UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS
FACULDADE DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Francisco Carrera Nascimento Junior

Desenvolvimento de Hardware Reconfigurável Utilizando o Processador Intel/Altera Nios II Softcore para Execução do Algoritmo de Detecção de Face de Viola-Jones

Belém – Pará

2018

Francisco Carrera Nascimento Junior

Desenvolvimento de Hardware Reconfigurável Utilizando o Processador Intel/Altera Nios II Softcore para Execução do Algoritmo de Detecção de Face de Viola-Jones

Trabalho de Conclusão de Curso apresentado como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação. Universidade Federal do Pará. Faculdade de Computação. Instituto de Ciências Exatas e Naturais.

Orientador: Prof. Dr. Dionne Cavalcante Monteiro

Belém – Pará

2018

Francisco Carrera Nascimento Junior
Desenvolvimento de Hardware Reconfigurável Utilizando o Processador Intel/Altera Nios II Softcore para Execução do Algoritmo de Detecção de Face de Viola-Jones/ Francisco Carrera Nascimento Junior. – Belém – Pará, 2018-
83 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Dionne Cavalcante Monteiro

Trabalho de Conclusão de Curso – Universidade Federal do Pará – UFPa
Instituto de Ciências Exatas e Naturais – Icen
Faculdade de Computação – Facomp, 2018.
1. Sistemas Embarcados. 2. Computação Reconfigurável. 3. FPGA. 4. Nios II. 5. Viola-Jones. I. Prof. Dr. Dionne Cavalcante Monteiro. II. Universidade Federal do Pará. III. Faculdade de Comunicação. IV. Desenvolvimento de Hardware Reconfigurável Utilizando o Processador Intel/Altera Nios II Softcore para Execução do Algoritmo de Detecção de Face de Viola-Jones.

CDU - - : - - - : - - - . -

Francisco Carrera Nascimento Junior

Desenvolvimento de Hardware Reconfigurável Utilizando o Processador Intel/Altera Nios II Softcore para Execução do Algoritmo de Detecção de Face de Viola-Jones

Trabalho de Conclusão de Curso apresentado como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação. Universidade Federal do Pará. Faculdade de Computação. Instituto de Ciências Exatas e Naturais.

Belém – Pará, _____ de _____ de 20_____.

Conceito: _____.

Prof. Dr. Dionne Cavalcante Monteiro
Orientador

Profª. Drª. Regiane Silva Kawasaki Francês
Avaliadora

Prof. Dr. Roberto Samarone dos Santos Araújo
Avaliador

Belém – Pará
2018

"Tudo posso naquele que me fortalece."
Carta de Paulo aos Filipenses, capítulo 4, versículo 13, Bíblia Sagrada

Dedico este trabalho à Deus
e aos meus Pais, por me guiarem
no caminho certo e por me
proporcionar tudo o que tenho
e tudo o que sou.

AGRADECIMENTOS

À Deus pelo dom da vida, por me guiar nos seus caminhos e nunca me permitir desviar dele, pelas vitórias alcançadas, livramentos e por me fortalecer a cada dia.

Aos meus Pais e demais membros da minha família, pelo cuidado, carinho, sustento, atenção, apoio moral e financeiro, motivação e conselhos necessários para alcançar meus objetivos.

Ao meu orientador Prof. Dr. Dionne Cavalcante Monteiro pelo suporte, oportunidades, conselhos e pela disposição em conduzir este trabalho.

Aos professores pelas disciplinas ministradas e orientações.

Aos colegas e amigos(as) da igreja e do curso pelo companheirismo, momentos de descontração e apoio nesta caminhada.

À Universidade Federal do Pará por fornecer toda a estrutura necessária para o processo de formação de seus estudantes.

RESUMO

A utilização de mecanismos computacionais, mais precisamente os que abordam os conceitos de visão computacional, são de grande relevância sendo vastamente aplicados em áreas como identificação biométrica, detecção e reconhecimento de objetos. Alguns trabalhos mostram estas aplicações como na identificação de faces (CHE; CHANG, 2010) e olhos (MANNAY; ABID, 2015). Para o processamento das imagens, são necessários dispositivos que possuem alto poder computacional, sendo assim o desenvolvimento de sistemas embarcados utilizando o paradigma de computação reconfigurável (RC - *Reconfigurable Computing*), como o FPGA (*Field Programmable Gate Array*), mostra-se promissor neste contexto. Diferentemente de outras abordagens como os ASICs (*Application Specific Integrated Circuits*) onde se desenvolve um hardware que posteriormente não pode ser reprogramado, os FPGAs fornecem a flexibilidade no desenvolvimento de funcionalidades específicas. Portanto, quando se necessita de futuras atualizações em projetos e adequações às novas tecnologias, oferece ao desenvolvedor a opção de sintetizar uma nova descrição de hardware (*HDL - Hardware Description Language*). Diante das vantagens oferecidas pelos FPGAs, foi adotado o mesmo com objetivo de realizar o processo de detecção facial, e optou-se pela utilização de um processador customizável *softcore* Nios II da Intel/Altera, memória SDRAM (*Synchronous Dynamic Random Access Memory*) e outros componentes necessários para o funcionamento do sistema, que serão responsáveis por armazenar e executar o algoritmo responsável pela referida detecção. É utilizado o algoritmo proposto por (VIOLA; JONES, 2001), desenvolvido em C/C++, para realizar o processamento da imagem e indicar a presença ou não de uma ou mais faces. Após o desenvolvimento do sistema e integração do algoritmo, foram realizadas execuções com a leitura de imagens previamente armazenadas no computador e transmitidas ao FPGA e também a captura proveniente da câmera Terasic TRDB-D5M. Foi observado o comportamento das detecções de face em diversas imagens com características como rosto na posição normal, inclinada, com e sem a utilização de óculos, diferentes formatos de rosto e cabelo. Obteve-se êxito das execuções nos cenários apresentados, bem como a detecção e demarcação da localização das faces.

Palavras-chaves: Sistemas Embarcados, Computação Reconfigurável, FPGA, Nios II, Viola-Jones.

ABSTRACT

The use of computational mechanisms, more precisely those that approach the concepts of computer vision, are of great relevance being widely applied in areas such as biometric identification, detection and recognition of objects. Some works show these applications as in the identification of faces (CHE; CHANG, 2010) and eyes (MANNAY; ABID, 2015). For the processing of the images, devices that have high computational power are required, thus the development of embedded systems using the reconfigurable computing paradigm (RC), such as the FPGA (*Field Programmable Gate Array*), is promising in this context. Unlike other approaches such as Application Specific Integrated Circuits (ASICs) where hardware is developed that can not subsequently be reprogrammed, FPGAs provide flexibility in the development of specific functionalities. Therefore, when future design updates and upgrades to new technologies are required, it offers the developer the option of synthesizing a new hardware description (HDL). In the face of the advantages offered by FPGAs, we adopted the same with the objective of performing the facial detection process, and opted for the use of a customizable Intel/Altera Nios II processor, SDRAM (*Synchronous Dynamic Random Access Memory*) and other components necessary for the operation of the system, which will be responsible for storing and executing the algorithm responsible for said detection. We use the algorithm proposed by (VIOLA; JONES, 2001), developed in C/C++, to perform the image processing and indicate the presence or absence of one or more faces. After the system development and integration of the algorithm, executions were performed with the reading of images previously stored in the computer and transmitted to the FPGA and also the capture from the Terasic TRDB-D5M camera. It was observed the behavior of face detections in several images with characteristics such as face in normal position, inclined, with and without the use of glasses, different formats of face and hair. We succeeded in executing the scenarios presented, as well as the detection and demarcation of the faces locations.

Key-words: Embedded Systems, Reconfigurable Computing, FPGA, Nios II, Viola-Jones.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

ASIC	*Application Specific Integrated Circuits*, em português "Circuitos Integrados de Aplicação Especifica"
BSP	*Board Support Package*, em português "Pacote de Suporte de Placa"
DMA	*Direct Memory Access*, em português "Acesso Direto à Memória"
FPGA	*Field Programmable Gate Array*, em português "Matriz de Portas Programáveis em Campo"
HDL	*Hardware Description Language*, em português "Linguagem de Descrição de Hardware"
I/O	*Input/Output*, em português "Entrada/Saída"
LCD	*Liquid Crystal Display*, em português "Display de Cristal Líquido"
LE	*Logic Elements*, em português "Elementos Lógicos"
LED	*Light Emitting Diode*, em português "Diodo Emissor de Luz"
LUT	*Look up Table*, em português "Tabela de Pesquisa"
NTSC	*National Television System Committee*, em português "Comitê Nacional do Sistema de Televisão"
PAL	*Phase Alternation Line*, em português "Linha de Fase Alternada"
PC	*Personal Computer*, em português "Computador Pessoal"
PLL	*Phase-Locked Loop*, em português "Malha de Captura de Fase"
PGM	*Portable Gray Map*, em português "Mapa de Imagem em Escala de Cinza Portátil"
RAM	*Random Access Memory*, em português "Memória de Acesso Aleatório"
RC	*Reconfigurable Computing*, em português "Computação Reconfigurável"
SBT	*Software Build Tools*, em português "Ferramentas de Compilação de Software"
SD	*Secure Digital Card*, em português "Cartão Digital Seguro"

SDRAM	*Synchronous Dynamic Random Access Memory*, em português "Memória de Acesso Aleatório Dinâmico Síncrono"
SECAM	*Séquentiel Couleur à Mémoire*, em português "Cor Sequencial com Memória"
SRAM	*Static Random Access Memory*, em português "Memória Estática de Acesso Aleatório"
SSRAM	*Synchronous Static Random Access Memory*, em português "Memória de Acesso Aleatório Estático Síncrono"
USB	*Universal Serial Bus*, em português "Barramento Serial Universal"
VGA	*Video Graphics Array*, em português "Padrão de Disposição Gráfica para Vídeo"
VHDL	*VHSIC Hardware Description Language*, em português "Linguagem de Descrição de Hardware para VHSIC"
VHSIC	*Very High Speed Integrated Circuits*, em português "Circuitos Integrados de Altíssima Velocidade"

SUMÁRIO

1 INTRODUÇÃO

A Visão Computacional é o ramo da Ciência da Computação que possui a finalidade de possibilitar que imagens sejam interpretadas por sistemas artificiais implementados em computadores (MAIA, 2010). A utilização de mecanismos de detecção e reconhecimento de imagens têm-se tornado cada vez mais importantes, sua aplicação segue deste a identificação biométrica como a face (CHE; CHANG, 2010) e olhos (MANNAY; ABID, 2015), que podem ser utilizados no processo de autenticação em sistemas computacionais; na interação humano-computador, funcionando assim, como uma interface no processo de comunicação entre o usuário e o sistema; na identificação de face em fotografias e vídeos, que possuem diversas aplicações com identificação de pessoas ou até mesmo suspeitos em crimes; e na identificação em tempo real de sinais de trânsito (SOUKI; BOUSSAID; ABID, 2008), pedestres ou até outros tipos de objetos externos que, no contexto de carros inteligentes, permitem auxiliar motoristas na condução de seus veículos, sendo assim, um recurso que permite a segurança.

Para o funcionamento destes mecanismos utilizam-se algoritmos computacionais, podendo ser no contexto de hardware ou software, que executam diversos processos que permitem identificar formas, cantos, cores ou o agrupamento destes elementos que auxiliam a construção de significados a estes objetos, ou seja, a sua detecção. Porém o desempenho destes algoritmos dependem de sua complexidade e podem exigir alto custo computacional ou até mesmo um hardware dedicado (embarcado) para o seu funcionamento.

Para a execução destes algoritmos foi proposto a utilização de um hardware reconfigurável Intel/Altera Cyclone II (DE2-70) para projetar SoCs (*System-on-a-Chip*) que trabalhará com as linguagens de alto nível C/C++ e também será incorporado ao projeto uma câmera (VGA) com objetivo de capturar imagens de um determinado objeto para serem analisadas pelo hardware.

1.1 Justificativa

A utilização do hardware reconfigurável possibilita a operação para um objetivo em específico, sendo assim, diferentemente de um sistema computacional comum, o algoritmo que será executado neste hardware não sofrerá influência de outros processos executados em concorrência a este. Com isso têm-se um ganho de desempenho por causa da utilização de um sistema embarcado e um processamento dedicado à execução do algoritmo de detecção de imagem.

Também, como justificativa, têm-se a complexidade destes algoritmos em razão das muitas operações com o tipo de tratamento das imagens que exige um alto poder computacional, necessitando assim, de um dispositivo que atenda de forma específica os requisitos que estes algoritmos requerem.

A utilização da Terasic DE2-70 justifica-se pelo seu baixo custo financeiro em contrapartida possui alta escalabilidade devido ao fato de ser reconfigurável, sendo que este possui 70.000 elementos lógicos utilizados para configuração de um sistema de uso específico. Com isso, por meio da ferramenta QSYS disponibilizada no *framework* Quartus II da Intel/Altera, pode-se customizar o sistema que executará os algoritmos de detecção facial, tornando sua operação mais otimizada.

Dentre os diversos algoritmos de detecção facial existentes, optou-se por utilizar o proposto por (VIOLA; JONES, 2001) para ser executado na Terasic DE2-70. Pois este apresenta uma seleção eficiente de características que permite a detecção de face ou até mesmo outros objetos, se treinado para tal. Também sua detecção é invariante aos fatores de escala e posição do objeto em uma imagem, possui baixa taxa de falsos positivos, alta taxa de acertos e baixo custo computacional.

1.2 Motivação

A utilização de sistemas embarcados e hardware reconfigurável permitem o maior desempenho para a execução de determinadas tarefas, sendo assim, bastante útil para aplicação no presente trabalho. Outro fator de grande importância é sua estrutura compacta que permitem o baixo custo do hardware tanto de consumo de energia quanto financeiro. Também por conceituar-se como FPGA (estrutura reconfigurável) permite que o desenvolvedor construa seu hardware conforme necessidades, bastando que previamente obtenha os conhecimentos técnicos para tal desenvolvimento.

A utilização deste dispositivo acoplado com a funcionalidade de processamento de imagem, permite uma gama de aplicações práticas e bastante úteis, como a inserção na área de acessibilidade, mais precisamente o monitoramento de PCD (pessoas com deficiência).

1.3 Objetivos

1.3.1 Objetivo Geral

Desenvolvimento de um sistema embarcado baseado no paradigma computação reconfigurável, com a utilização de um FPGA e uma câmera, que serão programados para efetuar a captura de imagens e posteriormente, com a execução do algoritmo de

processamento de imagens de Viola-Jones em um processador *softcore*, realizar o processo de detecção facial.

1.3.2 Objetivos Específicos

Para alcançar o objetivo geral, os seguintes objetivos específicos tornam-se necessários:

- Verificar viabilidade da utilização do FPGA para o processamento de imagens por meio de trabalhos relacionados;
- Projetar sistema com o FPGA;
- Implementação e configuração dos componentes do subsistema que permite a execução de algoritmos de alto-nível (C/C++);
- Implementação e configuração dos componentes do subsistema que permite o funcionamento de conexão de vídeo externo;
- Implementação e configuração dos componentes que permitem a captura e o armazenamento das imagens;
- Verificar qual algoritmo será utilizado para o processo de detecção facial;
- Implementar o algoritmo de detecção facial definido, de modo que seja compatível com o funcionamento do FPGA; e
- Testar o funcionamento sistema já com o algoritmo integrado.

1.4 Metodologia

Foram realizados levantamentos bibliográficos acerca da utilização do hardware reconfigurável, mais especificamente o processador Nios II, com o objetivo de verificar a viabilidade desta nova proposta, assim como fundamentá-la. Também foram verificados, via literatura, quais algoritmos de detecção de objetos em imagens serão utilizados para serem executados nesta plataforma a fim de verificar seu comportamento. Após essa verificação, foram configurados os componentes a ser utilizados na plataforma de acordo com as necessidades do projeto, como a inclusão do processador *softcore* Nios II e demais componentes necessários ao funcionamento do sistema como o JTAG UART, SDRAM entre outros requeridos ao longo do desenvolvimento.

Posteriormente foi observado o comportamento de todo o sistema, com isso verificou-se a sua eficácia. Também foi realizada a depuração de todo o conjunto com os devidos

ajustes para o seu melhor funcionamento. Foram realizados testes de desempenho verificando assim o tempo de execução e respostas para as devidas entradas de dados, e de posse destes resultados realizou-se comparações com outras plataformas como em computadores de propósito geral para verificar as discrepâncias no tempo de execução, bem como o custo computacional utilizado.

Procurou-se identificar quão vantajoso é a utilização da Intel/Altera Cyclone II em relação à plataforma citada anteriormente. Após os resultados obtidos dos passos anteriores, se espera a definição do espaço de aplicação, podendo esta ser direcionada a área de tecnologias assistivas, autenticação e afins.

1.5 Organização do Trabalho

São apresentados no Capítulo 2 os trabalhos relacionados, que contêm outras abordagens e modelos de implementação de sistemas de processamento de imagens bem como o processo de detecção facial, ao final deste Capítulo são apresentadas comparações dos mesmos.

No Capítulo 3 são descritos os conceitos necessários para o desenvolvimento deste trabalho como sistemas embarcados, computação reconfigurável, FPGA e *Soft-Processor*. Será abordado, também, o algoritmo de Viola-Jones, bem como suas características e funcionamento.

No Capítulo 4 estão descritas todas as etapas do desenvolvimento, como a inserção dos componentes necessários para o funcionamento dos subsistemas para execução de algoritmos de alto-nível (C/C++), exibição de imagens em dispositivo externo, captura de imagens da câmera e integração do algoritmo de detecção facial de Viola-Jones. São relatadas as execuções parciais, para verificar o funcionamento dos subsistemas, e do sistema como um todo juntamente com o algoritmo supracitado, resultando na detecção facial das imagens obtidas da câmera Terasic TRDB-D5M.

E por último, no Capítulo 5, têm-se as conclusões obtidas pelo desenvolvimento deste trabalho e também propostas para trabalhos futuros.

2 TRABALHOS CORRELATOS

Este Capítulo apresenta trabalhos que realizam algum tipo de processamento de imagens e que fazem uso do FPGA como base para seu desenvolvimento. Há implementações que utilizam somente a descrição de hardware (HDL) para o desenvolvimento do sistema e outros que utilizam o processador *softcore* Nios II da Intel/Altera e realizam um co-desenvolvimento entre a utilização e execução de algoritmos em C/C++ e a aceleração fornecida pelo *hardware*, desenvolvida em HDL.

2.1 Design and Implementation of an Open Image Processing System Based on Nios II and Terasic DE2-70 Board

(PYRGAS; KALANTZOPOULOS; ZIGOURIS, 2016) apresentam o desenvolvimento de um sistema de processamento de imagens baseado no processador *softcore* Nios II da Intel/Altera. O sistema é de arquitetura aberta e desenvolvido em FPGA, utiliza-se como plataforma a placa DE2-70 da Terasic, um sensor de captura de imagens TRDB-D5M, um leitor de cartão SD (*Security Data*) e uma tela sensível ao toque, TRDB-LTM LCD (*Liquid Crystal Display*), que é a interface de controle do sistema e exibição dos resultados do processamento.

O sistema proposto integra quatro algoritmos, sendo que três foram implementados pelos autores em VHDL (*VHSIC Hardware Description Language*), utilizando o ambiente de desenvolvimento Quartus, incluindo *negative colour* (cor negativa), *median filter* (filtro mediano) e *sharpen convolution filter* (filtro de nitidez por convolução); e o último algoritmo, *Sobel edge detection* (detecção de bordas Sobel), foi incluído ao projeto, porém pertence a biblioteca da Intel/Altera.

A arquitetura do sistema consiste em três subsistemas principais: *Camera sub-system*, *Image Processing sub-system* e *Touch Panel sub-system*. O primeiro subsistema é responsável por inicializar a câmera e gerir o processo de captura de *frames*. O segundo subsistema encarrega-se de processar a(s) imagem(ns) oriunda(s) da câmera ou de um dispositivo SD *card* por meio da execução dos algoritmos de processamento de imagens supracitados, desenvolvidos em *hardware*. E por último o *Touch Panel sub-system* é o responsável pela inicialização e operação da tela sensível ao toque, exibição das imagens previamente processadas e pela interação humano-sistema. O processador *softcore* Nios II é o responsável por controlar todos os componentes de hardware do sistema.

Durante o processo de inicialização do sistema, o processador carrega na memória as imagens salvas no cartão SD, também realiza a inicialização dos dispositivos TRDB-

D5M e TRDB-LTM LCD e define o modo *default* (padrão). O sistema opera em dois modos: *Video Mode* (Modo Vídeo) e *Photo Mode* (Modo Foto). No primeiro, que é o modo padrão, o *frame* de imagem capturado pela câmera é transferido, por meio do *Camera sub-system*, para o *Pixel Buffer DMA Controller* e este componente armazena o *frame* atual (temporariamente) na memória SRAM (*Static RAM - Random Access Memory*). Já no segundo modo o processo é semelhante, com exceção da origem da imagem, pois esta é obtida de um armazenamento externo SD *card* previamente salva pelo usuário. Em ambos os modos a imagem que foi salva na memória é transferida ao *Image Processing sub-system* para ser realizada a aplicação dos algoritmos de processamento de imagens e posteriormente a imagem resultante é encaminhada ao subsistema responsável pela tela sensível ao toque, para exibição.

O usuário é o responsável por operar o sistema, sendo assim, este pode selecionar o algoritmo a ser utilizado e também visualizar o resultado de sua aplicação nas imagens. Conforme citado no trabalho, como exemplo, foram executados os algoritmos *negative colour* e *edge detection*, posteriormente foram apresentados os resultados da execução conforme imagem exibida no dispositivo TRDB-LTM LCD. Sendo assim, o sistema proposto opera como uma plataforma para implementação e teste de algoritmos de processamento de imagem customizáveis desenvolvidos em *hardware*.

2.2 A Hardware/Software Co-design of a Face Detection Algorithm Based on FPGA

(CHE; CHANG, 2010) propõem o desenvolvimento de um algoritmo para detecção facial que irá compor um sistema de controle de *mouse* por meio de movimentos dos olhos. Portanto, é requerido por este sistema um trabalho em tempo real, sendo assim, foram utilizadas versões otimizadas dos algoritmos *skin color module* e *binary image projection* para garantir que tal requisito seja efetivado. É utilizada a metodologia *co-design*, ou seja, uma co-participação *hardware* dedicado, baseado em FPGA, que resolve partes do algoritmo com alto custo computacional e um microprocessador que gerencia os processos de controle e executa o restante do algoritmo.

O processo de detecção facial é o primeiro passo realizado pelo sistema e este funciona como base para posterior aquisição do movimento dos olhos. Sendo assim, como sub-rotina é realizada a tarefa de segmentação da cor da pele das imagens de entrada no espaço de cor YCbCr (luminância-crominância) e posteriormente, para evitar a influência da iluminação, é convertido para YCb'Cr'. Nesta etapa há duas classes de problemas, entre elas, quais são os pixels que fazem parte ou não de uma área de pele. Para esta questão foi proposto método L.U.T. (*look-up table*), que utiliza os componentes Cb' e Cr' dos *pixels* como parâmetros para detecção de pele, portanto, define-se este *pixel* como

pertencente a cor da pele quando a condição da Equação 2.1 for atendida:

$$\begin{cases} 80 \leq Cb' \leq 127 \\ 137 \leq Cr' \leq 165 \end{cases} \tag{2.1}$$

Após este processo é necessário delimitar a área que contém a face, sendo assim, foi utilizado o método de projeção integral (*integral projection method*) para calcular o retângulo que indicará a localização desta face. Assumindo que o tamanho da imagem seja MxN, e $f(x, y)$ os *pixels* de uma imagem, então o cálculo da projeção horizontal e vertical são realizados, respectivamente, pelas Equações 2.2 e 2.3:

$$Py(i) = \sum_{x=0}^{M-1} f(x, i) \tag{2.2}$$

$$Px(i) = \sum_{y=0}^{N-1} f(i, y) \tag{2.3}$$

Então o valor máximo de Py (MaxPy) é obtido como a largura da face. Utilizando a Equação 2.4:

$$Sx(i) = \sum_{i}^{i+MaxPy-1} Px(i) \tag{2.4}$$

para calcular a integral de Px e obter o valor máximo de Sx como limite esquerdo da face. Por meio do mesmo método, a projeção horizontal na faixa dos limites esquerdo e direito deve obter o limite superior bem como a altura da face. Então o limite direito e inferior pode ser estimado por parâmetros biofísicos.

Verificou-se o funcionamento do algoritmo em dois cenários, o primeiro com a implementação puramente em *software* e executado em um processador com arquitetura ARM9 e *clock* 400MHz; e o segundo com o desenvolvimento *hardware-software* baseado na Intel/Altera Cyclone II FPGA (EP2C70) com o processador *softcore* Nios II e *clock* de 100MHz. Verificou-se, no primeiro, que 85% do tempo de execução total destinava-se ao algoritmo *skin color* e 15% ao *binary image projection*. Portanto, conforme o Quadro 1, obteve-se os seguintes tempos de execução para os algoritmos em questão:

Quadro 1 – Tempo de Execução Utilizando o Processador com Arquitetura ARM9

Algoritmo	Tempo de Execução
Skin color	315ms
binary image projection	61ms
Tempo total de execução	376ms

Fonte: Adaptado de (CHE; CHANG, 2010)

Já no segundo cenário, como propõe este trabalho, o algoritmo *skin color* foi desenvolvido em *hardware* com objetivo de acelerar a execução do mesmo, e o *binary image projection* foi desenvolvido para ser executado no processador Nios. Verificou-se a diminuição do tempo total de execução em comparação ao primeiro cenário como é mostrado no Quadro 2:

Quadro 2 – Tempo de Execução Utilizando Aceleração por Hardware e Nios

Algoritmo	Tempo de Execução
Skin color	13ms
binary image projection	83ms
Tempo total de execução	96ms

Fonte: Adaptado de (CHE; CHANG, 2010)

2.3 The Design of an embedded system (SOPC) for an image processing application

(FRADI; YOUSSEF; MOHSEN, 2017) desenvolvem um *embedded system* (sistema embarcado) para o processamento de imagens no contexto médico. Para tal procedimento realiza o tratamento de imagens com objetivo de melhorar a nitidez, qualidade e reprodutibilidade. Neste trabalho é apresentado um Sistema em Chip Programável (SOPC). É desenvolvido um código descrevendo uma cadeia de processamento de imagem em linguagem C/C++ usando as funções fornecidas pela biblioteca OpenCV. Depois desta etapa, utiliza uma versão do Linux para sistemas embarcados denominado uClinux, este visa facilitar a implementação da cadeia de processamento de imagens feita com a biblioteca OpenCV e é executado no processador Nios II da placa Cyclone III FPGA da Intel/Altera. Após o processamento da imagem o resultado, ou seja, a imagem já tratada, é exibido em um *display* LCD do Kit de Avaliação Embarcado (NEEK) da Intel/Altera.

2.4 Hardware Acceleration of a Face Detection System on FPGA

(DHARAN; KHALIL-HANI; SHAIKH-HUSIN, 2015) propõem um sistema de detecção de rosto acelerado por *hardware* usando o método de segmentação de cores de pele no FPGA. Os processos de pré-processamento de segmentação de pele, filtragem e rotulagem de componentes conectados foram projetados usando arquitetura de *hardware* orientada por fluxo. Para avaliar o desempenho do sistema em termos de velocidade de execução, o algoritmo também foi implementado completamente em *software* e executado pelo processador *softcore* Nios II, com o FPGA Cyclone IV da Intel/Altera.

2.5 Comentários Finais

Verificou-se, com a revisão de literatura, que o processador Nios II obteve um desempenho satisfatório na execução dos algoritmos desenvolvidos, sejam os que realizam a detecção de face, realização de filtros ou como interface para o *hardware*. Porém quando se almeja o desenvolvimento de sistemas que se espera um mínimo tempo de resposta, como em tempo real, exige-se a otimização por meio da aceleração em hardware.

Os trabalhos relacionados apresentados neste Capítulo com exceção do (FRADI; YOUSSEF; MOHSEN, 2017), realizam esta aceleração por hardware. No Quadro 3 são mostradas as características em comum e diferenças dentre os artigos. Porém, por meio das similaridades e paradigma utilizado, observa-se as características que são importantes para o processo de execução de algoritmos de processamento de imagens e, sendo assim, foram destacadas.

Quadro 3 – Comparação dos Trabalhos

Título do Trabalho	Algoritmo de Processamento de Imagem Utilizado	Tipo de Execução no Nios	Tipo de Aceleração por Hardware	Versão do FPGA
Design and Implementation of an Open Image Processing System Based on Nios II and Terasic DE2-70 Board	Negative Colour Median Filter Sharpen Convolution Filter Sobel Edge Detection	Controle dos Componentes do Sistema	Aceleração Realizada em Todos os Algoritmos de Processamento de Imagens	Cyclone II
A Hardware/Software Co-design of a Face Detection Algorithm Based on FPGA	Skin Color Module Binary Image Projection	Execução do Algoritmo Binary Image Projection	Aceleração do Algoritmo Skin Color Module	Cyclone II
The Design of an embedded system (SOPC) for an image processing application	Biblioteca do Open CV	Execução do Sistema uClinux que Permite a Utilização da Biblioteca do Open CV	Não Realiza Aceleração por Hardware	MAX 10
Hardware Acceleration of a Face Detection System on FPGA	Skin Color Segmentation	Execução de Todos os Algoritmos de Processamento de Imagens para Fins de Comparação	Aceleração Realizada em Todos os Algoritmos de Processamento de Imagens	Cyclone IV
Desenvolvimento de Hardware Reconfigurável Utilizando o Processador Intel/Altera Nios II Softcore para Execução do Algoritmo de Detecção de Face de Viola-Jones	Viola-Jones	Execução do Algoritmo de Processamento de Imagens de Viola-Jones	Não Realiza Aceleração por Hardware	Cyclone II

Fonte: Elaborada pelo Autor

3 FUNDAMENTAÇÃO TEÓRICA

Neste Capítulo serão abordados os conceitos de sistemas embarcados, como um breve histórico, requisitos e aplicações. Também é discorrido sobre o paradigma computação reconfigurável, como as vantagens de sua utilização ao invés dos ASICs (*Application Specific Integrated Circuits*), como sua flexibilidade de desenvolvimento. É dissertado sobre o FPGA, informando que este utiliza o paradigma citado anteriormente e também a sua estrutura. Posteriormente será comentado acerca das características dos processadores implementados por *software*. E ao final será apresentado o algoritmo de Viola-Jones, bem como as etapas de seu funcionamento.

3.1 Sistemas Embarcados

Sistemas embarcados são sistemas computacionais encarregados de executar uma determinada tarefa ou função específica para a qual foi desenvolvido. Normalmente são utilizados em aplicações onde demandam baixo consumo de energia, tarefas consideradas repetitivas e que se exige alto poder de processamento.

O primeiro sistema embarcado que se têm notícia foi o AGC (*Apollo Guidance Computer*), este foi inventado pelo laboratório de instrumentação do MIT (*Massachusetts Institute of Technology*) nos Estados Unidos e a partir do ano de 1966 foi fabricado pela empresa Raytheon. Este dispositivo era um computador de bordo utilizado nas aeronaves do projeto Apollo nas décadas de 60 e 70 e foi responsável pelo controle de navegação, orientação e funcionamento do módulo de comando e lunar, por isso era considerado um dos itens mais importantes e extremamente crítico (HALL, 1996).

Segundo (HALLINAN, 2007) um sistema embarcado é projetado para um determinado fim, portanto possui recursos limitados de memória e processamento. Normalmente, sistemas desenvolvidos com esta característica são desenvolvidos para aplicações que não requeresse intervenção humana.

Grande parte dos sistemas embarcados possuem como requisito o mínimo tempo de resposta em suas funções (aplicações em tempo real), sendo assim neste tipo de aplicação a não execução de uma tarefa em tempo determinado ou até mesmo sua não execução pode acarretar em supressão de dados, problemas graves que ocasionam perda de confiabilidade e danos ao sistema. Porém, o desenvolvimento de ferramentas que garantam que o sistema cumpra os requisitos supracitados, bem como técnicas que objetivam a melhoria de confiabilidade, não são vastamente utilizados em sistemas embarcados (RAVI et al., 2004).

Geralmente as aplicações embarcadas são implementadas para realizar apenas uma determinada função, não sendo permitidas alterações oriundas do usuário final. O mesmo pode até configurar ou alterar a forma como o sistema se comporta, entretanto não pode interferir no objetivo específico para que o sistema foi desenvolvido (STUDNIA et al., 2013) (KERMANI et al., 2013).

3.1.1 Computação Reconfigurável

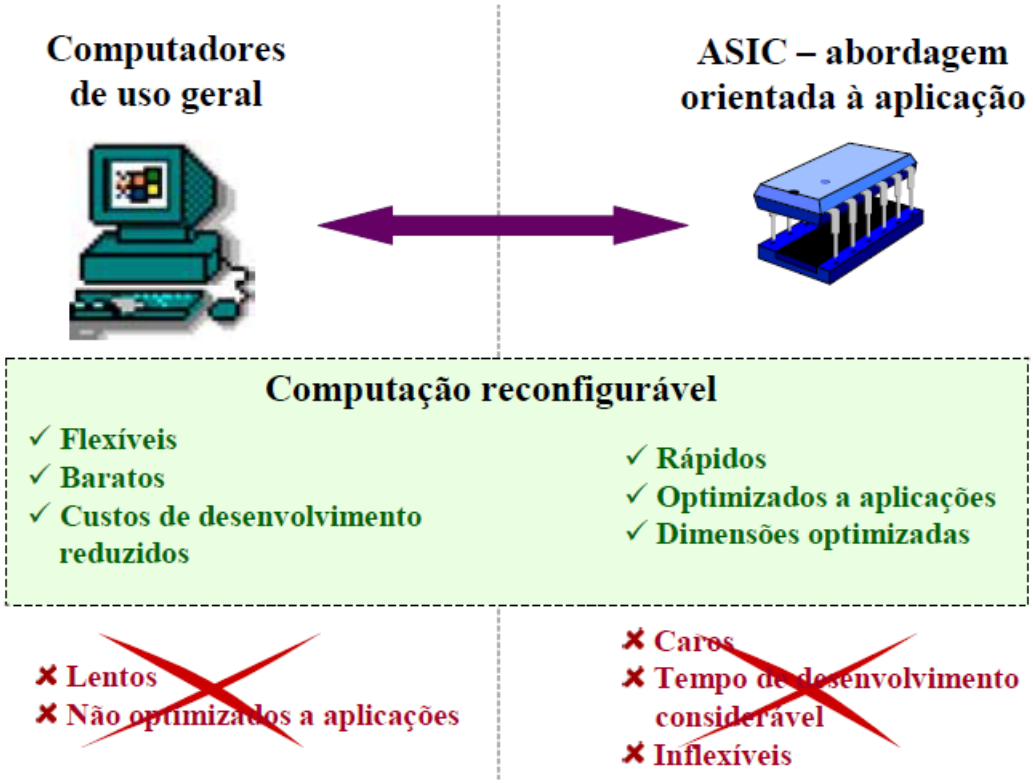
A utilização de dispositivos de uso específico (embarcados) possui o benefício de ser otimizado para uma determinada aplicação, sendo assim, quando bem desenvolvidos, resultam em um hardware final de alto desempenho em comparação aos computadores de uso geral, os ASICs são exemplos mais comuns da utilização desta abordagem. Porém, devido aos elevados custos de projeto, desenvolvimento e o tempo para realização dos mesmos, a viabilidade de produção destes dispositivos somente justifica-se quando realizada em grandes quantidades.

Outros fatores de grande relevância que desfavorecem a utilização de soluções baseadas em ASICs são o risco do *hardware* produzido tornar-se obsoleto devido ao rápido advento de novas tecnologias que tornam mais velozes os computadores de uso geral, já que o primeiro (ASIC) é uma opção para substituição destes computadores; e a inflexibilidade do *hardware* produzido, visto que a funcionalidade não pode ser alterada após a fabricação do circuito integrado.

Diferentemente dos ASICs, os computadores convencionais (uso geral) são flexíveis, pois possuem o benefício da utilização de diversos *frameworks* e bibliotecas que facilitam sua programação e também permitem a alteração dos algoritmos implementados e até mesmo a inclusão de novas funcionalidades. Entretanto, a utilização deste computador, quando se aplica a um escopo específico, não é uma solução que se garanta o melhor desempenho.

Diante das desvantagens supracitadas viu-se necessário a utilização de uma arquitetura que possibilita combinação do ganho de velocidade proporcionado pela utilização de um *hardware* dedicado com a flexibilidade dos computadores de uso geral, que permitem futuras atualizações no desenvolvimento das funcionalidades (SKLIAROVA; FERRARI, 2003). Assim sendo, surge a computação reconfigurável que engloba estes benefícios possibilitando eliminar as desvantagens relacionadas as duas abordagens, conforme Figura 1.

Figura 1 – Vantagens e Desvantagens dos Computadores de Uso Geral e ASICs



Fonte: (SKLIAROVA; FERRARI, 2003)

Segundo (SKLIAROVA; FERRARI, 2003) a computação reconfigurável têm como base dispositivos lógicos reprogramáveis que, dependendo de sua implementação, podem alcançar níveis altos de desempenho comparados aos dispositivos ASIC, também possuem a vantagem de programação a nível de portas lógicas.

Para (BOBDA, 2007), a computação reconfigurável é o estudo da computação que engloba os dispositivos considerados reconfiguráveis, juntamente com suas características como arquiteturas e aplicações, o mesmo também considera esta abordagem como uma forma de programação que utiliza estes dispositivos, configurados de acordo com descrições de *hardware*, para desenvolvimento de circuitos eletrônicos. Sendo assim, o objetivo deste paradigma é compensar as desvantagens existentes entre o *software* e o *hardware* aproveitando os benefícios de cada abordagem, permitindo assim um desempenho maior que da solução por *software*, enquanto mantém um nível de flexibilidade maior que do *hardware*.

3.1.2 FPGA (Field Programmable Gate Arrays)

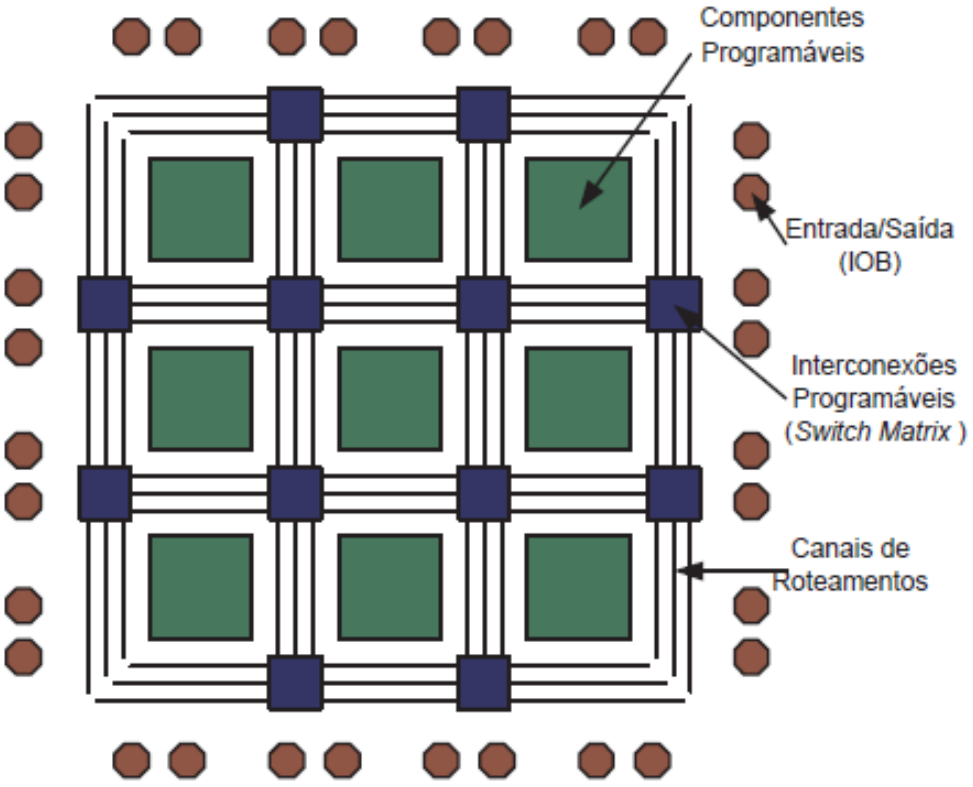
Atualmente, os dispositivos mais comuns que utilizam o paradigma da computação reconfigurável são os FPGAs. Estes são, inicialmente, utilizados para prototipação de circuitos. O desempenho e os recursos disponíveis nos FPGAs são análogos aos ASICs,

porém com o benefício da flexibilidade inerentes às implementações em software.

Um FPGA é um circuito integrado programável pelo usuário da qual podem ser configurados, por *software*, de acordo com as necessidades de projeto da qual será empregado. Os FPGAs são formados por um arranjo de células lógicas em um circuito integrado e permitem a implementação de uma variedade de portas lógicas básicas como AND, OR, NOR, XOR.

Sua estrutura é constituída, basicamente, por uma matriz de blocos lógicos configuráveis (*Configurable Logic Block* – CLB) que são organizados de forma bidimensional; interconexões programáveis (*switch matrix*) que permitem a comunicação entre os CLBs, este juntamente com os CLBs são programáveis pelo de acordo com as aplicações do usuário (programador); canais de roteamento, que são organizados em forma de trilhas verticais e horizontais que possuem interconexões programáveis; e por último têm-se os blocos de entrada e saída (*Input/Output Block* – IOB) dispostos nas extremidades da matriz e são responsáveis pelo interfaceamento com componentes externos (ANDRADE et al., 2006) (PEDRONI, 2010), conforme é exibido na Figura 2.

Figura 2 – Matriz de Blocos Lógicos do FPGA



Fonte: (ANDRADE et al., 2006)

3.1.3 Soft-Processor

Em sistemas embarcados, é comum a utilização de um microcontrolador ou um processador que trabalhe em conjunto com o FPGA, neste caso têm-se uma parte do sistema que é implementada e executada em *software* neste processador, com o benefício de se utilizar a abstração proporcionada pelo mesmo; e a vantagem da comunicação com a lógica programável no FPGA, para outras partes que necessitam dos benefícios proporcionados pelo *hardware*, como requisitos em tempo real.

Como exemplo de implementação com esta característica têm-se a utilização mínima de dois *chips* um contendo o processador (SoC - *System on Chip*) e outro com o próprio FPGA. Porém, há desvantagem nesta configuração, pois em casos onde necessita-se uma alta velocidade na comunicação entre o *software* e a lógica programável ou até mesmo onde é preciso dimensões físicas mínimas, esta abordagem torna-se pouco eficiente. Portanto, para contornar este problema, necessita-se que o processador e o FPGA sejam integrados no mesmo *chip*.

Uma solução que atende à necessidade supracitada é a utilização dos processadores *softcore* que são processadores sintetizados utilizando lógica programável, pois este ocupa espaço dentro do próprio chip do FPGA. Os *soft processors* possuem o benefício da alta customização, pois abordam os conceitos de computação reconfigurável descritos na Seção 3.1.1, portanto podem ser adaptáveis de modo que operem com menor frequência, ocupem um menor número de *logic elements* (elementos lógicos) ou até mesmo a utilização de recursos complexos como FPU (*Floating-point unit*) (NUNES, 2018). Exemplos de processadores *softcore* são o Nios II, da Intel/Altera, e o MicroBlaze, da Xilinx.

3.2 Algoritmo de Detecção Facial de Viola-Jones

Os pesquisadores Paul Viola e Michael Jones (VIOLA; JONES, 2001) propuseram, no ano de 2001, uma abordagem que permite a detecção de objetos em imagens. Seu funcionamento ocorre como um reconhecedor de padrões que indicam a presença de uma determinada característica. A principal abordagem de Viola e Jones foi aplicar o algoritmo no processo de detecção facial.

O método de detecção consiste na aplicação de três conceitos: imagem integral, treinamento de classificadores utilizando *boosting* e a utilização de classificadores em cascata. Objetiva-se por este algoritmo realizar uma varredura da imagem por meio de um detector, que é uma janela 24x24 pixels, que percorre toda a imagem da esquerda para direita e de cima para baixo com o intuito de encontrar as chamadas *haar features* (classificadores) que são unidades de detecção e que o conjunto destas possibilitam indicar a presença ou não de uma face. Para (ARAUJO, 2010) esta técnica possui baixo índice de falso-positivo e, portanto, uma alta taxa de acerto e baixo custo computacional.

3.2.1 *Integral Image*

A imagem integral é uma técnica proposta por Frank Crow, e introduzida por Viola e Jones em seu algoritmo. Ela consiste em realizar o somatório das intensidades dos níveis de cinza em todos os *pixels* que estejam em sub-região da imagem com o objetivo de acelerar o cálculo do valor de uma *feature* (será descrito na Seção 3.2.2).

Para realização do cálculo da integral utiliza-se a Equação 3.1:

$$ii(x, y) = \sum_{\substack{x' \leq x \\ y' \leq y}} i(x', y') \tag{3.1}$$

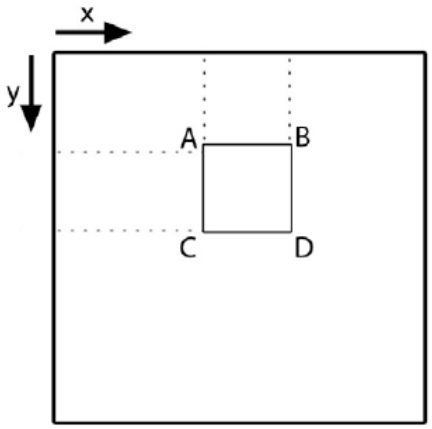
onde ii(x,y) indica a imagem integral de uma determinada coordenada (x, y) e i(x, y) a imagem original. O cálculo da imagem integral resulta em somar os valores dos *pixels* que se encontram acima de y e à esquerda de x, considerando que o ponto (0, 0) localiza-se no canto superior esquerdo de uma imagem. Sendo assim, origina-se uma tabela de varredura que contêm a mesma dimensão do *grid* da imagem e pode ser calculada em uma única execução conforme a utilização da Equação 3.2:

$$ii(x, y) = i(x, y) + ii(x - 1, y) + ii(x, y - 1) - ii(x - 1, y - 1) \tag{3.2}$$

Para evitar que o algoritmo processe uma determinada coordenada que não consta nos limites da imagem, definiu-se (x, -1) = 0 e (-1, y) = 0. Desta forma, obtêm-se a soma de área seja qual for a região retangular da imagem. Sendo assim, dada uma área retangular ABCD pertencente a uma imagem, é realizada a soma das intensidades dos *pixels* por meio da Equação 3.3. Na Figura 3 têm-se uma ilustração do funcionamento deste processo:

$$\sum_{\substack{(x,y) \\ \in ABCD}} i(x, y) = ii(A) + ii(D) - ii(B) - ii(C) \tag{3.3}$$

Figura 3 – Cálculo da Soma de Intensidade em uma Região da Matriz de *Pixels*

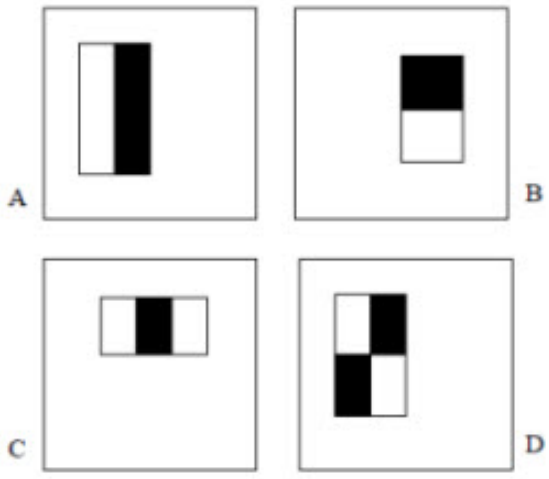


Fonte: Elaborada pelo autor

3.2.2 *Haar Features*

O procedimento de detecção de objetos classifica as imagens baseadas no valor de *features*. Estas *features* são unidades retangulares que indicam a diferença de intensidade dos *pixels* de uma determinada área. Portanto, é realizada a subtração de valores dos *pixels* de uma região em relação a outra. De forma visual, as *features* funcionam como uma máscara onde a região (em preto) indicam os *pixels* de maior intensidade e a outra (em branco) indicam os de menor intensidades. Na Figura 5 são exibidos quatro tipos de *features*.

Figura 4 – Representação Visual de Quatro Tipos de *Features*

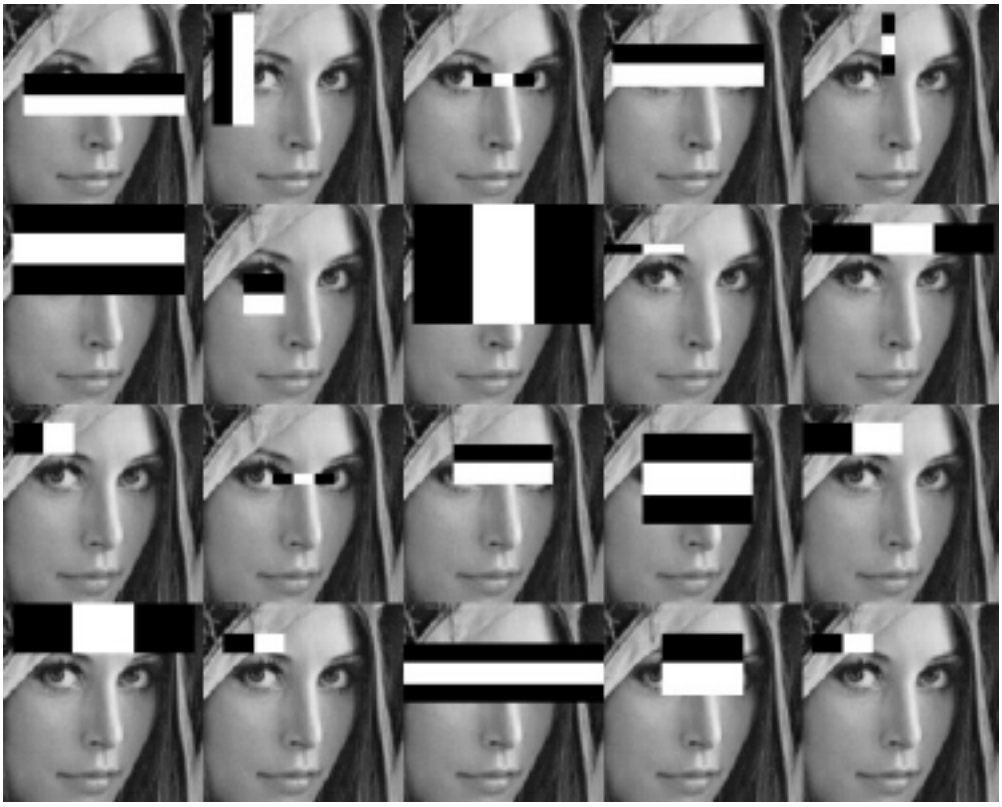


Fonte: Adaptado de (VIOLA; JONES, 2001)

Estas *features* podem ter diferentes tamanhos e posições dentro do detector, sendo assim pode haver mais de 100.000 possíveis unidades para serem testadas em uma imagem.

Cada unidade permite identificar um determinado padrão, por exemplo, no item B da Figura 5 auxilia na detecção de uma área onde há um contraste de intensidade entre uma parte superior e inferior em uma imagem, e aplicando-se na detecção facial, pode indicar a localização da região dos olhos visto que, frequentemente, a mesma é mais escura do que a região inferior (bochechas). Na Figura 5 têm-se a ilustração de algumas detecções.

Figura 5 – Ilustrações da Utilização das *Features* na Indicação de Características



Fonte: (BUKIS et al., 2011)

3.2.3 *Adaboost*

Após o cálculo da imagem integral e com a definição das *features*, é feito o treinamento de classificadores. Este processo consiste em treinar o sistema com imagens positivas e negativas, ou seja, que contém ou não contém faces. Para tal procedimento é utilizada uma variação da técnica de aprendizagem proposta por (FREUND; SCHAPIRE, 1997) intitulada *Adaboost*, seu funcionamento resulta em encontrar um classificador que seja preciso mediante a combinação linear de vários classificadores considerados fracos, ou seja, de precisão mediana, porém com uma taxa de acertos de, no mínimo, 50%. Na Equação 3.4 têm-se o cálculo da combinação supracitada:

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x) \tag{3.4}$$

onde a função que representa os classificadores fracos é dada por $h_t(x)$ e pode assumir os valores 0 ou 1 para exemplos negativos e positivos. A janela de detecção é comumente definida por 24x24 *pixels* e é representada pela variável x da Equação. O classificador fraco consiste de uma *feature* f, de um *threshold* (limiar) representado por θ e uma paridade p que indica a direção do sinal de desigualdade, como é visto na Equação 3.5:

$$h(x, f, p, \theta) = \begin{cases} 1 & : \text{ se } pf(x) < p\theta \\ 0 & : \text{ caso contrário} \end{cases} \tag{3.5}$$

A definição do classificador forte é dado pela função $H(x)$ e o peso do classificador fraco, conforme a Equação 3.6, é dado por α_t:

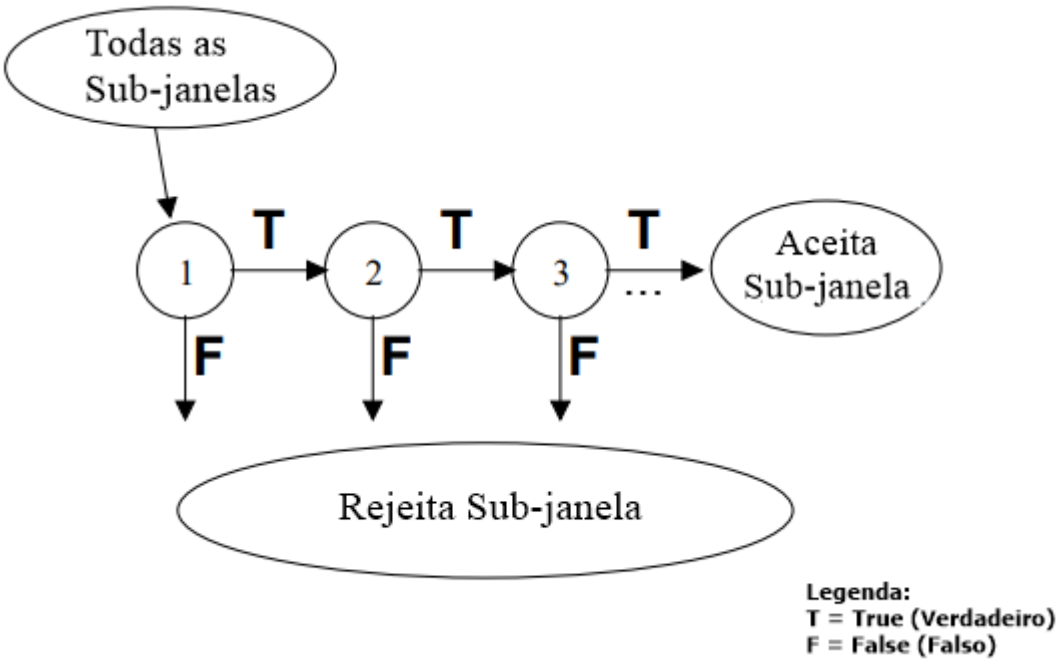
$$H(x) = \begin{cases} 1 & : f(x) \geq \frac{1}{2} \sum \alpha_t \\ 0 & : \text{caso contrário} \end{cases} \tag{3.6}$$

O algoritmo *Adaboost* atua na escolha de uma *feature* mais adequada ao tipo de detecção que se objetiva e também permite treinar os classificadores com esta característica, permitindo que o tempo de execução seja reduzido devido os milhares de opções possíveis de uma *feature* em uma janela de detecção.

3.2.4 *Cascade Classifier*

A última etapa do algoritmo de Viola e Jones é a combinação, em cascata, dos classificadores fortes. Esta tarefa possibilita, de forma eficiente, a busca de um padrão na imagem. A cascata é dividida em estágios, em cada uma são aplicados um classificador que seja mais específico e complexo que o anterior. Isto permite que o algoritmo realize a rejeição de regiões da imagem, de forma ágil, e que sejam muito diferentes da característica procurada. Quando uma rejeição ocorre, há o término do processo de procura de padrões, evitando que estágios posteriores sejam executados de forma desnecessária. Nos primeiros estágios, normalmente, são descartados elementos da imagem como planos de fundo e afins, já nos estágios mais avançados são mantidos elementos que possuem alta probabilidade de ser uma face. Quanto mais se avança nas etapas mais exaustiva é a análise, e menor são os elementos que serão processados. As subimagens que forem aceitas no último estágio de processamento, ou seja rotulada de maneira positiva pelos classificadores da cascata, serão consideradas faces conforme mostra a Figura 6. Na Figura 7 têm-se uma ilustração do processo de varredura em busca de padrões na imagem.

Figura 6 – Classificador em Cascata



Fonte: Adaptado de (VIOLA; JONES, 2001)

Figura 7 – Ilustração do Processo de Busca por Padrões na Imagem

Fonte: (BRAGA et al., 2013)

4 DESENVOLVIMENTO, IMPLEMENTAÇÃO E EXECUÇÕES

Neste Capítulo será apresentado o desenvolvimento do sistema, começando pela utilização da Terasic DE0-NANO como primeiro protótipo, a integração da placa ao monitor de vídeo por meio de um circuito e também a inserção dos componentes que permitem habilitar o dispositivo para execução de algoritmos de alto-nível.

Posteriormente será demonstrada a utilização da biblioteca do OpenCV para realização de testes de detecção de face e olhos, bem como a execução realizada no computador, com a utilização do algoritmo *Cascade Classification* e apresentação dos resultados com a devida demarcação destes elementos.

Como etapa seguinte serão descritas a implementação, com a utilização da Terasic DE2-70, como a inclusão dos componentes que irão compor os subsistemas responsáveis pela execução de algoritmos de alto-nível por meio do processador Nios, exibição das imagens processadas em dispositivo externo (monitor) e susbistema responsável pela captura e adequação das imagens obtidas da câmera Terasic TRDB-D5M.

Ao final serão expostos os resultados do processo de detecção facial e o tempo de execução nos cenários como a utilização de imagens previamente armazenadas; identificação de face em imagens obtidas por meio da câmera em situações como a utilização de óculos, sem a utilização do mesmo, rosto em posição inclinada e em posição normal.

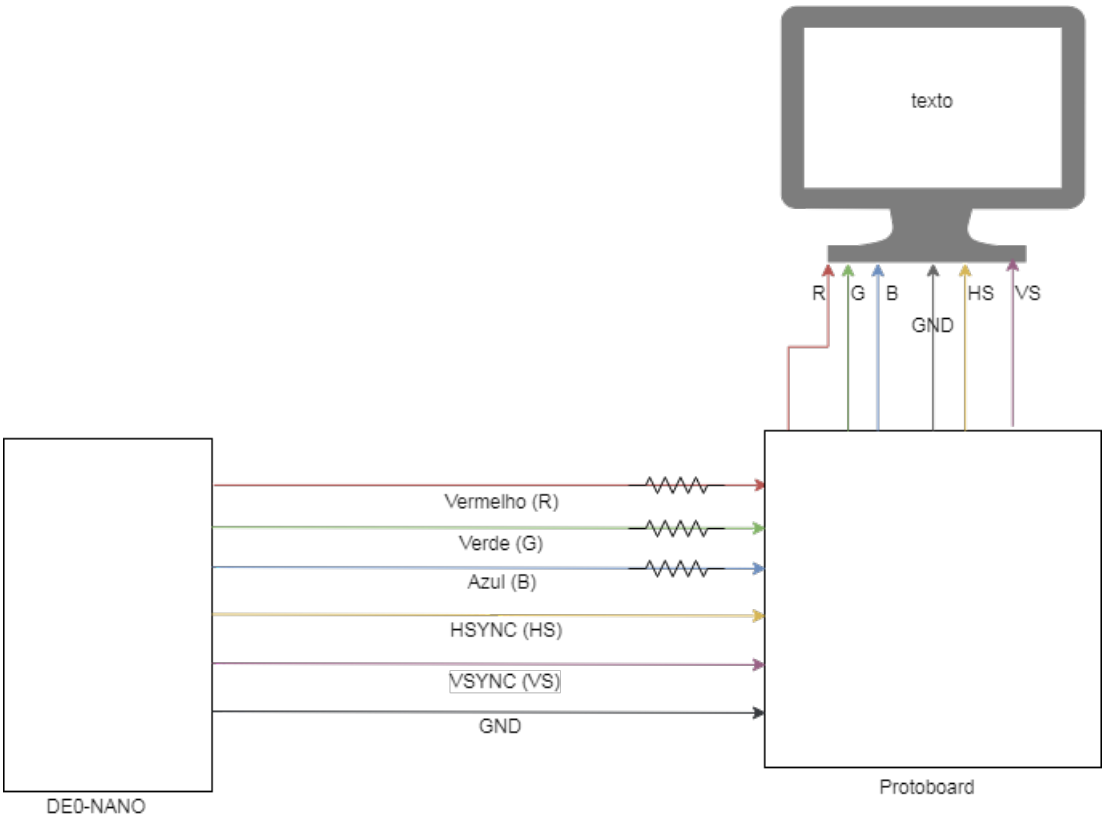
4.1 Terasic DE0-NANO

Na primeira etapa de desenvolvimento foi realizada a integração de um dispositivo reconfigurável, que neste trabalho é a placa DE0-NANO da Terasic (TERASIC, 2013), conectada a um monitor, com o objetivo de testar a saída das imagens processadas pelo *hardware*. Porém a DE0-NANO não possui uma interface padrão de saída como VGA ou HDMI, entretanto este dispositivo dispõe de conexões por meio de pinos, identificados como GPIO, que possuem sua funcionalidade configurada pelo desenvolvedor. Sendo assim, foi necessário implementar um circuito para possibilitar a conexão com o VGA (*Video Graphics Array*).

Para o desenvolvimento desta atividade fez-se necessário a adição de alguns componentes para sua execução. Além do próprio *hardware* reconfigurável (DE0-NANO), têm-se uma *protoboard*, fios para fazer a conexão da placa com os componentes, três resistores de 1k, um cabo no padrão VGA e um monitor compatível com este padrão. Para o pro-

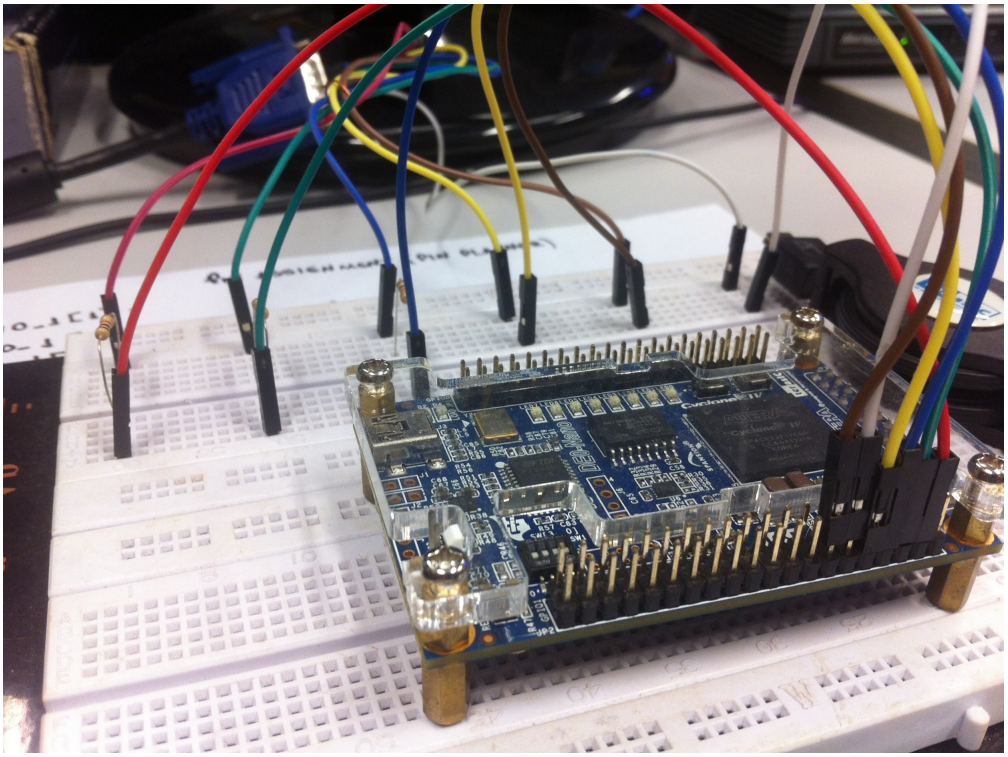
cesso de implementação foi utilizado o *software* Quartus II 64 bit Version 15.0.0 Web Edition. Na Figura 8 têm-se a ilustração do circuito e nas Figuras 9, 10 e 11 os registros do protótipo já com as devidas conexões.

Figura 8 – Visão Geral - Protótipo



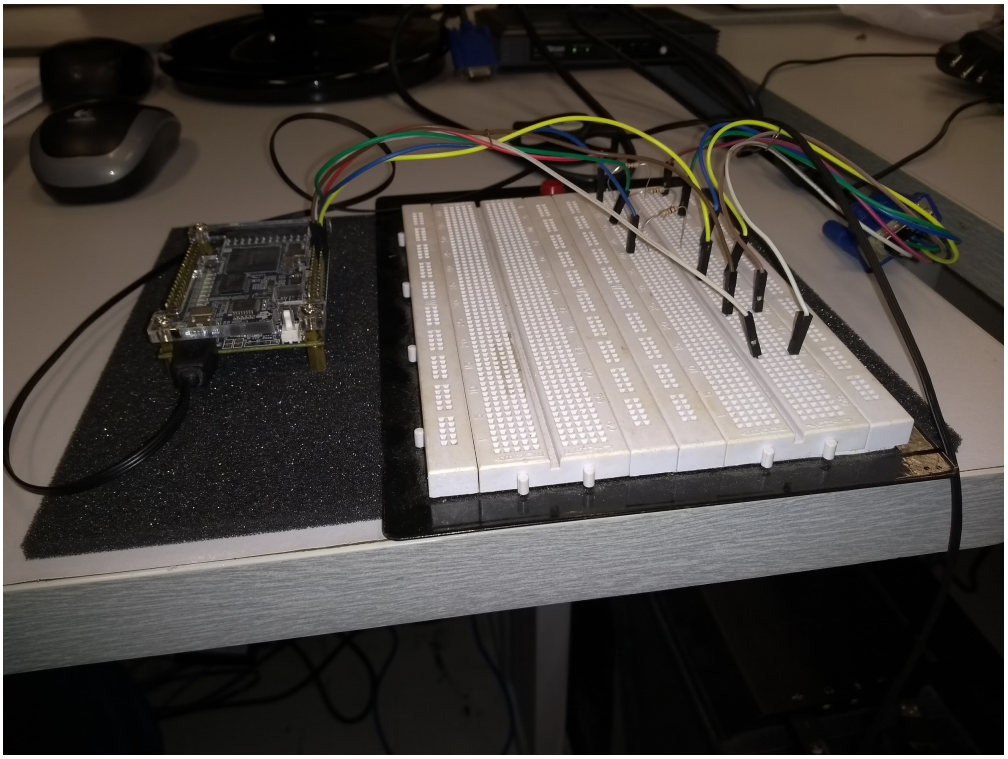
Fonte: Elaborada pelo autor

Figura 9 – DE0-NANO, Protoboard e Conexões



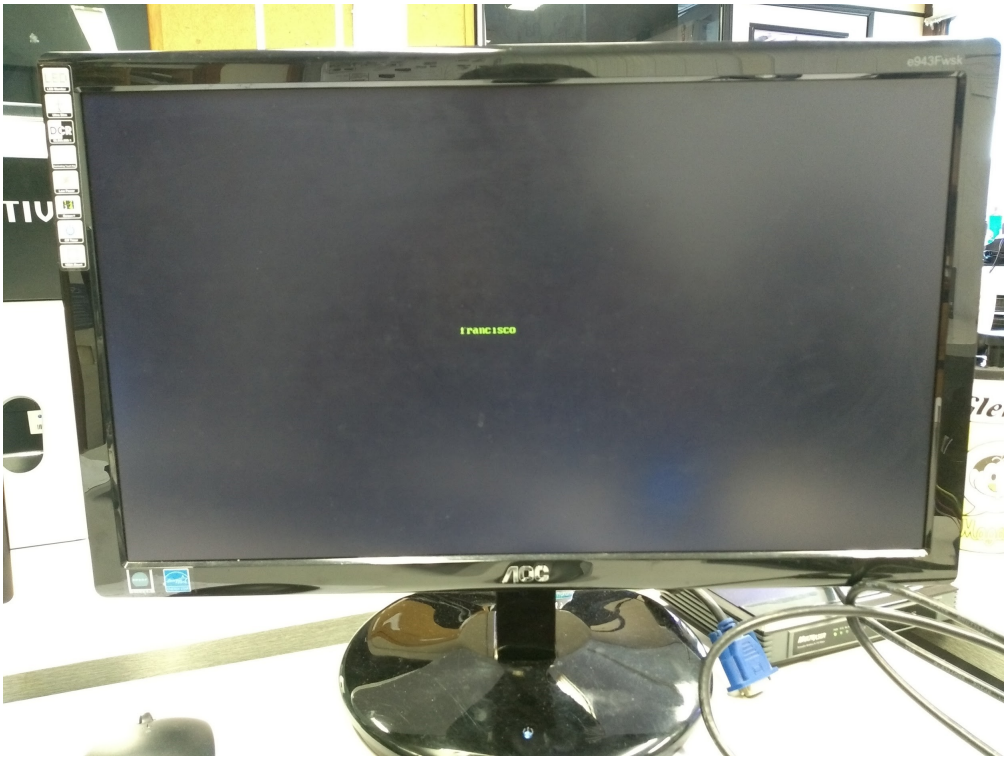
Fonte: Elaborada pelo autor

Figura 10 – DE0-NANO, Protoboard e Conexões



Fonte: Elaborada pelo autor

Figura 11 – Exibição de Texto no Monitor



Fonte: Elaborada pelo autor

Para o devido funcionamento do dispositivo DE0-NANO com os componentes externos é necessário que sejam configurados os pinos GPIO para que estejam habilitados ao funcionamento desejado, para tal processo o *software* Quartus possui uma ferramenta intitulada *Pin Planner* que permite que todos os pinos definidos na programação sejam vinculados aos componentes físicos da placa. Na Figura 12 apresenta a configuração realizada nesta etapa de desenvolvimento. Destacam nesta Figura os pinos responsáveis pela conexão dos componentes RGB (*Red*, *Green* e *Blue*), HSYNC (*Vertical Synchronization*) e VSYNC (*Horizontal Synchronization*):

Figura 12 – Configuração dos Pinos da Placa

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard
B	Output	PIN_T12	4	B4_N0	PIN_T12	2.5 V
CLOCK_50	Input	PIN_R8	3	B3_N0	PIN_R8	3.3-V LVTTL
G	Output	PIN_T13	4	B4_N0	PIN_T13	2.5 V
H_SYNC	Output	PIN_T11	4	B4_N0	PIN_T11	2.5 V
R	Output	PIN_T15	4	B4_N0	PIN_T15	2.5 V
Reset	Input	PIN_J15	5	B5_N0	PIN_J15	2.5 V
V_SYNC	Output	PIN_R11	4	B4_N0	PIN_R11	2.5 V

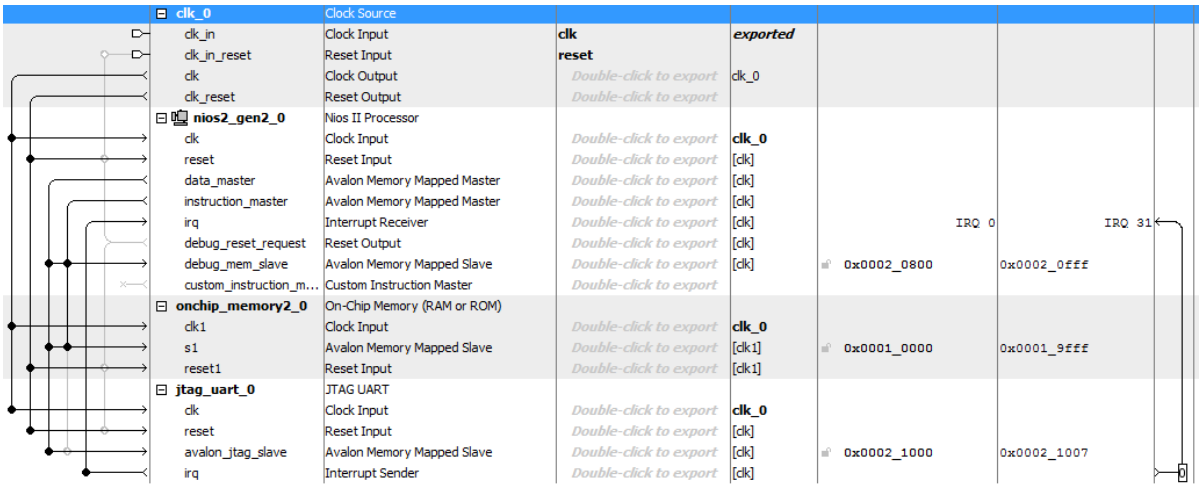
Fonte: Elaborada pelo autor

Posteriormente foi realizada a configuração e implantação do Nios para execução de algoritmos que serão utilizados no processamento de imagens. Para tal desenvolvimento, foi utilizado, como referência, o documento *Bootable Embedded Systems for the DE0-*

NANO Board (INTEL/ALTERA, 2013), sendo assim, foi desenvolvido, com as devidas adaptações, o sistema embarcado que permitirá o correto funcionamento do sistema.

Foi implementado um sistema simples, do tipo *hello world*, para verificação do seu comportamento, sendo assim resultou-se no correto funcionamento do mesmo. Na Figura 13 é visto o sistema desenvolvido no QSYS com as devidas conexões entre seus componentes.

Figura 13 – Componentes do Sistema - QSYS

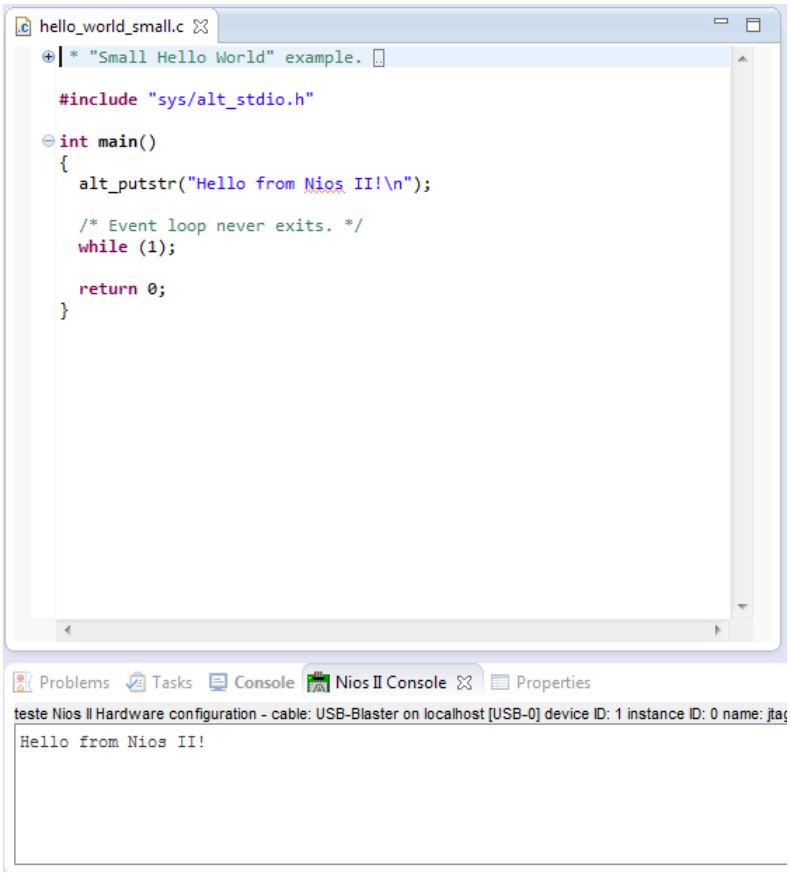


Fonte: Elaborada pelo autor

Nesta etapa somente os componentes **clk_0**, **nios2_gen2_0**, **onchip_memory2_0** e **jtag_uart_0** foram inseridos. Basicamente, estes componentes são necessários para geração de *clock*, **clk_0**; configuração e customização do processador, **nios2_gen2_0**; uma memória *onboard* (RAM-*Random Access Memory*), **onchip_memory2_0**; e uma interface para comunicação via porta *serial*, **jtag_uart_0**.

Na Figura 14 têm-se o código fonte, na linguagem C, e a saída do programa exibindo uma simples *String*.

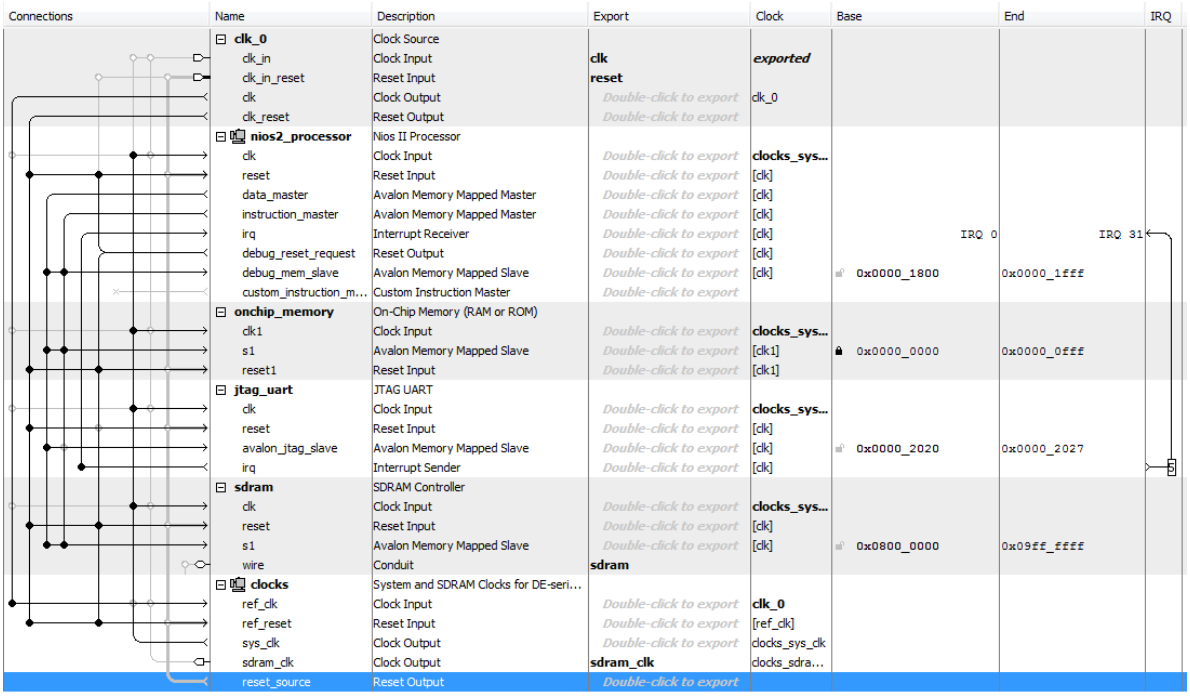
Figura 14 – Código-Fonte e Execução do Algoritmo Teste



Fonte: Elaborada pelo autor

Porém, devido a limitações de capacidade do componente **onchip_memory2_0**, foi proposto a utilização da **SDRAM** (*Synchronous dynamic random access memory*) por esta comportar maior quantidade de dados, sendo assim, uma nova versão deste sistema foi desenvolvida assumindo a integração deste componente e também foi realizada refatoração de código. Na Figura 15 têm-se o console do QSYS com o novo componente.

Figura 15 – Inclusão da Memória SDRAM



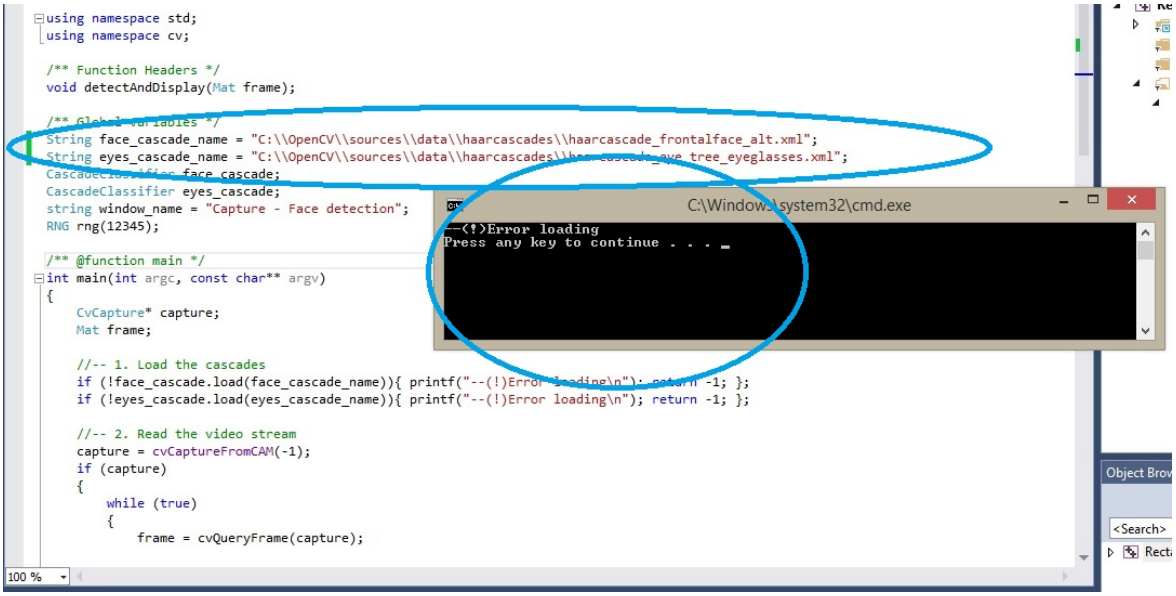
Fonte: Elaborada pelo autor

4.2 Testes com a biblioteca OpenCV

Com o objetivo de entender o processo de detecção de características humanas, optou-se pela utilização do OpenCV (versão 3.0) devido à sua biblioteca que facilita as operações de manipulação das imagens e processamento das mesmas. Concentrou-se na detecção de faces, objetivo deste trabalho, e olhos, e para tal foi utilizado o algoritmo *Cascade Classification* (OPENCV, 2018), pertencente a biblioteca. Foram realizados testes no computador (PC) juntamente com uma câmera e posterior auxílio na implementação no FPGA. É ressaltado que a biblioteca OpenCV não será incorporada ao FPGA.

Porém, a execução deste algoritmo, no estado em que se encontra, não foi bem-sucedida, pois ocorreram erros de leitura dos arquivos (em formato .xml) que contêm padrões pré reconhecidos de face e olhos, estes são necessários para o processo de detecção. Sendo assim, para correção deste problema *Error Loading* (*cascades*) conforme Figura 16, ajustes no código-fonte foram necessários como alterações nas atribuições das seguintes *strings* **face_cascade_name** e **eyes_cascade_name** que passaram a receber o caminho completo, com a utilização de barras invertidas (\\) para separação dos diretórios, dos arquivos **haarcascade_frontalface_alt.xml** e **haarcascade_eye_tree_eyeglasses.xml**.

Figura 16 – Error Loading (*Cascades*)



Fonte: Elaborada pelo autor

Ainda com a impossibilidade do processo de detecção devido a não captura das imagens da câmera, outras alterações no código-fonte foram requeridas, pois o algoritmo fornecido (OPENCV, 2018) é compatível apenas com versões 2.4.X do OpenCV e neste trabalho utilizou-se a versão 3.0. Portanto adequou-se os seguintes trechos de códigos que são responsáveis, respectivamente, pela captura de imagens como a criação do objeto do tipo **VideoCapture**, teste de início de transmissão e aquisição de *frames* (quadros) que serão utilizados para detecção. No Quadro 4 são exibidas essas alterações:

Quadro 4 – Alterações no Código-fonte do *Cascade Classifier*

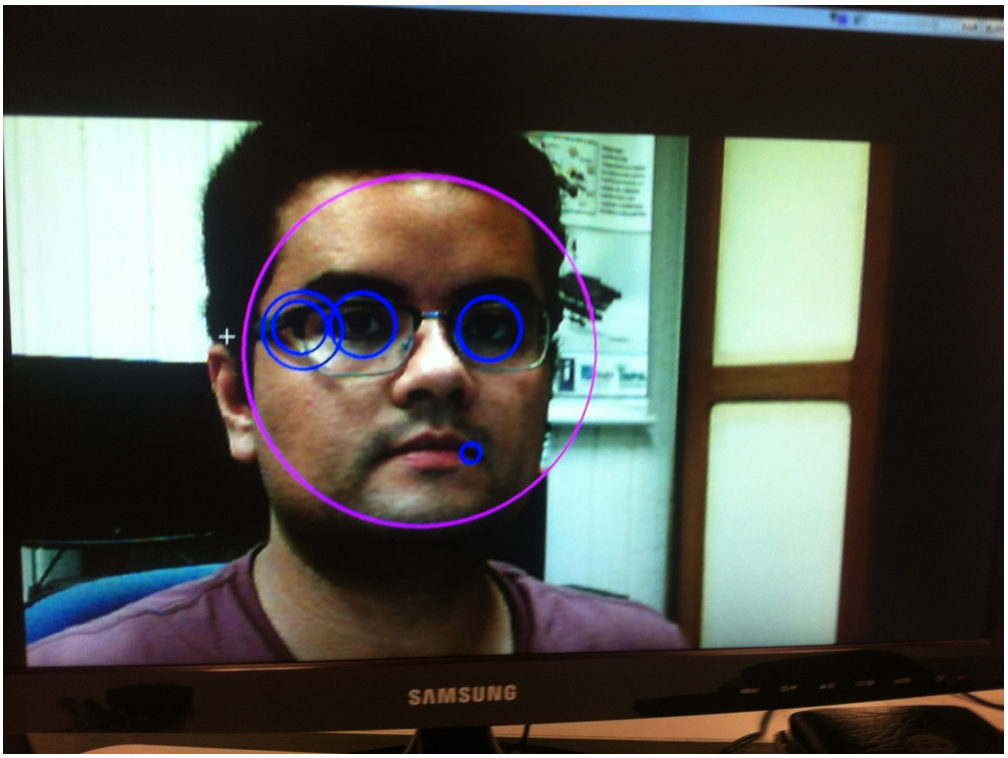
Original	Alterado para
CvCapture* capture;	VideoCapture capture(-1);
capture = cvCaptureFromCAM(-1); e if (capture);	if (capture.isOpened();
frame = cvQueryFrame(capture);	capture.read(frame);

Fonte: Elaborada pelo autor

Posteriormente, foram realizados testes de execução para verificar o comportamento do algoritmo em alguns cenários como

- Identificação de face e olhos, com a utilização de óculos e olhando para câmera (Figura 17);
- Identificação de face e olhos, sem a utilização de óculos e olhando para câmera (Figura 18);
- Identificação de face e olhos, com a utilização de óculos e sem olhar para câmera (Figura 19);

Figura 17 – Identificação de Face e Olhos, com a Utilização de Óculos e Observando a Câmera



Fonte: Elaborada pelo autor

Figura 18 – Identificação de Face e Olhos, sem a Utilização de Óculos e Observando a Câmera

Fonte: Elaborada pelo autor

Figura 19 – Identificação de Face e Olhos, com a Utilização de Óculos e sem Observar a Câmera

Fonte: Elaborada pelo autor

Obteve-se, como esperado, o êxito das execuções com a demarcação das áreas em que se designam as detecções. Após o resultado desta etapa, foram realizados testes com a cabeça inclinada e também a implementado do tracejo de uma reta entre os centros dos círculos dos olhos para destacar o ângulo desta inclinação. Portanto, a Figura 20 exibe o resultado das atividades em questão.

Para inserção da reta que fica localizada entre os pontos centrais dos olhos, foi necessário limitar a quantidade de círculos que indicam a posição dos olhos somente para dois (antes eram de acordo com as detecções). Para isso foi alterado o laço **for** para que este percorra somente a referência de um olho e a adição do outro laço **for** para percorrer a do outro olho. Em cada laço **for** foi inserido um objeto do tipo **Point** para armazenar o centro dos círculos, e no segundo laço **for** cria-se a linha, tendo como dois dos parâmetros as localizações dos centros dos círculos. Na Figura 21 são exibidos os resultados destas alterações.

Figura 20 – Identificação de Face e Olhos, sem a Utilização de Óculos e com o Rosto Inclinado

Fonte: Elaborada pelo autor

Figura 21 – Alterações nos Laços **for** Referente ao Desenho da Reta Entre os Olhos

```
for (size_t j = 0; j < eyes.size(); j++)
{
        Point center(faces[i].x + eyes[j].x + eyes[j].width*0.5, faces[i].y + eyes[j].y + eyes[j].height*0.5);
        int radius = cvRound((eyes[j].width + eyes[j].height)*0.25);
        circle(frame, center, radius, Scalar(255, 0, 0), 4, 8, 0);
        Point a(center.x, center.y);
        line(frame, a, a, Scalar(110, 220, 0), 2, 8);
        for (size_t k = 1; k < eyes.size(); k++)
        {
                Point center(faces[i].x + eyes[k].x + eyes[k].width*0.5, faces[i].y + eyes[k].y + eyes[k].height*0.5);
                int radius = cvRound((eyes[k].width + eyes[k].height)*0.25);
                circle(frame, center, radius, Scalar(255, 0, 0), 4, 8, 0);
                Point b(center.x, center.y);
                line(frame, a, b, Scalar(110, 220, 0), 2, 8);
        }

}
```

Fonte: Elaborada pelo autor

Após algumas execuções do algoritmo, mesmo com as detecções esperadas, ocorria o erro *debug assertion failed* que interrompia a execução do mesmo. Isto aconteceu por causa do acesso a uma posição inexistente do vetor **eyes**. Para correção deste erro foi necessário mudar os laços **for**, que percorrem cada posição do vetor **eyes**, com a condição

de parada **j < eyes.size()** e **k < eyes.size()** respectivamente aos laços, sendo assim, não há acesso a uma posição maior do que o tamanho do vetor supracitado. Foram realizadas novas execuções do programa, novamente para verificar seu comportamento na detecção de face nos sentidos inclinado e normal, detecção dos olhos e *tracking* (rastreamento) destes itens, bem como a reta que interliga os centros dos olhos, observou-se a inexistência do erro em questão juntamente com as corretas detecções.

4.3 Terasic DE2-70

Neste trabalho utilizou-se a placa DE2-70 da Terasic, ver Figura 22. Este dispositivo contém recursos que podem ser utilizados em diversos tipos de aplicações, como o desenvolvimento de simples circuitos até mesmo projetos complexos como de cunho multimídia, processamento de imagem, entre outras aplicações que são utilizadas conforme necessidade de projetistas de sistemas embarcados. A placa DE2-70 possui o chip FPGA Cyclone®II 2C70 da Intel/Altera que possibilita o desenvolvimento de aplicações embarcadas nos moldes dos conceitos descritos nas seções 3.1.1 (computação reconfigurável) e 3.1.2 (FPGA), portando oferece os seguintes componentes (TERASIC, 2008):

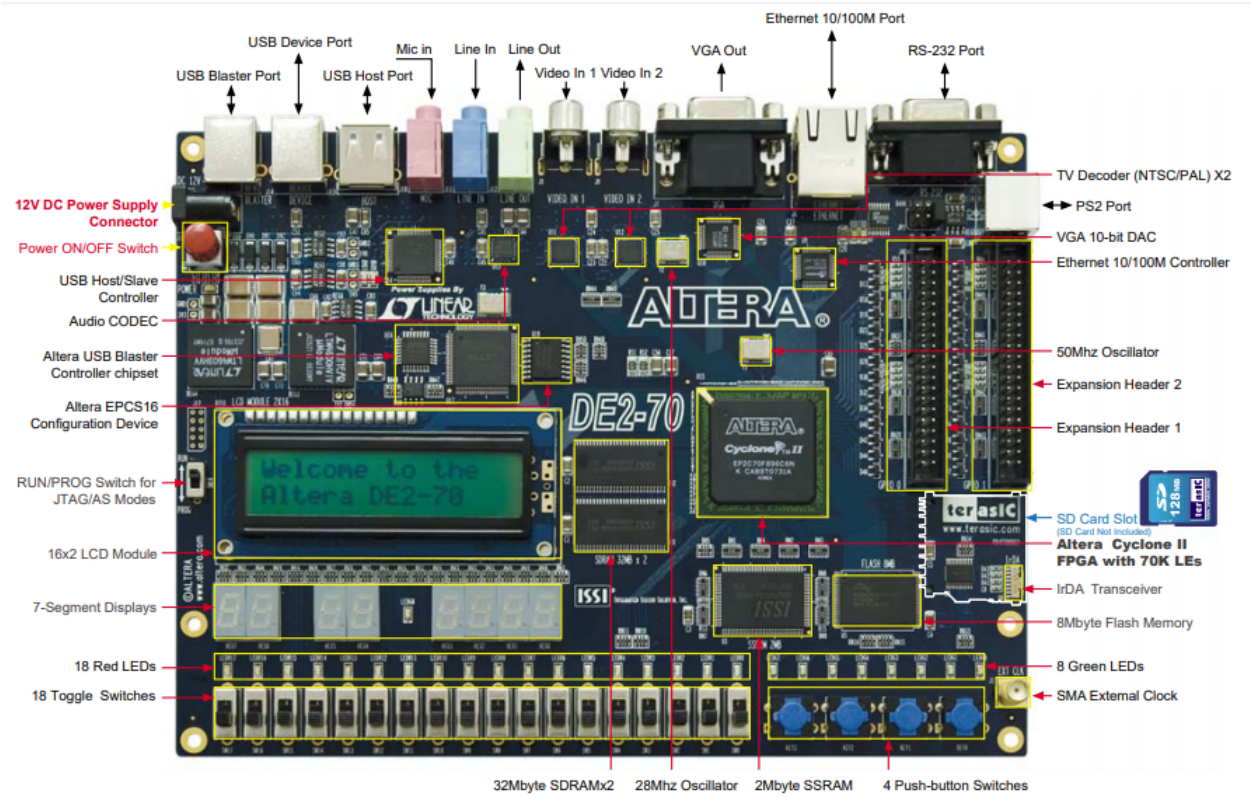
- 68.416 LE (elementos lógicos);
- 250 blocos RAM M4K;
- 1.152.000 *bits* de memória RAM;
- 150 multiplicadores embarcados;
- 4 PLLs (*Phase-Locked Loop*);
- 622 pinos de E/S (entrada e saída) disponíveis para o usuário; e
- Pacote FineLine BGA de 896 pinos.

Outros componentes de *hardware* da Terasic DE2-70, que trabalham juntamente com o FPGA, são descritos a seguir e fornecem recursos as mais diversas aplicações:

- Altera/Intel *Serial Configuration* - EPCS16;
- USB Blaster (*onboard*) para programação do dispositivo, *debug* (depuração via JTAG - *Joint Test Action Group*) e comunicação *serial*;
- 2 *MByte* de memória SSRAM (*Synchronous Static RAM*);
- Duas memórias SDRAM (*Synchronous Dynamic RAM*) de 32MBytes cada;

- 8 *MByte* de memória *flash*;
- Suporte para cartão de memória SD (*Secure Digital Card*);
- 4 botões pressionáveis;
- 18 interruptores;
- 18 LEDs (*Light Emitting Diode*) vermelhos;
- 9 LEDs (*Light Emitting Diode*) verde;
- um oscilador de 50MHz e outro de 28,63MHz para fonte de *clock* (relógio);
- *Codec* de 24 *bits* com conectores de entrada e saída;
- Controlador VGA com conector;
- 2 decodificadores para os sistemas de cores (NTSC (*National Television System(s) Committee*), PAL (*Phase Alternating Line*) e SECAM (*Séquentiel Couleur à Mémoire*)) e conector de entrada para TV;
- Controlador ETHERNET 10/100 *megabits*/seg com conector padrão RJ-45;
- Controlador USB (*Universal Serial Bus*) com conector;
- Transceptor RS-232 e conector de 9 pinos;
- Conector PS/2 para conexão de *mouse* e teclado;
- Transceptor infravermelho (IrDa);
- 1 Conector SMA (SubMiniature *version* A); e
- Dois expansores de 40 pinos com proteção por diodo.

Figura 22 – Visão Geral dos Componentes de *Hardware* da Placa Tesasic DE2-70



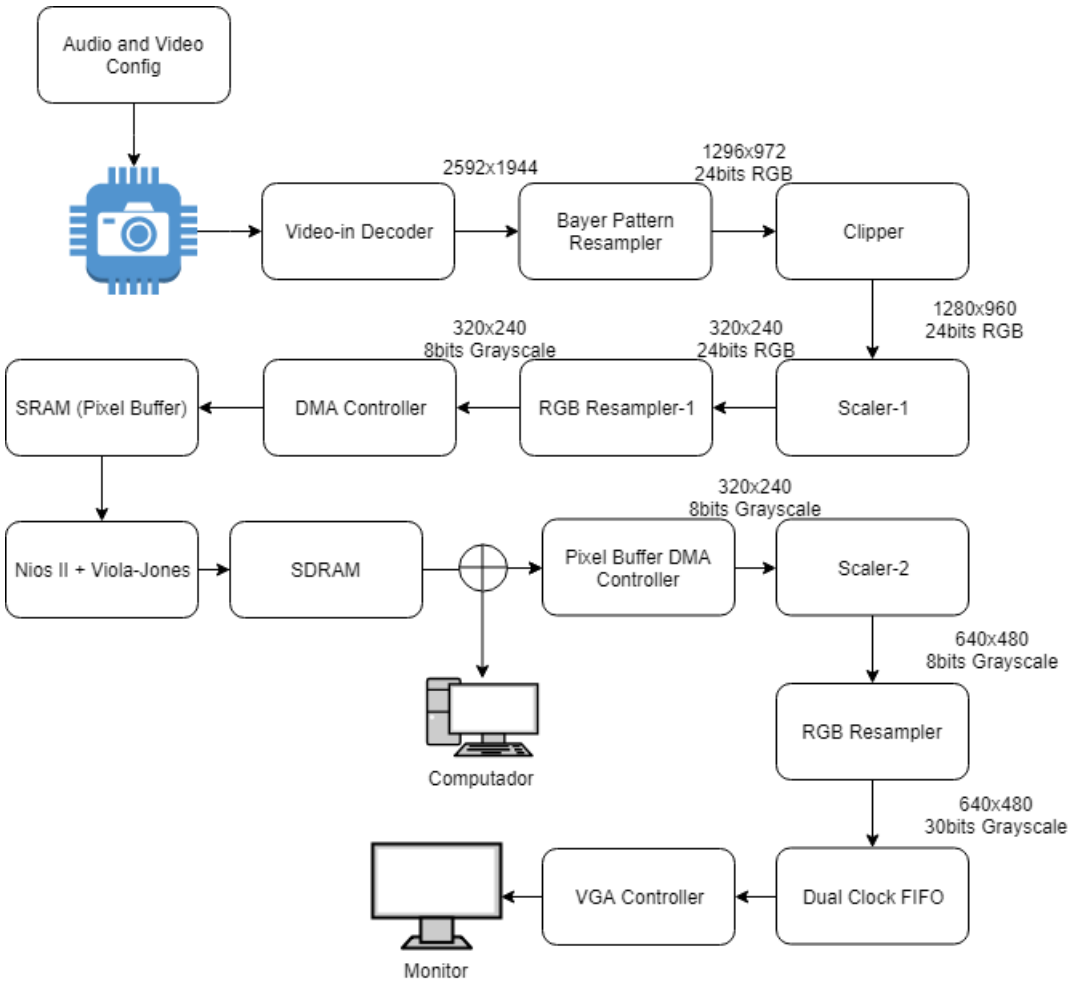
Fonte: (TERASIC, 2008)

Para o desenvolvimento do sistema, optou-se por subdividi-lo em três subsistemas. Na Seção 4.3.1 serão demonstrados a inserção dos componentes que permitirão a execução do algoritmo de Viola-Jones (C/C++), dentre estes destacam-se o fornecedor de *clock* (**Clock Source**) para o sistema, o **Nios II Processor**, memórias (**On-chip Memory (RAM or ROM)** e **SDRAM**) e conexão serial (**JTAG UART**).

Na Seção 4.3.2 serão detalhados os componentes e suas configurações que permitirão a conexão da placa ao monitor de vídeo. Sendo assim, é necessário a utilização do **Pixel Buffer DMA Controller**, **VGA Controller**, **SRAM/SSRAM Controller**, entre outros.

E por último, na Seção 4.3.3, serão apresentados os componentes que irão habilitar a conexão do FPGA à câmera, permitindo a transmissão dos dados de imagens e adequação dos mesmos aos padrões requeridos pelo sistema. Sendo assim, dentre os componentes necessários ao funcionamento do subsistema temos o **Audio and Video Config**, **Video-In Decoder**, **Scaler** e **Clipper**. Na Figura 23 têm-se o fluxograma que mostra uma visão geral do funcionamento destes componentes.

Figura 23 – Fluxograma do Sistema



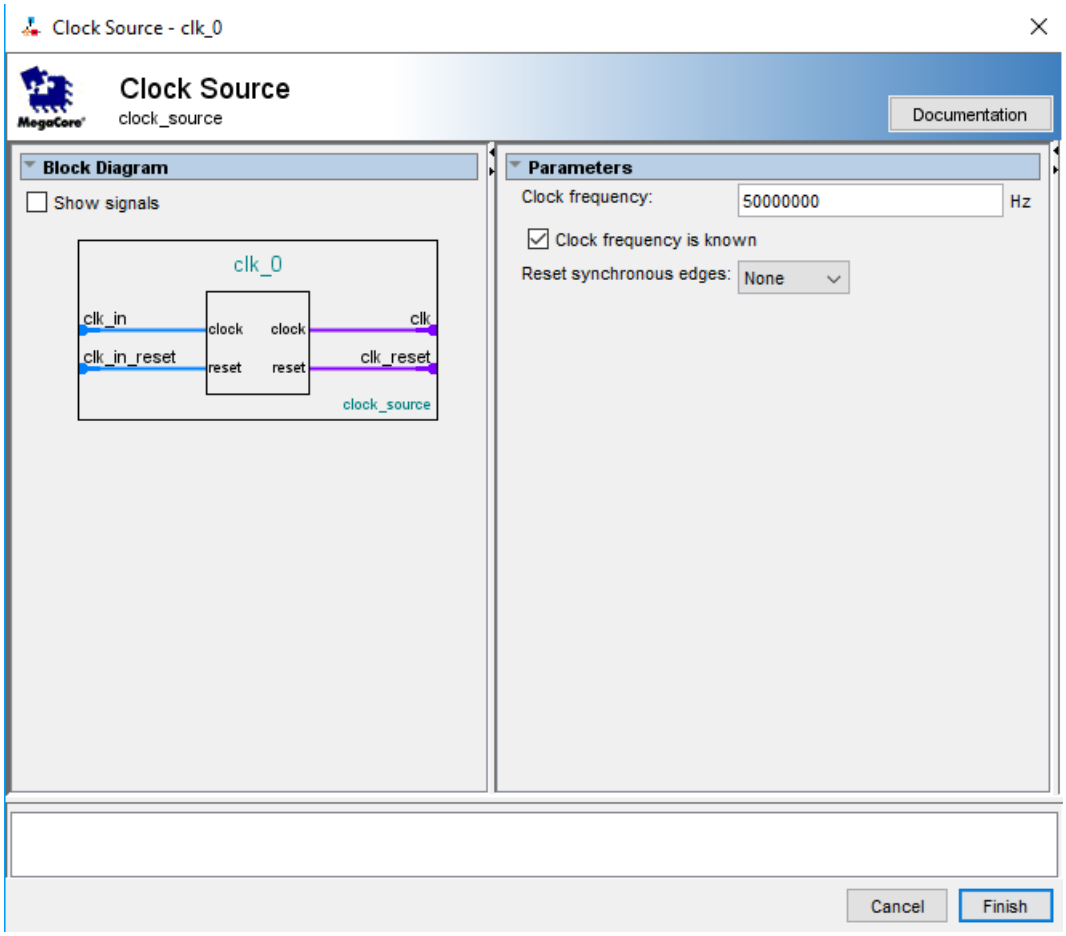
Fonte: Elaborada pelo autor

4.3.1 Desenvolvimento do subsistema para Execução de Algoritmos por meio do Processador *Softcore* Nios II

Nesta primeira etapa do trabalho foi realizada a criação de um novo projeto no *software* Quartus 13.0, no mesmo foram definidas configurações como nome do projeto e o dispositivo a ser utilizado, que neste caso é a placa DE2-70 da qual será programado pela ferramenta. Após a criação do projeto, foi utilizado o aplicativo QSYS para inserção dos componentes que farão parte do sistema e que permitirá a execução de algoritmos de alto-nível como C/C++ pelo mesmo. Os seguintes componentes foram requeridos e, portanto, inseridos ao sistema, juntamente com as devidas configurações e conexões

- **Clock Source (clk_0)**: Este componente fornece o clock de 50MHz para o sistema. Na Figura 24 têm-se a inclusão e configuração do mesmo.

Figura 24 – QSYS: Inclusão do Componente **Clock Source**



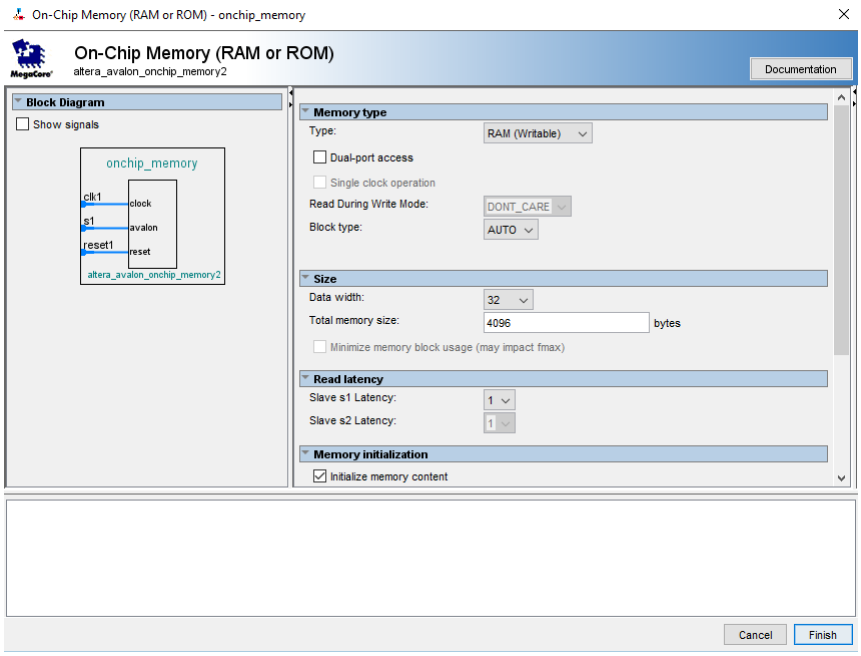
Fonte: Elaborada pelo autor

- **On-chip Memory (RAM or ROM) (onchip_memory)**: Este componente funciona como uma memória de acesso randômico (*random access memory*) para o sistema, sendo assim configurada para tal no parâmetro *Memory Type*. Portanto, possibilita armazenar o programa, previamente implementado utilizando o Nios II *Software Build Tools for Eclipse* (SBT), e conteúdo das variáveis. Seu tamanho foi definido para 4.096 *bytes*, porém possui limite para armazenar até 144.000 *bytes* de conteúdo. Foram testados diferentes valores para o **onchip_memory**, porém não se obteve melhorias de desempenho. A Figura 25 mostra esta configuração.

- **Nios II Processor (nios2_processor)**: por meio deste processador serão executados os algoritmos desenvolvidos em C/C++, este opera com *clock* de 50MHz conforme o oscilador físico fornecido pelo **Clock Source**. É utilizada a versão *fast* (rápida) do Nios, pois esta possui maior desempenho se comparado com as versões *standard* (padrão) e *economy* (econômica). Também foram configurados os parâmetros *Reset vector memory* e *Exception vector memory* para utilização da memória do FPGA (**On-chip memory**), respectivamente nos endereços de memória 0x00000000

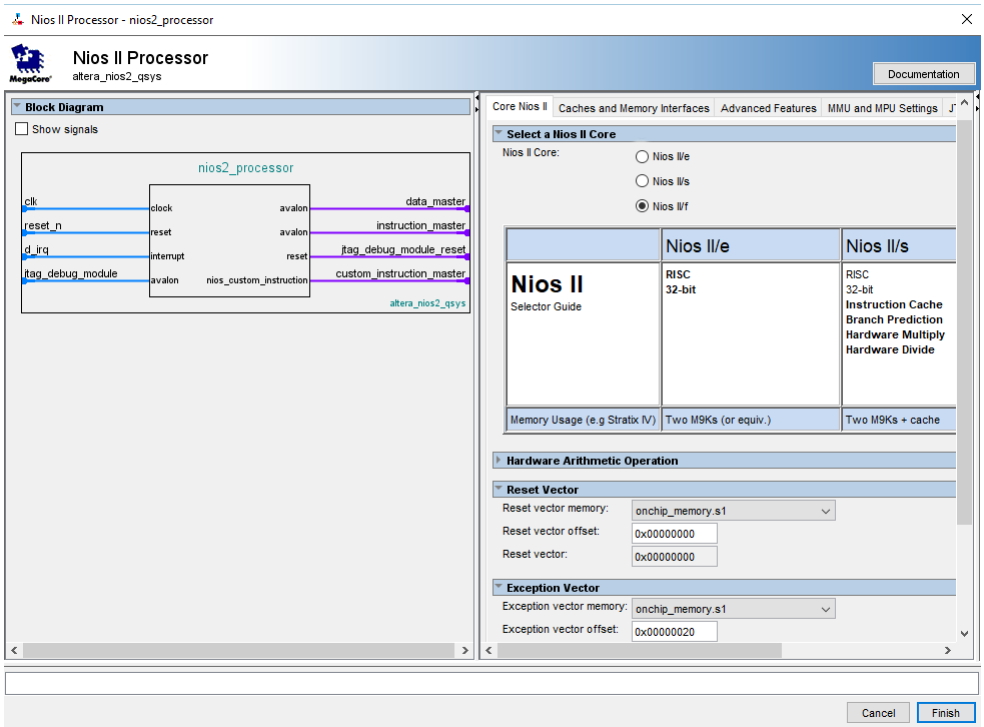
e 0x00000020, onde serão realizados a leitura e escrita de dados e instruções, conforme indica a Figura 26.

Figura 25 – QSYS: Inclusão do Componente *On-chip Memory*



Fonte: Elaborada pelo autor

Figura 26 – QSYS: Inclusão do Componente **Nios Processor**

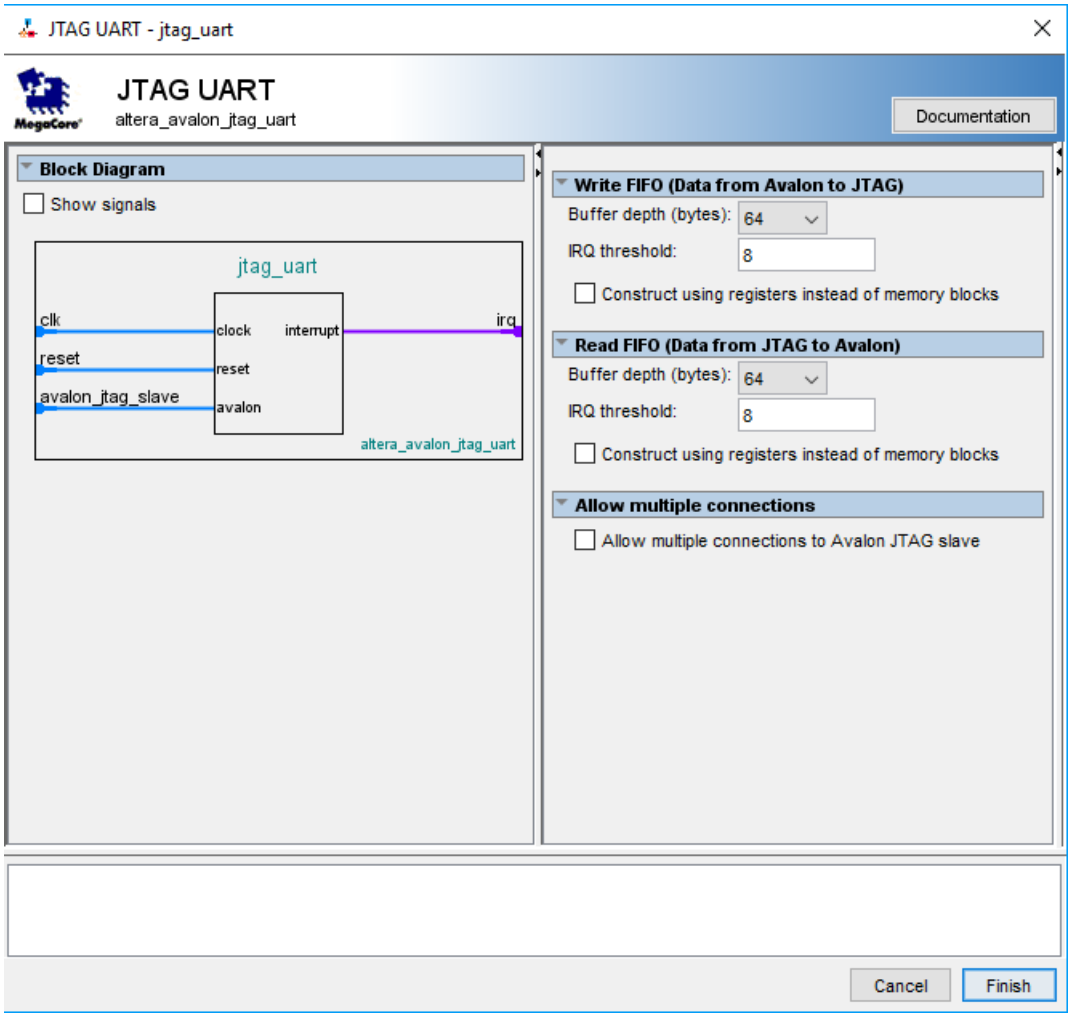


Fonte: Elaborada pelo autor

- **JTAG UART (jtag_uart)**: Este componente proporciona a comunicação serial

(*Universal Serial Bus*) entre o computador e a Terasic DE2-70 (USB blaster). Sendo assim, permite a transmissão do programa desenvolvido no computador, para a memória da placa. Também possibilita ao computador monitorar e controlar o processador Nios, como iniciar e parar sua execução, bem como a realização de depuração (*debug*), como mostra a Figura 27.

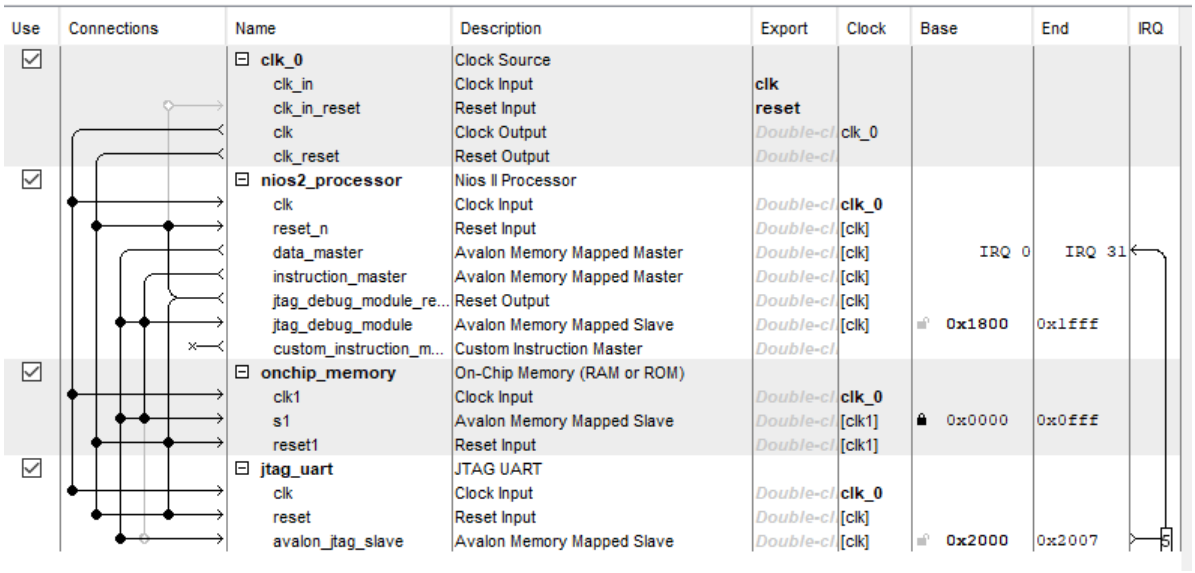
Figura 27 – QSYS: Inclusão do Componente **JTAG UART**



Fonte: Elaborada pelo autor

Após a inclusão de todos os componentes a serem utilizados pelo sistema, é necessário a atribuição das conexões dentre os mesmos conforme apresenta a Figura 28, sem este procedimento não há transmissão de instruções, dados e nem o a detecção da memória pelo processador. Após a modelagem do sistema é necessário definir os endereços dos componentes do mesmo, para tal ação seleciona-se a opção *Assign Base Addresses* no menu *System* do QSYS, isto é necessário para que não ocorra conflitos de endereçamento. Na coluna *Base* (Figura 28) observa-se os endereços atribuídos aos componentes **Nios II Processor**, **On-chip Memory (RAM or ROM)** e **JTAG UART**.

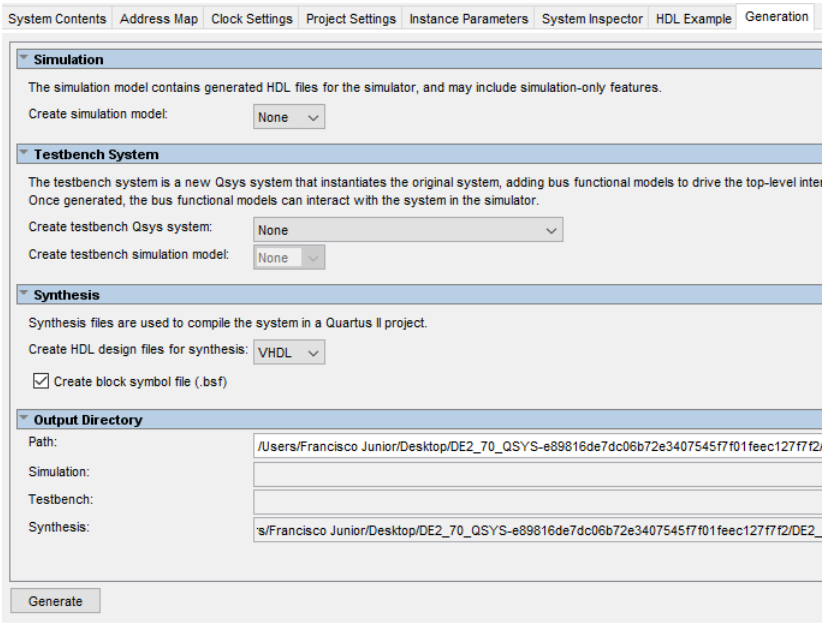
Figura 28 – QSYS: Visão Geral do Sistema com as Conexões



Fonte: Elaborada pelo autor

Após a finalização das atribuições deve-se salvar o sistema, que criará um arquivo com a extensão .qsys. Nas etapas descritas anteriormente, foram realizados os processos de modelagem do sistema, porém para utilização do mesmo no *software* Quartus é necessário o processo de geração (*Generation*). Desta forma serão criados os arquivos a serem utilizados na síntese do projeto, na pasta indicada em *Output Directory*, para posteriormente serem compilados no Quartus. Para realização deste procedimento bastam selecionar a aba *Generation* e o botão *Generate*, conforme é exibido na Figura 29.

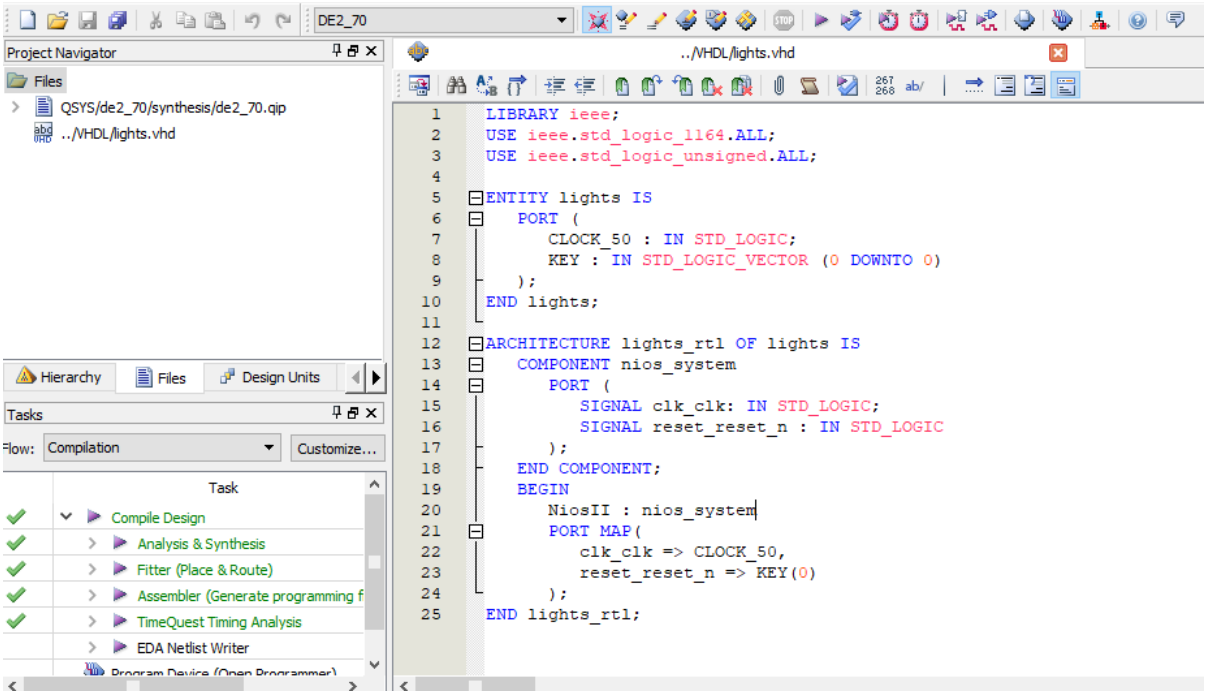
Figura 29 – QSYS: Geração do Sistema



Fonte: Elaborada pelo autor

No programa Quartus deve ser incluído o arquivo de síntese gerado pelo QSYS, de extensão .qip, para a pasta (*Files*) do *workspace*. Posteriormente cria-se o arquivo .vhd que conterá as bibliotecas, entidade (*Entity*), a arquitetura e a instância do sistema desenvolvido no QSYS (Figura 30). Posteriormente, é feita a compilação do projeto para verificação de possíveis erros.

Figura 30 – Quartus: Instanciação do Sistema

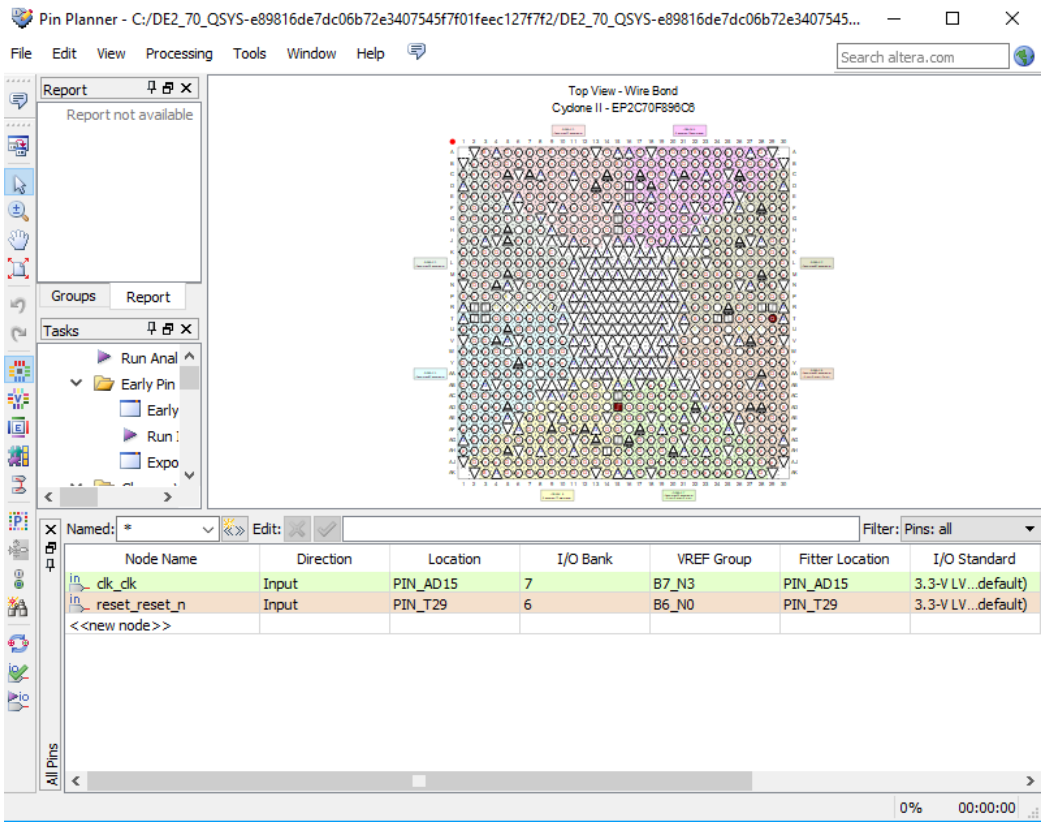


Fonte: Elaborada pelo autor

Para utilização dos componentes de *hardware* da placa, como sinal de *clock* e *reset*, botões (*push buttons*), chaves (*switches*), memórias (SSRAM e SDRAM), controlador VGA e demais recursos que são externos ao FPGA, é necessário a atribuição de pinos por meio do *Pin Planner*, ou seja, indicar a localização destes componentes para que ocorra conexão dos mesmos com o sistema criado por meio das linguagens de descrição de *hardware* (VHDL). Na Figura 31 têm-se a configuração dos pinos referente ao *clock* e *reset* que recebem, respectivamente, as localizações PIN_AD15 e PIN_T29 da placa.

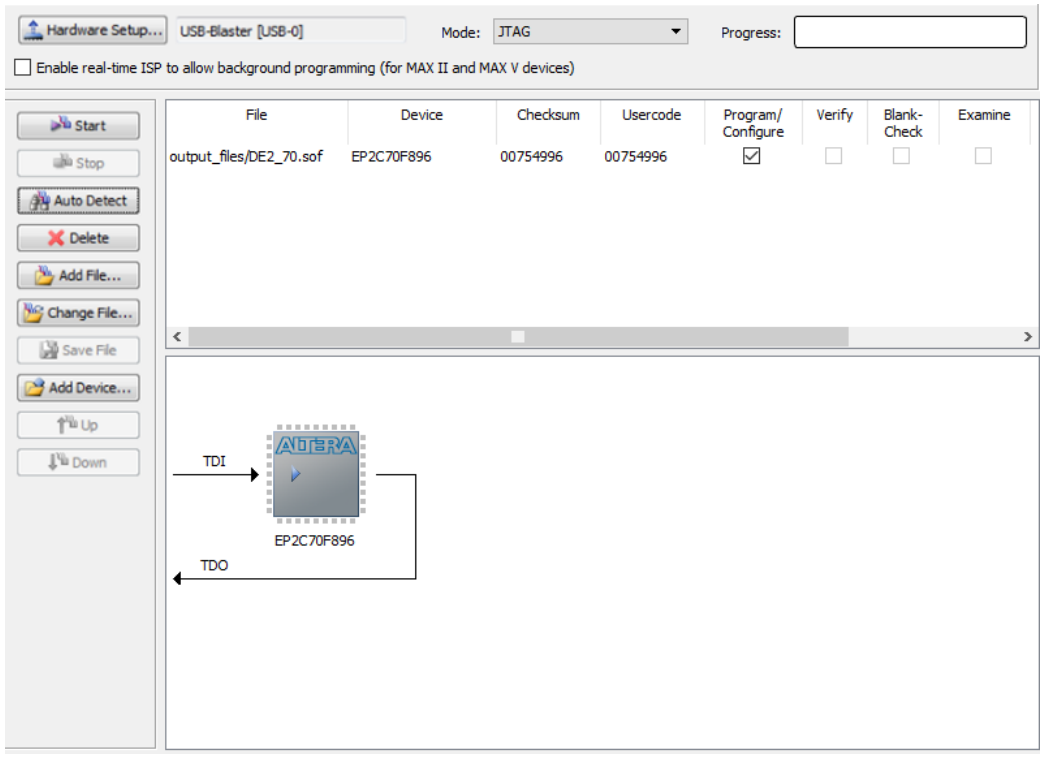
Após a compilação, cria-se o arquivo de síntese, com extensão .sof, este contêm as informações do FPGA e o programa que será gravado no chip (desenvolvido em linguagem de descrição de *hardware*). Portanto, é necessário que este arquivo seja transferido ao FPGA. A opção *Programmer* permite que o arquivo descrito anteriormente seja selecionado para gravação, bem como a identificação do dispositivo (DE2-70) e o modo de programação (JTAG), conforme é mostrado na Figura 32.

Figura 31 – Quartus: Atribuições dos Sinais do Sistema aos Componentes de *Hardware* da Terasic DE2-70



Fonte: Elaborada pelo autor

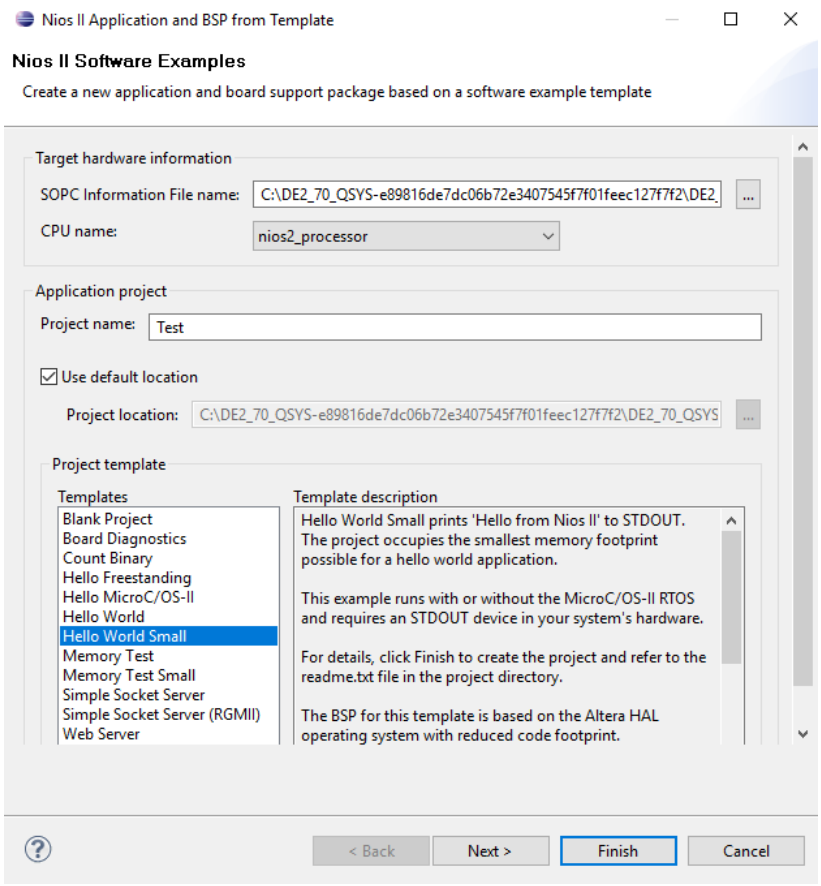
Figura 32 – Quartus: Gravação do Programa (descrito em *hardware*) no FPGA



Fonte: Elaborada pelo autor

O Eclipse SBT fornece o ambiente de programação para linguagens C/C++, e por meio da mesma foram desenvolvidos os algoritmos que serão executados no processador Nios. Na ato de criação do projeto, é necessária a indicação do arquivo .sopc que contêm as informações do sistema desenvolvido por meio do programa QSYS, como mostrado na Figura 33. Ao selecioná-lo é informado o nome do processador. Neste momento se insere o título do projeto (*Test*) e o *template* (modelo).

Figura 33 – Eclipse: Criação do Projeto no Eclipse SBT

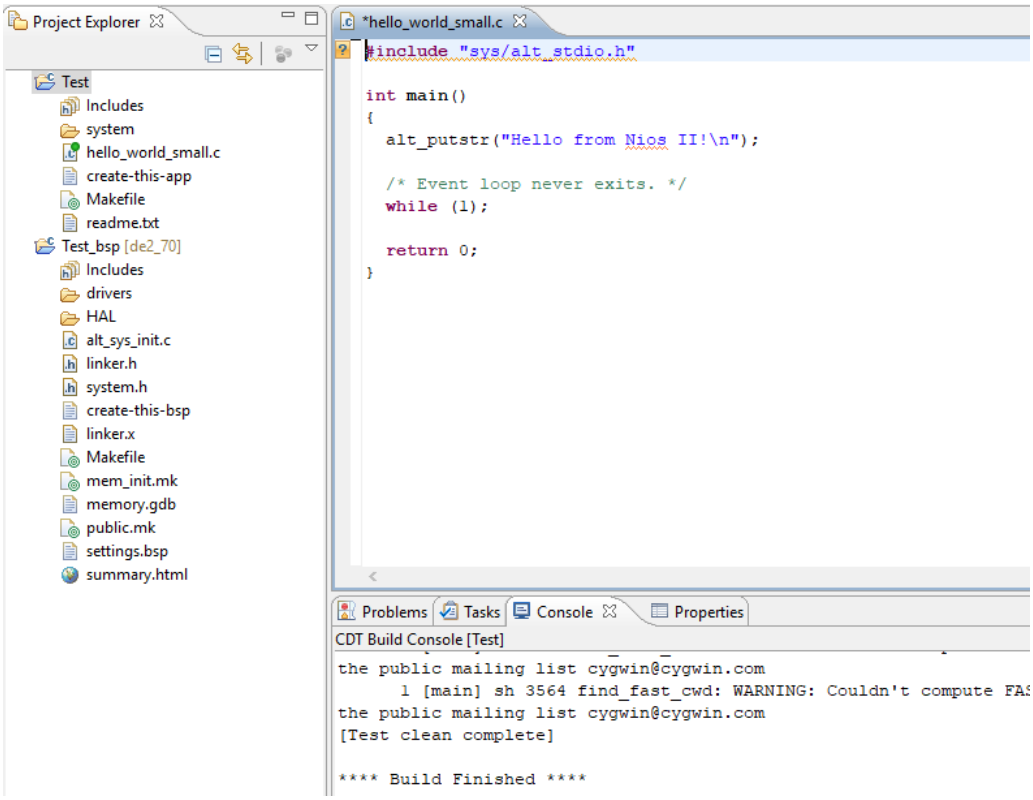


Fonte: Elaborada pelo autor

Após a criação do projeto têm-se, conforme mostra a Figura 34, o código-fonte baseado no *template*, e também a criação do *board support package* (BSP), que é uma camada de *software* que contém os drivers que serão necessários para acessar os recursos de *hardware*.

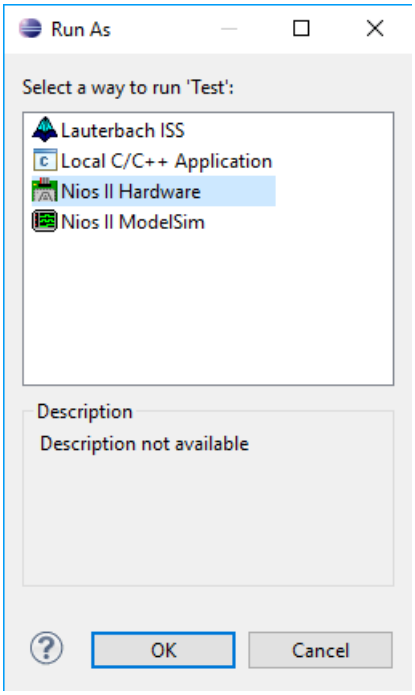
Por último é realizada a execução por meio da seleção da opção *Run As*, na aba *Run*, e a indicação do *Nios II Hardware* para que o mesmo inicie a execução do algoritmo (Figura 35). Na Figura 36 têm-se o resultado da execução.

Figura 34 – Eclipse: Exemplo Código-Fonte



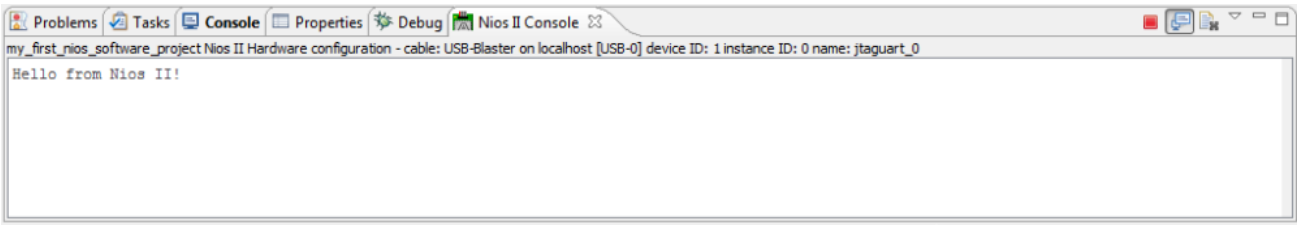
Fonte: Elaborada pelo autor

Figura 35 – Eclipse: Seleção do *Hardware* para Execução



Fonte: Elaborada pelo autor

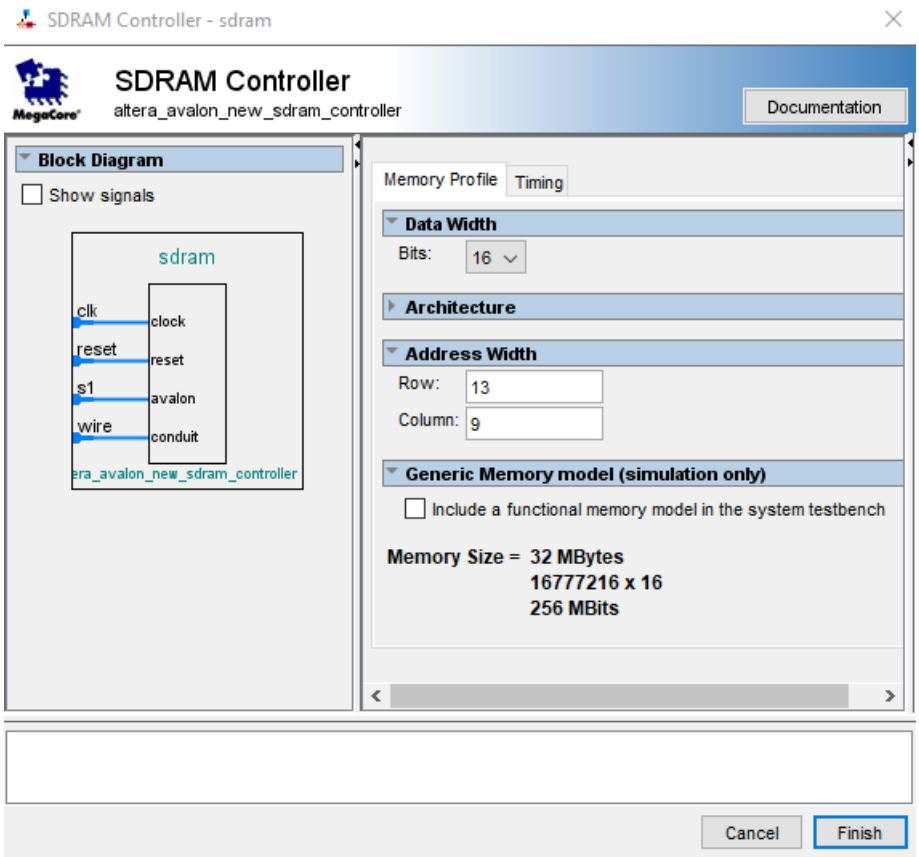
Figura 36 – Eclipse: Execução do Código Fonte no Nios



Fonte: Elaborada pelo autor

Devido a limitações de armazenamento da memória do FPGA (**On-chip Memory**), foi necessário a substituição do mesmo por uma de maior capacidade, portanto foi incluído o controlador que permitirá o acesso ao chip de memória **SDRAM**. Na configuração do controlador foi definido o valor de largura do barramento de dados (*Data Width*) para 16 *bits*, em *Address Width* foram atribuídos os números de *bits* de endereçamento de linhas e colunas do chip para 13 e 9, respectivamente (Figura 37).

Figura 37 – QSYS: Inclusão do **Controlador SDRAM**

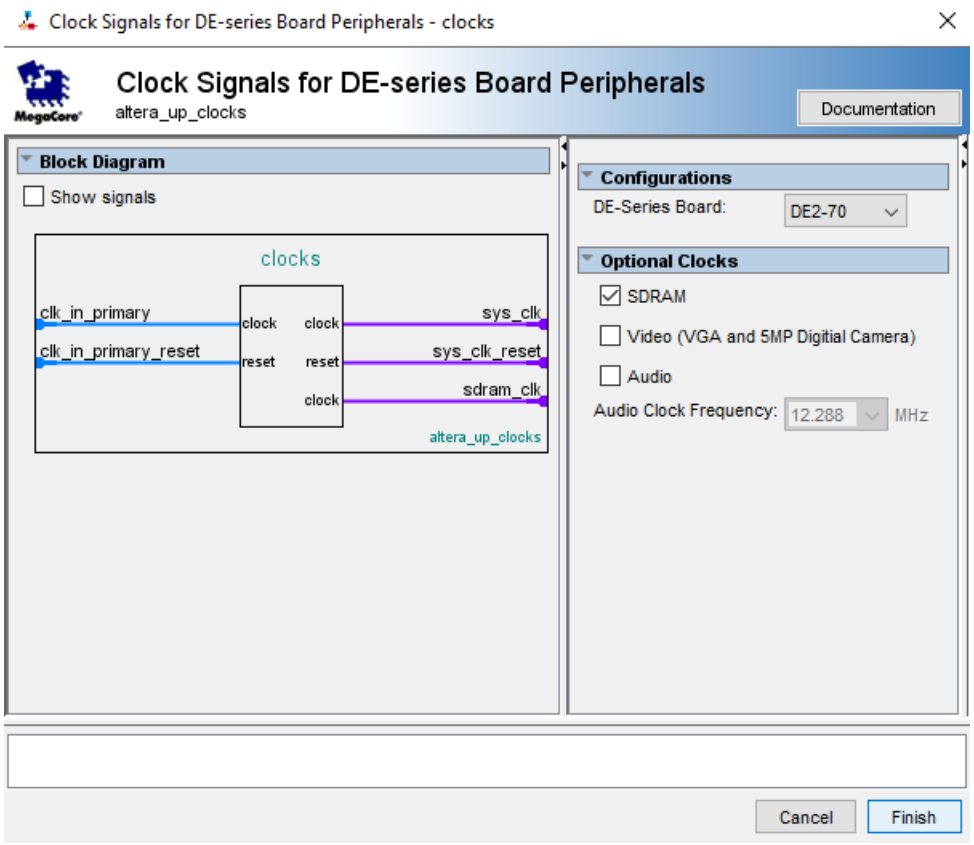


Fonte: Elaborada pelo autor

Devido a inclusão do **controlador SDRAM**, ocorreu o problema de dessincronização (*Clock Skew*), ou seja, o instante em que o sinal de *clock* oriundo do **Clock Source** que chega a este controlador é diferente dos demais componentes do sistema. Portanto,

para evitar a incidência deste problema, foi requerida a utilização do componente **Clock Signals for DE-series Board Peripherals**, este fornece o sinal de *clock* aos componentes do sistema (ex. Nios, SDRAM, etc.) de forma sincronizada evitando que haja o problema supracitado. Na Figura 38 mostra a configuração do mesmo, como a seleção da placa e indicação da **SDRAM**.

Figura 38 – QSYS: Inclusão do Componente **Clock Signals for DE-series Board Peripherals**



Fonte: Elaborada pelo autor

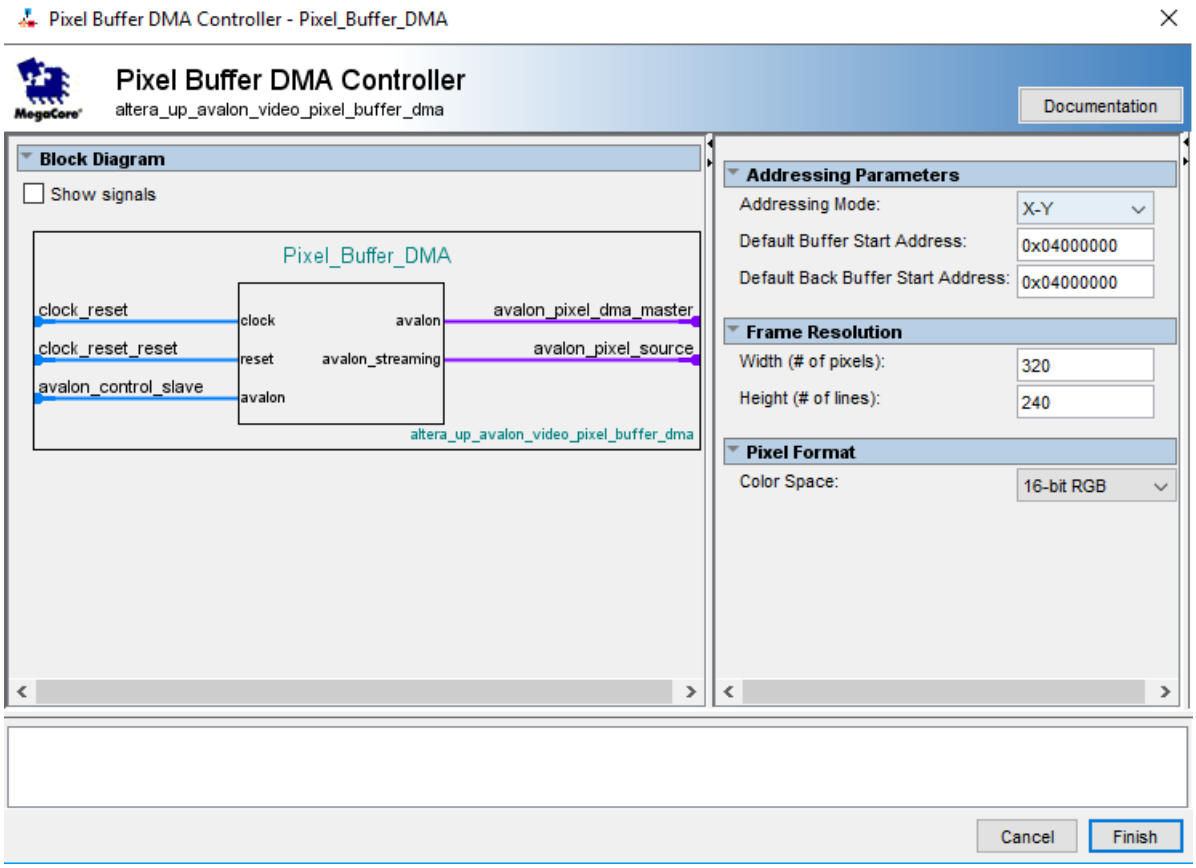
4.3.2 Desenvolvimento do Subsistema para Exibição de Imagem em Dispositivo Externo (Monitor)

Este subsistema é responsável por permitir a visualização das imagens capturadas da câmera Terasic TRDB-D5M, permitindo observar se o processo de captura das imagens realiza-se de forma correta. Para este desenvolvimento, foram necessários a inclusão, por meio do QSYS, dos seguintes componentes

- **Pixel Buffer DMA Controller (Pixel_Buffer_DMA)**: Este componente permite transmitir os dados de imagem que estão armazenados em memória (*Pixel Buffer*) e os envia para o **controlador VGA**. Em *Addressing Parameters* têm-se nos campos *Default Buffer Start Address Default Back Buffer Start Address* os en-

dereços base (0x04000000) do *Pixel Buffer* de onde será realizada a leitura, este é atribuído automaticamente pelo QSYS. Em *Frame Resolution* têm-se a configuração do quadro de *pixels*, ou seja, a resolução da imagem (320 x 240). Em *Pixel Format* contêm as informações de cores do *pixel*, foi definido o valor de 16 *bits* para representá-las (Figura 39).

Figura 39 – QSYS: Inclusão do Componente **Pixel Buffer DMA Controller**



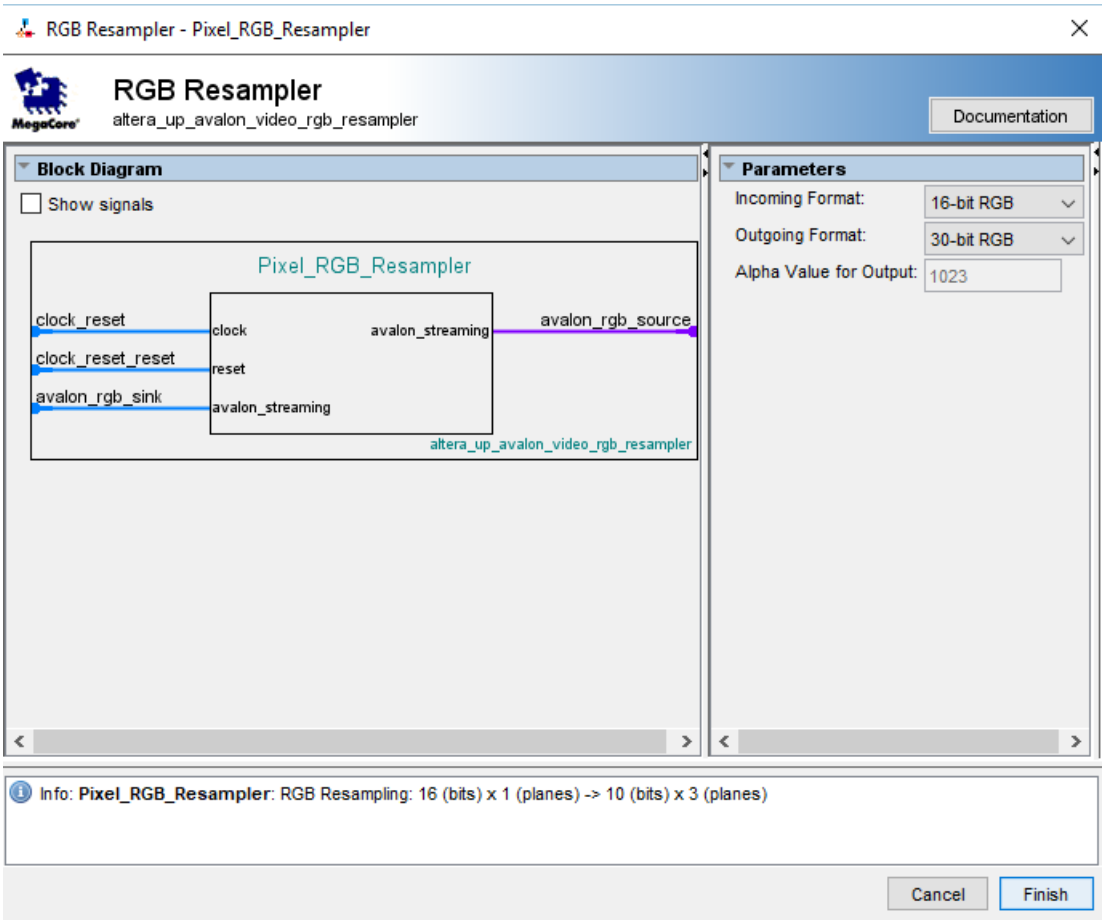
Fonte: Elaborada pelo autor

- **RGB Resampler (Pixel_RGB_Resampler)**: Este componente realiza a conversão do número de *bits* que representa cada *pixel*. Este procedimento é necessário, pois o controlador VGA exige uma imagem de entrada com valor de 30 *bits* por *pixel*. Portanto, o **RGB Resampler** realiza esta tarefa e converte a imagem de 16 *bits* obtida do **Pixel Buffer**, por meio do **Pixel Buffer DMA Controller**, para 30 *bits*. Na Figura 40 é apresentado, em *Incoming Format*, o valor referente a imagem de entrada, e em *Outgoing Format* o valor para o qual se deseja converter.

- **Scaler (Pixel_Scaler)**: Este componente possibilita a conversão da resolução da imagem, pois o **controlador VGA** exige uma imagem de entrada com a resolução 640 x 480, o dobro da configuração presente na imagem que está armazenada no **Pixel Buffer**, que é de 320 x 240. Para tal procedimento, o **Pixel Scaler** realiza a replicação de cada *pixel* 4 vezes, sem alterar o número de *bits* por *pixel*, permitindo

assim a compatibilidade da imagem com o padrão de saída (VGA). A Figura 41 apresenta a configuração dos parâmetros referente ao fator de escala (multiplicação) (*Width Scaling Factor* e *Height Scaling Factor*), e em *Incoming Frame Resolution* e *Pixel Format* têm-se as configurações da imagem de entrada, resolução (320 x 240) e número de *bits* por *pixel* (10 *bits* por componente RGB), respectivamente.

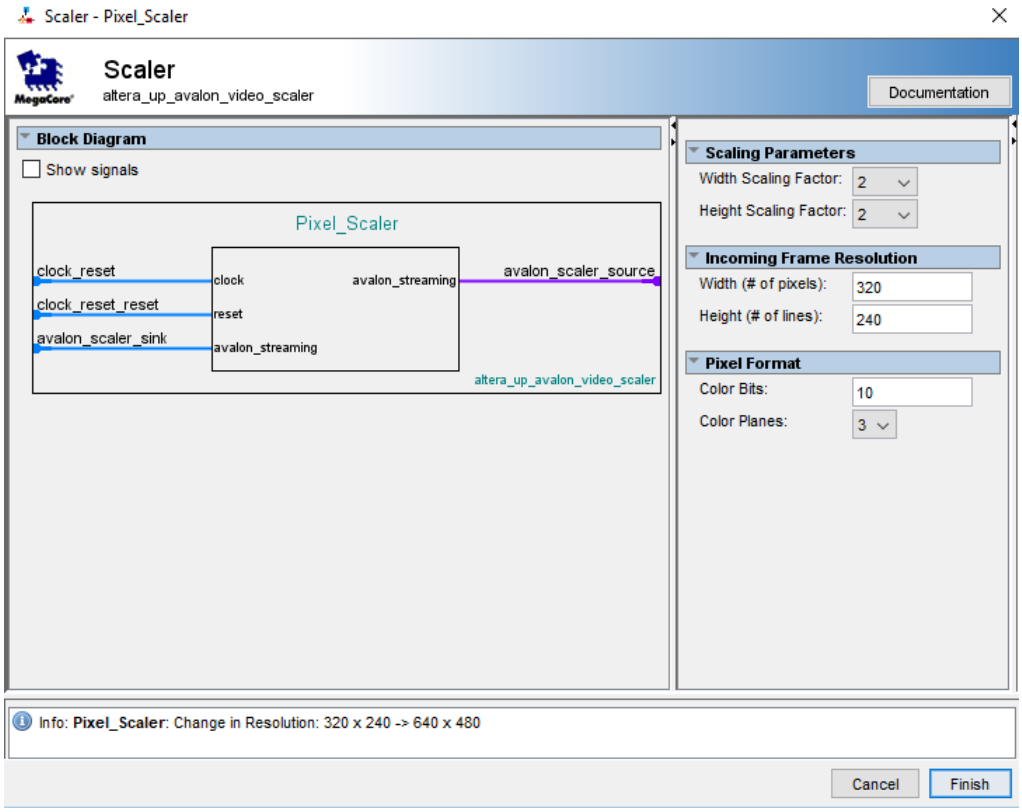
Figura 40 – QSYS: Inclusão do Componente **RGB Resampler**



Fonte: Elaborada pelo autor

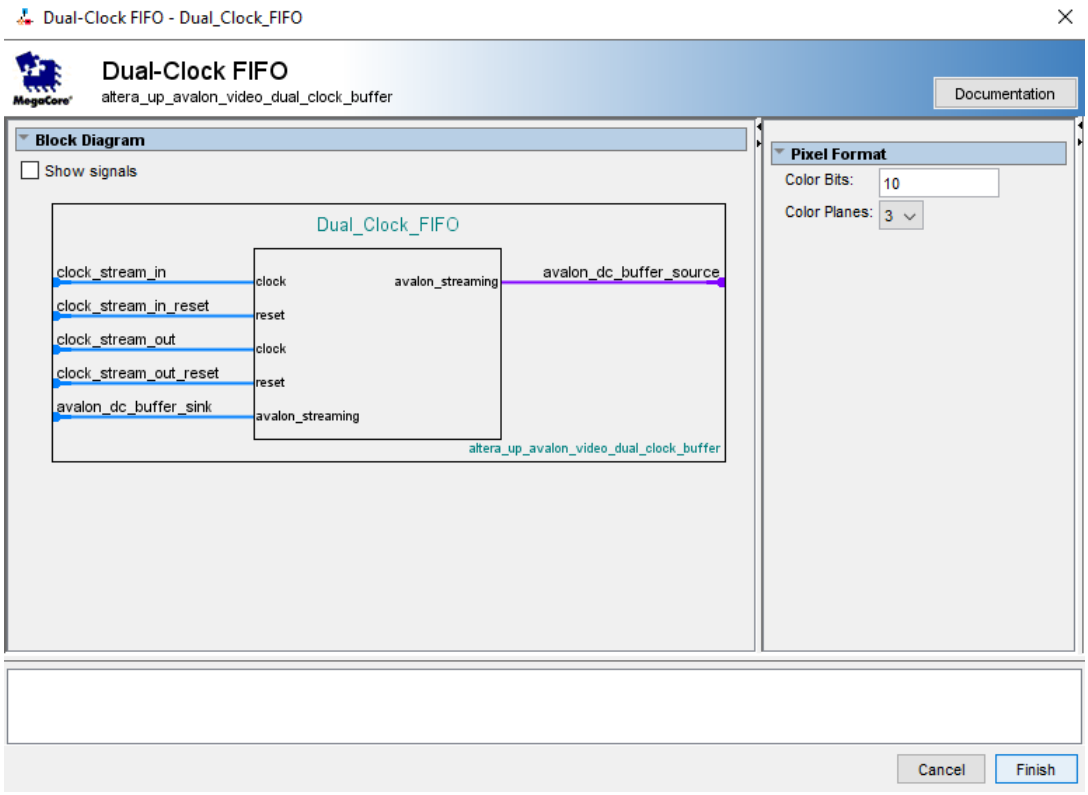
- **Dual-Clock FIFO (Dual_ClocK_FIFO)**: o uso deste componente permite garantir que a imagem modificada pelo **Scaler**, que está funcionando com o *clock* de 50 MHz, possa ser transmitida ao **controlador VGA**, que possui a metade deste sinal (25 MHz), sem que ocorra problema de incompatibilidade. Em sua configuração, há informações do formato do *pixel* (*Pixel Format*), como o número de 10 *bits* (*Color Bits*) e plano de cor (*Color Planes*) para o valor 3, como mostra a Figura 42.

Figura 41 – QSYS: Inclusão do Componente **Scaler**



Fonte: Elaborada pelo autor

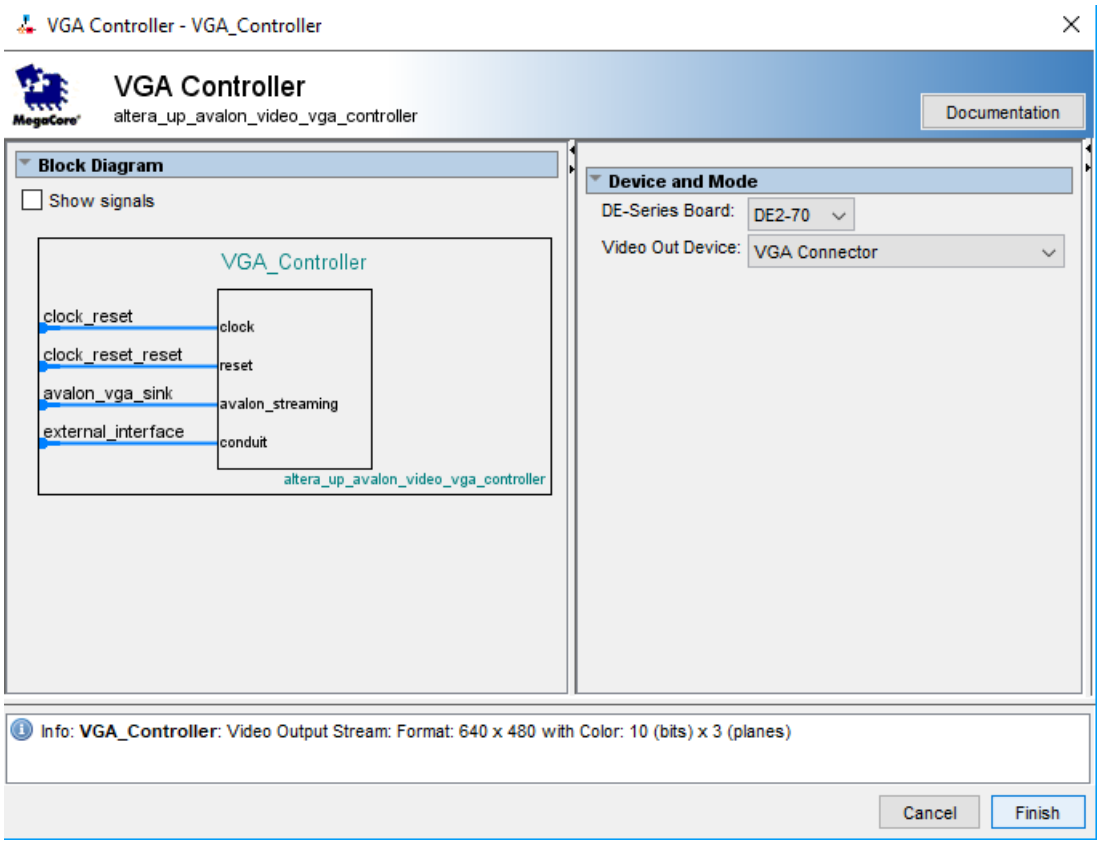
Figura 42 – QSYS: Inclusão do Componente **Dual-Clock FIFO**



Fonte: Elaborada pelo autor

- **VGA Controller (VGA_Controller)**: O controlador VGA permite que os dados recebidos pelo **Dual-Clock FIFO** sejam transmitidos para o conector físico do VGA. Portanto, este componente gerencia todo o tráfego de informações até um dispositivo externo (Ex. monitor). Em sua configuração é necessário apenas informar o dispositivo a ser utilizado (DE-Series Board), neste caso é a Terasic DE2-70, e qual tipo de saída receberá os dados (*Video Out Device*), conector VGA (*VGA Connector*), conforme a Figura 43.

Figura 43 – QSYS: Inclusão do Componente **VGA Controller**

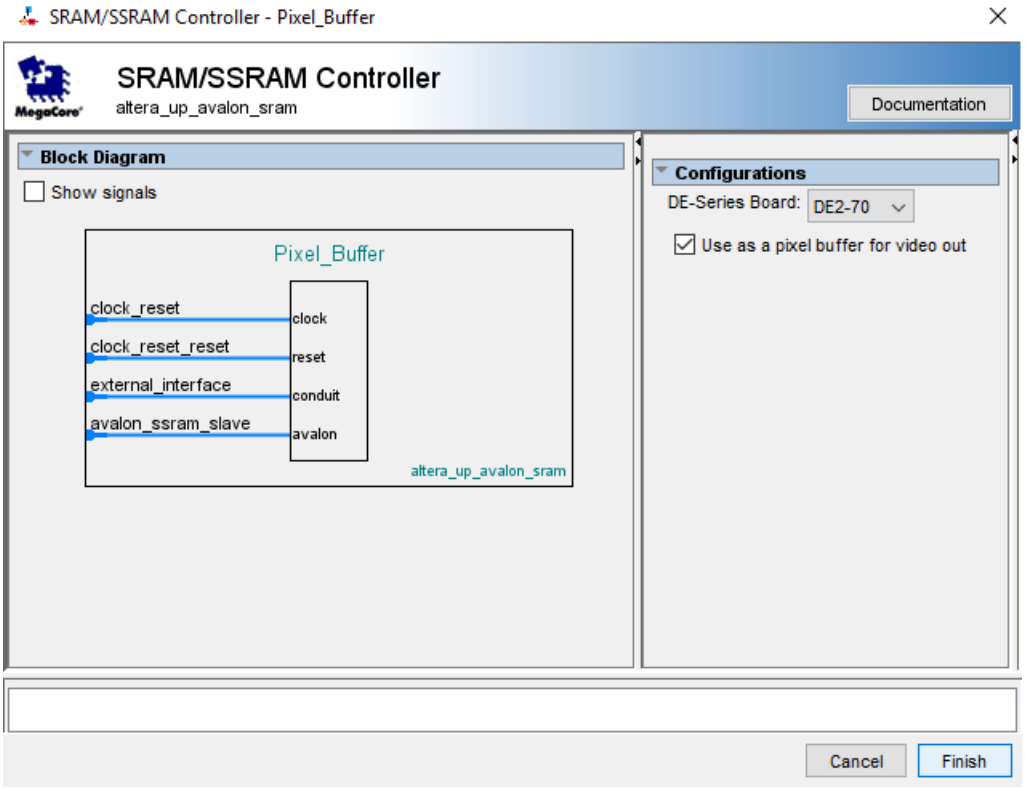


Fonte: Elaborada pelo autor

- **SRAM/SSRAM Controller (Pixel_Buffer)**: O **controlador SRAM/SSRAM** permite o acesso ao chip de memória, no caso da Terasic DE2-70 o **SSRAM**, contido no *hardware* da mesma. Este componente será utilizado como **Pixel Buffer**, conforme indicação em sua configuração (*Configurations*) juntamente com a seleção do modelo do *hardware* (DE-Series Board) (Figura 44).

  Após o desenvolvimento deste subsistema, foi implementado, por meio do Eclipse SBT, um algoritmo que realiza o desenho de um retângulo. Isto foi desenvolvido com o intuito de verificar a corretude da conexão do sistema com o dispositivo de vídeo externo (monitor). Na Figura 45 têm-se o resultado desta implementação.

Figura 44 – QSYS: Inclusão do Componente **SRAM/SSRAM Controller**



Fonte: Elaborada pelo autor

Figura 45 – Teste com a Saída de Vídeo no Monitor

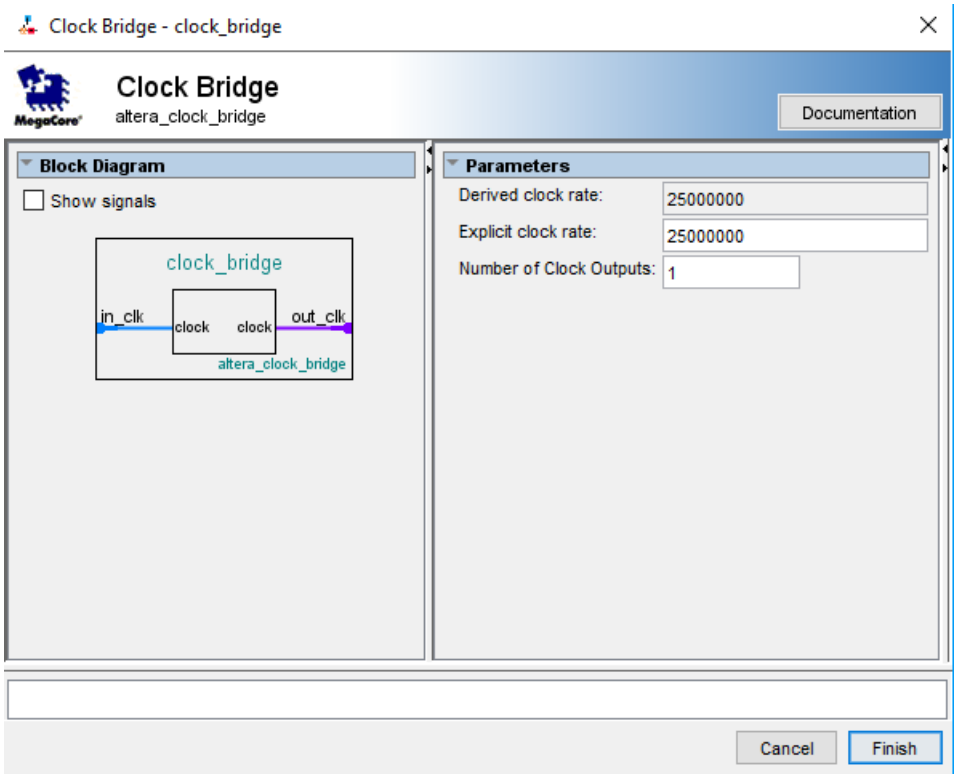
Fonte: Elaborada pelo autor

4.3.3 Desenvolvimento do Subsistema para Captura de Imagens por meio da Terasic TRDB-D5M

Este subsistema é responsável por efetuar a configuração e captura de imagens por meio da câmera (Terasic TRDB-D5M), conversão e adequação dos dados de imagem recebidos para o armazenamento na memória e posterior manipulação (processamento). Para seu funcionamento os seguintes componentes necessitam ser incluídos ao subsistema

- **Clock Bridge (clock_bridge)**: Este componente fornece o *clock* necessário para funcionamento da câmera (TRDB-D5M). Nos parâmetros de configuração (*Parameters*) é definido o valor de 25MHz e o número de saídas (*Number of Clock Outputs*) com o valor unitário, conforme Figura 46.

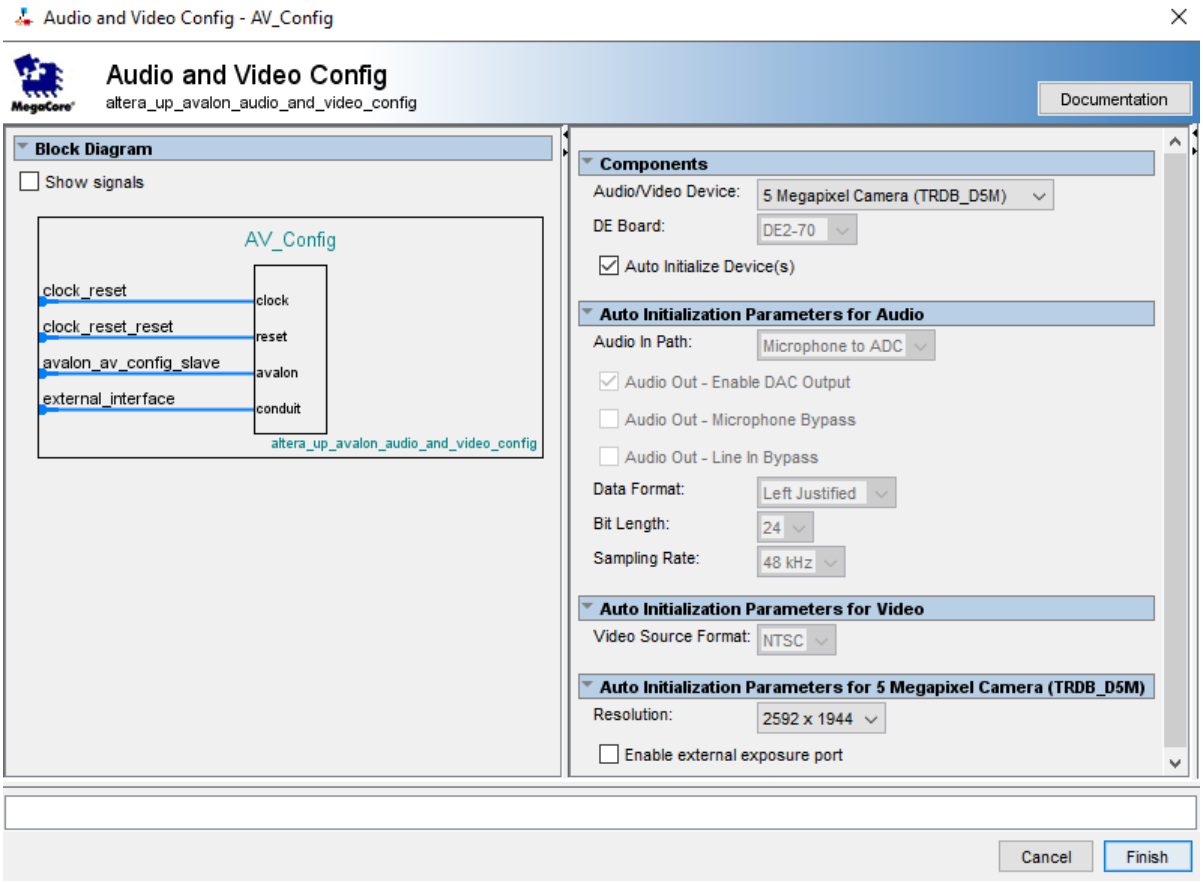
Figura 46 – QSYS: Inclusão do Componente **Clock Bridge**



Fonte: Elaborada pelo autor

- **Audio and Video Config (AV_Config)**: O **audio e vídeo config** inicializa o chip responsável pela conversão analógico-digital (*Analog-to-Digital*), presente na Terasic DE2-70, com as configurações necessárias para o uso de dispositivo analógico de entrada. Dentre as atribuições estão selecionados que tipo de dispositivo (câmera) (*Audio/Video Device*) que realizará a captura de imagens e a resolução de vídeo do mesmo (*Resolution*), dado por 2592 x 1944, conforme é indicado na Figura 47.

Figura 47 – QSYS: Inclusão do Componente **Audio and Video Config**

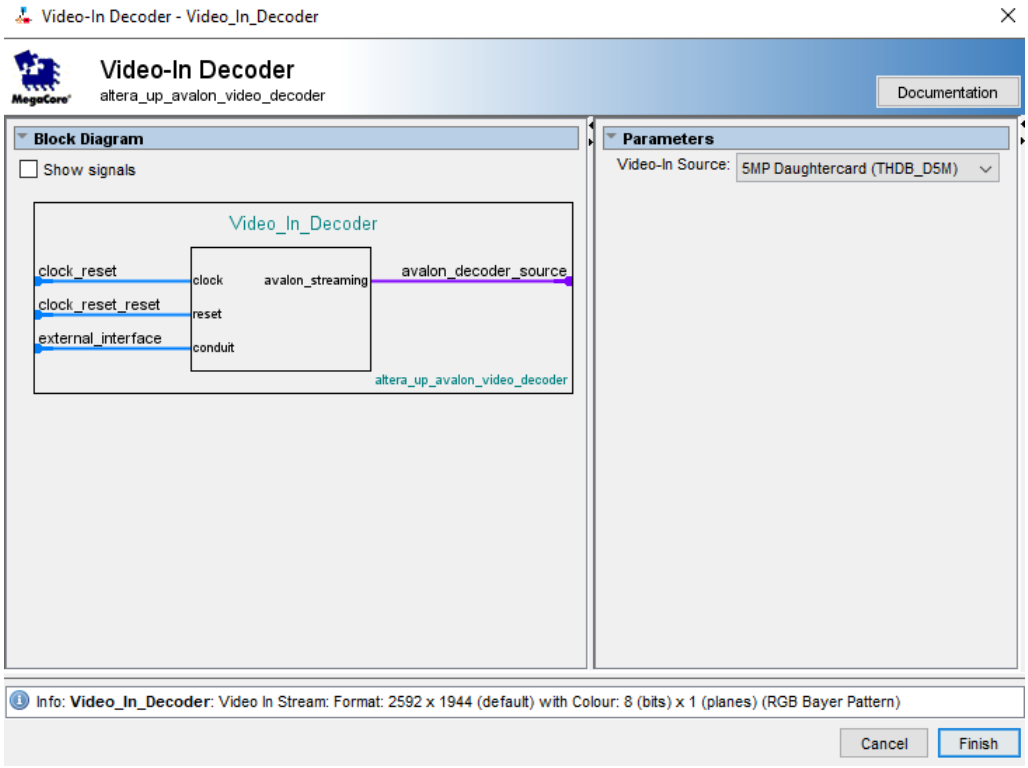


Fonte: Elaborada pelo autor

- **Video-In Decoder (Video_In_Decoder)**: este componente converte os dados de vídeo obtidos do conversor (ADC) em pacotes que serão enviados ao controlador DMA (*Direct Memory Access*). Sua configuração resulta em informar qual o dispositivo de entrada de vídeo (*Video-In Source*) será utilizado, neste caso a câmera (Figura 48).

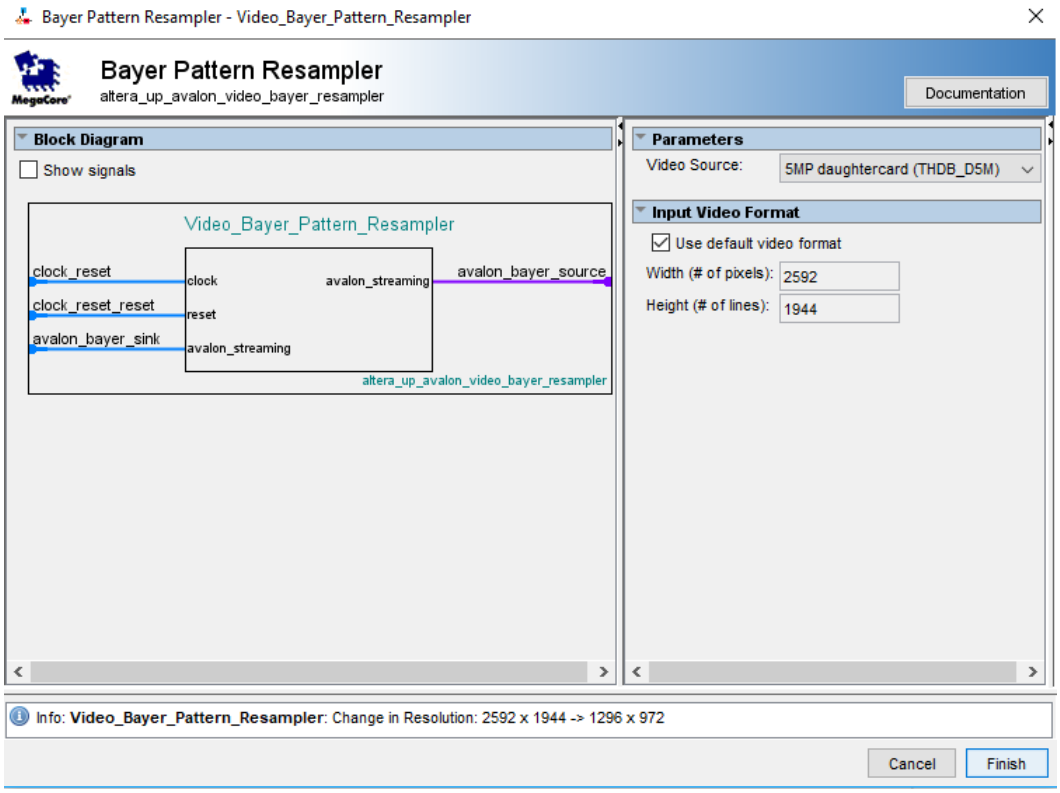
- **Bayer Pattern Resampler (Video_Bayer_Pattern_Resampler)**: O **Bayer Pattern** é um caso especial do espaço de cores RGB, pois cada *pixel* possui um único valor para uma cor RGB (vermelho, verde e azul), e as demais são contidas em *pixels* de sua adjacência. Sendo assim é necessário que este espaço de cores esteja no formato RGB de 24 *bits*. Para esta conversão utiliza-se o componente **Bayer Pattern Resampler**, que combinará os *pixels* em um único plano que contêm os *bits* referentes as três cores. Na Figura 49 é exibido os parâmetros de entrada ao componente, como a fonte de vídeo (*Video Source*) e a resolução do quadro (2592 x 1944).

Figura 48 – QSYS: Inclusão do Componente **Video-In Decoder**



Fonte: Elaborada pelo autor

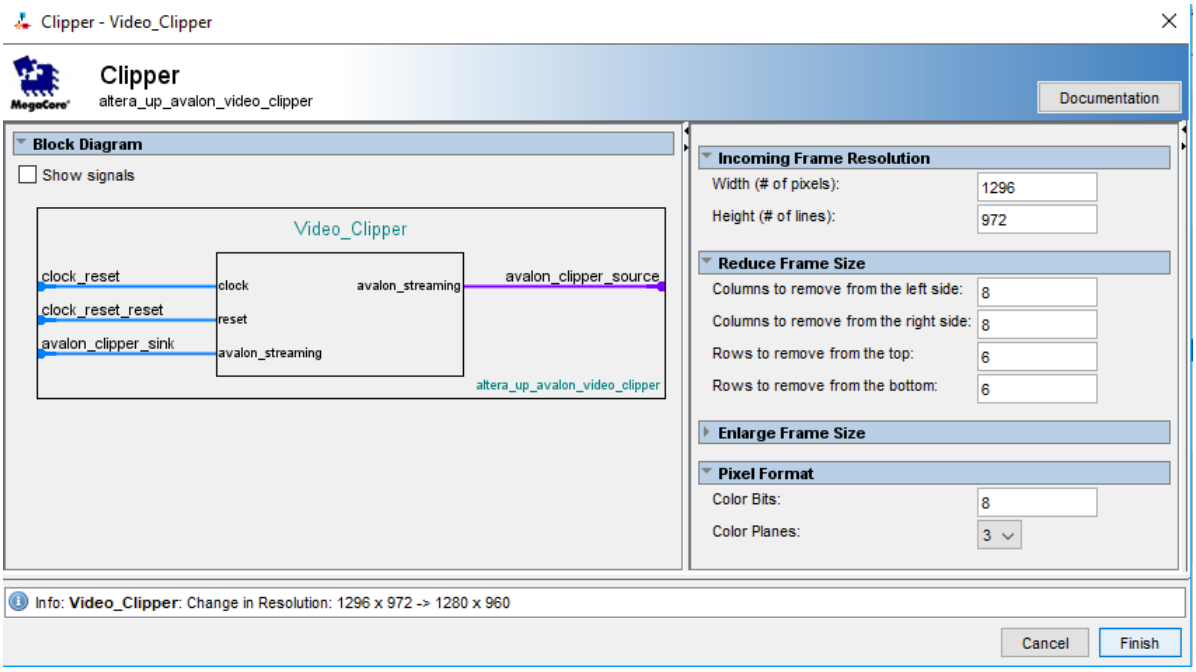
Figura 49 – QSYS: Inclusão do Componente **Bayer Pattern Resampler**



Fonte: Elaborada pelo autor

- **Clipper (Video_Clipper)**: O **Clipper** realiza a redução do tamanho do *frame* transmitido de 1296 x 972 (*Incoming Frame Resolution*) para 1280 x 960 para adequação à taxa de proporção compatível com o VGA, para tal procedimento são removidas 8 colunas de cada lado deste *frame*, e 6 linhas da base e do topo do mesmo. Na Figura 50 têm-se a definição dos parâmetros do componente.

Figura 50 – QSYS: Inclusão do Componente **Clipper**

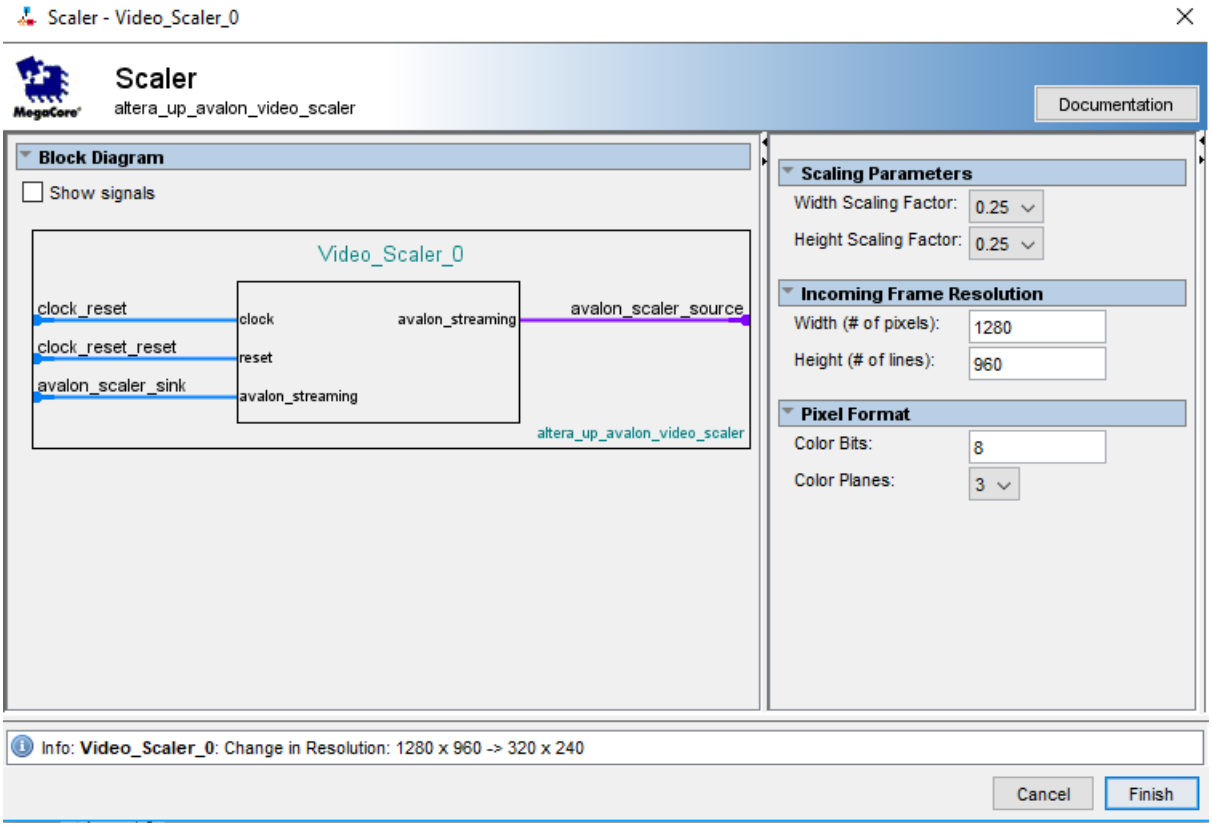


Fonte: Elaborada pelo autor

- **Scaler (Video_Scaler_0)**: O **scaler** realiza o redimensionamento do *frame* de imagem por meio de um fator de multiplicação, no componente são configurados os fatores relacionados à altura e largura deste *frame* (*Width Scaling Factor* e *Height Scaling Factor*) que diminuirá o quadro para 1/4 de seu tamanho, isto é requerido devido limitações do *frame buffer* (Figura 51).

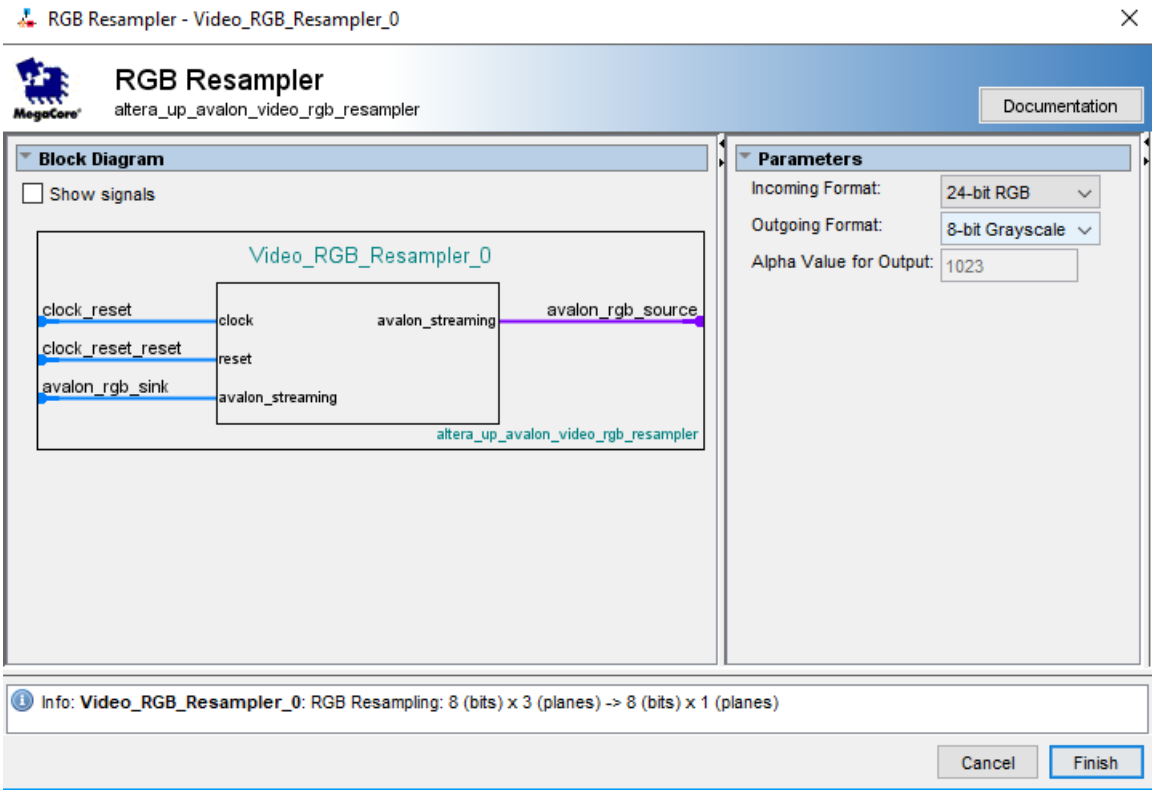
- **RGB Resampler (Video_RGB_Resampler_0)**: Como descrito na Seção 4.3.2, este componente realiza a conversão do número de *bits*. Porém neste subsistema este componente possui também a função de converter a imagem para escala de cinza. Como entrada (*Incoming Format*) têm-se o formato de cor da imagem (24 *bits*) em RGB, e como saída (*Outgoing Format*) resulta-se em uma imagem com 8 *bits* (*grayscale*), conforme indica a Figura 52.

Figura 51 – QSYS: Inclusão do Componente **Scaler**



Fonte: Elaborada pelo autor

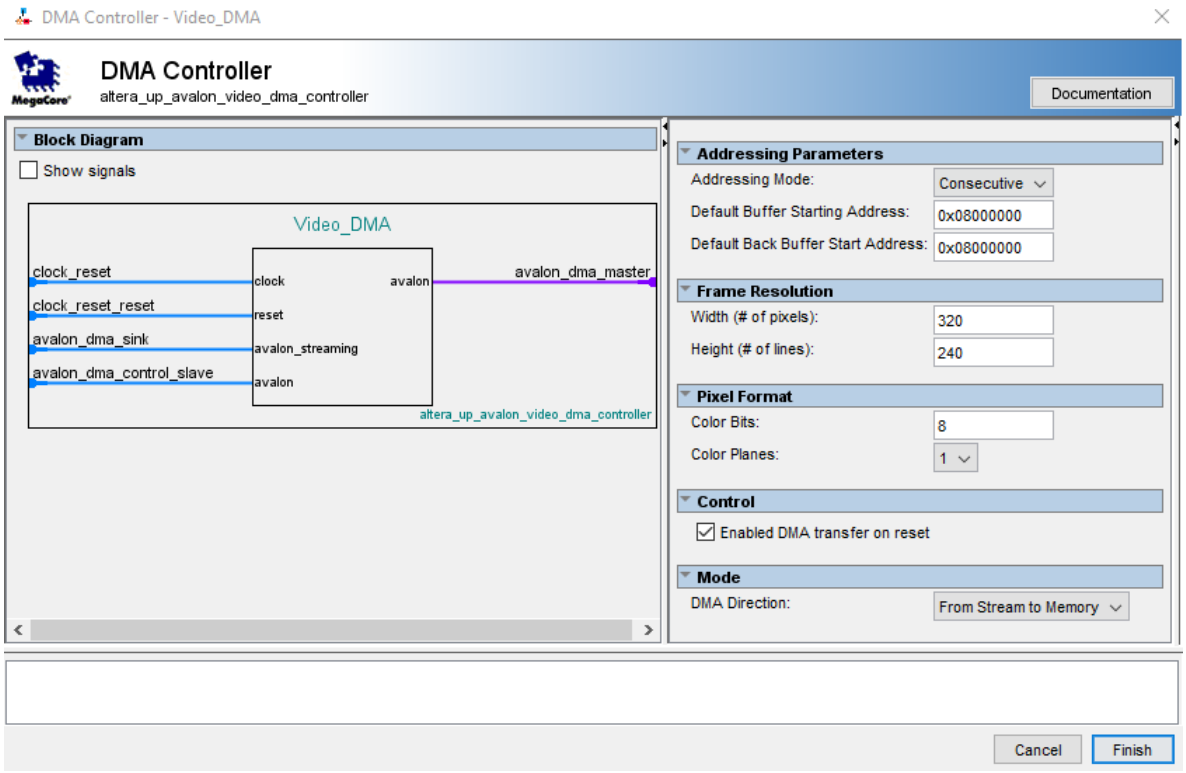
Figura 52 – QSYS: Inclusão do Componente **RGB Resampler**



Fonte: Elaborada pelo autor

- **DMA Controller (Video_DMA)**: O **Controlador DMA** transfere a imagem (*stream*), previamente ajustada pelos componentes supracitados, para o **frame buffer** (**SRAM/SSRAM**). Sua configuração resulta em definir o endereço da memória (0x08000000) a ser escrita em *Addressing Parameters*; a resolução do frame (*Frame Resolution*) (320 x 240); o formato do pixel (*Pixel Format*) (8 *bits* com um plano de cor); e a direção do tráfego de dados, do *stream* para memória (*Mode*). Na Figura 53 são apresentadas estas configurações.

Figura 53 – QSYS: Inclusão do Componente **DMA Controller**



Fonte: Elaborada pelo autor

4.4 Resultados do Experimento

Após o desenvolvimento do sistema, houve a integração do algoritmo de Viola-Jones para ser executado no processador Intel/Altera Nios II. Posteriormente, foram definidos em dois os cenários de testes, o primeiro (Seção 4.4.1) é verificar as execuções do algoritmo em imagens previamente armazenadas em disco. No segundo (Seção 4.4.2), foi verificado o funcionamento do algoritmo ao processar imagens obtidas da câmera.

4.4.1 Detecção de Imagens Previamente Armazenadas

Foram utilizados para testes 15 imagens em escala de cinza e convertidas para o formato .PGM (*Portable Gray Map*), estas imagens foram obtidas do mural da 5ª

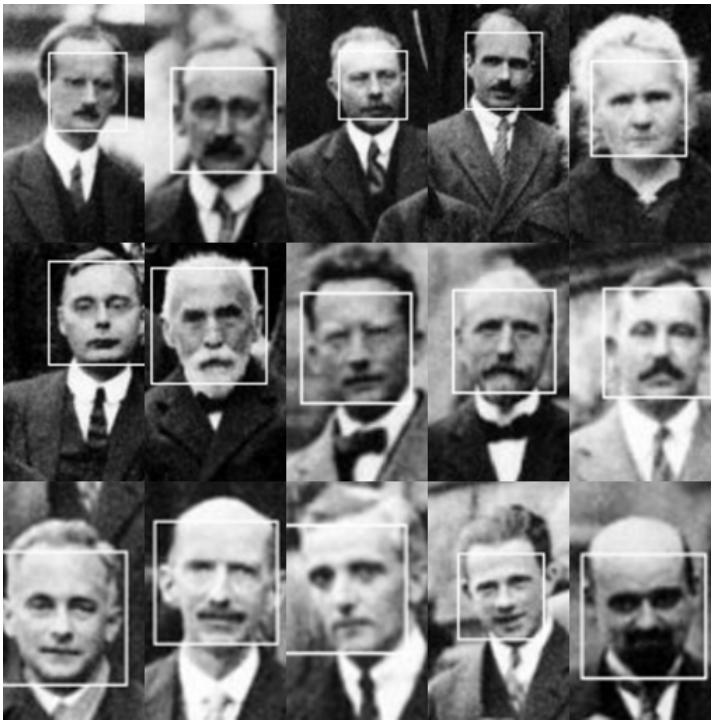
Conferência de Solvay (SOLVAY, 1927). Utilizou-se esta imagem devido às diferentes características dos participantes, como formato do rosto, cabelo, bigode, uso de óculos, entre outros.

A execução do algoritmo ocorreu por meio do Nios/f (*fast*), portanto este controla todo o sistema como a transmissão, via porta serial, de cada imagem do computador (PC) para memória SDRAM da Terasic DE2-70, para posteriormente ser processada pelo algoritmo. Em cada execução foi cronometrado pelo relógio o tempo necessário desde a transmissão do arquivo de imagem, processamento pelo algoritmo de Viola-Jones até a geração da imagem com a demarcação da face encontrada. A Figura 54 apresenta as detecções faciais (da esquerda para direita, de cima para baixo, enumeradas de 1 a 15) e no Quadro 5 são descritos o tempo e a média de todas as execuções.

Pôde-se observar os elevados tempos de execução devido as limitações de *clock* do dispositivo (50MHz). Sendo assim, há a diminuição de desempenho em comparação a outras abordagens como a utilização de computadores pessoais que operam com relógios na faixa de *gigahertz*. Outro fator relevante que justifica este tempo é a não aceleração por hardware do algoritmo de Viola-Jones, pois este é executado exclusivamente em um processador embarcado sem que ocorra otimizações em relação a tecnologia utilizada (FPGA).

Foi realizada, também, a execução do algoritmo de Viola-Jones para detectar várias faces em um único arquivo. A Figura 55 foi utilizada para verificar o funcionamento. Observou-se a detecção de todas as faces, porém o tempo de execução foi maior que o exibido no Quadro 5, sendo este com o total de **57 minutos**, **02 segundos** e **55 milésimos**. Isto ocorreu devido ao processamento de múltiplas faces em uma mesma execução.

Figura 54 – Sequência de Faces Detectadas



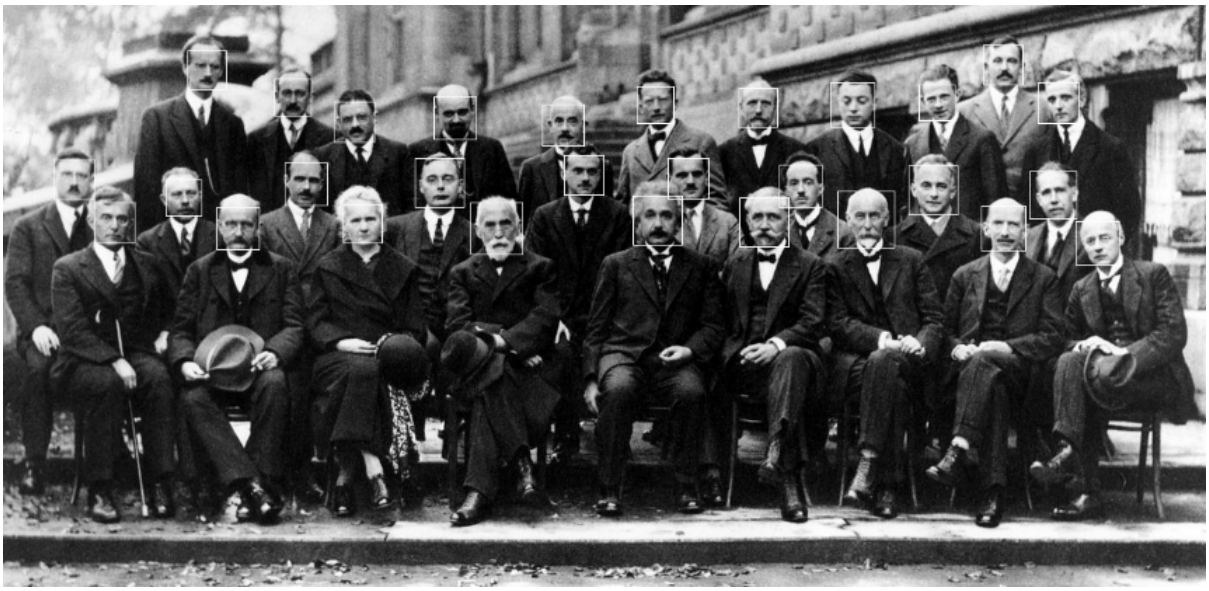
Fonte: Elaborada pelo autor

Quadro 5 – Tempos de Execução

Figura	Tempo de Execução (mm:ss:ms)
1	04:05:57
2	04:00:22
3	04:02:59
4	04:07:01
5	04:00:06
6	04:10:52
7	04:11:55
8	04:11:46
9	04:03:47
10	04:03:55
11	04:02:54
12	04:07:08
13	04:05:56
14	04:09:55
15	04:04:52
Média dos Tempos de Execução	04:05:58

Fonte: Elaborada pelo autor

Figura 55 – Detecção de Várias Faces em uma Única Execução



Fonte: Elaborada pelo autor

4.4.2 Detecção de Imagens Obtidas da Câmera Terasic TRDB-D5M

Foram realizados testes com a captura de imagens da câmera Terasic TRDB-D5M, as imagens obtidas são armazenadas na memória SRAM e, com a utilização do Nios, são processadas pelo algoritmo de detecção facial. Após o processamento da imagem, a mesma é armazenada no formato .PGM e exibida também, por meio do controlador VGA, no monitor de vídeo.

Para verificar o comportamento do algoritmo, optou-se por definir cenários para testes, e diante dos mesmos, foram obtidas diversas imagens da internet para utilização nestes testes, um total de 20 imagens foram utilizadas. A seguir são apresentados oito cenários que foram definidos:

1. Utilização do Nios *fast*, com o rosto em posição normal (longitudinal) e sem utilização de óculos;
2. Utilização do Nios *standard*, com o rosto em posição normal (longitudinal) e sem utilização de óculos;
3. Utilização do Nios *fast*, com o rosto em posição normal (longitudinal) e com utilização de óculos;
4. Utilização do Nios *standard*, com o rosto em posição normal (longitudinal) e com utilização de óculos;
5. Utilização do Nios *fast*, com o rosto em posição inclinada (em torno de 45º) e com utilização de óculos;

6. Utilização do Nios *standard*, com o rosto em posição inclinada (em torno de 45º) e com utilização de óculos;
7. Utilização do Nios *fast*, com o rosto em posição inclinada (em torno de 45º) e sem utilização de óculos; e
8. Utilização do Nios *standard*, com o rosto em posição inclinada (em torno de 45º) e sem utilização de óculos.

Nas Figuras 56, 57, 58 59 são exibidos os resultados das detecções, conforme os cenários listados anteriormente. Porém, foi decidido inserir imagens considerando somente a utilização de uma versão do Nios, pois não há diferença na detecção em si, com exceção do tempo de execução.

Figura 56 – Detecção Face: Rosto em Posição Normal e sem Utilização de Óculos

Fonte: Elaborada pelo autor

Figura 57 – Detecção Face: Rosto em Posição Normal e com Utilização de Óculos

Fonte: Elaborada pelo autor

Figura 58 – Detecção Face: Rosto em Posição Inclinada e com Utilização de Óculos

Fonte: Elaborada pelo autor

Figura 59 – Detecção Face: Rosto em Posição Inclinada e sem Utilização de Óculos

Fonte: Elaborada pelo autor

Todas as faces das imagens processas pelo algoritmo foram detectadas, mesmo com variações em aspectos físicos dos personagens como formato do rosto, olhos, utilização de franja, também com expressões faciais, diferenças no tom de pele, utilização de óculos e inclinação. Foi verificada a demarcação da área pertencente ao rosto por meio da indicação (quadrado) sobre o mesmo.

Em relação ao tempo de execução do algoritmo há pequenas variações no processamento de cada imagem. Porém, se comparada a mudança na versão do Nios (da versão padrão para rápida) é observada uma diminuição substancial do tempo de detecção, devido ao aumento de recursos que esta segunda versão oferece, como o cache de dados. Independentemente da versão utilizada, o elevado tempo de execução justifica-se pelo mesmo motivo informado na Seção 4.4.1. Nos Quadros 6, 7, 8 e 9 são apresentados os resultados de todas as execuções, em todos cenários supracitados.

Quadro 6 – Utilização do Nios *standard* (s) e *fast* (f), com o rosto em posição normal e sem utilização de óculos

Figura	Tempo de Execução utilizando o Nios/s (mm:ss:ms)	Tempo de Execução utilizando o Nios/f (mm:ss:ms)
1	05:01:59	02:56:43
2	05:00:06	02:54:55
3	05:03:22	02:55:55
4	05:02:44	02:57:10
5	05:01:44	02:55:24
Tempo Médio	05:01:59	02:56:01

Fonte: Elaborada pelo autor

Quadro 7 – Utilização do Nios *standard* (s) e *fast* (f), com o rosto em posição normal e utilização de óculos

Figura	Tempo de Execução utilizando o Nios/s (mm:ss:ms)	Tempo de Execução utilizando o Nios/f (mm:ss:ms)
1	05:05:56	02:56:58
2	05:05:13	02:56:02
3	05:06:01	02:55:55
4	05:05:21	02:54:23
5	05:06:24	02:56:30
Tempo Médio	05:05:47	02:55:58

Fonte: Elaborada pelo autor

Quadro 8 – Utilização do Nios *standard* (s) e *fast* (f), com o rosto em posição inclinado e utilização de óculos

Figura	Tempo de Execução utilizando o Nios/s (mm:ss:ms)	Tempo de Execução utilizando o Nios/f (mm:ss:ms)
1	04:45:55	02:47:54
2	04:46:22	02:47:01
3	04:44:57	02:47:30
4	04:45:30	02:46:45
5	04:45:45	02:48:05
Tempo Médio	04:45:42	02:47:27

Fonte: Elaborada pelo autor

Quadro 9 – Utilização do Nios *standard* (s) e *fast* (f), com o Rosto em Posição Inclinada e sem Utilização de Óculos

Figura	Tempo de Execução utilizando o Nios/s (mm:ss:ms)	Tempo de Execução utilizando o Nios/f (mm:ss:ms)
1	04:46:48	02:48:53
2	04:46:00	02:46:24
3	04:44:42	02:48:59
4	04:45:55	02:45:54
5	04:46:47	02:47:49
Tempo Médio	04:46:02	02:47:36

Fonte: Elaborada pelo autor

5 CONCLUSÃO

Este trabalho propôs a utilização de um FPGA (*Field Programmable Gate Array*) acoplado à uma câmera (Terasic TRDB-D5M) para o desenvolvimento de um sistema de detecção facial utilizando o algoritmo proposto por Viola-Jones. Foi utilizado o processador *softcore* Nios II da Intel/Altera, para realização do processamento do algoritmo, e demais componentes que compõem a arquitetura deste sistema.

Foi possível compreender que o desenvolvimento de um dispositivo embarcado, especificamente de arquitetura reconfigurável no contexto de processamento de imagens que necessite uma solução compacta e de alto poder computacional, é bastante promissor devido à sua flexibilidade de implementação. Pois permite centralizar em necessidades específicas de pequenos e grandes projetos. Devido aos avanços tecnológicos e computacionais, é necessário que aplicações sejam otimizadas e atualizadas, em alguns casos até mesmo ocasionando em alterações de funcionalidade.

Mediante os resultados apresentados, conclui-se que o algoritmo proposto por Viola-Jones foi bastante promissor no processo de detecção das imagens fornecidas ao mesmo, e que a utilização do Nios para execução deste algoritmo foi de grande importância por permitir sua customização e por permitir a abstração do *hardware*. Mesmo com os tempos de execução e suas médias, tanto nas versões *standard* (padrão) e *fast* (rápida), atingirem a casa de minutos, foi observado que o resultado esperado, ou seja a detecção de faces, foram alcançados.

Conclui-se, também, que mediante as execuções nos diversos cenários, a placa Terasic DE2-70 comportou-se de forma equilibrada, ou seja, não houveram discrepâncias consideráveis nos tempos de processamento com as diversas imagens que foram analisadas. O único cenário onde é observado variação no tempo foi na alteração da versão do Nios de *standard* para *fast* que resultou em uma diminuição, em média, de **02 minutos**, **03 segundos** e **07 milésimos**. Entende-se que melhorias são necessárias, como o aumento do *clock* utilizado no Nios e até mesmo a aceleração por *hardware* para que este tempo diminua ao espaço de segundos ou até mesmo de milésimos, e que cumpra possíveis necessidades e requisitos de aplicações, aonde for integrado.

5.1 Trabalhos futuros

Como trabalhos futuros é esperado elevar o desempenho do processo de detecção facial em imagens e sua aplicação em contexto real, como a utilização na área de tecnologias assistivas e na realização de frequência automática por meio do reconhecimento

facial. Sendo assim, as seguintes alterações e inclusões são necessárias:

- Analisar o algoritmo proposto por Viola-Jones com o intuito de verificar quais são as funções e trechos de código de maior custo computacional, e com isto realizar a aceleração por *hardware*;
- Alterar o *clock* utilizado no Nios por meio do PLL (*Phase Locked Loop*) ou até mesmo a utilização de oscilador externo; e
- Alterar e testar outros parâmetros utilizados nas configurações dos componentes com o objetivo de melhorar o desempenho dos mesmos.

REFERÊNCIAS

ANDRADE, M. C. et al. Rtev-ambiente de desenvolvimento de aplicações reconfiguráveis com o kernel de tempo real virtuoso. Universidade Federal de São Carlos, 2006.

ARAUJO, G. M. Algoritmo para reconhecimento de características faciais baseado em filtros de correlação. *Master's thesis. PPEE-UFRJ*, 2010.

BOBDA, C. *Introduction to reconfigurable computing: architectures, algorithms, and applications*. [S.l.]: Springer Science & Business Media, 2007.

BRAGA, L. F. Z. et al. *Sistemas de Reconhecimento Facial*. Tese (Doutorado) — UNIVERSIDADE DE SÃO PAULO, 2013.

BUKIS, A.; PROSCEVIČIUS, T.; RAUDONIS, V.; SIMUTIS, R. Survey of face detection and recognition methods. In: *Proceeding of International Conference on Electrical and Control Technologies, Kaunas, Lithuania*. [S.l.: s.n.], 2011. v. 20, p. 51–56.

CHE, M.; CHANG, Y. A hardware/software co-design of a face detection algorithm based on fpga. In: IEEE. *Measuring Technology and Mechatronics Automation (ICMTMA), 2010 International Conference on*. [S.l.], 2010. v. 1, p. 109–112.

DHARAN, S. V.; KHALIL-HANI, M.; SHAIKH-HUSIN, N. Hardware acceleration of a face detection system on fpga. In: IEEE. *Research and Development (SCOReD), 2015 IEEE Student Conference on*. [S.l.], 2015. p. 283–288.

FRADI, M.; YOUSSEF, W. E.; MOHSEN, M. The design of an embedded system (sopc) for an image processing application. In: IEEE. *Advanced Systems and Electric Technologies (IC_ASET), 2017 International Conference on*. [S.l.], 2017. p. 307–311.

FREUND, Y.; SCHAPIRE, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, Elsevier, v. 55, n. 1, p. 119–139, 1997.

HALL, E. C. *Journey to the moon: the history of the Apollo guidance computer*. Reston, Virginia, EUA: American Institute of Aeronautics and Astronautics, 1996. 196 p. ISBN 156347185X.

HALLINAN, C. *Embedded Linux Primer: A Practical Real-World Approach*. [S.l.]: Pearson Education India, 2007.

INTEL/ALTERA. *Bootable Embedded Systems for the DE0-nano Board*. 2013. Disponível em: <ftp://ftp.altera.com/up/pub/Intel_Material/14.0/Tutorials/DE0-Nano/Using_DE0-Nano_Flash.pdf>. Acesso em: 28 de Junho de 2017.

KERMANI, M. M.; ZHANG, M.; RAGHUNATHAN, A.; JHA, N. K. Emerging frontiers in embedded security. In: IEEE. *VLSI Design and 2013 12th International Conference on Embedded Systems (VLSID), 2013 26th International Conference on*. [S.l.], 2013. p. 203–208.

MAIA, J. G. R. *Detecção e reconhecimento de objetos usando descritores locais*. Tese (Doutorado) — Departamento de Computação, Centro de Ciências, Universidade Federal do Ceará, Porto Alegre, 2010.

MANNAY, Y. A. K.; ABID, M. Embedded system of the iris segmentation module. In: . Sousse, Tunisia: [s.n.], 2015. p. 1–4.

NUNES, É. d. M. Uma plataforma para agentes em hardware utilizando reconfiguração parcial. 2018.

OPENCV. *Cascade Classification*. 2018. Disponível em: <https://docs.opencv.org/2.4/doc/tutorials/objdetect/cascade_classifier/cascade_classifier.html>. Acesso em: 10 de Agosto de 2018.

PEDRONI, V. A. Eletrônica digital moderna e vhdl. *Rio de Janeiro, RJ: Elsevier*, 2010.

PYRGAS, L.; KALANTZOPOULOS, A.; ZIGOURIS, E. Design and implementation of an open image processing system based on nios ii and altera de2-70 board. *Journal of Engineering Science & Technology Review*, v. 9, n. 5, 2016.

RAVI, S.; RAGHUNATHAN, A.; KOCHER, P.; HATTANGADY, S. Security in embedded systems: Design challenges. *ACM Transactions on Embedded Computing Systems (TECS)*, ACM, v. 3, n. 3, p. 461–491, 2004.

SKLIAROVA, I.; FERRARI, A. B. Introdução à computação reconfigurável. *Electrónica e Telecomunicações*, v. 4, n. 1, p. 103–119, 2003.

SOLVAY. *Conferência de Solvay*. 1927. Disponível em: <https://pt.wikipedia.org/wiki/Ficheiro:Solvay_conference_1927.jpg>. Acesso em: 04 de Dezembro de 2018.

SOUKI, M.; BOUSSAID, L.; ABID, M. An embedded system for real-time traffic sign recognizing. In: IEEE. *Design and Test Workshop, 2008. IDT 2008. 3rd International*. [S.l.], 2008. p. 273–276.

STUDNIA, I.; NICOMETTE, V.; ALATA, E.; DESWARTE, Y.; KAANICHE, M.; LAAROUCHI, Y. Survey on security threats and protection mechanisms in embedded automotive networks. In: IEEE. *2013 43rd Annual IEEE/IFIP Conference on Dependable Systems and Networks Workshop (DSN-W)*. [S.l.], 2013. p. 1–12.

TERASIC. *DE2-70 User Manual*. 2008. Disponível em: <https://www.terasic.com.tw/attachment/archive/226/DE2_70_User_manual_v105.pdf>. Acesso em: 10 de Agosto de 2018.

TERASIC. *DE0-nano User Manual*. 2013. Disponível em: <https://www.altera.com/en_US/pdfs/literature/ug/DE0_Nano_User_Manual_v1.9.pdf>. Acesso em: 15 de setembro de 2017.

VIOLA, P.; JONES, M. Rapid object detection using a boosted cascade of simple features. In: IEEE. *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*. [S.l.], 2001. v. 1, p. I–I.