João Paulo Fernandes Barbosa

PROJETO E DESENVOLVIMENTO DE UMA ARQUITETURA EM HARDWARE RECONFIGURÁVEL PARA SEGMENTAÇÃO DE VÍDEOS

UNIVERSIDADE FEDERAL DE PERNAMBUCO
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE
2016

João Paulo Fernandes Barbosa

PROJETO E DESENVOLVIMENTO DE UMA ARQUITETURA EM HARDWARE RECONFIGURÁVEL PARA SEGMENTAÇÃO DE VÍDEOS

Dissertação de Mestrado apresentada à Universidade Federal de Pernambuco, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do Título de Mestre.

Orientador: Manoel Eusebio de Lima

RECIFE
201

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

B238p Barbosa, João Paulo Fernandes
Projeto e desenvolvimento de uma arquitetura em hardware reconfigurável para segmentação de vídeos / João Paulo Fernandes Barbosa. – 2016.
123 f.: il., fig., tab.

Orientador: Manoel Eusebio de Lima.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Informação, Recife, 2016.
Inclui referências.

1. Arquitetura de computadores. 2. Segmentação de vídeo. I. Lima, Manoel Eusebio de (orientador). II. Título.

004.22 CDD (23. ed.) UFPE- MEI 2017-77

1.

João Paulo Fernandes Barbosa

PROJETO E DESENVOLVIMENTO DE UMA ARQUITETURA EM HARDWARE RECONFIGURÁVEL PARA SEGMENTAÇÃO DE VÍDEOS

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação

Aprovado em: 26/02/2016.

BANCA EXAMINADORA

_____

Profa. Dra. Edna Natividade da Silva Barros
Centro de Informática / UFPE

_____

Prof. Dr. Victor Wanderley Costa de Medeiros
Departamento de Estatística e Informática / UFRPE

_____

Prof. Dr. Manoel Eusebio de Lima
Centro de Informática / UFPE
(**Orientador**)

AGRADECIMENTOS

Agradeço a Deus, El-Elion, porque d'Ele e para Ele são todas as coisas. Agradeço à minha linda esposa Luanda Augusta por toda sua dedicação, por sonhar comigo os meus sonhos que acabam se tornando nossos sonhos e também por todo suporte e amor que foram a mim dedicados durante o desenvolvimento deste trabalho. Agradeço também aos meus pais Severino e Zélia e aos meus irmãos Fábio e Lucélia por suas contribuições em minha formação pessoal, acadêmica e por toda educação que me foi dada e à minha sogra Paula Pinheiro por todo carinho e incentivo.

Quero agradecer especialmente a Antonyus Pyetro e a Rodrigo Camarotti por toda ajuda e apoio no desenvolvimento deste trabalho. Sem eles a construção deste seria bem mais árdua. Agradeço também a Bruno Pessoa, Erika Spencer, João Gabriel Machado e a Severino Barros que me ajudaram de forma singular no desenvolvimento deste projeto e à Joelma França por sua prontidão em ajudar sempre que isso se fez necessário.

Agradeço ao professor Manoel Eusebio de Lima por sempre acreditar em meu trabalho, por me orientar e por me dar a oportunidade de desenvolver meus conhecimentos através do grupo HPCIn. Agradeço também à professora Edna Barros por servir como um grande modelo profissional, por todos os ensinamentos que ela me transmitiu durante esses anos de convivência e por ter me dado à oportunidade de desenvolver este trabalho no Lincs.

Agradeço aos amigos do LINCS/CETENE que de alguma forma contribuíram para que fosse possível a conclusão desta dissertação: Josivan Reis, Maria Silveira e Djeefther Albuquerque. Agradeço também aos amigos de sempre do HPCIn: Abner, João Cléber e Victor.

Por fim agradeço aos professores do Centro de Informática que contribuíram com toda minha formação acadêmica. Agradeço de forma especial à professora Renata Souza e ao professor Tsang Ing Ren por terem servido como exemplo de profissionalismo e de cuidado com seus alunos.

RESUMO

A detecção de objetos e a segmentação de sequências de vídeo são os primeiros passos em algumas aplicações e sistemas de visão computacional. Bons resultados tem sido alcançados com a utilização de *General Propose Graphic Processor Unit (GPGPU)* e de *Field Gate Programmable Array (FPGA)* na implementação de aplicações científicas de alto desempenho e esta têm sido uma alternativa à implementação convencional baseada em uma Central Única de Processamento *(CPU)*. Neste contexto, este trabalho apresenta uma arquitetura heterogênea baseada em *CPU* e *FPGA*, que explora o máximo de paralelismo, para o processamento da segmentação de *frames* de vídeo utilizando a análise do espectro de fase de Fourier. O algoritmo de segmentação de vídeos implementado neste trabalho inclui entre suas operações o processamento de uma *FFT 3-D*, o cálculo do espectro de fase e o cálculo da *IFFT 2-D* em uma sequência de vídeo. O desempenho da arquitetura baseada em *CPU* e *FPGA* é comparado com a mesma implementação do algoritmo que utiliza a biblioteca *cuFFT* em um sistema baseado em *CPU* e *GPU*. A arquitetura desenvolvida em um *FPGA Stratix IV* (EP4SE530H35C2) é capaz de segmentar objetos em uma sequencia de vídeo a uma taxa de 1.800 *frames* por segundos.

Palavras-chave: Segmentação de Vídeo. Arquitetura de Hardware. FFT. IFFT. FPGA

ABSTRACT

Object detection and video sequences segmentation are the first step in some applications and computer vision systems. Hardware accelerators such General Propose Graphic Processor Unit (GPGPU) and Field Gate Programmable Array (FPGA) have been used as an alternative to conventional CPU architectures in high-performance scientific applications, and have achieved good speed-up results. Within this context, this work presents a heterogeneous architecture for computing based on CPU and FPGA, which explores the maximum parallelism for processing video segmentation using the video signal Fourier phase spectrum analysis. The video segmentation algorithm includes processing the 3-D FFT, calculating the phase spectrum and the 2-D IFFT operation in a video sequence. The performance of the architecture based on CPU and FPGA is compared with the implementation of the same algorithm with the cuFFT library in a system based on CPU and GPU. The prototyped architecture in a Stratix IV (EP4SE530H35C2) FPGA is able to segment objects in video sequences at 1.800 frames per second.

Keywords: Video Segmentation. Hardware Architecture. FFT. IFFT. FPGA

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

CETENE	Centro de Tecnologias do Nordeste
CLB	Configurable Logical Block
CPU	Central Processor Unit
DDR	Double Data Rate
DFT	Discrete Fourier Transform
DSP	Digital Signal Processor
FFT	Fast Fourier Transform
FFTW	Fast Fourier Transform West
FIFO	Firs in, First Out
FPGA	Field Gate Programmable Array
FPS	Frames Per Second
FSM	Finite State Machine
GMM	Gaussian Mixed Mixture
GPGPU	General Propose Graphic Processor Unit
HDL	Hardware Description Language
IDFT	Inverse Discrete Fourier Transform
IEEE	Institute of Electrical and Electronics Engineers
IFFT	Inverse Fast Fourier Transform
LINCS	Laboratório para Integração de Circuitos e sistemas
LUT	Look-up table
RAM	Random-Access Memory
RGB	Red, Green and Blue
ROM	Read-Only Memory
SNR	Signal-to-Noise Ratio
SRAM	Static Random-Access Memory
UFPE	Universidade Federal de Pernambuco

SUMÁRIO

1. Introdução

Esta dissertação tem como tema o desenvolvimento de uma arquitetura computacional heterogênea, baseada em *CPU* e *FPGA*, capaz de executar a detecção de movimentos em uma sequencia de vídeo e a segmentação das regiões onde o movimento é detectado. Este capítulo apresenta de uma forma geral os conceitos que estão relacionados a esse tema e também é apresentada a motivação que norteou o desenvolvimento deste trabalho, os objetivos a serem alcançados e a organização do texto.

O processamento de imagens e vídeos pode ser empregado em diversas linhas de pesquisa e em aplicações comerciais. Em geral, devem ser aplicadas técnicas de processamento de imagem e visão computacional em sistemas onde é necessária a percepção e extração de informações contidas em uma imagem ou vídeo. O processamento destes tipos de sinais pode ser empregado, por exemplo, na indústria petrolífera para o estudo e mapeamento de regiões onde são encontrados combustíveis fósseis (ROCHA, 2010).

Na área de segurança, o processamento de imagens ou sinais de vídeo pode ser utilizado para controlar o acesso a áreas ou sistemas restritos. O reconhecimento de íris (HEMATIAN et al., 2012) é exemplo de uma das técnicas de identificação biométricas mais precisas, que pode ser utilizado, dentre outras aplicações, para acesso a informações bancárias. O reconhecimento de emoções (EKMAN, 1999) através da análise de vídeos vem sendo desenvolvido e utilizado por algumas empresas para identificar a reação dos clientes quando lhe são apresentadas novas peças publicitárias que serão lançadas no mercado (AFFECTIVA, 2016). O robô *Pepper* (SOFTBANK, 2014) mapeia o ambiente através de sinais de vídeo e tem a capacidade de reconhecer expressões faciais e interagir com pessoas no ambiente que está inserido.

Esses são exemplos de como o processamento de imagens e vídeos pode ser utilizado em diversas aplicações comerciais e de segurança. Para construção de qualquer um dos sistemas supracitados deve ser aplicado um conjunto de técnicas e ferramentas que varia de acordo com o objetivo a ser alcançado. A segmentação de imagens e *frames* de vídeo é uma etapa crucial e indispensável para a construção de todas as aplicações que foram anteriormente citadas.

Implementar a segmentação de uma imagem ou vídeo pode não ser uma tarefa trivial, pois determinar o que deve ou não ser segmentado pode ser um grande desafio. Tal desafio pode ganhar uma proporção ainda maior quando parte da imagem ou vídeo que deve ser considerada como plano de fundo ou *background*, que são regiões das imagens ou *frames* que não são de interesse das aplicações de processamento de imagens e vídeos, apresenta um variado conjunto de texturas, o que pode dificultar a identificação desse tipo de *background* (GONZALEZ e WOODS, 2010). Um bom algoritmo de segmentação, seja ele de imagem ou de vídeo, deve ser capaz de separar de forma satisfatória as regiões de interesse daquelas consideradas irrelevantes.

Uma das mais importantes operações relacionadas à visão computacional é a detecção de objetos, que necessita obrigatoriamente da segmentação de imagens ou vídeos (KRYJAC et al., 2012). A detecção de objetos ou pessoas pode ser utilizada em uma variada

gama de aplicações. A detecção de pessoas, por exemplo, pode ser utilizada por um sistema de segurança para identificar a movimentação em áreas com acesso limitado. A detecção de carros, por sua vez, pode ser utilizada para monitoramento de tráfego em vias públicas ou para identificar infrações de trânsito. Esses são dois dos inúmeros exemplos que podem ser citados para destacar o quão importante uma boa segmentação pode ser.

Um bom algoritmo de segmentação pode ser composto por um grande conjunto de etapas que, por sua vez, podem demandar um grande esforço computacional. Em muitas aplicações existe a necessidade de processamento de sinais de vídeo em tempo real, portanto, o tempo total gasto na segmentação, que geralmente é uma das primeiras etapas de processamento a ser realizada, não deve prejudicar o desempenho total de um sistema desenvolvido para o processamento de sequências de vídeo. Neste contexto, ganham importância o tipo de segmentação a ser realizada e o dispositivo computacional que é empregado para processar tal segmentação.

A segmentação de vídeos através da modelagem e subtração de *background* é uma técnica bem popular que vem sendo desenvolvida ao longo dos últimos anos. Nela um modelo ou representação de *background* é gerado e posteriormente subtraído de uma nova imagem que compõe a cena que está sendo avaliada. Dessa forma os objetos que não pertencem ao modelo de *background* são destacados e a segmentação pode ser executada com sucesso (ELHBIAN e EL-SAYED, 2008). Geralmente as técnicas desenvolvidas com essa abordagem obtêm um bom resultado para situações onde a câmera permanece estática e a luminosidade varia pouco ou de forma lenta.

Quando a segmentação de vídeos por modelagem e subtração de *background* é implementada é preciso levar em consideração a diferença entre algoritmos de segmentação recursivos e não recursivos. Em um algoritmo de segmentação recursivo, um modelo de *background* é gerado durante uma etapa de pré-processamento. Tal modelo deve ser atualizado durante todo o tempo de processamento para que mudanças de iluminação ou mudanças na composição do *background* não afetem negativamente o desempenho do sistema. Podem ser citados como alguns dos principais algoritmos de segmentação recursivos: *Multiple of Gaussian* (STAUFFER e GRIMSON., 1999), *Codebook* (KIM et al., 2005), *Clustering* (BUTLER et al., 2003) e o *Método Sigma-Delta* (MCFARLANE e SCHOLFIELD, 1995).

Nos algoritmos não recursivos de subtração de *background* não existe a necessidade de um pré-processamento para gerar um modelo de plano de fundo e, consequentemente, não existe a necessidade de atualização de tal modelo durante as diferentes etapas de

processamento. Nesse tipo de algoritmo todo movimento de objetos presentes em uma sequência de vídeo é modelado a partir de um conjunto t de *frames* que compõem tal sequência. Cada novo *frame* avaliado pelo algoritmo de segmentação também é utilizado para modelar o movimento presente nos *pixels*. Tal modelo de movimento, por sua fez, será utilizado para avaliar a movimentação no próximo *frame* da sequência de vídeo que está sendo processada.

Nesta dissertação apresentamos a implementação, em um dispositivo lógico reconfigurável, de um algoritmo não recursivo (LI et al., 2009) para segmentação de *frames* de vídeo. Esta técnica modela a quantidade de movimento nos pixels através da análise do espectro de fase de Fourier. O algoritmo proposto por (LI et al., 2009) foi escolhido para ser implementado nesta dissertação por apresentar a capacidade de paralelismo operacional, ou seja, o mesmo pode ser acelerado através da execução paralela de algumas de suas operações, e por apresentar resultados satisfatórios quando existe variação de iluminação durante a captação do vídeo a ser processado.

No algoritmo proposto por (LI et al., 2009) o maior esforço computacional é desprendido para análise no domínio da frequência, portanto, a aceleração de tal análise leva a uma diminuição global de todo o tempo de processamento gasto na segmentação de frames de vídeo. Dessa forma, uma das principais motivações deste trabalho foi desenvolver uma arquitetura para segmentação de vídeos capaz de executar de forma paralela e rápida a análise no domínio da frequência proposta por (LI et al., 2009).

Devemos levar em conta, quando consideramos a possibilidade de implementação em dispositivos reconfiguráveis como os *Field Programmable Gate Arrays (FPGA)*, de um algoritmo não recursivo, que este tipo de algoritmo pode demandar um grande tráfego de dados entre o dispositivo e sua memória externa. Para que isso não ocorra é necessário o desenvolvimento de técnicas de armazenamento interno e acesso otimizado aos dados que são necessários durante o processamento. Por outro lado, os algoritmos recursivos podem gerar modelos complexos de *background* que precisam ser constantemente atualizados o que pode exigir um grande volume de operações que, se implementadas em *FPGA*, podem requerer uma grande quantidade de recursos lógicos.

A principal motivação para o desenvolvimento deste trabalho foi, portanto, construir um sistema eficiente, baseado em *CPU* e *FPGA*, capaz de segmentar uma grande quantidade de *frames* pertencentes a uma sequência de vídeo no menor espaço de tempo possível. Tal sistema deve ter uma grande capacidade computacional, para ser empregada, por exemplo, no processamento de sinais de vídeo captados por câmeras de segurança, que costumeiramente

são utilizadas para o monitoramento de pedestres e do tráfego de carros. Assim, a segmentação dos *frames* poderia ser realizada rapidamente pelo sistema aqui proposto e os recursos computacionais, que eram dispensados para a sua execução, poderiam ser direcionados à realização de outro tipo de processamento. Normalmente em arquiteturas como esta, um *host*, uma *CPU* de propósito geral, gerencia a plataforma, enquanto que o componente acelerador, neste caso um *FPGA*, funciona como um coprocessador de propósito específico.

A computação reconfigurável é uma linha de pesquisa que vem crescendo proporcionalmente ao incremento de complexidade dos *FPGAs* disponíveis no mercado (DUTRA, 2010). Cada vez mais os principais fabricantes, Xilinx e Altera, se esforçam para desenvolver circuitos com maior quantidade de elementos lógicos, dispositivos rápidos para processamento de sinais digitais (*DSP*) (ALTERA, 2010a) e memória interna. Isso permite que possam ser desenvolvidos elementos complexos baseados em *FPGA* para a computação paralela e eficiente de alguns algoritmos a serem executado sobre uma grande massa de dados.

Os trabalhos desenvolvidos por (ROCHA, 2010) e (MEDEIROS et al., 2012) são exemplos de como o *FPGA* pode ser utilizado para computação massiva de dados. Tais trabalhos descrevem sistemas de computação heterogêneos, implementados em um computador de propósito geral e *FPGA*, para processamento de dados sísmicos. Para o processador foram alocadas as tarefas de divisão, envio e recepção dos dados, enquanto que o *FPGA* fica responsável por executar o algoritmo de migração reversa no tempo (FERNANDES et al., 2009) para análise sísmica dos dados que recebe.

Alguns pesquisadores tem se debruçado a desenvolver sistemas baseados em *CPU* e *FPGA* para detectar movimento e segmentar de forma eficiente *frames* em uma sequência de vídeo. Os trabalhos desenvolvidos por (GENOVESE e NAPOLI, 2013), (KRYJAK et al., 2012) e (GOMEZ et al., 2012) apresentam exemplos de sistemas onde uma parte da computação necessária para a detecção de movimento e a segmentação dos *frames* de vídeo é desviada para ser executada no *FPGA* enquanto outra parte da computação fica restrita ao processamento via *CPU*.

O *FPGA* também pode ser utilizado para acelerar a análise de impressões digitais (LINDOSO e ETRENA, 2007) ou para calcular de forma otimizada uma *FFT* bidimensional sobre imagens com elevada quantidade de *pixels* (CHI-LI e LANPING, 2009). Esses são exemplos de como plataformas reconfiguráveis podem ser utilizadas para desenvolver

diferentes tipos de aplicações capazes de calcular de forma acelerada algoritmos com grande custo computacional.

A partir deste contexto, neste trabalho é apresentado o desenvolvimento e a implementação de um sistema heterogêneo, baseado em *CPU* e *FPGA*, capaz de executar de forma paralela e eficiente a segmentação de *frames* de vídeos. Este sistema é formado por uma arquitetura onde um processador de propósito geral trabalha de forma conjunta com um *FPGA* para acelerar a computação do algoritmo de segmentação de vídeos proposto por (LI et al., 2009). Neste algoritmo, que é não recursivo, a movimentação em cada *pixel* de um *frame* em uma sequência de vídeo é quantizada através do espectro de fase de Fourier, que é resultante da aplicação da *FFT 3-D* sobre um conjunto t de frames de entrada. A matriz resultante do processo de determinação do espectro de fase é então utilizada com entrada para o cálculo de uma *FFT 2-D*. O resultado deste cálculo é uma matriz com dados complexos que são utilizados para estimar a quantidade de movimento relacionado a cada *pixel*.

No sistema proposto nesta dissertação o *FPGA* é utilizado como uma tecnologia de computação massiva de dados. Sendo tarefa do dispositivo reconfigurável calcular a *FFT 3-D*, o espectro de fase e a *IFFT 2-D* a partir de uma sequência de *frames* de vídeo de entrada, uma vez que estas são as partes mais custosas do algoritmo implementado (LI et al., 2009). No decorrer deste texto serão descritos os conceitos e os detalhes envolvidos no projeto bem como no desenvolvimento de uma arquitetura em hardware reconfigurável para segmentação de frames de vídeos.

1.1. Objetivos do trabalho

O principal objetivo deste trabalho foi projetar e desenvolver um sistema de arquitetura heterogênea, baseada em *CPU* e *FPGA*, de alto desempenho computacional, para a computação em tempo real do algoritmo de segmentação de sinais de vídeo desenvolvido por (LI et al., 2009). A realização dos objetivos secundários elencados a seguir foi fundamental para a implementação e validação da arquitetura apresentada no quarto capítulo desta dissertação:

a. Projetar e implementar um *IP-Core* próprio para o cálculo da *FFT/IFFT* unidimensional. Tal *IP-Core* foi implementado para que não fosse preciso executar uma mudança na representação dos dados de entrada do sistema, uma vez que, os *IP-*

cores disponíveis para o desenvolvimento da arquitetura proposta neste trabalho realizavam o cálculo da *FFT/IFFT* sobre dados inteiros ou representados como números de ponto fixo. Assim, manter a precisão dos dados de entrada inalterada evita a inserção de ruídos que podem ser gerados, durante as etapas de processamento, pela modificação de tal representação.

b. Projetar e implementar o armazenamento eficiente dos resultados intermediários que seriam gerados entre as diferentes etapas do processamento que são necessárias para a execução do algoritmo proposto por (LI et al., 2009). Isso evitaria a leitura e escrita de uma quantidade massiva de dados na memória externa do *FPGA*, que, neste caso, já estariam internamente armazenados no *FPGA* e disponíveis para o uso nas diversas etapas do processamento.
c. Desenvolver e utilizar na arquitetura o armazenamento em *ROM* dos resultados de operações aritméticas que são executadas sobre números com limitado range valores. Isso poderia reduzir a quantidade de *DSP* utilizada na implementação deste trabalho, assim como poderia reduzir a quantidade de lógica dispensada para a implementação de tais operações.
d. Implementar e executar com conjunto de estudos de caso para avaliar a velocidade operacional e a qualidade dos resultados produzidos pela arquitetura para segmentação de sinais de vídeo desenvolvida neste trabalho.
e. Implementar e executar o conjunto de testes que deveriam ser realizados para a definição dos parâmetros de configuração da arquitetura para segmentação de vídeos.

1.2. Organização da Dissertação

O restante desta dissertação está organizado da seguinte forma: no segundo capítulo é apresentada a fundamentação teórica que será útil para o entendimento dos principais conceitos relacionados ao tema desta dissertação. No terceiro capítulo são apresentados os principais trabalhos que estão relacionados com o trabalho desenvolvido e são destacados os seus aspectos mais relevantes. A partir de tais aspectos é fundamentada a implementação do processador de propósito específico para a segmentação de *frames* de vídeo proposto nesta dissertação.

No quarto capítulo são apresentados em detalhes à concepção e o desenvolvimento de uma arquitetura heterogênea, baseada em *CPU* e *FPGA*, para segmentação de vídeos. O

quinto capítulo aborda os experimentos e testes aos quais a arquitetura desenvolvida foi submetida. Assim, no quinto capítulo fica demonstrada a velocidade operacional e a qualidade dos resultados produzidos pelo sistema proposto nesta dissertação. Por último, o sexto capítulo apresenta as conclusões que são retiradas a partir do que foi discutido neste texto e levanta sugestões de trabalhos futuros para dar seguimento a este trabalho.

2. Fundamentação Teórica

Neste capítulo são apresentados os conceitos que formaram a base para o desenvolvimento deste trabalho. O algoritmo para segmentação de vídeos (LI, et al., 2009) no qual o movimento de objetos é modelado através do espectro de fase de Fourier, que foi implementado em *FPGA*, é analisado na parte inicial deste capítulo. Em seguida é feita uma análise do tempo gasto na execução de cada parte do algoritmo de segmentação e como a aceleração de uma dessas partes pode influenciar positivamente no desempenho total do mesmo. Logo depois a Transformada Discreta de Fourier é apresentada, em sua forma direta e inversa, juntamente com algumas de suas propriedades mais relevantes para esse trabalho. Além disso, será apresentado a *FFT/IFFT* que é o algoritmo para cálculo rápido da *DFT/IDFT*. Por fim, o conceito *de Field Programmable Gate Array (FPGA)* e suas principais características são abordados.

2.1. A Transformada Discreta de Fourier

A Transformada de Fourier em sua forma contínua ou discreta *(DFT)* é uma ferramenta poderosa que pode ser empregada para resolver problemas na área de matemática, ciência e engenharia. Na engenharia elétrica, por exemplo, a análise de Fourier é fundamental para o estudo e desenvolvimento de técnicas de comunicação e estudo de antenas, além de ser empregada em processamento de diversos tipos de sinais (LATHI, 2007). Os avanços na computação digital fizeram a Transformada Discreta de Fourier ganhar uma importância ainda maior, pois ela é a ferramenta essencial para a análise de sinais digitais no domínio da frequência. Porém, a complexidade computacional, que chega a ser $O(N^2)$ para uma transformada unidimensional, poderia tornar a análise de sinais através da Transformada de Fourier em uma tarefa de elevado custo computacional.

Por outro lado, a Transformada Rápida de Fourier *(FFT)* (COOLEY e TUKEY, 1965) foi desenvolvida para calcular, não apenas de forma mais rápida, mas também de forma menos custosa, $O(N \log N)$, a Transformada de Fourier convencional. Se valendo da propriedade da linearidade, (COOLEY e TUKEY, 1965) desenvolveram este algoritmo, onde o sinal a ser processado é dividido em partes menores; em seguida a transformada é calculada sobre essas partes menores; e então a soma dessas transformadas é a Transformada de Fourier do sinal original. Um sinal a ser processado pode, portanto, ser subdividido em partes cada vez menores para reduzir a quantidade de operações necessárias para realizar o cálculo da sua transformada. O resultado disso é uma redução de complexidade de cálculo da ordem de $O(N^2)$ para $O(N \log N)$ sem a necessidade de ser empregado qualquer método numérico de aproximação, ou seja, o resultado da *FFT* é idêntico ao que seria obtido com o cálculo direto da *DFT*.

Tal resultado permite que *DFT* seja empregada nas mais diferentes áreas da computação e da engenharia. A *FFT* é empregada, por exemplo, no desenvolvimento de um sistema para reconhecimento de íris (JAIN et al., 2012), que pode ser utilizado na área de segurança para controle de acesso a locais onde apenas pessoas autorizadas podem entrar; compressão de imagens (HU et al., 2011) e para tratamento de imagens de satélites (CHAN e LIM, 2008); na detecção de movimentos em uma sequência de vídeo (GHANEM e AHUJA, 2007).

Existem diversas implementações em software da *FFT* para computadores de propósito geral, tais como: a *FFTW* (FRIGO e JOHNSON, 1998), *Spiral* (PÜSCHEL et al.,

2005), *Intel IPP* (INTEL, 2015a) e *Intel MKL* (INTEL, 2015b). O trabalhos desenvolvidos por (FRACHETTI, 2009) e (ELEFTHERIOU et al., 2009) utilizam computadores com processadores multicore para implementar de forma otimizada o processamento da *FFT*. Para supercomputadores temos as implementações da *FFT* discutidas em (FANG et al., 2007) e para *General Propose Graphic Processor Unit (GPGPU)* a biblioteca mais conhecida é a *CuFFT* (NVIDIA, 2014). Além disso, podemos encontrar alguns trabalhos que implementam ou utilizam uma *FFT* em *FPGA* para processamento de sinais digitais (UZUN et al., 2005).

A Transformada Discreta de Fourier, muito por causa da forma de processamento rápida e econômica desenvolvida por (COOLEY e TUKEY, 1965), está presente em várias aplicações da Engenharia. Assim como outros tipos de transformadas, basicamente essa transformada decompõe um sinal em somas de várias componentes, que neste caso são componentes senoidais (LATHI, 2007). A Transformada Discreta de Fourier de uma função $f(x)$, para $u = 0,1,2, \dots, M-1$, pode ser expressa através da equação (2.1).

$$F(u) = \sum_{x=0}^{M-1} f(x) e^{-j2\pi ux/M} \tag{2.1}$$

A Transformada Inversa Discreta de Fourier de uma função $F(u)$, para $x = 0,1,2, \dots, M-1$, pode ser expressa através da equação (2.2).

$$f(x) = \frac{1}{M} \sum_{u=0}^{M-1} F(u) e^{j2\pi ux/M} \tag{2.2}$$

A partir de uma rápida análise da equação (2.1) podemos notar que o cálculo de uma amostra de $F(u)$ requer que sejam realizadas M multiplicações complexas e $M-1$ somas complexas. Continuando com essa mesma análise, observamos que para calcular a *DTF* de um sinal com M pontos são necessárias M^2 multiplicações complexas e $M(M-1)$ somas complexas, o que nos permite concluir que o número de cálculos necessários para a determinação de uma *DFT* de M pontos é da ordem de $O(M^2)$.

A *DFT 2-D* de uma função $f(x,y)$, para $u = 0,1,2, \dots, M-1$ e $v = 0,1,2, \dots, N-1$, apresentada na equação (2.3), pode ser utilizada para o caso onde é necessário operar sobre dados bidimensionais, imagens, por exemplo. Através de uma rápida análise da equação (2.3) podemos observar que são necessárias NM^2 operações de multiplicação de números

complexos para que seja computada a *DFT 2-D* a partir de uma massa de dados disposta em uma matriz com N linhas e M colunas.

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-j2\pi(ux/M+vy/N)} \tag{2.3}$$

A Transformada Inversa de Fourier de uma função $F(u,v)$, para $x = 0,1,2,\dots,M-1$ e $y = 0,1,2,\dots,N-1$, é apresentada na equação (2.4).

$$f(x,y) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} F(u,v) e^{-j2\pi(ux/M+vy/N)} \tag{2.4}$$

Os valores das exponenciais encontrados nas equações (2.1), (2.2), (2.3) e (2.4) podem ser agrupados em fatores que podem ser calculados de forma separada das demais partes da *DFT*. Tais fatores, apresentados de forma reduzida na **Tabela 1**, são também conhecidos como *twiddles*. Em aplicações onde o tempo de execução é crucial os *twiddles* podem ser calculados previamente e acessados através de tabelas em tempo de execução, diminuindo assim o tempo total gasto para o cálculo da *DFT* e da *IDFT*. Mais adiante neste capítulo é apresentado como a arquitetura para segmentação de vídeos proposta nesta dissertação é capaz de tirar proveito da possibilidade de cálculo prévio dos *twiddles* para diminuir a quantidade de recursos de um *FPGA* utilizado em sua implementação.

Tabela 1 - Fatores *twiddles* da DFT/IDFT

$e^{-j2\pi ux/M}$	W_M^x
$e^{j2\pi ux/M}$	W_x^M
$e^{-j2\pi vy/N)}$	W_N^y
$e^{j2\pi vy/N)}$	W_y^N

Quando é necessário calcular a *DFT* para o processamento de uma massa de dados *3-D*, podemos utilizar a *DFT* tridimensional expressa através da equação (2.5). Podemos notar que neste caso os *twiddles* estão representados em sua forma reduzida e com uma análise simples é possível concluir que a quantidade de cálculo necessária para o processamento da

DFT 3-D é da ordem $O(MxNxP)$, ou seja, $O(M^3)$ quando as três dimensões tem o mesmo valor e são iguais a M

$$F\big((u,v,t)\big) = \sum_{x=0}^{M-1}\{W_M^x \sum_{y=0}^{N-1}[W_N^y \sum_{t=0}^{P-1} W_P^z f(x,y,z)]\} \tag{2.5}$$

Dentre todas as propriedades da Transformada de Fourier vamos destacar a propriedade de *Linearidade* e a *Simetria de Conjugado*, pois estas são as mais relevantes para o desenvolvimento deste trabalho. Para um maior aprofundamento sobre todas as propriedades da Transformada de Fourier é recomendável a consulta do livro *Sinais e Sistemas Lineares* (LATHI, 2007).

Segundo a propriedade da *Linearidade* a Transformada de Fourier de um sinal com N_0 pontos pode ser expressa através da soma de duas ou mais Transformadas de Fourier calculadas a partir de uma ou mais subdivisões dos N_0 pontos originais. Tal propriedade é expressa matematicamente da seguinte forma:

Se $x_n \Leftrightarrow X_r$ e $g_n \Leftrightarrow G_r$, então:

$$a_1 x_n + a_2 g_n = a_1 X_r + a_2 G_r \tag{2.6}$$

A *Simetria de Conjugado*, que também é conhecida como *Simetria Hermitiana* pode ser demonstrada da seguinte forma:

Da propriedade de conjugação $x^*(t) \Leftrightarrow X^*(-\omega)$, temos que:

$$x_n^* \leftrightarrow X_{-r}^* \tag{2.7}$$

A partir dessa equação e da propriedade de reversão no tempo, obtemos:

$$x_{-n}^* \leftrightarrow X_r^* \tag{2.8}$$

Quando $x(t)$ é real a propriedade de simetria de conjugado afirma que $X^*(\omega) = X(-\omega)$. Logo, para x_n real,

$$X_n^* = X_{-r} \tag{2.9}$$

Além disso, X_r é periódica com período N_0 logo:

$$X_r^* = X_{N_0 - r} \tag{2.10}$$

O resultado apresentado na equação (2.10) indica que, quando as N_0 pontos utilizados para o cálculo da Transformada de Fourier são puramente reais, precisamos calcular apenas $(N_0/2) + 1$ pontos resultantes da transformada, pois o resultado completo com N_0 pontos pode ser obtido a partir destes $(N_0/2) + 1$ pontos. Essa propriedade é muito importante para o desenvolvimento deste trabalho, pois a partir dela foi possível reduzir a quantidade de cálculo realizado dentro do *FPGA* para a computação de uma *DFT* tridimensional. Além disso, foi reduzida de forma considerável, aproximadamente 50%, a quantidade de memória interna utilizada para armazenamento de resultados intermediários entre as etapas de computação de tal *DFT* tridimensional.

A quantidade de operações com números complexos, que é necessária para calcular a *DFT* sobre qualquer massa de dados, poderia ser um para a inviabilização de seu uso em aplicações computacionais que requerem bom desempenho computacional. A *FFT* é o que torna a *DFT* acessível para o processamento digital de sinais, pois esta reduz o número necessário de cálculos para a computação de uma *DFT* unidimensional da ordem $O(N^2)$ de para $O(N \log N)$ (LATHI, 2007).

A economia de recursos obtida com o uso da *FFT* unidimensional também pode ser alcançada com a *FFT 2-D* e a *FFT 3-D*, uma vez que o cálculo de uma *FFT* bidimensional e uma *FFT* tridimensional pode ser computado através de aplicações sucessivas da *FFT* unidimensional nas diferentes dimensões dos dados (GONZALEZ e WOODS, 2010). Devemos destacar que a economia de recursos obtida com o uso da *FFT* também é alcançada quando se faz necessário o cálculo da *IDFT* e a *IFFT* é o algoritmo escolhido para o sua computação.

A *Linearidade* da Transformada de Fourier também é o fator determinante para que o algoritmo da *FFT* possa computar, sem nenhuma aproximação, o valor da *DFT* com uma quantidade de operações reduzida. Por exemplo, considerando um sinal $f(x_n)$, é possível calcular a sua transformada a partir da soma das transformadas de $f(x_r)$ e $f(x_l)$, desde que a

soma de $f(x_r)$ e $f(x_l)$ resulte no sinal original. Apesar de existirem diversas variações do algoritmo original de (COOLEY e TUKEY, 1965), podemos agrupar tais variações em dois tipos básicos: decimação em tempo e decimação em frequência (LATHI, 2007).

Segundo (LATHI, 2007), no algoritmo de decimação no tempo, podemos dividir a sequência de dados x_n com N_0 pontos em duas sequências com $N_o/2$ pontos constituídas pela parte par e ímpar do sinal original como segue:

$$\underbrace{x_0, x_2, x_4, \dots, x_{N_0-2}}_{sequência\ g_n}, \underbrace{x_1, x_3, x_5, \dots, x_{N_0-1}}_{sequência\ h_n}$$

Separamos então o cálculo da transformada em duas porções:

$$X_r = \sum_{n=0}^{(N_0/2)-1} x_{2n} W_{N_0}^{2nr} + \sum_{n=0}^{(N_0/2)-1} x_{2n+1} W_{N_0}^{(2n+1)r} \tag{2.11}$$

Como $W_{N_0/2} = W_{N_0}^2$, temos:

$$X_r = \sum_{n=0}^{(N_0/2)-1} x_{2n} W_{N_0/2}^{nr} + W_{N_0}^r \sum_{n=0}^{(N_0/2)-1} x_{2n+1} W_{N_0/2}^{nr} \tag{2.12}$$

$$= G_r + W_{N_0}^r H_r \quad 0 \le r \le N_0 - 1$$

na qual G_r e H_r são as Transformadas de Fourier de $N_0/2$ pontos das sequências de números pares e ímpares, g_n e h_n. Além disso, como G_r e H_r são Transformadas de Fourier de $N_0/2$ pontos, possuem período $N_0/2$. Logo:

$$G_{r+(N_0/2)} = G_r \tag{2.13}$$

$$H_{r+(N_0/2)} = H_r$$

Além disso,

$$W_{N_0}^{r+(N_0/2)} = W_{N_0}^{N_0/2} W_{N_0}^r = e^{-j\pi} W_{N_0}^r = -W_{N_0}^r \tag{2.14}$$

A partir das equações (2.12), (2.13) e (2.14), obtemos (LATHI; 2007):

$$X_{r+(N_0/2)} = G_r - W_{N_0}^r H_r \qquad (2.15)$$

A partir do que foi apresentado acima, podemos, por conta da propriedade da linearidade, reduzir a quantidade de cálculos necessários para calcular uma *DFT* de N_0 pontos. Para isso, primeiro calculamos os $N_0/2$ pontos de X_r usando a equação (2.12) e os últimos $N_0/2$ pontos usando a equação (2.15). Portanto, o número necessário de cálculos para a computação de uma DFT unidimensional de N_0 pontos passa da ordem $O(N^2)$ de para a ordem $O(N\log N)$.

Como exemplo dessa redução da necessidade de operações considere um sinal com $N_0 = 128$ pontos. Para o cálculo de uma *DFT* sem a utilização da *FFT* seriam necessárias 16.384 (128^2) multiplicações complexas e 16.256 $(128(128-1))$ adições complexas. Usando a ideia por trás da *FFT* poderíamos separar o sinal original em duas porções de $N_0/2 = 64$ pontos. Assim, para cada *DFT* de 64 pontos, seriam necessárias 4.096 multiplicações e 4.032 adições. Neste caso, para promover o mesmo cálculo da *DFT* de um sinal com $N_0 = 128$ pontos, seriam necessárias 8.192 multiplicações de 8.064 adições. Isto representa um número de operações consideravelmente menor do que o necessário originalmente. Este número pode ser reduzido por seguidas subdivisões dos pontos em partes menores até chegar, por exemplo, em uma *DFT* de 2 ou 4 pontos.

2.2. Algoritmo para Segmentação de Vídeos

O artigo "*Dynamic Texture Segmentation Using Fourier Transform"* (LI et. al, 2009) apresenta uma representação simples e eficiente da movimentação de objetos em uma sequência de vídeo. Inicialmente os autores definem como texturas dinâmicas variações temporárias em sequências de frames de vídeos, que apresentam certas propriedades espaciais e temporais, e que podem ser modeladas de diferentes formas. Os autores, a partir de uma modelagem das texturas dinâmicas através do espectro de fase de Fourier, propõem um algoritmo simples, porém muito eficiente, para detecção de movimentos e segmentação de vídeos. O objetivo dos autores é separar de um background complexo objetos que apresentam algum tipo de movimento.

O algoritmo desenvolvido por (LI et. al., 2009) foi escolhido para ser a base para o desenvolvimento deste trabalho de mestrado por ter apresentado bons resultados na segmentação dos vídeos com os quais foi testado, por ter um grande potencial de paralelização de suas operações e por ser um algoritmo não recursivo, ou seja, por não manipular um modelo de *background* que deve ser constantemente acessado e atualizado durante o processamento dos *frames* de vídeo. O algoritmo para segmentação de vídeo pode ser resumido em dez passos apresentados na lista a seguir.

	Algoritmo para segmentação de vídeos (LI et. al.; 2009)
1	*Abra o vídeo com N frames*
2	*Aplique um filtro Gaussiano em cada frame para a remoção de ruídos*
3	*Diminua o tamanho original dos frames para diminuir o esforço de processamento*
4	*Escolha uma profundidade temporal n para o cálculo das próximas etapas*
5	*A partir do frame 0 até o frame N agrupe n frames*
	5.1 Aplique a FFT 3-D sobre os n frames
	5.2 Sobre o frame n resultante do cálculo da FFT 3-D calcule o ângulo de fase
	5.3 A partir plano formado pelos ângulos de fase calcule a IFFT 2-D
	5.4 Armazene o resultado da IFFT 2-D para pós-processamento
	5.5 Descarte o frame mais antigo e componha os n frames com os anteriores e o frame $n + 1$
6	*Sobre os frames resultantes da IFFT 2-D construa máscaras binárias a partir de um limiar* l
7	*Aplique operações morfológicas sobre as máscaras para remoção de ruídos*
8	*Redimensione a máscara para o tamanho original do frame*
9	*Aplique as máscaras sobre os frames de entrada*
10	*Apresente o vídeo com os resultados da segmentação*

No primeiro passo o vídeo no qual será aplicado o processamento é aberto e decomposto em seus N *frames*. Quando os *frames* de entrada não estão em escala de cinza é necessário converte-los para tal representação, pois é menos custoso trabalhar com *frames* onde cada *pixel* é representado por apenas um número, ou seja, escala de cinza, do que trabalhar com *pixels* representados, por exemplo, por três componentes de cores, como no caso do espaço de cores *RGB*. Em seguida é aplicado um filtro *Gaussiano* para a remoção de ruídos que podem ser introduzidos nos dados a partir do processo de captura do vídeo a ser processado.

Após o redimensionamento dos frames para reduzir seu tamanho e, consequentemente, reduzir a quantidade de processamento necessário para a execução do algoritmo, deve ser escolhida a profundidade temporal utilizada durante a segmentação de vídeos. Aqui é importante destacar que quanto maior for o valor de n maior será a quantidade de dados a ser manipulada no quinto passo do algoritmo de segmentação de vídeos, pois o número de frames a ser processado em tal etapa será igual a n. Como no quinto passo existe a necessidade de um maior esforço computacional, é importante escolher um valor de n que

proporcione uma boa segmentação ao mesmo tempo em que não seja exigido um grande esforço computacional que torne inviável a execução do algoritmo em tempo real.

Na quinta etapa do algoritmo de segmentação a *FFT 3-D* é aplicada sobre um conjunto n de *frames* para que a análise no domínio da frequência seja realizada nas próximas etapas do algoritmo. A equação (2.16) apresenta a *FFT 3-D* que é aplicada sobre o conjunto $n = t$ de *frames* de vídeo. Neste caso, cada frame possui x linhas e y colunas. A equação (2.16) inicialmente é calculada a *FFT* na dimensão das x linhas, em seguida é calculada a *FFT* na dimensão das y colunas e por fim é calculada a *FFT* na dimensão dos t de *frames*. Neste caso, fica claro que a *FFT 3-D* pode ser obtida através do cálculo da *FFT* unidimensional em cada uma das três dimensões dos dados a serem processados. Os valores W_X^{k1}, W_Y^{k2} e W_T^{k3} encontrados na equação (2.16) são fatores de multiplicação conhecidos como *twiddles* e podem ser calculados previamente ou durante o cálculo da *FFT 3-D*.

$$F\big(I(x,y,t)\big) = \sum_{x=0}^{X-1}(W_X^{k1}\sum_{y=0}^{Y-1}(W_Y^{k2}\sum_{t=0}^{T-1}W_T^{k3}I(x,y,z))) \tag{2.16}$$

Na sequência do quinto passo de processamento o resultado da *FFT 3-D* é utilizado para extrair a informação de movimento relacionada a cada *pixel* do *frame* que primeiro entrou no pipeline temporal composto por n *frames*. Quando $n = 4$, por exemplo, e temos um total de N *frames*, a *FFT 3-D* é calculada para os quatro primeiros *frames*, a estimativa de movimento estará relacionada ao primeiro do total de *N frames*. Dessa mesma forma, a segunda estimativa de movimento vai estar relacionada ao segundo *frame* do total de *N frames* e assim por diante.

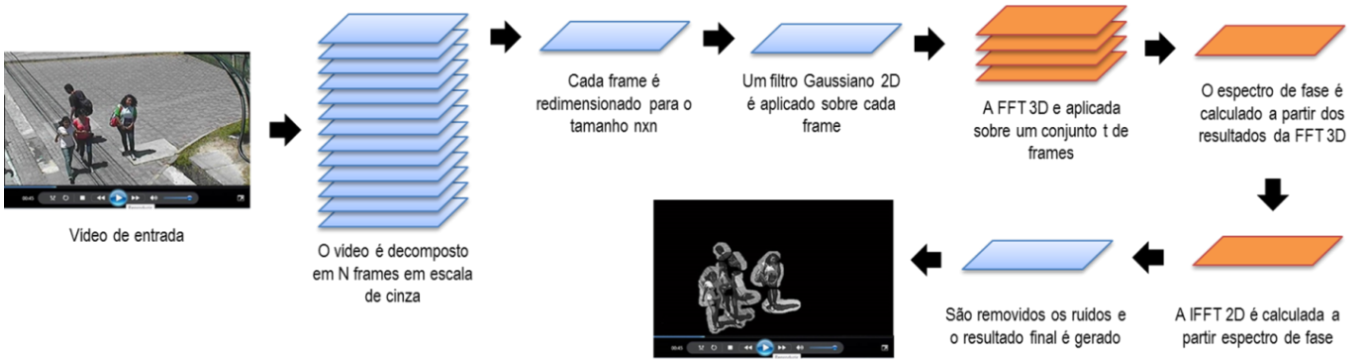
Para que seja possível estimar a quantidade de movimento em cada *pixel* é preciso calcular a *IFFT 2-D* sobre o resultado do cálculo do espectro de fase de Fourier obtido no passo 5.2. A equação (2.17) representa o restante do cálculo realizado no quinto passo do algoritmo. Nesta equação *P* é a fase obtida a partir dos valores de $F[I(x,y,t)]$ e $\hat{\mathrm{I}}(x,y,t)$ é o resultado da aplicação da *IFFT* sobre os valores de fase. Devemos destacar que os valores de *P* devem ser calculados a partir do arco-tangente de quatro quadrantes, que no *Matlab* pode ser obtido através da função *atan2* (GONZALEZ e WOODS, 2010).

$$\hat{\mathrm{I}}(x,y,t) = \left|F^{-1}\{e^{iP[F\left(I(x,y,t)\right)]}\}\right|^2 \tag{2.17}$$

No sexto passo de processamento ocorre a construção das máscaras binárias relacionadas a cada frame de entrada. Para tal construção é determinado um valor limiar l. Quando a quantidade de movimento estimada para um pixel excede tal valor de limiar é considerado que neste pixel existe movimento, portanto, na máscara binária tal pixel recebe valor igual a um. Caso contrário, não é detectado movimento relacionado ao pixel em questão e na máscara binária tal pixel receberá valor igual a zero.

No sétimo passo são aplicadas operações morfológicas sobre as máscaras binárias para remover os ruídos que podem ter sido inseridos durante todo processamento realizado nos passos anteriores. No oitavo passo as máscaras binárias são redimensionadas para ter o mesmo tamanho dos frames originais aos quais estão relacionadas. No nono e no décimo passo as máscaras binárias são aplicadas nos frames de entrada para gerar o resultado final do algoritmo. Na **Figura 1** é ilustrado graficamente o algoritmo proposto por (LI et al.; 2009) e as etapas supracitadas.

Figura 1 - Etapas do algoritmo para segmentação de vídeos



Para avaliar a eficiência computacional do algoritmo de segmentação de vídeos proposto em (LI et al., 2009), foram aplicados os dez passos de processamento em vídeos capturados através de câmeras de monitoramento de trânsito. O vídeo, que possui frames em *RGB*, foi processado com a intenção de detectar pessoas andando em uma calçada. O resultado obtido com a aplicação do algoritmo pode ser observado na **Figura 2**. Podemos notar que foi alcançado o objetivo segmentar as partes do vídeo onde as pessoas estão andando, ou seja, foi detectado o movimento dos pedestres na sequência de *frames* e estes foram corretamente segmentados. Esse resultado foi alcançado através da escolha da profundidade temporal igual a 6, que é a mesma adotada em (LI et. al., 2009). Além disso, para diminuir a quantidade de dados a serem processados em cada etapa do algoritmo, os

frames de entrada, que tinham dimensão igual a $480x704$ *pixels*, tiveram seu tamanho reduzido para $240x352$ *pixels*.

Figura 2 - Segmentação das regiões com movimentos no vídeo

Durante a execução da rotina computacional, que implementa o algoritmo de segmentação de vídeos (LI et al., 2009), foram medidos os tempos consumidos em cada uma das etapas que foram relacionadas no início deste capítulo. Na **Tabela 2** são apresentados os tempos gastos em cada parte do algoritmo de segmentação. Podemos notar que o tempo para calcular a quinta etapa do processamento corresponde a aproximadamente 94% de todo o tempo gasto para a execução da segmentação. Para mensurar esses tempos, foi desenvolvida uma aplicação em *C++* que utiliza da biblioteca *FFTW* (INTEL, 2016) para o cálculo da Transformada Rápida de Fourier e o algoritmo de segmentação foi executado sobre um vídeo com aproximadamente 3000 *frames* de resolução igual a $480x704$ *pixels*.

Tabela 2 - Tempo gasto na execução do algoritmo de segmentação

	Etapa 1	Etapa 2	Etapa 3	Etapa 4	Etapa 5	Etapa 6	Etapa 7	Etapa 8	Etapa 9	Etapa 10	Total
Tempo (seg)	0,08	0,06	0,08	0	11,89	0,15	0,21	0,08	0	0,05	12,6

A Lei de Amdahl (HENNESSY e PATTERSON, 2010) pode ser utilizada para avaliar o impacto que a melhoria da quinta etapa do algoritmo, a mais custosa, tem sobre o desempenho de total o mesmo. Se, por exemplo, esta etapa for melhorada em apenas duas vezes, o tempo total de processamento e acelerado em $1{,}89x$. Como o percentual do tempo gasto na análise no domínio da frequência, quinta etapa do algoritmo, é muito elevado, a aceleração de tal etapa leva a uma aceleração quase igual de todo o tempo de processamento. Esse foi um dos motivos que levaram ao desenvolvimento da arquitetura para segmentação de vídeos que é apresentada neste trabalho e que foi projetada para calcular de forma eficiente a quinta etapa algoritmo supracitado (LI et al., 2009).

No quarto capítulo deste texto são apresentadas as variações feitas no algoritmo de segmentação de vídeos original para tornar mais simples a sua implementação em uma plataforma reconfigurável. Tais modificações não alteram a quantidade de cálculos necessários para a obtenção dos resultados finais, muito menos modificam as ferramentas utilizadas para o processamento dos sinais digitais presentes em cada *frame*.

2.3. Field Programmable Gate Array *(FPGA)*

O primeiro *Field Programmable Gate Array (FPGA)* foi desenvolvido e lançado pela Xilinx Inc. em 1985 (XILINX, 2012b). A ideia que motivou o projeto do primeiro *FPGA* foi a de desenvolver um dispositivo de hardware que pudesse ser configurado diversas vezes para implementar as funcionalidades de diferentes tipos de circuitos. Tal configuração seria feita via software e poderia ser aplicada na implementação rápida de protótipos de circuitos eletrônicos em geral (GOKHALE e GRAHAM, 2005).

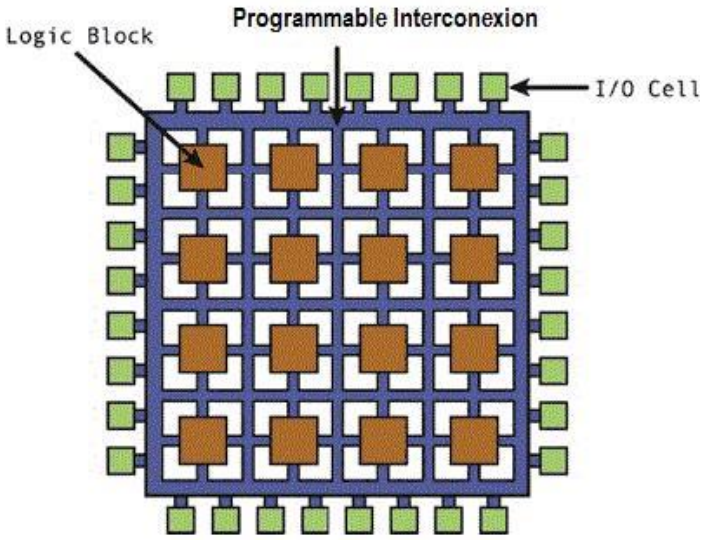
Com o passar do tempo a tecnologia de fabricação dos *FPGAs* ficou mais sofisticada. Outras empresas, dentre elas podemos destacar a Altera, passaram a dedicar seus esforços para fabricar *FPGAs* com uma maior quantidade de recursos que poderiam ser utilizados por seus programadores. Dentro deste contexto surgiu uma nova linha de pesquisa e desenvolvimento de projetos, a computação reconfigurável, na qual o *FPGA* é a tecnologia de prototipação básica para o desenvolvimento de pesquisas e projetos. Geralmente, nesta linha de pesquisa são desenvolvidas aplicações nas quais diferentes tipos de algoritmos, em sua totalidade ou de forma parcial, são implementados em *FPGA*. Dessa forma o *FPGA* pode atuar como um elemento de computação específico capaz de processar de forma paralela uma quantidade massiva de dados.

Atualmente os *FPGAs* são utilizados em aplicações relacionadas à área de *High-Performance Computing* (ALTERA, 2013). Dentre tais aplicações podemos citar uma arquitetura heterogênea, baseada *CPU* e *FPGA*, que é capaz de processar de forma paralela e eficiente dados sísmicos (ROCHA, 2010). Na área de processamento digital de sinais os *FPGAs* são utilizados em diferentes tipos de aplicações. Como exemplo disso, podemos citar a implementação de um algoritmo para segmentação de vídeos em *FPGA*, onde o movimento de objetos é modelado através de *Misturas Gaussianas* (GENOVESE e NAPOLI, 2013). Podemos citar, como exemplo de uso do *FPGA* como um dispositivo de processamento

específico, a implementação de uma arquitetura capaz de executar o cálculo de uma *DFT 2-D* a partir de uma massa de dados bidimensional (CHI-LI e LANPING, 2009).

O *FPGA* é constituído internamente por três estruturas básicas: blocos lógicos, elementos de roteamento e blocos de entrada e saída. A ativação e interconexão entre essas estruturas básicas é o que permite que no *FPGA* seja implementada uma grande variedade de circuitos para os mais diferentes propósitos. A **Figura 3** mostra de forma simples a estrutura interna de um *FPGA*.

Figura 3 - Estrutura interna de um FPGA



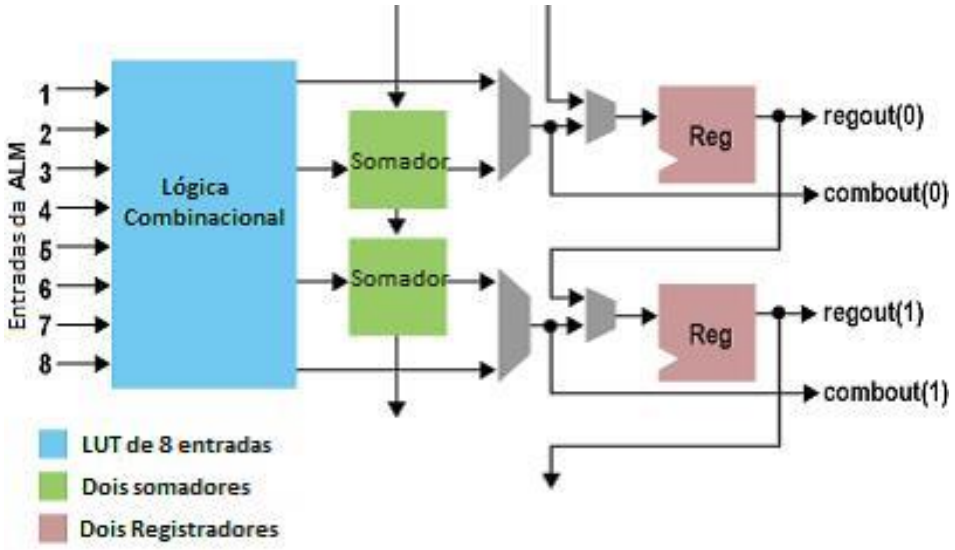
Fonte: (DUTRA; 2010)

O *Array of Logical Blocks* (ALTERA, 2009a) e o *Configurable Logical Block (CLB)* (XILINX, 2012b) são os nomes dados aos blocos lógicos que compõem um *FPGA* da Altera e da Xilinx, respectivamente. Os blocos lógicos, de uma forma geral, são constituídos por uma unidade lógica combinacional programável e um conjunto de somadores, registradores, *flip-flops* e *latches*. Cada fabricante determina a organização interna do seu bloco lógico e seu comportamento, mas na maioria das vezes a estrutura interna dos blocos lógicos pode ser representada através da **Figura 4**. Existem diferentes formas de implementar a unidade lógica combinacional. O método mais comum consiste em usar *look-up tables (LUTs)* (ALTERA, 2009b), que são células de memórias conectadas a um multiplexador que seleciona uma de tais células como a saída da unidade lógica combinacional. Na *LUT* ficam armazenadas possíveis respostas de partes de um sistema a um conjunto de estímulos. O sinal com tais estímulos é utilizado como o seletor do multiplexador que fica localizado na saída da unidade

lógica combinacional. Assim, de acordo com o estímulo, é selecionada a resposta correta a ser encaminhada para a saída da unidade lógica combinacional (ALTERA, 2009a).

Os *FPGAs* possuem blocos de memória embarcados *(BRAM)* que ficam conectadas aos blocos lógicos e que podem ser configurados para funcionar como *SRAM* ou *ROM*. Tais unidades de memória servem para prover o armazenamento de dados no *FPGA*. Isso é importante, pois, como é descrito no quarto capítulo, o armazenamento interno dos resultados intermediários do processamento realizado no *FPGA* elimina a necessidade de que tais resultados sejam escritos e lidos constantemente da memória externa que é conectada ao *FPGA*. Assim, todo o dado necessário para o processamento é lido apenas uma vez e o resultado e escrito apenas uma vez na memória externa conectada ao *FPGA*, o que provoca uma redução da necessidade de largura de banda de comunicação entre *FPGA* e memória.

Figura 4 - Organização interna de um bloco lógico

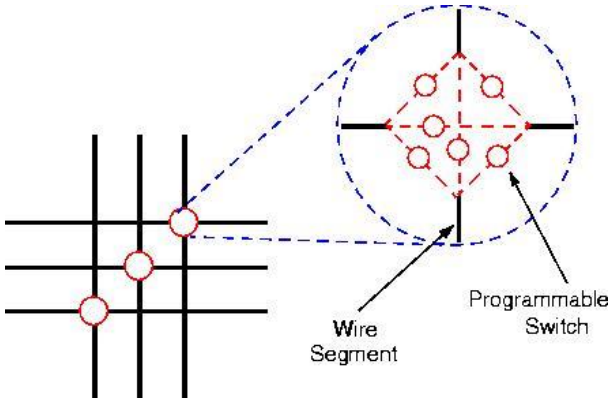


Fonte: (DUTRA; 2010)

A interconexão entre os blocos lógicos permite que sejam construídos circuitos complexos em *FPGA*, tal interconexão é provida pelos elementos de roteamento. Estes elementos são distribuídos por todo *FPGA* e são programados, na maioria das vezes, por ferramentas de software (ALTERA, 2010b), que são capazes de encontrar os melhores caminhos e conexões a serem ativados para que os recursos disponíveis no *FPGA* sejam utilizados da forma mais otimizada possível. A programação dos elementos de roteamento é feita através da configuração de uma estrutura conhecida como *Switch Matrix*, que é apresentada na **Figura 5**. Com o uso de tal estrutura, para ativar uma conexão entre dois

elementos lógicos basta ativar um ou mais caminhos que levam sinais de um elemento para o outro.

Figura 5 - Estrutura de uma Switch Matrix



Fonte: (ROCHA, 2010)

Os blocos de entrada e saída implementam a comunicação do *FPGA* com elementos externos. Através dos blocos de entrada e saída é possível enviar sinais para serem processados dentro do *FPGA* e receber as respostas geradas por tal processamento. Além das estruturas básicas discutidas anteriormente, os *FPGAs* atuais possuem ainda em sua estrutura um conjunto maior de tipos de elementos para aplicações especiais, como, por exemplo: unidades DSP (ALTERA, 2010a), para processamento rápido de sinais digitais; unidades de memória, para armazenamento interno de dados (DUTRA, 2010); canais de comunicação de alta velocidade para a troca de dados entre *FPGAs*; e cores de *CPU*, que podem ser utilizados para executar aplicações no dispositivo reconfigurável. Todos estes são fatores que têm contribuído para que este tipo de tecnologia se torne cada vez mais uma opção atraente para o desenvolvimento de projetos que seguem a linha da computação reconfigurável.

2.4. Conclusões

Neste capítulo foram apresentados os conceitos mais relevantes para o desenvolvimento desta dissertação de mestrado. Inicialmente foi abordado o algoritmo para a segmentação de vídeos apresentado no artigo "*Dynamic Texture Segmentation Using Fourier Transform"*(LI, et. al, 2009). Foi explicado de forma sucinta como o movimento de objetos em uma sequência de vídeo pode ser modelado através do espectro de fase de Fourier. Além disso, através de uma sequência de passos foram explicadas as etapas de processamento que

devem ser realizadas para que a segmentação de sinais de vídeo seja realizada de forma coerente com o explicado no artigo anteriormente citado.

Na sequência deste capítulo foi apresentado o conceito de Transformada Discreta de Fourier e foram apresentadas a propriedade da *Linearidade* e a propriedade da *Simetria Hermitiana*, que são duas das propriedades da Transformada de Fourier que foram mais relevantes para o desenvolvimento deste trabalho. Em seguida abordamos o algoritmo proposto por (COOLEY e TUKEY, 1965): a Transformada rápida de Fourier. Mostramos como tal algoritmo permite que o cálculo da *DFT* e *IDFT* seja realizado com um número reduzido de multiplicações e adições complexas. Além disso, ficou claro que a *FFT/IFFT* permite que o cálculo da *DFT/IDFT* seja realizado sem qualquer tipo de aproximação numérica.

Por fim, neste capítulo, foram apresentadas as características básicas da arquitetura interna de um *FPGA* e sua relevância para o processamento de sinais digitais com alto desempenho. Dessa forma, foram cobertos, neste capítulo, todos os conceitos básicos para a compreensão desta dissertação.

3. Trabalhos Relacionados

Neste capítulo são apresentados os principais trabalhos que estão relacionados ao tema desta dissertação. Todos estes trabalhos abordam a construção de sistemas heterogêneos baseados em computação reconfigurável, nos quais o *FPGA* atua como um processador de propósito específico, para o processamento de imagens e vídeos. No final do capítulo é apresentada uma análise comparativa entre os artigos aqui discutidos, são destacados os seus pontos principais e, a partir disso, são apresentadas as características principais do sistema proposto nesta dissertação e sua contribuição em relação aos demais.

Boa parte dos modelos de *FPGA* disponíveis atualmente no mercado conta com uma grande quantidade de blocos lógicos, memória interna, elementos para interconexão de blocos e *DSP* (DUTRA, 2010). Tudo isso permite que sejam desenvolvidas, em *FPGA*, estruturas complexas de processamento capazes de manipular vários tipos de sinais para os mais diferentes fins. Neste contexto, vários pesquisadores estão desenvolvendo trabalhos cujo tema central é a computação reconfigurável e a utilização do *FPGA* como um elemento de processamento capaz de executar computação massiva de dados de forma otimizada. Em uma busca por artigos que implementam em dispositivos reconfiguráveis processadores de sinais de imagem e vídeo é possível encontrar uma boa quantidade de resultados relevantes, dentre os quais podemos destacar os trabalhos apresentados neste capítulo.

(GENOVESE e NAPOLI, 2013) propõem a implementação em *FPGA* de um elemento de processamento capaz de segmentar *frames* de vídeo com o intuito de destacar objetos que se movimentam em uma cena, em seguida, o sinal resultante da segmentação passa por um processo de filtragem espacial, também implementado em *FPGA*, para a remoção de ruídos que podem ser gerados pelo processo de segmentação.

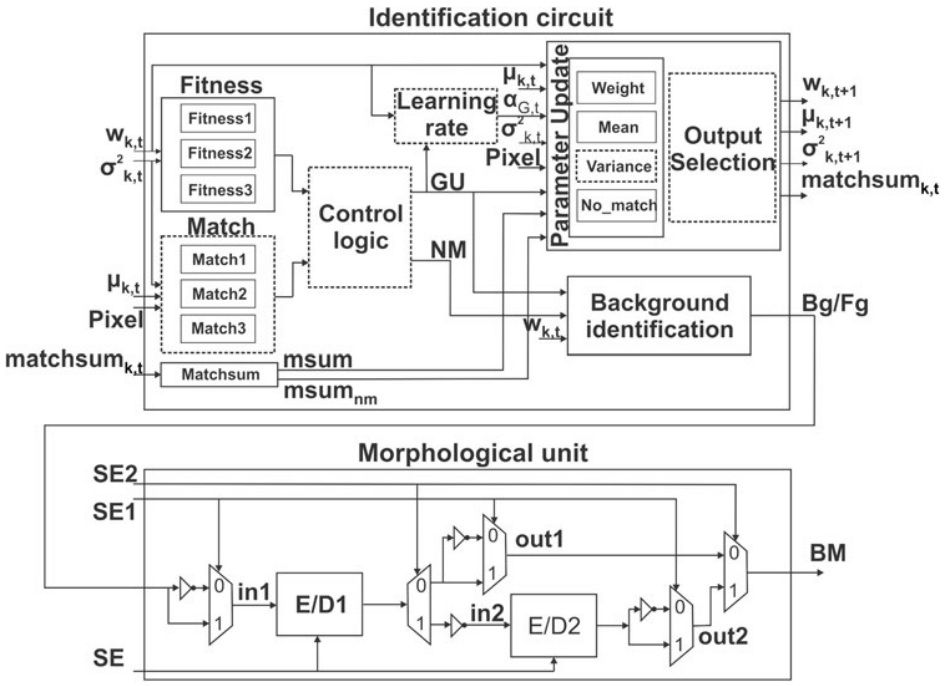
No sistema proposto por (GENOVESE e NAPOLI, 2013) a segmentação dos *frames* de vídeo, que fica a cargo de um dispositivo reconfigurável, é realizada utilizando um modelo estatístico baseado em Misturas de Gaussianas (STAUFFER e GRIMSON, 1999) para descrever o *background* e detectar regiões onde existe movimento. Foram utilizados operadores morfológicos (GONZALEZ e WOODS, 2010) para a etapa de remoção de ruídos. Tais operadores executam a erosão, a dilatação, a abertura e o fechamento sobre as imagens resultantes do processo de segmentação.

O algoritmo para segmentação de sinais de vídeo (STAUFFER e GRIMSON, 1999) é um dos componentes da biblioteca *OpenCV* (INTEL, 2014), que é um conjunto de ferramentas desenvolvido pela Intel e utilizado por desenvolvedores de todo o mundo para construir aplicações relacionadas à Visão Computacional. Em tal algoritmo um modelo de *background* é utilizado para detectar a movimentação em uma sequência de vídeo. Nesse modelo, que precisa ser construído antes do início do processamento e atualizado a partir de cada novo *frame* processado, o *background* é composto por um conjunto de *pixels* que são representados por um modelo estatístico formado pela soma de K distribuições gaussianas.

Cada distribuição gaussiana é representada por quatro parâmetros: peso (ω), média (μ), variância (σ^2) e *matchsum*, que é um contador que indica quantas vezes a distribuição foi a que teve o maior peso na classificação do *pixel* como de *background* ou não. Para desenvolver o seu trabalho (GENOVESE e NAPOLI, 2013) consideraram $K = 3$ e que a

computação executada pelo hardware seria feita de forma idêntica ao algoritmo de segmentação baseada em *GMM* que está implementada na biblioteca *OpenCV*. Na **Figura 6** temos uma visão geral do diagrama de blocos da arquitetura desenvolvida para a segmentação de sinais de vídeo.

Figura 6 - Diagrama de blocos

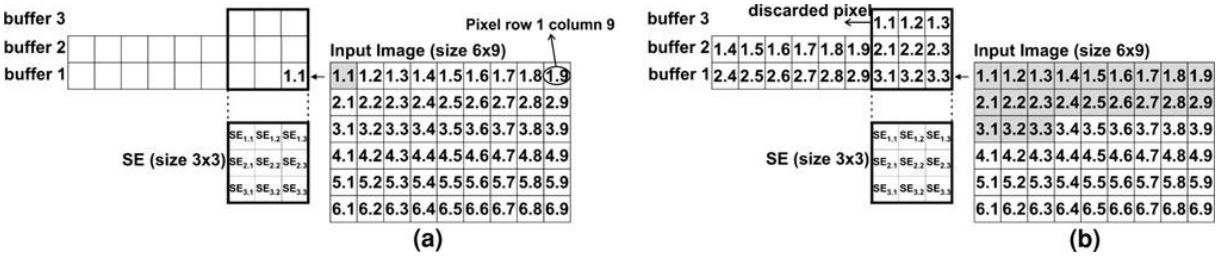


Fonte: (GENOVESE e NAPOLI, 2013)

A arquitetura é separada em dois blocos principais: *Identification Circuit*, que implementa a detecção de movimento e atualiza o modelo de *background* para cada *pixel*, e a *Morphological Unit*, que implementa a filtragem espacial para a remoção de ruídos. Devemos destacar que neste sistema toda a computação necessária para a execução do algoritmo de segmentação de vídeos baseado em *GMM*, exceto a construção inicial do modelo de background, é realizada pelo *FPGA*. Outro ponto que merece destaque é o fato de que nenhum resultado intermediário é armazenado internamente no *FPGA*, ou seja, o modelo de *background* é sempre lido da memória externa conectada ao dispositivo reconfigurável e sua atualização é sempre escrita de volta em tal memória a cada novo *frame* que é processado. Isso pode requerer uma considerável taxa de transferência entre o *FPGA* e a sua memória quando o número de distribuições gaussianas utilizadas para representar cada *pixel* cresce e por isso os autores consideraram $K = 3$.

Na *Morphological Unit* ocorre a filtragem espacial para a remoção de ruídos. Para implementar tal filtragem com apenas o operador de dilatação, os autores deste trabalho levaram em consideração que os operadores morfológicos de abertura e fechamento podem ser derivados dos operadores erosão e dilatação e que tais operadores são complementares entre si. Na **Figura 7** podemos ver que na *Morphological Unit* os dados são organizados em *buffers* de forma que a filtragem espacial é computada de forma contínua, ou seja, não são gerados atrasos para a organização da janela onde a filtragem será aplicada.

Figura 7 - Morphological Unit



Fonte: (GENOVESE e NAPOLI, 2013)

A arquitetura para a segmentação de vídeos desenvolvida por (GENOVESE e NAPOLI, 2013) foi testada em diferentes *FPGAs* da Xilinx e da Altera e o melhor resultado, uma taxa de processamento de 47 fps para *frames* com tamanho $1024x1024$ *pixels*, foi obtido em uma placa VirtexII (XC2V100). Foi necessária a adoção de algumas técnicas para que a arquitetura fosse capaz de produzir seus resultados com tal taxa de processamento. Os dados utilizados no processamento tiveram sua representação modificada de ponto flutuante, cada um composto por 32 bits, para ponto fixo, cada um composto por 16 bits. Isso foi feito para reduzir a transferência de dados entre o *FPGA* e sua memória externa. Os resultados de algumas operações, como, por exemplo, a raiz quadra que é necessária para o cálculo do modelo de *background*, foram mapeados em uma memória *ROM* que é lida durante o fluxo de processamento realizado nos módulos da *Identification Unit*. Assim, foram economizados recursos do *FPGA* que seriam gastos para a implementação de tais operações.

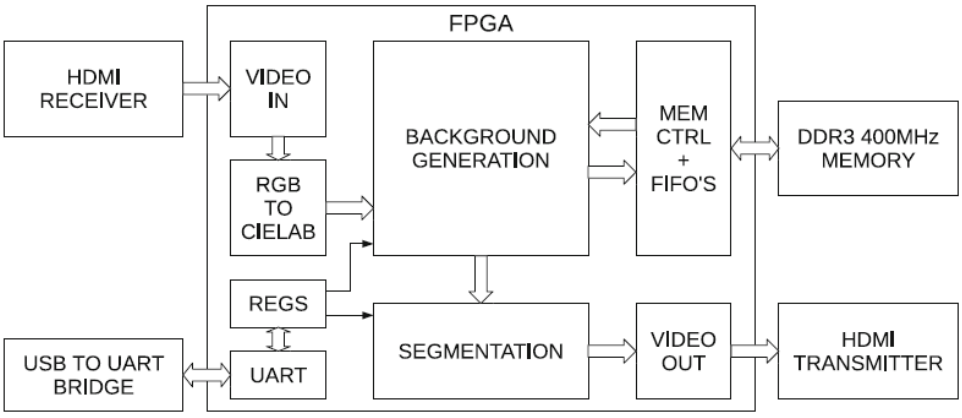
Uma nova versão desta arquitetura para segmentação de sinais vídeo foi descrita em (GENOVESE e NAPOLI, 2013). Nela foram desenvolvidas mudanças na unidade responsável por identificar o *background* e nas unidades responsáveis por promover as atualizações do modelo de *background*. Isso foi feito para adequar a arquitetura anterior à nova implementação do algoritmo de segmentação de vídeos baseado em *GMM* que foi divulgada pela Intel e incluída na biblioteca *OpenCV* (INTEL, 2014). Mesmo com tais

mudanças, a arquitetura para segmentação de vídeos alcançou o máximo de 47 fps como taxa de processamento para *frames* com tamanho $1024x1024$ *pixels*. O que é mais do que suficiente para processamento de vídeos em tempo real. O trabalho de (GENOVESE e NAPOLI, 2013) apresenta um conjunto de conceitos e técnicas que devem ser levadas em consideração na construção de um sistema para processamento de sinais em um dispositivo reconfigurável, por isso foi relevante para o desenvolvimento desta dissertação.

O trabalho desenvolvido por (KRYJAK et al., 2012) propõe uma arquitetura para segmentação de sinais de vídeo de alta definição que utiliza uma técnica de *Clustering* (BUTLER, 2007) para construir um modelo de *background*. Tal modelo é atualizado durante o processamento de uma sequência de vídeo ao mesmo tempo em que é utilizado para detectar movimentação de objetos. O sistema desenvolvido neste trabalho deve receber como entrada frames que estão codificados segundo o espaço de cores *RGB.*

As características utilizadas para agrupar e classificar cada *pixel* estão relacionadas às texturas dos elementos que compõem cada *frame*. Foi adotada uma conversão do espaço de cores *RGB* para o espaço de cores *CIE Lab* (BENEDEK, 2007), pois tal espaço mapeia as cores vermelho, verde e azul em uma componente de luminância e duas componentes de crominância. Dessa forma, o movimento dos objetos em uma cena é detectado através das componentes de crominância, por outro lado, a variação da luminosidade em uma sequência de vídeo não interfere nos resultados gerados pelo processamento, uma vez que, a componente de luminância não é utilizada para gerar o espaço de características descritoras dos *pixels*. A **Figura 8** apresenta o diagrama de blocos da arquitetura para segmentação de vídeos proposta por (KRYJAK et al., 2012).

Figura 8 - Diagrama de blocos



Fonte: (KRYJAK et al., 2012)

No sistema proposto por (KRYJAK, et al, 2012) toda computação necessária para segmentar os *frames* de vídeo, exceto na fase de construção inicial do modelo de *background*, é realizada no *FPGA*, ou seja, não existe a necessidade de que uma aplicação em software fique executando partes do algoritmo em tempo real. No diagrama de blocos da **Figura 8** a conversão para o espaço de cores *CIE Lab* (BENEDEK, 2007) é executada logo na entrada do sistema. Nesta etapa ocorre também uma conversão dos dados de entrada, que estão em formato de ponto flutuante, para um formato de ponto fixo. Isso é feito para reduzir a quantidade de elementos lógicos e *DSP* que são necessários para implementar as operações aritméticas relacionadas com a execução da transformação do espaço de cores e com as demais etapas do algoritmo de segmentação.

Neste sistema a etapa de geração de *background* é executada em um módulo distinto do utilizado para a segmentação. O componente do diagrama de blocos denominado *Backgound Generation* é responsável por ler, a cada nova entrada de *frame*, o modelo de background que está armazenado na memória externa do *FPGA*. Além disso, ele executa os cálculos necessários para a atualização do modelo de *background* e envia o resultado da classificação de cada *pixel* de entrada para o módulo responsável por fazer a segmentação. Após ser atualizado, o modelo de *background* deve ser escrito novamente na memória externa do *FPGA*, ou seja, não existe o armazenamento interno de resultados intermediários.

A análise de desempenho do algoritmo foi realizada com diferentes *FPGAs* da Xilinx. O melhor resultado foi obtido com uma plataforma dotada de um *FPGA Virtex 6* (XC6VLX240T1FF1156), na qual foi possível processar um *frame* de $1920x1080$ *pixels* a uma taxa de 60 fps. Esta taxa é mais do que a necessária para que o processamento seja executado em tempo real e isso vem a comprovar a eficiência do trabalho desenvolvido por (KRYJAK et al., 2012). Devido ao seu resultado de desempenho e por apresentar um apanhado de conceitos úteis para o desenvolvimento de sistemas de computação reconfigurável, este trabalho foi relevante para a construção desta dissertação.

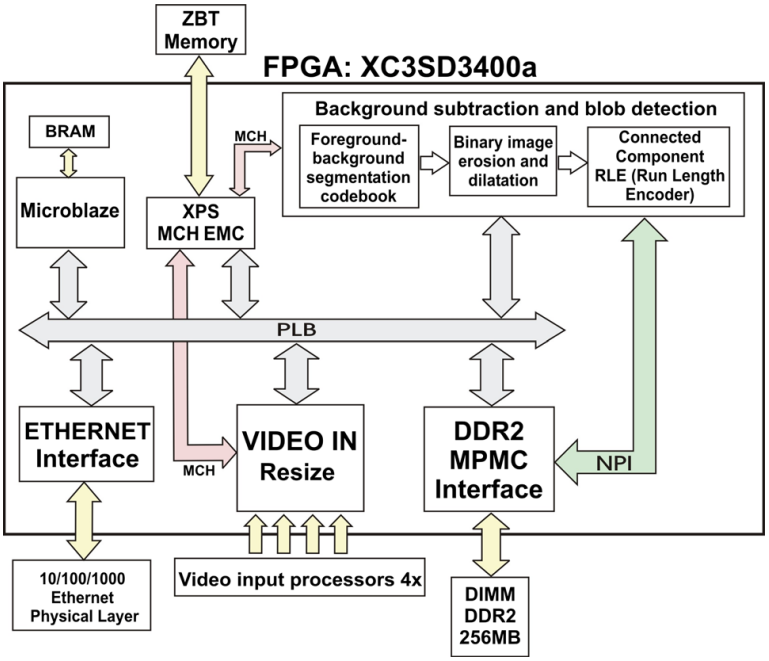
Nesta revisão bibliográfica deve ser destacado o trabalho desenvolvido por (GOMEZ et al., 2012), que propõe a construção de um sistema baseado em *FPGA* para a segmentação de sinais de vídeo. Neste trabalho o algoritmo *Codebook* (KIM et al., 2005) é implementado em *FPGA* para ser utilizado durante a etapa de subtração de *background* em uma sequência de vídeo.

No algoritmo *Codebook* cada *pixel* é representado por um *codebook*, que por sua vez é composto por um conjunto de *L codewords*. Cada *codeword* contém informações relacionadas à média das componentes *RGB* que o *pixel* apresenta durante uma sequência de

vídeo. Além disso, na *codeword* são armazenadas informações relacionadas à sua frequência de atualização e a primeira e última vez que tal atualização foi executada.

O algoritmo *Codebook* se baseia em um modelo de *background* que deve ser construído em uma etapa prévia à sua execução. O modelo de *background* deve ser formado obrigatoriamente com *frames* nos quais não existe a presença de objetos que se movimentam. No sistema proposto por (GOMEZ et al., 2012) essa etapa do processamento deve ser realizada antes do início da segmentação e o seu resultado deve ser transferido para a memória interna do *FPGA* para ser utilizado durante o processamento dos sinais de vídeo. Inicialmente o algoritmo *Codebook* considerava que o modelo de *background* não mudava durante o tempo, porém (KIM et al., 2005) desenvolveu uma forma de atualização do modelo de *background* que deve ser executado durante o processamento dos *frames* de vídeo. No sistema proposto por (GOMEZ et al., 2012) o resultado da atualização do *background* é armazenado na memória interna do *FPGA* e acessado durante todo o processamento. Na **Figura 9** é possível ver o diagrama de blocos que representa este sistema para segmentação de vídeos.

Figura 9 - Sistema para segmentação de vídeos



Fonte: (GOMEZ et al., 2012)

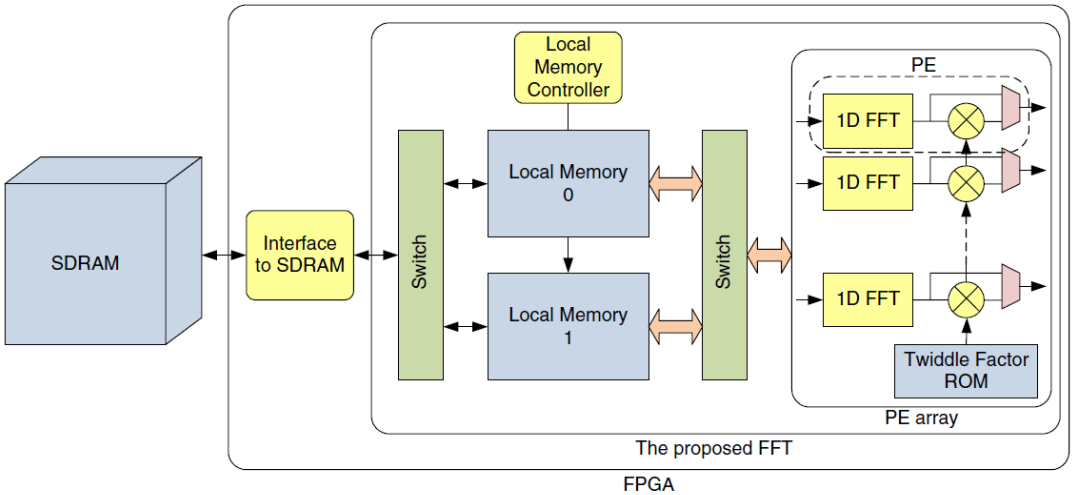
Na construção do módulo para a subtração de *background* e identificação de movimentação foi adotada a estratégia de mapear os resultados das operações de divisão e raiz quadrada em *ROM*. Assim, para ter acesso a tais resultados, é necessário indexar e ler a

memória na qual estão armazenados. Isso permite uma considerável redução de utilização de recursos do *FPGA* que seriam gastos na implementação dessas operações. O mapeamento desses resultados só é possível porque a faixa de valores de entrada é limitada entre zero e um, que é o valor atribuído a cada componente *RGB*. Transformar as entradas de ponto flutuante para uma representação de ponto fixo foi outra estratégia adotada por (GOMEZ et al., 2012) para a redução do custo de implementação.

O sistema para segmentação de vídeos foi descrito com o uso da linguagem *ImpulseC* (IMPULSEC, 2016) e implementado em um *FGPA* Xilinx (XC3SD3400aFG676). O melhor resultado obtido foi uma taxa de processamento máximo de 50 fps para *frames* de vídeo com tamanho $768x576$ *pixels*. Assim como nos outros trabalhos aqui apresentados, essa arquitetura é capaz de processar *frames* de vídeo em tempo real, o que é de grande importância para a maioria das aplicações que necessitam de detecção de movimento.

(CHI-LI e LANPING, 2009) propõe a construção de um sistema misto de processamento, composto por um computador de propósito geral e um *FPGA*, para computar a *FFT* (COOLEY e TUKEY, 1965) sobre uma massa de dados bidimensional. Tal massa de dados pode ser uma imagem ou uma sequência de *frames* de vídeo. A arquitetura do sistema proposto pode ser observada através do diagrama de blocos da **Figura 10**.

Figura 10 - Diagrama de blocos



Fonte: (CHI-LI e LANPING, 2009)

A ideia por trás do trabalho de (CHI-LI e LANPING, 2009) é proceder o cálculo da *FFT* de forma independente em cada uma das direções, linhas e colunas, dos dados de entrada. A operação de transposição deve ser aplicada sobre os dados após o cálculo da *FFT*

na direção da primeira dimensão (LENARD et al., 2008). Para isso, os resultados intermediários são armazenados nas memórias internas do *FPGA* e acessados em diferentes direções. Tal estratégia de acesso aos dados também é adotada na arquitetura apresentada neste trabalho.

Os módulos responsáveis por calcular a *FFT* na arquitetura proposta por (CHI-LI e LANPING, 2009) foram desenvolvidos pela Xilinx (XILINX, 2012a) e estão disponíveis para serem utilizados em projetos que envolvam *FPGAs* desse fabricante. Uma das características mais relevantes deste módulo é o fato de suas entradas e saídas serem formatadas em uma representação de ponto fixo, o que obrigatoriamente leva à necessidade de uma modificação de formatação dos dados quando os mesmos são números de ponto flutuante. Por isso, existe uma modificação da representação dos dados de entrada na arquitetura proposta. Como estratégia para redução de custo de implementação, os autores decidiram armazenar em *ROM* os valores dos *twiddles* que são utilizados no cálculo da *FFT 2-D*. Esta técnica também é adotada no desenvolvimento da arquitetura proposta nesta dissertação e é apresentada em detalhes no quarto capítulo.

Após o cálculo ser executado é necessário reorganizar os dados produzidos pelo *FPGA*. Esta é uma tarefa para o computador ao qual o *FPGA* está ligado. Portanto, podemos dizer que nem todo o processamento é executado pelo *FPGA*, mesmo assim o sistema proposto por (CHI-LI e LANPING, 2009) apresenta um bom desempenho. Este projeto foi implementado em uma *FPGA* Xilinx (XC5FX200T) e obteve taxa de processamento máximo de 44 fps para *frames* de vídeo com tamanho $2048x2048$ *pixels*. O que mais uma vez é uma taxa de processamento mais que suficiente para que o sistema possa ser utilizado para processamento de sinais de vídeo em tempo real.

Os trabalhos que acabaram de ser apresentados e discutidos são atualmente os mais relevantes e alinhados com o tema desta dissertação, porém alguns outros também merecem ser destacados. Em (AKOUSHDEH et al, 2012) encontramos a implementação de um sistema para processamento de imagens baseado em *FPGA* que executa de forma paralela a computação das característica relacionadas às texturas (HARALICK, 1973) presentes nas imagens que são fornecidas como entrada para o sistema. O sistema proposto é capaz de executar seu processamento 458 vezes mais rápido que um computador de propósito geral.

Em (YASRI et al., 2012) é apresentada a implementação de uma arquitetura baseada em *FPGA* para detecção de bordas em imagens. Enquanto (BRAVO et al., 2013) propõe a construção de um sistema baseado em *FPGA* para segmentação de sinais de vídeo utilizando o mesmo conceito de *GMM* utilizado por (GENOVESE e NAPOLI, 2013). Em (MORIMOTO

et al., 2006), por sua vez, é apresentada a construção de um sistema baseado também em *FPGA* para detectar formas de objetos e a partir de tal detecção promover a segmentação de uma sequência de vídeo.

Em (RANBHOR e MALI, 2015) encontra-se mais uma vez a implementação de um sistema baseado em *FPGA* para a segmentação de sinais de vídeo utilizando algoritmo de Misturas de Gaussianas *(GMM)*, o que mostra o quão esse algoritmo é largamente estudado e implementado. (FERNANDES et al., 2012) apresenta a construção de sistema de comunicação baseado em *FPGA* que implementa uma *FFT* de 128 pontos que é utilizada na modulação de sinais. Por fim, (LINDOSO e ENTRENA, 2007) utilizam um *FPGA* para construir um sistema capaz de executar o algoritmo de *Template Matching* (BROWN, 1992) para encontrar semelhanças entre imagens de impressões digitais.

3.1 Conclusões

Este capítulo fez a apresentação dos principais trabalhos relacionados ao tema desta dissertação. Foram abordados trabalhos que implementam em *FPGA* sistemas capazes de executar a segmentação de frames de vídeo. Além disso, em cada sistema analisado o *background* e o movimento eram modelados de formas distintas.

Em (GENOVESE e NAPOLI, 2013) um modelo de *background* foi calculado e atualizado através de um modelo estatístico baseado em somas de distribuições gaussianas. Foi visto que para reduzir a quantidade de recursos gastos com operações aritméticas os autores optaram por modificar a representação dos dados de entrada e por mapear em *ROM* o resultado de algumas operações. A arquitetura apresentada conseguiu alcançar uma taxa de processamento superior a 30 fps, que é fundamental para o processamento em tempo real de sinais de vídeo. Outra arquitetura que conseguiu esta taxa de processamento foi desenvolvida por (KRYJAC et al., 2012). Neste trabalho um algoritmo de *clustering* é utilizado para gerar um modelo de *background* e para classificar os *pixels* de entrada em tal arquitetura. Assim como no trabalho de (GENOVESE e NAPOLI, 2013), o sistema desenvolvido por (KRYJAC et al., 2012) necessita que o modelo de *background* seja calculado previamente e atualizado durante a execução do algoritmo de segmentação de vídeos.

Em (GOMEZ et al., 2012) uma nova arquitetura para segmentação de vídeos que utiliza a conceito de *Codebooks* para representar cada *pixel* foi abordada. Tal sistema opera armazenando nas unidades de memória do *FPGA* os resultados intermediários de

processamento. Isso é importante, pois evita que sejam realizadas sucessivas leituras e escritas na memória acoplada ao *FPGA*. Em (CHI-LI e LANPING, 2009) foi apresentado um sistema capaz de executar o cálculo da *FFT* sobre uma massa de dados bidimensional. Em tal sistema parte da computação fica a cargo do computador ao qual o *FPGA* está ligado, no entanto, isso não afeta a velocidade taxa de processamento alcançada, pois a mesma é superior à taxa de 30 fps.

A seguir será feita uma análise comparativa entre os principais trabalhos relacionados ao tema desta dissertação e, a partir dessa análise, serão apresentadas as principais características da arquitetura para segmentação de vídeos que é descrita no quarto capítulo deste texto. Os trabalhos utilizados para comparação são: (GENOVESE e NAPOLI, 2013), (KRYJAC et al., 2012), (GOMEZ et al., 2012) e (CHI-LI e LANPING, 2009). A **Tabela 3** relaciona esses trabalhos, dispostos nas colunas, com suas principais características, que foram descritas no decorrer de suas apresentações e estão dispostas nas linhas. Quando um trabalho possui uma característica é marcado um ✓ na entrada correspondente da tabela.

Na primeira linha da **Tabela 3** é destacado o fato de a arquitetura receber como entrada um modelo previamente processado que é utilizado e atualizado durante a execução do algoritmo, um modelo de *background* por exemplo. Isso pode ser danoso ao sistema, pois pode gerar a necessidade de uma maior leitura de dados e consequentemente pode requerer uma grande largura de banda para a transferência de dados entre o *FPGA* e sua memória externa. A segunda linha aborda o fato de toda a computação ser ou não realizada pelo *FPGA*. Quando a computação não é totalmente realizada no *FPGA* fica a cargo do computador executar partes do algoritmo que o sistema implementa. O sistema corre o risco de sofrer degradação de desempenho caso o software responsável por executar parte do algoritmo seja lento.

Na terceira linha é abordado o fato do sistema de processamento ser capaz de gerar seus resultados a uma taxa superior a 30 fps, que é uma taxa necessária para a execução de processamento de sinais de vídeo em tempo real. Na quarta linha é abordado o fato do sistema desenvolvido armazenar internamente os resultados intermediários do processamento. Isso pode reduzir a necessidade de leituras e escritas de dados na memória externa acoplada ao *FPGA*, no entanto o armazenamento de dados intermediários pode degradar a frequência operacional do sistema por consumir uma grande quantidade de recursos disponíveis no *FPGA*.

A quinta e a sexta linha da **Tabela 3** estão relacionadas à maneira como os operandos e operadores aritméticos são manipulados e implementados, respectivamente, durante o desenvolvimento do sistema. A modificação da representação dos dados, transformar de ponto flutuante para ponto fixo, por exemplo, pode introduzir um erro no resultado produzido pelo sistema. Cabem ao desenvolvedor as tarefas de analisar e decidir se tal erro é suportável e não afeta de forma nociva a computação realizada no *FPGA*. A utilização do mapeamento em *ROM* de operações como a raiz quadrada e a divisão, por exemplo, pode reduzir a utilização de recursos de *FPGA* empregados na implementação de tais operações. Isso pode ser feito quando a o intervalo de valores que os dados de entrada assumem é limitado. Dessa forma o resultado é limitado a um conjunto reduzido de valores que podem ser armazenados em *ROM* e acessados durante a execução do processamento no *FPGA*.

Tabela 3 - Comparação dos trabalhos relacionados

	GENOVESE e NAPOLI	KRYJAK	GOMEZ	CHI-LI e LANPING
Utiliza um modelo de *background* previamente processado	✓	✓	✓	--
Realiza toda computação em tempo real no *FPGA*	✓	✓	✓	--
Obtém uma taxa de processamento maior ou igual a 30 fps	✓	✓	✓	✓
Armazena no *FPGA* os resultados intermediários	--	--	✓	✓
Mantém a precisão original dos dados	--	--	--	--
Busca reduzir o uso de *DSP* do *FPGA*	✓	✓	✓	✓

A arquitetura para a segmentação de vídeos apresentada no próximo capítulo deste texto implementa um algoritmo para segmentação de vídeos no qual o modelo de movimento de cada *pixel* é feito através do espectro de fase de Fourier (LI et al., 2009). Tal algoritmo é composto por várias etapas de processamento e por isso parte da computação será realizada no *FPGA* e outra parte será realizada em *CPU*. Também por esse motivo não será aplicada nenhuma modificação na representação dos dados de entrada, pois a grande quantidade de

etapas de processamento aliada à perda de informações inerente do processo de modificação de representação pode degradar a resposta final do sistema.

Os resultados intermediários serão armazenados internamente na arquitetura para segmentação de vídeos desenvolvida neste trabalho. Isso é feito para reduzir a necessidade de acessos à memória acoplada ao *FPGA*. Além disso, o algoritmo de segmentação de vídeos implementado não opera sobre um modelo de *background*, portanto não é necessário calcular e ler previamente qualquer tipo de modelo de *background*. Por fim, algumas operações serão mapeadas em *ROM* para reduzir a quantidade de recursos de *FPGA* utilizados na implementação das mesmas.

A arquitetura apresentada no próximo capítulo foi desenvolvida para alcançar uma taxa de processamento maior ou igual a 30 fps. O desempenho do sistema produzido é comparado ao desempenho de um sistema composto por *CPU* e *GPU* que implementa o mesmo algoritmo de segmentação de *frames* de vídeo (LI et al., 2009). Por falta de outros *FPGAs* com grande disponibilidade de dispositivos lógicos, não será possível comparar o desempenho do sistema desenvolvido em diferentes tipos de *FPGA*, porém isso não afeta os resultados deste trabalho, pois com a taxa máxima de processamento alcançada é possível mensurar a eficiência do sistema para segmentação de sinais de vídeo.

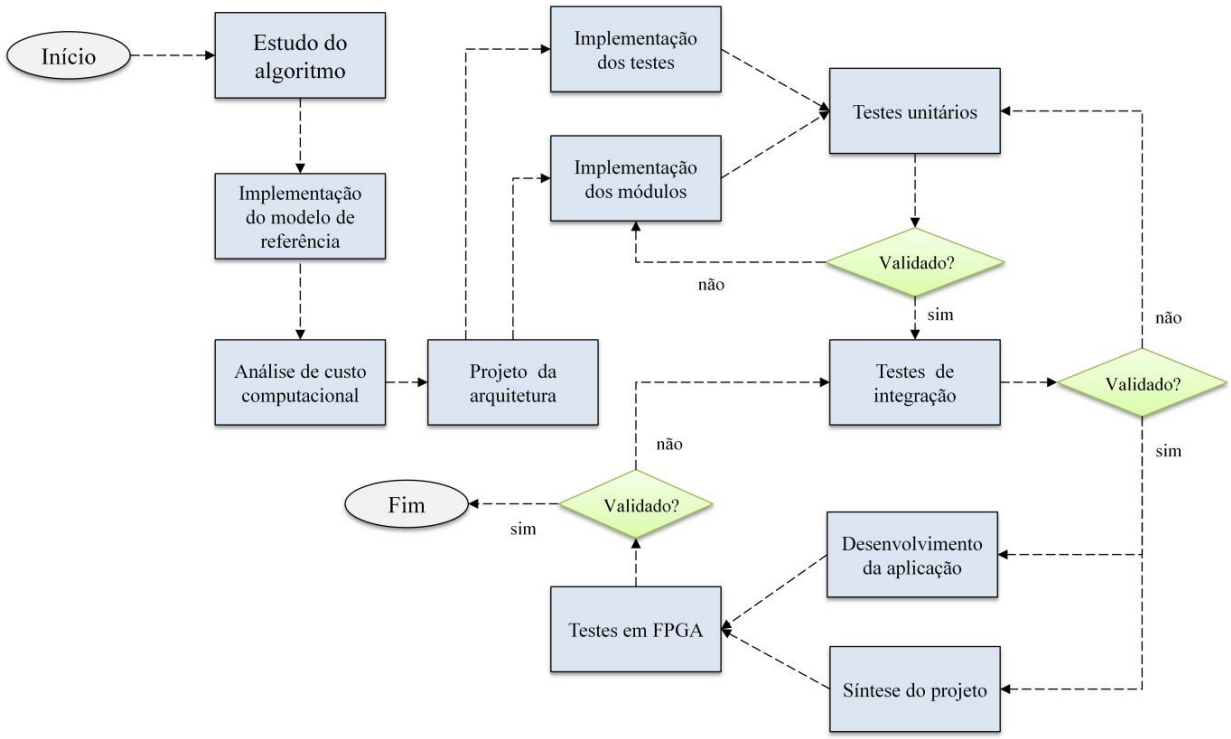
4. Arquitetura para Segmentação de Vídeos

Este capítulo aborda todos os aspectos relacionados ao desenvolvimento da plataforma e da arquitetura para segmentação de vídeos, tema desta dissertação, desde a metodologia utilizada até a implementação dos blocos aritméticos para o processamento da *FFT/IFFT*. Aqui são apresentados o projeto e a implementação de cada parte do elemento de processamento que fica no *FPGA*, bem como toda a estrutura computacional periférica, formada por um *host*, bancos de memória e barramento de comunicação. Também é demonstrado como o *FPGA* pode ser configurado para ser um elemento de processamento de propósito específico que, integrado a um computador de propósito geral (*host)*, consiste em uma tecnologia otimizada para a computação massiva de dados.

4.1. Metodologia de Desenvolvimento

Esta seção discute de forma sucinta a metodologia de desenvolvimento empregada neste projeto de mestrado. O objetivo aqui é explicar em linhas gerais como foi implementada cada parte do desenvolvimento do projeto da arquitetura capaz de executar a segmentação de sinais de vídeos. Para isso, começamos com a descrição do fluxo de desenvolvimento apresentado na **Figura 11**.

Figura 11 - Fluxo de desenvolvimento de projeto



Inicialmente, para o desenvolvimento da arquitetura heterogênea, baseada em *CPU* e *FPGA*, capaz de executar a segmentação de sinais de vídeo, foi executada uma etapa de estudo do algoritmo de segmentação (LI et al., 2009) com o intuito de identificar as partes do algoritmo que seriam candidatas à implementação em hardware. Em seguida, um modelo de referência foi implementado e foi feito o projeto dos módulos que iriam compor a arquitetura do processador de sinais de vídeo. Todos os módulos foram implementados e testados unitariamente e depois integrados e novamente testados. Os testes foram executados seguindo a metodologia de testes apresentada mais adiante nesta seção.

Após todos os testes funcionais e com restrição de tempo que foram executados nas etapas anteriores do fluxo de desenvolvimento, o projeto da arquitetura foi sintetizado

utilizando a ferramenta *Quartus II* (ALTERA, 2015b) e integrado com uma aplicação em software. Em seguida foram executados diversos testes com a arquitetura desenvolvida. Em tais testes, vídeos capturados por câmeras de segurança foram enviados para o processamento em *FGPA*, que neste projeto foi um *Stratix IV (EP4SE530H35C2)* (ALTERA, 2010b). Quando o resultado do processamento em *FPGA* não estava igual ao apresentado por um sistema em software desenvolvido em linguagem *C* que foi utilizado como referência, eram necessárias modificações nos módulos desenvolvidos e novos testes unitários e de integração. Dessa forma, a implementação da arquitetura produzida foi validada apenas quando a execução de testes em *FPGA* apresentaram os mesmos resultados gerados por um modelo de referência em software. A seguir são apresentados detalhes de cada etapa do fluxo de desenvolvimento do projeto que originou esta dissertação.

a) Estudo do Algoritmo a ser implementado:

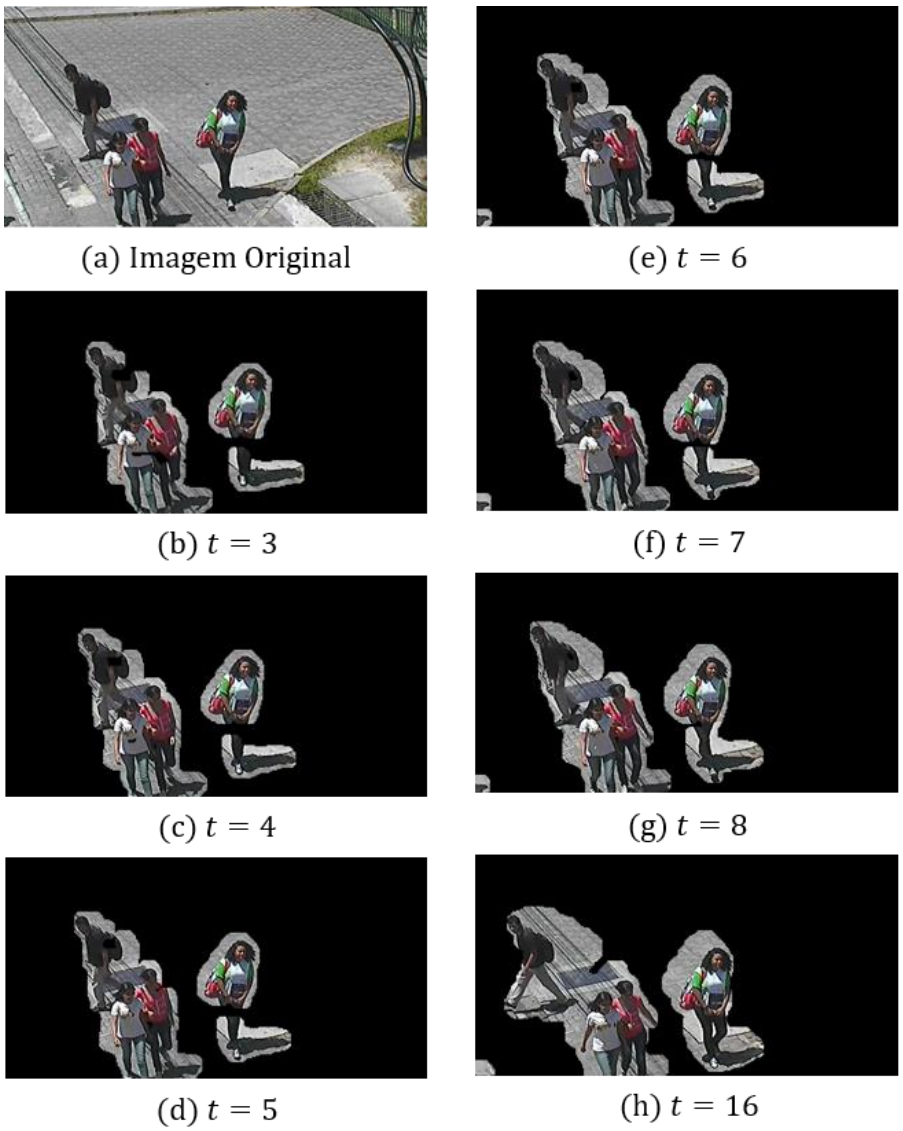
Foi executada uma avaliação detalhada de uma versão codificada para o *Matlab* do algoritmo de detecção de movimentos e segmentação de vídeos apresentado no artigo *"Dynamic Texture Segmentation Using Fourier Transform"* (LI et. al., 2009) antes da implementação em hardware da arquitetura apresentada neste capítulo. Isso foi feito para identificar as partes do algoritmo que consumiam maior tempo de processamento e possíveis otimizações que poderiam ser aplicadas para melhorar seu desempenho computacional.

A implementação do algoritmo de segmentação de vídeos analisada nesta etapa do projeto possuía um conjunto de parâmetros que eram utilizados, por exemplo, para configurar alguns filtros utilizados como atenuadores de ruídos. Os parâmetros eram utilizados também para controlar a profundidade temporal dos frames de vídeo que seriam utilizados para o cálculo da *FFT 3-D* e para controlar o tamanho que tais *frames* teriam após a etapa de redimensionamento que é executada logo no início do algoritmo.

O parâmetro de configuração utilizado para determinar a profundidade temporal t foi variado de um limite inferior igual a três até um limite superior igual a dezesseis. Tal variação foi executada para avaliar os resultados do algoritmo de segmentação de vídeo quando este parâmetro de configuração era variado. Na **Figura 12** podemos ver o resultado da segmentação de vídeos a partir da variação da profundidade temporal t. Podemos constatar que a variação de tal parâmetro pouco influi nos resultados alcançados, uma vez que, poucas diferenças são observadas nos frames apresentados na **Figura 12**. Dessa forma, podemos afirmar que não necessariamente uma grande profundidade temporal produz uma boa

qualidade visual dos resultados. Além disso, uma grande profundidade temporal torna o algoritmo mais lento, pois quanto maior tal profundidade maior também é a quantidade de dados a serem processados no cálculo da *FFT 3-D*. Observamos que não existiam diferenças significativas quando uma profundidade temporal igual a quatro ou igual a oito é utilizada. Esse resultado foi significativo para o desenvolvimento da arquitetura baseada em *FPGA* para segmentação de vídeos, pois foi através dele que ficou definido que a profundidade temporal utilizada para o processamento em *FPGA* seria fixa e igual a quatro.

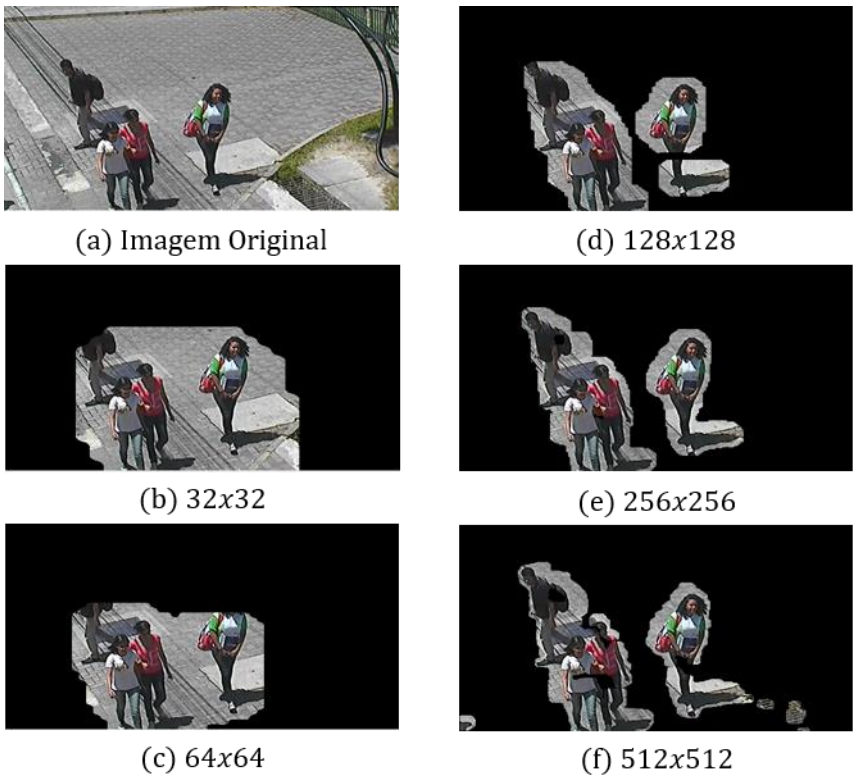
Figura 12 - Resultado da variação da profundidade temporal



Na **Figura 13** podemos observar o impacto da variação do tamanho dos *frames* que são de fato processados no algoritmo de segmentação. Nesta avaliação os *frames* originais tinham tamanho igual a $480x704$ *pixels* e o redimensionamento testado modificou a

dimensão dos *frames* para os seguintes tamanhos: $512x512$, $256x256$, $128x128$, $64x64$ e $32x32$. Observando a **Figura 13** é possível constatar um bom desempenho do algoritmo de segmentação quando os frames de entrada foram redimensionados para o tamanho $256x256$. Por outro lado, assumindo o tamanho $256x256$ como referência de bons resultados, é possível observar que quando os frames foram redimensionados para tamanhos iguais a $512x512$ e $128x128$ houve uma variação mínima no tamanho das regiões onde eram detectados pontos de movimento. Os resultados apresentados pelo algoritmo foram visualmente ruins quanto os *frames* de entrada foram redimensionados para os tamanhos $64x64$ e $32x32$, pois muitas regiões que não possuíam movimento foram detectadas como de movimento e regiões com movimento não foram detectadas e consequentemente não foram segmentadas.

Figura 13 - Resultado da variação do tamanho dos frames processados



A partir dessa análise ficou determinado que a arquitetura baseada em *FPGA* para a segmentação de vídeos deveria ter uma parâmetro global denominado *matrix_size*, que deveria ser ajustado antes da síntese ser executada, pois dessa forma a arquitetura apresentada neste trabalho poderia ser sintetizada para diferentes tipos de dispositivos com variadas

quantidade de memória interna e elementos lógicos. Assim, se houver espaço disponível no *FPGA*, a arquitetura pode ser configurada para operar sobre *frames* com grande tamanho.

A avaliação dos parâmetros de configuração citados anteriormente serviu para identificar o quão a arquitetura para segmentação de vídeos deveria ser flexível a suas variações. Tal avaliação foi muito importante na etapa inicial deste trabalho, pois permitiu que fosse adquirido um maior conhecimento sobre o algoritmo para segmentação de vídeos e possibilitou que fosse tomada a decisão de usar uma profundidade temporal fixa e a decisão de utilizar um parâmetro de síntese para decidir o tamanho dos *frames* a serem processado no *FPGA*. Tais decisões tiveram um impacto direto em todas as etapas subsequentes deste projeto e isso justifica a execução desta parte da metodologia de desenvolvimento.

b) Elaboração da arquitetura em alto nível:

Inicialmente foi elaborada através de diagramas de blocos uma arquitetura em alto nível para implementar o algoritmo a ser executado em *FPGA*. Isso foi feito levando em conta os *FPGAs* que estavam disponíveis para este projeto de mestrado e para os futuros testes que seriam executados nas etapas posteriores do desenvolvimento deste trabalho. Um dos requisitos básicos para definição de tal arquitetura foi a estimativa da largura de banda de dados necessária para a execução do algoritmo. Assim, a arquitetura foi projetada levando em consideração a quantidade de recursos (blocos lógicos, *DSP* e memória interna) presentes nos *FPGAs* disponíveis para o projeto e levando em consideração também a largura de banda necessária para executar o algoritmo para segmentação de vídeos com no máximo quatro cálculos de *FFT/IFFT* sendo executados em paralelo.

c) Desenvolvimento dos módulos componentes da arquitetura:

Nesta etapa do projeto foram desenvolvidos todos os componentes integrantes da arquitetura que foram elaborados em diagrama de blocos explicados na fase anterior desta metodologia. As máquinas de estados *(FSM)* de todos os módulos componentes do projeto que necessitavam de algum controle interno foram modeladas antes de sua implementação. Esse processo consistiu em elaborar o diagrama de estados de cada *FSM* e verificar seu comportamento básico manualmente. Tal avaliação de funcionamento é executada levando consideração a configuração da *FSM* a cada ciclo de relógio a partir de uma configuração inicial dos sinais internos da *FSM*, que foi estabelecida de forma prévia.

Após estes testes preliminares os módulos que compõem a arquitetura para segmentação de vídeos foram implementados utilizando *Verilog* (VERILOG, 2016) como a linguagem para descrição de hardware. Para a síntese dos módulos desenvolvidos foi utilizado o software *Quartus 15.0* da Altera (ALTERA, 2015b).

d) Execução de testes dos módulos componentes da arquitetura:

O procedimento para a execução dos testes de cada módulo desenvolvido neste projeto pode ser resumido através da **Figura 14**. Antes da implementação de cada módulo em hardware foi desenvolvida sua versão em software, denominada módulo de referência, que é capaz de executar as mesmas tarefas para qual o componente em hardware foi projetado. Geralmente o módulo de referência será o responsável por gerar dois tipos de arquivos de texto: os arquivos de entrada e os arquivos de referência.

Os arquivos de entrada são os estímulos fornecidos para o módulo em hardware. Os arquivos de referência, como o próprio nome diz, é uma referência de qual resposta deve ser fornecida pelo módulo sob teste quando são fornecidas as entradas armazenadas presentes nos arquivos de entrada.

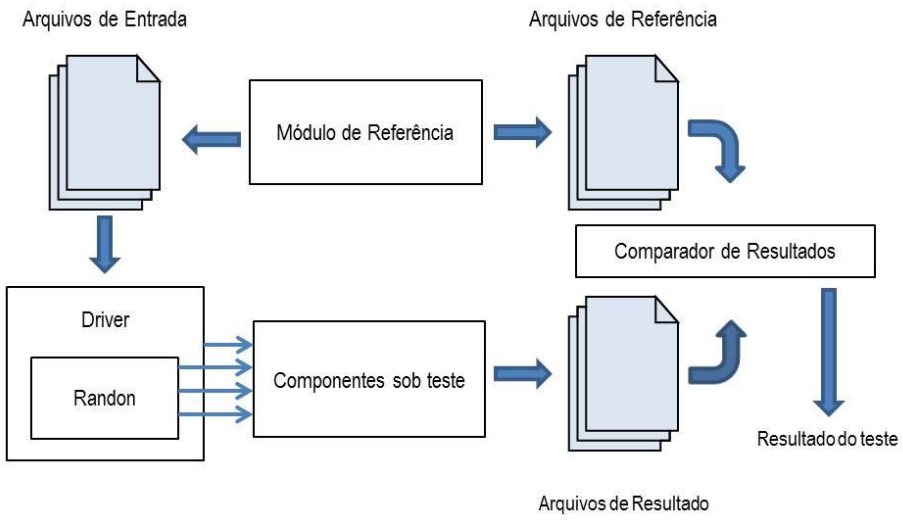
O *Driver* é a parte da estrutura de testes responsável por ler os arquivos de entrada, enviar, junto aos dados lidos, estímulos aleatórios de alguns sinais de entrada e, por fim, monitorar e escrever em arquivos de texto os resultados gerados pelo módulo sob teste. O comparador de resultados geralmente lê os arquivos de referência e os arquivos de resultados e, como o próprio nome diz, executa a comparação dos resultados que foram produzidos pelo módulo sob teste com os resultados produzidos pelo módulo de referência, produzindo dessa forma o resultado final do teste.

Todos os módulos em hardware desenvolvidos neste projeto foram submetidos de forma isolada a uma bateria de testes que possuía a estrutura apresentada na **Figura 14**. No teste de cada módulo as entradas foram geradas de forma aleatória para simular as diferentes situações de funcionamento que cada módulo seria submetido. Os componentes do projeto que ficaram responsáveis por calcular a *FFT 3-D*, o espectro de fase de Fourier e a *IFFT 2-D* foram testados com dados aleatórios de valor ilimitado e com dados obtidos a partir de frames de vídeo capturados por câmeras de segurança.

Durante a realização dos testes foram reparadas falhas do projeto inicial de cada módulo e em alguns casos foram adicionadas funcionalidades que não estavam previstas em seu projeto original. Devemos destacar que a estrutura de testes desenvolvida para cada

módulo foi implementada em *SystemVerilog* (SYSTEMVERILOG, 2016). Essa linguagem foi escolhida, pois existe um conjunto de ferramentas para testes funcionais de componentes de hardware que já foram desenvolvidas em *SystemVerilog*.

Figura 14 - Estrutura de testes



e) Testes de integração dos componentes da arquitetura:

Após a validação de cada componente isolado da arquitetura foi implementada a integração de cada um dos cinco passos de processamento que foram apresentados anteriormente neste texto. Cada passo de processamento foi então testado de forma isolada seguindo a mesma metodologia adotada para testes apresentada anteriormente.

A partir da validação de cada passo foi executada a integração dos passos de processamento de forma incremental. Inicialmente foram integrados o primeiro e o segundo passo e em seguida foram realizados os testes para avaliar a operação integrada do primeiro e com o segundo passo de processamento. Quando foi validada essa integração foi implementada a integração do terceiro passo aos dois passos anteriores. Novamente foram executados testes para validar, neste caso, a operação conjunta dos três primeiros passos de processamento. Foi adotada esta metodologia de testes de integração até a realização dos testes que envolvem os cinco passos de processamento.

f) Testes no *FPGA* dos componentes desenvolvidos:

Após a realização de cada um dos testes de integração, abordados anteriormente nesta seção, foram executados os testes de integração em *FGPA*. Isso foi feito para avaliar o comportamento em *FPGA* de cada parte do subsistema que foi desenvolvido.

Todos os módulos implementados neste projeto foram então integrados em uma plataforma única para que o teste final dos cinco passos de processamento fosse executado em *FPGA*. Após a síntese, o arquivo de configuração de *FPGA* foi baixado em uma plataforma alvo e a arquitetura para segmentação de vídeos foi testada e validada, completando assim todo o fluxo de desenvolvimento apresentado neste capítulo.

g) Elaboração de documentos:

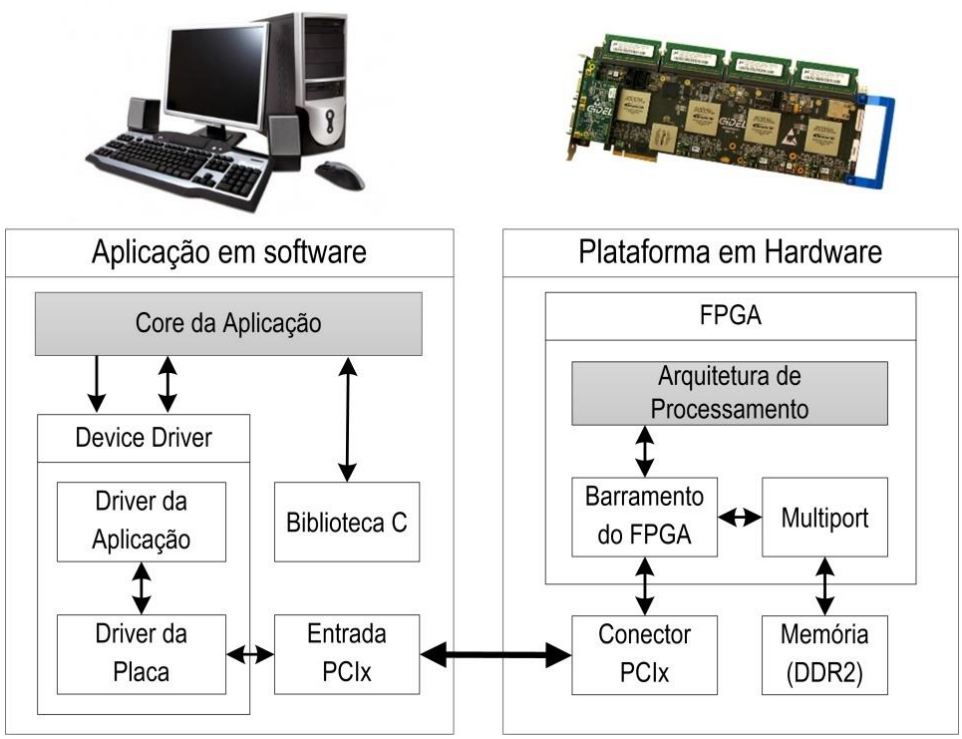
Durante todas as etapas deste projeto foram elaborados documentos que descreviam de forma simples os módulos de hardware desenvolvidos para produzir a arquitetura para segmentação de sinais de vídeo que foi apresentada neste capítulo. Através de tais documentos foi possível entender o funcionamento básico de cada módulo tanto de forma isolada, quanto de forma integrada a outros módulos. Tal documentação foi de grande ajuda durante o processo de elaboração desta dissertação, pois através dela foi possível relembrar detalhes da implementação de alguns módulos em hardware que foram desenvolvidos logo no início deste projeto. Além disso, foram documentadas todas as máquinas de estados finitos que foram projetadas e implementadas durante este trabalho de mestrado. Tais máquinas podem ser encontradas nos Anexos desta dissertação.

4.2. Visão Geral da Arquitetura para Segmentação de Vídeos

A arquitetura de processamento de sinais de vídeo que é apresentada neste capítulo se baseia em um sistema misto de computação que é formado através da junção de um computador de propósito geral e um *FPGA*. Este último é utilizado como elemento de processamento de propósito específico. Nesta abordagem, o *FPGA* é utilizado para prover um elemento de computação paralela e eficiente das partes de um algoritmo que demandam maior esforço computacional enquanto que para o computador ficam reservadas as tarefas que demandam menor esforço computacional. Na **Figura 15** temos um exemplo de como pode ser

construído um sistema de processamento misto, que se baseia na operação conjunta de uma *CPU* e um *FPGA*, para gerar um sistema para processamento massivo de dados (MEDEIROS, et. al., 2012).

Figura 15 - Sistema de computação formado por CPU e FPGA



A escolha de qual parte deve ficar em cada componente da plataforma depende da análise prévia do algoritmo (*profiling*). Em geral, o computador de propósito geral gerencia toda a plataforma. Ela possui uma aplicação desenvolvida, por exemplo, em *C*, que é responsável por executar tarefas relacionadas ao processamento de parte de algum algoritmo específico, em geral aquelas que não são críticos em tempo de execução, ou essencialmente seriais, e por enviar dados e requisição de processamento para o *FPGA* das partes do algoritmo que necessitam de grande esforço computacional e maior paralelismo.

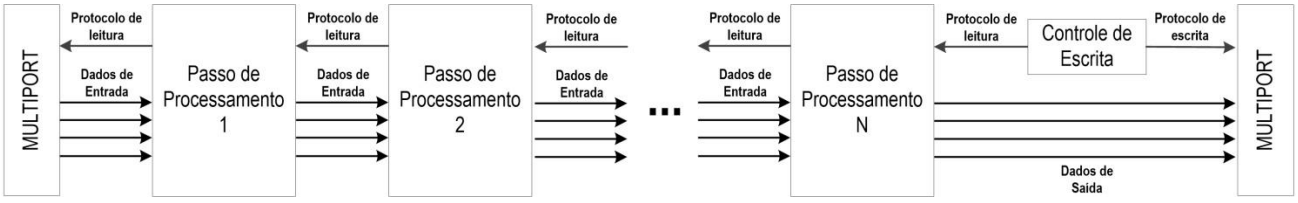
A plataforma em hardware, ou em *FPGA*, por sua vez, recebe os dados, executa o processamento dos mesmos e envia os resultados de volta ao computador de propósito geral. Os *drivers* das placas são os responsáveis por controlar o envio e a recepção de tais dados. Podemos notar que a conexão entre *CPU* e *FPGA* pode ser estabelecida através de um barramento *PCI-Express*, mas não necessariamente isso é uma obrigação, pois existem diversas plataformas de desenvolvimento que utilizam outros tipos de barramentos, como por exemplo, *PCI* ou *Ethernet*. Para construir um sistema semelhante ao apresentado na **Figura**

15 o desenvolvedor de uma arquitetura de processamento baseada em um sistema misto composto por *CPU* e *FPGA* deve se concentrar desenvolver o Core da Aplicação e a Arquitetura de Processamento que fica na plataforma de hardware, pois os outros elementos que aparecem na figura estão implementados e disponíveis para utilização.

A plataforma em hardware que aparece na **Figura 15**, por exemplo, é baseada na plataforma *ProcStar IV* da Gidel (GIDEL, 2010). Tal dispositivo possui quatro *FPGAs Stratix IV* (ALTERA, 2015) que podem ser utilizados de forma independente. Tal plataforma foi utilizada em todo o desenvolvimento deste trabalho por apresentar uma boa quantidade de elementos lógicos e por apresentar uma boa quantidade de memória interna que pode ser facilmente utilizada para atender as necessidades de armazenamento peculiares de cada projeto. Além disso, a Gidel (GIDEL, 2010) oferece uma ferramenta para acesso fácil aos dados que devem ser lidos e escritos pelo *FPGA*: o *Multiport*. Através de um simples protocolo de leitura e escrita o *Multiport* deixa transparente ao usuário a forma como as memórias externas ao *FPGA* podem ser acessadas e como os dados processados no *FPGA* são recebidos e enviados durante a comunicação estabelecida entre *FPGA* e o computador de propósito geral. Isso traz uma maior facilidade no desenvolvimento de projetos que demandam grandes quantidades de leitura e escrita dos dados.

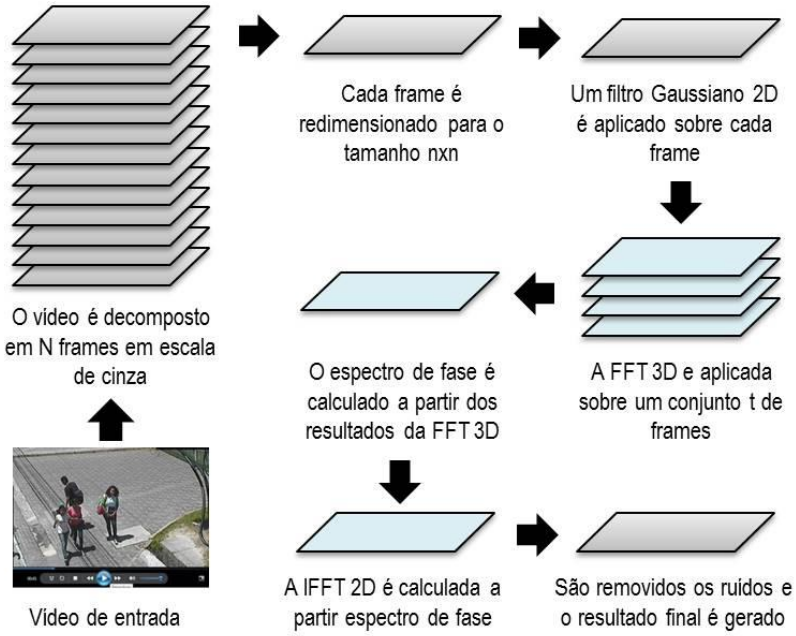
A arquitetura para segmentação de vídeos aqui apresentada transfere para o *FPGA* a tarefa de executar a parte mais custosa do algoritmo de segmentação (LI *et. al*.; 2009). A parte do algoritmo que é executada no *FPGA* pode ser subdividida em etapas ou passos de processamento para formar uma arquitetura de processamento como a demonstrada na **Figura 16**. Na organização interna desta arquitetura cada passo de processamento deve realizar a leitura dos dados através de um determinado protocolo de leitura, executar uma parte do processamento sobre os dados lidos e armazenar os valores resultantes para serem utilizados nos passos de processamento subsequentes. Ao término das *N* etapas de processamento um controlador de escrita deve receber os dados que serão enviados de volta para a aplicação em software e providenciar a sua escrita no Multiport.

Figura 16 - Arquitetura genérica para processamento de dados em FPGA



O algoritmo de segmentação de vídeos baseado no conceito de texturas dinâmicas (LI, et. al, 2009) que foi apresentado no segundo capítulo deste trabalho pode ser representado graficamente através da **Figura 17**. Inicialmente poderíamos pensar em implementar todas as etapas deste processamento diretamente no *FPGA*, porém, isso demandaria um tempo maior de projeto e um *FPGA* com grande quantidade de recursos lógicos e grande capacidade interna de armazenamento de dados. Além disso, nem todas as etapas do algoritmo podem ter um bom desempenho quando implementadas em *FPGA*, por isso, inicialmente foi feito um levantamento para avaliar quais partes do algoritmo consumiam mais tempo de processamento *(profiling)* e somente após tal avaliação foram escolhidas as partes do algoritmo que seriam implementadas para o *FPGA* e quais etapas ficariam para serem executadas no computador de propósito geral.

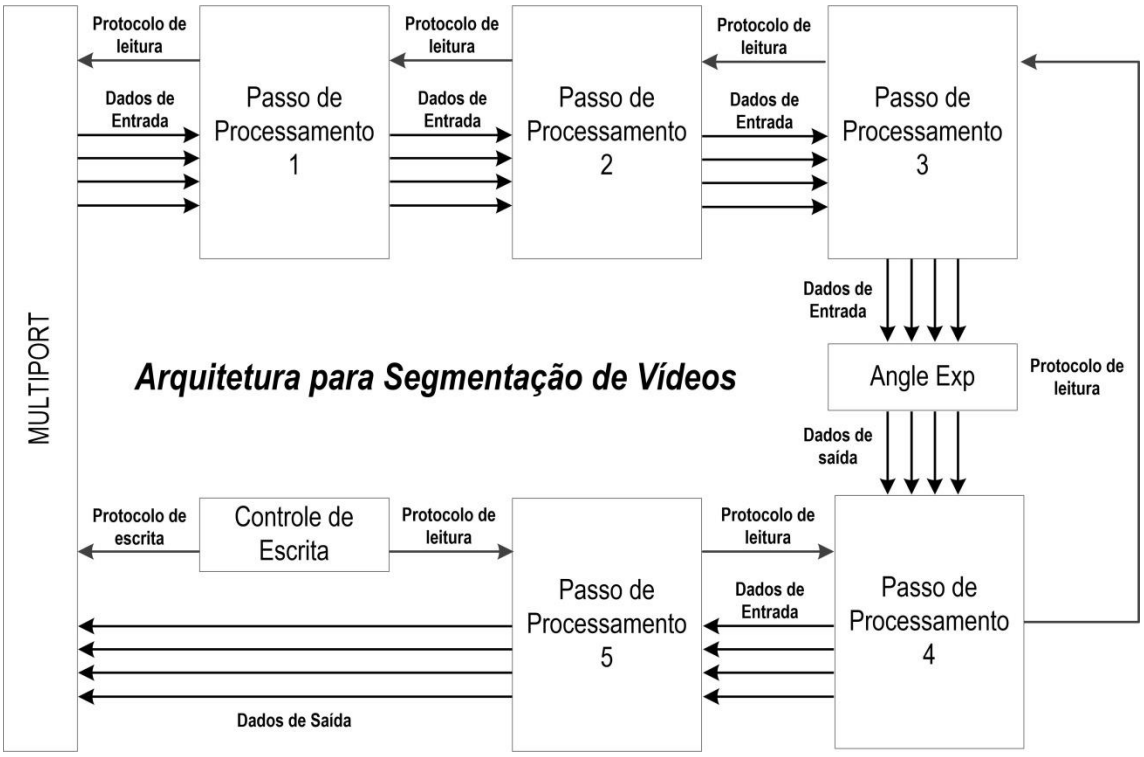
Figura 17 - Etapas do algoritmo de segmentação de vídeo



A análise do tempo de processamento gasto em cada etapa do algoritmo de segmentação de vídeos revelou que aproximadamente 94% do tempo são gastos nas etapas de cálculo da *FFT 3-D*, cálculo do espectro de fase e cálculo da *IFFT 2-D*. Portanto, acelerar estas partes do algoritmo levaria a uma redução considerável do tempo total de processamento. Por esse motivo estas três etapas foram escolhidas para serem executadas no *FPGA*, enquanto que as outras partes ficaram a cargo do computador ao qual a plataforma em hardware for conectada.

Levando em consideração a arquitetura genérica apresentada na **Figura 16** e as etapas do algoritmo de segmentação de vídeo que seriam implementados no *FPGA*, foi proposta e desenvolvida uma arquitetura para segmentação de vídeos apresentada na **Figura 18**. Nos primeiros três passos de processamento em tal arquitetura são executadas as operações necessárias para o cálculo da *FFT 3-D* sobre os dados lidos do módulo *Multiport*. Em seguida o cálculo do espectro de fase de Fourier é executado no módulo denominado *Angle Exp* e por fim, nos dois últimos passos, é executada a *IFFT 2-D*. O controlador de escrita se encarrega de receber os dados resultantes do processamento e executar a sua escrita no módulo *Multiport*.

Figura 18 - Arquitetura proposta para segmentação de vídeos



A arquitetura implementada para a segmentação de vídeos pode processar *frames* de vídeos de tamanhos variados. A dimensão do *frame* a ser processado pode ser definida em tempo de síntese através da configuração de um parâmetro denominado *matrix_size*, que é compartilhado por todos os módulos que compõem a arquitetura de segmentação. Os *frames* de entrada devem ser quadrados e de tamanho que seja uma potência de dois, pois isso torna mais simples os controladores de leitura e escrita de dados e permite que o cálculo da *FFT 3-D* e da *IFFT 2-D* seja realizada por apenas um tipo de *FFT/IFFT*, que terá número de pontos igual ao valor do parâmetro *matrix_size*. Isso indica que os *frames* de entrada devem ser

redimensionados para ter o mesmo número de linhas e colunas antes de serem processados no *FPGA*. A escolha por tamanhos quadrados de *frames* foi tomada com base na análise do desempenho do algoritmo a partir da variação da dimensão dos *frames* de entrada. Através de testes feitos em software foi constatado que modificar a dimensão dos *frames* para deixa-los quadrados não influenciava de forma negativa o desempenho e a qualidade dos resultados produzidos pelo algoritmo de segmentação. No quinto capítulo é apresentado um estudo de caso que deixa claro como foi feita a avaliação da variação do tamanho dos *frames* e da profundidade temporal e veremos como tal avaliação influenciou o desenvolvimento da arquitetura para segmentação de vídeos.

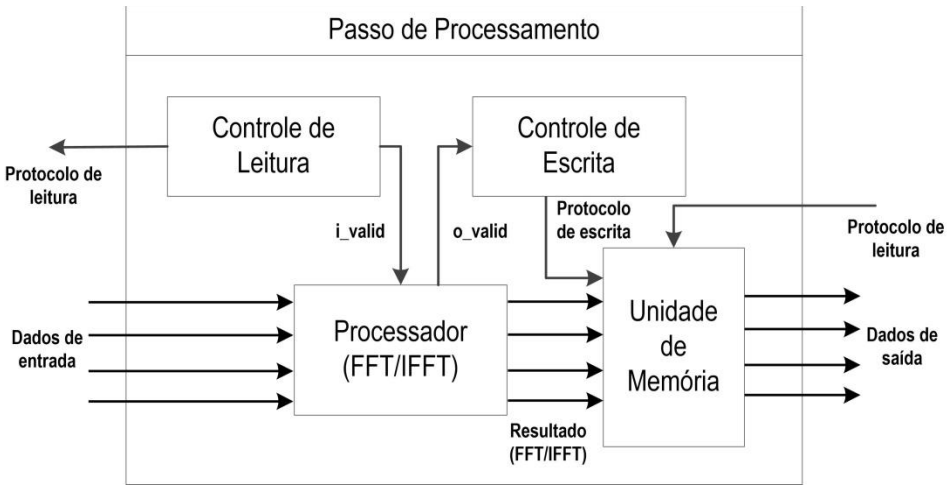
A profundidade temporal foi fixada em quatro *frames*. Foi escolhido este número após uma análise em software do quanto a variação da profundidade temporal influenciava na qualidade do resultado obtido com a segmentação. Como observado na seção anterior, visualmente foi constatado que variar a profundidade temporal entre quatro e oito *frames* não influenciava de forma significativa o resultado do processamento. Por outro lado, trabalhar com profundidade temporal menor do que quatro provocava uma degradação da qualidade do resultado do algoritmo de segmentação. Além disso, quanto maior a profundidade temporal, maior seria a quantidade de dados a serem armazenados no *FPGA* e a quantidade de processamento requerida para o cálculo da *FFT 3-D*. Este foi outro fator a ser levado em consideração na escolha de usar quatro *frames* de profundidade temporal no processamento da segmentação.

Cada ponto nos *frames* de entrada pode ser representado por um número com oito bits, isso porque os *frames* de vídeo que são enviados para o processamento no *FPGA* estão em escala de cinza. Desta forma na arquitetura proposta cada entrada de dados possui oito bits. Durante o desenvolvimento deste trabalho ficou definido que seriam processadas quatro linhas dos frames em paralelo em cada passo de processamento, portanto, a arquitetura aqui proposta possui quatro entradas de dados, temos a necessidade de leitura de 32 bits/ciclo de relógio. Internamente os dados inteiros são transformados em números complexos onde a parte imaginária e a parte real são números de ponto flutuante de precisão simples, ou seja, números de 32 bits. Cada uma das quatro saídas de dados na arquitetura proposta possui 64 bits, o que leva a necessidade de escrita de 256 bits/ciclo de relógio.

De uma forma geral cada passo de processamento pode ter uma estrutura genérica como a apresentada na **Figura 19**. Cada um dos cinco passos de processamento da arquitetura para segmentação de vídeos proposta neste trabalho possui estes quatro módulos principais. As variações do tipo de cada um desses módulos é o que torna um passo de processamento

diferente do outro. Durante o fluxo de execução do processamento realizado em cada passo o módulo que controla a leitura dos dados é responsável por fazer requisições de leitura de dados armazenados no passo anterior de processamento. O Processador *FFT/IFFT* é responsável por executar o cálculo da *FFT/IFFT* em uma determinada dimensão dos dados. O Controlador de Escrita coordena a escrita de dados na Unidade de Memória, que por sua vez armazena os dados resultantes do processamento até que o Controlador de Leitura do passo seguinte faça a requisição de tais dados.

Figura 19 - Estrutura genérica dos passos de processamento

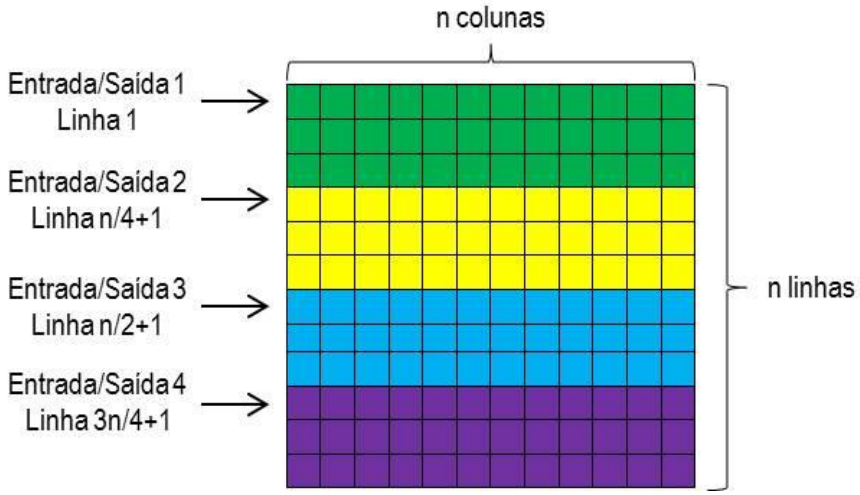


Os dados que são escritos nas Unidades de Memória de cada passo ficam armazenadas na memória interna do *FPGA*. Como o tamanho da memória interna do *FPGA* é muito reduzido quando comparado ao tamanho disponível para armazenamento externo, é importante adotar estratégias para reduzir o tamanho das Unidades de Memória de cada passo. Nas próximas seções veremos como a propriedade da Transformada de Fourier conhecida como simetria hermitiana pode ser utilizada para reduzir a quantidade de dados armazenados nas memórias do primeiro e do segundo passo de processamento.

Para uma melhor organização das Unidades de Memória do primeiro e do segundo passo e para armazenar uma menor quantidade de dados em tais unidades é preciso transpor os frames do vídeo a serem processados no *FPGA*. Sendo assim, o cálculo da *FFT* será executado primeiramente nas colunas dos *frames* de entrada e os resultados são armazenados novamente de forma transposta na Unidade de Memória do primeiro passo. No segundo passo o cálculo da *FFT* é executado no sentido das linhas dos frames de vídeo, completando assim o cálculo da *FFT 2-D* sobre os dados de entrada.

O armazenamento dos dados na memória externa do *FPGA* é outro aspecto importante que deve aqui ser destacado, pois tem impacto direto na disponibilidade de dados para processamento no *FPGA*. Para prover uma maior eficiência de leitura e escrita, os dados são separados em *buffers* de memória diferentes que podem ser acessados de forma independente através de quatro portas. Para a arquitetura de segmentação de vídeos apresentada neste trabalho foram utilizados dois *buffers* de leitura e dois *buffers* de escrita. Cada porta de leitura e escrita aponta para uma região de dados, assim como pode ser visto na **Figura 20**. Os *frames* de vídeo a serem processados são ordenados um após o outro na memória externa do *FPGA*. Portanto, a cada requisição de leitura são lidos quatro dados de um mesmo *frame*. Dessa mesma forma, a cada requisição de escrita são escritos quatro dados de um mesmo *frame*. Os *frames* de vídeo são dispostos de forma contínua em cada *buffer* citado anteriormente.

Figura 20 - Esquema de leitura e escrita de dados



Após a visão geral da arquitetura para a segmentação de vídeos proposta neste trabalho serão abordadas, nas próximas seções, as particularidades relacionadas a cada um dos cinco passos de processamento. Antes disso, porém, na próxima seção será descrita a implementação do módulo que executa o cálculo da *FFT/IFFT* sobre os dados manipulados na arquitetura para segmentação de vídeos. Deve ser destacado que os módulos *Controle de Leitura*, *Controle de Escrita* e a *Unidade de Memória* de cada passo foram desenvolvidos como máquinas de estados finitos *(FSM)*. Para uma melhor organização deste capítulo, os diagramas de estados de cada módulo desenvolvido neste projeto estão disponíveis nos Anexos desta dissertação.

4.3. Implementação da *FFT/IFFT*

Durante o desenvolvimento da arquitetura para segmentação de vídeos surgiu a necessidade de implementar módulos capazes de executar os cálculos da *FFT/IFFT* sobre os dados que são enviados para o processamento em *FPGA*. Isso se deu principalmente por não existir processadores de *FFT/IFFT* disponíveis livremente para uso e capazes de trabalhar com os dados da forma como era necessária neste projeto. Era necessário que os operadores de *FFT/IFFT* fossem capazes de receber como entrada números complexos nos quais a parte real e a parte imaginária são representadas por números de ponto flutuante de precisão simples (IEEE, 1985). Com a adoção dessa formatação não seriam perdidas informações por conta de mudanças de representação de dados, ou seja, os dados processados no *FPGA* teriam o mesmo formato dos dados processados em um computador de propósito geral. Era necessário também ter um maior controle sobre a quantidade de recursos de *FPGA* consumidos por cada processador de *FFT/IFFT*, ou seja, era importante ter domínio sobre a forma operacional para, caso necessário, promover mudanças nos processadores com o intuito de reduzir o consumo de recurso do *FPGA*. Essas foram as razões que levaram ao desenvolvimento dos processadores de *FFT/IFFT* que são apresentados nesta seção.

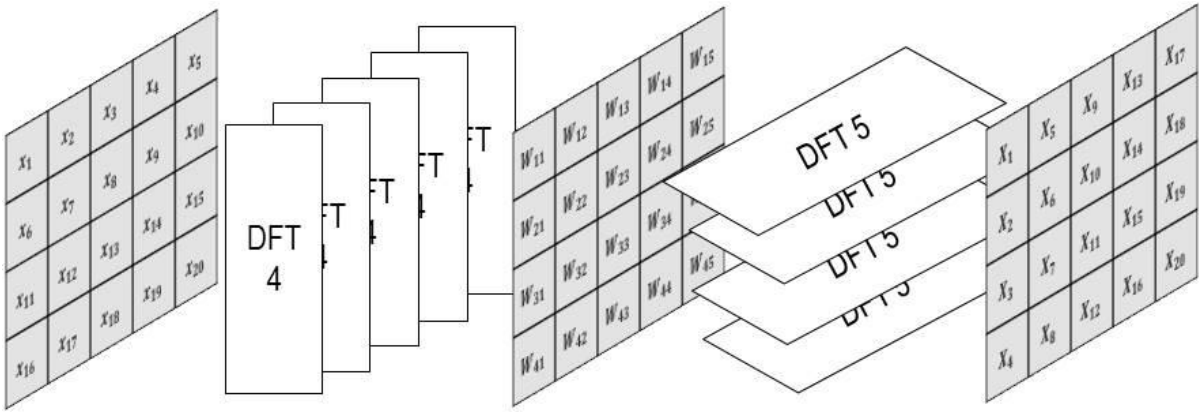
O algoritmo para o cálculo da *DFT* proposto por (COOLEY e TUKEY, 1965) ficou mundialmente conhecido como *Fast Fourier Transform (FFT)* por calcular de forma mais rápida do que a convencional uma *DFT* n de pontos e também por exigir um menor número de operações para tal cálculo, sem que para isso fosse necessário qualquer tipo de aproximação numérica.

(DUHAMEL e VETTERLI, 1990) mostraram, levando em consideração a propriedade da *Linearidade*, diferentes formas de como os n pontos de entrada de uma *FFT/IFFT* podem ser reorganizados de forma a serem processados por uma ou mais *FFTs/IFFTs* que operam sobre uma quantidade menor do que n pontos. A **Figura 21** mostra um exemplo de como n pontos podem ser reorganizados para que o processamento da *DFT* possa ser executado por operadores de dimensão menor que n. Neste caso o conjunto de dados de entrada possui vinte pontos.

Para o cálculo da *DFT* sobre um conjunto de vinte pontos é necessário reorganizar os dados de entrada para formar uma matriz com quatro linhas e cinco colunas. Após essa reorganização cada uma das cinco colunas pode ser processada por uma *DFT* de quatro pontos. As saídas das *DFTs* de quatro pontos são então multiplicadas pelos *twiddles* que

foram previamente calculados e estão também organizados na forma de uma matriz com quatro linhas e cinco colunas. O resultado da multiplicação é então reorganizado para que em cada linha da matriz seja calculada por uma *DFT* de cinco pontos. No final desta segunda etapa de processamento as saídas das *DFTs* de cinco pontos são reorganizadas para que a apresentação dos resultados seja feita de forma correta. Retornaremos a questão de como os dados devem ser organizados durante o processamento da *FFT/IFFT* (DUHAMEL e VETTERLI, 1990) através de um exemplo no final desta seção.

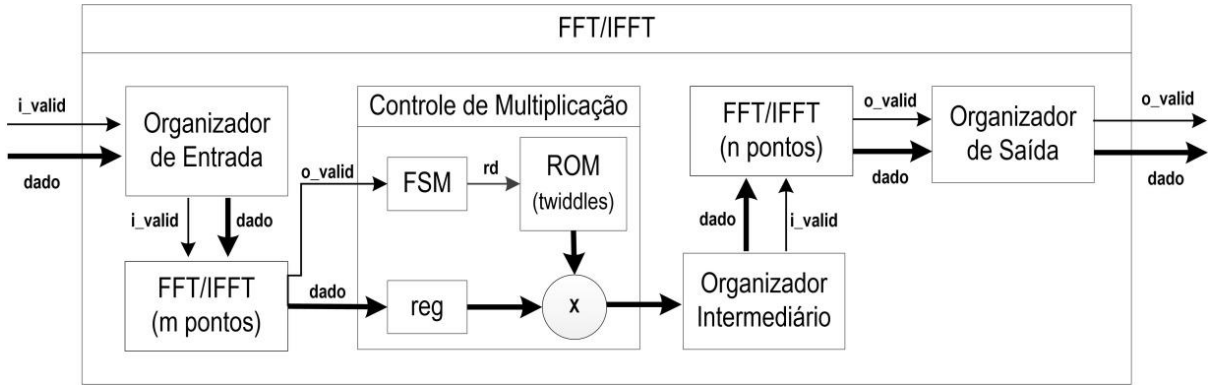
Figura 21 - Organização de pontos para o cálculo de uma FFT/IFFT



O módulo capaz de executar o processamento da *FFT/IFFT* desenvolvido neste projeto toma como base todas as ideias de reorganização de dados apresentadas acima. Na **Figura 22** temos uma representação genérica do diagrama de blocos de cada processador de *FFT/IFFT* construído neste projeto. Devemos destacar que essa organização leva em conta questões como a economia de recursos do *FPGA* onde o modulo é implementado, uma vez que o número de pontos processados em cada DFT pode requerer uma maior quantidade de recursos para sua implementação, e a velocidade operacional de cada componente do diagrama de blocos.

Os módulos *Organizador de Entrada*, *Organizador Intermediário* e *Organizador de Saída* manipulam os dados para que os mesmos fiquem corretamente organizados em todas as etapas do processamento. Por questões de economia de recursos do *FPGA* não é possível implementar a *FFT/IFFT* com várias *FFTs/IFFTs* de m e n pontos, por isso, no diagrama de blocos aparece apenas uma *FFT/IFFT* de m pontos e uma *FFT/IFFT* de n pontos. Isso torna ainda mais importante a forma como os módulos *Organizadores* manipulam os dados durante todas as etapas de cálculo da *FFT/IFFT*. Tal manipulação ficará mais clara através do exemplo apresentado no final desta seção.

Figura 22 - Diagrama de blocos genéricos para FFT/IFFT



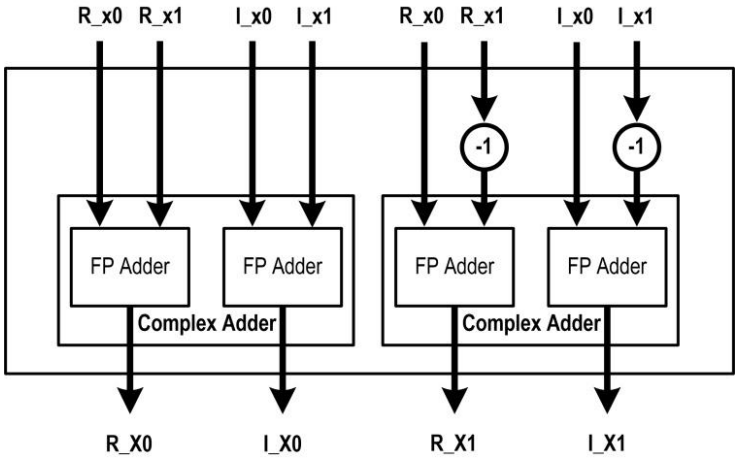
O Controle de Multiplicação monitora o recebimento dos dados resultantes do processamento da *FFT/IFFT* de m pontos e coordena a sua multiplicação pelos *twiddles*, que foram processados previamente e estão armazenados em uma pequena memória *ROM*, ou seja, o *Controlador de Multiplicação* é também responsável por executar a leitura dos dados armazenados na memória *ROM*. A organização matricial dos twiddles para o cálculo da FFT e IFFT é apresentado na **Figura 23**. Esta é a organização dos *twiddles* para quando os pontos a serem processados são organizados em uma matriz de NxN pontos.

Figura 23 - Organização matricial dos *twiddles*

1	1	1	1	...	1
1	W_N	W_N^2	W_N^3	...	$W_N^{(N-1)}$
1	W_N^2	W_N^4	W_N^6	...	$W_N^{2(n-1)}$
...	...	...	...	...	...
1	$W_N^{(N-1)}$	$W_N^{2(N-1)}$	$W_N^{3(N-1)}$	...	$W_N^{(N-1)(N-1)}$

Conforme citado anteriormente o módulo capaz de calcular a *FFT/IFFT* pode ser implementado pela união de dois outros módulos capazes de executar uma *FFT/IFFT* sobre um menor número de pontos que a original. A *FFT/IFFT* de dois pontos é a base para construção de qualquer outra *FFT/IFFT* que foi desenvolvida durante este trabalho. Seu diagrama de blocos é apresentado na **Figura 24**.

Figura 24 - Diagrama de blocos de uma FFT/IFFT de 2 pontos



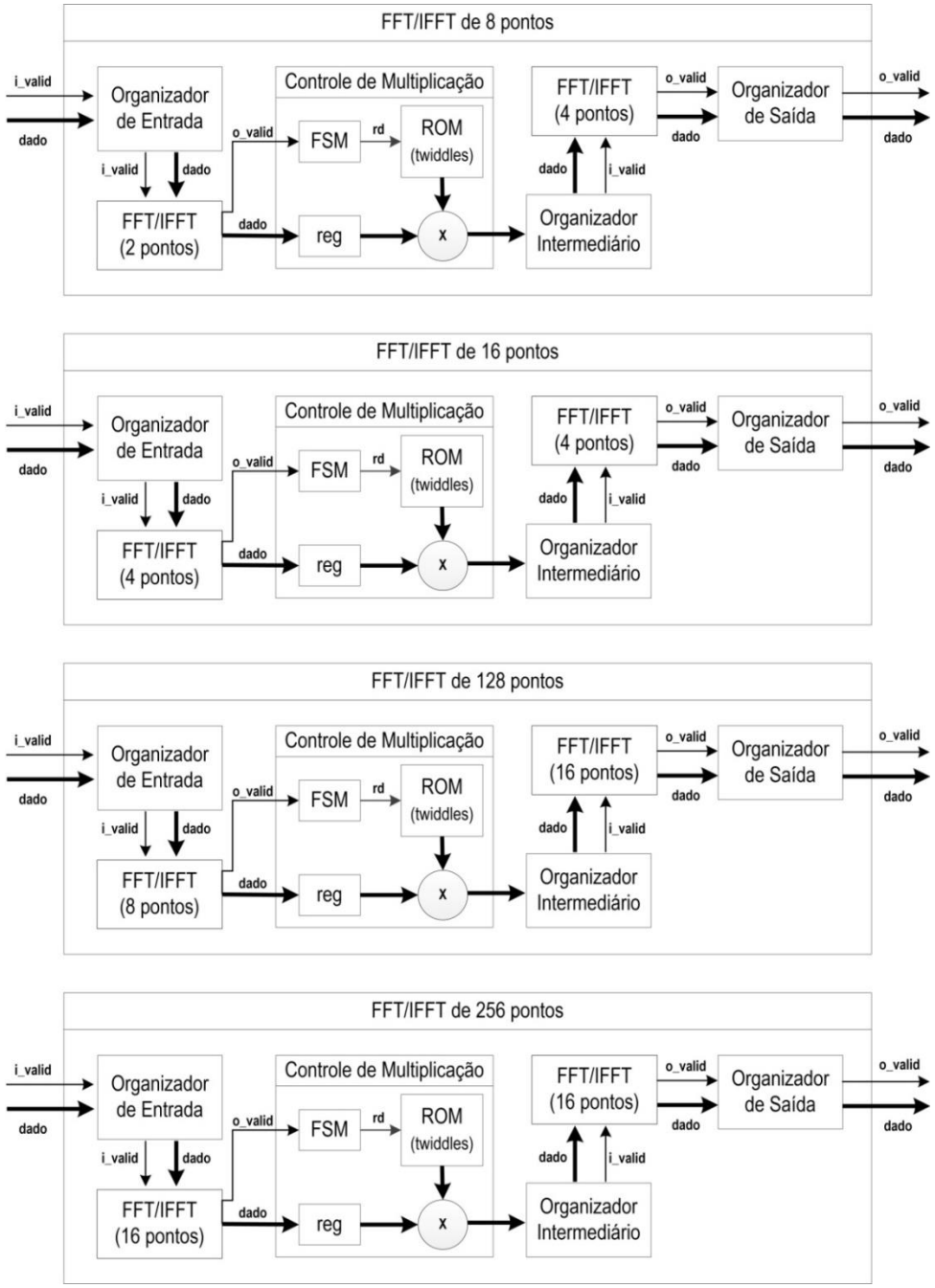
A *FFT/IFFT* de dois pontos foi implementada neste trabalho a partir do uso de somadores de números complexos desenvolvidos pela integração de somadores de números no formato de ponto flutuante. Tais somadores são fornecidos pela Altera (ALTERA, 2015a) para utilização em projetos que envolvem o cálculo de números que seguem tal formatação (IEEE, 1985).

A *FFT/IFFT* de quatro pontos foi desenvolvida neste projeto utilizando duas *FFT/IFFT* de dois pontos. Além desses dois tipos de *FFT/IFFT* foram implementadas neste projeto uma *FFT/IFFT* de oito pontos, uma *FFT/IFFT* de dezesseis pontos, uma *FFT/IFFT* de cento e vinte oito pontos e uma *FFT/IFFT* de duzentos e cinquenta e seis pontos. A **Figura 25** mostra a organização interna do diagrama de blocos para cada *FFT/IFFT*.

Toda *FFT/IFFT* desenvolvida nesse projeto recebe as suas entradas em streaming e disponibiliza suas saídas em streaming. Quando um dado deve ser utilizado no processamento o sinal *i_valid* deve ser ativado. Por outro lado o sinal *o_valid* é ativado sempre que um dado colocado na saída da *FFT/IFFT* é resultado do processamento.

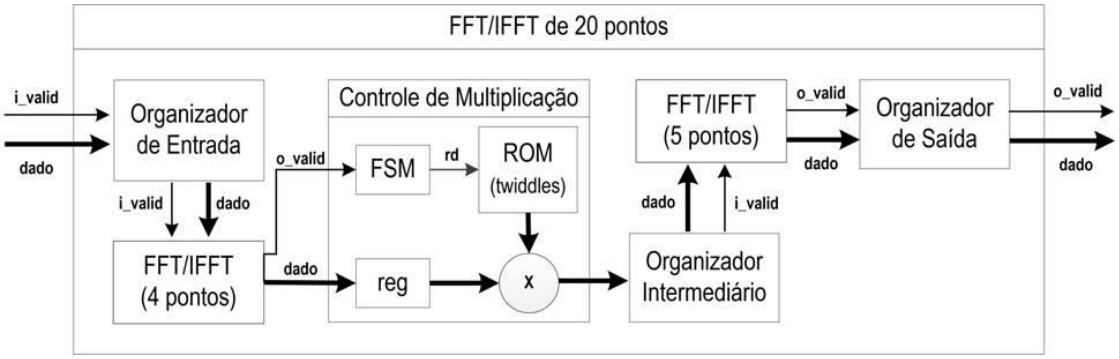
Como a *FFT* de dois pontos é a base para a construção de todas as outras citadas anteriormente, existe a necessidade de multiplicação de dados complexos apenas no módulo *Controlador de Multiplicação*. Assim, existe uma necessidade mínima de módulos multiplicadores complexos para a construção de uma *FFT/IFFT*. Isso é importante e é uma das contribuições deste trabalho, pois permite que cada *FFT/IFFT* seja sintetizada para vários tipos de *FPGA*, inclusive para aqueles que possuem poucas unidades de *DSP*. O multiplicador complexo desenvolvido neste trabalho utiliza multiplicadores de números em formato de ponto flutuante desenvolvidos e disponibilizados para uso pela Altera (ALTERA, 2016).

Figura 25 - Conjunto de diagramas de blocos



A **Figura 26** retoma a discussão de como o *Organizador de Entrada*, o *Organizador Intermediário* e o *Organizador de Saída* manipulam os dados que são processados em cada *FFT/IFFT* implementada neste projeto. Observando a disposição dos dados apresentada na **Figura 26** é possível compreender a configuração e a organização dos dados entre as etapas de processamento da uma *FFT/IFFT*. Para demonstrar que essa organização dos dados pode ser utilizada para diferentes tamanhos de *m*, neste exemplo foi utilizada uma *FFT/IFFT* de vinte pontos. Tal *FFT/IFFT* é formada por uma *FFT/IFFT* de quatro pontos e uma *FFT/IFFT* de cinco pontos.

Figura 26 - Organização intermediária dos dados em uma FFT/IFFT de 20 pontos



Dados de entrada da FFT de 20 pontos

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}	x_{17}	x_{18}	x_{19}	x_{20}

Dados de saída do módulo Organizador de Entrada

x_1	x_6	x_{11}	x_{16}	x_2	x_7	x_{12}	x_{17}	x_3	x_8	x_{13}	x_{18}	x_4	x_9	x_{14}	x_{19}	x_5	x_{10}	x_{15}	x_{20}

Dados de saída do módulo FFT de 4 pontos

F_1^4	F_6^4	F_{11}^4	F_{16}^4	F_2^4	F_7^4	F_{12}^4	F_{17}^4	F_3^4	F_8^4	F_{13}^4	F_{18}^4	F_4^4	F_9^4	F_{14}^4	F_{19}^4	F_5^4	F_{10}^4	F_{15}^4	F_{20}^4

Sequencia de leitura dos twiddles armazenados na ROM do módulo Controle de Multiplicação

W_{11}	W_{21}	W_{31}	W_{41}	W_{12}	W_{22}	W_{32}	W_{42}	W_{13}	W_{23}	W_{33}	W_{43}	W_{14}	W_{24}	W_{34}	W_{44}	W_{15}	W_{25}	W_{35}	W_{45}

Resultado do Controle de Multiplicação reorganizado pelo módulo Organizador Intermediário

M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}	M_{16}	M_{17}	M_{18}	M_{19}	M_{20}

Dados de saída do módulo FFT de 5 pontos

F_1^5	F_2^5	F_3^5	F_4^5	F_5^5	F_6^5	F_7^5	F_8^5	F_9^5	F_{10}^5	F_{11}^5	F_{12}^5	F_{13}^5	F_{14}^5	F_{15}^5	F_{16}^5	F_{17}^5	F_{18}^5	F_{19}^5	F_{20}^5

Dados de saída do módulo Organizador de Saída

F_1^5	F_6^5	F_{11}^5	F_{16}^5	F_2^5	F_7^5	F_{12}^5	F_{17}^5	F_3^5	F_8^5	F_{13}^5	F_{18}^5	F_4^5	F_9^5	F_{14}^5	F_{19}^5	F_5^5	F_{10}^5	F_{15}^5	F_{20}^5

Devemos destacar que a escolha e a ordem de utilização de cada *FFT/IFFT* vai alterar a organização dos dados entre as etapas de processamento, ou seja, utilizar primeiro uma *FFT/IFFT* de quatro pontos e depois uma *FFT/IFFT* de cinco pontos para construir uma *FFT/IFFT* de vinte pontos é diferente de utilizar uma *FFT/IFFT* de cinco pontos seguida de uma *FFT/IFFT* de quatro pontos. A forma de organização dos dados entre as etapas de processamento, porém, segue a mesma lógica observada na **Figura 26**.

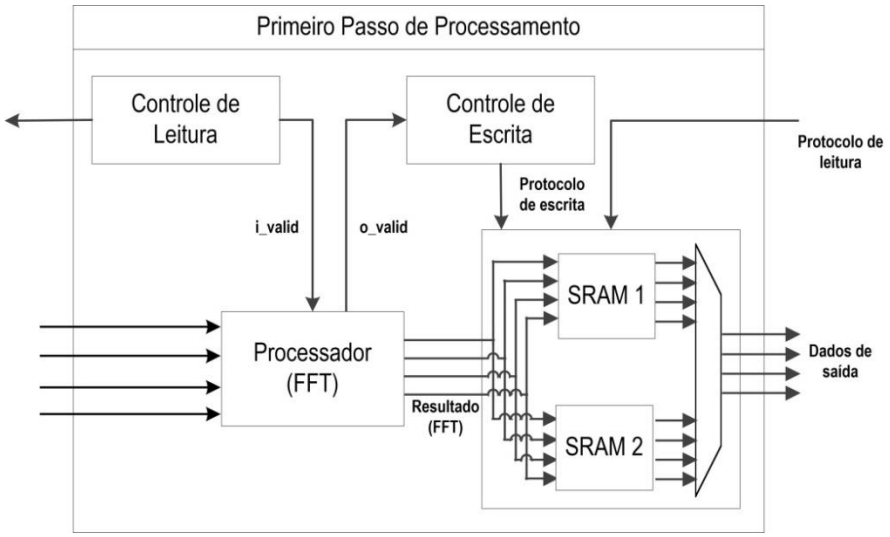
A partir das próximas seções serão abordados os módulos que compõem cada um dos passos de processamento que formam a arquitetura para segmentação de vídeos desenvolvida neste trabalho. Em cada um desses passos são instanciados quatro módulos do tipo *FFT/IFFT* apresentado nesta seção.

4.4. Primeiro Passo de Processamento

O primeiro passo de processamento é formado por um conjunto de módulos responsável por requisitar entrada de dados para todo o processamento que será executado no *FPGA*, por promover o cálculo da *FFT* na primeira dimensão dos dados e por armazenar os valores resultantes do processamento para serem utilizados no próximo passo. A **Figura 27** apresenta a estrutura interna do primeiro passo de processamento.

O *Controle de Leitura* é formado por uma *FSM* e se comunica diretamente com o módulo *Multiport* para fazer requisições de leitura dos dados que são a entrada para o algoritmo de segmentação de vídeos. Para isso, tal controlador foi projetado para executar todo o protocolo de requisição de dados definido pela Gidel para a leitura de dados junto ao módulo *Multiport* (GIDEL, 2010). Isso torna o *Controle de Leitura* do primeiro passo de processamento dependente da forma como os dados devem ser requisitados ao *Multiport*. Caso seja necessário fazer a requisição de leitura a um módulo de integração diferente do Multiport, é preciso adaptar a *FSM* do *Controlador de Leitura* para se adequar a tal módulo de integração. Levando em consideração a quantidade total de módulos desenvolvidos para este projeto, é importante ressaltar que a modificação de apenas um módulo, neste caso o *Controlador de Escrita* do primeiro passo de processamento, para a adaptação a uma nova plataforma de desenvolvimento não é um fator que inviabiliza a implementação da arquitetura aqui proposta em outras plataformas.

Figura 27 - Diagrama de blocos do primeiro passo de processamento



O *Controle de Leitura* também monitora a recepção dos dados vindo do *Multiport* e envia o sinal *i_valid* para o módulo *Processador* quando tais dados devem ser utilizados para o cálculo de *FFT* da primeira dimensão. A cada requisição de leitura são recebidos quatro dados, um de cada linha do *frame* que está sendo lido, portanto, são lidas quatro linhas independentes e o cálculo da *FFT* pode ser executado de forma paralela utilizando os dados de cada linha. Para isso o módulo *Processador* é composto por quatro módulos responsáveis por calcular a *FFT* de n pontos sobre a primeira dimensão dos *frames* de tamanho igual a $n\ x\ n$ *pixels*.

O *Controle de Escrita* monitora a saída de dados do *Processador* através do sinal *o_valid* e coordena a escrita de tais dados na unidade de memória do primeiro passo, que neste caso é composta por duas memórias *SRAM* da Altera. Aqui são utilizadas duas memórias para que o processamento do primeiro passo não fique a maior parte do tempo parado esperando que o *Controlador de Leitura* do segundo passo execute a leitura de todos os dados para a execução da *FFT* na segunda dimensão.

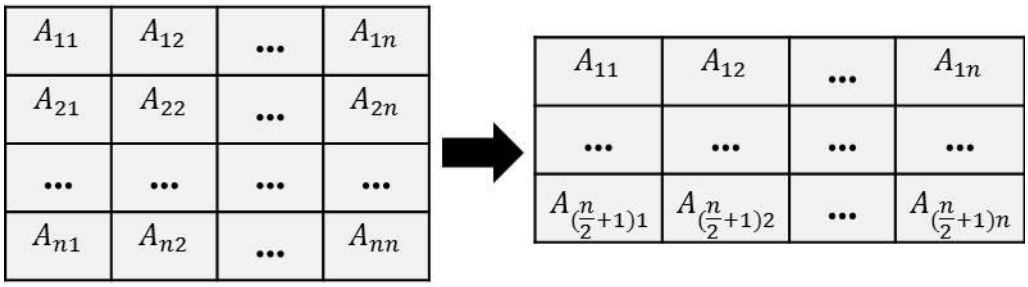
Assim que todos os dados de um *frame* são processados e escritos na primeira *SRAM* o *Controlador de Escrita* do primeiro passo envia um sinal para o *Controlador de Leitura* do segundo passo informando que os seus dados estão prontos para serem lidos. Dessa forma o *Controlador de Leitura* do segundo passo começa a ler os dados armazenados na primeira memória enquanto que o *Controlador de Escrita* do primeiro passo começa a monitorar a escrita dos dados na segunda *SRAM*.

Em cada memória utilizada neste passo existe um controlador interno que gerencia a leitura e a escrita dos dados. Tal controlador manipula os endereços de escrita para que os dados sejam escritos no sentido das colunas e manipula os endereços de leitura para que os dados sejam lidos no sentido das linhas. Dessa forma os dados que são escritos nas memórias ficarão transpostos em relação aos dados de entrada, anulando assim a transposição dos *frames* realizada na etapa de pré-processamento e que foi explicada na primeira seção deste capítulo.

A propriedade da *FFT* conhecida como *Simetria Hermitiana*, que foi apresentada no segundo capítulo deste trabalho, tem influência direta nos tamanhos das memórias que são instanciadas no primeiro passo de processamento e que são utilizadas para armazenamento dos resultados gerados pelo módulo *Processador*. Cada *pixel* dos *frames* de entrada é representado por um inteiro com oito bits, ou seja, cada ponto a ser processado na *FFT* do primeiro passo é puramente real. Logo para um resultado de processamento com N_0 pontos é necessário armazenar apenas $(N_0/2) + 1$ desses pontos, pois os demais podem ser

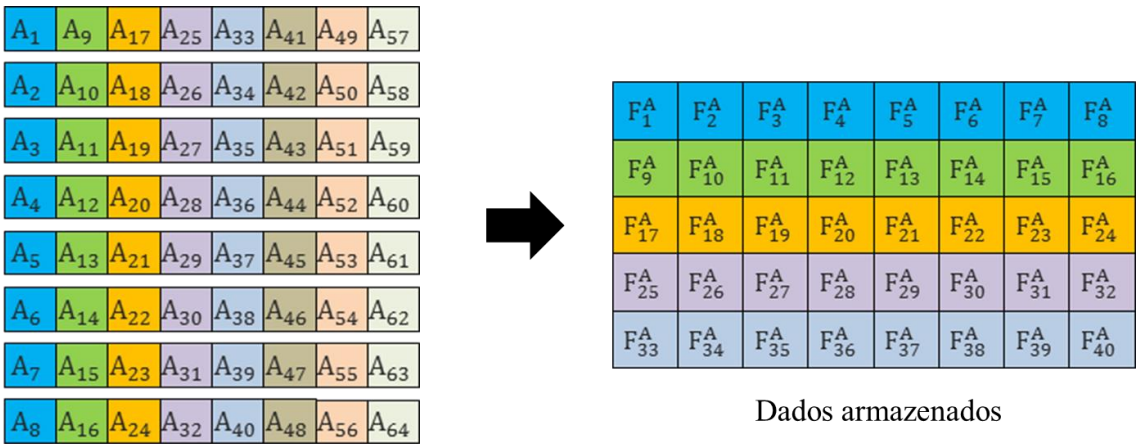
recuperados a partir daqueles que foram armazenados. Aqui vale destacar que os resultados gerados pelo módulo Processador em cada ponto são dois números de ponto flutuante de precisão simples, parte real e imaginária, para cada *pixel* de entrada, ou seja, sessenta e quatro bits. A **Figura 28** traz uma representação gráfica de como tal propriedade afeta a quantidade de dados que necessariamente precisam ser armazenados após o cálculo da *FFT* ser executado na primeira dimensão do *frame* de entrada.

Figura 28 - Representação gráfica do armazenamento do primeiro passo



Podemos notar pela **Figura 28** que a quantidade de dados a serem armazenados pode ser reduzida quase que pela metade. Quanto maior forem as dimensões do *frame* de entrada maior será a necessidade de armazenamento nas unidades de memória, portanto a *Simetria Hermitiana* ganha grande importância, pois através dela pode ser reduzida a quantidade de dados que ficam armazenados na memória interna do *FPGA*. Na **Figura 29** é apresentado um exemplo de como uma matriz entrada de tamanho igual $8x8$, previamente transposta, seria armazenada na unidade de memória do primeiro passo de processamento.

Figura 29 - Armazenamento dos dados no primeiro passo de processamento



Frames de Entrada

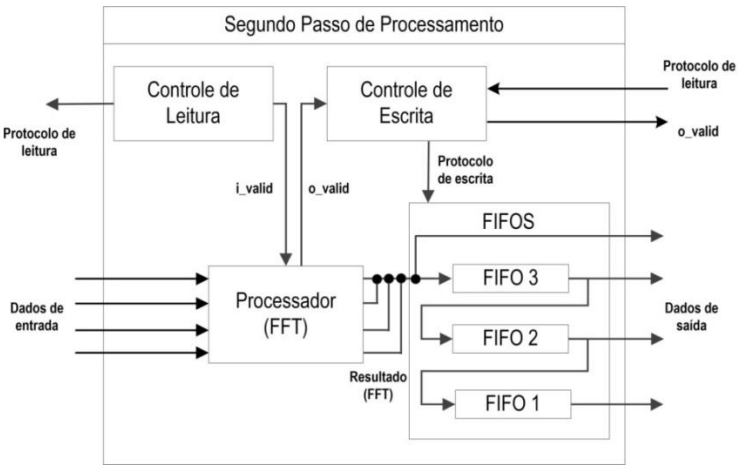
Dados armazenados

Tomando como exemplo um *frame* de vídeo com $128x128$ *pixels* e levando em consideração que após o processamento no primeiro passo cada *pixel* é representado por dois números em formato de ponto flutuante, ou seja, 32 bits para representar a parte real e 32 bits para a parte imaginária, é necessário armazenar nas unidades de memória do primeiro passo apenas 65 kbytes (532.480 bits) como resultado do processamento da *FFT* na primeira dimensão. Caso a *Simetria Hermitiana* não fosse levada em consideração, seria necessário armazenar 128 kbytes (1.048.576 bits) para esse mesmo tamanho de *frame*. Na seção seguinte é possível ver como essa redução de necessidade de armazenamento teve impacto direto no tamanho da unidade de memória utilizada no segundo passo de processamento.

4.5. Segundo Passo de Processamento

O segundo passo de processamento é o conjunto de módulos responsáveis por ler os dados armazenados do passo anterior, executar a *FFT* na segunda dimensão dos dados, organizar e enviar os resultados da *FFT 2-D* para o terceiro passo de processamento. A **Figura 30** mostra o diagrama de blocos do segundo passo de processamento. O módulo *Controlador de Leitura* requisita, através do protocolo de leitura, os dados que estão armazenados no primeiro passo de processamento e os encaminha juntamente com o sinal *i_valid* para o módulo *Processador*, que possui quatro *FFTs* de tamanho n definido, assim como nos outros passos, através do parâmetro de síntese denominado *matrix_size*. Dessa forma, são lidas quatro linhas de dados para serem processadas de forma independente e paralela nos quatro módulos capazes de calcular cada *FFT* de tamanho n.

Figura 30 - Diagrama de blocos do segundo passo de processamento



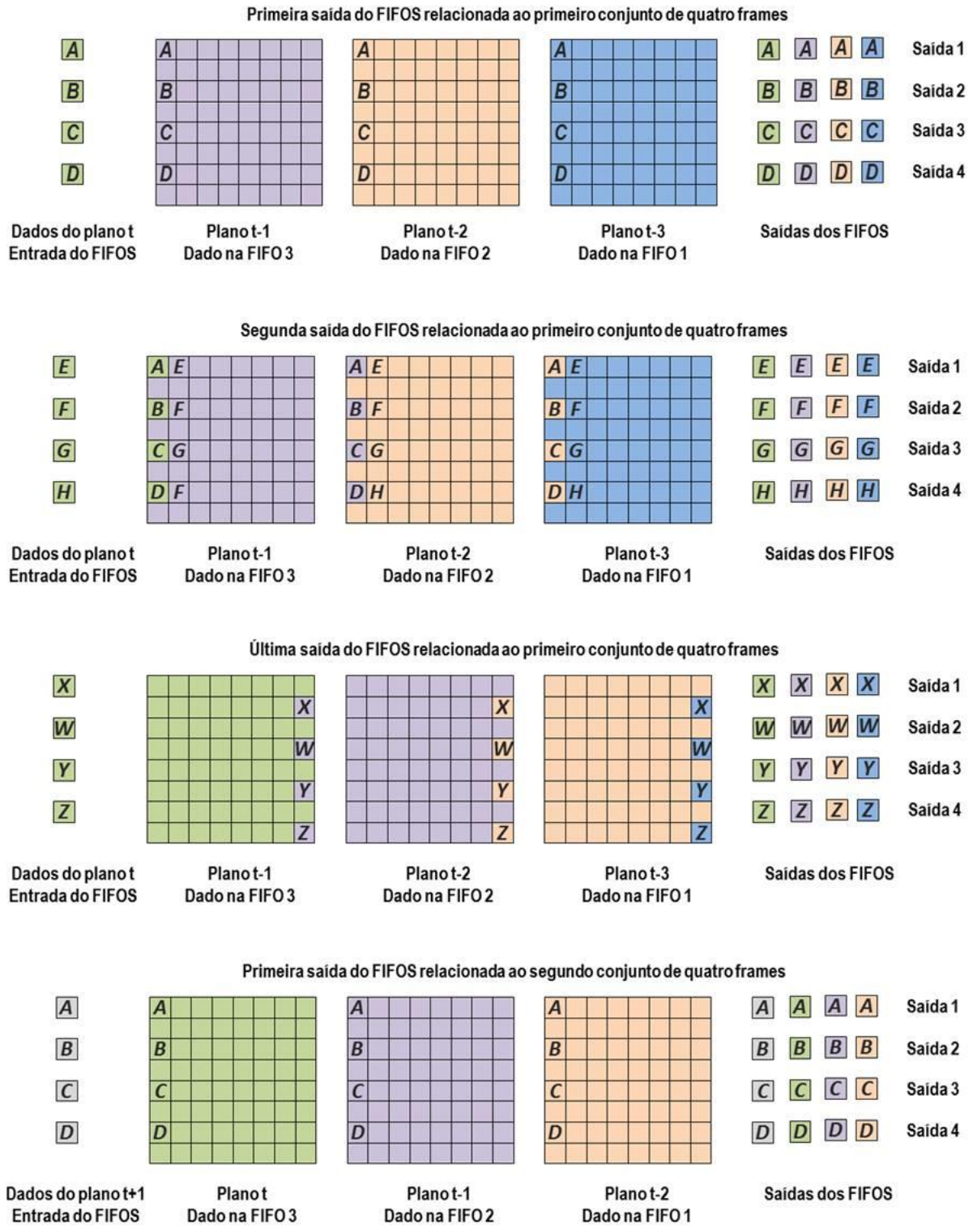
O módulo *Controlador de Escrita* monitora as saídas do módulo *Processador* e organiza a escrita de dados na *Unidade de Memória* do segundo passo, denominado *FIFOS*, que neste passo funciona como uma unidade de memória para armazenamento temporário de resultados intermediários do processamento. O *Controlador de Escrita* recebe o sinal *o_valid* vindo do *Processador* e providencia a escrita dos dados gerados em uma das unidades de armazenamento do módulo *FIFOS*. Dessa forma os dados ficam corretamente organizados para o processamento no próximo passo. Além disso, o *Controlador de Escrita* recebe o sinal com o protocolo de leitura vindo do terceiro passo. Quando as unidades de processamento do terceiro passo estão prontas para receber dados é tarefa do *Controlador de Escrita* do segundo passo permitir que os dados passem para a próxima etapa de processamento.

A *Unidade de Memória* do segundo passo de processamento, o módulo *FIFOS*, é responsável por armazenar todos os dados que serão necessários para executar a *FFT* na terceira dimensão dos *frames* de vídeo. Para isso são instanciadas e interligadas unidades de armazenamento da Altera que implementam o protocolo *First In,First Out (FIFO)*. Isso é feito para sincronizar e organizar os frames de vídeo que serão utilizados para o cálculo realizado no terceiro passo de processamento. A **Figura 31** mostra um exemplo de como *frames* de vídeo de tamanho $8x8$ trafegariam entre as unidades de armazenamento instanciadas para formar a *Unidade de Memória* do segundo passo.

Neste exemplo começamos, com uma situação onde a *FFT 2-D* já foi executada em três *frames* e o primeiro conjunto de quatro dados do *frame* t, que é o quarto *frame* resultante desse processamento *2-D*, está na entrada do módulo *FIFOS*. Podemos ver que os dados de entrada são diretamente encaminhados para a saída juntamente com os primeiros conjuntos de quatro dados dos três *frames* que já estavam armazenados no *FIFOS*. Ao mesmo tempo o primeiro conjunto de dados do *frame* $t-3$ é descartado da *FIFO 1* e em seu lugar o primeiro conjunto de dados do frame $t-2$ é armazenado. Algo semelhante acontece nas demais unidades de armazenamento para formar a segunda configuração do *FIFOS* apresentada na **Figura 31**.

Quando o último conjunto de quatro dados relacionados ao *frame* t é processado no segundo passo temos a terceira configuração de dados do *FIFOS* apresentada na **Figura 31**. Podemos perceber que os dados do *frame* $t-2$ estão quase completamente armazenados na *FIFO 1*, assim como os dados do *frame* $t-1$ estão na *FIFO 2* e os dados do *frame* t estão no *FIFO 3*. Quando o primeiro conjunto de quatro dados do *frame* $t+1$ chega à entrada do módulo *FIFOS* os *frames* já estão reorganizados para o início do processamento da *FFT* na terceira dimensão do segundo conjunto de quatro *frames*.

Figura 31 - Organização dos dados no módulo FIFOS



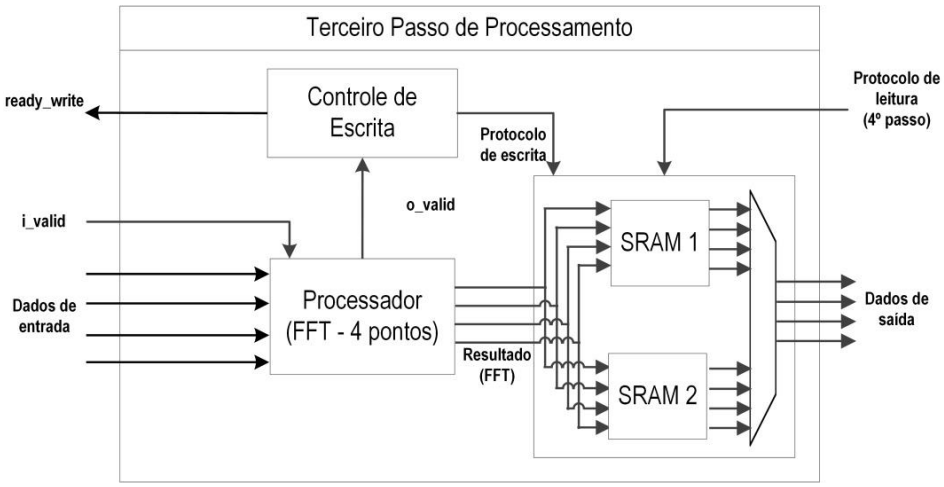
A organização interna dos dados no *FIFOS* garante que a necessidade de leitura dos dados na memória externa do *FPGA* seja mínima, pois tais dados ficam disponíveis para serem utilizados nos passos subsequentes da arquitetura e em nenhum momento é necessário escrever ou ler resultados intermediários na memória externa do *FPGA*. A propriedade da

Simetria Hermitiana apresentada no segundo capítulo deste trabalho também é utilizada para reduzir a quantidade de dados armazenados no módulo *FIFOS*. Como até o final do cálculo da *FFT 3-D* não é necessário recuperar todos os dados que compõem os *frames* resultantes do processamento, apenas um *frame* de tamanho $n(n/2+1)$ é armazenado em cada *FIFO*, isto para o caso de *frames* de entrada com tamanho $n\,x\,n$. Essa redução da necessidade de armazenamento permite uma economia de aproximadamente metade da memória interna do *FPGA* que é utilizada para a implementação do *FIFOS*, sendo ela crucial para que a arquitetura possa processar *frames* de vídeo de grandes dimensões em alta velocidade.

4.6. Terceiro Passo de Processamento

O terceiro passo de processamento possui uma organização interna que difere dos demais passos que compõem a arquitetura para segmentação de vídeos. Como pode ser visto no diagrama de blocos apresentado na **Figura 32**, não existe um módulo *Controlador de Leitura*. Tal ausência é justificada por ser o módulo *Controlador de Escrita* do segundo passo o responsável por enviar os dados necessários para o cálculo da *FFT* na terceira dimensão. Isso foi feito para que a *Unidade de Memória* do segundo passo não tenha que ficar esperando uma requisição de leitura para enviar os dados para o processamento no passo subsequente, tornando o processamento entre o segundo e o terceiro passo livre de retardos relacionados ao protocolo de leitura.

Figura 32 - Diagrama de blocos do terceiro passo de processamento



O sinal *ready_write*, que sai do *Controlador de Escrita* do terceiro passo de processamento, é enviado para o passo anterior é e informa que existe pelo menos uma unidade de memória disponível para receber os resultados do processamento realizado no terceiro passo. De posse de tal informação o *Controlador de Escritas* do segundo passo despacha os dados corretos para o processamento no passo seguinte.

O *Controlador de Escritas* do terceiro passo monitora os dados que saem do *Processador* através do sinal *o_valid* e coordena o seu armazenamento nas memórias *SRAM*. Como explicado anteriormente, esse controlador monitora a disponibilidade de espaço para a escrita de dados nas memórias e informa sobre tal disponibilidade ao segundo passo de processamento. O monitoramento da disponibilidade de espaço é realizado através de uma comunicação com o módulo *Controlador de Leitura* do quarto passo de processamento. Quando tal *Controlador de Leitura* termina de ler todos os dados de uma das memórias um sinal é enviado ao *Controlador de Escrita* do terceiro passo informando que a memória está disponível para ser utilizada, sendo essa a forma como o *Controlador de Escrita* do terceiro passo sabe quando as suas memórias estão disponíveis para a escrita de dados.

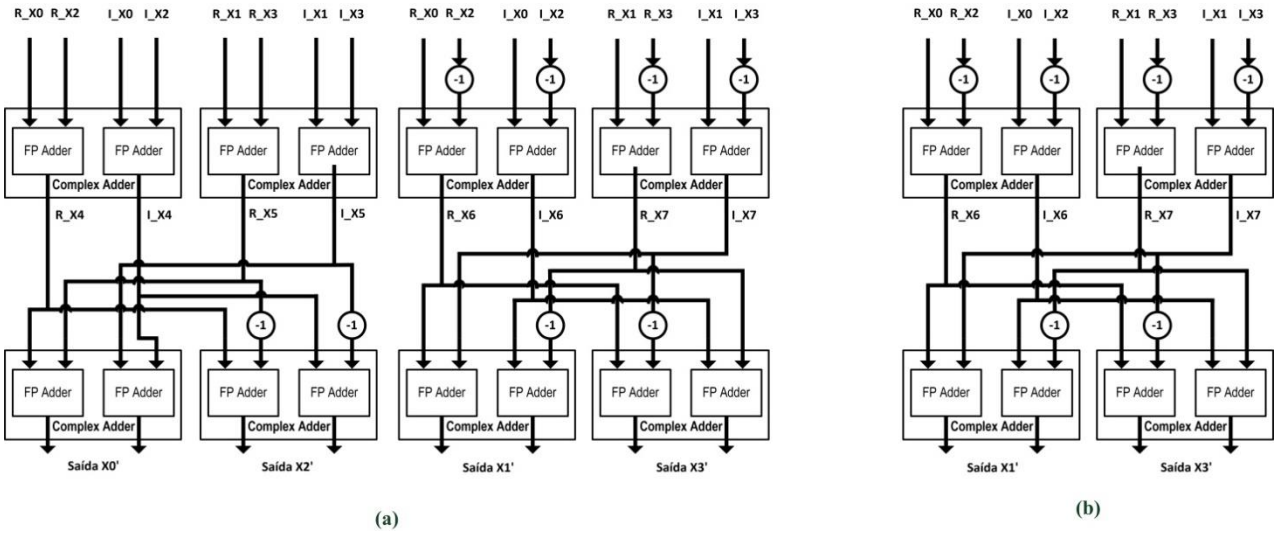
O módulo *Processador* é formado por quatro unidades que calculam uma *FFT* de quatro pontos. Diferentemente dos *Processadores* dos outros passos de processamento, cuja computação da *FFT/IFFT* é feita em *streaming*, no módulo *Processador* do terceiro passo o cálculo de *FFT* de quatro pontos é executado de forma paralela. Sendo assim, todas as entradas de dados necessárias para o processamento de cada *FFT* de quatro pontos devem estar disponíveis ao mesmo tempo na entrada do módulo *Processador* e todos os dados resultantes são disponibilizados ao mesmo tempo nas portas de saída de tal módulo.

Foi adotada a estratégia de executar em paralelo a computação de cada *FFT* de quatro pontos para aproveitar a organização e a quantidade de dados disponibilizados ao mesmo tempo pela unidade de memória do segundo passo de processamento. Além disso, a computação em paralelo tende a tornar mais rápido o processamento executado no terceiro passo, uma vez que, todas as saídas do *Processador* estão disponíveis ao mesmo tempo para serem escritas nas unidades de memória do terceiro passo.

Inicialmente foi projetado e desenvolvido para o cálculo da *FFT* de quatro pontos um módulo capaz de executar tal processamento de forma totalmente paralela, ou seja, quatro dados complexos são enviados para o módulo e como resposta saem quatro dados complexos da forma apresentada na **Figura 33 (a)**. Mais a frente neste texto é visto que apenas as saídas X_1' e X_2' da *FFT* de quatro pontos são relevantes para as próximas etapas do algoritmo de segmentação de vídeos, portanto, a *FFT* de quatro pontos utilizada neste projeto teve seu

tamanho reduzido pela metade, ficando com o seu diagrama de blocos como o apresentado na **Figura 33 (b)**.

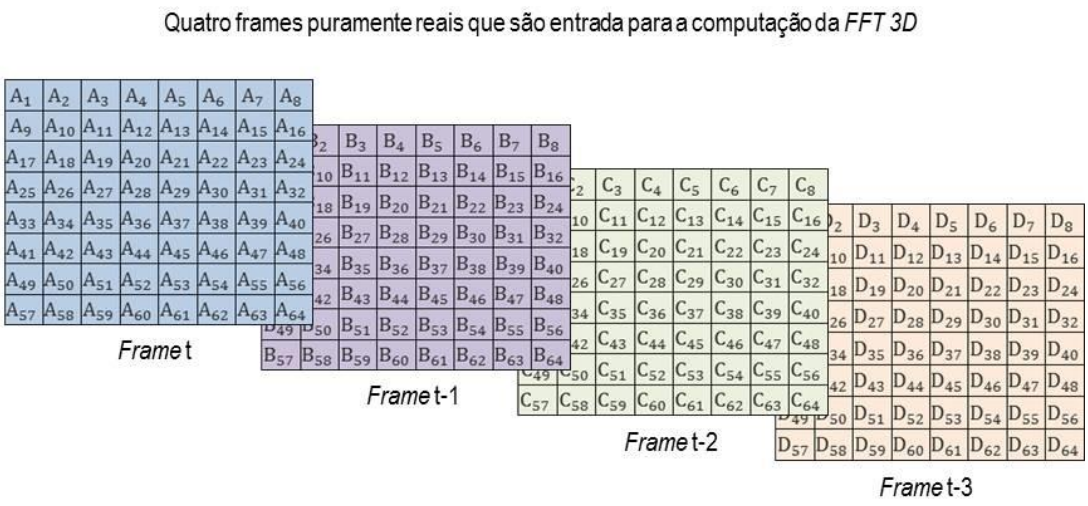
Figura 33 - Distintas implementações da FFT de quatro pontos

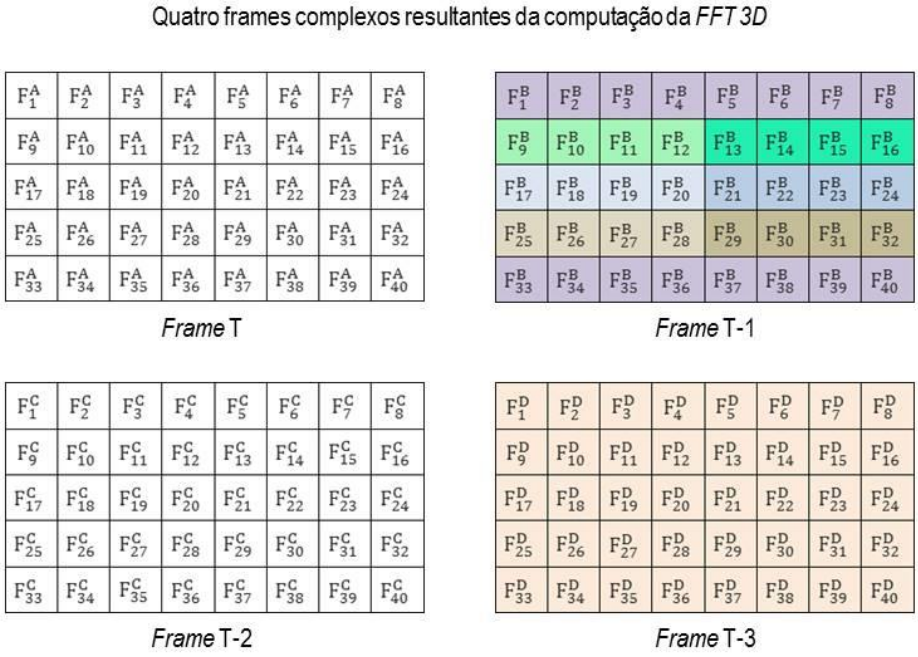


Todas as entradas e saídas da *FFT* de quatro pontos são números complexos onde tanto a parte real quanto a parte imaginária seguem o formato de notação de ponto flutuante de precisão simples especificado no padrão *IEEE-754* (IEEE, 1985). Essa precisão foi adotada para que a computação executada no *FPGA* não fosse afetada por perda de informações que poderiam ocorrer com a modificação do formato ou da precisão dos dados processados no terceiro passo. Como podemos notar a partir do diagrama de blocos da *FFT* de quatro pontos, existe uma grande necessidade de somadores para operar os sinais de entrada. Todos os somadores complexos que aparecem na **Figura 33 (b)** também foram implementados com somadores desenvolvidos e disponibilizados para uso pela Altera (ALTERA, 2015a).

Após o final do cálculo da *FFT* sobre a terceira dimensão de dados relacionados a um grupo de quatro frames (a profundidade temporal escolhida para o processamento) surge a necessidade de recuperar e armazenar todos os dados que foram descartados no primeiro e segundo passo de processamento por conta da propriedade de *Simetria Hermitiana*. Os resultados que são relevantes para o processamento realizado no quarto passo estarão disponíveis cada vez que o sinal *o_valid* for ativado na saída do *Processador*. A **Figura 34** mostra através de um exemplo a uma forma geral como é feita a composição do *frame* resultante do processamento da *FFT 3-D*, entrada para o cálculo do espectro de fase de Fourier.

Figura 34 - Composição dos frames de saída do terceiro passo





Na **Figura 34** temos um grupo de quatro frames de tamanho $8x8$ que é a entrada disponibilizada para o cálculo nos primeiros três passos de processamento. Na sequência são apresentados os quatro *frames* resultantes do cálculo da *FFT 3-D*. Podemos notar que tal resultado é formado por *frames* com cinco linhas e oito colunas. Como explicado anteriormente, a *Simetria Hermitiana* permite, para esse caso, que três das oito linhas sejam descartadas em cada *frame* que sai do primeiro passo, por isso os *frames* resultantes do cálculo da *FFT* na terceira dimensão dos dados também possui apenas cinco linhas.

Neste exemplo o resultado final da *FFT 3-D* que deve ser armazenado em uma das memórias do terceiro passo é formado apenas por dados relacionados aos *frames* de entrada $t-1$ e $t-3$. Os dados relacionados aos *frames* t e $t-2$ são utilizados durante o processamento, mas não são aproveitados para formar o *frame* resultante $T-3$. Por esse motivo cada *FFT* de quatro pontos teve seu tamanho reduzido pela metade da forma com foi explicado anteriormente. Observando o *frame* completo $T-3$ podemos notar que os dados das três primeiras linhas do frame $T-2$ sofreram uma rotação de 180º para compor o resultado do final da *FFT 3-D* aplicada sobre o conjunto de quatro frames fornecidos como entrada para seu processamento.

A *Simetria Hermitiana* é importante porque permite que a quantidade de dados armazenadas nas *Unidades de Memória* do primeiro e do segundo passos seja reduzida. Além disso, a quantidade de dados a serem processados em tais passos é reduzida quase que pela metade. Essas são vantagens que não devem ser deixadas de lado quando se constrói uma arquitetura de processamento baseada em *FPGA*, isso devido ao limite interno de recursos destes dispositivos.

A *Unidade de Memória* do terceiro passo de processamento é composta por duas memórias SRAM e um pequeno controlador baseado em uma máquina de estados finitos. Esta unidade organiza a escrita ordenada dos dados que saem do módulo *Processador*. Sendo assim, tal controlador implementa toda a recuperação dos dados para formar cada *frame* que deve ser escrito em uma das memórias. As *FSMs* relacionadas ao controlador das *Unidades de Memória* podem ser encontradas, assim como todas as outras, nos anexos deste trabalho.

4.7. Módulo para Cálculo do Espectro de Fase de Fourier

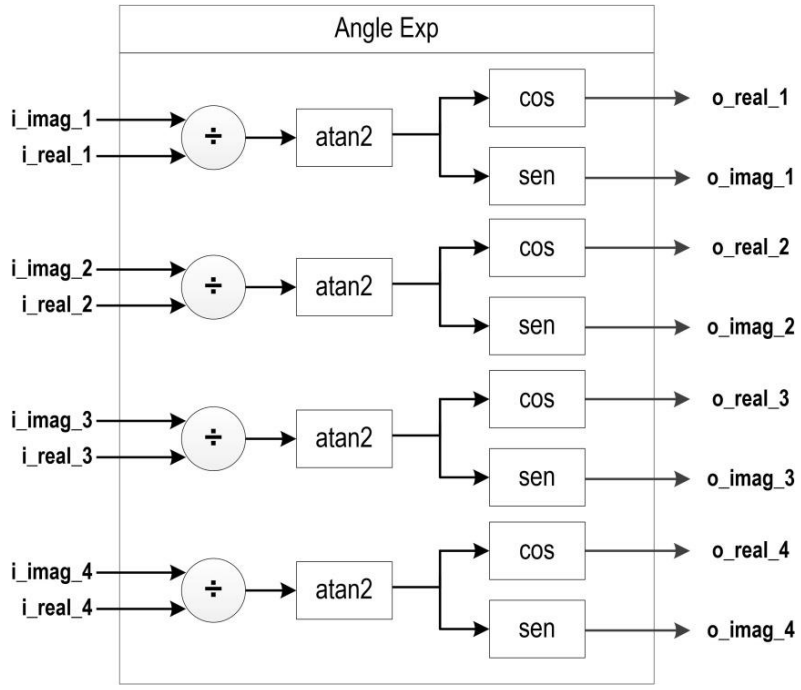
Após o processamento da *FFT 3-D* sobre um conjunto de t *frames* de vídeo é necessário extrair a informação relacionada ao movimento que está contida no primeiro *frame* do conjunto t que foi processado. Para o caso de $t=4$, o *frame* que será utilizado para extrair a informação de movimento é o $t-3$, ou seja, o primeiro sobre o qual foi processada a *FFT*

2-D. A equação (4.1) mostra como a informação relacionada ao movimento pode ser extraída dos dados resultantes da *FFT 3-D*. Em tal equação F é a Transformada de Fourier *3-D* calculada sobre o conjunto I de dados, P é o ângulo de fase de Fourier e F^{-1} é a Transformada Inversa de Fourier. O valor Î é o resultado final onde está mensurada a quantidade de movimento relacionada a cada pixel do frame de entrada.

$$\hat{I}(x,y,t) = \left|F^{-1}\{e^{iP[F(I(x,y,t))]}\}\right|^2 \qquad (4.1)$$

Na arquitetura desenvolvida neste trabalho o cálculo da Transformada Inversa de Fourier é executado no quarto e no quinto passo de processamento. O cálculo de $e^{iP[F(I(x,y,t))]}$ é executado no módulo *Angle Exp* cuja organização interna é demonstrada na **Figura 35**.

Figura 35 - Diagrama de blocos do módulo Angle Exp



Na arquitetura para segmentação de vídeos o módulo *Angle Exp*, que calcula o espectro de fase de Fourier após o cálculo da *FFT 3-D*, foi implementado de duas formas: uma utilizando *IP-Cores* da Altera, que operam com números de ponto flutuante, para o cálculo dos valores de seno e cosseno e a outra mapeando os valores de seno e cosseno em memórias do tipo *ROM*. A segunda abordagem pode fazer que o módulo *Angle Exp* gere resultados menos precisos do que quando são utilizados os *IP-Co*res da Altera, por outro lado,

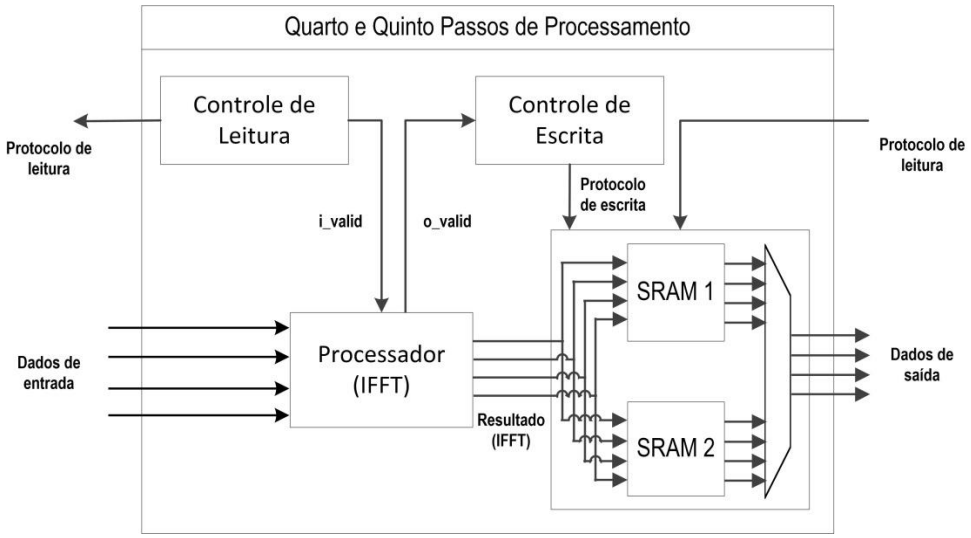
a segunda abordagem leva a um menor consumo de *DSP* e isso permite que a arquitetura seja sintetizada para FPGAs com um número reduzido destes recursos.

Podemos ver através da **Figura 35** que para cada entrada complexa fornecida para o módulo *Angle Exp* é utilizado um divisor para dividir o valor imaginário pelo valor real. O resultado da divisão é utilizado no módulo atan2 para calcular a tangente inversa, ou o ângulo de fase, a partir do valor complexo que originou tal divisão. Por fim, os módulos *sen* e *cos* são utilizados para calcular o valor da exponencial que será a entrada para o cálculo da *IFFT* nos próximos passos de processamento. Essa é uma descrição simples do funcionamento do *Angle Exp*.

4.8. Quarto e Quinto Passos de Processamento

O quarto e o quinto passos de processamento apresentam o mesmo conjunto de módulos e operam de forma igual sobre seus dados de entrada. Assim como nos outros passos, um *Controlador de Leitura* é responsável por ler os dados armazenados no passo anterior, monitorar a sua recepção e indicar para o *Processador* quando tais dados são válidos através do sinal *i_valid*. A **Figura 36** mostra o diagrama de blocos do quarto e do quinto passo de processamento.

Figura 36 - Diagrama de blocos do quarto e do quinto passo de processamento



O Processador instanciado no quarto e no quinto passo é formado por quatro elementos de computação que executam o cálculo da *IFFT*. O Controlador de Escrita

monitora os dados que saem do *Processador* observando o sinal *o_valid*. Quanto tal sinal se torna ativo o dado que está na saída do *Processador* é válido e o *Controlador de Escrita* envia a ordem de escrita para a *Unidade de Memória*, que nestes passos é formada por duas memórias *SRAM* que armazenam os dados na memória interna do *FPGA*. Foram utilizadas duas memórias para evitar que algum atraso nas leituras geradas pelo passo seguinte provocasse uma interrupção do processamento. Essa abordagem também foi adotada no primeiro e no terceiro passo de processamento.

A *IFFT* é calculada sobre as duas dimensões do plano com o espectro de fase que sai do módulo *Angle Exp*. Para tornar a leitura e a escrita de dados idênticas no quarto e no quinto passo de processamento foi implementado um pequeno controlador, baseado em uma máquina de estados, que opera juntamente às unidades de memória sendo responsável por manipular os endereços de leitura e escrita onde os dados de processamento resultante da *IFFT* serão armazenados. As escritas de dados são executadas na direção das colunas enquanto que a leituras são executadas na direção das linhas. Isso permite que o processamento seja executado sem a necessidade de transposição dos dados entre os passos de processamento, pois tal transposição já é feita quando são adotadas diferentes direções para a leitura e a escrita de dados. A *FSM* que controla as leituras e escritas nas unidades de memória do quarto e do quinto passo de processamento, assim como as outras de todo este projeto, estão nos anexos desta dissertação.

A *Unidade de Memória* do quinto passo armazena o resultado final do processamento executado no *FPGA*. Um módulo *Controlador de Escritas* que fica localizado entre o quinto passo e o módulo *Multiport* é responsável por ler os dados que estão armazenados no último passo de processamento e encaminha-los para a escrita no *Multiport*, que é realmente o módulo responsável por executar as leituras e escritas nas memórias externa do *FPGA*. A escrita em tais memórias encerra o fluxo de processamento realizado na arquitetura para segmentação de vídeos descrita neste capítulo.

4.9. Conclusões

Neste capítulo foram apresentados todos os aspectos relacionados ao desenvolvimento de uma arquitetura heterogênea, baseada em *CPU* e *FPGA*, capaz de segmentar regiões que apresentam movimento em sinais de vídeo.

No início desse capítulo foi apresentada a metodologia de desenvolvimento empregada durante a implementação da arquitetura para a segmentação de *frames* de vídeo. Foram explicados os aspectos relacionados à etapa de estudos do algoritmo de segmentação de vídeos e foi explicada de forma simples a estrutura de testes utilizada para executar os testes unitários e os testes de integração dos módulos de hardware desenvolvidos para formar a arquitetura heterogênea, baseada em *CPU* e *FPGA*, que foi desenvolvida neste trabalho.

Em seguida foi apresentada uma visão geral da arquitetura heterogênea. Foi apresentado o diagrama de blocos genérico que poderia ser utilizado para a construção de qualquer sistema de computação massiva de dados em *FPGA*. Na sequência foi proposta uma arquitetura formada por cinco passos de processamento que seria capaz de implementar o algoritmo descrito no artigo *"Dynamic Texture Segmentation Using Fourier Transform"* (LI, *et. al*, 2009).

A implementação do módulo capaz de calcular a *FFT/IFFT* sobre uma massa de dados com n pontos foi abordada após a apresentação da visão geral da arquitetura. Foi mostrado como a propriedade de linearidade permitiu que o cálculo da *FFT/IFFT* fosse executado em etapas e como os dados intermediários resultantes de cada etapa deveriam ser organizados para a correta apresentação dos resultados finais produzidos por cada processador de *FFT/IFFT*.

O primeiro passo de processamento foi abordado na sequência do texto. Foi dado destaque ao fato de que a propriedade da *Simetria Hermitiana* foi utilizada para reduzir a quantidade de dados armazenada na *Unidade de Memória* do primeiro passos de processamento. Sobre o segundo passo de processamento foi dado grande destaque a forma como os *frames* ficam armazenados na sua *Unidade Memória* e como tais *frames* são encaminhados para o terceiro passo de processamento.

Para o terceiro passo de processamento foi dado destaque ao módulo responsável por executar o processamento da *FFT* de quatro pontos. Além disso, foi dada grande relevância a forma como os dados são armazenados na *Unidade de Memória* deste passo. Sobre o módulo *Angle Exp* foi explicado de forma sucinta sua operação através da apresentação do seu diagrama de blocos. Por fim, sobre o quarto e o quinto passo de processamento foi levado em consideração como os dados são lidos e escritos na *Unidade de Memória* e como isso torna simples o processamento realizado nesses passos finais de processamento.

Após a apresentação de todo o sistema que foi desenvolvido para o processamento de *frames* de vídeo em tempo real, destacamos os pontos fortes deste trabalho tendo como referência as métricas de comparação apresentadas na **Tabela 3,** que está no terceiro capítulo

deste texto. A arquitetura desenvolvida não utiliza nenhum modelo de *background*, construído previamente, para a computação da segmentação. Por modelar dinamicamente o movimento das regiões nos *frames* de vídeo, na arquitetura desenvolvida não é necessário ler, modificar e escrever na memória externa do *FPGA* os dados relacionados a um modelo de *background*. Portanto, não existe a necessidade de acesso à memória externa do *FPGA* para manipular ou alterar os dados relacionados a um modelo de *background*.

Nenhuma alteração é feita nos dados de entrada do algoritmo para que os mesmos possam ser processados no *FPGA*, ou seja, os dados processados no *FPGA* estão no mesmo formato que os dados processados em qualquer computador de propósito geral que segue o padrão IEEE-754 (IEEE, 1984). Isso evita que informações sejam perdidas por causa da modificação de representação dos dados e, consequentemente, não é inserido ruído no processamento, que poderia ser originado por tal mudança de representação. No próximo capítulo serão apresentados um conjunto de experimentos que demonstram a importância da não mudança da representação dos dados que representam os *pixels* processados na arquitetura desenvolvida.

No quinto capítulo deste trabalho são apresentadas as medições de velocidade operacional do sistema desenvolvido e fica constatado, através dos resultados, que a arquitetura desenvolvida possui capacidade de processamento superior a 1800 *frames* por segundo, que é uma taxa mais do que suficiente para o processamento de sinais de vídeo em tempo real. Com essa capacidade de processamento é possível utilizar a arquitetura desenvolvida para processar vídeos oriundos de mais de uma câmera ao mesmo tempo.

5. Experimentos e seus Resultados

Neste capítulo é descrito em detalhes um conjunto de experimentos que tem o intuito validar o algoritmo e a arquitetura de alto desempenho desenvolvida para a segmentação de sequências de vídeo. Inicialmente são abordadas as métricas utilizadas para a avaliação dos resultados de todos os testes demonstrados neste capítulo. Logo depois, são apresentados os testes em software que permitiram a análise de algumas características do algoritmo de segmentação de vídeo e a escolha de alguns parâmetros que foram utilizados para o desenvolvimento deste trabalho. Em seguida, são abordados os testes e os relatórios de síntese dos módulos que foram desenvolvidos para calcular a *FFT/IFFT*. Por fim, são apresentados os testes realizados para avaliar a precisão e a velocidade de processamento alcançada pela arquitetura desenvolvida para segmentação de sinais de vídeo. Além disso, é feita uma comparação entre a velocidade operacional alcançada com o uso da arquitetura, que é baseada em *CPU* e *FPGA*, e a velocidade operacional obtida com o uso de um sistema composto por *CPU* e *GPU*, que implementa o mesmo algoritmo de segmentação de vídeos proposto por (LI et al., 2009).

Neste capítulo é apresentado um conjunto de experimentos que foram realizados para avaliar e validar a correta operação da arquitetura proposta neste trabalho. Uma vez que os resultados gerados pelo algoritmo de segmentação de vídeo (LI et al., 2009) são máscaras binárias relacionadas a cada *frame*, a avaliação dos resultados produzidos por tal algoritmo é feita através da comparação entre as máscaras binárias produzidas. Por outro lado, quando é necessário avaliar os resultados gerados pelos módulos desenvolvidos para o cálculo da *FFT/IFFT*, a métrica escolhida para tal avaliação é a Relação Sinal Ruído *(SNR)*.

Para a avaliação do desempenho operacional da arquitetura desenvolvida neste trabalho, foram enviados para processamento em *FPGA* vídeos com diferentes quantidades de frames. Foram escolhidos vídeos com 10, 100, 1000, 2000, 5000 e 8000 *frames* cada, pois dessa forma foi possível avaliar o comportamento da arquitetura desenvolvida quando poucos ou muitos frames são enviados para o processamento.

Neste contexto foram executados por 100 vezes o envio, o processamento e a recepção dos dados relacionados a cada sequência de vídeo supracitada. Isso foi feito para tornar mais precisa a avaliação estatística dos tempos medidos em cada uma dessas etapas. Por fim, foram calculadas a média e o desvio padrão de cada um desses tempos. Estes exemplos foram implementados em uma arquitetura baseada em *CPU* e *GPU* de tecnologia similar à plataforma baseada em *CPU* e *FPGA* utilizada para o desenvolvimento deste trabalho.

Os resultados das execuções gerados nas duas arquiteturas são comparados e discutidos em detalhes nos itens a seguir, assim como as métricas utilizadas para análise e comparação destes resultados.

5.1. Métricas de Avaliação

A avaliação das máscaras binárias produzidas pelo algoritmo de segmentação quando são variados seus parâmetros de configuração é baseada em medidas originadas na área de Recuperação de Informações (HUANG, 2011) onde geralmente é observado o desempenho de classificadores que lidam com classes desbalanceadas. Nas máscaras binárias produzidas como resultado do algoritmo de segmentação não há garantias de que a quantidade de *pixels* de *foreground* é aproximadamente igual à de *pixels* de *background*, dessa forma é necessária alguma técnica de avaliação que contorne tal condição. Assim sendo, o algoritmo de segmentação é avaliado como um classificador binário que é capaz de indicar se os pixels de

cada frame são de *foreground* ou de *background* e cada indicação será adotada como um verdadeiro ou falso positivo e negativo.

Teremos um caso de verdadeiro positivo *(TP)* quando um *pixel* de *foreground* é classificado como de *foreground*. Da mesma forma, um caso de verdadeiro negativo *(TN)* acontece quando um *pixel* de *background* é classificado como tal. Temos um falso positivo *(FP)* quando um *pixel* classificado com *foreground* é na verdade de *background* e um falso negativo *(FN)* quando um *pixel* classificado como *background* é verdadeiramente de *foreground*. A totalidade destes casos permite que sejam calculadas quatro medidas para avaliação da classificação dos *pixels* (HUANG, 2011) e (GOMEZ et al., 2012): *Sensitividade, Precisão, Medida F* e *Similaridade*.

A *Sensitividade* apresentada na equação (5.1) é valor percentual dos *pixels* de um frame que são classificados corretamente como *foreground* dentre todos que realmente são de *foreground*. A *Precisão*, que está apresentada na equação (5.2), é o valor percentual de *pixels* de um *frame* que são corretamente classificados como *foreground* dentre todos que foram classificados como *foreground*.

$$Sensitividade\ (S) = \frac{TP}{TP+FN} \tag{5.1}$$

$$Precisão\ (P) = \frac{TP}{TP+FP} \tag{5.2}$$

A partir da *Sensitividade* e da *Precisão* é possível calcular *Medida F* que está apresentada na equação (5.3). Com tal medida é possível avaliar o desempenho de um classificador levando em consideração a média harmônica da sua *Sensitividade* e *Precisão*. Para que o valor da *Medida F* seja elevado é preciso que tanto a *Sensitividade* quanto a *Precisão* apresentem valores elevados e essa pode ser a vantagem de utilizar a *Medida F* para avaliação de classificadores.

$$F = \frac{2.P.S}{P+S} \tag{5.3}$$

Quando consideramos dois *frames*, nos quais todos os seus *pixels* estão classificados como *foreground* ou *background*, podemos afirmar o quão iguais são tais *frames* através da

medida de *Similaridade* apresentada na equação (5.4). Para isso, adotamos um *frame* como a referência e os valores de *TP*, *TN*, *FP* e *FN* são calculados levando em consideração as igualdades e diferenças nas classificações de cada *pixel*. Essa abordagem foi adotada para avaliar a Similaridade dos resultados produzidos pelo algoritmo de segmentação quando foram variados a profundidade temporal e o redimensionamento dos *frames* a serem segmentados.

$$Similaridade\ (SM) = \frac{TP}{TP+FP+FN} \tag{5.4}$$

A avaliação da precisão dos resultados apresentados por cada *FFT/IFFT* que foi desenvolvida para compor o sistema para segmentação de vídeo é feita através da *Relação Sinal-Ruído (SNR)* descrita na equação (5.5).

$$SNR = 10 \log_{10} \frac{P_{sinal}}{P_{ruído}} (dB) \tag{5.5}$$

Em tal formulação P_{sinal} é o resultado de referência produzido por uma aplicação em software que executa o cálculo da *FFT/IFFT* sobre uma quantidade variada da *N* pontos e $P_{ruído}$ é a diferença entre os resultados gerados por um módulo de hardware a sua referência em software. Geralmente a *SNR* é expressa em decibéis e quanto maior seu valor menor é a diferença dos resultados produzidos pelo módulo de hardware e por sua referência em software.

Essa avaliação da *Relação Sinal-Ruído* foi a mesma abordagem adotada por (CHI-LI e LANPING, 2010) para avaliar o resultado produzido por sistema baseado em *FPGA* que calcula a *FFT* sobre uma massa bidimensional de dados.

5.2. Avaliação da Variação dos Parâmetros de Configuração

O algoritmo para segmentação de sinais de vídeo (LI et al., 2009) que foi apresentado no segundo capítulo desta dissertação possui um conjunto de parâmetros de configuração que devem ser ajustados antes do início do processamento. Tal ajuste tem influência direta nos resultados produzidos e podem ser a diferença entre uma boa ou má segmentação. São parâmetros de configuração: as dimensões do filtro gaussiano utilizado para atenuação de

ruído, o tamanho do redimensionamento aplicado em cada *frame* antes de seu processamento, a profundidade temporal adotada no algoritmo de segmentação e o tamanho e tipo dos operadores morfológicos utilizados para a remoção de ruídos das máscaras binárias que são geradas como produto do processamento.

Nesta seção será avaliada a variação dos resultados produzidos pelo algoritmo de segmentação de sinais de vídeo (LI et al., 2009) a partir da variação da profundidade temporal e da variação do redimensionamento dos *frames* de vídeo a serem processados. Estes são os parâmetros de configuração que estão diretamente ligados à arquitetura desenvolvida e que podem ser modificados na implementação apresentada neste trabalho. Como indicado no quarto capítulo deste trabalho, a profundidade temporal e a dimensão dos *frames* processados podem influenciar diretamente no resultado final do algoritmo de segmentação e também podem contribuir para aumentar ou diminuir o seu tempo total de execução. Aqui não são avaliadas a variação das dimensões do filtro gaussiano e a variação dos tamanhos e tipos dos operadores morfológicos utilizados para remoção de ruídos, pois estes parâmetros estão diretamente ligados às etapas de pré-processamento e pós-processamento que são executadas em software.

Quanto maiores forem a profundidade temporal e o tamanho dos *frames* processados, maior será o tempo total de processamento necessário para a execução do algoritmo de segmentação. Por outro lado, escolher elevados valores de profundidade temporal e de dimensão dos *frames* nem sempre pode gerar o melhor resultado de segmentação, pois o nível de detalhes captados com tais valores pode contribuir para a geração de muitas regiões identificadas como *foreground*, mesmo sendo de *background*, o que pode degradar o resultado visual produzido pela segmentação.

A avaliação da variação dos parâmetros anteriormente citados foi executada antes da construção da arquitetura para segmentação de vídeos que foi apresentada no quarto capítulo deste texto. Isso permitiu identificar qual seria a quantidade de *frames* que necessariamente deveriam ser armazenados na unidade de memória do segundo passo de processamento, ou seja, a profundidade temporal processada pela arquitetura baseada em *FPGA*. Além disso, tal avaliação permitiu a definição dos tamanhos dos frames que seriam processados no *FPGA* e, consequentemente, a dimensão da *FFT/IFFT* executada sobre os dados de entrada do sistema.

O impacto nos resultados produzidos pelo algoritmo de segmentação de vídeos a partir da variação da profundidade temporal e do redimensionamento dos frames de entrada é avaliado seguindo a mesma abordagem adotada por (KRYJAC et al., 2012) e que foi descrita na primeira seção deste capítulo.

A referência para a avaliação da variação dos parâmetros do algoritmo de segmentação foi construída a partir de uma sequência de vídeo capturada com uma câmera costumeiramente utilizada para monitoramento de tráfego de carros e pedestres. Os *frames* do vídeo foram redimensionados do tamanho inicial igual a $480x704$ *pixels* para o tamanho $240x352$ *pixels* e a profundidade temporal foi ajustada para ter valor igual a 6. Estes são os parâmetros de configuração utilizados por (LI et al., 2009) que levaram o algoritmo a produzir os melhores resultados. O vídeo foi então segmentado com esses valores de parâmetros e foi gerada uma sequência de máscaras binárias que possui a classificação de cada *pixel* em cada *frame* do vídeo utilizado como entrada para o algoritmo. Essas máscaras, juntamente àquelas produzidas a partir da variação da profundidade temporal e da dimensão dos *frames* a serem processados, foram então utilizadas para calcular a *Sensitividade*, a *Precisão*, a *Medida F* e a *Similaridade* dos resultados produzidos. Isso permitiu que a avaliação dos resultados da variação deixasse de ser feita apenas de forma visual, subjetiva, e passasse a ser feita de forma numérica e objetiva.

A profundidade temporal foi variada de 3 *frames* a 8 *frames* nos testes realizados, pois este são valores de profundidade temporal que não tornam muito lenta a execução do algoritmo de segmentação. Os *frames* de entrada foram redimensionados para os seguintes tamanhos: $32x32$, $64x64$, $128x128$, $256x256$ e $512x512$. Na **Tabela 4** são apresentados os valores médios da *Medida F* que foram calculados a partir de cada sequência de máscaras binárias gerada pela variação dos parâmetros de configuração do algoritmo.

Tabela 4 - Avaliação da medida F

Medida F						
Dimensão dos *frames*	Profundidade temporal					
	3	4	5	6	7	8
32x32	63,29%	64,57%	65,31%	65,68%	65,79%	65,79%
64x64	77,01%	78,27%	79,11%	79,53%	79,58%	79,38%
128x128	82,12%	85,05%	87,08%	88,28%	88,46%	88,18%
240x352	88,21%	91,99%	95,41%	**100,00%**	95,93%	93,64%
256x256	86,59%	90,09%	92,79%	94,40%	93,83%	92,60%
512x512	87,66%	89,28%	89,85%	89,74%	88,61%	87,11%

Deve ser destacado que na **Tabela 4** existe uma linha que apresenta os valores da *Medida F* que foram calculados a partir da variação da profundidade temporal e da

manutenção do tamanho dos *frames* igual $240x352$ *pixels*. Além disso, existe uma coluna onde a profundidade temporal foi mantida com valor 6 e foi variada a dimensão dos frames nas sequências de vídeo segmentadas. Essa linha e essa coluna são utilizadas como referência na avaliação dos resultados produzidos a partir da varação dos parâmetros de configuração do algoritmo de segmentação, pois estes foram os valores utilizados em (LI et al., 2009) para a execução dos testes do algoritmo de segmentação.

Podemos notar que a variação da *Sensitividade* e da *Precisão*, que é uma consequência direta, como não poderia deixar de ser, da variação da profundidade temporal e da dimensão dos *frames* processados, é mais acentuada quando a dimensão dos *frames* é variada. Quando mantemos a profundidade temporal fixa é possível notar que os valores percentuais da *Medida F* sofrem uma grande variação a partir da mudança da dimensão dos *frames*. Variação essa que é bem menor quando é mantida a dimensão do *frame* e o valor da profundidade temporal é variado. Isso mostra que para o cálculo da *Medida F* a variação da profundidade temporal não é tão relevante quanto à variação da dimensão dos *frames*.

Tabela 5 - Valores de similaridade

Similaridade						
Dimensão dos *frames*	Profundidade temporal					
	3	4	5	6	7	8
32x32	46,70%	48,05%	48,86%	49,26%	49,39%	49,41%
64x64	62,87%	64,58%	65,72%	66,28%	66,37%	66,12%
128x128	69,89%	74,18%	77,29%	79,18%	79,51%	79,10%
240x352	79,06%	85,25%	91,26%	**100,00%**	92,29%	88,24%
256x256	76,52%	82,07%	86,63%	89,47%	88,51%	86,42%
512x512	78,30%	80,99%	82,06%	81,95%	80,18%	77,84%

Na **Tabela 5** é apresentado o valor médio da *Similaridade* obtido a partir da sequência de máscaras binárias obtidas com a variação dos parâmetros de configuração do algoritmo de segmentação. Mais uma vez podemos notar que a variação da profundidade temporal não é tão relevante para a variação do valor da *Similaridade* quanto é relevante à variação da dimensão do *frame* a ser processado.

Os resultados observados na **Tabela 4** e **Tabela 5** foram decisivos para que a profundidade temporal da arquitetura baseada em *FPGA* para segmentação de vídeos fosse mantida constante e com valor igual a 4, pois esse valor permite que poucos dados fiquem

armazenados na unidade de memória do segundo passo de processamento, o que é importante, pois o *FPGA* utilizado para a síntese de todo o projeto, o *Stratix IV* (EP4SE530H35C2) (ALTERA, 2010b), apresenta uma quantidade limitada de memória interna. Ao mesmo tempo, ao adotar a profundidade temporal igual a 4, é possível notar que a *Similaridade* não é tão afetada por tal adoção. Por outro lado, os resultados de *Medida F* e *Similaridade* indicavam que era importante deixar a arquitetura para segmentação de vídeos pronta para executar o algoritmo sobre diferentes tamanhos de *frames*. Por isso, foi adotado que existiria um parâmetro de síntese chamado *matrix_size* que indicaria o tamanho dos *frames* que seriam processados no *FPGA*.

A avaliação da variação da profundidade temporal e do redimensionamento dos *frames* de vídeo apresentada nesta seção mostra o quão importante pode ser esse tipo de avaliação para a construção de sistemas baseados em *FPGA*. Através dela foi possível definir a profundidade temporal do algoritmo executado no *FPGA* e a forma como seria indicado o tamanho dos *frames* a serem processados. Esses são dois dos principais parâmetros de configuração da arquitetura para segmentação de vídeos apresentada no quarto capítulo desta dissertação.

5.3. Avaliação dos módulos para cálculo da FFT/IFFT

Nessa seção os módulos responsáveis por calcular a *FFT/IFFT*, que foram desenvolvidos e integrados ao projeto da arquitetura para segmentação de vídeos, serão avaliados levando em consideração dois aspectos principais: o total de recursos de *FPGA* gastos em sua implementação e a precisão operacional alcançada por cada um desses módulos. Como explicado no capítulo anterior deste texto, a construção de cada módulo responsável por calcular uma *FFT/IFFT* de N pontos foi feita através da junção de dois ou mais módulos que calculam uma *FFT/IFFT* sobre n pontos, com $N > n$, por isso, nesta análise serão considerados módulos capazes de calcular uma *FFT/IFFT* a partir de N pontos, com $N = 2, 4, 8, 16\ e\ 128$.

Na avaliação da quantidade de recursos de *FPGA* gastos com cada módulo *FFT/IFFT* foi utilizado o *FPGA Stratix IV* (EP4SE530H35C2) (ALTERA, 2010b). Na **Tabela 6** são apresentados os valores mais significativos retirados dos relatórios de síntese gerados pelo software *Quartus II* (ALTERA, 2015b). Para prover um parâmetro de comparação, em tal tabela foi adicionada a última coluna, que traz o resultado de síntese do

IP-Core da Altera que calcula a *FFT/IFFT* sobre um conjunto de 128 pontos (ALTERA, 2015a). Além disso, são apresentadas as quantidades de somadores e multiplicadores de números de ponto flutuante que foram utilizados em cada *FFT/IFFT* implementada neste projeto. Dessa forma é possível entender a quantidade total de *DSP* consumida por cada *FFT/IFFT*.

Tabela 6 - Resultado de Síntese - FFT/IFFT

Resultados de Síntese	FFT/IFFT 2	FFT/IFFT 4	FFT/IFFT 8	FFT/IFFT 16	FFT/IFFT 128	Altera FFT 128
Logic Utilization	<1%	<1%	2%	2%	5%	3%
Comb. ALUTs	916	2.053	4.784	6.019	14.366	7.327
Mem. ALUTs	0	0	640	512	2.944	192
Ded. Logic Registers	651	2.085	4.389	5.780	14.472	10.530
Total registers	651	2.085	4.389	5.780	14.742	10.530
Total bocks memory bits	0	0	0	2.048	16.384	24.576
DSP	0	0	16	16	48	24
Somador Complexo	1	2	2	4	6	--
FP_ADDER	2	4	4	8	12	--
Multiplicador Complexo	0	0	1	1	3	--
FP_ADDER	0	0	2	2	6	--
FP_MULT	0	0	4	4	12	--

Através dos resultados de síntese é possível perceber que a quantidade de recursos utilizados para a implementação da *FFT/IFFT* de 128 pontos é relativamente maior do que a quantidade dispensada para a *FFT/IFFT* de 128 pontos da Altera. Tal diferença é justificada a partir da forma de representação dos números adotada em cada módulo. Os números operados estão em formato de ponto flutuante obedecendo ao padrão IEEE-754 (IEEE, 1985) em cada módulo *FFT/IFFT* que foi desenvolvido. No *IP-Core* da Altera (ALTERA, 2015a) os números devem ser convertidos para uma representação própria de ponto fixo e operados internamente seguindo tal representação. A diferença de *DSP* utilizado em cada módulo para o cálculo da *FFT/IFFT* de 128 pontos também é justificada pelo formato dos números que são operados, uma vez que, operações com números em formato de ponto flutuante tendem a ser mais complexas do que as realizadas com números inteiros ou de ponto fixo.

A diferença de representação dos números que são operados em cada um dos módulos citados anteriormente é a principal justificativa para a não utilização do *IP-Core* da Altera para o cálculo da *FFT/IFFT* no projeto da arquitetura de segmentação de vídeo apresentada nesta dissertação. Além disso, tal *IP-Core* possui um complexo protocolo de operação o que levaria a necessidade de construção de unidades de controle complexas para controlar sua operação. Uma análise da precisão dos resultados gerados por cada *FFT/IFFT* pode contribuir

para entender melhor o impacto que a diferença de representação dos números pode ter no resultado final que é produzido pelo sistema para segmentação de sinais de vídeo. Tal análise é feita utilizando a Relação Sinal-Ruído *(SNR)* que foi apresentada na primeira seção deste capítulo.

A **Tabela 7** mostra a *SNR* alcançada a partir dos resultados produzidos por cada *FFT/IFFT* em hardware implementada para integrar a arquitetura de segmentação de sinais de vídeo abordada nesta dissertação. Foram fornecidas para uma aplicação em software e para cada módulo de hardware dois tipos de entradas. No primeiro tipo os dados são aleatórios e sem limitação alguma de seus valores, ou seja, podem assumir valores entre $-\infty$ e $+\infty$. Já no segundo tipo, as entradas são limitadas entre 0 e 255, que são os valores que podem ser assumidos por cada *pixel* dos *frames* de vídeo que serão processados. Para o teste de cada *FFT/IFFT* foram geradas 50.000 entradas de cada um dos tipos anteriormente citados.

Tabela 7 - Avaliação da precisão - FFT/IFFT

Tipo de Dado	FFT/IFFT 2	FFT/IFFT 4	FFT/IFFT 8	FFT/IFFT 16	FFT/IFFT 128
Aleatório	145,50 (dB)	141,98 (dB)	138,56 (dB)	135,86 (dB)	130,77 (dB)
Frame	∞	∞	151,24 (dB)	147,51 (dB)	139,35 (dB)

Através dos dados da **Tabela 7** podemos notar que os resultados produzidos pela *FFT/IFFT* de 2 e de 4 pontos geram uma *SNR* infinita quando as entradas tem seus valores limitados entre 0 e 255. Além disso, todos os módulos que calculam a *FFT/IFFT* apresentam um alto valor de *SNR* independentemente do tipo dos dados de entrada. Isso mostra o elevado grau de precisão apresentado por cada *FFT/IFFT* quando seus resultados são comparados com aqueles gerados por aplicações de software que executam em qualquer computador de propósito geral. Para melhor interpretar esses valores de *SNR* é importante avaliar os valores de *SRN* produzidos a partir dos resultados gerados através do uso do IP-Core da Altera para o cálculo da *FFT/IFFT* de 128 pontos.

Os resultados dos testes para avaliar a precisão do *IP-Core* da Altera que calcula a *FFT* de 128 pontos são apresentados na **Tabela 8**. Os dados de entrada deste *IP-Core* foram limitados a valores inteiros entre 0 e 255 pois o módulo da Altera recebe como entrada apenas valores inteiros ou codificados em sua representação própria de ponto fixo e esse intervalo de valores é o que pode ser assumido por cada *pixel* em um *frame* de vídeo.

Tabela 8 - Avaliação de precisão - IP-core da Altera

Tipo de Dado	FFT 128 Pontos da Altera
Dado	41,06 (dB)
Dado x 512	92,00 (dB)
Dado x 1024	97,82 (dB)

Nas duas linhas inferiores da **Tabela 8** são apresentados os valores de *SNR* obtidos quando os dados de entrada são multiplicados por 512 e 1024, respectivamente. Isso foi feito para observar como a *SNR* pode ser elevada a partir de uma simples multiplicação dos dados que são fornecidos ao *IP-Core* da Altera (ALTERA, 2015a).

Podemos notar que mesmo com a multiplicação dos dados de entrada por um fator que é uma potência de 2, o que é feito em números inteiros através de um simples deslocamento para a esquerda, o *IP-Core da* Altera não é tão preciso do que o implementado neste trabalho para executar a *FFT* sobre 128 pontos. Além disso, como a arquitetura para segmentação de vídeos é composta por vários passos de processamento e cada passo possui até quatro elementos para a computação de uma *FFT/IFFT*, tal diferença de precisão pode se propagar através dos passos o que pode levar a serem gerados resultados que muito diferem daqueles apresentados por uma referência em software. Portanto, todo contexto apresentado nesta seção foi motivador e justificador para a implementação e utilização de elementos de processamento próprios para a computação da *FFT/IFFT* na arquitetura abordada nesta dissertação, pois ter uma elevada precisão leva à geração de resultados mais próximos àqueles gerados em softwares que são executados em máquinas de propósito geral.

5.4. Segmentação de vídeos com a arquitetura baseada em *CPU* e *FPGA*

A plataforma *ProcStar IV* (GIDEL, 2016) vista na **Figura 37** foi utilizada para o desenvolvimento da arquitetura para segmentação de sinais de vídeo apresentada no quarto capítulo deste trabalho. Essa plataforma é equipada com quatro *FPGAs Stratix IV* (EP4SE530H35C2) (ALTERA, 2010b) aos quais está acoplado um módulo de memória do tipo *DDR2* com capacidade de armazenamento igual a $512MB$. Além disso, cada *FPGA* possui dois slots de memória do tipo *DDR2* aos quais podem ser conectadas memórias com até 4GB de capacidade. Na síntese e nos testes apresentados nesta seção foi utilizado apenas um *FPGA* da plataforma com dois módulos de memória *DDR2* com capacidade de 4GB a ele

conectado. Para a geração de componentes de integração na placa foi utilizada a ferramenta *ProcWizard* (GIDEL, 2016). A plataforma foi conectada através do barramento *PCI-Express* a um servidor *Supermicro* e uma aplicação foi desenvolvida em linguagem C++ para executar parte do algoritmo de segmentação e promover o envio e a recepção dos dados para o processamento no *FPGA*.

Figura 37 - Plataforma ProcStar IV

Começamos nossa avaliação da arquitetura para segmentação de vídeo apresentada no quarto capítulo dessa dissertação através do relatório de síntese produzido pelo software *Quartus II* (ALTERA, 20015b). A **Tabela 9** apresenta o relatório de síntese que foi gerado a partir da atribuição do valor 128 ao parâmetro *matrix_size*. Além disso, nessa síntese foram incorporados os componentes de integração gerados pela ferramenta *ProcWizard* (GIDEL, 2016).

Tabela 9 - Resultado de síntese da arquitetura para segmentação de vídeos

Resultados de Síntese	Arq. Segmentação	Total (EP4SE530H35C2)
Logic Utilization	97%	--
Comb. ALUTs	256.259 (60%)	424.960
Mem. ALUTs	39.320 (19%)	212.480
Ded. Logic Registers	304.240 (72%)	424.960
Total registers	310.342	--
Total pins	726 (98%)	744
Total bocks memory bits	11.605.008 (55%)	21.233.664
DSP	768 (75%)	1.024

O resultado de síntese aponta que o total de registradores foi o recurso mais consumido na implementação da arquitetura para segmentação de vídeos. Neste caso, não devemos levar em consideração a quantidade de pinos de *FPGA* consumidos, pois vários

destes estão relacionados aos módulos de integração na placa que foram gerados através do *ProcWizard* (GIDEL, 2016) e não é permitido ao desenvolvedor limitar a quantidade de pinos utilizados. Deve ser destacado que o parâmetro de síntese *matrix_size* teve seu valor alterado para 256, no entanto não foi possível obter o produto final da síntese, pois a quantidade de memória e registradores disponíveis no *FPGA* (EP4SE530H35C2) não era suficiente. Portanto, para construir uma arquitetura capaz de processar sinais de vídeo cujos *frames* após o redimensionamento devem ser maiores que $128x128$ *pixels* é necessária à escolha de outro *FPGA* para a implementação de tal arquitetura.

Após os resultados de síntese passamos a discutir sobre a velocidade operacional da arquitetura para segmentação de vídeos que foi apresentada nesta dissertação. Na análise da velocidade é importante capturar o tempo gasto no envio dos dados para o *FPGA*, o tempo de processamento e o tempo gasto na recepção dos dados vindos do *FPGA*, pois isto permite que seja mesurado o tempo que realmente é gasto a partir do momento que o fluxo de execução de um software é desviado para um elemento de processamento específico, neste caso o *FPGA*, até o momento que o fluxo de execução é retornado para o software. Assim, a taxa de processamento pode ser calculada de forma mais realista do que seria feito se fosse considerado apenas o tempo de processamento no *FPGA*.

Na **Tabela 10** e **Tabela 11** estão detalhados os tempos relacionados às etapas do algoritmo de segmentação que são executadas no *FPGA*. Devemos lembrar que no *FPGA* é executada a *FFT 3-D*, o cálculo do espectro de fase de Fourier e o cálculo da *IFFT 2-D* da forma explicada no quarto capítulo deste texto. Ao software ficam reservadas as tarefas de pré-processamento e pós-processamento no qual é gerada uma máscara binária, que deve ser utilizada na segmentação, relacionada a cada *frame* de entrada.

Tabela 10 - Velocidade operacional da arquitetura para segmentação de vídeo

	10 Frames		100 Frames		1.000 Frames	
	Média	Desvio	Média	Desvio	Média	Desvio
Tempo de Envio (seg)	0,000312	0,002195003	0,029796	0,005906065	0,25350056	0,008406764
Tempo de Process. (seg)	0	0	0,001716	0,00490567	0,047424	0,003072364
Tempo de Recepção (seg)	0,000156	0,00156	0,0234	0,007839295	0,23571615	0,011283284
Taxa de Process. (fps)	infefinido		58.275,0583		21.086,3698	
Tempo Total (seg)	0,0005		0,0549		0,5366	
Taxa Total (fps)	21.367,5214		1.821,0956		1.863,4442	

Tabela 11 - Velocidade operacional da arquitetura para segmentação de vídeo

	2.000 Frames		5.000 Frames		8.000 Frames	
	Média	Desvio	Média	Desvio	Média	Desvio
Tempo de Envio (seg)	0,520729	0,016999525	1,28123	0,038382456	2,03673994	0,046738176
Tempo de Process. (seg)	0,09438	0,003417069	0,23649615	0,005747992	0,379861	0,007478211
Tempo de Recepção (seg)	0,461137	0,02560175	1,221794	0,08052387	1,812879	0,030182405
Taxa de Process. (fps)	21.190,9303		21.141,9932		21.060,3352	
Tempo Total (seg)	1,0762		2,7395		4,2295	
Taxa Total (fps)	1.858,3112		1.825,1372		1.891,4855	

Para geração dos dados da **Tabela 10** e **Tabela 11** foram utilizadas sequências de vídeo com 10, 100, 1000, 2000, 5000 e 8000 *frames* cada e foram executados por 100 vezes o envio, o processamento e a recepção dos dados relacionados a cada sequência de vídeo, por fim foram então calculadas a média e o desvio padrão de cada um desses tempos e seus valores foram organizados em tais tabelas. A aplicação em software que executa esta tarefa foi desenvolvida em linguagem *C++* e utiliza a biblioteca *OpenCV* (INTEL, 2014) para uma rápida abertura do vídeo e envio e recepção dos sinais processados no *FPGA*. A frequência operacional do *FPGA* ficou configurada em 100 MHz durante todas as etapas de medição dos tempos de processamento, pois esta foi a frequência operacional que deixou estável a arquitetura que foi desenvolvida.

Uma análise da **Tabela 10** e da **Tabela 11** nos permite observar a grande taxa de processamento alcançada pela arquitetura de segmentação de vídeos. Na verdade, uma taxa total de processamento de aproximadamente 1.800 *frames* por segundo, que é a conseguida com o uso da arquitetura apresentada no quarto capítulo deste trabalho e está disposta na sexta linha da **Tabela 10** e da **Tabela 11**, é mais do que suficiente para que o processador baseado em *FPGA* seja utilizado como uma tecnologia eficiente para o processamento de sinais de vídeos em tempo real. Além disso, a maior parte do tempo total de processamento é gasto para o envio e a recepção dos dados que são manipulados no *FPGA* e quando consideramos apenas a taxa de processamento podemos notar que é possível processar mais de 21.000 *frames* por segundo e isso contribui ainda mais para mostrar a que a arquitetura para segmentação de vídeos é capaz de processar rapidamente as partes do algoritmo que lhe cabem mesmo com uma frequência operacional relativamente baixa de 100 MHz.

Na **Tabela 10** e na **Tabela 11** não devemos levar em consideração a taxa de processamento para o caso onde são processados 10 e 100 *frames*, pois essa quantidade de dados foi processada muito rapidamente no *FPGA* e na maioria das vezes a aplicação em software não conseguiu medir corretamente o tempo gasto no processamento em *FPGA*, por

isso não temos definidas a taxa de processamento e a taxa total de processamento para o caso onde são processados apenas 10 *frames* e não devemos considerar a taxa de processamento superior a 41.000 *frames* por segundo produzida no caso onde são processados 100 *frames*.

Os resultados discutidos até agora mostram que a arquitetura para segmentação de sequências de vídeo que foi desenvolvida neste trabalho é capaz de operar rapidamente uma grande quantidade de *frames* por segundo. Para uma melhor avaliação da velocidade operacional apresentada pelo sistema desenvolvido, foi implementada a mesma sequência de processamento para ser executada em *GPU* e foram comparadas as taxas de processamento produzidas pelo sistema baseado em *CPU* e *FPGA* e pelo sistema baseado em *CPU* e *GPU*.

5.5. Segmentação de vídeos com a arquitetura baseada em *CPU* e *GPU*

Nesta comparação o sistema baseado em *CPU* e *GPU* foi desenvolvido utilizando um computador de propósito geral que possui um processador Intel i5 e 16 GB de memória principal do tipo *DDR3*. Nesta máquina foi conectada, através de um barramento PCI-Express, uma *GPU GTX 760* com memória de 2GB *GDDR5* que é dedicada para processamento de vídeo. O computador possui sistema operacional Windows 7 e a aplicação que envia e recebe os dados que são processados na *GPU* foi implementada em C++ utilizando a biblioteca *CuFFT* (NVIDIA, 2014) e a biblioteca *OpenCV* (INTEL, 2014) para a manipulação dos vídeos processados.

A aplicação que envia dados para o processamento em *GPU* foi executada com vídeos que possuem 10, 100, 1000, 2000, 5000 e 8000 frames da mesma forma como foi feito nos testes para avaliar a velocidade operacional do sistema misto formado por *CPU* e *FPGA*. Além disso, os *frames* são fixados em tamanho igual a $128x128$ *pixels* e a profundidade temporal foi ajustada para ser igual a 4 *frames*. Os tempos de envio, processamento e recepção dos dados e o total gasto para a execução da aplicação, assim como as taxas de processamento e total, que foram medidos para cada execução, são apresentados na **Tabela 12**.

Tabela 12 - Velocidade de Operação para o sistema formado por CPU e GPU

Tempos (seg)	*10 Frames*	*100 Frames*	*1.000 Frames*	*2.000 Frames*	*5.000 Frames*	*8.000 Frames*
Tempo de Envio (seg)	0	0,005	0,045	0,189	0,293	0,464
Tempo de Process. (seg)	0,765	0,817	1,544	2,447	5,1	7,753
Tempo de Recepção (seg)	0	0,006	0,035	0,063	0,423	0,594
Tempo Total (seg)	0,765	0,836	1,718	2,948	6,391	9,682
Taxa Process. (fps)	13,072	122,399	647,668	817,327	980,392	1.031,86
Taxa Total (fps)	13,072	119,617	582,072	678,426	782,35	826,276

Podemos notar que a velocidade operacional foi superior que 30 frames por segundo e isso permite que o sistema formado por *CPU* e *GPU* seja utilizado para o processamento de vídeos em tempo real.

5.6. Discussão dos resultados

A comparação com o sistema misto formado por *CPU* e *FPGA* pode ser feito levando em consideração a taxa de processamento e a taxa total de processamento para os dois sistemas. Na **Tabela 13 es**tão os valores relacionados à taxa de processamento para diferentes quantidades de *frames* e na **Tabela 14** temos a taxa total de processamento obtida por cada um dos sistemas mistos.

Tabela 13 - Comparação das taxas de processamento geradas por GPU e FPGA

Dispositivo	*1.000 Frames*	*2.000 Frames*	*5.000 Frames*	*8.000 Frames*
FPGA (fps)	21.086	21.191	21.142	21.060
GPU (fps)	582	678	782	826

Tabela 14 - Comparação das taxas totais geradas por GPU e FPGA

Dispositivo	*100 Frames*	*1.000 Frames*	*2.000 Frames*	*5.000 Frames*	*8.000 Frames*
FPGA (fps)	1.821	1.863	1.858	1.825	1.891
GPU (fps)	120	582	678	782	826

Na **Tabela 13** são apresentados os tempos de apenas quatro execuções com diferentes quantidades de frames. Foram desconsiderados os casos onde foram processados 10 e 100 *frames*, pois, para tais tamanhos, a medição de tempo gasto na execução era muito imprecisa. Observando os resultados apresentados na tabela e na tabela podemos notar que a velocidade operacional alcançada pelo sistema composto por *CPU* e *FPGA* é superior, pelo

menos em uma ordem de grandeza, quando comprada ao sistema composto por *CPU* e *GPU*. Isso pode ser graficamente observado na **Figura 38** e na **Figura 39**.

Através destes gráficos podemos notar que a taxa total de processamento tende a ser constante para o sistema formado por *CPU* e *FPGA*. Por outro lado, a taxa total de processamento tende a se manter constante no sistema formado por *CPU* e *GPU* quando a quantidade de frames a serem processados aumenta. Esses resultados deixam claro que o sistema formado por *CPU* e *FPGA* possui um desempenho superior quando comparado ao sistema formado por *CPU* e *GPU*. Portanto, este é mais um resultado que valida o desenvolvimento da arquitetura para a segmentação de vídeos que é o tema desta dissertação.

Figura 38 - Gráfico comparativo da taxa de processamento

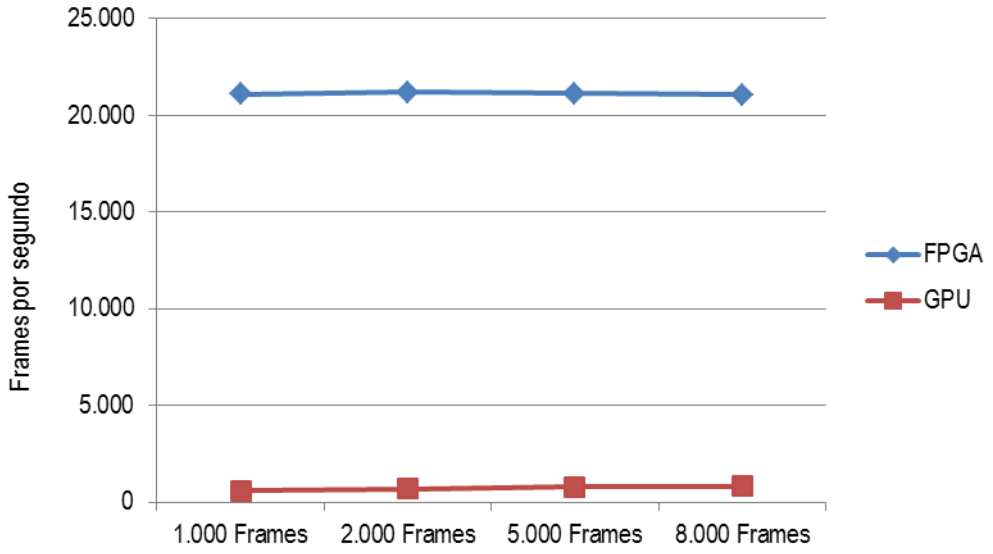
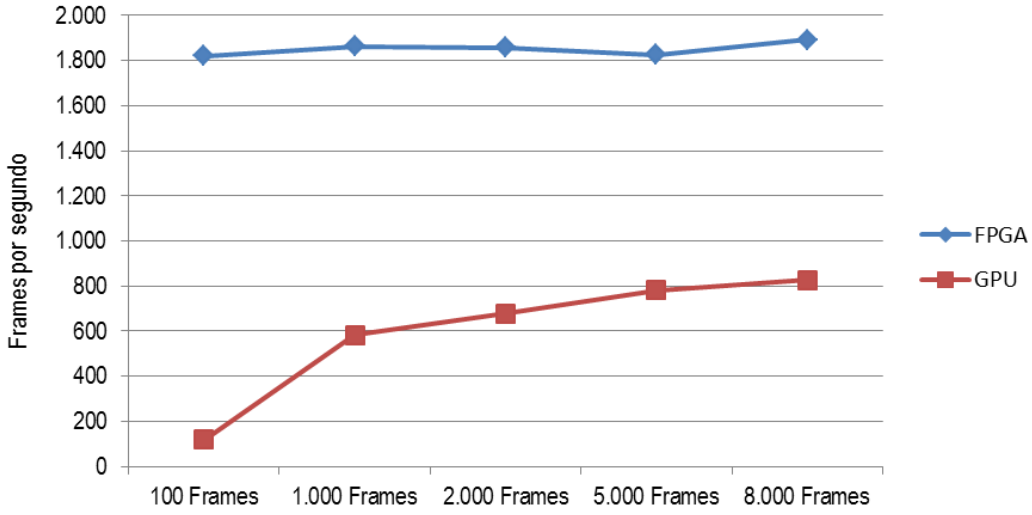


Figura 39 - Gráfico comparativo da taxa total de processamento



5.7. Avaliação dos resultados do processamento em *CPU* e *FPGA*

A avaliação da velocidade operacional deve ser acompanhada por uma avaliação da qualidade dos resultados produzidos pela arquitetura de segmentação de vídeos. Foi utilizado um vídeo com 50 *frames* para verificar a qualidade dos resultados produzidos pela arquitetura para segmentação de vídeo. Os *frames* do vídeo utilizado passaram pela etapa de pré-processamento, foram enviados para o *FPGA* e, após o seu recebimento, foram pós-processados para gerar as máscaras binárias utilizadas para a segmentação e que estão relacionadas a cada *frame* do vídeo original. O vídeo com 50 frames também foi inteiramente processado em software e as máscaras binárias resultantes foram utilizadas como referência para avaliar a qualidade do resultado produzido pelo processamento realizado no *FPGA*. Essa avaliação de qualidade foi feita seguindo a mesma abordagem adotada na primeira seção deste capítulo, ou seja, foi feita através do cálculo da sensitividade, da precisão, da *Medida F* e da *Similaridade*.

Como explicado no capítulo anterior na arquitetura para segmentação de vídeos o módulo *Angle Exp*, que calcula o espectro de fase de Fourier após o cálculo da *FFT 3-D*, foi implementado de duas formas: uma utilizando *IP-Cores* da Altera, que operam com números de ponto flutuante, para o cálculo dos valores de seno e cosseno e a outra mapeando os valores de seno e cosseno em memórias do tipo *ROM*. A qualidade dos resultados produzidos quando essas duas implementações do módulo *Angle Exp* foram utilizadas no desenvolvimento da arquitetura para segmentação de vídeos pode ser observada na **Tabela 15**. Em tal tabela temos os resultados obtidos quando o vídeo com 50 *frames* foi processado em *FPGA* e os resultados gerados por um software foi utilizado como referência. Como podemos notar, quando são mapeados em *ROM* os valores de seno e cosseno, a *Similaridade* entre as máscaras binárias produzidas em hardware e em software chega a ser 99,65% e quando foram utilizados os *IP-Cores* da Altera tal *Similaridade* chega a 100%. Esses valores de *Similaridade* mostram que os resultados produzidos com a utilização do processador de vídeos implementado em *FPGA* são praticamente ou totalmente iguais aos produzidos por uma aplicação que executa o processamento inteiramente em software. Isso comprova a qualidade dos resultados produzidos pelo *FPGA*.

Tabela 15 - Precisão operacional da arquitetura para segmentação de vídeos

Tipo de Opedor	Sensitividade	Precisão	Medida F	Similaridade
ROM	99,86%	99,79%	99,82%	99,65%
IP-Core da Altera	100,00%	100,00%	100,00%	100,00%

Os testes apresentados nesta seção comprovam a velocidade operacional e a qualidade dos resultados produzidos a partir do uso da arquitetura para segmentação de vídeo que foi apresentada no quarto capítulo deste texto e que foi o tema principal desta dissertação. A capacidade de processar dados a uma velocidade superior e 1.800 *frames* por segundo mostra que a arquitetura pode ser utilizada para o processamento em tempo real de *frames* de vídeo. Por outro lado, a *Similaridade* dos resultados produzidos em hardware com aqueles produzidos em software mostra que a arquitetura é capaz de processar corretamente os dados que são a ela enviados. Portanto, estes resultados colaboram para deixar claro o sucesso obtido com o projeto e o desenvolvimento da arquitetura para segmentação dos sinais de vídeo apresentada neste trabalho.

5.8. Conclusões

Neste capítulo foram apresentados os experimentos e resultados relevantes tanto para a definição de alguns parâmetros utilizados no projeto e desenvolvimento da arquitetura para segmentação de vídeo, quanto para comprovar a velocidade operacional e a qualidade dos resultados produzidos pelo processador de propósito específico, baseado em *FPGA*, que é o tema central desta dissertação. Além disso, foram explicadas todas as métricas utilizadas para a avaliação dos resultados gerados por cada experimento apresentado neste capítulo.

Inicialmente foi visto o impacto causado nos resultados do algoritmo de segmentação (LI et al., 2009) quando foi variada a profundidade temporal e a dimensão dos *frames* processados. Pode-se notar que a variação da profundidade temporal não provocava um impacto tão significativo quanto à variação da dimensão dos *frames* a serem processados e isso foi decisivo para que a arquitetura para segmentação de vídeo executasse o algoritmo de segmentação (LI et al., 2009) com profundidade temporal fixa e com tamanho variável de *frames* de vídeo.

Em seguida foi abordada a precisão dos resultados produzidos por cada módulo que calcula a *FFT/IFFT* que foi produzido para o projeto da arquitetura para segmentação de

vídeos e foi visto o porquê da escolha de utilizar esses módulos em detrimento ao uso do *IP-Core* da Altera que apresentavam uma precisão operacional inferior aos que foram desenvolvidos.

Finalmente foi abordado o relatório de síntese da arquitetura para segmentação de vídeos e foi comprovada a eficiência computacional e a qualidade dos resultados produzidos pelo processador de propósito específico, baseado em *FPGA*, que foi desenvolvido. Foi ainda comparado o desempenho do sistema composto por *CPU+FPGA* com o desempenho alcançado por um sistema composto por *CPU+GPU*. Tal comparação deixou clara a maior eficiência alcançada pelo sistema formado por *CPU+FPGA* quando comparado ao sistema formado por *CPU+GPU*.

6. Conclusões

Neste capítulo são abordadas as considerações finais sobre os principais tópicos discutidos nesta dissertação e são apresentadas as conclusões que podem ser obtidas a partir do trabalho aqui exposto, incluindo suas contribuições e os trabalhos futuros.

O primeiro capítulo desta dissertação apresentou de forma simples e direta os conceitos relacionados à segmentação de imagens e vídeos e abordou a forma como a transformada discreta de Fourier pode ser utilizada para o processamento de sinais digitais nas mais diferentes áreas da Engenharia. Por fim, a computação reconfigurável foi apresentada de forma sucinta e foram citados trabalhos nos quais foram construídos processadores de propósito específico, baseados em *FPGA*, para aceleração de algoritmos computacionais que demandam grande capacidade de processamento.

No segundo capítulo foram abordados os conceitos essenciais para a compreensão desta dissertação. O algoritmo de segmentação de vídeos proposto por (LI et al., 2009) foi detalhado de forma a deixar claro quais partes deste algoritmo seriam utilizadas para construir o processador de sinais de vídeo, que foi o tema central desta dissertação, e quais partes continuariam sendo executadas via software. Em seguida a Transformada discreta de Fourier foi discutida e duas de suas propriedades foram apresentadas, pois elas possuem uma ligação direta com algumas características da arquitetura para segmentação de vídeos que foi desenvolvida neste trabalho. Finalmente, foram abordados de forma simples os conceitos de *FPGA* e Computação Reconfigurável.

No terceiro capítulo foram discutidos quatro dos principais trabalhos que estão relacionados de alguma forma ao apresentado nesta dissertação. O trabalho proposto por (GENOVESE e NAPOLI, 2013) apresenta um sistema para a segmentação de sinais de vídeo, baseado em *FPGA*, onde a segmentação é feita através da subtração de um modelo de *background*. Em tal modelo cada *pixel* é representado por uma soma de distribuições gaussianas e a sua atualização se da pela modificação dos parâmetros relacionados a cada uma das distribuições. No trabalho de (KRYJAC et al., 2012) foi exposto um processador baseado em *FPGA* que implementava um algoritmo de segmentação de vídeos no qual um sistema de *clustering* era utilizado para agrupar os *pixels* que faziam parte do *background* e os *pixels* que estavam representando os objetos que se moviam nas sequências de vídeo. O artigo de (GOMEZ et al., 2012) mostra a construção de mais uma arquitetura para segmentação de vídeos e o trabalho apresentado em (CHI-LI e LANPING, 2009) aborda a construção de um sistema baseado em *FPGA* para cálculo rápido da *FFT* sobre uma massa bidimensional de dados. Todos estes trabalhos foram avaliados e através de tal avaliação foram destacadas as características principais que seriam levadas em consideração na avaliação do trabalho que foi apresentado neste texto.

No quarto capítulo foi abordada de forma minuciosa a arquitetura para segmentação de *frames* de vídeo proposta neste trabalho de mestrado. Foi discutido como tal arquitetura

pode ser formada por um conjunto de passos de processamento que possuem uma estrutura interna genérica, que pode ser adaptada para promover a computação das etapas do algoritmo de segmentação que foram implementadas em *FPGA*. Em seguida foram apresentadas as peculiaridades da estrutura de cada um dos cinco passos de processamento e tais peculiaridades foram relacionadas com as propriedades da Transformada Discreta de Fourier que foram discutidas no segundo capítulo. Ainda no quarto capítulo foram apresentados todos os componentes de hardware que foram produzidos para calcular a *FFT/IFFT* sobre um conjunto de pontos de tamanho variado e foi detalhado em quais passos de processamento é executada cada parte do algoritmo de segmentação de frames de vídeo proposto por (LI et al., 2009).

No quinto capítulo foram apresentados todos os testes relevantes para a definição de alguns parâmetros de configuração do algoritmo de segmentação. Tais parâmetros estavam diretamente relacionados ao projeto e ao desenvolvimento da arquitetura de processamento de sinais de vídeo, baseada a em *FPGA*, que foi o tema central desta dissertação, por isso foi fundamental entender como tais testes foram executados e como os seus resultados foram importantes para todo o desenvolvimento deste trabalho. Em seguida foram abordados os testes para avaliar a precisão dos módulos desenvolvidos para o calculo de cada *FFT/IFFT*. Na última seção do quinto capítulo foi apresentado o conjunto de testes que foi realizado para comprovar a eficiência computacional e a precisão dos resultados produzidos pelo processador de propósito especifico baseado em *FPGA* que foi desenvolvido.

Ainda no quinto capítulo, foi abordado um conjunto de testes que foram desenvolvidos para comparar o desempenho do sistema composto por *CPU* e *FPGA (Stratix IV, EP4SE530H35C2)* com o desempenho de um sistema composto por *CPU* e *GPU (G-Force GTX 760)* que também executa o algoritmo de segmentação proposto por (LI et al., 2009). Através da discussão de resultados, ficou claro que o sistema composto por *CPU* e *FPGA* apresenta um desempenho melhor que o sistema composto por *CPU* e *GPU*, mesmo operando com uma frequência de 100 MHz.

Uma das principais contribuições deste trabalho foi associar conceitos matemáticos relacionados às propriedades da Transformada Discreta de Fourier com a forma como o processador de propósito específico foi projetado e desenvolvido. O uso da *Simetria Hermitiana* permitiu que as unidades de memória do primeiro e do segundo passo de processamento fossem implementadas com aproximadamente 50% dos recursos do *FPGA* que seriam consumidos para tal implementação caso essa propriedade não fosse levada em consideração. Isso acarretou em uma importante economia de recursos de memória do *FPGA*.

Além disso, o uso da propriedade da *Linearidade* permitiu que os módulos que calculam a *FFT/IFFT* em cada passo de processamento fossem construídos com o menor número possível de *DSP* que é um recurso escasso na maioria dos *FPGAs* de mercado.

Outra contribuição importante deste trabalho foi apresentar o desenvolvimento de uma arquitetura complexa baseada em *FPGA* capaz de executar de forma eficiente, aproximadamente 1.800 *frames* por segundo com frequência operacional de 100 MHZ, a segmentação de *frames* de vídeo. Foi escolhido o FPGA para a implementação deste projeto porque este é um dispositivo que possui baixa frequência operacional, permite a implementação paralela do algoritmo de segmentação e tem baixo consumo de energia. Neste trabalho também foi comprovada, através de um conjunto coerente e objetivo de testes que utiliza como referência uma aplicação em software, a qualidade dos resultados produzidos a partir da utilização da arquitetura desenvolvida para segmentar sinais reais de vídeo. Os primeiros resultados deste trabalho são apresentados em (BARBOSA et. al., 2015).

6.1. Trabalhos Futuros

Futuramente é preciso integrar a arquitetura baseada em *FPGA* para segmentação de vídeo com uma aplicação em software capaz de fornecer um fluxo contínuo de *frames* de uma forma que o processamento seja realizado a partir de um *streaming* de vídeo e a arquitetura possa ser utilizada para o processamento de vídeos em tempo real. Além disso, como o sistema desenvolvido possui capacidade de processamento igual a 1.800 *frames* por segundo, será possível utiliza-lo para processar várias câmeras de vídeo ao mesmo tempo.

Poderão ser utilizadas outros tipos de *FPGA* para implementar o trabalho que foi descrito neste texto. Assim poderão ser processados *frames* de vídeo com dimensões maiores que $128x128$ *pixels* e a profundidade temporal poderá ser modificada a partir da possibilidade de maior capacidade de armazenamento interno destes outros tipos de *FPGA*.

Como a Transformada Discreta de Fourier em sua forma direta ou inversa é utilizada em diferentes tipos de algoritmos para os mais diferentes tipos de propósitos, os componentes do projeto da arquitetura para segmentação de vídeo poderão ser reorganizados ou reaproveitados para construir novas arquiteturas de processamento que utilizem a *FFT* ou a *IFFT* como principais ferramentas para computação de dados. Isso pode render a construção de novos sistemas de computação mistos baseados em *CPU* e *FPGA* para a computação veloz e eficiente de dados.

REFERÊNCIAS

AFFECTIVA. Affectiva, 2016. Disponivel em: <http://www.affectiva.com/>. Acesso em: 15 fevereiro 2016.

AKOUSHDEH, A. R.; SHAHBAHRAMI, A.; MAYBODY, B. M. High performance implementation of texture features extraction algorithms using FPGA architecture. **Journal of Real-Time Image Processing**, 9, 2012. 141-157.

ALTERA. ALM, 2009a. Disponivel em: <https://www.altera.com/products/fpga/features/stx-architecture.html>. Acesso em: 29 dez. 2015.

ALTERA. LUT, 2009b. Disponivel em: <https://www.altera.com/products/general/fpga/stratix-fpgas/stratixii/stratixii/features/architecture/st2-lut.html>. Acesso em: 29 dez. 2015.

ALTERA. DSP, 2010a. Disponivel em: <https://www.altera.com/products/fpga/features/stx-dsp-block.html>. Acesso em: 29 dez. 2015.

ALTERA. Overview StratixIV, 2010b. Disponivel em: <https://www.altera.com/products/fpga/stratix-series/stratix-iv/overview.html>. Acesso em: 4 fev. 2016.

ALTERA. **White Paper:** Accelerating High-Performance Computing With FPGAs. [S.l.]: [s.n.], 2013.

ALTERA. Stratix IV Device Handbook, Volume 4: Device Datasheet and Addendum, 2014. Disponivel em: <https://www.altera.com/en_US/pdfs/literature/hb/stratix-iv/stx4_5v4.pdf>. Acesso em: 20 mar. 2015.

ALTERA. Megafunciton, 2015a. Disponivel em: <https://www.altera.com/products/fpga/features/stx-dsp-block.html>. Acesso em: 25 jan. 2016.

ALTERA. QuartusII, 2015b. Disponivel em: <http://www.altera.com/literature/manual/intro_to_quartus2.pdf>. Acesso em: 20 jan. 2016.

BARBOSA, J. P. F. et al. A high performance hardware accelerator for dynamic texture segmentation. **Journal of Systems Architecture**, 61, n. 10, Novembro 2015. 639–645.

BENEDEK, C.; SZIRA´NYI, T. Study on color space selection for detecting cast shadows in video surveillance. **Journal of Imaging Systems and Technologies**, 17, 2007. 190–201.

BRAVO, A. L. et al. **FPGA-based video system for real time moving object detection**. IEEE 5th International Conference on Intelligent and Advanced Systems. [S.l.]: [s.n.]. 2013.

BROWN, L. G. A survey of image registration techniques. **ACM Computing Survey**, 24, 1992. 325–376.

BUTLER, D.; SRIDHARANM, S.; BOVE, V. **Real time adaptive background segmentation**. IEEE International Conference on Acoustics, Speech, and Signal Processing. [S.l.]: [s.n.]. 2003. p. 349–352.

CHAN, Y. K.; LIM, S. Y. Synthetic aperture radar signal generation. **Progress In Electromagnetics Research**, 1, 2008. 269–290.

CHI-LI, Y.; K., J.-S.; LANPING, D. **FPGA architecture for 2-D discrete Fourier transform based on 2-D decomposition for large-sized data**. IEEE Workshop on Signal Processing Systems. [S.l.]: [s.n.]. 2009. p. 121-126.

COOLEY, J.; TUKEY, J. An algorithm for the machine computation of the complex Fourier series. **Mathematics of Computation**, v. 19, n. 90, p. 297–301, 1965.

DUHAMEL, P.; VETTERLI, M. Fast Fourier Transforms: a Tutorial Review and State of the Art. **Journal of Signal Processing**, 19, 1990. 297–301.

DUTRA, B. H. T. C. Desenvolvimento de uma Plataforma com uma Arquitetura Escalável para Multiplicação de Matrizes Densas em Sistemas Reconfiguráveis de Alto Desempenho. **Dissertação de Mestrado**, Recife, 2010.

EKMAN, P. **Facial expressions. The handbook of cognition and emotion**. Sussex: John Wiley & Sons, 1999.

ELEFTHERIOU, M. . E. A. Scalable framework for 3D FFTs on the blue gene/l supercomputer: Implementation and early performance measurements. **IBM Journal of Research and Development**, 49, 2005. 457–464.

ELHBIAN, S. Y.; EL-SAYED, K. M. A. Moving object detection in spatial domain using background removal techniques—state-ofart. **Recent Patents Computation**, 1, 2008. 32–34.

FANG, B. Performance of the 3D FFT on the 6D network torus QCDOC parallel supercomputer. **Computer Physics Communications**, 176, n. 8, 2007. 531–538.

FERNANDES, B.; SARMENTO, H. FPGA implementation and testing of a 128 FFT for a MB-OFDM receiver. **Journal of Analog Integrated Circuits and Signal Process**, 70, 2012. 241-248.

FERNANDES, L. L.; CRUZ, J. C.; BARP, A. R. Modelagem Sísmica via Método das Diferenças Finitas - Caso Bacia do Amazonas. **Revista Acta Amazônica**, Manaus-AM, v. 39, p. 155-164, Junho 2009.

FRACHETTI, F. Discrete Fourier transform on multicore. **IEEE Signal Processing Magazine, Special Issue on "Signal Processing on Platforms with Multiple Cores"**, v. 26, n. 6, p. 90–102, 2009.

FRIGO, M.; JOHNSON, S. **FFTW:** An adaptive software of the FFT. Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing. [S.l.]: [s.n.]. 1998. p. 1381–1384.

GENOVESE, M.; NAPOLI, E. FPGA-based architecture for real time segmentation and donoising of HD video. **Journal of Real-Time Image Processing**, 8, 2013. 389-401.

GHANEM, B.; AHUJA, N. **Phase modelling of dynamic textures**. IEEE 11th International Conference on Computer Vision. [S.l.]: [s.n.]. 2007.

GIDEL. ProcStar IV: Product Brief, 2010. Disponivel em: <http://www.gidel.com/pdf/PROCStarIV%20Product%20Brief.pdf>. Acesso em: 20 Março 2015.

GIDEL. PROCWizard, 2010. Disponivel em: <http://www.gidel.com/procwizard.htm>. Acesso em: 04 Fevereiro 2016.

GOKHALE, M. B.; GRAHAM, P. S. **Reconfigurable Computing:** Accelerating Computations with Field-Programmable Gate Arrays. Dordretch: Springer, 2005.

GOMEZ, R. R. et al. Codebook hardware implementation on FPGA for background subtraction. **Journal of Real-Time Image Processing**, 10, 2012. 43-57.

GONZALEZ, R. C.; WOODS, R. E. **Processamento Digital de Imagens**. 3ª. ed. [S.l.]: Prentice Hall, 2010.

HARALICK, R. M.; SHANMUGAM, K.; DISTEIN, I. H. Textural features for image classification. **IEEE Trans Syst Man Cybern**, 3, 1973. 610–621.

HEMATIAN, A. et al. **Field programmable gate array system for real-time IRIS recognition**. IEEE Conference on Open Systems (ICOS). Kuala Lumpur: IEEE. 2012. p. 1 - 6.

HENNESSY, J. L.; PATTERSON, D. **Computer Architecture:** A Quantitative Approach. 4ª. ed. [S.l.]: Morgan Kaufmann, 2010.

HU, J.; DENG, J.; WU, J. Image compression based on improved FFT algorithm. **Journal of Networks**, 6, n. 7, 2011.

HUANG, S. C. An advanced motion detection algorithm with video quality analysis for video surveillance systems. **IEEE Transactions Circuits Systems and Video Technologies**, 21, n. 1, 2011. 1–14.

IEEE. **IEEE Standards for Binary Floating-Point Arithmetic**. IEEE Press. New York. 1985.

IMPULSEC. Impulse Accelerated Technologies, 2016. Disponivel em: <http://www.impulseaccelerated.com>. Acesso em: 31 janeiro 2016.

INTEL. OpenCV library reference, 2014. Disponivel em: <http://sourceforge.net/projects/opencvlibrary>. Acesso em: 29 janeiro 2016.

INTEL. IPP: Intel Integrated Performance Primitives, 2015a. Disponivel em: <http://software.intel.com/en-us/intel-ipp/>. Acesso em: 31 janeiro 2016.

INTEL. MKL: Intel Math Kernel Library, 2015b. Disponivel em: <http://software.intel.com/en-us/intel-mkl/>. Acesso em: 31 janeiro 2016.

JAIN, B.; GUPTA, M. K.; BHARTI, J. Efficient iris recognition algorithm using method of moments. **International Journal of Artificial Intelligence and Applications**, 3, n. 5, 2012.

KIM, K. et al. Realtime foreground–background segmentation using codebook model. **Journal of Real Time Imaging**, 11, n. 3, 2005. 172–185.

KRYJAC, T.; KOMORKIEWICZ, M.; GORGON, M. Real-time background generation and foreground object segmentation for high-definition colour video stream in FPGA device. **Journal of Real Time Image Processing**, 9, 2012. 61-77.

LATHI, B. P. **Sinais e Sistemas Lineares**. 2ª. ed. Porto Alegre: Bookman, 2007.

LENARD, T.; GUSTAFSSON, M.; OWALL, V. A hardware acceleration platform for digital holographic imaging. **Journal of Signal Processing System**, 3, 2008. 297-311.

LI, J.; CAI, Y.; CHEN, L. Dynamic texture segmentation using Fourier transform. **Modern Applied Science**, 3, n. 9, 2009.

LINDOSO, A.; ENTRENA, L. High performance FPGA-based image correlation. **Journal of Real Time Image Processing**, 2, 2007. 223-233.

MCFARLANE, N. J. B.; SCHOLFIELD, C. P. Segmentation and tracking of piglets in images. **Machine Vision Applied**, 8, 1995. 187–193.

MEDEIROS, V. W. C. et al. FPGA-based architecture to speed-up scientific computation in seismic aplications. **International Journal of High Performance Systems Architecture**, 4, n. 2, 2012. 65-77.

MORIMOTO, T. et al. An FPGA-based region-growing video segmentation system with boundary-scan-only LSI architecture. **IEEE APCCAS**, 2006.

NVIDIA CORP. Cuda C Programing Guide, 2014. Disponivel em: <http://docs.nvidia.com/cuda/cuda-c-programming-guide/#axzz3jU5qDHDT>. Acesso em: 20 agosto 2015.

PÜSCHEL, M. SPIRAL: Code generation for DSP transforms. **Proceedings of the IEEE**, 93, n. 2, 2005. 232–275.

RANBHOR, S. P.; MALI, S. N. Implementation of GMM for HD video segmentation using FPGA. **International Journal of Engineering Research & Technology,** 4, n. 2, 2015.

ROCHA, R. C. F. Desenvolvimento de uma Plataforma Reconfigurável para modelagem 2D, em Sísmica, Utilizando FPGA. **Dissertação de Mestrado**, Recife-PE, 2010.

SOFTBANK. **Pepper Robot**, 2014. Disponivel em: <http://www.softbank.jp/en/robot/>. Acesso em: 14 fevereiro 2016.

STAUFFER, C.; GRIMSON, W. Adaptive background mixture models for real time tracking. **IEEE Conference of Computer Vision and Pattern Recognition**, 2, 1999. 246-252.

UZUN, I. S.; AMIRA, A.; BOURIDANE, A. FPGA implementation of fast Fourier transforms for real-time signal and image processing. **IEEE Proceedings - Vision, Image, and Signal Process**, 152, n. 3, 2005.

XILINX. FFT Logicore, 2012a. Disponivel em: <http://www.xilinx.com/products/ipcenter/FFT.htm>. Acesso em: 31 jan. 2016.

XILINX. FPGA, 2012b. Disponivel em: <http://www.xilinx.com/fpga>. Acesso em: 29 dez. 2015.

YASRI, I.; HAMID, N. H.; ALI, N. B. Z. **VLSI based edge detection hardware accelerator for real time video segmentation system**. IEEE 4th International Conference on Intelligent and Advanced Systems. [S.l.]: [s.n.]. 2012.

Anexos

A seguir são apresentadas os diagramas de estados de todas as maquinas de estados finitos que foram implementadas para coordenas as operações de leitura e escrita de dados em algum tipo de elemento de armazenamento. As máquinas de estados finitos dos controladores de leitura do primeiro, segundo, quarto e quinto passo de processamento são apresentadas. Em seguida são mostrados os diagramas de estados dos controladores de escrita do primeiro, terceiro, quarto e quinto passo. Por fim, são apresentados os diagramas de estados das unidades de memória do primeiro, terceiro, quarto e quinto passo de processamento.

Figura 40 - FSM do Controlador de Leitura do Passo 1

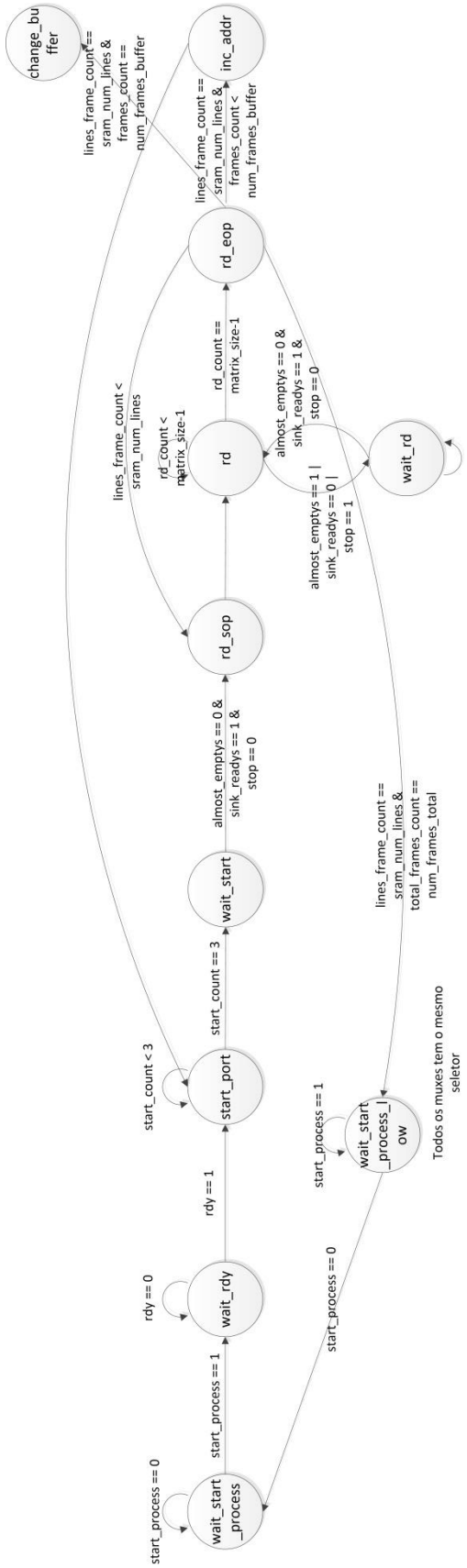


Figura 41 - FSM do Controlador de Leitura do Passo 2

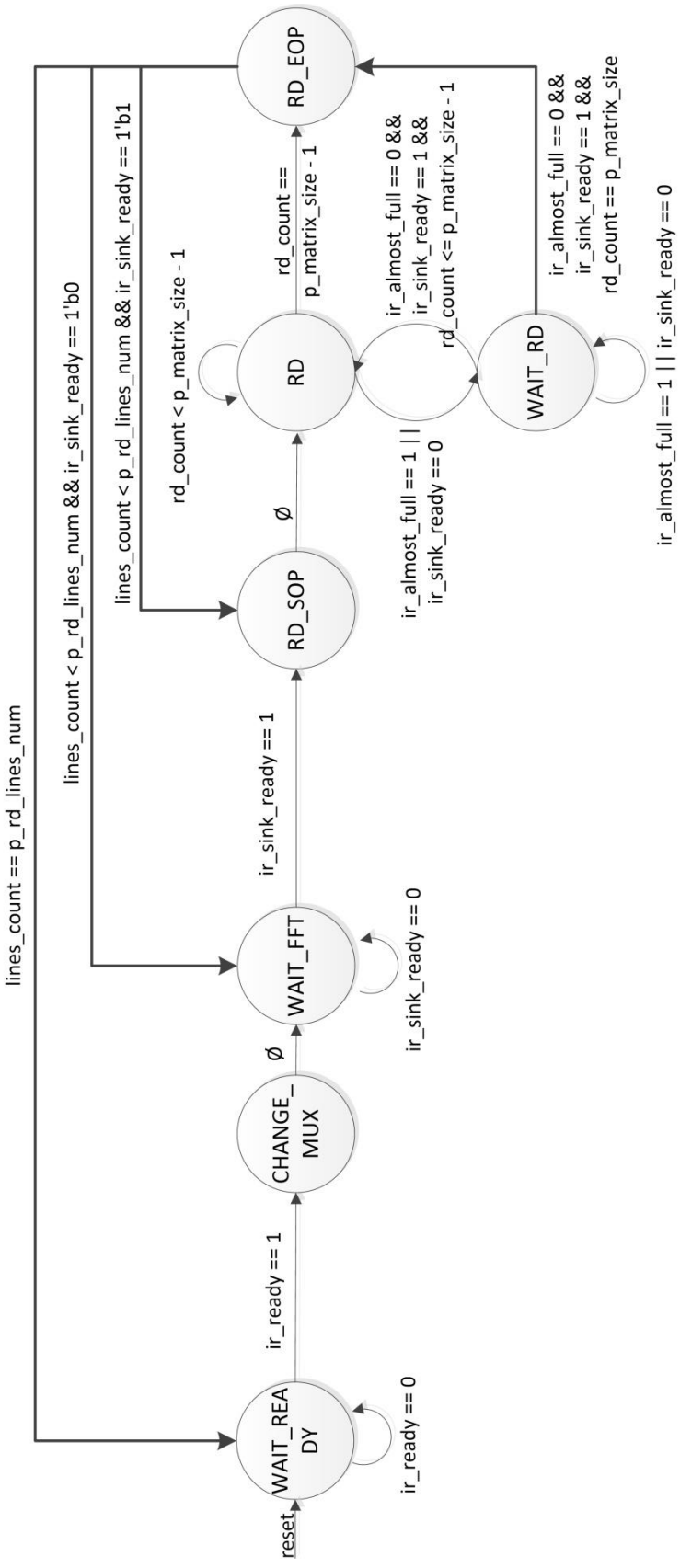


Figura 42 - FSM do Controlador de Leitura do Passo 4 e 5

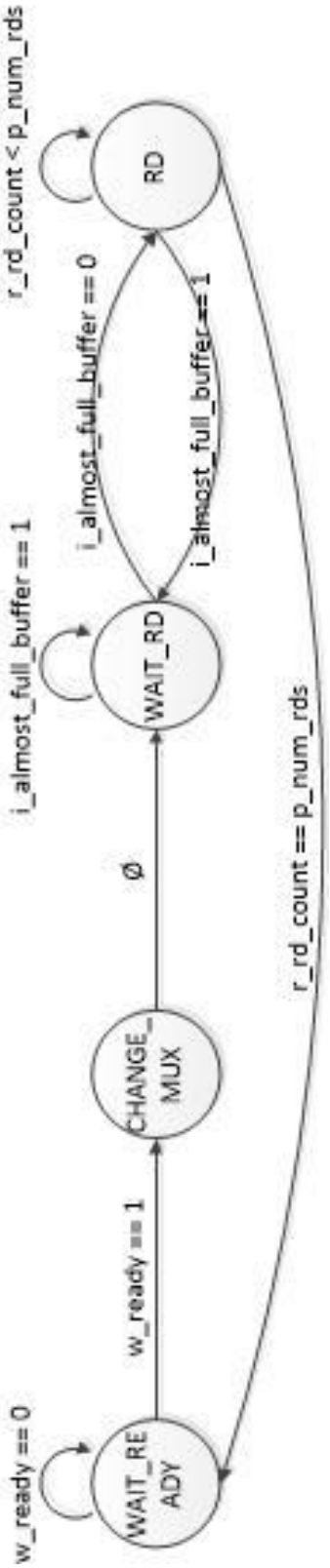


Figura 43 - FSM do Controlador de Escrita do Passo 1

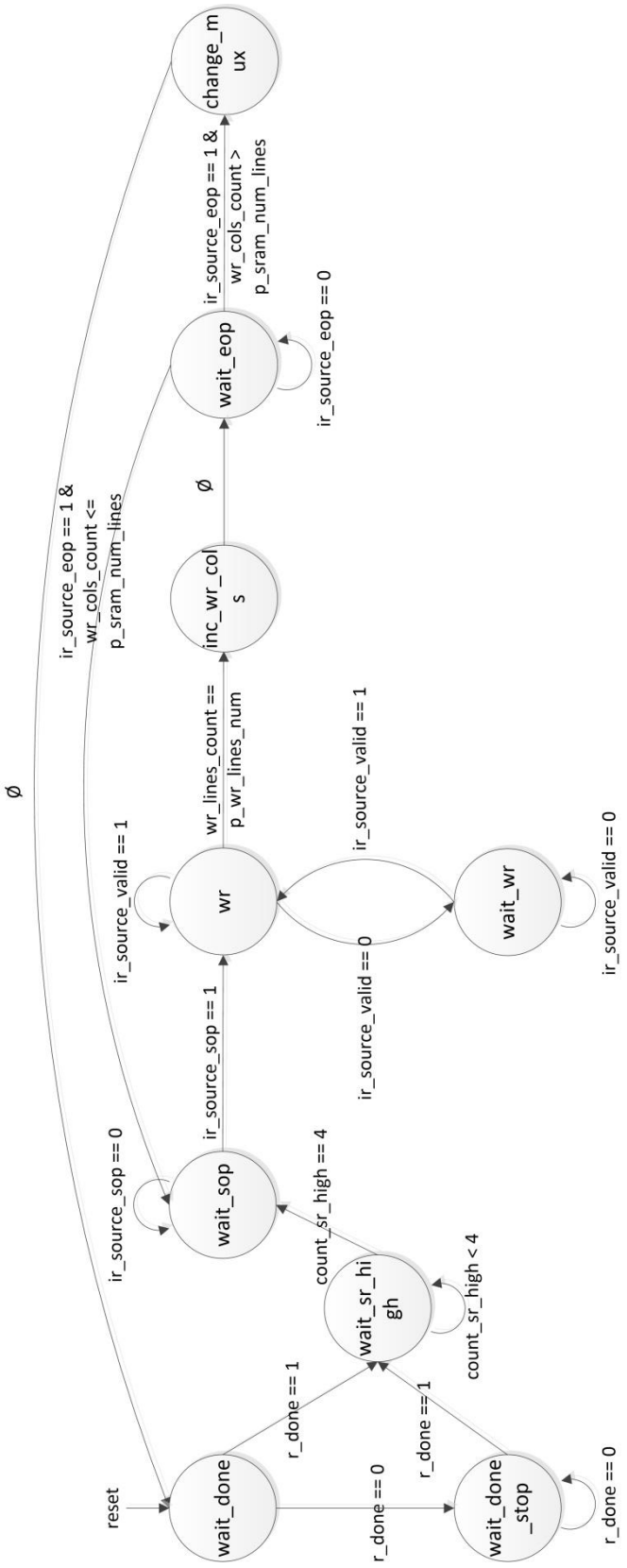


Figura 44 - FSM do Controlador de Escrita do Passo 2

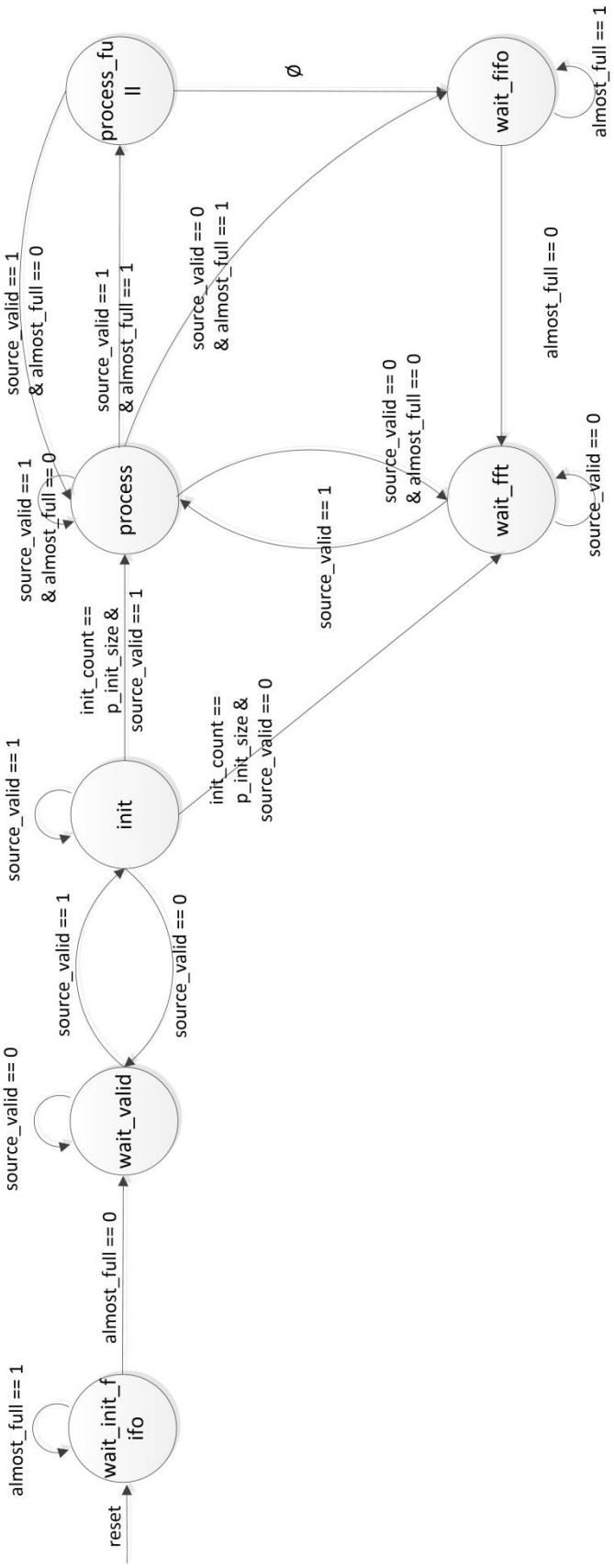


Figura 45 - FSM do Controlador de Escrita do Passo 3, 4 e 5

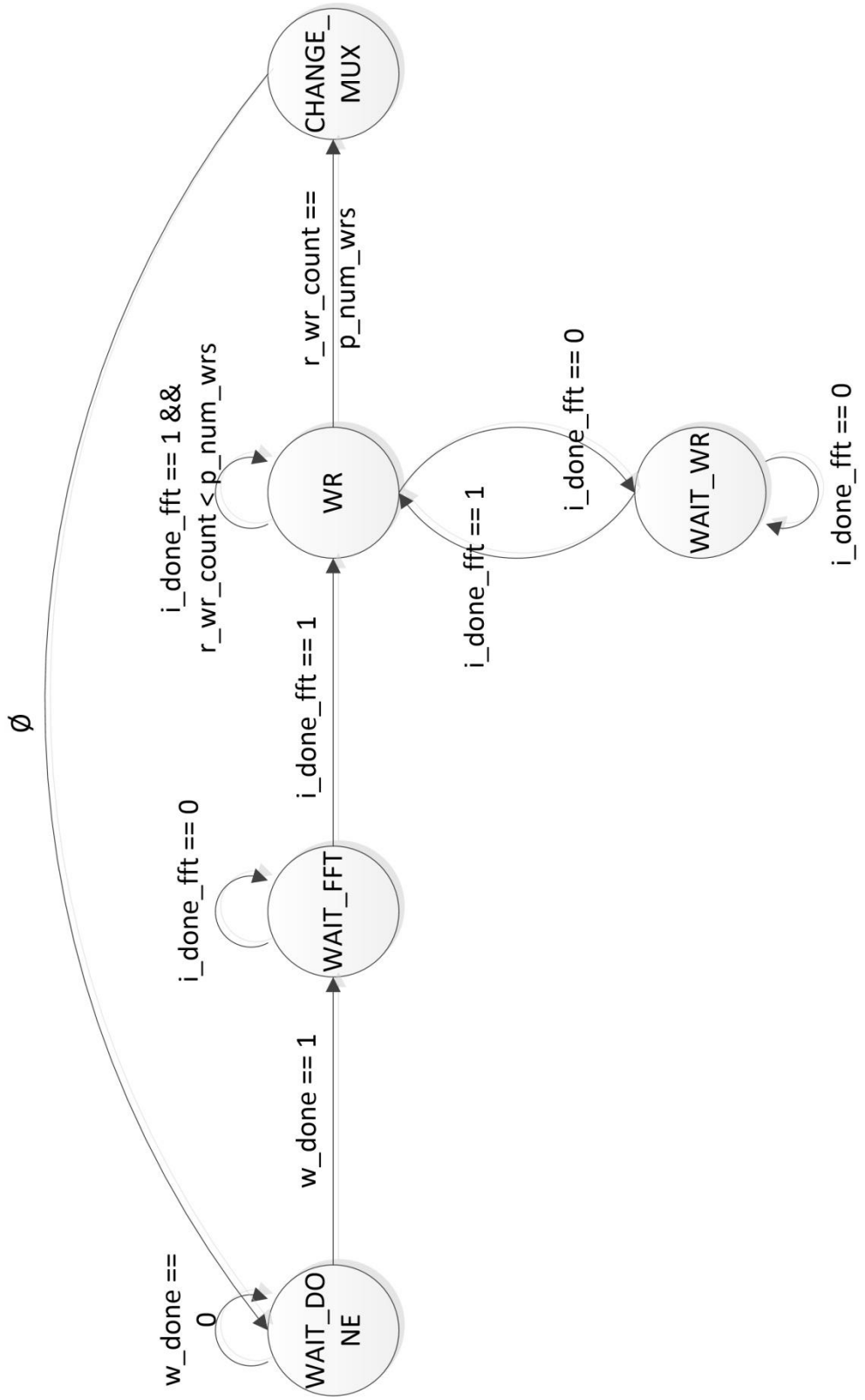


Figura 46 - FSM da Unidade de memória do Passo 1

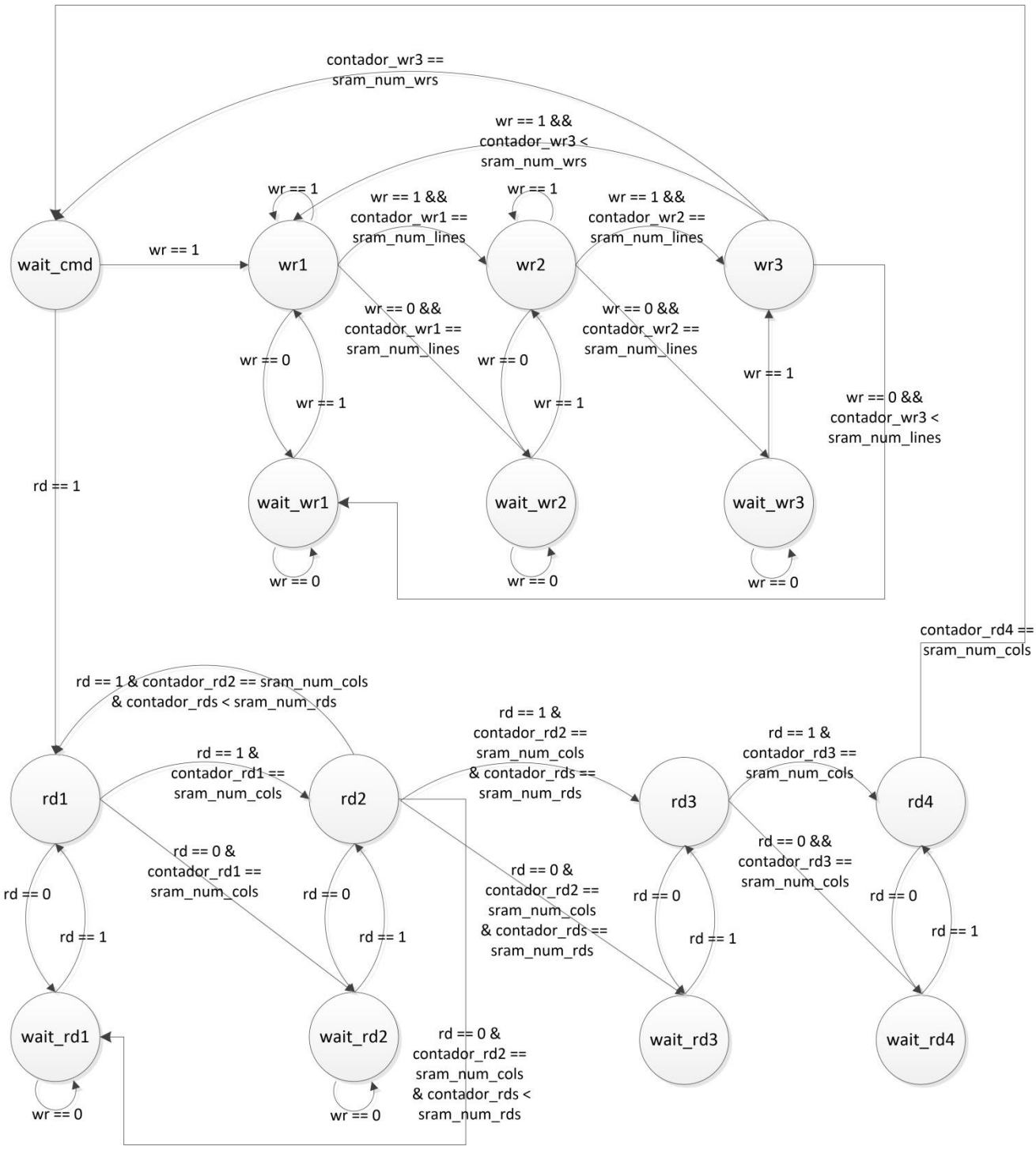


Figura 47 - FSM das Unidades de Memória do Passo 3, 4 e 5