**Filipe Sacchi da Silva**

# *Lógica programável aplicada ao processamento digital de imagens em tempo real*

Dissertação apresentada à Coordenação de Formação Científica do Centro Brasileiro de Pesquisas Físicas para obtenção do título de Mestre em Física com especialidade em Instrumentação Científica.

Dissertação defendida e aprovada em 08 de dezembro de 2014

**Composição da Banca:**

Presidente: Márcio Portes de Albuquerque – CBPF (orientador)
Herman Pessoa Lima Júnior – CBPF (coorientador)

Titular Interno: Marcelo Portes de Albuquerque – CBPF
Titular Externo: Luciano Manhães de Andrade Filho – UFJF

Centro Brasileiro de Pesquisas Físicas

Ministério da Ciência, Tecnologia e Inovação

# *Agradecimentos*

Em primeiro lugar, agradeço aos meus pais pelos exemplos de caráter e dedicação, agradeço por me mostrarem que o conhecimento é um de nossos bens mais valiosos.

Agradeço à minha querida esposa Aline pelo incentivo, compreensão e ajuda em todos os momentos de nossa vida.

Agradeço ao CBPF pela oportunidade de cursar o mestrado em instrumentação científica, e pela estrutura fornecida para o desenvolvimento deste trabalho. Agradeço aos professores da instituição que me direcionaram no estudo da física e de sua instrumentação.

Agradeço aos meus orientadores Márcio Portes de Albuquerque e Herman Pessoa Lima Júnior por acreditarem que eu seria capaz de desenvolver este trabalho. Agradeço a eles pelo grande conhecimento a que fui exposto. Agradeço pela paciência e por compreenderem que além de minhas responsabilidades acadêmicas também possuía responsabilidades profissionais.

Agradeço à *National Instruments Brasil* pelo apoio e entendimento de que a realização deste trabalho foi de suma importância para meu desenvolvimento pessoal e profissional.

Agradeço ao Técnico Fernando de Souza do LAFEX pela ajuda nas etapas iniciais deste trabalho.

Agradeço a todos os amigos, conhecidos e parentes pelos incentivos que de alguma forma contribuíram para que eu seguisse sempre em frente.

# *Resumo*

O processamento de imagens digitais é hoje uma importante ferramenta de análise quantitativa em várias áreas científicas. Entretanto, para situações que exigem características de tempo real e/ou altas velocidades, a aplicação de técnicas de processamento de imagens pode ser relativamente complexa e frequentemente custosa computacionalmente. Deste modo, o desenvolvimento em *hardware,* de certas etapas de uma cadeia de processamento de imagens, se torna uma alternativa atraente. Características como: (i) paralelismo real; (ii) alta velocidade de execução; (iii) alta densidade de circuitos, fazem dos FPGAs, dispositivos indicados para situações que exijam alta velocidade e determinismo[1] na execução dos algoritmos de processamento de imagens.

Este trabalho apresenta a caracterização de todo um ambiente de desenvolvimento em FPGA para aplicações de processamento de imagens que necessitam operação em alta velocidade e/ou tempo real. O desenvolvimento aqui realizado se baseou em trabalhos previamente propostos pelo CBPF, para a detecção em imagens sequênciais, através de algoritmos de *software*, do fenômeno *MARFE* que ocorre no interior de reatores de fusão nuclear.

Para aplicação em *hardware* dos algoritmos de processamento de imagens avaliados, foi utilizada a plataforma de desenvolvimento Altera Cyclone IV GX. Os resultados apresentados para a abordagem em *hardware,* quando comparados com aqueles obtidos pelas abordagens em *software* previamente propostas, permitem uma avaliação de todo o ambiente aqui desenvolvido. As análises por fim apresentadas levam a perspectivas de melhorias para este trabalho, que permitirão processar os algoritmos avaliados de forma determinística, e com taxas de até 67204 imagens/s.

**Palavras-Chave:** Processamento de imagens; tempo real; FPGA; PCIe; instrumentação científica; fusão nuclear.

[1] Um sistema é dito previsível no domínio lógico e temporal quando, independente de variações que ocorram em termos de *hardware* (i.e. desvios do relógio), da carga de execução e de falhas, o comportamento do sistema pode ser antecipado, antes de sua execução. O limite para esta previsibilidade é chamado de determinismo.

# *Abstract*

The processing of digital images has become an important tool for quantitative analysis in various scientific fields. However, for situations that require real-time and / or high speeds algorithms, the application of image processing techniques can be relatively complex and often computationally expensive. Thus, the hardware implementation of certain steps of the image processing chain becomes an attractive alternative. Features such as: (i) real parallelism; (ii) high speed execution; (iii) high density circuit, make the FPGA suitable devices for situations that require high speed and deterministic execution of algorithms for image processing.

This work presents the characterization of an entire development environment for FPGA image processing applications that need high-speed operation and / or real time feature. The development conducted here was based on work previously proposed by CBPF for detection in sequential images through software algorithms, the MARFE phenomenon that occurs inside nuclear fusion reactors.

For application of hardware image processing algorithms evaluated, the Cyclone IV GX development plataforma was used. The results presented for the approach in hardware, when compared with those obtained by previously proposed approaches in software, allows assessment of the entire environment developed here.

Finally, the analysis presented allows us to estimate the operating limits for these algorithms. With use of modern FPGAs, we can estimate that these deterministic algorithms can achieve rates up to 67,204 frames / s.

**Key-Words**: Image processing; real-time; FPGA; PCIe; scientific instrumentation; nuclear fusion.

# *Sumário*

# Lista de figuras

# Lista de tabelas

# Lista de abreviaturas

| | |
|---|---|
| ULA | Unidade Lógico Aritmética |
| CPU | *Central Processing Unit* |
| GPU | *Graphics Processing Unit* |
| FPGA | *Field Programmable Gate Array* |
| MARFE | *Multifaceted Asymmetric Radiation From the Edge* |
| JET | *Joint EuropeanTorus* |
| PCI | *Peripheral Component Interconnect* |
| PCIe | *Peripheral Component Interconnect Express* |
| RGB | *Red Green Blue* |
| SIMD | *Single Instruction Multiple Data* |
| SVM | *Support Vector Machine* |
| UCP | Unidade Central de Processamento |
| CBL | *Configurable Logic Blocks* |
| LUT | *Look-Up Table* |
| LVTTL | *Low Voltage Transistor-Transistor Logic* |
| LVCMOS | Low Voltage *Complementary Metal-Oxide-Semiconductor* |
| PLL | *Phase Lock Loop* |
| FIFO | *Fisrt-in First-out* |
| HDL | *Hardware Description Language* |
| VHDL | *Very High Speed Integrated Circuit Hardware Description Language* |
| JTAG | *Joint Test Action Group* |
| AGP | *Accelerated Graphics Port* |
| TL | *Transaction Layer* |
| DDL | *Data Link Layer* |
| PL | *Physical Layer* |

| | |
|---|---|
| TLP | *Transaction Layer Packet* |
| CRC | *Cyclic Redundancy Check* |
| BAR | *Base Address Register* |
| Bdf | *Block design file* |
| VISA | *Virtual Instrument Software Architecture* |
| GPIB | *General Purpose Interface Bus* |
| PXI | *PCI eXtensions for Instrumentation* |
| USB | *Universal Serial Bus* |
| ROI | *Region of Interest* |
| DMA | *Direct memory access* |
| SAv | *Average Subtraction* |
| MSB | *Most Significant Bit* |
| LabVIEW | *Laboratory Virtual Instrument Engineering Workbench* |
| HD | *Hard Disk* |

# Capítulo 1

# Introdução

## 1.1 Motivação

O processamento de imagens digitais é hoje uma importante ferramenta de análise quantitativa para a indústria, e para várias áreas de investigação científica, como por exemplo, ciência dos materiais (metalurgia, magnetismo etc.), ciências da terra (geologia), geografia, meteorologia, cartografia (fotografias aéreas e de satélites), astronomia, robótica etc. O que faz a análise de imagens uma disciplina comum a estas diferentes áreas é que as imagens são na realidade um suporte físico para troca e transporte de informações. Estas podem estar associadas a uma medida (um sinal associado a um fenômeno físico), ou podem estar associadas a um nível cognitivo (neste caso, obtenção de conhecimento). Uma imagem contém uma quantidade imensa de informações que um observador humano interpreta normalmente de um modo global e qualitativo.

Uma imagem dependerá basicamente do sensor utilizado para sua captura, não se restringindo apenas ao espectro visível, mas sim a quase toda faixa do espectro eletromagnético. Imagens digitais podem também ser obtidas de fontes não convencionais, como ultrassom, ressonância magnética e até mesmo assinaturas em poeira emitidas por objetos, como por exemplo, núcleos de cometas (AUER, 1982).

O processamento de uma imagem está relacionado diretamente ao processamento da informação nela presente. Em ambientes na qual está técnica é usada como instrumento científico, ele está na base do sistema de medida do fenômeno físico que se apresenta sob a forma de imagem (sinal bi ou tridimensional). As imagens são capturadas por câmeras ou sensores diversos e a informação passa por uma sequência de transformações com a finalidade da realização de uma medida.

As tarefas que compõem o processamento e análise de imagens podem ser agrupadas em diferentes etapas. Este agrupamento algumas vezes é referenciado como pirâmide de processamento (DOWNTON; CROOKES, 1998), conforme apresentado na Figura 1-1. Na base desta pirâmide está a aquisição e o pré-processamento com objetivo de melhorar a qualidade das imagens, realçando as características relevantes (nível dos *pixels*). No nível acima, se encontra a segmentação, com objetivo de separar na imagem suas partes constituintes. Cabe destacar que esta etapa é bastante subjetiva, pois é a passagem da representação da imagem na forma de um sinal (2D ou 3D) para os símbolos (informações) nela presente. As etapas anteriores à segmentação são normalmente conhecidas como etapas de pré-processamento e as posteriores de pós-processamento. Já a Classificação é a etapa usada para identificar objetos ou parte destes. Por fim, na etapa de

reconhecimento é derivada uma descrição ou alguma outra interpretação dos “objetos” analisados nas imagens. Em instrumentação cientifica é possível realizar medidas em todas as fases apresentadas.


**Figura 1-1 – Organização das etapas do processamento digital de imagens sob a forma de uma pirâmide. Na parte inferior da pirâmide estão as etapas mais próximas da captura da imagem. Na parte superior estão as etapas onde há uma representação de mais alto nível da informação presente na imagem.**

Tradicionalmente, as plataformas para processamento de imagens são baseadas em computadores de arquitetura serial, onde os algoritmos são divididos em sequências de operações lógicas e aritméticas realizadas pela ULA (Unidade Lógico Aritmética) do computador (BAILEY, 2011). Nos algoritmos de execução serial, o processador executa apenas uma tarefa de cada vez. Em algumas situações esta arquitetura limitará o desempenho do processamento. Felizmente, a estrutura das imagens digitais possibilita uma abordagem de processamento por meio de códigos de execução em paralelo, especialmente para as etapas que compõem os primeiros níveis da pirâmide de processamento. Assim, a utilização de CPUs dotadas de processadores com vários núcleos para o processamento de imagens pode então solucionar a limitação no tempo de execução dos algoritmos. O processamento paralelo dos processadores multicore é um candidato muito forte para alcançar um alto desempenho, principalmente devido à quantidade de bibliotecas computacionais existentes (ALBUQUERQUE *et al.*, 2013). No entanto, como o paralelismo é muito dependente do problema, é normalmente difícil desenvolver e analisar códigos computacionais em ambientes multicore.

Uma alternativa recente para as CPUs, quando se trata de processamento paralelo de imagens, é a utilização de GPUs[2] que inicialmente foram desenvolvidas para renderização gráfica, mas que têm sido usadas para processamento de imagens (COPE *et al.*, 2005). Em GPUs, através de um processo de *multithreads*[3], uma tarefa de processamento de imagens, normalmente nas etapas de pré-processamento, pode ser divida em pequenos passos a serem executados em paralelo. Para algoritmos paralelizáveis, GPUs podem apresentar ganhos significativos de desempenho quando

[2] GPU é a sigla em inglês para *Graphics Processing Unit*, um tipo de microprocessador especializado em processar gráficos em computadores pessoais e equipamentos de entretenimento.

[3] No modelo de execução *multithread*, um processo é dividido em tarefas menores denominadas *threads*. Em computadores com vários processadores, as *threads* podem ser executadas em paralelismo real, aumentando a velocidade de execução do processo.

comparados com CPUs, no entanto, o consumo de energia nas GPUs é uma limitação para muitos sistemas embarcados de processamento de imagens (BAILEY, 2011).

Sistemas de processamento de imagens baseados em FPGA (*Field-Programmable Gate Array*) se apresentam como boa alternativa para superar as limitações de tempo de execução dos algoritmos, já que os mesmos são genuinamente paralelos e permitem flexibilidade para alteração, via *software*, das funcionalidades do *hardware*. O processamento de imagens através de FPGA é uma boa alternativa não só para processos com baixo tempo de resposta, mas também para situações onde é necessária operação em tempo real, quando confiabilidade e determinismo são cruciais. Por se tratar de *hardware* dedicado, é possível realizar e obter os resultados em um mesmo ciclo de relógio, de várias operações executadas de forma paralela. A Figura 1-2 mostra um exemplo de uma placa de instrumentação modular da *National Instruments* que contém um FPGA programável, do fabricante *Xilinx*, para o desenvolvimento de projetos personalizados em *hardware*.


**Figura 1-2 – Módulo PXI 7842R da *National Instruments* para instrumentação modular. O módulo possui um FPGA do fabricante *Xilinx*, modelo Virtex-5 LX50, para personalização de projetos em *hardware*.**

Na literatura é possível encontrar trabalhos em diferentes áreas de conhecimento, relacionados à utilização de FPGA para processamento de imagens em situações onde é necessária uma abordagem de tempo real com alta velocidade de execução dos algoritmos de processamento. Em (NGUYEN *et al.*, 2006) é demonstrada a utilização de FPGA para reconhecimento facial. Já em (CAO; ELTON; DENG, 2012) utiliza-se a abordagem em FPGA para reconhecimento de placas em uma

rodovia. Em (HSU; MIAO; TSAI, 2010) é mostrado um sistema para aplicações na área de segurança, utilizando FPGA para rastreamento de objetos através de imagens. (DILLINGER *et al.*, 2006) mostra a utilização de FPGA no tratamento de imagens médicas para segmentação das áreas branca e cinza do cérebro. (MURARI *et al.*, 2010B) mostra a utilização de FPGA no tratamento de imagens obtidas de câmeras infravermelho instaladas em reatores de fusão nuclear, para diagnóstico em tempo real do plasma de fusão.

A utilização de processamento baseado em *hardware* configurável é uma opção para experimentos científicos onde é necessária a observação de fenômenos, aquisição e processamento de várias imagens, como mostrado em (SOUZA, 2013) e (ALBUQUERQUE *et al.*, 2012) onde taxas de até 10000 imagens/s são necessárias para detecção de um fenômeno denominado *MARFE* (*Multifaceted Asymmetric Radiation From the Edge*) (LIPSCHULTZ, B. et al., 1984) que ocorre no interior de reatores de fusão nuclear, como no laboratório Jet[4].

## 1.2 Objetivo

O presente trabalho tem por objetivo, estudar, desenvolver e avaliar a inserção, em dispositivos de lógica programável (FPGA), de algoritmos que compõem as etapas iniciais de uma cadeia de processamento de imagens. Serão apresentadas as principais características dos algoritmos a serem inseridos no FPGA, assim como a determinação do tempo de execução de cada uma deles, de forma a avaliar sua viabilidade quando comparados com outros trabalhos realizados.

Além disto, será estabelecida a metodologia de transferência das imagens através de um barramento PCIe. Apartir de medições para o intervalo de tempo de transmissão das imagens entre a memória de um computador e o FPGA serão feitas avaliações para o sistema de transferência de imagens desenvolvido.

O CBPF tem desenvolvido trabalhos na área de processamento de imagens, com abordagens de programação serial e paralela em computadores, com objetivo da detecção em tempo real do fenômeno de *MARFE* que ocorre em reatores de fusão nuclear. Este fenômeno se caracteriza por um anel de radiação luminosa que se desloca acompanhando a parede do reator e é responsável pela desestabilização e interrupção do processo de fusão. A Figura 1-3 mostra a sequência do fenômeno mencionado.

[4] O laboratório JET (Joint EuropeanTorus), localizado na Inglaterra, é um dos experimentos que investigam e avaliam o potencial da fusão nuclear como fonte de energia limpa e segura (WESSON, 2006).

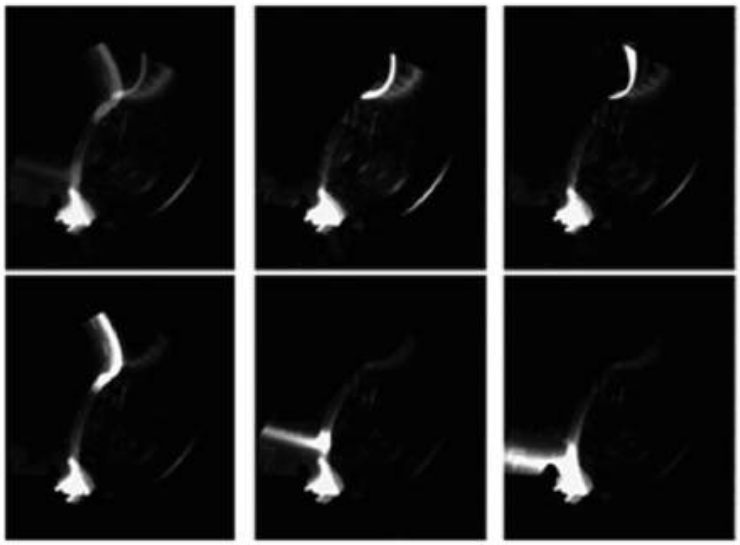


**Figura 1-3 – Da esquerda para direita, de cima para baixo, uma típica sequência do fenômeno *MARFE*, capturada com uma câmera dentro de um reator de fusão nuclear. Fonte: (ALBUQUERQUE *et al.*, 2013)**

Assim, para realização do presente trabalho, serão aplicadas, em *hardware* reconfigurável, algumas etapas que compõem o pré-processamento de um algoritmo de processamento de imagens previamente desenvolvido no CBPF, para detecção do fenômeno de *MARFE* já mencionado.

As etapas iniciais da cadeia de processamento de imagens inseridas no FPGA compõem o cálculo de imagem de fundo, a partir da média de imagens anteriores; subtração da imagem de fundo a partir da imagem corrente e, por fim, a segmentação em objetos das partes em movimento na imagem corrente.

## 1.3 Organização da Dissertação

No Capítulo 2 serão apresentados alguns conceitos teóricos sobre a fusão nuclear e o fenômeno de *MARFE*, assim como o conceito de sistemas de tempo real e como estes impõem requisitos em alguns ambientes onde são aplicadas técnicas de processamento de imagens. Serão também expostos conceitos relacionados à paralelização de algoritmos de processamento de imagens. Por fim, será apresentado o desenvolvimento já realizado pelo CBPF para detecção do fenômeno de *MARFE*, através da abordagem serial e paralela em computador, expondo as etapas do algoritmo de processamento de imagens utilizado e que serão embarcados no FPGA.

No Capítulo 3 serão apresentados conceitos teóricos das tecnologias envolvidas no desenvolvimento deste trabalho, especificamente PCIe e FPGA. Serão mostrados os ambientes de *software* utilizados para programação da plataforma de desenvolvimento e criação de uma interface que permite transferir as imagens, monitorar os recursos de memória do FPGA e para fins de comparação, realizar os mesmos algoritmos de processamento de imagens executados em *hardware*.

No Capítulo 4 serão mostrados os desenvolvimentos realizados neste trabalho. Inicialmente será mostrada a forma como foi realizado o transporte das imagens entre o computador e o FPGA, assim como a metodologia para medição do intervalo de tempo de transferência das mesmas. Estando as imagens já embarcadas no FPGA, será caracterizado o sistema de processamento de imagens inserido no *hardware* programável, bem como a metodologia para medição dos intervalos de tempos de execução de cada uma das etapas do algoritmo.

No Capítulo 5 serão apresentados os resultados obtidos para os intervalos de tempo de transferência das imagens entre um computador e o FPGA, através do barramento PCIe e também os tempos de execução de cada uma das etapas do processamento de imagens embarcado em *hardware*. Além disto, será feita uma comparação entre os resultados obtidos no processamento realizado em *hardware*, *software* e com aqueles obtidos em trabalhos anteriores.

Por fim, no Capítulo 6, serão apresentadas as conclusões, expondo as perspectivas futuras para diminuição do tempo de transferência das imagens entre computador e o FPGA e para o aumento do desempenho do processamento de imagens embarcado. Os resultados serão discutidos tanto em termos de memória interna do FPGA, ou seja, quantidade de imagens embarcadas, quanto na velocidade de execução dos algoritmos de processamento de imagens executados em *hardware*.

# Capítulo 2

# Processamento de Imagens em Experimentos de Fusão Nuclear

O presente trabalho está inserido no contexto de experimentos que utilizam processamento de imagens para o estudo de fusão nuclear. O entendimento dos conceitos teóricos envolvidos na geração de energia nuclear através de fusão, e dos desafios atualmente encontrados neste processo, leva naturalmente à necessidade de uma abordagem de tempo real para os sistemas de controle e análise que compõem o aparato de desenvolvimento da instrumentação para fusão nuclear.

Sistemas de tempo real para processamento de imagens, que necessitam de rápida resposta a eventos, podem se beneficiar do paralelismo encontrado em *hardware* reconfigurável, já que, algoritmos de processamento de imagens, principalmente que constituem etapas iniciais da cadeia de processamento, apresentam características que facilitam suas execuções de forma paralela.

O CBPF vem desenvolvendo através da abordagem de computação serial e paralela, algoritmos de processamento de imagens para detecção do fenômeno de *MARFE*, presente em reatores de fusão nuclear. Etapas desse desenvolvimento podem ser aplicadas e avaliadas em *hardware* reconfigurável, a fim de contribuir na superação dos desafios tecnológicos de hoje.

## 2.1 Conceitos básicos sobre fusão nuclear[5]

As estrelas produzem sua própria energia pela fusão nuclear de átomos leves. A luz e o calor provenientes do Sol são resultado da energia liberada a partir da fusão entre núcleos de hidrogênio. Ao longo de bilhões de anos, a força gravitacional que age no universo criou as características perfeitas para ocorrência da fusão nuclear nas estrelas. As nuvens de hidrogênio se fundiram em corpos estelares maciços com grande densidade e temperatura em seus núcleos, condições ideias para fusão nuclear.

### 2.1.1 Fusão nas estrelas

Quanto maior a temperatura dos átomos, mais rápido eles se movem. No núcleo do Sol, as temperaturas podem alcançar até 15.000.000$^{o}$ Celsius. Os átomos de hidrogênio estão em constante movimento, colidindo entre eles em altas velocidades. Com isto, consegue-se vencer a repulsão coulombiana entre cargas iguais e os átomos se fundem. A fusão de átomos leves de hidrogênio resulta em um átomo mais pesado de hélio.

[5] A maior parte do texto desta seção foi adaptada a partir de http://www.iter.org, http://www.plasma.inpe.br e http://www.efda.org/jet.

A massa do hélio resultante não é exatamente a soma dos dois átomos de hidrogênio iniciais, ou seja, uma parcela de massa é perdida e é gerada uma grande quantidade de energia. A cada segundo, o Sol funde cerca de 600 milhões de toneladas de hidrogênio em hélio, liberando uma enorme quantidade de energia.

### 2.1.2 Fusão em laboratório

A probabilidade de ocorrência da reação entre dois átomos de hidrogênio é muito baixa, liberando energia numa taxa extremamente lenta que não apresenta importância para produção de energia industrial. Esta reação é viável apenas onde existe grande quantidade disponível de combustível (hidrogênio), como ocorre no Sol e nas estrelas. Assim, identificou-se que as reações de fusão mais eficientes a serem realizadas em laboratório envolvem dois isótopos[6] de hidrogênio: deutério (D) e o trítio (T). O deutério apresenta um próton e um nêutron, enquanto que o trítio é formado por um próton e dois nêutrons. A Figura 2-1 mostra o esquema dos átomos para os isótopos do hidrogênio considerados.


**Figura 2-1 – Esquema dos átomos para dois isótopos do hidrogênio: deutério com um próton e um nêutron; trítio com um próton e dois nêutrons.**

A reação entre estes isótopos que ocorre mais facilmente e que libera a maior quantidade de energia é aquela em que o deutério se funde com o trítio, produzindo uma partícula alfa (núcleo de $He^4$), um nêutron e 17.6 Mev de energia, conforme visto na Equação 2-1 e Figura 2-2.

$$D^2 + T^3 => He^4 + neutron + 17.6Mev \qquad (2\text{-}1)$$

[6] Isótopos são variantes de um elemento químico particular. Enquanto todos os isótopos de um dado elemento compartilham o mesmo número de prótons, cada isótopo difere dos outros em seu número de nêutrons.

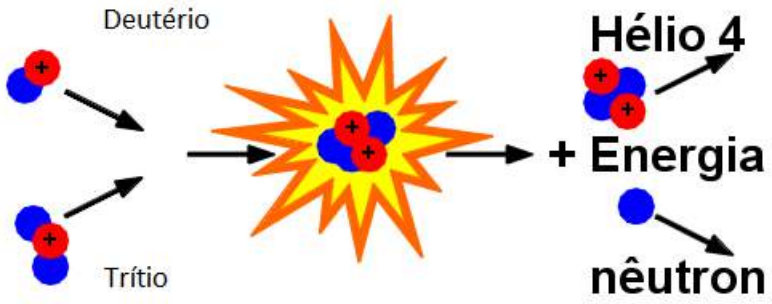



**Figura 2-2 – Reação D-T. Para que uma reação de fusão possa ocorrer, os núcleos devem colidir com energia suficiente para vencer a repulsão coulombiana entre cargas de mesmo sinal. A reação D-T apresenta alto tunelamento quântico[7], resultando em uma seção de choque[8] elevada para energias de impacto, relativamente baixas.**

Para ocorrência das reações de fusão nuclear é preciso que os núcleos possuam energia cinética muito grande, a fim de permitir o aumento da probabilidade da penetração da barreira coulombiana que os separa, o que é conseguido com o aumento da temperatura. Além disto, a densidade do combustível (núcleos) deve ser controlada em valores específicos.

Para a reação D-T, anteriormente mencionada, é preciso temperaturas entre 100 e 200 milhões de graus Celsius e densidade da ordem de 1 miligrama por $m^3$ (cerca de um milionésimo da densidade do ar).

## Plasma

As altas temperaturas colocam os combustíveis da fusão nuclear em um estado denominado plasma. Este pode ser considerado o quarto estado da matéria, quando a elevada temperatura faz-se com que átomos se tornem ionizados, com os elétrons se afastando de seus núcleos. O plasma possui todas as propriedades dinâmicas dos fluidos, além de também sofrerem influência de campos magnéticos, já que são formados de partículas carregadas. A fim de se sustentar um plasma, é preciso manter sua temperatura suficientemente alta, utilizar uma mistura de reagentes pouco densas, diminuindo a probabilidade de recombinação entre íons e elétrons, e submeter este plasma a ação de uma força externa de forma a mantê-lo confinado.

Como o plasma é sujeito a interações eletromagnéticas, ele pode ser confinado por campos magnéticos externos, que atuam como um recipiente invisível que não entra em contato direto com ele, mas o mantêm suspenso dentro da câmara de confinamento. A ação do campo magnético restringe o movimento das partículas carregadas e evita que estas atinjam as paredes do reator.

[7] Efeito quântico que consiste em uma partícula atravessar uma barreira de energia potencial maior que sua energia total.

[8] Definida como a área que mede a probabilidade de ocorrência de uma colisão entre partículas.

## Tokamaks

O caminho de maior sucesso para o confinamento magnético do plasma é o reator Tokamak. Este consiste num toróide (formato de um pneu) no qual uma câmara de vácuo contém um anel de plasma confinado por campos magnéticos, Figura 2-3.

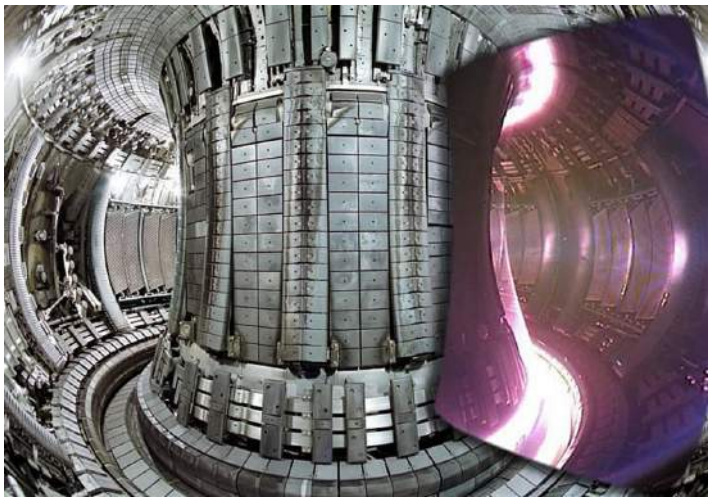


**Figura 2-3 – Visão das paredes internas de um Tokamak, sobreposta com uma imagem de um plasma obtida por uma câmera de vídeo (espectro visível). É possível perceber seu formato toroidal (Fonte: http://www.iter.org/sci/whatisfusion).**

Com a utilização de campos magnéticos, o plasma está isolado da parede do reator, com isto, evita-se o resfriamento do mesmo e também sua contaminação por outros átomos que podem contribuir para este resfriamento. Outro fator que pode resfriar o plasma é o acúmulo de "cinzas" de hélio (produto da combustão). À medida que o combustível vai sendo utilizado no processo de fusão, ele deve ser substituído, e o hélio desacelerado deve ser removido. Entretanto, inicialmente esses núcleos de hélio (conhecidos como partículas alfa) são fundamentais para manter a temperatura do plasma e, portanto dar continuidade às reações de fusão. Quando a potência dissipada pelas partículas alfa é suficiente para manter a temperatura do plasma, a reação se torna autossustentada, e essa condição é chamada de ignição.

O grande desafio das investigações de fusão controlada é atingir a condição de ignição, ou seja, garantir a combustão do plasma através da energia cinética dos subprodutos confinados da reação de fusão. Em um reator que funcione com uma mistura de D-T, as reações serão autossustentadas pela energia cinética dos átomos de hélio, que será suficiente para manter a temperatura e densidade de combustão, sem qualquer energia exterior. Dessa maneira, a energia liberada pelo reator será maior que a energia necessária para iniciar as reações de fusão (elevar a temperatura e confinar o plasma), e a razão entre essas energias é chamada de ganho (Q). Quando o ganho Q é maior que 1, o reator produz mais energia que aquela que tem de ser consumida para estabelecer o processo de fusão. Este procedimento parece simples, mas a tecnologia envolvida em

cada uma dessas etapas é um desafio tecnológico para esta e futuras gerações. Até os dias de hoje, o recorde de ganho é aproximadamente Q=0.65, obtido no laboratório JET (na cidade de Culham na Inglaterra) no ano de 1997.

## Sistemas de medição e controle para fusão nuclear em laboratório

Nos experimentos de fusão nuclear, os cientistas estão continuamente monitorando parâmetros como temperatura e densidade do plasma, os campos magnéticos de confinamento, velocidade das partículas no plasma etc. O Tokamak do Laboratório Jet, por exemplo, é constituído por mais de cem diferentes sistemas de diagnósticos, sendo que mais da metade deles são utilizados durante um experimento de fusão. Durante um dia de experimento, os sistemas de diagnósticos geram *terabytes* de dados.

O diagnóstico do plasma serve para dois propósitos principais: Primeiro, os dados são utilizados para aprender e analisar o comportamento do plasma. É observado, por exemplo, como o plasma responde aos sistemas de aquecimento e como as instabilidades e impurezas provenientes da parede do reator interagem com o plasma. Segundo, os sistemas de medição são utilizados para proteger a integridade do reator. Por exemplo, instabilidades, caso não sejam reconhecidas e controladas em tempo hábil, podem fazer com que o plasma se expanda e danifique as paredes do reator. A abordagem mais comum é o cálculo da geometria do plasma através de diferentes tipos de sensores magnéticos. Estes sensores servem como entrada para sistemas de controle, que, por exemplo, ajustam automaticamente os campos magnéticos de confinamento, mantendo sua forma e trajetória de acordo com padrões pré-estabelecidos (ALVES *et al.*, 2014).

## Processamento de imagens para experimentos de fusão nuclear

Conforme mencionado anteriormente, o aparato para desenvolvimento dos experimentos em fusão nuclear é constituído de inúmeros sistemas de diagnóstico e controle. Nos Tokamaks para confinamento magnético, as câmeras de vídeo têm se tornado instrumentos de diagnóstico para o plasma (MURARI *et al.*, 2010). Novas ferramentas e métodos de processamento de imagens são requeridos para manipular os frames de vídeos das câmaras instaladas no interior dos reatores. Algoritmos para processamento de imagens têm sido desenvolvidos com objetivo de identificação de padrões nas imagens capturadas dentro dos reatores.

Para controle das instabilidades no plasma, uma análise em tempo real dos vídeos capturados no interior dos reatores é de suma importância para atuação em tempo hábil nos sistemas de controle. Além disto, muitos sistemas de diagnóstico, além de requerem processamento em tempo real, necessitam rápida velocidade de execução dos algoritmos de análise dos dados.

## Sistemas de tempo real x Sistemas de execução rápida

É importante a apresentação de alguns conceitos sobre sistemas de tempo real e sistemas de rápida execução. O objetivo no desenvolvimento de algoritmos rápidos é construir ferramentas para minimizar o tempo médio de resposta de um determinado conjunto de tarefas. No entanto, o objetivo de computação em tempo real é cumprir uma exigência de tempo específico de cada tarefa. Diferente de rápido (que é um termo relativo), a propriedade mais importante de um sistema em

tempo real é a previsibilidade, ou seja, sua funcionalidade e comportamento com relação ao tempo devem ser tão precisos quanto o necessário para satisfazer as especificações do sistema principal. Computação rápida é útil para atender especificações de tempos rigorosos, mas a computação rápida em si só não garante a previsibilidade (MCKENNEY, 2009) e (GAMBIER, 2004).

A partir da perspectiva do processamento de imagens, um sistema de tempo real será aquele que captura regularmente as imagens, analisa, obtém informações e então utiliza estes dados para alguma ação de controle. Sendo que todos estes passos devem ocorrer dentro de um intervalo com um limite máximo de tempo (BAILEY, 2011).

Neste trabalho está sendo considerada a análise de imagens de um fenômeno denominado *MARFE*. Estes surgem nos Tokamaks à medida que o limite de densidade do plasma é alcançado. A presença dos mesmos restringe o confinamento e interrompe bruscamente o plasma, além de comprometer a integridade do reator. O *MARFE* é caracterizado como um anel brilhante concêntrico de radiação que se forma normalmente na parte superior do toróide e se movimenta acompanhando a parede do reator. Câmeras de alta velocidade no espectro visível, instaladas dentro dos reatores podem então capturar a ocorrência dos *MARFEs*. Conforme mostrado em (ALBUQUERQUE *et al.*, 2013), onde pela análise de um banco de dados de imagens de MARFES previamente capturadas em um reator de fusão nuclear, estima-se que duração média do fenômeno analisado é da ordem de $1.438 \pm 0.489 ms$ . Isto implica que a análise de apenas uma imagem dentro de um período de um *MARFE*, reduz muito a chance de detecção e atuação na correção da instabilidade (ALBUQUERQUE *et al.*, 2013).

Assim, o compromisso em detectar o fenômeno de *MARFE* e corrigir o plasma dentro dos limites de tempo, leva sem dúvida à necessidade de algoritmos de processamento de imagens de rápida execução e de operação em tempo real.

## 2.2 Paralelização aplicada ao processamento de imagens

Em princípio, cada etapa de um algoritmo de processamento de imagens pode ser executada em um processador de forma independente. No entanto, se um algoritmo é predominantemente serial, com cada etapa dependendo de dados gerados em passos anteriores, muito pouco pode ser obtido em termos de redução de tempo de execução, caso não sejam utilizadas técnicas de paralelismo de dados e/ou *pipeline*, assim descritas na Seção 2.2.2.

A ideia principal da paralelização dos algoritmos de processamento de imagens é dividir o problema em tarefas simples e então executá-las de forma concorrente a fim de que o tempo total possa ser dividido de acordo com o número de tarefas menores criadas (no melhor caso). No entanto, deve-se resaltar que nem todos os problemas podem ser tratados de forma paralela (SAXENA; SHARMA; SHARMA, 2013)

### 2.2.1 Características das imagens digitais

Uma imagem pode ser definida como uma função bidimensional, *f(x,y),* onde *x* e *y* são coordenadas no plano e a amplitude *f* em qualquer par de coordenadas *(x,y)* é chamada de

intensidade ou nível de cinza naquele ponto. Quando *x*, *y* e a intensidade têm valores finitos, e quantidades discretas, define-se uma imagem digital.

Uma imagem digital será então composta por um número finito de elementos, denominados *pixels*, com localização e valores definidos. A densidade espacial de *pixels* de uma imagem pode ser utilizada para informar sua resolução, sendo a convenção descrever a resolução como um conjunto de números inteiros positivos, em que o primeiro é a quantidade de colunas (largura) de *pixels* e o segundo é o número de linhas (altura) de *pixels*; algo como *640 X 480*, por exemplo. A Figura 2-4 mostra uma mesma imagem com diferentes resoluções. Para a resolução de *5 x 5*, por exemplo, a imagem é formada por 5 linhas, onde cada uma destas linhas possui 5 *pixels*, totalizando assim 25 *pixels* diferentes.

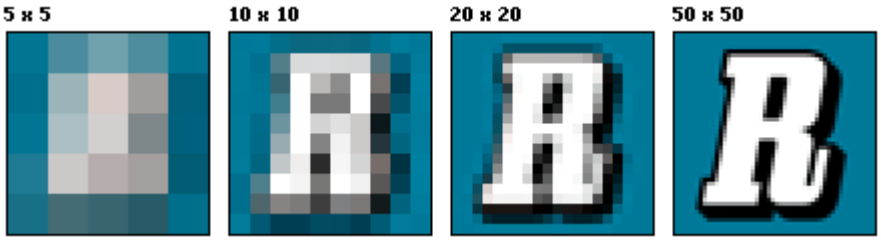



**Figura 2-4 – Exemplo de uma mesma imagem com diferentes resoluções espaciais.**

A profundidade em bits de uma imagem é definido de acordo com os valores que cada *pixel* pode assumir, e.g., em uma imagem de 8 bits em tons de cinza, os *pixels* podem assumir valores entre 0 e 255 em uma escala de cinza, onde por exemplo, o valor zero será o preto e o valor 255 será o branco. As imagens coloridas podem ser representadas no espaço de cores *RGB (RED, GREEN, BLUE)* (GONZALEZ; WOODS, 2008), onde cada *pixel* terá um valor proveniente da combinação entre as tonalidades vermelho, verde e azul. Em uma imagem *RGB* de 24 bits, os *pixels* serão constituídos de 3 conjuntos de 8 bits, um para cada uma das cores (vermelho, verde e azul), dando um alcance de 256 valores possíveis, ou intensidades para cada tom, o que resulta em mais de 16 milhões de cores diferentes, neste sistema, e.g., um *pixel* branco será representado por *RGB*(255,255,255) e o preto por *RGB*(0,0,0).

A Figura 2-5 mostra um exemplo para uma imagem de 8 bits em tons de cinza, com resolução de *341 x 341 pixels*, juntamente com seu histograma de níveis de cinza, o qual informa as quantidades de *pixels* da imagem distribuídas ao longo dos 256 valores de tonalidades possíveis. No histograma da figura é possível verificar a quantidade total de *pixels* da imagem, no caso 116281, além dos valores máximo, mínimo e médio dos *pixels*. Em particular é mostrado que existem 451 *pixels* com valor igual a 50.

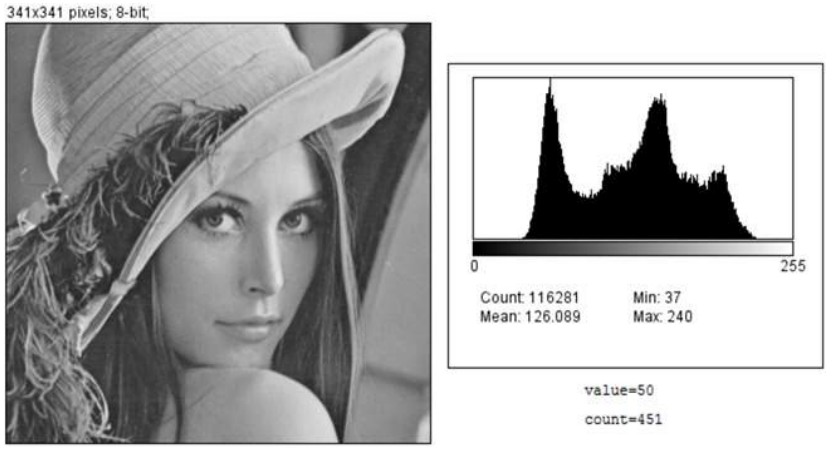



**Figura 2-5 – Esquerda: Exemplo da imagem Lena que é usada frequentemente na área de processamento de imagens. Todos os pixels desta imagem estão representados por 8 bits de tons de cinza. Direita: Histograma para imagem mostra a distribuição da quantidade de pixels dentre os 256 valores possíveis.**

### 2.2.2 Paralelismo de dados e *Pipeline*

Os algoritmos que compõem as etapas iniciais de uma cadeia de processamento de imagens são naturalmente paralelos, sendo esses classificados em: operadores pontuais; operadores de vizinhança e operadores globais (UMBAUGH, 1998). Esta classificação leva em conta a determinação dos *pixels* de saída de uma imagem apartir de *pixels* de entrada. O caso mais simples são os operadores pontuais. Neste, um *pixel* de uma imagem de saída depende apenas do valor do *pixel* na imagem de entrada. Os operadores de vizinhança calculam a imagem de saída a partir de uma operação envolvendo os *pixels* que estão na vizinhança dos *pixels* na imagem de entrada. Já os operadores globais calculam o valor de um *pixel* de uma imagem de saída utilizando todos os *pixels* da imagem de entrada. A Figura 2-6 mostra os três tipos de operadores mencionados.

No contexto deste trabalho, são utilizados apenas operadores pontuais, sendo estes detalhados na seção seguinte.

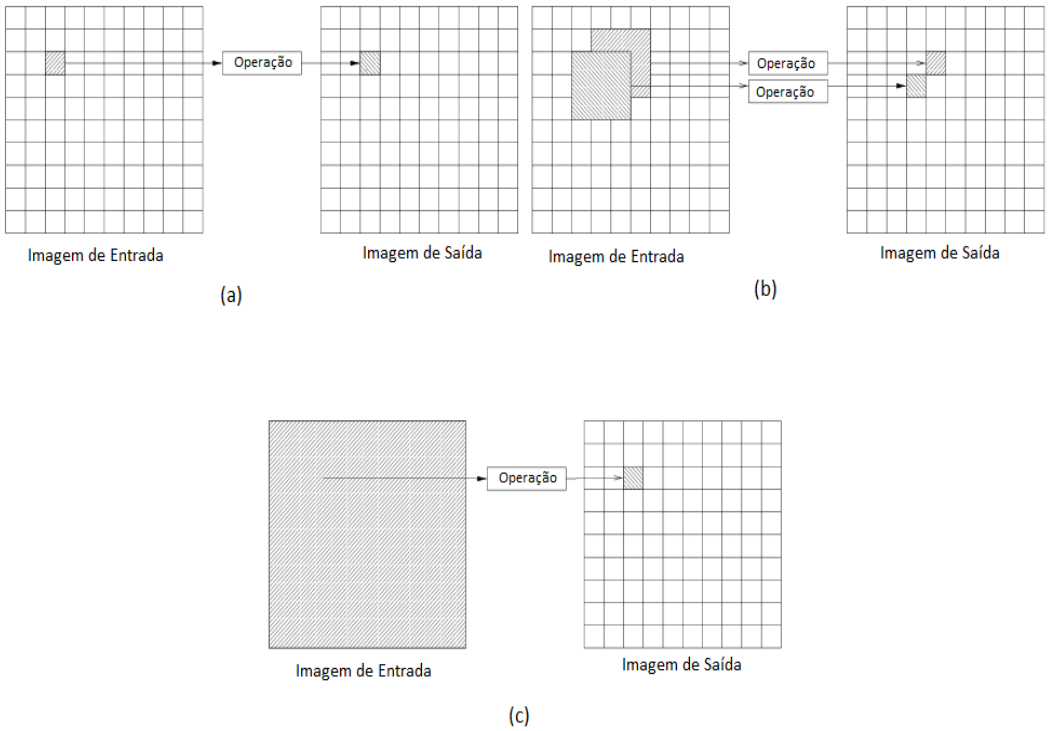



**Figura 2-6 – (a) Operador pontual, *pixel* resultante depende apenas de *pixel* de entrada. (b) Operador de vizinhança, *pixel* resultante depende de um conjunto de *pixels* de entrada. (c) Operador global, *pixel* resultante depende de todos os *pixels* da imagem de entrada.**

Observa-se então que a maioria dos operadores que compõem etapas inciais de uma cadeia de processamento de imagens exibem um paralelismo natural, já que a parcela da imagem de entrada necessária para cálculo da imagem de saída está espacialmente localizada. Esta característica é conhecida como paralelismo de dados (DOWNTON; CROOKES, 1998). Em termos de desenvolvimento, este paralelismo de dados pode ser explorado com a divisão das imagens e a utilização de processadores independentes para realizar operações concorrentes nos diferentes blocos que compõem a imagem. Esquemas comuns dividem as imagens em blocos de linhas, colunas ou blocos retangulares, conforme visto Figura 2-7.

Em casos extremos, pode-se utilizar um processador independente para cada um dos *pixels* que compõem a imagem, e.g., Processadores Massivamente Paralelos (BATCHER, 1980). Deste modo é realizada uma divisão e processamento de um imagem digital em sua unidade fundamental, i.e., um *pixel*.


**Figura 2-7 – Divisão das imagens para aplicação de paralelismo de dados nos algoritmos de processamento. Cada processador fica responsável por aplicar operações em cada um dos blocos que compõem a imagem. (a) Divisão em linhas. (b) Divisão em colunas. (c) Divisão em blocos retangulares**

A melhora no desempenho no quesito velocidade de execução aumenta então com a quantidade de divisões possíveis realizadas em uma imagem, no entanto, a possível comunicação entre os processadores responsáveis pelas operações em cada uma das partes e a necessidade destes em acessar recursos de memória compartilhados pode reduzir em muito o desempenho em termos de velocidade de execução dos algoritmos. Assim, faz-se necessário que cada processador possua uma quantidade de memória local para reduzir atrasos relacionados com o acesso de recursos compartilhados de memória.

A ideia do paralelismo de dados para o processamento de imagens é beneficiada quando os operadores responsáveis por calcularem as imagens de saída requerem dados de pequenas regiões que compõem uma imagem de entrada. Assim, os algoritmos que compõem etapas inciais de uma cadeia de processamento mais indicados para o paralelismo de dados são aqueles pertencentes ao grupo de operadores pontuais. Quando as operações realizadas em cada uma das regiões de uma imagem são idênticas, tem-se uma arquitetura de processamento paralelo denominado *SIMD (Single Instruction Multiple Data)* de acordo com a taxonomia de Flynn (FLYNN, 1972).

De acordo com a caracterização do paralelismo de dados anteriormente exposta, percebe-se que este pode ser aplicado nos diferentes operadores. No entanto, etapas de uma cadeia de processamento de imagens não são constituídas de apenas um operador e sim de uma série de operações realizadas muitas vezes de forma sequêncial, conforme Figura 2-8.

Observando a Figura 2-8 pode-se aplicar o paralelismo de dados dentro de cada operação. No entanto, para que se consiga o paralelismo entre operações, é necessário que se recorra a uma arquitetura de *pipeline.* Quando possível cada uma das operações deverá ser executa em um processador independente, conforme Figura 2-9. Esta figura apresenta o diagrama de tempo para uma sequência de quatro imagens a serem processadas por três processadores que executam três operações distintas. Ao final da operação 1 pelo processador 1 na imagem 1, este pode passar o resultado ao processador 2 e imediatamente iniciar o mesmo processamento (operação 1) na imagem 2.

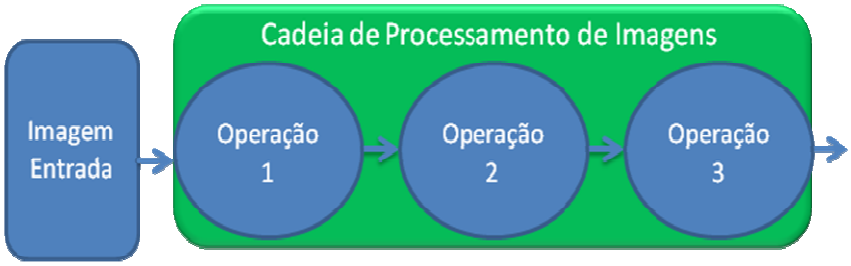



**Figura 2-8 – Sequência de operações em uma etapa de processamento de imagens. O resultado de cada operação é enviado para a operação seguinte.**

Algumas dificuldades podem ocorrer quando se utiliza a técnica de *pipeline* para algoritmos de processamento de imagens. Por exemplo, a taxa de entrada de imagens na cadeia de processamento será determinada pelo tempo de execução da operação mais lenta. Além disto, para algoritmos mais complexos, a existência de múltiplos caminhos paralelos, por exemplo, na Figura 2-9, caso o processador 3 necessite de dados também provenientes do processador 1, será necessária a inserção de linhas de atraso para sincronização dos dados, tornando mais complexa a cadeia.

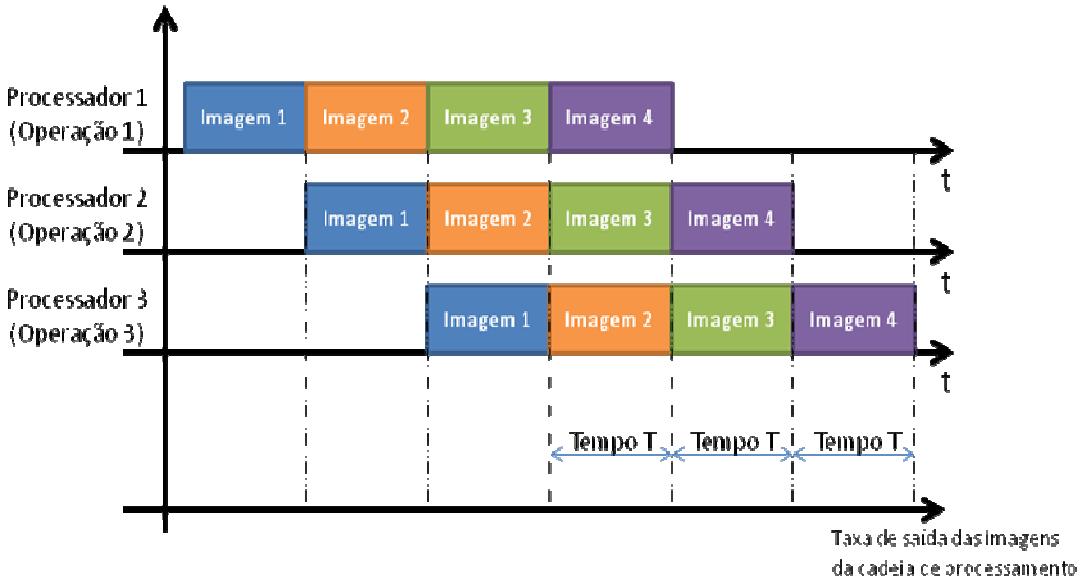



**Figura 2-9 – Paralelismo entre operações através de *pipeline*. Cada processador é responsável por uma operação na cadeia de processamento das imagens. Neste diagrama cada uma das operações é executada com o tempo T, o que resulta em uma taxa de saída das imagens da cadeia de processamento com o mesmo tempo T.**

Conforme apresentado no decorrer deste trabalho, será utilizado apenas o paralelismo de dados, tendo como proposta a divisão das imagens, para aumento da velocidade de execução das etapas iniciais da cadeia de processamento de imagens, aproveitando para isto, a característica de execução naturalmente paralela dos FPGAs.

## Operadores pontuais e processamento paralelo

No presente trabalho serão inseridos e executados em *hardware*, apenas operadores pontuais que compõem uma cadeia de processamento de imagens. Conforme já mencionado, estes operadores necessitam apenas de um *pixel* da imagem de entrada para o cálculo do *pixel* correspondente em uma imagem de saída. Deste modo, a quantidade de divisões que podem ser feitas uma imagem, para a exploração do paralelismo de dados, fica limitada apenas pelos recursos de *hardware* disponíveis para realização das operações em cada uma das partes criadas. Observando que no caso ideal, onde se considera recursos de *hardware* suficientemente grandes, poder-se-ia chegar a operações pontuais para cada *pixel* de uma imagem de entrada.

Operações pontuais podem ser classificadas em: (i) operações pontuais de imagem única e (ii) operações pontuais de múltiplas imagens (BAILEY, 2011). No primeiro caso, o cálculo da imagem da saída depende apenas de uma imagem de entrada. Já no segundo caso, o cálculo da imagem de saída dependerá da combinação de múltiplas imagens de entrada. A Figura 2-10 mostra o esquema da operação pontual para uma imagem e múltiplas imagens

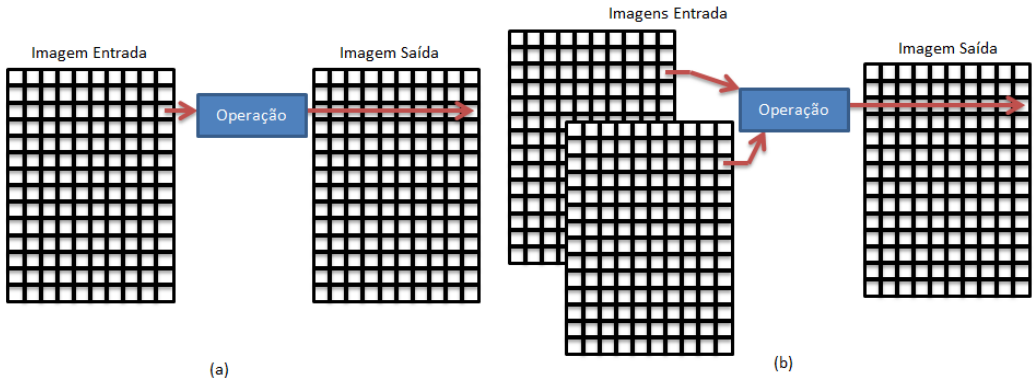



**Figura 2-10 – Operação Pontual. (a) Imagem Única, neste caso um pixel correspondente na imagem de entrada passa por uma operação que resulta no pixel correspondente na imagem de saída. (b) Múltiplas imagens, neste caso o pixel da imagem de saída é resultado da operação entre dois ou mais pixels correspondentes, provenientes de diferentes imagens de entrada.**

Nas situações reais, onde é impraticável a utilização de um processador de operações para cada *pixel*, pode-se dividir a imagem de entrada em uma quantidade de blocos de *pixels*, de acordo com os recursos disponíveis, e aplicar as operações de forma sequêncial em cada um dos *pixels* que forma um dado bloco. A Figura 2-11 mostra uma imagem dividida em três partes, cada operação paralela é responsável por realizar o cálculo nos *pixels* que constituem um dos blocos da imagem.

Neste caso, sendo a imagem formada por *N pixels* e considerando que a operação pontual é executada em um tempo *T,* tem-se um tempo total de $\frac{T \times N}{3}$, ou seja, 3 vezes menor caso não se utilize a divisão da imagem de entrada.

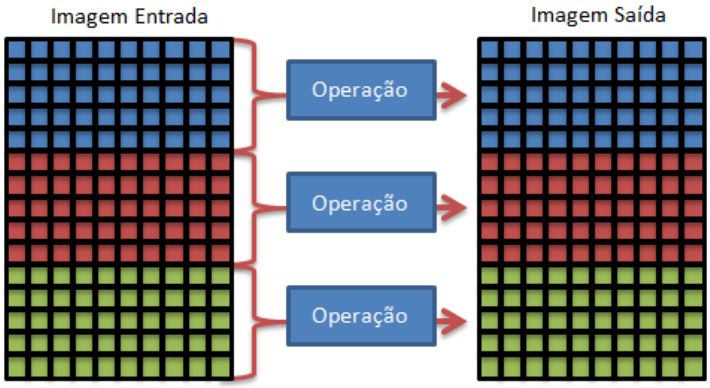



**Figura 2-11 – Imagem de entrada dividida com três partes. As operações paralelas calculam serialmente cada um dos pixels que forma um bloco específico da imagem.**

As operações pontuais podem ser aplicadas nas etapas iniciais de uma cadeia de processamento para melhorar a qualidade de uma imagem digital de entrada, de forma a facilitar a extração de informações nas etapas de reconhecimento e classificação em um algoritmo de processamento de imagens.

Devido as suas utilizações neste trabalho, serão detalhados aqui, o algoritmo de estimação de imagem de fundo e o algoritmo de subtração de uma imagem de fundo apartir de uma imagem corrente, ambos pertencentes ao grupo de operadores pontuais de múltiplas imagens. Será mostrado também o algoritmo de segmentação por *threshold* também conhecido como binarização, este pertencente ao grupo de operadores pontuais de imagem única.

## Segmentação por *Threshold*

Segmentação de imagens é o processo de separação de uma imagem em regiões homogêneas de *pixels*, que podem ser processadas como um grupo. Muitas aplicações em visão computacional utilizam técnicas de segmentação para reduzir as informações das imagens e com isto ajustar a eficiência de toda a cadeia de processamento (FITZGERALD; WILL; WILL, 2014).

A operação de segmentação por *threshold* compara cada *pixel* da imagem de entrada com um limiar de corte, resultando assim em um valor verdadeiro ou falso (branco ou preto) para o *pixel* de saída. Deste modo, é feita a separação de cada *pixel* em duas classes, sendo esta operação muitas vezes utilizada como técnica de segmentação entre um objeto que se quer localizar e o fundo que compõem uma imagem digital. A Figura 2-12 mostra um exemplo da técnica de segmentação por

*Threshold*, é possível perceber que a imagem de saída apresenta apenas dois grupos de *pixels*, i.e., *pixels* pretos e *pixels* brancos.

O nível apropriado para o limiar de *Threshold* pode ser selecionado a partir de uma análise estatística da imagem, e.g., análise dos valores dos *pixels* contidos no histograma da imagem (BAILEY, 2011).

$$\text{Pixel de Saída} = \begin{cases} 1, \text{se Pixel de Entrada} \geq \text{Limiar} \\ 0, \text{se Pixel de Entrada} < \text{Limiar} \end{cases}$$

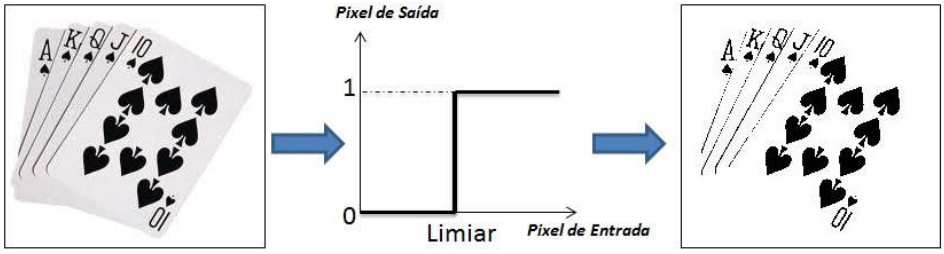



**Figura 2-12 – Segmentação através da técnica de *Thresholding* (Binarização). Cada *pixel* da imagem de entrada é comparado com um valor limite, resultando então em um *pixel* preto (zero) ou branco (um).**

## Estimação e subtração de imagem de fundo

Subtração de imagem de fundo é um procedimento usado normalmente para separação dos *pixels* da imagem em dois grupos: aqueles que pertencem a objetos em movimento e aqueles que representam um fundo estático. Este é um dos algoritmos mais utilizados na análise de imagens de vídeos para rastreamento do movimento de objetos e detecção automática de eventos (CZYżEWSKI *et al.*, 2011). A imagem de fundo é normalmente resultado de um modelo, que pode ser baseado, e.g., na análise estatística dos valores dos *pixels* provenientes de diferentes imagens (SZWOCH; ELLWART; CZYżEWSKI, 2012).

A necessidade de estimar e subtrair a imagem de fundo está presente no contexto de processamento de imagens sequências. Em situações onde são adquiridas imagens em modo dinâmico, é possível considerar as imagens anteriores a fim de estimar uma imagem de fundo (MURARI *et al.*, 2010A). Assim, em uma sequência de imagens onde existe um dado objeto em movimento, um dos métodos para calcular os *pixels* que representam a imagem de fundo para uma dada imagem no tempo *T,* será aplicação de um filtro de média (CUCCHIARA *et al.*, 2003). Neste caso, é realizada a média aritmética entre os *pixels* correspondentes, provenientes de *N* imagens obtidas nos tempos *T-1, T-2,...T-N.*

Observando a Figura 2-13 vê-se o efeito da média aritmética aplicada em uma sequência de oito imagens capturadas nos tempos *T-1* a *T-8*. Percebe-se que o objeto que se movimenta nas imagens, i.e., ponteiro dos segundos, praticamente desaparece (os valores dos *pixels* que formam o ponteiro tendem ao valor zero) na imagem de fundo estimada, a qual irá conter apenas os objetos

estáticos. Para qualquer *pixel* da imagem de fundo estimada, o valor deve se aproximar de zero para objetos em movimento ou ser igual ao valor dos respectivos *pixels* para as *N* imagens anteriores (objetos estáticos).

Não existe um valor ótimo para a quantidade de imagens anteriores necessárias para a estimação da imagem de fundo, estando este valor intimamente relacionado com o problema de processamento de imagens que se deseja aplicar esta técnica. Além disto, quanto mais imagens anteriores utilizadas, maior deverá ser o *buffer* de memória disponível para armazená-las.

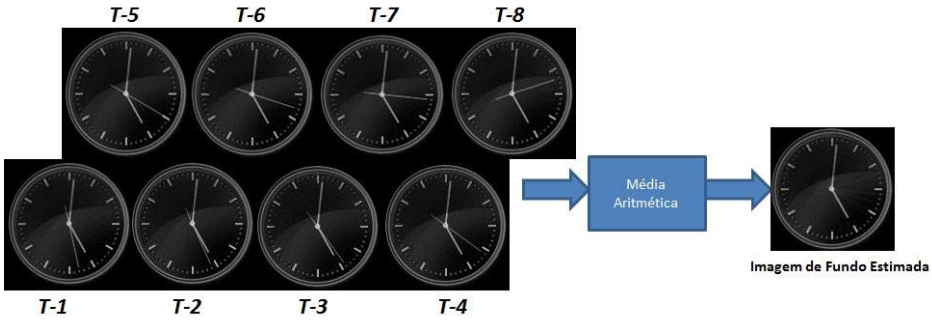



**Figura 2-13 – Exemplo de determinação de imagem de fundo a partir da média aritmética entre os pixels de imagens anteriores. A imagem de fundo estimada conterá apenas objetos estáticos**

A diferença entre a imagem de fundo estimada a partir da imagem corrente, ou seja, imagem no tempo *T irá* resultar então em uma imagem de saída que contém apenas o objeto em movimento. A Figura 2-14 mostra a ideia de subtração da imagem de fundo. O valor para os *pixels* resultantes da subtração devem ser maiores ou iguais a zero, onde o valor zero será obtido para os objetos estáticos e os valores maiores que zero representam os objetos em movimento.

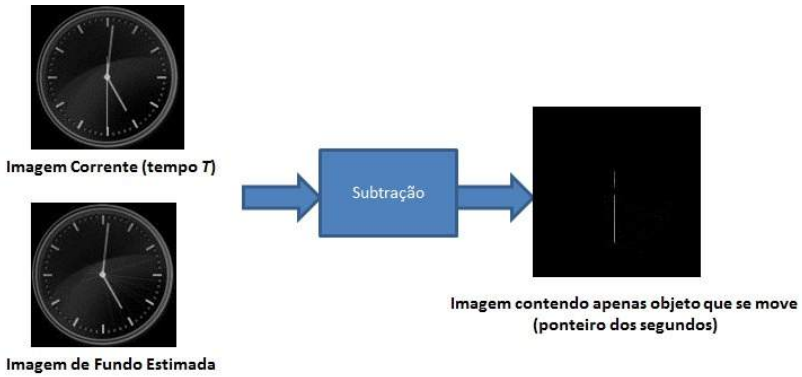



**Figura 2-14 – Subtração da imagem de fundo estimada (a partir de N imagens anteriores) da imagem corrente. O resultando é uma imagem que contem apenas o objeto que se movimenta na sequência de imagens.**

Espera-se que a subtração não resulte em *pixels* com valores negativos, já que neste caso o cálculo da média, para estimar a imagem de fundo, estaria aumentando o valor dos *pixels* para os objetos estáticos e/ou inserindo na imagem estática também os objetos em movimento. Valores negativos podem ocorrer caso, em alguma das imagens anteriores usadas para o cálculo da imagem de fundo existam *pixels* relativos a algum "objeto" inesperado. O resultado da subtração de imagem de fundo muitas vezes necessita passar por um pós-processamento, para remover ruído, preencher espaços vazios que foram deixados na imagem ou mesmo realçar os objetos que se deseja rastrear na sequência de imagens, para este fim pode-se utilizar a técnica de Segmentação por *Threshold* anteriormente descrita. A Figura 2-15 mostra aplicação da Binarização.


**Figura 2-15 – Segmentação através da técnica de *Thresholding* (Binarização por limiarização). Objeto que se movimenta (ponteiro dos segundos) foi realçado a fim de facilitar sua detecção pelos algoritmos de reconhecimento de padrões que compõem as etapas mais complexas de um algoritmo de processamento de imagens.**

Uma etapa de pré-processamento constituída pela subtração da imagem de fundo estimada a partir da imagem corrente, seguido pela segmentação por *threshold*, pode ser descrito pela Equação 2-3. Um *pixel* de saída $Q$ na posição (x,y) terá valor um, i.e., cor branca, caso a subtração do *pixel* $B$ na posição (x,y) da imagem de fundo estimada a partir do *pixel* $I$ na posição (x,y) da imagem corrente seja maior ou igual ao limiar de *Threshold* estabelecido, caso contrário, o *pixel* de saída $Q$ na posição (x,y) terá valor zero, i.e., cor preta.

$$Q_{x,y} = \begin{cases} 1, \text{se } \left(\mathrm{I}_{\mathrm{x,y}} - B_{x,y}\right) \geq Limiar \\ 0, \text{se } \left(\mathrm{I}_{\mathrm{x,y}} - B_{x,y}\right) < Limiar \end{cases} \qquad (2\text{-}3)$$

Além da técnica de estimação de imagem de fundo descrita, i.e, média de *N* imagens anteriores, existem outros métodos de modelagem que podem variar significativamente em termos de complexidade (MCIVOR; ZANG; KLETTE, 2001), e.g., situações onde existem variações nas condições da imagem de fundo exigem modelos adaptativos de estimação. Métodos mais complexos e avançados para estimação e subtração de imagem de fundo são revisados em (PICCARDI, 2004).

## 2.3 Processamento de imagens em ambiente computacional para detecção de *MARFE*

O presente trabalho é baseado em um algoritmo de processamento de imagens já desenvolvido pelo CBPF, para avaliação de imagens sequências, previamente obtidas em um reator Tokamak durante um experimento de fusão nuclear, com objetivo de detectar o fenômeno de *MARFE*.

A primeira etapa de desenvolvimento utilizou uma abordagem de execução serial em ambiente computacional. Posteriormente, passou-se para o estudo e desenvolvimento da execução utilizando técnicas de computação paralela e de características de tempo real.

A evolução na forma de execução, passando de uma abordagem em execução serial para execução paralela visou aumentar a velocidade de execução, além de inserir conceitos de computação em tempo real. Estas características serão requeridas para a uma aplicação prática deste algoritmo de processamento de imagens na detecção dos *MARFES* e correção das características do plasma de fusão, conforme já descrito na Seção 2.1.

### 2.3.1 Algoritmo serial

A Figura 2-16 mostra o diagrama em blocos contendo as etapas que compõem o algoritmo para detecção serial de *MARFES* em uma sequência de imagens.

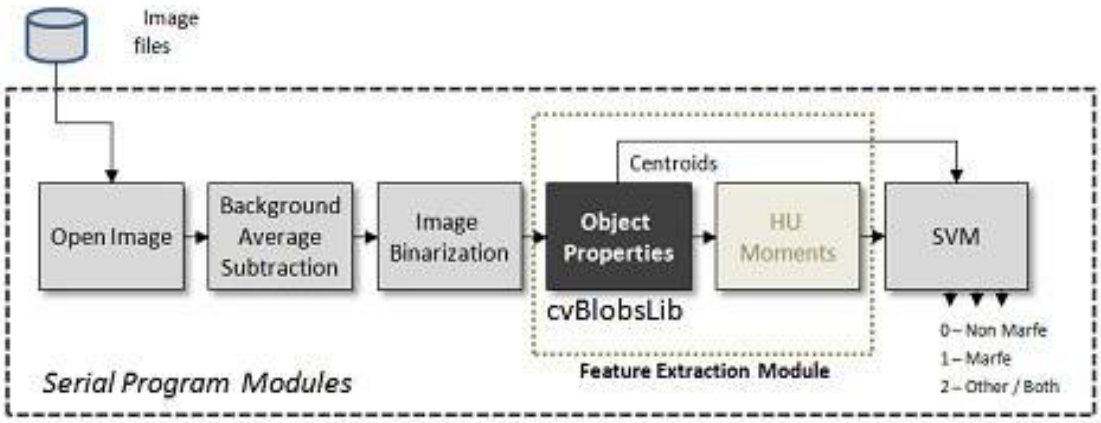



**Figura 2-16 – Diagrama em blocos da versão serial do algoritmo de detecção de Marfes, Fonte: (SOUZA, 2013)**

Este algoritmo avalia uma série de imagens previamente obtidas de um reator Tokamak e classifica os eventos em três categorias: *MARFE*, Não-*MARFE* e Outros. A Tabela 2-1 descreve sucintamente o objetivo de cada um dos módulos que compõem o algoritmo apresentado.

As etapas que formam a subtração de imagem de fundo e binarização constituem o pré-processamento. Após o pré-processamento, o objetivo da etapa de extração de características é

obter informações úteis, normalmente na imagem binária. Existem basicamente duas classes de medidas são elas: (i) atributos da Imagem como um todo (*Field Features*), ex.: número de objetos, área total de objetos, etc; (ii) atributos de região (*Region Features*) que se referem aos objetos independentes, ex.: área, perímetro, forma, etc.

Uma das etapas intermediárias na extração de características é chamada *Labelização* ou Rotulação. Após a etapa de segmentação obtem-se uma imagem onde as regiões correspondentes aos "objetos" estão separadas daquelas correspondentes ao "fundo" da imagem. Neste ponto do sistema de processamento, as regiões de interesse estão contiguamente agrupadas por pixels que se tocam (*blobs*). O próximo passo é dar um rótulo (ou *label*) para cada um desses grupos de pixels.

Para realização da etapa de extração de características foi utiliza a biblioteca cvBlob[9], esta biblioteca rotula imagens binárias com duas funcionalidades básicas: (i) extrair regiões com conectividade[10] 8 na forma binária ou em tons de cinza (conhecidas como blobs); (ii) filtar os blobs obtidos para posteriormente obter somente as características destes objetos.

Após a extração de características é realizada a etapa de classificação, esta foi baseada na técnica de vetores de suporte (SVM) (CRISTIANINI; TAYLOR, 2000). Nesta etapa é feita a classificação das regiões bnárias em três possíveis classes: *MARFE*, Não-*MARFE* e Outros. Em sistemas de SVMs existem duas fases. O primeiro é dedicado ao processo de formação que visa à definição de um modelo que contém as informações dos vetores de suporte. A segunda é o reconhecimento de padrões em si, que consiste em apresentar para o classificador as características obtidas pelos módulos anteriores da sequência de processamento de imagem.

| **Módulo** | **Objetivo** |
|---|---|
| *Open Image* | Inicia um arquivo de imagem. |
| *Background Average Subtraction* | Subtrai a média de N imagens anteriores, a partir da imagem atual (Subtração Imagem de Fundo). |
| *Image Binarization* | Segmentação por *Thresholding* da imagem obtida após a subtração. |
| *Feature Extraction* | Extração de características dos objetos existentes na imagem binária. |
| *SVM* | Módulo classificador para cada região, tendo como base as características extraídas. |

**Tabela 2-1 – Descrição simplificada dos módulos que compõem o algoritmo de detecção de MARFES.**

[9] http://code.google.com/p/cvblob

[10] Conceito usado para estabelecer fronteiras de objetos e regiões em uma imagem. Dois pixels são conectados se são adjacentes e se seus níveis de cinza satisfazem a um critério específico de similaridade.

A Figura 2-17 mostra os tempos de execução para cada uma das etapas mensuradas individualmente para a versão serial do algoritmo de detecção de *MARFES*.

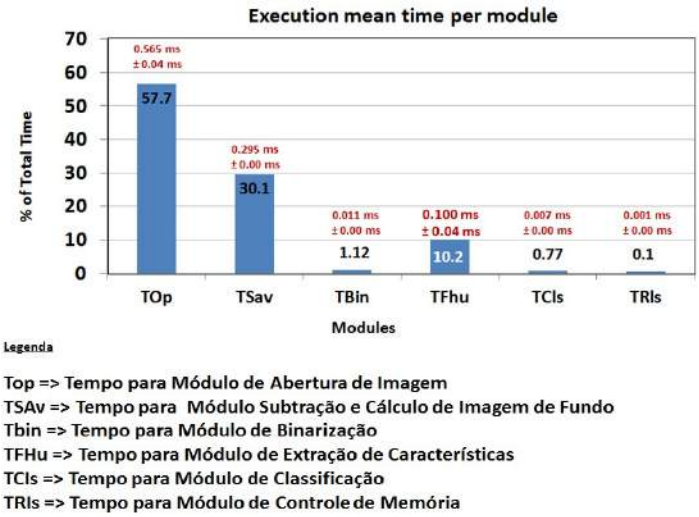



**Figura 2-17 – Tempos para execução de cada uma das etapas que formam o algoritmo de detecção serial de MARFES. Fonte: (SOUZA, 2013).**

## 2.3.2 Algoritmo paralelo

O desenvolvimento para execução em paralelo do algoritmo de detecção de *MARFES* se baseou nas técnicas de paralelismos de dados e *pipeline*, ambas descritas na Seção 2.2.2. Os módulos anteriormente desenvolvidos para o algoritmo serial, Figura 2-16, foram divididos em dois grupos, G1 e G2. O primeiro deles, constituído dos módulos de estimação e subtração de imagem de fundo e binarização. Já o segundo grupo constituído das etapas de extração de características e classificação. A Figura 2-18 mostra a divisão em grupos mencionada.

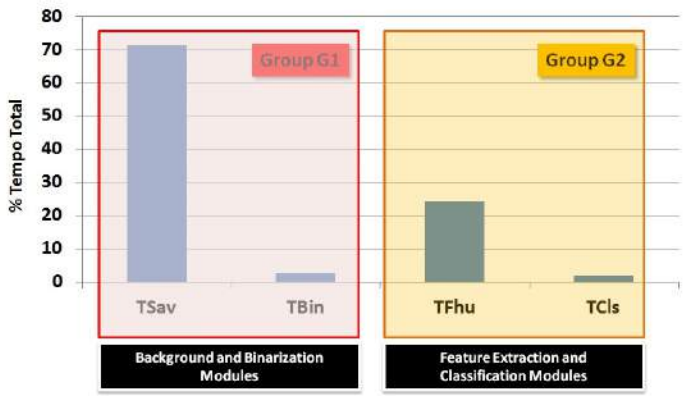



**Figura 2-18 – Divisão paralela primária em dois grupos de ações, responsáveis por diferentes etapas na detecção de *MARFES*. Fonte: (SOUZA, 2013)**

Para o primeiro grupo (G1) é utilizado o paralelismo de dados, com a divisão da imagem analisada em três partes, de forma que cada bloco da imagem é processado através de uma *thread* (SOUZA, 2013) diferente. A utilização de *threads* para o grupo G1 permitiu o aproveitamento de boa parte do código escrito anteriormente para este grupo em sua versão serial. A Figura 2-19 mostra a paralelização dos módulos do grupo G1 através da divisão da imagem em três blocos.

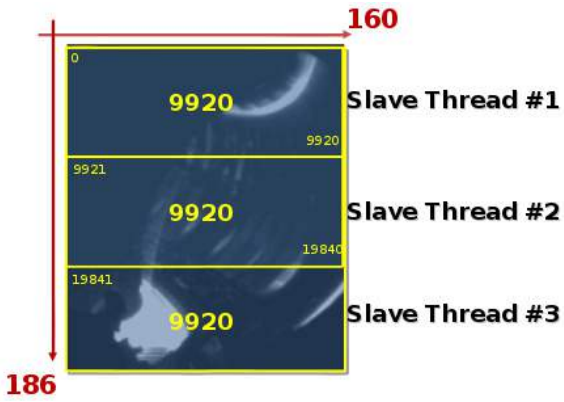



**Figura 2-19 – Paralelismo de Dados através da divisão da imagem em três grupos. Cada *thread* é responsável por executar as funções que compõem o grupo G1 para cada um dos blocos da imagem. Fonte: (SOUZA, 2013).**

Para as tarefas que compõem o segundo grupo (G2) é utilizada a técnica de *pipeline,* por meio de *forks* (SOUZA, 2013). Assim, também foi possível aproveitar boa parte do código escrito anteriormente para este grupo em sua versão serial. Em sistemas GNU/Linux e nas variantes do Unix, há a possibilidade de um processo[11] ser clonado, com a função *fork*. Basicamento uma chamada a instrução *fork* dentro de um programa cria uma cópia exata a partir do ponto que a instrução foi inserida e dois processos idênticos são executados simultaneamente. O processo principal, o qual chamou a instrução *fork*, é denominado processo pai e os processos que são gerados pelo pai são chamados de processos filhos. Os processos filhos ao serem criados ganham um novo espaço de memória onde terão variáveis globais diferentes das variáveis do processo pai, ainda que possuam os mesmos nomes na programação. Além disto, todos os processos podem ter acesso a uma região de memória compartilhada especialmente criada onde poderão compartilhar (ler, escrever e editar) informações.

A Figura 2-20 mostra o diagrama de tempo para a versão paralela do algoritmo de detecção de *MARFES* no processamento de três imagens (552, 553 e 554). Para obtenção destes tempos foi utilizada uma plataforma computacional com oito núcleos, onde estes foram divididos pelos processos dos grupos G1 e G2. O grupo G1 utiliza três processadores (*P#1-3*), cada uma responsável por uma *thread*. Já o grupo G2 é processado através de *forks*, utilizando para isto, os processadores

[11] Processo é um programa em execução o qual contém um fluxo ordenado de execução em um segmento de memória, possuindo suas próprias variáveis em memória e um identificador único.

*P#4-6*. O processador *P#0* controla todos os processos de sincronização de atividades dos outros processadores (*P#1-6*). O processador *P#7* é livre para trabalhar em outras tarefas do Sistema Operacional. Como as tarefas do grupo G2 são independentes entre si e dependentes do término das tarefas do grupo G1, os blocos de tempo foram distribuídos de modo a ter um núcleo do grupo G2 disponível ao final de cada conjunto de execução do grupo G1, ocupando os núcleos (G1 + G2) quase em tempo integral em uma arquitetura do tipo *pipeline* entre estes dois grupos.

As tarefas que compõem o grupo G1 são executadas em um tempo de 84 $\mu s$ representados pela somatória dos tempos de estimação e subtração de imagem de fundo (23 $\mu s$ ), segmentação por binarização (32 $\mu s$ ) e transferência da imagem binarizada para o grupo G2 (29 $\mu s$ ). As tarefas que compõem o grupo G2 são executadas com um tempo de 156 $\mu s$ .

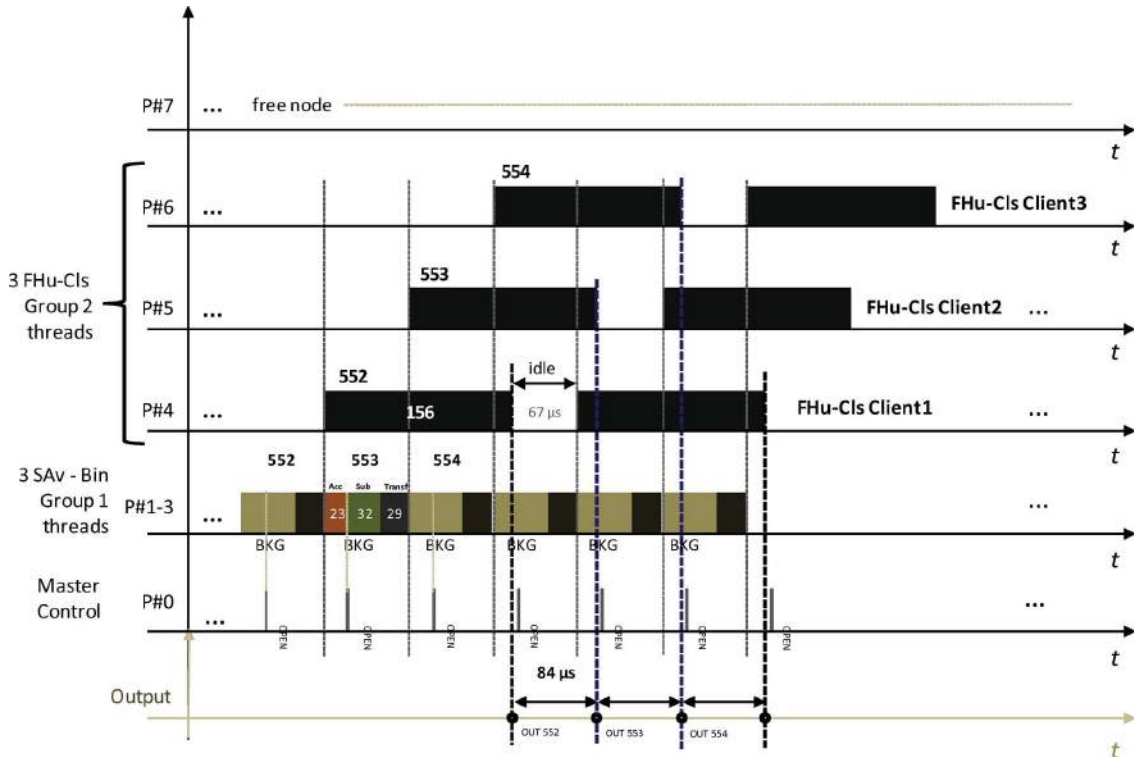



**Figura 2-20 – Diagrama de tempo da versão paralela do algoritmo de detecção de MARFES, para uma sequência de três imagens (552, 553 e 554). Os três números dentro da linha de tempo do grupo G1 (23, 32 e 29 $\mu s$ ) mostram o tempo gasto para estimação e subtração de imagem de fundo, binarização e transferência da imagem gerada pelo grupo G1 para o grupo G2. Fonte: (ALBUQUERQUE *et al.*, 2013)**

A execução em tempo real do algoritmo paralelo foi promovida por meio da elevação de privilégios (prioridade) dos processos, configurando-o como RT na fila de prioridades do sistema operacional. O direcionamento de cada processo para uma CPU específica foi feita pelas

determinações de afinidade[12] com as unidades de processamento, garantindo assim que cada destas ficasse responsável por seu processo específico.

Mais informações sobre as abordagens serial e paralelo do algoritmo de detecção de *MARFE* podem ser encontradas em (CHACON, 2012) e (SOUZA, 2013) respectivamente. Como o presente trabalho se dedica a inserção em FPGA de parte do algoritmo de detecção de *MARFE* apresentado, resultados como o desempenho do classificador para cada uma das versões desenvolvidas (algoritmo serial e algoritmo paralelo), foram aqui omitidos, no entanto, também podem ser encontrados nas literaturas anteriormente mencionadas.

O presente trabalho tem então como proposta a inclusão em FPGA das funções que correspondem ao grupo G1 anteriormente mencionado, Figura 2-21. Estas tarefas que compõem a etapa de pré-processamento são facilmente paralelizáveis, através da técnica de paralelismo de dados e podem explorar a natureza de execução paralela dos FPGAs. Além disto, como a execução é feita diretamente em *hardware* através de blocos lógicos, garante-se a previsibilidade e confiabilidade de execução, características essenciais para uma operação em tempo real.

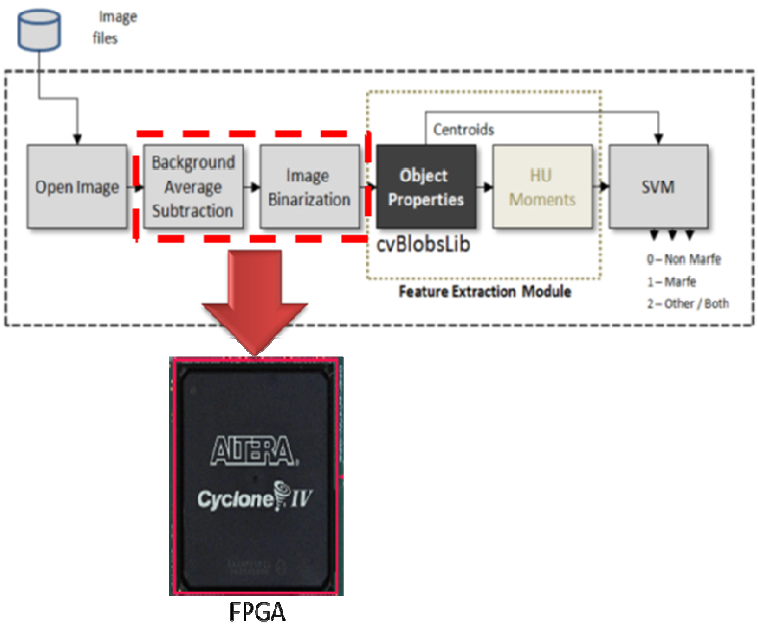



**Figura 2-21 – Etapas do algoritmo de processamento a serem inseridas em FPGA, para execução diretamente em hardware.**

[12] Configurar a afinidade com o processador significa que um aplicativo será executado apenas em determinado núcleo da unidade de processamento.

# Capítulo 3

# Tecnologias e ferramentas utilizadas no desenvolvimento do projeto

A inserção de algoritmos de processamento de imagens em FPGA passa pelo entendimento teórico do funcionamento do mesmo. Assim como é necessário explorar alguns conceitos relacionados com o barramento PCIe, utilizado para transferência das imagens entre o computador de desenvolvimento e o *hardware* configurável.

Para desenvolvimento do presente trabalho, ambientes de *software* foram utilizados tanto para programação do FPGA, quanto para criação de uma interface executada no computador de desenvolvimento. Esta interface é responsável pela seleção e transferência das imagens, pelo monitoramento da memória de armazenamento das mesmas no *hardware* configurável, e também pela execução em *software* dos mesmos algoritmos de imagens embarcados em *hardware.*

## 3.1 Conceitos teóricos sobre FPGA

A adoção de FPGA (*Field-programmable gate array*) referenciada muitas vezes como *hardware* configurável cresce a cada ano, e o mercado mundial para esta tecnologia atingiu 2,75 bilhões de dólares em 2010 (NATIONAL INSTRUMENTS, 2011). Desde sua invenção em meados dos anos 1980, os FPGAs passaram de um simples chip de lógica personalizada até realmente substituir circuitos integrados de aplicações específicas. Os dois maiores fabricantes para este tipo de tecnologia em termos de mercado são Xilinx e Altera Corporation. A Figura 3-1 mostra diferentes modelos de FPGAs para estes dois fabricantes.


**Figura 3-1 – Chips FPGA modelos Virtex-5 e Stratix V respectivamente dos fabricantes Xilinx e Altera Corporation**

### 3.1.1 Considerações gerais sobre o *hardware*

A ideia chave que caracteriza os FPGAs é a existência de um circuito eletrônico genérico que possui sua funcionalidade programada de acordo com a necessidade de dada aplicação, podendo esta ser desenvolvida como um conjunto de sistemas independentes e executados de forma paralela. Os computadores convencionais são baseados também nesta idéia, onde a ULA realiza uma data operação, de acordo com o que for selecionado em suas portas de controle. No entanto, uma ULA está limitada no ponto de poder realizar apenas uma operação por vez e de possuir uma arquitetura fixa de *hardware*, i.e., a unidade de controle presente na UCP (Unidade Central de Processamento) de um computador, através dos processos de busca e execução de instruções, pode solicitar que a ULA execute apenas as operações aritméticas e lógicas previamente definidas em sua arquitetura de *hardware.*

Nos FPGAs, diferentes operações não têm que competir pelos mesmos recursos. Cada tarefa de processamento é enviada para uma seção dedicada do chip e pode funcionar de forma autônoma sem nenhuma influência de outros blocos lógicos. Como resultado, o desempenho de uma parte da aplicação não é afetado quando mais tarefas são adicionadas. A Figura 3-2 mostra um esquemático, onde diferentes funções lógicas são distribuídas ao longo de diferentes partes de um chip de *hardware* configurável, evitando assim a concorrência de recursos.

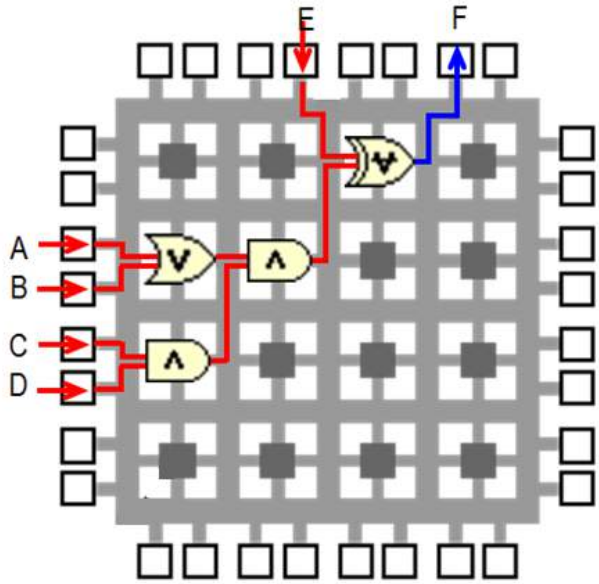



**Figura 3-2 – Funcionalidades lógicas distribuídas em diferentes partes do chip FPGA.**

Todo chip FPGA é constituído de um número finito de recursos de *hardware* pré-definidos, com interconexões programáveis por *software* e realizadas em *hardware*. Estes chips também possuem blocos de E/S que permitem que o circuito acesse o meio externo. A Figura 3-3 mostra as diferentes partes de um FPGA.

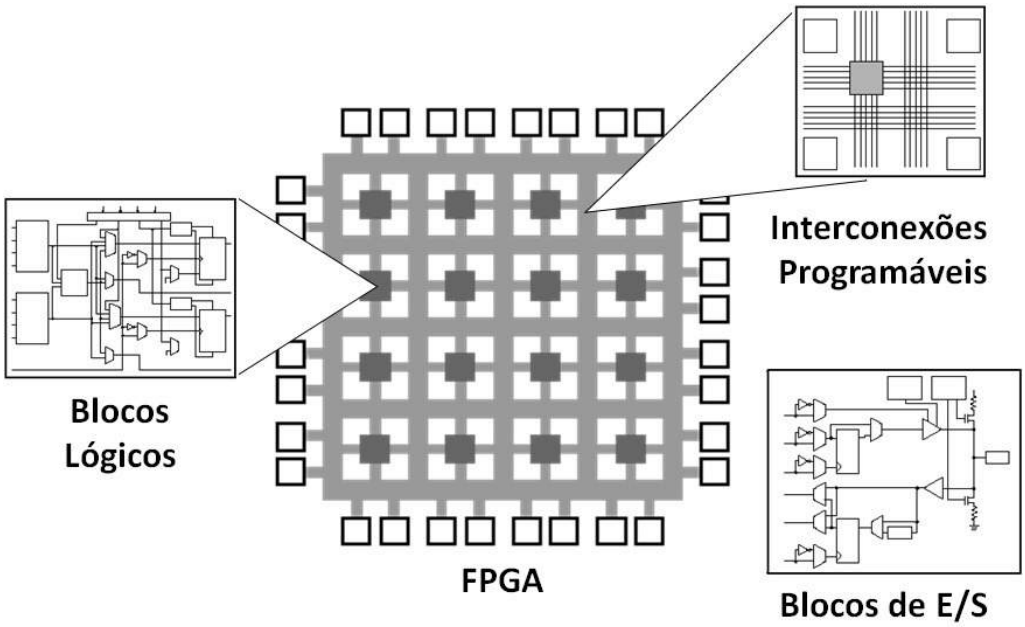
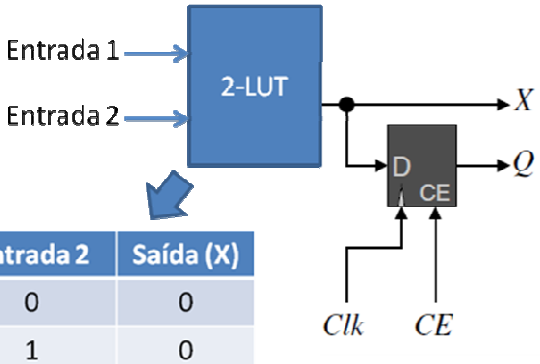



**Figura 3-3 – Blocos constituintes de um chip FPGA**

Observando a Figura 3-3, observa-se que um circuito reconfigurável é constituído de três elementos básicos: Blocos Lógicos, Blocos de E/S e Interconexões Programáveis.

## Blocos Lógicos

Os Blocos Lógicos (CBLs) formam as unidades lógicas básicas de qualquer chip FPGA. Algumas vezes são referidos também como células lógicas. Os CBLs são formados de dois componentes básicos, os *Flip-Flops* e *Lookup Tables* (LUTs). As diferentes famílias de FPGAs de diferentes fabricantes se diferem na forma como estes dois elementos são agrupados. As LUTs são constituídas por elementos de memória RAM, onde é possível determinar tabelas verdade para criação de lógica combinacional que responde de acordo com suas entradas. Os *Flip-Flops*, por sua vez, são responsáveis por registrar as saídas das LUTs, a fim de possibilitar um sincronismo entre os diversos blocos lógicos que compõem a lógica embarcada em *hardware*, permitindo a construção de máquinas de estados, contadores e pequenas lógicas registradas ideiais para *pipeline*. A Figura 3-4 mostra um exemplo de uma LUT de duas entradas, referenciada como 2-LUT, junto com sua tabela verdade, programada para realizar a operação de AND lógico. É possível verificar também seu respectivo registrador (*Flip-Flop*) na saída.




| Entrada 1 | Entrada 2 | Saída (X) |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**Figura 3-4 – LUT de duas entradas programada para realizar a operação de *AND* lógico.**

LUTs podem realizar qualquer função digital, combinacional ou lógica. Desenvolvimentos mais complexos envolvem o sequênciamento de múltiplos níveis de LUTs, conectando a saída de um bloco lógico à entrada do bloco subsequente. Fabricantes de FPGA enfrentam a questão de definir o tamanho ideal dos blocos lógicos, uma vez que a utilização de blocos lógicos complexos, para desenvolvimento de lógica simples, leva ao desperdício de recursos. Por outro lado, a definição de lógicas complexas a partir de blocos lógicos simples aumenta o atraso de propagação dos sinais entre os diferentes níveis de CBLs que devem ser conectados.

## Interconexões Programáveis

O segundo constituinte básico de um FPGA são as interconexões programáveis. Estas permitem que diferentes blocos lógicos sejam conectados para criação da funcionalidade desejada. Obviamente é impossível prover uma conexão direta e dedicada entre todas as entradas e saídas de CBLs existentes em um chip FPGA. Para qualquer aplicação, apenas uma pequena parcela destas conexões serão realmente requeridas. Assim, a abordagem utilizada é a existência de um pequeno conjunto de recursos de roteamento compartilhados que podem ser utilizados para criar as conexões desejadas, um pouco semelhante ao realizado em uma rede telefônica.

A rede de interconexões para a maioria das FPGAs é baseado em uma estrutura matricial de conexões, conforme visto na Figura 3-5. Em cada ponto de interseção existem chaves que permitem criar as conexões entre as linhas de roteamento vertical e horizontal. O chaveamento correto permite criar lógicas complexas.

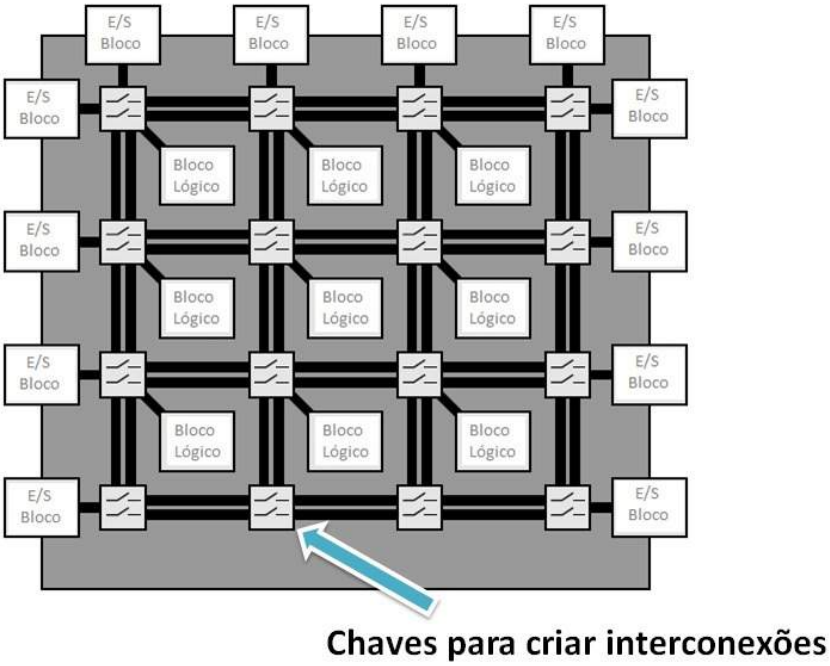



**Figura 3-5 – Matriz de conexões entre os blocos lógicos em um chip FPGA.**

Cada uma das chaves de conexão é realizada através de um transistor NMOS (MOSFET tipo N) (BROWN; VRANESIC, 2008). Quando o terminal de *Gate* deste transistor recebe lógica zero, é aberta a conexão entre as linhas horizontal e vertical, quando o *Gate* recebe lógica um, é fechada a conexão. A Figura 3-6 mostra a utilização dos transistores para o chaveamento entre as linhas de roteamento vertical e horizontal no FPGA. Neste caso, a saída do CBL está conectada na primeira linha vertical de roteamento.

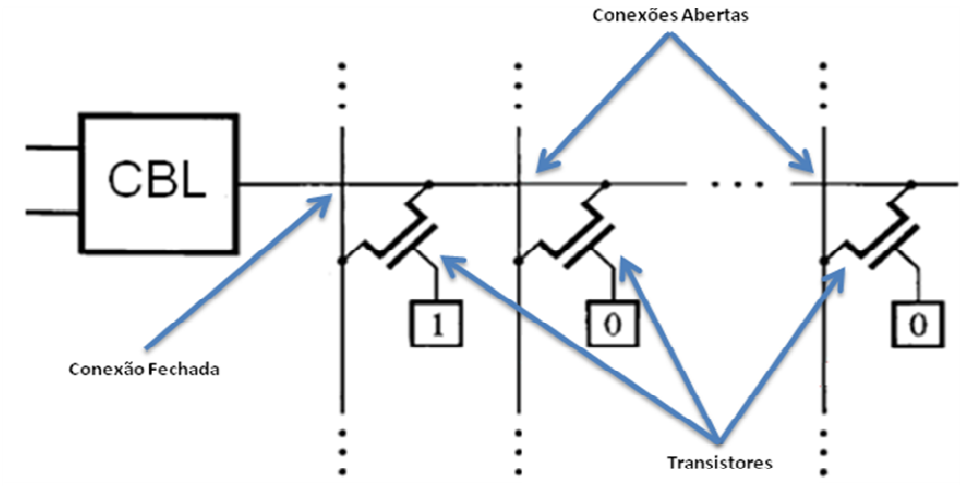



**Figura 3-6 – Transistores NMOS para criar conexões entre blocos lógicos de um FPGA**

É importante que apenas uma saída seja direcionada para uma linha de roteamento. Caso duas saídas com níveis diferentes, por exemplo, alto e baixo sejam conectadas juntas, ocorrerá um curto circuito, podendo danificar o chip FPGA. Além disto, a cada chave de conexão que um determinado sinal é roteado, o atraso de propagação sofre um acréscimo.

## Blocos de E/S

O último constituinte básico que completa o *hardware* de um FPGA são os blocos de entrada/saída (E/S), que permitem a conexão de sinais entre o *hardware* configurável o os dispositivos externos. A maioria dos pinos de interface pode ser configurada com entrada, saída ou ambos e podem ser conectados diretamente à lógica interna programada no FPGA. Esta gama de funcionalidades pode ser realizada por um bloco genérico de E/S, conforme visto na Figura 3-7. A complexidade dos blocos aumenta devido a grande quantidade de diferentes padrões de sinais utilizados pelos diferentes dispositivos externos. Os padrões mais comuns são o LVTTL (Low Voltage Transistor-Transistor Logic) e LVCMOS (Low Voltage CMOS).

Quando a comunicação com dispositivos externos requer a utilização de protocolos de comunicação de alto nível, uma grande quantidade de recursos de *hardware* será necessária para gerenciar o protocolo. Nestes casos, chips de interface dedicados podem ser utilizados para a comunicação entre FPGA e os dispositivos externos. No entanto, para alguns protocolos padrão como PCIe e Gigabit Ethernet, certos FPGAs contêm estas interfaces diretamente em suas estruturas de *hardware*.

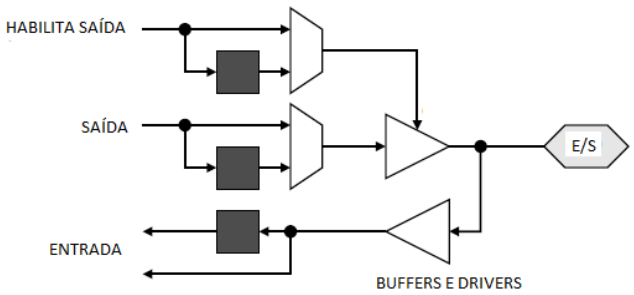



**Figura 3-7 – Bloco de E/S genérico. Funcionalidade de entrada ou saída pode ser definida pela porta de controle**

## Outras considerações do *hardware*

Muitos dos primeiros FPGAs contavam com uma arquitetura homogênea, onde apenas estavam presentes os blocos básicos anteriormente mencionados. No entanto, com o desenvolvimento da eletrônica, os FPGAs passaram a uma arquitetura heterogênea, onde existe uma quantidade de blocos básicos, mas também blocos de funções específicas, como por exemplo,

somadores, PLLs, multiplicadores, memórias RAM e FIFO. Existem famílias de FPGA que chegam a possuir processadores seriais de alto desempenho construídos como blocos de *hardware*. Esta arquitetura heterogênea permite redução de tempo do projeto com os FPGAs.

### 3.1.2 Considerações gerais sobre a programação e configuração

Todas as chaves de interconexão, estruturas e opções dentro de um FPGA são controladas através de bits de programação. Cada interseção vista na Figura 3-5 representa uma conexão programável. Os primeiros dispositivos de lógica programável em 1970 eram programados através de uma configuração manual em uma grade de bits 0s e 1s indicando os estados de chave aberta ou chave fechada para cada uma das interseções (TOCCI; WIDMER; MOSS, 2011).

O advento das linguagens descritivas de *hardware* permitiu que os programadores de dispositivos de lógica configurável utilizassem um computador para geração dos padrões de bits usados para programar os dispositivos de *hardware* reais. O VHDL (*Very High Speed Integrated Circuit Hardware Description Language*) (ASHENDEN, 2008), que tem suas raízes no departamento de defesa Norte Americano, se tornou uma das principais linguagens de descrição de *hardware* de alto nível para projetar e desenvolver circuitos digitais, tendo sido padronizada pelo IEEE em 1897. A primeira versão padronizada da linguagem é muitas vezes referenciada como VDHL-87. Como todos os padrões do IEEE, o padrão do VHDL está sujeito a revisões de tempos em tempos, sendo que a versão mais recente é referenciada por IEEE 1076-2008.

O VHDL foi pensado com objetivo de suprir diversas necessidades no processo de desenvolvimento de sistemas digitais: (i) permite descrever a estrutura de um sistema digital, a partir diferentes subsistemas conectados; (ii) permite especificar a funcionalidade de um sistema digital através de uma linguagem de programação com aspectos conhecidos; (iii) permite que o projeto para o sistema digital seja simulado em ambiente de *software* antes do real desenvolvimento em *hardware,* reduzindo gastos e tempo de desenvolvimento.

A linguagem de descrição de *hardware* fornece então um modo conciso e conveniente para se descrever a operação do circuito em um formato que um computador pessoal possa manejar e armazenar adequadamente. O computador roda um sintentizador para traduzir a linguagem de descrição de *hardware* para grade de bits 0s e 1s a ser carregada no FPGA.

Os FPGAs são configurados através da transferência do arquivo de configuração para sua memória interna. Esta memória é estática e volátil, perdendo seu conteúdo quando o FPGA é desligado. Em aplicações embarcadas, o arquivo de configuração está presente em uma memória flash externa, conforme visto na Figura 3-8, e é carregado para memória interna do FPGA, permitindo assim, configuração do *hardware*, quando for iniciado. O FPGA pode ser programado serialmente através de uma interface *JTAG* (*Joint Test Action Group*) (SOUZA, 2012), sendo esta comumente utilizada quando o FPGA é programado a partir de um computador.

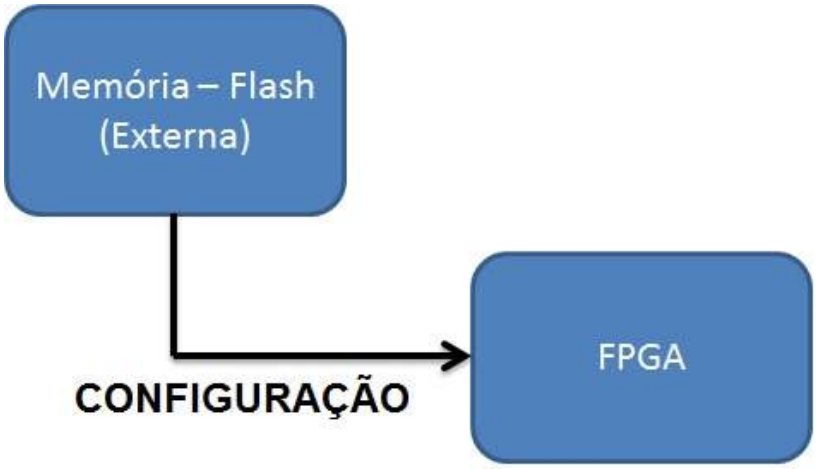



**Figura 3-8 – Configuração do FPGA para ambiente embarcado. Necessária memória Flash externa para armazenar arquivo de configuração do hardware**

Para o carregamento de nova funcionalidade para o *hardware* configurável. O arquivo de configuração deve ser transferido para memória interna do FPGA e o mesmo deve ser reiniciado. Nenhuma informação interna é mantida de uma configuração para outra – todos os *flip-flops* e memórias são reiniciados, de acordo com a nova configuração carregada.

### 3.1.3 FPGAs e processamento de imagens

Uma vez que nos FPGAs a lógica requerida para uma aplicação é construída com uma parte de *hardware* específico para cada função, FPGAs são naturalmente paralelos. Isto permite alta velocidade de execução, típicas de projetos em *hardware,* enquanto mantém a flexibilidade da programação de sua funcionalidade através de *software*. Estas características fazem dos FPGAs indicados para processamento de imagens, particularmente nas primeiras etapas da cadeia de processamento, onde é possível explorar o paralelismo intrínseco das imagens.

O paralelismo de dados pode ser explorado construindo múltiplas cópias do *hardware* de processamento e associando estes a diferentes partes de uma imagem de entrada. A Figura 3-9 mostra o conceito da aplicação do paralelismo de dados em *hardware* reconfigurável. As operações são executadas de forma paralela e independente em diferentes fatias do chip FPGA. Atualmente, o paralelismo massivo de dados, i.e., divisão da imagem digital em sua unidade básica (*pixel*), não pode ser desenvolvido na prática simplesmente porque o número de *pixels* de uma imagem excede a quantidade de recursos disponíveis no FPGA (BAILEY, 2011).

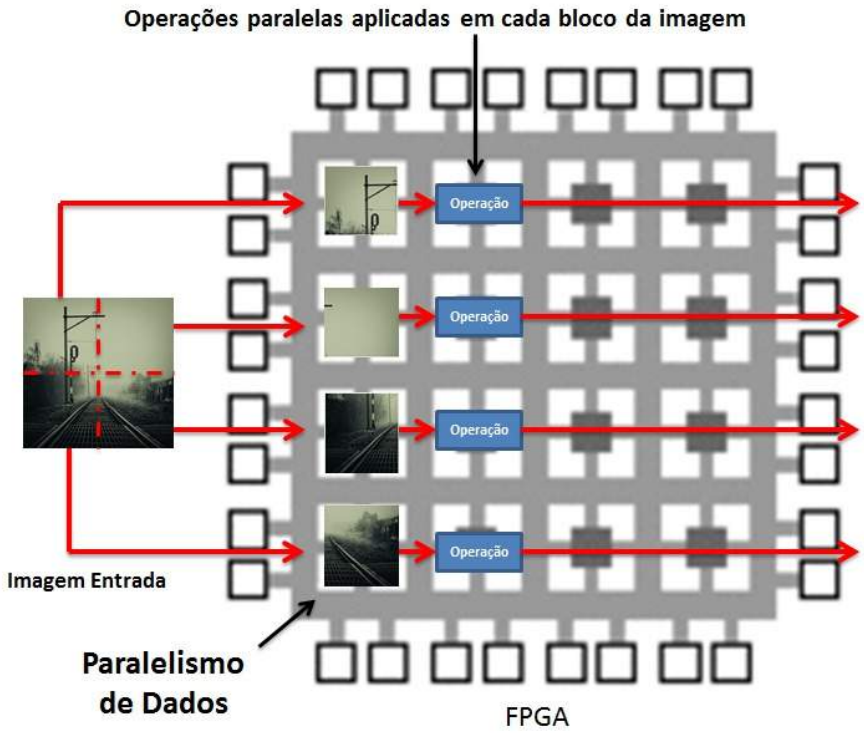



**Figura 3-9 – Paralelismo de Dados aplicado a uma imagem. Cada operação é executada de forma paralela e independente sobre cada um dos blocos da imagem**

## 3.2 Conceitos teóricos sobre barramento PCIe

O PCI *Express*, abreviado como PCIe, é um barramento serial de alta velocidade desenvolvido em 2004 pela Intel, Dell, HP e IBM, para substituir o antigos barramentos PCI e AGP. O PCIe tem várias melhorias quando comparado com padrões antigos, incluindo aumento na velocidade de transmissão de dados, menor quantidade de pinos de interface e melhores mecanismos de detecção de erro.

O PCIe é comumente usado para realizar interface e transferência de dados rápida e de alto desempenho entre aplicações de controle executadas em computadores e os FPGAs.

Dois dispositivos PCIe se comunicam através de uma conexão ponto a ponto denominada *link*. Em termos físicos esta conexão é constituída de um ou mais *lanes*. Um *lane* é constituído de dois pares de sinais diferencias: um para recepção e outro para envio de dados. Cada *lane* transporta pacotes entre os dois pontos finais do *link* em ambas direções. Estes pacotes são recebidos e enviados serialmente, e distribuídos através dos *lanes* disponíveis no link.

Os dispositivos PCIe podem conter de 1 a 32 *lanes*, em potência de dois (1, 2, 4, 8, 16, 32). A especificação dos dispositivos PCIe trás a quantidade de *lanes* de que é formado, com o prefixo *x.* Por exemplo, o prefixo *x16* indica um dispositivo PCIe com 16 *lanes*. A Figura 3-10 mostra o formato dos slots PCIe para dispositivos com 1, 4, 8 e 16 *lanes*.

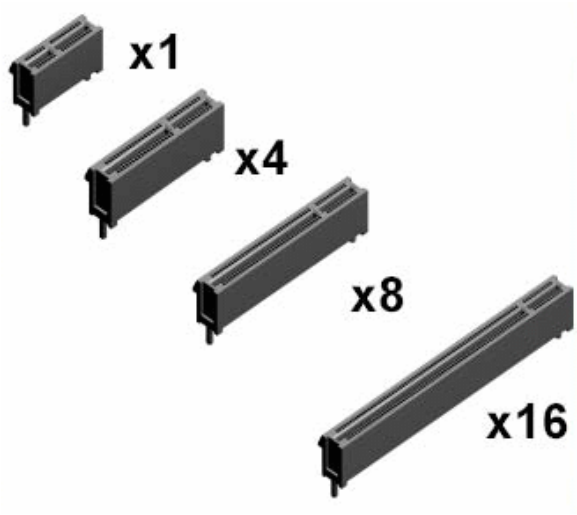



**Figura 3-10 – Slots PCIe para dispositivos com 1, 4, 8 e 16 *lanes* respectivamente**

O padrão PCIe 1.0 pode chegar a velocidades de transmissão de dados de até 2.5Gbits/s por *lane* por direção. No entanto, devido a codificação 8b/10b[13] necessária para transmissão do sinal de *clock* embarcado nos pacotes de dados, a taxa efetiva de transferência de dados chega até 250M*Bytes*/s por *lane* por direção.

A arquitetura para envio e recepção dos pacotes de dados no protocolo PCIe é baseada na operação de três camadas*, Transaction Layer* (TL), *Data Link Layer* (DDL) *e Physical Layer* (PL).

O PCIe através de transações de escrita e leitura, utiliza espaços de endereços relacionados à memória e configuração para trocar informações entre dois dispositivos. Transações podem ser definidas como uma sequência de um ou mais pacotes requeridos para completar a transferência de informações.

### 3.2.1 Arquitetura em Camadas do Protocolo PCIe

A especificação do PCIe define o funcionamento do barramento a partir de uma arquitetura em três camadas, conforme visto na Figura 3-11.

[13] Codificação 8b/10b é um algoritmo para codificar cada *byte* de dados em um caractere de 10 bits. Utilizada em transmissão serial de alta velocidade, os dados são codificados no transmissor e decodificados no receptor. O processo de codificação, além de outros benefícios, garante que informação suficiente de *clock* estará presente no conjunto de dados transmitido, permitindo assim sincronizar o receptor.

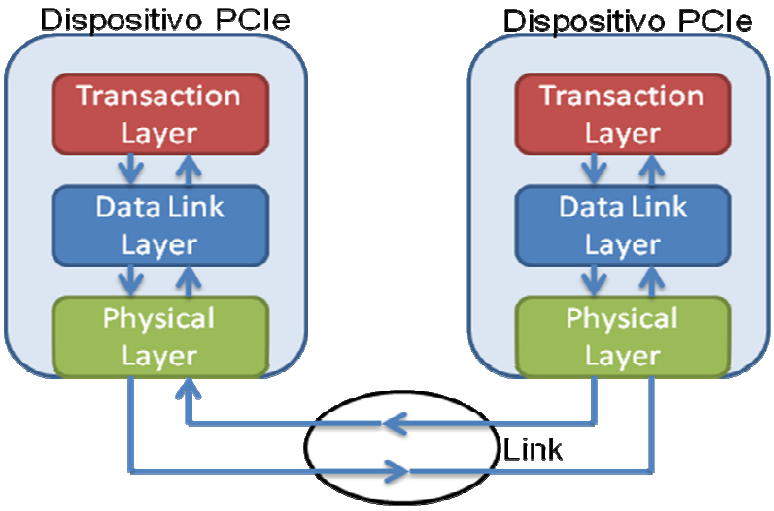



**Figura 3-11 – Estrutura em três camadas do protocolo PCIe. Cada camada é responsável por inserir/retirar informações para compor um pacote de transmissão**

*A Transaction Layer* (TL) é a primeira camada do protocolo PCIe. A TL é responsável por iniciar o processo de transmissão de dados, transformando uma requisição de um dispositivo em um TLP (*Transaction Layer Packet*). Um TLP é um pacote de informações composto por um cabeçalho para identificar o tipo de transação, e opcionalmente pelo *payload*[14] (XILINX , 2008). A Figura 3-12 mostra a estrutura de um TLP.

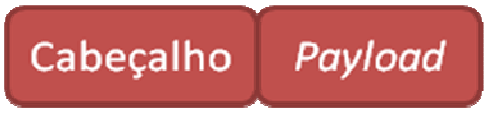



**Figura 3-12 – Estrutura de um pacote TLP gerado na camada TL do protocolo PCIe**

O TLP é enviado da TL de um dispositivo transmissor para a TL de um dispositivo receptor. Por exemplo, em uma requisição de leitura de dados na memória, a TL transmissora codifica esta requisição em um pacote TLP e o repassa através do *link*. A TL do dispositivo receptor decodifica a requisição e repassa para o dispositivo para ser processada. Quando os dados solicitados são disponibilizados, a TL receptora organiza-os em um TLP de resposta e este é encaminhado através do *link* para a TL solicitante. Cada TLP tem uma identificação individual que permite que os dados sejam direcionados para o solicitante correto.

[14] *Payload* representa a informação efetiva a ser trocada entre dois dispositivos PCIe. A especificação do protocolo PCIe permite *payload* de até 4096 bytes em um TLP. Todos os outros dados inseridos pelas diferentes camadas que compõem o protocolo PCIe são utilizados apenas para o correto *"handshake"* do protocolo.

A *Data Link Layer* (DLL) é a camada intermediária que compõe a arquitetura do protocolo PCIe. O principal objetivo desta camada é garantir a integridade dos dados que circulam através do *link*. A DLL adiciona uma sequência numérica no início e uma codificação, através da técnica de CRC, para verificação de erro sobre o pacote gerado pela TL. A Figura 3-13 mostra a estrutura do pacote gerado pela DLL.


**Figura 3-13 – Pacote gerado pela DLL. Pacote inicial gerado pela TL mais inclusão de mecanismos de confiabilidade de transmissão**

A *Physical Layer* (PL) é a última camada da arquitetura do protocolo PCIe, sendo responsável pelo envio efetivo dos pacotes de dados para o *link*. Esta camada interage com a DLL e com *link* físico PCIe, assim possui circuitos como casadores de impedância*, buffers* e PLLs que permitem a operação física da interface.

No bloco transmissor, a PL recebe a informação da DLL e a converte em um formato serial, no qual é inserido o sinal de *clock*, através da codificação 8b/10b. Também é adicionado um conjunto de caracteres em uma configuração que indica o início e fim do pacote. A Figura 3-14 mostra a estrutura do pacote completo PCIe.


**Figura 3-14 – Pacote PCIe completo. Contém informações inseridas pelas três camadas da arquitetura do protocolo PCIe**

Durante a transmissão, a PL divide os dados em *bytes* através dos *lanes*. Este processo de divisão dos dados é transparente para as demais camadas do protocolo. Durante a inicialização, cada *link* PCIe se ajusta de maneira a buscar uma melhor configuração de velocidade de operação entre os dispositivos localizados nas extremidades do *link*. Nenhum *firmware* ou sistema operacional são envolvidos neste processo. No dispositivo receptor, a PL recebe os dados seriais do *link* PCIe, os transforma novamente em pacotes de dados e os repassa para DLL receptora.

Conforme observado anteriormente, para a correta operação do protocolo PCIe é necessário a inclusão de diversas informações através de suas camadas de operação. Estas informações representam a redução da taxa efetiva de transmissão de dados, conforme pode ser visto em (XILINX , 2008) onde são realizados vários cálculos para determinação do desempenho teórico de um

barramento PCIe levando em consideração as características de *overhead* introduzidas pelo protocolo. Neste artigo alcança-se uma taxa de 200*MBytes/s por lane* para a Gen1.0[15] do protocolo PCIe, quando considerado apenas o barramento de transmissão. Este valor é esperado já que a banda teórica de 250M*Bytes*/s deve ser utilizada não só para o transporte de dados, mas também para as outras informações que permitem a operação do protocolo.

### 3.2.2 Espaços de Endereços do Protocolo PCIe

Dispositivos que se comunicam através do protocolo PCIe necessitam acessar a memória compartilhada existente entre eles. Por exemplo, quando um computador se comunica com um dispositivo PCIe, esta memória é utilizada pelos *drivers* para o controle e troca de informações entre o computador e o dispositivo PCIe.

Os sistemas de memória dos computadores podem ser utilizados para compartilhar espaço de troca de dados com os dispositivos PCIe, no entanto, a cada acesso deste à memória, o computador deve aguardar a finalização do acesso. Esta abordagem pode gerar lentidão no sistema, não sendo uma boa pr, permitir que os dispositivos PCIe acessem a memória principal do computador, de uma maneira desordenada.

Assim, os dispositivos PCIe têm seus próprios espaços de memória, que podem ser mapeados e acessados, através de *drivers*, por um computador para troca de informações. Além dos espaços utilizados para troca de informações, existe também um espaço destinado à configuração, o qual é utilizado na etapa de inicialização do dispositivo PCIe. A Figura 3-15 mostra o espaço de configuração do protocolo PCIe e seus principais registros.

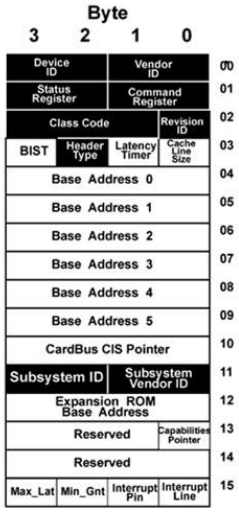



**Figura 3-15 – Espaço de Configuração para comunicação direta entre dois dispositivos com protocolo PCIe.**

[15] Na especificação do barramento PCIe 1.0, denominada Gen1, a velocidade de transmissão é de 250MB/s por canal de transferência (lane)

No escopo do presente trabalho, cabe destaque para os registros *Device ID* e *Vendor ID*, estes identificam o fabricante e o dispositivo PCIe. Estes registros são importantes para a criação dos *drivers* de acesso entre o computador e um dispositivo PCIe. Além destes, os registros *Base Address*, conhecidos como BARs, são responsáveis por determinar o tipo, a quantidade e os endereços de memória que serão utilizados pelo dispositivo PCIe para troca de informações.

(SHANLEY; ANDERSON; BUDRUK, 2003) pode ser consultado para mais informações sobre o funcionamento e configuração dos registros do protocolo PCIe.

## 3.3 Ambientes de *software* e *hardware*

Este trabalho apresenta o desenvolvimento, aplicação e caracterização de uma ferramenta para um problema de processamento de imagens, ainda nas etapas iniciais da pirâmide de processamento, a fim de se avaliar seu potencial de aplicação em sistemas de tempo real.

No desenvolvimento e caracterização tanto do processo de transferência de imagens entre um computador e a memória de um FPGA, quanto das etapas do algoritmo de processamento de imagens aplicadas em *hardware*, utilizou-se componentes de *hardware* e *software.* Estes permitiram, além da implementação do barramento de transmissão PCIe e dos algoritmos de processamento de imagens em *hardware*, aplicar o pré-processamento necessário às imagens antes da transferência destas para o FPGA.

### 3.3.1 Plataforma de desenvolvimento

O item de *hardware* utilizado foi a plataforma Altera Cyclone IV GX (ALTERA CORPORATION, 2010). Este possui um FPGA modelo EP4CGX150DF31, interfaces de memória e periféricos que permitem a comunicação com dispositivos externos a plataforma.

Para sistemas onde o FPGA é instalado como um módulo dentro de um computador padrão, este normalmente é utilizado para gerenciar a captura de imagens, o transporte destas ao FPGA, e posteriormente, permitir a visualização do processamento que foi realizado no *hardware* reconfigurável. Nestas situações, normalmente utiliza-se o barramento PCIe para a comunicação entre computador e FPGA (BAILEY, 2011). Assim, uma característica desta plataforma é a existência de uma porta de comunicação com barramento PCIe x4 1.0, a qual permite que ele seja diretamente conectado ao barramento PCIe da placa-mãe no computador de desenvolvimento. A Figura 3-16 mostra a plataforma e seus recursos disponíveis.

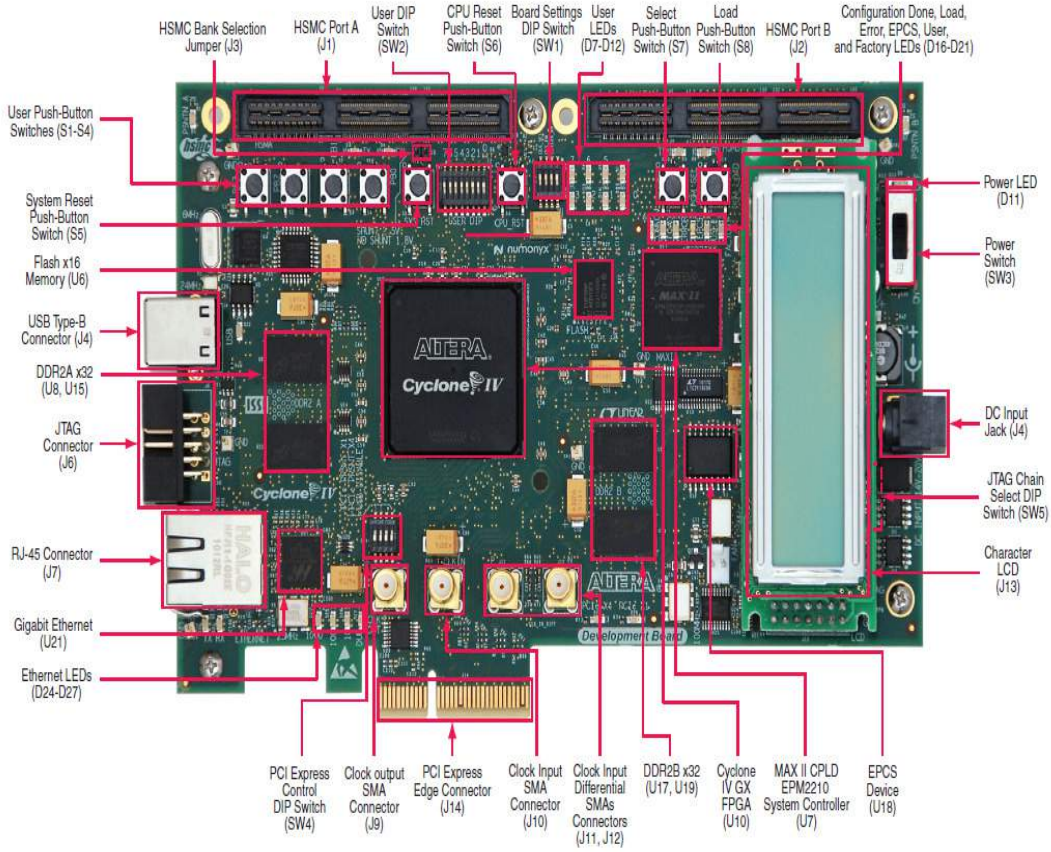



**Figura 3-16 – Plataforma de Desenvolvimento Altera Cyclone IV GX. Fonte: (ALTERA CORPORATION, 2010)**

## 3.3.2 Ferramenta Quartus II

Para o desenvolvimento de projetos com FPGA, a Altera disponibiliza a ferramenta de *software* Quartus II (ALTERA CORPORATION, 2012), a qual pode ser obtida gratuitamente de seu site, na versão *Web Edition.*

Com esta ferramenta é possível desenvolver os projetos em linguagem HDL, simular, compilar e através de uma ferramenta de programação, realizar a configuração do FPGA.

O ambiente de desenvolvimento do Quartus II permite a integração de arquivos provenientes de diferentes fontes para a criação da lógica a ser sintetizada no dispositivo. É possível inserir algoritmos criados diretamente em linguagem HDL ou, através da ferramenta *MegaWizard Plug-IN,* trabalhar com blocos lógicos previamente criados que representam funções como portas lógicas, somadores e memórias.

Um recurso importante que faz parte do ambiente Quartus II é a ferramenta de integração Qsys. Esta, através de um alto nível de abstração e automatização, cria as interconexões entre blocos

de funções pré-existentes (*IP blocks*), poupando o programador de realizar, através de linguagem HDL, todas as interfaces necessárias.

Um projeto dentro do ambiente Quartus II pode ser visualizado através de um arquivo bdf(*block design file*). Através deste, toda lógica é vista por meio de um diagrama em blocos, o que facilita a verificação das interconexões entre os mesmos e a inserção de mais blocos funcionais. A Figura 3-17 mostra o ambiente de desenvolvimento Quartus II, e a visualização dos componentes lógicos através de blocos.

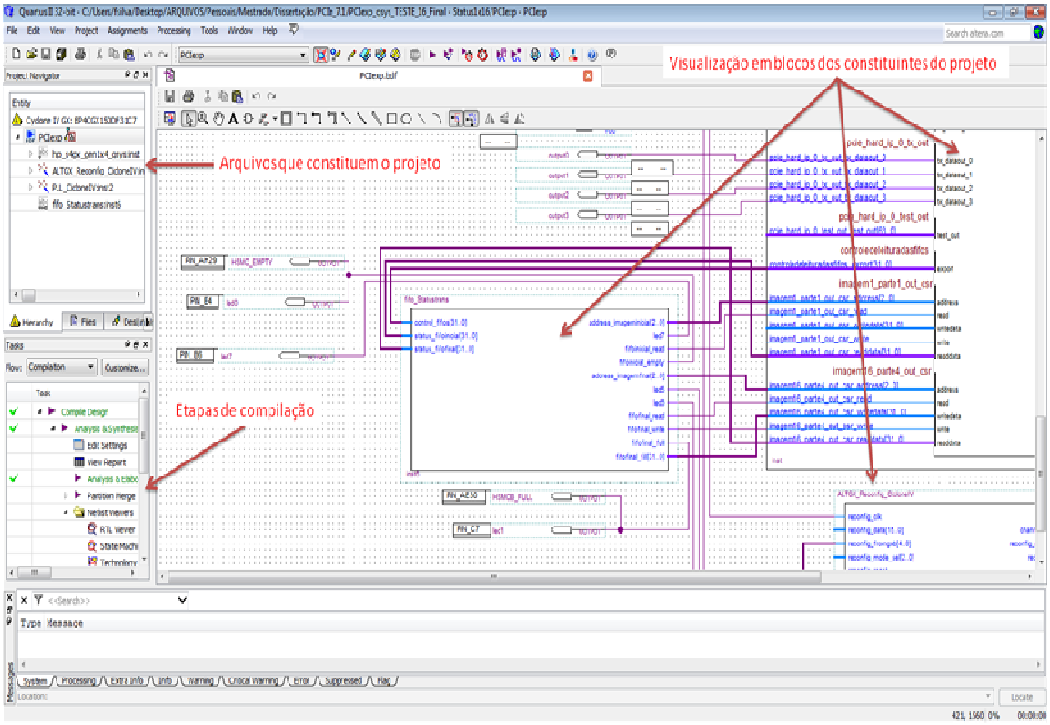



**Figura 3-17 – Visão geral do ambiente de desenvolvimento Quartus II**

### 3.3.3 Ferramenta LabVIEW

Foi necessário o desenvolvimento de um sistema capaz de importar as imagens de um arquivo, realizar o pré-processamento nas mesmas e acessar o barramento PCIe de forma ordenada para a correta transferência das imagens para o plataforma de desenvolvimento.

Para validação de execução dos algoritmos de processamento de imagens executados em FPGA, o sistema de *software* deve executar os mesmos algoritmos realizados em *hardware* configurável. Assim é possível realizar uma comparação dos resultados obtidos em *software* e *hardware* para as diferentes etapas do algoritmo de processamento de imagens avaliado.

Para elaboração das etapas anteriormente mencionadas, utilizou-se a ferramenta de desenvolvimento LabVIEW da empresa *National Instruments*, com a qual foi possível, de forma gráfica, programar o algoritmo de *software* necessário.

Uma característica desta ferramenta é a facilidade na elaboração de interfaces visuais de controle, as quais permitem aos usuários interação com os algoritmos criados, inserindo e obtendo informações dos mesmos.

O LabVIEW conta com uma biblioteca de funções denominada VISA (*Virtual Instrument Software Architecture*) desenvolvida pela *National Instruments*, que permite o acesso a dispositivos externos de instrumentação, através de interfaces GPIB, PXI, PCI, PCIe, Ethernet, USB e Serial.

Um elemento interessante do pacote VISA é o *NI-VISA Driver Wizard* (NATIONAL INSTRUMENTS, 2013), que permite a criação de *drivers* para dispositivos externos USB e PCI(e)/PXI(e). Esta ferramenta foi utilizada para gerar o *driver* de comunicação PCIe entre o algoritmo programado no FPGA e o computador de desenvolvimento.

O Sistema de Acesso ao FPGA, para gerenciamento, transferência das imagens e execução em *software* dos algoritmos de processamento de imagens foi desenvolvido e testado em um computador Pentium com CPU 2.80GHz, 1GB de RAM e HD de 220 G*Bytes*, rodando sistema operacional MS-Windows XP Professional, versão 2002/Service Pack 3.

# Capítulo 4

# Aplicação dos algoritmos de processamento de imagens em FPGA

O desenvolvimento deste trabalho pode ser dividido em duas etapas. A primeira delas com objetivo de estudar, desenvolver e validar o meio de transmissão das imagens entre o computador de desenvolvimento e o FPGA, resultando na caracterização do tempo de transferência. Já a segunda etapa envolve a inclusão em *hardware* dos algoritmos de processamentos de imagens avaliados, a fim de se caracterizar seus tempos de execução e seus resultados quando comparados com suas aplicações em *software*.

## 4.1 Transferência das imagens para a plataforma de desenvolvimento

A Figura 4-1 mostra um diagrama com todos os elementos e suas principais funções, que constituem esta etapa de desenvolvimento.

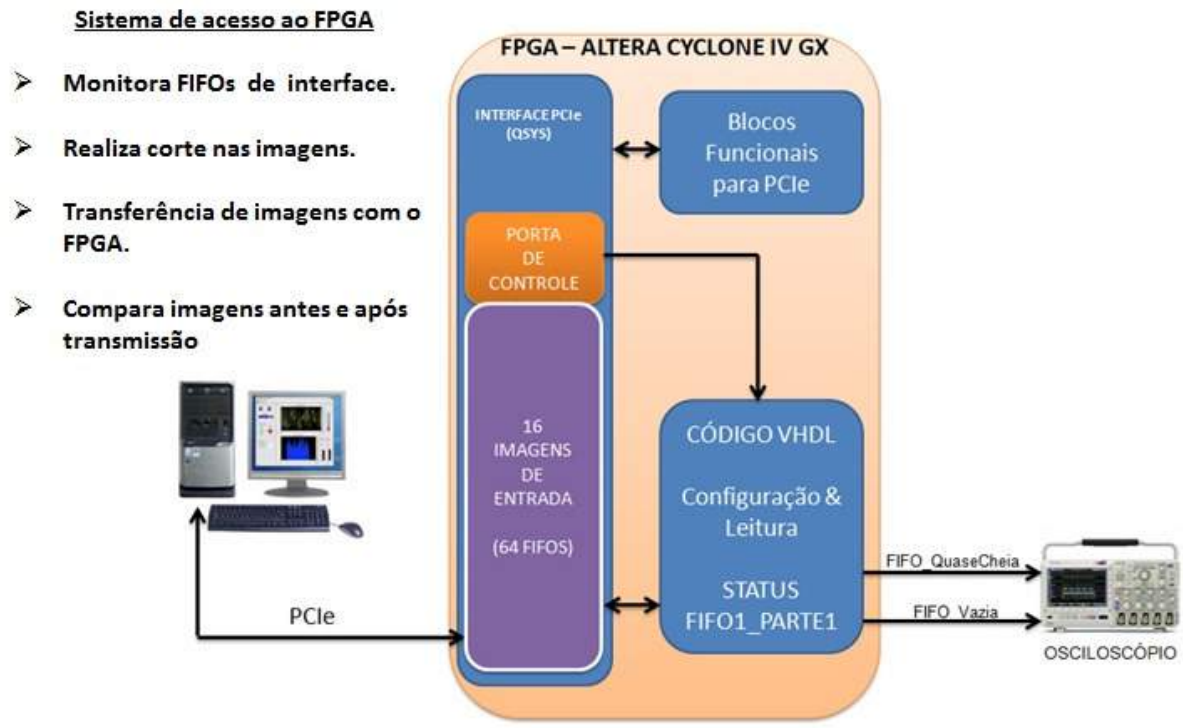



**Figura 4-1 – Transferência das imagens entre computador e FPGA e medição do tempo de transferência**

Conforme visto na Figura 4-1, para elaboração da transferência de imagens entre computador local e o FPGA e da medição do tempo de transporte das imagens é necessário desenvolver a interface que permite carregar as imagens da memória do computador, processá-las e acessar o barramento PCIe de forma ordenada para transmissão. Além disto, tem-se a programação do FPGA para recebimento das imagens e medição do tempo de transferência através de um osciloscópio.

Os constituintes da Figura 4-1 serão detalhados nas próximas seções, no entanto, inicialmente, é necessário se obter as seguintes características das imagens a serem transmitidas: (i) tamanho em *pixel*; (ii) quantidade de *bytes* por *pixel*; (iii) região de interesse dentro das imagens. As características mencionadas permitem definir como as imagens serão transferidas e armazenadas na plataforma de desenvolvimento, além da quantidade de memória que será utilizada durante a segunda etapa que trata da programação do FPGA.

### 4.1.1 Características das imagens utilizadas neste trabalho

As imagens utilizadas neste trabalho foram obtidas a partir do texto do trabalho de (SOUZA, 2013). Elas foram ajustadas para a resolução de 224 x 256 *pixels*, totalizando 57344 *pixels* com 8-bits em tons de cinza, i.e., cada *pixel* é formado por 1 *byte*, tendo seu valor variando entre zero e 255, onde zero representa o preto e 255 representa o branco em uma escala de cinza.

Frequentemente dentro de uma imagem, os eventos de interesse estão limitados em uma região específica, denominada região de interesse (ROI). Assim, o pré-processamento realizado em computador permite reduzir o tamanho das imagens, levando a uma redução no uso de memória no FPGA, e na quantidade de dados trocados entre o computador de desenvolvimento e a plataforma de desenvolvimento. Já que o fenômeno de *MARFE* ocorre apenas em uma região reduzida das imagens avaliadas, define-se, a região de interesse vista na Figura 4-2.

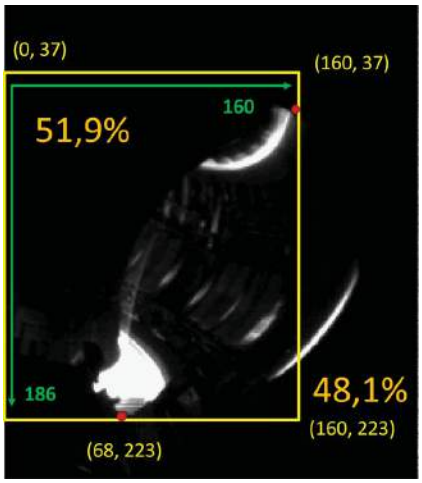



**Figura 4-2 – Região de interesse da imagem original. O tamanho da imagem final após sua redução é 160 x 186 pixels. Fonte: (ALBUQUERQUE *et al.*, 2013)**

A região de interesse observada, Figura 4-2, foi obtida pela análise de um banco de dados de imagens com o fenômeno de *MARFE,* conforme descrito em (ALBUQUERQUE *et al.*, 2013), deste modo esta ROI, para todas as imagens está sempre em uma mesma posição e possui sempre o mesmo tamanho.

Após o pré-processamento mencionado, cada uma das imagens apresenta um tamanho de 160x186 *pixels*. Ou seja, uma redução de 48,1% no tamanho original de uma imagem, que passou de 57344 *Bytes* para 29760 *Bytes*, i.e., *pixels*.

### 4.1.2 Programação da plataforma de desenvolvimento para transferência das imagens

O projeto desenvolvido para o FPGA nesta etapa de elaboração da interface de transferência das imagens, é constituído de uma interface PCIe programada a partir de um *IP block (IP_Compiler for PCI)* (ALTERA CORPORATION, 2011B) que permite a transferência de imagens entre o computador e o plataforma de desenvolvimento e vice-versa. O projeto possui também memórias FIFO (ALTERA CORPORATION, 2011) responsáveis por armazenar as regiões de interesse obtidas das imagens a partir de seu pré-processamento realizado em computador.

O processo de retorno das imagens do FPGA para o computador é necessário nesta etapa de desenvolvimento, para verificar se não ocorreu nenhuma alteração nas imagens quando transportadas através do barramento PCIe. Esta verificação é feita apartir de uma comparação *pixel* a *pixel* entre as imagens antes e após seu envio ao FPGA na interface de acesso no computador. Já na segunda etapa de desenvolvimento, o processo de retorno das imagens é necessário para obter a imagem resultante do processamento realizado no FPGA.

Sabendo que cada uma das imagens após pré-processamento apresenta 29760 *Bytes*, e que cada uma das FIFOs inseridas na ferramenta Qsys pode possuir tamanho máximo de 8192 elementos de 8 *bits*, utilizou-se uma arquitetura onde cada imagem ocupa 4 FIFOs consecutivas. Assim, cada imagem dentro do FPGA ocupa uma memória de 32768 *Bytes*, o que representa 4% da memória total do dispositivo (810000 *Bytes*). A partir destes números, estipulou-se a inserção de 64 FIFOs na ferramenta Qsys. Estas FIFOs realizam a interface de comunicação com o barramento PCIe, permitindo o armazenamento de até 16 imagens, o que resulta em 64% da memória total. Deste modo, tem-se 36% de memória disponível para desenvolvimento das etapas de processamento embarcado.

Cada uma destas FIFOs é responsável por armazenar 25% da quantidade total de *pixels* de uma imagem, i.e., 7440 *pixels*. Dentro da ferramenta Qsys, cada FIFO recebeu um rótulo que identifica a qual imagem e qual parte desta, a FIFO está associada, por exemplo, a FIFO identificada por “Imagem1_Parte1”, armazena os primeiros 7440 *pixels* da primeira imagem. Foram então gerados 64 rótulos, desde “Imagem1_Parte1” até “Imagem16_Parte4”. A Figura 4-3 mostra a divisão de uma imagem para armazenamento em suas quatro FIFOs independentes.

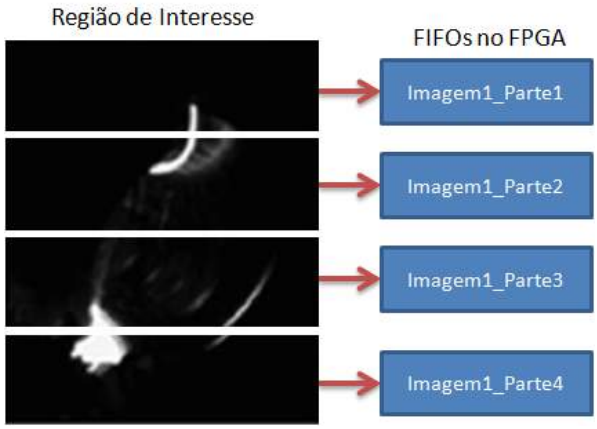



**Figura 4-3 – Divisão de uma imagem em 4 FIFOs independentes.**

É importante mencionar que esta divisão das imagens em blocos independentes, i.e., 4 FIFOS, além de ser necessária devido ao comprimento máximo das FIFOs na ferramenta Qsys, permite explorar a técnica de paralelismo de dados para as etapas de processamento de imagens embarcado em FPGA, conforme será visto na próxima seção.

A interface PCIe foi desenvolvida através da programação do *IP Block* específico. Esta interface é interna ao FPGA e utiliza uma abordagem via *hard IP* (ALTERA CORPORATION, 2011B). Para configuração *do IP_Compiler for PCI* é necessário inserir informações como identidade do fabricante e do dispositivo. Estas mesmas informações devem ser utilizadas para criação do *driver* de comunicação, a partir da ferramenta *NI-VISA Driver Wizard.* Foi determinada a utilização de 3 endereços base de registro (BARs), através dos quais é possível realizar escrita e leitura no barramento PCIe.

As FIFOs foram configuradas para operar em modo *dual clock*, o que permite associar dois sinais de *clock* diferentes e criar duas portas de *status* associados à escrita e leitura dos dados. O sinal de *clock* para controle de escrita e leitura das FIFOs foi obtido a partir do *IP_Compiler for PCI,* já que este foi definido como o *master* do sistema, sendo especificada a frequência de 125MHz, máximo valor possível para o FPGA utilizado (ALTERA CORPORATION, 2011B).

As portas de *status*, dentre outras informações, oferecem um *flag* denominado *FIFO_QuaseCheia*, este informa o momento quando a memória alcança uma quantidade de elementos pré-configurada. Este indicador é interessante, pois para a transmissão das imagens, as filas de 8192 posições devem ser consideradas cheias quando alcançarem 7440 elementos.

O trabalho com a ferramenta Qsys exige que o programador execute as ligações entre *IP Blocks,* definindo características como *clock*, *reset* e interfaces de comunicação. As portas de entrada das FIFOs foram conectadas no endereço base um (BAR 1), enquanto as portas que informam o *status* de escrita, como *FIFO_QuaseCheia* e *FIFO_Vazia* foram conectados no endereço base dois (BAR 2). Já as portas de saída das FIFOS foram associadas ao endereço base três (BAR 3).

Cada uma das FIFOs conectadas aos endereços base (BARs) da interface PCIe recebeu um endereço específico. Isto permite que o programa que é executado no computador local tenha capacidade de acesso independente para cada informação de entrada/saída disponibilizada pelas FIFOs. Esta tarefa de alocação manual de endereços foi realizada diretamente na ferramenta Qsys. A Figura 4-4 mostra a ferramenta para determinação dos endereços associados a cada BAR do barramento PCIe. Por questão de espaço, nesta imagem está sendo mostrada a alocação de apenas alguns dos endereços utilizados.

Observando a Figura 4-4, percebe-se que o endereço 0x00 associado ao BAR1 está relacionado ao *Controle_Acesso_Fifos*, que foi definido como uma porta de controle para o algoritmo sintetizado no FPGA, i.e., através do acesso ao BAR1, endereço 0x00, é possível enviar até 32 bits de controle a partir do programa que roda no computador local. Assim, a partir de uma ação do usuário na interface de controle é possível disparar ações dentro do FPGA.

A ferramenta Qsys permite que informações referentes a entradas e saídas dos blocos funcionais sejam exportadas para uso com outros circuitos dentro e/ou fora do FPGA. Como exemplo, nesta etapa do desenvolvimento foi realizado a exportação dos dados referentes aos *status* de leitura da primeira FIFO (Imagem1_Parte1). A fim de se associar pinos digitais disponíveis no plataforma de desenvolvimento para medida com osciloscópio dos sinais relativos a informações de *FIFO_Vazia* e *FIFO_QuaseCheia*. O intervalo entre a transição destes dois sinais será uma estimativa do tempo total de transferência.

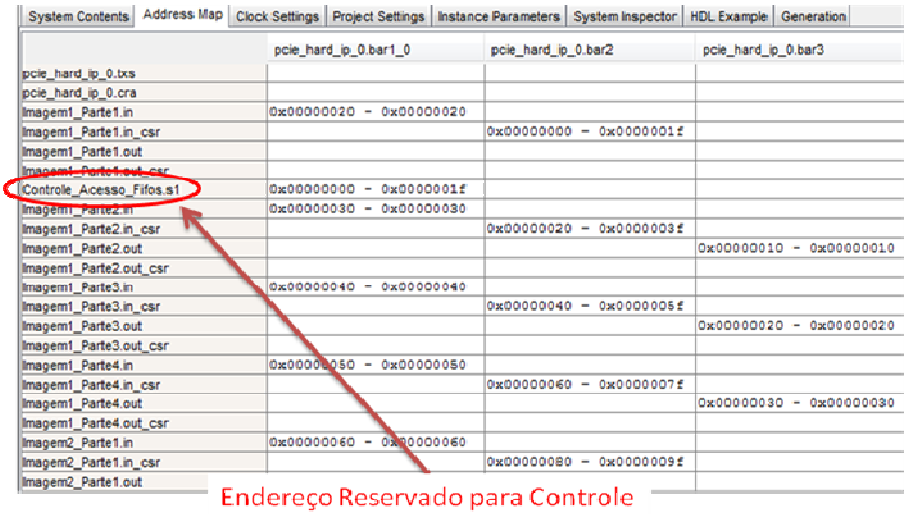



**Figura 4-4 – Mapa de endereços do barramento PCIe observados a partir da ferramenta Qsys/QuartusII**

Foram adicionados blocos funcionais como PLLs e interfaces de saída digital, além de uma função personalizada, programada em VHDL, criada exclusivamente para configurar o parâmetro *FIFO_QuaseCheia*. Este parâmetro está disponível na porta *status* de leitura da primeira FIFO (Imagem1_Parte1), e permite a verificação das *flags* através de interfaces digitais de saída. O

controle desta função, para seleção de escrita de configuração da *FIFO_QuaseCheia* ou leitura dos dados de *status*, é realizado pela interface no computador local, via porta de controle *Controle_Acesso_Fifos*. Esta porta é configurada no endereço BAR 1 do barramento PCIe.

É importante mencionar que os parâmetros de *status* relacionados às portas de escrita de todas as FIFOs são lido pelo *software* executado no computador local. Assim, para leitura do sinal via osciloscópio, convencionou-se utilizar o registro de *status* relacionado à porta de leitura destas FIFOs.

### 4.1.3 Interface para transferência das imagens e monitoramento da memória embarcada.

Para o acesso ao FPGA, para envio e recebimento das imagens, foi desenvolvido um programa em LabVIEW. Este utiliza a biblioteca VISA que permite o acesso aos registros base de dispositivos conectados a diversos barramentos, dentre eles PCIe.

Inicialmente, foi gerado o *driver* de comunicação através da ferramenta *NI-VISA Driver Wizard*. Uma vez instalado o *driver*, o FPGA já sintetizado com o protocolo PCIe passou a ser reconhecido como um instrumento conectado ao computador de desenvolvimento. A Figura 4-5 mostra uma imagem da tela do aplicativo "*Measurement & Automation*" da National Instruments que gerencia todos os dispositivos associados ao computador local. Nesta figura é possível observar a FPGA CycloneIV reconhecida como um instrumento conectado ao computador.

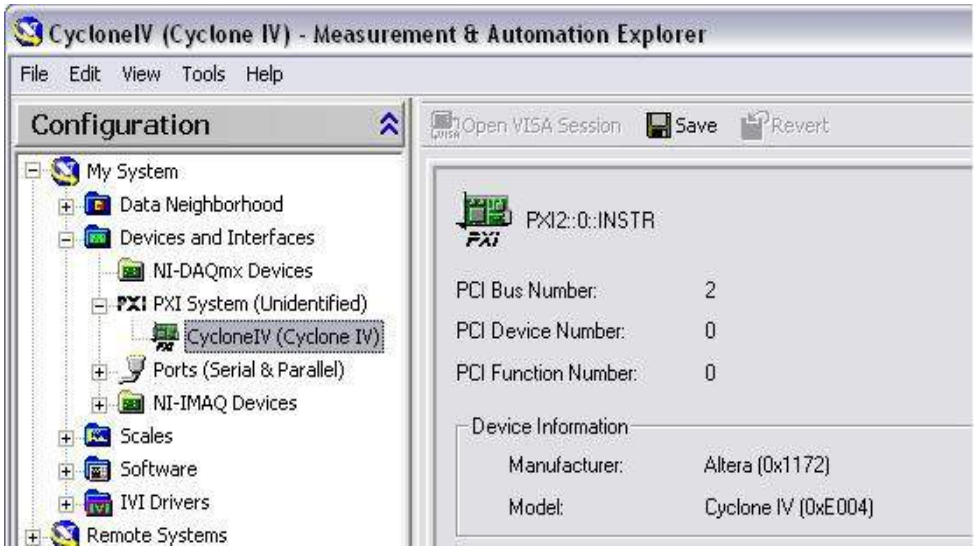



**Figura 4-5 – FPGA Cyclone IV é reconhecida como um instrumento conectado ao computador de desenvolvimento, por meio do programa de gerenciamento de dispositivos "*Measurement & Automation*" da *National Instruments*.**

A interface para acesso ao FPGA apresenta, conforme visto na Figura 4-6, cinco abas com respectivas ações e informações para operação do sistema. Na seção "OPERAÇÃO LOCAL", é possível

carregar um conjunto de imagens do disco local e realizar operações de corte de tamanho configurável sobre as mesmas, a fim de se obter suas regiões de interesse. A interface também informa em *Pixels* (*Bytes*), o tamanho dos dados, antes e após o corte. Na seção "COMUNICAÇÃO COM FPGA", é possível selecionar as imagens previamente carregadas e cortadas para transferência à memória do dispositivo de lógica programável, sendo possível a transferência de até 16 imagens. Informações como o *status* de comunicação com o FPGA são vistos nesta aba. Caso necessário, é possível realizar a transferência dos dados a partir do FPGA para memória local, esta ação é tomada através do botão "Transferir para PC". O botão "Verificar FIFOS de entrada" roda uma verificação sobre o *status* de todas as FIFOS, informando então na aba "ESTADO DAS FIFOS DE INTERFACE NO FPGA" quantos elementos estão contidos em cada uma delas e se as mesmas encontram-se cheias ou vazias. Na seção "REGISTRO DE ACESSO_PCIe" é possível configurar os endereços para acesso às portas de entrada, portas de *status* de escrita e portas de saída de todas as FIFOs. Estes endereços devem ser os mesmos daqueles configurados no mapa de endereços da ferramenta Qsys. Por fim na seção "ANÁLISE DAS IMAGENS" é possível executar uma verificação *pixel a pixel* entre as imagens antes e após seu envio, a fim de validar o processo de transmissão via barramento PCIe.

A programação para realização das tarefas anteriormente mencionadas foi feita a partir de uma linguagem gráfica. Os algoritmos seguem um funcionamento baseado em máquina de estado orientada por eventos de usuário, para resposta às interações com os botões da interface.

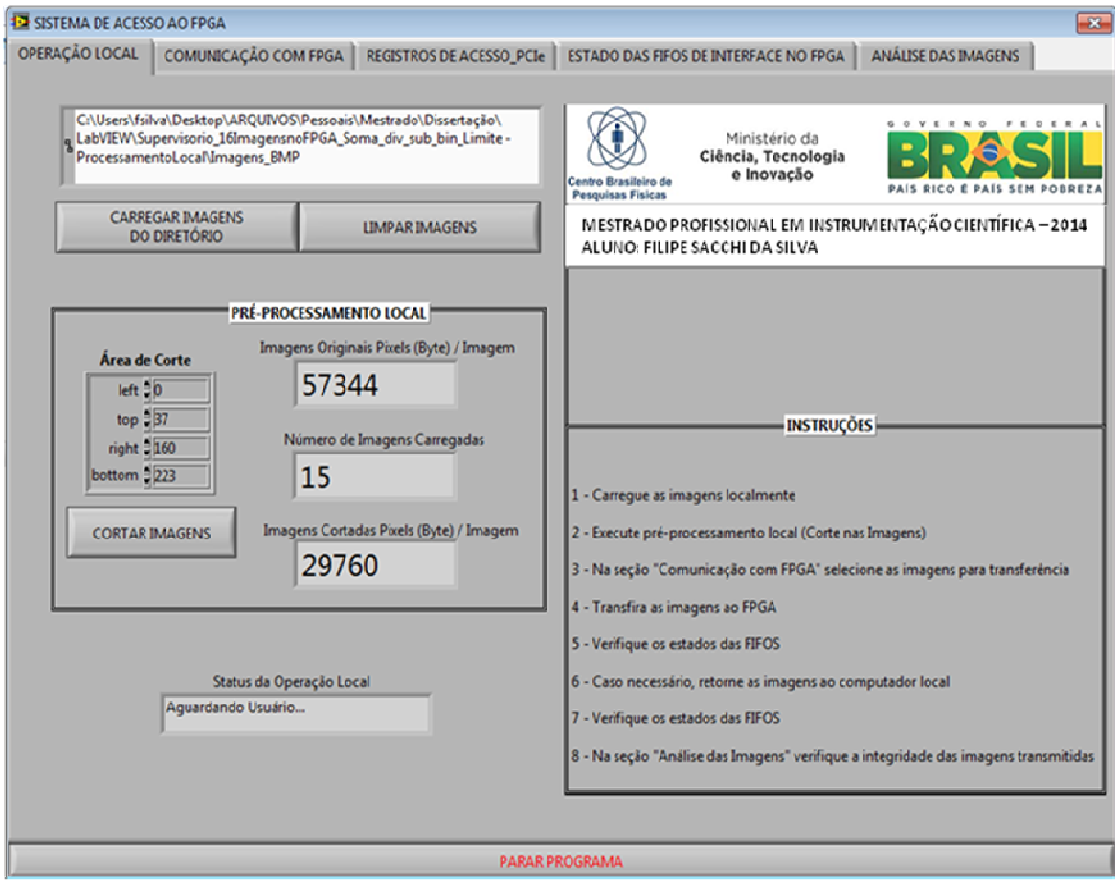



**Figura 4-6 – Visão geral da interface do programa desenvolvido para acesso ao FPGA.**

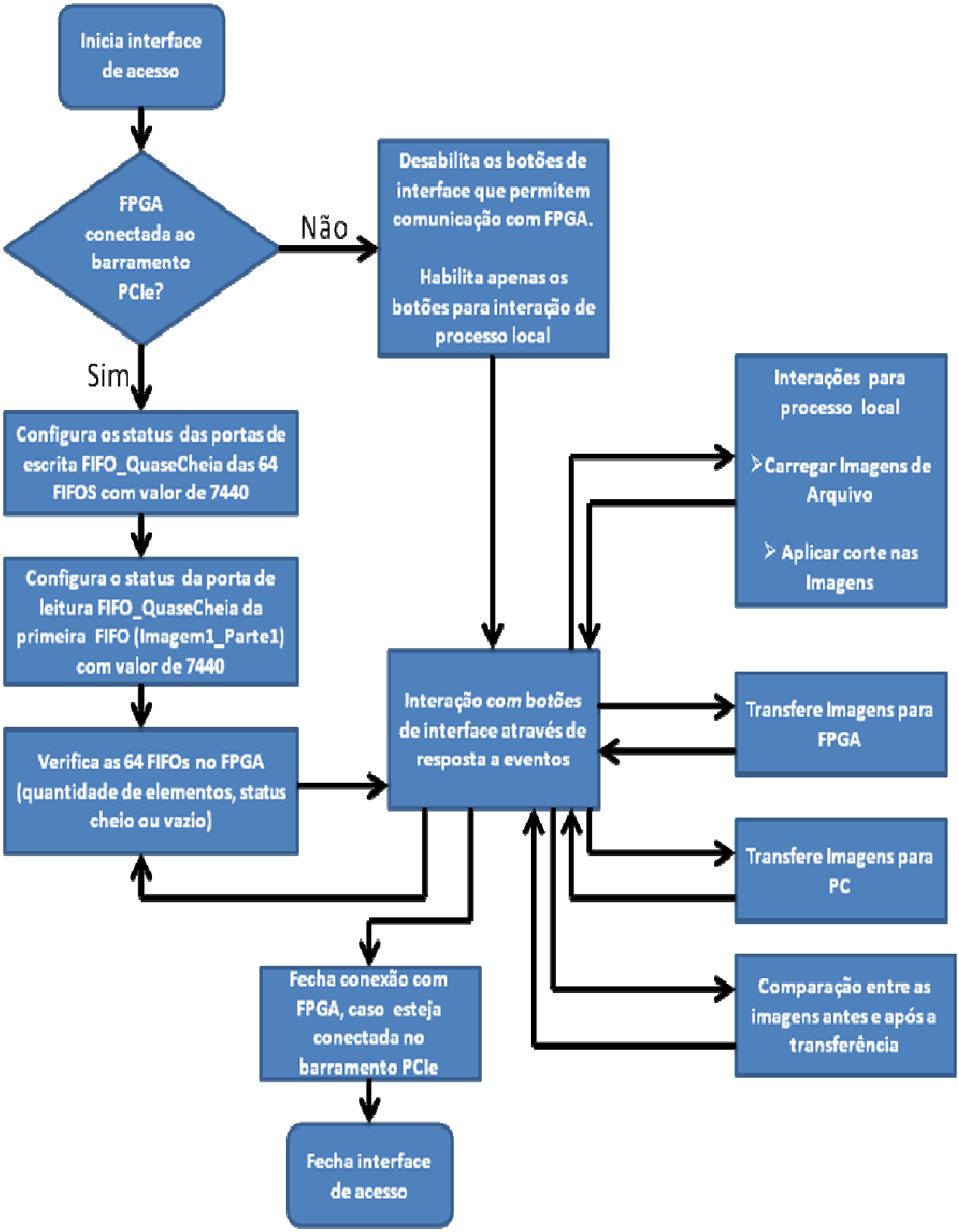



**Figura 4-7 – Inicialização e ações disponíveis ao usuário a partir da interface de acesso.**

Uma característica importante é que o estado de inicialização do programa deve procurar pelo FPGA conectado no barramento PCIe e acessar os endereços de *status* de escrita de todas as filas, para configuração do parâmetro *FIFO_QuaseCheia* com um valor de 7440. Assim as FIFOs podem ser consideradas cheias quando alcançam 7440 elementos (*pixels*) escritos. A Figura 4-7 mostra um diagrama com as etapas de inicialização e ações disponíveis ao usuário pela interface de acesso.

### 4.1.4 Metodologia para estimativa do tempo de transferência das imagens

O *driver* VISA para comunicação com barramento PCIe através da interface de acesso, possibilita interações de escrita e leitura em espaços de memória de instrumentos externos, no caso o Plataforma de desenvolvimento contendo FPGA. Conforme referência (VXIPLUG&PLAY SYSTEMS ALLIANCE AND IVI FOUNDATION , October 2012) interações do *driver* VISA ocorrem em acessos de 32 bits no barramento PCIe. Assim, mesmo que o *payload* do protocolo PCIe seja configurado através da ferramenta Qsys, para o valor de 256 *Bytes* (máximo valor para o Plataforma de desenvolvimento utilizado (ALTERA CORPORATION, 2011B) apenas acessos independentes de 32 bits são realizados.

O *driver* VISA, por se tratar de uma ferramenta de *software* cujo código não pode ser alterado pelos usuários, impossibilita a configuração de parâmetros (endereços de escrita/leitura, quantidade de transferências a serem realizadas) de controladores de DMA (XILINX , 2008) que por ventura sejam inseridos na FPGA conectada ao barramento PCIe. O driver VISA é otimizado para transferência em modo rajada (burst) (ALTERA CORPORATION, 2011B) apenas para pacotes de dados com 32bits no barramento PCIe (VXIPLUG&PLAY SYSTEMS ALLIANCE AND IVI FOUNDATION , October 2012). Deste modo, mesmo possuindo um controlador de DMA, o FPGA não pode operar como *Bus Master* conforme descrito em (BITTNER, 2012), inviabilizando então transferências otimizadas de mais de 32bits de dados em modo rajada (*burst*) para a interface de acesso via *driver* VISA.

Sabe-se que a utilização de controladores DMA para transferência em modo rajada, acelera as transmissões de dados entre a memória do computador e do dispositivo externo conectado ao barramento PCIe e permite a exploração das bandas de transmissão do barramento PCIe (QIANG *et al.*, 2009). No entanto, devido às características do *driver* VISA anteriormente mencionadas, este trabalho limita-se em realizar transferências individuais em modo rajada de pacotes de dados contendo 32bits (i.e. 4 *pixels* da imagem transferida).

A metodologia para medição do tempo de transmissão considera então que o sistema de transferência das imagens é constituído pelo *driver* de acesso VISA, barramento PCIe (contendo todo o *overhead* inserido pelo protocolo PCIe (XILINX , 2008)) e a lógica embarcada no FPGA para armazenamento das imagens, a qual é executa com um *clock* de 125MHz, conforme mencionado na Seção 4.1.2.

A Figura 4-8 mostra as camadas de *software* e *hardware* que compõem o sistema de transferência, considerando uma transmissão no sentido PC para FPGA. Percebe-se que a cada chamada do *driver*, a camada de aplicação direciona um bloco de 1860 elementos de 32 bits para serem transmitidos pelo barramento PCIe. Convencionou-se utilizar esta quantidade de elementos, pois resulta em 7440 *Bytes* (i.e. 7440 *pixels*), o que é a quantidade de um bloco individual de uma imagem. Esta quantidade de dados pode ser transferida sem a interferência da camada de aplicação,

uma vez que todos os elementos estão direcionados a um mesmo endereço (i.e., uma FIFO) dentro do FPGA.

Transferências de blocos de dados a partir da camada de aplicação são possíveis devido a função vi*MOVEOUT32* presente na biblioteca VISA. Esta apresenta o melhor desempenho em termos de velocidade quando comparado com as demais funções de acesso a registros da biblioteca VISA (NATIONAL INSTRUMENTS, 2001).

Quando a camada de *driver* é chamada, está realiza 1860 acessos ao barramento PCIe transferindo os pacotes de 32 bits sem interferência da camada de aplicação. Devido a este fato, a metodologia para medição de tempo desconsidera o tempo da camada de aplicação responsável por montar o bloco de 1860 elementos de 32 bits, conforme indicação da linha pontilhada na Figura 4-8.

Para a medição do tempo de transferência do bloco contendo os 1860 elementos de 32 bits utiliza-se os indicadores *FIFO_Vazia* e *FIFO_QuaseCheia* para a primeira FIFO (Imagem1_Parte1). O tempo de transição entre estes dois sinais indicará o tempo necessário para preencher toda uma FIFO com 7440 *pixels*.

Para realização da tarefa mencionada é preciso que na inicialização do *software* de acesso ao FPGA seja realizada a configuração do parâmetro *FIFO_QuaseCheia* do *status* de leitura da primeira FIFO (Imagem1_Parte1), para o valor 7440. Esta etapa pode ser vista no diagrama da Figura 4-7.

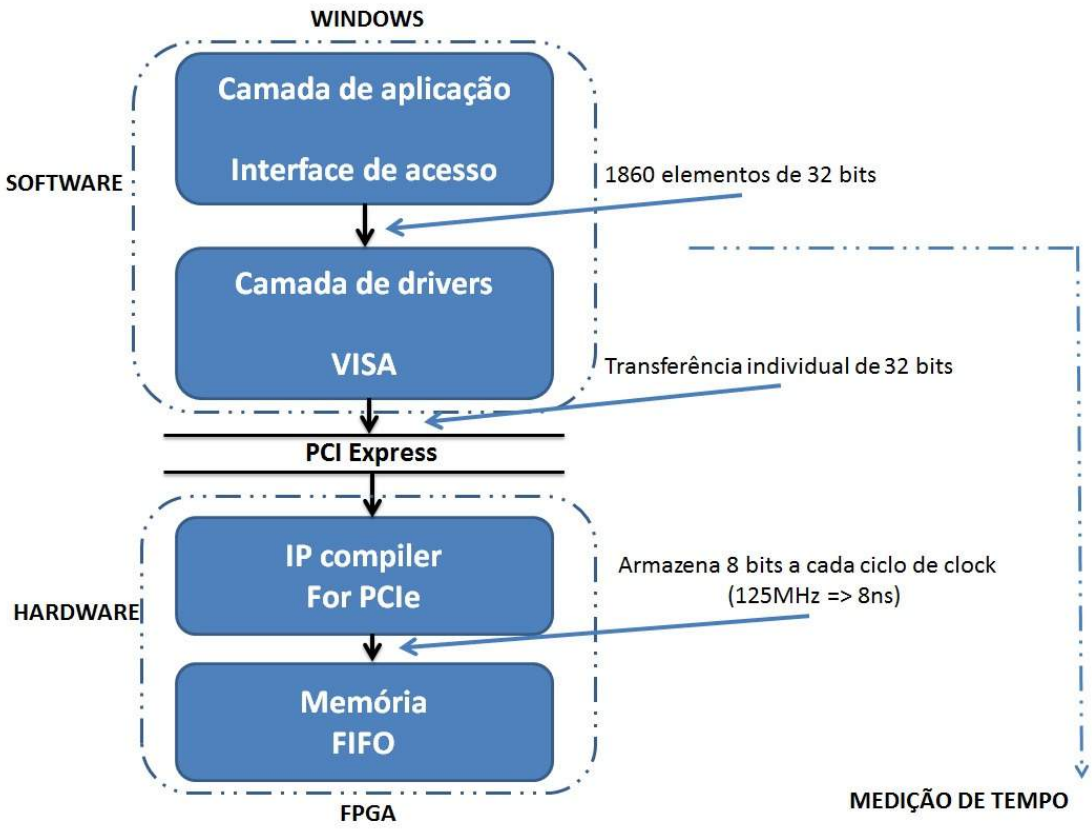



**Figura 4-8 – Camadas de *software* e *hardware* que formam o sistema de transferência de imagens.**

A Figura 4-9 mostra o *status* de escrita para todas as FIFOs antes da transmissão das imagens. As informações são expressas em formato de uma matriz com 64 elementos (4 x 16), onde cada um deles informa a situação para cada uma das FIFOs (i.e, cheia, vazia e a quantidade de elementos contidos nessas filas). As colunas da matriz representam a quantidade de imagens e as linhas informam as FIFOs para cada uma delas. A matriz pode ser percorrida através dos controladores de indexação localizados em seu canto superior esquerdo.


**Figura 4-9 – A imagem apresenta o Sistema de Acesso ao FPGA. É possível ver o status de 12 das 64 FIFOS antes da transmissão. Neste instante todas as 12 FIFOS estão vazias.**

## 4.2 Programação dos algoritmos de processamento de imagens no FPGA

Esta etapa envolve a configuração em *hardware* dos algoritmos de processamento de imagens avaliados: o cálculo de imagem de fundo, a partir da média de imagens anteriores; subtração da imagem de fundo a partir da imagem corrente e, por fim, a segmentação em objetos das partes em movimento na imagem corrente. Foram utilizados como ponto de partida os

componentes de *hardware* e *software* para transferência de imagens entre computador e FPGA, e vice-versa, inicialmente desenvolvidos e validados na primeira etapa e descritos na seção anterior.

Durante o desenvolvimento da primeira etapa estipulou-se a transferência de até 16 imagens entre a plataforma de desenvolvimento e a interface de acesso, através da utilização de 64 FIFOs programadas via ferramenta *Qsys*. Para esta segunda etapa de desenvolvimento, 60 das 64 FIFOs são utilizadas para armazenar as imagens transferidas do computador de desenvolvimento para FPGA, em um total de 15 imagens de entrada para os algoritmos de processamento de imagens em *hardware.* As quatro FIFOs restantes são utilizadas como saída, armazenando a imagem resultante do processamento de imagens em FPGA, para a transferência da FPGA para o computador.

As 15 imagens de entrada são imagens sequênciais do fenômeno de *MARFE* avaliado, de forma que a imagem mais nova, referenciada também como imagem corrente, estará na décima quinta posição. A Figura 4-10 mostra um diagrama de blocos dos algoritmos de processamento de imagens programados no FPGA. O cálculo da imagem de fundo é realizado através da média de 14 imagens. A imagem de fundo é então armazenada em quatro FIFOs intermediárias, sendo esta posteriormente subtraída da imagem corrente, resultando na imagem conhecida como *SAv* (*Background Image Subtraction*). Por fim, esta imagem *SAv* passa pelo algoritmo de segmentação por *threshold,* resultando em uma imagem binária a ser retornada para o computador de desenvolvimento, para comparação e validação.

Conforme visto na Figura 4-10, a interface de acesso deve realizar em *software* o mesmo processamento de imagens realizado em *hardware*, a fim de permitir uma comparação de resultados e validação da execução dos algoritmos inseridos no FPGA. Para execução dos algoritmos embarcados, a interface de acesso é responsável por: (i) enviar 15 imagens de entrada ao FPGA, e receber deste, 1 imagem com o resultado do processamento; (ii) disparo sequêncial de execução das etapas que compõem o algoritmo embarcado; (iii) determinação do limiar de *threshold* utilizado para a etapa de segmentação.

Conforme descrito anteriormente, e de acordo com a Figura 4-10, para o desenvolvimento desta segunda etapa foram necessárias modificações, tanto na interface de acesso quanto na programação do FPGA. Assim, as próximas seções têm por objetivo descrever de uma maneira mais detalhada os componentes vistos na Figura 4-10. Outro ponto também a ser mostrado será a forma de validação do intervalo de tempo de execução de cada uma das etapas do processamento embarcado de imagens.

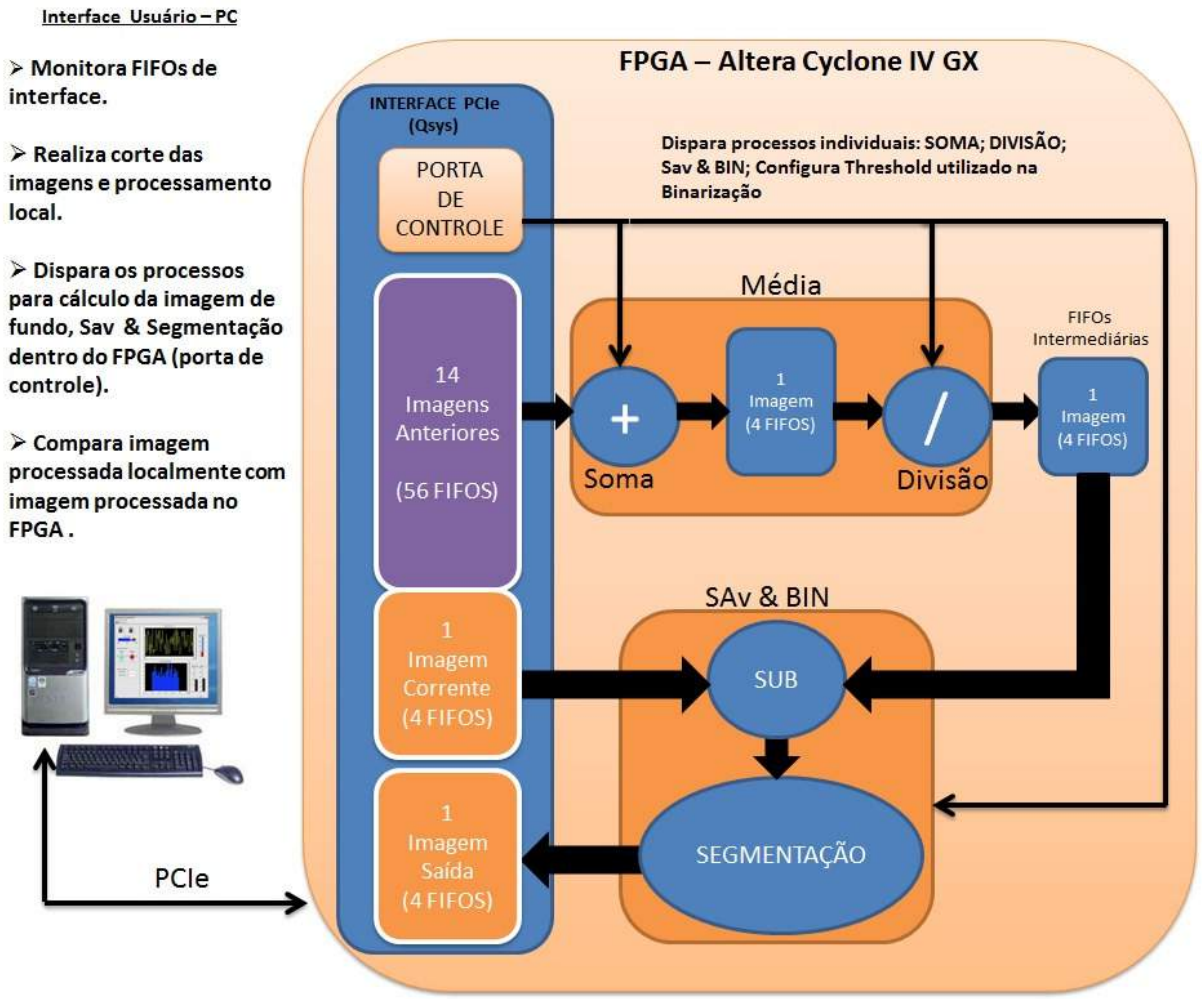



**Figura 4-10 – Diagrama em blocos dos algoritmos de processamento de imagens inseridos no FPGA.**

### 4.2.1 Programação da plataforma de desenvolvimento para o processamento de imagens embarcado

Para a inclusão dos algoritmos de processamento de imagens em *hardware*, foi modificada a forma de alocação dos endereços base de registros (BARs) descrita na Seção 4.1.2. As FIFOs referenciadas como “Imagem16_Parte1”, “Imagem16_Parte2”, “Imagem16_Parte3” e “Imagem16_Parte4” são utilizadas para a transferência, da plataforma de desenvolvimento para computador, da imagem resultante do processamento embarcado. Assim, o BAR3 estará associado apenas às portas de saída das FIFOs anteriormente mencionadas. O BAR1 é utilizado para transferência do computador para a plataforma de desenvolvimento das 15 imagens de entrada. Assim, BAR1 estará associado às portas de entrada das 60 FIFOs referenciadas como “Imagem1_Parte1” até “Imagem15_Parte4”. Já o BAR2, da mesma forma que descrito na Seção 4.1.2, é utilizado para monitoramento das portas de *status* de escrita, como por exemplo, *FIFO_QuaseCheia* e *FIFO_Vazia* de todas as 64 FIFOs utilizadas para transferência de imagens entre computador e plataforma de desenvolvimento, e vice-versa. O endereço 0x00 associado ao BAR1,

assim como descrito na seção 4.1.2, está relacionado ao *Controle_Acesso_Fifos*, que foi definido como uma porta de controle para o algoritmo sintetizado no FPGA. Desta forma, através do acesso ao BAR1 (endereço 0x00) é possível controlar sequêncialmente as etapas do algoritmo de processamento embarcado e determinar o limiar de *threshold* utilizado para a etapa de segmentação. A Figura 4-11 mostra uma visão geral da alocação dos BARs para transferência de imagens e informações de *status* entre computador e a plataforma de desenvolvimento. Nesta figura, para maior clareza, são mostrados apenas alguns dos endereços associados aos *status* de escrita de todas as FIFOs associados ao BAR2. É possível ver também a associação de todos os endereços utilizados para a transferência da imagem resultante da plataforma de desenvolvimento para o computador, através dos endereços associados ao BAR3.

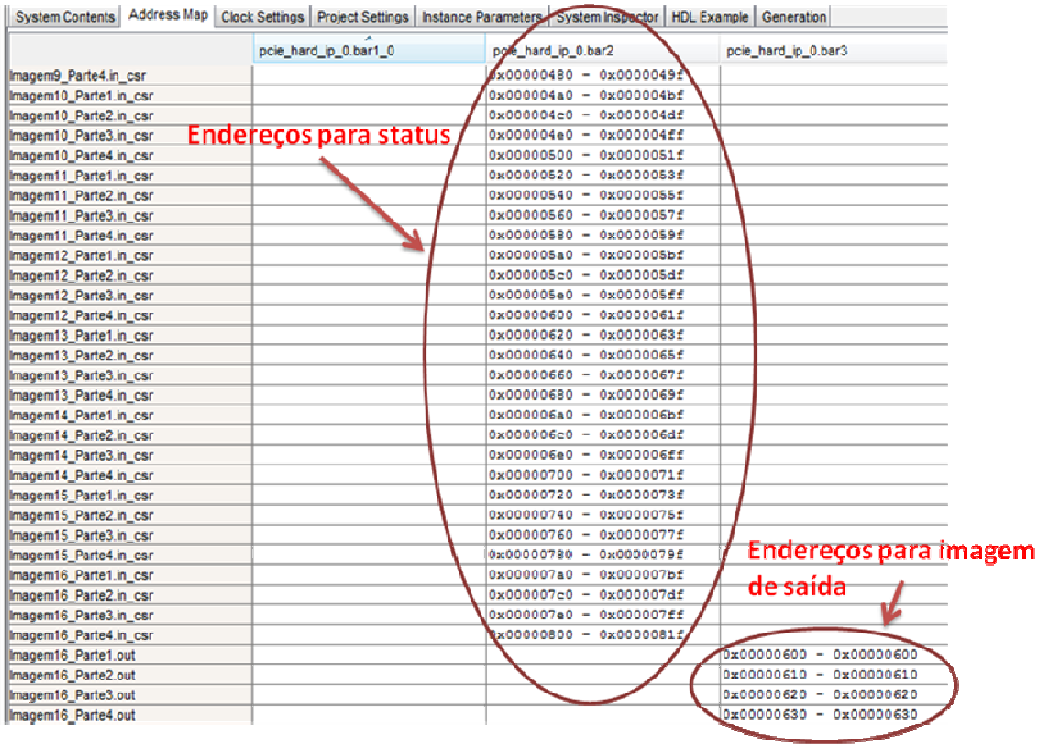



**Figura 4-11 – Visão geral da alocação de endereços dos BARs na ferramenta Qsys.**

As FIFOs foram mantidas com a configuração em modo *dual clock.* O sinal de *clock* para escrita nas 60 FIFOs (transferência das 15 imagens de entrada) e para leitura das 4 FIFOs, associadas à imagem resultante do processamento foi definido como 125MHz, conforme necessidade descrita na Seção 4.1.2. As portas de saída das 60 FIFOs e as portas de entrada das 4 FIFOs foram exportadas para fora do ambiente *Qsys*, onde os algoritmos de processamento de imagens foram inseridos. Para os testes de execução dos algoritmos de imagens embarcados, convencionou-se utilizar o mesmo *clock* de 125MHz. A Figura 4-12 mostra um diagrama com as FIFOs utilizadas para a transferência das imagens entre o computador e a plataforma de desenvolvimento.

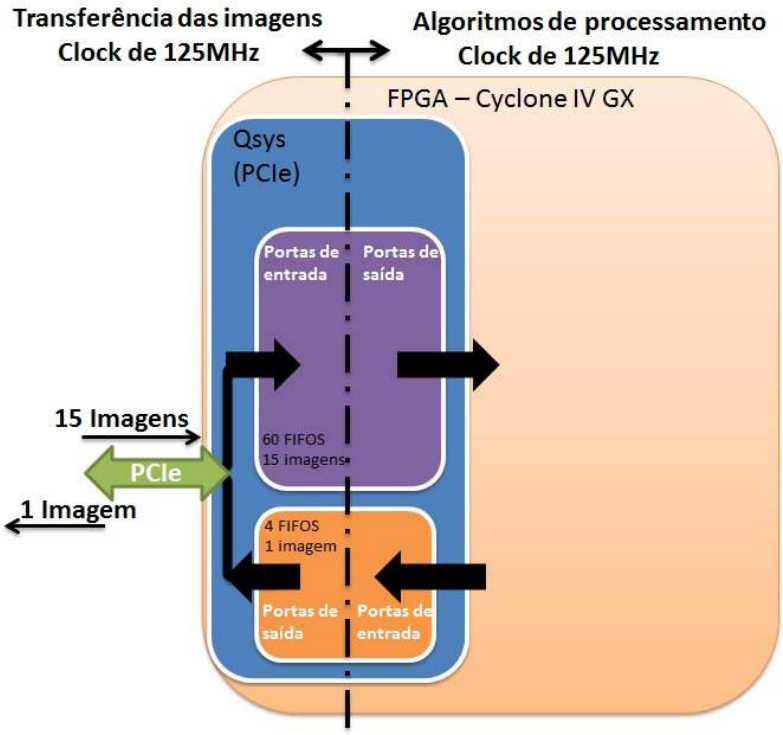



**Figura 4-12 – Visão geral das FIFOs utilizadas para a transferência de imagens entre computador e plataforma de desenvolvimento.**

Foram geradas três diferentes versões de projeto para programação do FPGA. A primeira compilação realiza apenas a estimação da imagem de fundo, retornando para a interface de acesso o vetor de *pixels* que forma a imagem de fundo estimada. A segunda compilação realiza no FPGA a estimação e subtração da imagem de fundo a partir da imagem corrente, retornando à interface de acesso o respectivo vetor de *pixels*. Já a terceira compilação realiza todos os passos do algoritmo, i.e., estimação da imagem de fundo; subtração da imagem de fundo a partir da imagem corrente; segmentação em objetos das partes em movimento na imagem corrente, retornando à interface de acesso, o vetor de *pixels* da imagem binarizada. A seguir será descrito com mais detalhes como cada uma das etapas foi programada no FPGA.

## Estimação da imagem de fundo

A imagem de fundo para o fenômeno de *MARFE* avaliado é obtida pela média aritmética de N imagens sequênciais. (CHACON, 2012) mostra que o valor ótimo para N é de 23, no entanto, devido a restrições de recursos de *hardware* no FPGA, este cálculo foi realizado com N igual a 14. Foram utilizados 4 somadores de 14 entradas cada um. Estes somadores apresentam entradas com 8 bits, ou seja, a mesma resolução dos *pixels* das imagens de entrada, e saída com 12 bits, suficiente para acomodar o maior caso de soma, i.e., quando todas as entradas apresentarem valor igual a $FF_{16}$. Estes somadores operam em paralelo, explorando a técnica de paralelismo de dados, sendo cada um deles responsável pela soma sequêncial dos 7440 *pixels* que formam cada um dos quatro blocos que compõem as imagens de entrada. Os *pixels* resultantes de cada um dos somadores são armazenados

em quatro memórias FIFOs que armazenam a imagem que contém a soma. A Figura 4-13 mostra a etapa de soma para estimação da imagem de fundo. O *clock* para leitura das 56 FIFOs (14 imagens de entrada) e para escrita das 4 FIFOs, que contêm a imagem de soma, é de 125MHz.

A etapa de soma das imagens no FPGA é habilitada pela interface de acesso no computador de desenvolvimento, através da porta de controle. Assim é feita a leitura e escrita das respectivas FIFOs que formam as imagens de entrada e a imagem que contem a soma. Como os quatro somadores operam em paralelo, o intervalo de tempo para a soma das 14 imagens, formadas por 29760 *pixels,* é divido por quatro, i.e., o intervalo de tempo total necessário é igual ao intervalo de tempo para a soma sequêncial dos 7440 pixels que compõem um dos blocos da imagem.

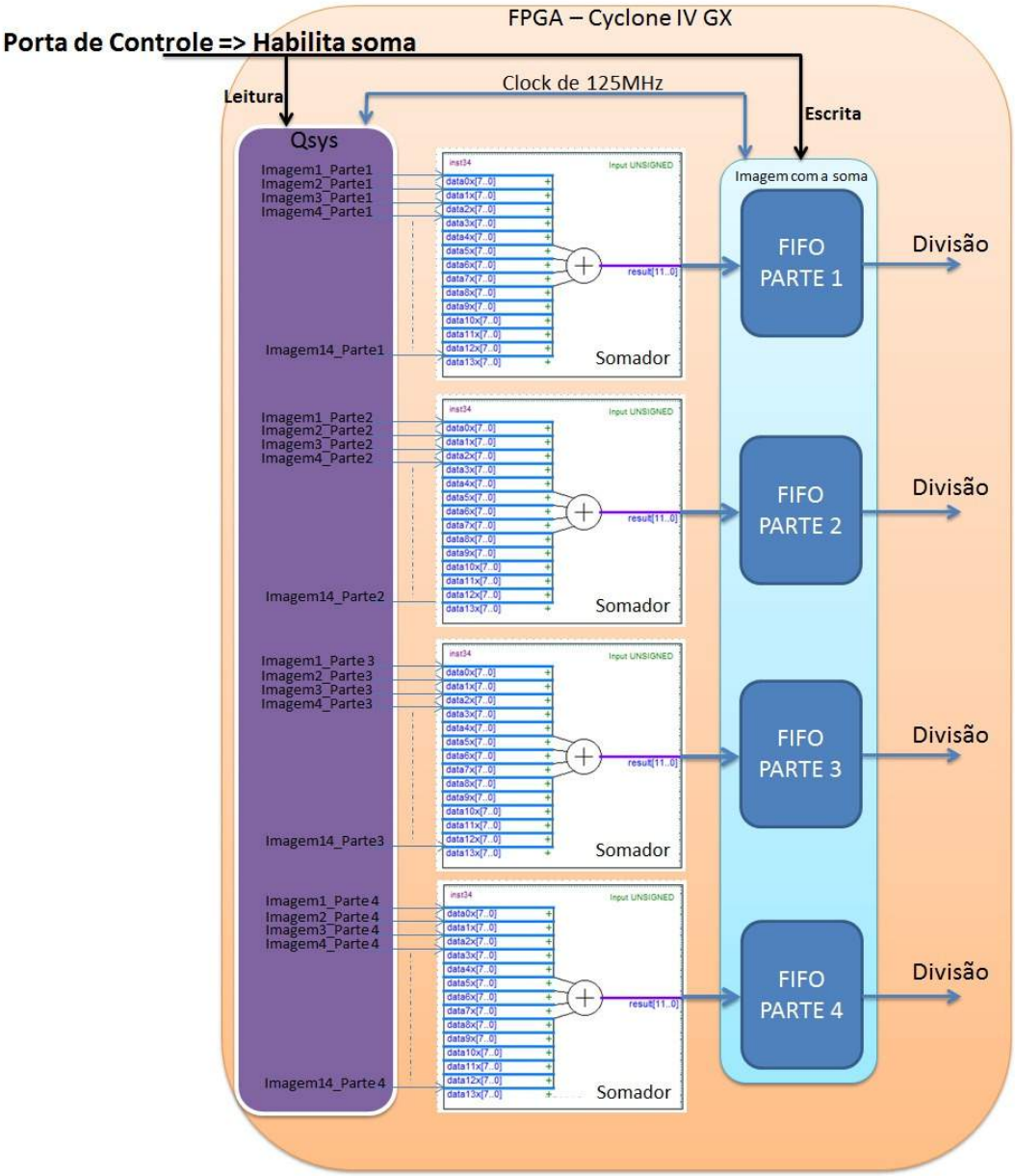



**Figura 4-13 – Etapa de soma das imagens anteriores para estimar imagem de fundo.**

Os somadores e as memórias FIFO vistos na Figura 4-13 foram inseridos na programação do FPGA pelo ambiente Quartus II, utilizando-se a ferramenta *MegaWizard Plug-In*. Isto evitou a necessidade de programação direta dos mesmos através de linguagem VHDL, simplificando o desenvolvimento. As FIFOs foram configuradas para armazenar 8192 elementos de 12 bits, e da mesma forma que realizado na seção 4.1.2, seu parâmetro *FIFO_QuaseCheia* foi definido para 7440 elementos, isto é, 25% da quantidade total de *pixels* de uma imagem.

Após a soma das 14 imagens, os *pixels* para cada um dos blocos da imagem de soma, armazenados nas FIFOs correspondentes, são divididos pela quantidade de imagens somadas. A Figura 4-14 mostra a etapa de divisão para estimação da imagem de fundo. O *clock* de leitura para as quatro FIFOs que contem a soma das imagens, assim como o *clock* de escrita para as FIFOs que contem a imagem de fundo estimada, é de 125MHz. Da mesma forma que feito para os somadores, os quatro divisores e as FIFOs que contem os resultados da divisão foram inseridos na programação do FPGA pelo ambiente Quartus II através da ferramenta *MegaWizard Plug-In*. Nos divisores, as entradas para os numeradores foram configuradas para 12 bits e as entradas para os denominadores para 4 bits, onde foi estabelecido um valor constante de $14_{10}$ ($E_{16}$). A saída para cada um dos divisores apresenta 12 bits, sendo o quociente da divisão entre os dois números inteiros de entrada, o valor de resto foi desconsiderado. Como o valor máximo esperado para a divisão é de $255_{10}$, inseriu-se uma função programada em VDHL, que obtém os 8 primeiros bits dos 12 bits produzidos pela divisão. Na Figura 4-14 esta função é referenciada como “12bits => 8bits”.

A etapa de divisão das imagens no FPGA é habilitada pela interface de acesso no computador de desenvolvimento, através da porta de controle. Assim, é feita a leitura e escrita das respectivas FIFOs que contêm a soma das imagens e a imagem de fundo estimada. Da mesma forma que ocorre na etapa de soma, os quatro divisores operam em paralelo, e assim o intervalo de tempo para a divisão de uma imagem constituída de 29760 *pixels* é divido por quatro.

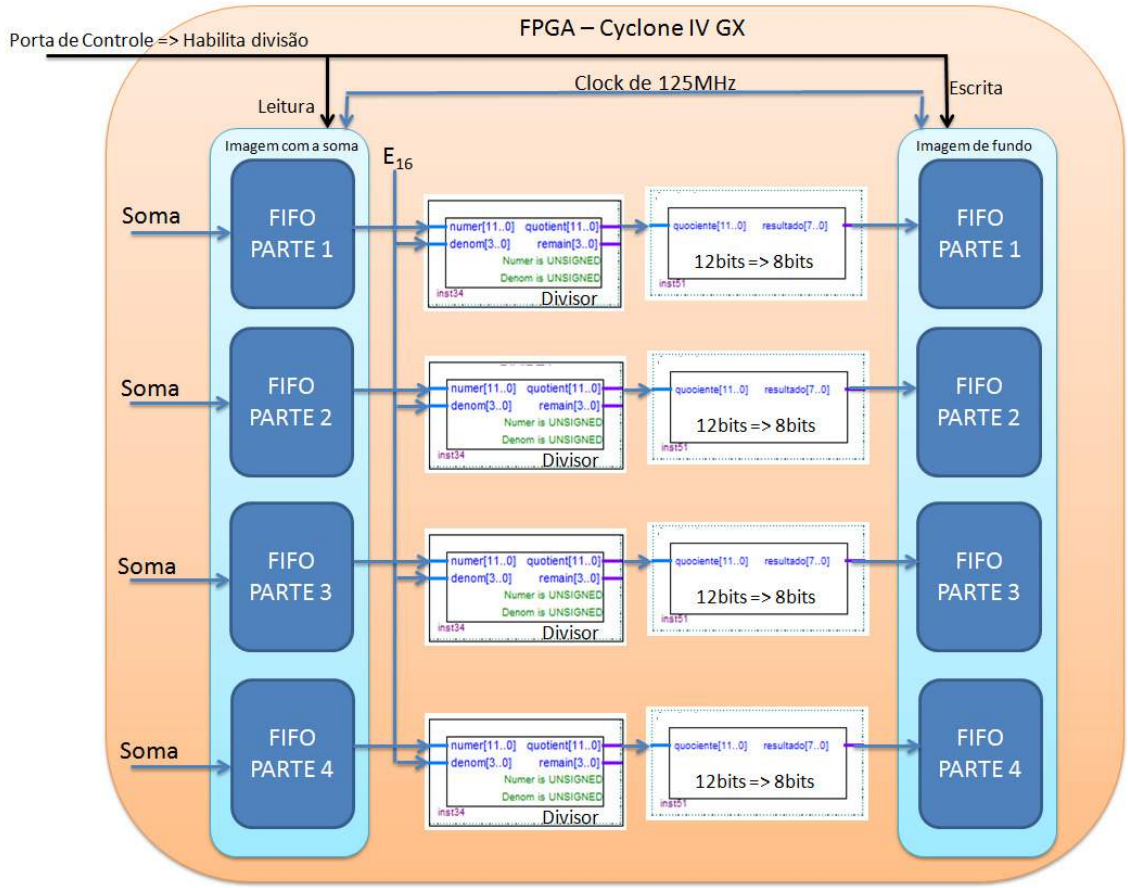



**Figura 4-14 – Etapa de divisão para estimar a imagem de fundo.**

## Subtração da imagem de fundo a partir da imagem corrente

Após a estimação da imagem de fundo, é necessário realizar a subtração da mesma a partir da imagem corrente. Esta imagem está armazenada nas FIFOs localizadas no ambiente Qsys, especificamente em “Imagem15_Parte1”, “Imagem15_Parte2”, ”Imagem15_Parte3” e “Imagem15_parte4”. A Figura 4-15 mostra a etapa de subtração da imagem de fundo estimada a partir da imagem corrente. A porta de controle habilita a leitura das FIFOs que armazenam a imagem corrente (ambiente Qsys) e a leitura das FIFOs que armazenam a imagem de fundo estimada. Convencionou-se realizar a subtração e a segmentação em um único passo. Deste modo, a porta de controle também habilita a escrita nas FIFOs posteriores à segmentação, conforme será visto a seguir.

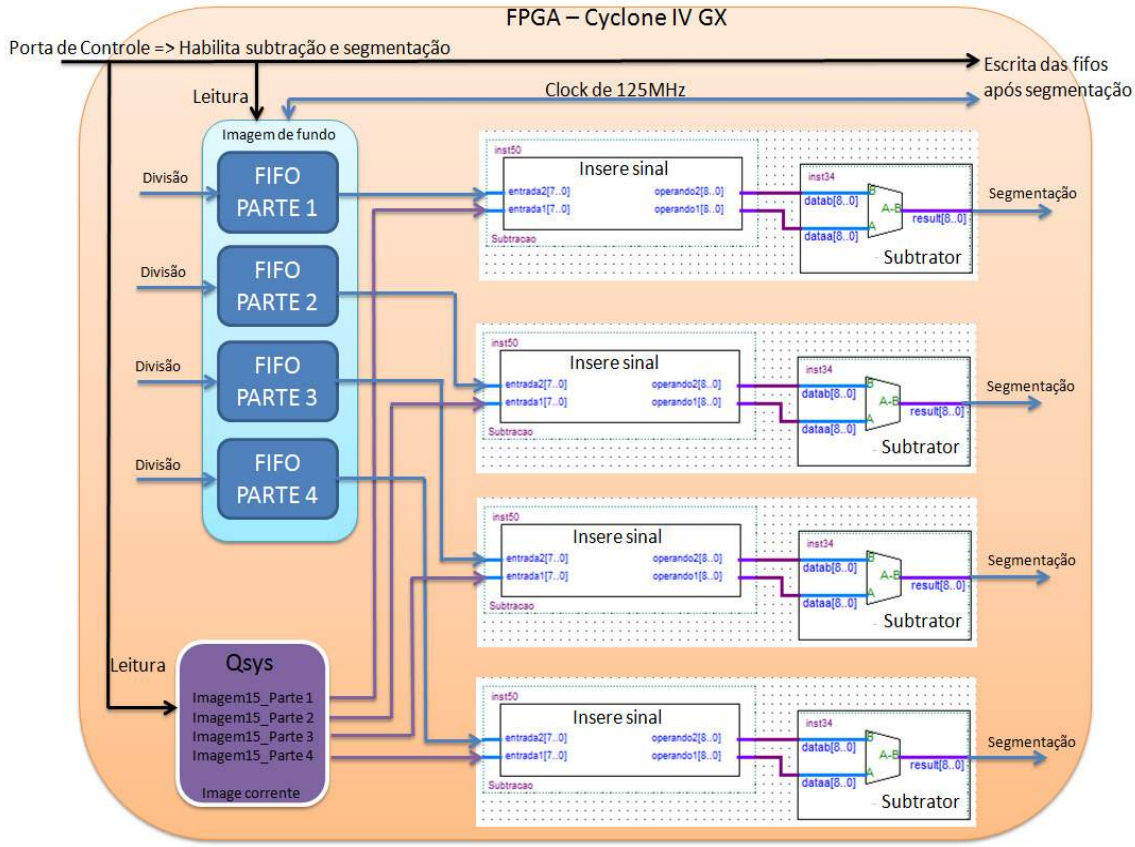



**Figura 4-15 – Etapa de subtração da imagem de fundo a partir da imagem corrente.**

Os subtratores foram inseridos na programação do FPGA no ambiente do Quartus II através do *MegaWizard Plug-In.* As entradas de dados foram configuradas para o formato *signed*. Deste modo, os subtratores interpretam o minuendo e subtraendo no formato de complemento de 2 (ALTERA CORPORATION, 2013). A representação de números com sinal usando complemento de 2, significa que o bit mais significativo (*MSB – most significant bit)* seja utilizado para representar o sinal, ou seja, *MSB* terá valor zero para números positivos e terá valor um para números negativos. Deste modo, foi necessária a inserção do bit de sinal nos valores dos *pixels* que formam a imagem de fundo e a imagem corrente. Para isto, foi utilizada uma função escrita em VHDL, que pode ser vista na Figura 4-15, referenciada como "Insere sinal". Esta função insere um zero na posição *MSB* dos *pixels* de entrada resultando em palavras de 9 bits. Assim, os subtratores realizam a subtração de um valor positivo (*pixel* da imagem estimada) a partir de outro valor positivo (*pixel* da imagem corrente).

## Segmentação

A última etapa no processamento embarcado das imagens é a segmentação. Esta etapa recebe os *pixels* resultantes da subtração da imagem de fundo, a partir da imagem corrente e os compara com um limiar de *Threshold* estabelecido, resultando em um valor igual a 255 para *pixels* com valores maiores ou iguais ao limiar e em um valor igual a zero para *pixels* de valor menor que o limiar. A Figura 4-16 mostra a etapa de segmentação, que é realizada a partir de uma função

programada em VHDL e referenciada nesta figura como “Segmentação”. A porta de controle habilita a escrita das FIFOs responsáveis por armazenar a imagem resultante do processamento embarcado. A porta de controle configura também o Limiar de *threshold.*

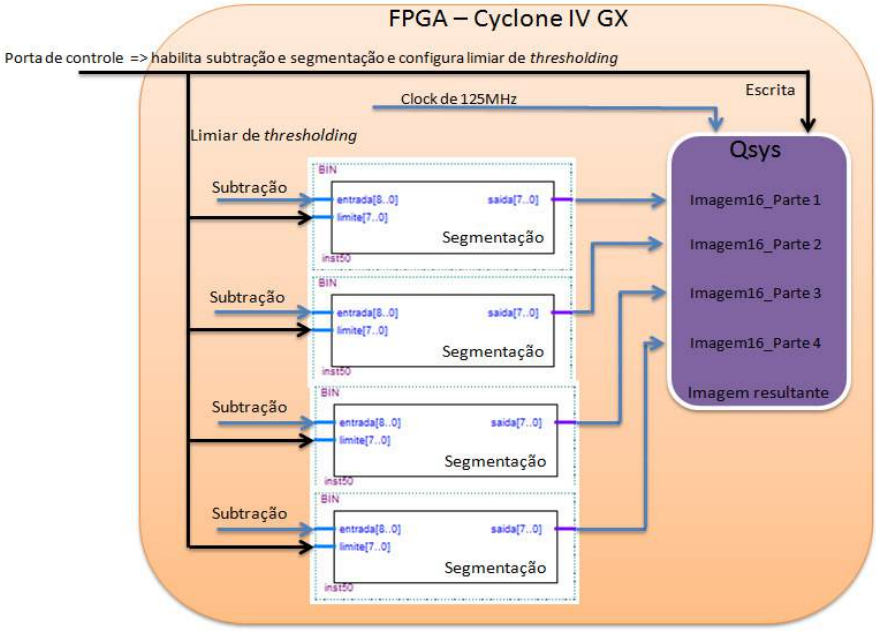



**Figura 4-16 – Etapa de segmentação.**

A função de segmentação apresenta entradas de dados de 9 bits, já que a subtração da imagem de fundo a partir da imagem corrente fornece valores em complemento de 2, ou seja, o *MSB* representa o sinal. A função de segmentação descarta o *MSB* dos valores de entrada, ou seja, o bit de sinal, e aplica o limiar apenas nos 8 bits menos significativos. A imagem resultante do processamento embarcado armazenada nas últimas quatro FIFOs, “Imagem16_Parte1”, “Imagem16_Parte2”, ”Imagem16_Parte3” e “Imagem16_parte4”, é então transferida para o computador através da interface de acesso, conforme mostrado nas próximas seções. Semelhante à etapa de soma, o intervalo de tempo de execução para a etapa formada pela subtração e segmentação também é dividido por quatro.

## Medição do intervalo de tempo de execução para as etapas do algoritmo embarcado

O intervalo de tempo de execução para cada uma das etapas do algoritmo de processamento de imagens embarcado deverá ser igual ao período de *clock* multiplicado pela quantidade de *pixels* que compõem um bloco de uma imagem, isto é, 7440. Espera-se que este intervalo de tempo de execução seja determinístico, já que os algoritmos estão sendo executados diretamente em *hardware* dedicado. Conforme descrito nas seções anteriores, as etapas do algoritmo estão localizadas entre as FIFOs que armazenam os resultados gerados por cada um dos cálculos. O *clock*

de escrita e leitura das FIFOs mencionadas foi definido como 125MHz. Portanto, espera-se que o intervalo de tempo de execução para cada etapa seja igual: $8ns \times 7440 = 59{,}52\mu s$ .

Para validação do tempo de execução esperado, mediu-se o intervalo de tempo entre a alternância dos *flags* de *status FIFO_QuaseCheia* para as FIFOs localizadas antes e após cada etapa de execução, para um dos blocos que compõe uma imagem, conforme ilustra a Figura 4-17. O instante de tempo que o *flag* de *status FIFO_QuaseCheia* da FIFO anterior ao cálculo transitar de "1" para "0" indica o ínício do cálculo para a etapa avaliada. Já o instante de tempo que o *flag* de *status FIFO_QuaseCheia* da FIFO posterior ao cálculo transitar de "0" para "1" indica o fim do cálculo para a etapa avaliada. Para medição do tempo de transferência das imagens entre o computador e o plataforma de desenvolvimento, os *flags* anteriormente mencionados foram exportados para fora do FPGA e associados a pinos presentes na plataforma de desenvolvimento, para monitoração com osciloscópio. A possibilidade do controle independente de execução para cada etapa do algoritmo embarcado permite realizar a medida de tempo de forma controlada e independente para cada cálculo realizado.

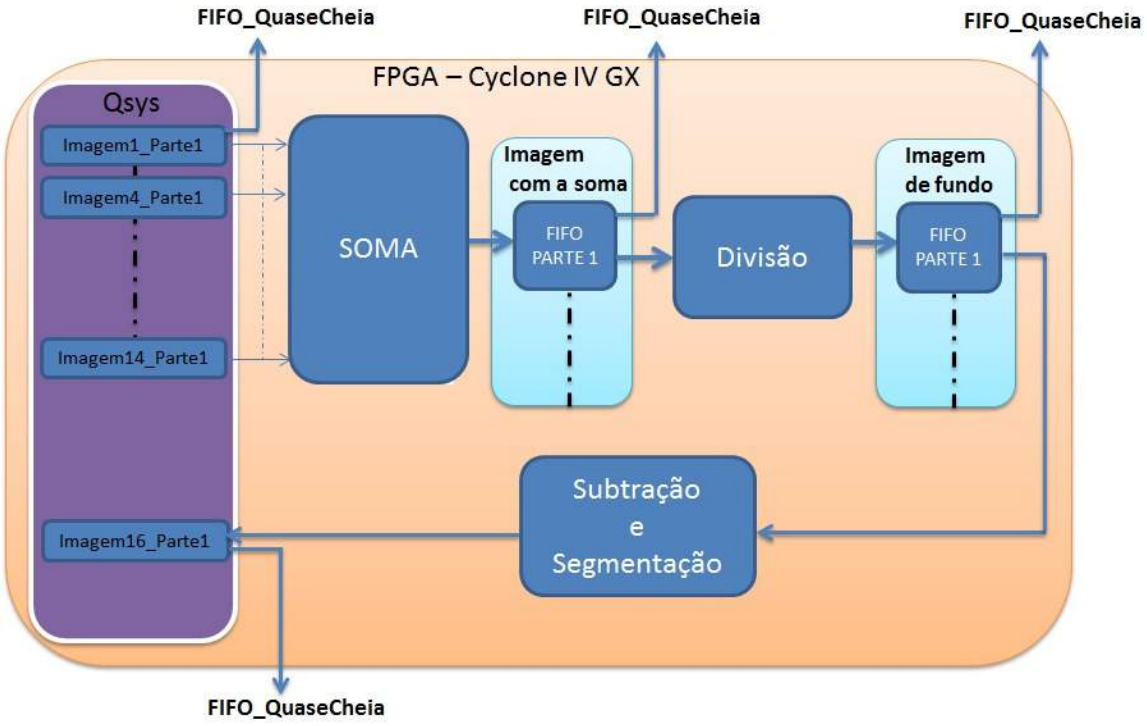



**Figura 4-17 – Medição para validação do tempo de execução de cada etapa embarcada**

### 4.2.2 Interface para transferência das imagens, monitoramento de memória e validação dos algoritmos de processamento embarcados

A interface de acesso desenvolvida para a transferência de imagens e monitoramento dos recursos de memória embarcada foi alterada para conter novas funcionalidades. Todas as etapas de

processamento de imagens realizadas no FPGA foram inseridas na interface de acesso. Isto possibilita fazer uma comparação dos resultados obtidos em *hardware* com resultados obtidos em *software*. Além disto, a interface de acesso deve permitir o controle de execução de cada passo do algoritmo embarcado, além de possibilitar que seja especificado o limiar de *threshold* utilizado pelo algoritmo de segmentação no FPGA. A Figura 4-18 mostra a tela inicial da interface desenvolvida para acesso ao FPGA. Percebe-se que na aba "Operação Local" agora é possível realizar em *software* as etapas do algoritmo de imagens embarcado em *hardware.* Além da resposta através de imagens, conforme visto na Figura 4-18, o *software* também armazena em três diferentes vetores, os valores dos *pixels* para cada uma das etapas realizadas, i.e., imagem de fundo estimada, imagem SAv e imagem binária. Este armazenamento permite a comparação destes valores com aqueles retornados do FPGA. Na aba "Comunicação com FPGA" além das funcionalidades explicadas na Seção 4.1.3, para monitoramento dos recursos de memória e transferência de imagens, foram inseridos comandos que permitem o controle de execução de cada etapa do algoritmo embarcado, conforme visto na Figura 4-19.

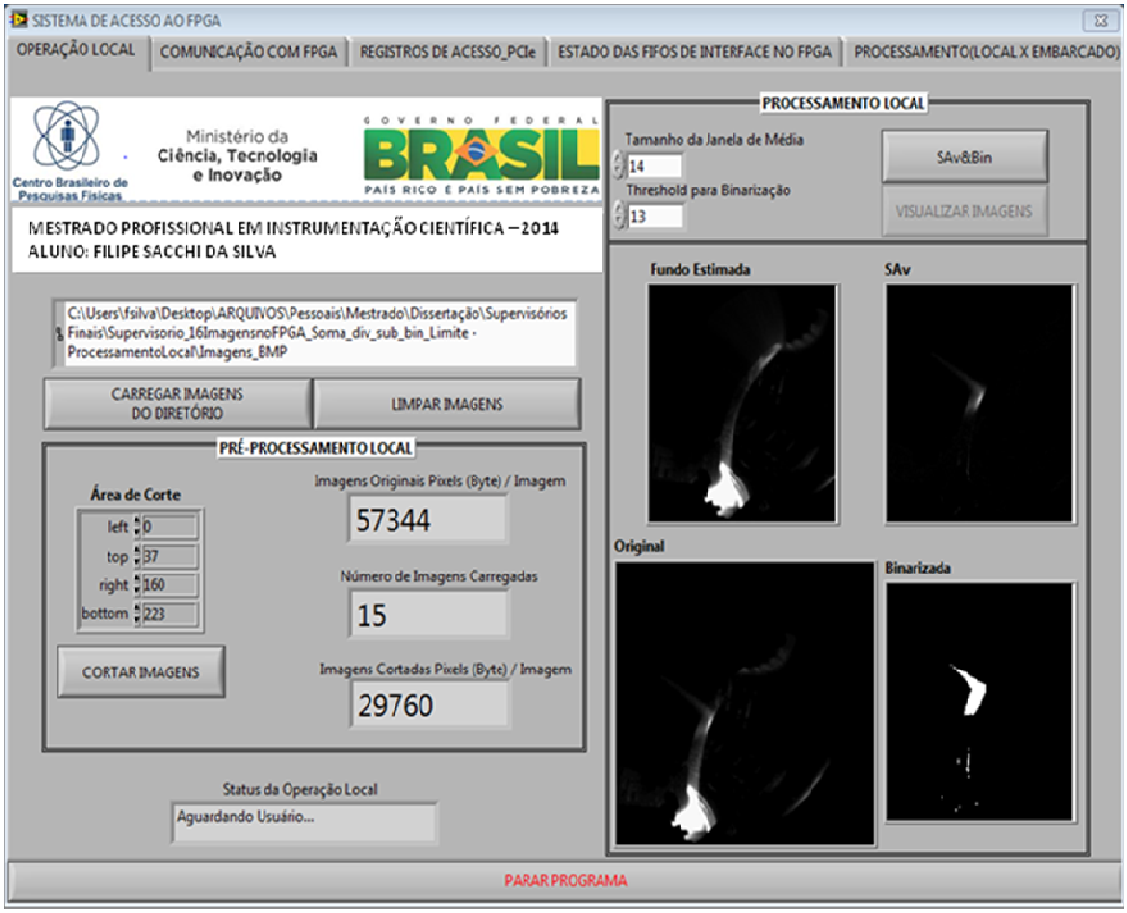



**Figura 4-18 – Visão Geral do programa desenvolvido para etapa de processamento de imagens embarcado.**

O retorno da imagem resultante para a interface de acesso gera um vetor de *pixels*, que pode também ser exibido diretamente em formato de imagens, conforme Figura 4-19. Este vetor é formado por 29760 *pixels.* Por questão de espaço, a interface apresenta apenas seis destes *pixels* simultaneamente. No entanto, todos os valores podem ser verificados através do controle de indexação. A comparação para análise dos vetores gerados pelo processamento local com o vetor produzido pelo processamento embarcado é realizada através da aba "Processamento (Local x Embarcado)".

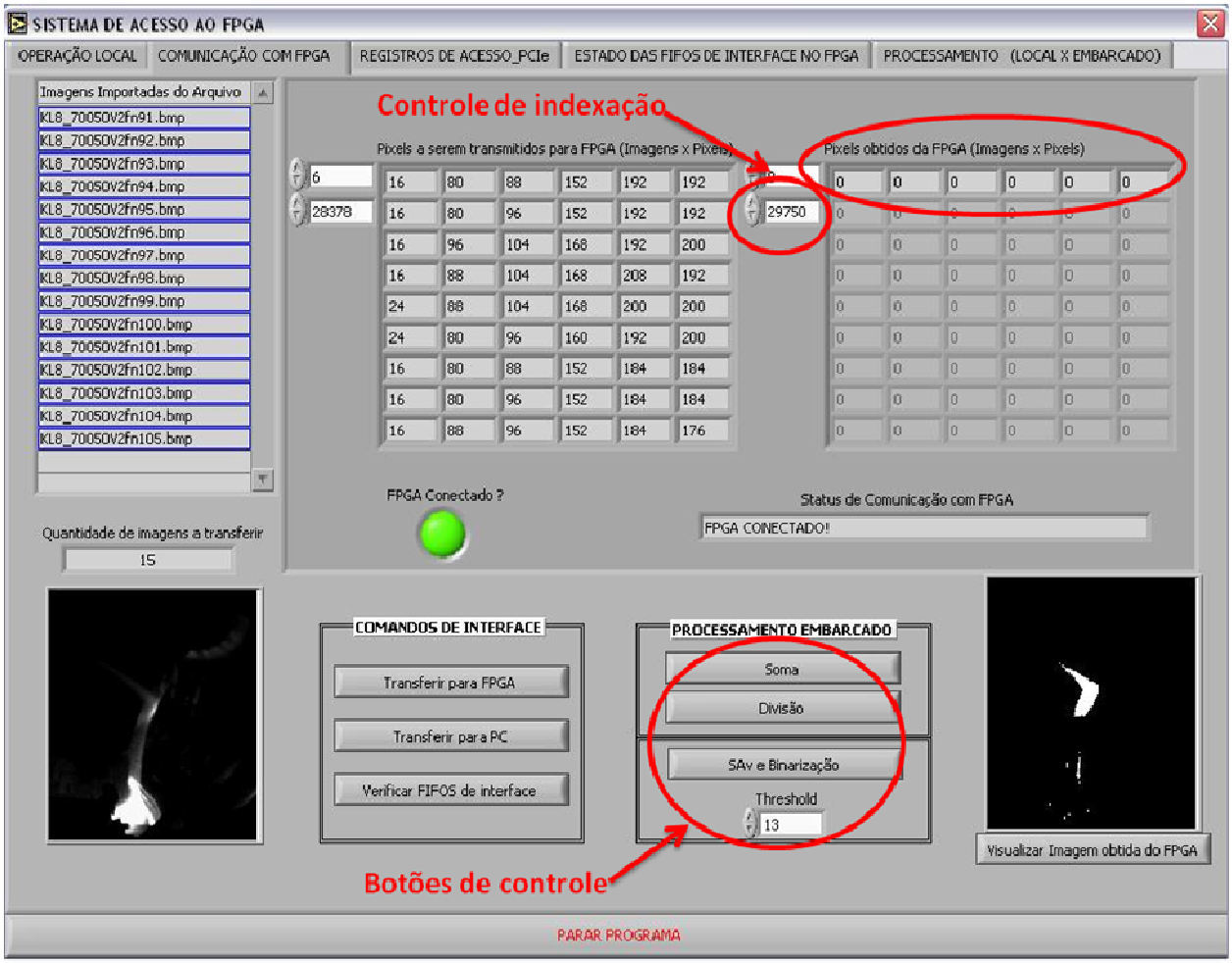



**Figura 4-19 – Visão da interface de acesso ao FPGA.**

# Capítulo 5

# Resultados

O sistema desenvolvido para o processamento de imagens embarcado no FPGA é constituído pelo processo de transferência das imagens entre computador e o plataforma de desenvolvimento e pelas etapas do algoritmo embarcado no *hardware.* Espera-se obter uma estimativa para o tempo de transferência das imagens, que permita uma avaliação dos componentes de *hardware* e *software* utilizados neste processo. Para os algoritmos de processamento de imagens, é necessária uma comparação dos resultados obtidos em FPGA com os obtidos em *software.* Além disto, o tempo de execução de cada etapa do processamento embarcado deve ser confrontado com o valor esperado de $59{,}52\mu s$, conforme discutido na subseção da Seção 4.2, dedicada à medição do intervalo de tempo de execução das etapas do algoritmo embarcado.

O entendimento dos resultados obtidos tem por objetivo a proposta de melhorias para o sistema desenvolvido, a fim de que em trabalhos futuros se consiga superar os desafios de hoje.

## 5.1 Resultados e análise para transferência de imagens entre computador e plataforma de desenvolvimento

A transferência dos dados para a memória do dispositivo de lógica programável foi validada a partir da utilização do Sistema de Acesso, desenvolvido neste trabalho. A confirmação da integridade dos dados recebidos e armazenados na memória do dispositivo foi possível a partir da verificação das imagens retornadas para o aplicativo (no LabVIEW). A Figura 5-1 apresenta as etapas de avaliação do procedimento de transferência. A Figura 5-2 mostra a tela de *status* após o envio das imagens. Nesta figura, é possível perceber a quantidade de elementos para cada uma das FIFOs, juntamente com seus estados atuais.

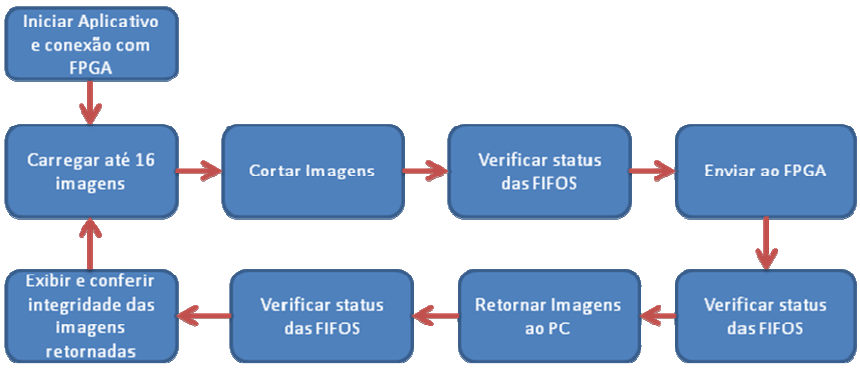



**Figura 5-1 – Esquema para teste de transferência das imagens**

Quando o procedimento de retorno é realizado, uma matriz de dados contendo as imagens obtidas é preenchida no Sistema de Acesso. Esta matriz possui até *n* x 29760 elementos, onde *n* é a quantidade de imagens retornadas do FPGA. Esta matriz de *pixels* pode ser então apresentada no formato de uma imagem, utilizando-se o botão "*Visualizar Imagens obtidas do FPGA*", na tela do Sistema de Acesso. Outra característica é que a interface informa também a matriz de *pixels* antes do envio para o FPGA, permitindo assim uma verificação dos dados antes e após a transmissão. A Figura 5-3 mostra a aba da interface com as matrizes anteriormente mencionadas, e a visualização dos *pixels* em formato de imagens.

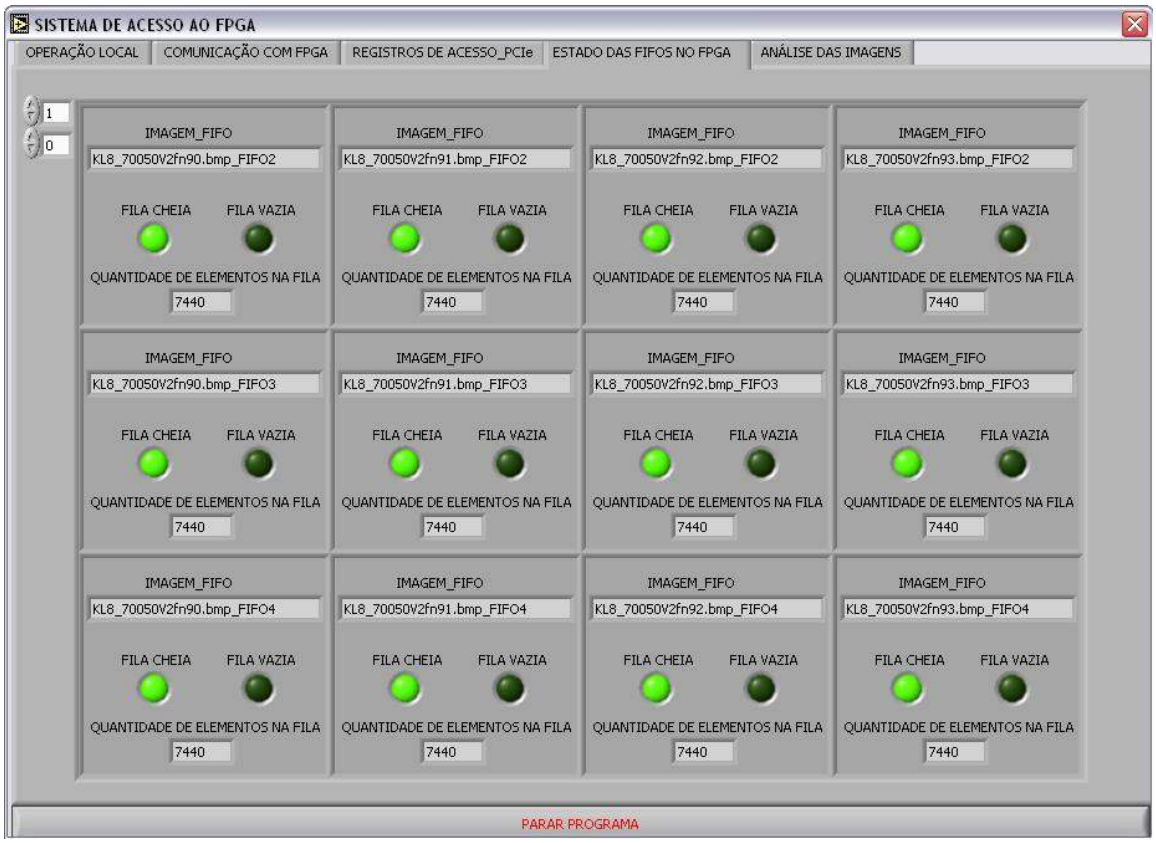



**Figura 5-2 – Verificação dos status e da quantidade de elementos para as FIFOs embarcadas no FPGA após a transmissão das imagens.**

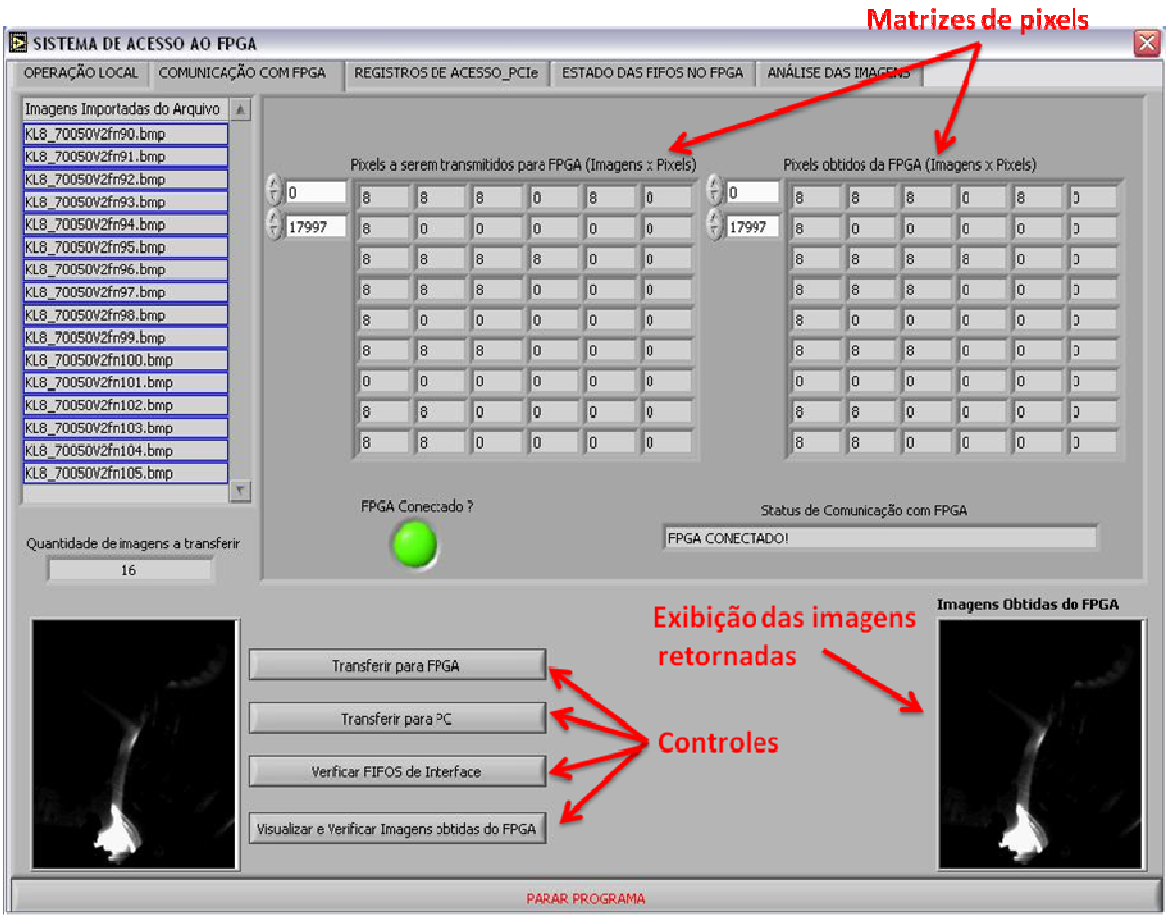



**Figura 5-3 – Sistema de acesso ao FPGA: visualização das imagens retornadas do FPGA**

Durante os testes de transferência foram realizadas medições de tempo de envio de um conjunto de 7440 *pixels* (7440 *Bytes*), através de um osciloscópio Tektronix modelo TDS 2022B. Foram medidos os sinais de *flags* de *FIFO_QuaseCheia* e *FIFO_Vazia*, para a primeira FIFO (i.e. Imagem1_Parte1). A Figura 5-4 mostra um exemplo de captura dos sinais com o osciloscópio. Nesta mesma figura o intervalo de tempo é diretamente obtido com utilização de cursores posicionados nos instantes em que sinais dos *flags* de *status* começam a mudar de nível.

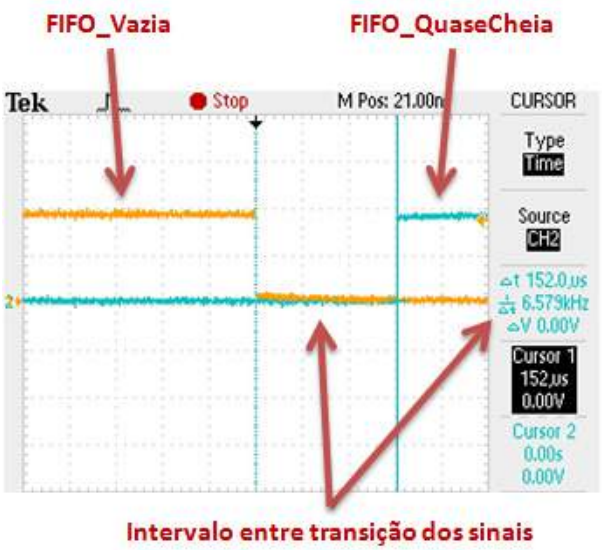



**Figura 5-4 – Medição do intervalo de tempo entre os *flags* de status**

Foram realizadas 100 transferências a fim de se obter a média e o desvio médio para o intervalo de tempo medido. Estes dados medidos são mostrados no Histograma 5-1. A partir dos dados coletados, o valor médio obtido para o tempo de transferência e armazenamento de um bloco de 7440 *pixels* foi de 152,12 µs**.** Assim pode-se estimar o tempo para uma imagem completa formada por quatro blocos de 7440 *pixels* e se obter a taxa em *MBytes/s* do sistema de transferência e armazenamento de imagens. Estes dados encontram-se na Tabela 5-1.


**Histograma 5-1 – Valores medidos para tempos de transferência de um bloco de dados com 7440 *pixels* (7440 bytes)**

O valor obtido para taxa de transferência/armazenamento de 48,91 *MBytes/s* para um bloco de 1860 elementos transmitidos é esperado quando comparado com dados de *benchmark* obtidos junto com a equipe de suporte técnico da *National Instruments*, para a transferência de blocos de

dados utilizando a função vi*MOVEOUT32*. Estes dados podem ser vistos na Tabela 5-2. Para correta comparação deve-se ter em mente que este teste de *benchmark* é antigo, tendo sido o único encontrado. Este foi realizado com uma plataforma computacional com *clock* da ordem de 400MHz, bem inferior ao que foi utilizado neste trabalho. O que justifica assim o maior valor encontrado para as medições aqui realizadas.

| | |
|---|---|
| **Tempo para um bloco de 7440 pixels (µs)** | **152,12** |
| **Tempo para uma imagem completa (µs)** | **608,48** |
| **Taxa de Transferência/Armazenamento (*MBytes/s*)** | **48,91** |

**Tabela 5-1 – Intervalo de tempo para bloco de 7440 *pixels,* para uma imagem completa (4 blocos de 7440 *pixels*) e taxa de transferência e armazenamento dos dados em *MBytes/s.***

| **Quantidade de elementos transmitidos** | **Taxa (*MBytes/s*)** |
|---|---|
| 1024 | 2,90 |
| 2048 | 5,00 |

**Tabela 5-2 – Dados de *benchmark* obtidos junto à *National Instruments* para transferência de dados utilizando a função vi*MOVEOUT32.***

O desvio médio para os dados medidos é de 21,02µs, o que resulta em uma taxa de transmissão de $1643 \pm 230$ imagens/s. Quando observado todo o sistema de transferência das imagens, conforme Figura 4-8, vê-se que o desvio médio encontrado, na ordem de 14% do valor médio, provavelmente está associado à variação nos tempos de execução do *driver* VISA durante suas interações com o barramento PCIe. Esta variação pode ser explicada pela não utilização de DMA na transmissão dos dados, assim a cada interação do *driver* VISA para escrita em rajada de 32 bits, a CPU do computador é exigida para interface entre memória RAM e barramento PCIe, o que para sistemas não dedicados, apresenta tempos variáveis. Além disto, a não utilização de DMA impede que o barramento PCIe que é compartilhado, seja alocado para uma transmissão contínua de todo os 1860 elementos de 32bits (7440 bytes).

## 5.2 Resultados e análises para os algoritmos de processamento de imagens no FPGA

Para avaliação das etapas do processamento de imagens realizado em *hardware*, e conforme descrito no capítulo anterior, foram gerados três diferentes projetos para a programação do FPGA: (i) estimação da imagem de fundo; (ii) subtração da imagem de fundo a partir da imagem corrente; (iii)

controle de todo processo. O vetor que contém os *pixels* resultantes da execução de cada uma destas programações é retornado para a interface de acesso. Esta realiza em *software* o mesmo processamento de imagens feito em *hardware* e produz como resultado os mesmos três vetores de *pixels* gerados pelas três programações mencionadas. Deste modo, é possível comparar e avaliar os resultados de *hardware* e *software* para cada etapa do algoritmo embarcado.

### 5.2.1 Estimação da imagem de fundo

A partir desta programação foi possível realizar a comparação entre os vetores de *pixels* para a imagem de fundo estimada gerada pelo FPGA e pela interface de acesso. A primeira comparação levou a uma quantidade de 7875 elementos diferentes entre os dois vetores analisados, ou seja, 26,5% do total de *pixels* que compõem uma imagem. Este elevado valor levou a uma análise mais apurada dos vetores de *pixels* comparados. Conforme Figura 5-5 constatou-se que o vetor de *pixels* retornando do FPGA para cada um dos blocos que forma uma imagem completa apresentou seus valores deslocados em 3 posições quando comparados com o vetor de *pixels* produzido pela interface de acesso.

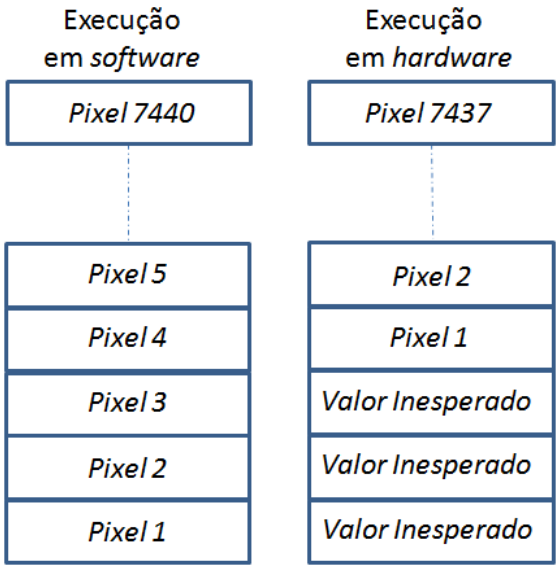



**Figura 5-5 – Comparação entre vetores de *pixels* gerados pelas execuções em *software* e *hardware*. O vetor de *pixels* gerado pela execução em *hardware* está deslocado em 3 posições quando comparado com o vetor produzido em *software*.**

Este deslocamento ocorre no momento da escrita em cada uma das FIFOs intermediárias que compõem o processamento embarcado. Na programação realizada para teste da estimação da imagem de fundo existem 3 FIFOs entre os processos de soma, divisão e retorno da imagem resultante para cada um dos blocos que compõem a imagem, assim observou-se o deslocamento em três posições para cada um dos blocos da imagem resultante.

Diante do fato mencionado anteriormente, modificou-se a função executada na interface de acesso que recebe os dados do FPGA. Foi feito o correto alinhamento do vetor de *pixels* recebidos do

*hardware* removendo os três primeiros elementos de valor inesperado e inserindo os mesmos nas três últimas posições. Deste modo, para a etapa de estimação da imagem de fundo esperou-se obter uma divergência máxima de 12 *pixels* (3 últimos *pixels* de cada um dos blocos da imagem) entre as execuções em *hardware* e *software*.

Após a comparação com os vetores de *pixels* corretamente alinhados, obteve-se apenas 3 elementos divergentes, o que representa 0,01% da quantidade total de *pixels* em uma imagem completa. Como era de se esperar, os três *pixels* divergentes estão localizados nas três últimas posições de um dos blocos da imagem. A necessidade de sincronismo dos *pixels* que formam a imagem de fundo estimada levou a inserção de um deslocamento (indexação) na programação feita para a etapa de subtração da imagem de fundo, conforme mostrado na Figura 5-6.

### 5.2.2 Subtração da imagem de fundo a partir da imagem corrente

Antes da comparação dos vetores de *pixels* gerados pelas execuções em *software* e *hardware*, programou-se no FPGA um mecanismo de sincronismo entre as FIFOs que contem a imagem de fundo estimada e a imagem corrente. Conforme descrito na seção anterior, sabe-se que para a programação realizada, cada processo de escrita em uma FIFO intermediária provoca o deslocamento em 1 posição dos *pixels* que formam um dos blocos da imagem. Assim, de acordo com a programação feita para o FPGA e descrita no capítulo anterior, cada uma das FIFOs que contem a imagem de fundo estimada apresenta seus elementos deslocados em duas posições. Para a correta subtração dos elementos, fez-se o sincronismo das posições dos *pixels* da imagem de fundo e da imagem corrente, conforme esquema mostrado na Figura 5-6.

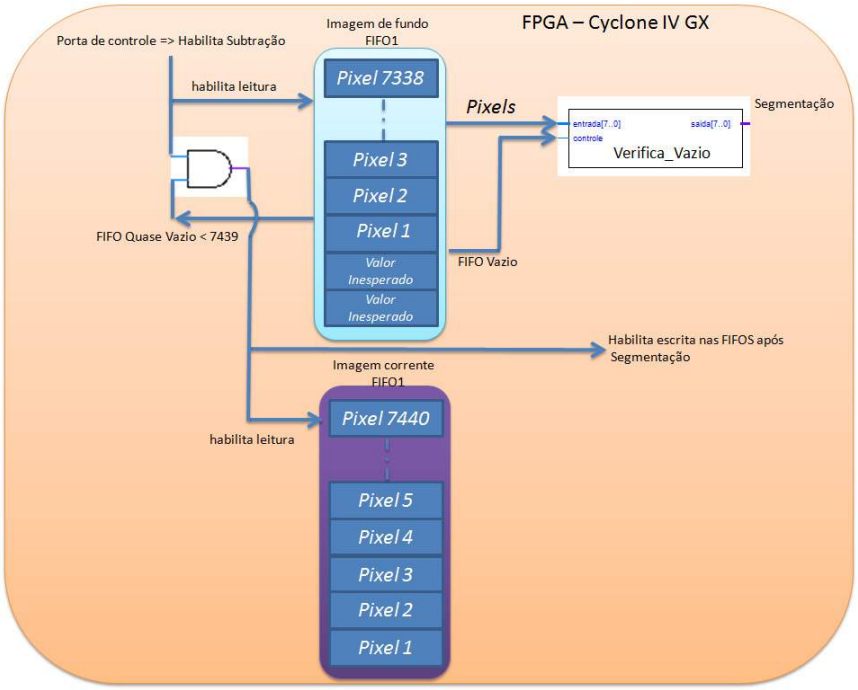



**Figura 5-6 – Esquema de sincronização dos pixels da imagem de fundo e imagem corrente. Os dois primeiros *pixels* de valor zero para imagem de fundo são inseridos devido ao processo de escrita nas duas FIFOs anteriores. Este procedimento descarta estes valores na subtração.**

O sincronismo é realizado com a leitura do *flag* de *status FIFO_QuaseVazia* para as FIFOs que contém a imagem de fundo estimada. Este *flag* foi configurado para possuir valor 1 quando a quantidade de elementos da referida FIFO é menor que 7439 elementos. Deste modo, consegue-se a subtração correta entre os *pixels.* Este método tem como desvantagem a necessidade de inclusão de uma função que verifica quando as FIFOs que contêm a imagem de fundo estão vazias, já que quando todos os elementos destas FIFOs forem lidos ainda restarão os dois *pixels* finais nas FIFOs que contem a imagem corrente. A função "Verifica_Vazio" programada em VHDL e vista na Figura 5-6, retorna o valor zero, caso sua entrada de controle (FIFO_Vazio) receba "1", ou retorna o valor do *pixel* do entrada, caso receba "0". Espera-se uma divergência de até 12 *pixels* na comparação entre os resultados da subtração da imagem de fundo para as execuções em *software* e *hardware.* Para cada bloco da imagem, o sincronismo necessário para a correta subtração insere até 2 *pixels* diferentes. A escrita na última FIFO para transferência da imagem resultante ao computador insere até 1 *pixel* diferente.

A comparação entre os vetores de *pixels* produzidos pelas execuções em *software* e *hardware* resultou em apenas 2 valores diferentes, que representam 0,0067% de uma imagem completa.

### 5.2.3 Segmentação

Conforme descrito no capítulo anterior, a segmentação recebe diretamente os valores resultantes da subtração da imagem de fundo. Assim, a quantidade máxima de elementos diferentes

após a comparação de execução em *software* e *hardware* será de 12 *pixels*. No entanto, para as imagens analisadas, a subtração da imagem de fundo resultou em apenas 2 *pixels* diferentes. A Tabela 5-3 abaixo mostra os valores destes *pixels* para as execuções em *software* e *hardware.*

Segundo (CHACON, 2012), o valor de *threshold* ótimo para a segmentação no processamento das imagens de MARFE analisadas deve ser igual a 13. Observando os valores apresentados, vê-se que, ambos os valores da primeira linha encontram-se acima do *threshold* e ambos os valores da segunda linha encontram-se abaixo deste limiar, deste modo, a etapa de segmentação ignora os pequenos erros produzidos pelas etapas anteriores.

| ***Pixel*** | **Execução em Hardware** | **Execução em Software** |
|---|---|---|
| 1 | 84 | 103 |
| 2 | 0 | 3 |

**Tabela 5-3 – *Pixels* diferentes gerados para a subtração da imagem de fundo pelas execuções em *Hardware e Software.***

Deste modo, quando executada toda a cadeia de processamento de imagens em *hardware*, os *pixels* resultantes serão todos iguais àqueles produzidos pela execução em *software* do mesmo algoritmo. Destaca-se que a quantidade de *pixels* resultantes diferentes do esperado dependerá do conjunto de imagens utilizado, no entanto esta quantidade nunca será superior a 12 *pixels.*

### 5.2.4 Intervalo de tempo de execução para as etapas do algoritmo embarcado

Conforme descrito no capítulo anterior, o intervalo de tempo de execução para as etapas do algoritmo embarcado depende do *clock* utilizado para leitura das FIFOs localizadas antes e após cada cálculo realizado e da quantidade de *pixels* que compõe um dos quatro blocos que formam uma imagem. Por se tratar de uma execução em *hardware,* onde não existe a necessidade de sistema operacional, espera-se que o intervalo de tempo de execução seja praticamente invariável e muito próximo ao esperado para todas as vezes que se executem os algoritmos, caracterizando assim uma execução em tempo real. Para validação do anteriormente exposto, mediu-se o intervalo de tempo entre a alternância dos *flags* de *status FIFO_QuaseCheia* para as FIFOs localizadas antes e após cada etapa de execução para um dos blocos que compõe uma imagem.

Para permitir a visualização no osciloscópio, a escala do tempo foi configurada para $25\mu s / divisão$. Assim, espera-se observar um valor medido de $60\mu s$, para o intervalo de tempo de execução de cada etapa.

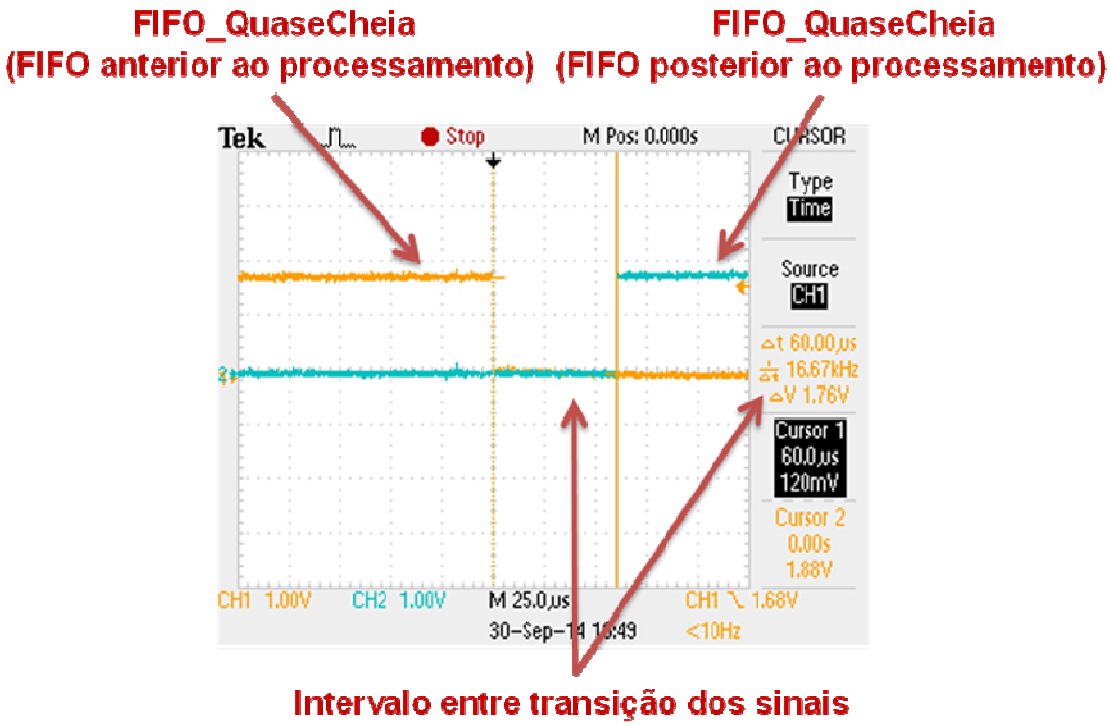



**Figura 5-7 – Medição do intervalo de tempo entre os *flags* de status. O intervalo de tempo de 60µS é constante para todas as execuções das etapas do algoritmo embarcado, caracterizando assim um sistema determinístico.**

O intervalo de tempo de $60\mu s$ medido, conforme Figura 5-7, comprova o esperado: o intervalo de tempo de execução do cálculo sobre cada *pixel* que compõe um dos quatro blocos da imagem é dado por um ciclo de *clock.* Observando a arquitetura de programação utilizada para o FPGA, conforme Figura 4-17, tem-se que os intervalos de tempo de execução para a etapa de estimação de imagem de fundo seguido pela etapa de subtração e segmentação serão respectivamente: $119{,}04\mu s$ e $59{,}52\mu s$. Deste modo, o tempo total de execução será de $178{,}56\mu s$, que resulta em uma taxa de processamento de 5600 imagens/s.

Para a correta análise da taxa de execução do sistema desenvolvido no presente trabalho, deve-se considerar que além da execução dos algoritmos, existe também a etapa de transferência das imagens entre o computador e a plataforma de desenvolvimento. Conforme Figura 5-8, a taxa de execução será limitada pelo processo mais lento, que no caso é a transferência das imagens, resultando assim, para um melhor caso, em uma taxa de execução do sistema de 1873 imagens/s. Neste mesmo gráfico são mostradas as etapas de execução determinística e não determinística, para o primeiro caso tem-se os algoritmos embarcados de execução puramente em *hardware,* já para o segundo caso, tem-se a etapa de transferência das imagens que sofre grande influencia do *driver* de comunicação executado no sistema operacional MS-*Windows*.

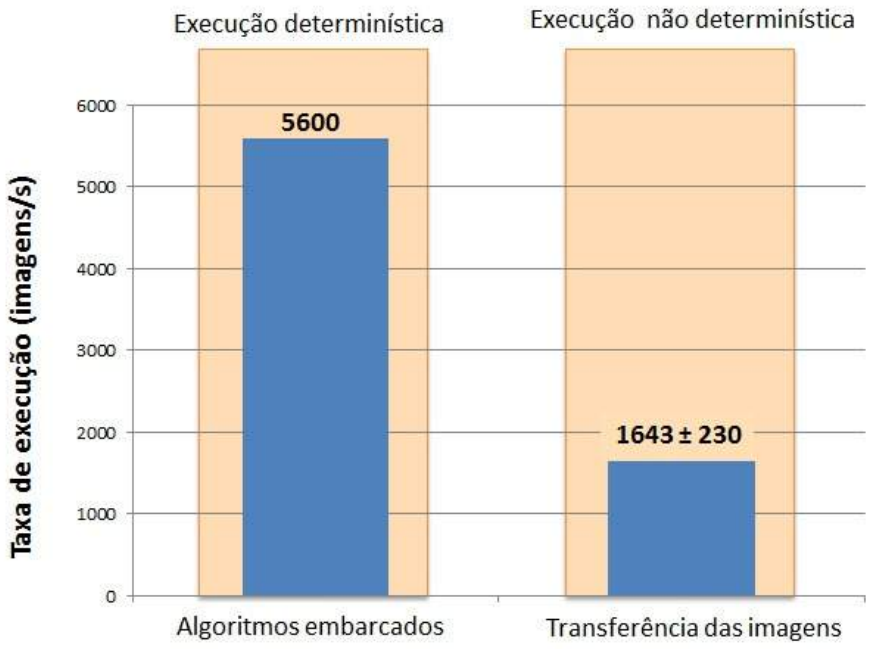



**Figura 5-8 – Intervalos de tempo para as etapas de transferência das imagens e execução dos algoritmos embarcados no FPGA.**

Quando observado os intervalos de tempos de execução para os algoritmos em *software* desenvolvidos pelo CBPF, e considerando apenas as etapas que neste trabalho foram inseridas em *hardware*, tem-se, conforme capítulo 2, os respectivos valores para as abordagens serial e paralela: $871\mu s$ e $84\mu s$. Os valores anteriores resultam em taxas de execução de 1148 imagens/s e 11905 imagens/s. A taxa de execução para o sistema desenvolvido no presente trabalho é superior àquela encontrada para a abordagem de execução serial. No entanto, para que se supere o desenvolvimento paralelo já realizado, e para que a abordagem em FPGA possa ser aplicada na prática, serão necessárias melhorias, conforme descrito no capítulo seguinte, levando em conta não apenas a arquitetura de programação para o FPGA, mas também a plataforma de desenvolvimento utilizada.

# Capítulo 6

# Conclusões e perspectivas

Este trabalho propôs o estudo e avaliação da abordagem das etapas iniciais de uma cadeia de processamento de imagens através do uso de FPGA. O desenvolvimento aqui realizado se baseou em trabalhos previamente propostos pelo CBPF para a detecção em imagens sequênciais, em tempo real, do fenômeno *MARFE* que ocorre no interior de reatores de fusão nuclear. A contribuição deste trabalho é a caracterização de todo um ambiente de desenvolvimento em FPGA para aplicações de processamento de imagens que exigem operação em alta velocidade e/ou tempo real.

O processamento de imagens através de *hardware configurável* requer o desenvolvimento de uma metodologia para transferência destas entre o computador (ou câmera) e o dispositivo responsável pelo processamento embarcado. Para esta tarefa, desenvolveu-se neste trabalho, um método de transferência utilizando o barramento PCIe, para envio, a partir de um computador, das imagens a serem processadas no FPGA e recepção a partir deste, da imagem resultante do processamento.

A inserção em FPGA dos algoritmos de processamento que foram aqui avaliados (estimação da imagem de fundo e de sua subtração da imagem corrente e posterior segmentação em objetos) explorou a técnica de paralelismo de dados. As imagens foram divididas em blocos menores e o processamento foi aplicado de forma paralela em cada um destes blocos. A execução naturalmente paralela dos FPGAs permitiu explorar de forma mais eficiente, a técnica de paralelismo antes mencionada. O intervalo de tempo de execução para cada etapa de processamento avaliada foi de $59{,}52\mu s$, devido à implementação dos algoritmos em *hardware.* Esta é uma importante característica que reforça a utilização de *hardware* para problemas que requerem uma abordagem rápida e/ou tempo real, como é o caso da detecção de *MARFES,* onde este trabalho pode ser aplicado.

O *hardware* utilizado foi a plataforma de desenvolvimento comercial Altera Cyclone IV GX que contém um FPGA e interfaces de comunicação, dentre as quais o PCIe. Para a comunicação entre o computador e o FPGA foi desenvolvido um programa de acesso que permite: (i) a transferência de imagens entre computador e FPGA e vice-versa; (ii) monitoramento das memórias que fazem interface com o barramento PCIe para transferência de imagens; (iii) controle da execução das etapas do algoritmo embarcado; (iv) determinação do limiar de segmentação para o algoritmo embarcado; (v) para fins de comparação, execução em *software* do mesmo algoritmo de processamento de imagens executado em *hardware.*

Quando comparadas a imagem resultante da execução do processamento de imagens em *hardware* e *software* espera-se com o sistema desenvolvido uma diferença entre as duas de até 12 *pixels*. Esta diferença é devido ao processo de escrita nas FIFOs que compõem o sistema embarcado. Para as imagens analisadas, o processo de subtração de imagem de fundo em *hardware* produziu apenas 2 *pixels* diferentes daqueles encontradas para sua execução em *software.* No entanto, estes

*pixels* diferentes encontram-se ou acima ou abaixo do limiar de segmentação, deste modo, esta etapa classifica os *pixels* diferentes da mesma forma nos dois procedimentos usados nesta avaliação (*hardware e software).*

Quando observado todo o sistema desenvolvido, i.e., transmissão e armazenamento das imagens no FPGA, seguido pelas etapas do processamento embarcado, a taxa de processamento final será limitada por ambos (transferência e armazenamento). No melhor caso, só será possível alcançar 1873 imagens/s em comparação com a velocidade de execução do processamento embarcado que tem uma taxa de 5600 imagens/s. Quando observado as abordagens serial e paralelo de programação em *software* desenvolvidas em outros trabalhos do CBPF (CHACON, 2012) e (SOUZA, 2013), em comparação com as etapas avaliadas em *hardware* neste trabalho, aqueles alcançam taxas de 1148 imagens/s para primeira (serial) e 11905 imagens/s para a segunda (paralelo) respectivamente. Apesar da taxa de processamento de imagens atualmente alcançada neste trabalho, ser superior àquela obtida para a abordagem serial, melhorias podem ser desenvolvidas para que se supere a taxa obtida pela abordagem de programação paralela.

A primeira perspectiva de melhoria para o presente trabalho seria o aumento de velocidade na transferência das imagens, com a utilização de um *driver* de acesso que fornecesse suporte a troca de dados por meio de DMA. O protocolo PCIe Gen 1.0 tem como limite teórico taxas de transferência da ordem de 250*MBytes/s por Lane*. No entanto, para o desenvolvimento de um sistema de transferência de dados que consiga explorar de forma efetiva as reais taxas do barramento (taxas reais são mais baixas devido ao *overhead* introduzido pelo protocolo), deve-se atentar a todos os constituintes que formam o canal de transmissão deste sistema (i.e. não apenas ao protocolo de transmissão utilizado) (ALTERA CORPORATION, 2014). Por exemplo, a possibilidade ou não de se utilizar DMA no *driver* de acesso e a limitação de frequência de operação na recepção dos dados a partir do barramento PCIe devem também ser levadas em consideração.

Em (ALTERA CORPORATION, 2014) são apresentados dados de desempenho para o plataforma Cyclone IV, o mesmo utilizado neste trabalho, para transmissão via barramento PCIe entre computador e memórias embarcadas no FPGA, utilizando DMA. Neste relatório técnico são atingidas taxas de 220*MBytes/s* para a Gen1.0 do protocolo PCIe x1. Caso fosse possível atingir a taxa de transmissão mencionada anteriormente para este trabalho, o *clock* interno do FPGA para armazenamento dos dados na FIFO, configurado para 125MHz, limitaria o desempenho total do sistema de transferência e armazenamento de imagens para uma taxa teórica de 4200 imagens/s. Uma possível abordagem para aumentar a frequência de escrita nas FIFOs, é utilizar uma fonte de *clock* gerada fora do ambiente *Qsys* e então importá-la para dentro deste ambiente através de uma *clock bridge* (ALTERA CORPORATION, 2014B). Desta forma, foi compilado um projeto para programação do FPGA para uma frequência de 500MHz para escrita nas FIFOs dentro do ambiente *Qsys*. No entanto, a quantidade de elementos lógicos necessários, 300541, superou a quantidade de 149760 disponíveis na Plataforma Cyclone IV GX utilizado (ALTERA CORPORATION, 2014C).

A discussão anterior leva à segunda perspectiva de melhoria, a qual impõe a necessidade de utilização de um FPGA com mais recursos de *hardware*. Para este fim, poder-se-ia utilizar, por exemplo, a plataforma Stratix IV GX. Este plataforma pode chegar a 531200 blocos lógicos para o FPGA (ALTERA CORPORATION, 2012B), o que comportaria o projeto compilado para frequência de

500MHz de escrita nas FIFOs no ambiente *Qsys.* Outro ponto importante é que segundo (ALTERA CORPORATION, 2014), este Plataforma pode alcançar taxas de até 1706*MBytes/s* para Gen1.0 PCIe *x8*, em transmissões utilizando DMA. Deste modo, o processo de transferência e armazenamento das imagens no FPGA, para a plataforma proposta, alcançaria uma taxa de 16801 imagens/s.

Quando observado a execução dos algoritmos embarcados, para o presente trabalho, tem-se uma taxa de execução em 5600 imagens/s. Ademais, devido a restrições de memória embarcada no FPGA, para plataforma Cyclone IV GX utilizado, para a etapa de estimação da imagem de fundo, realiza-se o cálculo da média com 14 imagens anteriores. Este valor é abaixo do valor ótimo, que conforme apresentado anteriormente, é de 23 imagens anteriores. As características antes mencionadas reforçam a perspectiva de utilização de um FPGA com mais recursos de *hardware*, como é o caso da plataforma Stratix IV GX já citado. Este pode chegar a conter 2592kbytes de memória embarcada, o que permitiria a manipulação em *hardware* de até 79 imagens, mais que o triplo conseguido com a atual plataforma. Além disto, mais recursos de *hardware* permitiriam um aumento do paralelismo de dados, por exemplo, caso as imagens fossem divididas em 32 blocos de 930 *pixels*, ao invés de quatro blocos de 7440, como é feito atualmente. A taxa de processamento para as etapas do algoritmo embarcado alcançaria 44803 imagens/s, sem necessidade de aumento para o valor do *clock* atual.

A terceira perspectiva de melhoria está relacionada a uma diferente arquitetura de *hardware* programado no FPGA, além do paralelismo de dados já mencionado, poder-se-ia incluir a técnica de *pipeline* entre as duas tarefas: (i) cálculo da imagem de fundo seguido pela subração desta apartir da imagem corrente; (ii) segmentação. Neste caso, e considerando um paralelismo de dados com a divisão das imagens em 32 blocos de 930 pixels, poder-se-ia alcançar uma taxa de execução de 67204 imagens/s, para o mesmo valor de *clock* utilizado neste trabalho.

Diante das discussões realizadas e dos resultados encontrados e estimados, vê-se que os FPGAs são uma escolha natural para certas etapas de uma cadeia de processamento de imagens que exigem características de tempo real e altas taxas de execução dos algoritmos. Diferente do que ocorre em sistemas constituídos apenas de *software,* a busca pelos resultados desejados implica que as limitações de recursos de *hardware* em FPGAs, forcem uma escolha criteriosa do *hardware* a ser utilizado, da arquitetura a ser programada, e das características de comunicação que permitem integrar o *hardware* com os ambientes de *software* que compõem todo um sistema de processamento de imagens.

# Capítulo 7

# Referências bibliográficas

**ALBUQUERQUE, M. et al.** High Speed Image Processing Algorithms for Real Time Detection of MARFEs on JET. IEEE Trans. Plasma Sci, v. 40, n. 12, p. 3485-3492, Dec. 2012.

**ALBUQUERQUE, M. et al.** A 10000 image per second parallel algorithm for real time detection of Marfes on Jet. IEEE Trans. Plasma Sci., v. 41, n. 2, p. 341-349, Feb 2013.

**ALTERA CORPORATION.** Cyclone IV GX FPGA Development Board - Reference Manual. San Jose. 2010.

**ALTERA CORPORATION.** Embedded Peripherals IP - User Guide. San Jose. 2011.

**ALTERA CORPORATION.** IP Compiler for PCI Express - User Guide. San Jose. 2011B.

**ALTERA CORPORATION.** Quartus II Handbook Version 12.0. Sao Jose. 2012.

**ALTERA CORPORATION.** Stratix IV Device Handbook - Volume 1. [S.l.]. 2012B.

**ALTERA CORPORATION**. Integer Arithmetic Megafunctions User. Sao Jose. 2013.

**ALTERA CORPORATION.** PCI Express High Performance Reference Design. San Jose. 2014.

**ALTERA CORPORATION.** Qsys System Design Components - QII51025. San Jose. 2014B.

**ALTERA CORPORATION.** Cyclone IV Device Handbook - Volume 1. [S.l.]. 2014C.

**ALVES, D. et al.** A Real-Time Architecture for the Identification of Faulty Magnetic Sensors in the JET Tokamak. Nuclear Science, IEEE Transactions on, Lisboa, v. 61, n. 3, p. 1228-1235 , Jun 2014.

**AMDAHL, G.** Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. AFIPS Spring Joint Computer Conference. Atlantic City, New Jersey: [s.n.]. 1967. p. 483-485.

**ASHENDEN, P.** The Designer's Guide to VHDL (Systems on Silicon). 3rd. ed. [S.l.]: MORGAN KAUFMANN PUBLISHERS, 2008. ISBN 978-0120887859.

**AUER, S.** Imaging by dust rays: a dust ray camera. Optica Acta, Moorhead, v. 29, n. 10, p. 1421-1426, April 1982.

**BAILEY, D.** Design for Embedded Image Processing on FPGAs. 1st. ed. [S.l.]: John Wiley & Sons (Asia), 2011. 12 p.

**BATCHER, K.** Design of a Massively Parallel Processsor. Computers, IEEE Transactions on, v. C-29, n. 9, p. 836-840, September 1980.

**BITTNER, R.** Speedy Bus Mastering PCI Express. 22nd International Conference on Field Programmable Logic and Applications (FPL 2012). [S.l.]: [s.n.]. 2012.

**BROWN, S.; VRANESIC, Z.** Fundamentals of digital logic with VHDL design. 2nd. ed. [S.l.]: McGraw-Hill, 2008. ISBN 978-0077221430.

**CAO, T.; ELTON, D.; DENG, G.** Fast buffering for FPGA implementation of vision-based object recognition systems. Journal of Real-Time Image Processing, v. 7, p. 173-183, April 2012.

**CHACON, G.** Aplicação de técnicas de processamento digital de imagens para a detecção de MARFEs no JET. Centro Brasileiro de Pesquisas Físicas (Dissertação de Mestrado). Rio de Janeiro. 2012.

**CHACON, G. et al.** Aplicação da técnica de momentos invariantes no reconhecimento de padrões em imagens digitais. Centro Brasileiro de Pesquisas Físicas (Nota Ténica). Rio de Janeiro. 2011.

**COPE, B. et al.** Have GPUs made FPGAs redundant in the field of video processing? IEEE International Conference on Field-Programmable Technology. Singapore: [s.n.]. 2005. p. 111-118.

**CRISTIANINI, N.; TAYLOR, J.** An Introduction to Support Vector Machines and Other Kernel-based Learning Methods. 1. ed. [S.l.]: Cambridge University Press, 2000. ISBN ISBN-13: 978-0521780193.

**CUCCHIARA, R. et al.** Detecting moving objects, ghosts, and shadows in video streams. Pattern Analysis and Machine Intelligence, IEEE Transactions on, v. 25, n. 10, p. 1337 - 1342, October 2003. ISSN ISSN:0162-8828.

**CZYżEWSKI, A. et al.** Multi-stage Video Analysis Framework. Lin, W. (ed.) Video Surveillance. [S.l.]: Intech. 2011. p. 147–172.

**DILLINGER, P. et al.** FPGA-Based Real-Time Image Segmentation for Medical Systems and Data Processing. Nuclear Science, IEEE Transactions on , v. 53, n. 4, p. 2097-2101, Aug. 2006.

**DOWNTON, A.; CROOKES, D.** Parallel architectures for image processing. Electronics & Communications Engineering Journal, Colchester, v. 10, n. 3, p. 139-151, Jun 1998.

**DOWNTON, A.; CROOKES, D.** Parallel architectures for image processing. Electronics & Communication Engineering Journal, v. 10, n. 3, p. 139-151, Jun 1998. ISSN 0954-0695.

**FITZGERALD, D.; WILL, L.; WILL, D.** Real-time, parallel segmentation of high-resolution images on multi-core platforms. Journal of Real-Time Image Processing, May 2014. ISSN DOI 10.1007/s11554-014-0432-z.

**FLYNN, M.** Some Computer Organizations and Their Effectiveness. IEEE Transactions on Computer, v. C-21, n. 9, p. 948-960, September 1972.

**GAMBIER, A.** Real-time Control Systems: A Tutorial. 2004 5th Asian Control Conference. Melbourne, Victoria, Australia: [s.n.]. 2004. p. 1024- 1031.

**GONZALEZ, R.; WOODS, R.** Digital Image Processing. 3th. ed. [S.l.]: Pearson Prentice Hall, 2008.

**HSU, Y.; MIAO, H.; TSAI, C.** FPGA Implementation of a Real-Time Image Tracking System. SICE Annual Conference. Taipei: [s.n.]. 2010.

**LIPSCHULTZ, B. et al.** Marfe: an edge plasma phenomenon. Nucl. Fusion, v. 24, n. 8, 1984.[Online],Disponível:http://www.psfc.mit.edu/library1/catalog/reports/1980/83ja/83ja033/83ja033_full.pdf.

**MCIVOR, A.; ZANG, Q.; KLETTE, R.** The Background Subtraction Problem for Video Surveillance Systems. Lecture Notes in Computer Science, 2001. pp 176-183.

**MCKENNEY, P.** Real Time vs. Real Fast How to Choose? Proceedings of the 11th Linux Symposium. Dresden, Germany: [s.n.]. 2009.

**MURARI, A. et al.** Image Manipulation for High Temperature Plasmas. EFDA–JET. EFDA, Culham Science Centre, Abingdon, Oxon, OX14 3DB, UK. 2010.

**MURARI, A. et al.** Algorithms for the Automatic Identification of MARFEs and UFOs in JET Database of Visible Camera Videos. Plasma Science, IEEE Transactions on, v. 38, n. 12, p. 3409 - 3418, Outubro 2010A. ISSN ISSN:0093-3813.

**MURARI, A.** et al. Image Processing with Cellular Nonlinear Networks Implemented on Field-Programmable Gate Arrays for Real Time Applications in Nuclear Fusion. EFDA-JET. [S.l.]. 2010B.

**NATIONAL INSTRUMENTS.** NI-VISA User Manual. [S.l.]. 2001.

**NATIONAL INSTRUMENTS.** Introdução à Tecnologia FPGA. Disponivel online: http://www.ni.com/white-paper/6984/pt/. [S.l.]. 2011.

**NATIONAL INSTRUMENTS.** Using the NI-VISA Driver Wizard and NI-VISA to Develop a PXI(e)/PCI(e) Driver in Windows. Austin. 2013.

**NGUYEN, D. et al.** Real-time face detection and lip feature extraction using field-programmable gate arrays. IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART B: CYBERNETICS, v. 36, n. 4, p. 902-912, August 2006.

**PICCARDI, M.** Background subtraction techniques: a review. Systems, Man and Cybernetics, 2004 IEEE International Conference on (Volume:4 ). [S.l.]: IEE. 2004. p. 3099 - 3104 vol.4.

**QIANG, W. et al.** The research and implementation of interfacing based on PCI express. Electronic Measurement & Instruments, 2009. ICEMI '09. 9th International Conference on. Beijing: IEEE. 2009. p. 3-116 - 3-121.

**SAXENA, S.; SHARMA, N.; SHARMA, S.** Image Processing Tasks using Parallel Computing in Multi core Architecture and its Applications in Medical Imaging. International Journal of Advanced Research in Computer and Communication Engineering, v. 2, n. 4, April 2013.

**SHANLEY, T.; ANDERSON, D.; BUDRUK, R.** PCI Express System Architecture. [S.l.]: Addison-Wesley, 2003.

**SOUZA, F.** Lógica programável aplicada à aquisição e transmissão serial de dados em alta velocidade via barramento PCI Express. Centro Brasileiro de Pesquisas Físicas. Rio de Janeiro, p. 37. 2012.

**SOUZA, M.** Paralelismo computacional de processamento digital de imagens aplicado à detecção de MARFEs no JET. Centro Brasileiro de Pesquisas Físicas (Dissertação de Mestrado). Rio de Janeiro. 2013.

**SZWOCH, G.; ELLWART, D.; CZYżEWSKI,** A. Parallel implementation of background subtraction algorithms. Journal of Real-Time Image Processing, 2012. ISSN 10.1007/s11554-012-0310-5.

**TOCCI, R.; WIDMER, N.; MOSS, G.** Sistemas Digitais Princípios e Aplicações. 11. ed. [S.l.]: Pearson Prentice Hall, 2011.

**UMBAUGH, S.** Computer Vision and Image Processing: A Practical Approach Using Cviptools. [S.l.]: [s.n.], 1998.

**VXIPLUG&PLAY SYSTEMS ALLIANCE AND IVI FOUNDATION.** VPP-4.3: The VISA Library 5.1. [S.l.]. October 2012.

**WESSON, J.** THE SCIENCE OF JET The achievements of the scientists and engineers who worked on the Joint European Torus. s.l. : JET Joint Undertaking, Abingdon, Oxon, OX14 3EA. [S.l.]. 2006.

**XILINX.** Understanding Performance of PCI Express Systems - WP350 (v1.1). [S.l.]. 2008.