UNIVERSIDAD DE CONCEPCIÓN

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

Profesor Patrocinante:
Dr. Miguel Figueroa

Informe de Memoria de Título
para optar al título de:
Ingeniero Civil Electrónico

Implementación en Hardware de algoritmo de reconocimiento de rostros BDPCA+LDA

Concepción, Octubre de 2011

Pablo Andrés Pizarro Jara

Universidad de Concepción
Facultad de Ingeniería
Departamento de Ingeniería Eléctrica

Profesor Patrocinante:
Dr. Miguel Figueroa

Implementación en Hardware de algoritmo de reconocimiento de rostros BDPCA+LDA

Pablo Andrés Pizarro Jara

Informe de Memoria de Título
para optar al Título de

Ingeniero Civil Electrónico

26 de octubre de 2011

Resumen

Debido al auge en la computación en las últimas décadas, ha tomado cada vez más relevancia la posibilidad de lograr un reconocimiento automático de personas. Los métodos de autentificación mediante huellas digitales o oculares son sólo dos de los métodos utilizados. Es en este contexto que el reconocimiento de rostros se presenta como un alternativa importante, en especial debido a que no requiere una acción particular por parte del sujeto, basta con tomar una imagen de su rostro para lograr la clasificación. Muchos de estos métodos se pueden aplicar también en otros ámbitos del reconocimiento automatizado; el reconocimiento de letras manuscritas o piezas de una máquina por ejemplo.

Dada la gran utilidad del reconocimiento automatizado, ha aumentado el interés y requerimientos de métodos cada vez más rápidos, eficaces e, idealmente, con menor consumo de energía. Frente a este escenario, las implementaciones en *hardware* dedicado presentan atractivas ventajas.

En el presente trabajo, se revisan dos métodos de reconocimiento de rostros con el fin de implementar uno de ellos en una FPGA y comparar el rendimiento con la misma implementación en PC.

Los resultados muestran una clara ventaja de la FPGA por sobre el PC, despejando el camino a complementar y mejorar la implementación utilizando, por ejemplo, sistemas más complejos de clasificación.

A mi hijo Nicolás
el impulsor de mi actuar

Agradecimientos

A mis padres y hermana, Marjorie, Oscar y Valentina, por guiarme y apoyarme incondicionalmente desde el inicio.

A mi pareja, Cristina, por la paciencia y confianza durante nuestra historia, así como la constante preocupación y soporte durante estos años.

A mis compañeros y amigos, por su incondicionalidad y compañia durante estos años.

Al Departamento de Ingeniería Eléctrica de la Facultad de Ingeniería, por el soporte institucional dado para la realización de este trabajo.

Al Doctor Miguel Figueroa Toro por su asesoría y dirección en el trabajo de investigación.

Y a todos aquellos que no he nombrado pero han participado y colaborado de distintas formas para llevar este proyecto a buen puerto.

Índice general

Índice de figuras

Índice de tablas

Nomenclatura

Matrices

$\mathbf{A}$: Matriz de proyección a subespacio *2DPCA*

$\mathbf{G_t}$: Matriz de covarianza

$\mathbf{S_B}$: Matriz de varianza entre clases (*Fisherfaces*)

$\mathbf{S_T}$: Matriz de varianza para *Eigenfaces*

$\mathbf{S_{T_y}}$: Matriz de varianza de muestras proyectadas

$\mathbf{S_W}$: Matriz de varianza interior a la clases (*Fisherfaces*)

$\mathbf{W}$: Matriz de proyección a subespacio principal (*Eigenfaces*)

$\mathbf{W_C}$: Matriz de proyección por filas (*BDPCA*)

$\mathbf{W_R}$: Matriz de proyección por columnas (*BDPCA*)

$\mathbf{W_{opt}}$: Matriz de proyección óptima

$\mathbf{W_{PCA}}$: Matriz de proyección *Eigenfaces*

$\mathbf{W_{LDA}}$: Matriz de proyección *Fisherfaces*

$\mathbf{X}$: Set de observaciones (imágenes) (*Eigenfaces*) o imagen a proyectar (*BDPCA*)

$\mathbf{Y}$: Imagen individual proyectada (*BDPCA*)

Vectores

$\mathbf{x_i}$: Imagen individual

$\mathbf{y_i}$: Imagen individual proyectada

μ : Promedio imágenes

$\mu_\mathbf{y}$: Promedio proyecciones

$\mu_\mathbf{i}$: Promedio imágenes de clase i

Escalares

d : Distancia entre puntos

x_n : Componente de vector

Abreviaciones

FPGA : Field Programmable Gate Array

PCA : Principal Component Analysis

LDA : Linear Discriminant Analysis

FLD : Fishers Lineal Discriminant

BDPCA : BiDimensional PCA

2DPCA : Two - Dimensional PCA

ICA : Indepent Component Analysis

USB : Universal Serial Bus

RAM : Random Access Memory

ROM : Read Only Memory

PC : Personal Computer

PSDRAM : Pseudo-Static Dynamic RAM

LED : Ligth Emitting Diode

Capítulo 1

Introducción

Gracias a la vida que me ha dado tanto.
Me dio dos luceros que, cuando los abro,
perfecto distingo lo negro del blanco,
y en el alto cielo su fondo estrellado
y en las multitudes el hombre que yo amo.
...
Violeta Parra.

Desde que surge la vida, ha existido la necesidad de identificación, para reconocer alimentos, amenazas, un amigo o la pareja. Con el desarrollo de la sociedad, la identificación se ha vuelto cada vez más necesaria y más compleja. Para nosotros los humanos y probablemente para otros mamíferos, reconocer a otros es algo cotidiano, lo hacemos sin pensar ni esforzarnos en forma consiente. No obstante, una simple caminata por la calle es un gran esfuerzo para el cerebro, que constantemente está utilizando los sentidos buscando entradas conocidas, reconociendo olores, sonidos, imágenes, sensaciones, etc. Es así como somos capaces de distinguir personas en una multitud, letras en un dibujo o palabras en una canción. A partir de esta habilidad que poseemos los seres vivos, surge la inquietud de implementar sistemas capaces de reconocer patrones en forma automática, ya sean caras, sonidos, o figuras.

El reconocimiento de rostros ha acaparado la atención de desarrolladores de visión automatizada durante las últimas décadas debido a su gran cantidad de usos, abarcando desde aplicaciones en seguridad (identificación y autentificación), hasta aplicaciones en entretenimiento casero (interacciones hombre-máquina). Esta atención se traduce en sistemas de reconocimiento

altamente efectivos llegando incluso a superar a humanos [25].

No obstante este desarrollo viene acompañado de un aumento en la complejidad de los algoritmos y mayores necesidades de hardware, desde más y mejores cámaras hasta computadores de última generación para operar en tiempo real. Debido a los altos requerimientos, estos nuevos métodos se vuelven inaplicables en sistemas portátiles, donde los recursos de energía y potencia de cálculo son limitados. Dicha limitante ha fomentado el desarrollo de sistemas de reconocimiento de rostros implementados directamente en hardware, explotando al máximo la capacidad de paralelismo existente en circuitos integrados para lograr altos rendimientos con menores consumos que utilizando procesadores multipropósito.

Una implementación en hardware se favorece fuertemente por cálculos regulares y comunicación local o estructurada, es por esto que muchos de los algoritmos implementados más eficientes, usan redes neuronales basadas en subespacios [16]. Normalmente el entrenamiento de las redes se hace offline y fuera del chip, para luego transferir los coeficientes. Si bien este método impide el entrenamiento de la red durante la operación, permite utilizar algoritmos de aprendizaje más complejos y con menores errores numéricos, que los posibles con la etapa de aprendizaje realizada en el chip. Para reentrenar el algoritmo, basta con ejecutar el algoritmo de aprendizaje en un computador y traspasar los nuevos coeficientes al chip.

En esta memoria se evalúa un algoritmo de clasificación de rostros basado en subespacios, para luego implementar sobre una FPGA y comparar el rendimiento de la implementación con el mismo algoritmo en PC. Este algoritmo usa PCA Bi-Direccional (BDPCA) y Análisis de Discriminante Lineal (LDA) para la reducción de dimensionalidad y extracción de características, utilizando luego una medición de la distancia Manhattan para la clasificación. Como plataforma se elige una tarjeta de desarrollo Nexys, la cual incluye una FPGA Spartan 3 XC3S1000 donde se implementará el algoritmo de clasificación y una memoria RAM de 16MB donde se almacenará la imagen a clasificar, del resto del hardware presente en la placa sólo se utilizará la interfaz USB para programar la FPGA y el generador de reloj.

Se obtienen resultados muy similares a la implementación en punto flotante sobre *Matlab* (la diferencia es menor al 1 %), pero con consumos estimados de energía un orden de magnitud menor y una velocidad de operación superior a la implementación sobre PC.

1.1. Objetivos

1.1.1. Objetivos generales

El objetivo del presente trabajo es evaluar los algoritmos BDPCA y BDPCA + LDA, para luego implementar uno de ellos sobre una FPGA y evaluar el rendimiento.

1.1.2. Objetivos específicos

- Estudiar y evaluar algoritmos BDPCA y BDPCA + LDA y seleccionar uno de ellos para la implementación en FPGA.
- Desarrollar y evaluar una arquitectura en hardware específica para la plataforma utilizada.

1.2. Alcances y limitaciones

Esta memoria incluye el estudio y la evaluación del algoritmo BDPCA y BDPCA+LDA para el reconocimiento de rostros. Una vez seleccionado uno de los algoritmos y la reducción dimensional correspondiente, se implementará en hardware de la manera más directa posible, buscando hacer un uso eficiente de los recursos de la FPGA.

Una de las limitaciones de este trabajo consiste en que no se incluye una interfaz con un dispositivo de adquisición de imágenes. Se asumirá que se dispone de la imagen en una memoria de la plataforma externa a la FPGA. La lectura de ésta se realizará tomando en consideración que no necesariamente está operando en forma sincrónica, para evitar restringir la implementación a la plataforma de desarrollo específica. Los cálculos de los coeficientes de proyección se realizaran fuera de línea, en una computadora externa usando Matlab, esto pues los cálculos requeridos son de alta complejidad y no se tiene una real ventaja de lograr calcular los coeficientes en linea; se deja abierta la posibilidad de editar los coeficientes por un módulo externo a aquellos donde se usan para permitir futuros trabajos.

La evaluación del algoritmo se hará tomando en consideración la plataforma donde será implementada, vale decir la selección será enfocada en maximizar el rendimiento con los recursos de la plataforma en cuestión, no se entrará en el detalle sobre la efectividad del método bajo condiciones especificas.

1.3. Organización del informe

El temario de este informe se describe a continuación:

- El capítulo 2 consiste en una revisión de las bases teóricas de los algoritmos Eigenfaces, FisherFaces y BDPCA. Además este capítulo incluye una presentación de las distintas bases de datos de rostros usadas.
- El capítulo 3 describe las características de la tarjeta de desarrollo utilizada, sus limitaciones y ventajas.
- El capítulo 4 presenta los resultados de los métodos mencionados en el capítulo 2 y la selección del método a implementar.
- El capítulo 5 presenta y describe el diseño de la arquitectura a implementar.
- El capítulo 6 presenta el resultado del algoritmo sobre la FPGA y su comparación con la implementación en PC.
- El capítulo 7 entrega las principales conclusiones del tema y presenta algunas propuestas de posibles trabajos a futuro.

Capítulo 2

Métodos y bases de datos

Los métodos de reconocimiento facial pueden ser divididos en dos lineas: el reconocimiento biométrico y el reconocimiento basado en escena. El primero utiliza las características del rostro como la posición de los ojos, nariz, orejas, etc. Este método es el que obtiene los mejores resultados, pero con una alta demanda de hardware y energía. En cambio, los métodos basados en escena utilizan la imagen como un todo, clasificando por similitud de la imagen completa. Esto amplía la utilidad de estos métodos a cualquier tipo de patrón, ya sea letra manuscrita o la imagen de una pieza mecánica. En esta memoria se utilizará exclusivamente para clasificar caras, pero se podría esperar un comportamiento similar para cualquier tipo de patrón.

El trabajar con la imagen completa presenta dos grandes problemas. Para tener suficiente información para la clasificación, se requiere una cantidad en el orden de los miles de datos, lo que requiere grandes espacios en memoria y una gran cantidad de operaciones para realizar la comparación. El segundo problema tiene que ver con la distribución de la información.

Una imagen es un mapa en dos dimensiones de la información, la forma en que se ordena es netamente espacial, vale decir, allí donde se capta más luz el valor es mayor. Esta forma de ordenar es útil para interpretar visualmente la imagen, mas no es la mejor forma de ordenar la información para realizar comparaciones del contenido.

Al ordenar la información de esta manera es imposible discriminar su importancia, por lo tanto no se puede eliminar aquella irrelevante o excesiva. Alternativamente, se pueden ordenar los elementos según su relevancia, simplificando considerablemente la tarea de eliminar aquellos que no contengan información útil.

Fig. 2.1: Imágenes base de datos *AT&T* [19]

Este trabajo se enfoca en los métodos *Eigenfaces*, BDPCA, *Fisherfaces* y BDPCA+LDA; los dos primeros se basan en el análisis de componentes principales para reducir la dimensionalidad de los datos, con el fin de mantener la mayor de cantidad de información relevante. El tercer y cuarto método utilizan análisis de discriminantes lineales (LDA), específicamente los discriminantes lineales de Fisher, para encontrar una rotación que agrupe las imágenes de un mismo sujeto; facilitando considerablemente el reconocimiento.

En los métodos mencionados y, en general en los métodos basados en escena, la primera etapa consiste en calcular los coeficientes de proyección. Esta etapa, comúnmente llamada entrenamiento, se puede hacer tanto fuera de linea como en linea; no obstante como se mencionó en el capítulo anterior, resulta más efectivo realizar esta etapa fuera de linea para reducir la complejidad del hardware y reducir los errores de cálculo.

Paralelamente al desarrollo de los métodos de reconocimiento de rostros, se han creado numerosas bases de datos con la intención de comparar la efectividad de dichos métodos. Utilizando diferentes bases es posible evaluar el rendimiento de un algoritmo ante variados escenarios y optimizarlo para alguna situación en particular. Las principales diferencias entre bases de datos incluyen: la variación en posición y ángulo del rostro, el tiempo transcurrido entre cada imagen y otros.

Existen además diferentes opciones para clasificar las imágenes. Las más usadas son las medidas de distancia debido a los bajos requerimientos en hardware que implica su uso, principalmente por la regularidad y, en los casos cubiertos en este trabajo, simplicidad de los cálculos.

(a) Imagen no centrada (b) imagen centrada

Fig. 2.2: Imágenes base de datos Yale [3]

2.1. Bases de datos

La primera base de datos utilizada corresponde a la base de *AT&T* (también conocida como *ORL*), se trabajó con una versión de imágenes centradas (Fig. 2.1). Esta base de datos consta de 10 imágenes por persona, con un total de 40 personas (400 imágenes en total), tomadas en un laboratorio con diferencias en iluminación, expresiones y detalles faciales (como lentes o barba); todas las imágenes fueron tomadas de frente con pequeñas variaciones de movimientos laterales. La imágenes que se utilizarán están centradas y escaladas a 64 x 64 pixeles.

La segunda base de datos utilizada es la conocida como base *Yale*. Ésta cuenta con 11 imágenes por persona, con fuertes variaciones en luminosidad y expresión; a una resolución de 320 x 243 y un total de 15 personas (165 imágenes en total). Esta base presenta el inconveniente de que las imágenes no están centradas (figura 2.2(a)), por lo que se procesaron previamente a su uso. El procesamiento consistió en centrar y recortar cada imagen utilizando *GIMP* en forma manual (figura 2.2(b)), para luego redimensionarlas a 61 x 49 pixeles utilizando *Matlab Image Processing toolbox*, lo que reduce considerablemente los requisitos de memoria, sin impactar, en forma considerable, en el rendimiento [5].

Se evaluó utilizar también la base de datos *FERET*, la cual contiene 14051 imágenes, con una gran variedad en posiciones, iluminación, periodo entre fotografías, etc. Sin embargo, debido a la poca consistencia de la base de datos (algunos sujetos tienen 1 foto mientras otros tienen más de 60, algunas fotografías son a cuerpo completo, otras medio cuerpo o sólo el rostro), su uso requiere un alto nivel de preprocesamiento que escapa al tema del presente trabajo.

2.2. Clasificación de imágenes

La clasificación de una imagen se puede dividir en dos partes. La primera parte consta de un acondicionamiento de la información que reduce el número de elementos a niveles manejables, además de ordenar dichos elementos según la cantidad de información que aportan. La proyección LDA busca además agrupar las imágenes según a quien pertenecen (esta información debe ser conocida al momento de calcular los coeficientes).

La segunda parte es la clasificación en sí. Se debe decidir a quién, o a qué clase, pertenece la imagen acondicionada en la primera parte, mediante la aplicación de un método de clasificación. Los métodos más usados se basan en medidas de distancia, para comparar con una base de datos previamente conocida; también se pueden clasificar mediante el uso de redes neuronales entrenadas específicamente para mencionado fin.

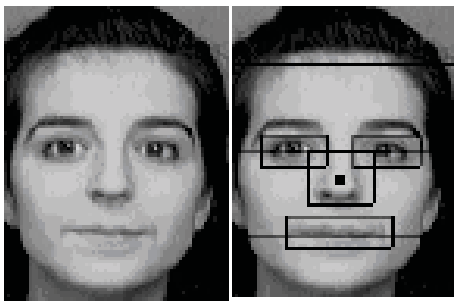
2.2.1. Pre-procesamiento de las imágenes

Como se mencionó, la clasificación se divide en dos partes, de las cuales el preprocesamiento de la imagen es, posiblemente, el más importante. Principalmente debido a la importante reducción de dimensionalidad que permite trabajar con sólo una pequeña fracción de los elementos y con mejores resultados [23]. A continuación se presentaran algunas de las diferentes opciones, mas solo se expandirá en las que son de interés para el presente trabajo.

Métodos biométricos

La reducción de dimensionalidad más simple de una fotografía es realizar una descripción del contenido de ésta; en el caso de una cara un resumen puede ser la posición de ojos, la boca y la nariz (figura 2.3(a)). En base a estas (y otras) referencias o *landmarks* se puede intentar discriminar a quien pertenece la fotografía con variados resultados [1]. Estos métodos tienen la ventaja de ser robustos con respecto a variaciones en iluminación, cambios en resolución de la imagen, ruido y, dependiendo de la implementación, variaciones en postura y expresión.

Existen tres caminos a seguir una vez se dispone de la información sobre el rostro; lo más simple es utilizar esta información y comparar la similitud de dichas relaciones [17], un segundo método consiste en generar un modelo (bidimensional o tridimensional) del rostro y utilizar los parámetros resultantes para clasificar [4, 12].



(a) *Landmarks* en rostro [17]

(b) Ejemplo vista sintetizada [4]

Fig. 2.3: Ejemplos método biométrico

La tercera vía consiste en, a partir de un modelo del rostro, generar vistas sintéticas "normalizando" pose, iluminación y expresión. Luego se utilizan estas vistas como imágenes a clasificar (figura 2.3(b)) [20]. Este última vía se considera híbrida pues utiliza los *landmarks* del rostro para normalizar la imagen y luego aplicar un método basado en escena.

Un problema de los métodos biométricos es su sensibilidad a las referencias utilizadas y a la cantidad de puntos designados para representarlas; no es lo mismo utilizar un punto para el ojo, que utilizar 25 puntos para delinear el ojo. Mucha de la información de identidad se encuentra también en las cejas, forma del rostro y color de la piel; no sólo en los ojos, nariz y boca [22].

Métodos basados en escena

A diferencia de los métodos biométricos, los algoritmos basados en escenas no buscan ni identifican un rostro, sino comparan similitudes de la imagen completa, ya sea que tenga una cara, una letra o un dibujo. Es importante entonces un procesamiento previo que normalice y centre las imágenes.

Se han desarrollado variados métodos para cuantificar las similitudes, algunos utilizando el Análisis de Componentes Principales (PCA), complementado con Análisis de Discriminantes Lineales (LDA) (también conocidos como *Eigenfaces* y *Fisherfaces* respectivamente) [23, 3]. Otros parten del supuesto de que un rostro se compone de componentes independientes y realiza un Análisis de Componentes Independientes (ICA) [2] o busca detalles en la imagen utilizando

wavelets [6, 24]. La cantidad de métodos diferentes es vasta (redes neuronales [18], implementaciones no lineales [14], etc.), cada uno presenta ventajas y desventajas dependiendo del escenario y plataforma utilizada [21, 7, 8].

Todos estos métodos utilizan las imágenes como puntos en espacios multidimensionales. Alternativamente, se puede procesar la imagen como matriz, aprovechando así la información espacial inherente en la imagen. A continuación se desarrollan los métodos de *Eigenfaces* y *Fisherfaces* para luego presentar dos métodos que utilizan la imagen como matriz, para realizar la reducción de dimensionalidad.

Métodos basados en escena: *Eigenfaces* y *Fisherfaces*

Dos métodos clásicos son *Eigenfaces* y *Fisherfaces*. Estos algoritmos buscan una proyección sobre un subespacio que cumpla con condiciones específicas.

El método de *Eigenfaces* es, en principio, un análisis de componentes principales (PCA). Éste se basa en utilizar una transformación o proyección lineal, que maximice la varianza entre las muestras (puntos en el espacio).

Partiendo con un conjunto de observaciones $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$ con $\mathbf{x}_i \in \mathbb{R}^n$, consideremos también una matriz $\mathbf{W} \in \mathbb{R}^{(n \times m)}$ con columnas ortonormales y $m < n$. Se puede definir una proyección lineal $\mathbf{y}_k$ como en (2.1) y una matriz de varianza $\mathbf{S_T}$ como en (2.2).

$$\mathbf{y}_k = \mathbf{W}^T \cdot \mathbf{x}_k \qquad k = 1, 2, ..., N \tag{2.1}$$

$$\mathbf{S_T} = \sum_{k=1}^{N} (\mathbf{x}_k - \mu) \cdot (\mathbf{x}_k - \mu)^T \tag{2.2}$$

donde $\mu = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n$. Si se aplica la transformación 2.1 a cada elemento del conjunto de observaciones, $\mathbf{X}$, resulta un nuevo conjunto $\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_N\}$, cuya matriz de varianza se puede calcular reemplazando (2.1) en (2.3) y finalmente, reeordenando se obtiene (2.6)

$$\mathbf{S_{T_y}} = \sum_{k=1}^{N} (\mathbf{y}_k - \mu_y) \cdot (\mathbf{y}_k - \mu_y)^T \tag{2.3}$$

$$\mathbf{S_{T_y}} = \sum_{k=1}^{N} (\mathbf{W}^T \mathbf{x}_k - \mathbf{W}^T \mu) \cdot (\mathbf{W}^T \mathbf{x}_k - \mathbf{W}^T \mu)^T \tag{2.4}$$

$$\mathbf{S_{T_y}} = \sum_{k=1}^{N} \mathbf{W}^T (\mathbf{x}_k - \mu) \cdot (\mathbf{W}^T (\mathbf{x}_k - \mu))^T \tag{2.5}$$

$$\mathbf{S_{T_y}} = \mathbf{W}^T (\sum_{k=1}^{N} (\mathbf{x}_k - \mu) \cdot (\mathbf{x}_k - \mu)^T) \mathbf{W} = \mathbf{W}^T \cdot \mathbf{S_T} \cdot \mathbf{W} \tag{2.6}$$

Luego, como la intención es maximizar la varianza, se debe resolver el problema de optimización presentado en (2.7). Se demuestra que la solución a este problema es una matriz compuesta por los vectores propios en orden descendiente [3].

$$\mathbf{W}_{opt} = \underset{\mathbf{W}}{\operatorname{argmax}} |\mathbf{W}^T \mathbf{S_T} \mathbf{W}| \tag{2.7}$$

Para reducir la dimensionalidad, se descartan los vectores propios asociados a los menores valores propios (en valor absoluto), pues contienen la menor cantidad de información. La cantidad a descartar no es fija, sino depende de las características de las imágenes que se están utilizando; en *Fisherfaces* se define un máximo como se verá a continuación

El método de *Fisherfaces* es un análisis de discriminantes lineales (LDA), particularmente los discriminantes lineales de Fisher, donde se busca aprovechar la información, normalmente disponible, sobre la clasificación de las imágenes de entrenamiento, para buscar una proyección que maximice la separación entre imágenes de diferentes personas (o clases) y minimice la distancia entre imágenes de una misma clase; así logra concentrar las imágenes mejorando, en forma importante, la tasa de reconocimiento [3].

Se define la matriz de varianza entre clases (imágenes de personas distintas) como en (2.8) y la varianza dentro de la clase (imágenes de una misma persona) como en (2.9). Donde μ_i corresponde a la imagen promedio de la clase χ_i, $|\chi_i|$ al número de puntos dentro de la clase χ_i y μ es el promedio de todas las imágenes.

$$\mathbf{S_B} = \sum_{i=1}^{c} |\chi_i| (\mu_i - \mu)(\mu_i - \mu)^T \tag{2.8}$$

$$\mathbf{S_W} = \sum_{i=1}^{c} \sum_{\mathbf{x}_k \in \chi_i} (\mathbf{x}_k - \mu_i)(\mathbf{x}_k - \mu_i)^T \tag{2.9}$$

Luego, en forma similar a PCA, se debe encontrar la matriz de proyección $\mathbf{W} \in \mathbb{R}^{(n \times m)}$, resolviendo el problema de optimización presentado en (2.10), donde se debe minimizar $\mathbf{S}_W$ se maximizar $\mathbf{S}_B$. El resultado de dicho problema es una matriz con los vectores propios de $\mathbf{S_B S_W}^{-1}$ [3].

$$\mathbf{W}_{opt} = \underset{\mathbf{W}}{\text{argmax}} \frac{|\mathbf{W}^T \mathbf{S_B} \mathbf{W}|}{|\mathbf{W}^T \mathbf{S_W} \mathbf{W}|} \tag{2.10}$$

Normalmente la solución mencionada no existe, pues $\mathbf{S_W}$ es singular; debido a una dimensionalidad mucho mayor a la cantidad de imágenes. Una práctica común es aplicar una reducción de dimensionalidad previo al análisis con discriminantes lineales, así se logra que $\mathbf{S_W}$ pierda su característica de no singular. Para esto se debe reducir la dimensionalidad de las imágenes a no más de *número de imágenes - número de clases* componentes. Luego se puede aplicar LDA para reducir a *número de clases - 1* elementos y agrupar las imágenes según la clase a la que pertenecen. Si la primera reducción se realiza utilizando *Eigenfaces*, al método se le llama *Fisherfaces*.

Así el método de *Fisherfaces* se puede separar en dos partes. Primero resolver (2.7) para *Eigenfaces* y luego resolver (2.10) con $\mathbf{S_B}$ y $\mathbf{S_W}$ creadas a partir de las imágenes proyectadas sobre el subespacio de *Eigenfaces*. En forma equivalente se puede resolver (2.11) con $\mathbf{W_{PCA}}$ la matriz resultado de (2.7).

$$\mathbf{W}_{opt} = \underset{\mathbf{W}}{\text{argmax}} \frac{|\mathbf{W}^T \mathbf{W_{PCA}}^T \mathbf{S_B} \mathbf{W_{PCA}} \mathbf{W}|}{|\mathbf{W}^T \mathbf{W_{PCA}}^T \mathbf{S_W} \mathbf{W_{PCA}} \mathbf{W}|} \tag{2.11}$$

Éstos métodos han mostrado gran eficiencia y popularidad debido a la relativa simpleza. Sin embargo, la cantidad de cálculos y requerimientos de memoria son muy altos debido a las grandes dimensionalidades de las imágenes.

En la figura 2.4 se presentan 12 imágenes pertenecientes a 4 clases (3 por clase) y su proyección a los subespacios *Eigenfaces* y *Fisherfaces*. Para esta última proyección se utilizó primero *Eigenfaces* reduciendo a cuatro elementos para comparar con la proyección BDPCA+LDA expuesta más abajo. Resulta claro como aplicar la proyección *Fisherfaces* agrupa las imágenes, eventualmente facilitando la clasificación.

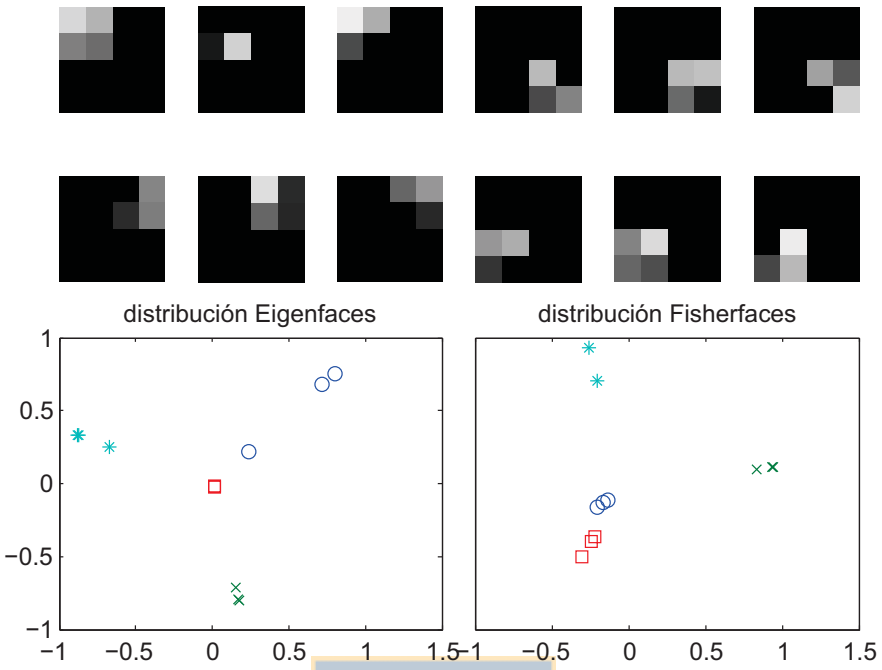


Fig. 2.4: Ejemplo proyección *Eigenfaces* y *Fisherfaces*

Métodos basados en escena: BDPCA y BDPCA+LDA

Debido a las restricciones de los métodos discutidos, se han desarrollados métodos alternativos para lograr proyecciones más eficientes en cálculos sin sacrificar rendimiento o incluso mejorándolo. Existen implementaciones con redes neuronales que logran buenos resultados. Sin embargo el entrenamiento se dificulta debido, nuevamente, al tamaño de las matrices.

Alternativamente, existe un método de PCA por bloques, donde se divide la imagen en bloques y se aplica PCA a dichos bloques. Un caso particular es el llamado *2DPCA* [27, 13] donde se trabaja con las imágenes como matrices en lugar de puntos en el espacio, este método logra resultados similares pero con un importante ahorro en cálculos.

La idea tras *2DPCA* es buscar una proyecciones que genere una reducción por filas, o por columnas, de la imagen como matriz. Esto es, obtener $\mathbf{Y}$ como una proyección de $\mathbf{X}$ sobre el espacio generado por $\mathbf{A}$ (2.12).

$$\mathbf{Y} = \mathbf{X} \cdot \mathbf{A} \tag{2.12}$$

El problema nuevamente se traduce en elegir una matriz $\mathbf{A}$ adecuada. Similar a PCA se busca maximizar la varianza entre las imágenes, se utiliza la covarianza de imágenes, ecuación (2.13).

$$\mathbf{G}_t = \frac{1}{M} \sum_{j=1}^{M} (\mathbf{X}_j - \bar{\mathbf{X}}^T) \cdot (\mathbf{X}_j - \bar{\mathbf{X}}) \tag{2.13}$$

Se puede demostrar que la matriz $\mathbf{A}$ se compone de los primeros d vectores propios de la matriz $\mathbf{G}_t$ [27]. Este método reduce la dimensionalidad de la matriz en una de las dimensiones (reduce la cantidad de filas). El método BDPCA se puede describir como dos problemas *2DPCA*, uno para reducir filas y uno para reducir columnas (2.14).

$$\mathbf{Y} = \mathbf{W_C}^T \mathbf{X} \mathbf{W_R} \tag{2.14}$$

Con $\mathbf{W_R}$ y $\mathbf{W_C}$ matrices de proyección por columnas y por filas respectivamente, *2DPCA* asume que $\mathbf{W_C}$ es una matriz identidad de tamaño apropiado; el algoritmo BDPCA calcula dicha matriz y reduce filas y columnas. Para el cálculo de $\mathbf{W_R}$ y $\mathbf{W_C}$ existen tres caminos:

Estrategia Jerárquica: Consiste en primero realizar la proyección en un sentido y luego proyectar la matriz reducida en el otro sentido. Este método requiere de seleccionar un sentido por sobre el otro.

Estrategia Iterativa: Consiste en fijar un $\mathbf{W_C}$ inicial para luego actualizar $\mathbf{W_R}$ y, a partir de este valor, volver a actualizar $\mathbf{W_C}$. Este método teóricamente asegura encontrar un óptimo local.

Suposición de independencia: Este método asume que el cálculo de $\mathbf{W_C}$ y $\mathbf{W_R}$ son independientes por lo cual realiza los cálculos como dos problemas de *2DPCA*.

Se ha comprobado que la eficacia es prácticamente independiente de la forma de calcular las matrices, no obstante, el primer y tercer métodos son considerablemente más rápidos [28]. Ambos consisten de dos proyecciones *2DPCA*; por filas y por columnas. El tercer método tiene una ventaja al momento de implementar, al considerar que ambas proyecciones son independientes otorga independencia en el orden de la multiplicación.

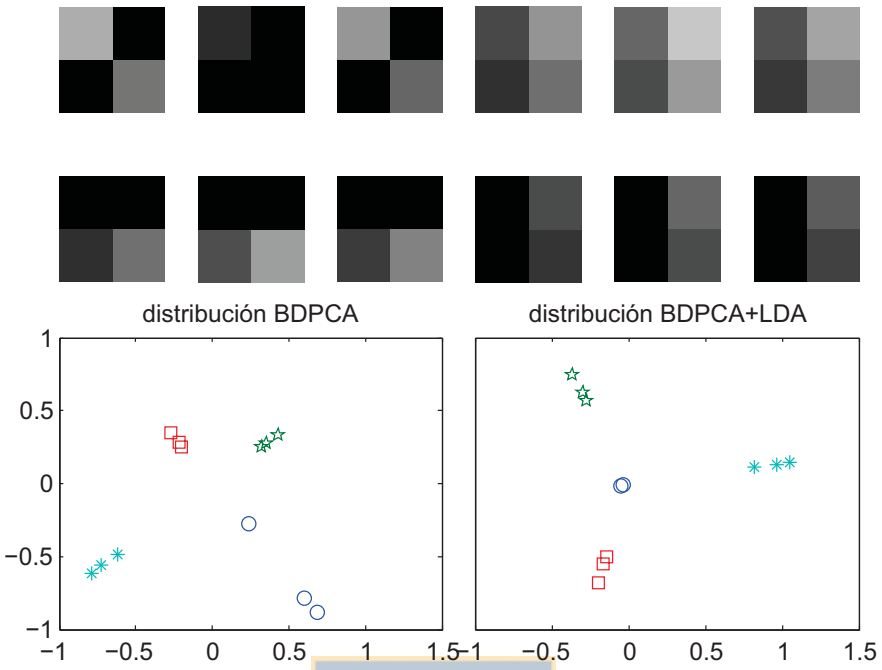
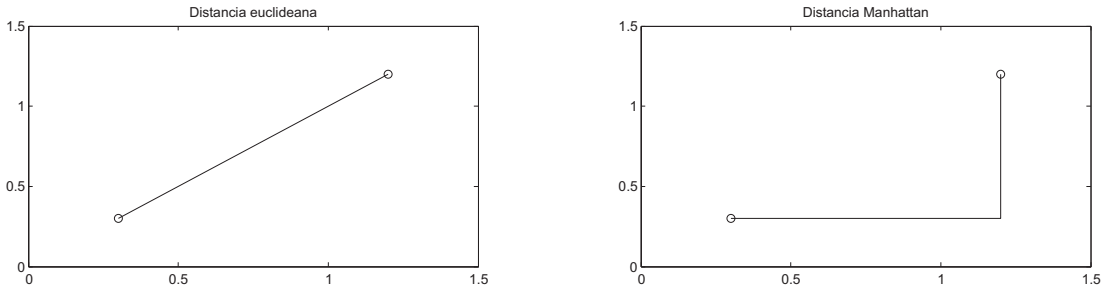


Fig. 2.5: Ejemplo proyección BDPCA y BDPCA+LDA

En forma similar a PCA, BDPCA ignora la información sobre clasificación que pueda existir. Para mejorar la tasa de reconocimiento se puede aplicar una clasificación con LDA (como en *Fisherfaces*) para una clusterización y reducción final, a esto se le llama BDPCA+LDA [29].

En la figura 2.5 se presenta un ejemplo de la proyección BDPCA y BDPCA+LDA. Las 12 imágenes superiores son el resultado de la proyección BDPCA a cuatro componentes de las imágenes de 16 componentes presentadas anteriormente (fig. 2.4), de éstos se toman los dos primeros componentes para graficar la distribución BDPCA; es importante notar que los coeficientes no están ordenados según su importancia (como ocurre con PCA). La proyección LDA se realiza utilizando los cuatro elementos de la proyección BDPCA. Se observa claramente como la segunda proyección agrupa las diferentes clases lo cual facilitaría considerablemente su clasificación, incluso mejor que *Fisherfaces* para el set de imágenes presentado.



(a) Ejemplo distancia euclidiana (b) Ejemplo distancia Manhattan

Fig. 2.6: Mediciones de distancia

2.2.2. Clasificación de las imágenes

Una vez transformadas las imágenes se procede a la clasificación en sí, para realizar ésto existen varios métodos, desde simples cálculos de distancia hasta clasificación mediante redes neuronales. En este trabajo se prueban dos medidas de vecino más cercano y se comparan en términos de simplicidad y efectividad.

El método de vecino más cercano consiste en evaluar la distancia entre un punto a clasificar y cada uno de los puntos cuya clase se conoce; se puede asumir entonces que el punto más cercano corresponde a la misma clase que el punto a clasificar.

Para este método existen dos formas de clasificar: Una es utilizando todas las imágenes y comparar con respecto a todo el set de entrenamiento. El segundo método consiste en utilizar un promedio de la clase como referencia y medir la distancia respecto a dichos promedios. Claramente, el utilizar un promedio significa un ahorro considerable de memoria y elimina errores causados por imágenes muy dispersas. Por ello sólo se evaluará este último método.

La forma de medir la distancia tiene un gran impacto en la complejidad de los cálculos que se deben realizar, por esto se utilizarán dos métodos. La distancia euclidiana expresada en (2.15) es la distancia en *linea recta* entre los dos puntos. No obstante, esta medida requiere calcular n cuadrados y una raíz. Alternativamente se puede prescindir de la raíz pues interesa la comparación de las distancias, no la distancia en sí.

$$d = \sqrt{(\Delta x_1)^2 + (\Delta x_2)^2 + ... + (\Delta x_n)^2} \tag{2.15}$$

La segunda distancia a probar es la distancia Manhattan, donde la distancia se representa por la suma de las distancias en los diferentes ejes (2.16), éste método es mucho más simple de implementar, mas no es equivalente a la distancia euclidiana y por ende entregará resultados diferentes.

$$d = |\Delta\, x_1| + |\Delta\, x_2| + ... + |\Delta\, x_n| \tag{2.16}$$

La figura 2.6 muestra la distancia medida por cada método mencionado. Realizando la medición resulta claro que las distancias no son las mismas, en el caso de la distancia euclidiana (fig. 2.6(a)) la distancia es de 1,27 y en la distancia Manhattan (fig. 2.6(b)) 1,80. El impacto en la clasificación de imágenes de mencionada diferencia se verificará más adelante.

Capítulo 3

Descripción del hardware

Existen variadas tarjetas de desarrollo de hardware programable disponibles en el mercado, con diferentes chips, accesorios y precios. Para el presente proyecto se utiliza una tarjeta Digilent Nexys que cuenta con un chip Spartan 3, una memoria RAM de 16MB y múltiples vías de entrada y salida; de las cuales se destaca una interfaz USB para la programación de la FPGA.

Si bien no es un requisito, durante el desarrollo del proyecto se necesita controlar el flujo de operación para depurar y verificar resultados. Para este fin la tarjeta Nexys cuenta con cuatro visores de siete segmentos, ocho LEDs, ocho interruptores y dos pares de pulsadores.

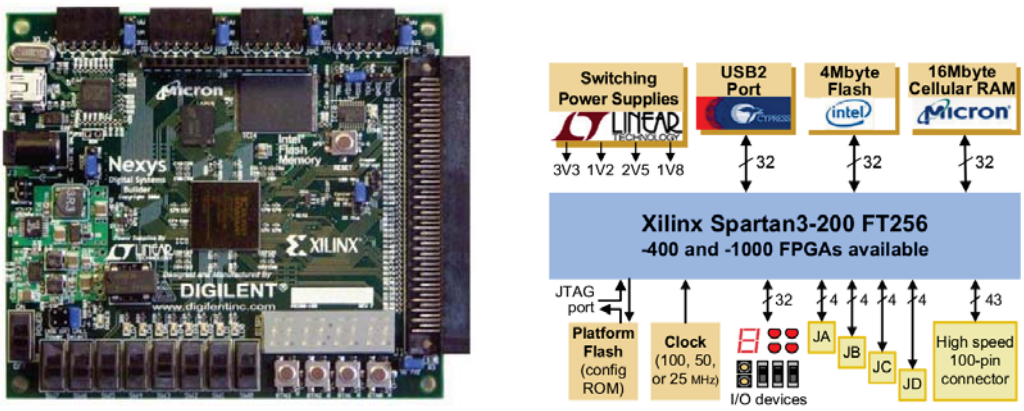
3.1. Tarjeta de desarrollo Nexys

La tarjeta de desarrollo Nexys proporciona todos los elementos básicos para el desarrollo de un proyecto sobre una FPGA y cuenta con numerosas conexiones de entrada y salida para anexar más dispositivos. En el presente caso, los componentes integrados en la tarjeta son suficientes para suplir los requerimientos del proyecto a realizar, manteniendo siempre un bajo consumo de energía (el consumo máximo total de la tarjeta de desarrollo Nexys es de $495mW$ [9])

Para el desarrollo del proyecto se necesita, además de la FPGA en sí, una memoria donde almacenar la imagen a clasificar; esta memoria debe ser, de preferencia, externa a la FPGA, pues debe ser modificada por un dispositivo externo para cada nueva imagen a clasificar. La tarjeta Nexys cuenta con dos memorias que sirven para este propósito: Una Intel Flash ROM de 4MB y una Micron PSDRAM de 16MB, para el proyecto se utiliza la memoria PSDRAM.

Tabla 3.1: Resumen características de tarjeta de desarrollo Nexys

Chip:	XC3S200 Spartan3 FPGA con 1000K puertas disponibles.
Conector:	Conector de 100-pin Hirose FX2
Programación:	A través del conector JTAG utilizando el puerto USB2 (a través del software gratuito disponible); a través de conector JTAG con el puerto paralelo; además existen numerosas opciones de configuración de la FPGA.
Características:	▪ FPGA de 1000K puertas Xilinx XC3S200 con una operación de 500+MHz. ▪ Configuración y transferencia de datos a alta velocidad a través del puerto USB ▪ Alimentación a través del Puerto USB (baterías o fuente de poder también pueden ser utilizados) ▪ Cuenta con 16MB de PSDRAM Micron y 4MB de ROM flash Intel StartaFlash. ▪ ROM Flash con plataforma Xilinx que guarda la configuración de la FPGA indefinidamente. ▪ Tecnología de alta eficiencia en fuente de alimentación (excelente para aplicaciones donde se debe utilizar baterías) ▪ Oscilador de 50MHz y 100Hz ▪ Conector para panel LCD (resolución 1/8 VGA) o pantalla LCD de 16×2 caracteres ▪ Cuenta con 60 entradas/salidas de la FPGA para ser utilizadas como expansión (un conector de alta velocidad Hirose FX2 y cuatro conectores de 6 pines) ▪ Además se cuenta con 8 LEDs, display de siete segmentos con cuatro dígitos, 4 pulsadores, y 8 interruptores.



(a) Placa de desarrollo Nexys (b) Diagrama de bloques de placa Nexys

Fig. 3.1: Placa Nexys [9]

Dado que la FPGA operará esencialmente independiente de un PC, se requiere un método para depuración y verificación de resultados directamente en la placa. Para esto se utilizan: cuatro visores de siete segmentos, ocho leds, ocho interruptores y cuatro pulsadores incluidos en la plataforma Nexys.

Para la programación de la FPGA y la memoria RAM, la tarjeta cuenta con una interfaz USB y un software para plataformas Microsoft Windows, que permiten la carga del archivo de programación en forma simple. Un resumen de las características de la tarjeta de desarrollo se encuentra en la tabla (3.1).

La figura 3.1(a) muestra la placa de desarrollo a utilizar, donde se puede ver claramente los visores de siete segmentos, interruptores, pulsadores y, en el centro, la FPGA Spartan 3. En la figura 3.1(b) se presenta el diagrama en bloques de la tarjeta, donde se aprecian los componentes y su conexión a la FPGA [9].

3.2. Chip FPGA Spartan 3

Sin duda el componente más importante para el diseño y desarrollo del proyecto es la FPGA, en este caso se utiliza un Spartan 3 de Xilinx; este chip presenta un diseño con buena relación tamaño-capacidad y costo-capacidad. Un breve resumen de sus características se presenta en la tabla 3.2 [26].

Tabla 3.2: Características Spartan3 XC3S1000

Puertas del sistema		1M
Celdas lógicas equivalentes		17.280
CLB (un CLB = cuatro Slices)	Filas	48
	Columnas	40
	CLBs Total	1.920
Bits de RAM distribuido (K=1024)		120K
Bits de bloque de RAM (K=1024)		432K
Multiplicadores dedicados		24
DCMs		4
Máximo de E/S		391
Máximo de E/S diferenciales		175

3.2.1. Características

Dado que la FPGA Spartan 3 es un chip económico, carece de elementos dedicados para cálculos complejos, tales cómo aritmética en punto flotante o raíz cuadrada; más aún, dispone sólo de 24 multiplicadores de 18 bits con signo. Estas son las restricciones más importantes al momento de evaluar los métodos y diseñar la implementación, significa que se debe utilizar aritmética en punto fijo o entera y no más de 24 multiplicaciones de 18 sbits simultaneas. A continuación se presenta un desglose general de las características y capacidades de mencionado chip.

- Bajo costo, solución lógica de alto rendimiento orientada a aplicaciones de consumo,
  - Densidad de 17.280 celdas lógicas
- Selección de interfaces de entradas-salidas,
  - 391 pines de entrada-salida
  - 622 Mb/s de trasferencia por puerto
  - 18 estándares simples.
  - 8 estándares diferenciales, incluyendo LVDS, RSDS
  - Terminaciones con controladores digitales de impedancia
  - Rango de variación de señales desde 1.14V a 3.465V
  - Soporte DDR, DDR2 y SDRAM sobre 333Mbps

- Recursos de lógica,
  - Abundancia de celdas lógicas con capacidad de desplazamiento
  - Multiplexores rápidos
  - 24 multiplicadores dedicados de 18bits
  - Lógica JTAG compatible con IEEE 1149.1/1532
- Selección jerárquica de RAM
  - 432 Kbits de bloques de RAM
  - 120 Kbits de RAM distribuida.
- Director de reloj digital (sobre 4 DCMs)
  - Eliminación de distorsión
  - Síntesis de frecuencia
  - Alta resolución de cambio de fase
- 8 líneas globales de reloj y abundancia de rutas

3.2.2. Visión general de la arquitectura

La arquitectura de la familia de FPGAs Spartan 3 consiste de cinco elementos funcionales programables fundamentales.

- Bloques lógicos configurables (CLBs): Contienen tablas de acceso rápido (LUTs), para implementar la lógica y almacenamiento de elementos que pueden ser usados como flip-flops o latches. Los CLBs pueden ser programados para realizar una gran variedad de funciones lógicas o también para guardar datos.
- Bloques entrada/salida (IOBs): Controlan el flujo de información entre pines de entrada/-salida y la lógica interna del dispositivo. Cada IOBs soporta flujo bidireccional y operación en 3 estados. 26 estándares diferentes de señales se encuentran disponibles. El controlador digital de impedancia (DCI) permite una terminación automática, simplificando el diseño de la tarjeta.
- Bloques de RAM: Permiten guardar datos en bloques duales de 18Kbit.

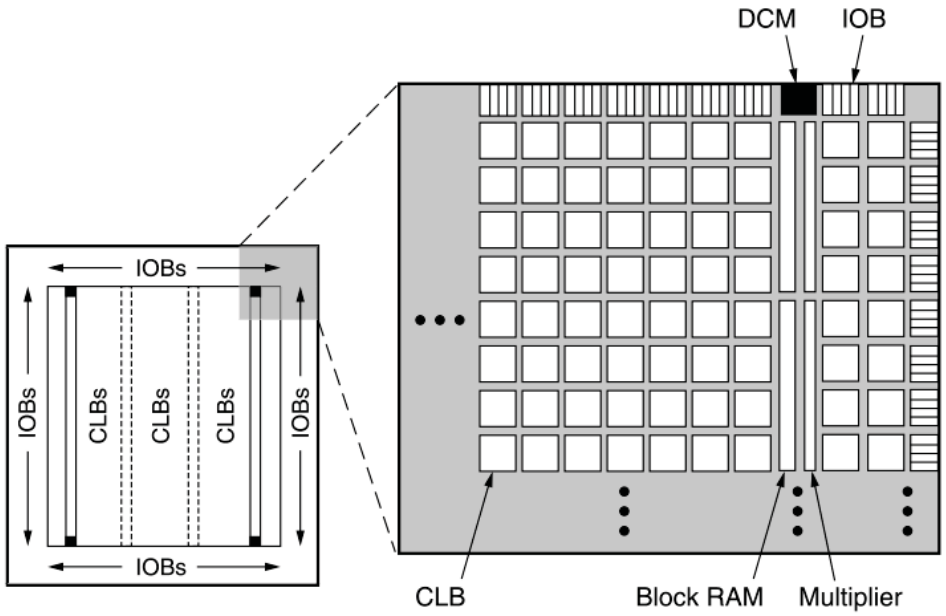


Fig. 3.2: Diagrama FPGA [26]

- Bloques de multiplicadores: Aceptan dos números binarios de 18-bits como entradas para calcular el producto.
- Controlador de reloj digital (DCM): Proporciona una completa, y auto-calibrada, solución digital para la distribución, desplazamiento, multiplicación, división y desfase de señales de reloj.

Todos estos elementos se encuentran organizados como se muestra en la figura 3.2. Un anillo de IOBs rodea un arreglo regular de CLBs que, dependiendo de la versión del chip, puede tener una, dos o cuatro columnas de block-RAM. Cada columna se encuentra compuesta por un bloque RAM de 18Kbit y se encuentra asociada a un multiplicador dedicado. Los DCMs se encuentran posicionados en los extremos de las columnas de block-ram externas.

Para la interconexión de los elementos funcionales mencionados, la familia de FPGA Spartan 3 posee una red de lineas e interruptores para la transmisión de señales entre ellos. Cada elemento tiene una matriz de interruptores asociada para permitir múltiples conexiones al rutear.

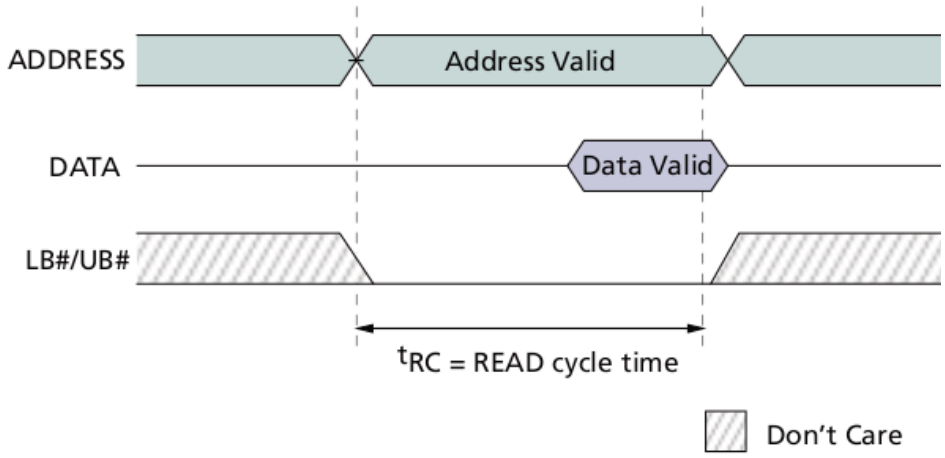


Fig. 3.3: Comportamiento de señales para lectura asincrónica [15]

3.3. Memoria Micron PSDRAM 16MB

La FPGA por sí sola no es suficiente para efectuar un reconocimiento de rostros efectivo, es necesaria además una memoria externa a la FPGA que la alimente con el rostro a clasificar. Para este fin se utiliza la memoria Micron PSDRAM de 16MB presente en la tarjeta de desarrollo Nexys.

La memoria Micron CellularRAM es una memoria CMOS de alta velocidad de acceso aleatorio pseudo-estática, diseñada especialmente para aplicaciones portátiles y de bajo consumo (menor a 48,8mW [15]). La unidad utilizada posee un núcleo de 128Mb organizado en 8M de 16b. A pesar de contar con una interfaz de alta velocidad mediante *bursts*, solo se solicitará un dato por consulta, pues se busca lograr una implementación independiente de la memoria utilizada. Así mismo se aprovecharán los mecanismos de auto refresco que permiten una operación similar a la de una memoria flash, sin afectar el rendimiento de lectura/escritura [15].

En la figura 3.3 se observan el comportamiento de las señales para la lectura de de la memoria, el valor de t_{RC} es de 80ns. En otras palabras, entre cada lectura de la memoria se debe esperar por lo menos 80ns.

Capítulo 4

Implementación en PC

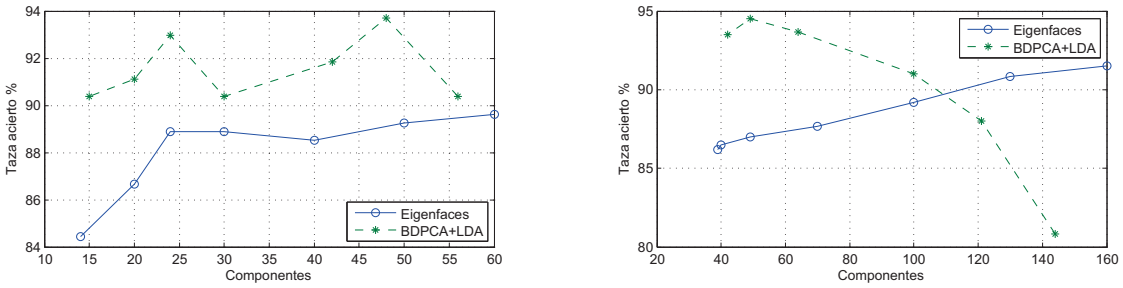
Para realizar la selección del algoritmo a implementar, se evaluarán los métodos mencionados en el capítulo 2. Al momento de evaluar se tomará en consideración que el algoritmo finalmente se implementará en el hardware presentado en la sección 3, por lo que se dará importancia no a la eficacia, sino a la eficiencia respecto al uso de memoria y de operaciones necesarias.

4.1. Método de prueba

Para la evaluación de los métodos mencionados se utilizarán las bases de datos *Yale* centrada y *AT&T* (descritas en 2.1); cada método se prueba tres veces, para luego calcular un promedio. En cada ejecución se seleccionan al azar 5 imágenes de cada clase para el entrenamiento y el resto (5 por clase en el caso de la base *AT&T* y 6 en las base *Yale*) para evaluar los métodos.

Lo primero que se fija es el tamaño inicial de las imágenes; los intentos de verificar tamaños muy por sobre los 4000 elementos no es posible por limitaciones de memoria en la plataforma donde se trabaja. Por esto se decide utilizar 61x49 para la base *Yale* y 64x64 para la base *AT&T*. De [5] se sabe además que resoluciones mayores no mejoran considerablemente el rendimiento, mas resoluciones menores comienzan a perder rendimiento.

En la figura 4.1 se muestran los rendimientos con distintas reducciones, se comparan sólo *Eigenfaces* y BDPCA+LDA con distancia Manhattan para la medida de distancia; se encuentra que 24 y 49 es una reducción con que da buenos resultados para BDPCA+LDA.



(a) Base *Yale* (b) Base *AT&T*

Fig. 4.1: Variación rendimiento PCA según elementos de reducción

A partir de los resultados expuestos, se decide, para los métodos *Eigenfaces* y BDPCA, utilizar 24 y 49 elementos para las bases *Yale* y *AT&T* respectivamente. Para la proyección LDA (*Fisherfaces* y BDPCA+LDA) se utilizan *número de clases - 1* coeficientes (14 y 39 coeficientes respectivamente) [3, 29].

Para la clasificación de las imágenes proyectadas se evalúan distancia Manhattan y distancia euclidiana. En ambos casos, se compara la distancia con el promedio de cada clase. La idea de utilizar un representante por clase es reducir en forma importante la memoria necesaria para almacenar la base de datos, acelerar la etapa de clasificación disminuyendo la cantidad de comparaciones necesarias y eliminar errores por imágenes que se alejen del conjunto.

Para la implementación en hardware, es importante conocer una estimación de los requisitos de memoria y la cantidad operaciones aritméticas para cada método. Esta estimación se basa en la dimensionalidad de las matrices de proyección y la base de datos para la clasificación.

Una vez seleccionado el método se medirá la velocidad de clasificación en PC con el fin de tener una referencia con la cual comparar la implementación en hardware.

Finalmente, se evalúa el impacto de la reducción de bits en los coeficientes, para seleccionar la representación que mejor se adecue al hardware sin sacrificar rendimiento.

4.2. Resultados y análisis

Como se puede apreciar en las tablas 4.1 y 4.2, el rendimiento varia fuertemente dependiendo de la base de datos utilizada. En el caso de la base de datos *Yale* el método más eficaz es

Tabla 4.1: Rendimiento y costo para diferentes métodos, base de datos *Yale*.

Distancia	Algoritmo	Rendimiento		Uso de recursos		
		Hit rate [%]	SD [%]	Mem [kw]	Mults [kops]	Adds [kops]
Manhattan	PCA	88.9	2.2	72.1	71.7	72.1
	LDA	97.8	1.9	42.1	41.8	42.1
	BDPCA	82.2	3.6	0.9	13.4	12.7
	BDPCA+LDA	93.0	3.4	1.1	13.8	12.9
Euclidean	PCA	85.2	1.7	72.1	72.5	72.1
	LDA	97.8	1.1	42.1	42.3	42.1
	BDPCA	82.6	3.4	0.9	14.1	12.7
	BDPCA+LDA	91.1	1.1	1.1	14.2	12.9

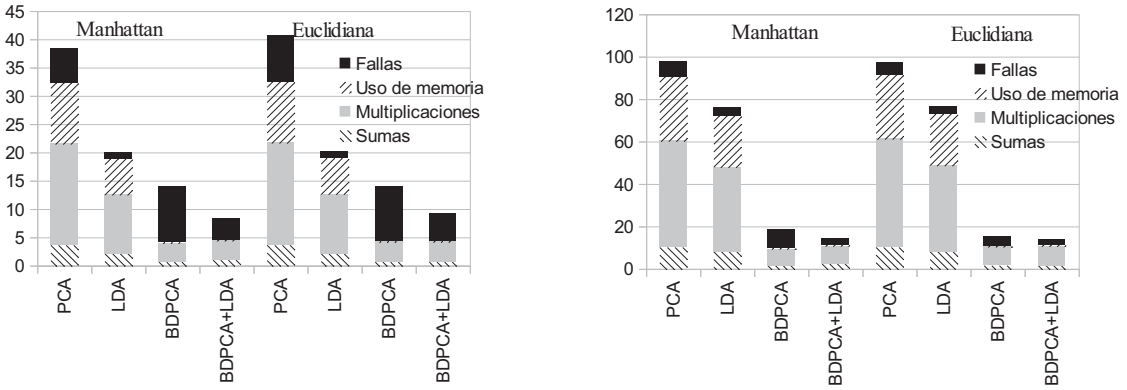
Tabla 4.2: Rendimiento y costo para diferentes métodos, base de datos *AT&T*.

Distancia	Algoritmo	Rendimiento		Uso de recursos		
		Hit rate [%]	SD [%]	Mem [kw]	Mults [kops]	Adds [kops]
Manhattan	PCA	87	2.5	202.7	200.7	202.7
	LDA	92.8	1.6	161.3	159.7	161.3
	BDPCA	84.3	1.4	2.9	31.8	30.3
	BDPCA+LDA	94.5	1.7	4.4	33.7	48.6
Euclidean	PCA	89.5	0.5	202.7	204.6	202.7
	LDA	93.8	1.3	161.3	162.9	161.3
	BDPCA	91.5	0.9	2.9	35.8	30.3
	BDPCA+LDA	95.7	1.3	4.4	36.7	31.8

Fisherfaces; casi un 5 % por sobre el segundo. Por el contrario, en la base *AT&T* el método más eficaz es BDPCA+LDA con distancia euclidiana, superando por 1 % al mismo método usando distancia Manhattan y por 2 % a *Fisherfaces* usando distancia euclidiana. El segundo parámetro considerado, los requisitos de memoria y cantidad de operaciónes utilizada, dan por claro vencedor a BDPCA y BDPCA+LDA, que requieren menos del 3 % de la memoria de PCA o *Fisherfaces* y menos de un quinto de los cálculos.

Para comparar los métodos se asignan pesos de importancia a las características evaluadas. La más importante es, sin duda, la taza de aciertos, pues si el sistema no es capaz de clasificar correctamente pierde utilidad; se le asigna un 55 % a la tasa de fallos (100 % - taza de acierto).

Los siguientes tres aspectos son importante según el hardware donde serán implementados. Para este caso, debido a la importante limitación en multiplicadores, la cantidad de multiplica-



(a) Base *Yale* (b) Base *AT&T*

Fig. 4.2: Comparación métodos

ciones representa el 25 % de la decisión. El segundo recurso que se debe cuidar es la memoria, si bien hay suficiente para cualquiera de los métodos, está dividida en 24 bloques, esto significa que se pueden obtener hasta 24 elementos por ciclo; se le da un 15 % de la decisión. Finalmente, la cantidad de sumas se llevan el 5 % restante, pues si bien vuelven al sistema más complejo, la cantidad de sumadores paralelos es prácticamente ilimitada.

En la figura 4.2 se pueden visualizar los resultados de la evaluación; cada aspecto se multiplicó por la fracción mencionada. Resulta claro que, a pesar de tener mejor taza de acierto, el método de *Fisherfaces* no es el más apropiado para implementar. Debido principalmente a su alto costo en memoria y en operaciones de multiplicación. Si bien el método de BDPCA aparenta ser un buen candidato para la implementación, ostenta la mayor tasa de fallos al usar la base *Yale*. Por el contrario BDPCA+LDA presenta una efectividad similar en ambas bases y ocupa el primer lugar en la comparación. Finalmente se decide implementar el método BDPCA+LDA usando distancia Manhattan; se prefiere distancia Manhattan principalmente debido a su menor costo en multiplicaciones sin sacrificar en forma importante la eficacia.

4.3. Velocidad de clasificación

Después de tener seleccionado el método se procede a medir la velocidad del algoritmo. Para establecer un buen parámetro las multiplicaciones se programan en C, utilizando la libreria *GNU Scientific Library* (GSL) [10]. Se probarán dos escenarios, primero se supondrá que las

Tabla 4.3: Velocidad y consumo de implementación en PC

Velocidad con imagen en disco	1 imagen	254.1 μs
	flujo	3,935 imágenes/segundo
Velocidad con imagen en RAM	1 imagen	44.4 μs
	flujo	22,500 imágenes/segundo
Consumo estimado de potencia		17.2 W

matrices están disponibles en memoria RAM. El segundo escenario, tendrá la imagen en un disco duro y las matrices de proyección en memoria.

Debido a que los tiempos de proyección son cortos, se realizan 90 proyecciones y se calcula el tiempo de una proyección a partir del total. La ejecución del programa se realiza en un notebook de última generación (Intel Core i3 @2.13GHz, 3GB RAM DDR3, Windows 7), debido a que el tiempo varía levemente en cada ejecución se repite tres veces el experimento y se considera la velocidad promedio.

Además se realiza una estimación del consumo de energía utilizando la herramienta *Joulemeter*, disponible en el sitio de *Microsoft Research* [11].

En los resultados presentados en la tabla 4.3, se observa una velocidad muy superior si la imagen se almacena en RAM; esto pues el acceso a disco es muy lento en comparación al acceso a RAM. Considerando que para la implementación en FPGA, la imagen se leerá desde una memoria externa, se espera un cuello de botella similar.

4.4. Análisis de ancho de bits

Un factor importante al momento de trasladar el algoritmo a hardware, es la representación que se utilizará para los coeficientes y la base de datos almacenada. Debido a que no se dispone, en hardware, de multiplicadores en punto flotante, se amplifican los valores de los coeficientes por un exponente de dos apropiado, con tal de expresar cada valor en una cantidad de bits suficiente, para no generar un impacto significativo en el rendimiento.

La memoria interna de la FPGA utilizada permite una organización en valores de 32, 16 u ocho bits [26], sin embargo, debido a que los multiplicadores son de 18 bits (y se prefiere no usar más de uno por cada multiplicación), se evaluará utilizar valores de 16 bits, 8 bits y 4 bits, con tal de buscar el óptimo en términos de memoria y efectividad.

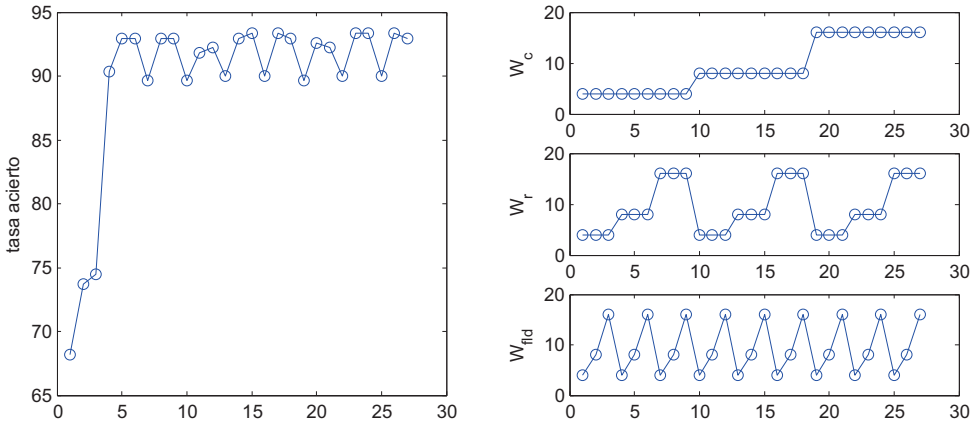


Fig. 4.3: Tasa acierto para diferentes cantidades de bits

Para realizar la comparación se divide el método BDPCA+LDA en tres partes, primero la proyección BDPCA, que consta de dos multiplicaciones de matrices, cada matriz, $\mathbf{W_C}$ y $\mathbf{W_R}$, puede ser almacenada con una cantidad de bits diferente. A continuación, para la proyección LDA, se asegura que los valores resultantes de BDPCA estén en no más de 16 bits y se optimiza la cantidad de bits para éste método. Finalmente, para la clasificación, se tiene más de libertad, pues solo requiere de sumadores y por tanto se puede hacer sin problemas en 32 bits.

La implementación se realizará para la base de datos *Yale*, pues se considera que es más representativa de lo que se podría encontrar en una situación real. Las pruebas de ancho de bits se realizan sobre esta misma base.

En la figura 4.3 se puede apreciar la variación de la tasa de acierto para los diferentes anchos de bit en $\mathbf{W_C}$, $\mathbf{W_R}$ y $\mathbf{W_{LDA}}$. Se elige utilizar 8 bits para $\mathbf{W_C}$ y $\mathbf{W_R}$ y 16 bits para $\mathbf{W_{LDA}}$. Pues es el resultado con menor uso de memoria y mayor tasa de acierto; superando levemente al resultado obtenido con la implementación en punto flotante.

4.5. Conclusiones

A partir de los resultados obtenidos se resuelve implementar el método de BDPCA+LDA utilizando la base *Yale* redimensionada a 61x49. Primero se realizará la proyección BDPCA para reducir a 24 elementos y luego se aplicará LDA para reducir a 14 elementos, finalmente se clasificará utilizando distancia manhattan para comparar con un promedio de la clase.

Para la implementación se decide almacenar las matrices $\mathbf{W_C}$ y $\mathbf{W_R}$ en ocho bits y $\mathbf{W_{LDA}}$ en 16 bits, para finalmente comparar con la base de datos utilizando 32 bits.

En base al resultado de la sección 4.4, se espera una tasa de acierto muy cercana al 93 % o, incluso, levemente superior; duplicando el resultado de la simulación.

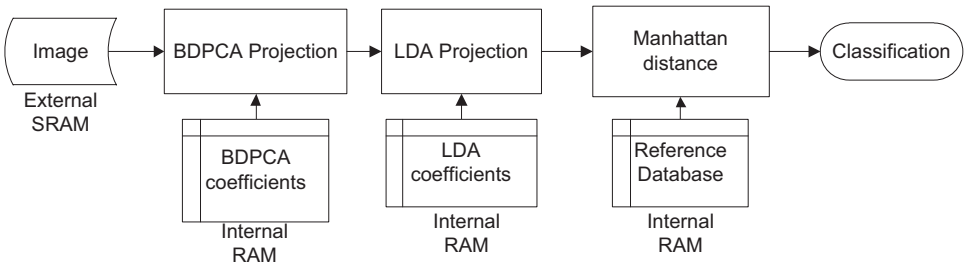
Capítulo 5

Arquitectura e implementación

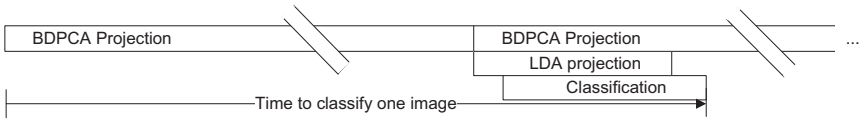
El método BDPCA+LDA se puede separar en tres partes capaces de operar en forma paralela. La figura 5.1(a) muestra un esquema de la operación, desde que se inicia, con la imagen en la memoria externa, hasta que finaliza con la clasificación. En la imagen 5.1(b) se puede apreciar la secuencia de operación de los diferentes módulos. Destaca la capacidad de realizar la proyección LDA y una segunda proyección BDPCA en forma simultánea mejorando el rendimiento al clasificar varias imágenes. El diseño del proyector LDA permite, además, una operación en paralelo con el clasificador, calculando la distancia manhattan a medida que se dispone de los elementos del resultado, sin esperar la proyección LDA completa.

5.1. Módulo BDPCA

La proyección BDPCA se implementa como la multiplicación de tres matrices, $\mathbf{W_C}$, $\mathbf{W_R}$ y la imagen. De estas tres, la imagen está en una memoria externa la cual puede entregar un dato de 16 bits cada $80ns$. Este dato es en realidad dos pixeles de 8 bits cada uno. Con este antecedente se puede establecer una secuencia para la multiplicación (5.1) que aproveche al máximo el paralelismo disponible en el hardware, sin desaprovechar el tiempo de espera entre cada lectura.



(a) Esquema implementación BDPCA+LDA



(b) Diagrama de tiempos BDPCA+LDA

Fig. 5.1: Algoritmo BDPCA+LDA

$$\mathbf{W}_{\mathrm{C}}^{\mathrm{T}} \times \mathbf{X} \times \mathbf{W}_{\mathrm{R}} = \begin{pmatrix} a_1 & a_2 & a_3 & \dots \\ b_1 & b_2 & b_3 & \dots \\ c_1 & c_2 & c_3 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \times \begin{pmatrix} X_1 & \dots \\ X_2 & \dots \\ X_3 & \dots \\ \vdots & \ddots \end{pmatrix} \times \begin{pmatrix} W_1 & W_2 & W_3 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \tag{5.1}$$

La multiplicación se realiza repitiendo dos pasos, primero dos pixeles de $\mathbf{X}$ se multiplican con dos columnas de la matriz $\mathbf{W_C}$ y se almacenan en registros; a esta multiplicación llamaremos $\mathbf{L}$ (ecuación (5.2)). Para que esta multiplicación se realice en un solo ciclo, la matriz $\mathbf{W_C}$ se almacena en cuatro memorias distintas; una por cada columna.

$$\begin{pmatrix} \boxed{a_1} & \boxed{a_2} & a_3 & \dots \\ \boxed{b_1} & \boxed{b_2} & b_3 & \dots \\ \boxed{c_1} & \boxed{c_2} & c_3 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \times \begin{pmatrix} \boxed{X_1} & \dots \\ \boxed{X_2} & \dots \\ X_3 & \dots \\ \vdots & \ddots \end{pmatrix} = \begin{pmatrix} a_1 \times X_1 + a_2 \times X_2 \\ b_1 \times X_1 + b_2 \times X_2 \\ c_1 \times X_1 + c_2 \times X_2 \\ \vdots \end{pmatrix} = \begin{pmatrix} L_1 \\ L_2 \\ L_3 \\ \vdots \end{pmatrix} \tag{5.2}$$

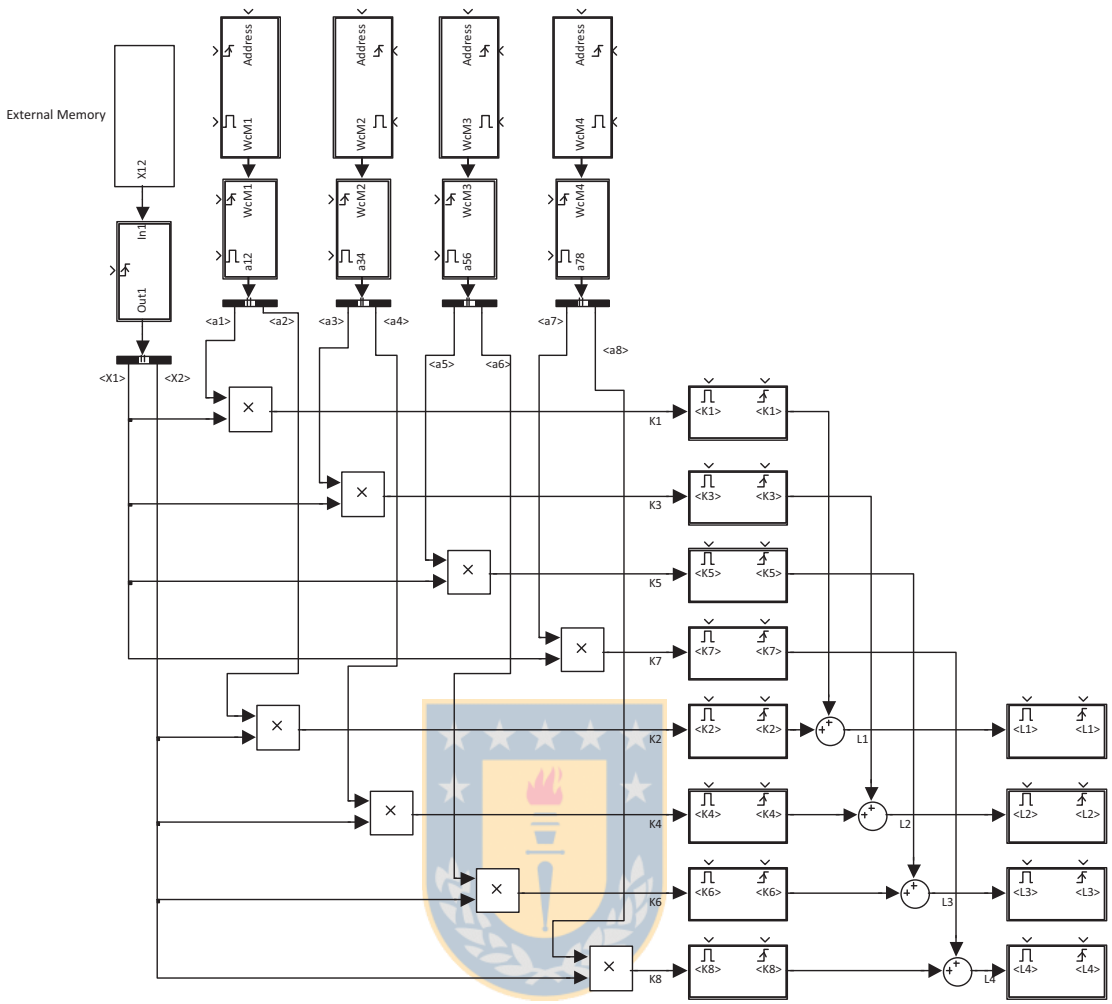


Fig. 5.2: Diagrama implementación primera parte BDPCA

La figura 5.2 ejemplifica el diseño de la primera multiplicación. Se pueden observar las cuatro memorias donde se almacenan los coeficientes de $\mathbf{W_C}$; recordar que se almacenan como valores de utilizando 16 bits, ocho bits para cada elemento. El resultado se almacena en registros de 17 bits, desde donde se rescatan para la segunda multiplicación.

El segundo paso consiste en multiplicar el resultado por una fila completa de $\mathbf{W_R}$, aprovechando que entre cada lectura de $\mathbf{X}$ se requieren $80ns$, esta multiplicación se realiza en tres iteraciones, cada una con un par de elementos de la fila de $\mathbf{W_R}$ (ecuación (5.3). La matriz de coeficientes $\mathbf{W_R}$ se almacena por filas en una sola memoria desde donde se leen de a dos valores de 8 bits en cada lectura de 16 bits.

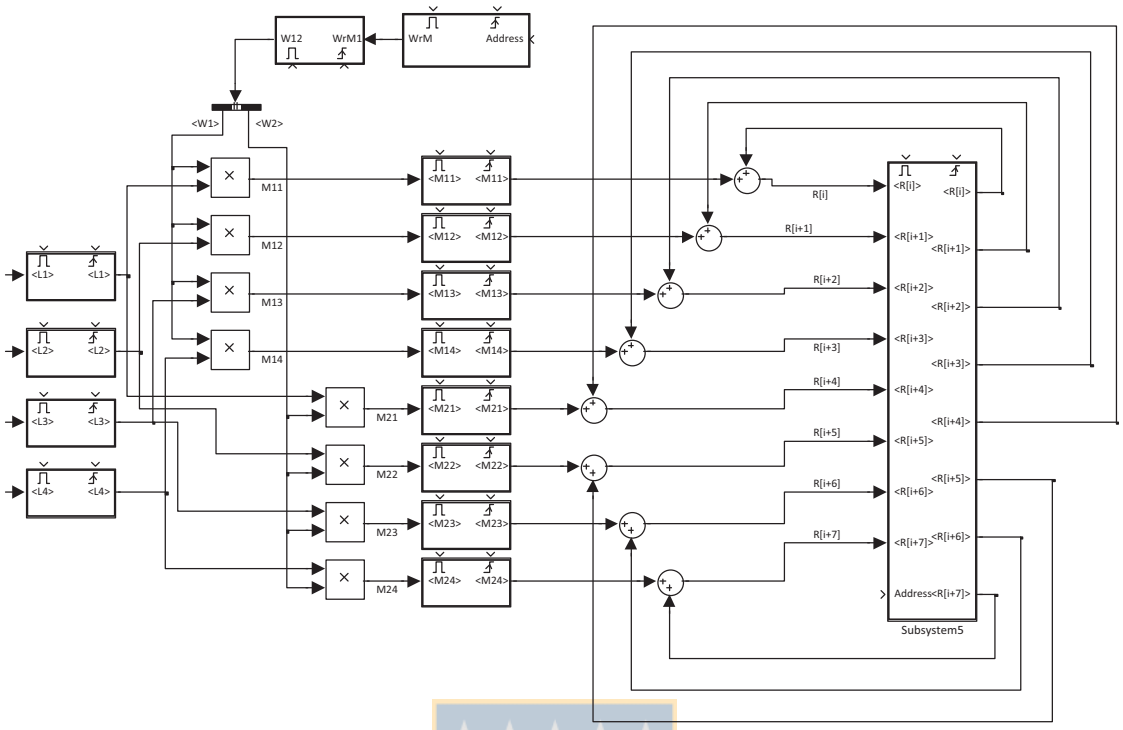


Fig. 5.3: Diagrama implementación segunda parte BDPCA

$$\begin{pmatrix} L_1 \\ L_2 \\ L_3 \\ \vdots \end{pmatrix} \times \begin{pmatrix} W_1 & W_2 & W_3 & W_4 \dots \end{pmatrix} = \begin{pmatrix} L_1 \times W_1 & L_1 \times W_2 & \dots \\ L_2 \times W_1 & L_2 \times W_2 & \dots \\ L_3 \times W_1 & L_3 \times W_2 & \dots \\ \vdots & \vdots & \vdots \quad \ddots \end{pmatrix} \tag{5.3}$$

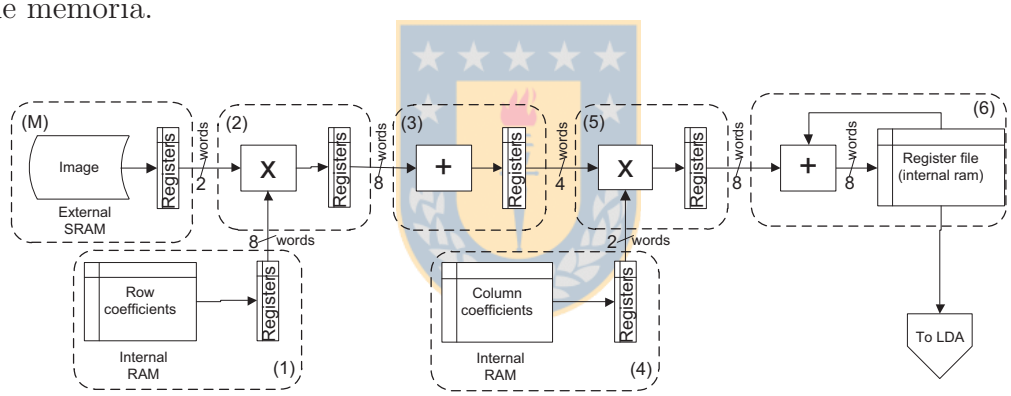
Los resultados de estas multiplicaciones se van acumulando en una memoria interna, así al terminar de procesar la imagen, se tienen disponibles los 24 elementos de la proyección BDPCA. La figura 5.3 presenta una esquema del diseño, donde resulta clara la multiplicación y acumulación; vale destacar que no se presenta el diseño del control.

En la figura 5.4(a) se puede apreciar el diagrama en bloques del módulo BDPCA. En ella se pueden distinguir los distintos pasos de la multiplicación. El bloque (M), representa la lectura de la memoria externa, que se multiplica con los coeficientes de $\mathbf{W_C}$ en el bloque (2). En el bloque (3) se finaliza la multiplicación (5.2). Los bloques (1) a (3) se pueden ejecutar en forma simultánea cada vez que se realiza una lectura de la memoria externa y corresponden a la figura 5.2.

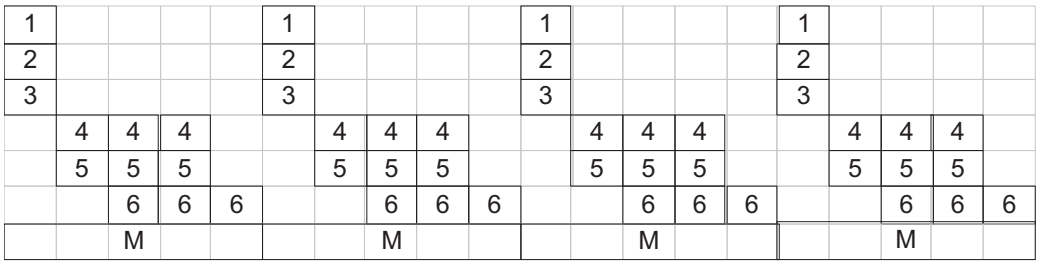
Los bloques (4) y (5) realizan la multiplicación (5.3) y el bloque (6), finalmente, va acumulando los resultados. Para multiplicar la fila completa se repite la operación tres veces, considerando la acumulación se requieren, entonces, cuatro ciclos entre cada lectura. En caso de no estar lista la lectura (por ejemplo, utilizando frecuencias mayores a 50MHz), el módulo espera a que la lectura se realice; estos tres bloques son representados en la figura 5.3.

Recordando de la sección 4.4, $\mathbf{W_C}$, $\mathbf{W_R}$ y $\mathbf{X}$ están en 8 bits por elemento, esto significa que al salir del bloque (2) se utilizan 16 bits, al realizar la suma de (3) el resultado se expresa en 17 bits, luego al multiplicar nuevamente en (5) el total se representa en 25 bits, en la acumulación final se debe agregar un bit por cada suma realizada. Sin embargo, se encuentra que con 32 bits es suficiente para representar los valores resultantes.

La figura 5.4(b) muestra un desglose de la secuencia de operación de los bloques durante el funcionamiento del módulo. Debido a dependencia de datos ((2) depende de (1) y de (M), (5) depende de (4) y de (3) y (6) depende de (5)) se genera una burbuja importante al esperar la lectura de memoria.



(a) Diagrama módulo BDPCA



(b) Secuencia módulo BDPCA

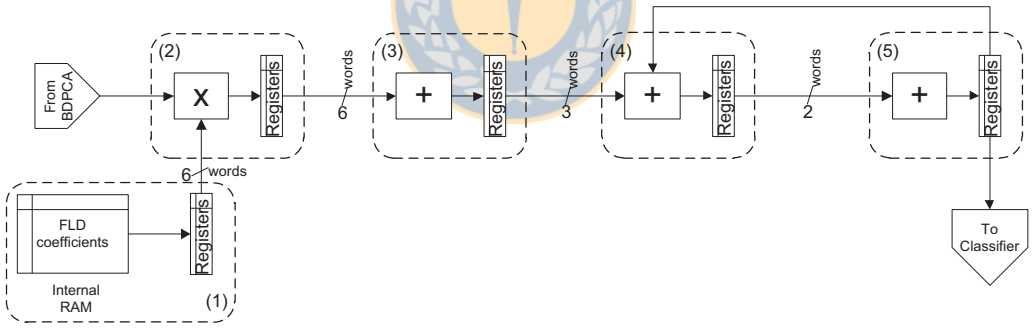
Fig. 5.4: Módulo BDPCA

5.2. Módulo LDA

En cuanto el módulo BDPCA anuncia el fin de la proyección, comienza la proyección LDA tomando una copia, escalada a 16 bits, de la proyección. Con esta entrada; un vector de 24 elementos, se procede a realizar la multiplicación (5.4).

$$\mathbf{W}_{\mathrm{LDA}}^{\mathrm{T}} \times \mathbf{Y} = \begin{pmatrix} F_1 & F_2 & F_3 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \times \begin{pmatrix} Y_1 \\ Y_2 \\ Y_3 \\ \vdots \end{pmatrix} = \begin{pmatrix} F_1 \times Y_1 + F_2 \times Y_2 + \dots \\ \vdots \end{pmatrix} \tag{5.4}$$

Para aprovechar que la entrada está almacenada en memoria distribuida, la multiplicación se realiza utilizando una fila completa de $\mathbf{W_{LDA}}$, multiplicando por todo el vector. Esto presenta una importante ventaja en el orden de los resultados, pues en lugar de obtener todo el vector en forma simultanea (como ocurre con el módulo BDPCA), se obtiene el resultado elemento a elemento. Esto permite al módulo clasificador operar en paralelo aprovechando el tiempo entre cada resultado.



(a) Diagrama módulo LDA

1		1		1		1		1		1		1		1		1		1	
	2		2		2		2		2		2		2		2		2		2
3		3		3		3		3		3		3		3		3		3	
	4		4		4		4		4		4		4		4		4		4
5		5		5		5		5		5		5		5		5		5	
R								R								R			

(b) Secuencia módulo LDA

Fig. 5.5: Módulo LDA

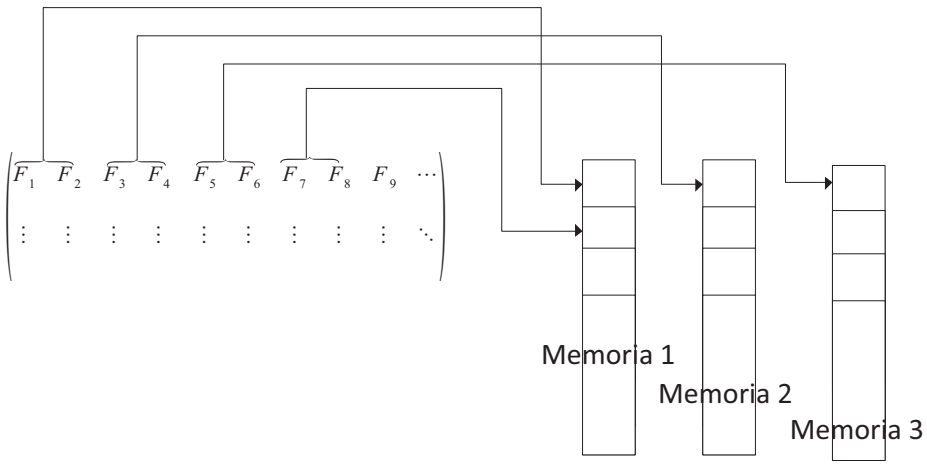


Fig. 5.6: Diagrama almacenamiento matriz $\mathbf{W_{LDA}}$

Con la cantidad limitada de block RAMs en mente, se decide distribuir en 3 memorias la matriz de coeficientes. La matriz se almacena por filas distribuidas en las tres memorias, proveyendo 6 elementos de la fila en cada lectura, completando una fila en 4 lecturas. Esto se ejemplifica en la figura 5.6.

La figura 5.5 muestra el diagrama en bloques y la secuencia de operación en ejecución. Los bloques (1) y (2) realizan la multiplicación, mientras que los bloques (3) a (5) suman los resultados; si en lugar de utilizar un pipeline las sumas se realizan en un sólo ciclo, se genera un camino crítico que limita importantemente la frecuencia de operación. La dependencia entre el bloque (4) y (5) fuerza la inclusión de la burbuja observable en 5.5(b) elevando a 8 ciclos el tiempo entre resultados (R). En la figura 5.7 se exhibe el diseño de la implementación, donde destaca las sumas sucesivas mencionadas

Cabe recordar que $\mathbf{W_{LDA}}$ está almacenada en 16 bits, por lo que el resultado de las multiplicaciones está en 32 bits. Se encuentra que estos 32 bits son suficientes para realizar las sumas (3), (4) y (5) sin generar overflow.

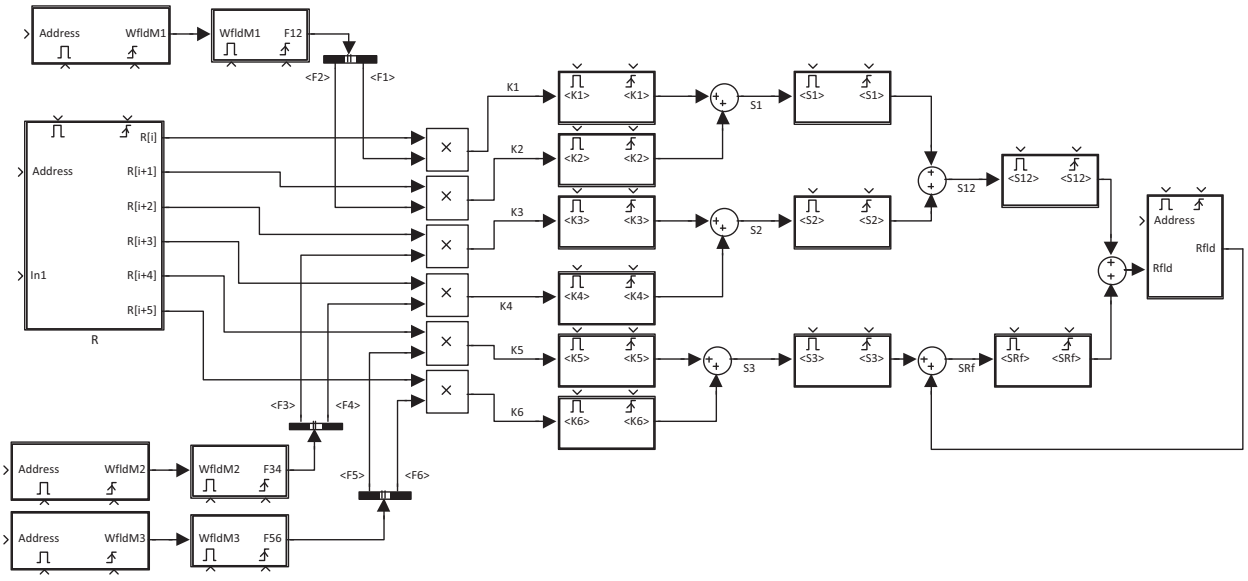


Fig. 5.7: Diagrama implementación proyección LDA

5.3. Módulo clasificador

Como se mencionó en la sección anterior, el módulo clasificador comienza a trabajar en cuanto el módulo LDA entrega un resultado. Esta operación en paralelo permite aprovechar los 8 ciclos entre resultados para realizar el cálculo de la distancia.

Como se vió en la sección 2.2.2, la distancia Manhattan se calcula como la suma de los valores absolutos de las diferencias entre cada elemento de la proyección con cada representante de clase. Por esto se distribuye en cuatro memorias la base de datos de referencia, así en cada lectura se pueden realizar cuatro cálculos, la figura 5.8 muestra la implentación del cálculo de la distancia manhattan, el bloque de memorias representa el arreglo de 16 espacios, de 32 bits cada uno, donde se almacenan los resultados. En cada iteración se calcula la diferencia en un eje, de cuatro clases simultáneamente.

En la figura 5.9 se esquematiza la operación del módulo. Los bloques (1) y (2) calculan la diferencia, luego el bloque (3) acumula los valores absolutos. Estos tres bloques operan en paralelo con el módulo LDA; debido a que los cálculos demoran menos que la proyección, el clasificador debe esperar al nuevo dato (bloque(W)). Una vez terminada la proyección LDA, se calcula la última distancia y se busca la mínima (bloque (4)). La búsqueda se hace comparando por pares y eliminando las distancias mayores.

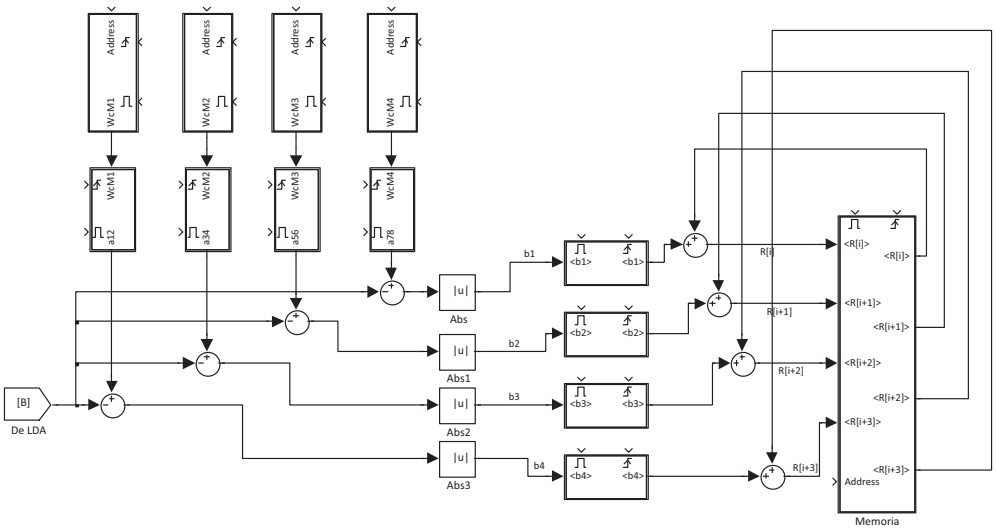
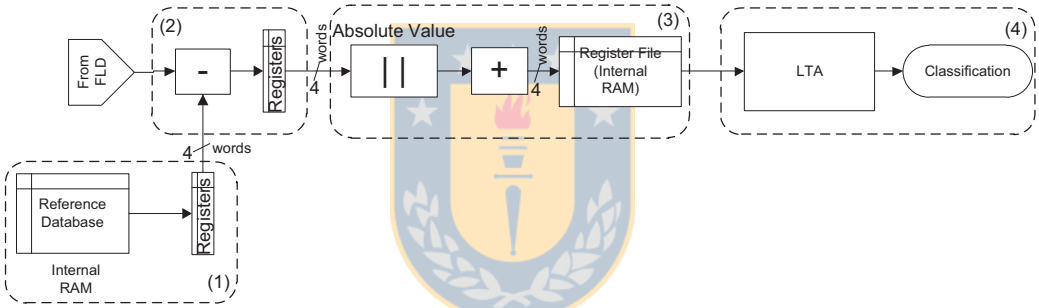
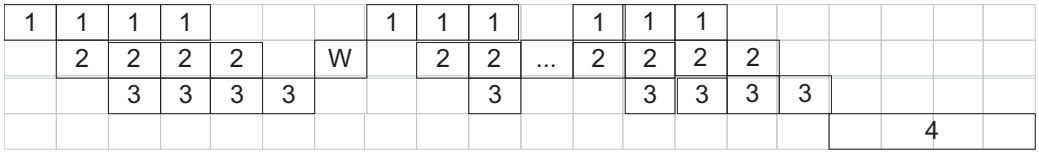


Fig. 5.8: Diagrama implementación cálculo distancia manhattan



(a) Diagrama módulo clasificador



(b) Secuencia módulo clasificador

Fig. 5.9: Módulo clasificador

Para realizar la medición manhattan, las bases de datos se almacenan en forma intercalada en cuatro memorias. Así, en cuatro ciclos se obtienen 16 datos; uno por clase más un dato en blanco.

5.4. Estimación de la duración de los módulos

La duración de cada módulo se puede definir en base a la cantidad de ciclos necesarios para la operación. En el caso de los módulos LDA y clasificador, la duración está definida exclusivamente por el diseño. Basándose en la figura 5.5(b), se puede estimar la duración del módulo LDA en 118 ciclos (8 por elemento de salida, más 6 para llegar a operación estable).

La duración del módulo de clasificación es de 4 ciclos por dimensión del vector de entrada, mas, dado que opera en forma paralela al módulo LDA, el largo efectivo es el tiempo en calcular la diferencia del último elemento, más el largo en buscar el mínimo, en total son 10 ciclos.

Finalmente el módulo más complejo de calcular la duración, es el módulo BDPCA donde la duración, en ciclos, es dependiente de la frecuencia de operación. El tiempo de acceso a la memoria externa, es siempre de $80ns$. Por lo cual para conocer la cantidad de ciclos, se debe multiplicar (y redondear hacia el ciclo entero más alto) por la frecuencia de operación.

La duración del sistema completo para diferentes frecuencias de operación está presentado en 5.10, se observan claramente mínimos locales en ciertas frecuencias, este comportamiento se debe al tiempo de acceso a la memoria externa. Al momento de seleccionar una frecuencia de operación se debe, idealmente, buscar la más cercana al mínimo local, para maximizar la velocidad de operación.

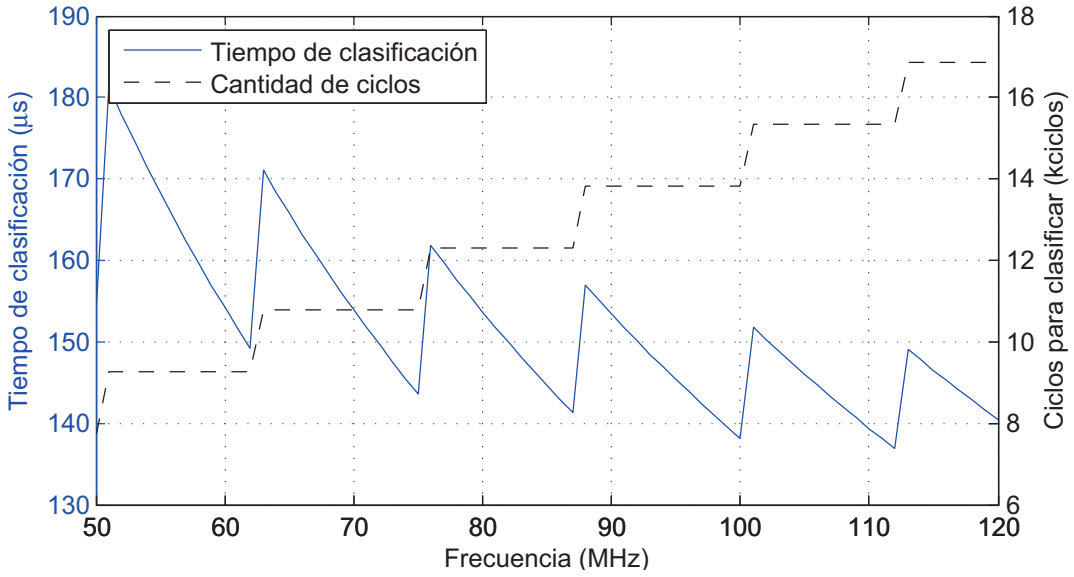


Fig. 5.10: Duración de clasificación para diferentes frecuencias

Capítulo 6

Resultados

Se sintetizó la implementación en Verilog HDL utilizando el compilador de hardware *Synopsys Synplify* y el mapeo a la FPGA Xillinx Spartan 3 se realizó con la herramienta de *place-and-route* de Xillinx.

6.1. Método de pruebas

Para la evaluación de la implementación es importante considerar tres aspectos: la tasa de aciertos, la velocidad de clasificación y la potencia requerida.

Se busca un equilibrio de estas tres variables, pues si una de ellas resulta con un valor muy bajo (o elevado en el caso de la potencia), inmediatamente se debe excluir de ciertas aplicaciones. Un consumo de energía excesivo, por ejemplo, impide su aplicación en sistemas portátiles.

Para evaluar la tasa de aciertos se repite el experimento de la sección 4.1. Esto es, utilizando la base *Yale*, realizar tres clasificaciones de la base completa, variando el set de entrenamiento y de testeo en cada ocasión. Para una comparación adecuada, se utiliza la misma selección de imágenes para entrenar y testear que se utilizó al clasificar en la implementación en punto flotante; es de interés, además, conocer si el sistema falla en las mismas imágenes.

El segundo aspecto, la velocidad de clasificación, se estima tomando como base la frecuencia máxima calculada por *Synopsys Synplify* y la duración, en ciclos, de cada módulo (el largo de cada módulo se estima en la sección 5.4). Es de interés conocer la velocidad de clasificación de

Tabla 6.1: Resultados de implementación a 75[MHz]

Tasa de acierto			93.3 %
Velocidad estimada	Limitada por RAM	1 imagen	143.7 μs
		flujo	7,045 imágenes/segundo
	No limitada por RAM	1 imagen	62.5 μs
		flujo	16,447 imágenes/segundo
Consumo estimado		En reposo	99.41 mW
		Dinámico	57.83 mW
		Total	157.24 mW

una imagen y el flujo (en imágenes por segundo) de clasificación en operación estable; se calcula, además, la velocidad de operación en el caso que la memoria no fuera limitante.

Finalmente la potencia requerida se estima utilizando la herramienta *Xillinx Xpower*, que realiza una aproximación basada en la síntesis de hardware y valores predeterminados sobre el consumo para el chip en la frecuencia de operación. Vale destacar que la estimación es sólo del consumo de la FPGA, sin incluir consumos fuera del chip, como puede ser: memoria ram, generador de reloj o LEDs.

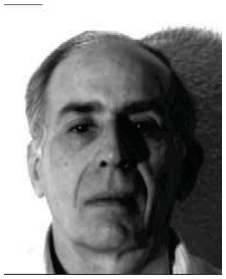
6.2. Resultados

Como se mencionó. a partir de la síntesis se puede estimar una frecuencia máxima de operación; para el presente trabajo la herramienta *Synopsys Synplify* calculó una frecuencia máxima de 75,9MHz, mas se utiliza 75MHz para operar en el mínimo local cercano (figura 5.10).

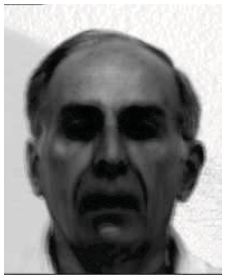
A partir de la frecuencia se puede obtener una aproximación al consumo de la implementación y a la velocidad de clasificación. En la tabla 6.1 se presenta un resumen de los resultados. Para la estimación de las velocidades sin limitación por la RAM, se reorganiza el módulo BDPCA logrando un resultado cada 3 ciclos, más 6 de latencia.

Tabla 6.2: Errores en clasificación en PC y FPGA

Imagen	17	19	**29**	**36**	38	42	46	**61**	79	85
Clase	3	4	**5**	**6**	7	7	8	**11**	14	15
Clasificación PC	2	15	**12**	**8**	2	3	15	**11**	2	4
Clasificación FPGA	13	9	**5**	**6**	13	5	5	**9**	9	5



(a) Imagen 29



(b) Representante clase 5: Clasificación correcta

(c) Representante clase 12: Clasificación PC

Fig. 6.1: Imagen 29

La efectividad de la implementación concuerda con la simulación realizada en 4.4. Para analizar con más detalle esta variación, se tabulan los errores de la primera iteración en 6.2; se revisa sólo la primera iteración pues es la única con tasas de reconocimiento diferentes en PC y FPGA.

De la tabla 6.2 se destacan las imágenes 29, 36 y 61. La clasificación de estas imágenes se realiza en forma correcta por sólo uno de los métodos, por lo que resulta interesante observar dichas imágenes.

Las figuras 6.1 presenta la imagen 29 y el representante de clase (el promedio del conjunto) para la clasificación correcta y para la categoría que entregó, en forma errada, la implementación

(a) Imagen 36

(b) Representante clase 6: Clasificación correcta

(c) Representante clase 8: Clasificación PC

Fig. 6.2: Imagen 36

(a) Imagen 61

(b) Representante clase 11: Clasificación correcta

(c) Representante clase 9: Clasificación FPGA

Fig. 6.3: Imagen 61

en punto flotante. La figura 6.2, repite lo mismo, pero para la imagen 36. En la figura 6.3, la imagen 61 es clasificada correctamente por el PC, mas no por la FPGA.

Observando los casos donde el PC no logra clasificar correctamente y la FPGA sí, se puede extrae que la reducción de precisión, al usar punto fijo en las matrices de proyección, mejora el discernimiento entre dos clases parecidas. Ésto pues el PC decide por una imagen con cierta similitud (pelo y forma de la cara por ejemplo) a la clasificación correcta, mientras la FPGA clasifica en forma correcta. Esto sugiere, por lo tanto, que dichas clases están cercanas en el espacio de BDPCA+LDA.

No siempre la reducción de precisión significó mejora en la capacidad de clasificación, la imagen 61 (figura 6.3(a)) es asignada a una clase notoriamente diferente (figura 6.3(c)). Ésto sugiere que el resultado de la proyección diverge fuertemente en relación a la implementación en punto flotante.

El segundo aspecto evaluado es la velocidad de clasificación. Comparando con la implementación en PC (tabla 4.3), la FPGA logra cerca del doble de la velocidad considerando los casos más lentos de ambas plataformas (254,1μs en pc y 143,7μs en FPGA). Inversamente si se consideran los casos más rápidos, la implementación en PC es poco más de un tercio más veloz que la versión en FPGA (44,4μs y 62,5μs respectivamente). Este resultado reitera la importancia del acceso expedito a la imagen y confirma la superioridad de una plataforma de alto rendimiento por sobre una FPGA en términos de cálculos matriciales.

Tabla 6.3: Distribución consumo de energía (Dinámico)

Elemento	consumo
Reloj	17.9 mW
Lógica	8.95 mW
Señales	29.2 mW
IO's	1.55 mW
Multiplicadores	0.15 mW

Es el tercer factor de la evaluación que definitivamente coloca la implementación en FPGA por sobre la implementación en PC. El consumo de energía estimado en la FPGA (157,24mW) es, por lo menos, un orden de magnitud menor que el consumo de un PC (17,2W en tabla 4.3). Se debe destacar que la estimación no considera la memoria externa, ni el resto de los circuitos de la FPGA. En la tabla 6.3 se desglosa el consumo de energía. Se observa que la mayor cantidad de energía se disipa en señales y se destaca la eficiencia de los multiplicadores.

La evaluación de la implementación en hardware, también debe considerar el nivel de uso de recursos en el chip. En caso de ser bajo, significa que existen recursos disponibles para otras tareas o que se puede utilizar un chip más simple. En la tabla 6.4 se presenta un resumen de la capacidad utilizada.

Se pueden extraer dos detalles importantes al observar la utilización de elementos. Primero, la gran limitante son los multiplicadores, pues se utiliza más del 90 % de ellos. Segundo, se utilizan menos block-RAMs de las esperadas por el diseño del módulo BDPCA (se esperaba usar 5), esto significa que los datos fueron almacenados en forma distruibuida en la FPGA; esto plantea nuevas posibilidades para optimizar la multiplicación.

Tabla 6.4: Utilización de recursos

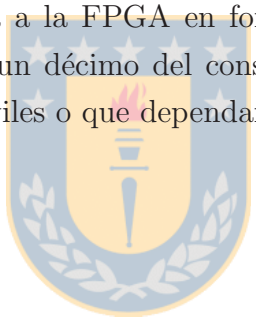
Recurso	BDPCA		LDA		Classifier		Total	
	Número	%	Número	%	Número	%	Número	%
Slices	278	3.62	1353	17.62	1722	22.42	3353	43.66
FF	394	2.57	1115	7.26	1113	7.25	2622	17.07
LUT	312	2.03	2253	14.67	3243	21.11	5808	37.81
BRAM	2	8.33	3	12.50	4	16.67	9	37.50
MULT	16	66.67	6	25	0	0	22	91.67
IOB	–	–	–	–	–	–	62	35.84

6.3. Conclusiones

Se verifica de los resultados que método implementado resiste muy bien el traspaso de punto flotante a punto fijo, logrando una tasa de acierto casi idéntica en ambos casos. Al analizar las diferencias, se encuentra que al reducir la precisión de las matrices de proyección, la capacidad de diferenciar entre clases mejora levemente. Sin embargo, no asegura un mejor resultado pues la proyección puede divergir notoriamente de la clasificación apropiada.

La buena eficacia de la implementación se ve complementada con una buena velocidad de operación, superando a la versión en lenguaje *C* operando sobre un PC. Sin embargo, se debe destacar que el PC logra velocidades mayores al considerar que la imagen se lee directamente desde la memoria RAM, en lugar de acceder al disco para cada lectura. Este resultado no es del todo inesperado pues el PC opera con un reloj casi 30 veces superior y optimiza las operaciones para maximizar la velocidad.

Finalmente el aspecto que coloca a la FPGA en forma clara por sobre el PC, es el bajo consumo de energía. Con menos de un décimo del consumo, la implementación en FPGA es más apropiada para aplicaciones móviles o que dependan de una batería para operar.



Capítulo 7

Conclusiones y trabajo a futuro

En este trabajo se presentó y evaluó una implementación en una FPGA Xillinx Spartan 3 del algoritmo BDPCA+LDA en conjunto con clasificación con distancia Mahattan para el reconocimiento de rostros. La implementación en hardware se realiza tomando en cuenta las limitaciones del chip utilizado, tratando de mantener independencia con respecto a los elementos externos a la plataforma.

7.1. Conclusiones

La implementación en hardware dedicado de un sistema de reconocimiento de rostros tiene varias ventajas. Ésto en conjunto con el desarrollo de nuevos métodos, motivó el diseño e implementación de una sistema de reconocimiento de rostros en una FPGA.

La clasificación se separó en dos etapas. Primero una reducción de dimensionalidad; para eliminar información redundante o muy detallada y reducir el esparcimiento de las imágenes. Segundo, un proceso de clasificación como tal, donde se utilizaron y compararon dos distancias; Manhattan y euclidiana. Para evaluar los métodos se usaron dos bases de datos para verificar el comportamiento de los algoritmos en condiciones distintas.

La evaluación de un algoritmo no puede estar ajeno al hardware para el cual se diseñará. Se eligió una FPGA Xilinx Spartan 3, embebida en una placa de desarrollo Digilent Nexys. La elección de este hardware fue en base a su bajo costo y a que se adecua a la tarea en cuestión.

Para la verificación en PC se implementaron los métodos en Matlab y se evaluaron según tasa de reconocimiento y requerimientos de recursos, específicamente memoria, multiplicadores y sumadores. Tras obtener los resultados se optó por implementar el método BDPCA+LDA y utilizar distancia Manhattan para la clasificación, pues obtuvo la mejor combinación entre tasa de acierto y consumo de recursos.

Una vez decidido el método, se diseñó la arquitectura para aprovechar al máximo los recursos de la FPGA. Cada módulo se planteó en forma que los cálculos se realizarán lo más rápido posible; aprovechando la capacidad de operar en paralelo propia de una implementación en hardware. Debido a que el acceso a la memoria externa es importante, se aprovechó dicho espacio para ocultar ocultar una gran cantidad de los cálculos del módulo BDPCA. Adicionalmente los módulos *LDA* y clasificador fueron diseñados para operar en paralelo entre sí y con la etapa BDPCA de una nueva imagen, reduciendo el tiempo de clasificación en operación continua.

Finalmente se sintetizó la implementación en Verilog HDL, utilizando el compliador de hardware Synopsys Synplify y se mapeó a la FPGA Xillinx Spartan 3 con la herramienta de place-and-route de Xilinx. Con ayuda de estas herramientas se estimó una frecuencia máxima de operación de $75.9MHz$, aunque se optó por una frecuencia de $75.0MHz$ para minimizar el tiempo de clasificación sincronizando el tiempo de acceso a memoria con la cantidad de ciclos requerida.

Una vez realizada las pruebas se encontró que la implementación en FPGA supera ampliamente a un PC. Si bien el resultado es prácticamente el mismo (93.3 % y 93.0 % respectivamente), el consumo es un orden de magnitud menor y la velocidad cercana al doble en el caso más lento del PC o cercana a dos tercios sí se consideran los casos más rápidos (para la FPGA, sin considerar las restricciones de memoria y para el PC, considerando las imágenes en memoria RAM)

Finalmente se confirma la efectividad del método en una implementación en hardware, a pesar de las limitaciones propias del hardware utilizado. Es importante destacar que, sin considerar los multiplicadores, gran parte de los recursos de la FPGA no se utilizaron, por lo que queda abierta la posibilidad de utilizar mencionados recursos para implementar módulos anexos, ya sea para un preprocesamiento de la imagen o un sistema de clasificación alternativo.

7.2. Trabajo a futuro

El siguiente paso es experimentar con sistemas más complejos de clasificación, principalmente porque analizando los errores de clasificación se encuentra que considerando más de una imagen por clase (por ejemplo, realizando tres reconocimientos independientes y clasificar según mayoría) se pueden alcanzar tasas del 100 %. Esto lleva a cuestionar la desición de utilizar un representante por clase, pues utilizando varias imágenes de cada persona y luego clasificando según mayoria se debería lograr resultados mucho mejores.

Un segundo aspecto donde se puede continuar, es la optimización del módulo BDPCA para aprovechar el hecho de que almacena varios de los datos en memoria distribuida, lo cual permitiría acceder a varios en forma simultánea y más rápido.

Otro punto con potencial para optimizar, es el consumo de energía. Reestructurando el diseño debería ser posible reducir la cantidad de señales, en especial tomando en consideración una mejora en el almacenamiento de las matrices de proyección.

Durante el desarrollo del proyecto, se encontró que para una misma cantidad de elementos finales, el rendimiento de BDPCA y BDPCA+LDA depende en gran medida de la razón entre las aristas de la imagen reducida. Se deja abierta la posibilidad de encontrar e implementar una mejor razón que la utilizada en este trabajo.

Finalmente queda el espacio para diseñar e implementar un sistema para la adquisición y preprocesamiento de la imagen, con lo cual el sistema podría operar en forma completamente autónoma.

Bibliografía

[1] A. F. Abate, M. Nappi, D. Riccio, and G. Sabatino. 2d and 3d face recognition: A survey. *Pattern Recogn. Lett.*, 28:1885–1906, October 2007.

[2] M. S. Bartlett, J. R. Movellan, and T. J. Sejnowski. Face recognition by independent component analysis. *IEEE Transactions on Neural Networks*, pages 1450–1464, 2002.

[3] P. Belhumeur, J. Hespanha, and D. Kriegman. Eigenfaces vs. fisherfaces: recognition using class specific linear projection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):711 –720, jul 1997.

[4] V. Blanz and T. Vetter. Face recognition based on fitting a 3d morphable model. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25:1063–1074, September 2003.

[5] B. Boom, G. Beumer, L. Spreeuwers, and R. Veldhuis. The effect of image resolution on the performance of a face recognition system. In *Control, Automation, Robotics and Vision, 2006. ICARCV '06. 9th International Conference on*, pages 1 –6, dec. 2006.

[6] J.-T. Chien and C.-C. Wu. Discriminant waveletfaces and nearest feature classifiers for face recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(12):1644 – 1649, dec 2002.

[7] K. Delac, M. Grgic, and S. Grgic. Statistics in face recognition: analyzing probability distributions of pca, ica and lda performance results. In *Image and Signal Processing and Analysis, 2005. ISPA 2005. Procs. of the 4th International Symp. on*, pages 289–294, 2005.

[8] K. Delac, M. Grgic, and P. Liatsis. Appearance-based statistical methods for face recognition. In *ELMAR, 2005. 47th International Symp.*, pages 151–158, 2005.

[9] Digilent. *Digilent Nexys Board Reference Manual*, 2007.

[10] M. Galassi et al. *GNU Scientific Library Reference Manual (3rd Ed.)*. Network Theory Ltd, 1999.

[11] M. Goraczko, A. Kansal, J. Liu, and F. Zhao. Joulemeter: Computational energy measurement and optimization. `http://research.microsoft.com/en-us/projects/joulemeter/`, sep 2011.

[12] Y. Hu, D. Jiang, S. Yan, L. Zhang, and H. zhang. Automatic 3d reconstruction for face recognition. In *Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on*, pages 843 – 848, may 2004.

[13] W. Li-Wei, W. Xiao, C. Ming, and F. Ju-Fu. Is two-dimensional pca a new technique? *Acta Automatica Sinica*, 31(5):782–787, 2005.

[14] Q. Liu, R. Huang, H. Lu, and S. Ma. Face recognition using kernel-based fisher discriminant analysis. In *Automatic Face and Gesture Recognition, 2002. Proceedings. Fifth IEEE International Conference on*, pages 197 –201, may 2002.

[15] Micron. *Async/Page/Burst CellularRAM^{TM} 1.5*, 2004.

[16] A. Mohan, N. Sudha, and P. Meher. An embedded face recognition system on a vlsi array architecture and its fpga implementation. In *Industrial Electronics, 2008. IECON 2008. 34th Annual Conference of IEEE*, pages 2432 –2437, nov. 2008.

[17] S. Nagamalla and B. Dhara. A novel face recognition method using facial landmarks. In *Advances in Pattern Recognition, 2009. ICAPR '09. Seventh International Conference on*, pages 445 –448, feb. 2009.

[18] S. Nazeer, N. Omar, and M. Khalid. Face recognition system using artificial neural networks approach. In *Signal Processing, Communications and Networking, 2007. ICSCN '07. International Conference on*, pages 420 –425, feb. 2007.

[19] F. Samaria and A. Harter. Parameterisation of a stochastic model for human face identification. In *Applications of Computer Vision, 1994., Proceedings of the Second IEEE Workshop on*, pages 138 –142, dec 1994.

[20] A. Scheenstra, A. Ruifrok, and R. Veltkamp. A survey of 3d face recognition methods. In T. Kanade, A. Jain, and N. Ratha, editors, *Audio- and Video-Based Biometric Person Authentication*, volume 3546 of *Lecture Notes in Computer Science*, pages 325–345. Springer Berlin / Heidelberg, 2005. 10.1007/11527923_93.

[21] G. Shakhnarovich and B. Moghaddam. Face Recognition in Subspaces. In S. Z. Li and A. K. Jain, editors, *Handbook of Face Recognition*, pages 141–168. Springer, 2004.

[22] P. Sinha, B. Balas, Y. Ostrovsky, and R. Russell. Face recognition by humans: 20 results all computer vision researchers should know about. *Department of Brain and Cognitive Sciences Massachusetts Institute of Technology Cambridge, MA*, 2005.

[23] M. Turk and A. Pentland. Face recognition using eigenfaces. In *Computer Vision and Pattern Recognition, 1991. Procs. CVPR '91., IEEE Computer Society Conference on*, pages 586 –591, June 1991.

[24] K. Vinay and B. Shreyas. Face recognition using gabor wavelets. In *Signals, Systems and Computers, 2006. ACSSC '06. Fortieth Asilomar Conference on*, pages 593 –597, 29 2006-nov. 1 2006.

[25] M. Williams. Better face-recognition software. `http://www.technologyreview.com/computing/18796/`, May 2007.

[26] Xilinx. *Spartan-3 Generation FPGA User Guide*, 2008.

[27] J. Yang, D. Zhang, A. Frangi, and J. yu Yang. Two-dimensional pca: a new approach to appearance-based face representation and recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(1):131 –137, Jan. 2004.

[28] W. Zuo, K. Wang, and H. Zhang. Subspace methods for face recognition: Singularity, regularization, and robustness. In M. Chacon, editor, *State of the Art in Face Recognition*, pages 25–50. In-Teh, 2009.

[29] W. Zuo, D. Zhang, J. Yang, and K. Wang. BDPCA plus LDA: a novel fast feature extraction technique for face recognition. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 36(4):946 –953, Aug. 2006.