UNIVERSIDAD AUTÓNOMA DE QUERÉTARO
FACULTAD DE INGENIERÍA

"Desarrollo de protocolo Ethernet en FPGA para el procesamiento digital de datos de cámara termográfica"

TESIS

Que como parte de los requisitos para obtener el grado de Maestro en Ciencias

PRESENTA:

Erick Manuel Lugo Álvarez

DIRIGIDO POR:

Dr. René de Jesús Romero Troncoso

Santiago de Querétaro, mayo de 2013.

Universidad Autónoma de Querétaro
Facultad de Ingeniería
Maestría en Mecatrónica

DESARROLLO DE PROTOCOLO ETHERNET EN FPGA PARA EL PROCESAMIENTO DIGITAL DE DATOS DE CÁMARA TERMOGRÁFICA

TESIS

Que como parte de los requisitos para obtener el grado de

Maestro en Ciencias Mecatrónica

Presenta:
Erick Manuel Lugo Álvarez

Dirigido por:
René de Jesús Romero Troncoso

SINODALES

Dr. René de Jesús Romero Troncoso
Presidente — Firma

Dr. Luis Morales Velázquez
Secretario — Firma

Dr. Luis Alberto Morales Hernández
Vocal — Firma

Dr. Miguel Trejo Hernández
Suplente — Firma

Dr. Carlos Rodríguez Doñate
Suplente — Firma

Dr. Aurelio Domínguez González
Director de la Facultad

Dr. Irineo Torres Pacheco
Director de Investigación y Posgrado

Centro Universitario
Querétaro, Qro.
Mayo de 2013
México

RESUMEN

Con el surgimiento de la tecnología FPGA (Field Programmable Gate Array, Arreglo de Compuertas Programables en Campo) el diseño de hardware se ha visto transformado de forma radical de la simple interconexión de componentes electrónicos a la reconfiguración electrónica por software de un circuito integrado especializado; el tamaño y velocidad de los FPGA son equiparables a los ASIC (Application-Specific Integrated Circuit, Circuitos integrados de Aplicación Especifica) pero los FPGA son más flexibles y su velocidad de procesamiento es mayor debido a su arquitectura paralela de interconexión de compuertas permitiendo así la ejecución de varios procesos de forma simultánea. Pese a sus virtudes los FPGA precisan contar con periféricos de comunicación que les permitan explotar plenamente sus cualidades de alta velocidad de procesamiento en aplicaciones que requieren la transferencia de grandes volúmenes de datos; la forma común de solventar este inconveniente es la incorporación de tarjetas de comunicación comerciales pero tienen la desventaja de ser poco adaptables a aplicaciones específicas además de generar costos significativos. Una alternativa a esta problemática es la creación de sistemas a la medida mediante el desarrollo núcleos de propiedad intelectual (IP Cores) de unidades de lógica que otorguen a los FPGA la capacidad de comunicación mediante un protocolo estándar de alta capacidad como lo es Ethernet, haciendo uso de sus prestaciones de velocidad y de confiabilidad, eliminando el uso de equipos comerciales y logrando de este modo independencia tecnológica. La aportación del presente trabajo es implementación de una aplicación Cliente/Servidor entre un FPGA y un Microcontrolador mediante el empleo de un Módulo Ethernet ENC28J60, logrando transferencias de datos a velocidades superiores a 100 Kbps en la generación de interfaces de usuario para la transmisión de datos de cámaras de visión termográficas, como una solución alternativa al procesamiento de imágenes aplicado al análisis termográfico de circuitos eléctricos y equipos mecánicos.

Palabras clave— protocolo Ethernet, FPGA, microcontrolador, aplicación Cliente/Servidor, análisis termográfico.

SUMMARY

With the emergence of the FPGA (Field Programmable Gate Array) technology hardware design has been radically transformed from a simple interconnection of electronic components to the electronic reconfiguration by software of a specialized integrated circuit; size and speed of FPGA are comparable to the ASIC (Application-Specific Integrated Circuit), but FPGA are more flexible and their processing speed is higher since they are composed of a parallel architecture of interconnection of gates thus allowing the execution of multiple processes simultaneously. Despite their virtues FPGA require communication peripherals that allow them to take advantage of their quality of high-speed processing in applications that require the transference of large volumes of data; the common way to overcome this inconvenience is the incorporation of commercial communication cards, but the disadvantage is that their adaptability is limited to certain applications as well as they generate high costs. An alternative to this problem is the creation of custom systems by developing cores of intellectual property (IP Cores) of units of logic which provide to the FPGA communication skills using a high capacity standard protocol such as Ethernet, making use of its speed and reliability benefits, eliminating the use of commercial equipment and thereby achieving technological independence. The contribution of this study is the implementation of a Client/Server application between an FPGA and a microcontroller through the employment of a module Ethernet ENC28J60, achieving the transference of data at speeds up to 100 Kbps in the generation of user interfaces for data transmission of thermal vision cameras, as an alternative to image processing applied to the thermographic analysis of electrical circuits and mechanical equipment.

Keywords— Ethernet protocol, FPGA, microcontroller, Client/Server application, thermographic analysis.

AGRADECIMIENTOS

Quiero agradecer plenamente a la vida ya que en todas las etapas de mi desarrollo me ha rodeado de personas excepcionales que en su conjunto son poseedoras de todas las virtudes. Es imposible citarlos a todos en unas cuantas líneas pero puedo decir con certeza, que no poseería ninguna fortaleza de no haberlos conocido y que debido a su ejemplo y su apoyo hacen que cada día exista dentro mí un deseo de ser alguien mejor. Todo lo que soy y lo que quiero ser se lo debo a ustedes.

A mis padres por enseñarme con su invaluable ejemplo, desde el día que nací hasta el día presente, que no existe problema tan grande que la fuerza de la familia no pueda vencer. Por velar cada minuto por mi bienestar, por estar siempre dispuestos a dar más de sí. No existen palabras para describir la admiración que siento por ustedes, para describir el valor con que se enfrentan a los problemas. Siempre serán una parte importante de mí.

A mí hermano por toda la ayuda que me ha dado desde que éramos niños, por su amistad, su generosidad y por todo el apoyo que mostró en nuestra familia en los momentos difíciles ya que sin su esfuerzo hubiera sido muy complicado concluir mis estudios de licenciatura. Espero que siempre estemos en contacto.

A mi novia Ana Laura por ser la inspiración para desear construir un futuro más grande, por tu cariño, comprensión y por la energía que posees que se ve reflejada en todo lo que haces. Es una de las alegrías más grandes de mi vida y un privilegio haberte conocido.

A mis compañeros de posgrado que más simples compañeros de aula los considero mis amigos. Fue un placer compartir y sortear hombro a hombro los obstáculos que nos presentó este recorrido. Mis mejores deseos en su vida y su carrera.

A todos los mis profesores que me otorgaron la oportunidad de compartir sus conocimientos conmigo, mostrándome campos de estudio que jamás había imaginado, aumentando con ello mi capacidad de análisis. Disfrute mucho entender un poco más como funciona todo lo que nos rodea.

Y finalmente a la Universidad Autónoma de Querétaro y al Consejo Nacional de Ciencia y Tecnología por brindarme la valiosa oportunidad realizar mis estudios de posgrado y concluir así una meta más en mi desarrollo profesional. Espero que este tipo de programas perduren y sigan extendiéndose, apoyando cada vez a más estudiantes en la formación de mejores personas.

"Seamos realistas y hagamos lo imposible."

Ernesto Guevara.

ÍNDICE

ÍNDICE DE FIGURAS

1. INTRODUCCION

La gran mayoría de los problemas y fallas que se generan dentro de la industria, ya sea de tipo mecánico, eléctrico o de fabricación, se encuentran precedidos por cambios de temperatura en los objetos. Es aquí donde nace la necesidad de contar con nuevas herramientas que puedan ayudar a eliminar, prevenir y minimizar el riesgo de fallas intempestivas y sus respectivas consecuencias, que, como es lógico pensar llevan consigo altos costos en reparación de equipos e instalaciones. El análisis termográfico es una técnica cada vez más utilizada en el ámbito industrial ya que es extremadamente útil en el mantenimiento preventivo, predictivo y correctivo.

El principio de funcionamiento se basa en una cámara termográfica mediante la cual se obtiene una imagen térmica o termograma, totalmente radiométrica que nos permite determinar la temperatura en cualquier punto del cuerpo de estudio. Su uso se ha extendido ampliamente por sus grandes ventajas ante los métodos tradicionales de medición de temperatura; entre estas ventajas se pueden mencionar que sus ensayos son no destructivos, es decir no dañan el objeto analizado, no es necesario el contacto físico, lo que resulta en una mayor seguridad para los operarios, además de que cambios mínimos en la temperatura pueden ser detectados rápidamente y con un alto grado de precisión.

Obtenido el termograma es posible detectar las anomalías térmicas que afectan a los cuerpos y que pueden ocasionar un mal funcionamiento, pero en un enfoque más moderno, un análisis termográfico va más allá de la captura de imágenes infrarrojas. Actualmente posterior a la captura de imágenes, se puede realizar un minucioso análisis por software de procesamiento digital, potencializando así las aplicaciones del termograma, que pueden ir desde la corrección automática según la emisividad específica de los cuerpos, eliminación de reflejos, hasta la creación de un reporte detallado.

El procesamiento digital de imágenes tiene como objetivo mejorar la calidad del estudio o facilitar la manipulación de información, aunque una limitante en esta técnica es que debido a la gran cantidad de datos que deben ser analizados y almacenados la velocidad de procesamiento debe ser igualmente alta.

Actualmente dicho procesamiento se lleva a cabo mediante aplicaciones comerciales que utilizan generalmente como plataforma de ejecución computadores personales, esto se ha vuelto un importante inconveniente, ya que por su limitada velocidad de procesamiento, es cada vez más complicado tratar de procesar volúmenes considerables de información en tiempos razonables.

El presente trabajo pretende como objetivo ofrecer una solución alternativa al problema de la velocidad de procesamiento de imágenes aplicado al análisis termográfico utilizando, primero, como medio de de almacenamiento un FPGA que por su capacidad y velocidad de procesamiento puede desempeñarse significativamente mejor que una computadora personal y segundo, como medio de enlace, una red de comunicación industrial de protocolo Ethernet, tratando de garantizar una plataforma de de intercambio de datos confiable, estable y veloz.

1.1 OBJETIVOS

1.1.1 OBJETIVO GENERAL

Desarrollar e implementar el protocolo Ethernet para la comunicación entre una cámara de visión termográfica y un FPGA aplicado al monitoreo de energía eléctrica y equipos mecánicos.

1.1.2 OBJETIVOS PARTICULARES

a) Desarrollar un algoritmo de comunicación que sea capaz de interpretar los datos de una red de comunicación SPI procedentes de un circuito contralado

de protocolo Ethernet mediante un FPGA para el procesamiento e intercambio de datos con la red.

b) Diseñar una tarjeta electrónica que permita hacer la interconexión de un puerto RJ45 de tipo Ethernet con el puerto de comunicación de un FPGA y un puerto SPI de un circuito controlador Ethernet con funciones Cliente/Servidor, para que funcione como el enlace de capa física entre los dispositivos conectados en la red.

c) Almacenar dentro de un FPGA la información en formato de imagen procedente de una cámara de visión termográfica en el monitoreo de energía eléctrica y sistemas mecánicos.

d) Transmitir bajo el protocolo Ethernet a una PC la información almacenada en el FPGA para su visualización.

1.2 JUSTIFICACIÓN

Los estudios de termográfica infrarroja consisten en tomar y analizar la energía térmica que emiten los cuerpos. Esta energía térmica pone de manifiesto posibles averías en equipos, previendo presentarse indeseables situaciones futuras como pueden ser desperfecto por deterioro, rotura de maquinaria o incluso equipamiento en estado crítico de falla inminente. Un análisis termográfico va mas allá de la captura de imágenes infrarrojas, posteriormente a la captura de imágenes se realiza un análisis por software, para corregir la emisividad de los cuerpos así como reflejos, para obtener una correcta interpretación de los datos, entre otras. Las aplicaciones son muy diversas dado que en la gran mayoría de las situaciones de falla de un sistema, las anomalías comienzan manifestándose como un incremento de temperatura en determinados puntos.

Los sistemas actuales de procesamiento de imágenes provenientes de cámara termográficas basan su funcionamiento en aplicaciones comerciales proporcionados por los mismos fabricantes de los dispositivos, que además de no

ser modificables para el usuario, repercuten en altos costos; dicho software se ejecuta dentro de ordenadores y esto representa una desventaja clara en el tiempo de ejecución del procesamiento, comparado con otros dispositivos como son los FPGA, esto debido a que un ordenador realiza el procesamiento de la información de manera serial, es decir espera a que un proceso sea terminado para continuar con el siguiente y así de manera sucesiva, por el contrario el FPGA tiene la capacidad de ejecutar todos los procesos de manera paralela lo que se traduce en una mayor velocidad de procesamiento.

A pesar de esta evidente ventaja de realizar el procesamiento digital de imágenes dentro de un FPGA, es necesario aún contar con un enlace confiable, un protocolo de comunicación que permita transferir a alta velocidad la información entre una cámara de visión termográfica y los FPGA.

En el presente, el protocolo de comunicación Ethernet se sitúa como el principal protocolo del nivel de enlace de comunicación de redes LAN (Local Area Network, Red de Área Local), por ser uno de los más rápidos y económicos, además de ser el más práctico ya que su uso se ha extendido en la mayoría de la aplicaciones tanto de telecomunicaciones como industriales.

Ethernet es un protocolo estándar adoptado por la IEEE (Institute of Electrical and Electronics Engineers, Instituto de Ingenieros Eléctricos y Electrónicos) y la ISO (International Organization for Standardization, Organización Internacional para la Estandarización), además, las tendencias tecnológicas permiten pronosticar que será un protocolo que seguirá utilizándose y desarrollándose durante muchos años más.

Debido a las circunstancias anteriormente expuestas surge la necesidad de diseñar una plataforma de núcleo de propiedad intelectual dentro de la Universidad Autónoma de Querétaro y contar así con independencia tecnológica para realizar la interconexión de los FPGA con diferentes dispositivos, para llevar a cabo el procesamiento de datos en un medio de intercambio de datos confiable, de alta inmunidad al ruido electromagnético, veloz, económico y flexible, además

de ser modificable para distintas aplicaciones, garantizando que seguirá vigente evitando así su pronta obsolescencia. El uso de cámaras termográficas utilizando como vía de enlace el protocolo Ethernet y apoyados de un FPGA como medio de procesamiento de datos, puede aumentar la velocidad de ejecución con respecto a los ordenadores usados actualmente, potencializando las aplicaciones que se ven limitadas por el procesamiento de una gran cantidad de información.

Es necesario, por lo tanto, desarrollar una tecnología fundamentada en FPGA aprovechando sus cualidades de procesamiento como DSP (Digital Signal Processor, Procesador de Señal Digital), que pueda trabajar en conjunto con cámaras de visión termográfica, comunicándose a través de un protocolo de comunicación confiable, que permita el correcto procesamiento y almacenamiento de imágenes arrojadas por una cámara de visión termográfica, de una manera más rápida y eficiente para llevar a cabo la tarea de monitoreo e inclusive la de control de un proceso.

1.3 ANTECEDENTES

Muchos son los factores que provocan que las conexiones eléctricas y mecánicas fallen, no solo por la sobre carga de un circuito, sino también por las vibraciones, la fatiga mecánica o simplemente el deterioro al transcurrir el tiempo, generados por las condiciones medioambientales en las cuales se mantienen. El análisis termográfico sirve para localizar todos estos daños, ya que en el caso de no repararse, estas fallas provocarán averías generalmente mayores y más costosas en plazos cortos y medianos.

Conforme se ha dado el desarrollo de la tecnología electrónica, en la aplicación de cámaras de visión infrarrojas más potentes y de mayor resolución, se ha ido a su vez incrementando la complejidad en el procesamiento de un número cada vez más grande de información que es entregada por estos dispositivos. Actualmente el procesamiento de esta información se ha centrado en el uso de ordenadores mediante aplicaciones de software comercial, que se ven

limitados por la velocidad de ejecución de los procesadores actuales resultando en tiempos de procesamiento largos. La aplicación de los FPGA para esta tarea puede ser indiscutible para dar una respuesta al problema del lento procesamiento de datos, ya que este dispositivo debido a su arquitectura interna puede procesar la información de forma paralela a velocidades mucho mayores.

Es en este punto donde se vuelve crucial el desarrollo de núcleos de propiedad intelectual dentro de los FPGA, que puedan interpretar protocolos de comunicación industrial para el intercambio, es decir, transmisión y recepción de datos; actualmente no existe protocolo comercial que mejore las prestaciones y desempeño del protocolo Ethernet, ya que es esté el más comúnmente utilizado por los equipos modernos, incluyendo a las cámaras de visión termográfica, por ser de ser el de mayor robustez, velocidad y además permitir la interconexión de múltiples dispositivos de forma simultánea.

Por la parte de desarrollo de protocolos de comunicación dentro de nuestra institución de educación superior, anteriormente se han realizado trabajos de investigación logrando desarrollar IP Cores de protocolos de comunicación estándar, mencionando puntualmente el protocolo USB (Universal Serial Bus, Bus Universal en Serie), que define los cables, conectores y algoritmos usados en un bus para conectar, comunicar y proveer de alimentación eléctrica entre ordenadores, periféricos y dispositivos electrónicos.

Hace algunos años, Morales (2007) logró implementar de manera exitosa en una tarjeta electrónica Xilinx® Spartan-3 (XC3S1000) la comunicación vía protocolo USB de un FPGA, para llevar a cabo el control de un intercambiador automático de herramientas, valiéndose de drivers generados mediante el software WinDriver® y el lenguaje de programación C++ Builder® como interfaz de control.

Fuera de esta casa de estudios, dentro del Instituto Politécnico Nacional, González (2008) realizó también un trabajo de desarrollo de IP Cores basado en un sistema de almacenamiento de datos de memoria SDRAM (Synchronous

Dynamic Random-Access Memory, Memoria Dinámica Síncrona de Acceso Aleatorio) en un FPGA, para implementar el control de las señales de acceso de memoria y comunicación USB, para la transferencia de datos desde y hacia una PC (Personal Computer, Computadora Personal). Su proyecto se encuentra fundamentado en el uso de controladores USB de la marca Cypress®, en concreto en el driver CY7C68013A (EZ-USB FX2LP), mientras que el almacenamiento de memoria se lleva a cabo con una memoria SDRAM (MT48LC16M16A2) de Micron®. La configuración y acceso de memoria la realizó también mediante una tarjeta Xilinx® Spartan-3. El código utilizado en el controlador es C.

Posteriormente Campos (2010) retoma el trabajo de Velázquez (2007) para el desarrollo de un IP Core de un controlador PID (Proportional Integral Derivative, Proporcional Integral Derivativo). El importante logro que representó el desarrollo de estas aplicaciones mediante el protocolo USB y sus posteriores implementaciones, se ve limitado en cuanto a velocidad de transmisión pero sobretodo porque es incapaz de permitir la conectividad simultánea entre diferentes dispositivos; cabe agregar que el protocolo USB no puede ser usado en aplicaciones industriales en campo por su alta sensibilidad al ruido electromagnético.

Un año antes Guzmán y Velazco (2009) implementaron un proyecto para monitoreo y control de un sistema de riego mediante una interfaz en Labview®, donde la adquisición de datos en campo se realizó mediante en una red Modbus plus® a través de una red Interbus®; esta interfaz de software lleva a cabo la ejecución del control mediante un ordenador utilizando su procesador y periféricos, sirviéndose del puerto Ethernet propio para realizar la intercomunicación dentro de una red en base en la integración de desarrollos comerciales existentes.

Internacionalmente se han realizado diferentes estudios para el desarrollo de la interconectividad de los FPGA mediante redes industriales. Lu (2002) diseñó algoritmos de comunicación TCP/IP (Transmission Control Protocol/Internet Protocol, Protocolo de Control de Transmisión/Protocolo de Internet) basados en la implementación de la tecnología CRC (Cyclic Redundancy Check,

Comprobación de Redundancia Cíclica), como una respuesta a una necesidad global para la interconexión de equipos electrónicos, al observar la problemática de la baja velocidad de procesamiento de dispositivos GPP (General Porpuse Processors, Procesadores de Propósito General). Este trabajo se fundamentó en describir las diferencias entre las conexiones de redes en la capa de nivel de transporte de tipo TCP y UDT (User Datagram Protocol, Protocolo de Datagramas de Usuario), arrojando como resultado la generación e implementación de un algoritmo de transferencia de datos de tipo TCP por presentar este último mayores ventajas como la detección cíclica de errores. Dicha implementación se realizó dentro de las plataformas Xilinx Virtex® y Virtex II®.

Herrmann et al. (2009) realizaron exitosamente el desarrollo de un IP Core para la transferencia de datos entre un FPGA y una red Ethernet de estándar abierto UDT/IP, logrando velocidades de transferencia de 1960 Mbps. Esta vez la implementación se realizó en una tarjeta Xilinx Espartan 3E®, trabajando sobre las capas de transporte, red y enlace del modelo OSI (Open Systems Interconnection, Interconexión de Sistemas Abiertos). Este tipo de estudios demuestra la alta capacidad de procesamiento de un FPGA en conjunto a una red industrial de tipo Ethernet.

Paralelamente se realizaron otros trabajos enfocados en la creación de tarjetas de comunicación de tipo NIC (Network Interface Card, Tarjeta de Interface de Red), para la implementación de FPGA bajo el protocolo Ethernet. Tinoosh (2004) presentó el diseño y la evaluación de una NIC Gigabit/Ethernet utilizando FPGA, además de incluir memorias de tipo SDRAM y SODIMM (Small Outline Dual In-line Memory Module, Pequeño Módulo de Memoria Dual en Línea), para agregar servicios de red. El trabajo fue implementado mediante un decodificador Ethernet/PCI Intel LTX1000® y una tarjeta Xilinx Virtex II® además de agregar periféricos diversos como RS-232 (Recommended Standard 232, Estándar Recomendado 232). Este sistema presentó la cualidad de almacenar la información a una tasa alta de velocidad de transferencia, proveniente de una red

Ethernet para su posterior procesamiento, valiéndose de la integración de diversos dispositivos comerciales pero trabajando únicamente sobre la capa de transporte.

Actualmente nuevos trabajos se han enfocado al estudio de la interconectividad de la red Ethernet en su estándar TCP/IP pero no solo en las capas de transporte y red del modelo OSI sino que también se estudia la comunicación en la capa de aplicación, que es responsable de poder enviar y recibir datos a través de Internet. Ariza y Javier (2010) lograron desarrollar un servidor Web dentro de un FPGA; el sistema permite monitorear y controlar variables conectadas a un dispositivo físico desde una página Web por medio de un navegador comercial desde un ordenador conectado a Internet. Como es lógico pensar de esta función se puede desprender una gran cantidad de posibles aplicaciones, como la ejecución y monitoreo de procesos de forma remota. La base fundamental de este proyecto fue la herramienta de diseño Altium Designer® y la plataforma de implementación física es el módulo Nanoboard® sobre un dispositivo Xilinx Spartan 3®.

Así mismo han sido varios los trabajos que han tratado de optimizar el procesamiento del análisis termográfico, mediante el uso y aplicación de procesamiento de imágenes aunados a un sistema de adquisición de datos. Hashimoto (1984) realizó el análisis de imágenes térmicas aplicado a la botánica, específicamente para el caso de estudio del estrés que sufren los girasoles a la falta de agua, haciendo uso del puerto paralelo de un ordenador como medio de recepción de datos y realizando mediciones a intervalos de dos minutos aproximadamente. Así mismo Hernández (2004) desarrolló un sistema para la clasificación de granos de café, según su etapa de maduración, utilizando una ANN (Artificial Neural Network, Red Neuronal Artificial). Como herramienta de clasificación se utilizaron estructuras de tipo perceptrón multi-capa. La estructura fue diseñada e implementada sobre C++, usando el algoritmo de aprendizaje con retro-propagación de error. Esta estructura fue implementada sobre una tarjeta Altera Flex 10K®. Pérez (2005) utilizó un FPGA para el tratamiento estereoscópico de imágenes, como una respuesta a la deficiencia de potencia de cálculo en

aplicaciones de monitoreo en secuencias de imágenes en tiempo real con el uso de una de tarjeta Altera APEX 20K400E®. Por su parte Loheiden et al. (2006) estudiaron el cambio climático regional y la relación que existe con la falta de agua en el subsuelo, demostrando la importancia de mantener en equilibrio los ecosistemas; este estudio se llevo a cabo mediante la captura periódica de imágenes termográficas, durante lapsos de algunos años. Por su parte Barranco et al. (2010) presentaron un entorno de software que actuó como interfaz entre la captación de imágenes a través de cámaras Web y una placa de coprocesamiento conectada a un ordenador a través del bus PCI. Dicha placa extrajo las características primitivas de visión como son la energía, orientación, fase, disparidad y estimación de flujo óptico en tiempo real, obteniendo mediante una aplicación de software los resultados procesados para visualizarlos o almacenarlos. El objetivo de su proyecto fue encontrar herramientas que optimizaran las velocidades de procesamiento de los ordenares convencionales en el procesamiento de imágenes. Finalmente Martin et al. (2010) diseñaron un IP Core en una tarjeta Xilinx Virtex-5®, mediante comunicación PCI para la adquisición de imágenes termográficas, para el monitoreo de piezas industriales sometidas a cortes por plasma como un método para diagnosticar fallas como fatigas en los elementos.

2. REVISIÓN DE LITERATURA

2.1 ESTADO DEL ARTE

En sus orígenes y durante varias décadas, la mayoría de los fabricantes de equipos electrónicos desarrollaron protocolos de comunicación basándose únicamente en las necesidades de interconexión de sus propios dispositivos, con el pasar del tiempo y en un mundo cada vez más globalizado y cambiante, las necesidades de interconexión fueron creciendo a tal grado que fue necesario crear protocolos estandarizados donde se definieran reglas comunes que permitieran el intercambio de datos entre dispositivos de diferentes fabricantes, ya que compartir la información se ha vuelto una necesidad preponderante.

Los protocolos de comunicación hoy día deben cumplir no solo la tarea de permitir la transferencia de información entre diversos equipos, ahora debido al tráfico masivo de datos, es necesario contar con protocolos más veloces, que puedan disminuir significativamente los tiempos de transmisión y recepción, deben ser a la vez más confiables, los errores producidos durante la transferencia deben minimizarse y en caso de existir deben ser corregidos para evitar la pérdida de información, además deben buscar ser inmunes a las perturbaciones medioambientales.

Día a día nacen nuevas aplicaciones para las telecomunicaciones, que se ven favorecidas no solo por la estandarización de las redes de comunicación sino también a la investigación y creación de dispositivos semiconductores y tarjetas electrónicas, que cada vez pueden procesar volúmenes mayores de información a la vez de poder alcanzar velocidades de ejecución cada vez mayores.

Es en este punto donde podemos ver, que dentro de las múltiples aplicaciones para el intercambio de información electrónica y la gran capacidad de procesamiento de los dispositivos semiconductores de avanzada, existe la posibilidad de generar nuevas técnicas, que puedan solucionar la demanda de

procesamiento de imágenes, de formas cada vez más veloces y confiables, en la aplicación de análisis termográficos de mayor complejidad.

2.2 ANALISIS TERMOGRÁFICO

La termografía infrarroja es una poderosa herramienta para realizar el mantenimiento preventivo a herramientas y equipos, no obstante la toma de muestras ha sido, hasta hace algunos años, muy costosa y poco accesible a la mayoría de sectores interesados. En los últimos años esta tecnología se ha tornado más viable económicamente y empiezan a aparecer numerosas aplicaciones que explotan sus ventajas. Con ello se pueden analizar partes sometidas a estrés mecánico, térmico y eléctrico, principalmente en motores de vehículos y en equipos industriales, con el fin de establecer de manera rápida y confiable los puntos en donde es necesario revisar cuidadosamente los aparatos, para impedir daños futuros, conocido como mantenimiento preventivo, así como disminuir las secciones de mantenimiento correctivo. Se presentan varios casos en los que la imagen térmica resulta ser un factor fundamental, para obtener información de la magnitud y localización del daño presente en el instrumento analizado.

Debido a las exigencias de competitividad y eficiencia a las que se ven sometidas diferentes empresas, se observa la necesidad de desarrollar nuevas herramientas que permitan diagnosticar el estado de los instrumentos, para así ejercer un control sobre ellos, aumentando su disponibilidad, y reduciendo las fallas intempestivas, alcanzando así la optimización de la calidad y la reducción de costos de mantenimiento, potenciando así el mantenimiento predictivo.

La termografía provee una técnica no muy costosa para reunir datos necesarios, para determinar pérdidas de temperatura, debido a que en la industria la cuantificación de las pérdidas de calor es de relevante importancia, debido a la enorme cantidad de energía consumida en los procesos mecánicos, térmicos y eléctricos y al alto costo subsecuente de la energía desperdiciada.

De modo general, la medición de flujos térmicos involucra la medición de temperatura, siendo necesario entonces un dispositivo que pueda desplegar un cambio de color en un pequeño rango de temperatura, haciéndose necesaria la utilización la TI (Termografía Infrarroja), que permite la realización de un correcto mapeo superficial de temperatura aun en presencia de altos gradientes de temperatura espacial del flujo térmico.

La TI básicamente incluye una cámara infrarroja y un computador. El núcleo de la cámara es el detector infrarrojo, el cual absorbe la energía IR (Infrared Energy, Energía Infrarroja), emitida por el objeto, cuya temperatura superficial está siendo medida, convirtiendo esta en un voltaje o una corriente eléctrica. Algunos objetos emiten energía proporcional a la temperatura de su superficie. Sin embargo, la energía detectada por el detector IR depende del coeficiente de emisividad de la superficie que se encuentra bajo medición.

Puesto que la TI es una técnica de no contacto, la radiación IR necesita viajar cierta distancia desde el objeto a ser medido hasta el aparato de medición, a través de un medio con propiedades infra-ópticas que pueden afectar el resultado de la medición. En la mayoría de los casos, este medio es el aire, pero también pueden ser otros materiales. En el caso del aire, este tiene muchos componentes, tales como el vapor de agua y el dióxido de carbono que afectan la transmitancía IR.

El nivel de transmitancía del aire depende de la longitud de onda. Los detectores IR, generalmente realizan las mediciones en dos ventanas o rangos de trabajo, la banda de 8 a 12 µm llamada LW (Long Wave, Onda Larga) y la banda entre 3 y 5 µm, denominada como SW (Short Wave, Onda Corta). La figura número uno pone en manifiesto como queda distribuida las diferentes longitudes de onda dentro del espectro electromagnético. Se indican también los puntos donde trabajan las ondas cortas y largas de los detectores utilizados por las cámaras infrarrojas.

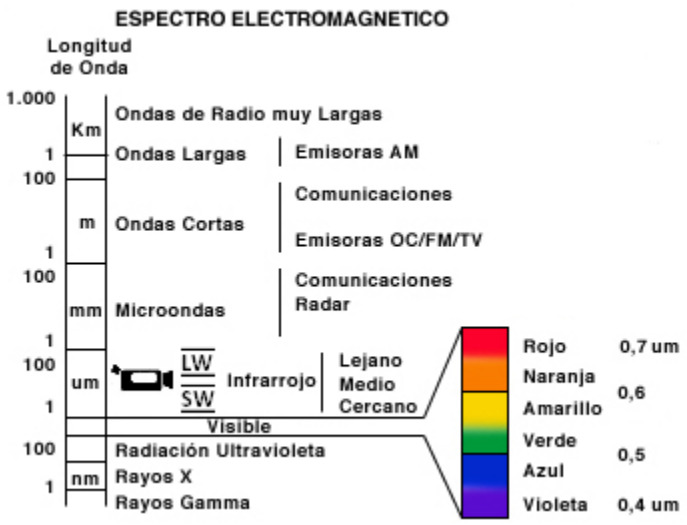

Figura 1. División del espectro electromagnético.

Fuente: http://www7.uc.cl/sw_educ/geo_mar/html/glosario.html

El equipo empleado en este método de inspección es la cámara termográfica, que registra la emisión natural de radiación infrarroja procedente de un objeto y genera una imagen térmica. La cámara termográfica se sitúa delante del objeto a inspeccionar para recibir la energía infrarroja emitida. Esa energía es la suma de tres componentes:

a) La energía infrarroja, proveniente del objeto.

b) La energía reflejada por dicho objeto.

c) La energía emitida por el ambiente (Muñoz et al., 2009).

2.2.1 ADQUISICIÓN

La figura número dos ilustra el proceso de adquisición de imágenes por TI y puede resumirse de la siguiente manera, primeramente un contraste térmico es producido en la superficie de la muestra bajo inspección. En general, este contraste se produce tras el envío de un frente de calor, que es el enfoque

comúnmente adoptado, señalando que la utilización de un frente frío es igualmente válido. Esto puede lograrse ya sea en régimen transitorio, con un impulso de energía o en régimen permanente, con el envío de ondas periódicas, dependiendo de la aplicación.

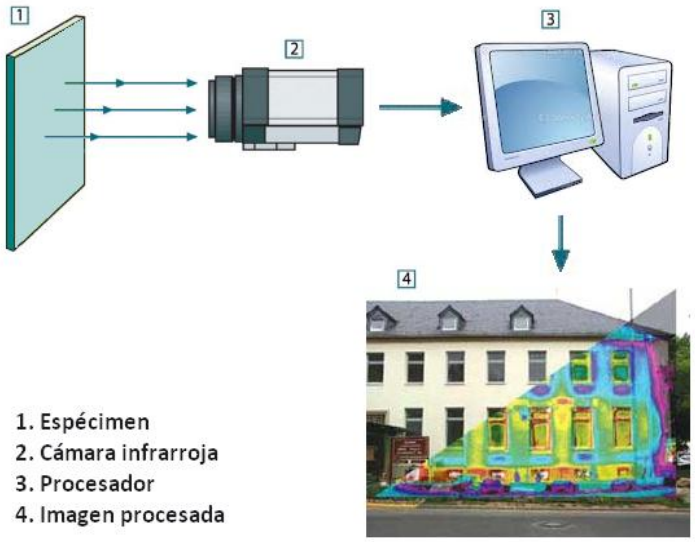

Figura 2. Sistema de adquisición de imágenes por Termografía Infrarroja. Fuente: Análisis de imágenes en Termografía Infrarroja, (Ibarra et al., 2005).

De igual manera, la adquisición de imágenes puede realizarse en modo reflexión, mediante el reflejo ocasionado por el espécimen o en modo transmisión, atravesando el mismo. En general, el modo reflexión es utilizado para defectos internos relativamente cercanos a la superficie, mientras que el modo transmisión lo es para defectos profundos, siempre y cuando se tenga acceso a los dos lados del objeto. Una vez que el calor entra en contacto con la superficie del espécimen, la energía es transmitida por conducción a su interior.

El principio de detección de defectos se basa en el hecho de que el frente caliente se propaga homogéneamente a través del material, salvo en presencia de un cuerpo con propiedades térmicas diferentes lo que determina un defecto. Cuando el frente hace contacto con este cuerpo, se produce una atenuación o un incremento, dependiendo de las propiedades térmicas del material y del defecto

en la conducción de calor, que tiene una relación directa con el tiempo de propagación del frente. Esto se refleja en la superficie de la muestra por un aumento o disminución de temperatura, que indica la presencia de un defecto y que puede ser detectado con la ayuda de una cámara infrarroja.

Existe un gran número de fuentes de degradación de la señal infrarroja, tales como ruido ligado al equipo de adquisición, ruido eléctrico, calentamiento no uniforme debido a imperfecciones de la fuente de energía, ruido térmico, diferencias de emisividad e irregularidades en la superficie de la pieza inspeccionada, deficiencia estructural, reflexiones provenientes de otros objetos y atenuación atmosférica. De este modo, una vez que las imágenes han sido adquiridas diferentes técnicas de tratamiento pueden ser utilizadas para realzar el contraste, detectar la presencia de defectos y caracterizarlos (Ibarra et al., 2005).

2.3 FPGA

Son la última generación de dispositivos de lógica programable, simplemente son matrices de puertas eléctricamente programables que contienen múltiples niveles de lógica; se basan en una cantidad muy grande de celdas lógicas, muy elementales. Mientras más pequeñas resulten las celdas, mayor aprovechamiento se puede tener de las mismas, aunque los retardos en la conducción pueden aumentar.

Los FPGA se caracterizan por altas densidades de puerta, alto rendimiento, un número grande de entradas y salidas definibles por el usuario, un esquema de interconexión flexible y un entorno de diseño similar al de matriz de puertas. No están limitadas a la típica matriz AND-OR. Las celdas básicas de un FPGA son mucho más simples que las macro-celdas de un PLD (Programmable Logic Device, Dispositivo Lógico Programable), y dependiendo del fabricante de los circuitos, puede cambiar su constitución. Cada proveedor y cada familia de FPGA ofrecen una versión diferente de la celda básica y de modo fundamental solo se presenta un ejemplo típico de las mismas.

Todo FPGA tiene tres grupos de celdas básicas que son:

a) Celda lógica
b) Celda de entrada/salida
c) Celda de distribución

La celda lógica es la que existe en mayor cantidad y son pequeños bloques lógicos de compuertas programables. Las celdas de entrada/salida están dedicadas a proporcionar la interconectividad entre el FPGA y el exterior. Finalmente, las celdas de distribución se encargan de manejar las señales de reloj hacia dentro del circuito.

Al contrario de lo que ocurre con los PLD, la interconectividad en un FPGA está restringida a celdas adyacentes exclusivamente. Con esto se permite un mucho menor consumo de potencia y por consiguiente, un mayor nivel de integración (Guitierrez et al., 2008). El diagrama a bloques de la interconectividad típica en un FPGA se muestra en la figura tres.

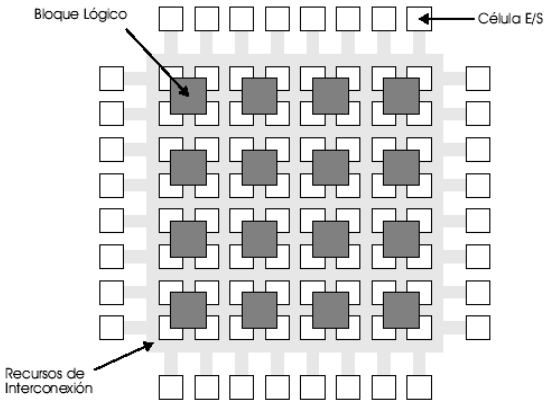

Figura 3. Estructura general de un FPGA. Fuente: Aplicación y soluciones para lógica programable (FPGA), Gutiérrez et al. (2008).

2.3.1 PRINCIPALES BENEFICIOS DE TECNOLOGÍA FPGA

Rendimiento. Aprovechando del paralelismo del hardware, los FPGA exceden la potencia de cómputo de los DSP, rompiendo el paradigma de

ejecución secuencial y logrando más operaciones por cada ciclo de reloj. El controlar entradas y salidas (E/S) a nivel de hardware ofrece tiempos de respuesta más rapidos y funcionalidad especializada que coincide con los requerimientos de una aplicación.

Tiempo en llegar al mercado. La tecnología FPGA ofrece flexibilidad y capacidades de rápido desarrollo de prototipos, para enfrentar los retos de que un producto se libere tarde al mercado. Se puede probar una idea o un concepto y verificarlo en hardware sin tener que pasar por el largo proceso de fabricación, por el que pasa forzosamente un diseño personalizado de ASIC. Posteriormente se puede implementar cambios y realizar iteraciones de un diseño FPGA en cuestión de horas en vez de semanas. También existe hardware comercial listo para usarse con diferentes tipos de E/S ya conectados a un chip FPGA programable por el usuario. El aumento en disponibilidad de herramientas de software de alto nivel disminuye la curva de aprendizaje con niveles de abstracción. Estas herramientas frecuentemente incluyen importantes núcleos IP y funciones pre-construidas, para control avanzado y procesamiento de señales.

Precio. El precio de la ingeniería no recurrente de un diseño personalizado ASIC excede considerablemente al de las soluciones de hardware basadas en FPGA. La fuerte inversión inicial de los ASIC es fácilmente justificable para los fabricantes de equipos originales que producen miles de chips por año, aunque muchos usuarios finales necesitan la funcionalidad de un hardware personalizado solo para decenas o cientos de sistemas en desarrollo. La misma naturaleza cambiante de la tecnología implica que no hay existe precio de fabricación fijo a largo plazo por lo que los requerimientos de un sistema van cambiando con el tiempo. El precio de cambiar los diseños FPGA es insignificante al compararlo con el precio de implementar cambios en un ASIC antes de su lanzamiento.

Fiabilidad. Mientras que las herramientas de software ofrecen un entorno de programación, los circuitos de un FPGA son una implementación segura de la ejecución de un programa. Los sistemas basados en procesadores frecuentemente implican varios niveles de abstracción para auxiliar a programar

las tareas y compartir los recursos entre procesos múltiples. El software a nivel driver se encarga de administrar los recursos de hardware y el sistema operativo administra la memoria y el ancho de banda del procesador. El núcleo de un procesador sólo puede ejecutar una instrucción a la vez y los sistemas basados en procesadores están siempre en riesgo de que sus tareas se obstruyan entre sí. Los FPGA, que no necesitan sistemas operativos, minimizan los retos de fiabilidad con ejecución paralela y hardware preciso dedicado a cada tarea.

Mantenimiento a largo plazo. Como se mencionó anteriormente, los chips FPGA son actualizables en campo y no requieren el tiempo y el precio que implica rediseñar un ASIC. Los protocolos de comunicación digital por ejemplo, tienen especificaciones que podrían cambiar con el tiempo, y las interfaces basadas en ASIC podrían causar retos de mantenimiento y en la capacidad de actualización. Los chips FPGA, al ser reconfigurables, son capaces de mantenerse al tanto con modificaciones a futuro que pudieran ser necesarias. Mientras el producto o sistema se va desarrollando, se pueden implementar mejoras funcionales sin la necesidad de invertir tiempo rediseñando el hardware o modificando el diseño de la tarjeta (Thompson, 2004).

2.4 ETHERNET

El uso de Ethernet en plantas industriales se está extendiendo, entre otras razones, por su buena relación prestación-costo y por su capacidad para soportar el uso de tecnologías de fibra óptica, cables eléctricos y comunicaciones inalámbricas en un único sistema. Otra ventaja es que las tecnologías TCP/IP asociadas a Ethernet proporcionan una infraestructura de red que se puede gestionar de forma unitaria. Esto racionaliza el despliegue y mantenimiento de la infraestructura y consigue importantes ahorros en la formación y en el suministro de repuestos.

Un requisito típico del mundo de la industria es la respuesta de control en tiempo real. Si las soluciones de comunicación afectan a un bucle de control, el

tiempo de respuesta es vital. El retardo admisible en el tiempo de respuesta está determinado por las leyes físicas o químicas que rigen el proceso objeto de control. Las soluciones de comunicación han de responder a esta variedad requisitos, sea con una única solución o combinando varias tecnologías (Hansen, 2006).

2.4.1 EL ESTÁNDAR ETHERNET

El sistema de Ethernet incluye cuatro bloques de construcción, que cuando son combinados permiten que este modelo pueda trabajar, los cuales son:

a) La trama de Ethernet, el cual estandariza un set de bits para ser usados por el sistema.

b) El protocolo de control de acceso medio, el cual consiste en un set de reglas unidas a la interfaz de Ethernet, la cual permite a múltiples computadoras acceder al canal Ethernet compartido en una manera efectiva.

c) Los componentes de señalización, los cuales consisten en la estandarización de los dispositivos electrónicos, que se utilizarán para enviar y recibir señales en un canal Ethernet.

d) El medio físico, el cual consiste en los cables y cualquier otro tipo de hardware utilizado para transportar las señales entre los equipos de la red.

2.4.2 TRANSMISIÓN Y TRAMA DEL PROTOCLO ETHERNET

El núcleo del sistema Ethernet radica en su trama de datos, el cual es utilizado para transportar la información entre dispositivos. La trama Ethernet consiste en un set de bits organizados en algunos campos, estos campos incluyen campos de dirección, un campo de datos que puede soportar 46 hasta 1500 Bytes de carga útil y un campo que chequea la integridad de los bits en la trama, para

asegurar que la trama fue recibida intacta (Rubio, 2009). La distribución de la trama Ethernet y su longitud puede ser observada en la figura número cuatro y se describen a continuación:

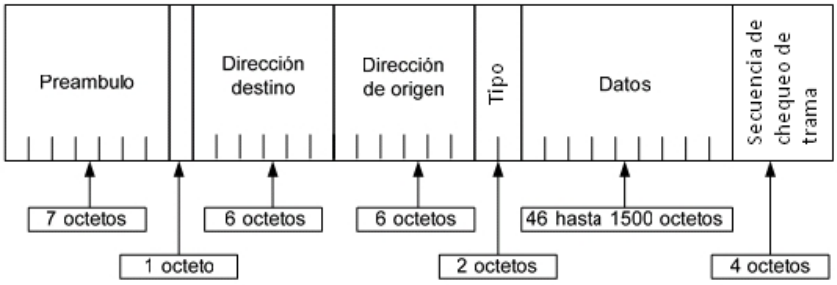

Figura 4. Formato de la trama Ethernet. Fuente: Arquitectura estándar y principios del protocolo Giga Ethernet, (Rubio, 2009).

a) Preámbulo (8 bytes). Se utiliza para sincronizar los relojes del emisor y del receptor que miden la duración de los bits del marco.

b) Dirección física destino (6 bytes). Es la dirección del adaptador destino.

c) Dirección física fuente (6 bytes). Es la dirección del adaptador origen.

d) Tipo (2 bytes). Identifica el protocolo usado en la capa de red (IP, Novell IPX, AppleTalk, etc.).

e) Datos (entre 46 y 1.500 bytes). Es el paquete de datos transportados. La capa de red se encarga de la segmentación y del relleno con ceros si estos fueran necesarios.

f) CRC (4 bytes). Sirve para detectar errores de transmisión (González, 2009).

2.5 MODELO OSI

Para simplificar la comunicación entre programas de aplicaciones de distintos equipos, se definió en 1984 el Modelo OSI por la ISO, el cual especifica

siete distintas capas de abstracción. Con ello, cada capa desarrolla una función específica con un alcance definido. Siguiendo el esquema de este modelo se crearon numerosos protocolos. Algunas veces se agrupan las capas de nivel de presentación, nivel de sesión y nivel de transporte en una sola que se denomina como esta última, para el caso concreto de redes TCP/IP como Internet. La figura número cinco muestra relación entre una red de tipo TCP/IP y su equivalente en el modelo OSI por sus niveles de abstracción.

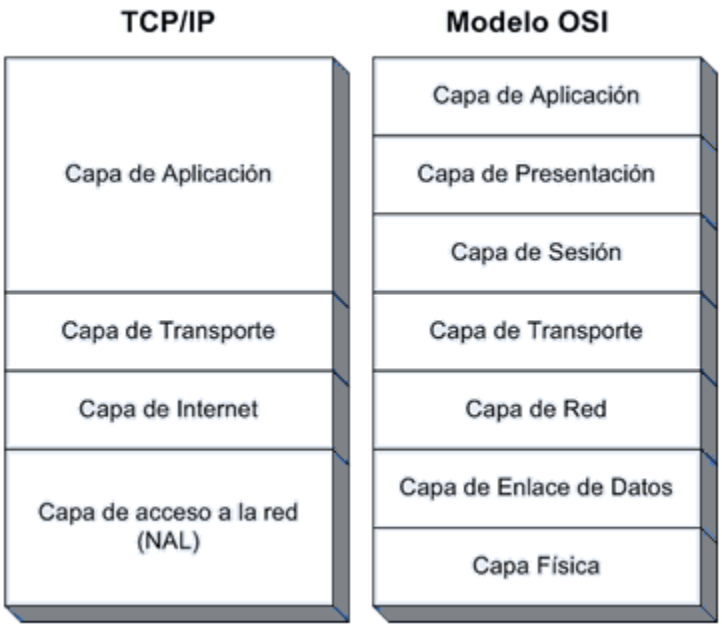

Figura 5. Niveles de abstracción o capas. Fuente: Informática: “Redes e Internet”, Arteaga (2001).

2.5.1 COMETIDO POR CAPA

a) Capa aplicación. Brinda los servicios de comunicación a los usuarios. Es una caja de herramientas normalizadas. Esta capa debe poseer protocolos que sean capaces de crear un terminal virtual de red abstracta, la cual debe realizar la adaptación de los diferentes programas de aplicaciones que poseen las máquinas de una red, con el fin de lograr la compatibilidad de las mismas.

b) Capa presentación. Se hace cargo de las diferentes sintaxis abstractas o de transferencia, facilitando el trabajo de las entidades de la capa aplicación, así también como de la semántica de los datos intercambiados. Sus servicios incluyen las conversiones de código y de formatos de datos así como la compresión y la encriptación de los datos.

c) Capa sesión. Permite establecer una relación entre dos aplicaciones, organizar y sincronizar el diálogo, permitiendo un intercambio full duplex, semiduplex o simplex. Si el tráfico es en un solo sentido a la vez, esta capa puede ayudar a llevar el control de los turnos. Un servicio relacionado es el manejo de fichas, ya que para algunos protocolos es esencial que dos máquinas no intenten la misma operación al mismo tiempo, para ello la capa sesión otorga fichas que se pueden intercambiar. Solo el lado que posea la ficha podrá efectuar la operación. Gestiona las modalidades de recuperación en caso de incidente.

d) Capa transporte. Se ubica en la frontera de las capas orientadas a transmisión y a tratamiento. Su función es ofrecer un servicio constante para las entidades de sesión, independientemente de la QoS (Quality of Service, Calidad del Servicio) de la red, asegurando un servicio punto a punto. Crea una conexión de red distinta para cada conexión de transporte que requiera la capa sesión. Si el volumen de transmisión es alto, esta capa puede crear múltiples conexiones de red, dividiendo los datos entre las conexiones o puede multiplexar varias conexiones de transporte en la misma conexión de red para reducir el costo, el multiplexado debe ser transparente a la capa sesión.

e) Capa de red. Controla que en la subred no se encuentren presentes demasiados paquetes a la vez, formando los cuellos de botella. El software debe contar cuántos paquetes o caracteres o bits envía cada cliente para producir información de facturación. La capa sesión debe lograr la comunicación entre redes heterogéneas. En las redes de difusión el ruteo es simple y esta capa con frecuencia es delgada o incluso inexistente.

f) Capa enlace de datos. Transmite datos sin error, sin duplicación, sin pérdida entre sistemas adyacentes. Enmascara a las capas superiores de las imperfecciones de los medios de transmisión utilizados. Toma un medio de transmisión en bruto y lo transforma en una línea que parezca libre de errores de transmisión no detectados en la capa de red.

g) Capa física. Se hace cargo de la transmisión de series de bits sobre el medio físico de interconexión, brinda las funciones de comando de los circuitos de datos. (Bísaro et al., 2005).

2.6 MODELO CLIENTE/SERVIDOR

Desde el punto de vista funcional, se puede definir el modelo Cliente/Servidor como una arquitectura distribuida que permite a los usuarios finales obtener acceso a la información, en forma transparente aún en entornos multiplataforma. En el modelo Cliente/Servidor, el Cliente envía un mensaje solicitando un determinado servicio a un Servidor, hace una petición, y este envía uno o varios mensajes con la respuesta, es decir provee el servicio. Si esto se aplica tanto a Clientes como Servidores se entiende que la forma estándar de aplicación y uso de sistemas Cliente/Servidor es mediante la explotación de las PC a través de interfaces gráficas de usuario. Usualmente la mayoría del trabajo pesado se hace en el proceso llamado Servidor y los procesos Cliente sólo se ocupan de la interacción con el usuario.

En esta arquitectura la capacidad de proceso está repartida entre los Clientes y los Servidores, aunque son más importantes las ventajas de tipo organizativo debidas a la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita y clarifica el diseño del sistema. La separación entre Cliente y Servidor es una separación de tipo lógico, donde el Servidor no se ejecuta necesariamente sobre una sola máquina ni es necesariamente un sólo programa. Los tipos específicos de Servidores incluyen los Servidores Web, los Servidores de archivo, los Servidores del correo, etc.,

mientras que sus propósitos varían de unos servicios a otros, la arquitectura básica seguirá siendo la misma. Una disposición muy común son los sistemas multicapa en los que el Servidor se descompone en diferentes programas que pueden ser ejecutados por diferentes computadoras, aumentando así el grado de distribución del sistema. La arquitectura Cliente/Servidor sustituye a la arquitectura monolítica en la que no hay distribución, tanto a nivel físico como a nivel lógico.

2.6.1 RED CLIENTE/SERVIDOR

Es aquella red de comunicaciones en la que todos los Clientes están conectados a un Servidor, en el que se centralizan los diversos recursos y aplicaciones con que se cuenta y que los pone a disposición de los Clientes cada vez que estos son solicitados. Esto significa que todas las gestiones que se realizan se concentran en el Servidor, de manera que en él se disponen los requerimientos provenientes de los Clientes que tienen prioridad, los archivos que son de uso público y los que son de uso restringido, los archivos que son de sólo lectura y los que, por el contrario, pueden ser modificados. Este tipo de red puede utilizarse conjuntamente en caso de que se esté utilizando en una red mixta.

2.6.2 CLIENTE

Es el que inicia un requerimiento de servicio. El requerimiento inicial puede convertirse en múltiples requerimientos de trabajo a través de redes LAN o WAN. La ubicación de los datos o de las aplicaciones es totalmente transparente para el Cliente. En la arquitectura Cliente/Servidor el remitente de una solicitud es conocido como Cliente.

Características:

a) Es quien inicia solicitudes o peticiones, tienen por tanto un papel activo en la comunicación. Es un dispositivo de tipo maestro o amo.

b) Espera y recibe las respuestas del Servidor.

c) Por lo general, puede conectarse a varios Servidores a la vez.

d) Normalmente interactúa directamente con los usuarios finales mediante una interfaz gráfica de usuario.

2.6.3 SERVIDOR

Es cualquier recurso de cómputo dedicado a responder a los requerimientos del Cliente. Los Servidores pueden estar conectados a los Clientes a través de redes, para proveer de múltiples servicios a los Clientes y funciones tales como impresión, acceso a bases de datos, fax, procesamiento de imágenes. Al receptor de la solicitud enviada por Cliente se conoce como Servidor.

Características.

a) Al iniciarse esperan a que lleguen las solicitudes de los Clientes, desempeñan entonces un papel pasivo en la comunicación. Es un dispositivo de tipo esclavo.

b) Tras la recepción de una solicitud, la procesan y luego envían la respuesta al Cliente.

c) Por lo general, aceptan conexiones desde un gran número de Clientes, en ciertos casos el número máximo de peticiones puede estar limitado.

d) No es frecuente que interactúen directamente con los usuarios finales.

En la figura seis se pueden hacer notar cómo se da la comunicación en este tipo de modelo. El Servidor se encuentra siempre a la espera de una solicitud por parte del Cliente quien es quien determina y propone el intercambio de información, una vez generada la solicitud el Servidor la procesa y es en ese momento que el Cliente queda a la espera que su solicitud sea respondida. Una

vez procesada la información el Servidor la devuelve y el ciclo se repite una vez más.

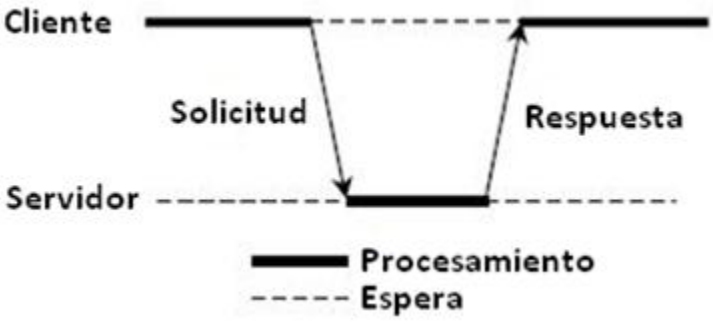

Figura 6. Comunicación Cliente/Servidor. Fuente: Modelo OSI - TCP/IP, Bísaro (2005).

2.6.4 VENTAJAS DEL MODELO CLIENTE/SERVIDOR

Entre las características más importantes que vuelven al modelo Cliente/Servidor en una de las tecnologías más utilizadas actualmente podemos mencionar.

a) La existencia de plataformas de hardware cada vez más baratas en las que puede ser implementado este modelo.

b) Facilita la integración entre sistemas diferentes y comparte información permitiendo, por ejemplo que las máquinas ya existentes puedan ser utilizadas pero utilizando interfaces mas amigables al usuario.

c) Al favorecer el uso de interfaces gráficas interactivas, los sistemas construidos bajo este esquema tienen mayor interacción más intuitiva con el usuario.

d) Es más rápido el mantenimiento y el desarrollo de aplicaciones, pues se pueden emplear las herramientas existentes, por ejemplo los Servidores de SQL o las herramientas de más bajo nivel como los sockets.

e) La estructura inherentemente modular facilita además la integración de nuevas tecnologías y el crecimiento de la infraestructura computacional, favoreciendo así la escalabilidad de las soluciones.

f) El esquema Cliente/Servidor contribuye además, a proporcionar, a los diferentes departamentos de una organización, soluciones locales pero permitiendo la integración de la información relevante a nivel global.

g) Permite distribuir físicamente los procesos y los datos en forma más eficiente, lo que en computación distribuida afecta directamente el tráfico de la red, reduciéndolo de forma considerable (Bísaro, 2005).

2.7 PROTOCOLO SPI

SPI es un bus de tres líneas, sobre el cual se transmiten paquetes de información de 8 bits. Cada una de estas tres líneas porta la información entre los diferentes dispositivos conectados al bus. Cada dispositivo conectado al bus puede actuar como transmisor y receptor al mismo tiempo, por lo que este tipo de comunicación serial es full duplex. Dos de estas líneas trasfieren los datos, una en cada dirección, y la tercer línea es la del reloj. Algunos dispositivos solo pueden ser transmisores y otros solo receptores, generalmente un dispositivo que tramite datos también puede recibir.

Los dispositivos conectados al bus son definidos como maestros y esclavos. Un maestro es aquel que inicia la transferencia de información sobre el bus y genera las señales de reloj y control.

Un esclavo es un dispositivo controlado por el maestro. Cada esclavo es controlado sobre el bus a través de una línea selectora llamada CS (Chip Select, Selección de Chip) o SS (Select Slave, Selccion de Esclavo), por lo tanto el esclavo es activado solo cuando esta línea es seleccionada. Generalmente una línea de selección es dedicada para cada esclavo.

En un tiempo determinado, solo podrá existir un maestro sobre el bus. Cualquier dispositivo esclavo que no esté seleccionado, debe deshabilitarse (ponerlo en alta impedancia) a través de la línea selectora CS. El bus SPI emplea un simple registro de desplazamiento para transmitir la información.

2.7.1 ESPECIFICACIONES DEL BUS

Todas las líneas del bus transmiten la información sobre una sola dirección.

a) La señal CS es la encargada de determinar que dispositivo esclavo debe interactuar con el dispositivo maestro. La selección debe ser única para cada transferencia de información.

b) La señal SCLK (Señal de Sincronización de Reloj) es generada por el maestro y sincroniza la transferencia de datos.

c) La línea MOSI (Master Out Slave In, Salida del Maestro, Entrada del Esclavo) transporta los datos del maestro hacia el esclavo.

d) La línea MISO (Master In Slave Out, Entrada del Maestro, Salida del Esclavo) transporta los datos del esclavo hacia el maestro.

Cada esclavo es seleccionado por un nivel lógico ‘0’ a través de la línea CS o SS. Los datos sobre este bus pueden ser transmitidos a una razón de unos pocos bps hasta 1 Mbps. Los datos son transferidos en bloques de 8 bits, en donde MSB (Bit Más Significativo) se transmite primero. La conexión se da en paralelo para las señales de CLK, MOSI y MISO a excepción de la señal CS donde debe de existir una línea por cada dispositivos esclavo que se conecta al dispositivo maestro. La figura número siete expone el cableado en detalle para una red con un dispositivo maestro y tres esclavos.

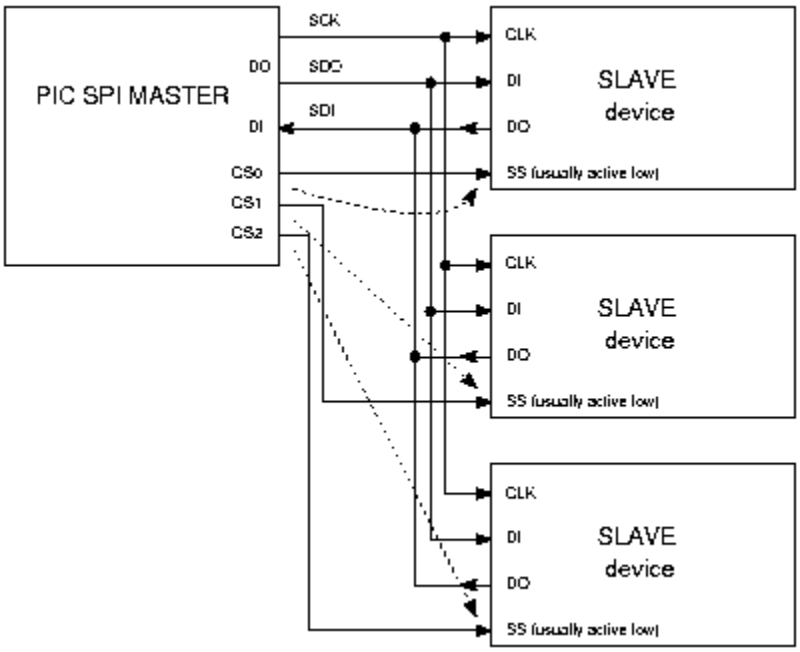

Figura 7. BUS SPI. Fuente: Protocolo SPI (Serial Peripherical Interface), López (2005).

2.7.2 MODOS DEL RELOJ

Todas las transferencias de los datos, son sincronizadas por la línea de reloj de este bus. Un bit es transferido por cada ciclo de reloj.

La mayoría de las interfaces SPI tienen dos bits de configuración, llamados CPOL (Clock Polarity, Polaridad de Reloj) y CPHA (Clock Phase, Reloj de Fase). CPOL determina si el estado Idle (Desocupado) de la línea de reloj esta en bajo CPOL=’0’ o si se encuentra en un estado alto CPOL=’1’. CPHA determina en que flanco de reloj los datos son desplazados hacia dentro o hacia fuera.

Si CPHA=’0’ los datos sobre la línea MOSI son detectados durante cada flanco descendente y los datos sobre la línea MISO son detectados durante cada flanco ascendente.

Cada bit tiene dos estados, lo cual permite cuatro diferentes combinaciones, las cuales son incompatibles una de la otra. Por lo que si dos dispositivos SPI

desean comunicarse entre sí, estos deben tener el mismo la misma polaridad de reloj CPOL y la misma fase de reloj CPHA.

Existen cuatro modos de reloj definidos por el protocolo SPI, estos modos son conocidos como Modo A, B, C y D. Estos determinan el valor de la polaridad del reloj CPOL y el bit de fase del reloj CPHA. La mayoría de los dispositivos SPI pueden soportar al menos dos modos de los cuatro antes mencionados.

Los cuatro modos de reloj en los que puede ser configurado el protocolo SPI se ilustran en las figuras número ocho, nueve, diez y once. López (2005).

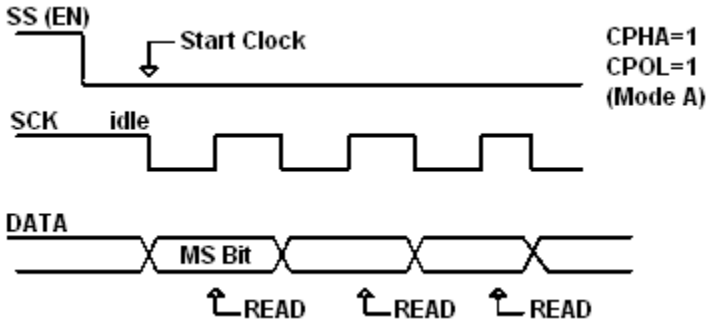

Figura 8. SPI modo A. Fuente: Protocolo SPI (Serial Peripherical Interface), López (2005).

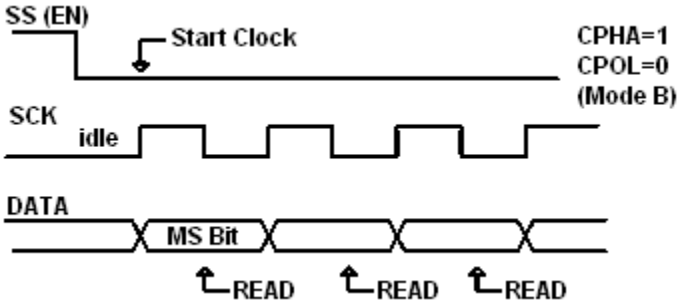

Figura 9. SPI modo B. Fuente: Protocolo SPI (Serial Peripherical Interface), López (2005).

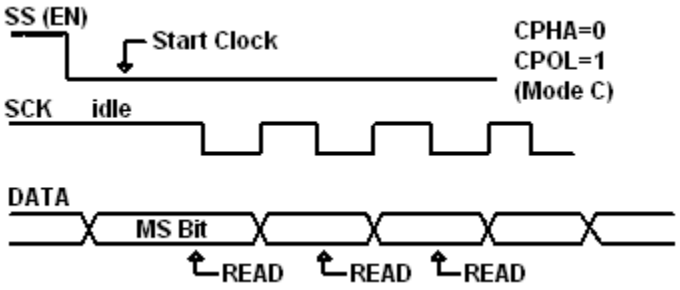

Figura 10. SPI modo C. Fuente: Protocolo SPI (Serial Peripherical Interface), López (2005).

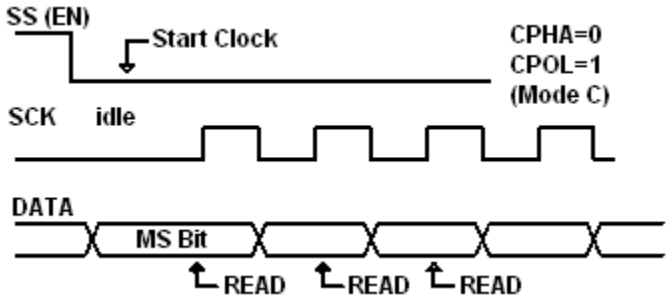

Figura 11. SPI modo D. Fuente: Protocolo SPI (Serial Peripherical Interface), López (2005).

3. METODOLOGÍA

3.1 DESARROLLO DE HARDWARE

En la parte de medición de temperatura se utilizó una cámara de visión termográfica FLIR® modelo A310® por sus prestaciones de alta precisión en la toma de lecturas, amplio rango de trabajo, fácil instalación y uso pero sobretodo su capacidad de conexión bajo el protocolo de comunicación IEE802.3, que la vuelve totalmente compatible con los dispositivos tipo Ethernet de nuestro sistema facilitando la interacción entre ellos. Entre sus características más notables podemos mencionar.

a) Intervalo de temperatura de trabajo de -25°C a +50°C.

b) Campo visual / distancia focal mínima de 25° × 18,8° / 0,4 m.

c) Sensibilidad térmica de 0,05°C a +30°C / 50 mK.

d) Resolución IR de 320 × 240 píxeles.

e) Precisión de ±2 °C ó ±2% por lectura.

e) Transmisión de vídeo vía Ethernet MPEG-4 ó M-JPEG.

Figura 12. Cámara termográfica FLIR A310.

Para el propósito de transmisión de datos y como etapa intermedia que permitiera el enlace entre el FPGA y la red Ethernet se utilizó un Módulo controlador Ethernet ENC28J60 de Microchip; este dispositivo soporta la operación del protocolo TCP/ IP, tambien bajo el estándar de comunicación IEEE 802.3 y que a su vez permite una conexión con el controlador a través de un bus SPI con una velocidad máxima de 10 Mbps. El stack o pila de instrucciones de control para este Módulo es de distribución libre y puede ser obtenido con el mismo fabricante. Entre sus características más importantes podemos mencionar.

a) Dirección MAC (Media Access Control, Control de Acceso al Medio) y conexión 10BASE-T.

b) Puerto dual de transmisión de paquete y Buffer SRAM de transmisión y recepción de 8 Kbyte.

c) Retransmisión automática programable para colisiones.

d) Generación CRC con relleno programable.

e) Rechazo automático programable de paquetes erróneos.

f) Soporta los modos Half y Full-Duplex.

Cabe mencionar que el Módulo controlador Ethernet también se comercializa por diversos fabricantes embebido dentro de tarjetas controladoras que facilitan el acoplamiento digital incorporando reguladores de voltaje, indicadores y periféricos. Para este caso en particular se utilizó la tarjeta ET-MINI® ENC28J60® del fabricante EET®.

Los bloques principales de funcionamiento en los que se encuentra divido el módulo ENC28J60 son la capa física que se conecta directamente al puerto Ethernet, la capa MAC que es la encargada de gestionar el trafico, codificación y decodificación de información entrante y saliente y por último un Buffer interno donde son almacenados todos los datos para que sean accesados por el puerto SPI cuando sean requeridos. La figura número trece ejemplifica esta estructura.

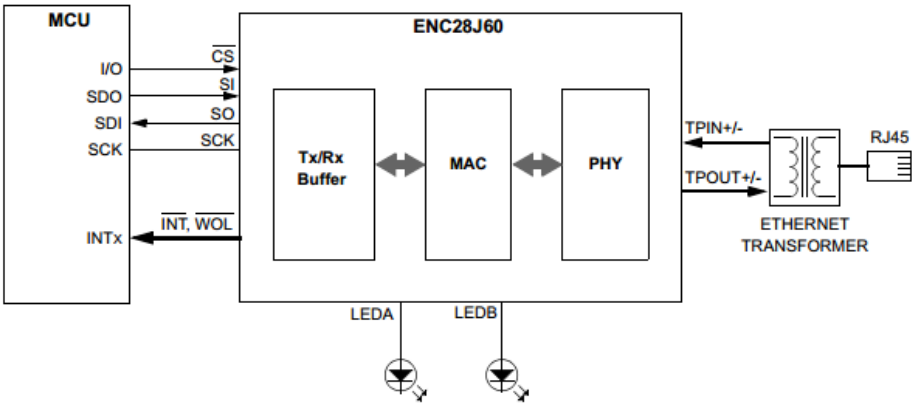

Figura 13. Diagrama a bloques del Módulo ENC28J60.

Como medio de procesamiento y control de la máquina CNC se utiliza un FPGA modelo Spartan3 XC3S1600E®. Caso homólogo al controlador Ethernet se decidió utilizar una tarjeta electrónica que facilitara las etapas de acoplamiento y comunicación; para este fin se hizo uso de una placa electrónica PLCUAQ816®, que fue diseñada e implementada por el grupo de trabajo HSP Digital y que se diseño especialmente como un controlador PLC para maquinaria CNC o robots, como un dispositivo basado en un FPGA de bajo costo y alta capacidad y que por su flexibilidad y robustez puede ser utilizada en un sin número de aplicaciones. Además, el sistema está diseñado para albergar múltiples periféricos digitales y también análogos de entradas y salidas, agregado a esto soporta comunicaciones adicionales mediante puerto serial RS232 o USB. Dentro de sus especificaciones se encuentran.

a) FPGA Spartan 3E de hasta 500,000 compuertas.

b) Oscilador de 48 MHz.

c) Memoria dinámica de 4 MByte.

d) Memoria estática de 512 KByte.

e) 8 Entradas analógicas de ±10 VDC.

f) 8 Salidas analógicas de ±10 VDC.

g) 16 Salidas digitales TTL.

h) 16 Entradas digitales TTL.

i) 1 Puerto USB

Debido al número de periféricos de este módulo, puede ser utilizado en diferentes trabajos industriales adaptándose a los fines que mas convengan.

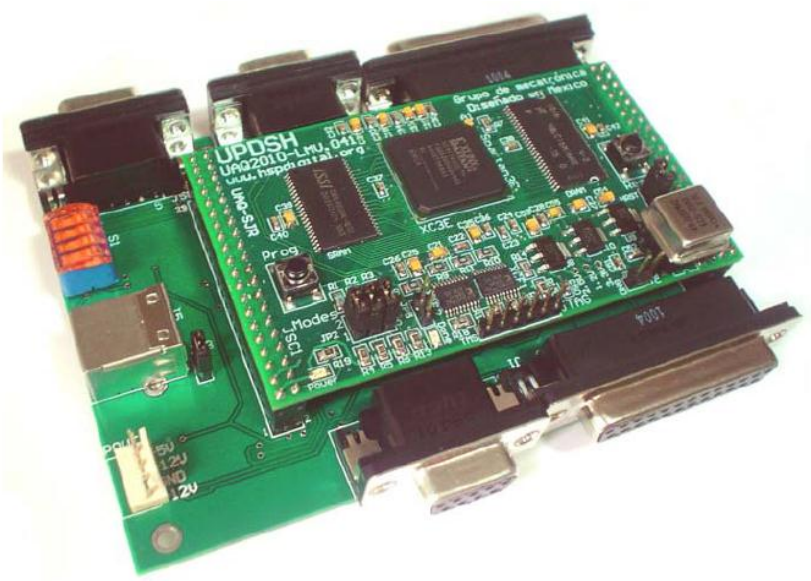

Figura 14. Tarjeta PLCUAQ816.

Finalmente el último el dispositivo fundamental encargado de integrar todo el hardware anteriormente visto, es el microcontrolador PIC18F46K22® de Microchip®; este controlador fue seleccionado por sus buenas prestaciones de memoria FLASH que lo hace buen candidato para poder alojar una aplicación de tipo Servidor, por otra parte es completamente compatible con el módulo controlador Ethernet ya que son producidos por el mimo fabricante reduciendo así la necesidad de desarrollar etapas intermedias de software para hacer la decodificación de la información. Agregado a esto, los microcontroladores de Microchip cuentan con una gran cantidad de herramientas de desarrollo para

simplificar su uso; disponen de un considerable número de periféricos para la transferencia de datos entre ellos SPI, tienen una buena relación de velocidad de procesamiento, son fáciles de obtener y su precio es muy accesible. Dentro de sus capacidades del module PIC18F46K22 se puede mencionar.

a) 64KB de memoria FLASH.

b) Memoria EPROM (Erasable Programmable Read-Only Memory, Memoria de Solo Lectura Reprogramable) de 1024Byte.

c) 36 Periféricos digitales reconfigurables en entradas y salidas.

d) 7 entradas análogas de tipo ADC (Analog to Digital Converter, Convertidor Análogo a Digital) de 13 Bits de resolución.

e) 2 Salidas de tipo PWM (Pulse Width Modulation, Modulador de Ancho de Pulso).

f) 1 Módulo de comunicación SPI, soporta configuración maestro o esclavo.

g) Entrada de oscilador externo para sincronización de envió de datos.

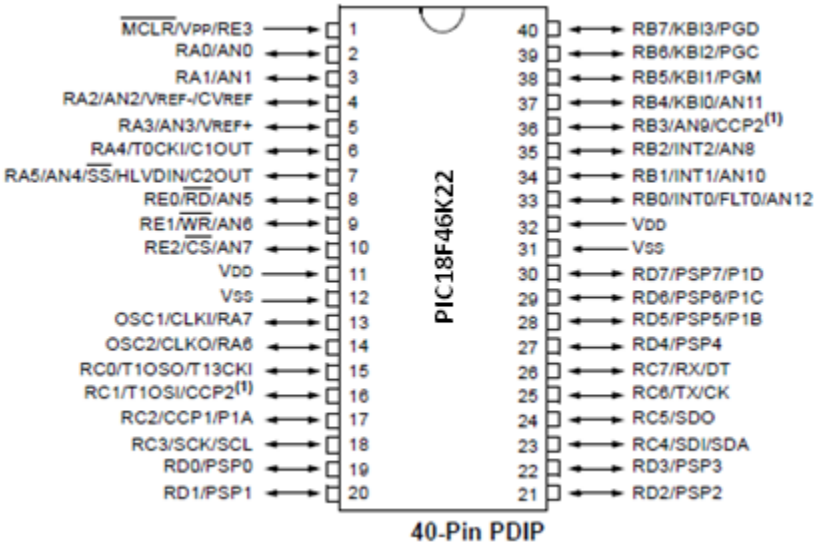

Figura 15. Diagrama de pines PIC18F46K22.

3.2 PLANTEAMIENTO GENERAL

El funcionamiento del sistema se basa en la transferencia de datos entre el microcontrolador y el FPGA que simultáneamente son enviados, recibidos y decodificados mediante el módulo Ethernet hacia y desde la red. La transferencia de información entre estos tres dispositivos se realiza mediante el protocolo SPI, siendo el controlador el encargado de gestionar el tráfico de datos es decir, desempeñándose como maestro y los restantes como esclavos. De forma general en la figura número dieciséis muestra la relación entre todas las partes.

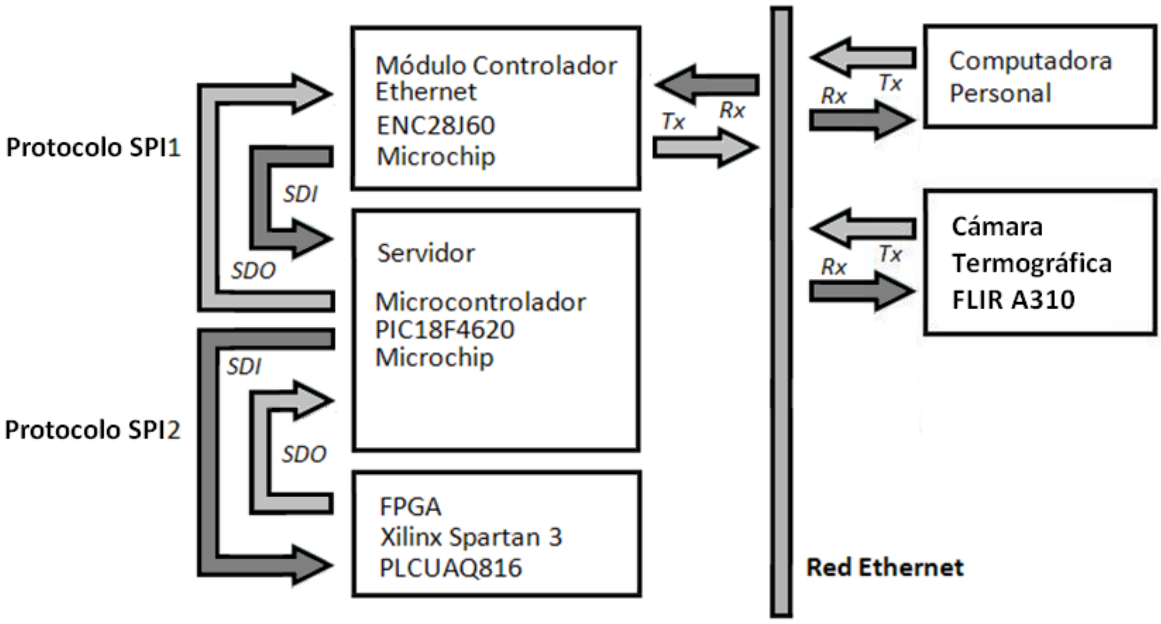

Figura 16. Arquitectura general.

Descripción de las funciones de los elementos del sistema.

Cámara de visión termográfica FLIR A310. Se encarga de realizar el análisis termográfico del objeto de estudio cómo puede ser un motor eléctrico o un sistema mecánico, captura la imagen y la decodifica a formato digital con la información correspondiente a la temperatura irradiada por el cuerpo. Hecho esto, queda a la espera de que un dispositivo solicite el acceso a la información

previamente adquirida y en ese momento la sube a la red mediante el módulo de conexión Ethernet que tiene incorporado.

Red Ethernet. Es la responsable de permitir el flujo de información entre los distintos dispositivos del sistema. Es el medio físico mediante el cual se da el intercambio de datos; toda la información que se comparte en la red entre el Cliente y el Servidor utiliza este medio para su transporte, como norma todos dispositivos conectados a ella deben poseer una dirección única conocida como IP para ser identificados y localizados dentro de la red.

Protocolo SPI. Es el medio de comunicación ente los tres dispositivos básicos del sistema. Esta constituido por un bus de cuatro líneas, sobre el cual se transmiten paquetes de información de 8 bits. Cada una de estas tres líneas porta la información entre los diferentes dispositivos conectados al bus. Cada dispositivo conectado al bus puede actuar como transmisor y receptor al mismo tiempo, por lo que este tipo de comunicación serial es Full-Dúplex.

Microcontrolador (Servidor). Su función es la gestionar tota la información que se maneja en el sistema; debe de entregar la información que es solicitada en la red Ethernet por el Cliente; de modo contrario también debe de cumplir la operación de tomar la los datos provenientes de dicho Cliente y entregarla al FPGA para su inmediata ejecución. Por otra parte se desempeña como dispositivo maestro entre el FPGA y el Módulo Ethernet haciendo uso del periférico según las prioridades del sistema.

Módulo controlador Ethernet. Debe decodificar la información procedente la red Ethernet y reconvertirla a protocolo SPI para que pueda ser identificada por el microcontrolador y ejecutada por el FPGA, caso inverso debe tomar la información que le entrega el microcontrolador mediante el protocolo SPI y convertirla a protocolo Ethernet y finalmente entregarla a el Cliente que se encuentran en ese momento solicitando dicha información.

FPGA Spartan 3. Solicita y procesa la información proveniente de la cámara termográfica en formato SPI mediante el puerto de comunicación que fue

sintetizado y tiene embebido. A si mismo se encarga de realizar el almacenamiento de la información previamente obtenida en sus módulos de memoria SDRAM; simultáneamente a este proceso, produce una señal SPI para la intercambio de datos con el microcontrolador para el monitoreo con la interface del operador.

Interface de operador (Cliente). Su tarea consiste en solicitar que la información ubicada en el Servidor le sea entregada a través de la red Ethernet para luego desplegarla en pantalla para que sea utilizada de esta forma por el usuario de la cámara termográfica. En síntesis cumple la función de capa de aplicación para que el usuario final pueda obtener de forma grafica la información.

3.3 DESARROLLO DE SOFTWARE

El realización del software se puede dividir en dos áreas fundamentales, la primera es la generación de un IP Core, que puede emular el protocolo SPI que utilizan tanto como el microcontrolador y el módulo Ethernet y así mismo realizar la conexión con la cámara termográfica; sin esta etapa sería imposible comunicar todos los dispositivos correctamente ya que es el periférico de transmisión estándar de alta velocidad con el que vienen integrados los microcontroladores, empleando otros protocolos internos del controlador se generarían atrasos en la transferencia de datos; por otra parte también se diseñaron interfaces de memoria que puedan alojarse en la SDRAM del PLCUAQ816 para poder tener almacenamiento de información actual del sistema para su posterior transmisión. Para este fin se utiliza la aplicación Project Navigator® que forma parte del Xilinx ISE Design Suite®.

La arquitectura de hardware del módulo SPI puede ser observada en la figura número diecisiete donde se muestran de forma detallada la interacción de todos los bloques teniendo como partes fundamentales un contador de módulo ocho, registros de desplazamiento y una máquina de estados que es la encargada de sincronizar todas la señales.

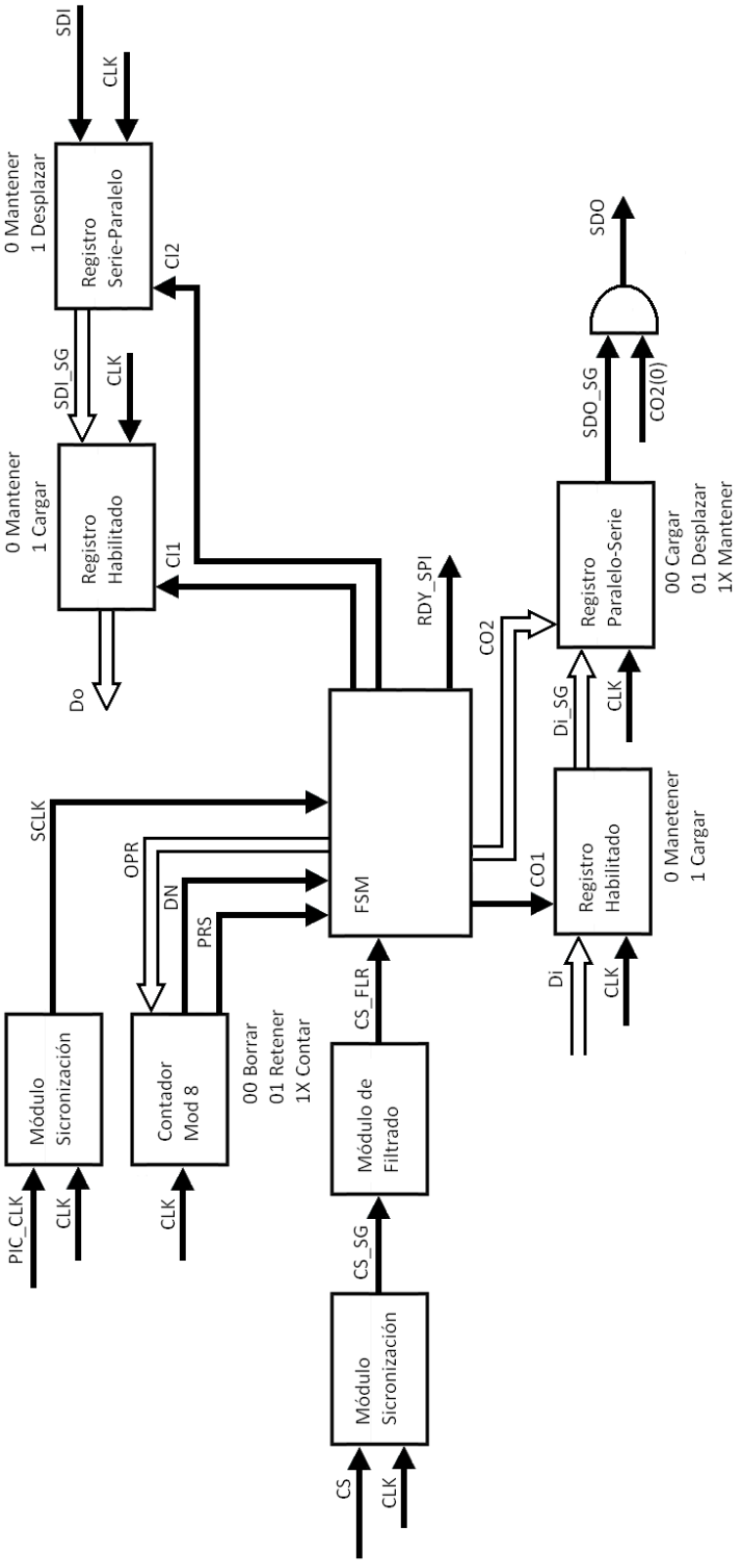

Figura 17. Arquitectura SPI.

De forma puntual se pueden observar cuatro señales principales de entrada y salida de este módulo y sus características son las siguientes:

CS. En sistemas que cuentan con más de un dispositivo esclavo se encarga de seleccionar cuál de ellos se encontrará en modo escucha para recibir o enviar información al dispositivo maestro. Así mismo determina el momento en que el dato que será transmitido por protocolo SPI es válido, convencionalmente esto esta designado como estado lógico en bajo.

PIC_CLK. Es la señal de sincronía que es usada para determinar cuándo debe ser capturado el bit de información transmitido, en el envió o en la recepción. Es generada siempre por el dispositivo maestro. Su función es la de eliminar la incertidumbre de saber si los bits transmitidos de forma serial son válidos al momento de la recepción o si sufrieron algún tipo de desfase con respecto al emisor.

SDI. Es la señal que contiene la información que es envidada del dispositivo maestro al dispositivo esclavo de forma serial; su forma es la de un tren de pulsos y debe encontrarse siempre en sincronía con la señal PIC_CLK.

SDO. Es la contraparte de la señal SDI. Es la señal que contiene la información que es envidada del dispositivo esclavo al dispositivo maestro. Debe cumplir las mismas características de la señal SDI.

En la figura dieciocho se muestran una simulación realizada para un envío y recepción de datos de forma simultánea. El ciclo comienza al terminar el pulso RST (Reset) y es en este momento en que el módulo queda a la espera de un pulso de la señal CS en bajo para comenzar a funcionar, acto seguido, la señal se sincroniza con el pulso de reloj y tira la señal RDY (Ready), indicando que el dato de recepción en ese momento se empieza a procesar y que no debe ser alimentado con un dato nuevo, así mismo indica que el dato transmitido no es válido en este momento y lo será hasta concluir el ciclo. Iniciado este proceso la señal de entrada SDI y la de salida SDO se sincronizan con la señal de reloj interno del FPGA pero también con la señal de reloj proveniente del dispositivo

maestro PIC_CLK. La señal SDI captura el estado del pulso al momento de cortar la señal de reloj del microcontrolador y la señal SDO comienza a transmitir el valor del dato alojado en la señal de entrada DI. Terminados los ocho pulsos de reloj correspondientes a los ocho bits configurados para la transmisión, la señal de Ready vuele a estado de alto y el dato DO despliega el valor capturado mediante la señal SDI. Finalmente el módulo queda a la espera de recibir una nueva señal CS que indique que el ciclo se repetirá una vez más.

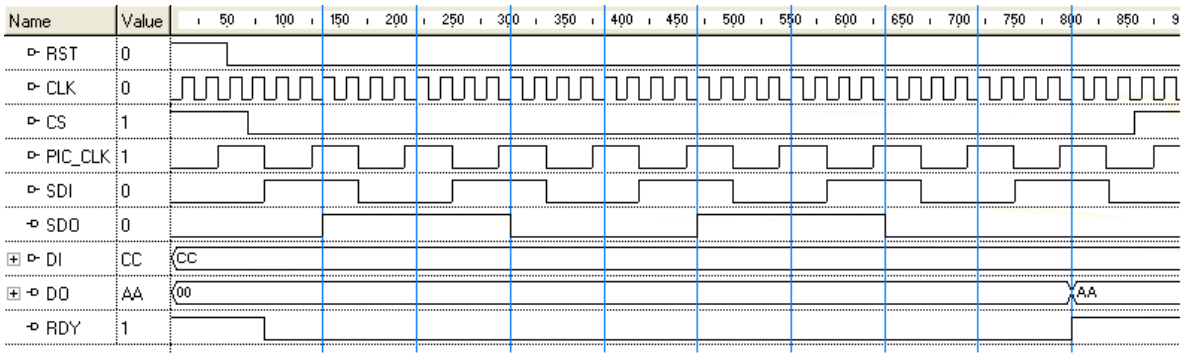

Figura 18. Protocolo SPI descrito en VHDL.

Por otra parte es necesario contar con una estructura de memoria que pueda albergar todos los datos que han sido introducidos mediante el módulo SPI para que de esta forma puedan se utilizaos para su posterior procesamiento, es por ello que fue seleccionado el módulo IS61LV5128AL® que se encuentra embebido en la tarjeta PLCUAQ816 y que es una memoria de alta velocidad de lectura y escritura de tipo estática, con una capacidad de 512K direcciones de memoria por 8 bits de espacio de almacenaje para cada una de las direcciones. Esta situación la vuelve idónea para nuestro propósito, por cumplir exactamente el tamaño de trama transmitido mediante el protocolo SPI de 8 bits y por tener suficientes localidades de memoria para alojar imágenes. El la figura número diecinueve se pueden observar de forma detallada todas las partes que conformar el módulo controlador de la memoria estática.

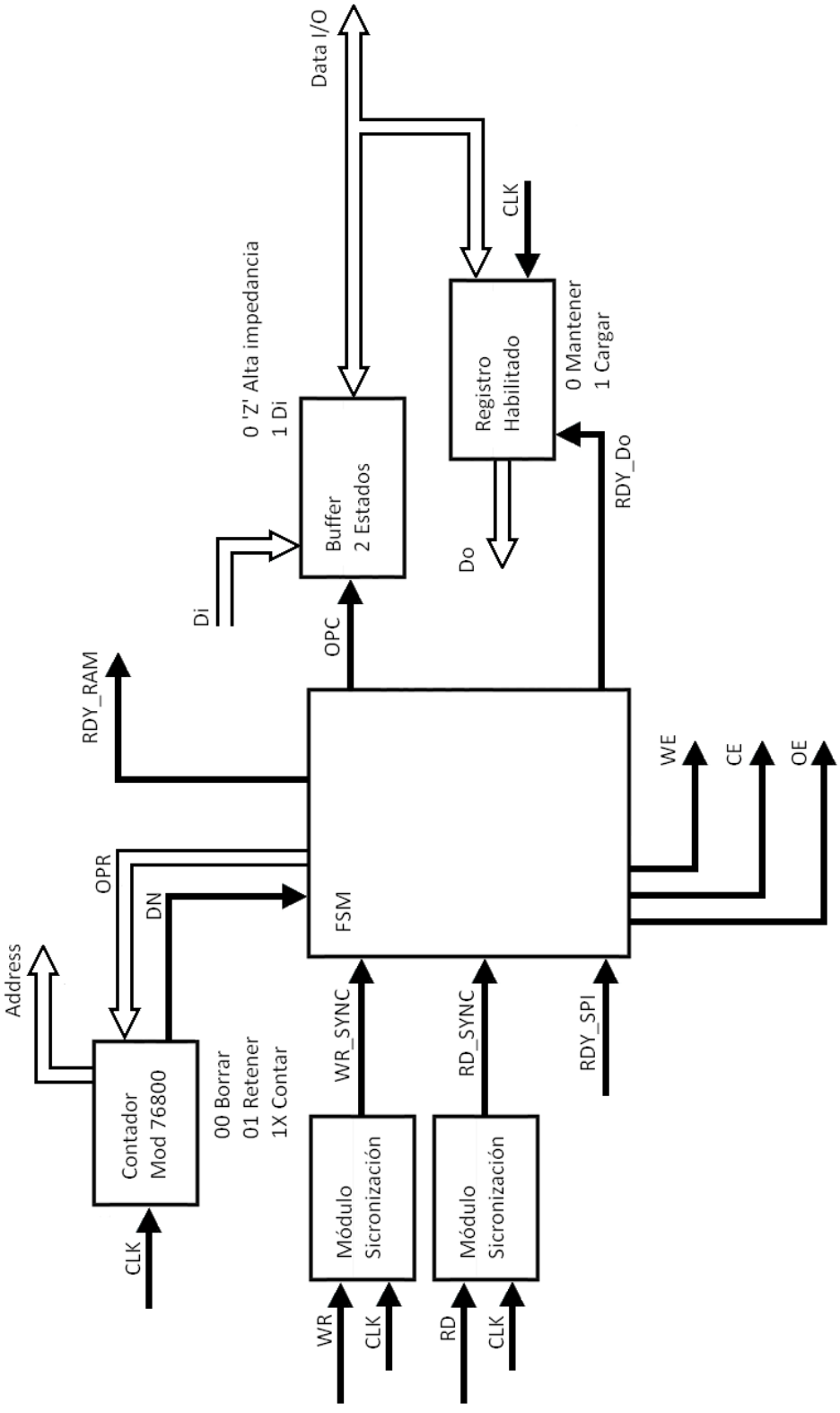

Figura 19. Módulo SRAM.

Entre sus partes fundamentales podemos citar:

WR. Es la señal de inicio para el modo escritura del módulo SRAM. Mientras esta señal siga presente se seguirán guardando los datos provenientes del módulo SPI o en caso contrario, hasta que sean llenadas todas las localidades de memoria necesarias para transmitir las imágenes.

RD. Es la señal de inicio para el modo lectura del módulo SRAM. Mientras esta señal se encuentra presente se envían los datos guardados en la memoria estática hacia el módulo SPI o en caso contrario, hasta finalizar de leer todas las localidades de memoria necesarias para transmitir las imágenes.

DI. Es el dato de entrada de la memoria estática y su longitud es de 8 bits. Su utilización se restringe al modo de escritura que es el momento en que se deben posicionar los datos en esta señal para ser salvados y transmitidos al módulo SPI.

DO. Es el dato de salida de la memoria estática y como es de esperar debido a los datos de entrada su longitud es de 8 bits. Su utilización se restringe al modo de lectura que es el momento en que se deben capturar los datos en esta señal para ser leídos y transmitidos al módulo SPI.

RDY_SPI. Determina cuando el dato localizado en las señales DI y DO es válido para ser escrito o accesado de forma correcta. Prescindir de esta señal causaría incertidumbre de la vigencia de la información al momento realizar un ciclo de escritura o de lectura ya que volvería al sistema asíncrono generando con ello muchos errores.

Data I/O. Es el bus de datos de entrada y que su vez pueden funcionar como salidas físicas de la memoria estática. La selección entre un modo de funcionamiento y otro es determinado por las señales de comando WE, CE y OE que rigen a la memoria.

Address. Es la dirección de localidad en la que se realizara un acceso ya sea de escritura o lectura de la memoria estática. Al momento de la captura debe encontrarse correctamente sincronizada con las señales DI y DO.

RDY_RAM. Determina cuando puede ser accedido el módulo de control de le memoria estática. Esta señal se encuentra en estado alto al momento de la inactividad o bien cuando la memoria haya sido escrita o leída en la totalidad de las localidades.

WE, CE, OE. Son las señales de comando que rigen el funcionamiento de la memoria estática, es decir permiten que la memoria se habilite, deshabilite, se ponga en modo lectura o escritura. El modo detallado seleccionar estas señales se puede observar en la tabla de verdad de la figura número veinte.

Mode	$\overline{WE}$	$\overline{CE}$	$\overline{OE}$	I/O Operation
Not Selected (Power-down)	X	H	X	High-Z
Output Disabled	H	L	H	High-Z
Read	H	L	L	D_{OUT}
Write	L	L	X	D_{IN}

Figura 20. Modos de funcionamiento de memoria estática.

Finalmente es necesario realizar la interconexión entre el módulo SPI y el módulo de control de la memoria estática de forma sincronizada, de tal modo que los datos transmitidos sean almacenados de forma ordenada y secuencial en la memoria, para que al momento de dar acceso se posible determinar la posición correcta de cada dato. Dicha sincronización se logra al limitar la transmisión de datos con la señal RDY_SPI que le indica al módulo de control de la memoria estática cuando debe escribir o leer la información de forma correcta. Por otra parte es necesario interconectar los buses de entrada de datos DI del módulo SPI con el de salida de datos DO del módulo de control de memoria, de forma análoga deben ser conectados los buses de salida de datos DO del módulo SPI con el de entada de datos DI del módulo de control de memoria. El funcionamiento completo de todo el sistema integrado se muestra en la figura número veintiuno.

La descripción de código referente al módulo SPI y módulo de control de memoria pueden ser consultados en el apéndice A.

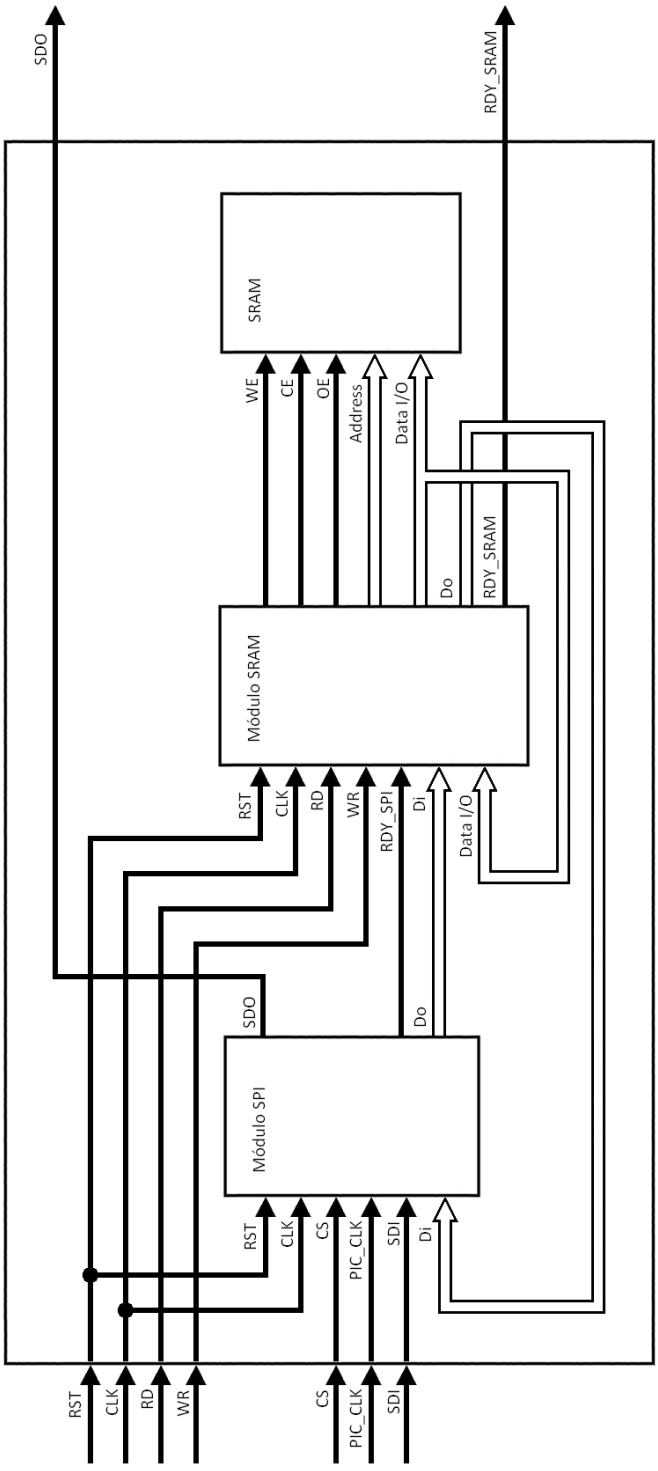

Figura 21. Integración de módulos.

La segunda área de desarrollo de software fue la programación la aplicación Cliente/Servidor y la subsecuente implementación en el PIC18F46K22; para ello se hizo uso de los stack que vienen preinstalados en la aplicación mikroC PRO for PIC®, se decidió utilizar esta paquetería debido al alto control que permite tener sobre el módulo ENC28J60 así como la simplicidad con la que puede ser manipulado, otros fabricantes ofrecen soluciones alternativas pero es necesario adquirir los stack específicos, diseñados para una aplicación particular limitando la gamma de utilidades que ofrece este dispositivo. Entre las librerías empleadas se incluyen la de configuración de servidor con configuración IPv4, configuración de servidor TCP/IP o UDP/IP, configuración de MAC, envío y recepción de paquetes.

De forma específica se pueden mencionar los comandos que intervienen la transmisión envío/recepción de los paquetes de datos:

SPI_Ethernet_Init. Inicializa y configura el puerto SPI que queda a disposición del módulo Ethernet ENC28J60. Entre sus parámetros encontramos la dirección MAC que le será asignada dispositivo, su dirección IP fija y el tipo de comunicación que se desea realizar ya sea full dúplex o half dúplex.

SPI_Ethernet_UserUDP. Es una rutina de tipo UDP y es llamada por la librería interna. El usuario accesa a las solicitudes realizadas por el cliente mediante la rutina de captura SPI_Ethernet_getByte. La función debe regresar la longitud en bytes de la respuesta UDP o cero si ningún paquete fue transmitido. Si no es necesario generar respuestas a las solicitudes debe definirse el parámetro de tipo return como cero.

SPI_Ethernet_getByte. Es una rutina de acceso al registro MAC. Captura un byte de la dirección que dirige al puntero actual del módulo ENC28J60. Para acceder a toda la trama de recepción es necesario generar un ciclo que se repita en la misma cantidad de veces como bytes contenga la longitud del paquete de recepción.

SPI_Ethernet_putBye. Es una rutina de acceso al registro MAC. Almacena un byte de la dirección que dirige al puntero actual del módulo ENC28J60 excluyendo

el carácter nulo. Para acceder a toda la trama de transmisión es necesario generar un ciclo que se repita en la misma cantidad de veces como bytes contenga la longitud del paquete de transmisión.

SPI_Ethernet_doPacket. Procesa los paquetes recibidos y enviados por las tres funciones anteriores. Se localiza en el cuerpo del programa y debe ser llama a de forma continua o la comunicación corre el riesgo de perderse. Es necesario depurar el código para dar prioridad y periodicidad a este evento.

SPI_Init_Advanced. Configura e inicializa el módulo SPI dependiendo las características definidas por el usuario. Esta función debe ser llamada previamente a la utilización de cualquier módulo SPI y sus librerías derivadas. Los modos en que este parámetro puede ser modificado se muestran en la figura veintidós.

Description	Predefined library const
SPI work mode:	
Master clock = Fosc/4	_SPI_MASTER_OSC_DIV4
Master clock = Fosc/16	_SPI_MASTER_OSC_DIV16
Master clock = Fosc/64	_SPI_MASTER_OSC_DIV64
Master clock source TMR2	_SPI_MASTER_TMR2
Slave select enabled	_SPI_SLAVE_SS_ENABLE
Slave select disabled	_SPI_SLAVE_SS_DIS
Data sampling interval:	
Input data sampled in middle of interval	_SPI_DATA_SAMPLE_MIDDLE
Input data sampled at the end of interval	_SPI_DATA_SAMPLE_END
SPI clock idle state:	
Clock idle HIGH	_SPI_CLK_IDLE_HIGH
Clock idle LOW	_SPI_CLK_IDLE_LOW
Transmit edge:	
Data transmit on low to high edge	_SPI_LOW_2_HIGH
Data transmit on high to low edge	_SPI_HIGH_2_LOW

Figura 22. Parámetros de inicialización SPI.

SPI work mode. Determina la frecuencia de trabajo a la que serán generados los pulsos de reloj del dispositivo maestro o de caso inverso si el dispositivo tomará de forma externa los pulsos de reloj. Como es lógico pensar una mayor frecuencia de trabajo permite aumentar la velocidad de transmisión pero hay que considerar que trae consigo una mayor vulnerabilidad a las interferencias electromagnéticas. Para el presente proyecto se decidió utilizar un divisor por dieciséis de una frecuencia de reloj de 20 MHz resultando en una señal de reloj aproximada de 1.25 MHz por presentar una buena relación de velocidad de transferencia e inmunidad a interferencias.

Data sampling interval. Especifica el momento en que es válido el dato de entrada SDI o de salida SDO con respecto a la señal de reloj, puede configurarse a la mitad del pulso o al finalizar.

SPI clock idle state. Permite seleccionar el estado de reposo del pulso de reloj, es decir, el estado lógico en que permanecerá al momento de estar inactivo. Debido a que el pulso de reloj es una señal de alta frecuencia es común que reciba interferencia con líneas de datos cercanas a ella, por lo que es recomendable configurar este parámetro en alto para inmunizar de la mejor manera posible la señal, evitando falsos pulso de reloj.

Transmit edge. Determina el tipo de flanco ya sea bajo a alto o alto a bajo en que será manipulado el dato SDO.

SPI_Set_Active. Configura en modo activo el módulo SPI que será utilizado en caso de existir más de uno. Es importante conmutar entre cada puerto SPI cada vez que sea utilizado.

SPI_Read. Captura y decodifica un dato conformado por 8 bits que se encuentre presente en la señal SDI del puerto SPI seleccionado.

SPI_Write. Codifica y envía un dato conformado por 8 bits poniéndolo a disposición en la señal SDO del puerto SPI seleccionado.

El código completo referente al modelo Servidor del microcontrolador puede ser observado en el apéndice B.

Finalmente se desarrolló la interface de operador, llevando a cabo una aplicación de tipo Cliente UDP mediante el uso del lenguaje de programación orientado a eventos Visual Basic 6®, valiéndose de controles Winsock para el intercambio de datos. Es necesario recordar en este punto que un sistema Cliente/Servidor se basa en el hecho de que el Servidor se encuentra a la espera de las solicitudes que genera el Cliente y que únicamente cuando se produce una solicitud el Servidor generará una respuesta. Es por ello que el sistema Cliente debe ser el encargado de gestionar el tráfico de datos tomando en cuenta que solo existirá transmisión de datos cuando éste lo solicite.

La primera acción fundamental a realizar para la aplicación de tipo Cliente, es crear la conexión al Servidor, ya que solo se puede transmitir información si la conexión Cliente/Servidor si se encuentra activa. Las propiedades necesarias para crear la conexión son:

RemoteHost. Asigna la dirección del dispositivo externo o Servidor al que se desea conectar el dispositivo Cliente.

RemotePort. Determina el puerto al que deseamos conectar el RemoteHost o Servidor. El puerto dispone un canal único que será apartado para la transmisión de datos y que debe ser compatible con el puerto abierto por el Servidor.

A su vez el lenguaje Visual Basic utiliza métodos y eventos que son directivas que rigen el comportamiento del programa. Un método es una acción que deseamos que realice el programa, por otra parte un evento es el cambio de estado en las características de un objeto y que se dan como resultado del uso de un método.

En la generación de la conexión Cliente/Servidor es necesario utilizar los siguientes métodos:

Connect. Realiza una solicitud de conexión con el Servidor.

Close. Cierra la conexión con el Servidor.

Así mismo se involucran los siguientes eventos:

Connect. Ocurre cuando se ha establecido con éxito la conexión con el Servidor.

Close. Sucede cuando el Servidor cierra la conexión al Cliente.

Error. Resulta de la generación de un error en la conexión.

Es necesario llamar al método Connect para realizar la conexión, siempre asegurándonos que el Socket no esté siendo utilizado. Para ello se utiliza el método Close que se encarga de cerrar toda conexión pendiente en el Socket. Si la conexión se realiza con éxito se dispara un evento para tal fin, en donde es posible realizar acciones inmediatas en el momento preciso en que se logra establecer la conexión con el servidor. En este punto se han creado los lazos básicos para realizar cualquier intercambio de datos con el servidor, ya sea texto ASCII o datos binarios. Se debe tener presente que en cualquier momento el Servidor puede cerrar la conexión, o bien cerrarse por algún error, para ello se cuenta con el evento Close, que se dispara al perder la conexión con el Servidor, en cambio si se desea que el Cliente cierre la conexión con el Servidor basta con llamar al método Close directamente.

Una vez realizada con éxito la conexión, solo resta comenzar a transferir datos, cabe aclarar que estos datos se envían siempre en forma binaria aunque su captura sea de tipo texto, ya que un carácter de texto es en sí una representación gráfica de un número binario en su equivalencia del código ASCII, es importante tambien hacer notar que a través de un Socket se puede enviar texto normal o datos binarios, declarados como variables de tipo String mayormente conocidas como cadenas o como variables tipo Byte que son enteros cortos de 8 bits.

Los métodos necesarios para el envío y recepción de datos de datos son:

SendData. Envía datos al otro extremo de la conexión ó Socket remoto.

GetData. Recibe datos enviados por el extremo remoto de la conexión ó Socket remoto.

Y los eventos involucrados en la transferencia de datos son.

DataArrival. Sucede cuando el Socket remoto envía un paquete de datos por el puerto que se encuentra abierto. Este evento es consecuencia de la respuesta del Servidor al método SendData.

Error. Ocurre en caso de una falla en la transmisión o recepción de un paquete de datos. Un caso común es cuando el servidor no entrega respuesta a una solicitud del Cliente.

El método SendData debe contener una cadena de caracteres o en otro caso un arreglo de enteros cortos, de lo contrario generará un error ya que resulta imposible asignar un orden establecido en el paquete a un tipo de datos diferente a los anteriores, comprometiendo la integridad de la información. Hecho esto lo envía inmediatamente al Socket remoto. Cuando el Socket remoto envía un paquete de datos, es decir de forma reciproca a la solicitud realizada, se genera el evento DataArrival indicando que existe nueva información disponible en el puerto. Por último es necesario capturar el paquete de datos en cola generado por el Servidor mediante el método GetData que de forma análoga, debe ser almacenado en una variable de tipo cadena o un arreglo de enteros cortos.

Por último punto es importante tomar cierta acción cuando se produzca algún error, aunque esta acción tan solo sea cerrar la conexión e informar al usuario de lo ocurrido. Para el manejo de errores producidos durante la conexión se cuenta con un evento dedicado, llamado Error el cual retorna varios valores para darnos información al respecto ya sea mediante un código numérico o con una descripción breve del error. En caso de producirse algún error la acción más comúnmente realizada es simplemente cerrar la conexión con el método Close y después de cierto tiempo repetir el método de conexión Connect. De persistir este evento es posible que el Servidor haya sufrido un desperfecto o que por alguna razón haya recibido la instrucción de cerrar definitivamente el puerto.

El código completo referente al modelo Cliente del ordenador puede ser observado en el apéndice C.

Por la parte de adquisición de imágenes provenientes de la cámara termográfica fue utilizado el ActiveX de la librería SDK 2.6 Thermovision®, que es un kit de desarrollo de software y que distribuye FLIR para la interconexión de sus productos con sistemas a la medida y que es generalmente utilizado por integradores de hardware. Estas librerías se encuentran disponibles para Microsoft Visual Studio® por lo que las vuelven idóneas para ser adaptadas a la etapa de transmisión Cliente/Servidor que se estudio anteriormente.

Antes de intentar realizar la comunicación de la librería es necesario realizar la conexión entre la red, la cámara FLIR A310 y la PC como lo muestra la figura veintitrés.

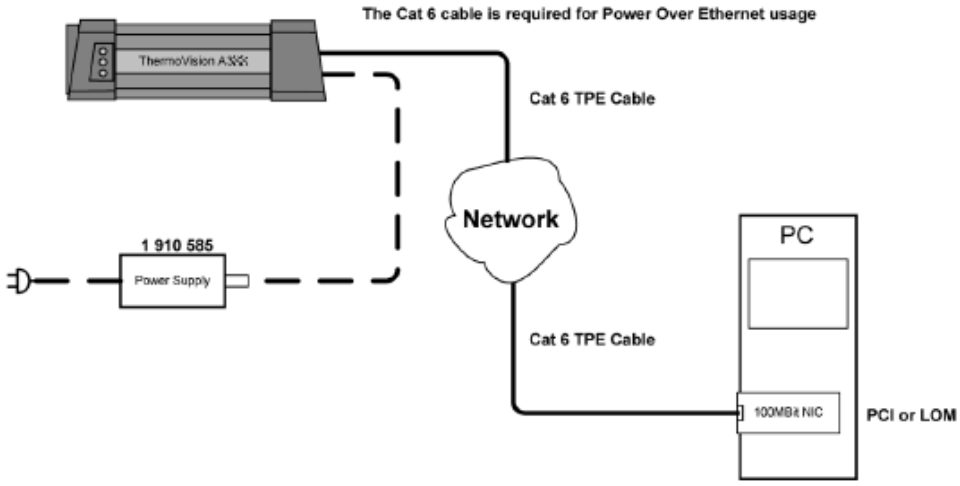

Figura 23. Conexión cámara FLIR A310.

Para el control de todas las prestaciones de la cámara la manipulación de la información la librería se encuentra dividida en tres tipos de comandos:

Estatus. Entrega el estado en que se encuentra la cámara, esta información es exclusivamente de solo lectura, entre sus capacidades se pueden mencionar por ejemplo adquirir el modelo de la cámara que se encuentra conectada y su número

de serie, si la conexión se realizó con éxito, existencia de un error de transferencia de datos o el estado actual de la batería entre otros.

Control. Este tipo de comando determina la acción que debe realizar la cámara según sea la intención de programación del usuario, estos parámetros se encuentran abiertos a modo escritura y lectura. Entre ellos se encuentran el parámetro de emisividad que se encuentra seleccionado, tipo de filtro aplicado o la configuración de disparo de captura por entrada externa.

Datos. Básicamente es la información de la imagen capturada por la cámara y trasformada en un arreglo de datos correspondientes a la digitalización de dicha imagen. Puede accederse como un código de color o un arreglo de flotantes correspondientes a los grados kelvin de cada pixel.

De forma análoga al sistema Cliente/Servidor el control de estos comandos se da en Visual Basic mediante métodos. Los principales métodos utilizados para la transmisión de datos de la cámara infrarroja son:

Connect. Establece en modo activo la conexión con la cámara infrarroja, se debe utilizar siempre este método antes de la adquisición de una imagen. Entre sus parámetros se incluyen tipo de cámara a conectar, puerto de conexión, tipo de conexión, interface e IP destino.

DoCameraAction. Desempeña una acción específica relacionada con los ajustes ópticos de la cámara. En otros están el modo fotografía, modo grabación, auto enfoque, calibración así como corrección de imagen.

GetCameraProperty. Captura los parámetros de calibración de la cámara, su uso es exclusivo al modo solo lectura. Entre estos se encuentran, coeficiente de emisividad, coeficiente de reflectividad, temperatura atmosférica, distancia al objeto, unidades de representación y paleta de color aplicada.

SetCameraProperty. Configura los parámetros de calibración de la cámara, su uso está restringido al modo de solo escritura. Su función es la reciproca al método GetCameraProperty.

GetError. Devuelve el código de error generado por la cámara en caso de presentarse alguna falla. Es una método muy importa ya que en caso de algún desperfecto nos permite encontrar una solución apropiada. De los códigos devueltos podemos destacar el de dispositivo no presente, dispositivo ocupado, falta de controlador del dispositivo, falla de configuración, error en buffer de alojamiento, imagen inválida y error en hardware de la cámara.

GetImage. Captura una imagen proveniente de la cámara y la almacena en un arreglo de dos dimensiones de datos tipo variante, el tamaño de la imagen depende de la resolución del tipo de cámara. Es unos de los métodos principales de la librería ya que nos permite realizar la digitalización de las imágenes para su posterior utilización. El arreglo de datos devuelto puede configurarse como imagen absoluta de 16 bits de resolución de tipo entero, imagen de señal de tipo flotante, imagen de temperatura de tipo flotante y por último imagen de temperatura relativa de 8 bits de tipo entero.

GetLut. Su funcionamiento es similar al método GetImage con la diferencia de que en lugar de generar un arreglo de datos correspondiente a la imagen capturada, este método genera una tabla de equivalencia de temperatura en grados Kelvin para cada pixel de la imagen. El rango de precisión de la conversión puede ser configurado en resoluciones de 8, 15 y 16 bits según convenga al usuario.

CameraEvent. Este evento ocurre cuando el estado de la cámara por alguna razón cambia, puede generarse ya sea por una solicitud de conexión o desconexión, por una conexión rota o inclusivo al capturar una imagen y almacenarla en el buffer de salida.

Para una ejemplificación más clara de los métodos y eventos del ActiveX de la librería de FLIR el código completo concerniente puede ser examinado en el apéndice D.

3.4 IMPLEMENTACIÓN DE DISPOSITVOS

Como último apartado de este capítulo se muestra del diseño de hardware de todo el sistema; es posible apreciar en la figura veinticuatro el diagrama eléctrico de interconexión entre todos los dispositivos que lo componen. Se han añadido diodos Zener de 3.9 VDC en las señales CS, CLK, SDI y SDO del puerto SPI que comunica el microcontrolador y el FPGA para la eliminación de interferencia electromagnética. Es necesario también tomar en cuenta que la alimentación de todos los dispositivos están reguladas a +5VDC y que todas las tierras son comunes.

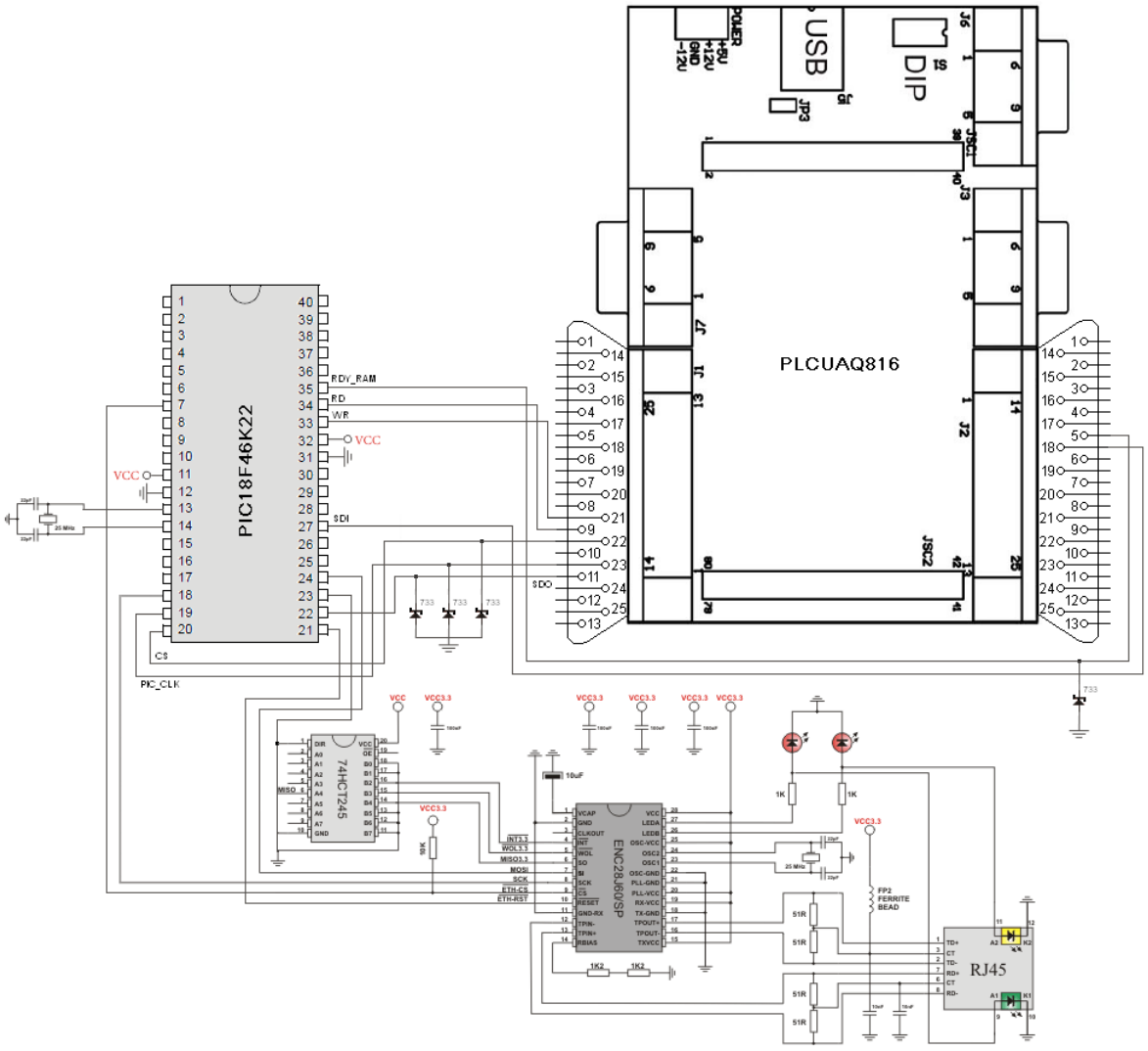

Figura 24. Diagrama eléctrico.

Finalmente la última etapa de diseño consistió en crear un PCB (Printed Circuit Board, Tarjeta de Circuito Impreso) capaz de alojar a todos los dispositivos electrónicos descritos anteriormente. Este tipo de tarjetas son de gran importancia debido a sus diferentes cualidades, entre ellas se pueden citar la disminución considerable del tamaño de las conexiones y a su vez del espacio necesario para montar todos los dispositivos, proporcionan un anclaje mecánico firme, permiten una buena disipación de calor producido por los circuitos integrados, así mismo, atenúan el ruido electromagnético debido a que por la disposición de las pistas, su grosor y sus ángulos, disminuyen la sensibilidad eléctrica entre una pista y otra reduciendo así las fallas en la trasmisión de datos.

La constitución del PCB fue diseñado con la herramienta ARES® que es un complemento del software de aplicación PROTEUS®. Esta aplicación genera a partir del circuito eléctrico el PCB de forma automática, haciendo caso a características específicas definidas por el usuario como pueden ser la disposición física de los elementos, el número de capas del circuito, tamaño de las pistas, tamaño de las juntas de soldadura, entre otras. En la figura veinticinco se muestra el diseño terminado del PCB.

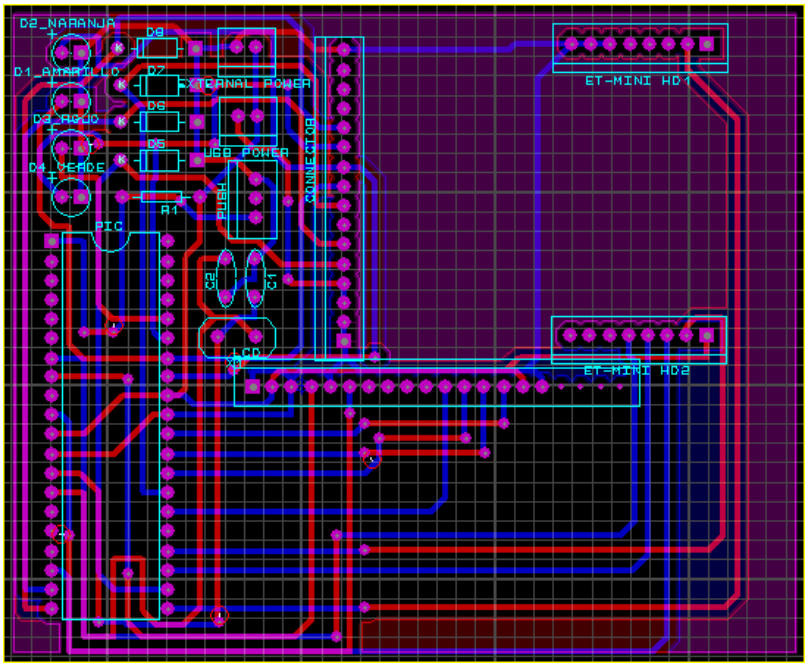

Figura 25. Diseño de PCB

De este diseño se desprende la fabricación física de la tarjeta mediante el uso de una placa fenólica con superficie de cobre y la colocación de los diferentes dispositivos. Como se puede observar en la figura veintiséis, la tarjeta terminada presenta sistema muy compacto que permite su fácil manipulación. Para su funcionamiento su conexión también se vuelve más sencilla, ya que únicamente es necesario conectar el puerto de comunicaciones con el FPGA y el puerto de transmisión Ethernet, debido a que su alimentación la obtiene directamente de la conexión en paralelo con la fuente de alimentación del FPGA.

Figura 26. Tarjeta electrónica.

4. PRUEBAS Y RESULTADOS

Como es lógico pensar ya que el presente proyecto se encuentra conformado por varias etapas fue necesario realizar pruebas de funcionamiento a cada una de ellas, de forma individual, para luego hacerlas trabajar en su conjunto. Básicamente es posible dividir el funcionamiento del sistema completo en tres etapas principales, la comunicación entre el microcontrolador y la PC y comunicación entre el microcontrolador y el FPGA y la comunicación entre cámara termográfica y la PC. Todas las etapas deben funcionar de forma bidireccional ya que deben enviar y recibir paquetes de información ya sea de control o de datos.

La primera área que fue puesta a prueba fue la comunicación SPI, debido a que es una de las partes más importantes del proyecto por representar el modo de comunicación entre el microcontrolador y el FPGA, que es el fin del presente proyecto. Como se comentaba anteriormente debido a que todas las comunicaciones se dan en forma bidireccional, fue necesario realizar pruebas tanto de transmisión de datos como de recepción para el protocolo SPI. La forma más sencilla de corroborar el buen funcionamiento, es programar un dato fijo en el microcontrolador y enviarlo hacia el FPGA y verificar si este último lo recibió de forma adecuada, tomando en cuenta que es importante visualizar mediante el uso de un osciloscopio la forma de onda que se genera esperando se comporte de igual forma como fue programada. En este punto debemos hacer uso de del puerto USB de la tarjeta PLCUAQ816, de tal forma que nos permita leer el dato que se está enviando vía SPI por el microcontrolador y que es recibido por el FPGA. Con la finalidad de hacer esta etapa más evidente y para evitar errores de captura, se integró al microcontrolador una rutina de control para un display de tipo LCD de 16x2 líneas de caracteres alfanuméricos. Cabe recordar que los datos que pueden ser enviados tienen una longitud máxima de 8 bits y que son interpretados como enteros sin signo, es decir, son valores que pueden ir desde el 0 hasta el 255 en base 10. Así por ejemplo si se desea realizar una transmisión con el dato 56 base 10 es esperable encontrar en la señal SDO los dígitos “0011100” que es

su equivalente base 2 en un formato de 8 bits. La figura veintisiete muestra el dato desplegado en el LCD que fue enviado como prueba desde el microcontrolador.

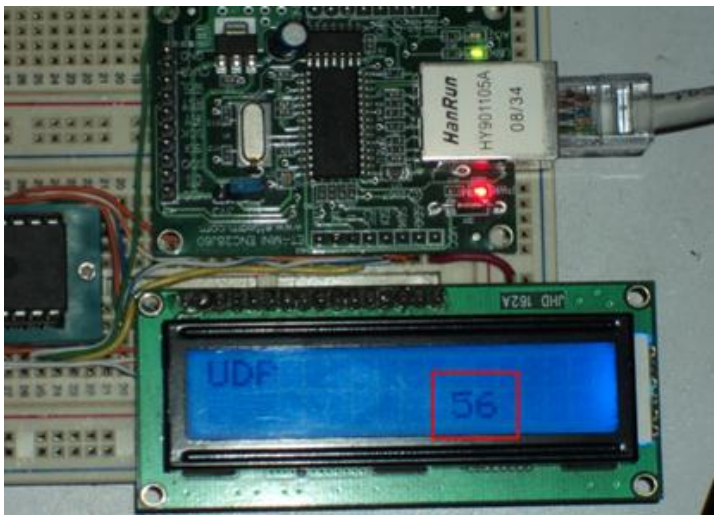

Figura 27. Dato SPI enviado desde el microcontrolador.

La transmisión del dato se hace más evidente en la figura veintiocho donde se puede observar la sincronía entre el la señal de reloj generada por el microcontrolador y la señal serial de salida SDO.

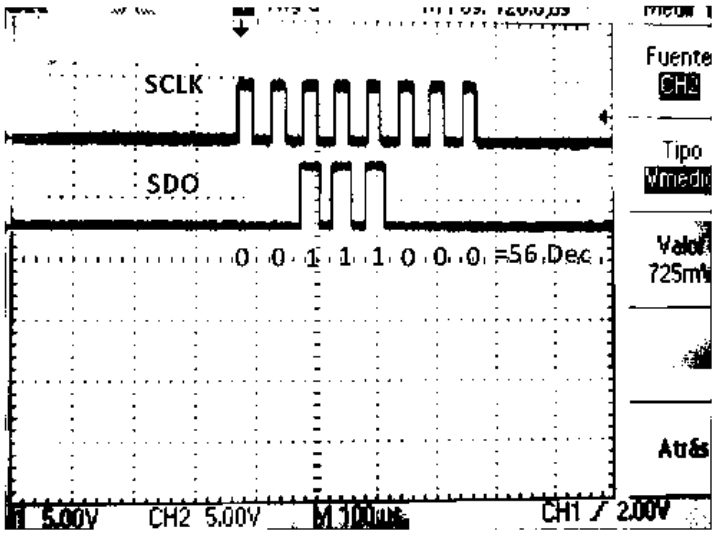

Figura 28. Señal SDO.

Como se puede notar a cada pulso de reloj le corresponde un pulso de la señal SDO ya sea en estado alto o bajo dependiendo del dato que se desea transmitir. Se puede señalar que esta etapa presento muchos retos debido a que la señal de reloj PIC_CLK es de alta frecuencia, de aproximadamente 1.25 MHz, lo que genera inducción electromagnética en las señales de baja frecuencia, entre ellas CS. Debido a que CS es la señal de arranque de ciclo, al presentarse la interferencia provoca que el FPGA interprete esto como si la señal de CS estuviera activa cuando solo se trata de ruido eléctrico y por esta razón comienza un ciclo en falso produciendo datos erróneos y corrimiento de direcciones. Una forma en la que se trató de eliminar este inconveniente fue usar filtros analógicos pero presentan el inconveniente de retrasar la onda, además de deformarla a tal grado que resulta difícil distinguir cuando la señal cambia estado. Un dispositivo electrónico que resultó más útil fue el diodo zener implementado como regulador de voltaje, por su alta velocidad de conmutación. Así mismo se agregaron retardos en el código que validaran que un pulso sostenido durante cierta cantidad de tiempo puede interpretarse realmente como un “1” lógico.

Una vez enviado el dato es posible observar si fue adquirido correctamente por el FPGA, accediendo a este mediante los IP Cores para USB que fueron creados para este propósito. En la figura veintinueve se muestran los datos capturados por el FPGA, de lo que puede observar que fue registrado el número “56” decimal correspondiente al dato enviado previamente.

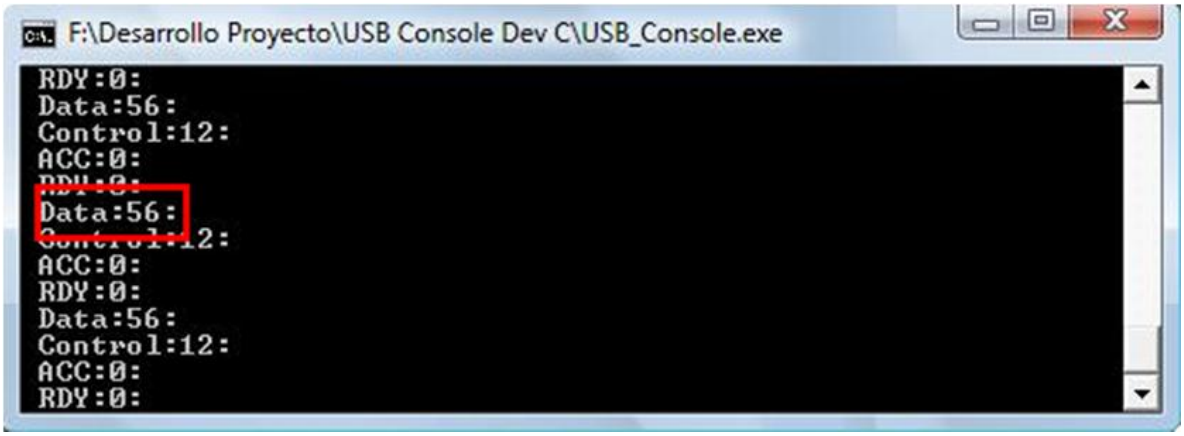

Figura 29. Dato SPI recibido en FPGA.

Una vez corroborada una buena transferencia de la señal SDO se probó una correcta recepción por medio de la señal SDI. La consistencia de esta prueba trata en seguir los mismos pasos de la prueba anterior pero en sentido inverso. En este caso comenzamos depositando un dato en el FPGA para ejemplo el número “124” decimal y obtenemos su equivalencia en binario en formato de entero sin signo de 8 bits que es “01111100”. La figura treinta muestra la captura de este dato.

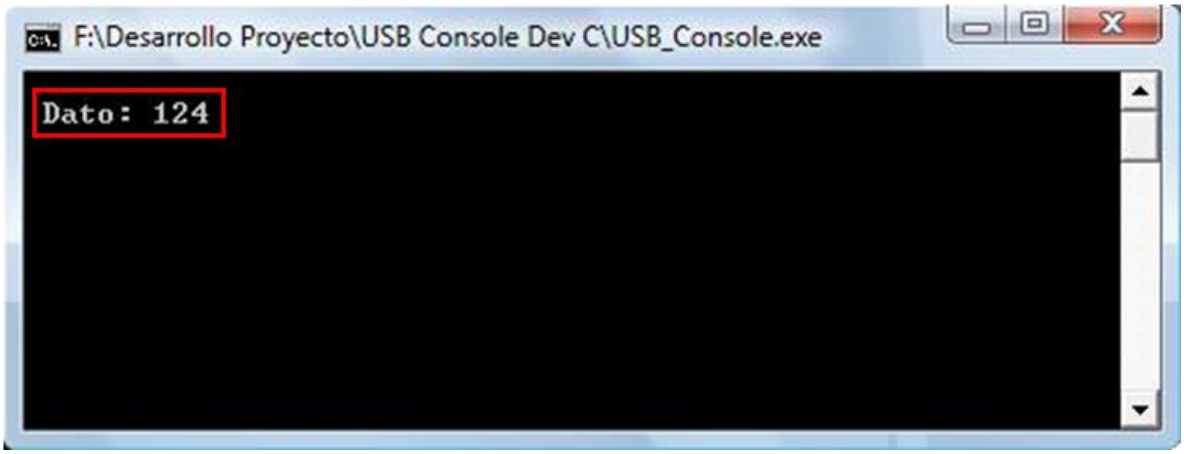

Figura 30. Dato SPI enviado desde FPGA.

Posteriormente el FPGA debe iniciar la secuencia de transferencia hacia el microcontrolador generando un tren de pulsos correspondiente al dato que se desea enviar. La diferencia en este caso es que el FPGA funciona en este momento en modo de esclavo y debe sincronizarse con la señal de pulso de reloj generada por el microcontrolador. Como en el caso de la transferencia a cada pulso de reloj debe corresponder un pulso de señal de entrada serial SDI con un valor de estado en alto o en bajo según corresponda al dato que se envío previamente. Es esperar tomar en cuenta la gran importancia de lograr un sincronización con un alto grado de exactitud ya que de lo contrario es posible que existan perdidas de paquetes o corrimiento de datos. En la figura treinta y uno se muestra como se da esta transferencia. El pulso ancho que aparece en la señal SDI corresponde a la unión de varios pulsos más pequeños.

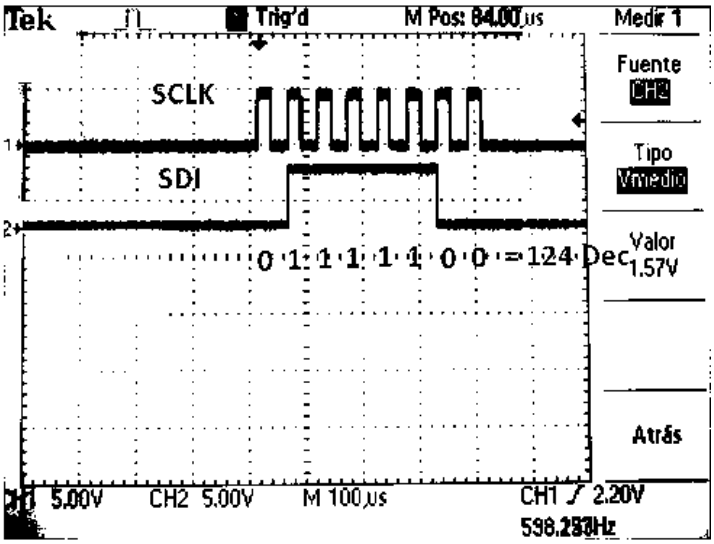

Figura 31. Señal SDI.

Como último paso de esta etapa solo queda verificar en el microcontrolador que la transferencia ha sido realizada con éxito. Nuevamente nos valemos del display LCD adaptado al microcontrolador para poder observar el dato que se ha registrado. En la figura número treinta y dos puede verse el dato para este ejemplo el número “124” base diez que fue capturado por el microcontrolador.

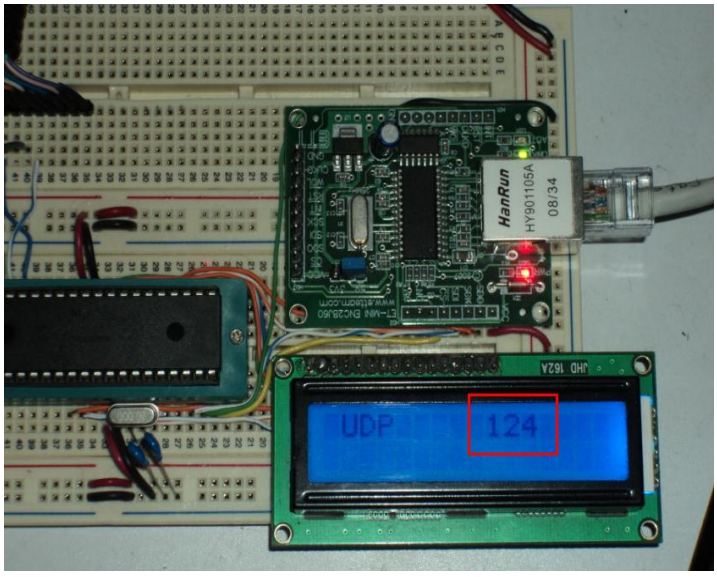

Figura 32. Dato recibido en microcontrolador.

Por la parte de diseño de hardware la cantidad de recursos utilizados para la implementación de las etapas anteriores fue un total de 141 Bloques de Lógica Configurables (Configurable Logic Blocks) también conocidos como Slices de 9312 disponibles, representando esto apenas un 1% de la capacidad de la tarjeta FPGA Spartan3 XC3S500E. Se puede observar de los datos anteriores que la implementación de un protocolo de comunicación Ethernet asistida con un microcontrolador, necesita únicamente cantidades mínimas de memoria que pueden de otra forma aprovecharse en tareas dirigidas a procesos. La figura treinta y tres muestra el número de recursos utilizados en la implementación de la aplicación SPI que aparece en el reporte generado por la herramienta de síntesis.

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	141	9,312	1%
Number of 4 input LUTs	199	9,312	2%
Number of occupied Slices	140	4,656	3%
Number of Slices containing only related logic	140	140	100%
Number of Slices containing unrelated logic	0	140	0%
Total Number of 4 input LUTs	217	9,312	2%
Number used as logic	199		
Number used as a route-thru	18		
Number of bonded IOBs	44	232	18%
Number of BUFGMUXs	1	24	4%
Average Fanout of Non-Clock Nets	3.56		

Figura 33. Uso de Recursos.

Para corroborar la transferencia de datos entre el microcontrolador y la PC, se realizaron pruebas de velocidad de transmisión mediante envíos de paquetes de control de 32 Bytes obteniendo como resultado velocidades promedio de 256 Kbps sin presentar ninguna pérdida. Para realizar dichas pruebas se hizo uso del comando Ping (Packet Internet Groper) que es un rastreador paquetes en redes Ethernet, que comprueba el estado de la conexión del host local con uno o varios equipos remotos de una red, por medio del envío de paquetes de solicitud y de

respuesta y que viene incluido de forma predeterminada entre las herramientas de MS-DOS. La utilización de esta herramienta puede ser observada en la figura treinta y cuatro.

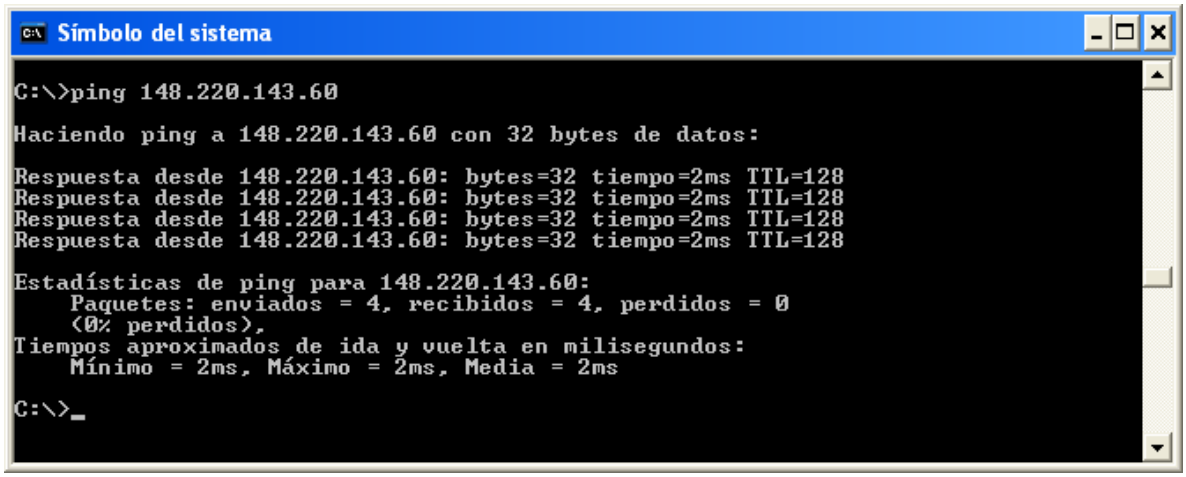

Figura 34. Envío de datos de control.

Así mismo se llevaron a cabo múltiples pruebas de comunicación mediante el envío masivo de datos, específicamente para transferencias de 3600 Bytes, dando como respuesta velocidades de hasta 1.2 Mbps. En la figura número treinta y cinco se pueden observar treinta ensayos de transferencia masiva de datos que fueron calculados a partir del tiempo de respuesta entre la trasmisión y captura de de paquetes de información, mediante la utilización de la herramienta de envío de paquetes de datos sendData y recepción de datos getData, del lenguaje de programación Visual Basic 6. La prueba consistió en enviar paquetes de 3600 Bytes de longitud alojándolos en el microcontrolador y que acto seguido regresa la tramada de datos a la interfaz de Visual Basic, en el menor tiempo posible, sin llevar acabo procesamiento alguno. Al momento de iniciar el envío se activó de forma simultánea un temporizador, que fue el responsable de determinar el tiempo que transcurrió en volver el dato después de su transmisión. Como se puede notar la tasa de transferencia arrojada por esta prueba se comporta de forma estable, con ligeras oscilaciones entre 1 Mbps y 1.2 Mbps.

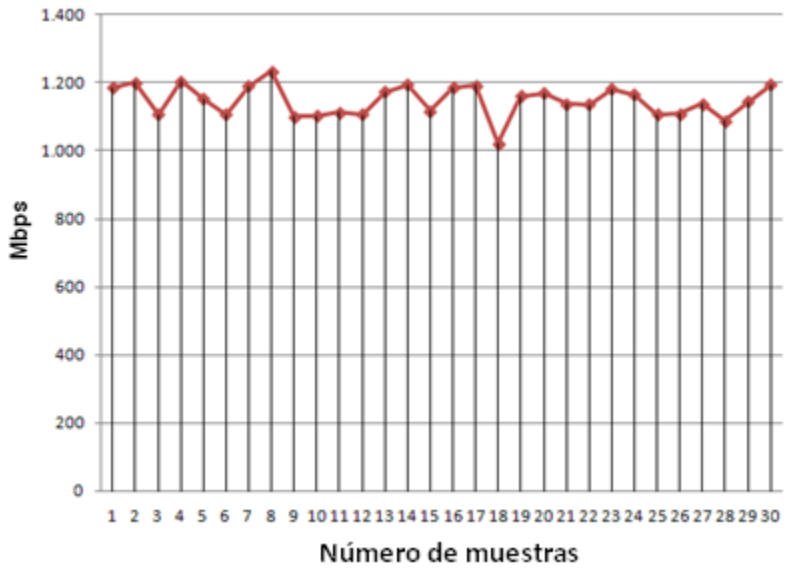

Figura 35. Tasa de transferencia.

Por la parte de la comunicación entre la cámara termográfica y la computadora personal, solo fue necesario utilizar y corroborar el buen funcionamiento del ActiveX que proporciona FLIR para el control de sus cámaras termográficas, mediante el uso de algún lenguaje de programación, de forma precisa para este caso mediante Visual Basic 6. Para este fin solamente es necesario ejecutar el paquete de instalación para que la librería sea automáticamente reconocida. Realizado este paso se debe agregar el ActiveX como componente en la pestaña de proyecto de Visual Basic. Al momento de ejecutar el código aparece un cuadro de dialogo, que nos permite seleccionar que tipo de conexión deseamos establecer, para el caso de la cámara A310 seleccionamos el tipo A320 con configuración de comunicación de tipo Ethernet por ser el que más se asemeja a nuestro dispositivo, al no existir exactamente uno para este. En la figura treinta y seis se puede observar el cuadro de dialogo de selección que de forma específica se encuentra en la pestaña denominada “Main”. Este es el menú principal ya que debe ser inicializado antes de tratar de realizar ninguna otra acción con la cámara termográfica. En ella como se puede observar un botón de conexión y un cuadro de texto con el estatus actual del dispositivo, así como la información de la configuración.

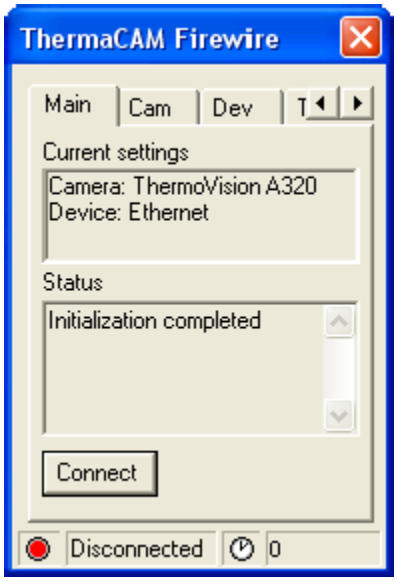

Figura 36. Menú principal.

Al momento de oprimir el botón de conexión, si los dispositivos de la red se encuentran físicamente presentes, aparecerá una ventana emergente que muestra todos los dispositivos que fueron detectados automáticamente pero únicamente los que representan alguna cámara termográfica FLIR, así como su dirección IP que los identifica como dispositivos únicos dentro de la red. Se debe tener cuidado de seleccionar el dispositivo adecuado y no confundirlo con otro. En la figura treinta y siete se hace notar el cuadro emergente con dos dispositivos presentes en la red, que como se puede observar ambos son de tipo A310. La franja azul representa cual de los dispositivos esta seleccionado.

Figura 37. Selección de dispositivo.

Si la conexión ha resultado exitosa podremos observar como el indicador ubicado en la parte inferior izquierda, cambia de estado de desconectado a conectado, mostrando que el dispositivo está listo para ser manipulado. Si la conexión fallara por algún motivo, el indicador regresa a su estado original. La siguiente pestaña denominada “Camera” que se muestra en la figura treinta y ocho contiene menús desplegables, que nos permiten por ejemplo seleccionar el rango absoluto de temperatura de trabajo de la cámara, aplicar la corrección de de imagen, reducción de ruido en la misma o cambiar el foco del lente.

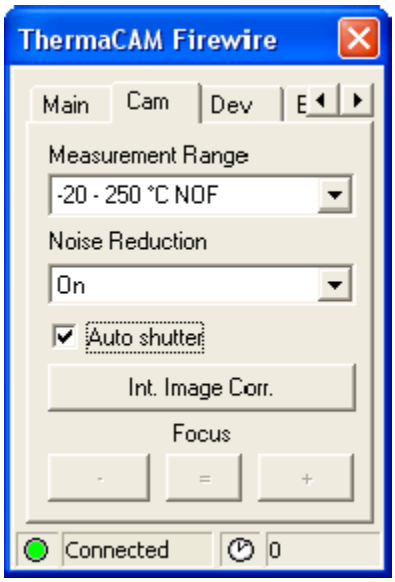

Figura 38. Menú cámara.

El último menú seleccionable es el de “Device”, que nos permite por ejemplo seleccionar la frecuencia de captura de los cuadros o imágenes, obtener información específica de la cámara termográfica, como puede ser su número de serie y la revisión de firmware que se encuentra instalado, un informe detallado del estatus y más importante aún, permite mediante el botón “Settings” realizar modificaciones de parámetros de emisividad, reflexión o temperatura ambiente así como modificar el sistema de unidades de medida, ya sea sistema internacional o inglés según convenga al operario. Todas estas características pueden verse dentro del cuadro de dialogo mostrado en la figura numero treinta y nueve.

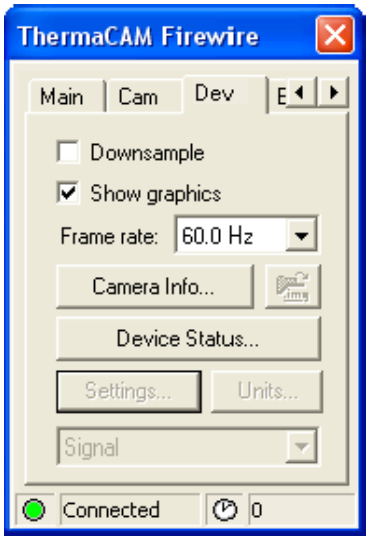

Figura 39. Menú dispositivo.

Finalmente se realizaron pruebas al sistema en su conjunto utilizando todas las herramientas descritas en los capítulos anteriores. Para ello se diseñó una interfaz de usuario, que es responsable de coordinar todas las etapas de procesamiento de datos. Así por ejemplo dicha interfaz se encarga de establecer comunicación con la cámara termográfica y dar un comando de captura de imagen, así mismo permite realizar ajustes como enfoque o modificar el índice de emisividad o distancia al objeto de estudio, para obtener imágenes de mayor calidad. También se añadió la opción de utilizar imágenes provenientes de un archivo de imagen, en caso que se desee procesar imágenes sin conexión a la cámara termográfica. Entre otras características la consola posee dos recuadros o "Frames" donde se localizan en el primer caso, la imagen que desea ser procesada y para el segundo caso, la imagen después de haber sido procesada. El tipo de procesamiento que es aplicado viene como resultado de la descripción de hardware que haya sido sintetizada en el FPGA. Para este efecto se crearon tres tipos diferentes de procesamiento, el primero simplemente regresa la imagen de igual forma como fue adquirida sin realizar modificación alguna, es una simple prueba de control para determinar si los datos han sido adquiridos de forma adecuada, la segunda es la generación del negativo de la imagen original, este

procesamiento se logra, de forma simple, al restar al valor 255 decimal el valor de cada pixel de la imagen, de este modo los valores más altos correspondientes a los tonos blancos de una imagen producen un menor residuo de dicha diferencia y los valores más bajos producen un mayor residuo, en realidad se trata de un procesamiento sencillo que solo requiere el uso mínimo de recursos. En las imágenes cuarenta y cuarenta y uno es claramente visible este efecto, en ellas se puede notar como las partes claras de la imagen se tornan obscuras y como de forma reciproca las partes obscuras se vuelven claras.

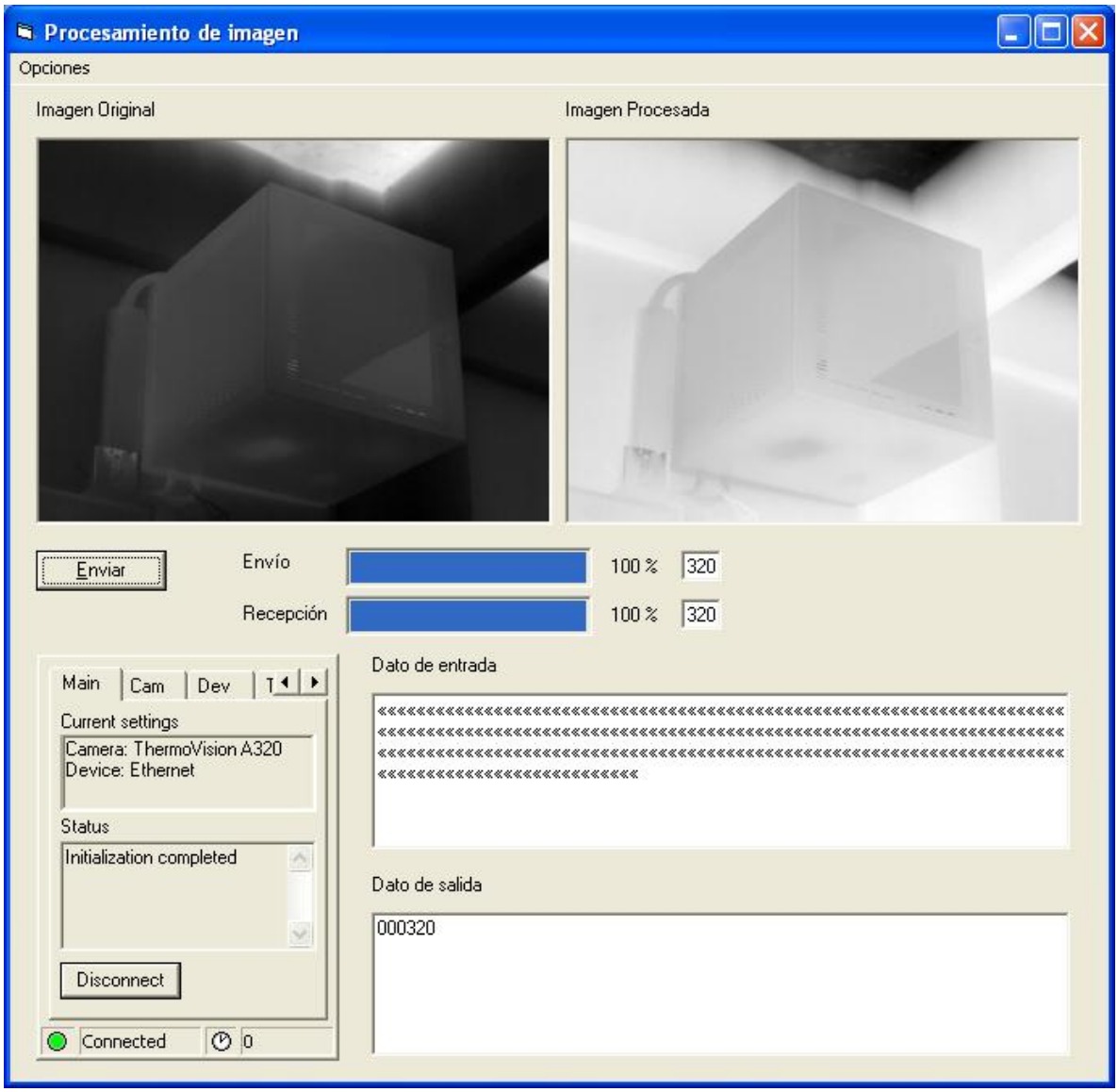

Figura 40. Procesamiento de imagen No. 1. Negativo.

Originalmente las tonalidades claras representan los puntos más calientes de una imagen y las tonalidades obscuras representan los puntos más fríos. La imagen cuarenta presenta un enrutador de redes dentro de un gabinete de acero, donde se puede observar que por dentro es ligeramente más caliente que en el exterior, de igual manera se puede notar que la lámpara encima del gabinete produce una mayor cantidad de calor que en las otras zonas de la imagen. En la imagen procesada se observa como las zonas calientes son ahora son representadas por zonas frías y las zonas frías por calientes.

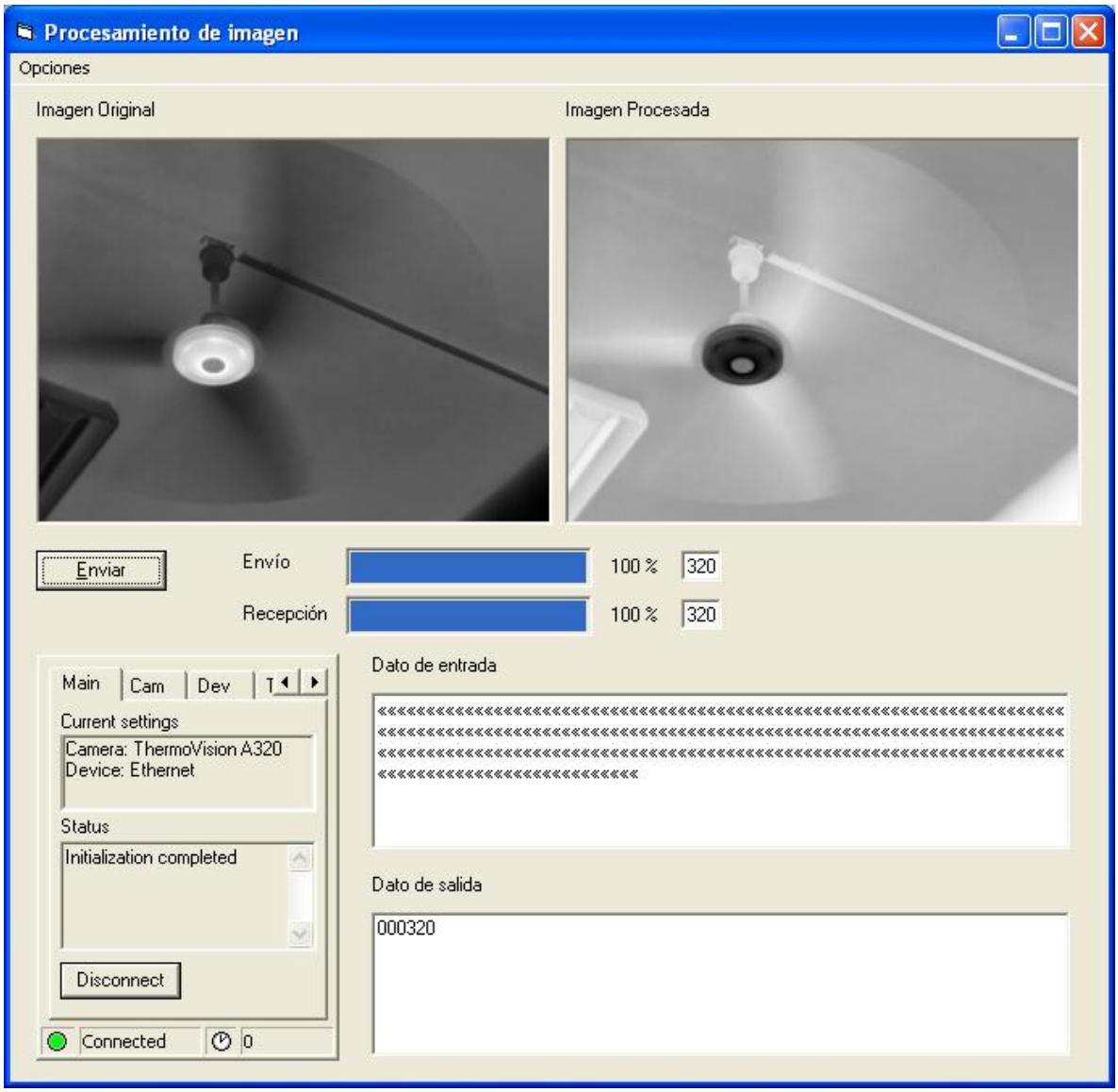

Figura 41. Procesamiento de imagen No. 2. Negativo.

La figura cuarenta y uno muestra la termografía de un ventilador en movimiento. Como se puede observar en ella, la zona donde se localiza el motor contrasta fuertemente con el fondo, además se hace notar que las aspas, debido al constante contacto con el aire, son mucho frías. Las figuras cuarenta y dos y cuarenta y tres muestran las mismas imágenes tomadas anteriormente con la cámara termográfica pero cambiando el tipo de procesamiento por un “Threshold” o mejor conocido como umbralizado, con un límite de fijado en un valor de “128” decimal.

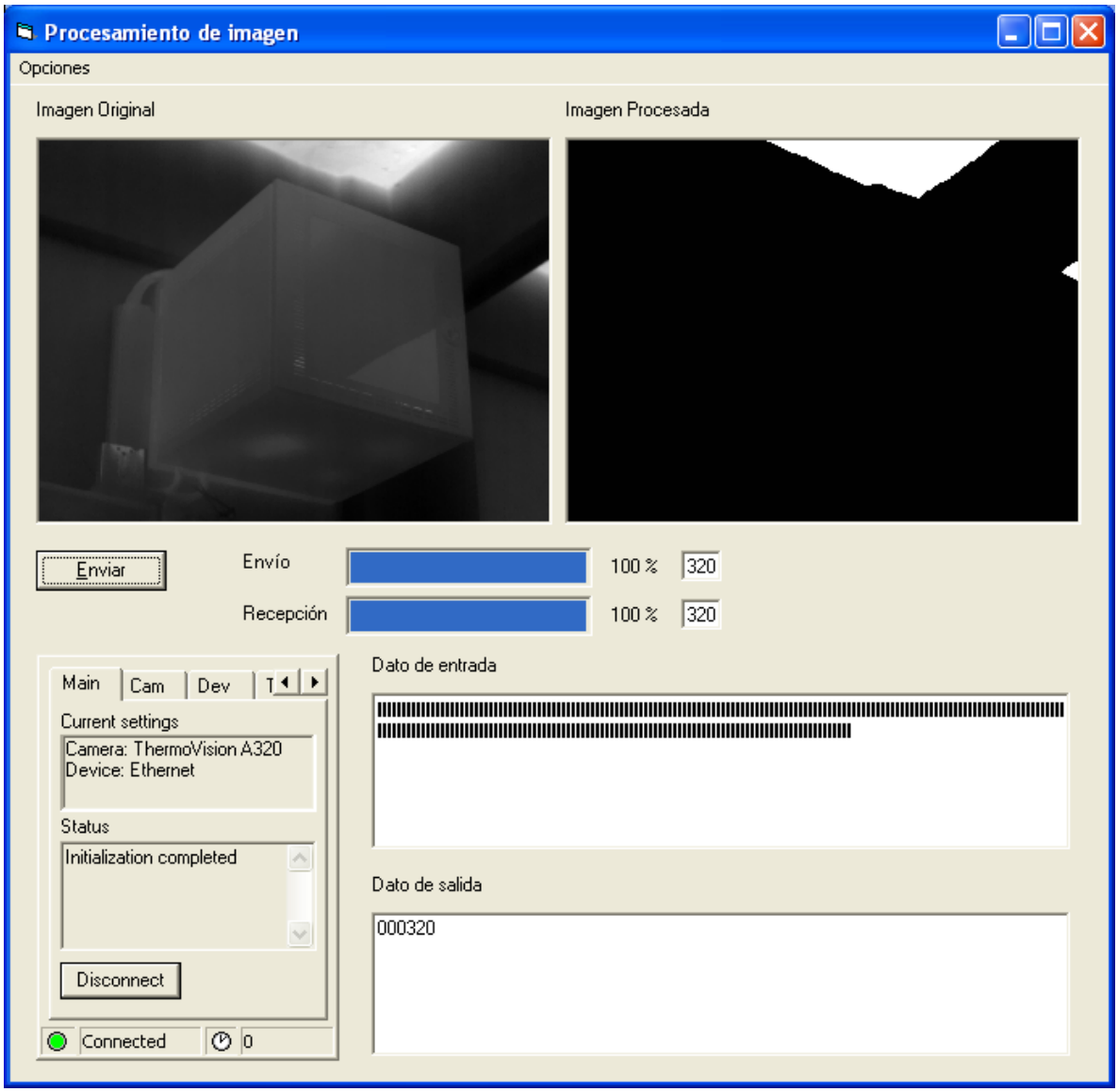

Figura 42. Procesamiento de imagen No. 1. Umbralizado.

De este modo los valores de los datos de cada pixel de la imagen serán redondeados a un valor de “255” cuando su valor es igual o superior a “128” y redondeados a un valor de cero cuando son menores a “128”. El efecto producido muestra como solo las zonas que se encuentran más calientes o que son de un blanco más intenso son representadas en la imagen procesada. Esta función puede ser utilizada en una operación de monitoreo, como un filtro que indique solo las zonas que han superado un umbral de cierta temperatura y que por tanto, se pueden considerar susceptibles a fallas inminentes.

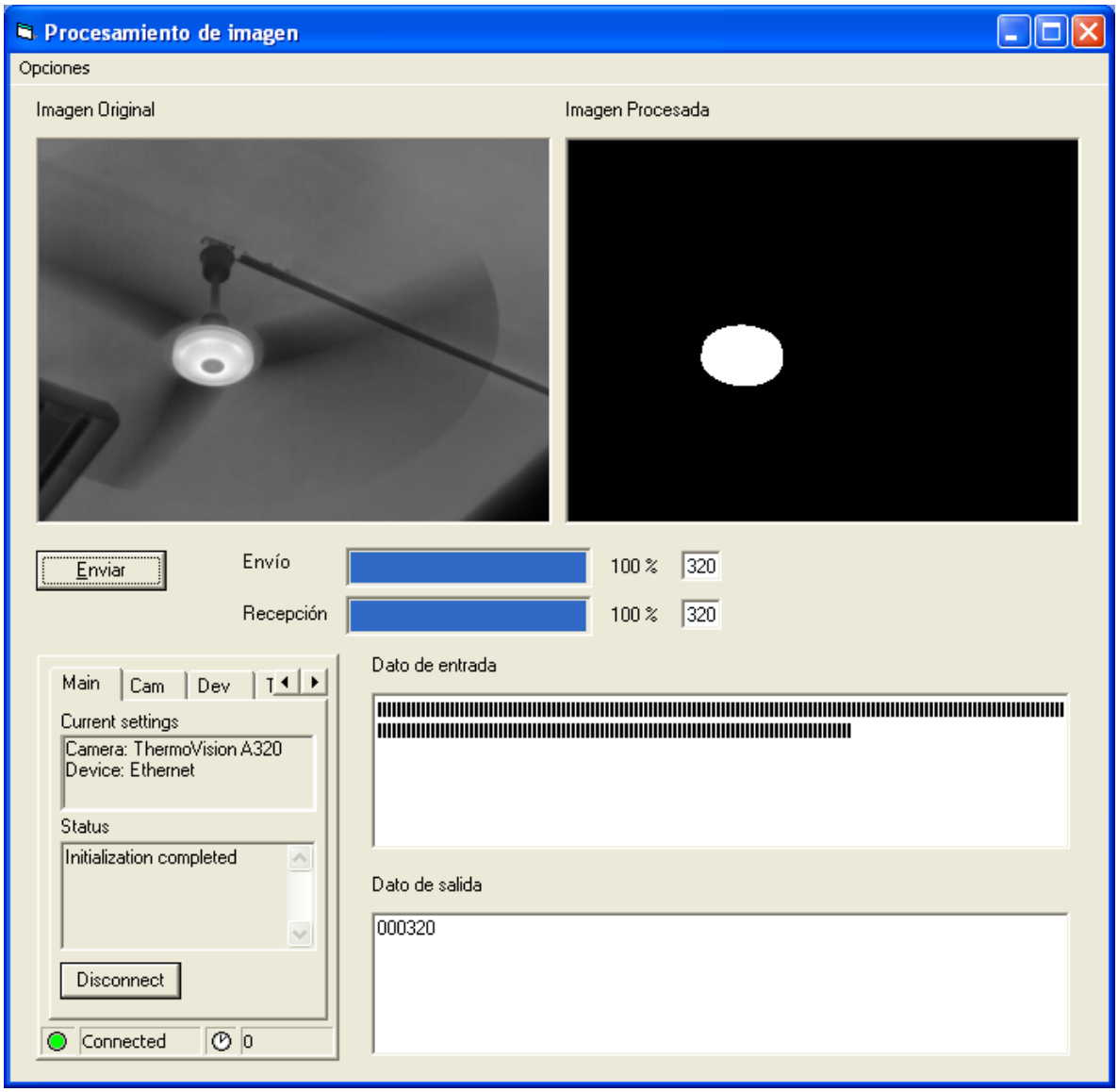

Figura 43. Procesamiento de imagen No. 2. Umbralizado.

De la imagen cuarenta y dos podemos hacer notar que solo la zona ocupada por las lámparas es la única que se logró superar el umbral, interpretándose que es la zona más caliente de la imagen, igualmente en la imagen cuarenta y tres es posible apreciar, que solo la parte que comprende la parte del motor se ve reflejada en la imagen procesada y como es lógico pensar debe ser la parte más caliente por ser una pieza sometida a fricción constante.

La transferencia de información incluyendo la etapa de transmisión y recepción de las imágenes tomando en cuenta la transmisión mediante el protocolo Ethernet y SPI, se logró en un tiempo aproximado de 12 segundos con un total de 153600 Bytes transmitidos, logrando con ello una tasa de transferencia de 102.4 Kbps, valor que puede ser observado en la figura cuarenta y cuatro para un número total de cincuenta muestras. Este valor es sensiblemente menor que los valores obtenidos en las pruebas obtenidas con anterioridad, para el intercambio de comunicación entre el microcontrolador y la PC pero hay que tener en cuenta que el sistema en su conjunto se ve reducido por la limitante de transmisión de datos vía protocolo SPI entre el FPGA y el microcontrolador y el tiempo que le lleva al controlador realizar los dos procesos.

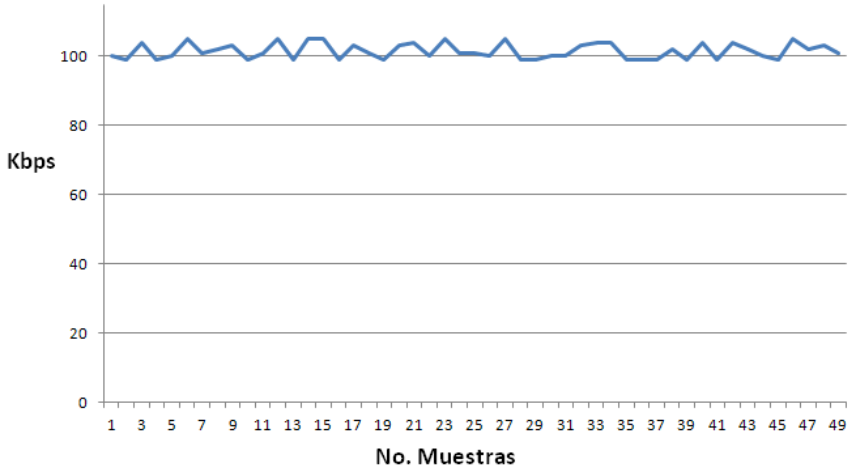

Figura 44. Tasa de transferencia total.

5. CONCLUSIONES

La utilización e implementación de una plataforma de comunicación estándar, robusta y ampliamente extendida como lo es Ethernet, para la comunicación entre dispositivos electrónicos de alto desempeño como los son los FPGA, permite un gran número de posibilidades para aplicaciones de control y monitoreo, donde el intercambio de datos de forma confiable y de alta velocidad es una prioridad, con la ventaja de ser un desarrollo de tipo modular que puede expandirse y ser ajustado a la medida, para aplicaciones especificas y más complejas. La fusión de diferentes tecnologías dedicadas a procesos especializados, presenta grandes ventajas a la simple utilización de las capacidades independientes de los dispositivos, potencializando con esto los ambientes en los que pueden ser empleados.

Es posible observar en este proyecto la gran capacidad que tienen los FPGA como procesadores de señales e imágenes, debido a su arquitectura de ejecución paralela, que permite disminuir considerablemente los tiempos de procesamiento. Es destacable también hacer notar las múltiples aplicaciones que se pueden dar a los FPGA cuando se les es añadido un medio de comunicación que les permite la transferencia de datos a otros dispositivos, de este modo su utilización puede expandirse a cualquiera que requiera un procesamiento de alta velocidad con posibilidad de transferencia de datos a uno o varios dispositivos. Así por ejemplo pueden darse todo tipo de aplicaciones industriales, como monitoreo en tiempo real, coordinación de controles para maquinaria, generación de reportes, entre otros. Cabe agregar que los circuitos integrados utilizados en este proyecto son dispositivos de fácil adquisición y de buenas prestaciones, que si bien no presentan las velocidades de transmisión más altas disponibles actualmente mediante tecnología de avanzada, permiten un útil medio de transmisión superando por mucho a tecnologías como RS232 o RS485 y siendo equiparables a USB 1.0, con la ventaja de no presentar desconexiones además de permitir la

generación de redes, que permiten el acceso a los dispositivos desde diferentes terminales.

El presente trabajo demuestra que es posible crear una alternativa en la generación de soluciones, con independencia tecnológica, de bajo costo y de optimización de recursos económicos y de memoria, con amplia adaptabilidad a aplicaciones específicas, en el diseño de plataformas de comunicación Ethernet para procesos de automatización y monitoreo, con respecto a las soluciones ofrecidas por los dispositivos de comunicación comerciales.

6. PROSPECTIVA

Como se comento con anterioridad el presente proyecto tienen una gama muy amplia de aplicaciones en las que puede ser adaptado para fines específicos, esta circunstancia vuelve aun más importante mencionar que son varias las mejoras que pueden ser realizadas para obtener un mejor desempeño. Así por ejemplo durante el desarrollo de las partes que integran el sistema, nos encontramos con la problemática de la velocidad de transferencia que se ve disminuida por el tiempo que le lleva al microcontrolador hacer la interpretación entre el protocolo Ethernet y después convertirlo a protocolo SPI, para llevar a cabo la comunicación con el FPGA. El protocolo SPI de modo general trabaja en promedio a velocidades de no más de 10 Mbps, en contra parte al protocolo Ethernet que puede hacerlo, dependiendo el estándar, desde 100 Mbps hasta aproximadamente 1 Gbps, como se puede notar de estas cifras esto deriva en un cuello de botella que limita el total aprovechamiento del protocolo Ethernet. Una forma lógica de solventar este problema, consistiría en lograr una transferencia más veloz, mediante el protocolo SPI aumentando la frecuencia del ciclo de reloj que para este proyecto fue de 1.25 MHz, sin embargo existe la desventaja de que a una mayor frecuencia de muestreo la señal se vuelve mas susceptible a interferencia electromagnética, además de provocar que las formas de onda se deformen, causando a su vez que una mayor número de paquetes de datos sean interpretados de forma errónea.

A pesar de que esta circunstancia presenta un sólido inconveniente, podría verse reducida, mediante el uso de filtros analógicos activos o con la aplicación de filtros digitales y de este modo mitigar la interferencia electromagnética y consecuentemente, aplicando algún tipo de buffer electrónico para disminuir las caídas de tensión. Por otra parte si se deseara eliminar de raíz este problema la recomendación obvia es omitir la etapa del microcontrolador y tratar de encontrar un dispositivo que pueda permitir una conexión de mayor velocidad, mediante la aplicación de otro protocolo de comunicación como PCI (Peripheral Component

Interconnect, Interconexión de Componentes Periféricos), que tiene la ventaja de permitir una conexión más directa con los FPGA, debido a que son tarjetas producidas por los mismos fabricantes, pensados para esta aplicación especifica, aunque llevan consigo la desventaja de ser sistemas de arquitectura cerrada además de elevar considerablemente los costos. Dicho de otro modo quedaría sujeto a establecer para cada aplicación si la velocidad de transferencia en una relación costo-beneficio resulta más importante que el costo del sistema.

7.1 APENDICE A

7.1.1 MÓDULO SPI

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity SPI_Control is
	------------------------------
	generic(n: integer := 8);
	------------------------------
	port(
		RST : in STD_LOGIC;
		CLK : in STD_LOGIC;
		PIC_CLK : in STD_LOGIC;
		CS : in STD_LOGIC;
		RFQ_IN : in STD_LOGIC;
		SDI : in STD_LOGIC;
		DI : in STD_LOGIC_VECTOR(n-1 downto 0);
		SDO : out STD_LOGIC;
		DO : out STD_LOGIC_VECTOR(n-1 downto 0);
		RFQ_OUT : out STD_LOGIC;
		RDY : out STD_LOGIC;
		SDI_DLY		: out STD_LOGIC;
		SCLK_OUT : out STD_LOGIC;
		ACC : out STD_LOGIC_VECTOR(n-1 downto 0);
		SCS_OUT : out STD_LOGIC
	  );
end SPI_Control;

architecture SPI_Control of SPI_Control is

component SPI_CLK_SYNC is
	 port(
		RST : in STD_LOGIC;
		CLK : in STD_LOGIC;
		IN_SYNC : in STD_LOGIC;
		OUT_SYNC : out STD_LOGIC
	  );
end component;

component SPI_CS_Control_FLR is
```

```
	-------------------------------
	generic(n: integer := 8);
	-------------------------------
	port(
		RST : in STD_LOGIC;
		CLK : in STD_LOGIC;
		CS_IN : in STD_LOGIC;
		CS_FR : out STD_LOGIC
	  );
end component;

component SPI_Counter_Enabled is
	-------------------------------
	generic(n: integer := 8);
	-------------------------------
	port(
		RST : in STD_LOGIC;
		CLK : in STD_LOGIC;
		OPR : in STD_LOGIC_VECTOR(1 downto 0);
		PRS : out STD_LOGIC;
		DN : out STD_LOGIC;
		ACC : out STD_LOGIC_VECTOR(n-1 downto 0)
	  );
end component;

component SPI_Reg_SP is
	-------------------------------
	generic(n:integer:=8);
	-------------------------------
	port(
		RST : in STD_LOGIC;
		CLK : in STD_LOGIC;
		OPR : in STD_LOGIC;
		Di : in STD_LOGIC;
		Do : out STD_LOGIC_VECTOR(n-1 downto 0)
	   );
end component;

component SPI_Reg_PS is
	-------------------------------
	generic(n:integer:=8);
	-------------------------------
	port(
		RST : in STD_LOGIC;
		CLK : in STD_LOGIC;
		OPR : in STD_LOGIC_VECTOR(1 downto 0);
		Di : in STD_LOGIC_VECTOR(n-1 downto 0);
		Do : out STD_LOGIC
```

```
            );
end component;

component SPI_Reg_Enabled is
        -------------------------------
        generic(n: integer := 8);
        -------------------------------
        port(
                RST : in STD_LOGIC;
                CLK : in STD_LOGIC;
                OPR : in STD_LOGIC;
                Di : in STD_LOGIC_VECTOR(n-1 downto 0);
                Do : out STD_LOGIC_VECTOR(n-1 downto 0)
          );
end component;

component DLY_Control is
        -------------------------------
        generic(n: integer := 8);
        -------------------------------
        port(
                RST : in STD_LOGIC;
                CLK : in STD_LOGIC;
                SDI : in STD_LOGIC;
                SDI_DLY : out STD_LOGIC
          );
end component;

component SPI_FSM is
        port(
                RST : in STD_LOGIC;
                CLK : in STD_LOGIC;
                SCLK: in STD_LOGIC;
                CS : in STD_LOGIC;
                PRS : in STD_LOGIC;
                DN : in STD_LOGIC;
                OPR : out STD_LOGIC_VECTOR(1 downto 0);
                CI1 : out STD_LOGIC;
                CI2 : out STD_LOGIC;
                CO1 : out STD_LOGIC;
                CO2 : out STD_LOGIC_VECTOR(1 downto 0);
                RDY : out STD_LOGIC
                );
end component;

-------------------------------------------------------------
component SPI_RDWR_Control_FLR is
        -------------------------------
```

```
        generic(n: integer := 8);
        ------------------------------
        port(
                RST : in STD_LOGIC;
                CLK : in STD_LOGIC;
                RDWR_IN : in STD_LOGIC;
                RDWR_FR : out STD_LOGIC
          );
end component;
-------------------------------------------------------------
component Signal_Control is
        port(
                RST : in STD_LOGIC;
                CLK : in STD_LOGIC;
                SG : in STD_LOGIC;
                SG_FR : out STD_LOGIC
          );
end component;
-------------------------------------------------------------

signal SCLK : STD_LOGIC;
signal OPR : STD_LOGIC_VECTOR(1 downto 0);
signal PRS : STD_LOGIC;
signal DN : STD_LOGIC;
signal CI1 : STD_LOGIC;
signal CI2 : STD_LOGIC;
signal CO1 : STD_LOGIC;
signal CO2 : STD_LOGIC_VECTOR(1 downto 0);
signal SDI_SG : STD_LOGIC_VECTOR(n-1 downto 0);
signal SDO_SG : STD_LOGIC;
signal DI_SG : STD_LOGIC_VECTOR(n-1 downto 0);
signal SDI_DLY_SG : STD_LOGIC;
signal SCS : STD_LOGIC;
signal RDY_SG : STD_LOGIC;
signal SCS_CLK : STD_LOGIC;

signal SDI_22 : STD_LOGIC;
signal SDI_DLY_SG_22 : STD_LOGIC;
signal SG_IN : STD_LOGIC;

--signal DI : STD_LOGIC_VECTOR(n-1 downto 0);
--signal DO : STD_LOGIC_VECTOR(n-1 downto 0);

begin

-- Sincronizacion de PIC_CLK con flanco de subida de CLK de un pulso de reloj de duracion
Bloque_1 : SPI_CLK_SYNC port map(
RST=>RST,
```

```
CLK=>CLK,
IN_SYNC=>PIC_CLK,
OUT_SYNC=>SCLK);

-- Salida por hardware de SCLK
SCLK_OUT<=SCLK;

SCS_CLK <= '0' when RST='1' else CS when rising_edge(CLK);

-- Sincronizacion de CS con flanco de subida de CLK        y filtro de tiempo en bajo
Bloque_2 : SPI_CS_Control_FLR generic map (n) port map(
RST=>RST,
CLK=>CLK,
CS_IN=>SCS_CLK,
--CS_FR=>SCS);
CS_FR=>SG_IN);

-------------------------------------------------------------
-- Filtro de tiempo para picos de 80 ns
Bloque_2FR : Signal_Control port map(
RST=>RST,
CLK=>CLK,
SG=>SG_IN,
SG_FR=>SCS);
-------------------------------------------------------------

SCS_OUT<=SCS;

-- Sincronizacion de SDI con flanco de subida de CLK e incremento de tamaño de pulso
Bloque_3_22 : DLY_Control generic map (n) port map(
RST=>RST,
CLK=>CLK,
SDI=>SDI_22,
SDI_DLY=>SDI_DLY_SG_22);

-------------------------------------------------------------
Bloque_3 : SPI_RDWR_Control_FLR generic map (n) port map(
RST=>RST,
CLK=>CLK,
RDWR_IN=>SDI,
RDWR_FR=>SDI_DLY_SG);
-------------------------------------------------------------

-- Salida por hardware de SDI_DLY
SDI_DLY<=SDI_DLY_SG;

Bloque_4 : SPI_Counter_Enabled generic map (n) port map(
RST=>RST,
```

```
CLK=>CLK,
OPR=>OPR,
PRS=>PRS,
DN=>DN,
ACC=>ACC);

Bloque_5 : SPI_Reg_SP generic map (n) port map(
RST=>RST,
CLK=>CLK,
OPR=>CI2,
--Di=>SDI,
Di=>SDI_DLY_SG,
Do=>SDI_SG);

Bloque_6 : SPI_Reg_Enabled generic map (n) port map(
RST=>RST,
CLK=>CLK,
OPR=>CI1,
Di=>SDI_SG,
Do=>DO);

Bloque_7 : SPI_Reg_Enabled generic map (n) port map(
RST=>RST,
CLK=>CLK,
OPR=>CO1,
Di=>DI,
Do=>DI_SG);

Bloque_8 : SPI_Reg_PS generic map (n) port map(
RST=>RST,
CLK=>CLK,
OPR=>CO2,
Di=>DI_SG,
Do=>SDO_SG);

SDO<=(SDO_SG and not(RDY_SG));
--SDO<=SDO_SG;

Bloque_9 : SPI_FSM port map(
RST=>RST,
CLK=>CLK,
SCLK=>SCLK,
CS=>SCS,
PRS=>PRS,
DN=>DN,
OPR=>OPR,
CI1=>CI1,
CI2=>CI2,
```

```
CO1=>CO1,
CO2=>CO2,
RDY=>RDY_SG);

RDY<=RDY_SG;

RFQ_OUT<=RFQ_IN;

end SPI_Control;
```

7.1.2 MÓDULO SRAM

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity SRAM_Control is
	--------------------------------
	generic(n: integer := 8);
	--------------------------------
	port(
		-- Configuracion
		RST : in STD_LOGIC;
		CLK : in STD_LOGIC;
		-- SRAM
		-- Control
		RD: in STD_LOGIC;
		WR: in STD_LOGIC;
		RDY_SPI: in STD_LOGIC;
		CE: out STD_LOGIC;
		OE: out STD_LOGIC;
		WE: out STD_LOGIC;
		RDY_SRAM: out STD_LOGIC;
		-- Datos
		IO: inout STD_LOGIC_VECTOR(7 downto 0);
		DI: in STD_LOGIC_VECTOR(7 downto 0);
		DO: out STD_LOGIC_VECTOR(7 downto 0);
		-- Direccion
		A0: out STD_LOGIC;
		A1: out STD_LOGIC;
		A2: out STD_LOGIC;
		A3: out STD_LOGIC;
		A4: out STD_LOGIC;
```

```
                A5: out STD_LOGIC;
                A6: out STD_LOGIC;
                A7: out STD_LOGIC;
                A8: out STD_LOGIC;
                A9: out STD_LOGIC;
                A10: out STD_LOGIC;
                A11: out STD_LOGIC;
                A12: out STD_LOGIC;
                A13: out STD_LOGIC;
                A14: out STD_LOGIC;
                A15: out STD_LOGIC;
                A16: out STD_LOGIC;
                A17: out STD_LOGIC;
                A18: out STD_LOGIC
                );
end SRAM_Control;

architecture SRAM_Control of SRAM_Control is

component SRAM_Counter_Enabled is
        --------------------------------
        generic(n: integer := 19);
        --------------------------------
        port(
                RST : in STD_LOGIC;
                CLK : in STD_LOGIC;
                OPR : in STD_LOGIC_VECTOR(1 downto 0);
                DN : out STD_LOGIC;
                ACC : out STD_LOGIC_VECTOR(n-1 downto 0)
          );
end component;

component SRAM_Buffer_3E is
        --------------------------------
        generic(n: integer := 8);
        --------------------------------
        port(
                OPC : in STD_LOGIC;
                DI : in STD_LOGIC_VECTOR(n-1 downto 0);
                IO : inout STD_LOGIC_VECTOR(n-1 downto 0)
                );
end component;
```

```
component SRAM_Reg_Enabled is
        ------------------------------
        generic(n: integer := 8);
        ------------------------------
        port(
                RST : in STD_LOGIC;
                CLK : in STD_LOGIC;
                OPR : in STD_LOGIC;
                Di : in STD_LOGIC_VECTOR(n-1 downto 0);
                Do : out STD_LOGIC_VECTOR(n-1 downto 0)
          );
end component;

component SRAM_RDY_SPI_SYNC is
       port(
                RST : in STD_LOGIC;
                CLK : in STD_LOGIC;
                IN_SYNC : in STD_LOGIC;
                OUT_SYNC : out STD_LOGIC
          );
end component;

component SRAM_IN_SYNC is
        port(
                RST : in STD_LOGIC;
                CLK : in STD_LOGIC;
                S_IN : in STD_LOGIC;
                S_OUT : out STD_LOGIC
                );
end component;

component SRAM_FSM is
        port(
                RST : in STD_LOGIC;
                CLK : in STD_LOGIC;
                WR : in STD_LOGIC;
                RD: in STD_LOGIC;
                RDY_SPI : in STD_LOGIC;
                DN : in STD_LOGIC;
                OPR : out STD_LOGIC_VECTOR(1 downto 0);
                OPC : out STD_LOGIC;
```

```
                RDY_SRAM : out STD_LOGIC;
                RDY_DO : out STD_LOGIC;
                WE : out STD_LOGIC;
                CE : out STD_LOGIC;
                OE : out STD_LOGIC
                );
end component;

signal WR_SYNC : STD_LOGIC;
signal RD_SYNC : STD_LOGIC;
signal RDY_SPI_SYNC : STD_LOGIC;
signal OPR : STD_LOGIC_VECTOR(1 downto 0);
signal OPC : STD_LOGIC;
signal DN : STD_LOGIC;
signal ACC : STD_LOGIC_VECTOR(18 downto 0);
signal RDY_DO : STD_LOGIC;

begin

Bloque_1 : SRAM_IN_SYNC port map(
RST=>RST,
CLK=>CLK,
S_IN=>WR,
S_OUT=>WR_SYNC);

Bloque_2 : SRAM_IN_SYNC port map(
RST=>RST,
CLK=>CLK,
S_IN=>RD,
S_OUT=>RD_SYNC);

Bloque_3 : SRAM_RDY_SPI_SYNC port map(
RST=>RST,
CLK=>CLK,
IN_SYNC=>RDY_SPI,
OUT_SYNC=>RDY_SPI_SYNC);

Bloque_4 : SRAM_Counter_Enabled generic map (19) port map(
RST=>RST,
CLK=>CLK,
OPR=>OPR,
DN=>DN,
```

```
ACC=>ACC);

A0<=ACC(0);
A1<=ACC(1);
A2<=ACC(2);
A3<=ACC(3);
A4<=ACC(4);
A5<=ACC(5);
A6<=ACC(6);
A7<=ACC(7);
A8<=ACC(8);
A9<=ACC(9);
A10<=ACC(10);
A11<=ACC(11);
A12<=ACC(12);
A13<=ACC(13);
A14<=ACC(14);
A15<=ACC(15);
A16<=ACC(16);
A17<=ACC(17);
A18<=ACC(18);

Bloque_5 : SRAM_Buffer_3E generic map (n) port map(
OPC=>OPC,
DI=>DI,
IO=>IO);

Bloque_6 : SRAM_Reg_Enabled generic map (n) port map(
RST=>RST,
CLK=>CLK,
OPR=>RDY_DO,
Di=>IO,
Do=>DO);

Bloque_7 : SRAM_FSM port map(
RST=>RST,
CLK=>CLK,
WR=>WR_SYNC,
RD=>RD_SYNC,
RDY_SPI=>RDY_SPI_SYNC,
DN=>DN,
OPR=>OPR,
```

```
OPC=>OPC,
RDY_SRAM=>RDY_SRAM,
RDY_DO=>RDY_DO,
WE=>WE,
CE=>CE,
OE=>OE);

end SRAM_Control;
```

7.2 APENDICE B

7.2.1 MODELO SERVIDOR EN MICROCONTROLADOR

```
                                // Tipo de comunicación Ethernet
#define SPI_Ethernet_HALFDUPLEX    0
#define SPI_Ethernet_FULLDUPLEX    1

                                // Conexiones módulo Ethernet ENC28J60
sfr sbit SPI_Ethernet_Rst at RD2_bit;
sfr sbit SPI_Ethernet_CS  at RA5_bit;
sfr sbit SPI_Ethernet_Rst_Direction at TRISD2_bit;
sfr sbit SPI_Ethernet_CS_Direction  at TRISA5_bit;

                                // Conexiones módulo SPI2
sbit Chip_Select at RD3_bit;
sbit Chip_Select_Direction at TRISD3_bit;

                                // Conexiones módulo LCD
sbit LCD_RS at RE0_bit;
sbit LCD_EN at RE2_bit;
sbit LCD_D4 at RD4_bit;
sbit LCD_D5 at RD5_bit;
sbit LCD_D6 at RD6_bit;
sbit LCD_D7 at RD7_bit;

sbit LCD_RS_Direction at TRISE0_bit;
sbit LCD_EN_Direction at TRISE2_bit;
sbit LCD_D4_Direction at TRISD4_bit;
sbit LCD_D5_Direction at TRISD5_bit;
sbit LCD_D6_Direction at TRISD6_bit;
```

```
sbit LCD_D7_Direction at TRISD7_bit;

                                        // Variables generales
int received_data;
int i=0;
int cy_ctr=0;
short take;
short buffer;
unsigned char txt_sg[6];

typedef struct {
  unsigned canCloseTCP: 1;              // Bandera para cerrar puerto TCP (Irrelevante para UDP)
  unsigned isBroadcast: 1;              // Paquete IP recivido via subnet broadcast
} TEthPktFlags;

                                        // Variables Ethernet
unsigned char txt[241];
char template_msg[] = "148.220.143.60", error_msg[] = "msg too long!";

                                        // Funciones Ethernet

unsigned int SPI_Ethernet_UserTCP(unsigned char *remoteHost, unsigned int remotePort,
                    unsigned int localPort, unsigned int reqLength, TEthPktFlags *flags) {
  return 0;                             // Regresando valor nulo por no utilizar servidor TCP
}

unsigned int SPI_Ethernet_UserUDP(unsigned char *remoteHost, unsigned int remotePort,
                    unsigned int destPort, unsigned int reqLength, TEthPktFlags *flags) {
  unsigned char len = 0;

  if (destPort != 10001) {              // Si el puerto de destino no corresponde al 10001
  //if (localPort != 10001) {           // Si el puerto de destino no corresponde al 10001
    return 0;                           // Se cancela la respuesta
  }

  if (reqLength > 242) {                // Si la solicitud es mayor de 242 caracteres
    Lcd_Out(2,1,error_msg);             // Se despliega el mensaje de error
    return sizeof(error_msg)-1;
  }

  //Lcd_Cmd(_LCD_CLEAR);                // Limpieza LCD
  //Lcd_Out(1, 2, template_msg);        // Despliegue de mensaje
```

```
if(cy_ctr<321)
 {
 cy_ctr++;
 }

if(cy_ctr<=320)
 {

 LATB.F0=1;

 for(i=0;i<240;i++)
  {

  txt[i] = SPI_Ethernet_getByte();                    // Lectura de solicitud UDP
  received_data=(int)txt[i];

  SPI_Set_Active(&SPI2_Read, &SPI2_Write);             // Puerto SPI2 en modo activo
  Chip_Select=0;
  SPI2_Write(received_data);
  Chip_Select=1;
  SPI_Set_Active(&SPI1_Read, &SPI1_Write);            // Puerto SPI1 en modo activo

  }

 IntToStrWithZeros(cy_ctr, txt);

 for(i=6;i<240;i++)
  {
  txt[i]=1;
  }

 txt[240]='\0';

 }

if(cy_ctr>=321)
 {
 LATB.F0=0;
 }

if(cy_ctr>=321 && cy_ctr<=641)
```

```
  {

  LATB.F1=1;

  for(i=0;i<240;i++)
   {

   if(i<6)
   {
   txt_sg[i] = SPI_Ethernet_getByte();                    // Lectura de solicitud UDP
   }

   SPI_Set_Active(&SPI2_Read, &SPI2_Write);               // Puerto SPI2 en modo activo
   Chip_Select=0;
   take = SPI2_Read(buffer);

   if(take==0)
    {
    take=1;
    }
   txt[i]=(char)take;
   Chip_Select=1;
   SPI_Set_Active(&SPI1_Read, &SPI1_Write);               // Puerto SPI1 en modo activo
   }

  txt[240]='\0';

                                                          // Conversion de dato entrante
  cy_ctr=((((int)txt_sg[3])-48)*100)+((((int)txt_sg[4])-48)*10)+((((int)txt_sg[5])-48)*1)+321;

  }

 if(cy_ctr>=641)
  {
  cy_ctr=0;
  LATB.F1=0;
  }

 return SPI_Ethernet_putString(txt);                      // Regresando al cliente la solicitud de dato
procesado
}
```

```
                                                    // Variables de la RAM

unsigned char myMacAddr[6] = {0x00, 0x14, 0xA5, 0x76, 0x19, 0x3f};
                                                    // Dirección MAC del PIC
//unsigned char myIpAddr[4] = {192, 168, 20, 60};
unsigned char myIpAddr[4] = {148, 220, 143, 60};
                                                    // Dirección IP del PIC

void main() {                                       // Rutina principal

 ANSELA = 0x00;                                      // Declarando todos los puertos digitales
 ANSELB = 0x00;
 ANSELC = 0x00;
 ANSELD = 0x00;
 ANSELE = 0x00;
 C1ON_bit = 0;                                      // Deshabiltación de comparadores
 C2ON_bit = 0;

 TRISA = 0x00;                                      // Configura el puerto A como salida
 PORTA = 0x00;                                       // Pone todos los pines en estado bajo

 TRISB = 0x04;                                      // Configura el puerto B como salida
 PORTB = 0x00;                                      // Pone todos los pines en estado bajo

 TRISC = 0x10;                                       // Configura el puerto C como salida
 TRISD = 0x02;                                      // Configura el puerto D como salida
 TRISE = 0x00;                                      // Configura el puerto E como salida

 //PORTD = 0x00;                                    // Pone todos los pines en estado bajo
 //PORTE = 0x00;                                    // Pone todos los pines en estado bajo

 Lcd_Init();                                        // Inicialización LCD
 Lcd_Cmd(_LCD_CLEAR);                               // Limpieza LCD
 Lcd_Cmd(_LCD_CURSOR_OFF);                          // Apagando el cursor
 LCD_Out(1, 2, template_msg);                       // Desplegando mensaje

 SPI1_Init();                                       // Inicialización puerto SPI1
 SPI_Ethernet_Init(myMacAddr, myIpAddr, SPI_Ethernet_FULLDUPLEX);
                                                    // Inicialización puerto Ethernet
 SPI2_Init_Advanced(_SPI_MASTER_OSC_DIV16, _SPI_DATA_SAMPLE_END, _SPI_CLK_IDLE_HIGH,
_SPI_LOW_2_HIGH);
                                                    // Inicialización puerto SPI2
```

```
 Chip_Select=1;                                  // Inicializando CS del SPI2 en alto

 while(1)                                        // Ciclo sin final
 {

   SPI_Set_Active(&SPI1_Read, &SPI1_Write);      // Puerto SPI1 en modo activo
   SPI_Ethernet_doPacket();                  // Procesando paquetes Ethernet de entrada

   if(PORTB.F2==1)
   {
   LATB.F3=0;
   }
   else
   {
   LATB.F3=1;
   }

 }
}
```

7.3 APENDICE C

7.3.1 MODELO CLIENTE EN PC

```
Option Explicit
Private Declare Function SetPixelV Lib "gdi32" (ByVal hdc As Long, ByVal X As Long, ByVal Y As
Long, ByVal crColor As Long) As Long
Private Declare Function GetPixel Lib "gdi32" (ByVal hdc As Long, ByVal X As Long, ByVal Y As Long)
As Long
Dim ImgData() As Long

Private Declare Sub Sleep Lib "kernel32" (ByVal millisecondsToSleep As Long)
Private Declare Function GetSystemTimes Lib "kernel32.dll" (ByRef idleTime As Currency, ByRef
kernelTime As Currency, ByRef userTime As Currency) As Boolean
Private Declare Function timeGetTime Lib "winmm.dll" () As Long

Dim Img_Format(0 To 319, 0 To 239) As Byte
Dim Img_Format_Vector(0 To 239) As Byte
Dim Img_Recive(0 To 319, 0 To 239) As Byte
Dim Img_Recive_Error(0 To 239) As Byte
```

```
Dim i As Integer, j As Integer
Dim k As Long
Dim r As Long, g As Long, b As Long
Dim RD As Integer, WR As Integer
Dim cy_ctr_int_tx As Integer
Dim Reply As String
Dim Hdr As String
Dim Data As String

'Camera Instructions
Dim playing As Boolean
Dim imageMode As Integer
Dim imageType As Integer

'Timer Control
Dim tr_before As Integer
Dim tr_present As Integer
Dim im As Integer

Private Sub Form_Load()

'Camera Instructions
playing = False
imageMode = 0
imageType = 0                    ' Raw image pixels
cmdGetImageA.Enabled = False
CmdSend.Enabled = False
cmdImgAdj.Enabled = False

'Timer Instructions
TimerFailure.Interval = 100
TimerFailure.Enabled = False
tr_present = 0
tr_before = 0

'Communicatons Instructions
Winsock1.Protocol = sckUDPProtocol
'Winsock1.RemoteHost = "192.168.20.60"
Winsock1.RemoteHost = "148.220.143.60"
Winsock1.RemotePort = 10001
Winsock1.LocalPort = 10002
'Winsock1.Bind 10002, "148.220.143.169"
```

```
Winsock1.Close
Winsock1.Connect

RD = 0
WR = 0

End Sub

Function GetRGB(colour As Long) As Integer()
Dim intRet(2) As Integer
intRet(0) = (colour& And &HFF0000) / 65536
intRet(1) = ((colour& And &HFF00) / 256&) Mod 256&
intRet(2) = colour& Mod 256
GetRGB = intRet
End Function

Private Sub mnuSalir_Click()

'Close Camera control
   Camera.Disconnect

   Unload Me

   End

End Sub

Private Sub mnuAbrirArchivo_Click()

Dim ImgData(0 To 2, 0 To 319, 0 To 239) As Long
Dim prm As Long
Dim Data As String
Dim Hdr As String
Dim lngCounter As Long
Dim Non_Selected As String
Dim Closed_By As Boolean

With CD
   .Filter = "Bitmaps and JPEGs|*.bmp;*.jpeg;*.jpg"
   .DialogTitle = "Seleccione una imagen"
   .ShowOpen
   Non_Selected = .FileName
```

```
   Closed_By = .CancelError
End With

'Picture1.Picture = LoadPicture(CD.FileName)
picIrImage.Picture = LoadPicture(CD.FileName)

If (Non_Selected = "" Or Closed_By = True) Then

Else

   mnuSalir.Enabled = False

   BarSend.Value = 0
   PercentSend = CStr(0) & " %"
   CounterSend.Text = "0"
   BarRecive.Value = 0
   PercentRecive = CStr(0) & " %"
   CounterRecive.Text = "0"

   Open "C:\Documents and Settings\Erick Lugo\Escritorio\Desarrollo Proyecto\Termografia VB
Integrated\Image_map.txt" For Output As #1

   k = 0

   For i = 0 To 319

      Data = ""
      Hdr = ""

         For j = 0 To 239

            r = GetRGB(GetPixel(picIrImage.hdc, i, j))(2)
            g = GetRGB(GetPixel(picIrImage.hdc, i, j))(1)
            b = GetRGB(GetPixel(picIrImage.hdc, i, j))(0)

            ImgData(0, i, j) = r
            ImgData(1, i, j) = g
            ImgData(2, i, j) = b

            prm = Round(((r + g + b) / 3), 0)

            If k < 10 Then
```

```
            Hdr = "0000" & CStr(k)
            ElseIf k < 100 Then
            Hdr = "000" & CStr(k)
            ElseIf k < 1000 Then
            Hdr = "00" & CStr(k)
            ElseIf k < 10000 Then
            Hdr = "0" & CStr(k)
            ElseIf k < 100000 Then
            Hdr = CStr(k)
            End If

            If prm < 10 Then
            Data = "00" & CStr(prm)
            ElseIf prm < 100 Then
            Data = "0" & CStr(prm)
            ElseIf prm < 1000 Then
            Data = CStr(prm)
            End If

            Img_Format(i, j) = prm

            Write #1, Hdr & " " & Data

            k = k + 1

            'SetPixelV Picture2.hdc, i, j, RGB(r, g, b)
            SetPixelV Picture2.hdc, i, j, RGB(prm, prm, prm)

            lngCounter = lngCounter + 1
            If Not (lngCounter Mod 1000) Then DoEvents
            If (lngCounter > 50000) Then lngCounter = 0

          Next j

        Me.Caption = "Procesamiento de imagen - " & Int(i * 100 / (320)) & "%"
        Picture2.Refresh

    Next i

Me.Caption = "Procesamiento de imagen"

Close #1
```

```
mnuSalir.Enabled = True

CmdSend.Enabled = True

End If

End Sub


Private Sub TimerFailure_Timer()

If tr_before = tr_present Then

   If RD = 1 Then

      Winsock1.SendData Img_Format_Vector

   End If

   If WR = 1 Then

      For im = 0 To 239
      SetPixelV Picture3.hdc, (cy_ctr_int_tx - 1), im, RGB(Img_Recive_Error(im),
Img_Recive_Error(im), Img_Recive_Error(im))
      Next im

      cy_ctr_int_tx = cy_ctr_int_tx + 1

      Write #2, Hdr & " " & Data

      Winsock1.SendData Reply

   End If

End If

tr_present = tr_before

End Sub

Private Sub Winsock1_Error( _
   ByVal Number As Integer, _
```

```
    Description As String, _
    ByVal Scode As Long, _
    ByVal Source As String, _
    ByVal HelpFile As String, _
    ByVal HelpContext As Long, _
    CancelDisplay As Boolean)

End Sub


Private Sub Winsock1_DataArrival(ByVal bytesTotal As Long)

Dim daAr() As Byte                      'Variable de entrada de datos
Dim daStrg As String
Dim Buffer As String

Dim cy_ctr_str As String
Dim ctr_str_char As String
Dim cy_ctr_int_rx As Integer
Dim Decoder As Integer

Winsock1.GetData daAr, vbArray + vbByte, 241
daStrg = StrConv(daAr, vbUnicode)
Buffer = Mid$(daStrg, 1, 240)

If RD = 1 Then

    cy_ctr_str = Mid$(Buffer, 4, 3)
    cy_ctr_int_rx = CInt(cy_ctr_str)

    CounterSend.Text = cy_ctr_int_rx
    Data_Input.Text = Buffer

    BarSend.Value = cy_ctr_int_rx
    PercentSend = CStr(Round(((cy_ctr_int_rx / 320) * 100), 0)) & " %"

    If cy_ctr_int_rx < 320 Then

        For j = 0 To 239
        Img_Format_Vector(j) = Img_Format(cy_ctr_int_rx, j)
        Next j
```

```
   Winsock1.SendData Img_Format_Vector

   If cy_ctr_int_rx = 1 Then
      TimerFailure.Enabled = True
   End If

   tr_before = cy_ctr_int_rx

   Data_Output.Text = Img_Format_Vector

   End If

   If cy_ctr_int_rx >= 320 Then
      RD = 0
      WR = 1
      cy_ctr_int_tx = 0
      k = 0
   End If

End If

If WR = 1 Then

   If cy_ctr_int_tx = 0 Then
   Open "C:\Documents and Settings\Erick Lugo\Escritorio\Desarrollo Proyecto\Termografia VB
Integrated\Image_Returned.txt" For Output As #2
   End If


   If cy_ctr_int_tx < 10 Then
   Reply = "00000" & CStr(cy_ctr_int_tx)
   ElseIf cy_ctr_int_tx < 100 Then
   Reply = "0000" & CStr(cy_ctr_int_tx)
   ElseIf cy_ctr_int_tx < 1000 Then
   Reply = "000" & CStr(cy_ctr_int_tx)
   End If

   CounterRecive.Text = cy_ctr_int_tx

   BarRecive.Value = cy_ctr_int_tx
   PercentRecive = CStr(Round(((cy_ctr_int_tx / 320) * 100), 0)) & " %"
```

```
If cy_ctr_int_tx <= 320 Then


   Winsock1.SendData Reply

   tr_before = cy_ctr_int_tx

   Data_Output.Text = Reply

End If

If cy_ctr_int_tx > 0 Then


   Data_Input.Text = Buffer

   For j = 0 To 239

   ctr_str_char = CStr(Mid$(Buffer, (j + 1), 1))
   Decoder = Asc(ctr_str_char)

   Hdr = ""
   Data = ""

   If k < 10 Then
   Hdr = "0000" & CStr(k)
   ElseIf k < 100 Then
   Hdr = "000" & CStr(k)
   ElseIf k < 1000 Then
   Hdr = "00" & CStr(k)
   ElseIf k < 10000 Then
   Hdr = "0" & CStr(k)
   ElseIf k < 100000 Then
   Hdr = CStr(k)
   End If

   If Decoder < 10 Then
   Data = "00" & CStr(Decoder)
   ElseIf Decoder < 100 Then
   Data = "0" & CStr(Decoder)
   ElseIf Decoder < 1000 Then
```

```
      Data = CStr(Decoder)
      End If


      Write #2, Hdr & " " & Data

      k = k + 1

      SetPixelV Picture3.hdc, (cy_ctr_int_tx - 1), j, RGB(Decoder, Decoder, Decoder)
      Img_Recive_Error(j) = Decoder

      Next j

      Picture3.Refresh

   End If

   If cy_ctr_int_tx = 320 Then
      RD = 0
      WR = 0

      TimerFailure.Enabled = False     'Paro de Timer
      tr_present = 0
      tr_before = 0

      Close #2
   End If

   cy_ctr_int_tx = cy_ctr_int_tx + 1

End If

End Sub

Private Sub CmdSend_Click()

BarSend.Value = 0
PercentSend = CStr(0) & " %"
CounterSend.Text = "0"
BarRecive.Value = 0
PercentRecive = CStr(0) & " %"
CounterRecive.Text = "0"
```

```
For j = 0 To 239
Img_Format_Vector(j) = Img_Format(0, j)
Next j

Winsock1.SendData Img_Format_Vector

Data_Output.Text = Img_Format_Vector

RD = 1
WR = 0

End Sub

Private Sub getRawImage()
   Dim varProp As Variant
   Dim varImage As Variant

   varProp = Camera.GetCameraProperty(68) 'Pixel size
   If (varProp = 0) Then
      imageType = 0
   Else
      imageType = 4
   End If

   varImage = Camera.GetImage(imageType)   ' Get absolute pixel image from camera control
   DrawPicture Me, varImage, imageType     ' Draw image in PictureBox control
End Sub
Private Sub cmdGetImageA_Click()

Dim ImgData(0 To 2, 0 To 319, 0 To 239) As Long
Dim prm As Long
Dim Data As String
Dim Hdr As String
Dim lngCounter As Long
Dim Non_Selected As String
Dim Closed_By As Boolean

Call getRawImage    'Camera Instruction

BarSend.Value = 0
PercentSend = CStr(0) & " %"
```

```
CounterSend.Text = "0"
BarRecive.Value = 0
PercentRecive = CStr(0) & " %"
CounterRecive.Text = "0"

Open "C:\Documents and Settings\Erick Lugo\Escritorio\Desarrollo Proyecto\Termografia VB
Integrated\Image_map.txt" For Output As #1

k = 0

   For i = 0 To 319

      Data = ""
      Hdr = ""

         For j = 0 To 239

            r = GetRGB(GetPixel(picIrImage.hdc, i, j))(2)
            g = GetRGB(GetPixel(picIrImage.hdc, i, j))(1)
            b = GetRGB(GetPixel(picIrImage.hdc, i, j))(0)

            ImgData(0, i, j) = r
            ImgData(1, i, j) = g
            ImgData(2, i, j) = b

            prm = Round(((r + g + b) / 3), 0)

            If k < 10 Then
            Hdr = "0000" & CStr(k)
            ElseIf k < 100 Then
            Hdr = "000" & CStr(k)
            ElseIf k < 1000 Then
            Hdr = "00" & CStr(k)
            ElseIf k < 10000 Then
            Hdr = "0" & CStr(k)
            ElseIf k < 100000 Then
            Hdr = CStr(k)
            End If

            If prm < 10 Then
            Data = "00" & CStr(prm)
            ElseIf prm < 100 Then
```

```
            Data = "0" & CStr(prm)
            ElseIf prm < 1000 Then
            Data = CStr(prm)
            End If

            Img_Format(i, j) = prm

            Write #1, Hdr & " " & Data

            k = k + 1

            'SetPixelV Picture2.hdc, i, j, RGB(r, g, b)
            SetPixelV Picture2.hdc, i, j, RGB(prm, prm, prm)

            lngCounter = lngCounter + 1
            If Not (lngCounter Mod 1000) Then DoEvents
            If (lngCounter > 50000) Then lngCounter = 0

          Next j

       Me.Caption = "Procesamiento de imagen - " & Int(i * 100 / (320)) & "%"
       Picture2.Refresh

    Next i

Me.Caption = "Procesamiento de imagen"

Close #1
mnuSalir.Enabled = True

CmdSend.Enabled = True

End Sub

Private Sub Camera_CameraEvent(ByVal lEvent As Long)

' Display events coming from the camera control
  Select Case lEvent
    Case 2:    cmdGetImageA.Enabled = True
               cmdImgAdj.Enabled = True
    Case 3:    cmdGetImageA.Enabled = False
               cmdImgAdj.Enabled = False
```

```
    Case 4:    cmdGetImageA.Enabled = False
               cmdImgAdj.Enabled = False
    Case 5:    cmdGetImageA.Enabled = True
               cmdImgAdj.Enabled = True
    Case 6:    cmdGetImageA.Enabled = False
               cmdImgAdj.Enabled = False
'   Case 2:    lstEvents.AddItem "Connected", 0
'   Case 3:    lstEvents.AddItem "Disconnected", 0
'   Case 4:    lstEvents.AddItem "Connection broken", 0
'   Case 5:    lstEvents.AddItem "Reconnected", 0
'   Case 6:    lstEvents.AddItem "Disconnecting...", 0
'   Case 7:    lstEvents.AddItem "Auto adjust", 0
'   Case 8:    lstEvents.AddItem "Start of recalibration", 0
'   Case 9:    lstEvents.AddItem "End of recalibration", 0
'   Case 10:   lstEvents.AddItem "LUT updated", 0
'   Case 11:   lstEvents.AddItem "Recording updated", 0
'   Case 12:   lstEvents.AddItem "Image captured", 0
'   Case 13:   lstEvents.AddItem "Init completed", 0
'   Case 14:   lstEvents.AddItem "Frame rate table available", 0
'   Case 15:   lstEvents.AddItem "Frame rate change completed", 0
'   Case 16:   lstEvents.AddItem "Range table available", 0
'   Case 17:   lstEvents.AddItem "Range change completed", 0
'   Case 18:   lstEvents.AddItem "Image size changed", 0
'   Case Else:  lstEvents.AddItem "Unknown event", 0
  End Select
End Sub

Private Sub cmdImgAdj_Click()
Camera.DoCameraAction (8)       ' Internal image correction
Camera.DoCameraAction (10)      ' Auto adjust of level and span
End Sub


Function IntegerToUnsigned(Value As Integer) As Long
   If Value < 0 Then
      IntegerToUnsigned = Value + 65536
   Else
      IntegerToUnsigned = Value
   End If
End Function
```

```
Private Sub Form_Terminate()
                        'Close Camera control
   Camera.Disconnect
End Sub

Private Sub Form_Unload(Cancel As Integer)
'Close Camera control
   Camera.Disconnect
   End
End Sub
```

7.4 APENDICE D

7.4.1 MÉTODOS Y EVENTOS ACTIVEX DE FLIR

```
Option Explicit
  ' The following Types, Declares, and Constants are only necessary
  ' for the DrawPicture routine
  '=====================================================================
   Type RGBQUAD
      rgbBlue As Byte
      rgbGreen As Byte
      rgbRed As Byte
      rgbReserved As Byte
   End Type

   Type BITMAP '14 bytes
      bmType As Long
      bmWidth As Long
      bmHeight As Long
      bmWidthBytes As Long
      bmPlanes As Integer
      bmBitsPixel As Integer
      bmBits As Long
```

```
 End Type

Type BITMAPINFOHEADER_TYPE
  biSize As Long
  biWidth As Long
  biHeight As Long
  biPlanes As Integer
  biBitCount As Integer
  biCompression As Long
  biSizeImage As Long
  biXPelsPerMeter As Long
  biYPelsPerMeter As Long
  biClrUsed As Long
  biClrImportant As Long
End Type

Type BITMAPINFO_TYPE
  BitmapInfoHeader As BITMAPINFOHEADER_TYPE
  bmiColors(256) As RGBQUAD
End Type

Declare Function CreateDIBSection Lib "gdi32" (ByVal hdc As Long, pBitmapInfo As BITMAPINFO_TYPE, ByVal un As Long, ByVal lplpVoid As Long, ByVal handle As Long, ByVal dw As Long) As Long
Declare Function MyGetObject Lib "gdi32" Alias "GetObjectA" (ByVal hObject As Long, ByVal nCount As Long, lpObject As Any) As Long
Declare Function CreateCompatibleDC Lib "gdi32" (ByVal hdc As Long) As Long
Declare Function BitBlt Lib "gdi32" (ByVal hDestDC As Long, ByVal X As Long, ByVal Y As Long, ByVal nWidth As Long, ByVal nHeight As Long, ByVal hSrcDC As Long, ByVal XSrc As Long, ByVal YSrc As Long, ByVal dwRop As Long) As Long
Declare Function SelectObject Lib "gdi32" (ByVal hdc As Long, ByVal hObject As Long) As Long
```

```
Declare Function CreateCompatibleBitmap Lib "gdi32" (ByVal hdc As Long, ByVal nWidth As Long, ByVal nHeight As Long) As Long
Declare Function StretchBlt Lib "gdi32" (ByVal hdc As Long, ByVal X As Long, ByVal Y As Long, ByVal nWidth As Long, ByVal nHeight As Long, ByVal hSrcDC As Long, ByVal XSrc As Long, ByVal YSrc As Long, ByVal nSrcWidth As Long, ByVal nSrcHeight As Long, ByVal dwRop As Long) As Long
Declare Function SetDIBits Lib "gdi32" (ByVal hdc As Long, ByVal hBitmap As Long, ByVal nStartScan As Long, ByVal nNumScans As Long, lpBits As Any, lpBI As BITMAPINFO_TYPE, ByVal wUsage As Long) As Long
Declare Function GetDIBits Lib "gdi32" (ByVal aHDC As Long, ByVal hBitmap As Long, ByVal nStartScan As Long, ByVal nNumScans As Long, lpBits As Any, lpBI As BITMAPINFO_TYPE, ByVal wUsage As Long) As Long
Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal Length As Long)
Declare Function StretchDIBits Lib "gdi32" (ByVal hdc As Long, ByVal X As Long, ByVal Y As Long, ByVal dx As Long, ByVal dy As Long, ByVal SrcX As Long, ByVal SrcY As Long, ByVal wSrcWidth As Long, ByVal wSrcHeight As Long, lpBits As Any, lpBitsInfo As BITMAPINFO_TYPE, ByVal wUsage As Long, ByVal dwRop As Long) As Long
Declare Function PatBlt Lib "gdi32" (ByVal hdc As Long, ByVal X As Long, ByVal Y As Long, ByVal nWidth As Long, ByVal nHeight As Long, ByVal dwRop As Long) As Long
Declare Function DeleteDC Lib "gdi32" (ByVal hdc As Long) As Long
Declare Function lstrcpy Lib "kernel32" Alias "lstrcpyA" (ByVal lpString1 As String, ByVal lpString2 As String) As Long
Declare Function ReleaseDC Lib "user32" (ByVal hwnd As Long, ByVal hdc As Long) As Long
Declare Function DeleteObject Lib "gdi32" (ByVal hObject As Long) As Long
Public Declare Function RealizePalette Lib "gdi32" (ByVal hdc As Long) As Long

Global Const SRCCOPY = &HCC0020
Global Const BI_RGB = 0
Global Const DIB_RGB_COLORS = 0
Global Const DIB_PAL_COLORS = 1
Global Const RASTERCAPS = 38
Global Const RC_STRETCHDIB = &H2000
```

```
Global Const CBM_INIT = 4           'initialize bitmap
Global Const PATCOPY = &HF00021      ' (DWORD) dest = pattern
Global Const PATINVERT = &H5A0049    ' (DWORD) dest = pattern XOR dest
Global Const WHITENESS = &HFF0062    ' (DWORD) dest = WHITE
Global Const BLACKNESS = &H42&       ' (DWORD) dest = BLACK

Private Const OFFSET_4 = 4294967296#
Private Const MAXINT_4 = 2147483647
Private Const OFFSET_2 = 65536
Private Const MAXINT_2 = 32767

Function IntegerToUnsigned(Value As Integer) As Long
    If Value < 0 Then
      IntegerToUnsigned = Value + OFFSET_2
    Else
      IntegerToUnsigned = Value
    End If
  End Function

Sub DrawPicture(frm As Form, varImage As Variant, imageType As Integer)

  Dim BitmapInfo As BITMAPINFO_TYPE
  Dim hMem As Long
  Dim lRetVal As Long
  Dim lngWidthIrImage As Long
  Dim lngHeightIrImage As Long
  Dim xAlignSize As Long
  Dim i As Integer
  Dim mSrcDc As Long
  Dim rsel As Long
  Dim hBitmap As Long
  Dim bm As BITMAP
```

```
Dim lpImg As Long
Dim intPixVal As Integer
Dim lngMinPixelValue As Long
Dim lngMaxPixelValue As Long
Dim lngUnsignedPixel As Long
Dim X As Long
Dim Y As Long
Dim lngSpan As Long
Dim bytPixel As Byte
Dim lngpicWidth As Long
Dim lngpicHeight As Long
frm.PaletteMode = vbPaletteModeCustom

' Picture Box should have autoredraw = True, ScaleMode = Pixel
frm.picIrImage.ScaleMode = 3 'Pixels
frm.picIrImage.AutoRedraw = True

If TypeName(varImage) = "Integer" Then
  Exit Sub
End If

lngWidthIrImage = UBound(varImage, 1) + 1
lngHeightIrImage = UBound(varImage, 2) + 1
xAlignSize = Int((lngWidthIrImage + 3) / 4) 'Must be 4 byte aligned
xAlignSize = xAlignSize * 4

BitmapInfo.BitmapInfoHeader.biSize = 40
BitmapInfo.BitmapInfoHeader.biWidth = xAlignSize
BitmapInfo.BitmapInfoHeader.biHeight = -lngHeightIrImage
BitmapInfo.BitmapInfoHeader.biPlanes = 1
BitmapInfo.BitmapInfoHeader.biBitCount = 8
BitmapInfo.BitmapInfoHeader.biCompression = BI_RGB
```

```
BitmapInfo.BitmapInfoHeader.biXPelsPerMeter = 0
BitmapInfo.BitmapInfoHeader.biYPelsPerMeter = 0
BitmapInfo.BitmapInfoHeader.biSizeImage = xAlignSize * lngHeightIrImage
BitmapInfo.BitmapInfoHeader.biClrUsed = 256
BitmapInfo.BitmapInfoHeader.biClrImportant = 0

' Create a standard 256 grey palette
For i = 0 To 255
  BitmapInfo.bmiColors(i).rgbBlue = CByte(i)
  BitmapInfo.bmiColors(i).rgbGreen = CByte(i)
  BitmapInfo.bmiColors(i).rgbRed = CByte(i)
  BitmapInfo.bmiColors(i).rgbReserved = 0
Next i

'Create compatible memory allocation
mSrcDc = CreateCompatibleDC(frm.picIrImage.hdc)
hBitmap = CreateDIBSection(0, BitmapInfo, DIB_RGB_COLORS, 0, 0, 0)

If imageType < 4 Then
  ' Find minimum and maximum pixels
  lngMinPixelValue = 65535
  lngMaxPixelValue = 0
  For Y = 0 To lngHeightIrImage - 1
     For X = 0 To lngWidthIrImage - 1
        intPixVal = varImage(X, Y)
        lngUnsignedPixel = IntegerToUnsigned(intPixVal) ' returns long
        Select Case lngUnsignedPixel
          Case Is < lngMinPixelValue
            lngMinPixelValue = lngUnsignedPixel
          Case Is > lngMaxPixelValue
            lngMaxPixelValue = lngUnsignedPixel
        End Select
```

```
        Next X
      Next Y
      lngSpan = lngMaxPixelValue - lngMinPixelValue
    End If

    rsel = SelectObject(mSrcDc, hBitmap)
    lRetVal = PatBlt(mSrcDc, 0, 0, frm.picIrImage.ScaleWidth, frm.picIrImage.ScaleHeight,
WHITENESS)   'Fill memory with white pattern
    Call MyGetObject(hBitmap, Len(bm), bm)

    If imageType < 4 Then
     ' Convert 16-bit pixels to color values and fill bitmap
     For Y = 0 To lngHeightIrImage - 1
      lpImg = bm.bmBits + (Y * xAlignSize)
      For X = 0 To lngWidthIrImage - 1
       intPixVal = varImage(X, Y)
       lngUnsignedPixel = IntegerToUnsigned(intPixVal) ' returns long
       bytPixel = CByte(Int(((lngUnsignedPixel - lngMinPixelValue) * 255) / lngSpan))
       CopyMemory ByVal lpImg, ByVal VarPtr(bytPixel), 1
       lpImg = lpImg + 1
      Next X
     Next Y
    Else
     ' Convert 8-bit pixels to bitmap
     For Y = 0 To lngHeightIrImage - 1
      lpImg = bm.bmBits + (Y * xAlignSize)
      For X = 0 To lngWidthIrImage - 1
       bytPixel = varImage(X, Y)
       CopyMemory ByVal lpImg, ByVal VarPtr(bytPixel), 1
       lpImg = lpImg + 1
      Next X
     Next Y
```

```
    End If

    'Bitmap is ready - show image
    Call SelectObject(mSrcDc, hBitmap)
    lngpicWidth = frm.picIrImage.ScaleWidth
    lngpicHeight = frm.picIrImage.ScaleHeight

   'lRetVal = StretchBlt(frm.IrImage.hdc, 0, 0, cw, ch, mydc, 0, 0, xAlignSize, Height, SRCCOPY)
   lRetVal = BitBlt(frm.picIrImage.hdc, 0, 0, frm.picIrImage.ScaleWidth,
frm.picIrImage.ScaleHeight, mSrcDc, 0, 0, SRCCOPY) ' Move Pixel by pixel to screen

    frm.picIrImage.Refresh
    'Cleanup Memory
   lRetVal = DeleteObject(rsel)
   lRetVal = SelectObject(mSrcDc, rsel)
   lRetVal = DeleteObject(hBitmap)
   lRetVal = DeleteDC(mSrcDc)
End Sub
```

8. REFERENCIAS

[1] Ariza, Miguel, Butraigo Javier, Servidor Web embebido en una FPGA con codiseño como metodología de diseño, Universidad Sergio Alboleda, Colombia, 2010.

[2] Arteaga Cardineau, Francisco. Javier, Informatica: “Redes e Internet”, 2001.

[3] Barranco, F. Entorno software para visualización y configuración de procesamiento de imágenes en tiempo real con plataformas reconfigurables, Universidad de granada, 2010.

[4] Bísaro, Mauricio, Danizio, Eduardo, Modelo OSI - TCP/IP, 2005.

[5] González Arenas, Julio. Sistema de almacenamiento de datos utilizando memoria SDRAM, un FPGA y comunicación USB, Tesis de Maestría, Instituto Politécnico Nacional, México, 2008.

[6] González Ruiz, Vicente. Redes de Computadoras, 2009.

[7] Gutiérrez Quintanilla, E., Pérez Bernabe, H.B., Rodríguez Doñate, R., Estructura general de un FPGA. Fuente: Aplicación y soluciones para lógica programable (FPGA), 2008.

[8] Hernández, Jorge. Clasificación de granos de café usando FPGA, Universidad Nacional de Colombia, 2004.

[9] Herrman, Fernando, Perin, Gilherme, José de Freitas, Josue, Bertagnolli, Rafael, dos Santos, Baptista, An UDP/IP Network stack in FPGA, Federal University of Santa Maria, Brazil, 2009.

[10] Hansen, Kai, Ethernet de alto rendimiento, ABB AS, Billingstad, Noruega, 2006.

[11] http://www7.uc.cl/sw_educ/geo_mar/html/glosario.html

[12] Ibarra Castañedo, Clemente, A. González, Daniel, Bendada, Hakim, Maldague, Xavier, Análisis de imágenes en Termografía Infrarroja, 2005.

[13] Loheide, Steven Quantifying Stream-Aquifer Interactions through the Analysis of Remotely Sensed Thermographic Profiles and In Situ Temperature Histories, Stanford University, 2006.

[14] López Pérez, Eric, Protocolo SPI (Serial Peripherical Interface), Ingeniería en Microcontroladores, 2005

[15] Lu, Weidong, Designing TCP/IP Functions in FPGA, Tesis de Maestría, Delft University of Technology, Holland, 2002.

[16] Mora, Omar Gregorio. IP Core controlador PID con comunicación USB basado en tecnología FPGA, Tesis de Licenciatura, Universidad Autónoma de Querétaro, México, 2010.

[17] Morales Velázquez, Luis. Unidad de USB de control de posición y generación de perfiles para un intercambiador automática de herramientas, Tesis de Maestría, Universidad Autónoma de Querétaro, México, 2007.

[18] Márquez Avendaño, Bertha Mariel, Implementación de un reconocedor de voz gratuito a el sistema de ayuda a invidentes Dos-Vox en español, Tesis profesional, Universidad de las Américas Puebla, México, 2004.

[19] Martin, V. New FPGA-based image-oriented acquisition and real-time processing applied to plasma facing component thermal monitoring, NRIA, 2010.

[20] Muñoz Potosi, A., Pencue Fierro, L., León Téllez, J. Análisis Termográfico para la determinación de puntos críticos en equipos mecánicos y eléctricos. Bistua: Revista de la Facultad de Ciencias Básicas, Vol. 7, Núm. 1, 2009, pp. 1-4, Universidad de Pamplona Colombia.

[21] Pérez, Madaín Un procesador basado en FPGA para el tratamiento de secuencias de imágenes: Aplicación a la estéreo visión densa, Université des Sciences et Technologies de Lille, 2005.

[22] Rubio Herrera, Carlos, Arquitectura estándar y principios del protocolo Giga Ethernet, Universidad de San Carlos de Guatemala Facultad de Ingeniería, Escuela de Ingeniería Mecánica Eléctrica (2009).

[23] Thompson, M., FPGA accelerate time to market for industrial designs, 2004.

[24] Tinoosh, Mohhsenin, Design and evaluation of FPGA-based Gigabit-Ethernet/PCI Network Interface Card, Tesis de Maestría, Rice University Houston TX, E.U., 2004.

[25] Velazco Mendoza, Rodrigo, Guzmán Gallegos, Ricardo. Monitoreo y control de riego mediante una red Ethernet con interfaz en Labview: Adquisición de datos en una red Modbus plus a través de una red Interbus, Tesis de Maestría, Universidad Autónoma de Querétaro, México, 2009.

[26] Yashimoto, Yasushi. Dynamic Analysis of Water Stress of Sunflower Leaves by Means of a Thermal Image Processing System, Duke University, 1984.