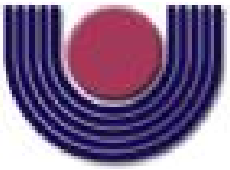


Unioeste - Universidade Estadual do Oeste do Paraná
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
Colegiado de Ciência da Computação
Curso de Bacharelado em Ciência da Computação

Utilizando co-projeto de Hardware e Software Para o Desenvolvimento de um Sistema de Reconhecimento Facial

Henrique Pedro de Oliveira

CASCAVEL
2016

Henrique Pedro de Oliveira

Utilizando co-projeto de Hardware e Software Para o Desenvolvimento de um Sistema de Reconhecimento Facial

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação, do Centro de Ciências Exatas e Tecnológicas da Universidade Estadual do Oeste do Paraná - Campus de Cascavel

Orientador: Prof. Marcio Seiji Oyamada

CASCAVEL
2016

Henrique Pedro de Oliveira

Utilizando co-projeto de Hardware e Software Para o Desenvolvimento de um Sistema de Reconhecimento Facial

Monografia apresentada como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação, pela Universidade Estadual do Oeste do Paraná, Campus de Cascavel, aprovada pela Comissão formada pelos professores:

Prof. Marcio Seiji Oyamada (Orientador)
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Adair Santa Catarina
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Edmar André Bellorini
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Thiago Berticelli Ló
IFPR

Cascavel, 7 de março de 2017

AGRADECIMENTOS

Agradeço ao meu pai, Cesar, minha família e amigos por me apoiarem durante o meu período de graduação na UNIOESTE. Agradeço aos professores que me ensinaram dentro e foram da sala de aula. Agradeço também ao meu orientador Prof. Marcio Oyamada por ter me guiado no desenvolvimento desse trabalho.

Lista de Figuras

Lista de Tabelas

Lista de Quadros

Lista de Abreviaturas e Siglas

CPU	*Central Processing Unit*
DMA	*Direct Memory Access*
E/S	Entrada/Saída
FIFO	*First In, First Out*
FPGA	*Field Programmable Gate Array*
GPIO	*General Purpose Input/Output*
GPU	*Graphics Processing Unit*
HDL	*Hardware Description Language*
HDMI	*High-Definition Multimedia Interface*
HSMC	*High Speed Mezzanine Card*
HW	*Hardware*
IP	*Intellectual Property*
IR	*InfraRed*
LCD	*Liquid-Crystal Display*
LE	*Logic Element*
LUT	*Look Up Table*
MAC	*Media Access Control*
PCA	*Principal Component Analysis*
PCIe	*Peripheral Component Interconnect Express*
PHY	*physical layer*
PLL	*phase-locked loop*
SATA	*Serial Advanced Technology Attachment*
SD	*Secure Digital*
SDRAM	*Synchronous dynamic random access memory*
SMA	*SubMiniature version A*
SO	*Sistema Operacional*
SRAM	*Static Random Access Memory*
SSRAM	*Synchronous Static Random Access Memory*
SW	*Software*
USB	*Universal Serial Bus*
VGA	*Video Graphics Array*
VHDL	VHSIC *Hardware Description Language*
VHSIC	*Very High Speed Integrated Circuits*

Lista de Símbolos

Γ	Face
Ψ	Face média
Φ	Diferença entre uma face e a média das faces
λ	Autovalor
ω	Peso de contribuição da face para um autovetor
Ω	Projeção da face no subespaço (conjunto de pesos)
ϵ	Distância entre a projeção da face de entrada e as projeções das faces de treinamento
θ_ϵ	Limiar da distância entre as projeções
P^T	Matriz P transposta

Sumário

Resumo

Este trabalho apresenta um sistema de reconhecimento facial que utiliza o método PCA (*Principal Component Analysis* - Análise de Componentes Principais), esse sistema foi desenvolvido parte em software e parte em hardware. O sistema foi implementado na plataforma DE2i-150 que possui um processador Intel Atom N2600 e um FPGA Cyclone IV GX. Foi implementado duas versões, uma puramente software e outra híbrida. A versão puramente software foi desenvolvida para averiguar qual etapa do reconhecimento é mais custosa em termos de tempo de execução. Os testes realizados mostraram que a projeção das faces no subespaço é a mais custosa levando 27,88 segundos para projetar 400 faces. Então, decidiu-se implementar as projeções em hardware visando acelerar a execução desta etapa. Os testes com a versão híbrida obteve um tempo de projeção de 241,76 segundos, desse tempo 34,62 segundos corresponde ao tempo de projeção no FPGA (sem contar as transferências de dados), 1,37 segundos corresponde ao tempo de transferência de dados entre a CPU e a SDRAM (incluindo o tempo de chamadas de sistema e *driver*) e o restante do tempo corresponde ao acesso da SDRAM pelo FPGA, o que corresponde a aproximadamente a 207 segundos. O menor desempenho do módulo em hardware pode ser atribuído ao excessivo número de acessos a SDRAM pelo FPGA, a diferença de frequência entre CPU e FPGA e o pouco paralelismo extraído na implementação em hardware.

Palavras-chave: lógica reconfigurável, análise de componente principal, desempenho

Capítulo 1

Introdução

Durante a década de 90 e início dos anos 2000 haviam casos onde os programadores não necessitavam otimizar o código caso o software não atendesse às especificações de desempenho. Em vez disso eles se apoiavam na Lei de Moore que dizia que a densidade de transistores dobrava a cada 18 meses; assim bastava esperar esse tempo e adquirir um novo processador fazendo com que o software atendesse as especificações [4].

Com o passar dos anos, o tempo para que a densidade de transistores dobre aumentou. Hoje em dia leva-se 24 meses ou mais para se comercializar uma nova geração de processadores e esse tempo tende a aumentar até um ponto onde não seja mais economicamente viável diminuir o tamanho do transistor. Algumas previsões feitas pela indústria indicam que tal situação ocorrerá entre 2020 e 2030 [4].

Se as companhias quiserem manter a sua competitividade terão de investir em outras soluções para aumentar o desempenho do processador e as principais apostas são os *multicores* e FPGAs (*Field Programmable Gate Array* - Arranjo de Portas Programáveis em Campo) [4]. Uma terceira solução que pode ser citada é a arquitetura 3-D de circuito integrado que consistem em empilhar camadas de *wafer* de silício, o que permitiria aumentar a densidade de transistores sem a necessidade de alterar o tamanho deles, porém há muitas questões para serem resolvidas [5].

Os *multicores* foram desenvolvidos em 2004 e utilizados até hoje. Com a diminuição do transistor, os processadores começaram a ter uma crescente perda de corrente e produção de calor. A solução adotada foi baixar a frequência de operação do processador e, portanto, para obter um crescente desempenho nominal empregou-se vários núcleos em um processador, criando-se os *multicores* [6].

A nova arquitetura trouxe desafios para os programadores que agora tinham que pensar em como construir aplicações que se beneficiassem de paralelismo sem aumentar o tempo de desenvolvimento ou diminuir sua qualidade. Algumas dessas aplicações são óbvias como multiplicação de matrizes (computação gráfica), multitarefa (transações de caixas eletrônicos, rastreamento de reserva em linhas aéreas) ou até mesmo buscas na Internet. Porém há certas aplicações que são sequenciais e divididas em etapas dependentes, para paralelizar uma aplicação dessas seria necessário prever o resultado da etapa anterior para decidir qual seria a próxima etapa, portanto não se beneficia do paralelismo [6].

Os FPGAs poderão ser usados de duas formas: uma delas seria usar o FPGA no lugar da CPU, pois com a diminuição do tamanho do transistor os erros que ocorrem durante a fabricação da CPU serão mais frequentes e com o FPGA será possível reorganizar o posicionamento dos recursos para que se evitem elementos defeituosos [4]. A segunda opção seria utilizar o FPGA em paralelo com a CPU; a configuração do FPGA pode ser feita de forma manual onde o desenvolvedor da aplicação deverá fornecer o código de configuração, ou de forma automática onde um módulo dentro da CPU analisaria o conjunto de operações do processo a ser executado a procura de operações custosas que poderiam ser executadas pelo arranjo configurável [7]. Em ambas as abordagens o problema do paralelismo persiste, porém devido a flexibilidade dos FPGAs é possível obter paralelismo em diferentes granularidades.

Várias aplicações podem se beneficiar de uma arquitetura CPU+FPGA, com o processamento dividido entre os dois chips; as instruções mais complexas seriam executadas no FPGA, que seria configurado com uma lógica específica para aquela aplicação, fazendo com que sua execução seja mais rápida do que na CPU. Entre alguns exemplos de aplicações pode-se citar a área aeroespacial, militar, automobilística, de *broadcast* e sistemas de segurança, que nos últimos anos tem se tornado de grande interesse devido a crescente ameaça de terrorismo.

Por esse motivo e por questões de controle de passageiros, os aeroportos de alguns países como Singapura, Reino Unido, Vietnã e o Brasil [8] instalaram sistemas de reconhecimento facial em portões eletrônicos que atendem viagens internacionais, assim toda vez que alguém usa um desses portões, é feita uma captura da imagem do rosto do indivíduo que será comparada com um banco de dados do órgão de segurança do país.

As técnicas de reconhecimento facial mais modernas costumam trabalhar com grande di-

mensionalidade de dados gerando um grande custo em termos de processamento. Ainda que sejam usadas técnicas de redução de dimensionalidade, como o PCA (*Principal Component Analysis* - Análise de Componentes Principais), este tipo de aplicação exige muito processamento [9], sendo possível de ser acelerado utilizando uma solução que combine CPU com FPGA.

1.1 Objetivos

O objetivo desse trabalho foi desenvolver um protótipo de sistema de reconhecimento facial baseado em PCA. Essa técnica foi escolhida pois o *framework* OpenCV possui bibliotecas para o reconhecimento facial que utiliza PCA, facilitando o desenvolvimento da pesquisa.

O protótipo foi dividido em dois módulos, software e hardware, onde no módulo hardware implementou-se uma das etapas com maior tempo de execução. A plataforma a ser utilizada neste trabalho é da Terasic, a DE2i-150, que possui uma CPU Intel Atom N2600 e um FPGA Altera Cyclone IV GX.

1.2 Organização do Texto

O Capítulo 2 apresenta a descrição do algoritmo PCA aplicado ao reconhecimento facial juntamente com um exemplo que procura mostrar a complexidade do algoritmo. No Capítulo 3 é apresentado a especificação da plataforma DE2i-150, como também a do FPGA Cyclone IV GX que compõe a plataforma. No Capítulo 4 é apresentado a proposta de implementação de um módulo de sistema de reconhecimento facial. O Capítulo 5 apresenta as conclusões e os trabalhos futuros.

Capítulo 2

Reconhecimento Facial

O livro *Handbook of Face Recognition* [10] define dois tipos distintos de sistema de reconhecimento facial, mas que podem ser combinados: verificação facial (ou autenticação) e identificação de face (ou reconhecimento). Verificação facial envolve comparação um-para-um, entre a imagem teste e o modelo facial da pessoa que se está tentando identificar. A identificação facial envolve comparação de um-para-vários, entre a imagem teste e todos os modelos da base de dados para determinar a identidade da pessoa representada na imagem teste. Um sistema de reconhecimento facial, seja ele de verificação ou identificação, é dividido em quatro etapas: detecção, alinhamento, extração de características e comparação.

A etapa de detecção extrai as faces de uma imagem gerando um conjunto de imagens, onde cada imagem contém uma única face. A etapa de alinhamento procura tornar mais preciso o processo de reconhecimento, visto que a detecção de face oferece uma estimativa grosseira da região e do tamanho das faces detectadas. Componentes faciais tais como olhos, nariz e boca são localizados por meio de pontos de localização; a imagem resultante da detecção de face é normalizada em relação a propriedades geométricas tais como tamanho e pose, utilizando transformações geométricas. A face pode, ainda, passar por mais processos de normalizações em relação a propriedades fotométricas como iluminação e escala de cinza.

A etapa de extração de características extrai as informações da imagem que possuem alta representatividade, visto que nem toda área da imagem de uma face possui características comparáveis. A etapa de comparação verifica se o vetor de características extraídas na etapa anterior é equivalente com os que estão na base de dados. O sistema tem como saída a identidade da face caso o resultado da comparação seja maior ou igual ao limiar mínimo, caso contrário o sistema acusará que a face não está registrada no sistema.

Uma das técnicas mais populares para o reconhecimento facial é o PCA, também conhecido como Transformada de *Karhunen-Loève*. É uma técnica estatística utilizada em muitas áreas da ciência e mais detalhes serão apresentados na Seção 2.1.

2.1 Análise de Componentes Principais

A ideia central do PCA [11] é reduzir a dimensionalidade de um conjunto de dados que possui um grande número de variáveis inter-relacionadas, e que podem portanto, serem representadas por um número menor de variáveis. No entanto, é necessário manter a representatividade do conjunto de dados. Essa redução é alcançada transformando o conjunto original em um novo conjunto de variáveis, onde os principais componentes são descorrelacionados e ordenados, de tal forma que os primeiros possuam a maior representatividade em relação ao conjunto original.

O PCA possui várias aplicações como detecção de anomalias numa rede de comunicações [12], classificação de imagens de satélites [13], cálculo de risco financeiro [14] e reconhecimento facial [15]. Neste último, o PCA participa da etapa de extração de características e comparação. Na etapa de extração de características encontram-se os principais componentes do conjunto de faces, ou seja, os autovetores da matriz de covariância das imagens de faces, onde cada coluna dessa matriz equivale a uma imagem de face, formando um espaço de alta dimensionalidade [15].

Esses autovetores podem ser considerados como um conjunto de características que diferenciam as imagens umas das outras. Cada ponto da imagem contribui mais ou menos com cada autovetor, que então pode ser representado como uma imagem que se assemelha a uma face, podendo ser chamado de autoface, do alemão *eigenface*. Cada face pode ser exatamente representada através da combinação linear das autofaces, e também podem ser representadas de forma aproximada utilizando apenas os melhores autovetores, que seriam aqueles que estão relacionados aos maiores autovalores [15]. O conjunto de autovetores forma um espaço de baixa dimensionalidade que pode ser chamando de autoespaço.

2.1.1 Cálculo de Autofaces

Seja o conjunto M de treinamento de imagens de faces $\Gamma_1, \Gamma_2, \Gamma_3, ..., \Gamma_m$, onde Γ_i possui dimensões de n-por-n pixels, trabalha-se como um vetor coluna de dimensão n^2, por exemplo

uma imagem de 100 pixels por 100 pixels dever ser representada como um vetor coluna de 10.000 elementos. Ao representar as imagens dessa forma, o conjunto M pode ser representado como uma matriz de dimensão n^2-por-m, um grande espaço amostral. Como imagens de faces são similares, os dados que caracterizam as faces não estarão randomicamente distribuídos, por isso o conjunto de imagens pode ser representado em um espaço amostral de menor dimensão [15].

O próximo passo é calcular a face média Ψ do conjunto que é definida por

$$\Psi = \frac{1}{m}\sum_{n=1}^{m}\Gamma_n \tag{2.1}$$

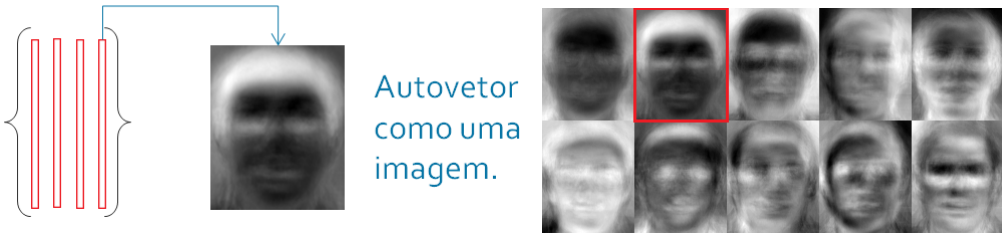
Em seguida calcula-se as diferenças Φ_i entre cada face Γ_i e a média Ψ que se da através de

$$\Phi_i = \Gamma_i - \Psi \tag{2.2}$$

O conjunto das diferenças é dado por $A = [\Phi_1, \Phi_2, ..., \Phi_m]$, onde A é uma matriz e as diferenças Φ_i são suas colunas. Esse conjunto de vetores é submetido a técnica do PCA, que consiste em gerar uma matriz C de covariância da qual se calcula os autovetores u_i e autovalores λ_i. A matriz de covariância é dada por

$$C = AA^T = \frac{1}{m}\sum_{n=1}^{m}\Phi_n\Phi_n^T \tag{2.3}$$

O conjunto de autovetores gerados a partir da matriz C forma uma matriz de transformação definida como $P = [u_1, u_2, ..., u_{m'}]$, onde $m' \leqslant m - 1$. Essa matriz é utilizada para projetar as imagens no autoespaço (espaço de baixa dimensionalidade). Como a matriz de covariância C é de dimensão n^2-por-n^2, logo, seus autovetores serão de tamanho n^2, sendo este o mesmo tamanho das imagens de faces. Se os autovetores forem transformados para matriz de dimensão n-por-n, poderão ser visualizados como uma imagem que lembra uma face (Figura 2.1).



(a) (b)

Figura 2.1: Conjunto de autovetores. (a) Matriz P de transformação formada por autovetores. (b) Autovetores representados como imagens.

Se o número de pixels de uma imagem for maior que o número de imagens ($m < n^2$), haverá $m - 1$ autovetores com significante representatividade, o restante será de autovetores associados a autovalores de 0 (zero). Os autovetores de dimensão n^2 podem ser obtidos através da transformação linear entre as diferenças Φ_i e os autovetores de uma matriz de dimensão m-por-m.

Considere os autovetores u_i e autovalores λ_i de C obedeçam a igualdade

$$Cu = \lambda u \tag{2.4}$$

Substitui-se C por AA^T:

$$AA^T u = \lambda u \tag{2.5}$$

Se for considerado que $u = Av$, onde v é um vetor coluna de tamanho m, e se for feita a substituição na equação 2.5 obtêm-se

$$AA^T Av = \lambda Av \tag{2.6}$$

Multiplicando ambos os lados da igualdade por A^{-1} obtêm-se

$$A^T Av = \lambda v \tag{2.7}$$

Portanto, seja L uma matriz de dimensão m-por-m definida pela Equação 2.8 e seus autovetores v_i, obtem-se os autovetores u_i de C através da Equação 2.9.

$$L = A^T A \tag{2.8}$$

$$u_i = Av_i, i = 1, 2, ..., m' \tag{2.9}$$

2.1.2 Reconhecimento facial utilizando autofaces

Cada autoface contribui com a caracterização de uma face presente no banco de faces, e o quão elas contribuem é definido por pesos ω_k que são calculados da seguinte forma

$$\omega_k = {u_k}^T(\Gamma_i - \Psi), k = 1, ..., m \tag{2.10}$$

Onde m é o número de autovetores escolhidos para representar o conjunto de faces. O conjunto desses pesos nada mais é que a projeção de uma imagem no autoespaço que dada por

$$\Omega_i = P^T(\Gamma_i - \Psi) \tag{2.11}$$

Para se determinar a que indivíduo uma face de entrada Γ pertence, deve-se projetar a face de entrada no autoespaço e encontrar a menor distância euclidiana ϵ_i entre as projeções das imagens que estão no banco de faces. O cálculo da distância se dá por:

$${\epsilon_i}^2 = \| \ (\Omega - \Omega_i) \ \|^2 \tag{2.12}$$

A imagem de entrada pertencerá ao indivíduo cuja imagem de face projetada ofereça a menor distância. Além disso, essa distância dever estar abaixo de um limar θ_ϵ, caso contrário a face será classificada como não pertencente a base de faces.

2.2 Resumo do processo de reconhecimento com PCA

O algoritmo de reconhecimento facial que utiliza o PCA é formado por duas etapas, a de treinamento (ou extração de características) e a de reconhecimento (ou comparação). A etapa de treinamento é formada por oito passos, essa etapa não precisa ser executada toda vez que se deseja reconhecer uma face. Os passos da etapa de treinamento estão listados abaixo:

1. Organizar as imagens de faces em vetores colunas, formando assim uma matriz M onde cada coluna representa uma face (Figura 2.2);

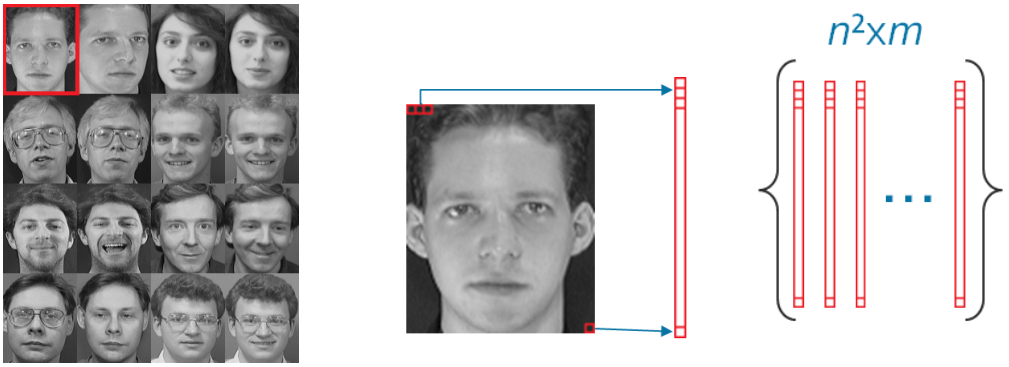


Figura 2.2: Faces de treinamento [1]. (a) Conjunto de amostras. (b) Transformação de uma face, de matriz para vetor. (c) Matriz M onde cada coluna representa uma face.

2. Calcular a face média através da Equação 2.1, a face média também é representada como um vetor colunar (ver Figura 2.3);

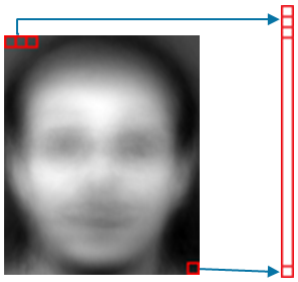


Figura 2.3: Imagem da face média.

3. Subtrair a face média de cada face do conjunto de treinamento gerando uma matriz A (ver Figura 2.4);

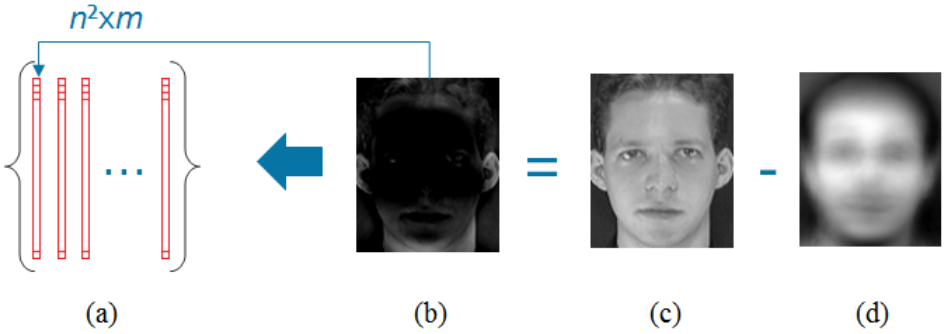


Figura 2.4: Cálculo das diferenças. (a) Matriz A de diferenças. (b) Diferença Φ_i. (c) Face Γ_i. (d) Face média Ψ.

4. Calcular a matriz L através da Equação 2.8 (ver Figura 2.5);

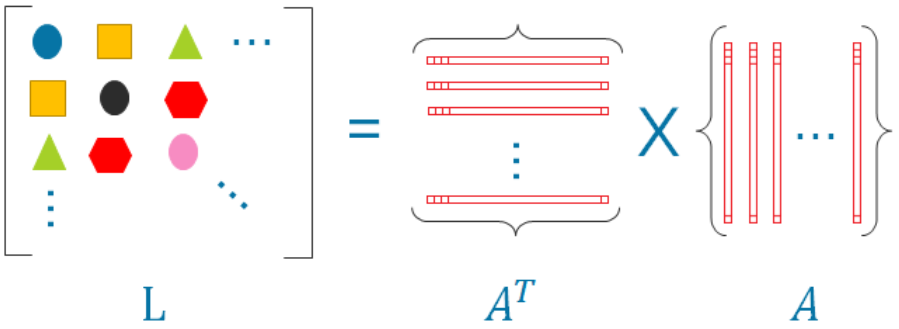


Figura 2.5: Cálculo da matriz L.

5. Calcular os autovetores v_i de L;

6. Calcular os autovetores de C através da Equação 2.9 (ver Figura 2.6);

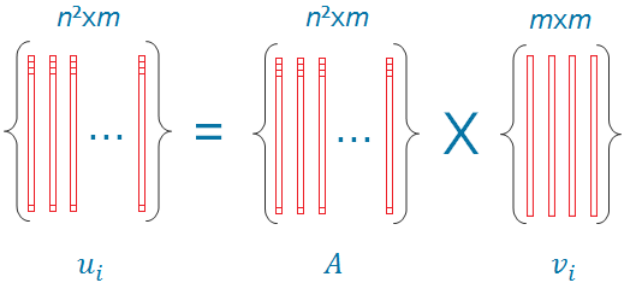


Figura 2.6: Cálculo dos autovetores da matriz C através dos autovetores da matriz L .

7. Montar a matriz de transformação P com os autovetores de C relacionados aos m' maiores autovalores, onde $m' \leqslant m$;

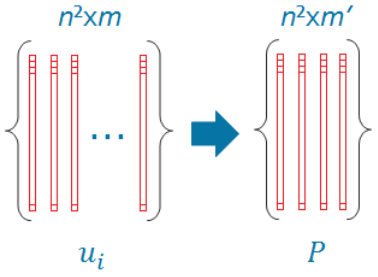


Figura 2.7: Matriz formada a partir dos autovetores da Matriz C.

8. Projetar no subespaço todas as imagens de faces usadas no treinamento (ver Figura 2.8).

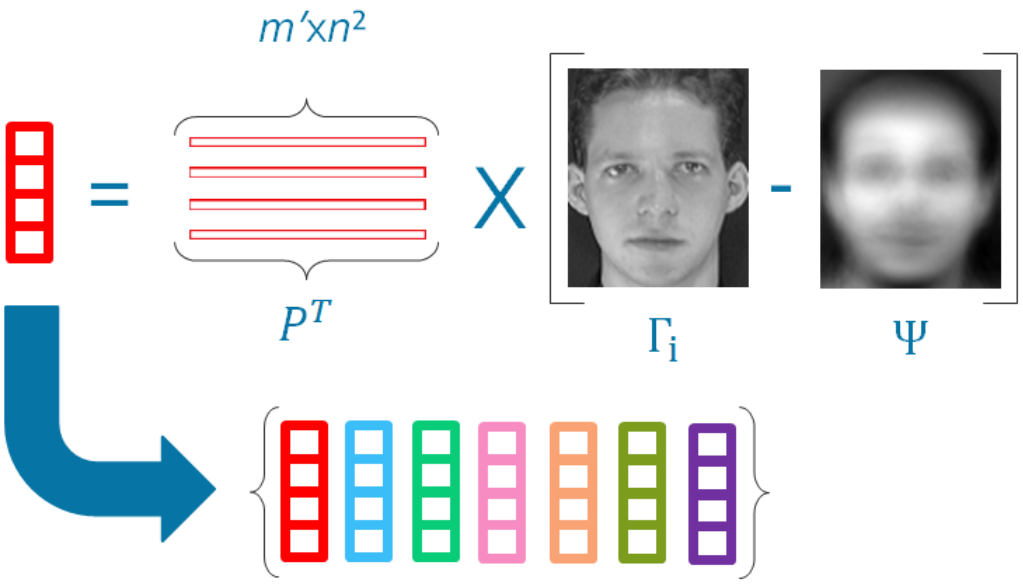


Figura 2.8: Cálculo das projeções das faces de treinamento no subespaço.

A etapa de reconhecimento é formada por três passos que estão listados abaixo:

1. Projetar a face de entrada Γ no subespaço de acordo com a Equação 2.11;

2. Calcular a distância euclidiana entre a projeção da face de entrada com todas as projeções das faces de treinamento de acordo com a Equação 2.12, a face de entrada pertencerá a pessoa cuja projeção de sua face forneceu a menor distância (ver Figura 2.9);

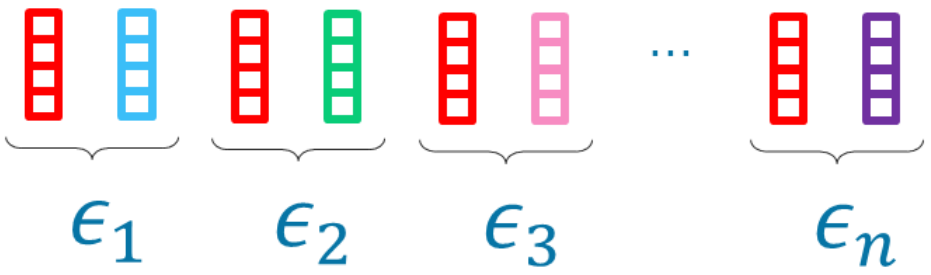


Figura 2.9: Comparações entre a projeção da face de entrada e as projeções das faces de treinamento.

3. Retornar a identidade da Face.

Capítulo 3

Plataforma DE2i-150

A plataforma DE2i-150 (Figura 3.1), produzida pela Terasic, é uma solução embarcada que combina um processador Intel Atom N2600 com o FPGA Cyclone IV GX da Altera, que será detalhado na Seção 3.1. O processador Intel Atom e o FPGA são ligados através de dois barramentos PCIe de alta velocidade [16].

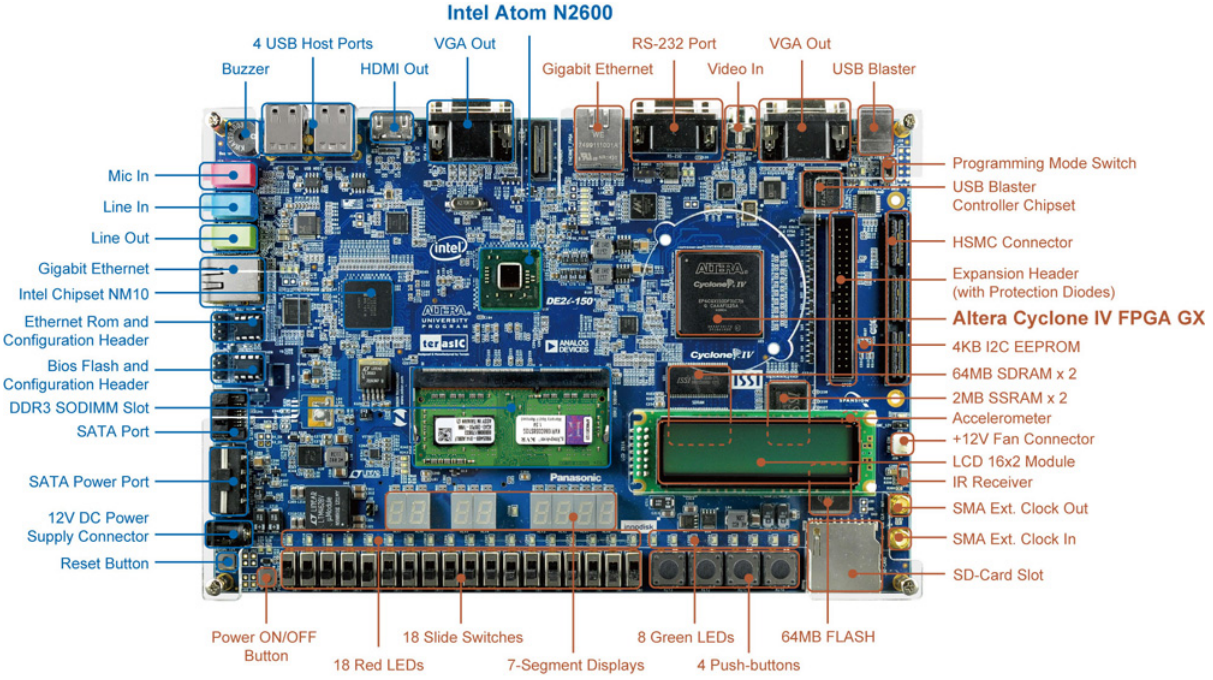


Figura 3.1: Plataforma de desenvolvimento DE2i-150 [2].

O Intel Atom N2600 é um processador de dois núcleos que funciona a uma frequência máxima de 1,6GHz e possui uma cache L2 de 1MB e um conjunto de instruções de 64-bit. Construído para aplicações de alto desempenho e baixo consumo, o N2600 também possui um

GPU integrado. A tabela 3.1 mostra os recursos que foram distribuídos entre CPU e FPGA.

Tabela 3.1: Alocação de recursos da placa entre CPU e FPGA

Recursos	**CPU**	**FPGA**
Porta VGA	1	1
Porta HDMI	1	0
Porta USB	4	1
Porta Ethernet	1	1
Porta SATA	1	0
Porta de alimentação da porta SATA	1	0
Porta RS-232	0	1
Conector HSMC	0	1
Porta GPIO	0	40
Memória RAM	2GB	0GB
Memória SDRAM	0MB	128MB
Memória Flash	0MB	64MB
Memória SSRAM	0MB	2MB
Acelerômetro	0	1
Módulo LCP 16x2	0	1
Receptor IR	0	1
Extensão SMA	0	2
Slot de cartão SD	0	1
Push-button	0	4
Switch	0	18
Display de sete segmentos	0	8
Buzzer	1	0
Chipset Intel NM10 Express	1	0
Intel Centrino Wireless-N 135	1	0
Realtek ALC272VA3-GR	1	0

3.1 Estrutura Cyclone IV EP4CGX150

Todas as informações desta seção foram obtidas de Cyclone IV Device Handbook [17].O Cyclone IV GX (EP4CGX150) é o FPGA com maior densidade da família Cyclone IV, possui aproximadamente 150 mil elementos lógicos. O EP4CGX150 também possui *hard* IPs como controlador PCIe, 720 blocos de memória M9k com 9 Kbits de capacidade de armazenamento cada, resultando num total de 6.480 Kbits de memória *on-chip*, e 360 multiplicadores 18x18. Outros recursos são descritos na Tabela 3.2.

Tabela 3.2: Recursos do Cyclone IV GX

Recursos	Quantidade
Elementos lógicos (LEs)	149.760
Memória embarcada (Kbits)	6.480
Multiplicador embarcado de 18x18	360
PLLs de propósito geral	4
PLLs de multipropósito(Gbits)	4
Redes de relógio global	30
Transceivers de alta velocidade	8
Taxa máxima de dados de um *Transceivers*	3.125
Bloco de comunicação PCIe (PIPE)	1
Portas de E/S	475

É possível configurar as portas dos blocos M9K em *single*, *dual*, e *true dual*, como também *buffers* FIFO. O modo *dual* permite uma leitura e uma escrita ao mesmo tempo enquanto que o modo *true dual* permite qualquer combinação de leitura e escrita: duas leituras, duas escritas ou leitura e escrita.O Cyclone IV usa células de SRAM para armazenar os dados de configuração, que devem ser carregados toda vez que a placa é ligada. A largura de dados também pode ser configurada de acordo com a Tabela 3.3. Os multiplicadores podem implementar um multiplicador 18x18 ou 9x9 em um único bloco.

Tabela 3.3: Largura de dados para os blocos M9K do Cyclone IV GX

Modo	Configuração de largura de dados
Single e *dual*	x1, x2, x4, x8/9, x16/18, e x32/36
True dual	x1, x2, x4, x8/9, e x16/18

O Cyclone IV GX incorpora um bloco de PCIe (PIPE) com largura de dados de x1, x2, x4, além disso este *hard* IP possui camadas de conexão de PHY-MAC[1], de dados e de transação. O bloco de IP ainda suporta configurações de *root-port* e *end-point*.

3.1.1 Elementos lógicos

O núcleo do EP4CGX150 é formado por elementos lógicos (LEs) que são responsáveis pelas operações lógicas necessárias para o FPGA funcionar, esses LEs são formados por LUTs (*Look Up Table*) de quatro entradas e também por registradores e multiplexadores.

[1]Componente que realiza a conexão de rede entre a camada física e a de enlace.

É possível configurar os registradores em cada LE para funcionar como um flip-flop do tipo D, T, JK ou SR. Cada registrador possui entradas de dado, relógio, ativador de relógio, e reset. Qualquer sinal que use a rede global de relógio, pinos de E/S de proposito geral, ou qualquer logica interna pode usar os sinais de controle de relógio e reset do registrador. Para funções combinacionais, a saída da LUT ignora o registrador e vai direto para as saídas do LE.

Cada LE tem quatro entradas e três saídas que se conectam aos canais de roteamento locais, de linha, e de coluna. Duas das saídas do LE conectam-se aos canais de linha ou coluna ligando diretamente as conexões de roteamento, enquanto que a terceira saída se conecta a recursos locais. Isso permite que a LUT se conecte com uma saída enquanto que o registrador se conecta a outra. Essa funcionalidade, chamada empacotamento de registradores, melhora a utilização do dispositivo, pois o dispositivo pode usar o registrador e a LUT para funções não relacionadas.

Os LEs do Cyclone IV GX possuem dois modos de operação: modo normal e modo aritmético. O modo normal é utilizado em aplicações logicas genéricas. O modo aritmético é utilizado para implementar somadores, contadores, acumuladores e comparadores.

3.2 Comunicação entre FPGA e Processador

O processador Atom e o FPGA se comunicam utilizando a interface PCIe, assim, há espaços endereçáveis dentro do FPGA que são comuns a ambos. Requisições de acesso a esses endereços são direcionados para a memória SDRAM do FPGA ou tratados por um componente implementado na lógica customizável.

O processador consegue acessar esses espaços por meio da interface PCIe, enquanto que esses espaços são acessíveis em todo FPGA. Por exemplo, se o Atom escrever um valor "0xfeedfeed"no endereço "0x00000000"através do barramento PCIe, um módulo lógico receberá essa requisição e poderá armazenar o valor em registradores. O número de registradores é definido pelo projetista na descrição SystemVerilog do componente.

Para que a comunicação entre processador e FPGA funcione deste modo, a aplicação precisa possuir um módulo escravo (que está representado na Figura 3.2 como Lógica Customizada) implementado no FPGA que seja capaz de receber e armazenar dados. A parte software da aplicação pode usar esses registradores para enviar dados ou sinais de controle visando o início ou o término de um processo na parte hardware. Outras funções podem ser atribuídas para os

registradores ficando a cargo do programador.

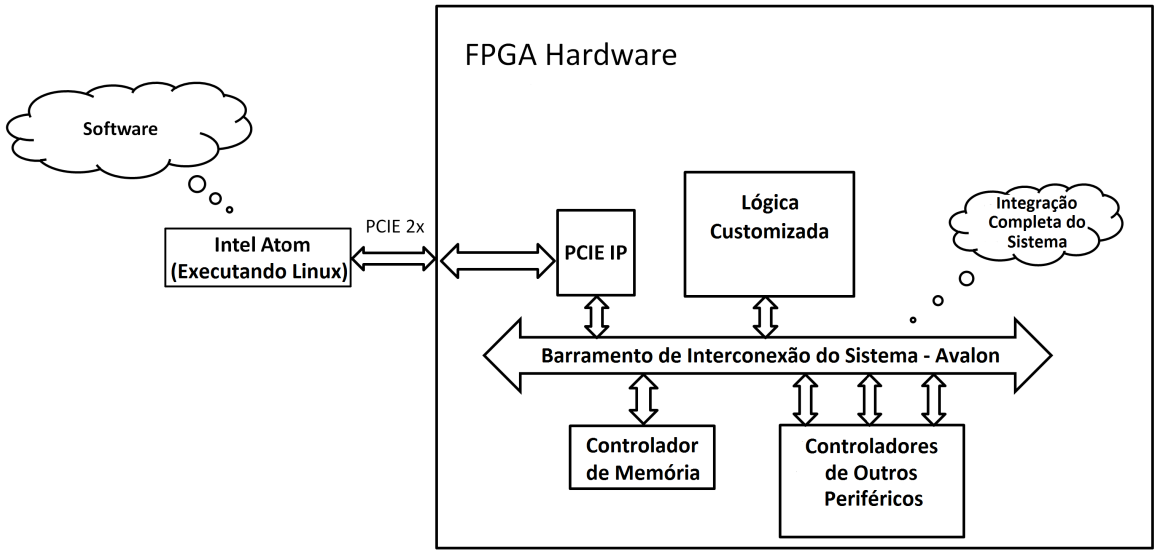


Figura 3.2: Comunicação processador e FPGA na plataforma DE2i-150. Adaptado de [3]

Para auxiliar o desenvolvimento de co-projetos de HW e SW a Altera fornece os *drivers* PCIe e a Terasic uma biblioteca na linguagem C que realiza a comunicação de uma aplicação com o FPGA. O *driver* PCIe permite que o processador se comunique com o FPGA como se fosse leitura e escrita de arquivos [3].

Para exemplificar o funcionamento do co-projeto do HW e SW toma-se como exemplo demo_master_slave fornecido por DE2i-150 at Purdue [3] que é um co-projeto de HW e SW simples que tem como objetivo mostrar como pode ser feita a troca de dados entre CPU, FPGA e SDRAM. Os códigos completos podem ser encontrados no repositório do GitHub no seguinte endereço: https://github.com/PurdueAtomDev/de2i150/tree/master/demo_master_slave.

Para que a comunicação possa ser realizada é necessário carregar a biblioteca PCIe na memória (Quadro 3.1), o trecho de código responsável por esse procedimento pode ser visto a partir da linha 29 a 34. Em seguida é preciso abrir o *file handler* para a biblioteca (linhas 36 a 41) visto que o SO interpreta o acesso ao PCIe como leitura e escrita de arquivos.

Na linha 44 é chamado o subprograma `test32` que realiza teste de leitura e escrita no módulo customizado. O subprograma `test32` (ver Quadro 3.2) escreve no módulo escravo um valor teste através do subprograma `PCIE_Write32` e em seguida lê-se de volta o dado através do subprograma `PCIE_Read32` para que possa fazer uma comparação de igualdade,

```
25 int main(void) {
26   void *lib_handle;
27   PCIE_HANDLE hPCIe;
28
29   lib_handle = PCIE_Load();    // Dynamically Load the PCIE library
30   if (!lib_handle)
31   {
32     printf("PCIE_Load failed\n");
33     return 0;
34   }
35
36   hPCIe = PCIE_Open(0,0,0);//Every device is a like a file in UNIX. Opens
       the PCIE device for reading/writing
37   if (!hPCIe)
38   {
39     printf("PCIE_Open failed\n");
40     return 0;
41   }
42
43   //test CRA
44   test32(hPCIe, CRA);      // Test the Configuration Registers for reads and
       writes
45   PCIE_Write32( hPCIe, pcie_bars[0], CRA, START_BYTE);
46   //test SDRAM
47   testDMA(hPCIe,SDRAM);      // Test the SDRAM for reads and writes
48
49   PCIE_Write32( hPCIe, pcie_bars[0], CRA, STOP_BYTE);
50   printf("\nPush up SW[16] to view data stored in SDRAM and use SW[3:0] to
       select different addresses.\n");
51   return 0;
52 }
```

Quadro 3.1: Subprograma main.

```
55 void test32( PCIE_HANDLE hPCIe, DWORD addr ) {
56   BOOL bPass;
57   DWORD testVal = 0xf;
58   DWORD readVal;
59
60   WORD i = 0;
61   for (i = 0; i < 16 ; i++ ) {
62     bPass = PCIE_Write32( hPCIe, pcie_bars[0], addr, testVal);
63     if (!bPass) {
64       printf("test FAILED: write did not return success");
65       return;
66     }
67
68     bPass = PCIE_Read32( hPCIe, pcie_bars[0], addr, &readVal);
69     if (!bPass) {
70       printf("test FAILED: read did not return success");
71       return;
72     }
73     printf("Testing register %d at addr %x with value %x: ", i, addr,
     testVal);
74
75     if (testVal == readVal) {
76       printf("Test PASSED: expected %x, received %x\n", testVal, readVal);
77     } else {
78       printf("Test FAILED: expected %x, received %x\n", testVal, readVal);
79     }
80     testVal = testVal + 1;
81     addr = addr + 4;
82   }
83   return;
84 }
```

Quadro 3.2: Subprograma test32.

o subprograma faz isso dezesseis vezes incrementando o endereço e o valor do dado a cada iteração.

Quando `PCIE_Write32` é executado na linha 45 do subprograma main ele escreve o valor `START_BYTE` no módulo escravo dando início a cópia dos dados que estão armazenados no módulo para a SDRAM. Então na linha 47 é chamado o subprograma `testDMA` que irá ler os dados armazenados na SDRAM através do subprograma `PCIE_DmaRead` e verificar se estão corretos. Na linha 49 é escrito o valor `STOP_BYTE` no módulo escravo. No Quadro 3.3 é descrito o subprograma `testDMA`.

A descrição do hardware que implementa o módulo escravo é apresentado no Quadro 3.4, ele é um processo sequencial ativado na subida do relógio. A primeira coisa que o processo faz é verificar se o sinal de reset está ativado, se não estiver ele é verificado se o sinal de

```
87  void testDMA( PCIE_HANDLE hPCIe, DWORD addr) {
88    BOOL bPass;
89    DWORD testArray[MAXDMA];
90    DWORD readArray[MAXDMA];
91
92    WORD i = 1;
93    testArray[0] = START_BYTE;
94    while ( i < MAXDMA ) {
95      testArray[i] = i  + 0x0f;
96      i++;
97    }
98
99    bPass = PCIE_DmaRead(hPCIe, addr, readArray, MAXDMA * RWSIZE );
100   if (!bPass){
101     printf("test FAILED: read did not return success");
102     return;
103   }
104
105   i = 0;
106   while ( i < MAXDMA ) {
107     printf("Testing SDRAM at addr %x: ", addr);
108     if (testArray[i] == readArray[i]) {
109       printf("Test PASSED: expected %x, received %x\n", testArray[i],
      readArray[i]);
110     } else {
111       printf("Test FAILED: expected %x, received %x\n", testArray[i],
      readArray[i]);
112     }
113     i++;
114     addr = addr + 4;
115   }
116   return;
117 }
```

Quadro 3.3: Função testDMA.

```
61 always_ff @ ( posedge clk ) begin
62   if(!reset_n) begin
63     slave_readdata <= 32'h0;
64     csr_registers <= '0;
65   end else begin
66     if(slave_write && slave_chipselect && (slave_address >= 0) && (
     slave_address < NUMREGS)) begin
67       csr_registers[slave_address] <= slave_writedata;  // Write a value to
      a CSR register
68     end else if(slave_read && slave_chipselect  && (slave_address >= 0) &&
     (slave_address < NUMREGS)) begin
69       slave_readdata <= csr_registers[slave_address];  // reading a CSR
     Register
70     end
71   end
72 end
```

Quadro 3.4: Módulo escravo.

leitura `slave_write` está, para que possa armazenar o valor que vem da CPU no registrador `csr_registers`. Caso o sinal `slave_write` não esteja ativado ele verifica se o sinal de leitura `slave_read` está ativado, para que possa ler o dado do registrador para a CPU.

O módulo mestre é uma máquina de estados que lê os dados dos registradores do módulo escravo e os copia para a SDRAM, essa máquina é dividida em três processos: um sequencial e dois combinacionais. A principal função do processo sequencial é realizar a troca de estados (Quadro 3.5), além disso, esse processo é responsável por atualizar o valor de sinais de endereço, índice de registradores e dados. Se o sinal de reset está ativado serão atribuídos valores iniciais aos sinais anteriormente citados. Esse processo também é responsável por armazenar os dados recém lidos da SDRAM em registradores dentro do módulo mestre (linha 89).

No Quadro 3.6 está representado o trecho em SystemVerilog que implementa o processo de lógica combinacional responsável por determinar o próximo estado, endereço e dado. O diagrama da máquina de estados pode ser visto na Figura 3.3 e as condições de transição na Tabela 3.4. Na linha 102 e na 104 a máquina de estados verifica se houve a escrita do valor `START_BYTE` ou `STOP_BYTE` no registrador, determinando o próximo estado, essa verificação ocorre no estado `IDLE`.

```
always_ff @ ( posedge clk ) begin
  if (!reset_n) begin
    address <= SDRAM_ADDR;
    reg_index <= 0;
    state <= IDLE;
    wr_data <= 0;
    read_data <= 32'hFEEDFEED;
    read_data_registers <= '0;
  end else begin
    state <= nextState;
    address <= nextAddress;
    reg_index <= nextRegIndex;
    wr_data <= nextData;
    if(new_data_flag)
      read_data_registers[reg_index] <= nextRead_data;
  end
end
```

Quadro 3.5: Processo de troca de estado.

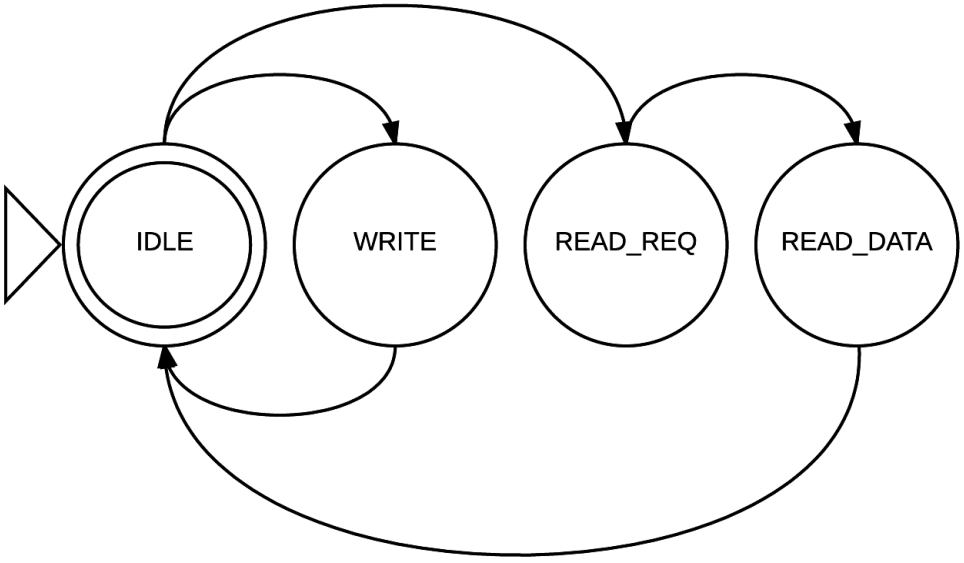


Figura 3.3: Diagrama de estados do hardware.

Tabela 3.4: Condições das transições de estados de demo_master_slave.

Transição	Condição
IDLE:WRITE	csr_registers = START_BYTE
IDLE:READ_REQ	csr_registers = STOP_BYTE
WRITE:IDLE	master_waitrequest = 0
READ_REQ:READ_DATA	master_waitrequest = 0
READ_DATA:IDLE	master_readdatavalid = 1

No estado `WRITE` o índice de registrador e o endereço são incrementados e o próximo estado será o `IDLE` (Quadro 3.6). Como `csr_registers[0]` possui o valor `START_BYTE` e `reg_index` é menor que o número de registradores (que no caso desse exemplo são oito), o próximo estado será `WRITE` novamente, isso ocorrerá até que `reg_index` seja igual ou maior que `NUMREGS` (ou `csr_registers[0]` com valor diferente de `START_BYTE`). Esse comportamento faz com que todos os registradores possam ser acessados na terceira máquina de estados.

Para que a máquina vá para o estado `READ_REQ` é preciso que seja escrito o valor `STOP_BYTE` em `csr_registers[0]` dando início a leitura da SDRAM pelo módulo mestre. Neste estado `reg_index` e `address` são decrementados e o próximo estado será o `READ_DATA`.

No estado `READ_DATA` o sinal `nextRead_data` recebe o valor recém lido da SDRAM (linha 130) e é atribuído ao sinal `new_data_flag` o valor `1`, fazendo com que o valor recém lido seja escrito no registrador do módulo mestre (linha 91). `IDLE` é definido como próximo estado para, assim, fechar o ciclo de leitura.

O terceiro processo, que pode ser visto no Quadro 3.7, é responsável por controlar os sinais da interface do módulo mestre. Nesse processo, durante o estado `WRITE` é atribuído o valor `1`(um) ao sinal `master_write` indicando que o módulo deve fazer uma escrita na SDRAM no endereço atribuído ao sinal `master_address` com o dado de `master_writedata`. No estado `READ_REQ` acontece algo similar, onde o sinal `master_read` indica que deve ser feita uma leitura da SDRAM, o dado lido será armazenado no sinal `master_readdata` que será usado no processo de lógica de próximo estado.

```
94  always_comb begin
95    nextState = state;
96    nextAddress = address;
97    nextRegIndex = reg_index;
98    nextRead_data = master_readdata;
99    new_data_flag = 0;
100   case( state )
101     IDLE : begin
102       if ( csr_registers[0] == START_BYTE && reg_index < NUMREGS) begin
103         nextState = WRITE;
104       end else if ( csr_registers[0] == STOP_BYTE && address >= SDRAM_ADDR)
      begin
105         nextState = READ_REQ;
106       end
107     end
108     WRITE: begin
109       if (!master_waitrequest) begin
110         nextRegIndex = reg_index + 1;
111         nextAddress = address + 4;
112         nextState = IDLE;
113       end
114     end
115     READ_REQ : begin
116       if (!master_waitrequest) begin
117         nextState = READ_DATA;
118         nextAddress = address - 4 ;
119         nextRegIndex = reg_index - 1;
120       end
121     end
122     READ_DATA : begin
123       if ( master_readdatavalid) begin
124         nextRead_data = master_readdata;
125         nextState = IDLE;
126         new_data_flag =1;
127       end
128     end
129   endcase
130 end
```

Quadro 3.6: Processo de lógica de próximo estado.

```
133 always_comb begin
134   master_write = 1'b0;
135   master_read = 1'b0;
136   master_writedata = 32'h0;
137   master_address = 32'hbad1bad1;
138   case(state)
139     WRITE : begin
140       master_write = 1;
141       master_address =  address;
142       master_writedata = csr_registers[reg_index];
143     end
144     READ_REQ : begin
145       master_address = address;
146       master_read = 1;
147     end
148   endcase
149 end
```

Quadro 3.7: Processo de lógica de saída.

Capítulo 4

Implementação

Este capítulo descreve a implementação do sistema de reconhecimento facial utilizado no trabalho. Inicialmente foi implementado um sistema em C++ de reconhecimento facial utilizando como base o exemplo e as bibliotecas de reconhecimento facial fornecido pelo *framework* OpenCV 2.4.9 [18]. Em relação ao exemplo, a única parte do código que foi utilizada foi o de carregamento das imagens de treinamento utilizando um arquivo .csv contendo o endereço de todas as imagens.

Para medir o tempo gasto em certo trecho de código foi utilizando as funções de `monotonic_clock` da biblioteca chrono [19]. Para que fosse possível utilizar essas funções foi preciso reescrever as classes `FaceRecognizer` e `PCA` do OpenCV utilizadas no reconhecimento facial. Os testes realizados marcaram o tempo de execução de trechos do código referentes a transformação das imagens em vetores coluna, cálculo da matriz de covariância, cálculo dos autovetores e autovalores, normalização dos dados, projeção das faces no subespaço e a etapa de reconhecimento.

A Tabela 4.1 mostra a média dos resultados dos testes do módulo de sistema de reconhecimento facial executados na placa DE2i-150 utilizando somente o processador Atom. A base de dados utilizada nos testes foi The ORL Database of Faces [1] que é formado por quatrocentas imagens de faces (imagens em tons de cinza, apenas um canal de cor, de tamanho 92x112 pixels) de quarenta indivíduos diferentes, as fotos foram tiradas em diferentes condições de iluminação, expressões faciais e outras características como barba e óculos.

Tabela 4.1: Tempos de execução de trecho de código do sistema de reconhecimento facial puramente software

Trecho de código	**Tempo (s)**	**Porcentagem (%)**
Transformação em vetor coluna	0,0840	0,09
Cálculo da matriz de covariância	21,8567	24,12
Cálculo de autovetores e autovalores	20,5443	22,67
Normalização dos dados	20,1932	22,28
Projeção no subespaço (400 amostras)	27,8774	30,76
Total da etapa de treinamento	90,5556	99,92
Reconhecimento	0,0730	0,08
Total	90,6286	100

Levando em consideração os dados da Tabela 4.1 e o tempo disponível para a implementação do projeto, optou-se por implementar a parte referente a projeção das faces no subespaço em FPGA, pois a projeção no subespaço tem o maior tempo de execução e é mais fácil de implementar do que cálculo de autovetores, assim, sua implementação levaria menos tempo e se encaixaria no cronograma.

4.1 O sistema

Inicialmente pretendia-se implementar um reconhecedor facial utilizando dados inteiros em vez de ponto flutuante, pois isso facilitaria o desenvolvimento do módulo hardware, porém ao testar a versão inteira o sistema perdeu a acurácia, ou seja não identificava corretamente um indivíduo. Foram feitos testes com uma versão inteira puramente software a fim de verificar se o erro de reconhecimento estava no módulo hardware ou se era por causa da perda de precisão devido as variáveis serem do tipo inteiro, a segunda suposição provou ser verdadeira.

Tendo em vista os problemas com a versão inteira optou-se por implementar um versão em ponto flutuante de 32bits utilizando os módulos de aritmética de ponto flutuante da Altera, com essa mudança o tamanho dos arranjos de dados teve de mudar, pois antes os pixels eram representados em palavras de 8bits. Também houve pequenas mudanças na máquina de estados, visto que os passos para a leitura e escrita são os mesmos nas duas versões. No que tange à máquina de estados as principais mudanças ocorreram no cálculo de endereço; outras mudanças também ocorreram nos módulos que serviam como registradores para os dados utilizados nos cálculos e no módulo responsável pelo cálculo que passou a utilizar os IPs da Altera.

Na versão inteira, módulo que armazena a face e a face média utilizava iteradores para determinar em que posição do arranjo o dado seria armazenado, na versão ponto flutuante a lógica de iteradores foi substituída pela lógica FIFO. O módulo que armazena os autovetores e o módulo que controla a saída de dados não mudaram a sua lógica, apenas foi ajustado as dimensões dos arranjos. Na versão inteira, o módulo que calcula a projeção executava 8 multiplicações básicas que levam um ciclo para serem executadas, na versão ponto flutuante são feita 8 multiplicações utilizando módulos de aritmética de ponto flutuante da Altera, esses módulos levam mais de um ciclo para realizar os cálculos. Os detalhes dos módulos da versão de ponto flutuante serão apresentados mais adiante.

O sistema puramente software utiliza 400 autovetores para realizar as projeções, porém o reconhecimento pode ser feito com um número menor de autovetores. Conforme testes realizados na versão em SW foi escolhido um número de 160 autovetores para fazer o reconhecimento a fim de economizar recursos do FPGA. O número de autovetores foi escolhido através de testes de reconstrução de projeções com diferentes quantidades de autovetores e com 160 autovetores foi possível atingir uma reconstrução bem próxima da original.

O Cyclone IV possui apenas 149.760 LEs, quantidade que não é o suficiente para armazenar as faces, a face média e os autovetores em registradores dentro do FPGA, por isso os dados deverem ser armazenados na SDRAM e lidos pelo FGPA a medida que fosse necessário. Depois de implementar algumas versões do módulo HW com diferentes configurações de tamanho de arranjos e matrizes verificou-se que a maior configuração que poderia ser sintetizada no FPGA seria dividir a face média e cada face em 322 partes com 32 elementos.

Os autovetores são divididos entre si em 20 grupos de 8 autovetores e como cada autovetor possui o mesmo tamanho de uma face eles também são divididos em 322 partes com 32 elementos. Desta forma, a matriz de autovetores é dividida em outras matrizes de menor tamanho com dimensões de 8 linhas por 32 colunas. São carregados 8 autovetores para que seja feito 8 multiplicações em paralelo. Tentou-se implementar uma configuração que utiliza-se mais autovetores, porém ela não compilava devido a falta de interconexões.Diferentemente das faces e autovetores as projeções não são divididas em grupo. A Figura 4.1 ilustra essa organização, onde R_n é a projeção da face n no subespaço.

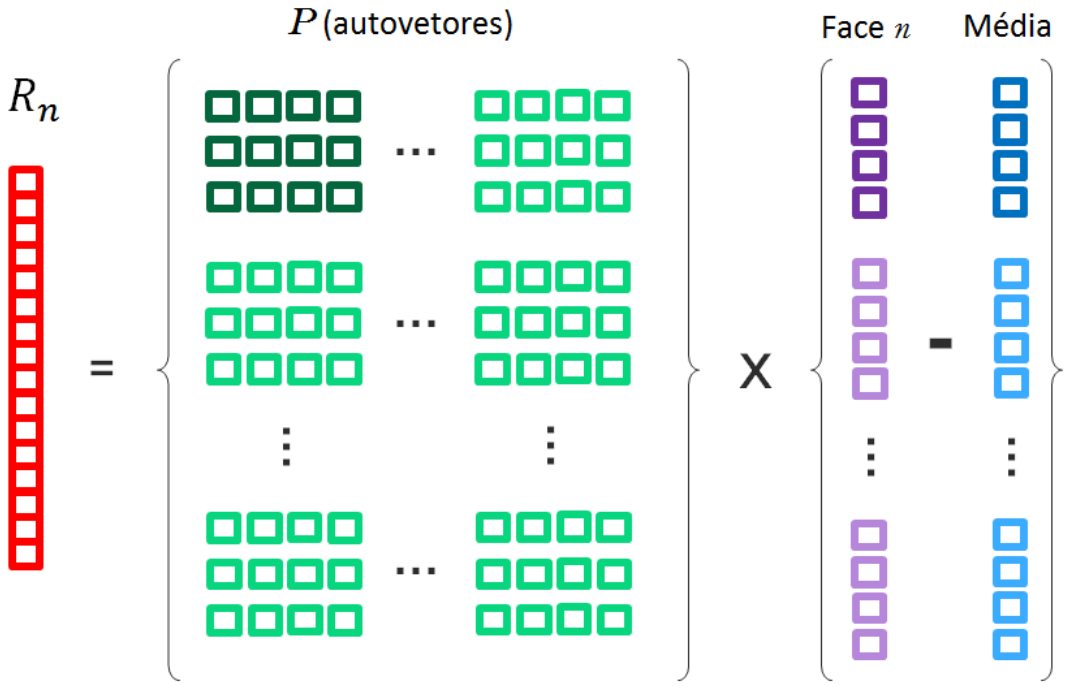


Figura 4.1: Ilustração do cálculo da projeção das faces em FPGA.

Os códigos dos módulos hardware e software e as faces utilizadas nos testes podem ser encontrados no repositório do GitHub no seguinte endereço: https://github.com/henrique789/Face_Recog_Co-design.

4.1.1 Módulo Software

O módulo software, implementado em C++ (gcc 4.4.7) e executado no processador Atom N2600, segue os passos descritos na Seção 2.2 do Capítulo 2. O trecho de código de maior interesse é a última parte do subprograma `train` da classe `Autofaces`. Os primeiros passos desse módulo são idênticos aos do exemplo citado do Capítulo 3, carregar a biblioteca PCIe na memória e abrir o *file handler* para a biblioteca. Em seguida é feita a chamada do subprograma `read_csv` que carregará todas as imagens de treinamento e atribuirá *labels* às imagens.

Depois é instanciado um objeto da classe `Autofaces` que equivale à classe `FaceRecognizer` do OpenCV. Então é chamado o subprograma `train` que é responsável por todo o processo de treinamento descrito no Capítulo 2. Terminada a execução do subprograma `train` o subprograma `predict`, que pertencente a classe `Autofaces`, é chamado

para realizar o reconhecimento. Esse subprograma retorna a *label* da face associada à imagem de entrada. Para ilustrar a sequência de passos do código foi feito um diagrama baseado no diagrama de sequência do UML, que pode ser visto na Figura 4.2.

Um dos primeiros passos do subprograma `train` é a chamada do subprograma `asRowMatrix` que transforma as imagens em vetores linha e depois os agrupa em uma matriz, além disso, nesse subprograma é possível determinar o tipo de dado que será a matriz resultante, desse modo as imagens são transformadas de caractere não-sinalizado (8bits) para ponto flutuante (32bits). Em seguida é determinado o número de autovetores que serão utilizados, que neste caso são 160, valor que se mostrou suficiente em testes feitos com a versão puramente software. O penúltimo passo importante desse subprograma é a instanciação do objeto da classe `ACP` análogo a classe `PCA` do OpenCV, dentro do subprograma de instanciação é calculada a face média, os autovalores e autovetores.

O último passo do subprograma `train` é a chamada do subprograma `FPGAsubspaceProject` que encapsula todos os passos de comunicação entre software e hardware. Esse subprograma primeiramente escreve na SDRAM a face média, as faces e os autovetores, nessa ordem. Em seguida escreve-se o valor de `START_BYTE` no registrador do módulo hardware dentro do FPGA dando início as computações no FPGA. Então, em um laço de repetição, é verificado se o valor `STOP_BYTE` foi escrito no registrador pelo FPGA, indicando que a computação do lado do hardware terminou, o laço termina e é feita a leitura das projeções e o subprograma `FPGAsubspaceProject` termina sua execução.

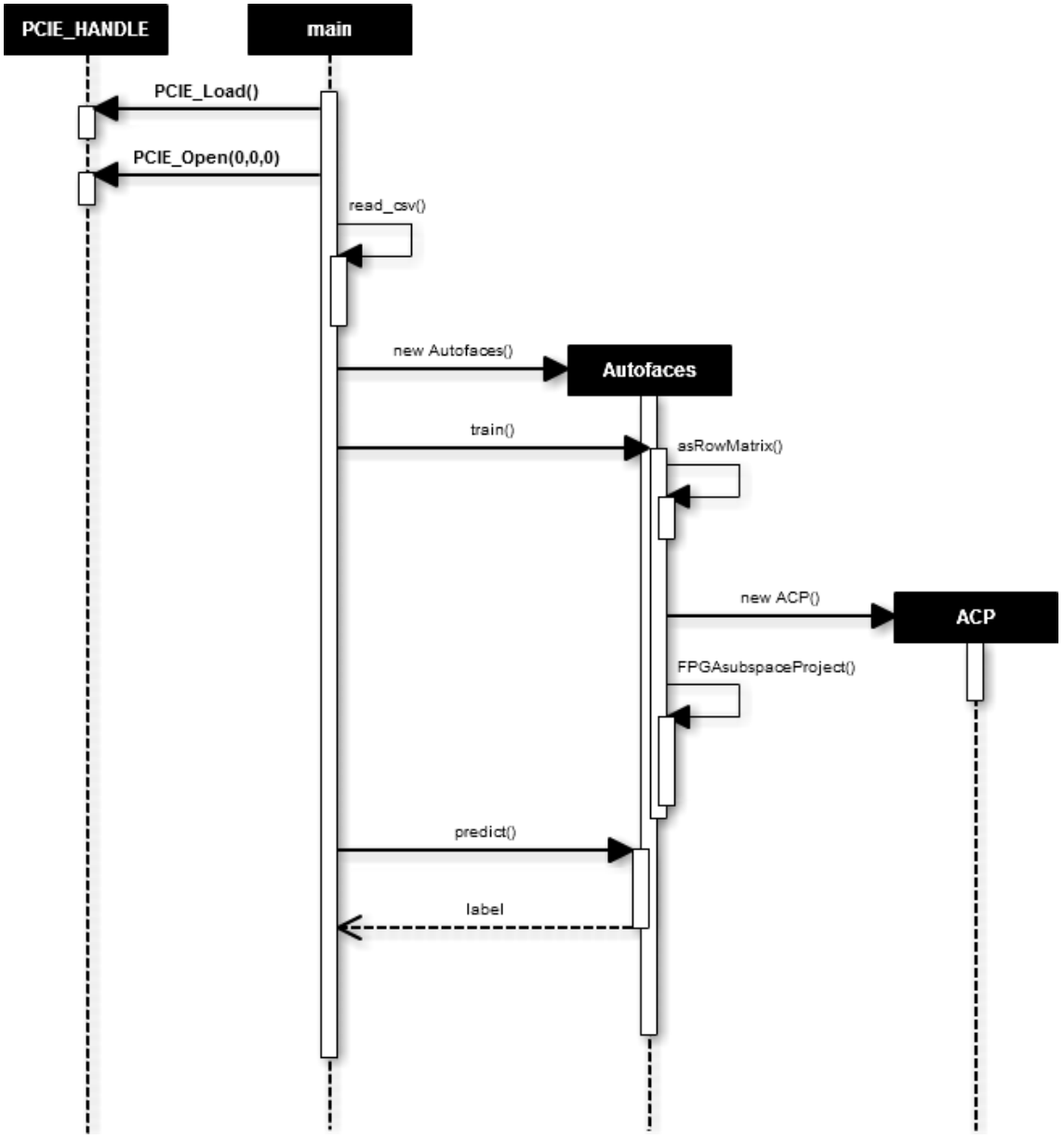


Figura 4.2: Diagrama de sequência do módulo software.

A fim de simplificar o código, os passos para escrita e leitura tanto na SDRAM quanto no FPGA foram divididos em outras funções (Quadro 4.1). As funções `WriteMeanDMA`, `WriteFacesDMA` e `WriteEigenfacesDMA` são responsáveis pela escrita da face média, as faces e os autovetores respectivamente, dentro de cada uma dessas funções, os dados são organizados em um grande arranjo, por exemplo, as 400 faces que estão organizadas em uma matriz de 400 por 10.304 são reorganizadas em um arranjo de 4.121.600 elementos. Para escrever na SDRAM essas funções utilizam

```
25 void Autofaces::FPGAsubspaceProject(PCIE_HANDLE hPCIe, Mat src, Mat mean,
       Mat egvec){
26   if(!WriteMeanDMA(hPCIe, mean))return;
27   if(!WriteFacesDMA(hPCIe, src))return;
28   if(!WriteEigenfacesDMA(hPCIe, egvec))return;
29
30   if(!WriteStartByte(hPCIe))return;
31
32   while(!checkImageDone(hPCIe));
33
34   if(!ReadProjsDMA(hPCIe, src)) return;
35 }
```

Quadro 4.1: Subprograma FPGAsubspaceProject.

o subprograma `PCIE_DmaWrite(PCIE_HANDLE hFPGA, PCIE_LOCAL_ADDRESS LocalAddress, void *pData, DWORD dwDataSize)` da biblioteca PCIe onde `hFPGA` é o *handler* da biblioteca, `LocalAddress` é o endereço na SDRAM, `pData` é o dado a ser escrito e `dwDataSize` é o tamanho do dado.

O subprograma `ReadProjsDMA` encapsula os passos da leitura das projeções e, diferentemente da escrita das faces, é realizada individualmente onde cada arranjo de bytes lido é transformado em uma matriz coluna que então é adicionada ao vetor de projeções. O subprograma utilizado para fazer a leitura via DMA é `PCIE_DmaRead` que funciona de modo análogo a escrita. A escrita do valor `START_BYTE` é feita pelo subprograma `WriteStartByte` que se utiliza do subprograma `PCIE_Write32` (descrita no Capítulo 3) para escrever no registrador do FPGA; já a leitura do valor `STOP_BYTE` é feito pelo subprograma `checkImageDone`, que utiliza o subprograma `PCIE_Read8` para ler do registrador do FPGA.

4.1.2 Módulo Hardware

O módulo customizado `user_module`, implementado em SystemVerilog e executado no FPGA Cyclone IV GX, inclui o módulo `top_levelu` que inclui outros cinco módulos (Figura 4.3): duas instâncias de `FPreg` (uma para armazenar parte da imagem a ser projetada no subespaço e outra para armazenar uma parte da face média), `EigenReg`, `ProjReg` e `outputlogic`. O módulo `user_module` pode controlar esses quatro módulos através da interface de `top_levelu`, utilizando sinais de controle. Esses cinco módulos mais internos possuem sinais `clk`, `rst_n`, `enable` e `clear`. Os sinais `rst_n` e `clear` possuem a mesma

função, porém o `rst_n` é um sinal global que reinicia todos os módulos de uma vez enquanto que o `clear` reinicia apenas um módulo. Além destes há os sinais únicos de entrada e saída de dados para cada módulo.

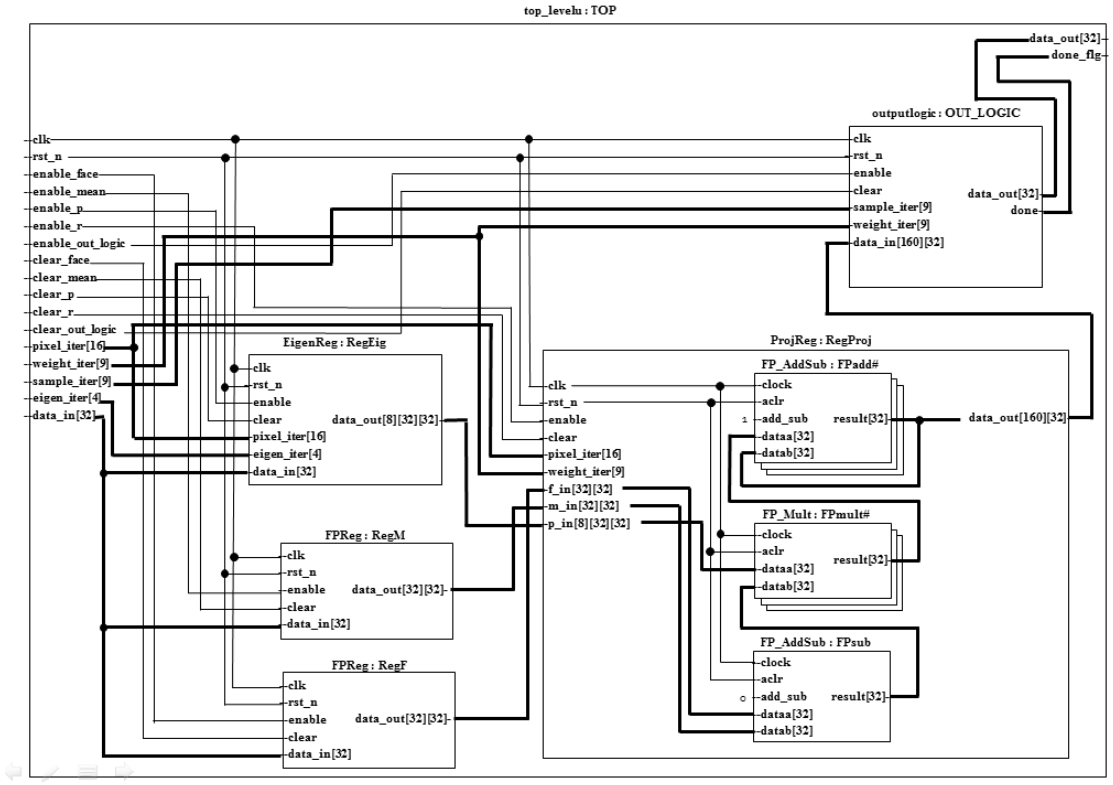


Figura 4.3: Diagrama de de blocos do módulo `top_levelu`.

O módulo `user_module` possui um processo escravo de tipo sequencial que funciona na subida do relógio, de modo praticamente idêntico ao descrito no Capítulo 3, com a adição de uma condição que verifica se o sinal `proj_done` está ativado; se estiver, atribui o valor `STOP_BYTE` para o sinal `csr_registers` indicando que a máquina terminou de projetar todas as faces. Esse processo possui sinais na interface do módulo para que possa atender as requisições de leitura e escrita da CPU.

O módulo `user_module` ainda possui dois processos mestre que são dependentes um do outro. Um deles é um processo sequencial que funciona na subida do relógio, esse processo realiza a troca de estados e atualiza o valor de sinais de endereço, índice de registradores, o intervalo de pixels e dados. O outro processo é de lógica combinacional e determinar o próximo estado (definindo assim o comportamento da máquina) e endereço, além de ativar os sinais

de carga dos módulos internos. O comportamento da máquina de estados será descrito mais adiante, depois da descrição dos módulos mais internos. Como o escravo, o mestre possui sinais na interface do módulo, porém o mestre realiza requisições de leitura e escrita na SDRAM.

O módulo `FPreg` é um registrador bidimensional de 32x32bits (um arranjo de palavras de 32bits) o qual possui uma interface de saída de mesma dimensão, porém sua interface de entrada possui apenas 32bits, pois os IPs que fazem a comunicação entre FPGA e SDRAM são de 32bits, fazendo com que a leitura e escrita seja em palavras de 32bits. O Preenchimento desse arranjo acontece da seguinte forma: quando o sinal `enable` sobe o dado que está na interface de entrada é atribuído para a última posição de um registrador interno a `regFP` que possui a mesma dimensão da interface de saída e está conectado a ela. Na próxima subida do sinal `enable` o dado que estava na última posição é atribuída para a penúltima, comportando-se como um canal FIFO.

O módulo `EigenReg` é um registrador tridimensional de 8x32x32 (uma matriz de palavras de 32bits) o qual possui uma interface de saída de mesma dimensão, esse módulo é utilizado para armazenar uma parte da matriz de autovetores (8 dos 160). A interface de entrada de dados é de 32bits, porém a atribuição dos dados é diferente de `FPreg`, aqui é utilizado dois iteradores (um para a linha, outro para a coluna) assim o primeiro elemento do primeiro autovetor é atribuído a posição de linha `0` e coluna `0`, o segundo a linha `0` e coluna `1` e assim por diante. Os iteradores são arranjos de bits que são acessíveis ao `user_module` através da interface do módulo.

O módulo `ProjReg` calcula e armazena o resultado da multiplicação entre a matriz de autovetores e o resultado da subtração entre face e face média, ou seja a projeção. A ideia inicial (versão inteira) era fazer a subtração da face média da face (com apenas um pixel dos 32), a multiplicação do resultado da subtração com um elemento de cada um dos 8 autovetores e a soma do resultado da multiplicação atual com o resultado das anteriores (Quadro 4.2) em apenas um ciclo. Porém, os módulos de adição e subtração de ponto flutuante, fornecidos pela Altera, necessitam de 7 ciclos para completar os cálculos e o módulo de multiplicação leva 5 ciclos. Além disso, esses módulos da Altera não possuem um sinal para indicar o fim das operações, por isso implementou-se uma máquina de estados dentro de `ProjReg`.

Para realizar a subtração foi instanciado um módulo de adição/subtração de ponto flutuante,

```
1   projecao[iterador_peso] <= projecao[iterador_peso] + (autovetor[0][
      iterador_col]*(face[iterador_col]-faceMedia[iterador_col]));
2   projecao[iterador_peso+1] <= projecao[iterador_peso+1] + (autovetor[1][
      iterador_col]*(face[iterador_col]-faceMedia[iterador_col]));
3   projecao[iterador_peso+2] <= projecao[iterador_peso+2] + (autovetor[2][
      iterador_col]*(face[iterador_col]-faceMedia[iterador_col]));
4   projecao[iterador_peso+3] <= projecao[iterador_peso+3] + (autovetor[3][
      iterador_col]*(face[iterador_col]-faceMedia[iterador_col]));
5   projecao[iterador_peso+4] <= projecao[iterador_peso+4] + (autovetor[4][
      iterador_col]*(face[iterador_col]-faceMedia[iterador_col]));
6   projecao[iterador_peso+5] <= projecao[iterador_peso+5] + (autovetor[5][
      iterador_col]*(face[iterador_col]-faceMedia[iterador_col]));
7   projecao[iterador_peso+6] <= projecao[iterador_peso+6] + (autovetor[6][
      iterador_col]*(face[iterador_col]-faceMedia[iterador_col]));
8   projecao[iterador_peso+7] <= projecao[iterador_peso+7] + (autovetor[7][
      iterador_col]*(face[iterador_col]-faceMedia[iterador_col]));
9 }
```

Quadro 4.2: Projeção parcial.

este módulo possui um sinal de entrada que indica qual das duas operações deve ser realizada e como essa instância realizará apenas subtrações o sinal ficará ativado em zero. Para a multiplicação foram feitas oito instâncias do módulo de multiplicação de ponto flutuante um para cada autovetor que será multiplicado pelo resultado da subtração calculado na etapa anterior. Para acumular os resultados parciais foram utilizadas oito instâncias do módulo de adição/subtração com os seus sinais de seleção de operação ativados em alto.

A máquina de estados de `ProjReg` é formada por 21 estados sendo o primeiro estado chamado de `IDLE` o qual fica a espera da subida do sinal `enable` para então avançar para o próximo estado. A partir de então os ciclos são contados para que se possa saber quando os dados na saída dos módulos somadores estarão corretos. Para fazer essa contagem de ciclos são necessários 19, estados 7 para adição e subtração e 5 para multiplicação, sendo a única condição para troca de estado a subida do relógio. Depois de contar os ciclos, há um último estado que atribui o resultado final para a saída do módulo `ProjReg`.

O módulo `outputlogic` controla a saída do resultado da projeção e indica o término de todas as projeções. Além dos sinais básicos esse módulo possui um contador para o número de faces a serem projetadas, um iterador, uma entrada de dados de 160x32bits que está ligado a saída de `ProjReg`, uma sinal de saída de 32bits e um sinal de saída que indica quando todas as faces foram projetadas. Como a interface de saída do módulo mestre é um sinal de 32bits, para escrever na SDRAM é preciso transmitir as projeções em palavras 32bits.

A Máquina de Estados de `user_module`

A máquina de estados implementada possui 38 estados, por isso o diagrama da máquina será dividido em hierarquias. A Figura 4.4 mostra a máquina completa. O sinal `pixel_iter` é um contador responsável por controlar a quantidade de pixels (tanto da média quanto da face) que foram lidos, em relação aos autovetores esse sinal também serve de iterador para os valores de cada autovetor. O sinal `eigen_iter` serve como iterador entre os autovetores. O sinal `weight_iter` é um contador que controla a quantidade de autovetores que já foram lidos. O sinal `sample_iter` é um contador que controla a quantidade de faces que já foram projetadas.

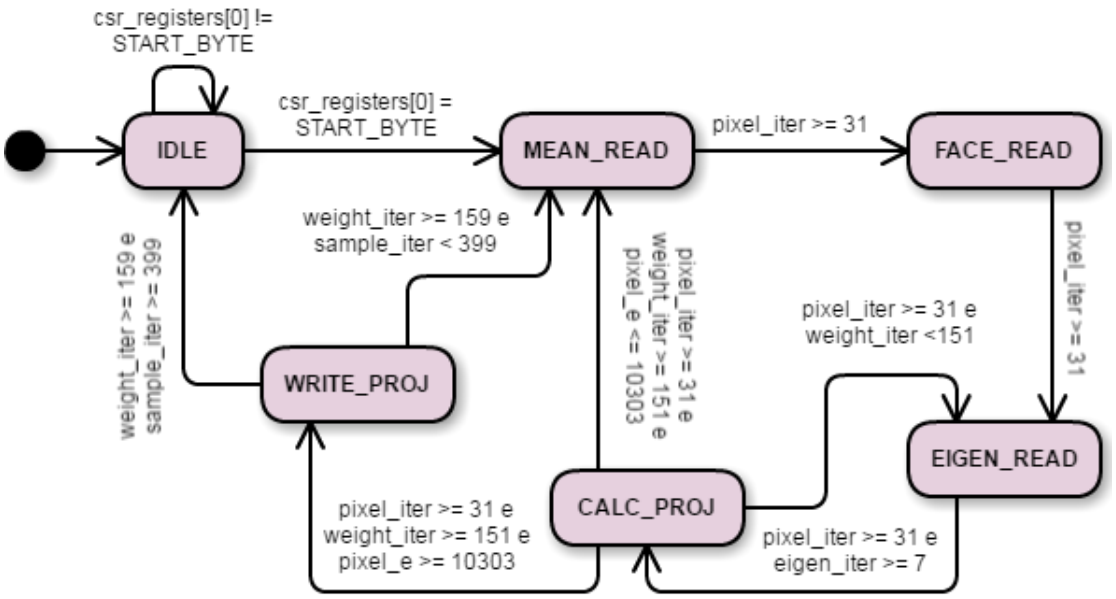


Figura 4.4: Diagrama da máquina de estados.

Uma leitura feita na SDRAM é dividida em 4 estados: requisição de leitura, leitura de dados, escrita do dados no registrador interno do módulo customizado e incremento de endereço. Tendo isso em vista, só serão detalhados os estados internos de `MEAN_READ`.

Depois do estado `IDLE` identificar o valor `START_BYTE` em `csr_registers` a máquina irá para o estado `M_READ_REQ`, dando início a leitura da face média. A Figura 4.5 mostra os estados internos a `MEAN_READ`. O sinal `master_readdatavalid` indica se o dado lido da SDRAM é valido, já o sinal `master_waitrequest` indica quando a interconexão do Avalon pode atender a requisição de leitura ou escrita.

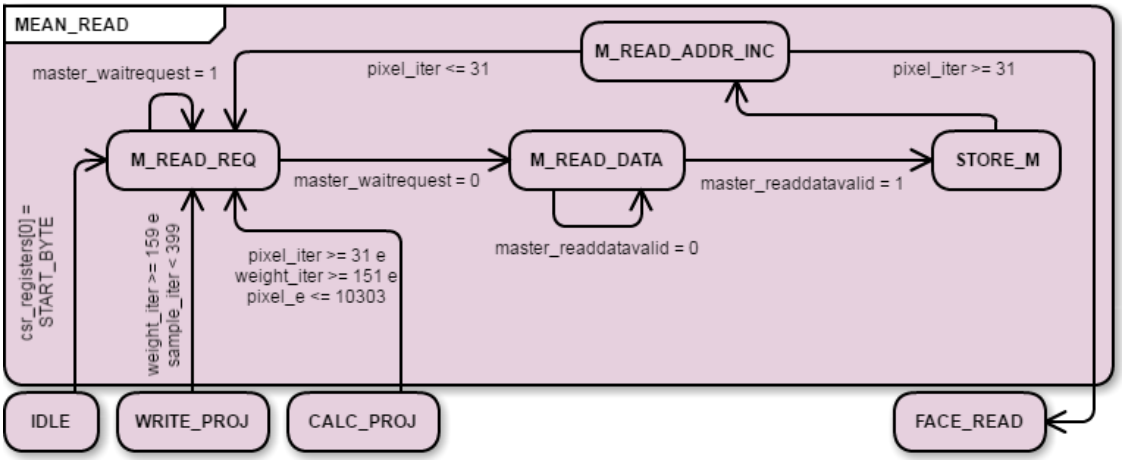


Figura 4.5: Diagrama de estados internos a `MEAN_READ`.

Depois da face média, da face e dos autovetores serem lidos dá-se inicio aos cálculos da projeção. O primeiro estado de `CALC_PROJ` é `CALC_R`, nesse estado o sinal `enable` do módulo `ProjReg` é ativado e a máquina vai para o estado `WAIT_CALC` que possui outros 20 estados internamente que servem apenas para contar o ciclos a fim de acompanhar a máquina de estados de `ProjReg`. Terminado `WAIT_CALC` a máquina irá para `CALC_R_NEXT_PIXEL`. A Figura 4.6 mostra essas transições..

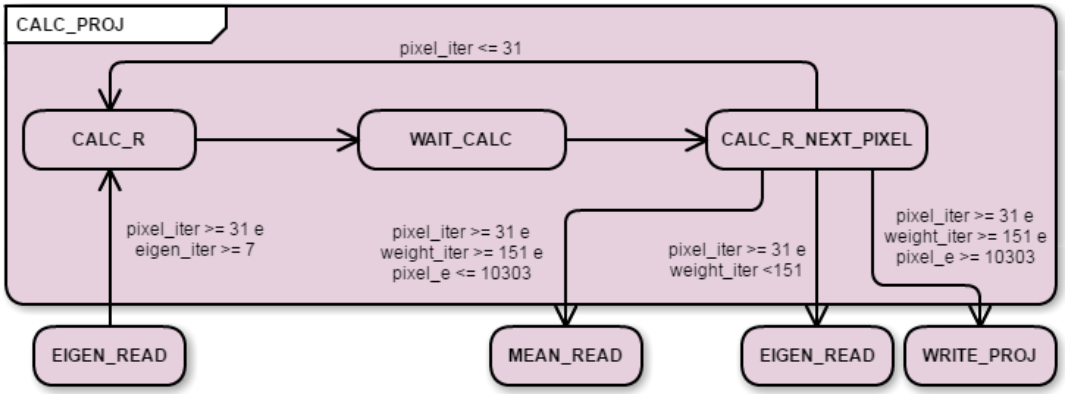


Figura 4.6: Diagrama de estados internos a `CALC_PROJ`.

O estado de alta hierarquia `WRITE_PROJ` escreve a projeção na SDRAM de elemento em elemento. Esse estado é formado por outros três estados que podem ser vistos na Figura 4.7.

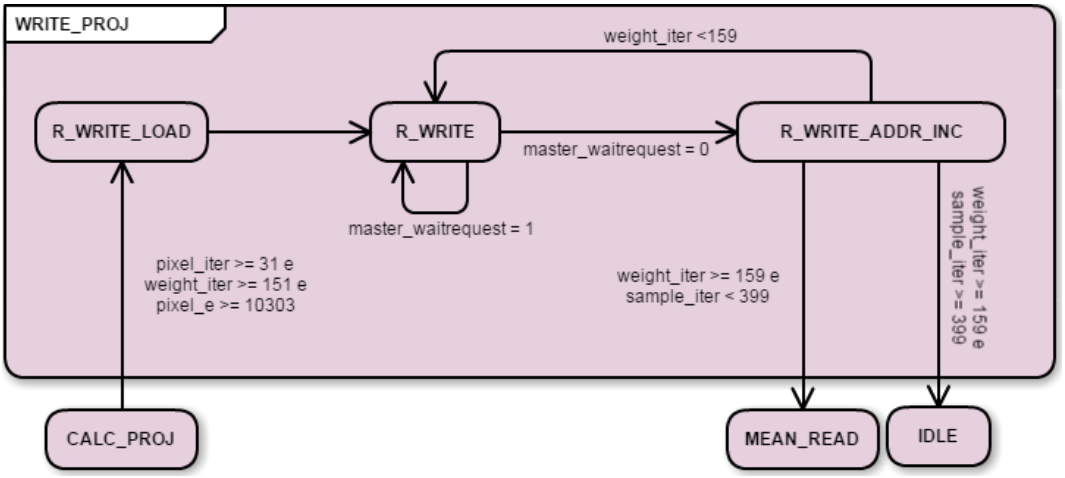


Figura 4.7: Diagrama de estados internos a `WRITE_PROJ`.

4.2 Resultados

A Tabela 4.2 mostra os recursos utilizados pelo módulo hardware. Como pode-se observar o uso de interconexões esta próximo do limite se comparado ao número de LEs utilizados, isso aconteceu devido ao projeto dos módulos registradores que guardam os dados da face, face média, autovetores e projeção. Esses módulos possuem interfaces de saída muito grande, gerando uso elevado de interconexões.

Tabela 4.2: Recursos utilizados pelo FPGA.

Recurso	**Uso**
LEs	69.266 / 149.760 (46%)
Bits de memória	270,099 / 6,635,520 (4%)
Multiplicadores 9-bit	56 / 720 (8%)
Registradores	33,057 / 152,165 (22%)
Blocos M9K	60 / 720 (8%)
Máximo uso de interconexões	79,5%

O uso elevado de interconexões tornou a etapa de roteamento mais complexa o que acabou refletindo no tempo de compilação do módulo hardware. Conforme a Tabela 4.3 essa etapa foi a mais demorada no processo de geração do arquivo de configuração do FPGA.

Tabela 4.3: Tempo de compilação do design.

Etapa	Tempo (mm:ss)
Síntese	11:28
Organização e roteamento	19:35
Geração de Netlist	01:04

Os testes realizados mostraram que a projeção no subespaço foi de 241,762 segundos, 867,23% mais lenta que os 27,877 segundos da projeção realizada pela versão puramente software apresentada na Tabela 4.1 indicando, assim, que não foi obtido o resultado desejado. O motivo principal é a diferença entre as frequências de operação da CPU (1,6GHz) e do FPGA (50MHz). A CPU é 32 vezes mais rápida que o FPGA, porém a versão HW/SW foi 8,6 vezes mais lenta. Para o FPGA superar a CPU com essa frequência de operação seria necessário uma paralelização maior, porém seria necessário mais LEs e uma remodelagem do projeto.

A comunicação entre CPU e SDRAM contribuiu para o alto tempo de execução da projeção, porém com uma parcela muito pequena como pode ser visto na Tabela 4.4 que mostra os tempos de leitura e escrita na SDRAM feitas pelo software via DMA. Analisando o tempo e a quantidade de dados transferida pode-se observar uma taxa de transferência bem abaixo dos 250 MB/s do PCIe, isso ocorre pois o PCIe compete com outros componentes como portas USB e SATA, o *buzzer*, o dispositivo de rede e o codec de áudio entre outras portas e dispositivos que estão conectados no mesmo barramento.

Os tempos registrados compreende o tempo desde as chamadas das funções de leitura e escrita, as requisições de acesso ao barramento PCIe, a transferência dos dados até o termino da execução das funções.

Analisando a Tabela 4.4 percebe-se que a taxa da leitura das projeções está bem abaixo da taxa dos outros dados, isso acontece pois é feita uma leitura para cada projeção, ou seja, são feitas várias chamadas da função `PCIE_DmaRead` o que contribui para o tempo total desta etapa.

Também pode-se observar uma diferença entre as taxas de escrita da face média e das faces. Essa diferença ocorreu devido a quantidade de dados transferida em cada escrita, enquanto a face média possui apenas 41.216 bytes, as faces possuem 1.648.640 bytes, ou seja, leva mais tempo para escrever todas as faces; outro detalhe é que em ambas as escritas é feito apenas uma vez a chamada da função `PCIE_DmaWrite`, ou seja, o tempo de chamada de função é o

mesmo em ambos os caso, porém esse tempo contribui, em termos de porcentagem, mais para o tempo de escrita da face média do que para a escrita das faces. Outra observação importante é que a porcentagem mostrada na tabela se refere ao tempo total de projeção.

Tabela 4.4: Tempo de transferência de dados do CPU para SDRAM no FPGA.

Trecho de código	**Tempo (s)**	**Porcentagem (%)**	**Dados (Bytes)**	**Taxa (MB/s)**
Escrita da face média	0,0024	0,001	41216	16,24
Escrita das faces	0,7182	0,3	16486400	21,89
Escrita dos autovetores	0,2932	0,12	6594560	21,44
Leitura das projeções (400)	0,3530	0,15	256000	0,69
Total	1,3668	0,57	-	-

Outro motivo que contribuiu para o tempo de execução elevado da projeção é a quantidade excessiva de acessos que o FPGA faz na SDRAM, porém a Tabela 4.5 mostra que no melhor caso os acessos tomariam apenas 80,12 segundos. Segundo a especificação do Avalon [20], no melhor dos casos o controlador da SDRAM leva 4 ciclos para realizar a leitura e a escrita, porém devido ao design da máquina de estados, o módulo implementado leva 6 ciclos para leitura e escrita. E como é conhecido a frequência do FPGA e o número de acessos foi possível calcular o tempo necessário para os acessos no melhor caso.

Tabela 4.5: Acesso teórico do FPGA à SDRAM no melhor caso.

	Leituras	**Escritas**	**ciclos**	**Tempo (s)**
Face média	4121600	-	24729600	0,49
Faces	4121600	-	24729600	0,49
Autovetores	659456000	-	3956736000	79,13
Projeções	-	64000	384000	0,01
Total	667699200	64000	4006515200	80,12

O número de ciclos que são necessários para calcular as projeções foi deduzido a partir do comportamento da máquina de estados, apresentado um número fixo de ciclos; e como é conhecido o número de imagens, o número de pixels, o número de autovetores e a frequência, foi possível calcular o tempo exato que levou o cálculo das projeções. O tempo decorrido dos cálculos das projeções pode ser calculado da seguinte forma:

$$t = (ni * np * (na/8) * nc)/f \tag{4.1}$$

onde t representa o tempo em segundos, ni o número de imagens, np o número de pixels, na o número de autovetores, nc o número de ciclos e f a frequência.

Aplicando-se os valores na Equação 4.1 obtém-se um tempo de projeção de 34,62 segundos. Comparando o tempo total das projeções, o tempo de acesso entre SDRAM, FPGA e CPU pode-se concluir que, de fato, a comunicação entre FPGA e SDRAM não ocorreu no melhor caso. A especificação do Avalon [20] não informa qual seria o caso médio e o pior caso, é dito apenas que não há limite, mas que se deve garantir que não ocorra acesso indefinidamente.

Na Tabela 4.6 são apresentados o tempo de execução das etapas de projeção. Observe que o tempo de comunicação entre FPGA e CPU apresentado na Tabela 4.6 foi calculado subtraindo o tempo de cálculo de projeção e comunicação entre CPU e SDRAM do tempo total de projeção, visto que não foi possível calcular o tempo exato de comunicação devido ao número variável de ciclos do protocolo de comunicação do Avalon.

Tabela 4.6: Tempo das Etapas da projeção.

Etapa	Tempo (s)	Porcentagem (%)
Comunicação CPU-SDRAM	1,366	0,57
Comunicação FPGA-SDRAM	205,776	85,12
Cálculo da projeção	34,620	14,32
Total	241,762	100

4.3 Trabalhos correlatos

Esta seção descreverá alguns trabalhos correlatos que são comparados na tabela 4.7.

Morizet *et al.* [21] desenvolveram um sistema de reconhecimento facial que utiliza o PCA como algoritmo de reconhecimento e que é executado inteiramente em FPGA, tendo como foco a etapa de reconhecimento. Esse sistema usa o softcore NIOS II para fazer o reconhecimento, porém a projeção da face que se quer identificar e o cálculo das distâncias é feito em um módulo a parte que tem como objetivo acelerar a etapa de reconhecimento. Os teste foram feitos utilizando o banco de faces FERET [22]. O tempo de reconhecimento foi de 60ms em uma frequência de 50MHz.

Visakhasart e Chitsobhuk [23] propuseram um sistema de reconhecimento facial baseado em diferentes técnicas de PCA: MPCA (Modular PCA), WMPCA (Weight MPCA) e Wavelet PCA. A técnica MPCA consiste em dividir as faces em N sub-imagens as quais é aplicado o

PCA. O WMPCA é muito parecido com a MPCA, a diferença é que para cada sub-imagem é atribuído um peso, visto que certas sub-imagens podem ter maior relevância em relação as outras. O Wavelet PCA consistem em aplicar wavelets nas imagens antes de processa-las a fim de tornar o reconhecimento mais robusto em relação a variação de iluminação e posição.

Do ponto de vista do hardware o processamento das sub-imagens pode ser paralelizado porém cada processo ainda seria sequencial, então a proposta Visakhasart e Chitsobhuk é de melhorar essa etapa com múltiplos pipelines (cada processo possuiria um pipeline). Todas essas técnicas implementadas tornam o reconhecimento mais rápido e mais preciso. Nos testes foram feitas várias combinações das técnicas usadas, as que obtiveram o menor tempo de reconhecimento foram aquelas onde as imagens foram particionadas simetricamente 4 vezes, sendo o menor tempo de aproximadamente 8,84 ms em uma frequência de 13,38 MHz. Como base de dados foi utilizado o banco de faces Yale [24] e o ORL [1].

Tran *et al.* [25] propuseram um acelerador de hardware para reconhecimento facial utilizando PCA como técnica de extração de características e VQ (quantização de vetores - *vector quantization*) como técnica de comparação. A técnica de quantização de vetores descrita por eles nada mais é que o cálculo da distância euclidiana entre a projeção de entrada e as projeções do banco que são organizadas em *codewords*. O sistema proposto é formado pelo NIOS II, um módulo para realizar o treinamento do PCA, um módulo de memória para armazenar as projeções (ou *codewords*), um módulo controlador do pipeline, *n* instâncias de um módulo responsável por calcular a distância euclidiana e um módulo para encontrar a menor distância. O NIOS é responsável por controlar os sinais do módulo de treinamento do PCA e do controlador do pipeline. O pipeline implementado possui dois estágios, um para o cálculo das distâncias e outro para a comparação das distâncias a fim de encontrar a menor.

O processo de reconhecimento acontece da seguinte forma: as projeções são passadas para o módulo de memória on-chip. Então os parâmetros de configuração são passados pelo NIOS que indica quantas projeções serão usadas, em seguida é passado o vetor imagem que contém a face a ser reconhecida para um FIFO. Então é calculada as distâncias e depois é determinado a menor distância. Os testes realizados foram feitos usando 100 *codewords* calculados com 5 autovetores e o tempo necessário para realizar o reconhecimento foi de 1,6 ms a 100 MHz.

Sardar e Babu [26] desenvolveram um sistema de tempo real que utiliza uma rede neural

baseada em RCE (*Restricted Coulomb Energy*), o PCA e DWT (*Discret Wavelet Transform*). O sistema consiste em uma câmera CMOS de 1 Mega-Pixel de resolução e uma taxa 37fps. O módulo da câmera possui um módulo de pre-processamento em tempo real implementado dentro do FPGA. A imagem da câmera é redimensionada de 1024x1024 para 128x128 e então é passada para o FPGA através da interface CamLink.

Como método de pré-processamento o sistema utiliza CLAHE (*contrast limited adaptive histogram equalization*) pois este método aumenta a precisão do sistema. Depois disso é aplicado a transformada wavelet discreta para que se possa realizar uma análise de várias resoluções de imagens das faces para então selecionar as características mais apropriadas. Então o PCA é aplicado para redução de dimensionalidade e extração de características. Os vetores de características provenientes do PCA são usados no treinamento da rede neural. O reconhecimento é feito no FPGA Virtex-6 e o resultado é passado para o processador MicroBlaze que mostra o resultado para o usuário através de uma GUI (*Graphic User Interface*). Nos testes foram utilizados diversos bancos de faces como ORL [1], Yale [24], FERET [22] entre outros. O sistema conseguiu realizar o reconhecimento em 18 ms a uma freqüência de 62,5 MHz.

Tabela 4.7: Comparação entre os trabalhos correlatos.

Autores	**Métodos**	**Tempo (ms)**	**frequência (MHz)**
Morizet *et al.*	PCA	60	50
Visakhasart e Chitsobhuk	PCA, MPCA, WMPCA e DWT	8,84	13,38
Tran *et al.*	PCA e VQ	1,6	100
Sardar e Babu	PCA, DWT, CLAHE e RCE-NN	18	62,5

Como é possível notar, a maioria dos trabalhos focam na etapa de reconhecimento, onde são implementados módulos completos em hardware que executam todas as etapas do reconhecimento (não confundir com o treinamento). O trabalho desenvolvido possui uma abordagem diferente, pois foca na etapa de treinamento não sendo possível fazer uma comparação direta com os trabalhos apresentados na tabela 4.7. Um outro ponto importante está relacionado com a forma como a solução foi desenvolvida, buscando uma arquitetura onde o FPGA seria responsável apenas por uma parte do processo de treinamento, necessitando assim da comunicação entre CPU e FPGA.

Capítulo 5

Conclusão

Este trabalho descreve o co-projeto de hardware e software de um sistema de reconhecimento facial desenvolvido utilizando plataforma DEi-150 composto pelo processador Intel Atom N2600 e o FPGA da Altera Cyclone IV GX que se comunicam através de um interface PCIe. O software do sistema foi desenvolvido em C++ e é executado no sistema operacional CentOS 6, o hardware foi desenvolvido em SystemVerilog e utiliza módulos de aritmética de ponto flutuante fornecidos pela Altera.

Os testes foram realizados através da execução do sistema implementado e como métrica de desempenho foi utilizado o tempo de execução mensurado em segundos. Através do software foi possível mensurar o tempo de acesso do software a SDRAM. O tempo de execução de certos passos da máquina de estados tiveram de ser calculados, visto que o módulo em hardware foi implementado de tal forma que ele apenas indica o término de todas as projeções através da escrita do valor `STOP_BYTE` no registrador do processo escravo do módulo hardware.

Os protocolos do Avalon para acesso a SDRAM não possui um número fixo de ciclos, fazendo com que não se possa calcular com exatidão o tempo total de acesso da SDRAM pelo FPGA, porém o tempo de execução dos cálculos das projeções podem ser calculados, visto que os módulos de aritmética de ponto flutuante possuem um número fixo de ciclos que são necessários para realizar as operações.

A análise dos resultados obtidos mostra que a projeção em hardware foi 867,23% mais lenta que a projeção em software. O tempo de acesso da SDRAM para envio de valores para o FPGA e leitura dos resultados foi de 1,37 segundos (0,45% em relação ao tempo de treinamento e 0,57% em relação ao tempo de projeção) indicando que esse não é um gargalo. O tempo de execução dos cálculos da projeção, sem considerar o tempo de comunicação entre SDRAM,

CPU e FPGA, foi calculado em 34,62 segundos considerando a frequência de operação de 50 MHz do FPGA (11,39% em relação ao tempo de treinamento e 14,32% em relação ao tempo de projeção), com esse dado pode-se concluir que o tempo de acesso a SDRAM pelo FPGA foi aproximadamente de 205 segundos, indicando um dos gargalos do sistema.

O que ocasionou esse gargalo foi a grande quantidade de acessos a SDRAM pelo FPGA. O excesso de acessos a SDRAM foi inevitável, pois os dados eram muito grandes e tiveram de ser armazenados na SDRAM e apenas uma parte desses dados eram carregados para o FGPA. Outro fator que contribuiu com esse resultado foi a diferença de frequência entre a CPU (1.6 GHz) e o FPGA (50 MHz).

Trabalhos futuros

Como trabalhos futuros várias melhorias no módulo de hardware e novas técnicas poderiam ser implementadas, por exemplo: mudar a interface de saída dos sub-módulo internos ao `top_levelu`, pois atualmente a interface de saída compreende o arranjo inteiro gerando um grande roteamento de fios e impossibilitando o uso de todos os LEs. Com a disponibilidade de todos os LEs seria possível aumentar o número de operações paralelas e assim reduzir o tempo de projeção.

A implementação de um pipeline que possibilite que os cálculos das projeções sejam executados em paralelo a leitura e escrita dos dados na SDRAM é um trabalho futuro. Reduzir o tamanho das imagens e o número de autovetores utilizados nas projeções é outra possibilidade a ser analisada em projetos futuros.

Referências Bibliográficas

[1] HOPPER, A. *The Database of Faces*. 2002. Consultado na INTERNET: http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html, 2016.

[2] Femto Logic Design (P) Ltd. *Femto Logic Design*. 2016. Consultado na INTERNET: http://femtologicdesign.com/Products/de2i-150-fpga/, 2016.

[3] JOHNSON, M. *DE2i-150 at Purdue*. 2015. Consultado na INTERNET: https://sites.google.com/site/de2i150atpurdue, 2016.

[4] HUANG, A. bunnie. *The Death of Moore's Law Will Spur Innovation*. Mar 2015. Consultado na INTERNET: http://spectrum.ieee.org/semiconductors/design/the-death-of-moores-law-will-spur-innovation, 2016.

[5] COURTLAND, R. *The Rise of the Monolithic 3-D Chip*. Feb 2014. Consultado na INTERNET: http://spectrum.ieee.org/semiconductors/design/the-rise-of-the-monolithic-3d-chip, 2016.

[6] PATTERSON, D. *The Trouble With Multicore*. Jun 2010. Consultado na INTERNET: http://spectrum.ieee.org/computing/software/the-trouble-with-multicore, 2016.

[7] ENZLER, R.; PLATZNER, M. *Dynamically Reconfigurable Processors*. 2000. Consultado na INTERNET: https://www2.ife.ee.ethz.ch/cag/pub/ep01.pdf, 2016.

[8] ANEAA. *GRU Airport instala tecnologia inédita no País para inspeção de passaporte brasileiro*. 2014. Consultado na INTERNET: http://aneaa.aero/gru-airport-instala-tecnologia-inedita-no-pais-para-inspecao-de-passaporte-brasileiro/, 2016.

[9] CHEN, D.; Cao, X.; Wen, F.; Sun, J. Blessing of dimensionality: High-dimensional feature and its efficient compression for face verification. In: *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*. 2013. p. 3025–3032. ISSN 1063-6919.

[10] LI, S. Z.; JAIN, A. K. *Handbook of Face Recognition*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005. ISBN 038740595X.

[11] JOLLIFFE, I. *Principal Component Analysis*. 2. ed. Springer, 2002. (Springer Series in Statistics). ISBN 9780387954424.

[12] LIU, D.; Lung, C. H.; Seddigh, N.; Nandy, B. Entropy-based robust pca for communication network anomaly detection. In: *2014 IEEE/CIC International Conference on Communications in China (ICCC)*. 2014. p. 171–175. ISSN 2377-8644.

[13] QIU, C.; Ren, H.; Zou, H.; Zhou, S. Performance comparison of target classification in sar images based on pca and 2d-pca features. In: *Synthetic Aperture Radar, 2009. APSAR 2009. 2nd Asian-Pacific Conference on*. 2009. p. 868–871.

[14] JAIN, A.; Bakshi, M.; Kalele, A.; Subramanian, E. On accelerating concurrent pca computations for financial risk applications. In: *2015 IEEE 22nd International Conference on High Performance Computing (HiPC)*. 2015. p. 175–184.

[15] TURK, M.; PENTLAND, A. Eigenfaces for recognition. *Jornal of Cognitive Neuroscience*, v. 3, n. 1, p. 71–86, Jan 1991. ISSN 0898-929X.

[16] TERASIC. *DE2i-150 FPGA Development Kit*. 2013. Consultado na INTERNET: http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=11&No=529, 2016.

[17] ALTERA. *Cyclone IV Device Handbook*. 101 Innovation Drive San Jose, CA 95134, 2016. v. 1.

[18] ITSEEZ. *OpenCV*. 2017. Consultado na INTERNET: http://opencv.org/, 2017.

[19] LTD, J. S. S. *Just::Thread - Complete C++ Standard Thread Library*. 2016. Consultado na INTERNET: http://www.stdthread.co.uk/doc/headers/chrono/monotonic_clock.html, 2016.

[20] ALTERA. *Avalon Interface Specifications*. 2015. Consultado na INTERNET: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/manual/mnl_avalon_spec.pdf, 2016.

[21] MORIZET, N.; Amiel, F.; Hamed, I. D.; Ea, T. A comparative implementation of pca face recognition algorithm. In: *2007 14th IEEE International Conference on Electronics, Circuits and Systems*. 2007. p. 865–868.

[22] NIST. *The Color FERET database*. 2011. Consultado na INTERNET: https://www.nist.gov/itl/iad/image-group/color-feret-database, 2016.

[23] VISAKHASART, S.; CHITSOBHUK, O. Multi-pipeline architecture for face recognition on fpga. In: *2009 International Conference on Digital Image Processing*. 2009. p. 152–156.

[24] NIST. *Yale face databases*. 2011. Consultado na INTERNET: http://vision.ucsd.edu/content/yale-face-database, 2016.

[25] TRAN, D.; To, T.; Huynh, T.; Nguyen, P. Designing a harware accelerator for face recognition using vector quantization and principal component analysis as a component of sopc. In: *2010 Fifth IEEE International Symposium on Electronic Design, Test Applications*. 2010. p. 82–86.

[26] SARDAR, S.; BABU, K. A. Hardware implementation of real-time, high performance, rce-nn based face recognition system. In: *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*. 2014. p. 174–179. ISSN 1063-9667.