
  else
    f5r <= "00000000" & x5;
    f5i <= "0000000000000000";
  end if;

  if(x6(7) = '1') then
    f6r <= "11111111" & x6;
    f6i <= "0000000000000000";
  else
    f6r <= "00000000" & x6;
    f6i <= "0000000000000000";
  end if;

    if(x7(7) = '1') then
    f7r <= "11111111" & x7;
    f7i <= "0000000000000000";
  else
    f7r <= "00000000" & x7;
    f7i <= "0000000000000000";
  end if;

  atual_estado <= state1;

when state1 =>
        f0r <= f0r + f4r;
  f1r <= f0r - f4r;

  f2r <= f2r + f6r;
  f3r <= f2r - f6r;
```

```
    f4r <= f1r + f5r;
    f5r <= f1r - f5r;

    f6r <= f3r + f7r;
    f7r <= f3r - f7r;

    atual_estado <= state2;

  when state2 =>
    f0r <= f0r + f2r;

    f1r <= f1r;
    f1i <= "0000000000000000" - f3r;

    f2r <= f0r - f2r;

    f3r <= f1r;
    f3i <= f3r;

    f4r <= f4r + f6r;

    f5r <= f5r;
    f5i <= "0000000000000000" - f7r;

    f6r <= f4r - f6r;

    f7r <= f5r;
    f7i <= f7r;

    atual_estado <= state3;

  when state3 =>

    f0r <= f0r + f4r;
    f4r <= f0r - f4r;

    f2r <= f2r;
    f2i <= "0000000000000000" - f6r;
```

```
                        f6r <= f2r;
                        f6i <= "0000000000000000" + f6r;

                        if(f5r(15) = '1') then
                          ctrl_inv(0) <= '1';
                          f5r <= "0000000000000000" - f5r;
                        end if;

                        if(f5i(15) = '1') then
                          ctrl_inv(1) <= '1';
                          f5i <= "0000000000000000" - f5i;
                        end if;

                        if(f7r(15) = '1') then
                          ctrl_inv(2) <= '1';
                          f7r <= "0000000000000000" - f7r;
                        end if;

                        if(f7i(15) = '1') then
                          ctrl_inv(3) <= '1';
                          f7i <= "0000000000000000" - f7i;
                        end if;



                        atual_estado <= state4;

                      when state4 =>

                        pronto <= '1';

                        if(ctrl_inv(1) = '1' and ctrl_inv(0) = '1') then
                           f1r <= f1r - (('0'&f5r(15 downto 1)) +
("00"&f5r(15 downto 2)))  - (('0'&f5i(15 downto 1)) + ("00"&f5i(15
downto 2)));
                             f1i <= f1i +  (('0'&f5r(15 downto 1)) +
("00"&f5r(15 downto 2))) - (('0'&f5i(15 downto 1)) + ("00"&f5i(15
downto 2)));
                        elsif(ctrl_inv(1) = '1') then
                           f1r <= f1r +  (('0'&f5r(15 downto 1)) +
("00"&f5r(15 downto 2))) - (('0'&f5i(15 downto 1)) + ("00"&f5i(15
downto 2)));
                             f1i <= f1i -  (('0'&f5r(15 downto 1)) +
("00"&f5r(15 downto 2))) - (('0'&f5i(15 downto 1)) + ("00"&f5i(15
```

```
downto 2)));
                        elsif(ctrl_inv(0) = '1') then
                           f1r <= f1r -  (('0'&f5r(15 downto 1)) +
("00"&f5r(15 downto 2))) + (('0'&f5i(15 downto 1)) + ("00"&f5i(15
downto 2)));
                           f1i <= f1i +  (('0'&f5r(15 downto 1)) +
("00"&f5r(15 downto 2))) + (('0'&f5i(15 downto 1)) + ("00"&f5i(15
downto 2)));
                        else
                           f1r <= f1r +  (('0'&f5r(15 downto 1)) +
("00"&f5r(15 downto 2))) + (('0'&f5i(15 downto 1)) + ("00"&f5i(15
downto 2)));
                           f1i <= f1i -  (('0'&f5r(15 downto 1)) +
("00"&f5r(15 downto 2))) + (('0'&f5i(15 downto 1)) + ("00"&f5i(15
downto 2)));
                        end if;




                        if(ctrl_inv(1) = '1' and ctrl_inv(0) = '1') then
                           f5r <= f1r + (('0'&f5r(15 downto 1)) +
("00"&f5r(15 downto 2)))  + (('0'&f5i(15 downto 1)) + ("00"&f5i(15
downto 2)));
                           f5i <= f1i -  (('0'&f5r(15 downto 1)) +
("00"&f5r(15 downto 2))) + (('0'&f5i(15 downto 1)) + ("00"&f5i(15
downto 2)));
                        elsif(ctrl_inv(1) = '1') then
                           f5r <= f1r -  (('0'&f5r(15 downto 1)) +
("00"&f5r(15 downto 2))) + (('0'&f5i(15 downto 1)) + ("00"&f5i(15
downto 2)));
                           f5i <= f1i +  (('0'&f5r(15 downto 1)) +
("00"&f5r(15 downto 2))) + (('0'&f5i(15 downto 1)) + ("00"&f5i(15
downto 2)));
                        elsif(ctrl_inv(0) = '1') then
                           f5r <= f1r +  (('0'&f5r(15 downto 1)) +
("00"&f5r(15 downto 2))) - (('0'&f5i(15 downto 1)) + ("00"&f5i(15
downto 2)));
                           f5i <= f1i -  (('0'&f5r(15 downto 1)) +
("00"&f5r(15 downto 2))) - (('0'&f5i(15 downto 1)) + ("00"&f5i(15
downto 2)));
                        else
                           f5r <= f1r -  (('0'&f5r(15 downto 1)) +
("00"&f5r(15 downto 2))) - (('0'&f5i(15 downto 1)) + ("00"&f5i(15
downto 2)));
                           f5i <= f1i +  (('0'&f5r(15 downto 1)) +
("00"&f5r(15 downto 2))) - (('0'&f5i(15 downto 1)) + ("00"&f5i(15
downto 2)));
                        end if;
```

```
                    if(ctrl_inv(3) = '1' and ctrl_inv(2) = '1') then
                       f3r <= f3r + (('0'&f7r(15 downto 1)) +
("00"&f7r(15 downto 2)))  - (('0'&f7i(15 downto 1)) + ("00"&f7i(15
downto 2)));
                       f3i <= f3i + (('0'&f7r(15 downto 1)) +
("00"&f7r(15 downto 2)))  + (('0'&f7i(15 downto 1)) + ("00"&f7i(15
downto 2)));
                    elsif(ctrl_inv(3) = '1') then
                       f3r <= f3r - (('0'&f7r(15 downto 1)) +
("00"&f7r(15 downto 2)))  - (('0'&f7i(15 downto 1)) + ("00"&f7i(15
downto 2)));
                       f3i <= f3i - (('0'&f7r(15 downto 1)) +
("00"&f7r(15 downto 2)))  + (('0'&f7i(15 downto 1)) + ("00"&f7i(15
downto 2)));
                    elsif(ctrl_inv(2) = '1') then
                       f3r <= f3r + (('0'&f7r(15 downto 1)) +
("00"&f7r(15 downto 2)))  + (('0'&f7i(15 downto 1)) + ("00"&f7i(15
downto 2)));
                       f3i <= f3i + (('0'&f7r(15 downto 1)) +
("00"&f7r(15 downto 2)))  - (('0'&f7i(15 downto 1)) + ("00"&f7i(15
downto 2)));
                    else
                       f3r <= f3r - (('0'&f7r(15 downto 1)) +
("00"&f7r(15 downto 2)))  + (('0'&f7i(15 downto 1)) + ("00"&f7i(15
downto 2)));
                       f3i <= f3i - (('0'&f7r(15 downto 1)) +
("00"&f7r(15 downto 2)))  - (('0'&f7i(15 downto 1)) + ("00"&f7i(15
downto 2)));
                    end if;



                    if(ctrl_inv(3) = '1' and ctrl_inv(2) = '1') then
                       f7r <= f3r  - (('0'&f7r(15 downto 1)) +
("00"&f7r(15 downto 2)))   + (('0'&f7i(15 downto 1)) + ("00"&f7i(15
downto 2)));
                       f7i <= f3i  - (('0'&f7r(15 downto 1)) +
("00"&f7r(15 downto 2)))    - (('0'&f7i(15 downto 1)) + ("00"&f7i(15
downto 2)));
                    elsif(ctrl_inv(3) = '1') then
                       f7r <= f3r  + (('0'&f7r(15 downto 1)) +
("00"&f7r(15 downto 2)))   + (('0'&f7i(15 downto 1)) + ("00"&f7i(15
downto 2)));
                       f7i <= f3i + (('0'&f7r(15 downto 1)) +
("00"&f7r(15 downto 2)))    - (('0'&f7i(15 downto 1)) + ("00"&f7i(15
downto 2)));
                    elsif(ctrl_inv(2) = '1') then
                       f7r <= f3r  - (('0'&f7r(15 downto 1)) +
("00"&f7r(15 downto 2)))   - (('0'&f7i(15 downto 1)) + ("00"&f7i(15
```

```
downto 2)));
                        f7i <= f3i - (('0'&f7r(15 downto 1)) +
("00"&f7r(15 downto 2)))    + (('0'&f7i(15 downto 1)) + ("00"&f7i(15
downto 2)));
                    else
                        f7r <= f3r  + (('0'&f7r(15 downto 1)) +
("00"&f7r(15 downto 2)))   - (('0'&f7i(15 downto 1)) + ("00"&f7i(15
downto 2)));
                        f7i <= f3i + (('0'&f7r(15 downto 1)) +
("00"&f7r(15 downto 2)))    + (('0'&f7i(15 downto 1)) + ("00"&f7i(15
downto 2)));
                    end if;



                    atual_estado <= state0;


                end case;
            end if;
 end process;
end behavior;
```

ANEXO B <Testbench básico para verificar funcionalidade>

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY fftsn_tb IS
END fftsn_tb;

ARCHITECTURE tb OF fftsn_tb IS



COMPONENT fftsn --IS
port  (
    clk , reset : in std_logic;
    x0, x1 , x2, x3, x4, x5 , x6 , x7 : in std_logic_vector(7 downto
0);
    F0_R, F0_I, F1_R, F1_I, F2_R, F2_I, F3_R, F3_I, F4_R, F4_I, F5_R,
F5_I, F6_R, F6_I, F7_R, F7_I : out std_logic_vector(15 downto 0);
    pronto : out std_logic);
END COMPONENT;


signal clk , reset :  std_logic;
signal x0, x1 , x2, x3, x4, x5 , x6 , x7 : std_logic_vector(7 downto
0);
signal F0_R, F0_I, F1_R, F1_I, F2_R, F2_I, F3_R, F3_I, F4_R, F4_I,
F5_R, F5_I, F6_R, F6_I, F7_R, F7_I : std_logic_vector(15 downto 0);
signal pronto: std_logic;


BEGIN


  DUT : fftsn PORT MAP (clk=>clk, reset=>reset, x0=>x0, x1=>x1,
x2=>x2, x3=>x3, x4=>x4, x5=>x5, x6=>x6, x7=>x7,
      F0_R=>F0_R,  F0_I=>F0_I,  F1_R=>F1_R,  F1_I=>F1_I,  F2_R=>F2_R,
F2_I=>F2_I,    F3_R=>F3_R,    F3_I=>F3_I,    F4_R=>F4_R,    F4_I=>F4_I,
F5_R=>F5_R,    F5_I=>F5_I,    F6_R=>F6_R,    F6_I=>F6_I,    F7_R=>F7_R,
F7_I=>F7_I, pronto => pronto);


  Generate_clk: PROCESS
  begin
    clk <= '0';
    wait for 500 ns;
    clk <= '1';
    wait for 500 ns;
  end process Generate_clk;
```

```
Reseting: PROCESS
begin
	reset <= '1';

	x0	<= "00000010";
	x4	<= "00000100";

	x2	<= "00000011";
	x6	<= "00000101";

	x1	<= "00010100";
	x5	<= "00000010";

	x3	<= "00000110";
	x7	<= "00000001";

	wait for 100 us;

	reset <= '0';

	wait for 10 us;

	x0	<= "11111100";
	x4	<= "00011000";

	x2	<= "00100000";
	x6	<= "00101101";

	x1	<= "10000000";
	x5	<= "01111111";

	x3	<= "00000010";
	x7	<= "11111111";


	wait for 1 ms;


	ASSERT false REPORT "Test Complete";
end process Reseting;
```

ANEXO C <Gerador de Testbench de 500 valores aleatórios>

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>


   FILE *pFile22;
   char binario[8];


int main(int argc, char *argv[])
{

   srand ( time(NULL) );
   int aleatorio = 0;



   pFile22 = fopen ( "fftsn_tb.vhd", "w");
   if (pFile22 == NULL) perror ("Error opening file");
   else {

       fprintf (pFile22," LIBRARY IEEE;\n" );
       fprintf (pFile22," USE IEEE.std_logic_1164.all;\n\n\n" );
       fprintf (pFile22," ENTITY fftsn_tb IS\n" );
       fprintf (pFile22," END fftsn_tb;\n\n\n" );
       fprintf (pFile22," ARCHITECTURE tb OF fftsn_tb IS\n\n" );
       fprintf (pFile22," COMPONENT fftsn \n" );
       fprintf (pFile22," port  (\n" );
       fprintf (pFile22,"        clk , reset : in std_logic;\n" );
       fprintf (pFile22,"            x0, x1 , x2, x3, x4, x5 , x6 , x7 : in
std_logic_vector(7 downto 0);\n" );
       fprintf (pFile22,"        F0_R, F0_I, F1_R, F1_I, F2_R, F2_I, F3_R,
F3_I, F4_R, F4_I, F5_R, F5_I, F6_R, F6_I, F7_R, F7_I : out
std_logic_vector(15 downto 0);\n" );
       fprintf (pFile22,"        pronto : out std_logic);\n" );
       fprintf (pFile22," END COMPONENT; \n\n\n" );
       fprintf (pFile22," signal clk , reset :  std_logic;\n" );
       fprintf (pFile22," signal x0, x1 , x2, x3, x4, x5 , x6 , x7 :
std_logic_vector(7 downto 0);\n" );
       fprintf (pFile22," signal F0_R, F0_I, F1_R, F1_I, F2_R, F2_I, F3_R,
F3_I, F4_R, F4_I, F5_R, F5_I, F6_R, F6_I, F7_R, F7_I : std_logic_vector(15
downto 0);\n" );
       fprintf (pFile22," signal pronto: std_logic;\n\n" );
       fprintf (pFile22," BEGIN \n" );
       fprintf (pFile22," DUT : fftsn PORT MAP (clk=>clk, reset=>reset,
x0=>x0, x1=>x1, x2=>x2, x3=>x3, x4=>x4, x5=>x5, x6=>x6, x7=>x7, F0_R=>F0_R,
F0_I=>F0_I, F1_R=>F1_R, F1_I=>F1_I, F2_R=>F2_R, F2_I=>F2_I, F3_R=>F3_R,
F3_I=>F3_I, F4_R=>F4_R, F4_I=>F4_I, F5_R=>F5_R, F5_I=>F5_I, F6_R=>F6_R,
F6_I=>F6_I, F7_R=>F7_R, F7_I=>F7_I, pronto => pronto); \n\n\n" );
       fprintf (pFile22," Generate_clk: PROCESS \n" );
       fprintf (pFile22,"   begin \n" );
       fprintf (pFile22,"     clk <= '0'; \n" );
       fprintf (pFile22,"     wait for 500 ns; \n" );
       fprintf (pFile22,"     clk <= '1'; \n" );
       fprintf (pFile22,"     wait for 500 ns; \n" );
       fprintf (pFile22," end process Generate_clk;\n\n\n" );
       fprintf (pFile22," Reseting: PROCESS \n");
       fprintf (pFile22,"   begin \n" );
             fprintf (pFile22,"     reset <= '1';\n");
```

```
        fprintf (pFile22,"     wait for 100 us;\n");
        fprintf (pFile22,"     reset <= '0';\n\n\n");

        int i = 0;
        while (i <= 500)
        {
               aleatorio = rand() % 10;
               //itoa(aleatorio,binario,2); //Converte para base 2
               //fprintf (pFile22," x <= %s; \n" , binario);
               fprintf (pFile22," x0 <= \"%d%d%d%d%d%d%d%d\"; \n" , rand()%2 ,
rand()%2, rand()%2, rand()%2, rand()%2 , rand()%2, rand()%2, rand()%2);
               fprintf (pFile22," x1 <= \"%d%d%d%d%d%d%d%d\"; \n" , rand()%2 ,
rand()%2, rand()%2, rand()%2, rand()%2 , rand()%2, rand()%2, rand()%2);
               fprintf (pFile22," x2 <= \"%d%d%d%d%d%d%d%d\"; \n" , rand()%2 ,
rand()%2, rand()%2, rand()%2, rand()%2 , rand()%2, rand()%2, rand()%2);
               fprintf (pFile22," x3 <= \"%d%d%d%d%d%d%d%d\"; \n" , rand()%2 ,
rand()%2, rand()%2, rand()%2, rand()%2 , rand()%2, rand()%2, rand()%2);
               fprintf (pFile22," x4 <= \"%d%d%d%d%d%d%d%d\"; \n" , rand()%2 ,
rand()%2, rand()%2, rand()%2, rand()%2 , rand()%2, rand()%2, rand()%2);
               fprintf (pFile22," x5 <= \"%d%d%d%d%d%d%d%d\"; \n" , rand()%2 ,
rand()%2, rand()%2, rand()%2, rand()%2 , rand()%2, rand()%2, rand()%2);
               fprintf (pFile22," x6 <= \"%d%d%d%d%d%d%d%d\"; \n" , rand()%2 ,
rand()%2, rand()%2, rand()%2, rand()%2 , rand()%2, rand()%2, rand()%2);
               fprintf (pFile22," x7 <= \"%d%d%d%d%d%d%d%d\"; \n" , rand()%2 ,
rand()%2, rand()%2, rand()%2, rand()%2 , rand()%2, rand()%2, rand()%2);

               fprintf (pFile22," wait for 10 us;\n\n");
               i = i + 1;
        }




        fprintf (pFile22," ASSERT false REPORT \"Test Complete\"; \n");
        fprintf (pFile22," end process Reseting; \n");
        fprintf (pFile22," END tb;\n");

        fclose (pFile22);

    }

    return 0;



}
```

ANEXO D <Arquivo fonte em linguagem Matlab que gera os gráficos deste trabalho>

```
t0p45 = [-40,0,25,80];
p0p45 = [7.6,5.5, 5.0, 6.4];
figure(1);
plot(t0p45,p0p45);
hold on;
plot(t0p45,p0p45,'k*');
plot (-40,7.6,'r*');
hold on;
t1p20 = [-40,0,25,80];
p1p20 = [26.6, 27.0, 28.5 33.8];
plot(t1p20,p1p20,'r');
hold on;
plot(t1p20,p1p20,'k*');
title('Potência X Temperatura (pelo testbench com VCD file)')
xlabel('C°')
ylabel('uW')

t0p45a = [-40,0,25,80];
p0p45a = [76.8,68.2, 62.5, 47.5];
figure(2);
plot(t0p45a,p0p45a);
hold on;
plot(t0p45a,p0p45a,'k*');
hold on;
t1p20a = [-40,0,25,80];
p1p20a = [243.4, 239.8, 249.4 255.4];
plot(t1p20a,p1p20a,'r');
hold on;
plot(t1p20a,p1p20a,'k*');
title('Potência X Temperatura (pelo rtl-compiler)')
xlabel('C°')
ylabel('uW')


p = [0.45,0.46,0.48,0.54,0.75,0.9,1.2];
v = [5.0,5.0, 6.4, 7.5,10.2,15.4,28.5];
figure(3);
plot(p,v,'g');
```

```
hold on;
plot(p,v,'k*');
hold on;
p1 = [0.45,0.46,0.48,0.54,0.75,0.9,1.2];
v1 = [62.5,62.1, 47.2, 61.0,93.5,132.2,249.4];
plot(p1,v1,'g');
hold on;
plot(p1,v1,'k*');
title('Potência X Tensão')
xlabel('V')
ylabel('uW')



v = [0.45,0.46,0.48,0.54,0.75,0.9,1.2];
nc = [18081,17609,18222, 17728,9519,9227,9381];
figure(4);
bar(v,nc,'g');
title('Instâncias de Células X Tensão')
xlabel('V')

p = [0.45,0.46,0.48,0.54,0.75,0.9,1.2];
ts = [187010,219846,235502,274450,294255,296499,297994];
figure(5);
bar(p,ts,'r');
title('Timing Slack x Tensão')
xlabel('V')
ylabel('ps')

t  = [-40,0,25,80];
nc = [16861,17650,18081,20011];
figure(6);
bar(t,nc,'b');
title('Instâncias de Células X Temperatura em 0.45V')
xlabel('V')

t  = [-40,0,25,80];
ts = [-1094045,1529,187010,242119];
figure(7);
```

```
bar(t,ts,'g');
title('Timing Slack X Temperatura em 0.45V')
xlabel('Cº')
ylabel('ps')



t  = [-40,0,25,80];
nc = [9165,9433,9381,9390];
figure(8);
bar(t,nc,'r');
title('Instâncias de Células X Temperatura em 1.20V')
xlabel('V')

t  = [-40,0,25,80];
ts = [297834,297901,297994,297913];
figure(9);
bar(t,ts,'b');
title('Timing Slack X Temperatura em 1.20V')
xlabel('Cº')
ylabel('ps')



figure(10);
t  = [-40,0,25,80];
ts = [297834,297901,297994,297913];
bar(t,ts,'r');
hold on;
ta = [-1094045,1529,187010,242119];
bar(t,ta,'b');
title('Timing Slack X Temperatura em 0.45V blue e 1.20 red.')
xlabel('Cº')
ylabel('ps')
```

ANEXO E <Arquivo Spice com as células básicas e suas capacitâncias>

```
********************************************************************************
.SUBCKT INVX1 VDD VSS A Y
*
*
*  caps2d version: 10
*
*
*       TRANSISTOR CARDS
*
*
MT1    Y     A     VSS   VSS   nfet  L=0.06U     W=0.15U
+ rgatemod=1 ptwell=0 par=1 nrs=0.986395 nrd=0.986395 nf=1 mSwitch=0
m=1 bentgate=0
MT0    Y     A     VDD   VDD   pfet  L=0.06U     W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1  bentgate=0
*
*
*       CAPACITOR CARDS
*
*
C1     VDD   VSS   8.669E-16
C2     A     VSS   9.238E-16
C3     Y     VSS   3.518E-16
*
*
.ENDS INVX1
********************************************************************************
.SUBCKT NAND2X1 VDD VSS A B Y
*
*
*  caps2d version: 10
*
*
*       TRANSISTOR CARDS
*
*
MT1    Y     B     net036      VSS   nfet  L=0.06U     W=0.15U
+ rgatemod=1 ptwell=0 par=1 nrs=0.986395 nrd=0.986395 nf=1 mSwitch=0
m=1  bentgate=0
```

```
MT0    net036      A     VSS   VSS   nfet  L=0.06U     W=0.15U
+ rgatemod=1 ptwell=0 par=1 nrs=0.986395 nrd=0.986395 nf=1 mSwitch=0
m=1  bentgate=0
MT3    Y     A     VDD   VDD   pfet  L=0.06U     W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1  bentgate=0
MT2    Y     B     VDD   VDD   pfet  L=0.06U     W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1  bentgate=0
*
*
*       CAPACITOR CARDS
*
*
C1     VDD   VSS   1.703E-15
C2     A     VSS   8.272E-16
C3     B     VSS   8.697E-16
C4     Y     VSS   5.925E-16
*
*
.ENDS NAND2X1
********************************************************************************
.SUBCKT NOR2X1 VDD VSS A B Y
*
*
*  caps2d version: 10
*
*
*       TRANSISTOR CARDS
*
*
MT2    Y     A     VSS   VSS   nfet  L=0.06U     W=0.15U
+ rgatemod=1 ptwell=0 par=1 nrs=0.986395 nrd=0.986395 nf=1 mSwitch=0
m=1  bentgate=0
MT3    Y     B     VSS   VSS   nfet  L=0.06U     W=0.15U
+ rgatemod=1 ptwell=0 par=1 nrs=0.986395 nrd=0.986395 nf=1 mSwitch=0
m=1  bentgate=0
MT1    net15 B     VDD   VDD   pfet  L=0.06U     W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1  bentgate=0
MT1@1  VDD   B     net15 VDD   pfet  L=0.06U     W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1  bentgate=0
```

```
MT0    Y     A     net15 VDD   pfet  L=0.06U     W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1  bentgate=0
MT0@1  net15 A     Y     VDD   pfet  L=0.06U     W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1  bentgate=0
*
*
*       CAPACITOR CARDS
*
*
C1     VDD   VSS   2.021E-15
C2     A     VSS   9.294E-16
C3     B     VSS   9.135E-16
C4     Y     VSS   5.374E-16
C5     net15 VSS   7.094E-16
*
*
.ENDS NOR2X1
********************************************************************************
.SUBCKT AND2X1 VDD VSS A B Y
*
*
*  caps2d version: 10
*
*
*       TRANSISTOR CARDS
*
*
MT0    net16 A     VSS   VSS   nfet  L=0.06U     W=0.15U
+ rgatemod=1 ptwell=0 par=1 nrs=0.986395 nrd=0.986395 nf=1 mSwitch=0
m=1  bentgate=0
MT1    net12 B     net16 VSS   nfet  L=0.06U     W=0.15U
+ rgatemod=1 ptwell=0 par=1 nrs=0.986395 nrd=0.986395 nf=1 mSwitch=0
m=1  bentgate=0
MT4    Y     net12 VSS   VSS   nfet  L=0.06U     W=0.15U
+ rgatemod=1 ptwell=0 par=1 nrs=0.986395 nrd=0.986395 nf=1 mSwitch=0
m=1  bentgate=0
MT3    net12 A     VDD   VDD   pfet  L=0.06U     W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1  bentgate=0
MT2    net12 B     VDD   VDD   pfet  L=0.06U     W=0.225U
```

```
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1  bentgate=0
MT5    Y     net12 VDD   VDD   pfet  L=0.06U     W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1  bentgate=0
*
*
*       CAPACITOR CARDS
*
*
C1     VDD   VSS   2.509E-15
C2     A     VSS   8.266E-16
C3     B     VSS   8.608E-16
C4     Y     VSS   3.119E-16
C5     net12 VSS   1.396E-15
*
*
.ENDS AND2X1
********************************************************************************
.SUBCKT OR2X1 VDD VSS A B Y
*
*
*  caps2d version: 10
*
*
*       TRANSISTOR CARDS
*
*
MT3    net16 B     VSS   VSS   nfet  L=0.06U     W=0.15U
+ rgatemod=1 ptwell=0 par=1 nrs=0.986395 nrd=0.986395 nf=1 mSwitch=0
m=1  bentgate=0
MT2    net16 A     VSS   VSS   nfet  L=0.06U     W=0.15U
+ rgatemod=1 ptwell=0 par=1 nrs=0.986395 nrd=0.986395 nf=1 mSwitch=0
m=1  bentgate=0
MT4    Y     net16 VSS   VSS   nfet  L=0.06U     W=0.15U
+ rgatemod=1 ptwell=0 par=1 nrs=0.986395 nrd=0.986395 nf=1 mSwitch=0
m=1  bentgate=0
MT1    net27 B     VDD   VDD   pfet  L=0.06U     W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1  bentgate=0
MT1@1  VDD   B     net27 VDD   pfet  L=0.06U     W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1  bentgate=0
```

```
MT0    net16 A     net27 VDD   pfet  L=0.06U     W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1  bentgate=0
MT0@1  net27 A     net16 VDD   pfet  L=0.06U     W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1  bentgate=0
MT5    Y     net16 VDD   VDD   pfet  L=0.06U     W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1  bentgate=0
*
*
*       CAPACITOR CARDS
*
*
C1     VDD   VSS   2.809E-15
C2     A     VSS   9.192E-16
C3     B     VSS   9.141E-16
C4     Y     VSS   3.017E-16
C5     net16 VSS   1.328E-15
C6     net27 VSS   7.284E-16
*
*
.ENDS OR2X1

********************************************************************************

********************************************************************************
*
.SUBCKT DFFRHQ VDD VSS CLK D RST Q QN
*
*
*  caps2d version: 10
*
*
*       TRANSISTOR CARDS
*
*
MT27   net0174     clkgat2     VSS   VSS   nfet  L=0.06U
+ W=0.15U
+ rgatemod=1 ptwell=0 par=1 nrs=0.986395 nrd=0.986395 nf=1 mSwitch=0
m=1  bentgate=0
MT9    b2    NCLK  net0174     VSS   nfet  L=0.06U     W=0.15U
```

```
+ rgatemod=1 ptwell=0 par=1 nrs=0.986395 nrd=0.986395 nf=1 mSwitch=0
m=1   bentgate=0

MT3    clkgat2     b2    VSS   VSS   nfet  L=0.06U     W=0.15U

+ rgatemod=1 ptwell=0 par=1 nrs=0.986395 nrd=0.986395 nf=1 mSwitch=0
m=1   bentgate=0

MT2    clkgat2     net13 VSS   VSS   nfet  L=0.06U     W=0.15U

+ rgatemod=1 ptwell=0 par=1 nrs=0.986395 nrd=0.986395 nf=1 mSwitch=0
m=1   bentgate=0

MT4    QN    clkgat2     VSS   VSS   nfet  L=0.06U     W=0.15U

+ rgatemod=1 ptwell=0 par=1 nrs=0.986395 nrd=0.986395 nf=1 mSwitch=0
m=1   bentgate=0

MT32   Q     QN    VSS   VSS   nfet  L=0.06U     W=0.15U

+ rgatemod=1 ptwell=0 par=1 nrs=0.986395 nrd=0.986395 nf=1 mSwitch=0
m=1   bentgate=0

MT10   b2    CLK   net0215     VSS   nfet  L=0.06U     W=0.15U

+ rgatemod=1 ptwell=0 par=1 nrs=0.986395 nrd=0.986395 nf=1 mSwitch=0
m=1   bentgate=0

MT28   net0215     clkgat1     VSS   VSS   nfet  L=0.06U

+ W=0.15U

+ rgatemod=1 ptwell=0 par=1 nrs=0.986395 nrd=0.986395 nf=1 mSwitch=0
m=1   bentgate=0

MT6    net0194     D     VSS   VSS   nfet  L=0.06U     W=0.15U

+ rgatemod=1 ptwell=0 par=1 nrs=0.986395 nrd=0.986395 nf=1 mSwitch=0
m=1   bentgate=0

MT21   net13 RST   VSS   VSS   nfet  L=0.06U     W=0.15U

+ rgatemod=1 ptwell=0 par=1 nrs=0.986395 nrd=0.986395 nf=1 mSwitch=0
m=1   bentgate=0

MT22   NCLK  CLK   VSS   VSS   nfet  L=0.06U     W=0.15U

+ rgatemod=1 ptwell=0 par=1 nrs=0.986395 nrd=0.986395 nf=1 mSwitch=0
m=1   bentgate=0

MT12   b1    NCLK  net0194     VSS   nfet  L=0.06U     W=0.15U

+ rgatemod=1 ptwell=0 par=1 nrs=0.986395 nrd=0.986395 nf=1 mSwitch=0
m=1   bentgate=0

MT25   net0178     clkgat1     VSS   VSS   nfet  L=0.06U

+ W=0.15U

+ rgatemod=1 ptwell=0 par=1 nrs=0.986395 nrd=0.986395 nf=1 mSwitch=0
m=1   bentgate=0

MT14   b1    CLK   net0178     VSS   nfet  L=0.06U     W=0.15U

+ rgatemod=1 ptwell=0 par=1 nrs=0.986395 nrd=0.986395 nf=1 mSwitch=0
m=1   bentgate=0

MT19   clkgat1     b1    VSS   VSS   nfet  L=0.06U     W=0.15U

+ rgatemod=1 ptwell=0 par=1 nrs=0.986395 nrd=0.986395 nf=1 mSwitch=0
m=1   bentgate=0

MT18   clkgat1     net13 VSS   VSS   nfet  L=0.06U     W=0.15U

+ rgatemod=1 ptwell=0 par=1 nrs=0.986395 nrd=0.986395 nf=1 mSwitch=0
```

```
m=1   bentgate=0
MT11   b2    NCLK  net0215     VDD   pfet  L=0.06U
+ W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1   bentgate=0
MT26   net0174     clkgat2     VDD   VDD   pfet  L=0.06U
+ W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1   bentgate=0
MT8    b2    CLK   net0174     VDD   pfet  L=0.06U
+ W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1   bentgate=0
MT1    net100      b2    VDD   VDD   pfet  L=0.06U     W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1   bentgate=0
MT1@1  VDD   b2    net100      VDD   pfet  L=0.06U     W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1   bentgate=0
MT0    clkgat2     net13 net100      VDD   pfet  L=0.06U
+ W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1   bentgate=0
MT0@1  net100      net13 clkgat2     VDD   pfet  L=0.06U
+ W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1   bentgate=0
MT5    QN    clkgat2     VDD   VDD   pfet  L=0.06U
+ W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1   bentgate=0
MT31   Q     QN    VDD   VDD   pfet  L=0.06U     W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1   bentgate=0
MT29   net0215     clkgat1     VDD   VDD   pfet  L=0.06U
+ W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1   bentgate=0
MT7    net0194     D     VDD   VDD   pfet  L=0.06U
+ W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1   bentgate=0
MT20   net13 RST   VDD   VDD   pfet  L=0.06U     W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
```

```
m=1   bentgate=0
MT23   NCLK  CLK   VDD   VDD   pfet  L=0.06U     W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1   bentgate=0
MT13   b1    CLK   net0194     VDD   pfet  L=0.06U
+ W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1   bentgate=0
MT24   net0178     clkgat1     VDD   VDD   pfet  L=0.06U
+ W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1   bentgate=0
MT15   b1    NCLK  net0178     VDD   pfet  L=0.06U
+ W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1   bentgate=0
MT17   net72 b1    VDD   VDD   pfet  L=0.06U     W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1   bentgate=0
MT17@1 VDD   b1    net72 VDD   pfet  L=0.06U     W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1   bentgate=0
MT16   clkgat1     net13 net72 VDD   pfet  L=0.06U
+ W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1   bentgate=0
MT16@1 net72 net13 clkgat1     VDD   pfet  L=0.06U
+ W=0.225U
+ rgatemod=1 ptwell=0 par=1 nrs=0.653153 nrd=0.653153 nf=1 mSwitch=0
m=1   bentgate=0
*
*
*       CAPACITOR CARDS
*
*
C1     VDD   VSS   1.216E-14
C2     CLK   VSS   3.239E-15
C3     D     VSS   5.431E-16
C4     RST   VSS   5.413E-16
C5     Q     VSS   3.016E-16
C6     QN    VSS   1.028E-15
C7     net0215     VSS   6.192E-16
C8     net0174     VSS   6.138E-16
```

```
C9      net0178      VSS    6.122E-16
C10     net0194      VSS    9.401E-16
C11     b1     VSS    2.144E-15
C12     net72 VSS    8.021E-16
C13     clkgat1      VSS    2.626E-15
C14     NCLK   VSS    2.798E-15
C15     b2     VSS    2.346E-15
C16     clkgat2      VSS    2.467E-15
C17     net100       VSS    7.621E-16
C18     net13 VSS    3.491E-15
*
*
.ENDS DFFRHQ
*
********************************************************************************
**
```

ANEXO F <Paper Submetido ao Congresso IEEE Latin American Symposium on Circuits and Systems. Uso da Biblioteca Std Cells em Near-Threshold –“ 61 pJ/sample Near-Threshold Notch Filter with Pole-Radius Variation” >

61 pJ/sample Near-Threshold Notch Filter with Pole-Radius Variation

Leonardo Soares[1]; Kleber Stangherlin[2]
PGMicro[1]/PPGC[2] – Informatics Institute
Federal University of Rio Grande do Sul
Porto Alegre, Brazil
{lbsoares, khstangherlin}@inf.ufrgs.br

Jorge de Mello; Sergio Bampi
Informatics Institute
Federal University of Rio Grande do Sul
Porto Alegre, Brazil

Abstract—**This paper presents results of digital CMOS design for ultra-low power filter that uses logic cells operating at near-threshold voltage supplies. The cells were designed in 65nm CMOS technology for ultra-low power. The hardware implementation of a pole-radius-varying Infinite Impulse Response (IIR) notch filter is addressed. Previous works have shown that digital design of pole-radius-varying (or Q factor-varying) IIR notch filters can suppress the transient effect. The proposed filter is synthesized using our custom 65 nm near-threshold voltage standard cells library. Comparisons from near-threshold operation to nominal voltages show large gains in energy-savings for this filter that make the logic architecture suitable for sub-nano-Joule per sample applications.**

I. Introduction

IIR digital notch filter has been used to reject narrow frequency bands in many applications. A common environment that needs the use of this approach is the AC power supply interference extraction from monitored electroencephalography (EEG) signals. The analyzed data is a mixture of EEG signal and 60/50 Hz interference. Since the brain activity produces frequencies ranging from 1.5Hz to 80Hz (from delta-frequency to gamma-frequency) [1], then a reject narrow band filter can be implemented to eliminate the interference.

The design of digital notch filters has been proposed for a long time [2]-[11]. References [2] and [3] performed an explanation about the basic theory of digital notch filters. Specifically in [3], a few configurations were proposed in order to reduce the number of multiplications. In fact, there are many possibilities to improve the implementation of digital IIR notch filters. This brings efforts to enhance features like transient response, stability and quantization sensitivity [4]-[11]. The transient effect is a result of the pole-radius choice. When the pole-radius gets larger, then the rejected frequency band turns out to be narrower. However, for narrower rejected bands the transient response has relevant magnitude and takes more time duration. In [10] and [11] a Q factor and pole-radius-varying were proposed respectively. The major gain is in transient effect suppression reducing the time duration. That is because when the pole-radius value is dynamically changed, then the transient is initially suppressed and after that the rejected band turns out to be narrower.

Regarding the ultra low-power point of view required for the medical and health-care applications, the solution explored in this work focus on voltage scaling down to the near-threshold region. It's well known that the minimum energy operating point relies in the sub-threshold region [13] - [14], however the huge performance penalties raised by weak inversion result in a low energy-efficiency. Thus, to achieve best energy-efficiency [15] - [16], we propose a circuit able to work in moderate inversion, i.e. at near-threshold voltages.

In this paper we highlight in our proposal the CMOS design of a digital second-order pole-radius-varying IIR notch filter that uses logic cells operating in near-threshold voltage. As a case study we applied the notch filter for the extraction of interference in EEG signals.

This paper is organized as follows: Section II overviews from the basic theory for the pole-radius-varying model for design of digital notch filters. Section III presents the proposed filter hardware and more information about our 65nm near-threshold voltage standard cells library project. Section IV shows results for the notch filter and logic synthesis. Finally, conclusions are drawn in Section V.

II. Digital IIR Notch Filter Design

Digital IIR Notch Filter's Transfer Function is derived from two parameters: The digital notch normalized angle θ_0 (1) and the digital -3dB reject bandwidth b (2).

$$\theta_0 = \frac{2\pi f_0}{f_s}\ [rad] \quad (1)$$

$$b = \frac{2\pi \Delta f}{f_s} \quad (2)$$

In (1) f_0 is the center frequency in Hz and in (2) Δf is ideal bandwidth in Hz for the rejected band. The sampled frequency is denoted by f_s.

Identify applicable sponsor/s here. *(sponsors)*

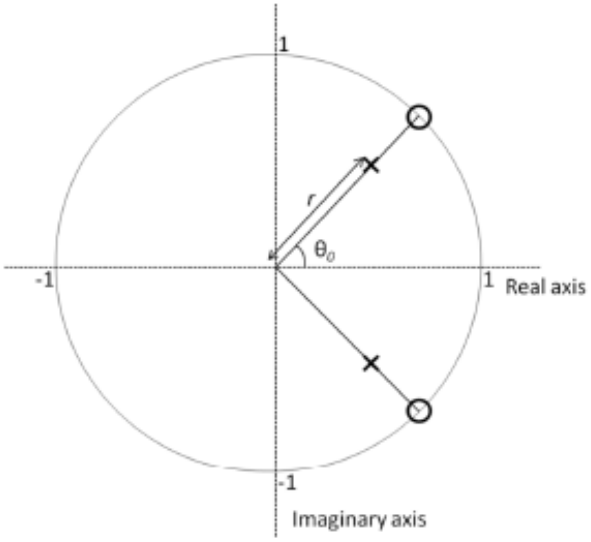


Figure 1. Poles and Zeros placement on the z-plane

The quality factor Q is then derived as in (3):

$$Q = \frac{f_0}{\Delta f} \tag{3}$$

It is easy to see that the higher Q factor is, the narrower the notch filter. That is the relation between Q factor and the narrow band reject adjustment. Hence, a way to suppress the transient response is varying the Q.

Moreover, the relation between Q factor and the pole-radius r is defined as in (4) [11].

$$Q \approx \frac{\pi f_0}{(1-r) f_s} \tag{4}$$

Therefore, as the r gets larger, the narrower the rejected band is. This brings that is possible to control the transient effect by varying r.

Based on the parameters previously defined, the second-order IIR digital notch filter's Transfer Function $H(z)$ is presented as in (5) [11]:

$$H(z) = \frac{1 - 2\cos(\theta_0)\, z^{-1} + z^{-2}}{1 - 2r\cos(\theta_0) z^{-1} + r^2 z^{-2}} \tag{5}$$

The zeros and poles of $H(z)$ are placed at the z-plane as shown in Fig. 1.

From Fig.1 is possible to identify the digital notch normalized angle θ_0 and the pole-radius r. Since is required a narrow band reject, then the poles must be placed near to the unit-circle (large values for r). However, this brings to the transient effect earlier discussed. Therefore, reference [11] developed an approach to deal with this problem: the pole-radius-varying technique. This r variation can be taken as proposed by [11] in (6):

$$r(n) = (r - \Delta r e^{-\alpha n}) \tag{6}$$

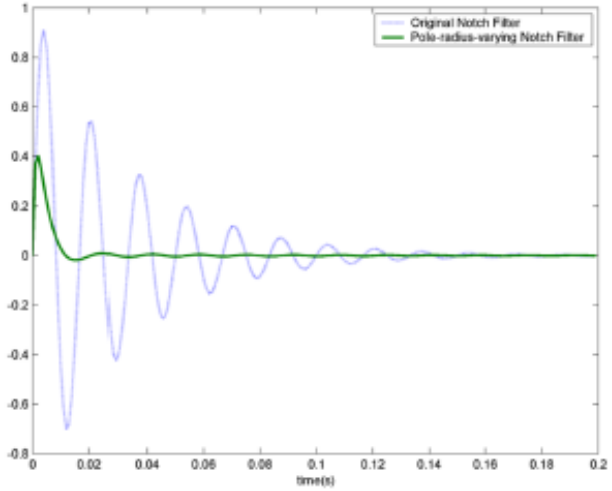


Figure 2. Transient response comparison between techniques

Where n is the sample index and $r(n)$ is the pole-radius value at the n^{th} sample. The r value is ranging from $(r - \Delta r)$ to r and α is the damping factor.

This change in r produces a modification in the original Transfer Function and the new difference equation is given by (7) [11]:

$$y(n) = x(n) - 2\cos(\theta_0)\, x(n-1) + x(n-2) + 2\cos(\theta_0)\, r(n) y(n-1) - r(n) r(n-1) y(n-2) \tag{7}$$

Notice from (7) that the product between $r(n)$ and $r(n\text{-}1)$ replaces the coefficient r^2 in (5).

In order to verify the transient response suppression with this technique, we applied the original IIR notch filter and the pole-radius-varying one to a unit amplitude 60 Hz sinusoidal signal sampled at 1 kHz. The original filter has a fixed r of 0.97 while the latter technique has the pole-radius ranging from 0.49 to 0.99 and a damping factor α of 0.1. Fig. 2 shows the resulting transient responses for both filters. As can be seen the transient effect is significantly attenuated by the latter technique.

III. Hardware Implementation and Near-Threshold Voltage Library Project

This section presents the hardware design and implementation of the pole-radius-varying notch filter.

A. Hardware Implementation

We propose a hardware design derived from the transposed direct-form II structure for second-order IIR filters (see Fig.3). In Figure 3 b_0 to b_2 and a_1 to a_2 represent the coefficients from numerator and denominator polynomial, defined in equation (5), respectively. Notice from the same equation that some simplifications can be taken to reduce the number of multipliers in the structure. That is because b_0 and b_2 has fixed unitary value.

Furthermore, the delay elements Z^1 and Z^2 are represented in hardware by registers and an additional output register is placed in $y[n]$.

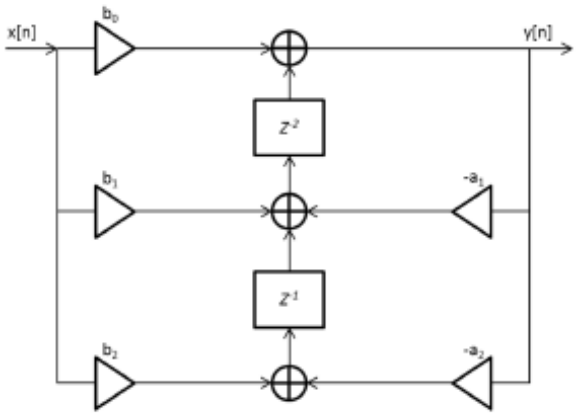


Figure 3. Transposed Direct-form II for second-order IIR filters

Putting all these information together, the hardware consists of three instances of registers, multipliers and adders. The registers, coefficients, inputs and output are 32 bits long represented in signed fixed-point format (Q2.30, 1-bit integer and 30-bit fraction). Also, the multiplication result is 64 bits long which is truncated to 32 bits.

From equation (7) it is possible to identify that coefficients a_1 and a_2 are dynamically changed as a function of time. That is because the pole-radius-varying technique alters these coefficients. As can be seen in equation (6), the pole-radius varies in exponential saturation rate. Hence, we first simulated in MatLab the maximum damping factor α that provides good transient response attenuation and fast saturation for the pole-radius values. After several simulations, we concluded that a damping factor α of 0.1 produces good results for both criteria. This results in 68 and 89 different values for a_1 and a_2 respectively until these coefficients remain constant as a consequence of the exponential saturation.

For the hardware design purpose we assume that a_1 and a_2 are inputs to the architecture. Therefore, these values are injected at the test bench level. Notice that if these coefficients were not inputs but stored in an internal ROM, the filtering process would be the same as proposed here.

B. Near-Threshold Voltage Library

The proposed hardware architecture is synthesized in our custom 65 nm near-threshold voltage standard cells library. The library includes basic NAND, NOR, INV and DFF cells which were manually designed in a layout editor and had the parasitics extracted using a commercial tool. The near-threshold library design flow continues with cells characterization at many different process corners, temperatures, loads and voltages.

For the notch filter design, the temperature was kept constant at 35 degrees Celsius due to the health application requirements. The chosen voltage for near-threshold operation in moderate inversion is 0.45V, which is near, but above the transistor threshold. For comparison purposes, that library has also been characterized at three other voltage corners: 0.55V, 0.80V and 1.2V (nominal).

IV. Results

This section presents the results on the filtering technique response and logical synthesis.

A. Notch Filter Results

Our case study to verify the notch filter functionality is the 60Hz interference elimination from recorded EEG signals sampled at 250 Hz and provided by the Colorado State University EEG Brain-computer interface Lab [12]. Figure 4 shows in *a*) the original signal and in *b*) its Fast Fourier Transform (FFT). Notice in *b*) there is an interference component of 60 Hz. After the notch filtering process, the filtered signal's FFT can be checked in *c*). Notice that the interference component was significantly attenuated.

B. Synthesis Results

The architecture was coded in VHDL. It uses three 32-bit multipliers, three 32-bit adders and 96 Flip-Flops (2 x 32-bit for state retention and 1 x 32-bit to the output). All synthesis share the same timing constraints, with a 5kHz clock. The power is evaluated considering a real world scenario where the activity factor is taken into account. The wiring is accounted for both timing and power, however the clock tree and memory power are not included in the analysis.

Figure 5 shows the energy per sample processed by the developed notch filter at different voltages. The points fit a quadratic equation dependent on the operating voltage, as expected. All designs are evaluated at the same 5 kHz frequency. Data points below 0.45V are missing in Fig. 5 since our library and operating frequency are not targeted for subthreshold operation. The minimum energy design obtained operates at 0.45V with 61 pJoules per sample, 11x energy-savings compared to the nominal voltage of 1.2V.

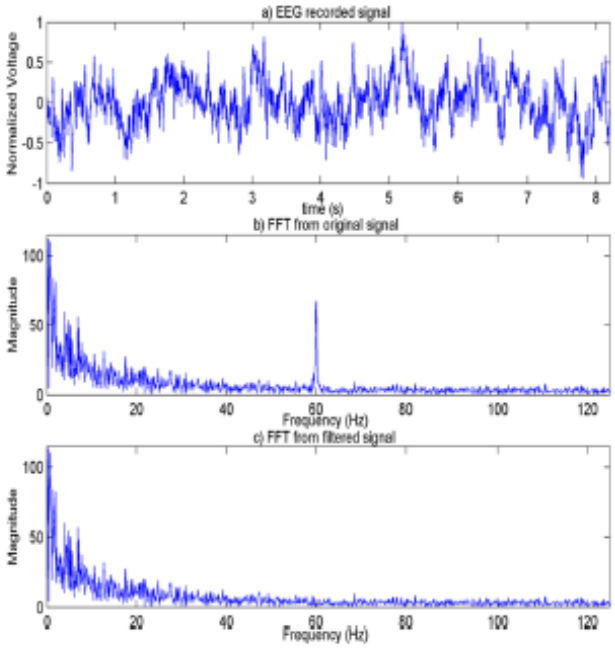


Figure 4. Notch Filter Results: a) EEG recorded signal; b) FFT from original signal; c) FFT from filtered signal

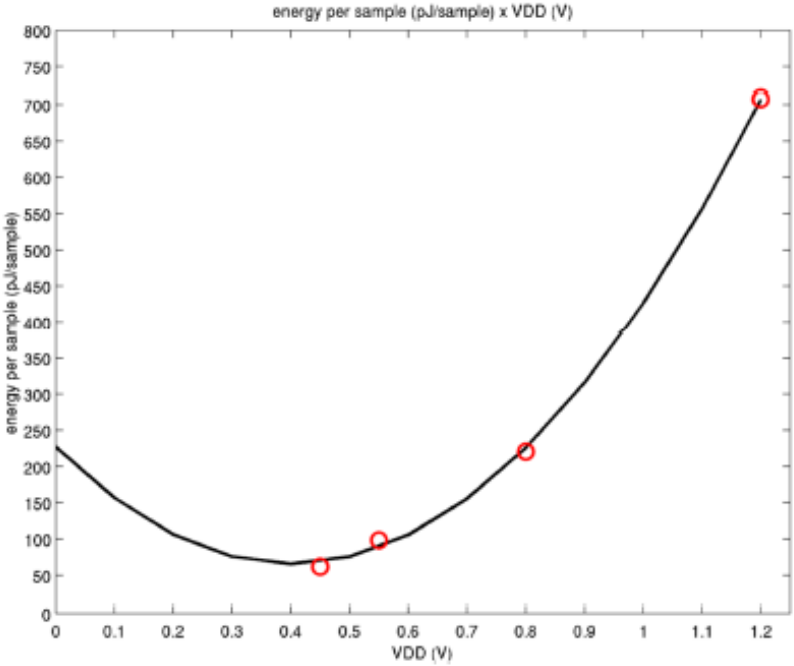


Figure 5. Synthesis Results for Energy per Sample @ f = 5kHz

V. CONCLUSION

A CMOS design of digital second-order pole-radius-varying IIR notch filter is proposed. We verified that the pole-radius-varying technique outperforms the pole-radius-fixed one if considered the filter's transient response. This feature is important when the rejected band has to be very narrow.

Since our case study is correlated with the medical and health-care applications, then the ultra-low power requirements are presented. The hardware implementation was synthesized with our custom near-threshold standard cells library, and achieves up to 11x energy-savings compared to the nominal voltage, with a 61pJoules per sample energy consumption.

ACKNOWLEDGMENT

The authors would like to thank CAPES, Brazil for financial support.

REFERENCES

[1] G. Buzsáki, Rhythms of the brain, Oxford University Press: New York, 2006.

[2] A.G. Constantinides, "Digital notch filters", in Electronics Letter, Vol. 5, 1969, pp. 198-199.

[3] K. Hirano, S. Nishimura, and S. Mitra, "Design of digital notch filters", in IEEE Transactions on Circuits and Systems, Vol. 21, 1974, pp. 540-546.

[4] G. Yan, "New digital notch filter structures with low coefficient sensitivity", in IEEE Transactions on Circuits and Systems, Vol. 31, 1984, pp. 825-828.

[5] A. Zalnieriunas, "Adaptive notch wave digital filters", in IEEE International Symposium on Circuits and Systems, Vol. 4, 1990, pp. 3158 - 3161.

[6] R. Wariar, "A tracking digital notch filter", in IEEE International Symposium on Circuits and Systems, Vol. 5, 1991, pp. 2810 - 2813.

[7] T.S. Ng, "IIR notch filtering-comparisons of four adaptive algorithms for frequency estimation", in IEEE International Symposium on Circuits and Systems, Vol. 2, 1995, pp. 865- 868.

[8] C. Tseng, "Stable IIR notch filter design with optimal pole placement", in IEEE Transactions on Signal Processing, Vol. 49, 2001, pp. 2673-2681.

[9] C. Tseng, "Analytical design of multidimensional IIR digital notch filter", in IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, Vol. 49, 2002, pp. 882 - 887.

[10] J. Piskorowski, "Digital notch filter with time-varying quality factor for the reduction of powerline interference", in IEEE International Symposium on Circuits and Systems, 2010, pp. 2706 - 2709.

[11] L. Tan, "Pole-Radius-Varying IIR Notch Filter With Transient Suppression", in IEEE Transactions on Instrumentation and Measurement, Vol. 61, 2012, pp. 1684 - 1691.

[12] CEBL, CSU EEG Brain-computer interface Lab, http://www.cs.colostate.edu/eeg/eegSoftware.html, Accessed in 2012.

[13] B.H. Calhoun, A. Wang, and A. Chandrakasan, "Modeling and sizing for minimum energy operation in subthreshold circuits", in IEEE Journal of Solid-State Circuits, Vol. 40, 2005, pp.1778 - 1786.

[14] A. Wang, and A. Chandrakasan, "A 180-mV subthreshold FFT processor using a minimum energy design methodology", in IEEE Journal of Solid-State Circuits, Vol. 40, 2005, pp.310 - 319.

[15] H. Kaul, "Near-threshold voltage (NTV) design – opportunities and challenges", in 49th Annual Design Automation Conference, 2012, pp. 1153 - 1158.

[16] D. Markovic, "Ultralow-power design in near-threshold region", in Proceedings of the IEEE, Vol. 98, 2010, pp. 237 - 252.

ANEXO G <Trabalho de Graduação 1>

Arquitetura de Hardware para Transformada Rápida de Fourier Aplicada ao Tratamento de Sinais do Sistema Nervoso

Jorge Wichrowski Krieger de Mello

Graduando em Engenharia de Computação

Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brasil

jwkmhdr@hotmail.com

Abstract: *This monography describes the first phase of the Undergraduate Research Work of Jorge Wichrowski Krieger de Mello, Computer Engineering student. The goal of the study is the development of an integrated circuit for calculating the Fast Fourier Transform at low power, focusing on the treatment of spinal cord signs.*

Resumo*. Esta monografia descreve a primeira fase da Pesquisa do Trabalho de Graduação de Jorge Wichrowski Krieger de Mello, estudante do curso de Engenharia de Computação. O objetivo do estudo é desenvolver um circuito integrado para cálculo da Transformada Rápida de Fourier em baixa potência, tendo como foco tratamento de sinais medulares.*

1. Introdução

Este trabalho de conclusão versa sobre a aplicação da FFT (Transformada Rápida de Fourier) a sinais obtidos para fins de tratamentos médicos. O primeiro passo é o estudo do algoritmo da FFT otimizado para execução em hardware *full-custom*. A seguir será descrito um circuito integrado em VHDL durante o trabalho para o cálculo da FFT. Será feita uma análise desse projeto para sua execução em baixa potência, visto que este é um requisito de projetos da área médica, pois instrumentos médicos que precisam ser implantados necessitam de grande autonomia de funcionamento. A área de aplicação especificamente é o tratamento de sinais da medula espinhal, para definição das características da arquitetura.

O foco desse estudo é uma solução para resolver o problema da paralisia por rompimento da medula espinhal que acomete algumas pessoas. Visto a tecnologia atualmente disponível é possível acreditar que seja apenas uma questão de tempo e não de capacidade para resolver estes casos médicos. No contexto de um trabalho final multidisciplinar será especificado um esboço do projeto onde a FFT está inserida. Para isso será feita uma rápida apresentação da ideia usando modelagem 3D.

O objetivo desta monografia é trabalhar em nível de hardware e de software, exatamente o que se é proposto a aprender no curso de Engenharia de Computação. Podemos separar este trabalho em quatro partes distintas:

Um estudo do problema em nível médico e biológico, complicações soluções disponíveis, análise do sinal a ser amostrado.

Em conjunto a este estudo será desenvolvido um projeto levando em conta os objetivos a serem alcançados e também serão traçados os requisitos da transformada de Fourier, necessária para tratar os sinais nervosos.

Numa segunda etapa será desenvolvido o projeto em VHDL da transformada rápida, simulação e sua síntese lógica para baixa potência.

Por fim será realizada a aplicação de ultra-baixa potência, em um projeto multidisciplinar.

2. Estudo do Problema

2.1 Impulso Nervoso

Na membrana da célula nervosa há a chamada **Bomba de Sódio (Na+) e Potássio (K+)** que constantemente fica bombeando Sódio (Na+) para fora da célula. Como sai muito mais Na+ do que entra de K+ a célula nervosa em repouso se torna eletronegativa em seu interior.

Há também o **Canal de Sódio (Na)** que permanece fechado em repouso.

O **impulso nervoso** se dá da seguinte maneira, o meio intracelular é rico em K (Potássio) e o meio extracelular é rico em Na (Sódio). O meio intracelular em estado de repouso apresenta -70mV de potencial elétrico, ao receber um pequeno estímulo o Na entra para dentro da célula a medida que um pouco de Potássio(K) sai, tornando o meio intracelular positivo aproximadamente +30mV. Em seguida o Sódio (Na) é jogado para fora de novo enquanto o Potássio(K) volta para dentro, esta inversão é transmitida ao longo do **axônio** e é chamada de **onda de polarização**.

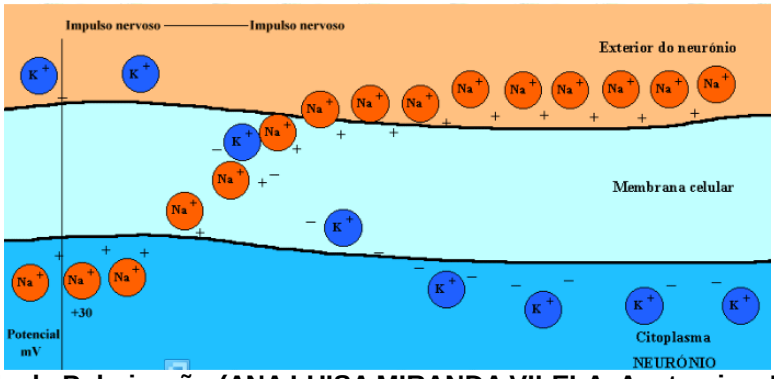


Figura 1: Onda de Polarização (ANA LUISA MIRANDA VILELA. Anatomia e Fisiologia Humanas. http://www.afh.bio.br/nervoso/nervoso4.asp#SNP.)

Impulsos nervosos ocorrem em média a cada 5ms, o que significa uma frequência média de 200Hz.

2.2 Sinapse

A **sinapse química** ocorre quando não há contato físico entre os neurônios, ou seja, na grande maioria deles, entre eles há uma fenda sináptica de 20 a 50nm.

A **energia elétrica** contida no neurônio é convertida para energia

química (são liberados **neurotransmissores**) e depois para elétrica no neurônio seguinte. Esta é sinapse mais comum e que ocorre com mais frequência, sua vantagem que ela é unidirecional, mas também é mais lenta. Na medula espinhal ela é predominante.

2.3 Medula Espinhal

A **medula espinhal** possui aproximadamente 40 cm de comprimento. Possui um sistema descendente que leva informação do cérebro aos músculos e possuí um sistema ascendente que leva informações sensoriais do corpo ao cérebro. Essa comunicação é realizada pelo **sistema periférico** que é formado pelos **31 pares de nervos raquidianos** que saem da **medula espinhal**.

O corpo dos neurônios fica no centro da medula ou massa cinzenta, já o axônio fica na massa branca.

Dado interessante é que se a lesão da medula atingir apenas a parte cinzenta, poderá ficar restrita a uma determinada porção do corpo sem afetar os membros mais abaixo. Agora a lesão que ocorre na massa branca paralisa tudo que estiver abaixo dela. A lesão em si vem acompanhada de outros problemas que acabam piorando a recuperação do paciente, por isso há necessidade de tecnologia avançada e médicos capacitados para reduzir as chances de sequelas.

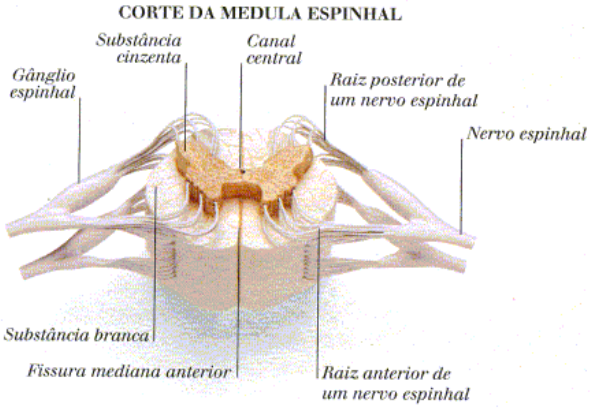


Figura 2:Corte Horizontal na Medula (ANA LUISA MIRANDA VILELA. Anatomia e Fisiologia Humanas. http://www.afh.bio.br/nervoso/nervoso4.asp#SNP.)

Abaixo na figura 3, a figura mostra os 31 pares de nervos raquidianos em uma espinha humana:

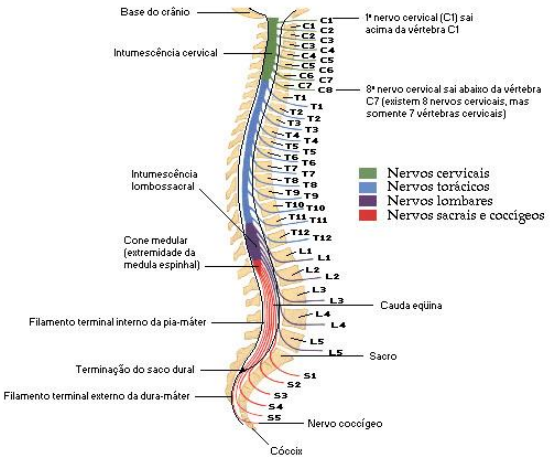


Figura 3: Coluna Vertebral 31 pares de Nervos Raquidianos (ANA LUISA MIRANDA)

3. Desenvolvimento da ideia de solução do problema - Especificação do

Projeto da Transformada de Fourier

Esta parte do trabalho é onde será usada toda capacidade criativa e intelectual na busca de uma solução viável e que parece boa para quem for apresentada a ideia.

Tanto para o projeto, como para definir os requisitos da Transformada de Fourier Rápida é necessária uma análise do sinal. O nosso sinal nervoso apresenta pulsos que duram um intervalo entre 2ms e 7ms com média de 4,5ms. Esta característica esta de acordo com os tempos citados na literatura: 1,5ms para alcançar o pico de voltagem e 3ms para voltar ao valor do estado de repouso. O inverso deste período é aproximadamente 220Hz. Na figura 4 é mostrado o sinal de um modelo aproximado (modelo de **Hodgkin-Huxley**) para as ações de potenciais elétricos em neurônios.

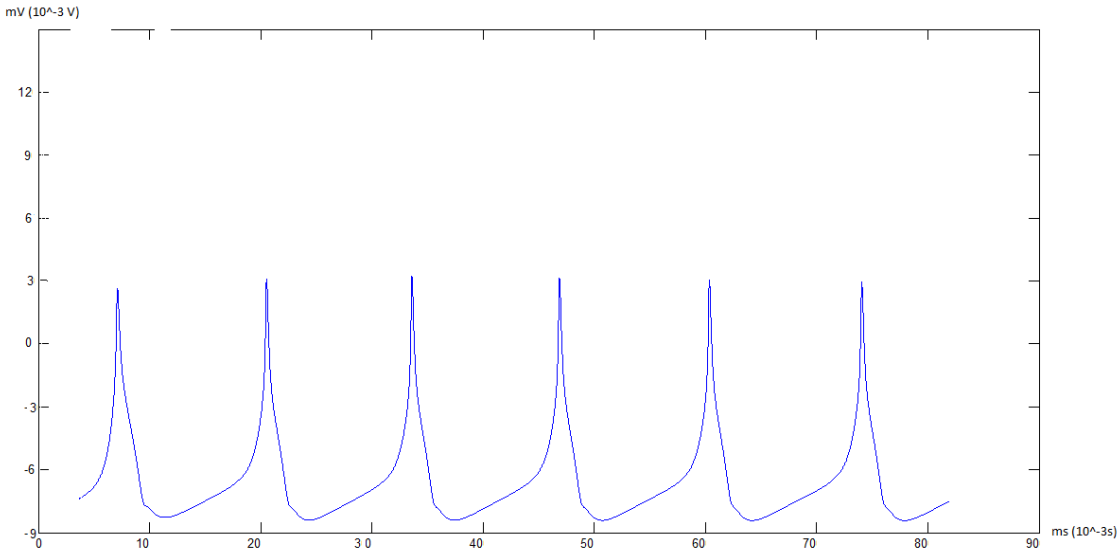


Figura 4: Forma de onda dos impulsos nervosos.

Se usarmos a definição do teorema de **Nyquist**, que diz que para um sinal ser posteriormente reconstruído com o mínimo de perda de informação devemos amostrá-lo com o dobro da frequência máxima que predomina no seu espectro, podemos chegar à conclusão de que necessitamos amostrar este sinal com um pulso de aproximadamente 500Hz.

Nosso sinal de interesse é apenas um pulso pertencente a mais alta frequência por isso pode definir uma janela pequena para se calcular a transformada de Fourier. Vamos começar com uma janela de oito pontos, caso nos estudos posteriores se verifique necessidade de aumentar o numero de pontos então isso será feito.

Um dos efeitos negativos de se usar uma janela muito grande seria que o espectro no domínio frequência teria pouca representatividade nas frequências de interesse em função do tempo de repouso do neurônio (baixas frequências) ser muito maior que o período de atividade. Na figura 5 é possível ver esse efeito analisando a magnitude das componentes espectrais (no domínio das frequências) do modelo de **Hodgkin-Huxley** para descrever ações de potenciais em neurônios, feitos no **MatLab**.

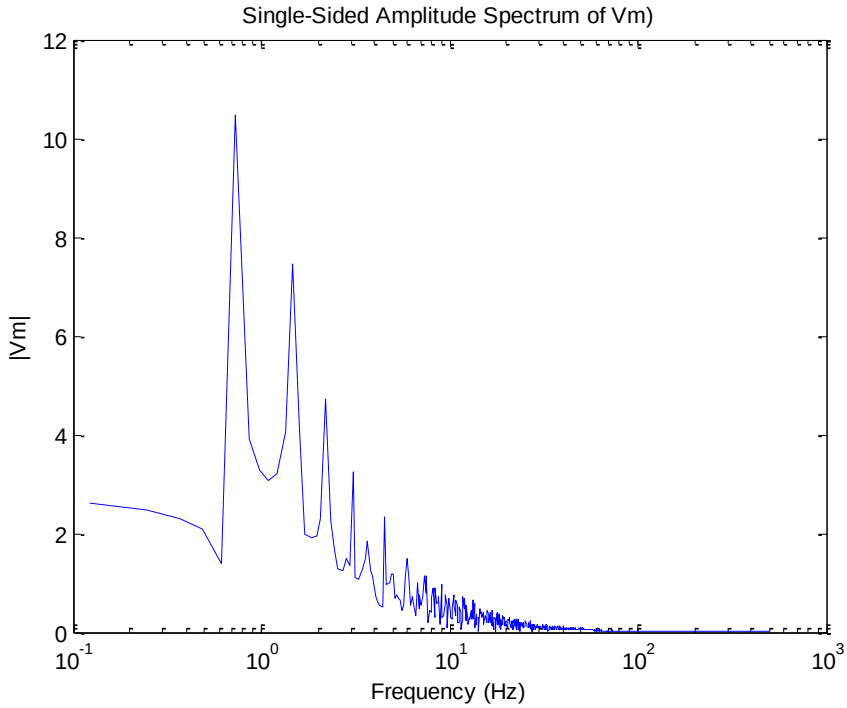


Figura 5: Efeito da transformada sobre uma janela de 8192 pontos

Pode-se observar que estamos trabalhando com frequências bem baixas para o padrão dos circuitos atuais. Logo é possível afirmar que não haverá problemas para bater timing e que estes circuitos provavelmente conseguirão operar a baixas potências. Pois quanto mais baixo é *clock*, menor a potência.

$$P = C\, f\, V^2$$

Potência é a multiplicação das capacitâncias internas, da frequência e da tensão de alimentação(V) ao quadrado. A frequência é baixa logo a potência dissipada será baixa, sendo a frequência baixa podemos operar com tensões menores o que nos permite reduzir ainda mais a potência. Inclusive um dos assuntos desse trabalho será operar a voltagens *Near-Threshold* (abaixo do valor limiar de funcionamento de um transistor).

4. Desenvolvimento de um dos módulos do projeto a FFT Rápida

4.1 – Transformada de Fourier

Esta parte do trabalho tem como objetivo entender e explicar o funcionamento do módulo mais usado atualmente para processamento de sinais a Transformada de Fourier.

A Transformada de Fourier tem como objetivo tomar um sinal com representação no domínio tempo e representá-lo no domínio frequência, isto é feito separando todas as frequências que aparecem no sinal no domínio tempo e medindo suas intensidades. Após isso você tem um espectro que contém pulsos maiores nas frequências que mais aparecem no domínio tempo.

Esse tipo de transformação é muito útil, pois torna mais rápido algumas operações matemáticas como a convolução (passagem de um sinal pelo

outro), este tipo de operação fica extremamente rápida, pois o que seria uma soma de produtos se torna apenas uma multiplicação e isto nos permite obter a resposta a um determinado impulso sobre o sinal de maneira eficaz. Também é facilitada a operação de filtragem do sinal, ou seja, obter apenas faixas de valores de interesse.

Existem quatro tipos de transformações de Fourier distintas, em ambas o princípio é o mesmo o que muda é o tipo original do sinal e com isso difere o modo como realizamos as transformadas, a Série de Fourier de Tempo Contínuo, a Série de Fourier de Tempo Discreto ou Transformada de Fourier Discreta (FTD), a Transformada de Fourier de Tempo Contínuo, a Transformada de Fourier de Tempo Discreto, este trabalho tratará da Transformada de Fourier Discreta (FTD), que é provavelmente a mais usada atualmente, pois ela trata de sinais periódicos e discretos, ou seja, para computadores que lidam apenas com sinais digitais o fato do sinal ser discreto é uma vantagem. Como a maioria dos sinais do nosso meio são contínuos o que se faz é amostrar eles para que o computador possa aplicar a transformada.

Tendo em vista a grande utilização da Transformada de Fourier Discreta foram desenvolvidos métodos para ela que reduzem o tempo de cálculo em diversas vezes. Estes métodos tem basicamente o mesmo princípio reorganizar os dados amostrados de modo a eliminar as redundâncias de cálculos matemáticos. Quando utilizamos esses algoritmos falamos que estamos realizando a Transformada Rápida de Fourier.

Vamos analisar os sinais para os quais estamos focando este trabalho, para tanto é necessário compreender as propriedades básicas dos pares de Fourier e neste contexto entender como um sinal no domínio tempo será representado no domínio frequência:

Contínuo	-	Não Periódico
Discreto	-	Periódico
Periódico	-	Discreto
Não Periódico	-	Contínuo

Vamos agora analisar o nosso sinal para decidir qual a melhor representação em Fourier para ele. Estamos falando de um sinal analógico com frequência máxima de 250Hz que deverá ser amostrado com uma frequência de corte de pelo menos 500Hz, como resultado da amostragem temos um sinal discreto. Este sinal iremos assumi-lo como periódico durante a realização de uma atividade repetitiva, como andar e correr, por exemplo. Então nossa representação escolhida como já mencionada anteriormente é FTD (Transformada de Fourier Discreta).

4.2 A Transformada Rápida de Fourier de 4 pontos

Quando falamos de Transformada Rápida de Fourier estamos falando de algoritmos para o cálculo da transformada que reduz sua complexidade, reorganizando as entradas de forma a evitar realizar cálculos redundantes.

Esta reorganização só é possível fixando um N (tamanho de amostras) de modo a poder separar os cálculos em blocos que podem ser subdivididos em blocos de mesmo tamanho até se ter blocos do tamanho mínimo, por isso normalmente este tamanho é base de 2 (2,4,8,16,32,64...).

Existem diversos algoritmos com diversas vantagens, aqui neste trabalho vamos tratar do mais conhecido o *Radix-2* ou algoritmo de *Cooley-Tukey*, neste algoritmo quebramos as amostras em pares e impares, o resultado nós separamos de novo em dois grupos, fazemos isso até obter o resultado final.

O N deve ser base 2, o aumento do desempenho em relação ao método original se torna mais significativo a medida que o N (numero de pontos aumenta).

Transformada de Fourier = N^2

Transformada Rápida de Fourier = $N \log_2(N)$

Estudaremos a FFT de 4 pontos para isso começamos pela equação da transformada de Fourier de tempo discreto:

$$X(k) = \sum_{n=0}^{N-1} x[n].e^{-\frac{j2\pi kn}{N}}$$

São 4 pontos então vamos de n=0 até n=3:

$$X(0) = \sum_{n=0}^{3} x[n].e^{-\frac{j2\pi 0n}{4}} = x(0) + x(1) + x(2) + x(3)$$

$$X(1) = \sum_{n=0}^{3} x[n].e^{-\frac{j2\pi 1n}{4}} = x[0] + x[1] * e^{-\frac{j2\pi 1}{4}} + x[2] * e^{-\frac{j2\pi 2}{4}} + x[3] * e^{-\frac{j2\pi 3}{4}}$$

$$X(2) = \sum_{n=0}^{3} x[n].e^{-\frac{j2\pi 2n}{4}} = x[0] + x[1] * e^{-\frac{j2\pi 2}{4}} + x[2] * e^{-\frac{j2\pi 4}{4}} + x[3] * e^{-\frac{j2\pi 6}{4}}$$

$$X(3) = \sum_{n=0}^{3} x[n].e^{-\frac{j2\pi 3n}{4}} = x[0] + x[1] * e^{-\frac{j2\pi 3}{4}} + x[2] * e^{-\frac{j2\pi 6}{4}} + x[3] * e^{-\frac{j2\pi 9}{4}}$$

Os resultados acima serão reescritos na forma matricial:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & e^{-\frac{j2\pi 1}{4}} & e^{-\frac{j2\pi 2}{4}} & e^{-\frac{j2\pi 3}{4}} \\ 1 & e^{-\frac{j2\pi 2}{4}} & e^{-\frac{j2\pi 4}{4}} & e^{-\frac{j2\pi 6}{4}} \\ 1 & e^{-\frac{j2\pi 3}{4}} & e^{-\frac{j2\pi 6}{4}} & e^{-\frac{j2\pi 9}{4}} \end{bmatrix} * \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ x[3] \end{bmatrix}$$

Simplificando os valores da matriz de coeficiente complexos, obtém-se:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} * \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ x[3] \end{bmatrix}$$

Partindo dessa Matriz deduziremos um dos algoritmos da Transformada

Rápida de Fourier e que será utilizado neste trabalho, o algoritmo de Cooley-Tukey também chamado de Radix-2 ou Decimação no Tempo. Neste algoritmo a Transformada de Fourier de N pontos é sucessivamente dividida em duas partes de N/2 pontos.

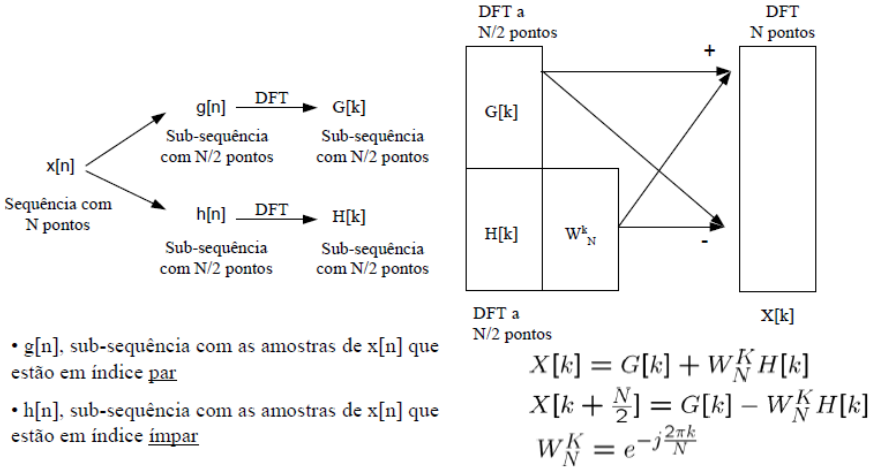


Figura 6: Radix-2(STDS e PSTR – Inverno 2007/2008)

Esta divisão é possível, pois separamos a parte par da parte impar isso pode ser visto através do *bit-reversal* que separa a sequência nas duas subsequências com as amostras de índice par e ímpar, designadas por g[n] e h[n], respectivamente.

Domínio Tempo	Domínio Frequência
00	00
01	10
10	01
11	11

Resolveremos a multiplicação de matrizes mostrada anteriormente:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} * \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} a+b+c+d \\ a-c-j(b-d) \\ a+c-(b+d) \\ a-c+j(b-d) \end{bmatrix}$$

Ao aplicar a FFT vemos que temos o mesmo que esta visto acima.

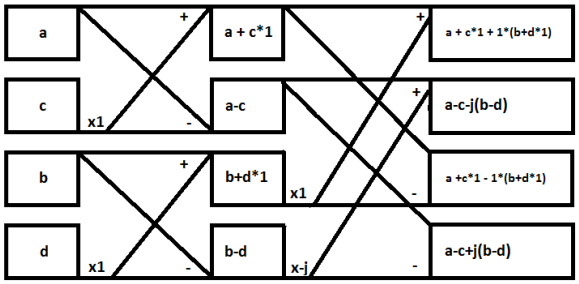


Figura 7: Algoritmo FFT 4 pontos

Pode-se observar que, ao invés da aplicação direta da transformada de Fourier que demanda 16 operações, a FFT reorganiza os cálculos de forma que se tenha apenas 8 operações. Conforme já foi falado acima esse desempenho aumenta com o aumento do N:

N	N^2 (DFT)	$N.\log_2 N$ (FFT)	Vantagem
2	4	2	2
4	16	8	2
8	64	24	2,67
16	256	64	4
32	1024	160	6,4
64	4096	384	10,67
128	16384	896	18,29
256	65536	2048	32
512	262144	4068	56,89
1024	1048576	10240	102,4
2048	4194304	22528	186,18
4096	16777216	49512	341,33
8192	671088964	106496	630,15

Figura 8: Complexidade FFT (UFPE. 2010.)

Vamos agora ver um exemplo simples com números para que não restem dúvidas de como é realizado o algoritmo da FFT:

$$x[n] = \{3, 1, 2, -1\}$$

$$g[n] = \{3, 2\}$$

$$h[n] = \{1, -1\}$$

$$X[k] = \{5, 1 - j2, 5, 1 + j2\}$$

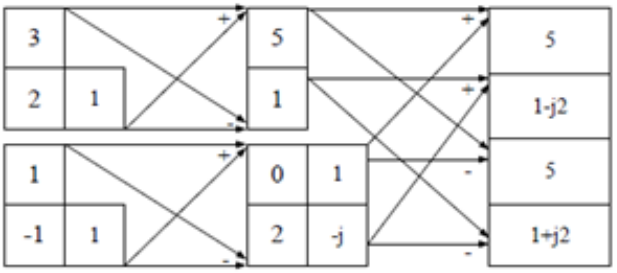


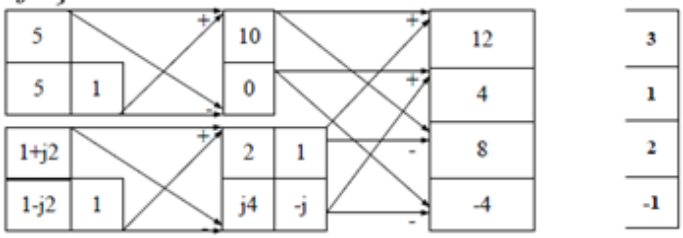
Figura 9: Exemplo FFT (STDS e PSTR – Inverno 2007/2008)

Vejamos apenas por curiosidade e aprendizado como se dá a volta da transformada de Fourier, ou melhor dizendo, a transformada inversa:

$X^*[k] = \{5, 1+j2, 5, 1-j2\}$

$G[k] = \{5, 5\}$

$H[k] = \{1+j2, 1-j2\}$



$x[n] = \{3, 1, 2, -1\}$

Figura 10: Exemplo FFT Inversa (STDS e PSTR – Inverno 2007/2008)

4.3 A Transformada Rápida de Fourier de 8 pontos

A Transformada Rápida de Fourier de 8 pontos será a que priori estudaremos e descreveremos neste trabalho, nossa janela tem 8 amostras do nosso sinal, cada amostra possui tamanho de 8 bits.

Abaixo segue um diagrama do algoritmo da FFT de 8 pontos, não iremos deduzi-lo, pois já fizemos para a FFT de 4 pontos, mas cabe aqui resaltar que o procedimento é o mesmo, usa-se o bit-reversal para determinar a organização dos cálculos, usa-se o calculo cruzado “borboleta”, o que muda é que temos um estágio a mais e o fator *Twiddle* difere para o último estágio.

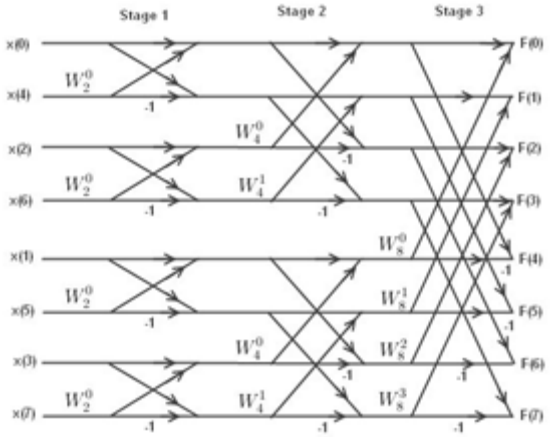


Figura 11: Algoritmo FFT 8 pontos (ALWAYSLEARN. – A DFT and FFT Tutorial. http://www.alwayslearn.com.)

Sobre o fator de *Twiddle* podemos dizer que ele pode ser calculado para cada estágio da FFT:

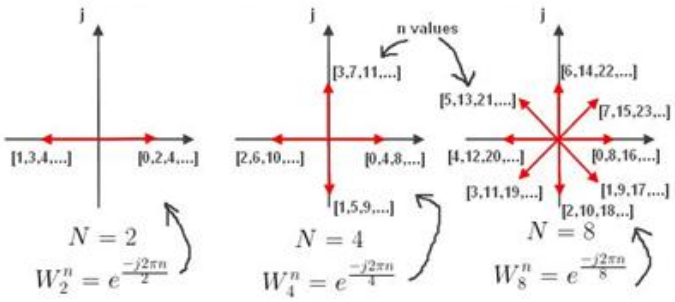


Figura 12: Fator Twiddle para diferentes N (ALWAYSLEARN. – A DFT and FFT Tutorial. http://www.alwayslearn.com.)

Dito isso apresentaremos uma tabela num molde um pouco mais claro do algoritmo que será implementado em VHDL. Depois criaremos um caso de teste e aplicaremos ele na tabela para ver a saída. Tendo o algoritmo bem documentado e um caso de teste podemos descrever o hardware em VHDL.

É preciso falar antes de apresentar os cálculos que neste trabalho vamos apenas tratar números de ponto fixo. Ou seja, aqueles resultados que tiverem números em ponto flutuante serão arredondados ou truncados. Também faremos uma aproximação para aumentar o desempenho que consiste na operação de multiplicar por 0.7 presente no terceiro estágio da FFT de 8 pontos, ao invés disso, dividiremos por 2 e depois dividiremos por 4 e somaremos os dois resultados, equivale a multiplicar por 0.75, mas como estamos tratando de números inteiros com o arredondamento ou truncamento dos resultados podemos na verdade estar obtendo o mesmo valor que multiplicando por 0.7. Essa aproximação garante mais velocidade, pois essas operações são apenas deslocamento de bits para direita com números positivos.

Esquematização do algoritmo e caso para teste:

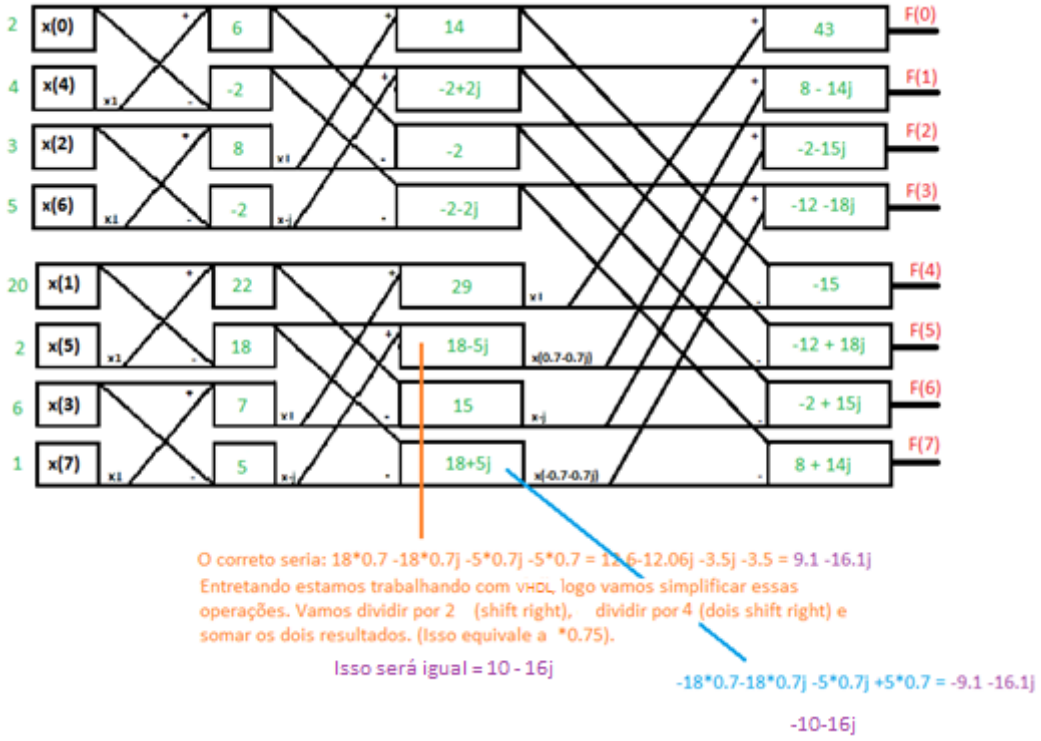


Figura 13: Caso para teste FFT

4.4 Perspectiva:

Principal parte prática desse trabalho será a criação e o aperfeiçoamento do VHDL da FFT, geração de mais casos teste, síntese lógica para 2 voltagens *near-threshold* e para 1 voltagem convencional, síntese para placa FPGA.

Tudo isso para exibir e comparar os resultados gerando dados que poderão ser útil no futuro não só para o projeto em questão, mas para outros projetos.

É importante lembrar a questão de trabalhar em baixa potência é essencial para circuitos androides já que não podemos colocar baterias dentro de pessoas.

4.5 Ferramental:

Na UFRGS no laboratório de pesquisa temos todas as ferramentas de síntese lógica e física da Cadence instaladas, possuímos também instalados o software ISE, Chipscope e temos a disposição placas da Xilinx de FPGA. Todo ferramental está pronto para uso. Apenas as bibliotecas de baixa potência para síntese lógica ainda não estão disponíveis, mas é apenas uma questão de tempo até elas ficarem prontas, uma vez que nosso laboratório já possuí todo conhecimento para geração de arquivos .lib a partir de layouts.

4.6 Pré-requisitos:

Cursei Sistemas Digitais, Concepção 1 e 2, atuei em I.C. em Codificação de Video em VHDL, agora atuo no desenvolvimento de lógica CMOS fazendo a geração de layouts em baixa potência. Possuo todos pré-requisitos necessários para utilizar as ferramentas e desenvolver esta tarefa.

5. Criação da apresentação da ideia

Como trabalho complementar para poder inserir o módulo da FFT no contexto o qual esta sendo estudado é criar e documentar uma rápida apresentação com modelagem e animação usando Blender, software gratuito que para propósitos de pesquisa esta perfeitamente enquadrado, segundo a ideia de projeto para solucionar o problema em questão.

5.1 Perspectiva:

Principalmente esperamos expor melhor a ideia do trabalho para as demais pessoas.

5.2 Ferramental:

Possuímos livros e o programa instalado nos computadores, é possível iniciar o trabalho imediatamente.

5.3 Pré-requisitos:

Não possuo nenhuma experiência específica, mas considero esta tarefa fácil e terei plena capacidade para aprender a desenvolvê-la.

6. Objetivo do Trabalho

Tentar introduzir uma linha de estudo voltado para esta questão. Em um objetivo um pouco maior servir como base de inicio de um trabalho que culmine com a cura total desse tipo de lesão.

A parte prática da transformada de Fourier também permitirá a geração de dados de pesquisa para auxiliar futuros trabalhos em outras áreas.

7. Cronograma

Tarefa	Prazo Estimado	Carga Horária
Estudo do Projeto	31 de Outubro	50 horas
Especificação do Projeto	31 de Outubro	10 horas
Desenvolvimento do	21 de Outubro	50 horas

Módulo da FFT Rápida		
Compilação da FFT para FPGA	25 de Novembro	10 horas
Síntese Lógica da FFT Tensão Convencional	15 de Novembro	10 horas
Síntese Lógica da FFT a 2(duas) Tensões de alimentação abaixo da nominal	15 de Novembro	10 horas
Desenvolvimento do Texto	15 de Dezembro	40 horas
Finalização Textual	1 de Dezembro	20 horas
Encaminhar documento para correção	15 de Dezembro	X

8. Conclusão

No contexto da Transformada Rápida de Fourier descreveremos em VHDL o algoritmo Radix-2 que é o mais conhecido, há outros algoritmos (Radix-4,Radix-8,Danielson-Lanczos) que apresentam pequenas melhoras dependendo do caso, mas a ideia é sempre a mesma dividir para evitar cálculos redundantes.

A proposta irá englobar software e hardware, será aplicado conhecimentos do tratamento de sinais para solucionar um problema da área médica, o atual trabalho de iniciação científica esta incluso no TCC uma vez que esses circuitos devem ser de baixa potência, ou seja, devemos gastar o mínimo de energia necessário para realizar a tarefa.

O trabalho esta bastante abrangente, caberá selecionar aquilo de mais útil para quem ler este trabalho com objetivo de iniciar uma pesquisa na área.

Referências

[1] ANA LUISA MIRANDA VILELA. Anatomia e Fisiologia Humanas. http://www.afh.bio.br/nervoso/nervoso4.asp#SNP.

[2] EUGENE IZHIKEVICH. Neuron Models on FPGA. 2003

[3] CARLOS ALEXANDRE MELLO. UFPE - FFT - Fast Fourier Transform.

[4] STDS e PSTR – Inverno 2007/2008.

[5] ALWAYSLEARN. – A DFT and FFT Tutorial. http://www.alwayslearn.com.

[6] FERNANDO S. SCHLINDWEIN. DFT / FFT, spectral estimation, some proofs and windows. 2012.

[8] WILLIAM PEREIRA ALVES. Blender 2.63 – Modelagem e Animação. 2012.

[9] NEURO INFORMATICS 2012. http://www.neuroinformatics2012.org. 2012.