Universidad Autónoma Metropolitana–Azcapotzalco

División de Ciencias Básicas e Ingeniería

Licenciatura en Ingeniería en Computación

Compresión de Imágenes JPEG con un FPGA para un Centro de Datos.

Reporte de Proyecto Tecnológico que presenta

Daniel Tapia Rodríguez

Matricula
207205344

Para obtener el título de:

Ingeniero en Computación

Asesor:

M. en C. José Ignacio Vega Luna

Co-asesor:

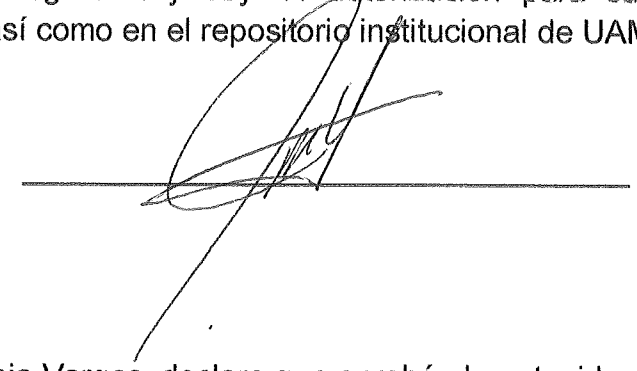
Ing. Víctor Noé Tapia Vargas

Trimestre 2017-Primavera

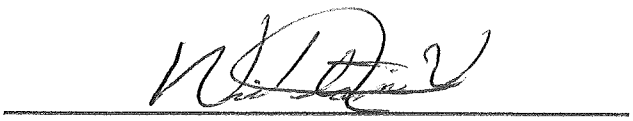
Ciudad de México a 31 de Agosto de 2017

Declaratoria

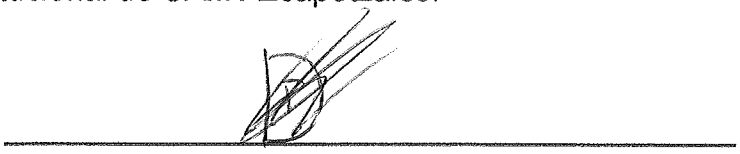
Yo, José Ignacio Vega Luna, declaro que aprobé el contenido del presente reporte de proyecto de integración y doy mi autorización para su publicación en la biblioteca digital, así como en el repositorio institucional de UAM Azcapotzalco.



Yo, Víctor Noé Tapia Vargas, declaro que aprobé el contenido del presente reporte de proyecto de integración y doy mi autorización para su publicación en la biblioteca digital, así como en el repositorio institucional de UAM Azcapotzalco.



Yo, Daniel Tapia Rodríguez, doy mi autorización a la coordinación de servicios de información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el repositorio institucional de UAM Azcapotzalco.



RESUMEN

Se presenta el diseño e implantación de un sistema cuyo objetivo fue comprimir a formato JPEG, usando un FPGA, la imagen del rostro de personas que intentan acceder a un área de un centro de datos y transmitirla a un servidor. La investigación y desarrollo de técnicas de compresión de imágenes se ha incrementado los últimos años por la necesidad de usar eficientemente el espacio de almacenamiento y ancho de banda de las redes de datos. En los centros de datos se usan diferentes procedimientos para registro y control de acceso a las instalaciones. Algunos de estos procedimientos capturan, usando una cámara de video, el rostro de personas que intentan acceder al centro de datos en un archivo con formato TIF y lo transmiten a una aplicación de reconocimiento de imágenes y control de acceso. Comúnmente, estas aplicaciones aceptan también archivos en formato JPEG los cuales procesan y analizan más rápido y eficientemente que los archivos TIF. El sistema desarrollado en este trabajo se basa en una tarjeta Altera DE2 y realiza las siguientes funciones; lee el identificador de la tarjeta RFiD, captura la imagen del rostro de la persona, implanta en un FPGA el algoritmo estándar JPEG para comprimir el archivo de la imagen y transmite el archivo JPEG al servidor. Se realizó una interfaz de usuario, la cual se ejecuta en el servidor, que muestra y almacena en un archivo el nombre de la persona y su imagen. Este archivo es usado por una aplicación, no realizada en este trabajo y con la cual ya dispone el centro de datos, de reconocimiento de imágenes que acepta tasas de compresión de 10% a 20%. La tasa de compresión obtenida fue 15%. Se pudo haber realizado una compresión aritmética para lograr una tasa de compresión menor pero el tiempo de ejecución y recursos necesarios serían mayores a los usados en este trabajo. También, se pudo haber realizado la compresión con un programa ejecutándose en una computadora pero el costo de la solución es mayor al del sistema aquí presentado.

TABLA DE CONTENIDO

INTRODUCCIÒN

Los centros de datos o data center por su equivalente en inglés, ofrecen servicios para concentrar y operar recursos y equipos necesarios para el procesamiento y almacenamiento de información de empresas e instituciones, así como equipos de telecomunicaciones. La confiabilidad, seguridad y disponibilidad de estos servicios garantiza la continuidad del negocio y procesos de clientes, empleados, ciudadanos, proveedores y empresas colaboradoras. Los centros de datos cuentan con mecanismos seguros y eficientes para la protección del equipo y acceso a instalaciones, ya que contienen información crítica y necesaria para las operaciones de empresas. Para acceder a instalaciones de un centro de datos, se usa una variedad de dispositivos y procedimientos de seguridad que incluyen: cerraduras electromagnéticas, torniquetes, videocámaras, detectores de movimiento, tarjetas de identificación, sistemas biométricos, teclados claves de acceso y sistemas de registro y reconocimiento facial. Se usa uno o varios de estos dispositivos dependiendo de los siguientes factores: tipo de área a proteger, cantidad de usuarios que pueden acceder al área, equipo alojado y confiabilidad del mecanismo de seguridad [1].

Recientemente se han empezado a usar sistemas de procesamiento de imágenes de rostros en los procesos de acceso a instalaciones y en otros sectores como: el militar, la salud y registro de población. La industria e instituciones de investigación han invertido mucho tiempo y recursos en este campo para el desarrollo de software y hardware para obtener soluciones eficientes y seguras. Generalmente, un sistema de registro y reconocimiento facial procesa la imagen del rostro humano para generar órdenes que actúan sobre un mecanismo de control [2].

La imagen del rostro contiene gran cantidad de información que consume: espacio significativo en un dispositivo de almacenamiento, ancho de banda al transmitirla por la red de datos y tiempo de procesamiento.

La realización del trabajo aquí presentado surgió a solicitud de los administradores de un centro de datos. Actualmente, algunas áreas del centro de datos cuentan con una cámara de video que puede capturar, en un archivo con formato TIF, el rostro de personas que se identifican con una tarjeta RFiD para poder acceder al área donde está instalada la cámara. El archivo es transmitido por la cámara a un servidor donde se ejecuta una aplicación que registra y procesa la imagen para permitir el acceso a las personas. Los archivos de imágenes TIF consumen mucho espacio en disco y la aplicación acepta que estén en formato JPEG. Se realizaron pruebas de la aplicación usando archivos con

formato JPEG y mostraron que su procesamiento es más rápido que las de formato TIF. La solicitud fue realizar un sistema cuyos objetivos son los siguientes: comprimir el archivo del rostro capturado por la cámara a formato JPEG usando un FPGA, transmitir el archivo comprimido e información de la tarjeta RFiD al servidor y realizar una interfaz de usuario que registre y muestre la imagen y nombre de las personas que han solicitado acceder. El propósito y aplicación de este trabajo es de seguridad en el centro de datos.

Un FPGA (Field Programmable Gate Array) es un dispositivo lógico programable que incorpora gran cantidad de compuertas lógicas y circuitos digitales cuya interconexión y funcionalidad puede configurarse al momento de usarse mediante un lenguaje de descripción de hardware o una herramienta gráfica. Esto los hace flexibles al flujo de diseño y presentan costos de adquisición bajos para pequeñas cantidades de dispositivos, además que el tiempo de desarrollo es menor en comparación con otros dispositivos digitales. El uso de un FPGA presenta tres ventajas: son dispositivos reprogramables, disminuyen el costo y tiempo de desarrollo y proporcionan mayor rendimiento a las aplicaciones. Debido a estas ventajas, así como a la frecuencia alta de trabajo y capacidad de procesamiento en paralelo, los FPGA exceden la potencia de cómputo de los procesadores digitales de señales (DSP). Terasic, tiene disponible la tarjeta Altera DE2 como la utilizada en este trabajo. Altera ofrece el software Quartus II para el diseño y simulación de circuitos en la tarjeta Altera DE2 y soporta el uso de los lenguajes de descripción de hardware VHDL y Verilog. La programación realizada en Quartus II se transfiere desde una computadora a la DE2 para probar inmediatamente el diseño. El procesador Nios II implantado en el FPGA y los diferentes componentes de la tarjeta Altera DE2 permiten crear un sistema digital completo [3].

El algoritmo JPEG fue creado como software libre por el Joint Photographic Experts Group (JPEG) ante la necesidad de transmitir por la Internet archivos de imágenes de manera rápida y confiable. Este algoritmo es fácil de entender y no fácil de implantar. Se diseñó para comprimir archivos de imágenes en escala de grises o imágenes de color con 24 bits. El algoritmo JPEG está basado en dos características del ojo humano. La primera es que el ojo es más sensible al cambio en la luminancia que en la crominancia, debido que capta más claramente los cambios de brillo que los de color. La segunda es que percibe más fácilmente cambios de brillo en áreas homogéneas que en zonas donde la variación es grande [4].

En este trabajo se solicitó que la compresión se realice usando el algoritmo JPEG estándar, para lo cual se usó el FPGA de una tarjeta Altera D2. Las ventajas

y aportaciones del sistema desarrollado son las siguientes: reducción de tamaño en los archivos que almacenan las imágenes de rostros, y por tanto el espacio en disco necesario, y reducción de tiempo de reconocimiento de rostros y apertura de la puerta de acceso en la aplicación usada en el centro de datos. No se ha realizado una solución como la aquí presentada, implantando el algoritmo JPEG con un FPGA e integrando un lector de tarjetas RFiD para un centro de datos.

ANTECEDENTES

Se realizó una búsqueda en los proyectos realizados tanto en la UAM Azcapotzalco como en otras instituciones, para encontrar aquellos que tienen cierta similitud con el proyecto que se está proponiendo. Enseguida se muestra una breve descripción de dichos proyectos.

Referencias Internas

Proyectos de Integración o Terminales

- *Clasificador de objetos en banda infinita por medio de procesamiento digital de imágenes.*

En este proyecto se desarrolló un algoritmo que fuese capaz de identificar objetos iguales e ir separándolos mediante el procesamiento digital de imágenes. La similitud que tiene es que se usa un procesamiento de imágenes obtenidas mediante una cámara. Difiere en que nosotros utilizaremos hardware (FPGA) para realizar la compresión de las imágenes JPEG [5].

- *Aplicación de filtros digitales en la compresión de imágenes*

Este proyecto consistió en aplicar un filtro optimizado de una imagen pequeña en una imagen de mayor tamaño. La similitud que tiene es que se realizó una compresión de imágenes. Difiere en que se utilizó lenguaje de programación JAVA y se usó la transformada de Wavelet Discreta Haar [6].

- *Implementación de una aplicación software para procesamiento de imágenes con wavelets y bancos de filtros paramétricos de reconstrucción perfecta.*

Este proyecto consistió en implementar un programa visual que permita al usuario manipular el espectro de energía y la entropía de una imagen por medio de la transformada wavelet discreta con bancos de filtros paramétricos de reconstrucción perfecta. La similitud con nuestro proyecto es que se realiza el

análisis y procesamiento de imágenes. Difiere en que es solo con Software además de usar el formato de imagen PGM y el algoritmo de la transformada Wavelet Discreta e Inversa [7].

Referencias Externas

Tesis

- *Algoritmo de detección de bordes e imágenes con nios II*

Este proyecto consiste implementar un algoritmo que sea capaz de obtener solo los bordes de una imagen a través de una tarjeta Altera DE2. Tiene cierta similitud en que también procesa imágenes con la tarjeta Altera DE2. Difiere en que está implementado en lenguaje de programación C y en que solo obtiene los bordes de la imagen [8].

- *Reproducción de una Imagen en un Monitor VGA utilizando un FPGA*

Este proyecto consistió en implementar un programa para capturar y enviar la información de una imagen a un FPGA el cual a su vez la envía a un Monitor VGA. Tiene cierta similitud ya que el FPGA procesara la imagen recibida por el programa y la envía a otro dispositivo. Difiere en que es un programa hecho en Visual Basic el que envía la imagen al FPGA y este a su vez solo la envía a otro periférico y no comprímela imagen y tampoco la transmite a un servidor [9].

- *Procesamiento y análisis digital de imágenes mediante dispositivos lógicos Programables.*

Este proyecto consistió en la implementación de un sistema de procesamiento y análisis digital de imágenes sobre un FPGA. La similitud es que ambos utilizan un FPGA para el procesamiento de imágenes. La diferencia es que solo es una herramienta que brindara una base sólida en el diseño e implementación de prototipos autónomos que incluyen algoritmos de procesamiento y/o análisis digital de imágenes, sólo tiene usos académicos e investigación y no realiza compresión de imágenes [10].

JUSTIFICACION

Ya se ha trabajado el tema de la compresión y encriptado de imágenes con diferentes formatos incluyendo JPEG. Casi todos los trabajos toman la imagen desde un archivo en memoria, la comprimen y el resultado lo depositan en otro archivo en memoria. Se han realizado trabajos en varias partes del planeta usando un FPGA y Verilog HDL (Hardware Description Language), pero la gran mayoría

concluyen al obtener el archivo de una imagen comprimido, no usan el archivo resultante para alguna aplicación práctica, los resultados obtenidos son teóricos, muchos trabajos usan una computadora para realizar la compresión.

La justificación más importante del proyecto propuesto en este documento es suministrar el archivo con formato JPEG a una aplicación que se ejecuta en un servidor de un centro de datos. Se propone usar una tarjeta con un FPGA por tener un precio más accesible que una computadora y porque es más fácil de instalar en la entrada de un centro de datos.

Para garantizar la continuidad de servicio a clientes, empleados, ciudadanos, proveedores y empresas colaboradoras, es vital que el Centro de Procesamiento de Datos cuente con mecanismos seguros y eficientes para el funcionamiento del equipo, ya que contiene información crítica y necesaria para las operaciones diarias de la empresa a fin de evitar poner en riesgo la productividad y el negocio. Periódicamente los centros de datos son auditados por organismos y empresas externas para poder estar certificados y ofrecer servicios garantizados a sus clientes. Un punto importante que consideran las auditorías son los procesos de control de acceso. Recientemente se han empezado a usar aplicaciones que procesan imágenes capturadas por las cámaras del CPD y que usan como entrada precisamente un archivo en formato JPEG.

OBJETIVOS

Objetivo general

Convertir archivos de imágenes a formato comprimido JPEG y transmitirlos a un servidor usando un FPGA.

Objetivos específicos:

- Adquirir una imagen en la memoria RAM de una tarjeta Altera DE2, enviada desde una cámara de video.

- Desarrollar algoritmos en un FPGA para efectuar la compresión en formato JPEG de un cuadro en escala de grises, de imágenes almacenadas en la memoria RAM de la tarjeta DE2.

- Realizar la transmisión de un archivo en formato JPEG comprimido a un servidor de imágenes, mediante la interface Ethernet de una tarjeta Altera DE2 y el protocolo FTP.

MARCO TEORICO

En aplicaciones de procesamiento de imágenes digitales, los archivos pueden tener diferentes formatos. Cada formato tiene una extensión diferente, de los cuales los más utilizados son: BMP, GIF, JPG, TIF y PNG. Cada uno utiliza un algoritmo de compresión diferente para reducir el espacio ocupado en disco y facilitar su transmisión a través de la red [11].

Con el algoritmo JPEG se puede ajustar el grado de compresión, de tal forma que al usar un grado de compresión alto se obtiene un archivo pequeño pero se pierde calidad. Con una tasa de compresión baja se obtiene un archivo de calidad similar al original pero de tamaño grande. El algoritmo JPEG está basado en dos características del ojo humano. La primera es que el ojo es más sensible al cambio en la luminancia que en la crominancia, debido que capta más claramente los cambios de brillo que los de color. La segunda es que percibe más fácilmente cambios de brillo en áreas homogéneas que en zonas donde la variación es grande. El algoritmo considera que las imágenes digitales se pueden representar como matrices para poder comprimirse a través de las cuatro etapas siguientes [12].

- Conversión de la imagen de formato de color RGB a formato de color YIQ. El canal Y representa la luminancia y los canales I y Q representan la crominancia.
- Transformación de la imagen. Se divide la imagen en matrices de 8x8 pixeles, se aplica la transformada de coseno discreta (DCT) a cada matriz y se redondea cada elemento de éstas al entero más cercano.
- Cuantificación digital. Se aprovecha que el ojo humano detecta bien cambios de brillo en áreas grandes y no cambios rápidos de brillo en áreas pequeñas, esto es, variación de alta frecuencia. Por tanto se eliminan altas frecuencias sin perder calidad visual, dividiendo cada componente de las matrices del paso anterior, en el dominio de la frecuencia, entre una constante y el resultado se redondea al entero más próximo.
- Codificación entrópica. Se realiza una compresión especial donde no existe pérdida de información. Usa el código Huffman en cada sub-imagen o

matriz de 8x8. El algoritmo Huffman toma los elementos de la matriz en forma de zig-zag para obtener una lista de elementos con ceros acumulados al final.

En este trabajo se solicitó que la compresión se realice usando el algoritmo JPEG estándar, para lo cual se usó el FPGA de una tarjeta Altera D2. Las ventajas y aportaciones del sistema desarrollado son las siguientes: reducción de tamaño en los archivos que almacenan las imágenes de rostros, y por tanto el espacio en disco necesario, y reducción de tiempo de reconocimiento de rostros y apertura de la puerta de acceso en la aplicación usada en el centro de datos. No se ha realizado una solución como la aquí presentada, implantando el algoritmo JPEG con un FPGA e integrando un lector de tarjetas RFiD para un centro de datos.

DESARROLLO DEL PROYECTO

La metodología seguida para el diseño del sistema fue dividirlo en dos partes: la programación del FPGA y la interfaz de usuario. La operación del sistema es la siguiente: cuando una persona se identifica con una tarjeta RFiD, a través de un lector instalado en la puerta de acceso del centro de datos, el sistema lee el identificador de la tarjeta (TID) y lo transmite, a través de la interfaz Ethernet de la tarjeta Altera DE2, a un servidor. A continuación, el sistema solicita a la cámara de video la captura de imagen del rostro de la persona y espera recibir de la cámara el archivo de la imagen en formato TIF. Posteriormente, la programación del FPGA comprime el archivo de la imagen usando el algoritmo JPEG y lo transmite al servidor. La interfaz de usuario, ejecutándose en el servidor, registra y muestra el nombre asociado con el TID leído así como la imagen de la persona. Las imágenes registradas son utilizadas por una aplicación, no realizada en este trabajo y adquirida por el centro de datos, para reconocimiento y validación de rostros de usuarios autorizados y apertura de la puerta. La Figura. 1 muestra el diagrama de bloques del sistema cuya funcionalidad es la explicada anteriormente.

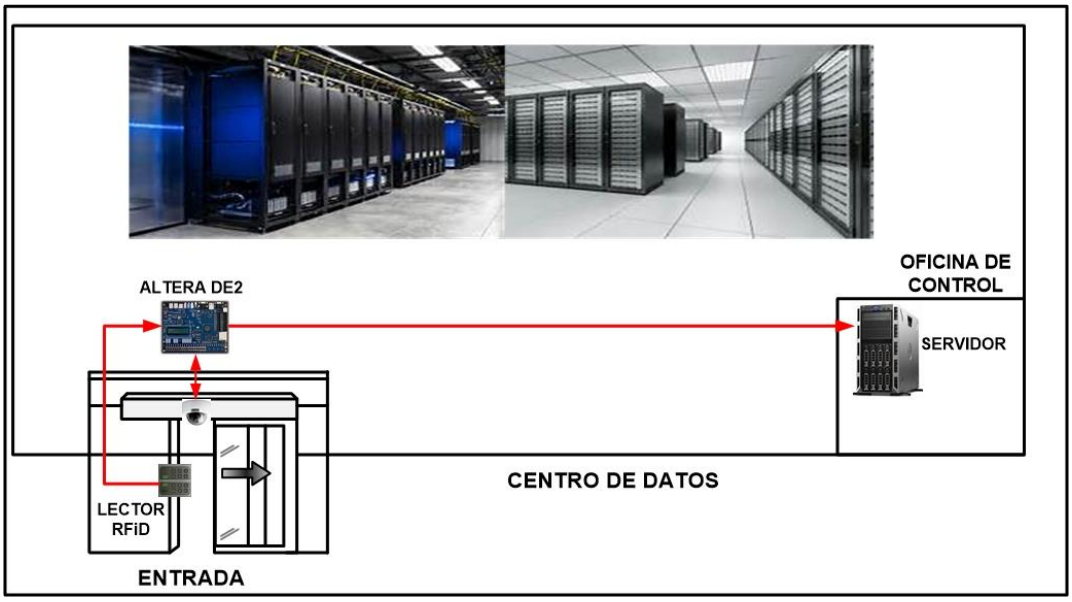


Fig. 1. Diagrama de bloques del sistema desarrollado

La programación del FPGA

Se utilizó una tarjeta Altera DE2, la cual tiene los siguientes componentes: FPGA Cyclone II EP2C35, 8 MB memoria SDRAM, 512 KB SRAM, 4 MB memoria Flash, 4 push-button, 18 interruptores DPDT, 27 leds, display LCD de 16x2, módulo de memoria SD e interfaz Ethernet 10/100 BT. Esta parte del sistema realiza la mayoría de funciones del sistema y está compuesta por los siguientes cuatro módulos: el módulo lector RFiD, el módulo captura de imagen, el módulo de compresión y el módulo Ethernet, como se indica en el diagrama de bloques de la Fig. 2.

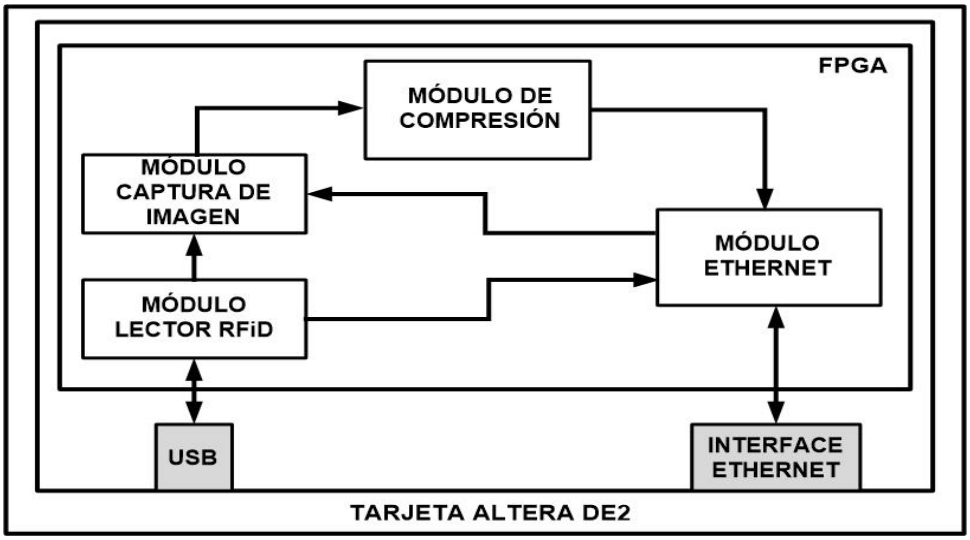


Fig. 2. Diagrama de bloques de la programación de la tarjeta Altera DE2

La función del *módulo lector RFiD* es obtener la información de la tarjeta RFiD usada por las personas que intentan acceder al área del centro de datos donde se encuentra instalado el sistema aquí presentado. Para lograr está función, se conectó un circuito SkyeModule Gemini al puerto USB de la tarjeta Altera DE2. El SkyeModule Gemini es un dispositivo que permite leer y escribir etiquetas RFiD del tipo MIFARE y tarjetas basadas en el estándar ISO14443A/B. La tarjeta Altera DE2 suministra la alimentación de 5 V al lector, a través del puerto USB, y la transmisión de información se lleva a cabo por las terminales USB_DP y USB_DN del puerto USB del SkyeModule Gemini. La programación del FPGA de este módulo realiza las siguientes tareas: establece la velocidad del puerto USB a 38.4 Kbps, envía el comando *Loop_F* al lector, indicando que continuamente explore el campo de detección RFiD en busca de un TID, entra a un ciclo del que sale cuando el lector ha detectado un TID, solicita al módulo Ethernet que trasmita el TID al servidor y finalmente transfiere el control al módulo de captura de imagen. La comunicación entre el módulo lector RFiD y el SkyeModule Gemini lleva a cabo usando el protocolo propietario SkyeTek Protocol (STPv3), el cual trabaja con el formato de comando-respuesta[13].

La función del *módulo de captura de imagen*, es obtener el archivo TIF de la imagen del rostro de la persona. Para llevar a cabo esta función, la programación de FPGA realiza las siguientes tareas: solicita al módulo Ethernet la transmisión del comando de captura a la cámara de video, entra a un ciclo donde espera del módulo Ethernet el archivo TIF enviado por la cámara, almacena en memoria RAM el archivo TIF recibido y transfiere el control al módulo de compresión.

La función del *módulo de compresión* es convertir el archivo de la imagen a formato JPEG. La programación del FPGA lee el archivo TIF de la memoria RAM y para comprimirlo implanta las cuatro etapas del algoritmo JPEG indicadas en el diagrama de bloques de la Fig. 3. Las tareas que realiza la programación del FPGA en cada etapa se explica a continuación.

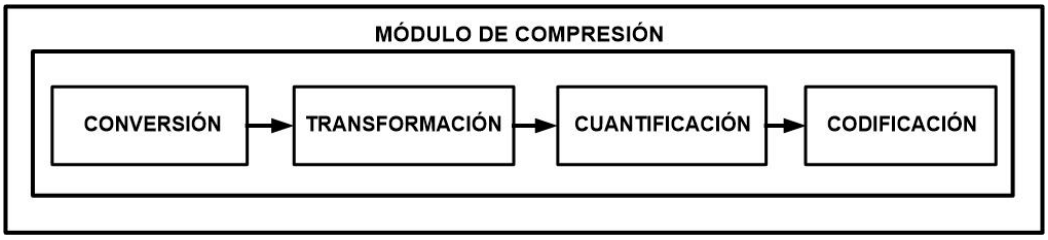


Fig. 3. Diagrama de bloques del módulo de compresión

Conversión de la imagen de formato de color RGB a formato de color YIQ. El formato de color YIQ indica la luminosidad e información del color. Las imágenes en formato de color RGB se almacenan en 3 canales independientes: rojo, verde y azul, usando 8 bits para indicar la intensidad. El formato de color YIQ representa una división entre la luminosidad (Y) y el color (I, Q). La programación del FPGA usa las siguientes ecuaciones para convertir el archivo de imagen de RGB a YIQ:

$$Y = (0.257 * R) + (0.504 * G) + (0.098 * B) + 16 \quad (1)$$
$$I = (-0.148 * R) - (0.291 * G) + (0.439 * B) + 128 \quad (2)$$
$$Q = (0.439 * R) - (0368 * G) + (0.071 * B) + 128 \quad (3)$$

Debido a que el ojo humano es más sensible a la luminosidad que al color, en esta etapa se realiza una reducción de información de los canales I y Q de la imagen [14].

Transformación de la imagen. En esta etapa, la programación del FPGA implanta un procesador Nios II cuyas funciones son las siguientes: dividir la imagen en matrices de 8x8 pixeles, aplicar la DCT y redondear al entero más cercano cada elemento de las matrices. Se divide la imagen para procesar cada matriz de 8x8 independientemente y disminuir el tiempo de procesamiento. Cada matriz de 8x8 se convierte al dominio de la frecuencia usando la DCT, comprimiendo la imagen en base a patrones de frecuencia. La transformada de una imagen genera una serie de datos que contiene la misma información que la imagen original, la cual puede usarse para obtener la imagen original aplicando la transformada inversa. La DCT expresa una secuencia finita de números como una sumatoria de términos de coseno con distintas frecuencias y amplitudes. La DCT está basada en la transformada discreta de Fourier (DFT) pero para números reales. La DFT representa funciones periódicas como una sumatoria de senos de diferente frecuencia. La DCT, y específicamente la DCT-II, es usada para compresión de imágenes por su propiedad de alta compactación de energía o información original. Gran parte de la información original la compacta usando una cantidad pequeña de bits. La DCT concentra la mayor parte de la información con pocos coeficientes transformados. Una vez convertida la imagen a formato de color YIQ *NxN*, se puede expresar como una función de dos variables *f(x,y)*. Donde *(x,y)* son las coordenadas de cada píxel, *x=0,1,...,N-1* e *y=0,1,...,N-1* y la función *f(x,y)* es la intensidad de color del píxel *(x,y)* [15].

La DCT sirve para obtener una matriz *F* a partir de la anterior. El dominio de *F* es el mismo que el de *f* para cada valor *(u,v)* con *u=0,1,2,...,N-1* y *v=0,1,2,...,N-1*, por lo que se obtiene la siguiente ecuación, la cual se implantó con la programación del FPGA de este módulo del sistema:

$$C(u,v) = \alpha(u)\alpha(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)cos\left(\frac{(2x+1)u\pi}{2N}\right)cos\left(\frac{(2y+1)v\pi}{2N}\right) \quad (3)$$

Por ejemplo, en la etapa anterior se obtuvo la matriz de color A en formato YIQ indicada en la Fig. 4.

31	23	60	93	49	57	57	35
31	21	74	73	55	52	45	44
25	16	82	65	61	38	50	51
26	38	91	61	62	28	49	50
17	85	70	76	50	34	37	43
29	90	68	53	43	25	30	49
116	184	108	25	52	26	38	36
151	144	182	129	44	23	41	39

Fig. 4. Matriz A, imagen de color YIQ

Al aplicar la DCT a la matriz A, se obtuvo la matriz B indicada en la Fig. 5, ambas de iguales dimensiones.

466.2500	100.9063	-40.3393	-91.13.15	-25.2500	2.6791	-3.3152	16.6482
-88.4065	-146.0661	-67.5059	26.1243	33.4120	44.3008	24.6658	-0.4861
69.5604	73.4218	17.6909	8.1087	-1.2285	31.3229	12.3824	-7.9303
-44.0839	-28.5187	4.3866	18.4638	-11.9507	-32.1620	-31.1853	-32.1898
27.5000	12.2552	-25.4491	-19.8833	9.5000	35.2306	3.2352	-10.1438
-2.2299	6.3770	26.7637	18.3775	-13.4471	-20.9802	-10.1116	19.2261
-10.4122	-22.5187	-20.8676	-15.9436	17.2859	25.7420	9.5591	-14.7470
8.0193	11.5534	7.5823	9.4544	-9.8801	-12.0629	-13.2570	-7.4175

Fig. 5. Matriz B, resultado de aplicar la DCT

Los valores más grandes de la matriz B se encuentran en la parte superior-izquierda. Esto indica que se concentra gran parte de la información en los primeros valores. Para almacenar la matriz B se realizó un proceso de normalización, usando la función *N(u,v)* tal que: *C(u,v)=Redondeo(F(u,v)/N(u,v))*, donde *C* es una matriz con una cantidad grande de ceros. El algoritmo JPEG recomienda el uso de una matriz de normalización estandarizada, como la indicada en la Fig. 6, para imágenes con una cantidad grande de niveles de intensidad, por lo que la programación del FPGA normaliza la matriz B usando la matriz de la Fig. 6, obteniendo como resultado la matriz D mostrada en la Fig. 7.

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Fig. 6. Matriz de normalización estandarizada

29	9	-4	-6	-1	0	0	0
-7	-12	-5	1	1	1	0	0
5	6	1	0	0	1	0	0
-3	-2	0	1	0	0	0	-1
2	1	-1	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Fig. 7. Matriz D, resultado de la normalización

Cuantificación digital. En esta etapa se aprovecha que el ojo humano detecta cambios de brillo en áreas grandes y no cambios rápidos de brillo en áreas pequeñas, esto es, variación de alta frecuencia. Por tanto se pueden eliminar altas frecuencias sin perder calidad visual. Para lograr esto, la programación del FPGA divide cada componente en el dominio de la frecuencia entre una constante y el resultado se redondea al entero más próximo. Es en este paso donde se generó la mayor pérdida de información. Los componentes de frecuencias altas tienden igualarse a cero y los restantes se convierten a números positivos o números negativos pequeños. Para realizar la división, el algoritmo JPEG recomienda usar matrices estándar para imágenes con 256 niveles de intensidad, una de ellas es la matriz de cuantificación típica de Losheller, la cual se indica en la Fig. 8 y es la utilizada en esta etapa. Los coeficientes de la matriz D transformada se dividieron entre los coeficientes de la matriz de Losheller obteniendo la matriz E redondeada, indicada en la Fig. 9. Esta matriz tiene una cantidad grande ceros en la parte inferior-derecha.

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Fig. 8. Matriz de Losheller

29	9	-4	-6	-1	0	0	0
-7	-12	-5	1	1	1	0	0
5	6	1	0	0	1	0	0
-3	-2	0	1	0	0	0	-1
2	1	-1	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Fig. 9. Matriz E, cuantificación digital

Codificación entrópica. En esta última etapa se realizó una compresión especial donde no hay pérdida de información. Se utilizó el algoritmo Huffman en cada sub-imagen o matriz de 8x8. La programación del FPGA implanta el algoritmo Huffman leyendo los elementos de la matriz en forma de zig-zag, como se indica en la Fig. 10, para obtener una lista de elementos con ceros acumulados al final. A continuación, conjunta los elementos de frecuencias similares e inserta ceros de codificación. Con esta codificación se utiliza menor cantidad de espacio eliminando los números más repetidos, la redundancia. Se puede usar algún método de codificación aritmética, más óptimo que el algoritmo Huffman, pero generalmente están patentados. En esta etapa se almacena la información de la imagen usando distintos grados de compresión. Si se almacenan los 64 elementos de la matriz, el grado de compresión es pequeño. Cómo los datos significativos están al inicio de la matriz, se pueden considerar sólo los primeros elementos de la misma y el grado de compresión es mayor.

Al implantar con el FPGA el algoritmo Huffman a la matriz E obtenida en el paso anterior, se obtuvo la siguiente secuencia: 29, 9, -7,5, -12, -4, -6, -5, 6, -3, 2, -2, 1, 1, -1, 0, 1, 0, 0, 1, 0, 0, 0, -1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, F. La letra F, indica que a partir de ese elemento son todos ceros hasta completar los 64 elementos de la lista. El algoritmo JPEG trunca la secuencia en la cadena anterior cuando el resto de los coeficientes son ceros para ahorrar espacio, de manera que la secuencia anterior se redujo de la siguiente forma: −26, −3, 0, −3, −2, −6, 2, −4, 1 −4, 1, 1, 5, 1, 2, −1, 1, −1, 2, 0, 0, 0, 0, 0, −1, −1, F. Finalmente, la programación del FPGA obtiene, para cada matriz E una secuencia como la anterior, para crear el archivo comprimido en formato JPEG en la memoria RAM de la tarjeta Altera DE2 y solicita al módulo Ethernet la transmisión de este archivo.

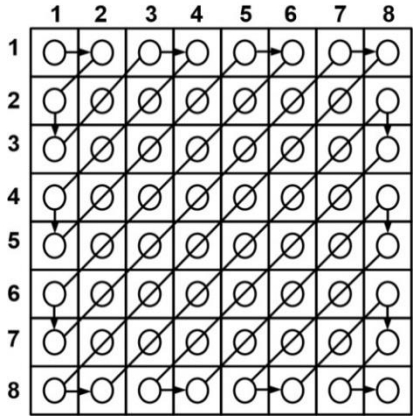


Fig. 10. Algoritmo Huffman

El último módulo de la programación del FPGA es el *módulo Ethernet*. La función de este módulo es transmitir el TID obtenido por el módulo lector RFiD y el archivo JPEG al servidor. Para llevar a cabo esta función, la programación del FPGA realiza las siguientes tareas: configura e inicializa la interfaz Ethernet DM9000A de la tarjeta Altera DE2 a 100 Mbps, entra en un ciclo donde espera solicitud de servicio del módulo lector RFiD o del módulo de compresión. Si el servicio lo solicita el primero de los módulos anteriores, establece la conexión vía ftp con la cámara de video para descargar en memoria RAM el archivo de la imagen TIF. Si el servicio lo solicita el módulo de compresión, establece la conexión vía ftp con el servidor para transferirle al archivo JPEG. Las direcciones MAC e IP usadas por la cámara, el servidor y la interfaz Ethernet de la tarjeta Altera DE2 son fijas.

La interfaz de usuario

La interfaz de usuario se realizó en Visual Basic y Python 3.6.0. Se ejecuta en el servidor ubicado en una oficina de control y monitoreo del centro de datos. La interfaz recibe de la tarjeta Altera DE2, vía un segmento Ethernet, el TID leído de la tarjeta RFiD y el archivo JPEG del rostro de la persona que intenta acceder al área del centro de dato. Posteriormente, almacena esta información en disco para estar disponible a la aplicación de reconocimiento de imágenes. La interfaz muestra el nombre de cada persona y la imagen de su rostro como se indica en la Figura. 11.

Fig. 11. Interfaz de usuario

RESULTADOS ANALISIS Y DISCUSIÒN DE RESULTADOS

Se realizaron tres grupos de pruebas. El objetivo del primer grupo fue determinar el alcance del lector RFiD. Los resultados de las pruebas mostraron que el alcance es 1.2 metros, suficiente para leer el TID de la tarjeta al momento de ubicarse la persona frente la puerta de acceso y bajo la cámara de video, por lo que no hubo necesidad de conectar al lector una antena externa. En el segundo grupo de pruebas se capturó la imagen del rostro de varias personas, dando como resultado un archivo TIF, para cada una, cuyo tamaño es en promedio 600 KB, por lo que la memoria SDRAM de 8 MB de la tarjeta Altera DE2 es suficiente para almacenar este archivo. Difícilmente se tendrán archivos TIF mayores a 1 MB en esta aplicación. La tasa de compresión obtenida en este sistema fue 15%. El tercer grupo de pruebas consistió en modificar las etapas de cuantificación y codificación del algoritmo JPEG para lograr una mejor tasa de compresión. En la etapa de cuantificación, donde se tiene mayor pérdida de información, los puntos de la imagen de frecuencia alta se eliminan. Las pruebas en esta etapa consistieron en realizar la división de matrices usando como divisor tres matrices estándar, recomendadas por el algoritmo, en lugar de usar la matriz de Losheller.

Las tasas de compresión obtenidas fueron 12%, 13% y 14%. En la etapa de codificación se eliminan los elementos de la matriz cuyo valor es cero. Las pruebas llevadas a cabo en esta etapa consistieron en incrementar el grado de compresión eliminando los elementos cuyo valor se encuentre entre 1 y 10. Los resultados obtenidos fueron tasas de compresión entre 10% y 12%. Como puede

observarse, las tasas de compresión obtenidas modificando estas dos etapas del algoritmo mejoran un poco la tasa lograda originalmente (15%), encontrándose todas en el rango aceptable de 10% a 20% de la aplicación de reconocimiento de imágenes del centro de datos. Se pudo haber realizado la etapa de transformación de forma más sencilla sin implantar el procesador Nios II en el FPGA, sin embargo, se optó por utilizar al Nios II para aprovechar sus características de paralelismo y arquitectura RISC de 32 bits y obtener la DCT más rápidamente. Aunque el algoritmo estándar JPEG no es nuevo, se ha trabajado mucho y se continúan realizando estudios sobre su eficiencia, no es fácil implantarlo en un FPGA y los beneficios que se obtienen son eficiencia en costo y velocidad de compresión.

CONCLUSIONES

Ya que el manejo de imágenes depende en gran medida de la velocidad del procesamiento de datos, debido a que requiere una gran cantidad de procesamiento de señales, la implantación del mismo en un FPGA fue una buena elección. En este trabajo se alcanzó el objetivo planteado, el cual fue crear un sistema de compresión de imágenes de bajo costo cuyo resultado es un archivo de mucho menor tamaño que el generado por la cámara de video usando un FPGA Cyclone II 2C35. El diseño del sistema surgió para resolver una necesidad específica y real, mostrando una aplicación completa terminada. No incluye reconocimiento de imágenes, lo cual está planeado implantarse en el FPGA a corto plazo. Con esta adición al sistema se determinará más rápidamente si debe o no abrirse la puerta de acceso. Se tiene planeado también incorporar al sistema, específicamente en el módulo Ethernet, la programación que determine las direcciones MAC e IP asociadas con la interface de red de la cámara y el servidor para hacer más flexible su uso y pueda usarse en otras aplicaciones como por ejemplo sistemas de seguridad biométricos y sistemas de tratamiento médico. La realización de este trabajo sirvió para establecer y generar la relación de confianza con el centro de datos y continuar trabajando en esta y otras aplicaciones.

REFERENCIAS BIBLIOGRAFICAS

[1] El Mir, I., D. S. Kim y A. Haqiq, "Security modeling and analysis of an intrusion tolerant cloud data center", Third World Conference on Complex Systems (WCCS), 1-6, Marrakech-Morocco (2015).

[2] Hackenberg, D., "The Plenum concept: Improving scalability, security, and efficiency for data centers", Fourteenth Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm), 1137-1144, Orlando, FL-USA (2014).

[3] Ayadi, W., W. Elhamzi y M. Atri, "A FPGA-based implementation of JPEG encoder", International Image Processing, Applications and Systems (IPAS), 1-4, Hammamet-Tunisia (2016).

[4] Luo, Z. e Y. Wan, "An efficient framework for lossless color image compression", International Conference on Audio, Language and Image Processing (ICALIP), 380-384, Shanghai-China (2016).

[5] Díaz Cabrera Fausto Mario, “Clasificador de Objetos en Banda Infinita por Medio de Procesamiento Digital de Imágenes”, Proyecto Tecnológico, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana, México, 2009.

[6] Moisés Chávez Pérez, “Aplicación de Filtros Digitales en la Compresión de Imágenes”, Proyecto Tecnológico, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana, México, 2015.

[7] Oscar Claudio Vargas, “Implementación de una aplicación software para procesamiento de imágenes con wavelets y bancos de filtros paramétricos de reconstrucción perfecta”, Proyecto Tecnológico, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana, México, 2011.

[8] Calle Andres, Pasmiño Paola, Ponguillo Ronald Ing., “Algoritmo de detección de bordes en imágenes con NIOS II”, Articulo Tesis Grado, Escuela Superior Politécnica del Litoral, 2014.

[9] Michael Alejandro Díaz Illa, Alfredo Granados Ly, “Reproducción de una Imagen en un Monitor VGA Utilizando un FPGA”, Facultad de Ingeniería Electrónica y Eléctrica, Universidad Nacional Mayor de San Marcos, Lima, Perú, 2007.

[10] C. Manuel Alejandro Mendoza Manzano, “Procesamiento y análisis digital de imágenes mediante dispositivos lógicos Programables”, Tesis, Universidad Tecnológica de la Mixteca, Oaxaca, 2009.

[11] Chouhan, A. y M. J. Nigam, "Double compression of JPEG image using DCT with estimated quality factor", IEEE 1st International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES), 1-3, Delhi-India (2016).

[12] Wibisono, S. S. y F. B. Setiawan, "Automatic doorstop safety system based on image processing using webcam and scanner", The 1st International Conference on Information Technology, Computer, and Electrical Engineering, 150-154, Semarang-Indonesia (2014).

[13] Federico Garcia Crespi, Otoniel Lopez. “Implementación de algoritmos de compresión de imágenes en FPGAs”, Departamento de Física y Arquitectura de Computadores de la Universidad Miguel Hernández de Elche.

[14] Gu, J. y Huayu, Y.; Real-Time Image Collection and Processing System Design, 2015 Fifth International Conference on Instrumentation and Measurement, Computer, Communication and Control (IMCCC), Pages: 1649-1652, 18-20 Sept., DOI: 10.1109/IMCCC.2015.350 (2015).

[15] Li, Y., W. Jia, B. Luan y Z. Mao, "A FPGA implementation of JPEG baseline encoder for wearable devices", 41st Annual Northeast Biomedical Engineering Conference (NEBEC), 1-2, Troy, NY-USA (2015).

ENTREGABLES (código fuente de cada módulo)

Código fuente módulo de la DCT (Transformada Discreta del Coseno)

```
# Programa  para calcular la DCT

module SAR12cal ( clk, por_n, start, cal_en, cal_reg_sel, cal_reg0

,cal_reg1, cal_reg2, cal_reg3, cal_reg4, cal_reg5, cal_reg6, cal_reg7,

cal_reg8, cal_reg9,cal_reg10, cal_reg11, comp, eocout, track, precharge,

precharge_n, qdacout, qdacout_cal, qoutout, qdacoutCt, sample, sample_n );

 //puertos in/out

 input clk;

 input por_n;
```

```
 input start;
 input cal_en;
 input [2:0] cal_reg_sel;
 input signed [`NBITS_CAL-1:0] cal_reg0, cal_reg1, cal_reg2, cal_reg3,
cal_reg4, cal_reg5, cal_reg6, cal_reg7, cal_reg8, cal_reg9, cal_reg10,
cal_reg11;
 input comp;
 output reg eocout;
 output reg track;
 output reg precharge;
 output reg precharge_n;
 output reg sample;
 output reg sample_n;
 output [`NBITS_SAR-1:0] qdacout;
 output reg qdacoutCt;
 output [`NBITS_CAL-1:0] qdacout_cal;
 output reg [`NBITS_SAR-1:0] qoutout;
 //parametros
 parameter sWait = 4'b0000;
 parameter sPrecharge = 4'b0100, sSample = 4'b0101, sConv = 4'b0110, sDone
= 4'b0111;
 parameter sPrecharge_Cal = 4'b1000, sSample_Cal = 4'b1001, sChargeEq_Cal
```

```
= 4'b1010, sConv_Cal = 4'b1011, sDone_Cal = 4'b1100;
 //variables internas
 reg [3:0] state; //almacena el estado actual
 reg [`NBITS_CAL-1:0] calibration_reg; //registro acumulador de la
calibracion
 reg [`NBITS_SAR-1:0] mask; //mascara para busqueda SAR del convDAC
 reg [`NBITS_CAL-1:0] mask_cal; //mascara para busqueda SAR del
calDAC
 reg [`NBITS_CAL-1:0] qoutout_cal; //variable busqueda SAR del calDAC
 reg [`NBITS_CAL-1:0] cal_bit; //salida del MUX
 reg [`NBITS_CAL-1:0] cal_reg; //salida del MUX
 //MAQUINA DE ESTADOS
 always @(posedge clk or negedge por_n) begin
 if (!por_n) begin
 state <= sWait;
 mask <= 12'h000;
 qoutout <= 12'h000;
 mask_cal <= 8'h80;
 qoutout_cal <= 8'h00;
 end
 else case (state) //paso al siguiente estado
 ///ESTADO DE ESPERA (y PRECARGA)
```

```
 sWait :
 begin
 mask <= 12'h000;
 qoutout <= 12'h000;
 mask_cal <= 8'h80;
 qoutout_cal <= 8'h00;
 //pasa al siguiente estado
 if (start && !cal_en) state <= sPrecharge; //CONV
 else if (start && cal_en) state <= sPrecharge_Cal; //BUSQ CAL
 else state <= sWait; //IDLE
 end
/// CONVERSION ////////////////////////////////////////////////////
 ///Estado de precarga
 sPrecharge :
 begin
 //siguiente estado
 state <= sSample;
end
 ///Estado de muestreo
 sSample :
 begin
 //siguiente estado
```

```
 state <= sConv;
 //actualizacion de salidas
 mask <= 12'h800;
 mask_cal <= cal_bit + cal_reg; //sumador registrado!
 end
///Estado de conversion
 sConv :
 begin
 //busqueda SAR convDAC
 if (!comp) qoutout <= qoutout | mask;
 if (mask[0] == 1'b0) begin
 mask <= mask >> 1; //siguiente 'bit activo'
 end
 else begin
 state <= sDone; //siguiente estado
 end
 //calibracion calDAC
 mask_cal <= cal_bit + cal_reg; //sumador registrado!
 end
 ///Estado de fin de conversion
 sDone :
 begin
```

```
//siguiente estado
state <= sWait;
end
/// BUSQ. CALIBRACION ////////////////////////////////////////////
///Estado de precarga
sPrecharge_Cal :
begin
//siguiente estado
state <= sSample_Cal;
//actualizacion de salidas
case(cal_reg_sel)
3'd7: mask <= 12'h7ff; //dacout <= 0111_1111_1111
3'd6: mask <= 12'h3ff; //dacout <= 0011_1111_1111
3'd5: mask <= 12'h1ff; //dacout <= 0001_1111_1111
3'd4: mask <= 12'h0ff; //dacout <= 0000_1111_1111
3'd3: mask <= 12'h07f; //dacout <= 0000_0111_1111
3'd2: mask <= 12'h03f; //dacout <= 0000_0011_1111
3'd1: mask <= 12'h01f; //dacout <= 0000_0001_1111
3'd0: mask <= 12'h00f; //dacout <= 0000_0000_1111
endcase
end
///Estado de muestreo
```

```
sSample_Cal :
begin
//siguiente estado
state <= sChargeEq_Cal;
end
///Estado de muestreo2
sChargeEq_Cal :
begin
//siguiente estado
state <= sConv_Cal;
//actualizacoin de salidas
case(cal_reg_sel)
3'd7: mask <= 12'h800; //dacout <= 1000_0000_0000
3'd6: mask <= 12'h400; //dacout <= 0100_0000_0000
3'd5: mask <= 12'h200; //dacout <= 0010_0000_0000
3'd4: mask <= 12'h100; //dacout <= 0001_0000_0000
3'd3: mask <= 12'h080; //dacout <= 0000_1000_0000
3'd2: mask <= 12'h040; //dacout <= 0000_0100_0000
3'd1: mask <= 12'h020; //dacout <= 0000_0010_0000
3'd0: mask <= 12'h010; //dacout <= 0000_0001_0000
endcase
end
```

```
///Estado de busqueda
sConv_Cal :
begin
//busqueda SAR calDAC
if (!comp) qoutout_cal <= qoutout_cal | mask_cal;
if (mask_cal[0] == 1'b0) begin
mask_cal <= mask_cal >> 1; //siguiente 'bit activo'
end
else begin
state <= sDone_Cal; //siguiente estado
qoutout[7:0] <= qoutout_cal; //valor encontrado
end
end
///Estado de almacenamiento
sDone_Cal :
begin
//siguiente estado
state <= sWait;
end
///Estado por defecto (reseta la FSM)
default:
begin
```

```
 state <= sWait;
 mask <= 12'h000;
 qoutout <= 12'h000;
 mask_cal <= 8'h80;
 qoutout_cal <= 8'h00;
 end
 endcase
 end
 //MUX: cal_bit
 always @(cal_reg11 or cal_reg10 or cal_reg9 or cal_reg8 or cal_reg7 or
cal_reg6 or cal_reg5 or cal_reg4 or cal_reg3 or cal_reg2 or cal_reg1 or
cal_reg0 or mask) begin
 case(mask)
 12'd2048: cal_bit = cal_reg10;
 12'd1024: cal_bit = cal_reg9;
 12'd512: cal_bit = cal_reg8;
 12'd256: cal_bit = cal_reg7;
 12'd128: cal_bit = cal_reg6;
 12'd64: cal_bit = cal_reg5;
 12'd32: cal_bit = cal_reg4;
 12'd16: cal_bit = cal_reg3;
 12'd8: cal_bit = cal_reg2;
```

```
 12'd4: cal_bit = cal_reg1;
 12'd2: cal_bit = cal_reg0;
 12'd0: cal_bit = cal_reg11;
 default: cal_bit = 8'h00;
 endcase
 end
//MUX: cal_reg
 always @(calibration_reg or mask_cal or comp) begin
 case(comp)
 1'd0: cal_reg = mask_cal;
 1'd1: cal_reg = calibration_reg;
 endcase
 end
 //REG: calibration_reg
 always @(posedge clk or negedge por_n) begin
 if (!por_n) calibration_reg <= 8'h80;
 else case(state)
 sWait: calibration_reg <= 8'h80;
 default: calibration_reg <= cal_reg;
 endcase
 end
 //SALIDAS COMBINACIONALES (registradas para evitar glitches!)
```

```
always @(posedge clk or negedge por_n) begin
if (!por_n) begin
precharge <= 1'b1;
precharge_n <= 1'b0;
sample <= 1'b0;
sample_n <= 1'b1;
track <= 1'b0;
eocout <= 1'b0;
qdacoutCt <= 1'b0;
end
else begin
precharge <= (state==sWait) | (state==sDone) | (state==sDone_Cal);
precharge_n <= !precharge;
sample <= (state==sPrecharge) | (state==sPrecharge_Cal);
sample_n <= !sample;
track <= (state==sPrecharge);
eocout <= (state==sDone) | (state==sDone_Cal);
qdacoutCt <= (state==sPrecharge_Cal) | (state==sSample_Cal);
end
end
assign qdacout = qout tout | mask; assign qdacout_cal = qoutout_cal | mask_cal;
endmodule
```

Código fuente módulo de Zig-zag

```
module Zigzag (
        //////////////////////  Clock Input        //////////////////
        input           CLOCK_27,                       //      27 MHz
        input           CLOCK_50,                       //      50 MHz
        input           EXT_CLOCK,                              //      External
Clock
        //////////////////////        Push Button   //////////////////
        input  [3:0]  KEY,                              //      Pushbutton[3:0]
        //////////////////////        DPDT Switch             //////////////////
        input  [17:0] SW,                               //      Toggle
Switch[17:0]
        //////////////////////        7-SEG Display      //////////////////
        output      [6:0]  HEX0,            //      Seven Segment Digit 0
        output      [6:0]  HEX1,            //      Seven Segment Digit 1
        output      [6:0]  HEX2,            //      Seven Segment Digit 2
        output      [6:0]  HEX3,            //      Seven Segment Digit 3
        output      [6:0]  HEX4,            //      Seven Segment Digit 4
        output      [6:0]  HEX5,            //      Seven Segment Digit 5
        output      [6:0]  HEX6,            //      Seven Segment Digit 6
        output      [6:0]  HEX7,            //      Seven Segment Digit 7

        ////////////////////////////  LED           ////////////////////////
        output      [8:0]  LEDG,                            //      LED
Green[8:0]
        output      [17:0] LEDR,            //      LED Red[17:0]
        ////////////////////////////  UART ////////////////////////////
        output                  UART_TXD,       //      UART Transmitter
        input                   UART_RXD,       //      UART Receiver
        ////////////////////////////  IRDA  ////////////////////////////
        output                  IRDA_TXD,       //      IRDA Transmitter
        input                   IRDA_RXD,       //      IRDA Receiver
        //////////////////////                SDRAM Interface   //////////////////
        inout  [15:0] DRAM_DQ,          //      SDRAM Data bus 16 Bits
        output[13:0]   DRAM_ADDR,               //      SDRAM Address bus 12 Bits
        output                  DRAM_LDQM,      //      SDRAM Low-byte
Data Mask
```

```
        output          DRAM_UDQM,      //      SDRAM High-byte
Data Mask
        output          DRAM_WE_N,      //      SDRAM Write Enable
        output          DRAM_CAS_N,     //SDRAM Column Address
Strobe
        output          DRAM_RAS_N,     //      SDRAM Row Address
Strobe
        output          DRAM_CS_N,              //      SDRAM Chip
Select
        output          DRAM_BA_0,              //      SDRAM Bank
Address 0
        output          DRAM_BA_1,              //      SDRAM Bank
Address 0
        output          DRAM_CLK,       //      SDRAM Clock
        output          DRAM_CKE,       //      SDRAM Clock Enable
        ////////////////////////    Flash Interface     ////////////////////
        inout  [7:0] FL_DQ,                     //      FLASH Data bus 8 Bits
        output      [21:0] FL_ADDR,     //      FLASH Address bus 22 Bits
        output          FL_WE_N,        //      FLASH Write Enable
        output          FL_RST_N,       //      FLASH Reset
        output          FL_OE_N,                //      FLASH Output Enable
        output          FL_CE_N,                //      FLASH Chip Enable
        ////////////////////////    SRAM Interface      ////////////////////
        inout  [15:0] SRAM_DQ,          //      SRAM Data bus 16 Bits
        output      [17:0] SRAM_ADDR,       //      SRAM Address bus 18 Bits
        output          SRAM_UB_N,      //      SRAM High-byte Data
Mask
        output          SRAM_LB_N,      //      SRAM Low-byte Data
Mask
        output          SRAM_WE_N,              //      SRAM Write
Enable
        output          SRAM_CE_N,              //      SRAM Chip
Enable
        output          SRAM_OE_N,              //      SRAM Output
Enable
        //////////////////// ISP1362 Interface   ////////////////////////
        inout  [15:0] OTG_DATA,         //      ISP1362 Data bus 16 Bits
        output      [1:0]  OTG_ADDR,        //      ISP1362 Address 2 Bits
        output          OTG_CS_N,       //      ISP1362 Chip Select
        output          OTG_RD_N,       //      ISP1362 Write
        output          OTG_WR_N,               //      ISP1362 Read
```

```
        output          OTG_RST_N,          //  ISP1362 Reset
        output          OTG_FSPEED,         //  USB Full
Speed,  0 = Enable, Z = Disable
        output          OTG_LSPEED,         //  USB Low
Speed,  0 = Enable, Z = Disable
        input       OTG_INT0,           //  ISP1362 Interrupt 0
        input       OTG_INT1,           //  ISP1362 Interrupt 1
        input       OTG_DREQ0,          //  ISP1362 DMA
Request 0
        input       OTG_DREQ1,          //  ISP1362 DMA
Request 1
        output          OTG_DACK0_N, // ISP1362 DMA Acknowledge
0
        output          OTG_DACK1_N,//  ISP1362 DMA Acknowledge
1


        ////////////////////  LCD Module 16X2  ////////////////////////
        inout  [7:0]  LCD_DATA,                 //  LCD Data bus 8 bits
        output          LCD_ON,             //  LCD Power ON/OFF
        output          LCD_BLON,           //  LCD Back Light
ON/OFF
        output          LCD_RW,             //  LCD Read/Write
Select, 0 = Write, 1 = Read
        output          LCD_EN,             //  LCD Enable
        output          LCD_RS,             //  LCD Command/Data
Select, 0 = Command, 1 = Data
        ////////////////////  SD Card Interface  ////////////////////////
        inout       SD_DAT,             //  SD Card Data
        inout       SD_DAT3,            //  SD Card Data 3
        inout       SD_CMD,             //  SD Card Command
Signal
        output          SD_CLK,             //  SD Card Clock
        ////////////////////////    I2C    ////////////////////////////
        inout       I2C_SDAT,                   //  I2C Data
        output          I2C_SCLK,               //  I2C Clock
        ////////////////////////    PS2    ////////////////////////////
        input       PS2_DAT,                    //  PS2 Data
        input       PS2_CLK,                    //  PS2 Clock
        ////////////////////  USB JTAG link    ////////////////////////
        input           TDI,                // CPLD -> FPGA (data in)
```

```
        input                   TCK,            // CPLD -> FPGA (clk)
        input                   TCS,            // CPLD -> FPGA (CS)
        output                  TDO,            // FPGA -> CPLD (data out)
        //////////////////////    VGA         //////////////////
        output                  VGA_CLK,        //      VGA Clock
        output                  VGA_HS,         //      VGA H_SYNC
        output                  VGA_VS,         //      VGA V_SYNC
        output                  VGA_BLANK,              //      VGA BLANK
        output                  VGA_SYNC,       //      VGA SYNC
        output      [9:0]   VGA_R,              //      VGA Red[9:0]
        output      [9:0]   VGA_G,              //      VGA Green[9:0]
        output      [9:0]   VGA_B,          //  VGA Blue[9:0]
        ////////////////    Ethernet Interface  ////////////////////////
        inout   [15:0] ENET_DATA,           //      DM9000A DATA bus 16Bits
        output                  ENET_CMD,//      DM9000A Command/Data
Select, 0 = Command, 1 = Data
        output                  ENET_CS_N,              //      DM9000A Chip
Select
        output                  ENET_WR_N,              //      DM9000A Write
        output                  ENET_RD_N,              //      DM9000A Read
        output                  ENET_RST_N,             //      DM9000A
Reset
        input               ENET_INT,               //      DM9000A Interrupt
        output                  ENET_CLK,       //      DM9000A Clock 25
MHz
        ////////////////////  Audio CODEC               ////////////////////////
        inout               AUD_ADCLRCK,    //      Audio CODEC ADC LR Clock
        input               AUD_ADCDAT,             //      Audio CODEC ADC
Data
        inout               AUD_DACLRCK,    //      Audio CODEC DAC LR Clock
        output                  AUD_DACDAT,             //      Audio CODEC
DAC Data
        inout               AUD_BCLK,       //      Audio CODEC Bit-Stream
Clock
        output                  AUD_XCK,    //      Audio CODEC Chip Clock
        ////////////////////  TV Devoder            ////////////////////////
        input   [7:0]   TD_DATA,                //      TV Decoder Data bus 8 bits
        input               TD_HS,              //      TV Decoder H_SYNC
        input               TD_VS,              //      TV Decoder V_SYNC
        output                  TD_RESET,       //      TV Decoder Reset
        //////////////////////    GPIO  ////////////////////////////////
```

```
        inout  [35:0] GPIO_0,                    //      GPIO Connection 0
        inout  [35:0] GPIO_1                     //      GPIO Connection 1
);

assign      LCD_ON          =       1'b1;
assign      LCD_BLON  =       1'b1;

//      Todos los puertos internos se convierten en tri-state
assign      DRAM_DQ         =       16'hzzzz;
assign      FL_DQ                   =       8'hzz;
assign      SRAM_DQ         =       16'hzzzz;
assign      OTG_DATA        =       16'hzzzz;
assign      SD_DAT          =       1'bz;
assign      ENET_DATA=      16'hzzzz;
assign      GPIO_0          =       36'hzzzzzzzzz;
assign      GPIO_1          =       36'hzzzzzzzzz;

//Salida de audio DAC
wire signed [15:0] audio_outL, audio_outR;
// Entrada de audio ADC
wire signed [15:0] audio_inL, audio_inR;

wire   AUD_CTRL_CLK;
wire   DLY_RST;
wire   I2C_END;
wire   NIOS_CLK;
wire   CLK;

assign      TD_RESET = 1'b1; // Permitir 27 MHz
assign      AUD_ADCLRCK = CLK;
assign      AUD_XCK = AUD_CTRL_CLK;
assign      AUD_DACLRCK = CLK;


// Interfaz entre el NIOS y el controlador FFT
wire [7:0]  fftaddr;
wire        fftstart;
wire        fftcomplete;
wire [15:0] fftcoeff;
wire [5:0]  fftlevel;
```

```
// Constantes del controlador FFT
parameter LEN   = 256;
parameter LBITS = 8;
parameter EBITS = 6;
parameter BITS  = 16;

wire lc;
wire [LBITS-1:0] sampleAddr;
wire [BITS-1:0] sample;

Reset_Delay r0 (
        .iCLK(CLOCK_50),
        .oRESET(DLY_RST)
);

audiopll p1 (
        .areset(~DLY_RST),
        .inclk0(CLOCK_27),
        .c0(AUD_CTRL_CLK)
);

I2C_AV_Config u3 (
        .iCLK(CLOCK_50),
        .iRST_N(KEY[0]),
        .I2C_SCLK(I2C_SCLK),
        .I2C_SDAT(I2C_SDAT)
);

niospll p2 (
        .inclk0(CLOCK_50),
        .c0(NIOS_CLK),
        .c1(DRAM_CLK)
);

NiosTimer cpu (
    .clk                    (NIOS_CLK),
    .reset_n                (KEY[0]),
    .in_port_to_the_iFFTCoeff       (fftcoeff),
    .in_port_to_the_iFFTComplete    (fftcomplete),
    .in_port_to_the_iFFTLevel       (fftlevel),
    .in_port_to_the_iKeys           (KEY[3:0]),
```

```
    .in_port_to_the_iSwitches       ({10'b0,SW[7:0]}),
    .in_port_to_the_niosclock       (NIOS_CLK),
    .in_port_to_the_reset           (KEY[0]),
        .in_port_to_the_lc                                  (lc),
    .out_port_from_the_oFFTAddress  (fftaddr),
    .out_port_from_the_oFFTStart    (fftstart),
    .out_port_from_the_oLEDG        ({1'b0,LEDG[7:0]}),
    .out_port_from_the_oLEDR        (LEDR[9:0]),
                //Señales LCD
    .LCD_E_from_the_lcd_0           (LCD_EN),
    .LCD_RS_from_the_lcd_0          (LCD_RS),
    .LCD_RW_from_the_lcd_0          (LCD_RW),
    .LCD_data_to_and_from_the_lcd_0 (LCD_DATA),
                //Señales SDRAM
    .zs_addr_from_the_sdram_0       (DRAM_ADDR),
    .zs_ba_from_the_sdram_0         ({DRAM_BA_1, DRAM_BA_0}),
    .zs_cas_n_from_the_sdram_0      (DRAM_CAS_N),
    .zs_cke_from_the_sdram_0        (DRAM_CKE),
    .zs_cs_n_from_the_sdram_0       (DRAM_CS_N),
    .zs_dq_to_and_from_the_sdram_0  (DRAM_DQ),
    .zs_dqm_from_the_sdram_0        ({DRAM_UDQM, DRAM_LDQM}),
    .zs_ras_n_from_the_sdram_0      (DRAM_RAS_N),
    .zs_we_n_from_the_sdram_0       (DRAM_WE_N)

);

FFTModule fftc (
        .iReset(~KEY[0]),
        .iStart(lc),
        .iStateClk(NIOS_CLK),
        .oSampAddr(sampleAddr),
        .iSamp(sample),
        .iReadAddr(fftaddr),
        .iReadClock(NIOS_CLK),
        .oPower(fftcoeff),
        .oExp(fftlevel),
        .oDone(fftcomplete)
);

MemoryModule memRAM (
        .iReset(~KEY[0]),
```

```
        .iStartLoad(fftstart),
        .iWriteClock(ramclk),
        .iSample(audio_inL),
        .iReadClock(NIOS_CLK),
        .iReadAddr(sampleAddr),
        .oValue(sample),
        .oLoadComplete(lc),
        .state(LEDR[17]),
        .oNum(temp),
        .oLED(LEDR[16:13])
);
wire ramclk;
wire [15:0] temp;
wire [15:0] temp1;

HexDigit h4  (HEX4,temp[3:0]);
HexDigit h5  (HEX5,temp[7:4]);
HexDigit h6  (HEX6,temp[11:8]);
HexDigit h7  (HEX7,temp[15:12]);
HexDigit1 h0 (HEX0,temp1[3:0]);
HexDigit1 h1 (HEX1,temp1[7:4]);
HexDigit1 h2 (HEX2,temp1[11:8]);
HexDigit1 h3 (HEX3,temp1[15:12]);

assign temp1 = LEDG[7] ? (16'hf4E5):((LEDG[0])?(16'hffd0):(16'hffff));

//Asignación de pines para Debug
assign audio_outL = audio_inL;
assign GPIO_0[0] =  CLK;
assign GPIO_0[1] = fftstart;
assign GPIO_0[2] = lc;
assign GPIO_0[3] = fftcomplete;
assign LEDG[8] = fftstart;
assign ramclk = (SW[17]) ? (~KEY[1]) : CLK;

endmodule

// Decodificar un dígito hexagonal para la visualización LED de 7 seg
module HexDigit(segs, num);
        input [3:0] num      ;              //El dígito hexadecimal que mostrará
        output [6:0] segs ;          //Segmentos de LED reales
```

```
        reg [6:0] segs ;
        always @ (num)
        begin
                case (num)
                                4'h0: segs = 7'b1000000;
                                4'h1: segs = 7'b1111001;
                                4'h2: segs = 7'b0100100;
                                4'h3: segs = 7'b0110000;
                                4'h4: segs = 7'b0011001;
                                4'h5: segs = 7'b0010010;
                                4'h6: segs = 7'b0000010;
                                4'h7: segs = 7'b1111000;
                                4'h8: segs = 7'b0000000;
                                4'h9: segs = 7'b0010000;
                                4'ha: segs = 7'b0001000;
                                4'hb: segs = 7'b0000011;
                                4'hc: segs = 7'b1000110;
                                4'hd: segs = 7'b0100001;
                                4'he: segs = 7'b0000110;
                                4'hf: segs = 7'b0001110;
                                default segs = 7'b1111111;
                endcase
        end

        endmodule

// Decodificar un dígito hexagonal para la visualización LED de 7 seg
module HexDigit1(segs, num);
        input [3:0] num     ;           //El dígito hexadecimal que mostrará
        output [6:0] segs ;         //Segmentos de LED reales
        reg [6:0] segs ;
        always @ (num)
        begin
                case (num)
                                4'h0: segs = 7'b1000000;
                                4'h1: segs = 7'b1111001;
                                4'h2: segs = 7'b0100100;
                                4'h3: segs = 7'b0110000;
                                4'h4: segs = 7'b0011001;
                                4'h5: segs = 7'b0010010;
                                4'h6: segs = 7'b0000010;
```

```
                    4'h7: segs = 7'b1111000;
                    4'h8: segs = 7'b0000000;
                    4'h9: segs = 7'b0010000;
                    4'ha: segs = 7'b0001000;
                    4'hb: segs = 7'b0000011;
                    4'hc: segs = 7'b1000110;
                    4'hd: segs = 7'b0101011;  //d will print n
                    4'he: segs = 7'b0000110;
                    4'hf: segs = 7'b1111111;   //f will blank
                    default segs = 7'b1111111;
            endcase
        end

        endmodule
s = 7'b1111001;
                    4'h2: segs = 7'b0100100;
                    4'h3: segs = 7'b0110000;
                    4'h4: segs = 7'b0011001;
                    4'h5: segs = 7'b0010010;
                    4'h6: segs = 7'b0000010;
                    4'h7: segs = 7'b1111000;
                    4'h8: segs = 7'b0000000;
                    4'h9: segs = 7'b0010000;
                    4'ha: segs = 7'b0001000;
                    4'hb: segs = 7'b0000011;
                    4'hc: segs =
```

Código fuente modulo convertidor de Huffman

```
/*
        Huffman Encoder (without pipelining)
        */
        //State variable names
        `define INIT 3'b111
        `define GET_DATA 3'b000
        `define BUILD_TREE 3'b001
        `define DECODE_TREE 3'b010
        `define SEND_SYMBOLS 3'b011
        `define SEND_CODE 3'b100
        `define SEND_LENGTH 3'b101
```

```
module huff_encoder(
                        input wire clock,
                    //Clock
                        input wire [bit_width:0]data_in,
            //Input data from another module
                        input wire data_enable,
            //This bit has to be high for data to be accepted
                        output reg [2*bit_width+2:0]data_out,
            //Output data from this module
                        output reg data_recv,
                    //This bit should remain high whenever data is being sent
                        output reg code_map_recv
                    //This bit should be high whenever code map is being sent
                        );

parameter bit_width = 7;
parameter col_length = 255;       //2 power bit width, no support for
2**bit_width in verilog 1995
parameter No_of_Data = 100;
reg [7:0]Prob_list[col_length:0];                       //Probability list
reg [7:0]temp2;
reg [bit_width:0]Sym_list[col_length:0];          //Symbols list
reg [bit_width:0]Sym,temp1;                                         //Symbol
holder variable
reg [0:2*bit_width+2]Code_list[col_length:0];          //Codes list
reg [bit_width:0]Code_length[col_length:0];            //Code lengths
reg [bit_width:0]Huff_list[col_length:0];          //List used to perform the
algorithm on
reg [bit_width:0]Pair_list[2*col_length+2:0];     //The pair list, an abstraction
for the tree concept. even - decode 0. odd - decode 1.
reg [2:0]state = `INIT;                                     //State variable
integer step = 0;                                                   //Number
of steps of tree building algorithm
reg [bit_width:0]pos,newpos = 0;                       //Variables to hold
values of positions in pair table
reg [bit_width:0]col = 'b0;                                 //Column length
reg [bit_width:0]Data[No_of_Data:0];
integer i= 32'h0;
integer j= 32'h0;
integer k= 32'h0;                                           //Loop variables
```

```
reg flag = 0;
        //Flag
integer pair_count= 0, sym_count = 0;
/*Steps for build_tree:
1)Add 2nd least + least probabilities
2)Add 2nd least and least in pair table (function add_pair does 2 and 3)
3)Remove least symbol from Huff_list
4)Push sort
Steps for decode_tree:
1)Search for each element from top.
2)If even, append symbol 0, else 1. Increment code length.
3)If 0, keep going. if 1, do pos - 1. Change variable and do as for 0.
*/
always @(posedge clock) begin
        case(state)


        `INIT: begin
        Sym_list[0] = 'b0;
        Prob_list[0] = 'b0;

        for(j=0;j<col_length;j=j+1) begin
        Code_list[j] = 'bz;
        Prob_list[j] = 'b0;
        Sym_list[j] = 'bz;
        Code_length[j] = 'b0;
        End


        data_out = 'bz;
        state = `GET_DATA;
        End


        `GET_DATA: begin
        if(data_enable) begin
                Data[i] = data_in;
                i=i+1'b1;

                        for(j=0;j<=col_length; j=j+1) begin
```

```
                    if(data_in == Sym_list[j]) begin
                        Prob_list[j] = Prob_list[j] + 1;

                        begin:SORT
                            for(k=j-1;k>=0;k=k-1) begin
                                if(Prob_list[k] <= Prob_list[j])
begin
                                    temp1 = Sym_list[j];
                                    temp2 = Prob_list[j];
                                    Sym_list[j] =
Sym_list[k];
                                    Prob_list[j] =
Prob_list[k];
                                    Sym_list[k] = temp1;
                                    Prob_list[k] = temp2;

                                    Huff_list[j] =
Sym_list[j];
                                    Huff_list[k] =
Sym_list[k];

                                end
                            end
                        end         //end of Sort
                        flag=1;
                    end    //End of if
                end        //End of for loop


                if(!flag) begin
                    Sym_list[col] = data_in;
                    Huff_list[col] = data_in;
                    Prob_list[col] = 'b1;
                    col = col+1;
                end

                flag= 0;

            if(i == No_of_Data) begin
            state = `BUILD_TREE;
```

```
                sym_count = col;
                //$display("col:",col);
                //for(i=0;i<col_length;i=i+1)
                //$display(Huff_list[i],"  ", Prob_list[i]);
                col = col -1 ;
                End
        End
        End


        `BUILD_TREE: begin
                code_map_recv = 0;
                data_recv = 0;
                if(col) begin                   //One step per cycle
                        Prob_list[col-1] = Prob_list[col] + Prob_list[col-1];
        //Added probabilities

                        Pair_list[step] = Huff_list[col-1];                     //Add in
pair table
                        Pair_list[step+1] = Huff_list[col];

                        col = col - 1;          //removing least symbol
                        pair_count = pair_count +2;

                        begin:SORT1
                                for(k=col-1;k>=0;k=k-1) begin
                                        if(Prob_list[k] < Prob_list[j]) begin
                                                temp1 = Huff_list[j];
                                                temp2 = Prob_list[j];
                                                Huff_list[j] = Huff_list[k];
                                                Prob_list[j] = Prob_list[k];
                                                Huff_list[k] = temp1;
                                                Prob_list[k] = temp2;
                                        end
                                end
                        end

                        step = step + 2;
                end
```

```
        else
            if(col == 0) begin
            state = `DECODE_TREE;
            //for(i=0;i<2*col_length;i=i+1)
            //$display(Pair_list[i]);
            //$display(sym_count, "  ",pair_count);
            i=0;
            j=0;

            Sym = Sym_list[0];
            end
        End


`DECODE_TREE: begin
        code_map_recv = 1;
        data_recv = 1;
        //One symbol per cycle decoding
        //i - symbol number, j - iteration for code

        if(Sym == Pair_list[j]) begin

            if(j%2 == 0) begin
                Code_list[i]= Code_list[i]<<1 | 'b0;
                j=j+2;
            end

            else begin
                Code_list[i]= Code_list[i]<<1 | 'b1;
                Sym = Pair_list[j-1];
                j=j+1;
            end

            Code_length[i] = Code_length[i] + 1;
        End

        else
            j=j+1;

        if(j>pair_count-1) begin
```

```
                        i=i+1;
                        j=0;
                        Sym = Sym_list[i];
                        end

                if(i==sym_count)     begin
                        state = `SEND_LENGTH;
                        //for(k=0;k<col_length;k=k+1)
                        //$display(Sym_list[k],"  ","%b",Code_list[k],"
",Code_length[k]);
                        i=0;
                End

        End
        `SEND_LENGTH: begin
        //send data in reverse order
                data_out = Code_length[i];
                i = i+1;

                if(i == sym_count) begin
                        state = `SEND_CODE;
                        i = 0;
                        end

                data_recv = 1;
                code_map_recv = 0;
        End

        `SEND_CODE: begin
                data_recv = 0;
                code_map_recv = 1;
                data_out = Code_list[i];
                i = i+1;

                if(i == sym_count) begin
                        state = `SEND_SYMBOLS;
                        i = 0;
                        end
                End
        `SEND_SYMBOLS: begin
```

```
                data_recv = 1;
                code_map_recv = 1;
                data_out = Sym_list[i];
                i = i+1;
                if(i == sym_count)
                        state = `GET_DATA;
                End

        endcase
end
Endmodule
```