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

LUCAS DAL PIAZ NUNES

Algoritmos para processamento de imagens visando implementação em FPGA

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

Prof. Dr. Carlos Eduardo Pereira
Orientador

Prof. Dr. Dionísio Doering
Co-orientador

Porto Alegre, 21 junho, de 2014

"Quem tem vontade já tem metade."
Ditado popular

AGRADECIMENTOS

Agradeço principalmente aos meus pais pelo amor incondicional, pela educação e oportunidades que me deram ao longo da vida. Aos meus irmãos, pela amizade e bons momentos que passamos juntos.

A todos os meus familiares, amigos de colégio e aos novos que adquiri nestes últimos anos.

SUMÁRIO

SIGLAS E ABREVIATURAS

CSR Controlled Shift Register

DFT Transforma discreta de Fourier

DFT2D Transforma discreta de Fourier 2D

FFT Transforma rápida de fourier

FIFO First in first out

FPGA Field-programmable gate array

HDL hardware description language

VANTs veículos aéreos não tripulados

LISTA DE FIGURAS

RESUMO

Este trabalho tem por objetivo a implementação de diferentes algoritmos de processamento de imagens visando sua implementação em FPGA. Para tal, ao invés de seguir um fluxo padrão de projeto onde o código seria programado usando uma linguagem de descrição de hardware, será utilizado o MATLAB e o Simulink como ambiente de validação e geração de código HDL. Com esta abordagem, espera-se verificar os benefícios em se utilizar engenharia baseada em modelos e suas eventuais restrições para este uso em particular (processamento de imagens).

Palavras-chave: Model based engineering, Image processing, DFT2D.

ABSTRACT

This work aims the implementation of different processing image algorithms in FPGA implementation. For this purpose, rather than follow a standard design flow where the code would be programmed using a hardware description language, will be used MATLAB and Simulink environment as possible validation and generation of HDL code. With this approach, it is expected to evaluate the benefits of using engineering-based models and their possible restrictions for this particular use.

Keywords: Model based engineering, Image processing, DFT2D.

1 INTRODUÇÃO

A complexidade de sistemas embarcados cresce à medida em que novas necessidades, dispositivos e restrições são impostas. Antigamente, grande parte destes sistemas eram compostos por um único processador que realizam tarefas simples. No entanto, hoje em dia o cenário é outro, havendo sistemas com múltiplos processadores e dispositivos que interagem entre si.

Os veículos aéreos não tripulados , possuem tais características e, em particular, possuem sistemas de visão embarcados. Ou seja, é necessário prover processamento de imagem em tempo real e de forma eficiente. Neste universo, será utilizado FPGAs visto a natureza paralela dos algoritmos de processamento de imagens, a possibilidade de reconfiguração dinâmica e o baixo custo de desenvolvimento.

No entanto, pontos negativos persistem em tal escolha, visto que as ferramentas de desenvolvimento para tais dispositivos possuem um baixo nível de abstração, de um conhecimento detalhado da arquitetura utilizada, assim como um ambiente de simulação específico para a representação de sinais.

Para contornar este problema, linguagens como C e C++ foram adaptadas para permitir a programação de FPGAs possibilitando uma melhor reutilização de código e manutenibilidade. Apesar dos avanços, algumas questões persistem:

- A falta de uma visão global geralmente limita os desenvolvedores para uma solução não otimizada (WEISS, 2004).
- É necessário atenção aos recursos da linguagem que serão utilizados para uma determinada solução (WEISS, 2004).

Diante destas restrições, a metodologia de software definida como engenharia baseada em modelos é proposta. Nela, o foco é a representação abstrata do conhecimento das atividades que governam um certo domínio particular da aplicação. A sua maior vantagem é que expressamos modelos usando conceitos que estão muito menos ligados às implementações tecnológicas e mais ligados ao domínio específico do problema. Estas características, fazem com que o modelo se torne mais fácil de se especificar, entender e manter. Fazendo com que especialistas, não necessariamente ligadas a programação, desenvolvam sistemas (BRAN SELIC. 2003).

Neste contexto, será utilizado o software Matlab/Simulink, visto que ele possui ferramentas de simulação e desenvolvimento guiado a modelos. Possui, também, um ambiente gráfico e uma série de blocos parametrizáveis com soluções parciais para certos domínios

(incluindo o de processamento de imagens). Além, da geração automática de código para algumas linguagens de programação. Neste trabalho, por exemplo, será utilizado o HDL coder para geração automática de código VHDL.

1.1 Motivação

O uso de processamento de imagens em tempo real possui diversas aplicações: reconhecimento de ambientes, objetos; monitoramento de tráfego, ambientes; inspeção de rodovias, pontes; operações de resgate em ambientes perigosos para humanos, entre outros. No entanto, o desenvolvimento de programas se torna um desafio visto que as linguagens mais utilizadas para programar FPGAs (VHDL/Verilog) não possuem o nível de abstração adequado.

O aumento constante da complexidade dos projetos e o tempo cada vez menor exigido para o desenvolvimento de um produto (time-to-market) exigem que um novo fluxo de projeto seja analisado. Desta forma, as etapas para o desenvolvimento de um projeto que geralmente incluem a especificação de requisitos, prototipação, implementação, testes e verificações são realizada simultaneamente ao utilizar-se engenharia baseada em modelos. Eventuais restrições exigidas pela escolha de um dispositivo, a simulação contínua durante o desenvolvimento do projeto, e a geração de código automática para a arquitetura alvo, tendem a diminuir o tempo de desenvolvimento.

1.2 Objetivos

O objetivo deste trabalho é implementar alguns dos principais algoritmos utilizados em processamento de imagens assim como o estudo de um problema em particular, de forma que sejam facilmente readaptados frente a mudanças de requisitos (tamanho da imagem de entrada, máscaras de processamento, blocos de processamento) visando implementação em FPGAs.

Pretende-se, portanto, desenvolver diferentes algoritmos usando engenharia baseada em modelos através das ferramentas Matlab/Simulink. Num primeiro momento, será validado os blocos básicos que permitem a confecção destes algoritmos. A qualidade do código HDL gerado, assim como as restrições e peculiaridades envolvidas no processo serão discutidas no decorrer do desenvolvimento de cada algoritmo.

1.3 Estrutura do texto

No capítulo [2] será detalhado o estudo de caso para o primeiro algoritmo desenvolvido. O restante, tem por objetivo de revisar a teoria necessária para os algoritmos de processamento de imagens no domínio frequencial. O capítulo [3] apresenta as principais ferramentas para o desenvolvimento deste trabalho. Em [4] alguns trabalhos relacionados são citados seguido de uma breve descrição. O capítulo [5] detalha os passos executados para a execução deste trabalho.

Em [6] há a implementação dos algoritmos propostos, os detalhes de implementação assim como os resultados de simulação seguido da conlusão e algumas idéias para

trabalho futuros.

2 REFERENCIAL TEÓRICO

Na seção [2.1] será abordado os principais termos e características da câmera, utilizados para os algoritmos que envolvem detecção de partículas. A seção [2.2] trata de revisar a teoria necessária para o desenvolvimento dos algorítmos discutidos a partir da seção [6].

2.1 Reconstrução de sinais na análise de detecção de partículas

2.1.1 Características da câmera

As principais características da câmera de onde foram obtidas as imagens para análise são:

- Possui sensores de imagem CCD (dispositivo de carga acoplada).
- Sua tecnologia de construção permite a absorção de raios X com energia de até 10 Kev.
- Possui uma taxa de leitura de 200 Mpixeis/s obtida através de uma leitura paralela do CCD na qual cada grupo de dez colunas possui um estágio de saída. Desta forma, aumenta-se o fluxo de dados através de uma leitura paralela dos mesmos. Evitando-se, desta forma, os efeitos indesejados do simples aumento de frequência de operação (ruído, potência).
- A matriz de detecção do sensor CCD é composto de 480_x480 pixeis. O tempo necessário (em segundos)para a leitura do CDD é obtida por:

$$T = \frac{N_y}{2}(T_v + (\frac{N_x}{m})T_h) \tag{2.1}$$

Sendo N_y e N_x a quantidade de pixels, m a quantidade de estágios de saída, T_v o tempo do clock que paraleliza as colunas e T_h o tempo necessário para serializar uma coluna. O fator de divisão 2, deve-se ao fato de haver m portas na parte superior e m na parte inferior do display.

Detalhes adicionais sobre a câmera podem ser obtidos em [P. Denes and Weizeorick, 2009].

2.1.2 Definições para a análise do algoritmo

Segue algumas definições utilizadas para auxiliar na explicação destes algoritmos:

- Cluster - Consiste de um grupo contínuo de pixeis que possui uma intensidade de energia recebida maior que o ruído presente em sua vizinhança. Geralmente, há um pixel central (seed pixel) cercado por sua vizinhança.
- Seed pixel - É definido como sendo o pixel com maior $\frac{Sinal}{Rudo}$ presente no cluster ou, como sendo o pixel de maior intensidade. Estes conceitos tendem a ser equivalentes caso haja uma distribuição uniforme do ruído sobre a matriz de pixeis.
- Cluster size - Refere-se a quantidade de pixeis além do seed pixel que satisfazem a relação $\frac{Sinal}{Rudo}$ por exemplo.
- Bad Pixel - Pixel que possui um mal funcionamento. Geralmente é reconhecido pela aquisição de imagens escuras e analise dos pixeis que possuem um ruído ligeiramente maior do que a média do ruído obtido.
- Hot Pixel - Apresenta um grande sinal recebido mesmo em sua absência.

Os algoritmos para reconstrução de sinais necessitam basicamente de duas variáveis externas. A primeira, refere-se ao limiar que será estabelecido para selecionar os pixeis de interesse. A segunda, ao tamanho da matriz analisada em torno dos pontos de interesse. Esta última, tem relação direta com a energia incidente em torno do ponto considerado.

Antes de realizarmos o processamento propriamente dito podemos falar em um pré-processamento que consiste na subtração das imagens escuras, equalização dos pixels considerando que não há uma distribuição homogênea do ruído e um mascaramento dos bad pixels.

A primeira operação de processamento consiste na obtenção dos Seed Pixeis obtida através da análise completa da matriz de pixeis. Logo em seguida, assegura-se que haja apenas um Seed pixel em torno de uma região de interesse. Para tal, compara-se este pixel com todos os seus vizinhos.

O ponto negativo de tal procedimento é que os hot Pixels podem ser processados caso não sejam eliminados anteriormente. Porém, estes podem ser eliminados ao notarmos uma frequência grande de determinados pixels em um dado conjunto de imagens em uma etapa de pós processamento, por exemplo.

2.1.3 Cluster

Conceitos utilizados para os clusters (região em torno de um ponto de interesse):

- Cluster integrity - Verifica se os pixeis são contínuos internamente ao cluster.
- Hot pixel identification - Após a análise de uma certa quantia de frames pode-se verificar se um pixel teve uma taxa alta de ocorrência se comparado aos demais.
- Cluster shape - A forma esperada para um cluster depende basicamente do ângulo em que um elétron incide na matriz do CCD. Caso houver uma incidência perpendicular, espera-se encontrar formas arredondas. Caso contrário, formas mais alongadas são esperadas. Esta propriedade é intrínseca da onda analisada.

2.2 Teoria Algorítmos

2.2.1 Processamento de Imagens no Domínio da Frequência

O principal objetivo deste estudo, refere-se ao fato de que as ferramentas utilizadas (Matlab e Simulink) oferecem inúmeras funções e facilidades para trabalhar com transformadas. Visto que estas possuem aplicações em diversos campos incluindo processamento de imagens, deseja-se verificar o que a ferramenta oferece em termos de geração de código HDL. Uma revisão com enfoque a processamento de imagens será feita afim de facilitar a compreensão e utilização do software.

Uma imagem é descrita matematicamente como uma função discreta de 2 dimensões. Logo, a transformada natural a ser analisada é a DFT2D. Antes de verificarmos sua forma, será dado uma revisão dos conceitos envolvidos em tal transformada.

2.2.2 Série e transformada de Fourier

A ideia fundamental envolvendo séries de Fourier vem do fato que uma função periódica $f(t)$ pode ser representada como uma série de senos e cossenos de diferentes frequências e diferentes amplitudes. Ou seja:

$$f_{(t)} = a_0 + \sum_{n=1}^{\infty}(a_n \cos(nw_0t) + b_n \sin(nw_0t)) \tag{2.2}$$

w_0, frequência fundamental; $n \in \mathbb{N}^*$

Os coeficientes a_0, a_n e b_n podem ser obtidos integrando-se $f(t)$ com uma função conveniente. Por exemplo, para obtermos a_n devemos notar que as integrais da função acima somente não são nulas quando $m = n$. $m \in \mathbb{N}^*$

$$\int_{t_0}^{t_0+T} f_{(t)} \cos(mw_0t)\mathrm{d}t = \int_{t_0}^{t_0+T} \cos(nw_0t)a_n \cos(mw_0t)\mathrm{d}t$$

$$a_n = \frac{2}{T}\int_{t_0}^{t_0+T} f_{(t)} \cos(nw_0t)\mathrm{d}t$$

$$b_n = \frac{2}{T}\int_{t_0}^{t_0+T} f_{(t)} \sin(nw_0t)\mathrm{d}t$$

$$a_0 = \frac{1}{T}\int_{t_0}^{t_0+T} f_{(t)}\mathrm{d}t$$

A equação [2.2] pode ser descrita de forma mais compacta em sua forma exponencial:

$$f_{(t)} = \sum_{n=-\infty}^{\infty} C_n e^{jnw_0t}$$

$$C_n = \frac{1}{T}\int_{t_0}^{t_0+T} f_{(t)}e^{-jnw_0t}\mathrm{d}t$$

$$C_n = \frac{1}{T}\int_{t_0}^{t_0+T} f_{(t)}[(\cos(nw_0t) - j\sin(nw_0t))\mathrm{d}t = \frac{a_n - jb_n}{2}$$

Para funções aperiódicas, utiliza-se a transformada de Fourier. Sua representação pode ser obtida a partir das séries de Fourier observando o fato de que funções aperiódicas podem ser vistas como um sinal periódico de período infinito. Logo:

$$\tilde{x}_{(t)} = \sum_{n=-\infty}^{\infty} C_n e^{jnw_0 t}$$

$$C_n = \frac{1}{T}\int_{t_0}^{t_0+T} \tilde{x}_{(t)} e^{-jnw_0 t}\mathrm{d}t = \frac{1}{T}\int_{-\infty}^{\infty} \tilde{x}_{(t)} e^{-jnw_0 t}\mathrm{d}t = \frac{1}{T}X(jnw_0)$$

$$\tilde{x}_{(t)} = \sum_{n=-\infty}^{\infty} \frac{1}{T}X(jnw_0)e^{jnw_0 t} = \sum_{n=-\infty}^{\infty} \frac{1}{2\pi}X(jnw_0)e^{jnw_0 t}w_0$$

$$\lim_{T\to\infty} \tilde{x}_{(t)} = x_{(t)} = \frac{1}{2\pi}\int_{n=-\infty}^{\infty} X(jw)e^{jwt}\mathrm{d}w$$

$$\lim_{T\to\infty} X(jnw_0) = X(jw) = \int_{-\infty}^{\infty} x_{(t)} e^{-jwt}\mathrm{d}t \tag{2.3}$$

2.2.3 Série e transformada de Fourier para sinais discretos

Para sinais discretos e periódicos, a ideia fundamental é a mesma que para sinais contínuos. Ou seja, uma série de senos e cossenos de diferentes frequências e diferentes amplitudes.

$$x[n] = \sum_{<k>} a_k e^{jkw_0 n} = \sum_{<k>} a_k e^{jk(\frac{2\pi}{N})n}|_{N=periodofundamental} \tag{2.4}$$

O notação $< k >$ significa que o somatório pode ser realizado em qualquer sequência de N valores visto que a função $e^{jk(\frac{2\pi}{N})n}$ é N periódica.

Os coeficientes a_k são denominados coeficientes espectrais de $x[n]$ e podem ser obtidos multiplicando-se ambos os lados da equação [2.4] por $e^{-jr(\frac{2\pi}{N})n}$ e somando-se os N elementos. Logo:

$$\sum_{n=<N>} x[n]e^{-jr(\frac{2\pi}{N})n} = \sum_{k=<N>} a_k \sum_{n=<N>} e^{-j(k-r)(\frac{2\pi}{N})n} \tag{2.5}$$

Da equação acima deve-se observar que:

$$\sum_{n=<N>} e^{jk(\frac{2\pi}{N})n} \begin{cases} N & N, k = 0, \pm N, \pm 2N, ... \\ 0 & 0, casocontrario \end{cases}$$

De [2.5], podemos notar que o lado esquerdo só não será 0 quando $k = r$ ou for um múltiplo de N. Logo, ao escolhermos valores para r na mesma faixa que k varia em algum momento teremos $k = r$ e [2.4] se reduz a:

$$a_r = \frac{1}{N}\sum_{n=<N>} x[n]e^{-jr(\frac{2\pi}{N})n} \tag{2.6}$$

Desta forma, tanto a equação [2.6] como a [2.4], são periódicas com período N.

Para funções aperiódicas e discretas aplica-se a mesmo ideia utilizada em funções contínuas, ou seja, funções aperiódicas podem ser vistas como um sinal periódico de período infinito.

$$\tilde{x}[n] = \sum_{n=<N>} a_k e^{jr(\frac{2\pi}{N})n} = \sum_{n=-N_1}^{N_2} a_k e^{jr(\frac{2\pi}{N})n}|_{-N_1 \leq n \leq N_2} \tag{2.7}$$

$$a_k = \frac{1}{N} \sum_{n=-N_1}^{N_2} x[n] e^{-jk(\frac{2\pi}{N})n} = \frac{1}{N} \sum_{n=-\infty}^{\infty} x[n] e^{-jk(\frac{2\pi}{N})n}$$

Definindo:

$$X(e^{jn}) = \sum_{n=-\infty}^{\infty} x[n] e^{-jwn} \tag{2.8}$$

$$a_k = \frac{1}{N} X(e^{jkw_0}) \tag{2.9}$$

Combinando [2.7] com [2.9] resulta em:

$$\tilde{x}[n] = \frac{1}{2\pi} \sum_{k=<N>} X(e^{jkw_0}) e^{jkw_0 n} w_0|_{w_0=\frac{2\pi}{N}} \tag{2.10}$$

No momento em que aumentamos N, diminui-se w_0 e quanto $N \to \infty$ a equação [2.10] torna-se uma integral e os limites de integração que antes correspondia a $\sum_{k=<N>}$ torna-se naturalmente $\int_{2\pi}$. Logo [2.10] torna-se:

$$x[n] = \frac{1}{2\pi} \int_{2\pi} X(e^{jkw}) e^{jkwn} \mathrm{d}w \tag{2.11}$$

2.2.4 Transformada de fourier de um sinal amostrado

Partindo-se da equação [2.3], deseja-se derivar a transformada de fourier para um sinal de N amostras separadas por um intervalo ΔT. Desta forma, teremos uma expressão que servirá como base para a derivação da DFT2D. Ao definirmos $w = 2\pi u$:

$$X(ju) = \int_{-\infty}^{\infty} \tilde{x}_{(t)} e^{-j2\pi ut} \mathrm{d}t = \int_{-\infty}^{\infty} \sum_{n=-\infty}^{\infty} x_{(t)} \delta(t - n\Delta T) e^{-j2\pi ut} \mathrm{d}t$$

$$X(ju) = \sum_{n=-\infty}^{\infty} \int_{-\infty}^{\infty} x_{(t)} \delta(t - n\Delta T) e^{-j2\pi ut} \mathrm{d}t = \sum_{n=-\infty}^{\infty} x_{[n]} e^{-j2\pi un\Delta T}$$

A equação $X(ju)$ é $\frac{1}{\Delta T}$ periódica. Ao definirmos N amostras igualmente espaçadas no intervalo $u = 0$ até $u = N - 1$:

$$u = \frac{m}{N\Delta T}, m = 0, 1, 2, ..., N - 1$$

Chegamos a equação desejada:

$$X[p] = \sum_{n=0}^{N-1} x_{[n]} e^{\frac{-j2\pi np}{N}} \tag{2.12}$$

Como estamos interessados na transformada de 2 dimensões a equação deve acima deve ser ajustada. As variáveis foram alteradas para facilitar a sua aplicação em imagens onde x corresponde as colunas e y as linhas.

$$F[u,v] = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f_{[x,y]} e^{\frac{-j2\pi ux}{M} + \frac{-j2\pi vy}{N}} \tag{2.13}$$

Enquanto que a TFD2D inversa:

$$f[x,y] = \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F_{[u,v]} e^{\frac{j2\pi ux}{N} + \frac{j2\pi vy}{N}} \tag{2.14}$$

2.2.5 FFT Transformada rápida de fourier

A DFT calculada diretamente da definição [2.12] possui complexidade $O(N^2)$. No entanto, o mesmo resultado pode ser obtido em $O(Nlog(N))$ ao utilizarmos o algoritmo conhecido por FFT. A DFT inversa, pode ser calculada readaptando-se a DFT visto que a única diferença é o sinal do exponencial e o fator de divisão $\frac{1}{N}$.

Existem diferentes formas de calcular a FFT. No entanto, todas elas baseiam-se no fato de que a expressão [2.12] possui inúmeras repetições visto a simetria do termo $e^{\frac{2\pi nr}{N}}$. Desta forma, a DFT pode ser dividida em diferentes estágios e interligados por uma estrutura conhecida como $butterfly$. Estas interligações, serão responsáveis pela redução do número de operações da DFT e, consequentemente, a redução da complexidade.

Já para o caso da DFT2D [2.13] podemos num primeiro momento aplicar o algoritmo da FFT linha a linha e, com o resultado, aplicar novamente o algoritmo coluna a coluna ou vice-versa. Ou seja:

$$F[u,v] = \sum_{y=0}^{M-1}[\sum_{x=0}^{N-1} f_{[y,x]} e^{\frac{-j2\pi ux}{N}}]e^{\frac{-j2\pi vy}{M}} \tag{2.15}$$

Por exemplo, numa matriz 2_X2, teríamos a seguinte sequência de operações para a componente DC caso aplicássemos diretamente a definição [2.13]:

$$W_N^{nu} = e^{\frac{-j2\pi nu}{N}}$$

$$F[0,0] = f_{[0,0]}W_N^{00}W_M^{00} + f_{[0,1]}W_N^{10}W_M^{00} + f_{[1,0]}W_N^{00}W_M^{10} + f_{[1,1]}W_N^{10}W_M^{10}$$

No entanto, utilizando a equação [2.15] com a intenção de aplicar a FFT podemos reescrever a expressão acima na forma:

$$F[0,0] = \begin{pmatrix} [f_{[0,0]}W_N^{00} + f_{[0,1]}W_N^{10}] & [f_{[1,0]}W_N^{00} + f_{[1,1]}W_N^{10}] \end{pmatrix} X \begin{pmatrix} W_M^{00} \\ W_M^{10} \end{pmatrix}$$

Portanto, utilizando esta técnica, precisamos aplicar $N + M$ FFT para gerar uma DFT2D.

3 FERRAMENTAS UTILIZADAS

3.1 Matlab

O Matlab é um software destinado a fazer cálculos com matrizes, possuindo um ambiente iterativo voltado para o cálculo numérico podendo ser utilizado em conjunto com o Simulink e o Stateflow. Com isso, aplicações que possuem processamento de imagens se adaptam bem a esta linguagem. Outra característica desta ferramenta é a geração automática de código para algumas arquiteturas. Neste trabalho, por exemplo, será utilizado o HDL coder para geração automática de código VHDL.

3.2 Simulink

O Simulink é uma ferramenta para modelagem, simulação e análise de sistemas dinâmicos. Sua interface primária é uma ferramenta de diagramação gráfica guiado a modelos e bibliotecas customizáveis de blocos. Dentre eles, há um conjunto de blocos específicos para processamento de imagens.

A Mathworks também prove a geração automática de código a partir de modelos Simulink. No entanto, alguns cuidados devem ser tomados, visto que nem todos os blocos são sintetizáveis.

Os modelos que possuem trigger e enable também devem ser utilizadas com certa cautela pois podem tirar a propriedade de cicle-accurate e bit-true da simulação com o código sintetizado. A primeira propriedade, cicle-accurate, ocorre quando não há uma correlação temporal entre o modelo e o código HDL gerado. Alguns blocos do Simulink quando otimizados para área podem introduzir atrasos, por exemplo. Já a segunda, bit-true, ocorre ao haver diferenças na composição interna de um bloco e o código gerado. Por exemplo, ao implementarmos um somador e após o otimizarmos tendo em vista performance, a estrutura interna pode ser rearranjada. Mais detalhes sobre o efeito destas restrições serão discutidas na implementação dos algoritmos.

3.3 Stateflow

Corresponde ao módulo presente no Simulink que permite a criação de máquinas de estado, diagramas de fluxo e tabelas verdade. O Stateflow utiliza uma variante de máquina de estados finita definida por David Harel, no qual permite representação hierárquica e

paralelismo entre diferentes fluxogramas.

3.4 Xilinx ISE

Xilinx ISE é um conjunto de ferramentas fornecida pela Xilinx que permite sintetizar e analisar código , habilitando o programador a sintetizar, examinar diagramas RTL, simular, configurar o dispositivo que será utilizado, além de realizar análise temporal.

Neste trabalho, o código HDL será gerado a partir do Matlab e importado no ISE. Em seguida, ele será sintetizado, gerando ,desta forma, uma lista dos recursos utilizados (registradores, somadores). Por fim, realiza-se a simulação do modelo.

4 TRABALHOS RELACIONADOS

O objetivo deste capítulo é demonstrar os principais trabalhos que fazem uso de alguma técnica no intuito de facilitar o desenvolvimento de sistemas que possuam FPGAs com foco em processamento de imagens. Sejam estes, na forma de um problema em particular, ou através de alguma ferramenta.

4.1 Algoritmos para processamento de imagens

No artigo [A. Toledo Moreo, 2004] e [H. Shan Neoh, 2005], há a implementação de diferentes algoritmos de processamento de imagens no domínio tempo utilizando um conjunto de blocos denominado Xilinx system generator que extende o ambiente Simulink. Várias configurações de máscaras são aplicadas (dimensões e finalidades) e, ao final de cada artigo, mostra-se a quantidade de recursos alocada para implementar um determinado algoritmo em função de um modelo de FPGA específico. No entanto, não há uma especificação detalhada de como os diferentes filtros utilizados impactam no modelo descrito além desta solução ser restrita a FPGAs fabricados pela Xilinx.

Em [I.S. Uzun and Bouridane, 2005] diferentes algoritmos para calcular a FFT são implementados: radix-2, radix-4, split-radix e fast Hartley transform. Todos foram implementados sobre o mesmo $framework$ auxiliando o usuário a escolher a transformada adequada em relação aos requerimentos do sistema. Utilizou-se Handel-C como linguagem de programação com o intuito de verificar o tempo de desenvolvimento de projeto além de deixar a FFT o mais flexível possível em relação ao tamanho dos dados de entrada e as unidades de processamento (número de FFT em paralelo).

4.2 Frameworks de programação

Os artigos [I.S. Uzun and Bouridane, 2005] e [D.Crookes and A.Benkrid, 2000] apesar de serem mais antigos, propõem alternativas diante do baixo nível de abstração das ferramentas disponíveis para programação de FPGAs utilizando C++ como linguagem de programação. O presente trabalho, ao utilizar o Simulink/Matlab também busca contornar estas restrições.

5 PROPOSTA DO TRABALHO

Este trabalho consiste na implementação de diferentes algoritmos para processamento de imagens em FPGAs. Diante das dificuldades de desenvolvimento já discutidas, será utilizado Matlab/Simulink como ferramenta de desenvolvimento, visto que esta possui um fluxo de projeto orientado a modelos e um conversor para HDL.

Num primeiro momento, pretende-se desenvolver os principais elementos presentes em qualquer circuito lógico/sequêncial: registradores, operadores lógicos, operadores aritméticos, máquinas de estado, etc no ambiente Simulink. Em seguida, a combinação destes elementos básicos, servirá como base para a construção de modelos mais complexos.

O primeiro algoritmo desenvolvido, visa o estudo de um problema em particular, no qual foram pesquisados os detalhes da câmera utilizada, o modo como os pixeis são lidos, e uma forma de filtrar os pixeis de interesse para um pós processamento. Em seguida, foram implementados alguns dos algoritmos largamente utilizados em processamento de imagens: binarização, covolução, transformada rápida de fourier 2D.

Por fim, pretende-se discutir as vantagens e desvantagens em utilizar tal metodologia de desenvolvimento.

6 VALIDAÇÃO EXPERIMENTAL

Os algoritmos implementados visam a utilização das estruturas mais utilizadas em projetos que envolvam HDL. A seção [6.1] objetiva a validação destes blocos básicos, servindo como base para a consolidação de estruturas mais complexas.

Em [6.2] utiliza-se os conceitos vistos anteriormente e aplica-se a um caso de estudo em particular. As seções [6.3] e [6.4] implementa alguns dos algoritmos amplamente utilizados em processamento de imagens.

6.1 Validação das estruturas básicas

Visto que alguns blocos utilizados no simulink geraram um comportamento inesperado - algoritmos desenvolvidos no TG1 -, em um primeiro momento, optou-se por validar os blocos comumente utilizados em projetos de circuitos digitais. Para tanto, verificou-se a simulação no próprio ambiente. Em seguida, gerou-se o respectivo código HDL e importou-se na ferramenta ISE. Por fim, foram gerados os arquivos de simulação e comparados os resultados entre as duas ferramentas (ISE e Matlab).

Pricipais blocos analisados:

6.1.1 Controlled Shift Register

Funcionamento similar a um Latch do tipo D sendo que o valor da entrada é salvo apenas quando o sinal de entrada (enable) estiver ativo. Na imagem [6.1], podemos visualizar o bloco analisado além de um contador e um comparador. Este último, controla em que momento os dados serão salvos.

6.1.2 Máquina de estado

Utilizada na lógica de controle de diferentes maneiras. Uma característica que facilita bastante a construção de máquinas de estados mais complexas é a possibilidade de inserção de variáveis locais. Na imagem [6.2], a variável aux é utilizada como um contador local e, só permite a troca de estado após atingir um determinado valor.

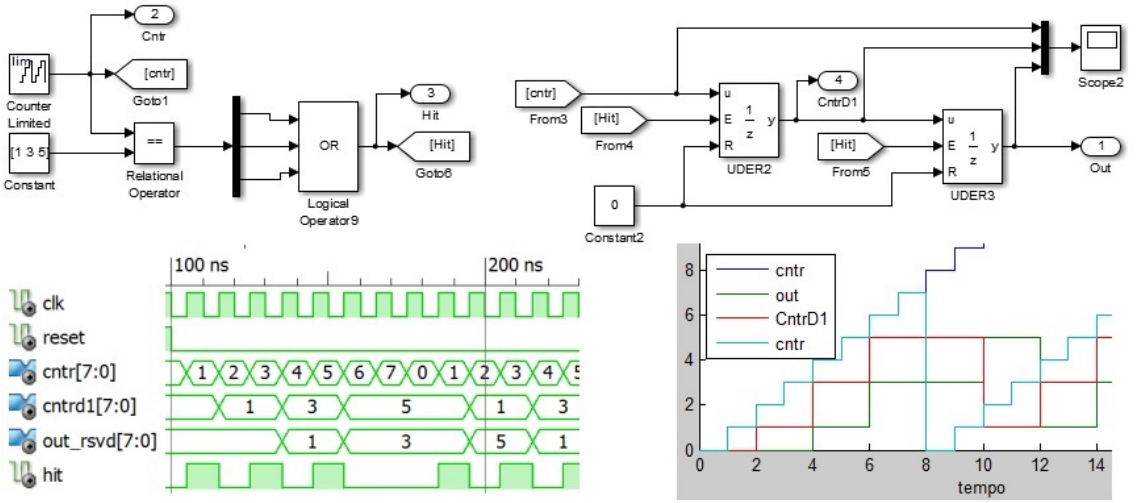

Figura 6.1: Validação Cont. Shift Register

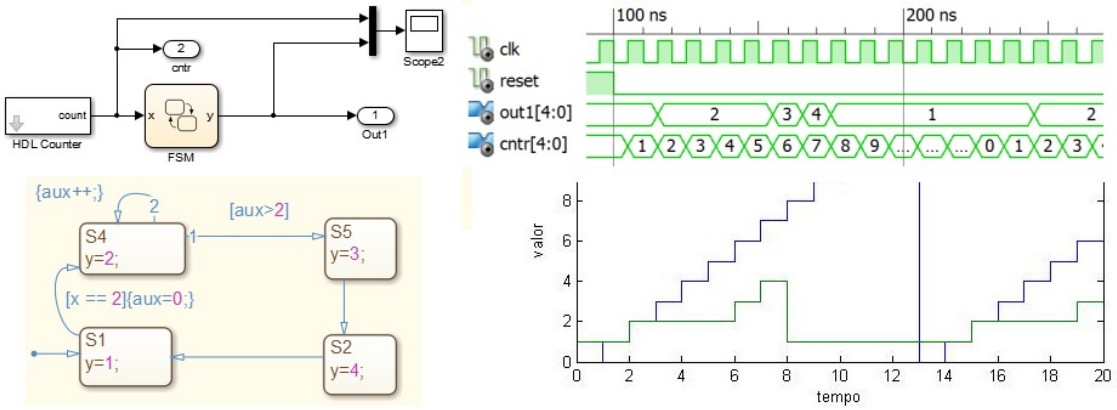

Figura 6.2: Validação Máquina de estado

6.1.3 Matlab function

Visto que alguns blocos no Simulink não são parametrizáveis e requerem mudanças a nível de layout, buscou-se uma forma alternativa de contornar este problema. Um exemplo de uso, seria um CSR variável onde o tamanho e os tipos do dado são explicitados em forma textual. A imagem [6.3] possui o modelo em simulink, o respectivo código que o descreve e as simulações. O código gerado para este bloco em particular apresentou um comportamento inesperado: ao se definir um trigger para ativar a função não foi gerado no código HDL um $PROCESS$ com este sinal. Ao invés, criou-se um trigger emulado no qual foram gerados 4 $PROCESS$. Portanto, prevendo uma geração de código mais concisa para este modelo, optou-se em deixar o sinal de enable como uma entrada comum. Desta forma, o código sequencial deste bloco é ativado toda vez que houver a mudança de estado de um dos pinos de entrada, porém, os valores só serão alterados caso a entrada enable estiver ativa.

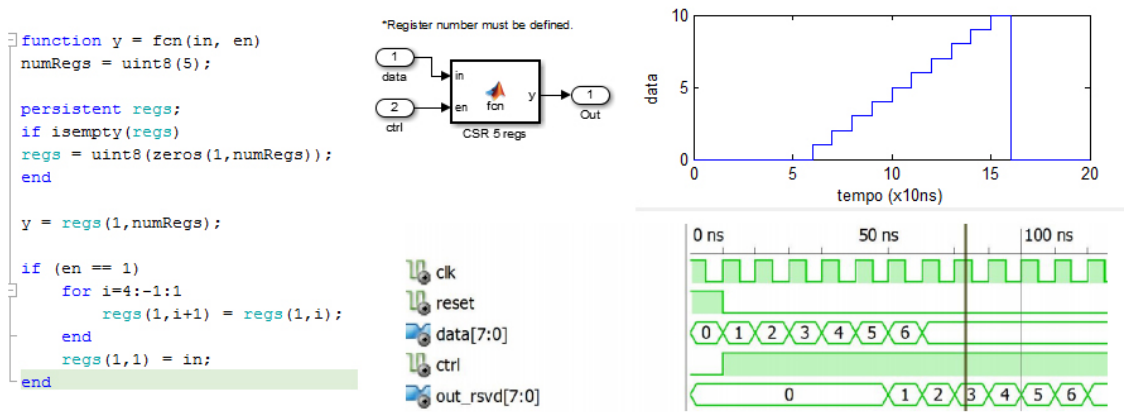

Figura 6.3: Validação Matlab function

6.1.4 Concatena dados e serializa saída

O Simulink disponibiliza um bloco denominado $vector\ concatenate$ para formar um barramento de dados do mesmo tipo. Por exemplo, três sinais de 8 bits irão formar um único sinal de 24 bits. No entanto, alguns blocos não aceitam como entrada um tipo de dado gerado a partir desta estrutura. É o caso de uma FIFO. Para contornar esta restrição, criou-se o bloco-função $concat$ que disponibiliza em sua saída um array contínuo de bits dada uma entrada qualquer. Desta forma, a FIFO aceita como entrada uma sequência qualquer de bits ,ou seja, qualquer tipo de dado.

Por fim, criou-se outra função denominada $ctrlShiftRegs$ que serializa os dados. Ou seja, no momento em que a entrada $load$ estiver ativa, copia-se os dados da entrada in para um registrador de bits pré-definido. A saída y, contém a informação relativa ao bit menos significativo deste registrador. Conforme o pino $shift$ for ativo, haverá uma rotação dos dados para a direita.

Para os blocos acima citados, definiu-se uma lógica de controle extremamente básica a partir de um contador. No momento em que o contador valer 1 o sinal $push$ é ativado havendo a cópia dos dados de entrada para a FIFO. Neste caso, 204 e 170, correspondendo a sequência binária 1100110010101010 ao considerarmos os sinais como sendo inteiros de 8 bits sem sinal. No instante 2, os dados são disponibilizados na saída da FIFO e copiados para o próximo bloco. Enfim, uma sequência de ativação do pino $shift$ permite a serialização dos dados. As simulações e os blocos podem ser vistos na figura [6.4].

6.2 Reconstrução de sinais na análise de detecção de partículas

A análise deste algoritmo, deve-se ao fato das câmeras científicas de alta velocidade possuírem cada vez mais pixeis e $frames/_{segundo}$ aumentando o fluxo de informações. Uma forma de se lidar com estas exigências é utilizar processamento de imagens em tempo real utilizando FPGA. O desafio, porém, é a que cada novo experimento exigem-se novos algoritmos de processamento de imagens. Neste contexto, será analisado uma forma de filtrar os dados relevantes a serem analisados, diminuindo, desta forma, o fluxo de dados e o pós processamento.

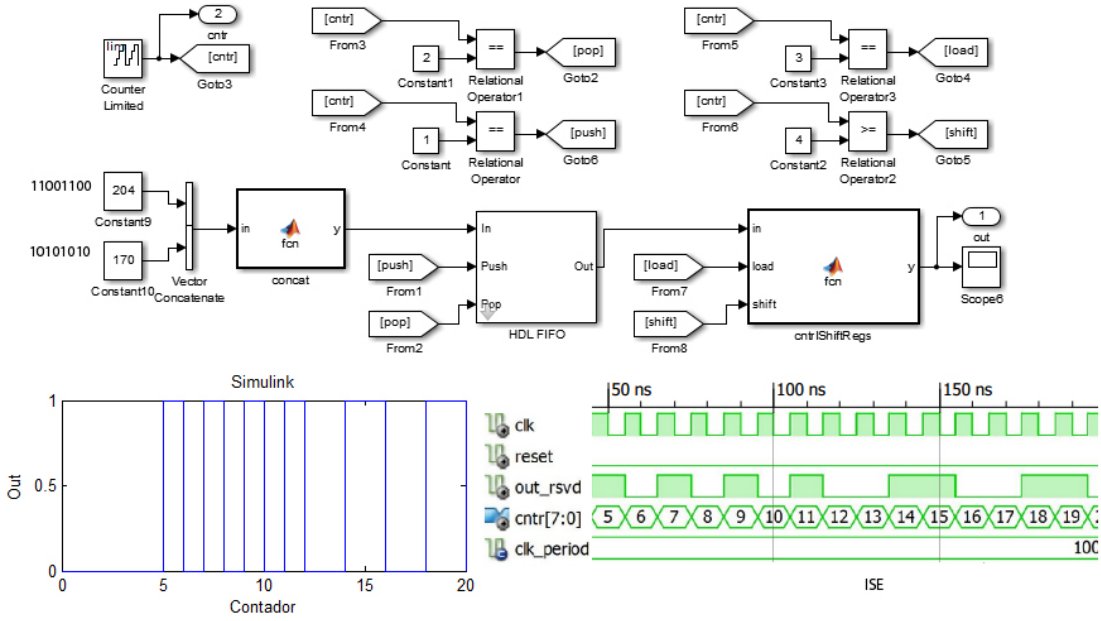

Figura 6.4: Concatena dados e serializa saída

6.2.1 Modelo com processamento serial

Nesta primeira análise, supomos que toda a imagem é obtida através de um único canal, que há apenas um sinal de clock e que a imagem é composta de 5_X10 pixeis. Apesar deste modelo não corresponder ao modelo real, ele já é solução parcial do problema global. Ou seja, além de servir como um passo intermediário para o objetivo final, será possível melhor analisar os blocos que precisaram ser readaptados e as eventuais dificuldade neste processo. A arquitetura do modelo principal é composta por cinco blocos principais:

- $\_Control$ - Composto por quatro contadores que servirão como controle para os demais blocos além de conter a lógica para a geração dos SeedPixeis.
- $\_PreProcessing$ - Etapa anterior ao cálculo dos SeedPixeis. Nela, a imagem analisada é subtraída do bloco Pedestal Memory além de haver a eliminação dos BadPixeis.
- $\_Mask3_X3$ - Estrutura responsável pela obtenção dos SeedPixeis. Composta por

$$N_{CSR} = (N_c + 1) \times (N_b - 1) \tag{6.1}$$

  onde N_{CSR} refere-se a quantidade de CSR, N_c ao número de colunas de um canal e N_b o tamanhao do bloco utilizado. Neste caso, usaremos um bloco de dimensão 3.
- $\_isSeed1$ - Bloco que efetivamente diz se um pixel é ou não SeedPixel. Além de verificar que o pixel central é o maior do que toda a vizinhança é necessário excluir as condições de fronteira.
- $\_SendDataToPosProcessing$ - Este bloco armazena os dados de interesse e os envia para uma próxima etapa de processamento. Para tanto, utiliza-se uma estrutura do tipo FIFO onde os dados são armazenados no momento em que o sinal

$isSeed$ estiver ativo. Já o envio dos dados é controlado por um contador/decrementador que age da seguinte forma: enquanto houver dados a serem processador ($counterPix < 50$) e o sinal $isSeed$ estiver ativo, incrementa-se o contador de uma unidade. Em seguida, ($counterPix >= 50$) decrementa-se e enquanto este não valer 0 mantém-se a operação pop da FIFO ativada. No momento em que ele chegar a 0 todos os dados forma enviados.

Estes blocos podem ser visto na image [6.5], assim como a simulação obtida. Este resultado, deve-se ao fato da imagem de entrada possuir 3 seedPixeis nos pixeis 11, 24, 32. O deslocamento temporal de $11unidades$, refere-se ao foto de que um seedPixel é determinado após

$$T = (N_p + N_c + \lfloor \frac{M_t}{2} \rfloor) \tag{6.2}$$

onde N_p refere-se ao pixel a ser analisado, N_c ao número de colunas do canal e M_t ao tamanho da máscara.

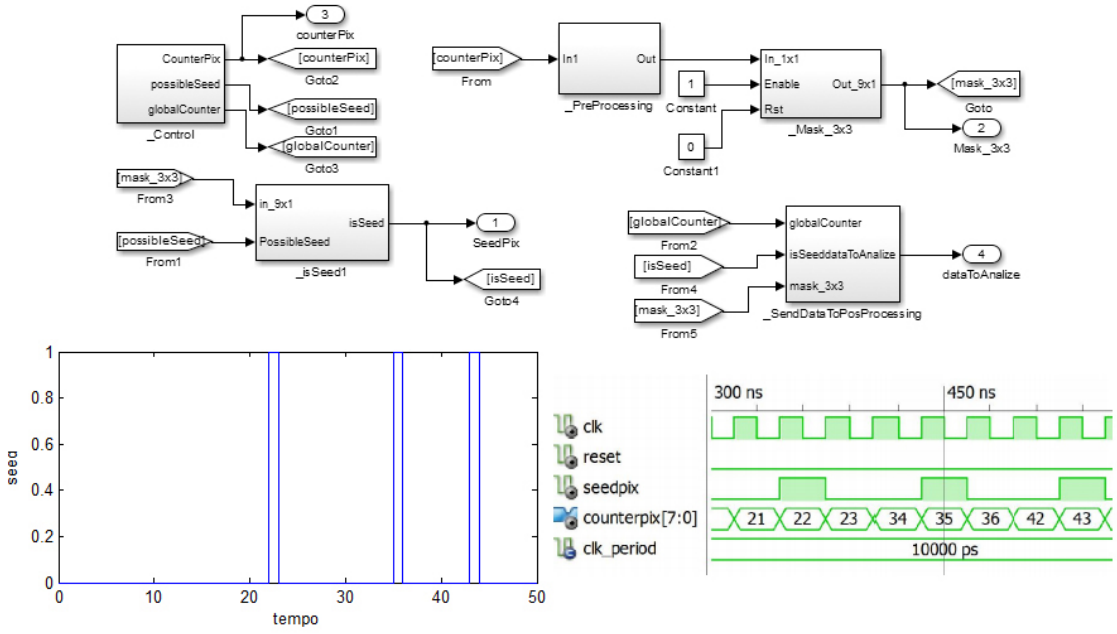

Figura 6.5: Modelo de 1 canal

A partir do momento em que obtemos os pixeis de interesse, podemos aplicar diferentes algoritmos com o intuito de melhor compreender a natureza física do sinal recebido. Com este objetivo, poderíamos nos perguntar qual a forma típica de um pixel de interesse, por exemplo. A ferramenta de modelagem utilizada nos permite obter esta resposta através da imagem [6.6].

Neste exemplo em particular, ao invés de mandarmos os 50 pixeis para processamento, iremos mandar 27 (para cada SeedPixel 9 pixeis). Ou seja, uma redução de 46% no volume de dados além de uma etapa a menos de processamento (detecção dos SeedPixeis).

6.2.2 Modelo com processamento serial utilizando uma imagem real

Uma imagem real obtida através da câmera em estudo é composta de 500_X576 pixeis. No entando, apenas 480_X480 serão efetivamente lidos. Os pixeis que serão descartados

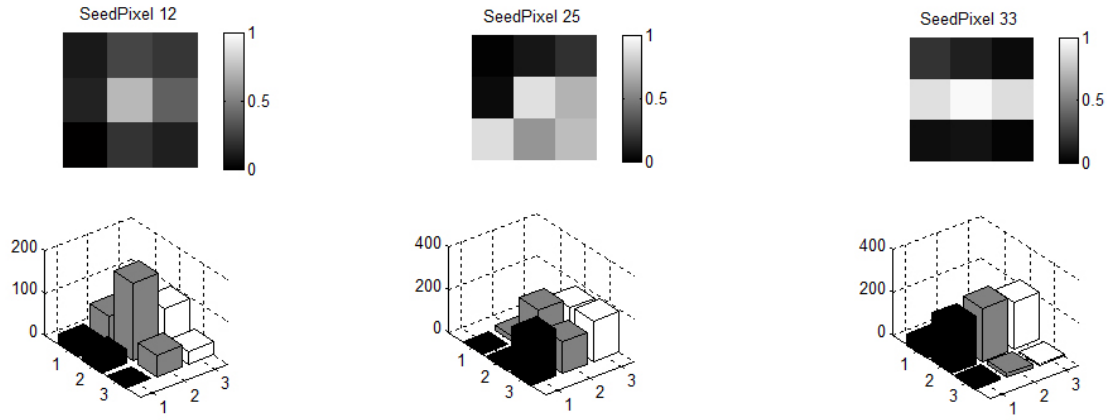

Figura 6.6: Análise de um SeedPixel gerado manualmente.

podem ser visualizados na imagem [6.7]. Ou seja, as primeiras 6 e as últimas 4 linhas além das últimas 2 colunas de cada canal são eliminadas.

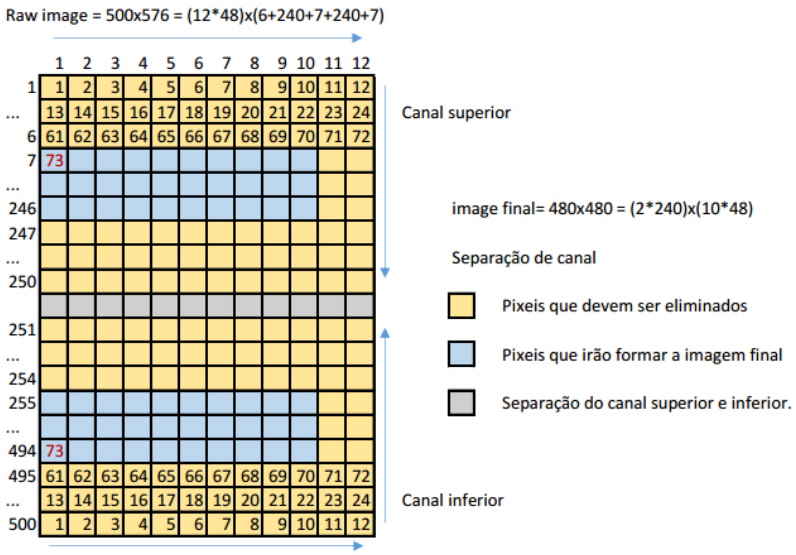

Figura 6.7: Formação da imagem.

Neste exemplo, será considerado que os pixeis de interesse já foram filtrados, ou seja, temos uma imagem de 480_X480 como entrada. Este processo foi realizado através de um script em Matlab. Após, o processo é realizado da mesma forma que o modelo discutido anteriormente, exceto que as dimensões da imagem mudaram.

As modificações necessárias partindo do modelo anterior foram:

- $\_Arquivodeconfigurao$ - Visto que as dimensões da imagem foram alteradas, as variáveis relacionadas também o foram. Adicionou-se um $threshold$ baseado no pixel de maior intensidade como condição extra para um pixel ser considerado Seed. Por fim, adicionou-se funções para ler a imagem.

- $\_Mask3_X3$ - Neste bloco utilizou-se um CSR de tamanho correspondente a linha de atraso, ou seja, 480. A modelagem do bloco CSR foi discutida em [6.1.3].

- $\_isSeed1$ - Agora a variável $threshold$ também influencia na definição de um Seed Pixel. Caso contrário iríamos pegar milhares de outros pixeis que deveriam ser descartados.
- $\_SendDataToPosProcessing$ - Bloco específico para este modelo. Ao ser encontrado um SeedPixel, os dados da máscara são salvos em uma estrutura do tipo FIFO, havendo uma FIFO para cada linha. Após o processamento de toda a imagem, os dados são enviados serialmente através das saídas $l1$, $l2$ e $l3$. O controle é realizado através de 2 contadores. O primeiro, armazena a quantidade de SeedPixeis encontradas. O segundo, gera os sinais de controle para a FIFO, para o serializador e para o primeiro contador.

Gerada a simulação para este modelo e definido um $threshold = (0.8xPixeldemaiorintensidade)$, obteu-se 4 seedPixeis que podem ser visualizados na imagem [6.8]

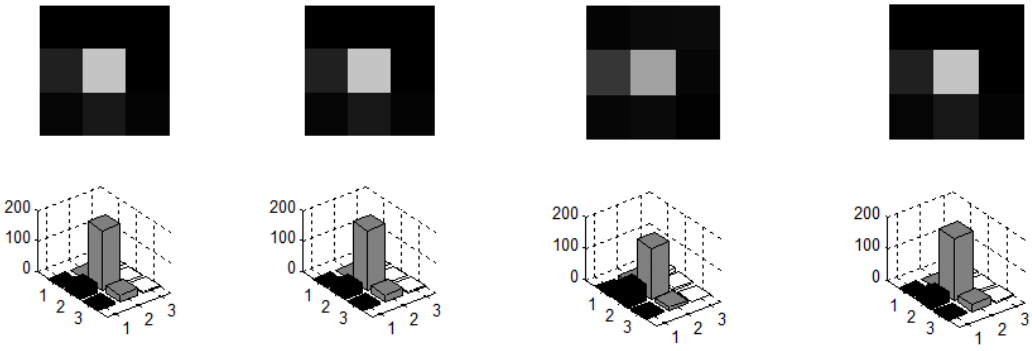

Figura 6.8: Seed Pixeis reais.

Para validar este modelo, compara-se os pinos de saída que serializam linha a linha a máscara dos seedPixeis [6.9]. Desta forma, podemos visualizar que o primeiro seedPixel encontrado, possui a seguinte vizinhança: $1^oSeed = \begin{bmatrix} 5 & 7 & 0 \\ 42 & 167 & 11 \\ 7 & 47 & 9 \end{bmatrix}$

Se no exemplo fictício [6.2.2] tivemos uma redução de apenas 46% no fluxo de dados, neste exemplo, dos 480_X480 que seriam enviados apenas 4_X9 serão. Ou seja, uma redução de 99.98%.

6.3 Processamento de imagens no domínio tempo

Dois algoritmos amplamente utilizados em processamento de imagens serão implementados: binarização e convolução. Num primeiro momento, será demonstrado como realizar operações de divisão em FPGAs visto que o algoritmo de convolução irá utilizar este recurso.

6.3.1 Divisão em FPGA

No escopo deste trabalho, apenas operações em ponto fixo serão realizadas para o algoritmo de convolução. Em HDL, a operação de divisão somente aceita potência de 2

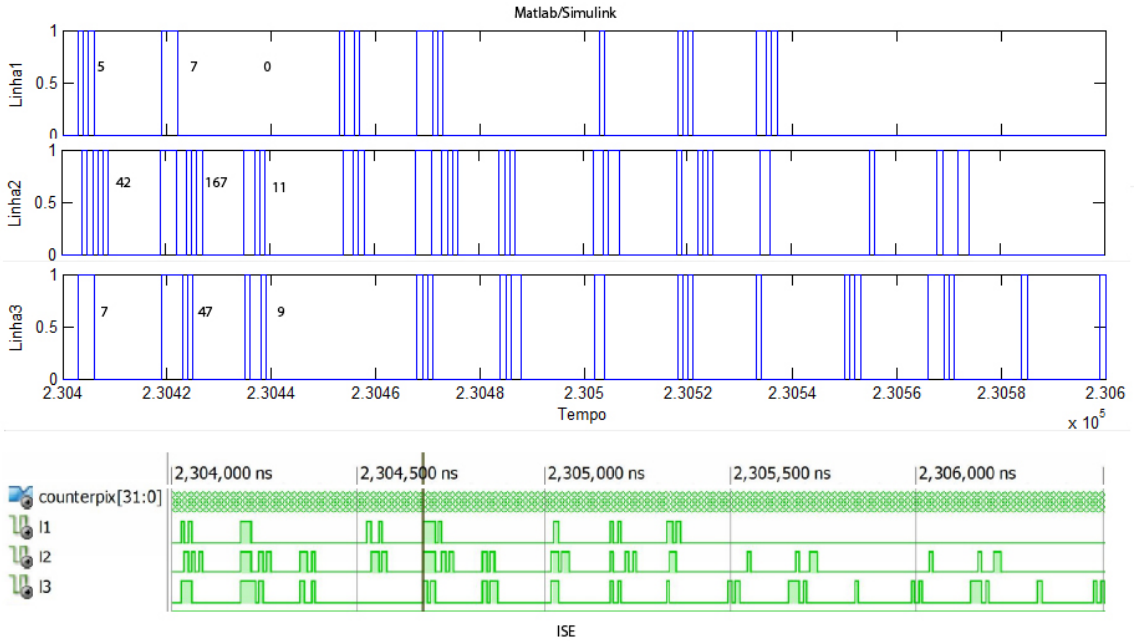

Figura 6.9: Validação processamento serial.

como divisor, ou seja, na realidade uma operação de shift register é realizada. Caso queiramos uma divisão por um valor diferente, devemos recorrer a alguma outra técnica. A imagem [6.10] mostra a relação envolvida para a divisão de um número inteiro qualquer.

Figura 6.10: Divisão em ponto fixo

Utilizando esta abordagem para realizar divisões inteiras, o valor d' e o $Errorelativo$ são calculados por:

$$\lfloor \frac{n}{d} \rfloor \approx \lfloor \frac{nd'}{2^p} \rfloor$$

$$\lfloor \frac{2^p}{d} \rfloor \approx d' \tag{6.3}$$

$$Error. = \frac{\left| \lfloor \frac{n}{d} \rfloor - \lfloor \frac{\lfloor \frac{2^p}{d} \rfloor}{2^p} \rfloor \right|}{\lfloor \frac{n}{d} \rfloor} \tag{6.4}$$

Ou seja, o erro relativo é dependente de n, d e p. Visando a melhor relação entre estes parâmetros a imagem [6.11] foi obtida onde varia-se apenas d e após p respectivamente.

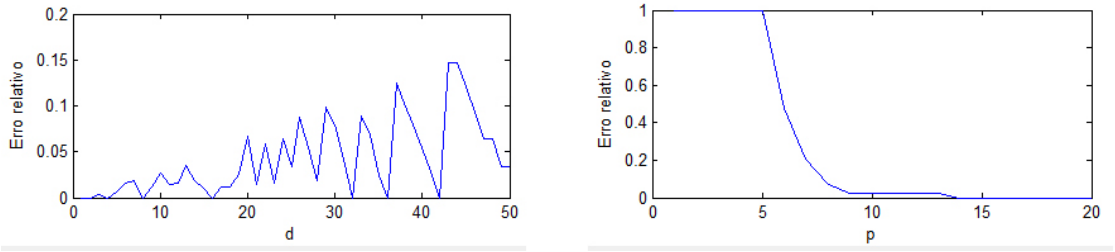

Figura 6.11: Análise erro relativo

6.3.2 Binarização

O processo de binarização consiste na transformação de uma imagem em tons de cinza para uma representação binária podendo ser descrita matematicamente pela função:

$$y = T(x) \begin{cases} 1 & \text{Se } x > T \\ 0 & \text{Se } x \leq T \end{cases}$$

no qual o valor T refere-se a um limiar escolhido. Neste exemplo, será dado ênfase apenas na operação de binarização em si, ou seja, assume-se que os dados de entrada estão na sequência correta para serem manipulados. Caso contrário, voltaríamos para um problema parecido com os discutimos acima. Na imagem [6.12] podemos visualizar todo o processo envolvido. O modelo é extremamente simples e possui dois parâmetros: T e o número de blocos. Neste exemplo, utilizou-se uma imagem de 256_X256 pixeis com 4 blocos.

A simulação em ambiente Matlab/Simulink é realizada em 3 etapas: obtém-se os dados referentes a imagem em uma representação matricial, readapta-se o formato desta matrix onde as colunas correspondem ao número de canais e as linhas ao próximo valor a ser lido e após volta-se a representação matricial original para podermos ver a imagem resultante. Já no ambiente ISE, a imagem é obtida através de um arquivo no qual os dados são mapeados para os respectivos pinos. Neste ambiente, temos apenas uma representação a nível de sinais e, para termos de validação, compara-se um dos canais de saída de cada plataforma.

6.3.3 Convolução

O processo de convolução dada uma imagem I, e um filtro F 3_X3, pode ser obtido da seguinte forma:

$$I_f[i,j] = \sum_{i=2}^{M-1} \sum_{j=2}^{N-1} \sum_{k=-1}^{1} \sum_{l=-1}^{1} I[i+k, j+l] F[k+2, l+2] \tag{6.5}$$

A arquitetura utilizada está presente em [H. Shan Neoh, 2005] e [C. T. Johnston, 2005] imagem [6.13], sendo que há um estágio de divisão extra na saída $outputpixels$.

A aplicação de diferentes filtros F na imagem I resulta em diferentes aplicações tais como filtros: passa-baixas, passa-faixas, passa-altas, dilatação, erosão, média, etc. Portanto, um modelo com um filtro facilmente adaptável possibilita a resolução de diferentes problemas. Neste contexto, os pixeis do filtro F não são fixeis. Após o cálculo de um

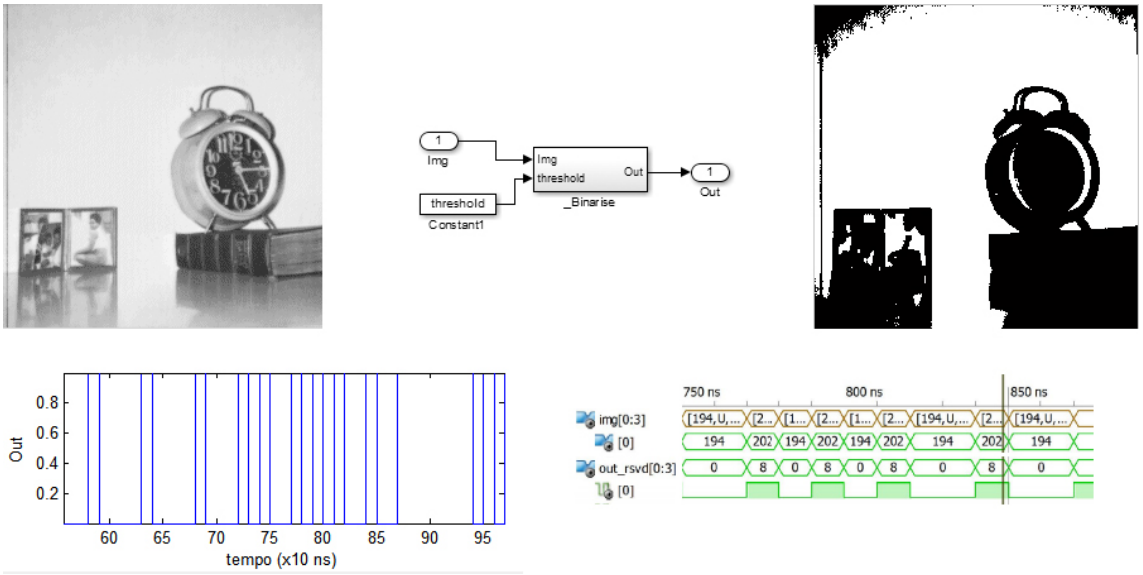

Figura 6.12: Imagem binarizada.

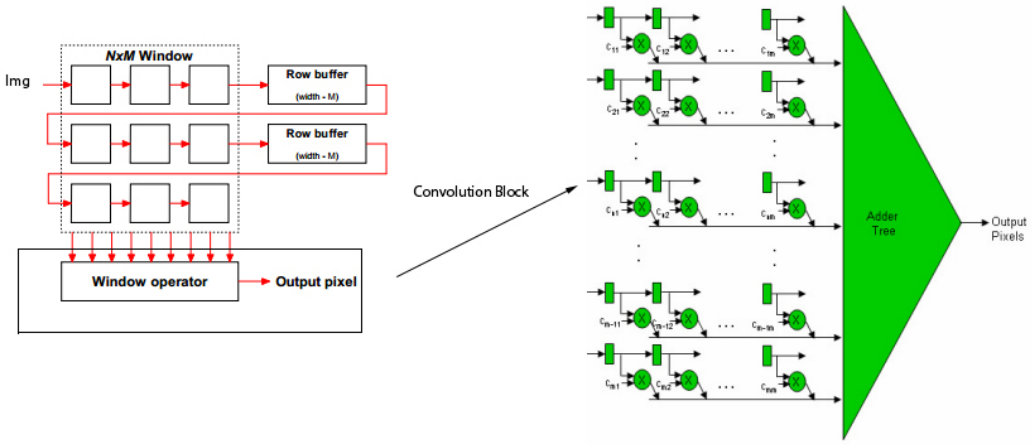

Figura 6.13: Arquitetura global

ponto $I_f[i,j]$, há um estágio de normalização que consiste de uma divisão com o intuito de deixar o filtro mais genérico. Esta última, foi implementada seguindo os passos da seção [6.2].

Os principais blocos do modelo podem ser visualizados em [6.14] assim como a arquitetura global. Em relação ao modelo visto em [6.2.2] reutilizam-se os blocos $\_Control$ e $\_Mask3x3$ sendo que há apenas modificações a nível de variáveis e não de layout. A operação de convolução ocorre efetivamente no bloco $\_convolution$.

As simulações abaixo foram realizadas definindo-se F como um filtro 3_X3 unitário e um fator de normalização 9. Ou seja, estamos calculando a média aritmética em torno de uma ponto de interesse. Logo, a variável d' definindo-se p igual a 16, vale 7281 obtida a partir da equação [6.3]. Para este filtro em particular, os valores $I_f[i,j]$ serão todos menores ou iguais a 255 visto que cada pixel pode assumir o valor máximo de 2^8 e, portanto, o pino $filter$ possui 8 bits.

Para termos de validação deste bloco, gerou-se as simulações no ambiente ISE utilizando o pino img como entrada para obtenção da imagem. Esta é carregada em tempo de

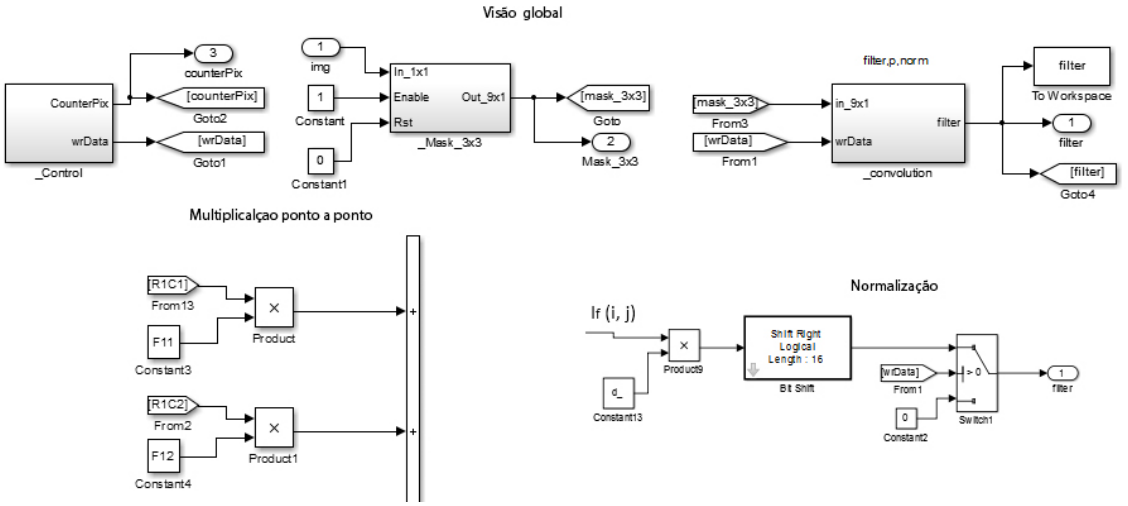

Figura 6.14: Visão global e blocos principais

simulação através de um arquivo texto. Os valores de saída são obtidos a partir do pino $filter$, sendo salvos em arquivo para serem comparados com a simulação obtidas a partir do simulink. Como última etapa, subtrai-se as duas imagens e, caso o processo estiver correto o valor desta nova "imagem"será zero. Este processo pode ser visto abaixo.

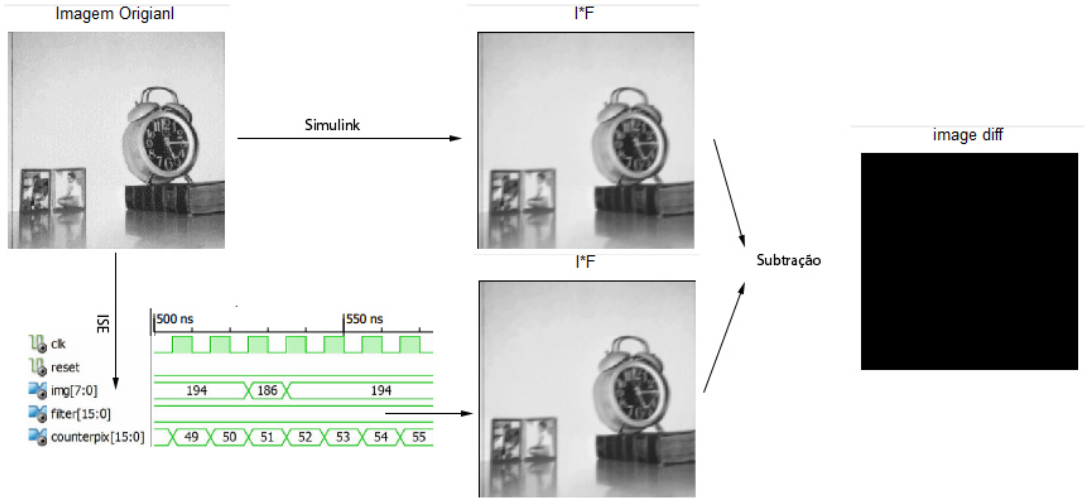

Figura 6.15: Validação filtro convolução

As mudanças necessárias para um filtro F com dimensões diferentes com base na imagem [6.13] compreendem: a modificação no tamanho da janela de obtenção dos pixeis (window operator), a alteração do tamanho de linha (row buffer), inserção de novos multiplicadores e somadores.

6.4 Processamento de imagens do domínio frequência

6.4.1 Transforma discreta de fourier

O HDL coder disponibiliza um bloco denominado $HDL\ streaming\ FFT$ que gera a transforma de Fourier de um sinal. O algoritmo utilizado pelo bloco é denominado

$radix-2\ decimation\ in\ time$. Para utilizá-lo, é necessário definir o tamanho da FFT desejada, ou seja, o número de amostras N. Após a entrada $Start$ receber um sinal lógico alto, N amostras são coletadas na entrada. Esta sequência de dados, corresponde a função $x[n]$ da equação [2.12], porém, com os limites definidos pelo número de amostras. Após um determinado tempo, o sinal $dvalid$ é ativado, informando que os dados da saída correspondem aos valores $X[n]$. Por fim, quando o sinal $ready$ estiver ativo, um novo frame pode ser calculado visto que todos os valores já foram determinados. O tempo total em ciclos necessário para todo o processo é dado por:

$$T = \frac{3N}{2} - 2 + log_2 N(\frac{N}{2} + 3); \tag{6.6}$$

Para gerar os sinais de controle e sincronização dos dados de entrada foi criado o seguinte modelo contendo uma máquina de estado para gerir tal tarefa - imagem [6.16].

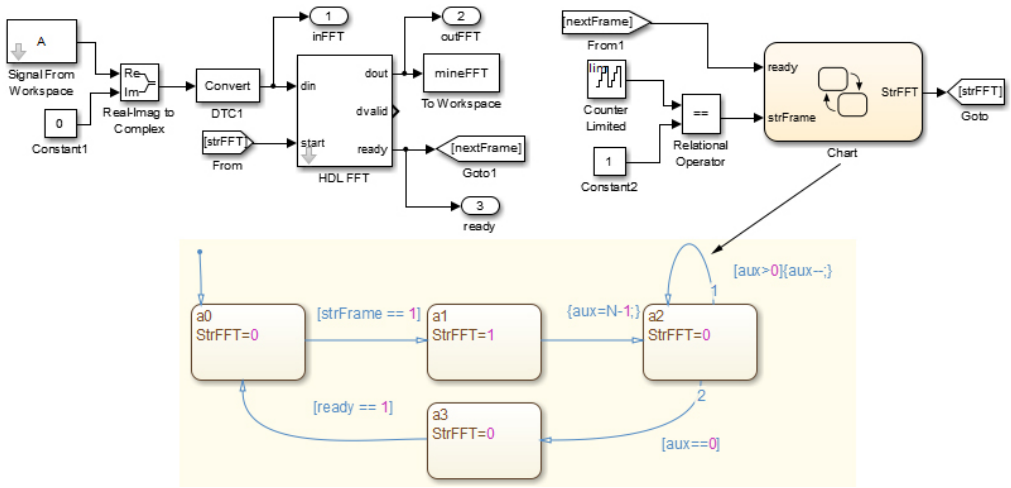

Figura 6.16: Modelo FFT 1D

Por exemplo, ao considerarmos o sinal $x[n] = 3 + cos(2\pi(m)) + cos(3_X 2\pi(m))$ com 8 amostras no qual m assume os valores $0 : 1/8 : 1 - 1/8$, esperamos como resposta uma componente $DC = 3$, um impulso em f_0 e um outro em $3f_0$. A imagem [6.17] corresponde a FFT1D deste exemplo em particular.

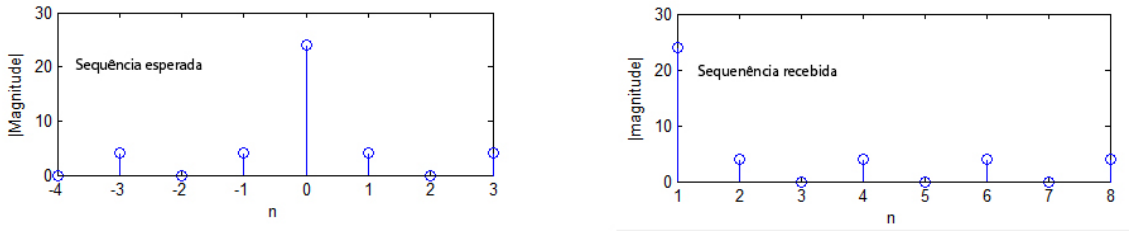

Figura 6.17: Espectro de frequência

Logo, é necessário ajustar os dados retornados pelo bloco $HDL\ streaming\ FFT$ pois eles não correspondem exatamente a equação [2.12], sendo necessário ajustar a magnitude do espectro de frequência (dividir pelo número de amostras) assim como a sequência de dados.

O processo de simulação corresponde basicamente a comparar os valores da resposta da FFT no momento em que os dados estão sendo processados com os valores desejados [6.17]. Os dados de entrada e saída possuem 32 bits sendo 16 para a parte fracionária. A simulação obtida através do ISE pode ser visualizada em [6.18] e podemos notar que são gerados pinos diferentes para a parte real e imaginária.

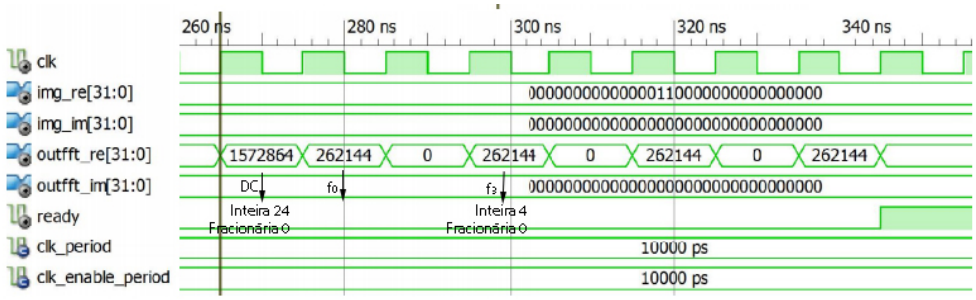

Figura 6.18: Simulação FFT ISE

6.4.2 Transforma discreta de fourier 2D

O HDL coder não disponibiliza 1 bloco para calcular a DFT2D. No entanto, como discutido em [6.4.1], é possível obtê-la através de sucessivas aplicações do algorítmo FFT. Nesta implementação, será calculada a FFT de cada linha da imagem de entrada e, com o resultado, aplica-se novamente a FFT, porém, agora em coluna. Portanto, o problema se resume a construir um modelo que realize as operações descritas acima. A imagem [6.19] o descreve:

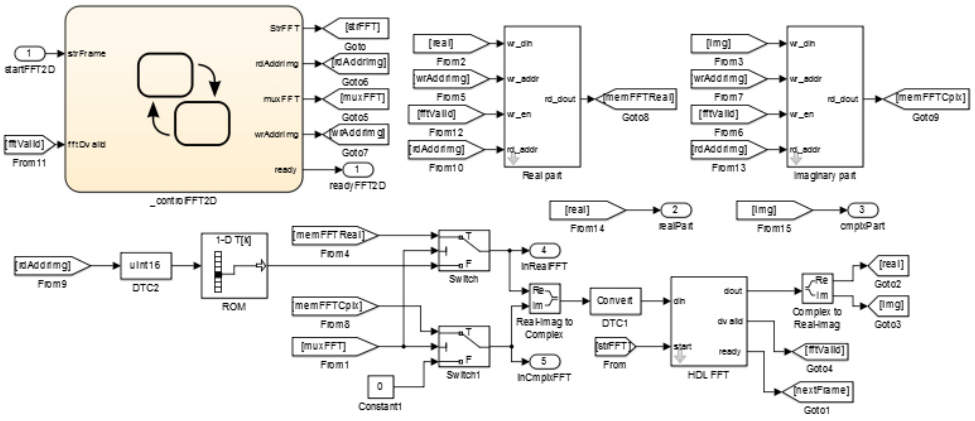

Figura 6.19: Modelo TDF2D

O modelo acima opera da seguinte forma: Após o bloco de controle $controlFFT2D$ receber um sinal alto na entrada $startFFT2D$ o processo inicia. O bloco $HDLFFT$ recebe os primeiros N valores correspondendo a primeira linha da imagem. Após 304 ciclos dado pela expressão [6.6] determina-se sua transformada, onde o resultado é salvo nos blocos de memória $RealPart$ e $ImaginaryPart$. Este processo se repete até a última linha da imagem. Completada esta etapa, o sinal $muxFFT$ é ativado e os valores armazenados em memória são utilizados como entrada para o bloco FFT. Desta forma, inicia-se o cálculo da FFT coluna a coluna. Ao final desta etapa, a DFT2D da imagem

de entrada estará armazenada nos mesmos blocos de memória responsáveis por reter a informação da FFT calculada linha a linha. Ou seja, o resultado final ($TDF2D$) pode ser obtido lendo-se diretamente os blocos de memória $RealPart$ e $ImaginaryPart$ ou logo após o cálculo da FFT por coluna. O sinal $readyFFT2D$ indica quando o processo acabou, possibilitando, desta forma, o processamento de uma nova imagem.

Para validar este modelo inseriu-se a função $x[n] = 127cos(3_X 2\pi(m))$ com 64 amostras no qual m assume os valores $(0 : 1 : 64 - 1)/64$. Definindo $x[n]$ como sendo uma coluna, replica-se a mesma função por N colunas constituindo a imagem de entrada. Esta imagem, assim como a sua DFT2D podem ser melhor visualizadas em [6.20].

Figura 6.20: Imagem de entrada e DFT2D associada.

Como esperado, a DFT2D possui apenas uma componente em $3f_0$ na direção y, visto que não há oscilação na direção x. A simulação no ambiente ISE, no momento em que as componentes não nulas são obtidas pode ser visualizada em [6.21]. Os dados possuem 32 bits nos quais apenas os 10 bits menos significativos correspondem a parte fracionária.

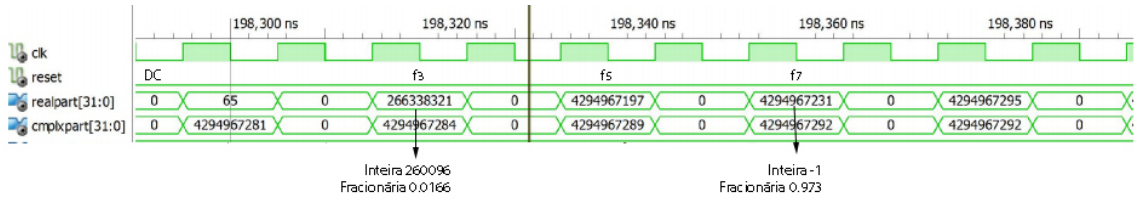

Figura 6.21: ISE simulação DFT2D

Lembrando que para chegarmos ao valor 127 esperado em $3f_0$ devemos realizar a seguinte sequencia de operações:

$$127 = \frac{260096}{N^2}2$$

6.5 Código HDL gerado

Os dados da tabela [6.5] referem-se aos recursos alocados para cada algoritmo visto anteriormente. Eles foram obtidos ao se utilizar um Virtex5, modelo XC5VLX20T.

Algoritmo	>=	x	+/-	Contadores	Reg.	F. Máx. Mhz	Acc	RAM
[6.1.1]	0	0	0	1	17	372.384	0	0
[6.1.2]	1	0	1	0	18	538.054	1	0
[6.1.3]	0	0	0	0	40	644.330	0	0
[6.1.4]	2	0	2	1	65	335.841	3	1 16x16
[6.2.2]	14	0	6	1	15798	223.359	13	3 16x48
[6.3.2]	4	0	0	0	0	Max.	0	0
[6.3.3]	0	1	8	3	4113	477.16	0	0
[6.4.1]	2	0	2	1	65	335.841	3	1 4x64

Em relação a transformada [6.4.1], o número de ciclos necessários para gerar a FFT de um frame contendo 1024 amostras é 6648 de acordo com [6.6]. No artigo [I.S. Uzun and Bouridane, 2005] a mesma transformada utilizada neste trabalho ($radix-2$ $decimation\ in\ time$) foi determinada em $121.6us$ considerando uma frequência de operação de $84Mhz$. Caso utilizasse-mos esta mesma frequência de operação a nível de comparação, conseguiríamos o mesmo resultado em $80us$. No entanto, no mesmo artigo, há a implementação de uma variação do algoritmo citado acima ($radix-4$) no qual necessita-se de $68us$ para determinar a FFT.

7 CONCLUSÃO E TRABALHOS FUTUROS

Neste trabalho, foi desenvolvido alguns dos principais algoritmos amplamente utilizados em processamento de imagens através do software Matlab/simulink visando implementação em FPGA.

Durante o desenvolvimento destes algoritmos, pode-se observar algumas das vantagens em se utilizar engenharia baseada em modelos:

- Visualização dos resultados em diferentes domínios – Dependendo da etapa de construção da solução diferentes testes são necessários para verificar o correto funcionamento do sistema. Por exemplo, após aplicar um filtro em uma imagem, gostaríamos de verificar o resultado da imagem resultante. Em um segundo momento, podemos desejar recuperar a lista de diferentes sinais de sincronização para verificar se foram gerados corretamente.

- A facilidade em readaptar um algoritmo frente a alguma mudança proposta - Em relação ao algoritmo de detecção de partículas, diferentes versões foram realizadas aumentando-se o nível de complexidade gradativamente. As mudanças envolvidas, geralmente são pontuais e facilmente identificadas devido à diagramação gráfica da ferramenta. A possibilidade de configuração dos blocos a partir de scripts também contribui bastante para deixar o modelo mais flexível.

- Visão global da solução – À medida que criamos modelos com funções bem definidas, podemos abstrair os detalhes do seu funcionamento a fim de aumentar o nível de abstração possibilitando, desta forma, a criação de diagramas cada vez mais complexos.

- Geração de código para arquitetura alvo – Neste trabalho, o código HDL gerado foi obtido diretamente a partir da própria ferramenta de desenvolvimento. Desta forma, o desenvolvimento em si se torna menos suscetível a erros.

Em relação às desvantagens, elas ficam direcionadas a plataforma utilizada e não a metodologia de desenvolvimento. Por exemplo, o gerador de código HDL não reproduziu o código esperado em algumas estruturas; a IDE utilizada não é estável ao se utilizar alguns recursos; a impossibilidade de tentar contornar alguns dos problemas citados acima devido a ferramenta ser proprietária.

O trabalho apresentado pode ser estendido visando o estudo de um problema específico envolvendo processamento de imagens. Em seguida, este pode ser implementado em

uma FPGA com o objetivo de fornecer novas funcionalidades aos VANTS discutidas em [1.1].

Ou ainda, um estudo mais detalhado sobre as transformadas a fim de se obter uma transformada de Fourier inversa visto que a ferramenta não a fornece.

REFERÊNCIAS

P. Donachy K. Alotaibi K. Benkrid A. Bouridane, D. Crookes. A high level fpga-based abstract machine for image processing. *journal of systems architecture*, 1999.

F. Soto Valles J. Muro C. Fernandez Andres A. Toledo Moreo, P. Navarro Lorente. Experiences on developing computer vision hardware algorithms using xilinx system generator. 2004.

Alan S. Willsky Alan V. Oppenheim. *Signal e Systems*.

D. G. Bailey C. T. Johnston, K. T. Gribbon. Adaptive edge detection for real-time video processing using fpgas. 2005.

N. ANDRESEN D. DOERING. High speed, direct detection lk frame-store. *IEEE Nuclear Science Symposium Conference Record*, 2011a.

N. ANDRESEN K. CHOW D. CONTARATO D. DOERING, Y.-D. CHUANG. Development of a compact fast ccd camera and resonant soft x-ray scattering endstation for time-resolved pump-probe experiments. *Rev. Sci. Instrum.*, 2011b. doi: 10.1063/1.3609862.

A.Bouridane-K.Alotaibi D.Crookes, K.Benkrid and A.Benkrid. Design and implementation of a high level programming environment for fpga-based image processing. *IEE Proceedings online no. 20000579 DOI: 10.1049/ip-vis:20000579*, 2000.

Dionisio Doering Devis Contarato. Clustering algorithms lbnl 04162010. 2010.

A. Hazanchuk H. Shan Neoh. Adaptive edge detection for real-time video processing using fpgas. 2005.

A. Amira I.S. Uzun and A. Bouridane. Fpga implementations of fast fourier transforms for real-time signal and image processing. *IEE Proc.-Vis. Image Signal Process., Vol. 152, No. 3*, 2005.

F. JOSEF RAMMIG M. AURELIO WEHRMEISTER, C. EDUARDO PEREIRA. An aspect-oriented model-driven engineering approach for distributed embedded real-time systems. *Computer Society*, 2009.

L. CARRO M. VOGUEL PINTO. Implementação do protocolo can utilizando simulink para geração automática de vhdl. 2011.

H. A. Padmore J.-P. Walter P. Denes, D. Doering and J. Weizeorick. A fast, direct x-ray detection charge-coupled device. *Rev. Sci. Instrum.*, 2009. doi: 10.1063/1.3187222.

R. Woods R. Gonzalez. *Digital Image Processing*.

D. C. SCHMIDT. Model driven engineering. *IEEE Computer Society*, 2006.

B. SELIC. The pragmatics of model-driven development. *Computer Society*, 2003.

K. WEISS. Reusable specification components for model-driven development. *IEEE Nuclear Science Symposium Conference Record*, 2003.

TRABALHO DE GRADUAÇÃO I
Algoritmos para processamento de imagens visando implementação em FPGA utilizando engenharia baseada em modelos.

Lucas Dal Piaz Nunes[1]

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

{lpnunes}@inf.ufrgs.br

Abstract. *This work aims the implementation of different processing image algorithms for FPGA implementation. To this purpose, rather than follow a standard design flow where the code would be programmed using a hardware description language, will be used MATLAB and Simulink environment as possible validation and generation of HDL code.With this approach, it is expected to evaluate the benefits of using engineering-based models and their possible restrictions for this particular use. The first case study detailed and proposed in this article, refers to the reconstruction of signals in particle detection analysis.*

Resumo. *Este trabalho tem por objetivo a implementação de diferentes algoritmos de processamento de imagens visando sua implementação em FPGA. Para tal, ao invés de seguir um fluxo padrão de projeto onde o código seria programado usando uma linguagem de descrição de hardware, será utilizado o MATLAB e o Simulink como ambiente de validação e geração de código HDL. Com esta abordagem, espera-se verificar os benefícios em se utilizar engenharia baseada em modelos e suas eventuais restrições para este uso em particular (processamento de imagens). O primeiro estudo de caso proposto e detalhado neste artigo, refere-se a reconstrução de sinais na análise de detecção de partículas.*

1. Introdução

A complexidade de sistemas embarcados cresce à medida em que novas necessidades, dispositivos e restrições são impostas. Antigamente, grande parte destes sistemas eram compostos por um único processador que realizam tarefas simples. No entanto, hoje em dia o cenário é outro, havendo sistemas com múltiplos processadores e dispositivos que interagem entre si.

Diante desta nova realidade, surgiram novas linguagens de programação com níveis de abstração mais elevadas. Por exemplo, grande parte dos desenvolvedores usam linguagens orientadas a objetos tais como C++, C#, java ao invés de C ou Fortran. Desta forma, a introdução de classes possibilitou uma melhor reutilização de código e mantenabilidade se comparada com as linguagens de gerações anterior. Apesar dos avanços, vários problemas persistem:

- A falta de uma visão global geralmente limita os desenvolvedores para uma solução não otimizada. [WEISS 2003]

- É necessário atenção aos recursos da linguagem que serão utilizados para uma determinada solução. Por exemplo, em C++ com a adição de novas características tais como herança, funções virtuais, polimorfismo, entre outras, acaba-se tirando o foco em performance e questões arquiteturais [WEISS 2003].

Diante destas restrições, a metodologia de software definida como engenharia baseada em modelos é proposta. Nela, o foco é a representação abstrata do conhecimento das atividades que governam um certo domínio particular de aplicação. A sua maior vantagem é que expressamos modelos usando conceitos que estão muito menos ligados às implementações tecnológicas e mais ligados ao domínio específico do problema. Estas características, fazem com que o modelo se torne mais fácil de se especificar, entender e manter. Fazendo com que especialistas, não necessariamente ligadas a programação, desenvolvam sistemas [SELIC 2003].

Neste contexto, será utilizado o MATLAB\Simulink como ferramenta, visto que ele possui ferramentas de simulação e desenvolvimento guiado a modelos. Possui, também, um ambiente gráfico e uma série de blocos parametrizáveis com soluções parciais para certos domínios (incluindo o de processamento de imagens). Além, da geração automática de código para algumas arquiteturas. Neste trabalho, por exemplo, será utilizado o HDL coder para geração automática de código VHDL.

O primeiro estudo de caso proposto, deve-se ao fato das câmeras científicas de alta velocidade possuírem cada vez mais pixeis e frames\s aumentando o fluxo de informações. Uma forma de se lidar com estas exigências é utilizar processamento de imagens em tempo real utilizando FPGA. O desafio, porém, é a que cada novo experimento exigem-se novos algoritmos de processamento de imagens. Neste âmbito, pretende-se utilizar um desenvolvimento baseado em modelos através dos softwares MATLAB\Simulink e verificar os benefícios e restrições em utilizar tal metodologia.

2. Revisão bibliográfica

Em um primeiro momento será dado uma descrição detalhada das ferramentas que serão utilizadas. Após, será abordado detalhes da câmera e do algoritmo, assim como alguns conceitos básicos que auxiliarão na melhor compreensão do algoritmo.

2.1. Ferramentas

O simulink é uma ferramenta gráfica de programação e simulação desenvolvida pela MathWorks que usa a arquitetura de software definida como fluxo de dados. Sua interface de programação é realizada através de blocos (em sua maioria parametrizáveis) no qual existe um conjunto de bibliotecas para diferentes propósitos. A biblioteca DSP System será a mais utilizada visto que ela possui as operações básicas para processamento de sinais. Caso seja necessário o desenvolvimento de uma máquina de estado para a solução será utilizado a biblioteca StateFlow.

Outro fator que estimula o uso do Simulink é a possibilidade de iteração com o ambiente de desenvolvimento Matlab. Desta forma, é possível a troca de dados entre os dois e a utilização das duas ferramentas simultaneamente. Em particular, existe um bloco do simulink que permite a escrita de funções usando a própria linguagem de programação Matlab.

A Mathworks também prove a geração automática de código a partir de modelos Simulink. O HDL Coder gera código Verilog ou VHDL a partir de modelos Simulink, códigos do Matlab e diagramas do Stateflow. No entanto, alguns cuidados devem ser tomados, visto que nem todos os blocos são sintetizáveis.

Os modelos que possuem trigger e enable também devem ser utilizadas com certa cautela pois podem tirar a propriedade de cicle-accurate e bit-true da simulação com o código sintetizado. A primeira propriedade, cicle-accurate, ocorre quando não há uma correlação temporal entre o modelo e o código HDL gerado. Alguns blocos do Simulink quando otimizados para área podem introduzir atrasos, por exemplo. Já a segunda, bit-true, ocorre ao haver diferenças na composição interna de um bloco e o código gerado. Por exemplo, ao implementarmos um somador e após o otimizarmos tendo em vista performance, a estrutura interna pode ser rearranjada.

2.2. Câmera utilizada para a obtenção de imagens

As principais características da câmera de onde foram obtidas as imagens para análise são:

- Possui sensores de imagem CCD (dispositivo de carga acoplada).
- Sua tecnologia de construção permite a absorção de raios X com energia de até 10 Kev.
- Possui uma taxa de leitura de 200 Mpixeis/s obtida através de uma leitura paralela do CCD na qual cada grupo de dez colunas possui um estágio de saída. Desta forma, aumenta-se o fluxo de dados através de uma leitura paralela dos mesmos. Evitando-se, desta forma, os efeitos indesejados do simples aumento de frequência de operação (ruído, potência).
- A matriz de detecção do sensor CCD é composto de 480x480 pixeis. O tempo necessário (em segundos)para a leitura do CDD é obtida por:

$$T = \frac{N_y}{2}(T_v + (\frac{N_x}{m})T_h) \tag{1}$$

Sendo N_y e N_x a quantidade de pixels, m a quantidade de estágios de saída, T_v o tempo do clock que paraleliza as colunas e T_h o tempo necessário para serializar uma coluna. O fator de divisão 2, deve-se ao fato de haver m portas na parte superior e m na parte inferior do display.

2.3. Algoritmos padrões utilizados para o posicionamento e reconstrução de sinais na análise de detecção de partículas

Segue algumas definições utilizadas para auxiliar na explicação destes algoritmos:

- Cluster - Consiste de um grupo contínuo de pixeis que possui uma intensidade de energia recebida maior que o ruído presente em sua vizinhança. Geralmente, há um pixel central (seed pixel) cercado por sua vizinhança.
- Seed pixel - É definido como sendo o pixel com maior Sinal/Ruído presente no cluster ou, como sendo o pixel de maior intensidade. Estes conceitos tendem a ser equivalentes caso haja uma distribuição uniforme do ruído sobre a matriz de pixeis.

- Cluster size - Refere-se a quantidade de pixeis além do seed pixel que satisfazem a relação S/R por exemplo.
- Bad Pixel - Pixel que possui um mal funcionamento. Geralmente é reconhecido pela aquisição de imagens escuras e analise dos pixeis que possuem um ruído ligeiramente maior do que a média do ruído obtido.
- Hot Pixel - Apresenta um grande sinal recebido mesmo em sua absência.

Os algoritmos para reconstrução de sinais necessitam basicamente de duas variáveis externas. A primeira, refere-se ao limiar que será estabelecido para selecionar os pixeis de interesse. A segunda, ao tamanho da matriz analisada em torno dos pontos de interesse. Esta última, tem relação direta com a energia incidente em torno do ponto considerado.

Antes de realizarmos o processamento propriamente dito podemos falar em um pré-processamento que consiste na subtração das imagens escuras, equalização dos pixels considerando que não há uma distribuição homogênea do ruído e um mascaramento dos bad pixels.

A primeira operação de processamento consiste na obtenção dos Seed Pixeis obtida através da análise completa da matriz de pixeis. Logo em seguida, assegura-se que haja apenas um Seed pixel em torno de uma região de interesse. Utiliza-se para tal uma lista decrescente no qual verifica-se se um outro ponto esta contida na região de interesse deste pixel. Caso estiver, elimina-se este pixel da lista e continua-se esta operação até a lista estar vazia.

O ponto negativo de tal procedimento é que os hot Pixels podem ser processados caso não sejam eliminados anteriormente. Porém, estes podem ser eliminados ao notarmos uma frequência grande de determinados pixels em um dado conjunto de imagens.

Após, analisa-se os clusters individualmente e caso forem satisfeitos aspectos como a relação sinal ruído e energia, segue-se para uma análise ou pós-processamento.

2.4. Cluster

Conceitos utilizados para os clusters (região em torno de um ponto de interesse):

- Cluster integrity - Verifica se os pixeis são contínuos internamente ao cluster.
- Hot pixel identification - Após a análise de uma certa quantia de frames pode-se verificar se um pixel teve uma taxa alta de ocorrência se comparado aos demais.
- Cluster shape - A forma esperada para um cluster depende basicamente do ângulo em que um elétron incide na matriz do CCD. Caso houver uma incidência perpendicular, espera-se encontrar formas arredondas. Caso contrário, formas mais alongadas são esperadas. Esta propriedade é intrínseca da onda analisada.

2.5. Algoritmos para reconstrução de posição

O impacto das partículas no detector permite a reconstrução da posição graças a carga coletada nesta região. O algoritmo mais simples, seria a simples detecção do seed pixel no qual nos garante uma precisão de $\frac{P}{\sqrt{12}}$ em uma abordagem de erro da raiz quadrada média. P, refere-se a distância entre pixeis. No entanto, ao coletarmos a informação completa do cluster podemos aplicar algum método de interpolação garantindo desta forma uma melhor precisão. Uma segunda proposta, é utilizar o centro de gravidade para reconstruir a posição do seed pixel.

2.6. Implementação em hardware

Nesta implementação propõe-se uma adaptação do algoritmo anterior visando a simplicidade da arquitetura gerada. Caso fosse gerado um valor médio de ruído para eliminação dos bad pixeis e equalização dos pixeis, por exemplo, seriam necessários recursos como aritmética de ponto flutuante. Ao supormos que o ruído em torno dos pixeis é constante assim como o seu PSR (Point Spread Function) espera-se um resultado equivalente ao utilizarmos um algoritmo baseado em corte de sinais de carga.

Na primeira etapa de pré-processamento calcula-se o ruído médio de cada pixel. Esta etapa propõe-se fazer em software e, após obtido o valor para cada pixel, carrega-se em hardware para as demais etapas de processamento.

O algoritmo mais simples seria aplicar um único threshold para todos os pixeis e mandar a posição destes para uma etapa posterior, sendo que a utilização de um threshold alto tende a selecionar os seed pixels diretamente e um valor baixo um cluster. A figura 1 exibe a arquitetura base e após temos a análise dos blocos:

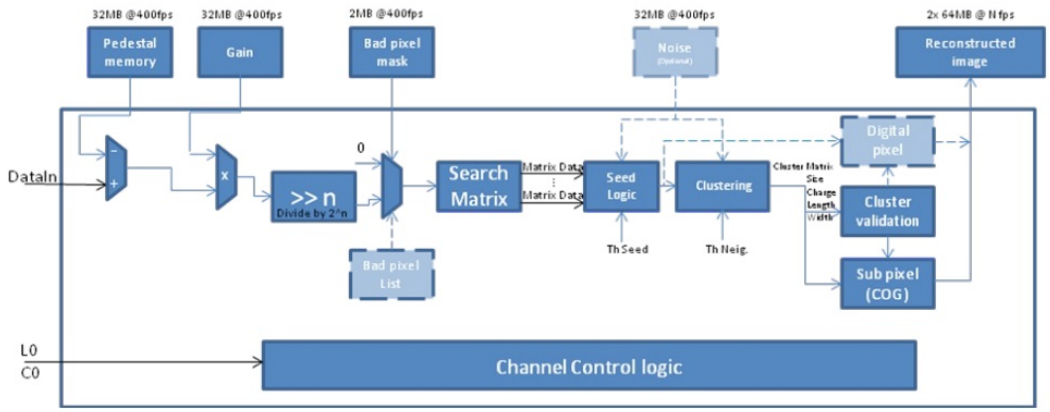

Figura 1. Diagrama de blocos da arquitetura analisada.

- Pedestal subtraction - O valor médio de um conjunto de imagens escuras é armazenado em pedestal memory para ser subtraído da imagem que se deseja analisar.
- Gain correction - A matriz Gain possui o ganho que deve ser aplicado a cada pixel. Logo, após a operação pedestal subtraction faz-se a multiplicação e em seguida uma divisão realizada através de um shifter register.
- Bad pixel removal - Remove os bad pixeis. Para isto. propõe duas alternativas. A primeira contém uma matriz dos bad pixeis e a segunda possui uma lista das coordenadas referentes aos bad pixeis. Apesar da segunda alternativa precisar de menos memória mantém a arquitetura mais simples por não precisar de um comparador extra além de uma lógica de controle adicional.
- Search Matrix - Etapa que irá armazenar os dados de um cluster. A figura 2 mostra a sua arquitetura considerando uma matriz 5x5.
- Seed logic and clustering - Irá encontrar o seed pixel. Para tanto, verifica se o pixel é um máximo local do cluster.

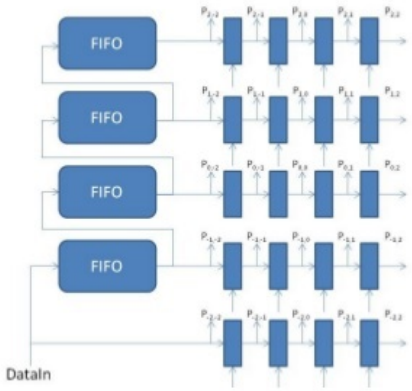

Figura 2. Diagrama para obtenção dos seed pixeis.

Por fim, escolhe-se um método para calcular o centro de gravidade e outras métricas que sejam de interesse.

3. Primeiros experimentos

3.1. Visão global da solução

Em um primeiro momento, buscou-se identificar um seed pixel a fim de melhor compreender a sua estrutura. Desta forma, antes de começar o desenvolvimento de modelos em simulink, já fora possível obter as imagens esperadas.

Abaixo podemos visualizar a forma típica de um cluster (figura 3) estando o seed pixel localizado no centro na imagem. Ao aplicarmos uma função interpolante obtemos a figura 4.

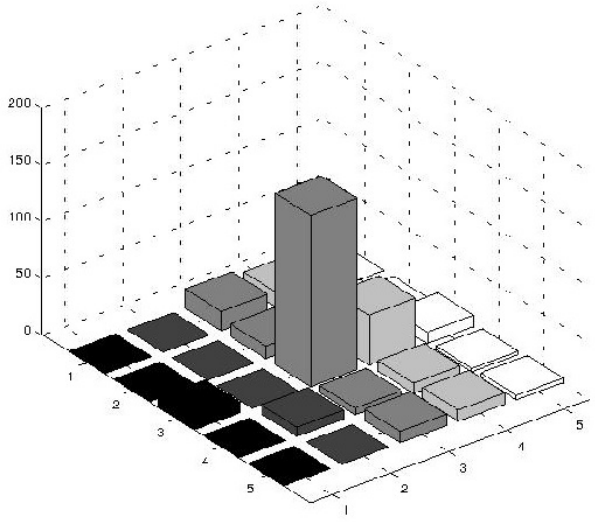

Figura 3. Seed pixeis.

Após estes primeiros testes, buscou-se construir modelos em simulink que simule e gere o algoritmo acima descrito visando sua implementação em FPGA.

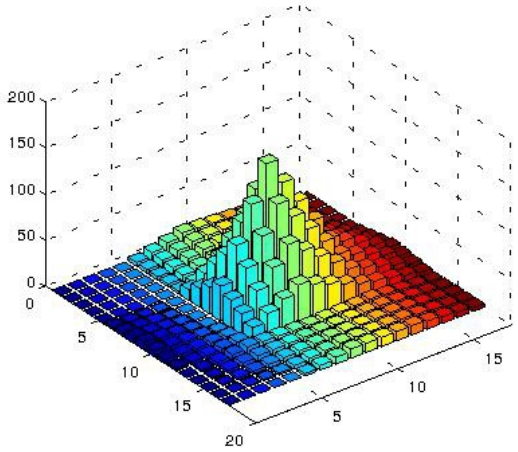

Figura 4. Seed pixeis obtido com função interpolante.

3.2. Modelo com processamento serial

Os primeiros experimentos utilizando Simulink foram voltados ao processamento de um único canal (figura 5) com o objetivo de determinar os seedPixeis. A única vantagem deste em relação ao modelo com múltiplos canais é a sua simplicidade de implementação.

As imagens de entradas e a visualização são importadas e exportadas afim de separar a lógica funcional da simulação. Cabe lembrar que as funções de pré-processamento (leitura das imagens e cálculo do ruído médio) já foram desenvolvidas em Matlab não havendo necessidade de conversão para Simulink.

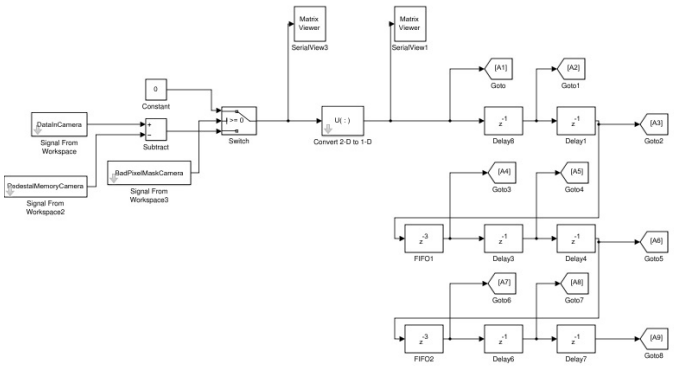

Figura 5. Modelo de um canal.

Neste primeiro diagrama, podemos verificar os blocos que foram obtidos a partir do pré-processamento. Por exemplo, ao inserirmos uma imagem 5x5 a fim de obter seus seedPixeis considerando uma mascára 3x3 em torno do ponto de interesse para tal, obtemos (figura 6):

Após identificado o seedPixel, segue-se a sequencia de processamento. Neste exemplo, somam-se todos os pixeis em torno de um seedPixel a fim de obter-se a energia presente neste cluster (figura 7).

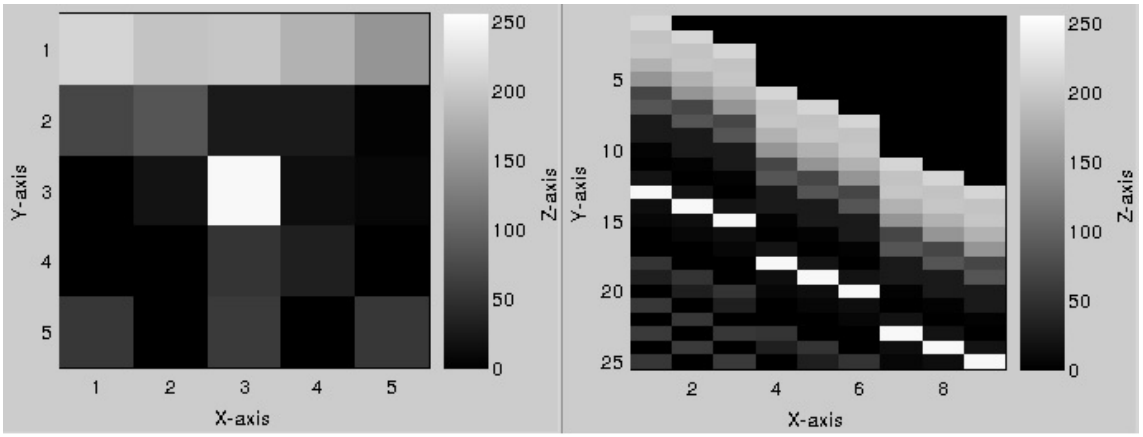

Figura 6. Imagem de entrada e valores internos da máscara 3x3 que serão utilizados para determinar os seedPixeis.

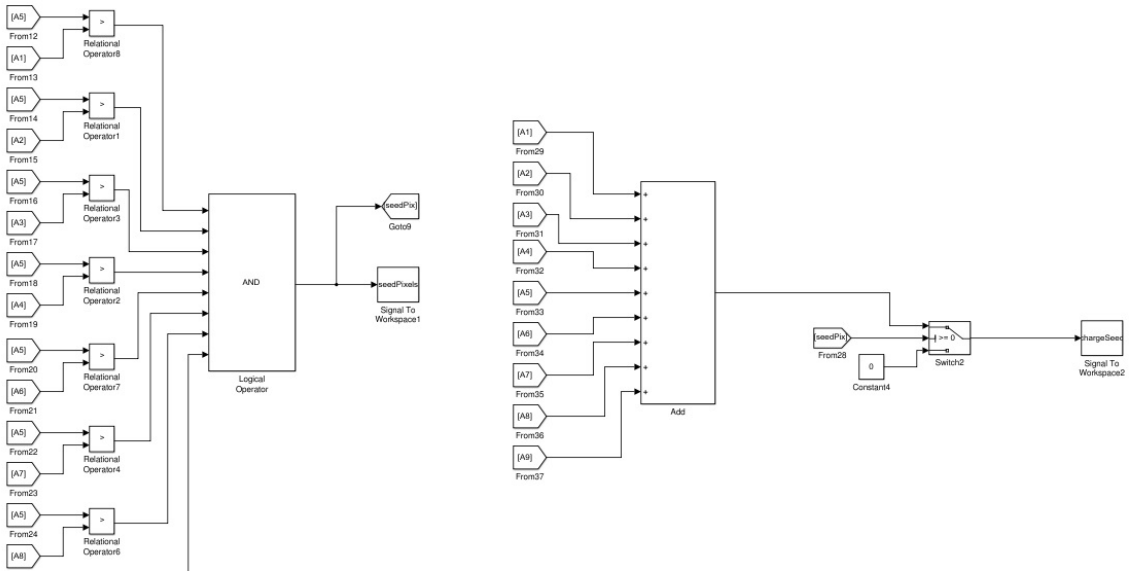

Figura 7. Avaliação do Seed pixel e a energia total de um cluster.

3.3. Modelo com processamento Paralelo

Tendo em vista que a captura de imagem da câmera utilizada é composta por um conjunto de canais lidos concorrentemente e que compartilham informações para a composição dos seedPixeis, devemos modificar a estrutura acima.

A principal vantagem deste modelo em relação ao serial,é a velocidade com que o processamento será realizado. Na câmera utilizada, por exemplo, como há 96 canais devemos esperar um ganho de velocidade de mesma magnitude. No entanto, será necessário a utilização de lógicas de controle e armazenamento extras para calcular os seedPixeis.

Por exemplo, ao considerarmos uma imagem 20x20 (figura 8) composta por 8 canais de 5x10 e que o filtro aplicado para geração será de 3x3, percebemos que apenas os pixeis em branco podem ser calculados localmente (dentro do seu próprio canal). Os demais (em colorido), necessitam de informações adicionais da sua vizinhança. Os valores presentes na matriz, referem-se ao momento em que a informação está disponível

para o cálculo do seed pixel e as diferentes cores referem-se as lógicas que deveriam ser aplicadas para sua obtenção.

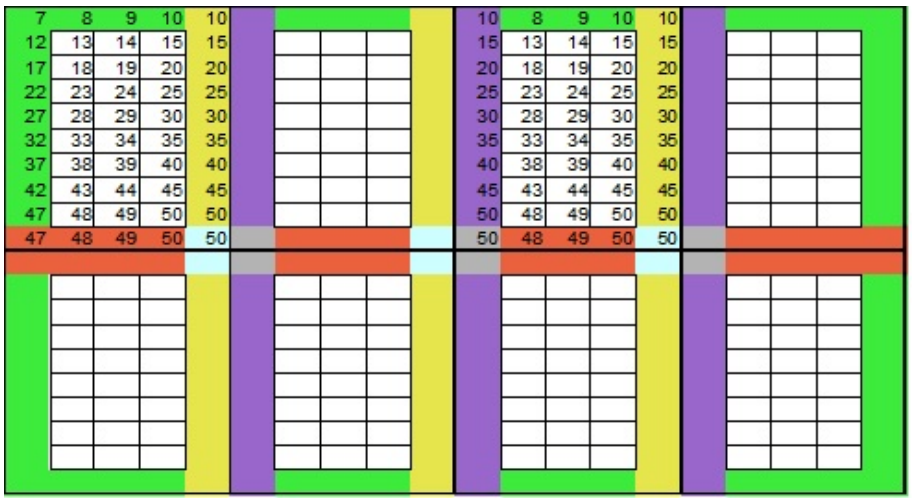

Figura 8. Pixel clocks para cálculo do seed pixel.

Visando a simplicidade, no entanto, expande-se as regiões de fronteira de um canal e aplica-se a mesma lógica para a geração dos seedPixeis como se fosse um único canal. Para tal, foi desenvolvido o modelo abaixo baseado nas dimensões da matriz acima demonstrada que possui a lógica de controle completa e é capaz de montar uma linha até o momento.

Abaixo, geração dos clocks e do contador que será utilizado para sincronizar os dados provenientes de diferentes canais (figura 9).

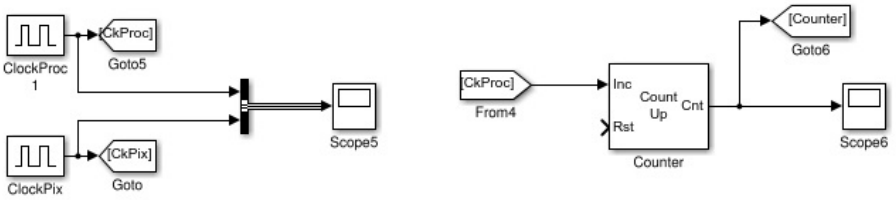

Figura 9. Oscilador e contador.

Em seguida, na figura 10, temos a obtenção dos dados (modelo a esquerda), lógica de controle para salvar os pixeis e removê-los da FIFO (modelos centrais) e, por fim, o seu armazenamento em estruturas do tipo FIFO (modelo a direita). Desta forma, cada canal possui uma FIFO para armazenar os dados de sua linha além de 6 registradores para os canais vizinhos que ainda precisa ser implementada.

Na última etapa, seleciona-se os pixeis provenientes de diferentes canais de acordo com a sequencia desejada a fim de formar uma única linha (figura 11). Para tanto, utiliza-se um multiplexador para selecionar os dados de diferentes canais que serão armazenados em registradores na etapa seguinte. A figura 12 mostra a disponibilidade dos dados na entrada do buffer.

A fim de validar este modelo, inseriu-se no fluxo de entrada um array crescente iniciando em 1 para todos os canais e a cada nova leitura de pixel incrementa-se de uma

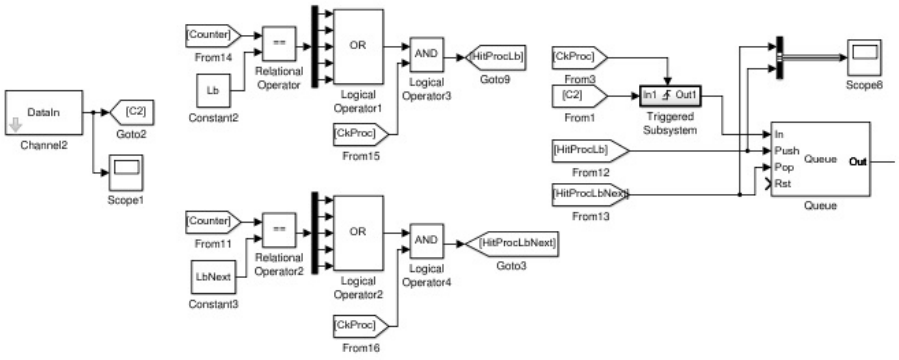

Figura 10. Armazenamento de dados para uma linha.

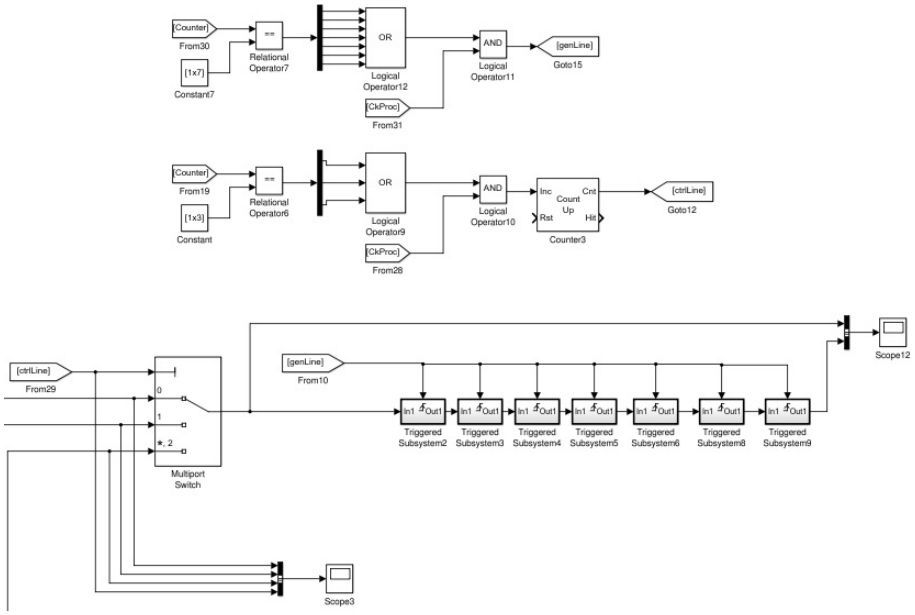

Figura 11. Geração de um linha completa de pixeis envolvendo diferentes canais.

unidade. Como resultados, obtemos a seguinte imagem ao final da segunda linha (figura 13):

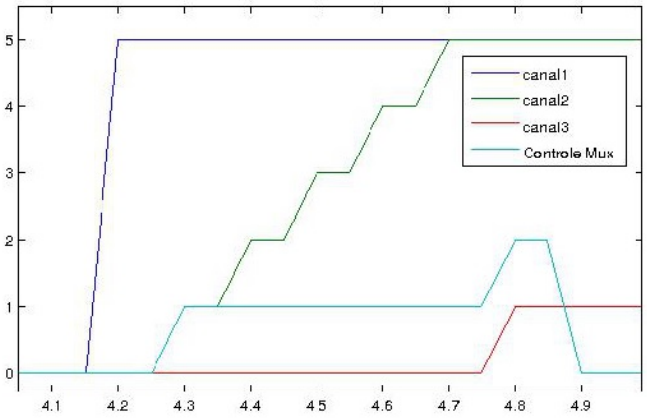

Figura 12. Valores dos pixeis lidos de diferentes canais presentes na entrada do MUX.

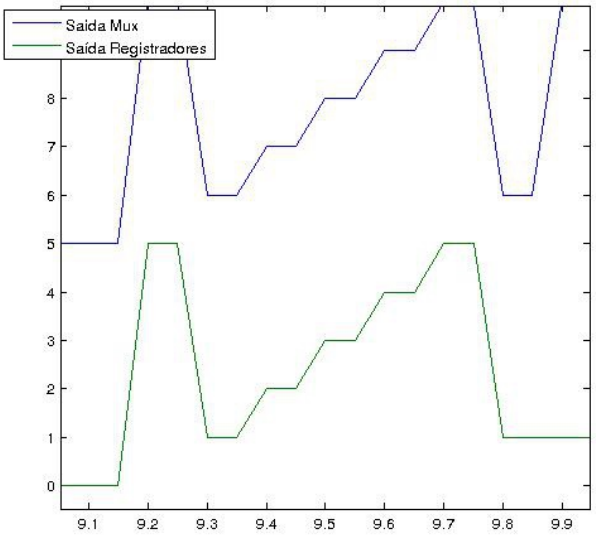

Figura 13. Linha gerada por diferentes canais.

4. Considerações finais

Neste artigo, foi apresentado as principais características da câmera e alguns algoritmos para reconstrução de sinais que já começaram a ser desenvolvidos. Num primeiro momento, utilizou-se uma abordagem serial para o processamento de imagens e , após, uma abordagem paralela.

Durante o desenvolvimento destes algoritmos, pode-se observar algumas vantagens em se utilizar a metodologia proposta:

- Visualização dos resultados em diferentes domínios - Dependendo da etapa de construção da solução diferentes testes são necessário para verificar o correto funcionamento do sistema. Por exemplo, após aplicar um filtro em uma imagem, gostaríamos de verificar o resultado da imagem resultante. Em um segundo momento, podemos desejar recuperar a lista de diferentes sinais de sincronização para verificar se foram gerados corretamente.
- A facilidade em readaptar um algoritmo frente a alguma mudança proposta - Ao passar para a abordagem multicanais, reaproveitou-se toda a parte desenvolvida anteriormente: obtenção das imagens e filtro. Bastando adicionar a lógica de controle e registradores a fim de realizar a sincronização dos dados.
- Visão global da solução - A medida que criamos modelos mais complexos e que possuem funções bem definidas, podemos abstrair os detalhes do seu funcionamento a fim de aumentar o nível de abstração e possibilitar a visualização de uma solução global.
- Geração de código para arquitetura alvo - Neste trabalho, será aproveitado o gerador HDL a partir de um modelo simulink. Embora, possa existir outras linguagens alvo.

Após a conclusão dos algoritmos envolvendo múltiplos canais, pretende-se utilizar o gerador HDL do simulink para verificar o código gerado. Acredita-se que esta será a etapa mais delicada do projeto pois nem todos os bloco possuem uma conversão direta para o código HDL.

Concluída estas etapas, novos algoritmos que envolvam processamento de imagens podem ser desenvolvidos assim como uma comparação do código VHDL gerado com uma solução desenvolvida manualmente.

Bibliografia

D. DOERING, N. A. (2011a). High speed, direct detection lk frame-store. *IEEE Nuclear Science Symposium Conference Record*.

D. DOERING, Y.-D. CHUANG, N. A. K. C. D. C. (2011b). Development of a compact fast ccd camera and resonant soft x-ray scattering endstation for time-resolved pump-probe experiments. *Rev. Sci. Instrum.*

Devis Contarato, D. D. (2010). Clustering algorithms lbnl 04162010.

M. AURELIO WEHRMEISTER, C. EDUARDO PEREIRA, F. J. R. (2009). An aspect-oriented model-driven engineering approach for distributed embedded real-time systems. *Computer Society*.

M. VOGUEL PINTO, L. C. (2011). ImplementaÃ§Ã£o do protocolo can utilizando simulink para geraÃ§Ã£o automÃ¡tica de vhdl.

P. Denes, D. Doering, H. A. P. J.-P. W. and Weizeorick, J. (2009). A fast, direct x-ray detection charge-coupled device. *Rev. Sci. Instrum.*

SCHMIDT, D. C. (2006). Model driven engineering. *IEEE Computer Society*.

SELIC, B. (2003). The pragmatics of model-driven development. *Computer Society*.

WEISS, K. (2003). Reusable specification components for model-driven development. *IEEE Nuclear Science Symposium Conference Record*.