Gabriel Calin

PROCESSAMENTO DE VÍDEO ESTEREOSCÓPICO EM TEMPO REAL PARA EXTRAÇÃO DE MAPA DE DISPARIDADES

Dissertação apresentada à Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do Título de Mestre em Engenharia Elétrica.

Orientador: Prof. Dr. Valentin O. Roda

São Carlos

2007

Um singelo obrigado aos meus pais e minha noiva, pela compreensão e apoio incondicional durante essa caminhada.

PÁGINA INTENCIONAMENTE DEIXADA EM BRANCO

AGRADECIMENTOS

Um agradecimento à Vanderlei Bonato por gentilmente ceder o material de pesquisa de sua dissertação “Projeto de um módulo de aquisição e pré-processamento de imagem colorida baseado em computação reconfigurável e aplicado a robôs móveis”, sendo esta de grande valia durante o desenvolvimento desta pesquisa.

A Universidade de São Paulo, pela infra-estrutura oferecida, e aos funcionários da graduação e pós-graduação pela pronta disposição na solução de problemas.

Ao meu orientador, Prof. Dr. Valentin Obac Roda, por toda ajuda e orientação despendida.

PÁGINA INTENCIONAMENTE DEIXADA EM BRANCO

PÁGINA INTENCIONAMENTE DEIXADA EM BRANCO

RESUMO

CALIN, G. (2007).

. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2007.

A análise em tempo real de pares de imagens estereoscópicas para extração de características dimensionais da cena tem apresentado crescente interesse, possibilitando robusta navegação robótica e identificação de objetos em cenários dinâmicos. A presente dissertação propõe um método que emprega a análise a e observação de janelas, em pares de imagens estereoscópicas, para extração de denso mapa de disparidades. A arquitetura de processamento proposta é única em sua constituição, misturando elementos de processamento concorrente e seqüencial. O algoritmo estrutura-se em processamento , permitindo sua implementação em dispositivos de lógica programável e obtenção de resultados em tempo real.

Palavras-chave: visão estereoscópica; lógica programável; processamento em tempo-real; processamento .

ABSTRACT

CALIN, G. (2007).

. M.Sc. Dissertation – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2007.

Real-time analysis of stereo images for extraction of dimensional features has been focus of great interest, providing means for autonomous robot navigation and identification of objects in dynamic environments. This work describes a method based in pixel-to-pixel and windows based matching analysis, in stereo images, for constructing dense disparity maps. The proposed processing structure is unique, mixing concurrent and sequential elements. Pipelines structure is employed, targeting implementation in FPGA devices and enabling real-time results.

Keywords: stereo vision; programmable logic; real-time processing; pipeline processing.

LISTA DE FIGURAS

LISTA DE TABELAS

PÁGINA INTENCIONAMENTE DEIXADA EM BRANCO

SUMÁRIO

1 INTRODUÇÃO

A navegação autônoma de robôs e veículos autônomos tem apresentado considerável dificuldade quando tratada em ambientes dinâmicos.

Sistemas robóticos capazes de se adaptar a ambientes desconhecidos devem observar o mundo que os cerca e extrair características relevantes para execução de suas tarefas (navegação, identificação de objetos, etc.).

Para resolução de tais problemas pesquisadores têm utilizado uma ampla gama de transdutores (ultra-som, infravermelho, interferômetros laser, etc.) para extração de características tridimensionais do cenário, ou seja, avaliação de distância, forma e mesmo velocidade relativa de objetos.

Dos sistemas pesquisados, grande atenção tem sido dada aos módulos de visão estereoscópica. Estes módulos, constituídos tipicamente de um par de câmeras, têm o objetivo de obter múltiplas imagens de uma mesma cena sob diferentes pontos de vista, e processando-as adequadamente, extrair informações dimensionais da cena, permitindo finalmente a elaboração de uma aproximação tridimensional dos objetos observados.

Alvo de estudo e desenvolvimento nesta pesquisa, se definiu um algoritmo básico para extração de características de pares de imagens 2D, visualizando aplicação em dispositivos de lógica programável.

Em especial, foram utilizados como principal referência os métodos propostos por BIRCHFIELD e TOMASI (1998) e os trabalhos de HILE e ZENG (1996).

Para desenvolvimento do projeto selecionaram-se diversas ferramentas, dentre elas a plataforma de programação Delphi, o software para FPGA Xilinx ISE 6.3, e o simulador VHDL MentorGraphics ModelSim.

A Figura 1 exibe de forma estruturada a organização dos estudos e desenvolvimento, referenciando a cada um dos capítulos desta tese.

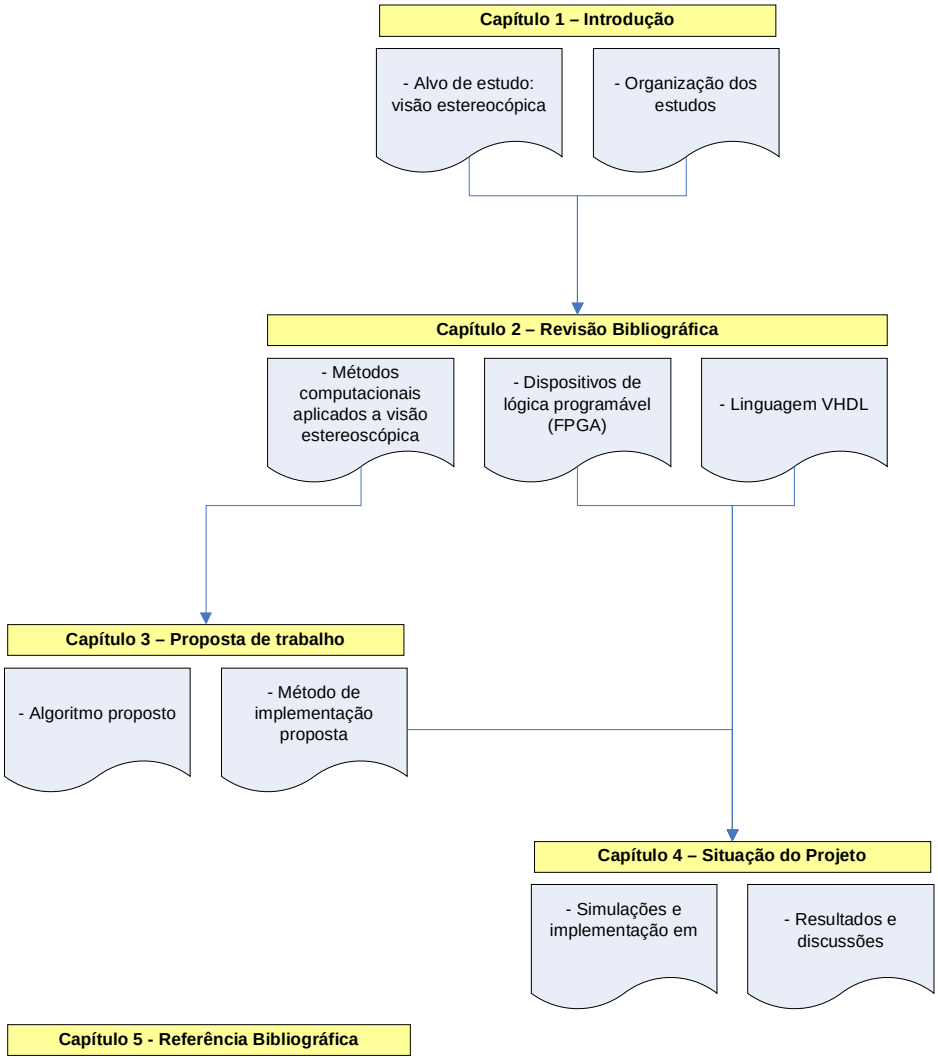


Figura 1 - Estrutura da monografia apresentada

2 REVISÃO BIBLIOGRÁFICA

2.1 Sistemas de visão estereoscópica

O núcleo funcional de qualquer sistema de visão estereoscópica está no processo de combinação das duas imagens com o objetivo de identificar pontos cuja origem é de um mesmo elemento da cena observada.

Este processo de identificação e quantização das distâncias entre as imagens apresenta uma série de dificuldades, especialmente devido à presença de ruído nas imagens, descontinuidades e oclusão de elementos.

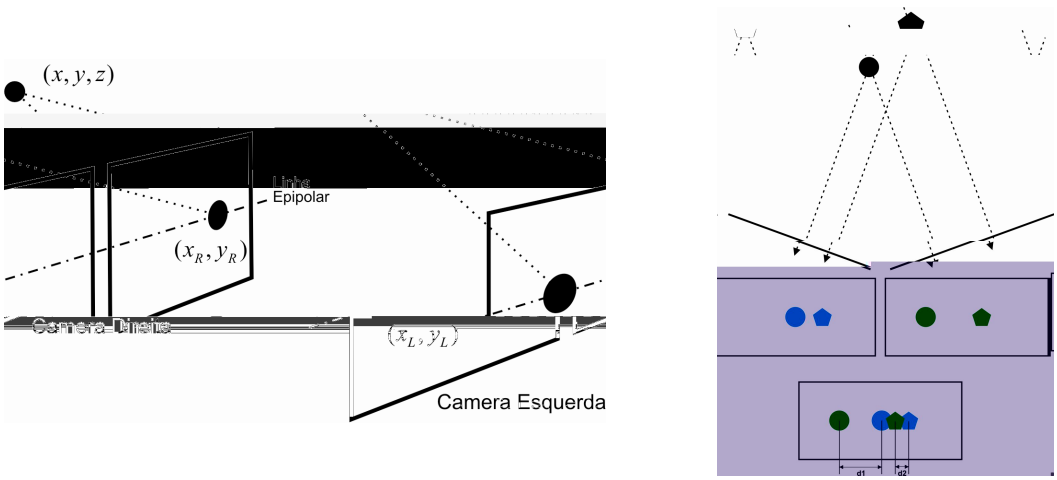


Figura 2 - Típica situação problema em visão estereoscópica

A Figura 2 exibe a típica situação problema em visão estereoscópica onde duas câmeras, observando uma cena, obtêm duas representações bidimensionais de um mesmo objeto tridimensional. Processamento adequado permite que as duas imagens sejam recombinadas, regenerando as informações tridimensionais perdidas.

Conforme modelamento do problema por GROSSO e TISTARELLI (1995), a extração da característica de profundidade pode ser obtida utilizando-se algumas observações trigonométricas.

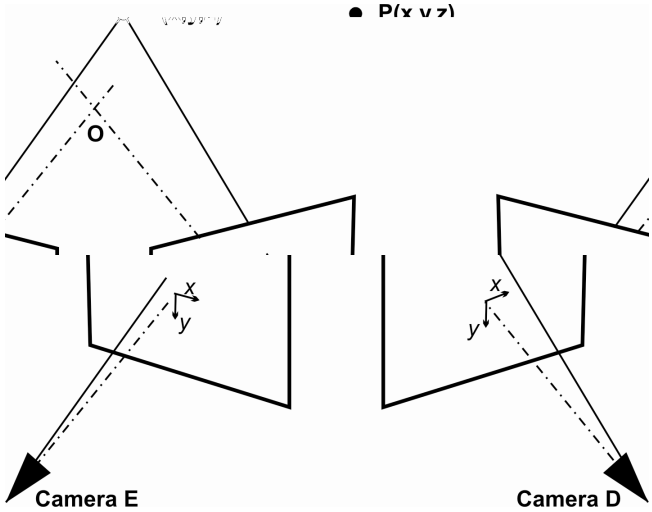


Figura 3 – Representação esquemática espacial de um sistema de visão estereoscópico.

Para análise do problema é definida uma representação espacial para a montagem típica de um sistema de visão estereoscópica, conforme mostrado na Figura 3. No esquema, o ponto P(x,y,z) é observado por duas câmeras e os eixos y das câmeras são paralelos e ortogonais ao plano definido pelos eixos ópticos. O ponto O indica o ponto de convergência focal das câmeras.

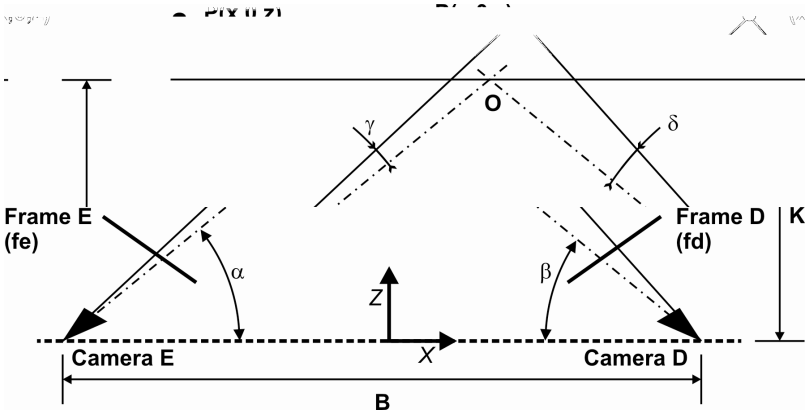


Figura 4 – Projeção (y=0) do modelo do sistema de visão estereoscópico.

Os parâmetros de interesse do sistema estudado estão identificados na Figura 4.

A fim de demonstrar que a profundidade do ponto P pode ser diretamente inferida dos parâmetros do sistema, será definida a função K, conforme indicado na eq

Atualmente existem três grandes vertentes de estudos para detecção e criação de mapas de disparidades. Suas bases fundamentam-se na análise de intensidade, características locais e fase.

BIRCHFIELD e TOMASI (1998), COX et al. (1996), BELHUMER (1993) propuseram no passado algoritmos cujo núcleo funcional se baseia na minimização de funções de custo, onde são computadas as diferenças absolutas ou quadráticas das intensidades de em pares de imagens epipolares. Estes métodos, apesar de resultados satisfatórios, são baseados em programação dinâmica, resultando custo computacional proibitivo para processamento em tempo real, sendo assim inadequados para implementação em .

HILE e ZENG (1996), YOSHIDA e HIROSE (1992), propõem algoritmos também baseados na observação da intensidade de , porém estruturados para implementação em dispositivos de lógica programáveis, o que confere processamento em tempo real, a custo de insensibilidade a oclusões e limitada região de análise para detecção de correspondências.

Outros pesquisadores, como TAKENO e HACHIYAMA (1991), MURRAY e JENNINGS (1991), GROSSO e TISTARELLI (1995) mostraram em seus estudos as novas possibilidades de caracterização de cena e navegação, fruto da obtenção em tempo real de informações tridimensionais do ambiente, mais especificadamente através da análise do mapa de disparidades.

O alvo de estudo desta pesquisa fundamenta-se na análise da intensidade local de , associada a estruturas de processamento concorrente.

2.1 Dispositivos de lógica programável - FPGAs

FPGAs ( ) são circuitos integrados passíveis de configuração por software, permitindo a implementação física de circuitos digitais, desde simples estruturas lógicas, como interfaces e decodificadores, até sistemas complexos, como processadores e controladores. Sua estrutura interna consiste de um arranjo fortemente condensado de blocos idênticos, compostos por portas lógicas, dispositivos e acionadores de Entrada/Saída. As conexões entre blocos são implementadas através de um mapa de conexões, realizado fisicamente (programação por fusíveis) ou espelhado em um dispositivo de memória.

Dispositivos modernos FPGA possuem basicamente três conjuntos de elementos configuráveis. O primeiro consiste de vários circuitos idênticos, compostos por e lógica combinacional, conhecidos por CLBs ( ). Os CLBs possuem os elementos funcionais básicos para implementação de lógicas seqüenciais e combinacionais. Usualmente são constituídos por 2 a 4 agregados a um conjunto genérico de lógicas combinacionais.

O segundo conjunto de elementos configuráveis consiste de circuitos de das saídas dos CLBs com o exterior do FPGA, conhecidos como IOBs ( ). Estes são constituídos por bidirecionais e saídas de alta-impedância. Através da programação adequada de um IOB, é possível obter pinos com função de entrada, saída, bidirecional ou coletor-aberto.

O terceiro e último grupo é composto pelas interconexões. As interconexões apresentam condensados de trilhas capazes de interconectar as entradas e saídas dos CLBs e IOBs para formar os circuitos digitais desejados. Geralmente a configuração é estabelecida por programação interna das células de memória estática, que determinam funções lógicas e conexões internas implementadas no FPGA. O processo de escolha das interconexões é chamado de roteamento e pode ser executado manualmente ou com auxílio de ferramentas de desenvolvimento.

2.2 Linguagem VHDL

Com a crescente sofisticação e diversificação tecnológica dos dispositivos de lógica programável tornou-se necessária a elaboração de uma linguagem de programação que permitisse a geração de circuitos lógicos a partir de descrições textuais algorítmicas.

Desenvolvido nos meados da década de 80 pelo Departamento de defesa dos EUA, a linguagem VHDL –

– tornou-se atualmente a principal ferramenta para programação de dispositivos de lógica programável, garantindo grande portabilidade no código e alto grau de abstração dos algoritmos programados.

Estruturado como uma linguagem de programação completa, os compiladores VHDL atuais permitem a programação de algoritmos estruturados baseado em modelos de hardware, verificação funcional através de ferramentas de simulação e por fim geração do mapeamento físico específico para implementação da lógica em um dispositivo ASIC ou FPGA.

A Figura 5 mostra um ciclo típico de desenvolvimento VHDL. Assim como uma linguagem de programação tradicional, o ciclo típico de programação parte de alta abstração (algorítmica) movendo-se em direção do código de máquina ou , no caso da linguagem VHDL.

Durante este processo, a depuração se torna possível por simuladores específicos, capazes de analisar características dos circuitos lógicos sem a necessidade de implementação no dispositivo físico.

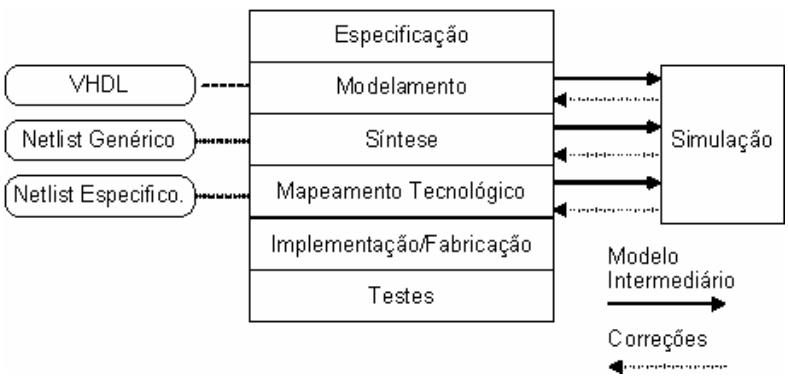


Figura 5 - Ciclo de vida típico de um projeto VHDL

3 PROPOSTA DE TRABALHO

3.1 Objetivos centrais

O projeto desenvolvido teve por objetivo a elaboração e implementação de um algoritmo para extração do mapa de disparidades de imagens estereoscópicas, focando futuras aplicações em dispositivos de lógica programável.

Neste sentido foi proposto um algoritmo que utiliza uma técnica de observação de intensidade dos constituintes de cada imagem e, mistura características de varredura de janelas e avaliação -a- para levantamento do mapa de disparidades.

A arquitetura proposta é única em sua construção uma vez que mistura elementos de processamento concorrente e seqüencial, visando sua implementação em dispositivos de lógica programável.

Para análise objetiva e comparação com resultados obtidos por outros pesquisadores – em especial BIRCHFIELD, S.; TOMASI, C. (1998) – foram utilizadas pares de fotos do repositório JISCT. Adicionalmente foram comparados resultados com pares de imagens estereoscópicas disponibilizadas pelo grupo de visão computacional da universidade de Tsukuba, e com pares de imagens estereoscópicas sintéticas (geradas através de algoritmos de) disponibilizadas pelo grupo de visão computacional da Universidade de Bonn.

Os objetivos deste trabalho foram realizados em duas fases distintas: simulação e implementação em lógica programável. Com isso procurou-se demonstrar as funcionalidades do algoritmo proposto, tanto como sua aplicabilidade em dispositivos de lógica programável.

3.2 Avaliação do algoritmo proposto

Um sistema típico de visão estereoscópica, constituído de um par de câmeras CMOS, uma placa de desenvolvimento (contendo um dispositivo de lógica programável), um computador (para monitoramento e programação) e interfaces de acessórias ao desenvolvimento, foram utilizados para avaliação do método proposto.

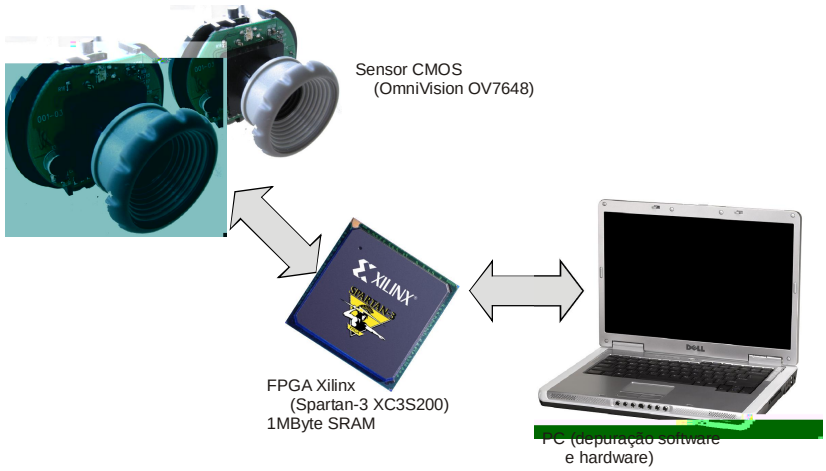


Figura 6 – Esquema geral do projeto proposto

Em uma primeira fase dos estudos o algoritmo proposto foi simulado – utilizando especialmente desenvolvido – e posteriormente implementado em um dispositivo de lógica programável, demonstrando sua capacidade de processar em tempo real as informações provenientes das câmeras CMOS e disponibilizar, em uma memória externa SRAM, as informação processadas.

4 MODELAMENTO, SIMULAÇÕES E IMPLEMENTAÇÃO

4.1 Parametrização e modelamento matemático

Para descrição do método proposto serão denotados F_e e F_d as imagens pré-processadas dos sensores CMOS, localizados respectivamente à direita e esquerda na montagem de ensaio. As imagens pré-processadas possuem largura de w por h de altura, sendo que cada destas imagens representa uma de 256 possíveis intensidades de cinza (quantização de 8-bit em níveis de cinza).

As imagens F_e e F_d serão tratadas na seqüência deste documento como sendo respectivamente os vetores $\vec{e}$ e $\vec{d}$, de comprimento $(w \cdot h)$.

$I_e(k)$ e $I_d(k)$ denotam as intensidades de um dado k nos vetores $\vec{e}$ e $\vec{d}$, onde $k \in \{0,1,...,(w \cdot h)-1\}$.

É definido o parâmetro ω como sendo a largura das janelas utilizadas, e β a região de observação em torno do k.

Define-se a função similaridade como sendo

$$S(j) = \sum_{i=k-\frac{1}{2}\varpi}^{k+\frac{1}{2}\varpi} [I_e(i) - I_d(i+j)]^2 \text{ com } i \in Z,\ \omega \in \{1,3,5,7,...\} \quad (5)$$

Define-se a função custo como sendo

$$C(k) = \min_{-\beta+1 \le j \le \beta} \{S(j)\} \quad (6)$$

O cálculo do mapa de disparidade ocorre a ; envolve, entretanto, a observação de janela centradas no em observação k, como mostra esquematicamente a Figura 7.

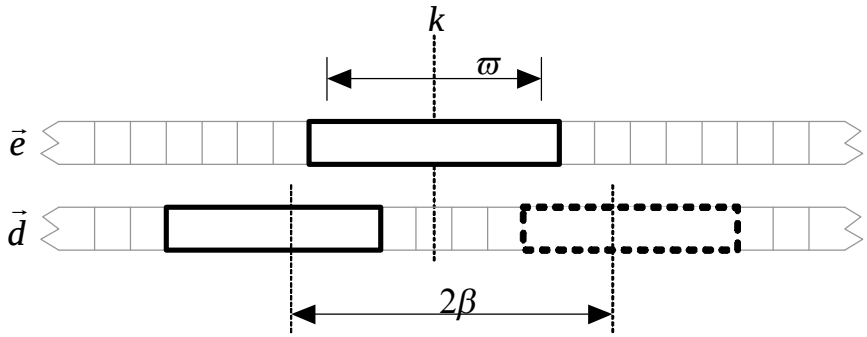


Figura 7 – Janelas observadas nos vetores $\vec{e}$ e $\vec{d}$

O modelamento algorítmico para processamento das funções similaridade e custo são apresentados nos tópicos a seguir.

4.1.1 – Pipelining e máquinas de estado

Para que a implementação do algoritmo no FPGA permita análise em tempo real das imagens, foi estudado e desenvolvido um forte arranjo híbrido, utilizando processamento em associado ao processamento seqüencial de diversas máquinas de estado. Isto permite que, após um desprezível atraso inicial (tipicamente inferior a 1μs, variando conforme profundidade dos de entrada), seja possível obter o mapa de disparidades da cena em tempo real.

Serão definidas diversas regiões de memória, utilizadas para processamento do algoritmo durante as diversas fases de e sub-processos seqüenciais.

As fases de foram estruturadas baseadas nos resultados de simulação, onde:

Dimensões de um quadro $(w \times h)$ =160x120 pixels

Largura da janela ω =11

Região de observação β =4

Como o algoritmo requer apenas uma limitada janela de observação para processamento da disparidade, é armazenada na memória interna do FPGA apenas uma pequena seqüência de provenientes das câmeras CMOS. Desta forma os vetores $\vec{e}$ e $\vec{d}$ podem ser limitadamente observados em torno de um dado k_0. Sendo assim, os vetores $\vec{e}$ e $\vec{d}$ serão tratados, para efeito de entendimento, como vetores das imagens das câmeras CMOS, possuindo entretanto apenas um limitado número de posições acessíveis para processamento.

Para alocação do vetor $\vec{e}$ serão utilizados internos do FPGA, consistindo de ω blocos com 8-bit de largura e designados por $e = \{ke_{-5}, ke_{-4}, ..., ke_0, ..., ke_{+5}\}$, conforme esquema da Figura 8.

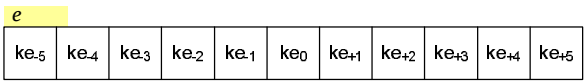


Figura 8 – Alocação do vetor $\vec{e}$ (88 bits)

De forma análoga, para alocação do vetor $\vec{d}$ serão utilizados internos do FPGA, consistindo de $\left[\omega + (2 \cdot \beta - 1)\right]$ blocos com 8-bit de largura e designados por $d = \{kd_{-8}, kd_{-7}, ..., kd_0, ..., ke_{+9}\}$, conforme esquema da Figura 9.

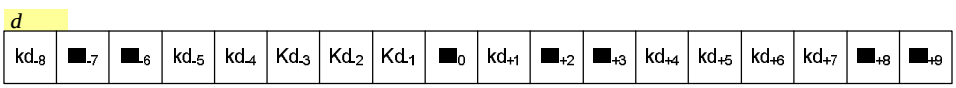


Figura 9 – Alocação do vetor $\vec{d}$ (144 bits)

Para execução do processamento síncrono das informações dos sensores CMOS e formação da cadeia em são necessários três sinais fundamentais de , como indicado na Figura 10.

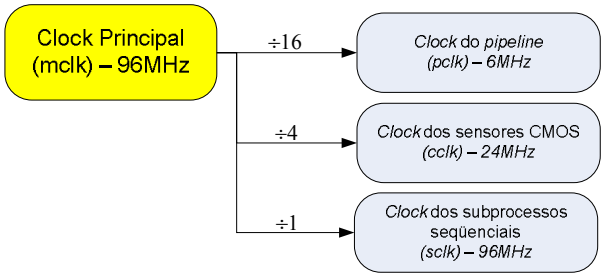


Figura 10 – Diagrama dos sinais fundamentais de *clock*

O principal (mclk) é sintetizado internamente ao FPGA, a partir do oscilador externo de 50MHz, disponível na placa de desenvolvimento. Este sinal de é base para geração dos demais sinais necessários, permitindo sincronização dos processos de transferência e máquinas de estado.

O do (pclk) é responsável pela execução de cada uma de suas fases, discutidas nas seções posteriores.

O dos sensores (cclk) é utilizado para obtenção e redução de 4:1 dos recebidos.

O dos sub-processos seqüenciais (sclk) é utilizado em diversas fases do processo em para execução de rotinas que não podem ser processadas concorrentemente, isto é, as máquinas de estado internas a cada uma das fases do processo de .

4.1.2 – Fases do processo em pipeline

As próximas seções detalham cada uma das fases do processo de , responsáveis pela movimentação, transformação e avaliação das informações obtidas das câmeras CMOS. Vale notar que, apesar da estruturação do processo em fases, estas ocorrem de forma concorrente, permitindo assim o fluxo contínuo de informações. Um esquema geral do processo é mostrado na Figura 11.

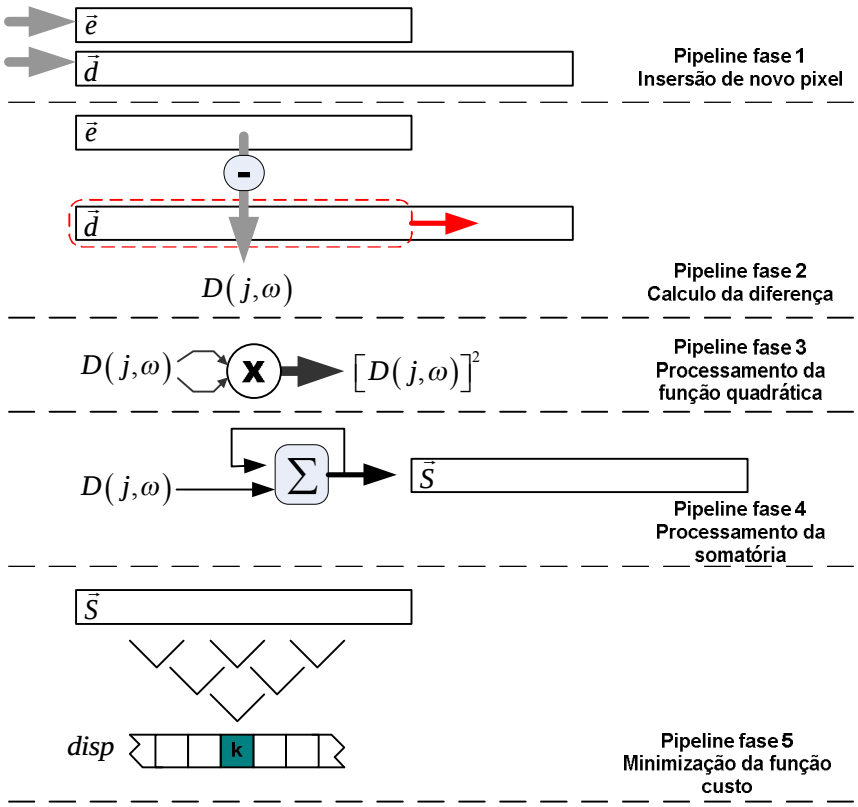


Figura 11 - Estrutura geral de processamento em pipeline para o algoritmo proposto

4.1.2.1 – Fase 1 – Movimentação dos vetores

A fase 1 do consiste em movimentar ambos os vetores para a esquerda, descartando os nas posições ke_{-5} e kd_{-8} e liberando as posições ke_{+5} e kd_{+9} para dois novos proveniente dos respectivos sensores CMOS.

Esta fase é realizada seqüencialmente em dois ciclos de , onde no primeiro sinal (sclk0), os vetores $\vec{e}$ e $\vec{d}$ são copiados para posições temporárias de memória, respectivamente e_{tmp} (alocado 80 bits) e d_{tmp} .(alocado 136 bits), indicado na Figura 12 e Figura 13.

O segundo da fase 1, sclk2, é sincronizado com o dos sensores CMOS (cclk), efetuando a cópia do conteúdo dos vetores temporários para os vetores originais e armazenando os novos disponíveis nas posições vagas, conforme indicado na Figura 14 e Figura 15.

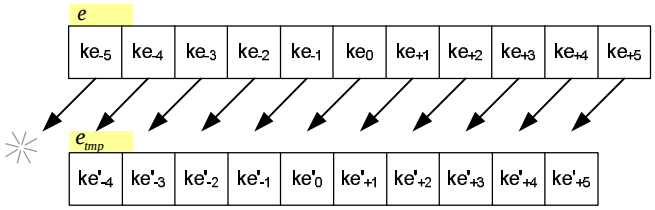


Figura 12 – Fase 1 do *pipeline* – Ciclo de *clock* sclk0 – Cópia do vetor $\vec{e}$

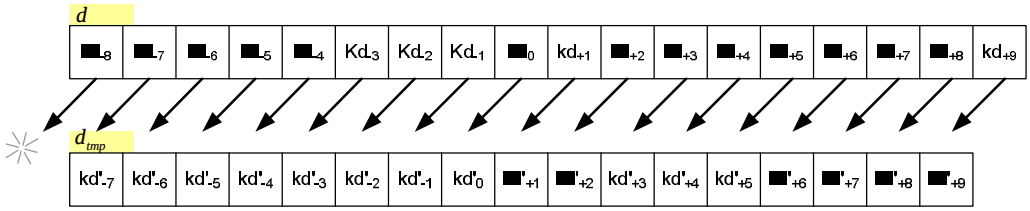


Figura 13 – Fase 1 do Pipeline – Ciclo de clock sclk0 – Cópia do vetor $\vec{d}$

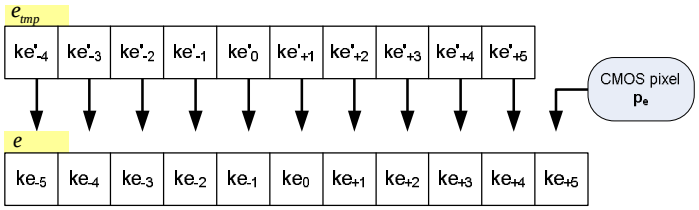


Figura 14 – Fase 1 do *pipeline* – Ciclo de *clock* sclk1 – Inserção do novo *pixel* p_e

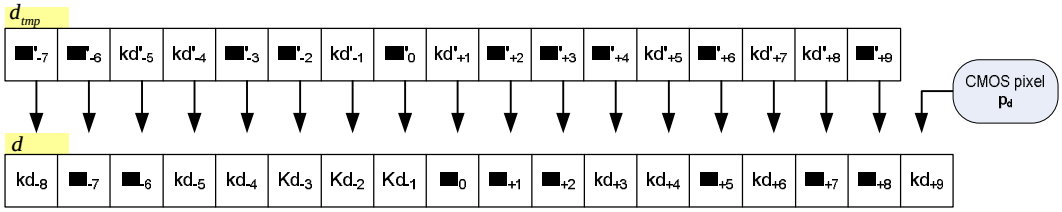


Figura 15 – Fase 1 do *pipeline* – Ciclo de *clock* sclk1 – Inserção do novo *pixel* p_d

Terminada a fase 1 os vetores $\vec{e}$ e $\vec{d}$ contêm novos provenientes dos respectivos sensores CMOS e estão prontos para a seqüência de cálculos de diferenças realizadas na fase 2.

4.1.2.2 – Fase 2 – Cálculo de diferenças entre vetores

Nesta fase são calculadas as diferenças de intensidade entre os [illegible]7.921TTj

ke e kd, observando-se para cada conjunto de diferenças o vetor $\vec{e}$ e uma das janelas de comparação do vetor $\vec{d}$.

As diferenças obtidas são armazenadas em blocos de memória interna do FPGA com largura de 9-bits (8-bits + sinal), cujo endereços foram designados:.

$$D(i,j) = I(ke_i) - I\left(kd_{(i+j)}\right) \tag{7}$$

onde i indica índice do vetor $\vec{e}$, com $i \in \{-5,-4,...,0,...,+5\}$ e j indica referência à janela utilizada no vetor $\vec{d}$, com $j \in \{-\beta+1,...,0,...,+\beta\}$ = $\{-3,-2,...,0,...,+4\}$.

Todos os cálculos ocorrem concorrentemente, sendo efetuado nesta fase de $\omega \cdot 2\beta = 88$ cálculos de diferenças, utilizando $88 \cdot 9 = 792$ bits da memória interna do FPGA.

A Figura 16 exibe esquematicamente o cálculo da diferença entre o vetor $\vec{e}$ e a janela $j = -\beta + 1$ no vetor $\vec{d}$.

De forma concorrente todas as janelas no vetor $\vec{d}$ são avaliadas frente ao vetor $\vec{e}$. A Figura 17 e Figura 18 exibem esquematicamente 7.216 Tc (90 Tc (a) 8

a ec (e) Ti0 Tc (s) Ti0.048 Tc (guc)f3.36 -1.44 Te a r

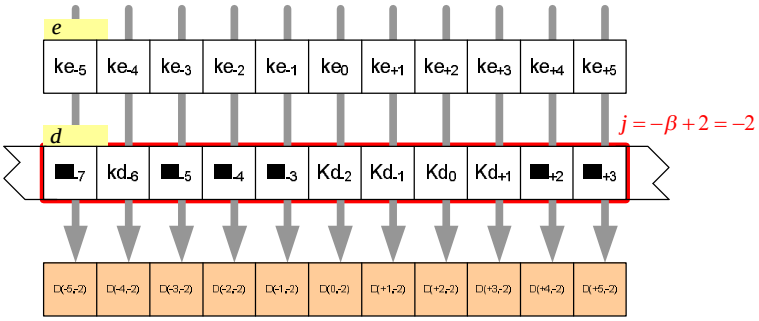


Figura 17 – Fase 2 do *pipeline* – Cálculo da diferença de intensidade – Janela $j = -\beta + 2$

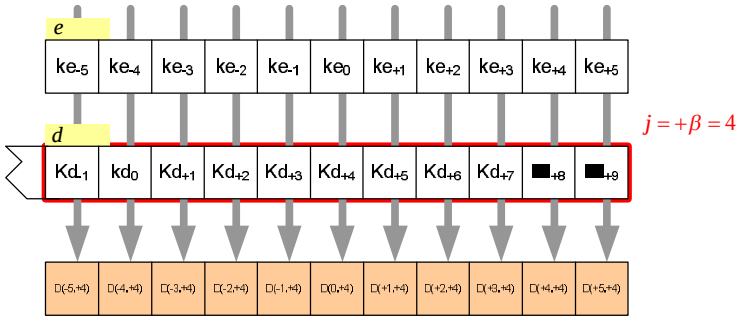


Figura 18 – Fase 2 do *pipeline* – Cálculo da diferença de intensidade – Janela $j = +\beta$

Como resultado da segunda fase do é obtido uma matriz contendo todas as diferenças calculadas (Figura 19).

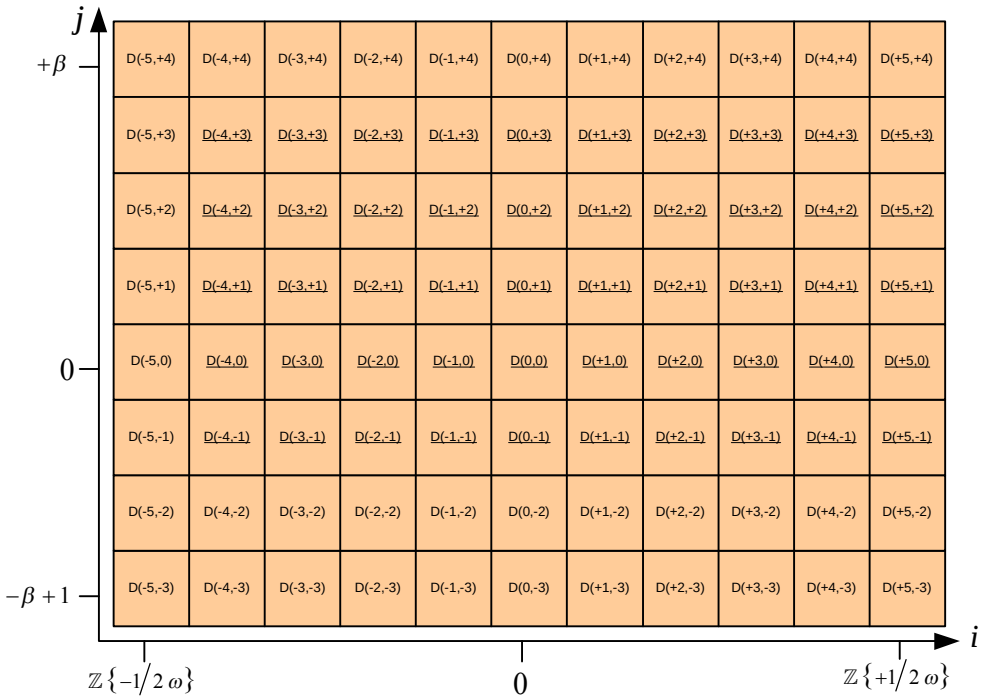


Figura 19 – Fase 2 do *pipeline* – Matriz resultante diferença $D(i, j)$

4.1.2.3 – Fase 3 – Processamento da função quadrática

A terceira fase do consiste em elevar ao quadrado cada uma das diferenças $D(i, j)$ calculadas na fase anterior. Esta fase é sequencialmente executada durante 8 ciclos de , sclk0 a sclk7.

O processamento utiliza concorrentemente, a cada ciclo de sclk 11 multiplicadores (dos 12 disponíveis no FPGA Xilinx XC3S200), multiplicando cada uma das diferenças por ela mesma. O resultado de cada uma destas multiplicações é armazenado nas posições de memória correspondentes $D^2(i, j)$, de 16-bit de largura.

Esta fase poderia ser condensada em um número menor de ciclos de caso houvesse um número maior de multiplicadores disponíveis (como, por exemplo, no modelo XC3S1500 da Xilinx que contém 32 multiplicadores disponíveis).

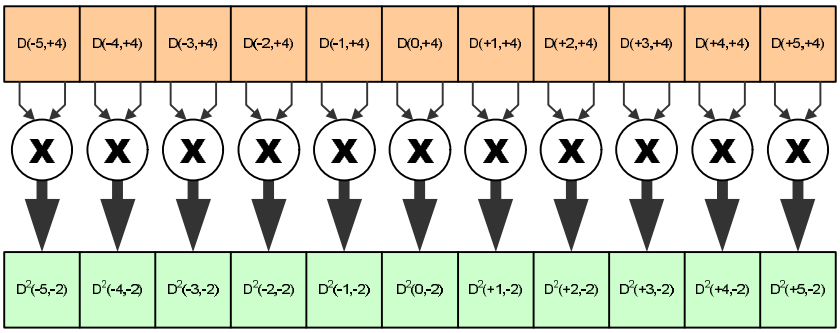


Figura 20 – Fase 3 do *pipeline* – Execução da função quadrática

Após término do sclk7 (oitavo ciclo de desta fase) estará disponível em memória o resultado de 88 operações de multiplicação, sendo alocado $88 \cdot 16 = 1408$ bits da memória do FPGA.

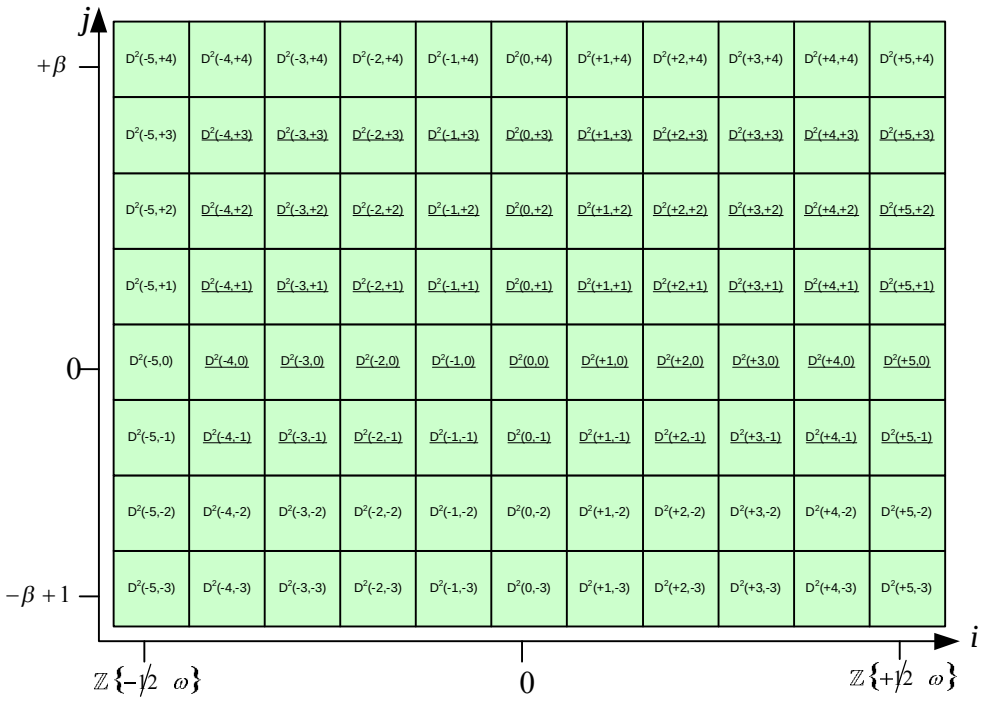


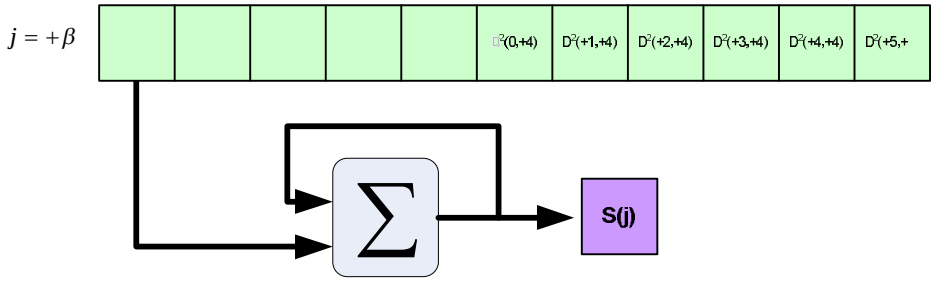
Figura 21 – Fase 3 do *pipeline* – Matriz resultante quadrática $D^2(i, j)$

4.1.2.4 – Fase 4 – Processamento da função somatório

A quarta fase do processo em consiste em efetuar a operação de somatório, necessária para avaliação da magnitude das janelas da fase 5.

São alocados 8 somadores, responsáveis por executar concorrentemente os processos de soma de cada uma das linhas da matriz $D^2(i, j)$.

O processo da fase 4 é efetuado sequencialmente, consumindo 11 ciclos de (sclk0 a sclk10), onde, a cada ciclo de são efetuadas 8 operações de soma dos valores $D^2(i, j)$, resultando, ao final, o vetor $S(j)$.


$j = +\beta$
D²(0,+4)
D²(+1,+4)
D²(+2,+4)
D²(+3,+4)
D²(+4,+4)
D²(+5,+
Σ
S(j)

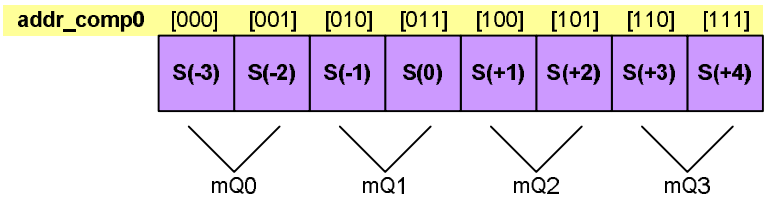


Figura 24 – Fase 5 do *pipeline* – Primeira comparação de magnitude

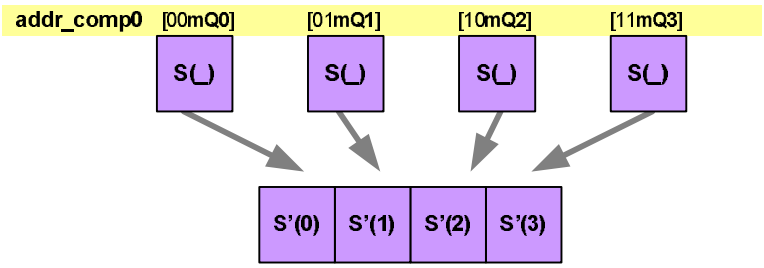


Figura 25 – Fase 5 do *pipeline* – Segunda comparação de magnitude

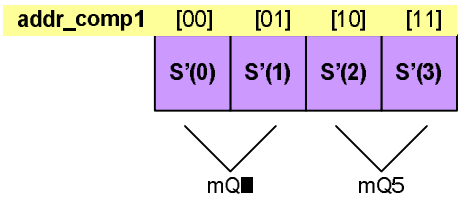


Figura 26 – Fase 5 do *pipeline* – Terceira comparação de magnitude

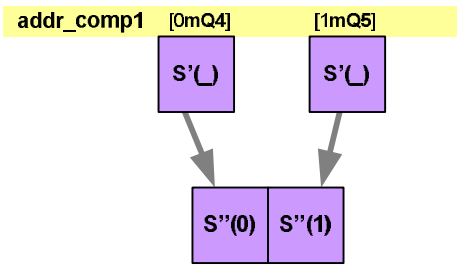


Figura 27 – Fase 5 do *pipeline* – Quarta comparação de magnitude

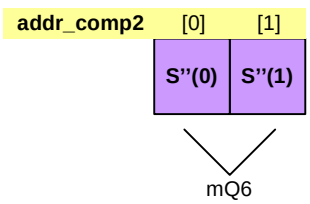


Figura 28 – Fase 5 do *pipeline* – Última comparação de magnitude

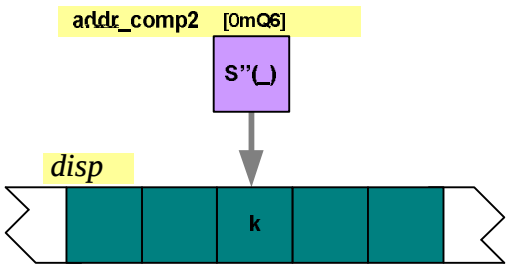


Figura 29 – Fase 5 do *pipeline* – Armazenamento da diferença mínima observada

4.1.3 – Simulações e resultados preliminares

Para avaliação do algoritmo proposto foi desenvolvido um programa de computador dedicado a emular funcionalmente o circuito digital a ser implementado no dispositivo FPGA.

O desenvolvimento deste programa dedicado surgiu das extremas limitações dos programas comerciais, como o Mentor Modelsim, em tratar informações complexas (quadros de vídeo, neste caso) e apresentar resultados graficamente representativos. Na maioria dos casos, o ciclo de simulação nestes incorre em processos complexos de tratamento dos dados de entrada e saída, adicionando complexidade desnecessária, especialmente durante as fases iniciais de avaliação de viabilidade de um algoritmo.

A estrutura funcional básica do programa de simulação é mostrada na Figura 30.

A janela do programa de simulação é apresentada na Figura 31. Seu desenvolvimento foi realizado utilizando a plataforma de desenvolvimento Delphi.

Apesar do trabalho inicial, decorrência natural do desenvolvimento de um programa específico, esta ferramenta acelerou consideravelmente a avaliação do algoritmo e permitiu visualização gráfica e em detalhes de cada um dos sub-processos empregados durante o processamento das imagens.

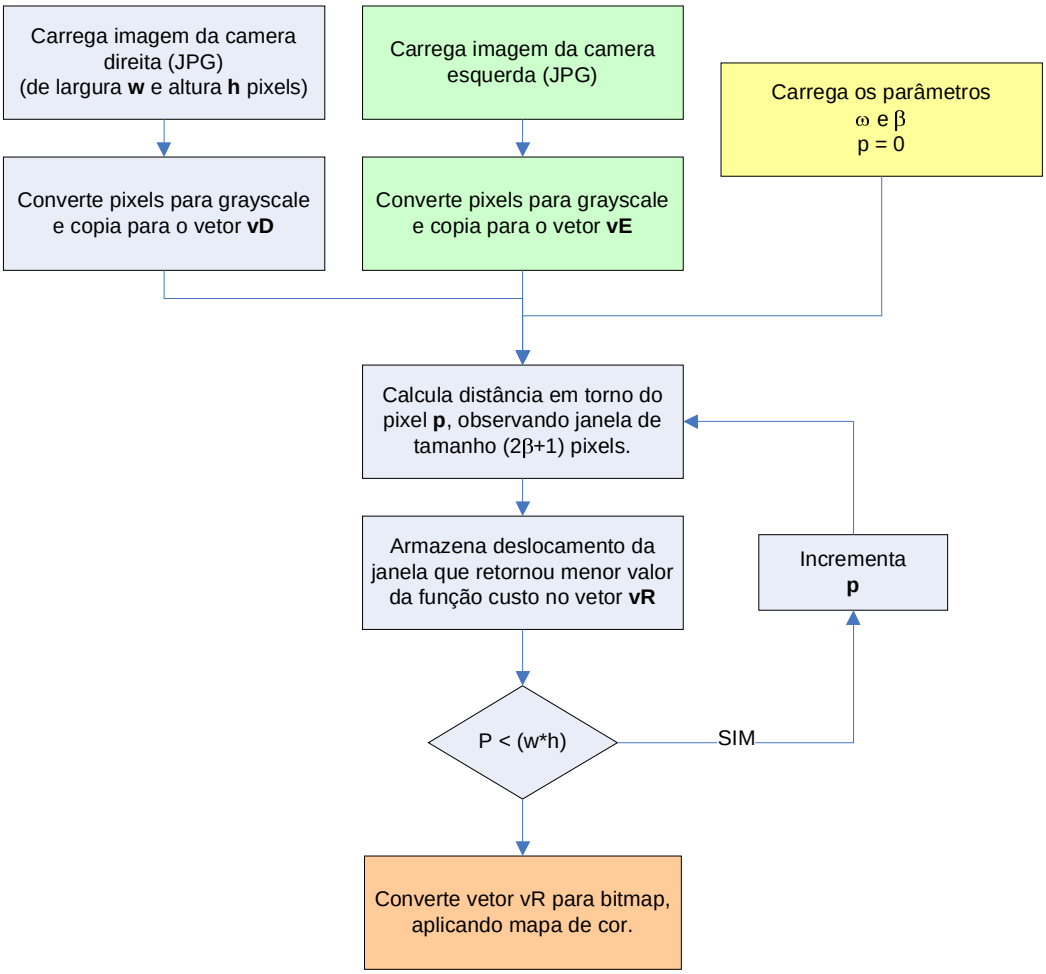


Figura 30 – Estrutura funcional básica do programa de simulação

Figura 31 - Janela do programa de simulação

4.2 Implementação

Durante a fase de estudos e modelamento do sistema proposto, foi desenvolvido uma série de sub-circuitos digitais aplicados ao FPGA Spartan-3, na placa de desenvolvimento .

Estes sub-circuitos possuem funções de e comunicação, sendo alguns deles diretamente necessários para implementação do algoritmo proposto, enquanto outros foram apenas utilizados durante a fase de depuração do código digital sintetizado.

Todos os códigos foram desenvolvidos utilizando linguagem de programação VHDL, garantindo portabilidade e simplificando a fase de depuração do circuito lógico.

4.2.1 – Kit de desenvolvimento

Para execução do projeto proposto foi escolhido o kit de desenvolvimento Spartan-3 Starter Board, fabricado pela empresa Digilent. Este kit integra um dispositivo FPGA Spartan-3 XC3S200, 2Mbyte de memória SRAM, interfaces JTAG, RS232C, VGA, PS2, além de micro-switches, displays de 7 segmentos e diversos leds auxiliares para uso durante o processo de depuração.

O modelo de FPGA XC3S200 disponível na placa de desenvolvimento conta com 200.000 blocos lógicos, 12 multiplicadores de 18-bits de largura, além de uma série de recursos para conexão e controle de periféricos externos. O kit também inclui uma versão da plataforma de desenvolvimento ISE 6, da própria Xilinx, empregada para o desenvolvimento do código VHDL deste projeto.

A Figura 32 exibe a montagem utilizada para avaliação do algoritmo, integrando duas câmeras CMOS à placa de desenvolvimento.

Um resumo das características técnicas do kit de desenvolvimento é mostrado na Figura 33.

Figura 32 – Hardware utilizado na pesquisa. Kit de desenvolvimento da Xilinx e duas câmeras CMOS modificadas.

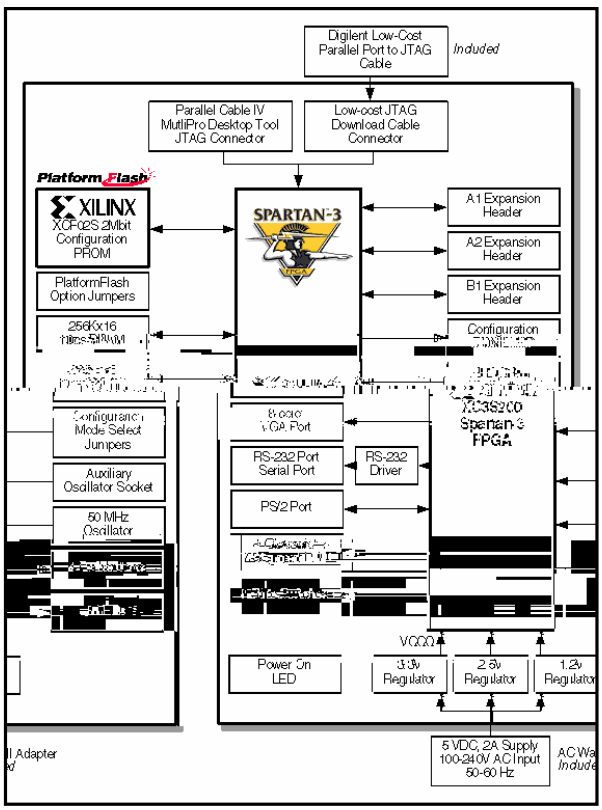


Figura 33 – Xilinx Spartan-3 XC3S200 Starter Kit

4.2.2 Câmeras CMOS

Para obtenção das imagens do ambiente foram empregadas duas câmeras CMOS. Os sensores de imagens CMOS (acrônimo de) utilizam uma matriz de , cada qual contendo um ou mais transistores, para quantização da imagem observada.

O sensor CMOS escolhido foi o OV7648, fabricado pela empresa Omnivision. Este sensor CMOS é empregado em diversas câmeras de baixo custo utilizadas em micro-computadores.

Por simplicidade de utilização e aplicação, foram adquiridas duas DSB-C310, da empresa D-Link, que empregam em seu interior o sensor CMOS escolhido.

As Figura 34 e Figura 35 exibem a câmera, assim como as modificações realizadas para acesso direto aos pinos de comunicação do sensor CMOS. Optou-se pela modificação de uma câmera comercial devido a dificuldade de obtenção de um kit de desenvolvimento contendo apenas o sensor CMOS. As modificações incluíram a remoção do circuito integrado OV519 (responsável pela interface USB e comunicação com o sensor CMOS OV7648) e adição de um conector externo, disponibilizando diretamente os pinos do sensor CMOS.

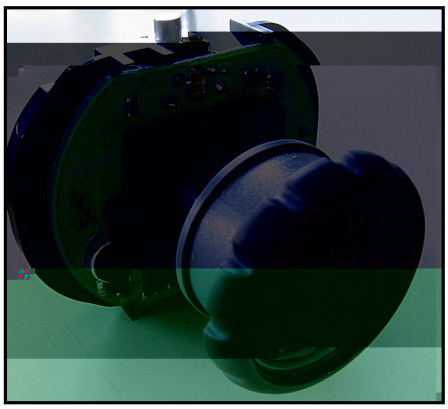


Figura 34 – Webcam D-Link DSB-C310

As modificações realizadas permitiram a conexão do sensor CMOS diretamente na placa de desenvolvimento utilizada no projeto, possibilitando a

aquisição em tempo real, sem compressão ou pré-processamento, das informações observadas pela câmera.

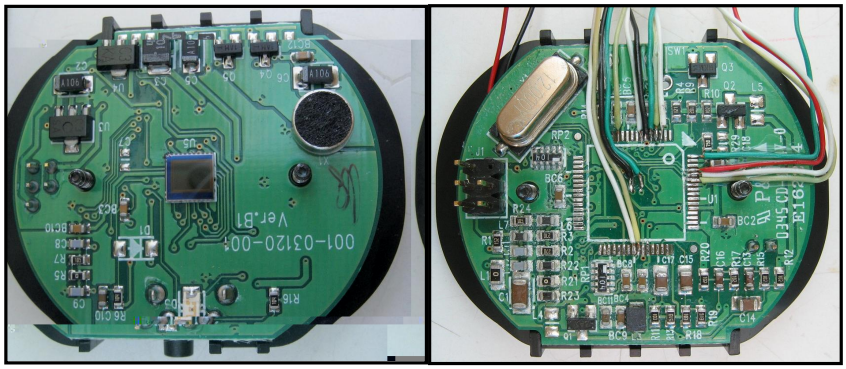


Figura 35 – Modificações para acesso aos sinais do sensor CMOS

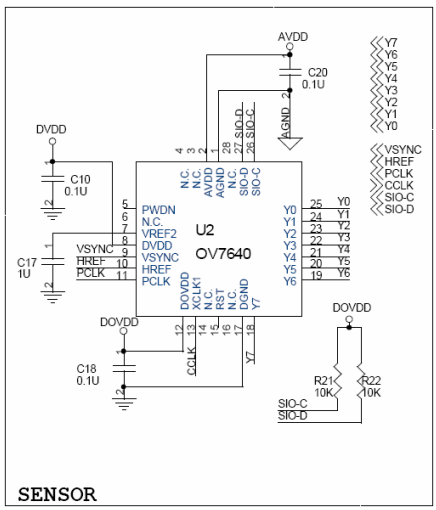


Figura 36 - Esquema elétrico de conexão com o sensor CMOS OV7648

O esquema elétrico da Figura 36 exibe os pinos de comunicação disponíveis no sensor CMOS utilizado. Os sinais VSYNC, HREF, PCLK e CCLK são utilizados para alinhamento dos quadros e para conexão dos sinais de necessários. O barramento serial SIO-C e SIO-D possibilita acesso a interface interna do sensor responsável pela programação de diversos parâmetros funcionais. O barramento paralelo Y[0..7] disponibiliza, sincronamente ao sinal PCLK, os dados componentes de cada um dos pixels do quadro.

4.2.3 Estrutura interna do FPGA

A Figura 37 exibe esquematicamente a estrutura interna implementada no dispositivo FPGA. Cada bloco representa um sub-dispositivo digital que foi desenvolvido e testado isoladamente.

Constituindo uma máquina síncrona, o esquema exibe uma fonte primária de de 50MHz (gerada externamente através de um oscilador), seguida por dois blocos sintetizadores de (respectivamente de 96MHz e 27MHz). Diversos sub-divisores foram empregados, disponibilizando todas as fontes de necessárias para movimentação dos processos e sub-processos da máquina síncrona.

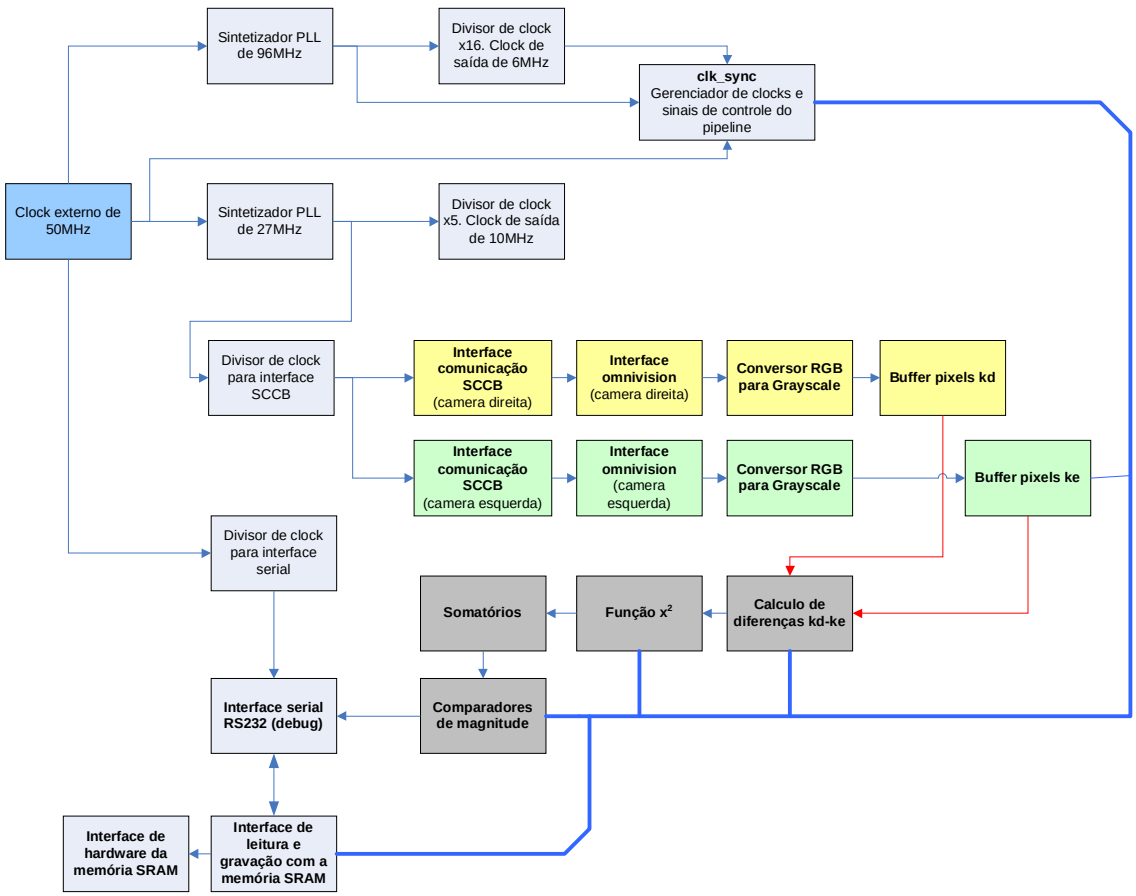


Figura 37 - Diagrama interno do algoritmo no FPGA

Representados por blocos amarelos e verdes, a Figura 37 exibe os sub-processos responsáveis pela inicialização, leitura e disponibilização dos da câmera direita e esquerda. Quando o sistema é executado pela primeira vez, o processo dispara o envio das configurações necessárias para os sensores CMOS (via interface serial SCCB). Terminado os processos de inicialização, estes sub-processos executam ciclicamente a leitura dos valores (RGB) que representam um , sua conversão para um único valor em tom de cinza, e armazenam finalmente este resultado nos buffers kd e ke.

Os blocos representados em cinza escuro identificam os sub-processos envolvidos diretamente no processamento do algoritmo. Os demais blocos representam as ferramentas utilizadas para armazenamento final e depuração.

A Figura 38 exibe o esquemático detalhado dos blocos e a representação utilizada pelo programa de desenvolvimento ISE durante projeto do FPGA.

4.2.3.1 Sintetizadores de clock

Para correto funcionamento, diversas fontes de precisaram ser empregadas. Entretanto, apenas uma fonte de 50MHz, gerada através de um oscilador externo, está disponível para uso na placa de desenvolvimento do FPGA Spartan-3.

Para obtenção dos demais sinais de empregaram-se sintetizadores de clock PLL, disponíveis internamente ao FPGA empregado.

Desta forma foi possível obter de 96MHz e 27Mhz, além dos demais sinais de clock submúltiplos, empregados em diversas máquinas de estado do .

Os sintetizadores de clock são disponibilizados na interface de desenvolvimento como blocos parametrizados não permitindo, portanto, observação do seu código fonte VHDL/Verilog.

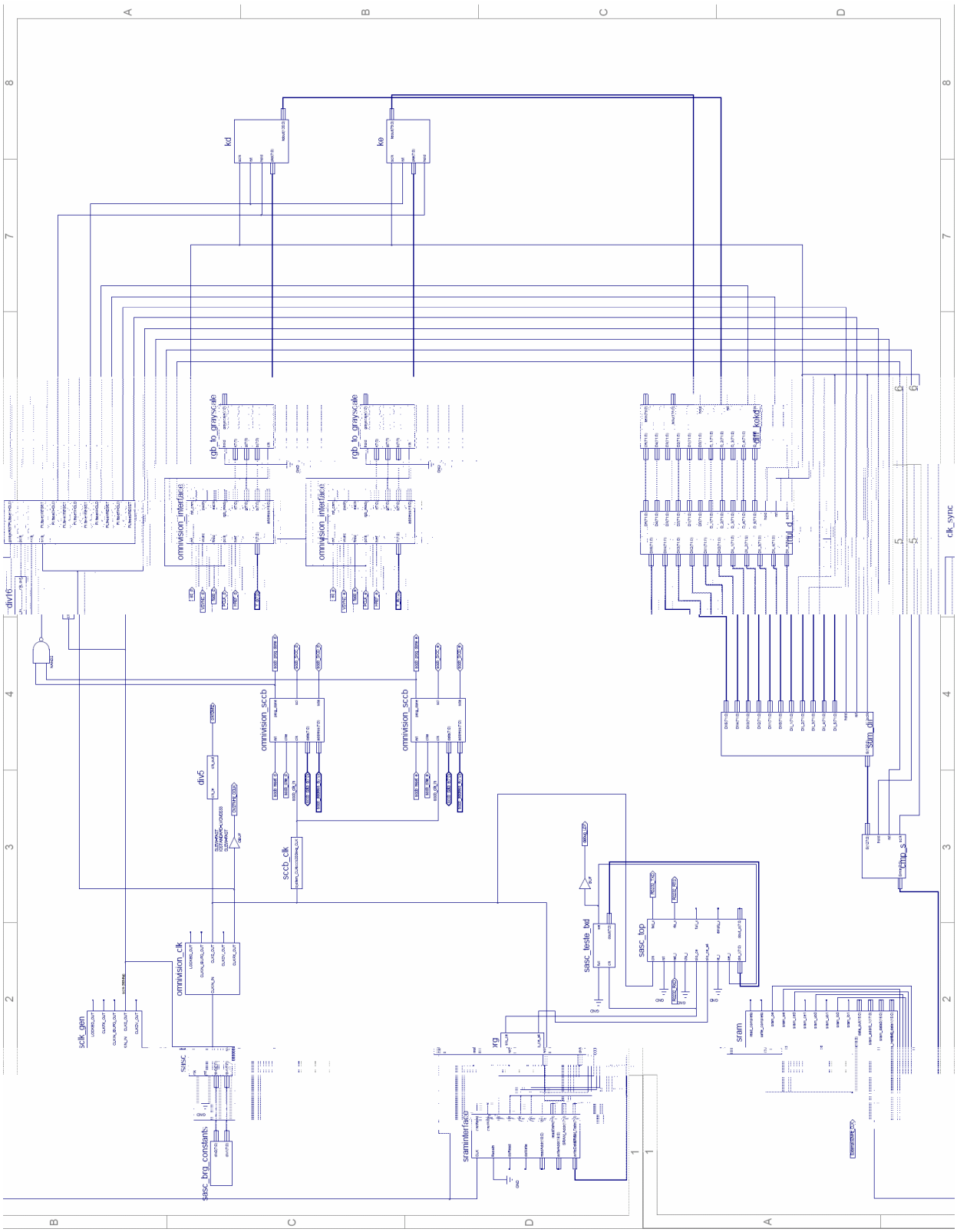


Figura 38 - main.sch - diagrama interno dos blocos constituintes do algoritmo no FPGA

4.2.3.2 Interface da câmera Omnivision

A câmera de vídeo utilizada emprega em seu núcleo um sensor CMOS da empresa Omnivision (OV7648). Este sensor disponibiliza duas de comunicação, uma paralela – utilizada para transferência do interno, e uma serial – utilizada para inicialização e ajuste das diversas opções disponíveis no sensor.

O sub-circuito apresentado na Figura 39 foi desenvolvido para conexão do sensor CMOS ao dispositivo FPGA, provendo todos os sinais de controle e barramento para recepção dos dados do sensor.

A interface serial VHDL contém anexada em seu interior uma seqüência padrão de comandos utilizada para inicialização do sensor, executados apenas fase de inicialização do sistema.

A interface paralela é agregada a circuitos lógicos e máquinas de estado adicionais para transferência dos dados recebidos para as células de memórias descritas no modelamento da seção 4.1 Parametrização e modelamento.

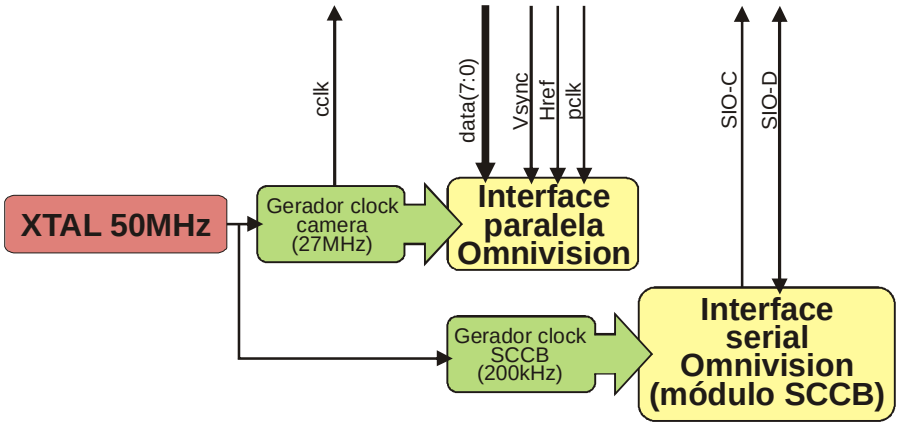


Figura 39 – Estrutura interna módulo de interface com o sensor CMOS Omnivision

O módulo de comunicação é composto dos arquivos fontes omniVision_CLK (gerado através do Xilinx Architecture Wizard), omnivision_interface (VHDL), omnivision_sccb (VHDL) e sccb_clk (VHDL).

A Tabela 1,

Tabela 2 e Tabela 3 exibem os resultados da síntese dos códigos fonte VHDL utilizados nestes módulos.

Tabela 1 – Resultado da síntese do módulo VHDL omnivision_interface.vhd para o FPGA Xilinx 3s200pq208-4

Resultados de alocação	
Número de SLICEs utilizados	32 de 1920 (1%)
Número de SLICEs flip-flops utilizados	53 de 3840 (1%)
Número de IOBs de interface externa utilizados	49 de 141 (34%)
Número de LUTs de 4 entradas utilizadas	27 de 3840 (0%)
Number of GCLKs utilizados	1 de 8 (12%)
Quantidade de Macro dispositivos funcionais sintetizados	
1	17-bit adder
3	1-bit register
2	17-bit register
3	8-bit register
Características temporais relevantes	
Freqüência máxima do clock global pclk	180,050MHz
Máximo atraso observável de uma saída após sinal do clock	5,835ns

Tabela 2 – Resultado da síntese do módulo VHDL omnivision_sccb.vhd para o FPGA Xilinx 3s200pq208-4

Resultados de alocação	
Número de SLICEs utilizados	43 de 1920 (2%)
Número de SLICEs flip-flops utilizados	57 de 3840 (1%)
Número de IOBs de interface externa utilizados	19 de 141 (13%)
Número de LUTs de 4 entradas utilizadas	73 de 3840 (1%)
Number of GCLKs utilizados	3 de 8 (37%)
Quantidade de Macro dispositivos funcionais sintetizados	
2	8-bit latch
1	34-bit register
7	1-bit register
Características temporais relevantes	
Freqüência máxima do clock global pclk	140,607MHz
Máximo atraso observável de uma saída após sinal do clock	6,283ns

Tabela 3 – Resultado da síntese do módulo VHDL sccb_clk.vhd para o FPGA Xilinx 3s200pq208-4

Resultados de alocação	
Número de SLICEs utilizados	12 de 1920 (0%)
Número de SLICEs flip-flops utilizados	9 de 3840 (0%)
Número de IOBs de interface externa utilizados	1 de 141 (0%)
Número de LUTs de 4 entradas utilizadas	21 de 3840 (0%)
Number of GCLKs utilizados	1 de 8 (12%)
Quantidade de Macro dispositivos funcionais sintetizados	
1	8-bit adder
1	1-bit register
1	8-bit register
Características temporais relevantes	
Freqüência máxima do clock global pclk	141,884MHz
Máximo atraso observável de uma saída após sinal do clock	6,060ns

4.2.3.4 Conversor RGB para nível de cinza

A interface paralela das câmeras disponibiliza três bytes de informação para cada novo pixel amostrado, representando respectivamente as cores vermelho, verde e azul.

O algoritmo proposto se baseia no processamento de imagens em tonalidades de cinza, requerendo portanto a conversão de modo RGB para níveis de cinza.

O algoritmo implementado para esse fim utiliza a curva padrão indicada pela equação (8), onde Y representa o valor resultante em níveis de cinza e R, G e B representam as componentes de cor do pixel analisado.

$$Y = 0,3 \cdot R + 0,59 \cdot G + 0,11 \cdot B \quad (8)$$

Visando simplificar a implementação do módulo, os valores da curva padrão foram aproximados por razões com coeficientes 2^n, como indicado na equação (9).

$$Y = \frac{32 \cdot R}{128} + \frac{64 \cdot G}{128} + \frac{8 \cdot B}{128} = 0,25 \cdot R + 0,5 \cdot G + 0,062 \cdot B \quad (9)$$

A Tabela 4 exibe o resultado da síntese do código fonte VHDL utilizado para implementação do conversor descrito.

Tabela 4 – Resultado da síntese do módulo VHDL rgb_to_grayscale.vhd para o FPGA Xilinx 3s200pq208-4

Resultados de alocação	
Número de SLICEs utilizados	27 de 1920 (1%)
Número de SLICEs flip-flops utilizados	40 de 3840 (1%)
Número de IOBs de interface externa utilizados	0 de 141 (0%)
Número de LUTs de 4 entradas utilizadas	16 de 3840 (0%)
Number of GCLKs utilizados	1 de 8 (12%)

Quantidade de Macro dispositivos funcionais sintetizados	
2	32-bit adder
1	32-bit up accumulator
1	8-bit register
1	32-bit register
1	32-bit comparator less
1	32-bit comparator greater
1	32-bit 2-to-1 multiplexer

Características temporais relevantes	
Freqüência máxima do clock global clk	229,358MHz
Máximo atraso observável de uma saída após sinal do clock	5,835ns

4.2.3.5 Buffers das câmeras

Os de entrada da câmera são formados por registradores de deslocamento FIFO, responsáveis por armazenar um novo pixel (1 byte) e por

descartar o pixel mais antigo a cada ciclo do clock . O processo ocorre em dois estágios síncronos, controlado por uma máquina de estados e o clock . O conteúdo dos registrados é disponibilizado paralelamente para o estágio seguinte, permitindo processamento concorrente de todos os dados dos buffers.

Os módulos VHDL kd.vhd e ke.vdh implementam as funcionalidades descritas. A Tabela 5 e Tabela 6 exibem os resultados da síntese dos códigos fonte VHDL utilizados.

Tabela 5 – Resultado da síntese do módulo VHDL kd.vhd para o FPGA Xilinx 3s200pq208-4

Resultados de alocação	
Número de SLICEs utilizados	157 de 1920 (8%)
Número de SLICEs flip-flops utilizados	273 de 3840 (7%)
Número de LUTs de 4 entradas utilizadas	17 de 3840 (0%)
Number of GCLKs utilizados	1 de 8 (12%)
Quantidade de Macro dispositivos funcionais sintetizados	
273	1-bit register
Características temporais relevantes	
Freqüência máxima do clock global clk	340,948MHz
Máximo atraso observável de uma saída após sinal do clock	6,060ns

Tabela 6 – Resultado da síntese do módulo VHDL ke.vhd para o FPGA Xilinx 3s200pq208-4

Resultados de alocação	
Número de SLICEs utilizados	93 de 1920 (4%)
Número de SLICEs flip-flops utilizados	161 de 3840 (4%)
Número de LUTs de 4 entradas utilizadas	11 de 3840 (0%)
Number of GCLKs utilizados	1 de 8 (12%)
Quantidade de Macro dispositivos funcionais sintetizados	
161	1-bit register
Características temporais relevantes	
Freqüência máxima do clock global clk	363,636MHz
Máximo atraso observável de uma saída após sinal do clock	6,060ns

4.2.3.6 Módulo de cálculo de diferenças

O módulo de cálculo de diferenças, implementado pelo código VHDL diff_kdke.vhd, recebe paralelamente 216 bits armazenados nos buffer kd e ke, e realiza concorrentemente todos os cálculos de diferença para as 2β janelas. Os resultados são disponibilizados para o módulo seguinte por 11 barramentos de 72 bits.

A Tabela 7 exibe os resultados da síntese do código fonte VHDL utilizado.

Tabela 7 – Resultado da síntese do módulo VHDL diff_kdke.vhd para o FPGA Xilinx 3s200pq208-4

Resultados de alocação	
Número de SLICEs utilizados	324 de 1920 (16%)
Número de LUTs de 4 entradas utilizadas	648 de 3840 (16%)
Quantidade de Macro dispositivos funcionais sintetizados	
72	9-bit subtractor
Características temporais relevantes	
Máximo atraso observável (considerando atraso do circuito combinacional)	14,257ns

4.2.3.7 Módulo de execução da função quadrática

O módulo VHDL de execução de função quadrática emprega os multiplicadores disponíveis em hardware para efetuar dezenas de cálculos, utilizando como dados de entrada as saídas do módulo de cálculo de diferenças.

Devido à limitação do número de multiplicadores disponíveis em hardware, o módulo foi estruturado em uma máquina de estado seqüencial (7 passos), onde a cada passo emprega 11 multiplicadores para processar uma fração dos dados disponibilizados pelo módulo anterior. O módulo resulta em 88 valores de saída, que são processados posteriormente pelo módulo da função somatório.

A Tabela 8 exibe os resultados da síntese do código fonte VHDL utilizado.

Tabela 8 – Resultado da síntese do módulo VHDL mul_d.vhd para o FPGA Xilinx 3s200pq208-4

Resultados de alocação	
Número de SLICEs utilizados	296 de 1920 (15%)
Número de SLICEs flip-flops utilizados	512 de 3840 (13%)
Número de IOBs de interface externa utilizados	0 de 141 (0%)
Número de LUTs de 4 entradas utilizadas	512 de 3840 (13%)
Number of GCLKs utilizados	1 de 8 (12%)
Quantidade de Macro dispositivos funcionais sintetizados	
1	32-bit adder
1	32-bit up accumulator
2	8-bit register
1	32-bit register
11	32-bit multiplier
1	32-bit 2-to-1 multiplexer
Características temporais relevantes	
Freqüência máxima do clock global clk	227,422MHz
Máximo atraso observável de uma saída após sinal do clock	5,115ns

4.2.3.8 Módulo de execução do somatório

Responsável pelo somatório dos diversos resultados disponibilizados pelo módulo de execução da função quadrática, o módulo do somatório foi estruturado como uma máquina de estado de 11 passos, onde cada passo é responsável por processar o somatório de 8 operações concorrentes de soma. Esta estrutura foi escolhida pela significativa redução da quantidade de dispositivos somatórios necessários para execução concorrente de soma de todos os dados.

A Tabela 9 exibe os resultados da síntese do código fonte VHDL utilizado.

Tabela 9 – Resultado da síntese do módulo VHDL sum_dii.vhd para o FPGA Xilinx 3s200pq208-4

Resultados de alocação	
Número de SLICEs utilizados	439 de 1920 (22%)
Número de SLICEs flip-flops utilizados	92 de 3840 (2%)
Número de IOBs de interface externa utilizados	0 de 141 (0%)
Número de LUTs de 4 entradas utilizadas	778 de 3840 (20%)
Number of GCLKs utilizados	1 de 8 (12%)

Quantidade de Macro dispositivos funcionais sintetizados	
8	16-bit adder
11	16-bit 2-to-1 multiplexer
83	1-bit register

Características temporais relevantes	
Freqüência máxima do clock global clk	202,388MHz
Máximo atraso observável de uma saída após sinal do clock	6,172ns

4.2.3.9 Comparadores de magnitude

Responsáveis pela detecção do mínimo peso dos resultados da função custo, isto é, detecção da janela que apresentou melhor correspondência entre os quadros.

Diversos comparadores de magnitude foram utilizados no módulo VHDL cmp_S.vhd. O módulo é estruturado em uma máquina de estados de 6 passos, onde cada passo compara dois a dois os resultados disponibilizados pelo módulo do somatório, encontrado sucessivamente o índice da janela que apresentou menor resultado da função custo.

O resultado disponibilizado a cada novo pixel processado representada diretamente a profundidade relativa observada e é armazenada sucessivamente para construção do denso mapa de disparidade.

A Tabela 10 exibe os resultados da síntese do código fonte VHDL utilizado.

Tabela 10 – Resultado da síntese do módulo VHDL cmp_S.vhd para o FPGA Xilinx 3s200pq208-4

Resultados de alocação	
Número de SLICEs utilizados	22 de 1920 (1%)
Número de SLICEs flip-flops utilizados	16 de 3840 (0%)
Número de IOBs de interface externa utilizados	0 de 141 (0%)
Número de LUTs de 4 entradas utilizadas	40 de 3840 (24%)
Number of GCLKs utilizados	1 de 8 (12%)
Quantidade de Macro dispositivos funcionais sintetizados	
1	16-bit comparator greater
1	32-bit register
1	8-bit register
6	1-bit register
Características temporais relevantes	

252 0.48 0.48 re f50.88

Internamente dois dutos paralelos são disponibilizados entrada e saída de dados, além de três sinais de controle, responsáveis pela sincronização e sinalização de eventos.

Este sub-circuito é especialmente útil no monitoramente do conteúdo da memória externa SRAM, permitindo a observação do seu conteúdo durante o funcionamento do algoritmo.

A Tabela 11 e Tabela 12 exibem os resultados da síntese dos códigos fonte VHDL utilizados nestes módulos.

Tabela 11 – Resultado da síntese do módulo Verilog sasc_brg.v para o FPGA Xilinx 3s200pq208-4

Resultados de alocação	
Número de SLICEs utilizados	21 de 1920 (1%)
Número de SLICEs flip-flops utilizados	25 de 3840 (0%)
Número de IOBs de interface externa utilizados	19 de 141 (10%)
Número de LUTs de 4 entradas utilizadas	32 de 3840 (0%)
Number of GCLKs utilizados	1 de 8 (12%)
Quantidade de Macro dispositivos funcionais sintetizados	
2	8-bit up counter
1	2-bit up counter
7	1-bit register
2	8-bit comparator equal
Características temporais relevantes	
Freqüência máxima do clock global clk	261,712MHz
Máximo atraso observável de uma saída após sinal do clock	5,835ns

Tabela 12 – Resultado da síntese do módulo Verilog sasc_top.v para o FPGA Xilinx 3s200pq208-4

Resultados de alocação	
Número de SLICEs utilizados	46 de 1920 (2%)
Número de SLICEs flip-flops utilizados	59 de 3840 (1%)
Número de IOBs de interface externa utilizados	27 de 173 (15%)
Número de LUTs de 4 entradas utilizadas	71 de 3840 (1%)
Number of GCLKs utilizados	1 de 8 (12%)

Tabela 12 (continuação)

Quantidade de Macro dispositivos funcionais sintetizados	
2	4x8-bit dual-port distributed RAM
4	2-bit adder
2	4-bit up counter
1	10-bit register
29	1-bit register
4	2-bit register
4	2-bit comparator equal
1	10-bit 2-to-1 multiplexer
1	1-bit xor2
Características temporais relevantes	
Freqüência máxima do clock global clk	231,803MHz
Máximo atraso observável de uma saída após sinal do clock	8,208ns

4.2.3.11 – Módulo de *interface* com os bancos de memória SRam

A placa de desenvolvimento Spartan-3 Starter Kit disponibiliza 2Mbits de memória externa através de dois bancos de memória SRam 256kx16 endereçados diretamente ao FPGA Spartan-3.

Devido a ausência de circuitos digitais gratuitos para acesso a estes módulos SRam, desenvolveu-se um sub-circuito digital síncrono capaz de ler e gravar dados aos bancos de memória.

O módulo FIFO disponibiliza dois dutos paralelos de 16-bits para entrada e saída de dados respectivamente, além de um duto de 18-bits para endereçamento contínuo de toda memória SRam disponível.

Uma máquina de estados permite a utilização sincronamente da memória SRam, simplificando a integração com o processo de proposto.

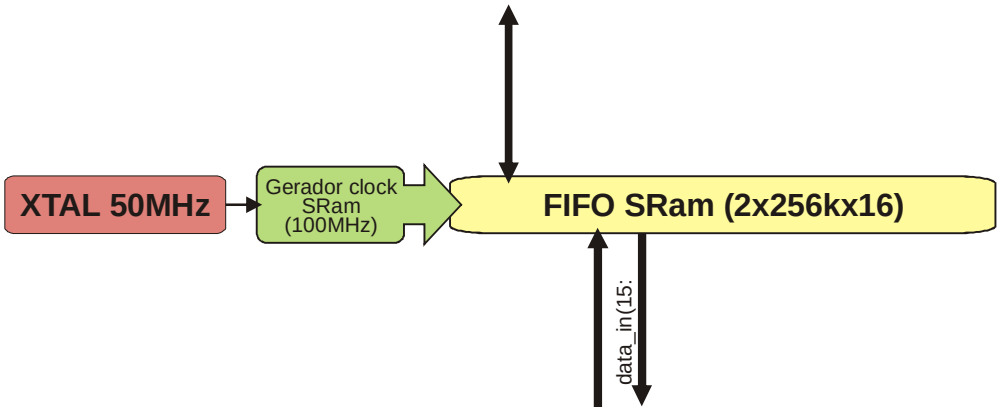

XTAL 50MHz
Gerador clock SRam (100MHz)
FIFO SRam (2x256kx16)
data_in(15:

Tabela 14 – Resultado da síntese do módulo VHDL sram_interface.vhd para o FPGA Xilinx 3s200pq208-4

Resultados de alocação	
Número de SLICEs utilizados	25 de 1920 (1%)
Número de SLICEs flip-flops utilizados	42 de 3840 (1%)
Número de IOBs de interface externa utilizados	90 de 173 (52%)
Número de LUTs de 4 entradas utilizadas	32 de 3840 (0%)
Number of GCLKs utilizados	1 de 8 (12%)
Quantidade de Macro dispositivos funcionais sintetizados	
5	1-bit register
1	18-bit register
1	8-bit register
3	2-to-1 multiplexer
1	8-bit tristate buffer
Características temporais relevantes	
Freqüência máxima do clock global clk	254,130MHz
Máximo atraso observável de uma saída após sinal do clock	7,883ns

4.2.3.12 – Módulo de *interface* Câmera-SRam

Este sub-circuito digital foi desenvolvido exclusivamente para fins de depuração, permitindo que os módulos de do sensor CMOS e de da memória SRam fossem interconectados, alocando duas matrizes na memória SRam com os conteúdos dos dos sensores CMOS.

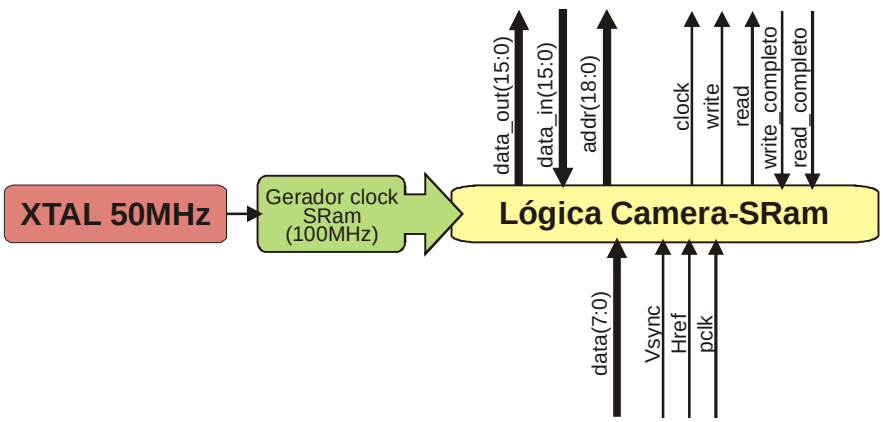


Figura 42 – Estrutura interna da interface sensor CMOS com a memória externa SRam

4.2.3.13 Módulo de *interface* UART-SRam

Este sub-circuito digital foi desenvolvido para monitoramento do conteúdo da memória SRam, permitindo assim depuração dos resultados dos outros módulos.

A gera os sinais necessários para os módulos UART e SRam, sincronizando o processo de envio para serial e leitura dos dados da SRam.

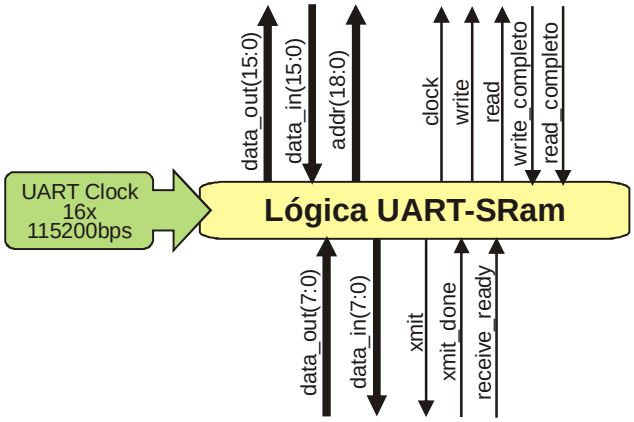


Figura 43 – Estrutura interna da interface de comunicação entre a memória externa SRam e o módulo de comunicação serial

4.2.3.14 Montagem do pipeline e módulos acessórios

A fase final de implementação do sistema consistiu em agrupar todos os módulos previamente desenvolvidos e avaliados em uma grande estrutura , capaz de processar as informações das câmeras em tempo real.

A Tabela 15 mostra a alocação de recursos do FPGA utilizado, sem interconexão dos módulos.

Durante a integração dos módulos, em decorrência da grande quantidade de interconexões necessárias, o sintetizador VHDL não conseguiu alocar toda a montagem dentro do FPGA disponível devido a falta de IOBs de interconexão interna. Esse problema poderia ser solucionado pela adoção de um FPGA de maior capacidade ou pela reestruturação dos processos lógicos e conexões.

Tabela 15 – Resultado da síntese parcial do sistema para o FPGA Xilinx 3s200pq208-4

Resultados de alocação	
Número de SLICEs utilizados	1629 de 1920 (85%)
Número de SLICEs flip-flops utilizados	1475 de 3840 (38%)
Número de IOBs de interface externa utilizados	122 de 141 (87%)
Número de LUTs de 4 entradas utilizadas	2412 de 3840 (63%)
Number of GCLKs utilizados	3 de 8 (38%)

Desta forma não foi possível avaliar o funcionamento contínuo do processo, sendo possível apenas a verificação, módulo a módulo das saídas esperadas.

Para levantamento dos resultados apresentados, diversas montagem parciais foram elaboradas, incluindo a interface RS232, interface de leitura e gravação da memória externa SRAM, o módulo com o sub-processo de interesse e um pequeno módulo de controle.

Dados previamente tratados no PC eram transferidos para memória SRAM e um sinal de controle gerado para execução do sub-processo. Os resultados eram então finalmente retornados ao PC para análise.

5 RESULTADOS FINAIS

Para verificação do algoritmo proposto utilizou-se o simulador e posterior aquisição dos dados dos sub-processos implementados no FPGA.

Para análise foram utilizadas pares de fotos do repositório JISCT, cuja base de dados é comumente utilizada para execução de testes e comparação de algoritmos.

Os pares de imagens foram pré-processados, sendo o tamanho reduzido em ¼ (por simples média 4:1) e um filtro gaussiano aplicado para redução dos componentes de alta freqüência das imagens (reduzindo consideravelmente o ruído contido nas imagens).

A Tabela 16 exibe o resultado do processamento de diversos pares de imagens. Alguns dos pares escolhidos foram utilizados nos trabalhos de BIRCHFIELD e TOMASI (1998), permitindo uma comparação visual de resultados.

Tabela 16 – (a,b) Pares de imagens do repositório JISCT. (c) Mapa de disparidade resultante.

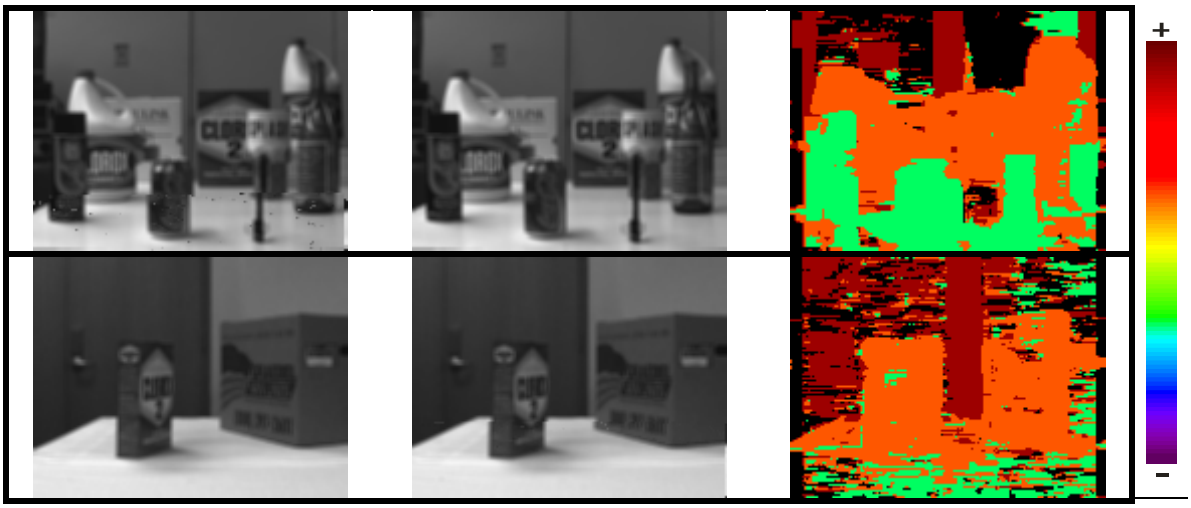
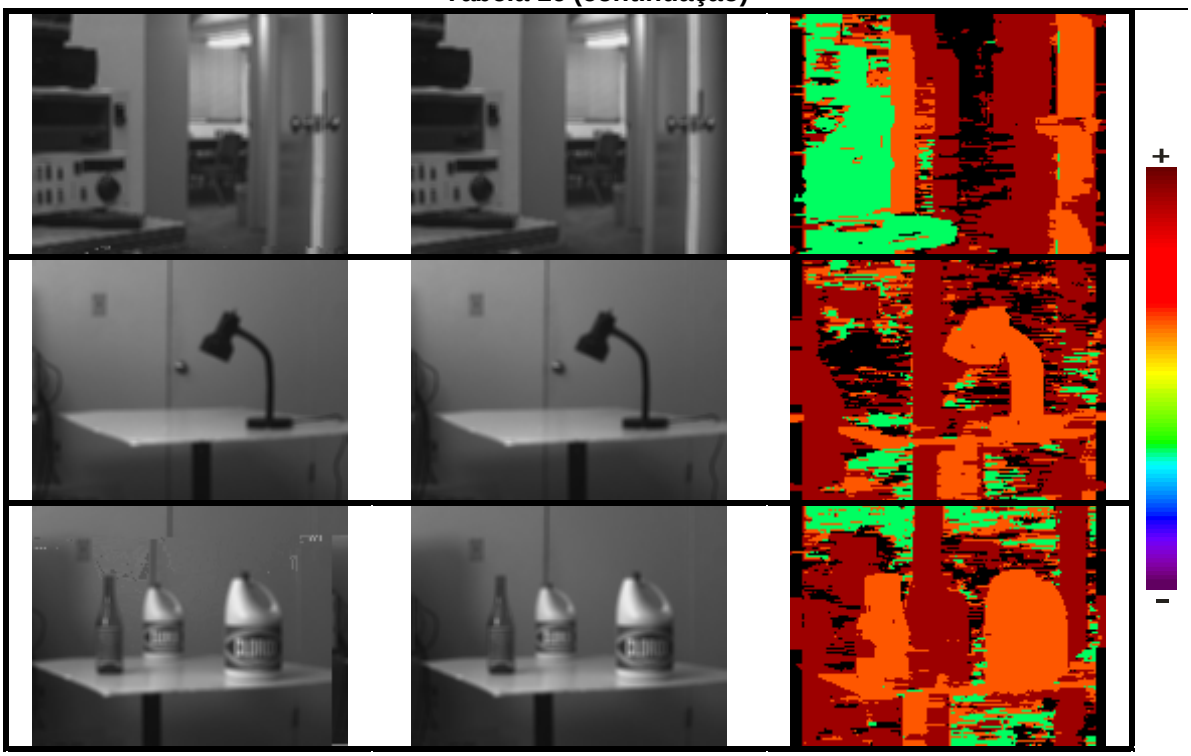


Tabela 16 (continuação)



(a) (b) (c)

Também foram avaliadas pares de imagem da universidade de Tsukuba. Diferentes valores dos coeficientes ω e β foram empregados devido às diferenças do FOV das câmeras. Parâmetro $\omega = 4$ e $\beta = 30$ foram utilizados para processamentos dos pares de imagens. Os resultados são apresentados na Tabela 17 – (a,b) Pares de imagens do repositório Tsukuba. (c) Mapa de disparidade resultante.

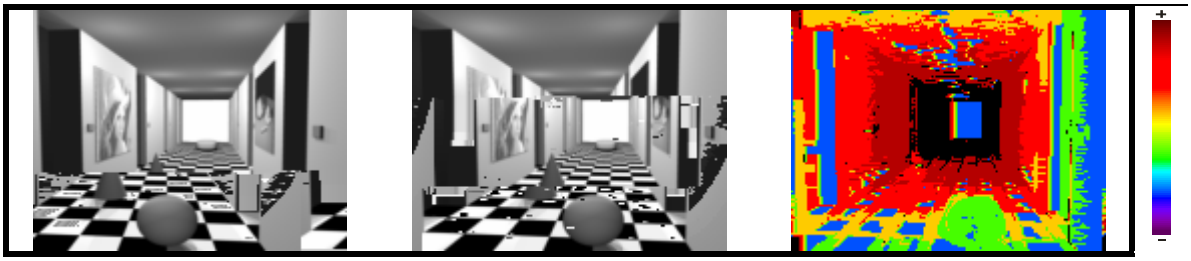
Tabela 17 – (a,b) Pares de imagens do repositório Tsukuba. (c) Mapa de disparidade resultante.

(a) (b) (c)

Pares de imagens sintéticas (geradas totalmente por algoritmos de), disponibilizadas pelo grupo de visão computacional da Universidade de Bonn, também foram analisadas.

Estas imagens sintéticas, geradas por um software de (MRTStereo –), apresentam regiões bastante definidas para avaliação do mapa de disparidade e efeito das oclusões no algoritmo. Os resultados são apresentados na Tabela 18. Parâmetros $\omega = 5$ e $\beta = 7$ foram empregados para processamento das imagens.

Tabela 18 – (a,b) Pares de imagens sintéticas do repositório Bonn. (c) Mapa de disparidade resultante



(a) (b) (c)

PÁGINA INTENCIONAMENTE DEIXADA EM BRANCO

6 CONCLUSÕES E TRABALHOS FUTUROS

Os resultados apresentados indicaram a viabilidade de implementação de um algoritmo de alto desempenho para visão computacional, mantendo simplicidade funcional e baixo-custo.

Apesar de apresentar limitações conhecidas (como a incapacidade de lidar com oclusões), o sistema se mostra adequado para obtenção dinâmica de referência de profundidade em dispositivos robóticos móveis. Em especial, os densos mapas de disparidade disponibilizados em tempo-real pelnj0.048 Tc (p) Tj0te

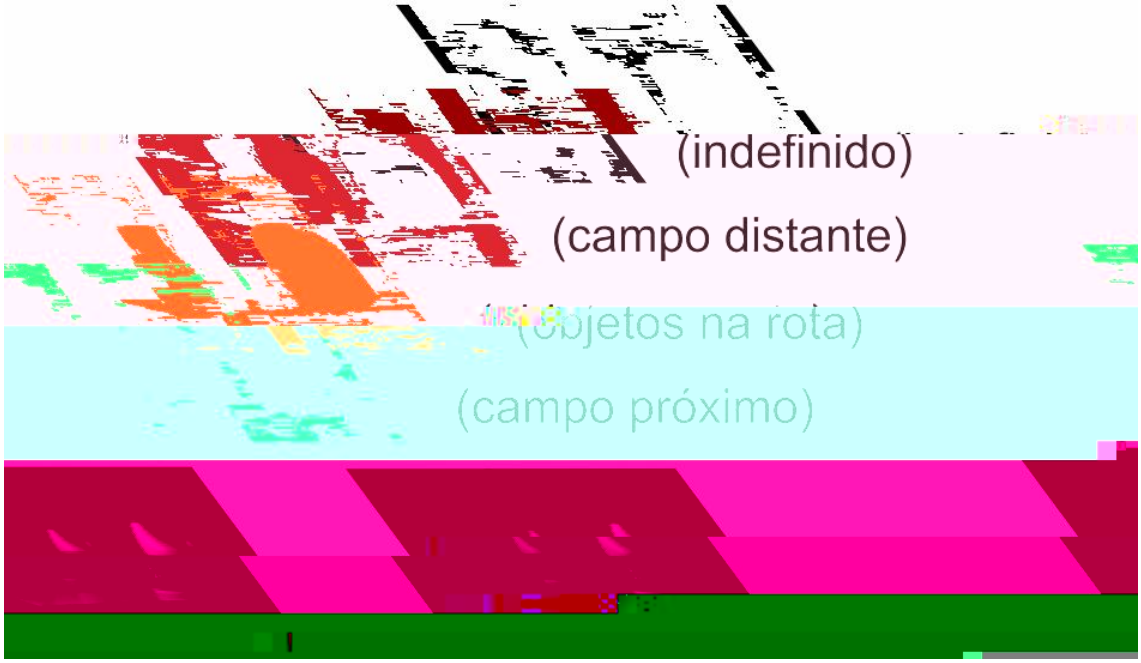


Figura 44 - Exemplo de aplicação do mapa de disparidade

Devido a dificuldades de ordem técnica, não foi possível obter completa integração de cada um dos sub-processos no FPGA, o que resultaria em um sistema funcional para utilização da câmera. Desta forma as verificações funcionais em consistiram em processar manualmente blocos de dados pré-definidos, e obtenção dos resultados pela interface RS232 implementada no sistema.

Os maiores problemas encontrados durante a fase de implementação no dispositivo FPGA foram decorrentes da falta de suporte e documentação pelo fabricante da placa de desenvolvimento e do dispositivo FPGA, assim como as limitações impostas pelo programa de simulação. Em especial não havia disponíveis gratuitamente (ou para avaliação) códigos funcionais básicos, como acesso à memória externa SRAM ou comunicação serial. Essas peças fundamentais para o desenvolvimento foram projetadas manualmente, ocasionando considerável atraso no cronograma do projeto.

A integração de todos os módulos também apresentou considerável dificuldade. Cada módulo separadamente utiliza uma pequena quantidade dos recursos disponíveis no FPGA empregado. Entretanto, durante a fase de integração, em decorrência da grande quantidade de interconexões

necessárias inter-módulos, o sintetizador VHDL não conseguiu alocar toda a montagem dentro do FPGA disponível. Revisão futura estará sendo feita com intuito de minimizar os processos lógicos e interconexões necessárias para execução do algoritmo no dispositivo.

Os trabalhos aqui apresentados resultaram em duas publicações. A primeira, disponível no Anexo A – Artigo Periódico LAAR, foi aceito para publicação no Periódico .

A segunda publicação, disponível no Anexo B – Artigo SPL2006 foi submetida e aceita na conferencia , realizada na cidade de Mar Del Plata, Argentina, em janeiro de 2006.

Trabalhos futuros prevêem solução dos diversos problemas de ordem técnica, possivelmente pela troca das ferramentas de desenvolvimento e do dispositivo FPGA, além da adição de um padrão que permita fácil conexão de filtros de pré e pós-processamento ao núcleo funcional do algoritmo proposto.

PÁGINA INTENCIONAMENTE DEIXADA EM BRANCO

7 REFERÊNCIAS

BELHUMER, P. N. (1993).
.
Proceedings of the Fourth International Conference on Computer Vision, pages 431-438, May 1993.

BIRCHFIELD, S.; TOMASI, C. (1998);
Proceedings of the 1998 IEEE International Conference on Computer Vision, Bombay, India.

_____. (1998);
IEEE Transactions on Pattern Analysis And Machine Intelligence, Vol. 20, No. 4, Apr. 1998.

BONATO, VANDERLEI ; FERNANDES, MÁRCIO MERINO ; MARQUES, EDUARDO .
. International Journal of Electronics, Londres, v. 93, p. 385-401, 2006.

COX, I. J.; HINGORANI, S. L.; RAO, S. B.; MAGGS, B. M. (1996).
. Computer Vision and Image Understanding, vol. 63, pages 542-567, May 1996.

DARABIHA, A., ROSE, J., MACLEAN, W. J. (2003).
. Proc. of the 2003 IEEE Computer Society Conf. on Computer Vision and Pattern Recognition, vol. 1, pages 203-210, June 2003.

GROSSO, E.; TISTARELLI, M. (1995). . IEEE Trans. On Pattern Analysis and Machine Intelligence, vol. 17, no. 9, pages 868-879, Sept. 1995.

HILE, H.; ZENG, C. (1996). . University of Washington.

MURRAY, D.; JENNINGS, C. (1997).
. Proceedings of the 1997 IEEE International Conference on Robotics and Automation, pages 1694-1699, Albuquerque, New Mexico, Apr. 1997.

SILVIA, L.C., PETRAGLIA, A., PETRAGLIA, M.R. (2003).
. Industrial Electronics, 2003. ISIE '03. 2003 IEEE Int. Symp, vol. 1, pages 607-611, June 2003.

SUNYOTO, H., VAN DER MARK, W., GAVRILA, D.M. (2004).
Symposium, 2004 IEEE, pages 319–324.

TAKENO, J.; HACHIYAMA, S. (1991).
. IEEE 1991.

YOSHIDA, K.; HIROSE, S. (1992).
. Proceedings of the 1997 IEEE International Conference on Robotics and Automation, pages 1765-1770, Nice, France, May 1992.

MENDONÇA, A.; ZELONOVKY, R.
http://www.mzeditora.com.br/artigos/fpga_fam.htm
Últimо acesso em 06/08/2005.

http://electronics.howstuffworks.com/digital-camera2.htm
Último acesso em 05/07/2005.

ANEXO A – ARTIGO PERIÓDICO LAAR

Artigo publicado no periódico LAAR – Latin American Applied Research (http://www.laar.uns.edu.ar).

Citação bibliográfica

CALIN, G.; RODA, V. O. (2006).
. Latin American Applied Research, v. 37, p. 21-24, 2006.

PÁGINA INTENCIONAMENTE DEIXADA EM BRANCO

REAL-TIME DISPARITY MAP EXTRACTION IN A DUAL HEAD STEREO VISION SYSTEM

G. CALIN† and V. O. RODA†

† *Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, Brazil*
gcalin@uol.com.br, valentin@sel.eesc.usp.br

***Abstract*— This paper describes the design of an algorithm for constructing dense disparity maps using the image streams from two CMOS camera sensors. The proposed algorithm extracts information from the images based on correlation and uses the epipolar constraint. For real-time performance, the processing structure of the algorithm was built targeting implementation on programmable logic, where pipelined structures and condensed logic blocks were used.**

***Keywords*— Stereo vision, disparity map, programmable logic.**

I. INTRODUCTION

Researchers have been giving especial attention to computer vision systems capable of delivering accurate 3D information of an observed scene, which leads to the construction of robust intelligent vehicles. Using low cost sensors, it has been possible to develop stereo vision systems capable of extracting 3D features by passive sensing of the environment.

Most stereo vision implementations are based on a two camera configuration setup, where each camera delivers a two 2D representation of a given scene, as show in Fig. 1. Stereo vision is achieved by extracting 3D information by processing two or more 2D images of a given scene. The processing for extracting the 3D information creates a map that describes which point in the 2D images corresponds to the same point in the 3D scene. Detailed description of the stereo vision problem has been widely studied in past and it is not presented. Refer, for instance, to Grosso *et al.* (1989) and Grosso and Tistarelli (1995) for a detailed study of this subject.

Several stereo algorithms have been proposed in recent years to solve the problem of finding the correspondence of the right and left image. Simple methods employ the measure of absolute or squared differences of the pixels intensities, to measure the similarity between the images (Sunyoto *et al.*, 2004). Other methods, in order to increase accuracy, employ window-based matching, where a cost function is evaluated around the pixel of interest to find the best match. These methods usually do not consider occlusions and present problems in regions displaying little or repetitive textures, leading to similar cost functions and being unable to find the proper match (Darabiha *et al.*, 2003; Silva *et al.*, 2003; Cox *et al.*, 1996).

Birchfield and Tomasi (1998a, b) employed dynamic programming to solve the matching problem, where each scanline –and in some cases in-between scanlines– are described as a dynamic cost function and evaluated with addition of some penalties criteria, like occlusions and large jumps in disparity.

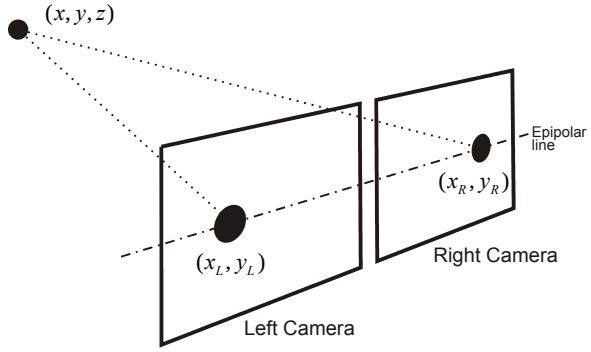


Figure 1. Typical Stereo Vision problem: two cameras, acquiring images of the same scene, have two different 2D representation of a common 3D point. With proper processing, the position and depth of the 3D point can be extracted from the images

II. PROPOSED METHOD

The proposed method employs a windows-based matching technique to find the disparity map on a pair of stereo images. Similar solutions were proposed in many past papers and represent a simple solution to the matching problem. Although it presents some known limitations, as being unable to process occlusions and large disparity jumps, the windows-based matching technique was chosen and used as base for this study due its potential of being ported to a small Field-Programmable Gate Array (FPGA) device.

For the proposed system, F_e and F_d denote the left and right frames from the CMOS camera sensors, located respectively at right and left sides of the scene. The source frame is w pixels large and of h pixels height, with 8-bit grayscale intensities.

The video frames F_e and F_d are addressed as vectors, $\vec{e}$ and $\vec{d}$ of $(w \cdot h)$ length, where the first vector position stores the intensity of the upper-left pixel and the last position stores the bottom-right pixel intensity.

$I_e(k)$ and $I_d(k)$ is the intensity of a given pixel k located respectively in vectors $\vec{e}$ and $\vec{d}$, where $k \in \{0, 1, ..., (w \cdot h) - 1\}$.

The parameter ω is defined as the window width and β the observation region around pixel k, as shown in Figure 2.

In Eq. (1) the similarity function $S(j)$ is defined. This function measures the squared distance from a window in vector $\vec{e}$ (centered in a given pixel k) and a window in vector $\vec{d}$, displaced by j pixels.

$$S(j) = \sum_{i=k-\frac{1}{2}\omega}^{k+\frac{1}{2}\omega} \left[I_e(i) - I_d(i+j) \right]^2, \quad (1)$$

where $i \in Z$, $\omega \in \{1,3,5,7,...\}$

$C(k)$, defined in Eq. (2), returns the best match (minimum distance) for a given window centered in pixel k, when compared with 2β windows in vector $\vec{d}$.

$$C(k) = \min_{-\beta+1 \le j \le \beta} \{ S(j) \}, \quad (2)$$

Since only a limited number of pixels are need for the matching process, vectors $\vec{e}$ and $\vec{d}$ can be trimmed to reduce the memory allocation.

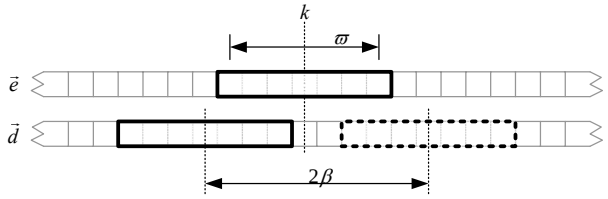


Figure 2. Windows based match. A windows around pixel k in the $\vec{e}$ vector is compared with 2β windows in vector $\vec{d}$

III. DESIGN OF A STEREO VISION SYSTEM

In this paper, for parallel, efficient and condensed hardware implementation of the proposed algorithm, some constrains have been included: 1) CMOS camera sensors with digital interface were used, avoiding implementation of a video interface to digitalize signals from a regular analog video camera; 2) The parameters ω and β were fixed, based in the results of the simulations and camera setup (typically $\omega = 11$ and $\beta = 4$ were used). This allowed a small use of resources, since just a small number of pixels must be present for processing; 3) External memory was used only for storing the final processed disparity map (for debugging purposes).

Also the simplicity of the formulation allowed implementation of all elemental functions with available in-hardware math blocks or by creating simple logic constructions, such as subtraction and magnitude comparison blocks.

A. Pipeline Structure

The developed architecture of the stereo vision system is illustrated in Fig. 3. It consists of a five stage pipeline process capable of processing 160x120 pixels grayscale frames at video rate speed (30 frames/s). Each new pixel, delivered by the cameras, moves the pipeline forward, creating a FIFO process at pixel-rate speed. This allowed a very condensed implementation in terms of logic allocation and memory, and the possibility for future addition of other post-processing algorithms.

Higher clocks frequencies were used to allow execution of machine states within a pixel-rate period. This was needed because some processes could not be parallelized in hardware and needed to be executed in sequential order.

B. Pipeline Stages

In pipeline **stage 1**, vectors $\vec{e}$ and $\vec{d}$ allocate limited memory, since a small history of pixels is needed to process the disparity. In the first pipeline state every new pixel available by the cameras are shifted in the vectors, and the oldest pixel is discarded. This process occurs in two steps, first shifting the vectors and then adding the new available pixels.

Stage 2 is responsible for measuring the all pixel distance from the two vectors, making data available for computing the squared distance in stage 3.

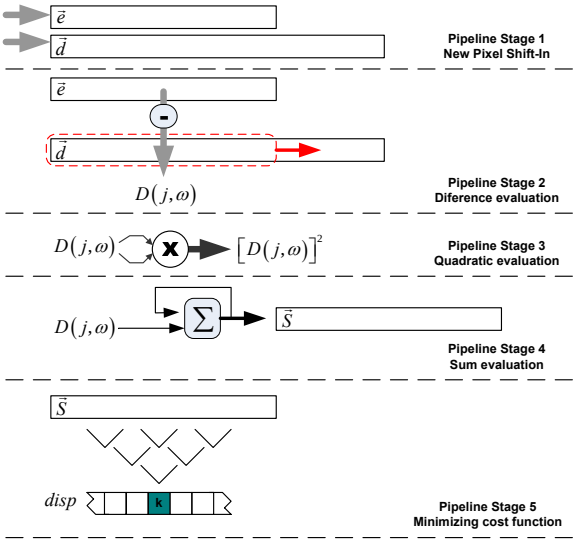


Figure 3. Pipeline structure used to process the proposed stereo vision algorithm

Stage 3 employs hardware multipliers to compute the square function of all data processed in state 2. Since only a limited number of multipliers are available, this stage was broken in several small processes.

Stage 4 computes the sum function for every compared window. This results in a set of data representing the cost function.

Stage 5 uses a tree comparison structure to analyze the data from stage 4. The minimum distance is detected and a disparity coefficient is assigned.

C. Pipeline Structure evaluation

Prior to hardware implementation, the pipeline structure was tested and evaluated using custom developed soft-

ware. Delphi development platform was used due its facilities to emulate low-level binary processes and possibility to quick output results in graphic form for analysis.

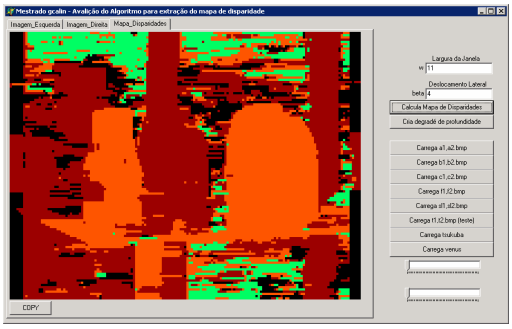
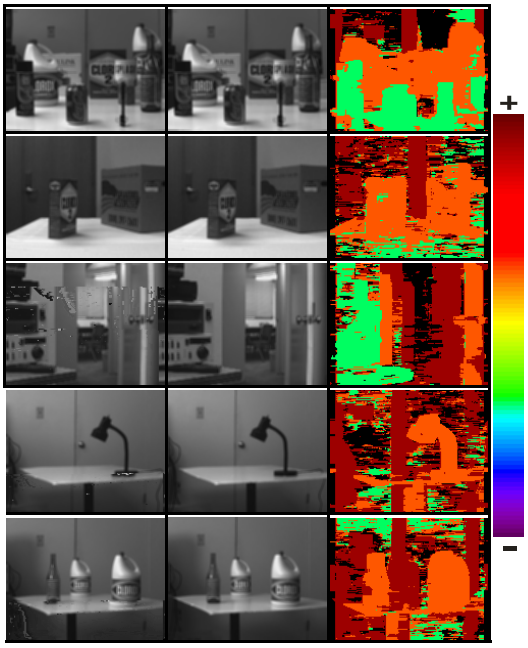


Figure 4. Custom software interface used to evaluate he presented pipeline structure prior to hardware implementation

The results obtained by software emulation were very important during hardware implementation, allow-

Table 1. Results for the proposed algorithm: (a) and (b) are the source pair of stereo images, from JISCT dataset. (c) Shows the dense depth map for each scene. The results were colored from violet (nearest) to red (farthest) to enhance the images contrast.



(a) (b) (c)

Table 2. Results for the proposed algorithm: (a) and (b) are the source pair of Tsukuba stereo images. (c) Shows the resulting dense depth map. The result was colored from violet (nearest) to red (farthest) to enhance the image contrast.

(a) (b) (c)

Table 3. Results for the proposed algorithm: (a) and (b) are the source pair of synthetic stereo images. (c) Shows the resulting dense depth map. The result was colored from violet (nearest) to red (farthest) to enhance the images contrast.

(a) (b) (c)

IV. CONCLUSIONS

This paper has shown the feasibility of constructing high performance stereo vision system on programmable logic, while maintaining construction simplicity and low-cost.

Although there are some limitations, the system seems suitable for serving as an artificial depth reference for autonomous vehicle guidance. In special, the dense real-time depth maps provided by programmable hardware may continuously delivery high amounts of information to navigation and mapping software, lowering the usual computational burden involved in decoding and processing stereo camera images by conventional software methods.

Also, the proposed structure enables addition of post and pre-processed with low impact on processing time, allowing implementation of filters, algorithms for feature tracking, object recognition, among other possible proposals.

REFERENCES

Birchfield, S. and C. Tomasi, "Depth Discontinuities by Pixel-to-Pixel Stereo," *Proc of the 1998 IEEE Int. Conf. on Computer Vision*, Bombay, India 1073-1080 (1998a).

Birchfield, S. and C. Tomasi, "A Pixel Dissimilarity Measure That is Insensitive to Image Sampling," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, **20**, 401-406 (1998b).

Cox, I.J., S.L. Hingorani, S.B. Rao and B.M. Maggs, "A Maximum Likelihood Stereo Algorithm," *Computer Vision and Image Understanding*, **63**, 542-567 (1996).

Darabiha, A., J. Rose and W.J. MacLean, "Video-Rate Stereo Depth Measurement on Programmable Hardware," *Proc. of the 2003 IEEE Computer Society Conf. on Computer Vision and Pattern Recognition*, **1**, 203-210 (2003).

Grosso, E., G. Sandini and M. Tistarelli, "3D object reconstruction using stereo and motion," *IEEE Trans. System, Man and Cybernetics*, **19**, 1465-1476 (1989).

Grosso, E. and M. Tistarelli, "Active/Dynamic Stereo Vision," *IEEE Trans. On Pattern Analysis and Machine Intelligence*, **17**, 868-879 (1995).

Silva, L.C., A. Petraglia and M.R. Petraglia, "Stereo vision system for real time inspection and 3D reconstruction," *Industrial Electronics, 2003. ISIE '03. 2003 IEEE Int. Symp.*, **1**, 607-611 (2003).

Sunyoto, H., W. van der Mark and D.M. Gavrila, "A comparative study of fast dense stereo vision algorithms," *IEEE Intelligent Vehicles Symposium*, 319–324 (2004)

Received: April 14, 2006.
Accepted: September 8, 2006.
Recommended by Special Issue Editors Hilda Larrondo, Gustavo Sutter.

ANEXO B – ARTIGO SPL2006

Artigo submetido e aceito na SPL 2006 (
) realizada entre os dias 8 a 10 de Março de 2006

PÁGINA INTENCIONAMENTE DEIXADA EM BRANCO

Real-time disparity map extraction in a dual head stereo vision system

Gabriel Calin, Valentin Obac Roda

Universidade de São Paulo – Escola de Engenharia de São Carlos, São Carlos, Brazil,
gcalin@uol.com.br
valentin@sel.eesc.usp.br
http://www.eesc.sc.usp.br

Abstract. This paper describes the design of an algorithm for constructing dense disparity maps using the image streams from two CMOS camera sensors. The proposed algorithm extracts information from the images based on correlation and uses the epipolar constraint. For real-time performance, the processing structure of the algorithm was built targeting implementation on programmable logic, where pipelined structures and condensed logic blocks were used.

1 Introduction

Researchers have been giving especial attention to computer vision systems capable of delivering accurate 3D information of an observed scene, which leads to the construction of robust intelligent vehicles. Using low cost sensors, it has been possible to develop stereo vision systems capable of extracting 3D features by passive sensing of the environment.

Most stereo vision implementations are based on a two camera configuration setup, where each camera delivers a two 2D representation of a given scene, as show in **Fig. 1**. Stereo vision is achieved by extracting 3D information by processing two or more 2D images of a given scene. The processing for extracting the 3D information creates a map that describes which point in the 2D images corresponds to the same point in the 3D scene.

Several stereo algorithms have been proposed in recent years to solve the problem of finding the correspondence of the right and left image. Simple methods employ the measure of absolute or squared differences of the pixels intensities, to measure the similarity between the images. Other methods, in order to increase accuracy, employ window-based matching, where a cost function is evaluated around the pixel of interest to find the best match. These methods usually do not consider occlusions and present problems in regions displaying little or repetitive textures, leading to similar cost functions and being unable to find the proper match.

Some researchers employ dynamic programming to solve the matching problem, where each scanline – and in some cases in-between scanlines – are described as a dynamic cost function and evaluated with addition of some penalties criteria, like occlusions and large jumps in disparity.

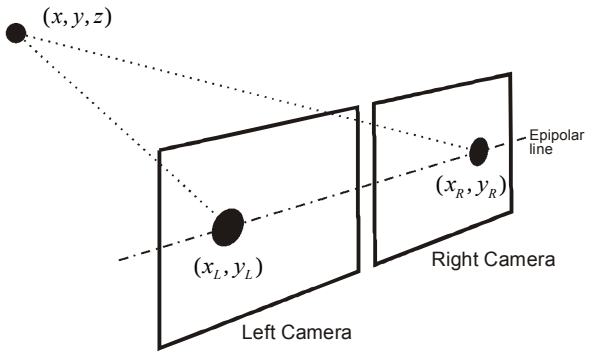


Fig. 1. Typical Stereo Vision problem: two cameras, acquiring images of the same scene, have two different 2D representation of a common 3D point. With proper processing, the position and depth of the 3D point can be extracted from the images

2 Proposed Method

The proposed method employs a windows-based matching technique to find the disparity map on a pair of stereo images. Similar solutions were proposed in many past papers and represent a simple solution to the matching problem. Although it has know limitations, as being unable to process occlusions and large disparity jumps, the windows-based matching technique was chosen and used as base for this study because its potential of being ported to a small Field-Programmable Gate Array (FPGA) devices.

For the proposed system, F_e and F_d denote the left and right frames from the CMOS camera sensors, located respectively at right and left sides of the scene. The source frame is w pixels large and of h pixels height, with 8-bit grayscale intensities.

The video frames F_e and F_d are addressed as vectors, $\vec{e}$ and $\vec{d}$ of $(w \cdot h)$ length, where the first vector position stores the intensity of the upper-left pixel and the last position stores the bottom-right pixel intensity.

$I_e(k)$ and $I_d(k)$ is the intensity of a given pixel k located respectively in vectors $\vec{e}$ and $\vec{d}$, where $k \in \{0,1,...,(w \cdot h)-1\}$.

The parameter ω is defined as the window width and β the observation region around pixel k, as shown in **Fig. 2**.

In equation **(1)** the similarity function $S(j)$ is defined. This function measures the squared distance from a window in vector $\vec{e}$ (centered in a given pixel k) and a window in vector $\vec{d}$, displaced by j pixels.

$$S(j)=\sum_{i=k-\frac{1}{2}\varpi}^{k+\frac{1}{2}\varpi}[I_e($$

$C(k)$, defined in equatio

window centered in pixel k, w

Since only a limited numb

$\vec{d}$ can be trimmed to reduce

d

k

e

3

2

Also the simplicity of the formulation allowed implementation of all elemental functions with available in-hardware math blocks or by creating simple logic constructions, such as subtraction and magnitude comparison blocks.

3.1 Pipeline Structure

The developed architecture of the stereo vision system is illustrated in **Fig. 3**. It consists of a five stage pipeline process capable of processing 160x120 pixels grayscale frames at video rate speed (30 frames/s). Each new pixel, delivered by the cameras, moves the pipeline forward, creating a FIFO process at pixel-rate speed. This allowed a very condensed implementation in terms of logic allocation and memory, and the possibility for future addition of other post-processing algorithms.

Higher clocks frequencies were used to allow execution of machine states within a pixel-rate period. This was needed because some processes could not be parallelized in hardware and needed to be executed in sequential order.

3.2 Pipeline Stages

In pipeline **stage 1**, vectors $\vec{e}$ and $\vec{d}$ allocate limited memory, since a small history of pixels is needed to process the disparity. In the first pipeline state very new pixel available by the cameras are shifted in the vectors, and the oldest pixel is discarded. This process occurs in two steps, first shifting the vectors and then adding the new available pixels.

Stage 2 is responsible for measuring the all pixel distance from the two vectors, making data available for computing the squared distance in stage 3.

Stage 3 employs hardware multipliers to compute the square function of all data processed in state 2. Since only a limited number of multipliers are available, this stage was broken in several small processes.

Stage 4 computes the sum function for every compared window. This results in a set of data representing the cost function.

Stage 5 uses a tree comparison structure to analyze the data from stage 4. The minimum distance is detected and a disparity coefficient is assigned.

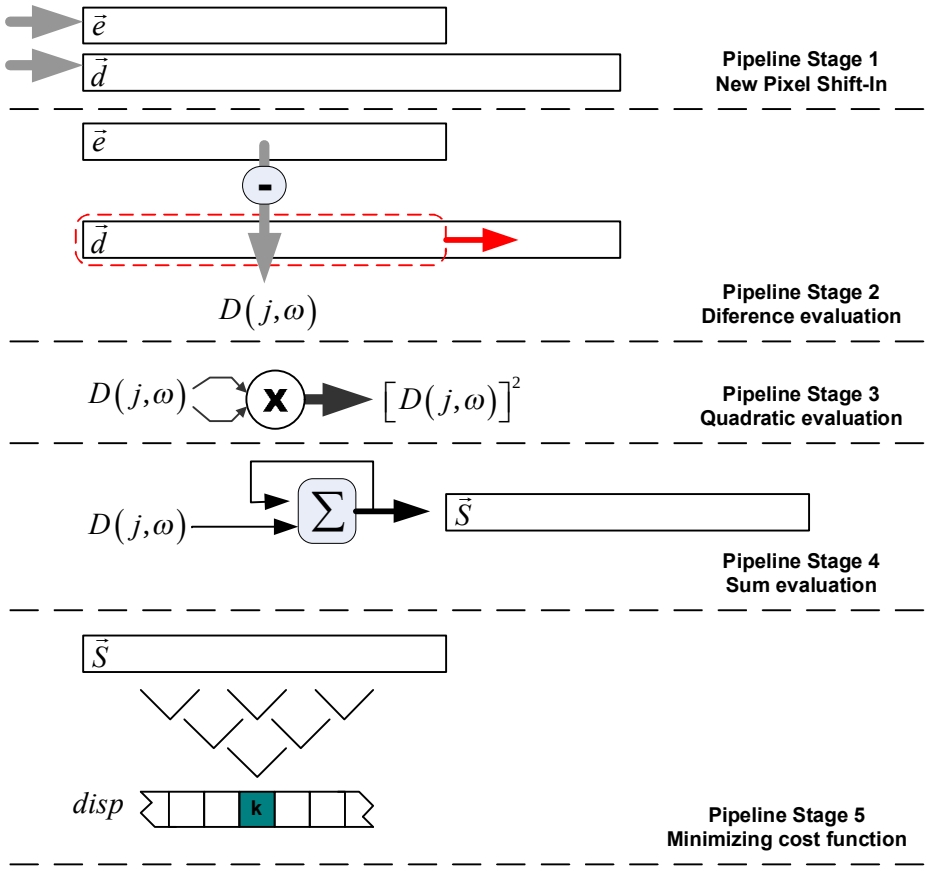


Fig. 3. Pipeline structure used to process the proposed stereo vision algorithm

3.3 Hardware

To test the algorithm, a Xilinx Spartan-3 development board was used. This board uses a Spartan-3 FPGA (model XC3S200), a low-cost programmable logic, with 200k logic blocks and twelve 18-bit hardware multipliers.

A base clock of 50MHz was used for synthesizing all need system clocks.

To acquire the images a pair of low cost OmniVision CMOS sensor were used. Two web-cams were disassembled and its image sensors wired to obtain the raw digital data, at video-rate speed.

In addition a serial interface was build for debugging purposes. The processed disparity map was stored in an external SRAM memory, also available in the development board.

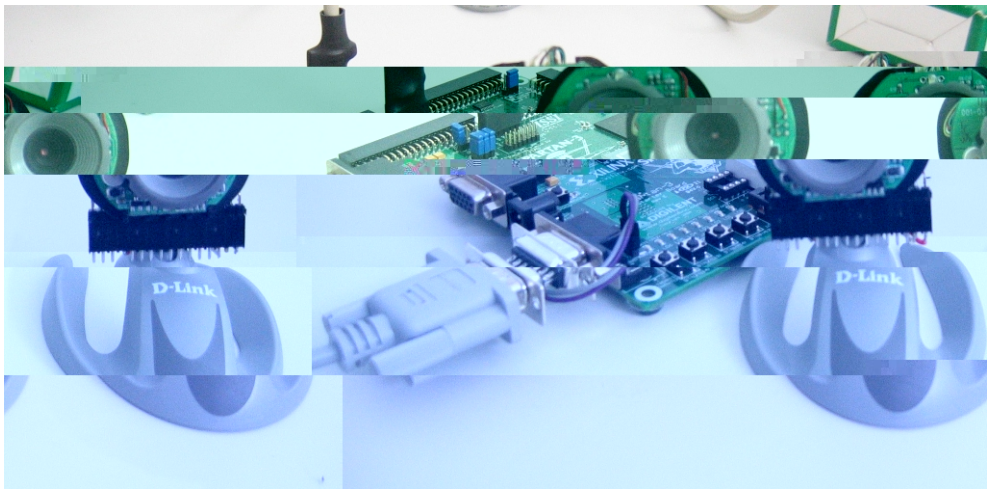


Fig. 4. Hardware setup, including the Xilinx FPGA development board and the two CMOS cameras.

4 Results

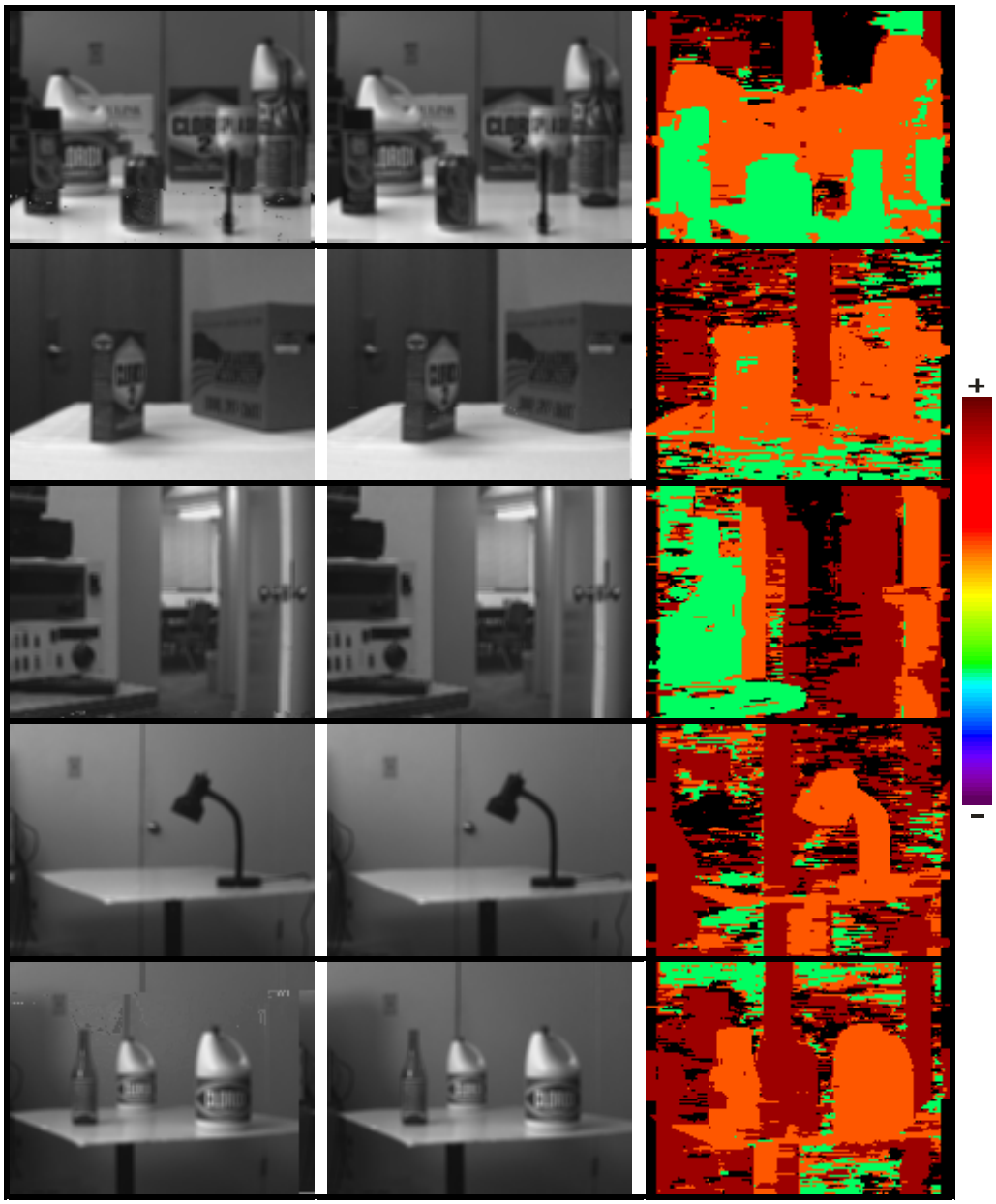
To verify the proposed algorithm, pairs of pictures from JISCT repository were used. The use of this dataset if specially interesting for future comparison of the obtained results with results of algorithms proposed by other researchers.

The source images were pre-processed, with a 4:1 reduction of size and use of an average anti-aliasing filter.

The results shown in **Table 1** (column c) were artificially colored to evidence the depth of the objects in the scene. An arbitrary color map was used, where the nearest to the furthermost objects were shown from violet to red tonalities respectively.

The pipelined structure, running at pixel-rate speed (576 kHz) could delivery the first disparity pixel after approximately 8.68μs. This allowed observing the disparity map of the scene without any noticeable delay or framing loss.

To test images from JISCT dataset, a special routine uploaded them to the external SRAM of the stereo vision system, and after 33ms the disparity map was downloaded back to the computer, for analysis.



(a) (b) (c)

Table 1. Results for the proposed algorithm: (a) and (b) are the source pair of stereo images, from JISCT dataset. (c) Shows the dense depth map for each scene. The results were colored from violet (nearest) to red (farthest) to better contrast of the images.

5 Conclusions

This paper has shown the feasibility of constructing high performance stereo vision system on programmable logic, while maintaining construction simplicity and low-cost.

Although there are some limitations, the system seems suitable for serving as an artificial depth reference for autonomous vehicle guidance.

Also, the proposed structure enables addition of post and pre-processed with low impact on processing time, allowing implementation of filters, algorithms for feature tracking, object recognition, among other possible proposals.

References

1. Belhumer, P. N.: A Binocular Stereo Algorithm for Reconstructing Slop: lrito, p lllritj0.10464 Tc (l) Tj-0k.0mp j0.10464 Tc (itj0

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo