Portada Interna de Tesis

Universidad Autónoma de Querétaro
Facultad de Ingeniería
Maestría Instrumentación y Control automático

IMPLEMENTACIÓN EN FPGA (FIELD PROGRAMMABLE GATE ARRAY) DE UN CONTROLADOR DIFUSO APLICADO EN LA AUTOMATIZACIÓN DE ALIMENTADORES PARA ACUICULTURA

TESIS

Que como parte de los requisitos para obtener el grado de

Maestro en Instrumentación y Control automático

Presenta:
Ing. Christian Reyes Gonzalez

Dirigido por:
M.C. Genaro Martin Soto Zarazúa

SINODALES

M.C. Genaro Martin Soto Zarazúa
Presidente — Firma

Dr. Gilberto Herrera Ruiz
Secretario — Firma

M.C. Manuel Toledano Ayala
Vocal — Firma

Dr. Enrique Rico García
Suplente — Firma

Dr. Edgar Rivas Araiza
Suplente — Firma

Dr. Gilberto Herrera Ruiz
Director de la Facultad

Dr. Luis Gerardo Hernández Sandoval
Director de Investigación y Posgrado

Centro Universitario
Querétaro, Qro.
Junio de 2010
México

RESUMEN

Para controlar de manera óptima el suministro de alimento en tanques acuícolas, se diseñó un sistema orientado a una tecnología de implementación FPGA. El diseño del sistema fue descrito en VHDL (VHSIC HDL, *very high-speed integrated circuit hardware description language*). El sistema fue descrito a nivel de comportamiento, por medio de componentes. El sistema consta de un número de salidas digitales a las cuales le pueden ser programados los tiempos de activación de alimentación o bien, que el sistema calcule la cantidad de alimento a proporcionar por medio de un controlador difuso que usa como entradas la temperatura y el oxigeno disuelto. Toda la matemática que modela y describe el comportamiento del controlador difuso se basa en la propuesta por Soto-Zarazúa et al., (2010). En el diseño se implemento un sistema de adquisición de datos utilizado para proporcionar al controlador los valores de las variables de entrada. El software para la configuración del chip fue desarrollado en C#. Tanto las programaciones de encendido del alimentador como la configuración del controlador difuso son almacenadas en un chip de memoria Flash. El controlador de la memoria Flash ha mostrado un desempeño de escritura de 32.3 KB/s; y de lectura de 14.6MB/s. El desempeño de lectura indica que la memoria (al ser de 8MB) podrá ser escaneada 1.8 veces por segundo en busca de programaciones de activación. La comunicación entre el chip y la computadora ha sido probada, con resultados satisfactorios, a una velocidad de 115,200 bps, con una trama de 345,600 bytes; sin perder ningún bit. La máxima frecuencia de operación del sistema obtenida del proceso de síntesis es de 156.5 MHz, el sistema ha sido diseñado para trabajar a 50 MHz. Siendo ésta menor que la máxima frecuencia de operación, nos asegura el correcto funcionamiento del sistema a velocidades menores o iguales a la máxima frecuencia. El sistema completo ocupa el 3.8% de la capacidad del FPGA. Con base en el desempeño general del diseño, el sistema puede procesar hasta 2’097,152 programaciones por segundo; lo que lo hace viable para procesar el contenido de la memoria en poco menos de un segundo. La respuesta del controlador difuso comparada con el resultado obtenido por Soto-Zarazúa et al., (2010) presenta una correlación R=1; lo que demuestra la correcta implementación del controlador difuso, sin embargo, la respuesta del controlador difuso fue discreta.

(**Palabras clave**: Control difuso, memoria Flash, VHDL, FPGA)

SUMMARY

To optimally control food supply in aquaculture tanks, it was designed a system designated to an FPGA implementation technology. The system's design was described in VHDL (VHSIC HDL, very high-speed integrated circuit hardware description language). The system was described at behavioral level, by means of components. The system comprises a number of digital outputs to which it may be scheduled the feeding activation time or, that the system estimates quantity of food to provide by means of a fuzzy controller that uses as inputs temperature and dissolved oxygen. All mathematics that models and describes the fuzzy controller behavior has been based on the proposed by Soto-Zarazúa et al., (2010). In the design it was implemented a data acquisition system utilized to provide to the controller the input parameters values. The software for chip configuration was developed on C#. Feeder schedule activations as well as fuzzy controller configuration are stored in a solid state storage device, specifically speaking, Flash memory. Flash memory controller has shown a writing performance of 32.3 KB/s; and reading of 14.6 MB/s. Reading performance points out that the memory (being of 8MB capacity) could be scanned 1.8 times by second in search of activation schedules. Communication between the chip and computer has been tested, with favorable results, at a speed of 115,200 bps, with a stream of 345,600 bytes; without losing a single bit. System's maximum operation frequency retrieved from synthesis process is 156.5 MHz, the system has been designed to operate at 50 MHz. Been this less than maximum operation frequency, ensure correct system functioning at maximum operation frequency or below. The whole system occupies 3.8% of FPGA capacity. Based on the design's general performance, the system is able to process up to 2'097,152 schedules per second; which make it profitable to process memory contents in little less than a second. Fuzzy controller response being compared against result obtained by Soto-Zarazúa et al., (2010) presents a correlation R=1; showing correct fuzzy controller implementation, nevertheless, fuzzy controller response was discrete.

(**Key words**: Fuzzy control, Flash memory, VHDL, FPGA)

A los interesados en líneas de investigación similares,
así como a todos los que me han apoyado.

AGRADECIMIENTOS

Al Consejo Nacional de Ciencia y Tecnología, por la beca de manutención otorgada para el apoyo de los gastos en el transcurso del estudio de la maestría. A la Universidad Autónoma de Querétaro por el pago de los créditos y algunas de las inscripciones por concepto de haber impartido el Laboratorio de Automatización 1. A Genaro Martin Soto Zarazúa por el gran trabajo de asesor desempeñado desde ya hace 3 años, así como el apoyo económico. A los sinodales por las observaciones que contribuyeron al mejoramiento de la escritura de esta tesis. Y a mi familia por el apoyo moral y económico.

I N D I C E

INDICE DE TABLAS

INDICE DE FIGURAS

I. CAPÍTULO 1

I.1 Introducción

El presente trabajo ha sido estructurado de la siguiente manera, teniendo en primera instancia la etapa de descripción del problema donde se explica la problemática a resolver y se da a conocer un panorama general acerca de la producción acuícola en general para la especie en la que se centra el trabajo de investigación. Así mismo se hace una revisión de antecedentes relacionados con el tema en base a artículos científicos relacionados con el tema así como se enuncia la existencia de aplicaciones comerciales acerca del control difuso.

En la siguiente subsección se plantea la justificación del trabajo remarcando tanto las principales ventajas del control difuso como sus aplicaciones, también se especifican las ventajas de utilizar el control difuso implementado en hardware comparándolo con su contraparte en software. Posteriormente se enuncian el objetivo general, objetivos particulares e hipótesis general, estos con el fin de dar a conocer lo que se quiere hacer e intenta comprobar.

I.2 Descripción del problema

Es bien sabido que los sistemas de acuicultura (extensivos, semi-intensivos e intensivos) representan uno de los principales sectores de producción de alimentos con alto contenido de proteínas para consumo humano, por esta razón sistemas de monitoreo y control han sido desarrollados para mejorar dos aspectos fundamentales de la acuicultura sustentable: incremento de productividad y optimización del uso de la energía. La primera es directamente relacionada con la redituabilidad económica y en consecuencia con los recursos económicos, y la ultima con el uso de la electricidad, personal, administración de alimentación y uso eficiente del agua. Basado en ello se enuncia la necesidad de mejorar los sistemas de alimentación por medio de nuevas estrategias de control.

I.2.1 Sector productivo

Según el Programa Maestro Nacional de Tilapia (2007), la tilapia es el segundo producto pesquero más importante en la acuacultura mundial por debajo de la producción de carpas; de acuerdo con el Programa Maestro Nacional de Tilapia (2007) durante 2004, la producción mundial fue de 2, 466,028 toneladas, en tanto a la situación nacional mientras en el 2003 la producción nacional de tilapia fue de 64,293 toneladas, peso vivo desembarcado, y un valor de $671,341 miles de pesos, desembarcado. En la producción nacional de acuacultura, la tilapia (mojarra) ocupa la segunda posición en términos de valor, con 608,080 miles de pesos, y un volumen total de 61,516 toneladas, con una aportación al total nacional del 13.7% y 29.6% respectivamente. De acuerdo a la Carta Nacional Pesquera 2004, en México existen 3,970 unidades de producción acuícola, de las cuales 3,429 son para autoconsumo. Además existen también 35 centros acuícolas. La superficie total cultivada es de 276,944.65 hectáreas para todas las especies que se cultivan adicionalmente a la tilapia.

I.2.2 Aspectos de producción

Generalmente los procesos automáticos necesitan en ciertos puntos de su rutina de funcionamiento tomar decisiones que afectan la dinámica del sistema. Estas decisiones, desde el punto de vista de control, pueden ser controladas conociendo los parámetros del sistema; principalmente aquellos parámetros que afectan de manera directa el fenómeno a controlar. La idea general del trabajo de investigación es controlar de manera óptima (evitando desperdicio de alimento) la alimentación del pez. En este trabajo se ha utilizado la relación que existe entre el porcentaje de oxígeno disuelto en el agua y la temperatura del estanque para relacionar ambos parámetros con el porcentaje de alimento a suministrar (Soto-Zarazúa et al. in press). En los cultivos intensivos de especies acuícolas es necesario controlar la cantidad de alimento a proveer, esto por dos principales

razones: evitar el desperdicio del alimento así como evitar la contaminación del estanque.

El sistema utilizado actualmente está basado en el uso de una computadora como elemento de control. Para que dicha computadora sea utilizada como tal se ha diseñado un software que controla 8 salidas. Los diagramas de operación del software son mostrados en las Figuras 1.1, 1.2 y 1.3. El control puede operar de dos maneras, por tiempo preprogramado o bien por medio del algoritmo difuso programado en el que se adquieren lecturas de los sensores y se determina de manera automática el tiempo de encendido que mejor le corresponda dependiendo de las condiciones ambientales.

El proceso principal, mostrado en la Figura 1.1, inicia con la ejecución del programa, se buscan archivos de configuración ya existentes, en caso de no encontrar ninguno se procede a la creación una tabla base vacía, finalmente se examina la tabla de horarios en busca de programaciones en tiempo de activación, en caso de existir alguna se ejecuta el proceso de activación.

Al inicio del proceso de activación se verifica el modo de operación configurado actualmente y en base a este de ejecuta la rutina correspondiente. Una de estas siendo la lectura de los tiempos de activación guardados en una tabla que indican cuanto tiempo y en qué momento se debe activar cada salida o bien calcular el tiempo de encendido de acuerdo a las condiciones ambientales del estanque. Los parámetros del controlador son cargados desde el disco duro. Respecto al uso de una computadora se derivan las siguientes cuestiones:

Desventajas

•El hecho de hacer uso de una computadora requiere demasiada energía (aproximadamente de 1kW).

•La computadora requiere de ventilación adecuada, así como de un ambiente de baja humedad.

•En el caso de restablecimiento de suministro eléctrico la computadora entrara en un ciclo de reinicio en el cual no es posible controlar el estado del puerto que controla en sistema.

•El sistema de control será afectado directamente por la inestabilidad del sistema operativo.

•Tiempo de depuración muy alto.

Debido al número de desventajas que presenta el uso de una computadora como elemento de control se propone entonces el desarrollo de un controlador difuso basado en FPGA (*Field Programmable Gate Array*). El diagrama general del nuevo sistema de control en donde se ilustran de manera tentativa los elementos del sistema, su organización, interconexión así como la dirección del flujo de las señales se muestra en la Figura 1.4.

Su consumo de energía es por mucho menor que el de una PC de escritorio, más estable debido al desuso de sistemas operativos que pueden causar mal funcionamiento de sistema de control, mayor grado de personalización al diseñar nuestra propia arquitectura que permita hacer un mejor controlador. Una posible desventaja, ahora sería el elevado tiempo de desarrollo de interfaces graficas para facilitar el uso del sistema; lo cual en este caso se contrarrestara haciendo uso de la computadora como interfaz del usuario.

Figura 1.1. Proceso principal, diagrama de solución implementada en PC.

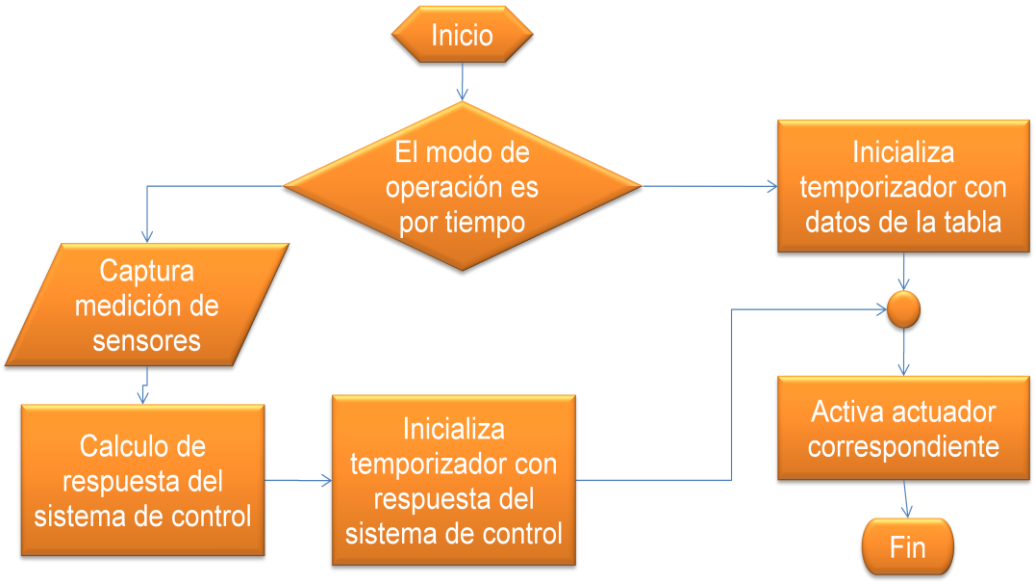

Figura 1.2. Proceso de activación, diagrama detallado de la ejecución del proceso de activación mostrado en la Figura 1.1.

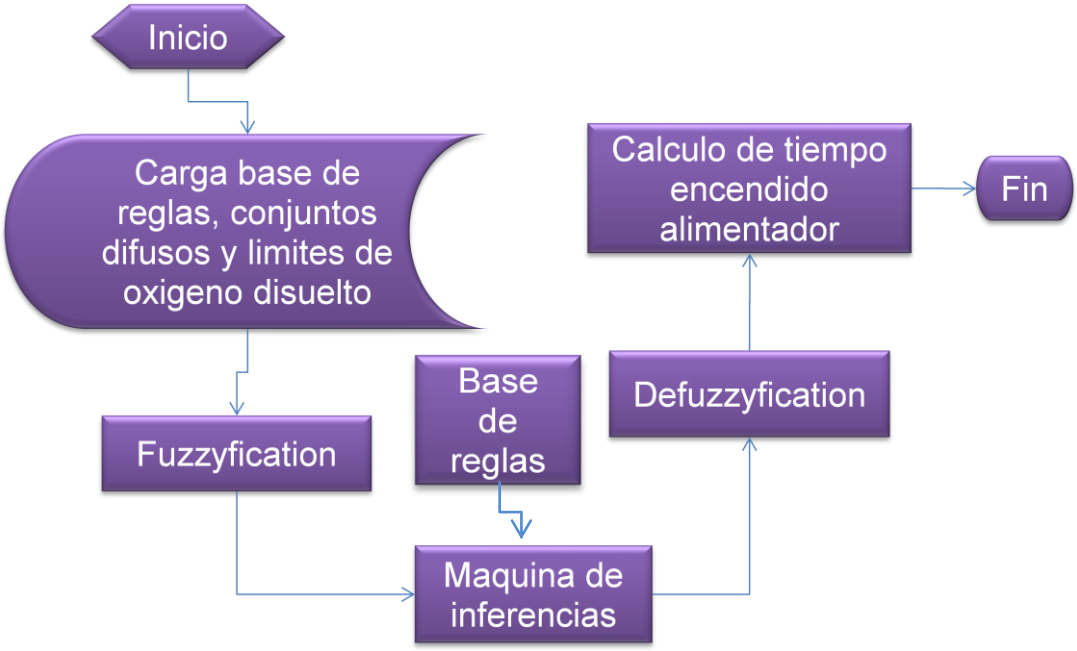

Figura 1.3. Cálculo de respuesta del sistema de control, subrutina para el cálculo del tiempo de activación.

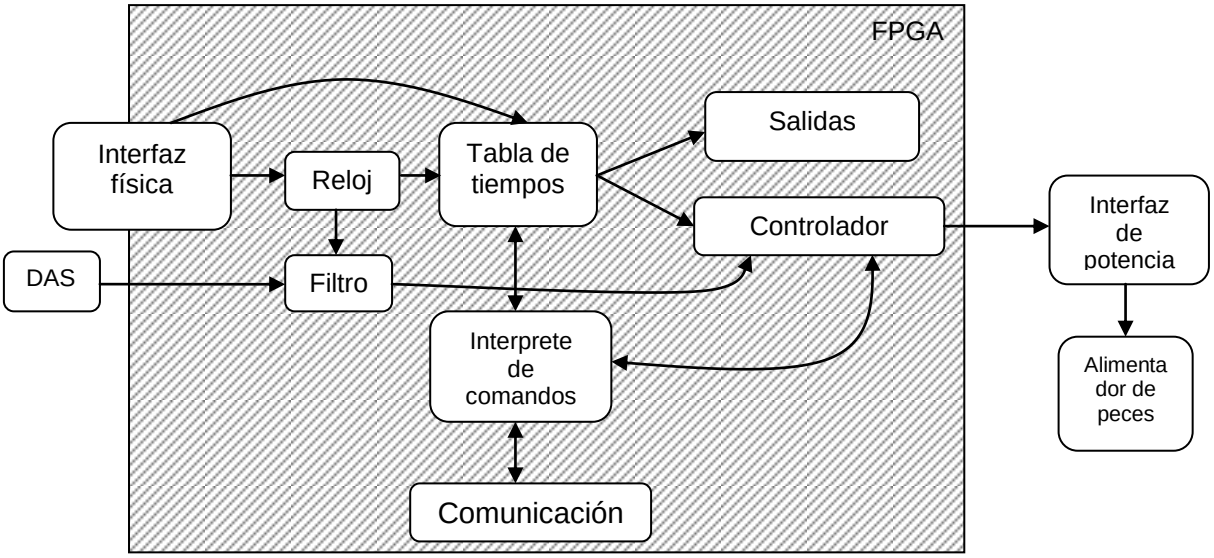

Figura 1.4. Diagrama de solución basada en FPGA.

I.3 Antecedentes

I.3.1 Alimentadores en acuicultura

Los distribuidores de alimento son equipo de alimentación especialmente diseñado. Existen dos principales maneras en las que la tecnología de alimentación de peces, y por tanto el equipo, difiere de aquellos generalmente utilizados en la ganadería. El alimento debe ser depositado en el agua y es usado alimento de tamaño muy pequeño (incluso en polvo).

Actualmente existen diversos y diferentes tipos de equipo de alimentación usados alrededor del mundo. Los alimentadores pueden ser divididos en grupos de acuerdo a las especies de peces cultivadas, edad del pez, o tipo de alimento, principalmente. Sin embargo, estos tipos se basan en algunos principios básicos. Los alimentadores de peces pueden ser divididos en dos grandes grupos: estacionarios (o fijos) y móviles.

El equipo de alimentación estacionario se subdivide en aquellos que requieren de suministro de energía (automáticos [eléctricos, neumáticos e hidráulicos]) y los que no (de demanda). En tanto a los móviles, existen botes y carros alimentadores. El controlador de alimentadores a desarrollar esta orientado a los alimentadores eléctricos. Los alimentadores eléctricos pueden ser clasificados de acuerdo a su suministro de energía siendo de tipo eléctrica, neumático e hidráulica.

Los alimentadores automáticos operados eléctricamente constan principalmente de tres partes contenedor de alimento, distribuidor de alimento y dispositivo de porcionado. Los diseños difieren entre cada uno en el dispositivo distribuidor de alimento y pueden ser agrupados como: operados por electroimán con movimiento alternativo, operados por motor con movimiento giratorio y alimentado por vibración.

En la Figura 1.5 se muestra un alimentador eléctrico operado por motor, de la compañía Aquaculture Equipment LTD modelo SOLO 1 Spinner Feeder. El alimentador puede proveer alimento hasta 8 metros de distancia.

Figura 1.5. Alimentador SOLO 1, Aquaculture Equipment LTD.

En la Figura 1.6 se muestra un alimentador vibratorio de la misma compañía, utilizando un plato de 4” y al que se le puede acoplar un tanque de 80 litros para aumentar su capacidad de almacenamiento de alimento.

Figura 1.6. Alimentador SOLO 2, Aquaculture Equipment LTD.

I.3.2 Antecedentes comerciales

En tanto a la implementación digital de procesadores especializados, recientes investigaciones en el balance entre desempeño y costo han permitido el desarrollo de arquitecturas solución con soporte específico de acuerdo con Reznik (1997), y han sido propuestos diversos coprocesadores dedicados a la lógica difusa y control. El hardware dedicado se puede considerar como la mejor técnica en términos de desempeño, pero este puede solamente cubrir un limitado rango de aplicaciones.

En vista de la falta de flexibilidad, la opción de soluciones completas en hardware especializado puede representar un camino efectivo, en particular para aplicaciones que requieren un gran número de reglas. El hardware difuso específico nos permite en varios casos alcanzar una mejor relación costo-desempeño debido a la explotación del paralelismo en el procesamiento difuso y la introducción de unidades de propósito especial.

Para dar una introducción de lo anterior permítase considerar dos procesadores especializados: FC110 de Togai InfraLogic y AL220 de Adaptive Logic^{TM}, el último de ellos siendo el más reciente. El FC110, procesador difuso digital de Togai InfraLogic fue desarrollado como un coprocesador difuso especializado. Es un solo chip suficientemente pequeño para aplicaciones embebidas su arquitectura supone altas posibilidades de comunicación para trabajar junto con un procesador maestro. Este no está orientado a un tipo de procesador específico y es flexible en posibles aplicaciones. El AL220 es un microcontrolador independiente que emplea control difuso, es económico y de alto desempeño. El dispositivo contiene cuatro enteradas analógicas de 8 bits de resolución, y un generador de reloj interno. El AL220 consume muy poca energía durante su operación normal y tiene un modo de ahorro de energía.

Así como el control difuso se hizo popular y numerosas aplicaciones de la vida real han sido desarrolladas. Como son un (Rice cooker) cocinador de arroz de Panasonic®/National® Fuzzy Logic en el que el proceso de cocinado es controlado por lógica difusa, autoajustado para las condiciones del arroz y agua. El pocillo térmico (Thermo pot) de National® Deluxe Electric Fuzzy Logic esta unidad representa la mejor tecnología disponible en producción de agua caliente en demanda para hacer té.

Sin embargo el control difuso también se ha aplicado en el control de transmisiones automáticas de automóviles como el Saturn™ SL1 y el Mitsubishi Galant™ ES y LS; desarrolladores de hardware han empezado a proponer soluciones complejas, integrando software y hardware difuso con PLCs (del inglés *Programmable Logic Controller*) controladores lógicos programables convencionales y DCS (del inglés *Distributed Control Systems*) sistemas de control distribuido. El objetivo es proveer una oportunidad para completar un diseño e implementar un controlador para cualquier aplicación en particular sin la necesidad de ningún software y hardware adicional.

Dos maneras de transformar el diseño de un controlador difuso en aplicaciones industriales reales son explicadas a continuación. La primera es llamada UNAC™ la cual fue desarrollada por CICS Automation. Ésta posee la combinación de difuso y tecnología a nivel 'macro'. UNAC consiste de un paquete de diseño, incluyendo metodología de control difuso, y una parte hardware SAAC™ que es descargada con un código controlador desarrollado por el software.

La segunda metodología es una solución a nivel 'micro', desarrollada por Inform Software y Klockner-Moeller, y consiste de dos chips, conexiones field bus e interfaces. Un ASIC (del inglés *Application Specific Integrated Circuit*) circuito integrado de aplicación especifica analógico que maneja las interfaces analógico/digital a un estándar industrial de 12 bits de resolución. Módulos

adicionales pueden extender los periféricos para aplicaciones grandes de hasta 100 señales. Una conexión field bus integrada, basada en RS485, provee futura extensión por red. El cómputo de la lógica difusa y convencional es manejado por un microcontrolador RISC (del inglés *Reduced Instruction Set Computer*) de 16-32 bits. El sistema operativo y rutinas de comunicación, desarrolladas por Klockner-Moeller, están basados en un núcleo multitareas de tiempo real comercial. La RAM (del inglés *Random Access Memory*) memoria de acceso aleatorio interna de 256 KB es expansible por tarjetas de memoria que usan tecnología flash. En base a lo anterior el fuzzyPLC^{TM} es capaz de resolver problemas complejos reales de la automatización industrial.

I.3.3 Antecedentes científicos

Mohammed y Hashim (2007) implementaron un controlador difuso basado en dos atributos de entrada y uno de salida con cinco funciones de correlación para cada entrada de las que resultaran veinticinco reglas difusas utilizando la función “Y” (AND lógico), en el que concluyen que la selección de la tarjeta adecuada depende de los atributos de entradas y salidas a utilizar en el sistema, el costo, el tiempo de desarrollo del sistema y su capacidad de reprogramabilidad. Por todo lo anterior sugieren el uso de la FPGA x2v4000 de la familia Xilinx para implementar controladores difusos ya que satisface todas las características a considerar en el desarrollo de los mismos.

Islam et al. (2007) trabajaron en el diseño e implementación de un controlador difuso aplicado al control de temperatura de un proceso industrial, el sistema de control fue simulado en software y hardware teniendo buena respuesta en ambas pruebas, el modelo matemático simulado en MatLab muestra una respuesta suave, para ambas entradas. La tarjeta utilizada fue una Altera FLEX10KEPF10K70.

Chemali et al. (2006) proponen una nueva técnica funcional de evaluación, la presente aproximación de diseño no solo es llevada a cabo para incrementar la velocidad de respuesta del controlador difuso sino para diseñar un IP Core bien estructurado. Finalmente 3 nuevas técnicas fueron utilizadas en el diseño de este controlador, estas son: múltiple acceso a las LUT, reglas de evaluación difusa y métodos “pipeline” para el cómputo de los datos de defuzzificación.

Masmoudi et al. (2007) proponen el uso de dos controladores difusos en serie para controlar la labor de estacionamiento paralelo, complejidades debidas a la no linealidad de la dinámica del coche le plantean como un gran desafío dependiendo de las condiciones del camino. Monroe (2007) identificó un problema al hacer la implementación en VHDL, esto principalmente debido a las matemáticas enteras, pero el uso de un offset de $10^{(n/4)}$ resolvió dos cuestiones con el valor de salida: incremento en la precisión durante el cálculo y este offset durante el proceso de derivación probó la entrega de un valor aceptable de salida que no requiere compensación. El siguiente paso en esta investigación es implementar el controlador en una base de movimiento y probarlo en el túnel de viento.

Mermoud et al. (2005) proponen el desarrollo de una plataforma de hardware cooperativo coevolutivo en el que se hace énfasis en la importancia de las implementaciones basadas en FPGA, que mientras las implementaciones basadas en microprocesadores son muy flexibles y las basadas en ASIC son las de mejor desempeño, los diseños basados en FPGA proveen ambas ventajas. Al concluir se muestran resultados que indican casi el mismo desempeño que una plataforma basada en microprocesadores.

I.4 Justificación

De acuerdo con lo anterior el uso de controladores difusos se hace presente cuando es difícil modelar la dinámica del sistema a controlar. Mientras la

implementación del sistema de control difuso basado en FPGA se valida con los tiempos de respuesta requeridos por el sistema a controlar. Aunque también otra razón es por la posibilidad de integrar en un solo chip varios controladores difusos. Para ello únicamente diseñando un solo controlador y propagando su uso dentro del chip.

No perdiendo de vista que el procesamiento del control difuso es de manera paralela su implementación en hardware es una mejor aproximación que su contraparte en software, además que conserva la misma flexibilidad por el hecho de que los conjuntos difusos y base de reglas son configurables vía software. En el caso de que se requiera más de un controlador, es posible integrarlo en el mismo chip sin hacer grandes cambios a la arquitectura de éste y conservando los mismos tiempos de respuesta del sistema.

Debido a que la alimentación de los peces pretende ser racionada por el cultivador experto, se puede considerar el control difuso como la mejor solución a este problema, así el cultivador especificará las reglas y conjuntos difusos más convenientes dependiendo de su experiencia en el cultivo de la especie. Por otro lado la idea de integrar el controlador en un solo chip se hace con el fin de minimizar el espacio y energía que este requiere y así poderlo integrar junto con el panel de control de potencia

I.5 Objetivos e hipótesis

I.5.1 Objetivo general

Desarrollar un controlador difuso basado en tecnología FPGA para automatizar alimentadores de peces en sistemas acuícolas cuya respuesta en la dinámica de las variables básicas en el tiempo/espacio real sea independiente de la computadora.

I.5.2 Objetivos particulares

1. Implementar una interfaz usuario que conserve el papel del usuario dentro del sistema de control diseñado previamente para preservar su facilidad de uso.
2. Aprovechar al máximo la interfaz usuario-controlador dentro del SOC (*system on a chip*) para disminuir en mayor medida el uso de periféricos auxiliares.
3. Reforzar la invariancia al escalado del sistema de control para aumentar su aplicabilidad.

I.5.3 Hipótesis general

Al reducir la dependencia de la computadora como elemento de control, así como el correcto diseño de la interfaz usuario-controlador y utilización de la interfaz de acuerdo a su capacidad se asegura la facilidad de su uso y será posible eliminar el uso excesivo de periféricos, por consiguiente un sistema invariante al escalado, es de mayor aplicabilidad.

II. CAPÍTULO 2

II.1 Importancia de la lógica digital

De manera creciente, circuitos y sistemas electrónicos están siendo desarrollados utilizando tecnologías que ofrecen desarrollo rápido de prototipos, programabilidad y capacidad de reutilización (reprogramabilidad y reciclamiento de componentes) para permitir a los sistemas ser desarrollados en un tiempo mínimo, así como permitir reconfiguración en servicio (para actualización de productos normales y mejorar desempeño, proveer capacidades de depuración del diseño, y para la inevitable remoción de errores de diseño), o inclusive reciclado de componentes electrónicos para otra aplicación. Estos aspectos requeridos por el reducido tiempo-mercado e incrementadas complejidades para aplicaciones – desde teléfonos celulares hasta computadoras, control, instrumentación y aplicaciones de prueba. El principal enfoque de desarrollar electrónicos con las capacidades previamente mencionadas ha estado en el dominio digital debido a las técnicas de diseño y naturaleza de las señales digitales de ser adecuadas para la reconfiguración.

II.2 Tipos de lógica programable

El SPLD fue introducido previo al CPLD y FPGA. Los tres principales tipos de arquitecturas SPLD – arreglo lógico programable (PLA), arreglo de lógica programable (PAL), y la lógica de arreglo genérico (GAL).

El PLA consiste de dos planos AND y OR. El plano AND consiste de interconexiones programables a lo largo con compuertas AND. El plano OR consiste de interconexiones programables a lo largo con compuertas OR. De acuerdo con ello, existen cuatro entradas en el PLA y cuatro salidas de la PLA. Cada una de estas entradas puede ser conectada a una compuerta AND con cualquier otra de las demás entradas al conectar los puntos de cruce de las líneas

de interconexión horizontales y verticales en la interconexión programable de la compuerta AND.

Inicialmente, los puntos de cruce no están eléctricamente conectados, pero al configurar el PLA se conectarán puntos particulares de interconexión. Desde esta perspectiva, la compuerta AND es vista con una sola línea a la entrada. Esta representación es por convención, pero ello representa que cualquiera de esas entradas puede ser conectada. Entonces, para un PLA de cuatro entradas, la compuerta AND también tiene cuatro entradas. La salida de cada una de las compuertas AND es aplicada a la interconexión programable de una compuerta OR.

Una vez más, inicialmente los puntos de cruce no están eléctricamente conectados, pero al configurar el PLA se conectaran puntos particulares de interconexión. En tal vista, la compuerta OR tiene una única línea de entrada, ello por convención, pero significa que cualquiera de las salidas de las compuertas AND puede ser conectada a la entrada de las compuertas OR.

El PAL es similar a la arquitectura del PLA, pero ahora solo existe un único plano programable, los planos AND y OR son combinados. Esta arquitectura es más simple que la del PLA y se elimina el tiempo de retraso asociado con el plano de interconexión OR, por tanto produciendo un diseño más rápido. Como siempre, esto viene a costo de la flexibilidad – el PAL es menos flexible en las rutas en que un diseño digital puede ser implementado en comparación con el PLA.

Las arquitecturas PLA y PAL permiten la implementación de diseños lógicos combinacionales. Si el diseño provee retroalimentación de las salidas a la entrada, entonces es posible implementar latches y registros biestables, como resultado también la implementación de circuitos de lógica secuencial. Esto es posible en algunos PAL comercialmente disponibles. Adicionalmente, algunos dispositivos PAL también proveen la salida disponible desde el plano OR o vía un registro

biestable conectado a esta misma. Así, incrementa el tipo de circuitos lógicos secuenciales que pueden ser implementados y por lo tanto incrementa la usabilidad de un dispositivo PAL.

El GAL, los dispositivos PLA y PAL son programables una sola vez (*one time programmable*, OTP) basados en PROM (*Programmable Read Only Memory*), por lo tanto la configuración de un PLA o PAL no puede ser cambiada una vez que han sido configurados. Esta limitación significa que el dispositivo configurado tendrá que ser descartado y configurar un nuevo dispositivo. La GAL, similar a la arquitectura de la PAL, usa EEPROM (*Electronically Erasable Programmable Read Only Memory*) y puede ser reconfigurado.

El CPLD está un paso arriba en complexidad al SPLD, éste se construye en la arquitectura del SPLD y crea un diseño más grande. Consecuentemente, el SPLD puede ser usado para integrar funciones de un número de CI discretos digitales en un solo dispositivo y el CPLD puede ser usado para integrar las funciones de un número de SPLDs en un solo dispositivo. La arquitectura CPLD se basa en un pequeño número de bloques lógicos y un interconector global programable. La arquitectura de un CPLD genérico es mostrada en la Figura 2.1, el CPLD se puede considerar el predecesor del FPGA. Existen ciertas similitudes como los bloques de interconexión programable y la equivalencia arquitectural entre las macroceldas y las celdas lógicas.

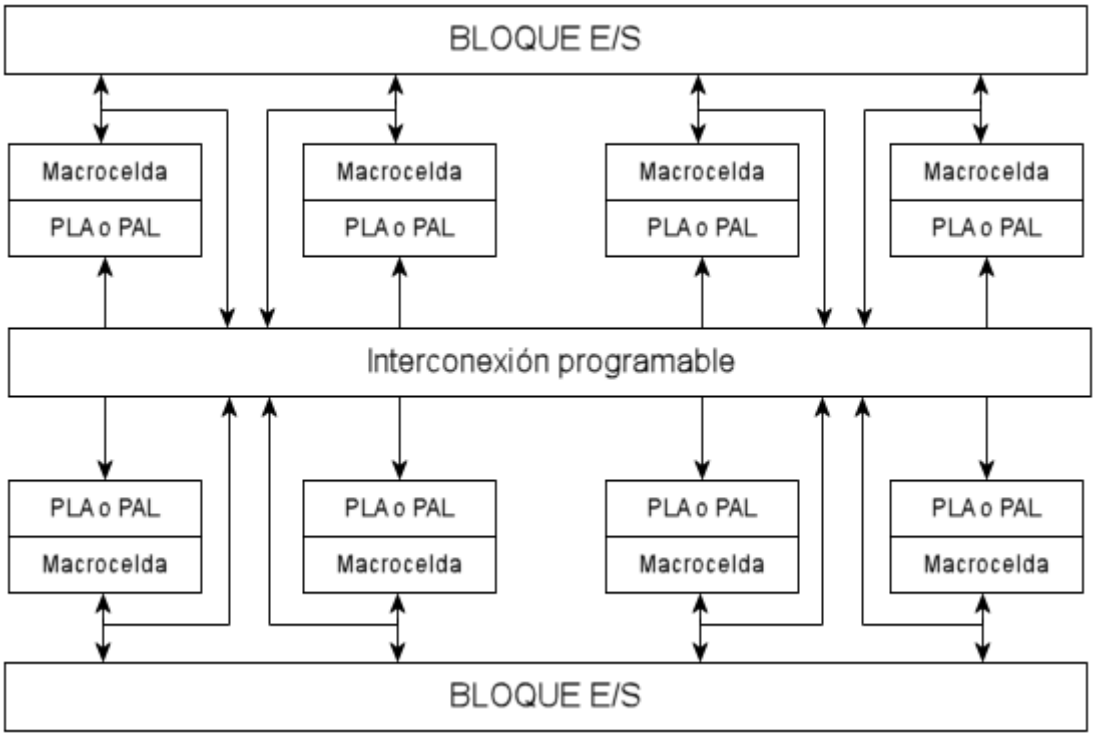

Figura 2.1. Arquitectura genérica CPLD.

El CPLD consiste de un número de bloques lógicos (algunas veces referidos como bloques funcionales), cada uno de los cuales contiene una macrocelda y un circuito de arreglo PLA o PAL. En esta vista, ocho bloques son mostrados. La macrocelda provee circuitería adicional para ajustar a la necesidad de salidas con registros, junto con el control de la polaridad de la señal. El número de bloques lógicos en un CPLD varía dependiendo del dispositivo; mientras más bloques lógicos, más grande el diseño que puede ser configurado.

En el centro del diseño se encuentra un interconector global programable. Este interconector permite conexiones a las macroceldas del bloque lógico y los arreglos de celdas de E/S (las celdas digitales de E/S del CPLD conectadas a los pines del empaquetado CPLD). El interconector programable es usualmente basado en un interconector basado en arreglos o multiplexores:

- Interconector basado en arreglos permite a cualquier señal en el interconector programable conectarse a cualquier bloque lógico en el CPLD. Ésto puede ser logrado al permitir ruteo horizontal y vertical

en el interconector programable y permitiendo a los puntos de cruce ser conectados o desconectados (la misma idea que con el PLA y PAL), dependiendo de la configuración del CPLD.

- Interconector basado en multiplexores usa multiplexores digitales conectados a cada una de las entradas de las macroceldas con los bloques lógicos. Señales específicas con el interconector programable son conectadas con entradas específicas de los multiplexores. Esto podría no ser práctico para conectar todas las señales internas con el interconector programable a las entradas de todos los multiplexores debido a consideraciones del tamaño y velocidad de operación.

Así como el CPLD, el FPGA está un paso arriba en complejidad al SPLD al crear un diseño mucho más grande; por el contrario a la arquitectura del CPLD, la arquitectura del FPGA fue desarrollada utilizando un concepto básico diferente. La arquitectura se basa en un arreglo regular de celdas lógicas programables básicas (LC) y una matriz interconectora programable rodeando las celdas lógicas. La arquitectura presentada en la Figura 2.2 es de mero propósito informativo y difiere entre fabricantes; Las celdas lógicas y bloques lógicos complejos son equivalentes a nivel arquitectural.

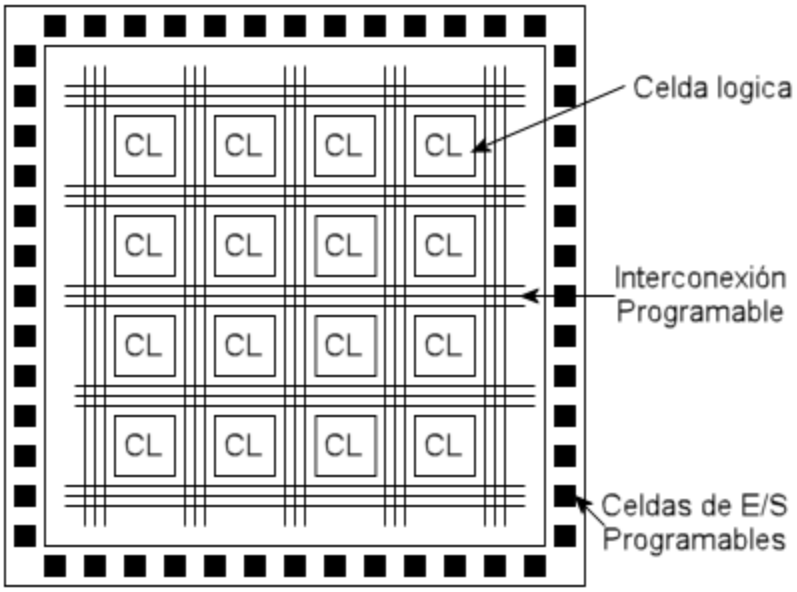

Figura 2.2. Arquitectura genérica FPGA.

El arreglo de celdas lógicas programables básicas y la matriz interconectora programable forman el núcleo del FPGA. Este está rodeado por celdas programables de E/S. El interconector programable se ubica en los canales de ruteo. Los detalles específicos del diseño para cada una de las funciones principales (celdas lógicas, interconector programable, y E/S programables) varían dependiendo del fabricante.

II.3 Tecnologías de configuración de PLD

El PLD es configurado descargando una configuración particular del circuito como una secuencia de valores de lógica binaria (secuencia de 0s y 1s). La configuración será contenida en un archivo de configuración en el PC o estación de trabajo en la que el diseño fue creado usando las herramientas EDA (*Electronic Design Automation*) requeridas. El software con la aplicación de descarga leerá el archivo de configuración y descargará el contenido en el PLD. Estos valores son almacenados en memoria con el dispositivo, en el cual la memoria puede ser volátil o no volátil:

- Memoria volátil: Cuando se almacenan datos en la memoria, la información es retenida en la memoria tanto como ésta se mantenga conectada al suministro de energía. Una vez que el suministro de energía le ha sido removido, los contenidos de la memoria (los datos) se pierden. La mayoría de los FPGAs utilizan memoria volátil basada en SRAM (*Static Random Access Memory*). Por lo tanto, siempre que el suministro de energía es desconectado del FPGA, la configuración del FPGA es pérdida y cuando el suministro de energía del FPGA es nuevamente conectado, entonces la configuración debe cargarse nuevamente en la SRAM.
- Memoria no volátil: Cuando se almacenan datos en la memoria, los datos se retienen en la memoria aun cuando el suministro de energía ha sido removido. Algunas FPGAs utilizan tecnología anti-fusible para almacenar la configuración del FPGA; FPGAs de nueva generación utilizan memoria flash.

II.4 Diseño digital

A pesar que el mundo en el que vivimos es analógico de natura; en los circuitos y sistemas electrónicos, los circuitos digitales son ampliamente usados y pueden ser diseñados para desempeñar diversas acciones que eran originalmente tomadas en circuitería analógica. Así proveyendo beneficios potenciales por encima de la operación de circuitos analógicos. Por lo tanto, una señal puede ser de uno de dos tipos: analógica o digital.

Una señal analógica es una señal continua o discreta en el dominio del tiempo con una amplitud continua y un valor entre un límite superior e inferior, pero puede ser continua en el tiempo o bien, discreta.

Una señal digital es una señal continua o discreta en el dominio del tiempo con valores discretos entre un límite superior e inferior. Tales valores discretos son

representados por valores numéricos y son adecuados para procesamiento digital de señales. Si la señal discreta en tiempo ha sido derivada de una señal continua en tiempo por muestreo, entonces la señal muestreada es convertida en una señal digital por cuantificación. La cuantificación produce un número finito de valores a partir de una señal de amplitud continua. Es común el uso del sistema de numeración binaria (dos valores, 0 o 1) para representar un número digitalmente.

En el dominio digital, la elección de la tecnología de implementación es esencialmente dirigida al uso de lógica digital de funcionalidad dedicada (y fija), uso de software programado o sistemas basados en procesadores (diseñados en base a microprocesadores, μP; microcontroladores, μC; o procesadores digitales de señales, DSP), o al uso de hardware-configurado; dispositivos de lógica programable (PLD), bien simple (SPLD), complejo (CPLD), o el arreglo de compuertas programable (FPGA). Memoria utilizada para el almacenamiento de datos y código programa es integral para diversos circuitos y sistemas digitales.

De acuerdo con Smith (1999) la elección inicial para implementar el circuito digital se encuentra entre un producto estándar IC y un ASIC:

- Producto estándar IC: un componente electrónico listo para ser utilizado que ha sido diseñado y manufacturado para un dado propósito, o rango de uso, y que esta comercialmente disponible. Éste es comprado de un proveedor de componentes o bien directamente del diseñador o fabricante.
- ASIC: un circuito integrado que ha sido específicamente diseñado y manufacturado para una aplicación en particular.

Para diversas aplicaciones, el desarrollo de un sistema electrónico basado en un producto estándar IC es más efectivo en costo que el diseño ASIC. Al tomar el proyecto de diseño de un ASIC también se requiere acceso a experiencia en el diseño de IC, herramientas de diseño de IC asistido por computadora (CAD), y

adecuadas capacidades de verificación y manufactura. Sin importar si se opta por la aproximación de diseño por medio de productos estándar IC o diseño ASIC, el tipo de IC usado o desarrollado será de uno de cuatro tipos:

- Funcionalidad fija: Estos IC han sido diseñados para implementar una funcionalidad específica y no puede ser cambiada. El diseñador usa un conjunto de estos IC para implementar una dada funcionalidad del circuito. Cambios al circuito requieren de un completo rediseño del circuito y el uso de diferentes IC de funcionalidad fija.
- Procesador: La mayoría de la gente está familiarizada con procesadores en uso diario; el corazón de una PC es un microprocesador. Este componente ejecuta un software programado para implementar una requerida funcionalidad. Al cambiar este software, el procesador operará una función diferente. Los tres tipos de procesadores son microprocesador (µP), microcontrolador (µC), y procesador digital de señales (DSP).
- Memoria: La memoria es usada para almacenar, proveer acceso a, y permitir modificación de datos y código de programa para uso con un circuito o sistema electrónico basado en procesador. Los dos tipos de memoria son ROM (*read only memory – memoria de solo lectura*) y RAM (*random access memory – memoria de acceso aleatorio*). La ROM es usada para mantener código de programa que debe ser retenido cuando el suministro de energía es removido; este es almacenamiento no volátil. El código puede ser programado cuando la memoria es fabricada (*mask programable ROM*), programable eléctricamente de una sola vez (*PROM, programmable ROM*) o electrónicamente programada de múltiples veces. Capacidad de múltiple programación requiere la habilidad de borrar antes de programar, que está disponible en EPROM (eléctricamente programable ROM, borrada usando luz ultravioleta [UV]), EEPROM o

E^2PROM (eléctricamente borrable PROM), o flash (también eléctricamente borrable). La RAM es usada para mantener datos y código de programa que requiere rápido acceso y la habilidad de modificar los contenidos durante operación normal. La RAM difiere de la memoria de solo lectura (ROM) en que ésta puede leer desde y escribir en el circuito de aplicación normal. De cualquier manera, la memoria flash puede ser también referida como RAM no volátil (NVRAM). La RAM es considerada por proveer almacenamiento volátil debido a que, por el contrario a la ROM, los contenidos de la RAM son perdidos cuando el suministro de energía es removido. Existen dos principales tipos de RAM: RAM estática (SRAM) y RAM dinámica (DRAM).

- PLD: El dispositivo lógico programable, es un IC que contiene celdas lógicas digitales e interconexión programable (Skahill, 1996; Maxfield 2004) para permitir al diseñador la configuración de las celdas lógicas y la interconexión programable en el IC para formar un circuito electrónico digital con un solo empaquetado IC. En éste, los recursos físicos (hardware disponible para uso) son configurados para desempeñar la funcionalidad requerida. Al cambiar la configuración del hardware, el PLD desempeña una función diferente. Existen tres tipos de PLD disponibles: el dispositivo lógico programable simple (SPLD), el dispositivo lógico programable complejo, y el arreglo de compuertas de campo programable (FPGA).

Ambos el procesador y el PLD permiten al diseñador, la implementación y cambio de la funcionalidad de un IC por medio del programa software o la configuración del hardware. Para evitar la potencial confusión, de estos términos se enuncian los siguientes términos utilizados para diferenciar el PLD del procesador:

- El PLD será configurado usando una configuración de hardware.

- El procesador será programado usando un programa software.

Un ASIC puede ser diseñado para crear cualquiera de los cuatro productos estándar IC (funcionalidad fija, procesador, memoria, o PLD). Un producto estándar IC es diseñado de la misma manera que un ASIC, por lo tanto cualquiera que haya participado en el diseño de un ASIC, fabricación, e instrumental de verificación puede crear un equivalente a un producto estándar IC (dada la patente y papeleo legal de IP [*propiedad intelectual*] para diseños existentes y dispositivos tomados en cuenta).

No importando el grado de complexidad del diseño de un circuito digital, y los tipos de operaciones que requiere tomar, la operación es basada en un reducido número de circuitos lógicos combinacionales y secuenciales básicos, elementos que son conectados para formar la requerida operación del circuito:

- Lógica combinacional: Un circuito de lógica combinacional es definido por una expresión Booleana, y la salida del circuito (en términos lógicos) es una función de los valores de las entradas lógicas, las compuertas lógicas usadas (AND, OR, etc.), y la manera en que las compuertas lógicas son conectadas (Stonham 1988; Tocci et al. 2004). La salida se viene a ser un valor final cuando las entradas cambian después de un tiempo finito, que es el tiempo requerido para que se propaguen los valores lógicos a través del circuito dados los retardos de propagación de la señal para cada una de las compuertas lógicas y cualquier retardo de las interconexiones entre las compuertas lógicas. Los elementos básicos de los circuitos lógicos combinacionales (compuertas) son:
    - Compuerta AND
    - Compuerta NAND
    - Compuerta OR
    - Compuerta NOR

    - Compuerta OR-Exclusiva (EX-OR)
    - Compuerta NOR-Exclusiva (EX-NOR)
    - Inversor
    - Buffer
- Lógica secuencial: En un circuito de lógica secuencial, la salida del circuito viene a ser un valor basado en los valores de las entradas lógicas, las compuertas lógicas usadas, la manera en que éstas son conectadas, y en el estado actual del circuito (Stonham 1988; Tocci et al. 2004). En un circuito de lógica secuencial síncrono, el cambio en la salida ocurre en el cambio del borde de una señal de reloj (de 0 a 1 o de 1 a 0) o en el nivel de la señal de reloj (0 ó 1 lógico). Por el contrario, un circuito de lógica secuencial asíncrono no usa entrada de reloj. En el circuito de lógica secuencial, el circuito mantendrá o recordará su valor actual (estado) y cambiará estado solamente en cambios de reloj o señales de entrada. Un circuito de lógica secuencial puede también contener señales de control adicionales para establecer o restablecer el circuito en un estado conocido cuando el circuito es inicialmente encendido o durante operación normal del circuito. Los elementos básicos de circuitos de lógica secuencial (compuertas) son:
    - Flip-flop S-R
    - Flip-flop J-K
    - Flip-flop de conmutación
    - D-latch
    - Flip-flop tipo D

II.5 FPGAs

Los FPGAs fueron el siguiente paso de los CPLD. En vez de un arreglo fijo de compuertas, el FPGA usa el concepto de un Bloque Lógico Complejo (CLB – Complex Logic Block). Este es configurable y permite no solo el ruteo en el

dispositivo, sino también cada bloque lógico puede ser configurado óptimamente. Un CLB típico es mostrado en la Figura 2.3 (Wilson, 2007), mostrando de manera esquemática sus componentes internos, este tiene la habilidad de trabajar de manera síncrona o asíncrona.

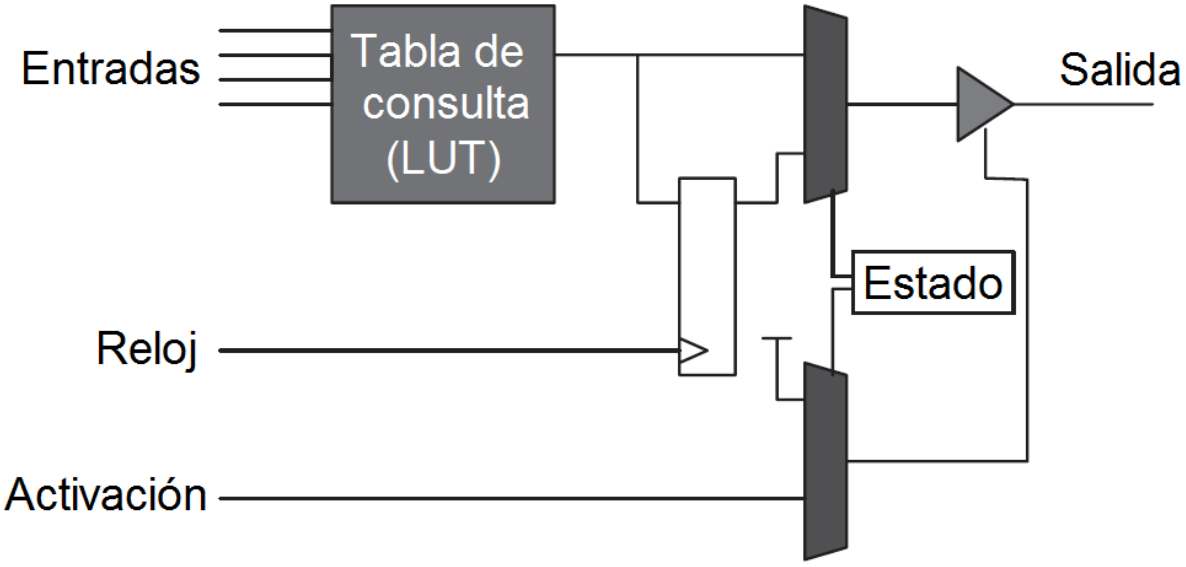

Figura 2.3. Bloque lógico complejo.

El CLB tiene una tabla de consulta (LUT – Look Up Table) que puede ser configurada para dar un tipo específico de función lógica cuando es programada. También tiene un flip-flop síncrono tipo D que permite al CLB ser combinacional (sin señal de reloj) o síncrono (con señal de reloj), y adicionalmente cuenta con una señal de activación. Un CLB de Xilinx es mostrado en la Figura 2.4, ésta muestra claramente las 2 LUTs de 4 entradas, varios multiplexores y flip-flops en un dispositivo real. En el caso de la arquitectura manejada por este fabricante permite la opción de controlar registros de manera asíncrona, en el caso del fabricante Altera esto no es posible.

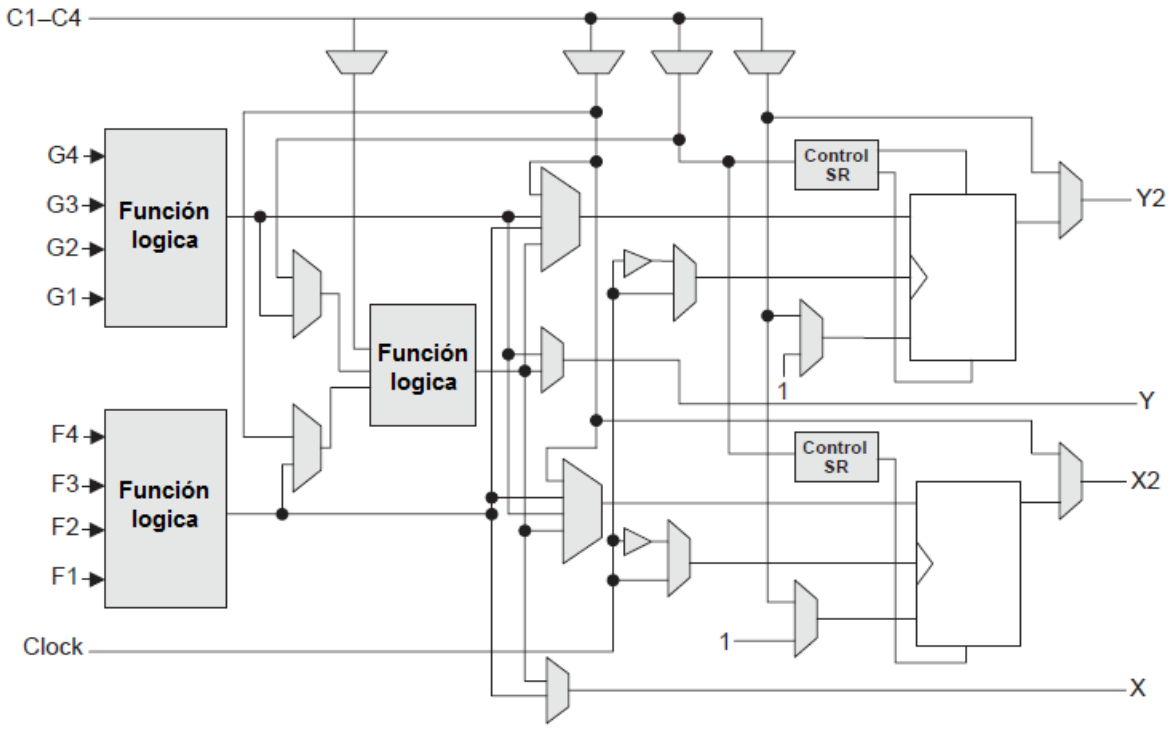

Figura 2.4. Bloque lógico complejo de FPGA del fabricante Xilinx.

Un FPGA típico tendrá cientos o miles de CLBs, de diferentes tipos, en un solo dispositivo permitiendo dispositivos bastante complejos ser implementados en un solo chip y configurado fácilmente. FPGAs modernos tienen capacidad suficiente para contener un número de procesadores de 32 bits en un solo dispositivo. El esquema de un FPGA típico (en términos de CLB) es mostrado en la Figura 2.5; los subbloques C, y S, indican bloques combinacionales y secuenciales, mientras el bloque CLKM indica la entrada de reloj maestro.

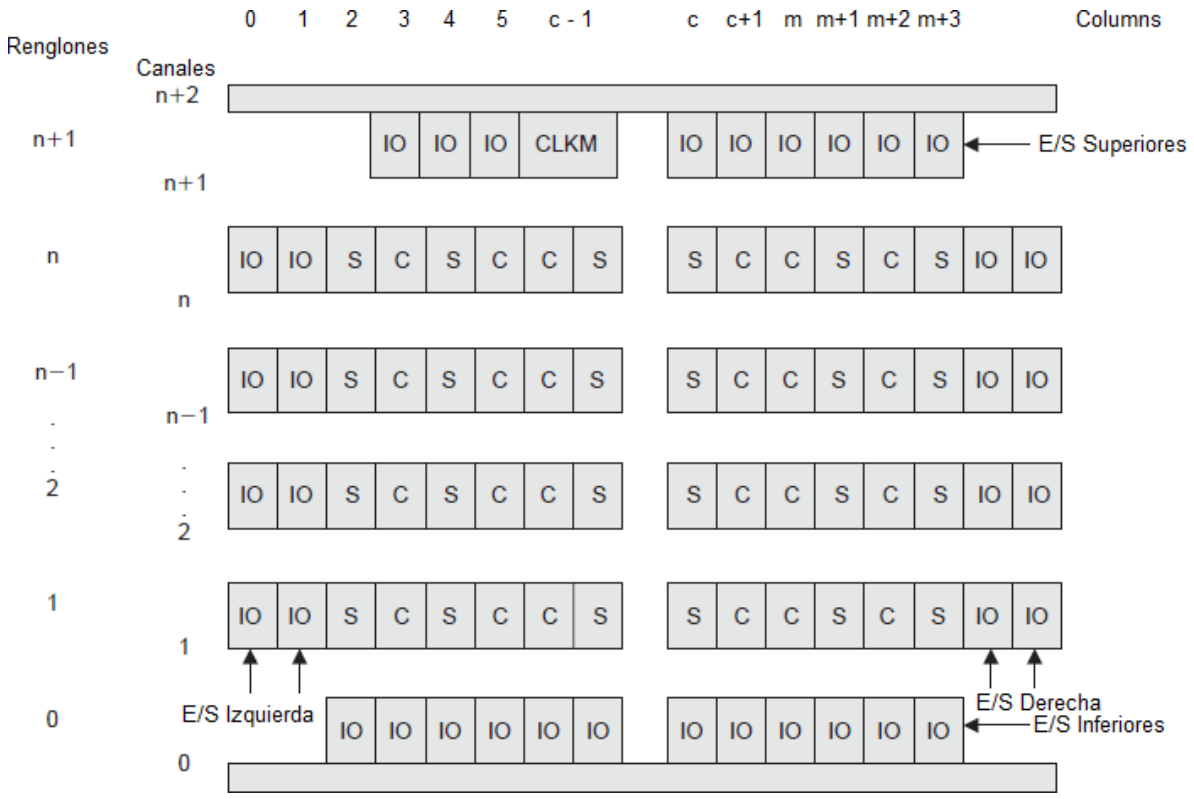

Figura 2.5. Estructura de CLBs dentro del FPGA.

II.6 Técnicas de diseño FPGA

Cuando se diseña usando VHDL, estas funciones necesitan ser mapeadas en los bloques lógicos de bajo nivel en el FPGA. En orden para hacer esto, se necesita llevar a cabo tres funciones específicas:

1. Mapeo: Funciones lógicas mapeadas en CLBs.
2. Colocación: CLBs colocados en FPGA.
3. Ruteo: Conexiones entre CLBs son ruteadas.

Claramente es imposible diseñar manualmente usando los complejos diseños de hoy día. Por lo tanto se cuenta en software de sintetizado para convertir la descripción del diseño en VHDL en funciones lógicas que puedan ser mapeadas en los CLBs del FPGA. Este flujo de diseño es un proceso interactivo incluyendo optimización e implicando un completo flujo de diseño.

II.6.1 Restricciones de diseño usando FPGAs

Es bastante fácil producir diseños irrealistas usando VHDL si la plataforma FPGA al que está dirigido no es considerada cuidadosamente. Obviamente las FPGAs tienen un número limitado de bloques lógicos y recursos de ruteo, y el diseño debe considerar esto. El estilo de código VHDL usado por el diseñador deberá hacer el mejor uso de los recursos, el código VHDL puede ser transferido entre tecnologías, pero podrá ser necesaria la reescritura de éste para mejores resultados debido a estas restricciones.

II.6.2 Diseñando con HDLs

La entrada de diseño en lenguaje de descripción de hardware (HDL) es basada en la creación y uso de descripciones basadas en texto de un circuito o sistema lógico digital. Aquí, usando un HDL particular (los dos estándares IEEE en uso común en la industria y academia son Verilog®-HDL y VHDL), la descripción de un circuito puede ser creada a diferentes niveles de abstracción desde la descripción de compuertas lógicas básicas de acuerdo con la sintaxis del lenguaje (el arreglo gramatical de las palabras y los símbolos utilizados en el lenguaje) y semántica (el significado de las palabras y símbolos usados en el lenguaje).

Verilog®-HDL y VHDL ambos establecidos en los estándares IEEE:

- Verilog®-HDL, IEEE Std 1364™-2005
- VHDL, IEEE Std 1076™-2002

Verilog®-HDL fue lanzado en 1983 por Gateway Design System Corporation, junto con un simulador Verilog®-HDL. En 1985, el lenguaje y simulador fueron mejorados con la introducción del simulador Verilog-XL®. En 1989, Cadence Design Systems, Inc. trajo Gateway Design System Corporation, y a principios de 1990, Verilog®-HDL y Verilog-XL® fueron separados para convertirse en dos productos independientes. Verilog®-HDL, hasta entonces un

lenguaje propietario, fue liberado en el dominio público para facilitar la diseminación del conocimiento relacionado a Verilog®-HDL y permitir a Verilog®-HDL competir con VHDL, ya existente como un lenguaje no propietario.

También en 1990, Open Verilog International (OVI) fue formado como un consorcio industrial consistiendo de vendedores de ingeniería asistida por computadora (computer-aided engineering - CAE) y usuarios de Verilog®-HDL para controlar las especificaciones del lenguaje. En 1995, Verilog®-HDL fue revisado y adoptado como estándar (Std) IEEE 1364 (convirtiéndose en IEEE 1364-1995). En 2001 y 2005, el estándar fue revisado y la versión actual es IEEE 1364™-2005.

VHDL (VHSIC HDL, *very high-speed integrated circuit hardware description language – Lenguaje de descripción de circuitos integrados de muy alta velocidad*) inicio su vida en 1980 bajo el requerimiento de un Departamento de Defensa de los Estados Unidos (DoD) para el diseño de circuitos digitales siguiendo una metodología de diseño común para proveer la habilidad de auto-documentación y reúso con nuevas tecnologías. El desarrollo de VHDL comenzó en 1983, y el lenguaje fue revisado en 1987 para convertirse en IEEE Std 1076-1987. El lenguaje ha sido revisado desde 1993, 2000, y 2002, siendo la última versión 1076-2002. VHDL también tiene un número de estándares asociados relacionados con el modelado y síntesis.

Cuando se diseña con HDLs, el diseñador debe decidir que lenguaje utilizará y a qué nivel de abstracción de diseño trabajar. Al considerar la elección de lenguaje, un número de factores intervienen, incluyendo:

- la disponibilidad de herramientas EDA (*Electronic Design Automation – Automatismo para Diseño Electrónico*) para soportar el uso del lenguaje (incluyendo capacidades de manejo del diseño y disponibilidad de uso de herramientas con el proyecto)
- conocimiento previo

- referencias personales
- disponibilidad de modelos de simulación
- capacidades de síntesis
- cuestiones comerciales
- reúso de diseño
- requerimientos para aprender un nuevo lenguaje y las capacidades del lenguaje
- flujos de diseño soportados con una organización
- existencia de estándares para el lenguaje
- acceso a los estándares del lenguaje
- legibilidad del código HDL resultante
- habilidad de crear los niveles de abstracción de diseño requeridos, y lenguaje y/o soporte de herramienta EDA para estos niveles de abstracción
- acceso a herramientas de soporte de diseño para el lenguaje (como la existencia de herramientas de verificación automática de código y herramientas de generación de documentación)

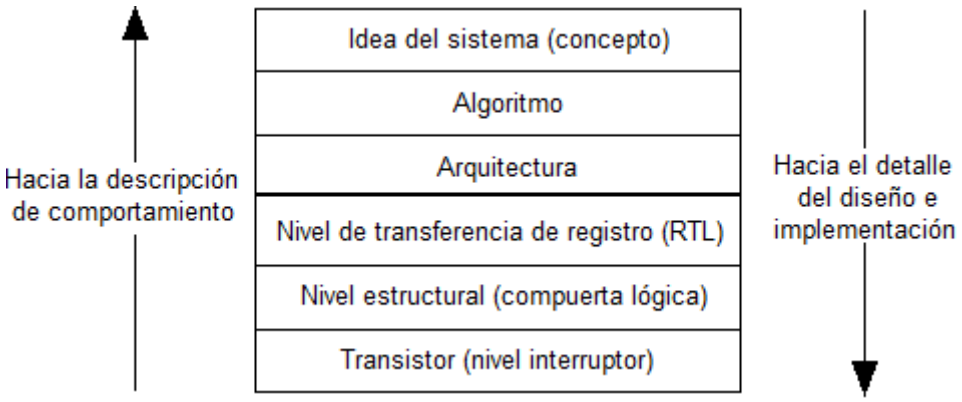

Figura 2.6. Niveles de abstracción en la descripción del diseño.

La Figura 2.6 muestra los diferentes niveles de abstracción del diseño que son usados. La selección del lenguaje a utilizar en el desarrollo de un diseño electrónico esta bastantemente relacionada con el nivel de abstracción al que se pretende hacer el desarrollo del diseño. Uno o más niveles son usados en un

proyecto de diseño típico. Iniciando por el más alto nivel de abstracción, la idea o concepto del sistema es el nivel alto inicial de descripción del diseño que provee las especificaciones de diseño.

El nivel algoritmo describe un comportamiento de alto nivel, una descripción de la operación del sistema a un nivel de descripción matemático de comportamiento. Ni la idea del sistema ni el algoritmo describen la manera en que el comportamiento del diseño será implementado. La estructura del algoritmo en hardware es descrito por la arquitectura, identificando los bloques funcionales de alto nivel a usar y la manera en que las funciones son conectadas entre sí. Los niveles algoritmo y arquitectura describen el comportamiento del diseño a ser verificado en la simulación.

El siguiente nivel, hacia abajo, de la arquitectura es el nivel de trasferencia de registro (*RTL – Register Transfer Level*), que describe el almacenamiento (en registros) y flujo de datos en el diseño, con operaciones lógicas efectuadas en los datos. En este nivel que es usualmente utilizado por las herramientas de síntesis para convertir la descripción del diseño en un nivel estructural (la lista de redes del diseño en términos de compuertas lógicas y cableado de interconexión entre las compuertas lógicas). Las compuertas lógicas son por sí mismas implementadas usando transistores. El HDL puede también soportar descripciones al nivel interruptor que modelan la operación de transistor como un interruptor (encendido/apagado).

El flujo de diseño típico para un sistema o circuito digital usando VHDL es mostrado en la Figura 2.7. En la mayoría de las ocasiones la decisión de usar VHDL sobre otros lenguajes como Verilog o SystemC, tendrá poco que ver con la elección del diseñador, y más con la disponibilidad de software. De la idea inicial del diseño, una descripción de comportamiento del diseño es escrita en VHDL. Ésta es simulada para verificar su operación y determinar que la descripción

coincida con la idea de diseño (con la idea de diseño en forma de una especificación de diseño).

En VHDL, el control de la simulación y los estimuladores de prueba a aplicar son creados con la plataforma de prueba. Cuando la descripción del diseño a nivel de comportamiento ha sido validada satisfactoria y completamente a través de la simulación, el diseño es traducido a código RTL. Esto puede ser desempeñado automática o manualmente, sí una herramienta de síntesis adecuada está disponible.

El código RTL VHDL resultante es entonces sintetizado usando el mismo conjunto de criterios que en el diseño a nivel de comportamiento, y los resultados de la simulación son comparados para probar que el código RTL se desempeña en la misma manera que el código a nivel de comportamiento. Si se perciben diferencias, entonces el código RTL es modificado para asegurar la equivalencia.

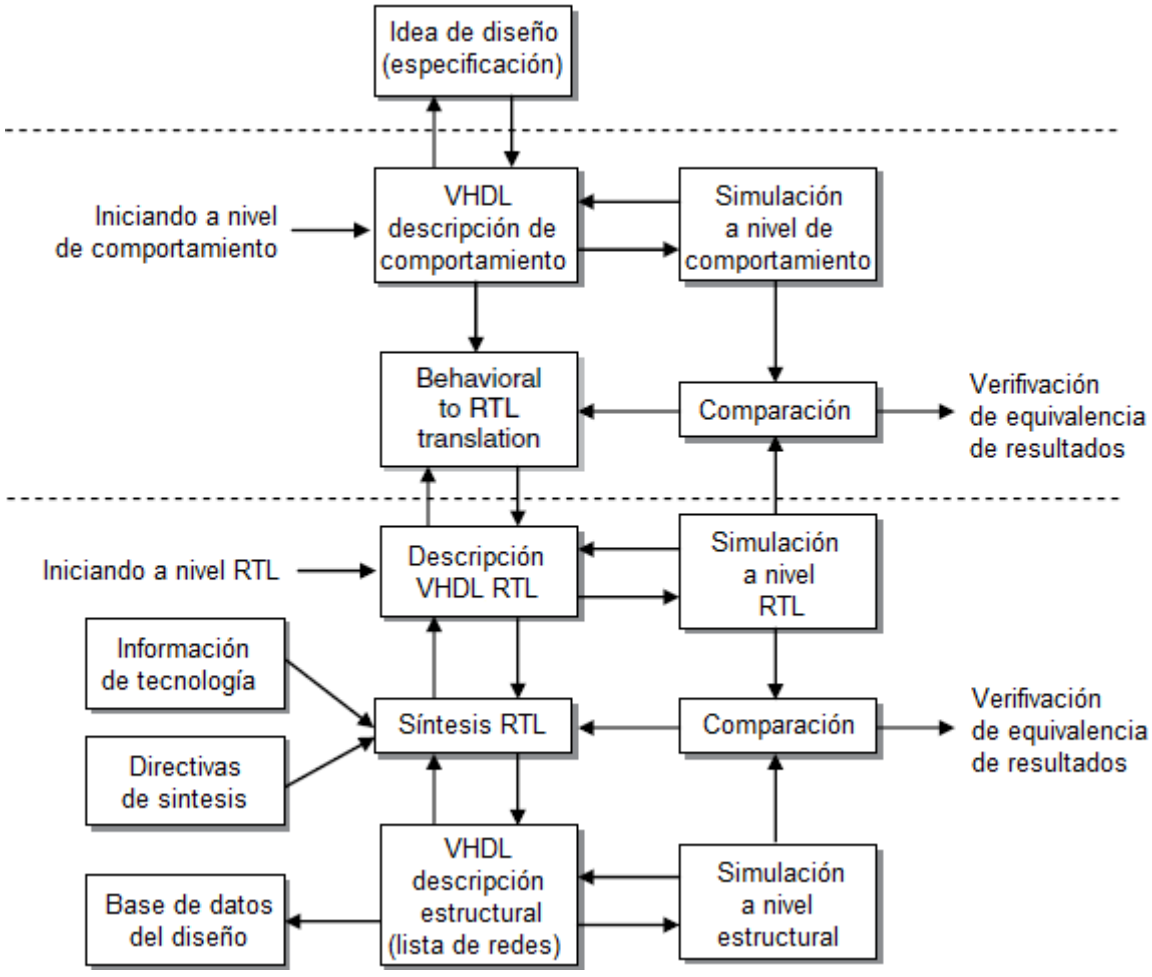

Figura 2.7. Flujo de diseño típico usando HDLs.

Resumiendo: el punto de entrada del diseño es la idea de diseño. En la Figura 2.7, la idea es después codificada en VHDL como una descripción de comportamiento, ésta es una aproximación. En diversas situaciones, el código RTL es generado directamente de la idea, omitiendo la etapa de descripción de comportamiento.

El siguiente paso es desempeñar la síntesis en el código RTL para producir una lista de redes de diseño para la tecnología destino (ASIC o PLD). Información acerca de la tecnología y un conjunto de directivas de síntesis por parte del usuario son requeridas para el control de la operación de la síntesis. Los diseños pre y post-síntesis son después comparados al simular la lista de redes de post-síntesis y checando esto contra el código a nivel RTL en busca de equivalencia.

En caso de realización satisfactoria en este paso, la lista de redes es almacenada en la base de datos del diseño para su uso.

II.6.3 Síntesis lógica

Una característica importante del diseño con HDLs es la necesidad de síntesis lógica, a partir de aquí simplemente llamada síntesis. Síntesis es los medios por los cuales una descripción HDL es convertida (traducida) a una lista de redes del circuito que identifica las compuertas lógicas usadas y el alambrado de interconexión. En el proceso de síntesis, un diseño inicial basado en HDL que es independiente de la tecnología (no describe nada relacionado con la tecnología final de implementación) es convertido a una lista de redes dependiente de una tecnología especifica. Como tal, solo en la etapa de síntesis el diseño es fijado a una tecnología de implementación en particular. Esta es una ventaja para el diseñador de manera que el mismo código HDL inicial puede ser utilizado para diferentes tecnologías, particularmente importante si un diseño será migrado a una nueva tecnología de implementación, como sucede frecuentemente en proyectos de diseño.

El proceso de diseño básico es mostrado en la Figura 2.8, los números en las flechas indican el orden en que el proceso es desempeñado, y cuyo producto final es designado a un fabricante en particular. Las directivas de síntesis indican si el diseño será optimizado para el uso del chip, consumo de energía o bien velocidad de operación del diseño. Una herramienta requiere información específica como rutinas de instalación de herramienta, librerías de tecnología, y directivas de síntesis. Las rutinas de instalación de herramienta configuran la herramienta de síntesis para la plataforma de cómputo específica en que esta es instalada. Las librerías de tecnología proveen información específica relacionada con la tecnología de implementación destino. Las directivas de síntesis son aplicadas por el usuario para dirigir la herramienta de síntesis durante la síntesis del diseño.

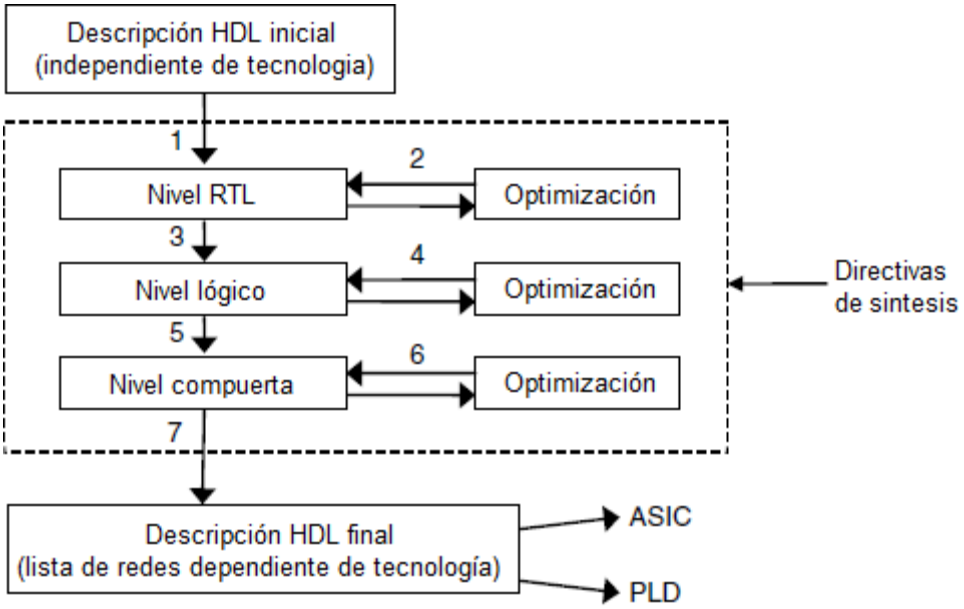

Figura 2.8. Proceso de síntesis básico.

La síntesis consiste de siete pasos, identificados en la Figura 2.8. La descripción HDL inicial es traducida (1) a un nivel de descripción RTL. Esta forma es optimizada (2) y después traducida a una descripción a nivel lógico (3). Esta es optimizada (4) y traducida (mapeada) a una descripción a nivel de compuerta (5). Esta es optimizada (6), y el resultado es traducido a la lista final de redes (7). En cada paso, la descripción creada es más cercana a la lista final de redes. El diseñador establece directivas de síntesis específicas para dirigir la herramienta de síntesis en la creación de la lista de redes del diseño. Las restricciones son típicamente tamaño, velocidad, y consumo de energía. Aplicando diferentes restricciones de diseño típicamente resulta en diferentes listas finales de redes.

Estas consideraciones aplican a la síntesis lógica (comúnmente referida como síntesis). Sin embargo, también existe la síntesis física que se relaciona a la síntesis automática de las capas de diseño en el diseño de circuitos integrados a nivel silicio.

Cuando código RTL es sintetizado, este es referido como síntesis RTL. Cuando código HDL a nivel comportamiento es sintetizado, este es referido como síntesis de comportamiento. Adicionalmente el código HDL inicial debe ser creado de tal manera que sea sintetizable. Una descripción de diseño HDL puede ser creada así como simulada y acertada la correcta operación, pero en ciertas circunstancias el código HDL escrito como tal podría no ser sintetizable.

II.7 Conceptos básicos VHDL

En la presente subsección se presenta en forma de resumen las estructuras de más común uso en VHDL. El código se divide principalmente en tres partes:

Parte superior identifica las referencias a librerías y paquetes a usar con el diseño.

Parte media identifica la entidad del diseño.

Parte inferior identifica la arquitectura del diseño.

II.7.1 Declaración de entidad

De acuerdo con el manual de referencia del estándar IEEE 1076-1987 (1987) el diseño de entidad es la abstracción primaria del hardware en VHDL. Esta representa una porción del diseño de hardware que tiene bien definidas entradas y salidas; y desempeña una función bien definida. Un diseño de entidad puede representar un sistema entero, un subsistema, una tarjeta, un chip, una macrocelda, una compuerta lógica, o cualquier nivel de abstracción intermedio.

Un diseño de entidad puede ser descrito en términos de bloques jerárquicos, cada uno de los cuales representa una porción del diseño entero. Un diseño de entidad puede también ser descrito en términos de componentes interconectados. Cada componente del diseño de entidad puede estar relacionado a un diseño de entidad de menor nivel para definir la estructura o comportamiento de tal componente.

La descomposición sucesiva de un diseño de entidad en componentes, y empaquetado de tales componentes a otros diseños de entidades que pueden ser descompuestos de manera similar, resulta en una jerarquía de diseño de entidades representando un diseño completo. Tal colección de diseños de entidades es llamada jerarquía de diseño.

La declaración de una entidad define la interfaz entre un dado diseño de entidad y el entrono en el que este es usado. Esta también puede especificar declaraciones y sentencias que son parte de la entidad. Una declaración de entidad puede ser compartida por varios diseños de entidades, cada uno de los cuales teniendo una arquitectura diferente. Por lo tanto una declaración de entidad puede potencialmente representar una clase de diseños de entidades, cada uno con la misma interfaz.

El templete básico para una entidad es mostrado en el Listado 2.1. El cuerpo de la entidad y <nombre> el identificador de la identidad siguiendo las reglas de nomenclatura para VHDL.

```
Encabezado_de_entidad
     [clausula_genérica_formal]
     [clausula_de_puertos_formal]
clausula_generica
     generic( lista_genérica )
clausula_de_puertos
     port( lista_de_puertos )
```

Listado 2.1. Templete básico de entidad.

La lista genérica en la clausula genérica formal define constantes genéricas cuyos valores puedan ser determinados por el entorno. La lista de puertos en la clausula formal de puertos formal define los puertos de entrada/salida del diseño de entidad. En ciertas circunstancias, los nombres de las constantes genéricas y puertos declarados en el encabezado del diseño de entidad se vuelven visibles fuera del diseño de entidad.

II.7.1.1 Genéricos

Los genéricos proveen un canal para que la información estática sea comunicada a un bloque desde su entorno. Lo siguiente aplica a ambos bloques externos definidos por diseños de entidades y a bloques internos definidos por sentencias block.

lista_genérica ::= lista_interfaz_genérica

Los genéricos de un bloque son definidos por una lista interfaz genérica. Cada elemento interfaz en dicha lista interfaz genérica declara un genérico formal. El valor de una constante genérica puede ser especificado por el actual correspondiente en una lista asociación genérica. Si tal actual no es especificado para un dado genérico formal, y si una expresión predeterminada es especificada para ese genérico, el valor para esta expresión es el valor para el genérico. Es un error si no se especifica actual para un dado genérico formal y hay presente expresión predeterminada en el elemento interfaz correspondiente.

Los genéricos pueden ser utilizados para controlar características estructurales, de flujo de datos o de comportamiento de un bloque, o pueden ser usados simplemente para documentación. En particular, los genéricos pueden ser usados para especificar el tamaño de los puertos, el número de subcomponentes en un bloque, las características de temporización de un bloque, o incluso las características físicas de un diseño como son temperatura, capacitancia, localización, etc.

II.7.1.2 Puertos

Los puertos proveen canales para comunicación dinámica entre un bloque y su entorno. Lo siguiente aplica a ambos bloques externos definidos por los diseños de entidades y a bloques internos definidos por sentencias block.

lista_de_puertos::=lista_interfaz_de_puertos

Los puertos de un bloque son definidos por una lista interfaz de puertos. Cada elemento interfaz en la lista interfaz de puertos declara un puerto formal. Los puertos de un bloque pueden ser asociados con señales en el entorno en que el bloque es usado, para comunicarse con otros bloques en ese entorno.

Un puerto es por sí mismo una señal, por lo tanto un puerto formal de un bloque puede ser asociado con un puerto de un super bloque que le contiene. El puerto o señal asociado con un dado puerto formal es llamado el actual correspondiendo con el formato de puerto. El actual debe ser denotado por un nombre estático.

Si, cuando una descripción dada es completamente elaborada, un puerto formal es asociado con un actual que es por sí mismo un puerto, entonces las siguientes restricciones aplican dependiendo en el modo del puerto formal:

Para un puerto formal de modo **in**, el actual asociado puede ser solamente un puerto de modo **in**, **inout**, o **buffer**. Para un puerto formal de modo **out**, el actual asociado puede ser solamente un puerto de modo **out** o **inout**. Para un puerto formal de modo **inout**, el actual asociado puede ser solamente un puerto de modo **inout**.

Si un puerto formal es asociado con un puerto actual o señal, entonces el puerto formal se dice conectado. Si un puerto formal es en vez asociado con la palabra reservada **open**, entonces el formal se dice desconectado. Un puerto de modo **in** puede estar desconectado solo si su declaración incluye una expresión predeterminada. Un puerto de cualquier otro modo que **in** puede permanecer desconectado siempre y cuando su tipo no sea un arreglo de tipo no acotado.

II.7.1.3 Entidad: parte declarativa

La parte declarativa de una dada declaración de entidad declara elementos que son comunes a todos los diseños de entidades cuyas interfaces son definidas por la declaración de entidad dada. En el Listado 2.2 se muestra la estructura de la parte declarativa y los posibles elementos que esta puede contener así como el orden de aparición. El listado no precisa los elementos a contener sino el orden de aparición.

```
     entidad_parte_declarativa::=
{elemento_declarativo_de_entidad}

     elemento_declarativo_de_entidad::=
     declaración_de_subprograma
     | cuerpo_de_subprograma
     | declaración_de_tipo
     | declaración_de_subtipo
     | declaración_de_constante
     | declaración_de_señal
     | declaración_de_archivo
     | declaración_de_alias
     | declaración_de_atributo
     | especificación_de_atributo
     | especificación_de_desconexión
     | clausula_use
```

Listado 2.2. Estructura de parte declarativa de entidad.

Los nombres declarados por elementos declarativos en la parte declarativa de la entidad de una dada declaración de entidad son visibles en los cuerpos de diseño de entidades correspondientes, así como con ciertas porciones de una declaración de configuración correspondiente.

II.7.2 Cuerpos de arquitectura

Un cuerpo de arquitectura define el cuerpo de un diseño de entidad. Este especifica las relaciones entre las entradas y salidas de un diseño de entidad, y puede ser expresada en términos de estructura, flujo de datos, o comportamiento. Tales especificaciones pueden ser parciales o completas. En el listado 2.3 se

muestra la estructura de la arquitectura del diseño de una entidad, cada uno de sus subelementos será descrito en las siguientes subsecciones. El listado incluye palabras reservadas resaltadas en negrillas que dan a conocer al revisor de sintaxis que se trata de la descripción de una arquitectura.

```
cuerpo_de_arquitectura::=
architecture identificador of nombre_de_entidad is
     parte_declarativa_de_arquitectura
begin
     arquitectura_apartado_de_sentencias
end[ nombre de arquitectura ];
```

Listado 2.3. Estructura de arquitectura del diseño de una entidad.

El identificador define simplemente el nombre del cuerpo de arquitectura; este simple nombre distingue cuerpos de arquitectura asociados con la misma declaración de entidad. El nombre de entidad identifica el nombre de la declaración de entidad que define la interfaz de este diseño de entidad. Para un dado diseño de entidad, ambos la declaración de entidad y el cuerpo de arquitectura asociado deben residir en la misma librería. Si un sólo nombre aparece al final de un cuerpo de arquitectura, este debe repetir el identificador del cuerpo de arquitectura.

Más de un cuerpo de arquitectura puede existir correspondiendo a una dada declaración de entidad. Cada una declara un diferente cuerpo con la misma interfaz, por tanto cada uno junto con la declaración de entidad representa un diferente diseño de entidad con la misma interfaz. Dos cuerpos de arquitectura que son asociados con diferentes declaraciones de entidad pueden tener el mismo nombre, aun si ambos cuerpos de arquitectura (y las correspondientes declaraciones de entidad) residen en la misma librería.

II.7.2.1 Arquitectura: parte declarativa

La parte declarativa de la arquitectura contiene declaraciones de elementos que están disponibles para su uso con el bloque definido por el diseño de entidad.

El listado 2.4 muestra la estructura de la parte declarativa de la arquitectura, donde el orden es el factor de mayor importancia, en las siguientes subsecciones se profundizara la descripción de cada uno de estos. Por el contrario con respecto del Listado 2.2 en éste se incluye la declaración de componente que tiene particular aplicación en el estructurado del diseño por componentes, así como la configuración de la especificación; sin embargo esta última no será utilizada en este proyecto de investigación.

```
parte_declarativa_de_arquitectura ::=
     {bloque_elemento_declarativo}

bloque_elemento_declarativo ::=
declaración_de_subprograma
| cuerpo_de_subprograma
| declaración_de_tipo
| declaración_de_subtipo
| declaración_de_constante
| declaración_de_señal
| declaración_de_archivo
| declaración_de_alias
| declaración_de_componente
| declaración_de_atributo
| especificación_de_atributo
| configuración_de_especificación
| especificación_de_desconexión
| clausula_use
```

Listado 2.4. Estructura de parte declarativa de arquitectura.

II.7.2.2 Arquitectura: apartado de sentencias

El apartado de sentencias de la arquitectura contiene sentencias que describen la organización interna y/u operación del bloque definido por el diseño de entidad.

```
apartado_de_sentencias_de_la_arquitectura ::=
        {sentencia_concurrente}
```

Todas las sentencias en el apartado de sentencias de la arquitectura son sentencias concurrentes, que se ejecutan asíncronamente con respecto a las otras.

II.7.3 Sentencias secuenciales

Sentencias secuenciales son usadas para definir algoritmos para la ejecución de un subprograma o proceso; estos se ejecutan en el orden en el que aparecen. En el Listado 2.5 se muestra el compendio de sentencias secuenciales, su uso es únicamente posible dentro de procesos o procedimientos, aunque la ejecución de cada proceso es concurrente.

```
secuencia_de_sentencias ::=
     {sentencia_secuencial}

sentencia_secuencial ::=
sentencia_wait
|sentencia_de_aserción
|sentencia_de_asignación_de_señal
|sentencia_de_asignación_de_variable
|sentencia_llamada_a_procedimiento
|sentencia_if
|sentencia_case
|sentencia_loop
|sentencia_next
|sentencia_exit
|sentencia_return
|sentencia_nula
```

Listado 2.5. Compendio de sentencias secuenciales.

Ciertas sentencias secuenciales pueden ser etiquetadas. Tales etiquetas son declaradas implícitamente al principio de la parte declarativa de la más interna sentencia de proceso o cuerpo de subprograma.

II.7.3.1 Sentencia if

Una sentencia **if** selecciona para ejecución una o ninguna de las secuencias de sentencias encerradas, dependiendo del valor de una o más

condiciones. En el listado 2.6 se muestra la estructura de una sentencia **if**, las palabras reservadas de esta estructura son resaltadas en negrilla, la condición puede utilizar operadores lógicos como <>, ==, and, or, xor, etc.

```
sentencia_if ::=
     if condición then
          secuencia_de_sentencias
     { elsif condición then
          secuencia_de_sentencias }
     [ else
          secuencia_de_sentencias ]
     end if;
```

Listado 2.6. Estructura de sentencia if.

Una expresión especificando una condición debe ser de tipo BOOLEAN. Para la ejecución de una sentencia **if**, la condición especificada después del **if**, y cualesquiera condiciones especificadas después del **elsif**, son evaluadas en sucesión (tratando un **else** final como **elsif** TRUE **then**), hasta que una se evalúa TRUE o todas las condiciones son evaluadas y apuntan a FALSE. Si una condición se evalua a TRUE, entonces la correspondiente secuencia de sentencias es ejecutada; de lo contrario ninguna de las secuencias de sentencias es ejecutada.

II.7.3.2 Sentencia case

Una sentencia **case** selecciona para ejecución una de un numero de secuencias de sentencias alternativas; la alternativa seleccionada es definida por el valor de una expresión. En el Listado 2.7 se muestra la estructura del la sentencia **case**, las palabras reservadas de esta estructura son resaltadas en negrilla, las alternativas deben de cubrir por completo los posibles valores del tipo de dato; de no ser así la alternativa **others** deberá ser usada para cubrir las opciones restantes.

```
sentencia_case ::=
     case expresión is
```

```
            alternativa_sentencia_case
            { alternativa_sentencia_case }
      end case;

alternativa_sentencia_case ::=
      when opciones =>
            secuencia_de_sentencias
```

Listado 2.7. Estructura de sentencia case.

La expresión debe ser de tipo discreto, o tipo arreglo de caracteres unidimensional (cuyos valores sean representables como literales cadena o cadena de bits). Éste tipo debe ser determinable independientemente del contexto en el que la expresión ocurre, pero empleando el hecho de que la expresión debe ser tipo discreto o arreglo de caracteres unidimensional. Cada opción en una alternativa sentencia **case** debe ser del mismo tipo que la expresión; la lista de opciones específica para que valores de la expresión la alternativa es seleccionada.

La opción **others** es solo permitida para la última alternativa y como su única opción; esta representa todos los valores (posiblemente ninguno) no dados en las opciones de las alternativas anteriores. La ejecución de una sentencia **case** consiste en la evaluación de la expresión seguida por la ejecución de la secuencia de sentencias seleccionada.

La ejecución de una sentencia **case** selecciona una y solo una alternativa, debido a que las opciones son exhaustivas y mutuamente exclusivas. La cualificación de la expresión de una sentencia **case** por un subtipo estático localmente puede ser usado a menudo para limitar el número de opciones que se necesita proporcionar explícitamente.

II.7.4 Sentencias concurrentes

Sentencias concurrentes son usadas para definir bloques interconectados y procesos que conjuntamente describen el comportamiento completo o estructura

de un diseño. Las sentencias concurrentes se ejecutan asíncronamente con respecto a las demás. El compendio de sentencias concurrentes se muestra en el Listado 2.8, principalmente los utilizados en el presente trabajo de investigación son las sentencias proceso y de instanciación de componente.

```
sentencia_concurrente ::=
     sentencia_block
     | sentencia_proceso
     | llamada_a_procedimiento_concurrente
     | sentencia_de_aserción_concurrente
     | sentencia_de_asignación_de_señal_concurrente
     | sentencia_de_instanciación_de_componente
     | sentencia_generate
```

Listado 2.8. Elementos pertenecientes a la clase de sentencias concurrentes.

Las sentencias concurrentes primarias son las sentencias block, que agrupa otras sentencias concurrentes, y la sentencia proceso, que representa un solo e independiente proceso secuencial. Sentencias concurrentes adicionales proveen sintaxis conveniente para representar simples, formas de procesos comúnmente ocurrentes, así como para representar descomposición estructural y descripciones regulares.

En un dado ciclo de simulación, una implementación puede ejecutar sentencias concurrentes en paralelo o en algún orden. El lenguaje no define el orden, de cualquiera, en que tales sentencias serán ejecutadas. Una descripción que depende en algún orden de ejecución particular de sentencias concurrentes es errónea.

Todas las sentencias concurrentes pueden ser etiquetadas. Tales etiquetas implícitamente declaradas al principio de la parte declarativa mas interna de la declaración de entidad, cuerpo de arquitectura o sentencia block.

II.7.4.1 Sentencia proceso

Una sentencia proceso define un proceso secuencial independiente representando el comportamiento de alguna parte del diseño. En el Listado 2.9 se muestra la estructura de la sentencia proceso.

```
sentencia_proceso ::=
     [ etiqueta_del_proceso : ]
          process[(lista_de_sensibilidad)]
               parte_declarativa_del_proceso
          begin
               apartado_de_sentencias_del_proceso
          end process [ etiqueta_del_proceso ];

parte_declarativa_del_proceso ::=
     { elemento_declarativo_del_proceso }

elemento_declarativo_del_proceso ::=

     declaración_de_subprograma
     | cuerpo_de_subprograma
     | declaración_de_tipo
     | declaración_de_subtipo
     | declaración_de_constante
     | declaración_de_variable
     | declaración_de_archivo
     | declaración_de_alias
     | declaración_de_atributo
     | especificación_de_atributo
     | clausula use

apartado_de_sentencias_del_proceso ::=
     { sentencia_secuencial }
```

Listado 2.9. Estructura de sentencia proceso.

El apartado de sentencias es ejecutado de manera secuencial mientras los procesos entre si son ejecutados concurrentemente. La lista de sensibilidad indica si se deberá esperar algún cambio en las señales listadas o ejecutar el proceso indefinidamente de manera repetida.

Si una lista de sensibilidad aparece después de la palabra reservada **process**, entonces se asume que la sentencia proceso contiene una sentencia de espera implícita como última sentencia del apartado de sentencias del proceso; esta sentencia de espera implícita es de la forma:

```
wait on lista_de_sensibilidad;
```

Donde la lista de sensibilidad de la sentencia de espera es aquello después de la palabra reservada **process**. Tal sentencia proceso puede no contener una sentencia de espera explicita. Similarmente, si tal sentencia proceso es pariente de un procedimiento, entonces ese procedimiento puede no contener una sentencia **wait**.

Solo nombres de señales estáticos pueden aparecer en la lista de sensibilidad de una sentencia proceso. Si una etiqueta aparece al final de una sentencia proceso, ésta debe repetir la etiqueta del proceso. La ejecución de una sentencia proceso consiste de la repetida ejecución de su secuencia de sentencias. Después de que la última sentencia en la secuencia de sentencias de una sentencia proceso es ejecutada, la ejecución continuara inmediatamente con la primera sentencia en la secuencia de sentencias.

Una sentencia proceso se dice ser proceso pasivo si el proceso por sí mismo, o cualquier procedimiento del cual el proceso es pariente, contienen una sentencia de asignación a señal. Tal proceso, o cualquier sentencia concurrente equivalente a tal proceso, pueden aparecer en el apartado de sentencias de la entidad de la declaración de entidad.

Las reglas de arriba implican que un proceso que tiene una lista de sensibilidad explicita siempre tiene exactamente una (implícita) sentencia de espera en él, y esa sentencia de espera aparece al final de la secuencia de sentencias en el apartado de sentencias del proceso. Por lo tanto un proceso con una lista de sensibilidad siempre espera al final de su apartado de sentencias;

cualquier evento en una señal nombrada en la lista de sensibilidad causara a tal proceso su ejecución desde el principio de su apartado de sentencias hacia abajo hasta el final, donde volverá a esperar nuevamente. Tal proceso se ejecuta a través una vez al principio de la simulación, suspendiendo por primera vez cuando este ejecuta la sentencia de espera implícita.

II.8.4.2 Sentencia de instanciación de componente

Una sentencia de instanciación de componente define un subcomponente del diseño de entidad en el cual este aparece y asocia señales con los puertos de ese subcomponente. Este subcomponente es una instancia de una clase de componentes definidos por una declaración de componente correspondiente. En el Listado 2.10 se muestra la estructura de la sentencia de instanciación de componente. Gracias al uso de componentes se reutilizan segmentos de la descripción del diseño electrónico.

```
sentencia_de_instanciación_de_componente ::=
     etiqueta_de_instanciacion :
          nombre_del_componente
               [ aspecto_de_mapeo_de_genérico ]
               [ aspecto_de_mapeo_de_puerto ];
```

Listado 2.10. Estructura de instanciación de componentes.

El nombre del componente debe de ser el nombre de un componente declarado en una declaración de componente. El aspecto de mapeo de genérico, si está presente, asocia un solo actual con cada genérico local (o subelemento de) en la correspondiente declaración de componente.

Cada genérico local (o subelemento de) debe ser asociado exactamente una vez. Similarmente el aspecto de mapeo de puerto, si está presente, asocia un solo actual con cada puerto local (o subelemento de) en la correspondiente declaración de componente. Cada puerto local (o subelemento de) debe ser asociado exactamente una vez.

La sentencia de instanciación de componente puede ser usada para implicar una organización estructural para un diseño de hardware. Al usar declaraciones de componentes, señales, y sentencias de instanciación de componentes, un dado bloque (interno o externo) puede ser descrito en términos de subcomponentes que están interconectados por señales.

La instanciación de componentes provee una forma de estructurar la descomposición lógica de un diseño. Las características estructurales o de comportamiento precisas de un dado subcomponente pueden ser descritas después. La instanciación de componentes provee también un mecanismo para reutilizar diseños existentes en una librería de diseños.

II.7.4.2.1 Instanciación de un componente

Una sentencia de instanciación de componente y una especificación de configuración correspondiente, tomadas juntas, implica que la jerarquía del bloque con el diseño de entidad conteniendo la instanciación del componente será extendido con una copia única del bloque definido por otro diseño de entidad. Los aspectos del mapeo de genéricos y de puertos en la sentencia de instanciación de componente y en la indicación de empaquetado de la especificación de configuración identifican las conexiones que serán hechas a fin de cumplir la extensión.

II.7.5 Elementos léxicos

El texto de una descripción consiste de uno o más archivos de diseño. El texto de un archivo de diseño es una secuencia de elementos léxicos, cada uno compuesto de caracteres. Debido a la extensión y reducida importancia el resto de esta subsección se incluye como el Apéndice 2.

III. METODOLOGIA

El trabajo de investigación fue desarrollado en dos distintos lugares de trabajo, intercalando el uso de las instalaciones de acuerdo a la carga de materias del semestre en curso, uno es el cubículo del área acuícola a un costado de la entrada peatonal de la Universidad Autónoma de Querétaro la cual se ubica en la esquina de la avenida Hidalgo y la avenida 5 de Febrero del Municipio de Querétaro, Querétaro, México; la otra área designada para el desarrollo del proyecto y utilizada principalmente en días sin carga de materias es el Laboratorio de Instrumentacion y Control ubicado enfrente de la comunidad de Amazcala en el municipio El Márquez, Querétaro, México.

El presente capitulo será dividido en dos parte una de ellas siendo la sección de materiales y la otra métodos; en el orden mencionado. En la sección de materiales será incluido a parte del hardware, el software utilizado en tareas como descripción, simulación y sintetizado del diseño electrónico; desarrollo de software interfaz para controlador, el software de sintetizado es proporcionado por el fabricante del FPGA. En la metodología se describirán los procesos llevados a cabo en el diseño electrónico y se referenciará alguna parte acerca del desarrollo de la interfaz usuario controlador en trabajos anteriores similares a esta etapa.

III.1 Materiales

III.1.1 Tarjeta de desarrollo FPGA

La tarjeta de desarrollo FPGA utilizada es el modelo DE2-70 del fabricante Terasic, esta tarjeta de desarrollo es un ensamble de circuito impreso (PCA) el cual contiene diversos componentes electrónicos, no todos ellos fueron usados por lo que a continuación se listarán y describirán las características de los usados.

En la Figura 3.1 se muestra la fotografía del kit, el kit está particularmente orientado a aplicaciones multimedia en las que se requiere suficientes recursos de memoria así como capacidad en el FPGA; debido a la capacidad de este kit es posible probar diversas técnicas de diseño y encontrar las que mejor satisfagan las restricciones del diseño.

- FPGA Cyclone II marca Altera EP2C70 68,416 LE.
- USB Blaster.
- Memoria estática síncrona de acceso aleatorio de 2 MB.
- Memoria flash de 8 MB.
- Pantalla de cristal liquido 2X16.
- 18 interruptores de palanca.
- 27 diodos emisores de luz.
- Cristal de 50 MHz.
- Transceptor RS-232 y conector DB9.
- Cabezal de expansión de 40-pines.

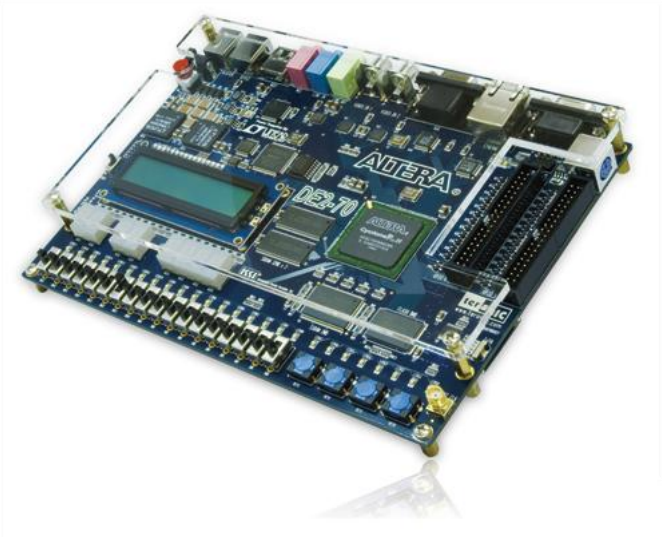

Figura 3.1. Kit de desarrollo Altera DE2-70.

III.1.1.1 FPGA

Numero de parte: EP2C70F896C6N.

Fabricante: Altera.

Caracteristicas: Arquitectura de alta densidad con 68,416 LE; Hasta 1.1 MBits de RAM disponible sin reducir lógica disponible; 150 multiplicadores embebidos de 18x18 bits cada uno de ellos configurable como 2 multiplicadores de 9 bits independientes, con operación de hasta 250 MHz; Estandar Multivolt^{tm} I/O soporte para interfaces 1.5, 1.8, 2.5, y 3.3 V; Soporte de propiedad intelectual: Altera Megafunction, Altera Megacore® y Altera Megafunctions Partners Program (AMPP^{sm}).

III.1.1.2 USB Blaster

El USB Blaster es un subsistema embebido en el kit, diseñado por el fabricante del kit de desarrollo. Este sirve para configurar el FPGA, a pesar de que la mayoría de los kit de desarrollo incluyen un sistema equivalente integrado, es importante denotar que tipo de interfaz ha sido utilizada para programar el FPGA.

III.1.1.3 Memoria estática síncrona de acceso aleatorio

La memoria modelo IS61LPS51236A del fabricante ISSI es de alta velocidad, bajo consumo, RAM estática síncrona diseñada para proveer uso continuo, de alto desempeño para aplicaciones en comunicaciones y redes. Este modelo está organizado en 524, 288 palabras de 36 bits. El dispositivo integra un contador de 2 bits para uso continuo, núcleo SRAM de alta velocidad, y alta capacidad de manejo de salidas. Todas las entradas síncronas pasan por registros controlados por el flanco positivo de una sola señal de reloj. Los ciclos de escritura son temporizados internamente y son iniciados por el flanco de subida de la entrada de reloj.

Los ciclos de escritura pueden ser de uno a cuatro bytes controlado por las entradas de control de escritura. Separados habilitadores de byte permiten la escritura de bytes individuales. La operación de escritura de byte es desempeñada

al usar la entrada del habilitador de escritura de byte (BWE) combinado con uno o más señales individuales de escritura de byte (BWx). Adicionalmente, la escritura global (GW) está disponible para escribir todos los bytes a un tiempo, no importando los controles de escritura de byte.

Los diagramas de tiempos son las características que deben de tener las señales de control, datos y direcciones, así como los tiempos que cada una de ellas debe de cumplir. El diagrama mostrado en la Figura 3.2 es de gran importancia debido a que en base a este se desarrolla el controlador para el chip de memoria, los tiempos correspondientes a las literales son mostrados en la Tabla 3.1.

El diagrama mostrado en la Figura 3.3 es el correspondiente al ciclo de escritura, los tiempos correspondientes a las literales son mostrados en la Tabla 3.1; mientras en la Tabla 3.2 se muestra a manera de resumen los modos de operación lectura y escritura con sus correspondientes señales de habilitación para escritura de bytes independientes. Para la operación de memoria estática resumida en la Tabla 3.2, el símbolo # indica que la señal es testada, H = 1 lógico, L = 0 lógico, X no importa. Los números de los bytes empiezan en 1.

La Tabla 3.1 indica el tiempo que representa cada símbolo. En el caso de que el máximo o mínimo sea representado con un guion medio “-” indica que este valor no tiene restricción para el máximo o mínimo.

III.1.1.3.1 Diagrama de tiempos: lectura

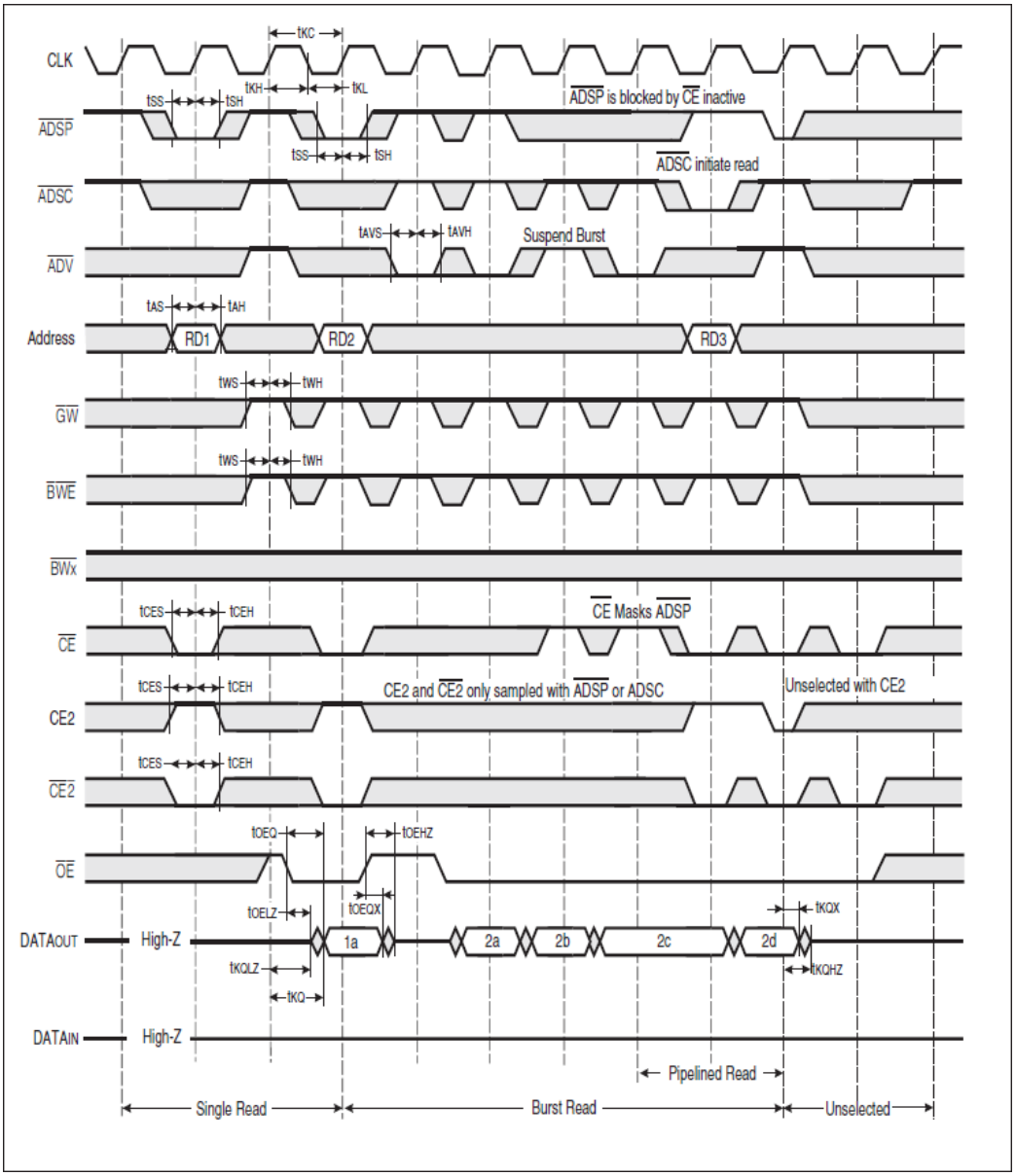

Figura 3.2. Diagrama de tiempos para ciclo de lectura sencillo, ciclo de lectura continuo y deselección.

III.1.1.3.2 Diagrama de tiempos: escritura

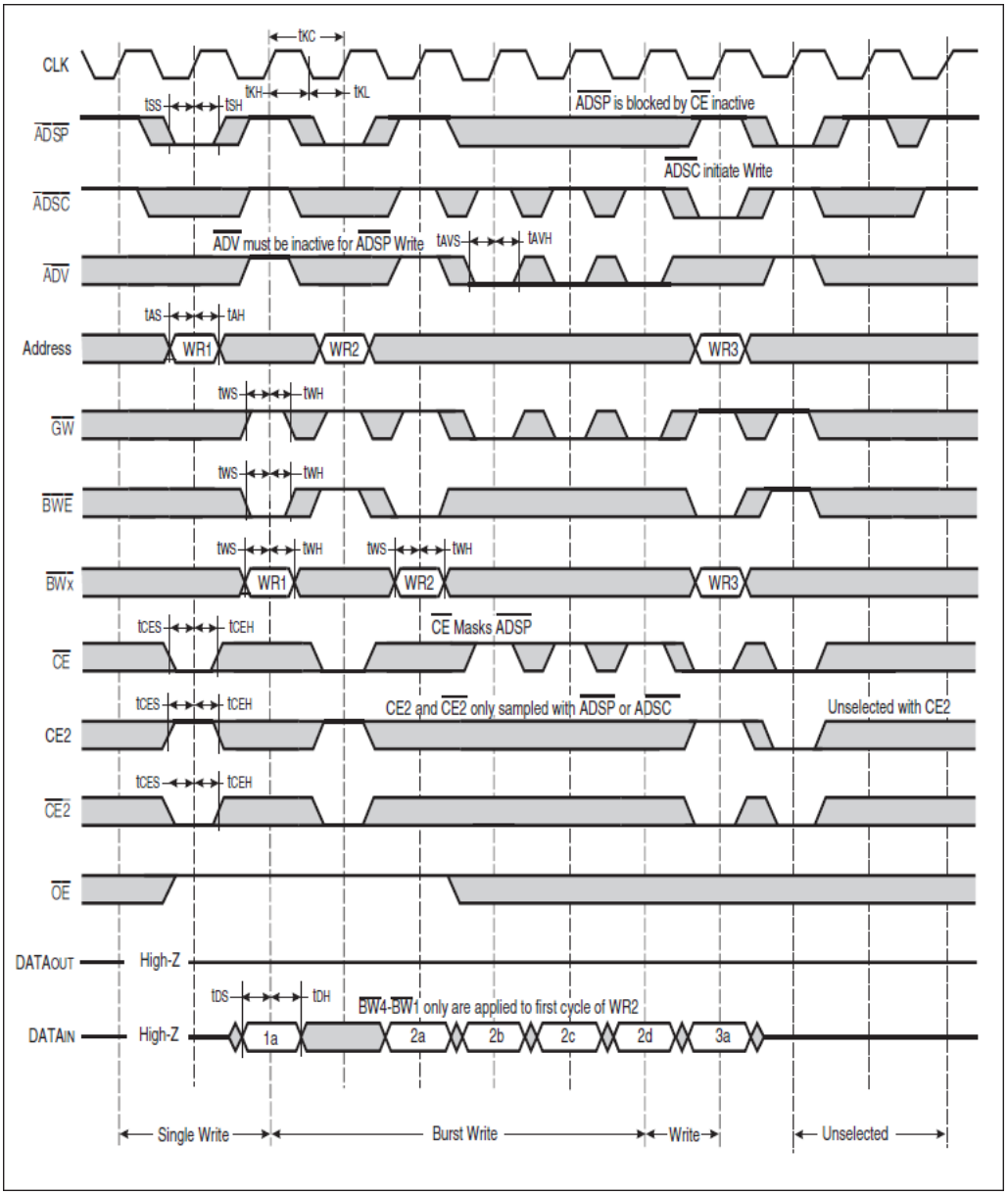

Figura 3.3. Diagrama de tiempos para ciclo de escritura sencillo, ciclo de escritura continuo y deselección.

Símbolo	Parámetro	Min	Max	Unidad
f_{MAX}	Frecuencia de reloj	-	200	MHz
t_{KC}	Tiempo de ciclo	5	-	ns
t_{KH}	Tiempo de reloj en alto	2	-	ns

t_{KL}	Tiempo de reloj en bajo	2	-	ns
t_{KQ}	Tiempo de acceso a reloj	-	3.1	ns
t_{KQX}	Reloj en alto a salida invalida	1.5	-	ns
t_{KQLZ}	Reloj en alto a salida Baja-Z	1	-	ns
t_{KQHZ}	Reloj en alto a salida Alta-Z	-	3.0	ns
t_{OEQ}	Habilitación de salida a salida valida	-	3.1	ns
t_{OELZ}	Habilitación de salida a salida Baja-Z	0	-	ns
t_{OEHZ}	Habilitación de salida a salida Alta-Z	-	3.0	ns
t_{AS}	Tiempo de establecimiento de dirección	1.4	-	ns
t_{WS}	Tiempo de establecimiento de escritura/lectura	1.4	-	ns
t_{CES}	Tiempo de establecimiento de habilitación de chip	1.4	-	ns
t_{AVS}	Tiempo de establecimiento de avance de dirección	1.4	-	ns
t_{DS}	Tiempo de establecimiento de datos	1.4	-	ns
t_{AH}	Tiempo de retención de dirección	0.4	-	ns
t_{WH}	Tiempo de retención de escritura	0.4	-	ns
t_{CEH}	Tiempo de retención de habilitación de chip	0.4	-	ns
t_{AVH}	Tiempo de retención de avance de dirección	0.4	-	ns
t_{DH}	Tiempo de retención de datos	0.4	-	ns
t_{PDS}	ZZ alto a apagado	-	2	ciclos
t_{PUS}	ZZ bajo a apagado	-	2	ciclos

Tabla 3.1. Listado de parámetros correspondiente a Figuras 3.2 y 3.3.

Función	**GW#**	**BWE#**	**BWa#**	**BWb#**	**BWc#**	**BWd#**
Leer	H	H	X	X	X	X
Leer	H	L	H	H	H	H
Escribir byte 1	H	L	L	H	H	H
Escribir bytes 2 y 4	H	L	H	L	H	L
Escribir todos los bytes	H	L	L	L	L	L
Escribir todos los bytes	L	X	X	X	X	X

Tabla 3.2. Tabla de verdad parcial.

III.1.1.4 Manejador/Receptor de línea RS232

El transceptor ADM3202 es de alta velocidad, 2 canales para dispositivos de interfaz RS-232/V.28 que operan desde una fuente sencilla de 3.3 V. Baja

potencia de consumo y opción de apagado los hace ideales para instrumentos portátiles alimentados por baterías. El dispositivo conforma las especificaciones EIA-232 y CCITT V.28 y opera a velocidades de hasta 460 kbps.

Cuatro capacitores externos son utilizados para el duplicador/inversor de voltaje, permitiendo la operación desde una fuente sencilla de 3.3 V. Sus principales características son:

Máxima velocidad 460 kbps.

3.3 V especificados

Cumple especificaciones EIA-232E

III.1.1.5 Memoria flash

La familia de dispositivos S29GL-A del fabricante Spansion son memorias Flash, alimentados por una única fuente de 3.0V, manufacturados usando tecnología MirrorBit de 200 nm. El S29GL064A es un dispositivo de 64 Mb organizado como 4, 194,304 palabras o 8, 388,608 bytes. Este dispositivo incorpora un bus de datos de 16 bits que puede ser utilizado como un bus de 8 bits al usar la entrada BYTE#; este dispositivo puede ser programado en el sistema anfitrión o bien con programadores estándar de EPROM.

El mínimo tiempo de acceso es de 90 ns. El dispositivo requiere solamente una fuente de alimentación de 3 V para ambas funciones leer y escribir. Los comandos son escritos al dispositivo usando tiempos de escritura de microprocesador estándar. También los ciclos de escritura internamente mantienen direcciones y datos necesarios para operaciones de programación y borrado.

La arquitectura de borrado por sector permite borrar sectores de memoria y ser reprogramados sin afectar los datos contenidos en otros sectores. El programado y borrado del dispositivo es iniciado a través de secuencias de

comandos. En la Figura 3.4 se muestra el símbolo de la memoria Flash representando sus señales de control, puerto de direcciones y de datos; este último bidireccional. El símbolo # indica que la señal esta testada, trabaja con lógica negada; las flechas indican el flujo de las señales entrada, salida o ambas. Por medio del puerto de direcciones se proporciona también la dirección de los sectores.

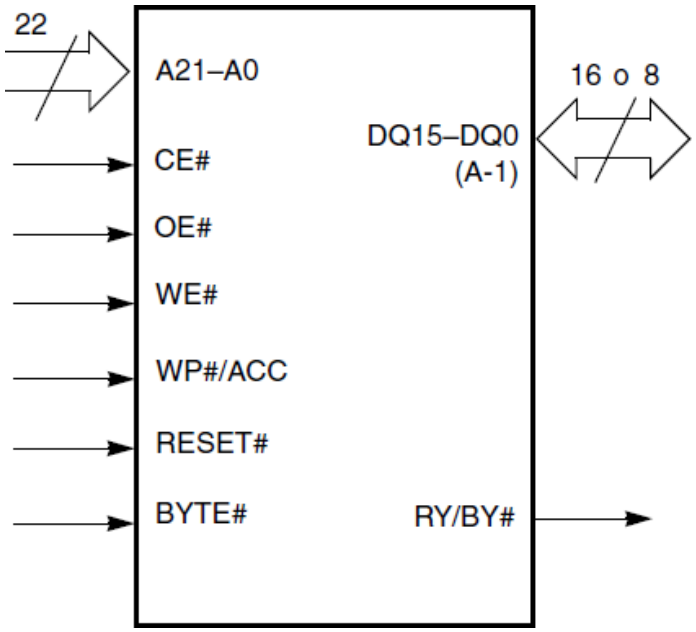

Figura 3.4. Símbolo de memoria Flash.

III.1.1.5.1 Operaciones de los buses

Las operaciones de los buses son iniciadas a través del registro de comando interno. El registro de comando no ocupa ninguna localidad de memoria. El registro almacena los comandos, junto con la dirección y datos necesarios para ejecutar el comando. Los contenidos del registro sirven como entrada a la máquina de estados interna. Las salidas de la máquina de estados indican la función del dispositivo. La Tabla 3.3 muestra las operaciones del bus, las entradas y niveles de control requeridos, y la salida resultante.

Para la Tabla 3.3, la leyenda es: L = Lógico bajo = V_{IL}, H = Lógico alto = V_{IH}, X = No importa, A_{IN} = Dirección de entrada, D_{OUT} = Salida de datos. Notas: 1) Las direcciones son Amax:A0 en modo palabra, Amax:A-1 en modo byte. Las direcciones de los sectores son Amax:A15 en ambos modos. 2) Si WP# = V_{IL}, los dos sectores de boteo mas externos son protegidos; Si WP# = V_{IH} los dos sectores de boteo mas externos son desprotegidos o protegidos. 3) D_{IN} o D_{OUT} como sean requeridos por la secuencia del comando, estado de datos, o algoritmo de protección de sectores

Operación	CE#	OE#	WE#	RESET#	WP#	ACC	Direcciones (Nota1)	DQ0-DQ7	DQ8-DQ15
Lectura	L	L	H	H	X	X	A_{IN}	D_{OUT}	D_{OUT}
Escritura(Programar/Borrar)	L	H	L	H	(Nota 2)	X	A_{IN}	(Nota 3)	(Nota 3)
Deshabilitación de salida	L	H	H	H	X	X	X	Alta-Z	Alta-Z
Reset	X	X	X	L	X	X	X	Alta-Z	Alta-Z

Tabla 3.3. Tabla de operaciones resumida, solo operaciones utilizadas.

III.1.1.5.1.1 Configuración palabra/byte

El Pin BYTE# controla en si los pines de datos E/S operan en la configuración byte o palabra. Si el pin BYTE# es establecido a 1 lógico, el dispositivo esta en configuración palabra, DQ0-DQ15 permanecen activos y son controlados por CE# y OE#.

Si el pin BYTE# es puesto a 0 lógico, el dispositivo esta en configuración byte, y solo los pines de datos DQ0-DQ7 están activos y son controlados por CE# y OE#. Los pines de datos DQ8-DQ14 son tri-estado, y el pin DQ15 es usado como entrada para la función de dirección LSB (A-1).

III.1.1.5.1.2 Escritura de comandos/Secuencias de comando

Para escribir un comando o secuencia de comando (el cual incluye programar datos en el dispositivo y borrar sectores de memoria), el sistema debe manejar WE# y CE# a V_{IL}, y OE# a V_{IH}.

Una operación de borrado puede borrar un sector, múltiples sectores o el dispositivo completo.

III.1.1.5.1.3 Modo suspensión automático

El modo suspensión minimiza el consumo de energía del dispositivo Flash. El dispositivo automáticamente activa este modo cuando las direcciones permanecen estables por t_{ACC} + 30 ns. El modo suspensión automático es independiente de las señales de control CE#, WE# y OE#. Tiempos estándar de acceso a direcciones proveen nuevos datos cuando las direcciones son cambiadas. En el modo suspensión, los datos de salida son conservados y siempre disponibles para el sistema.

III.1.1.5.1.4 RESET#: Pin de restablecimiento por hardware

El pin RESET# provee un método vía hardware para restablecer el dispositivo a lectura del arreglo de datos. Cuando el pin RESET# es manejado a bajo por al menos un periodo de t_{RP}, inmediatamente el dispositivo termina cualquier operación en progreso, configura todos los pines de salida a tri-estado e ignora todos los comandos de lectura/escritura por la duración del pulso en RESET#. El dispositivo también restablece la máquina de estados interna a lectura del arreglo de datos. La operación que fue interrumpida debe ser reinicializada una vez que el dispositivo está listo para aceptar otra secuencia de comandos, para asegurar la integridad de los datos.

En la Figura 3.5 se muestra el diagrama de tiempos para el restablecimiento vía hardware, mientras en la Tabla 3.4 se listan los valores de los parámetros; nótese que el parámetro t_{READY} tiene distintos valores para cada rutina de restablecimiento. El restablecimiento vía hardware responde de manera distinta en caso de que la máquina de estados interna del dispositivo Flash se encuentre procesando algún algoritmo embebido o no.

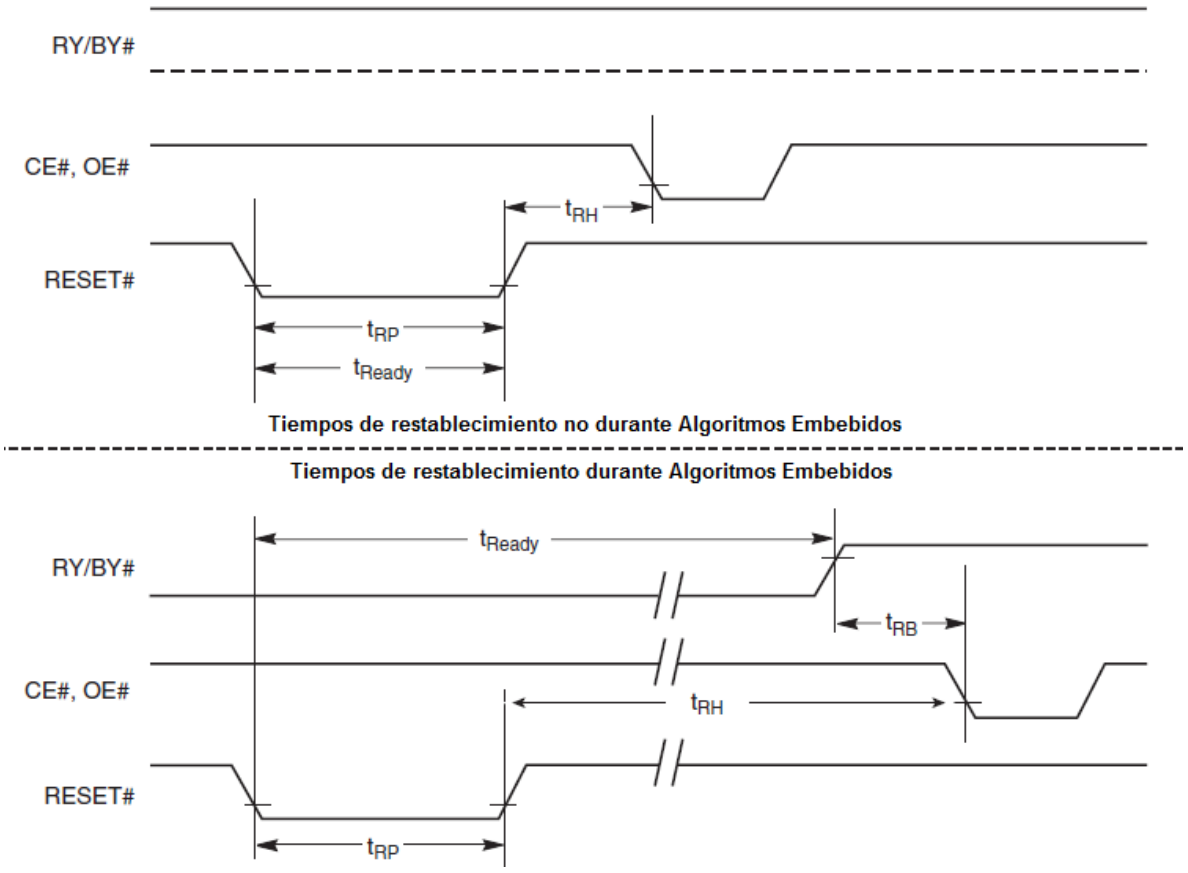

Figura 3.5. Diagrama de tiempos para restablecimiento vía hardware.

Parámetro	Descripción		Duración	Unidad
t_{READY}	Pin RESET# bajo (durante algoritmos embebidos) a modo lectura	Max	20	µs
t_{READY}	Pin RESET# bajo (no durante algoritmos embebidos) a modo lectura	Max	500	ns
t_{RP}	Ancho de pulso RESET#	Min	500	ns
t_{RH}	Tiempo en alto de restablecer antes de leer	Min	50	ns

t_{RPD}	Entrada RESET# en bajo a modo espera	Min	20	µs
t_{RB}	Salida RY/BY# en alto al pin CE#, OE# en bajo	Min	0	ns

Tabla 3.4. Lista de valores de parámetros correspondientes a Figura 3.5.

III.1.1.5.2 Definiciones de comandos

Al escribir direcciones y datos de comandos o secuencias específicas en el registro de comando inicializa operaciones del dispositivo. En la Tabla 3.5 se definen las secuencias para el registro de comandos validas. El escribir direcciones y valores de datos incorrectos o escribirlos en la secuencia inapropiada puede poner al dispositivo en un estado desconocido. Un comando de restablecimiento es entonces requerido para regresar al dispositivo a lectura del arreglo de datos.

Todas las direcciones son conservadas en el flanco de bajada de WE# o CE#, cualquiera que suceda al último. Todos los datos son captados en el flanco de subida de WE# o CE#, cualquiera que suceda primero.

Secuencia de comandos (Nota 1)	Ciclos	Ciclos del bus (Notas 2 a la 4)											
		Primero		Segundo		Tercero		Cuarto		Quinto		Sexto	
Lectura (Nota 4)	1	RA	RD										
Restablecimiento	1	XXX	F0										
Programar	4	555	AA	2AA	55	555	A0	PA	PD				
Borrar chip	6	555	AA	2AA	55	555	80	555	AA	2AA	55	555	10
Borrar sector	6	555	AA	2AA	55	555	80	555	AA	2AA	55	SA	30

Tabla 3.5. Definiciones de comandos, modo x16 con BYTE# = VIH.

Leyenda

X = No importa
RA = Dirección de lectura de localidad de memoria a leer
RD = Dato leído de la localidad RA durante la operación de lectura
PA = Dirección de programación. Las direcciones son captadas en el flanco de bajada del pulso WE# o CE#, cualquiera que suceda al último.
PD = Datos de programación para localidad PA. Los datos son captados en el flanco de subida del pulso WE# o CE#, cualquiera que suceda primero.
SA = Dirección de sector a ser borrado. Los bits de dirección A21-A15 seleccionan de manera única cualquier sector.

Notas

1) Refiérase a la Tabla 3.3 para descripción de las operaciones del bus.
2) Todos los valores están en notación hexadecimal.
3) Durante ciclos de comandos o desbloqueo, cuando los bits más bajos de la dirección son 555 o 2AA como es mostrado en la Tabla 3.5, los bits de la dirección arriba del A11 y bits de datos arriba de DQ7 no tienen importancia.
4) No se requieren ciclos de desbloqueo o comando cuando el dispositivo es en modo lectura.

III.1.1.5.2.1 Leyendo arreglo de datos

El dispositivo es automáticamente establecido a lectura del arreglo de datos después del encendido del dispositivo. No se requieren comandos para obtener datos. El dispositivo está listo para leer el arreglo de datos después de completar un algoritmo embebido de programación o borrado.

En la Figura 3.6 se muestra el diagrama de tiempos correspondiente al ciclo de lectura, mientras en la Tabla 3.6 se listan los valores correspondientes a los parámetros mostrados en la Figura 3.6. A pesar de que el chip de memoria proporciona diversos modos de lectura el mostrado en la Figura 3.6 es el más adecuado a la aplicación actual.

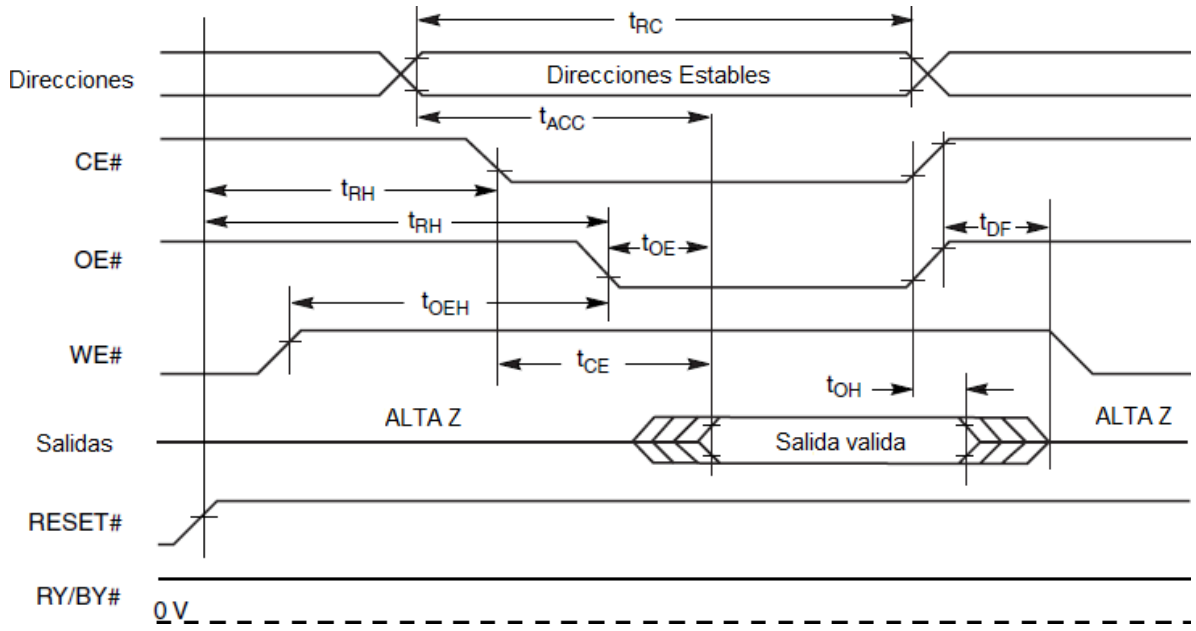

Figura 3.6. Diagrama de tiempos para ciclo de lectura sencillo.

Parámetro	Descripción		Condición de prueba		Velocidad	Unidades
t_{RC}	Tiempo de ciclo de lectura			Min	90	ns
t_{ACC}	Retraso dirección a salida		CE#, OE# = V_{IL}	Max	90	ns
t_{CE}	Retraso habilitación de chip a salida		OE# = V_{IL}	Max	90	ns
t_{PACC}	Tiempo de acceso a pagina			Max	25	ns
t_{OE}	Retraso habilitación de salida a salida			Max	25	ns
t_{DF}	Habilitación de chip a salida ALTA Z			Max	16	ns
t_{DF}	Habilitación de salida a salida ALTA Z			Max	16	ns
t_{OH}	Tiempo de retención de salida de direcciones a CE# u OE#, cualquiera que ocurra primero			Min	0	ns
t_{OEH}	Tiempo de retención de habilitación de salida	Lectura		Min	0	ns
		Conmutación y estado de Data#		Min	10	ns

Tabla 3.6. Tiempos límite comando de lectura.

III.1.1.5.2.2 Comando restablecer

El escribir el comando restablecer restablece el dispositivo al modo lectura o suspensión de borrado-leer. Los bits de dirección no importan para este comando.

El comando restablecer puede ser escrito entre los ciclos de secuencia en la secuencia de un comando de borrado antes de que el borrado empiece. Este restablece el dispositivo al modo lectura. Una vez que el borrado empieza, el dispositivo ignora comandos restablecer hasta que la operación es completada.

El comando restablecer puede ser escrito entre los ciclos de secuencia en la secuencia de un comando de programación antes de que la programación empiece. Este restablece el dispositivo al modo lectura. Una vez que la programación empieza, el dispositivo ignora comandos restablecer hasta que la operación es completada.

III.1.1.5.2.3 Secuencia de comando programación de palabra

La programación es una operación de cuatro ciclos de bus. La secuencia de comando programación es inicializada al escribir dos ciclos de desbloqueo, seguidos por el comando de establecimiento de programa. La dirección y datos a programar son escritos después, los que a su vez inician el algoritmo embebido de de programación. No se requiere que el sistema provea señales de control o tiempos. El dispositivo automáticamente provee pulsos de programación internamente generados y verifica el margen de la celda programada. En la Tabla 3.5 se muestran los requerimientos de direcciones y datos para la secuencia de comando programación.

Cuando el algoritmo de programación embebido es completado, el dispositivo regresa al modo lectura y las direcciones ya no son conservas.

Cualesquiera comandos escritos al dispositivo durante el algoritmo de programación embebido son ignorados. Nótese que un restablecimiento vía hardware inmediatamente termina la operación de programado. La secuencia de comando programación debe ser reinicializada una vez que el dispositivo regresa al modo lectura, para asegurar la integridad de los datos.

La programación se permite en cualquier secuencia de direcciones de localidades y a través de las fronteras de los sectores. La programación de palabras es soportada para compatibilidad con controladores Flash existentes en software y para escritura ocasional de palabras individuales. Cualquier bit en una palabra no puede ser programado de 0 a 1. Al intentar hacer eso puede causar al dispositivo indicar que la operación fue realizada. De cualquier manera, una lectura sucesiva muestra que el dato es aun 0. Solo operaciones de borrado pueden convertir un 0 a 1.

En la Figura 3.7 se muestra el diagrama de tiempos para la operación de programación, mientras en la Tabla 3.7 se muestra el listado de los parámetros y sus respectivos valores límite. A pesar de que el chip de memoria proporciona modos de programación más eficientes las limitaciones en el ancho de banda de la comunicación postulan a este como el más adecuado a la aplicación actual. Los valores de la Tabla 3.7 corresponden a los parámetros mostrados en las Figuras 3.7 y 3.8

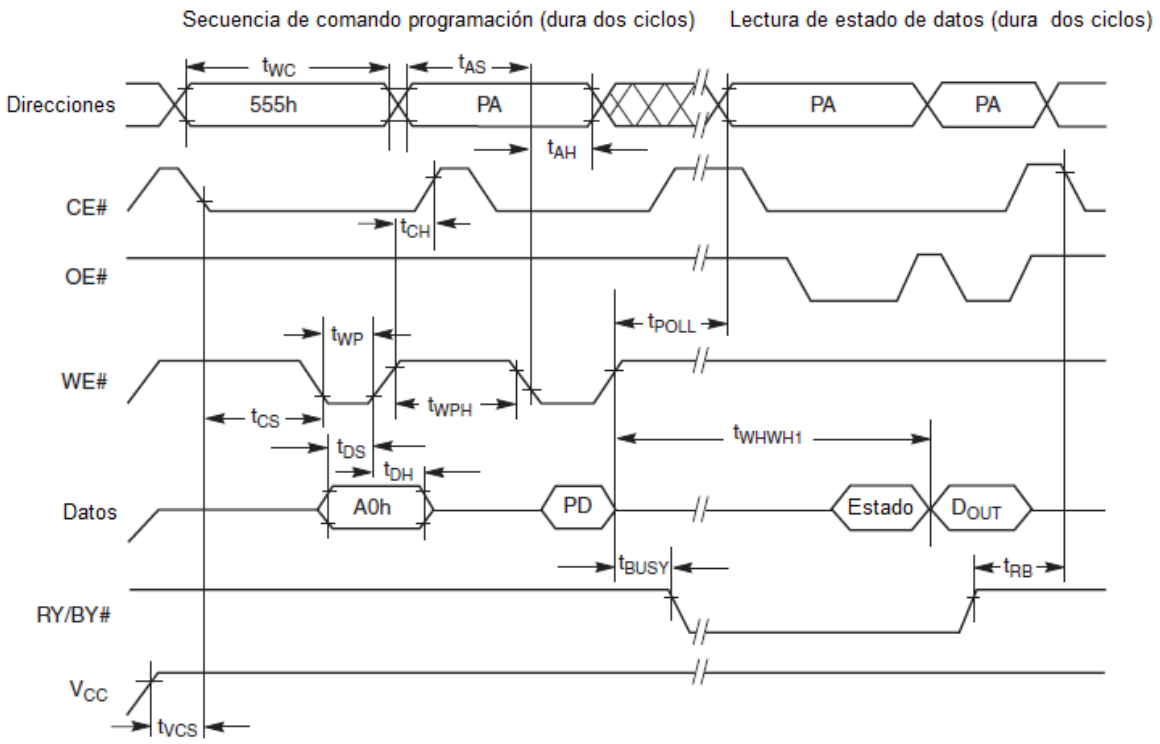

Figura 3.7. Diagrama de tiempos para ciclo de programación.

Parámetro	Descripción		Velocidad	Unidad
t_{WC}	Tiempo ciclo de escritura	Min	90	ns
t_{AS}	Tiempo de establecimiento de dirección	Min	0	ns
t_{ASO}	Tiempo de establecimiento de dirección para OE# bajo durante cambio de bit de estado	Min	15	ns
t_{AH}	Tiempo de retención de dirección	Min	45	ns
t_{AHT}	Tiempo de retención de dirección de CE# u OE# alto durante cambio de bit de estado	Min	0	ns
t_{DS}	Tiempo de establecimiento de datos	Min	35	ns
t_{DH}	Tiempo de retención de datos	Min	0	ns
t_{CEPH}	CE# alto durante cambio de bit de estado	Min	20	ns
t_{OEPH}	OE# alto durante cambio de bit de estado	Min	20	ns
t_{GHWL}	Tiempo de recuperación de lectura antes de escritura (OE# alto a WE# bajo)	Min	0	ns
t_{CS}	Tiempo de establecimiento CE#	Min	0	ns
t_{CH}	Tiempo de retención CE#	Min	0	ns

t_{WP}	Ancho de pulso de escritura	Min	35	ns
t_{WPH}	Ancho de pulso de escritura en alto	Min	30	ns
t_{WHWH1}	Operación de programación de palabra	Típico	60	µs
t_{WHWH2}	Operación de borrado de sector	Típico	0.5	s
t_{VHH}	Tiempo de subida y bajada V_{HH}	Min	250	ns
t_{VCS}	Tiempo de establecimiento V_{CC}	Min	50	µs
t_{BUSY}	WE# alto a RY/BY# bajo	Min	90	ns
t_{POLL}	Programación valida antes de requisición de estado	Max	4	µs

Tabla 3.7. Tiempos límite comando de programación/borrado.

III.1.1.5.2.4 Secuencia de comando borrado de chip

El borrado de chip es una operación de seis ciclos de bus. La secuencia de comando borrado de chip es inicializada al escribir dos ciclos de desbloqueo, seguidos por el establecimiento del comando. Dos ciclos de desbloqueo adicionales son escritos y son seguidos por el comando de borrado de chip, el que a su vez invoca el algoritmo de borrado embebido. El dispositivo no requiere que el sistema preprograme antes de borrar. El algoritmo de borrado embebido automáticamente preprograma y verifica la memoria en busca de un patrón de 0 para todos los datos previo al borrado eléctrico. No se requiere que el sistema provea ninguna señal de control o tiempos durante estas operaciones. En la Tabla 3.5 se muestran los requerimientos de dirección y datos para la secuencia de comando borrado de chip.

Cuando el algoritmo de borrado embebido es completado, el dispositivo regresa al modo lectura y las direcciones dejan de ser retenidas. Cualesquiera comandos escritos durante la operación de borrado del chip son ignorados. De cualquier manera, note que un restablecimiento vía hardware inmediatamente termina la operación de borrado. Si esto ocurre, la secuencia de comando borrado de chip debe ser reinicializada una vez que el dispositivo regresa a la lectura del arreglo de datos, para asegurar la integridad de los datos.

En la Figura 3.8 se muestra el diagrama de tiempos para operaciones de borrado, ya sea de chip o sector. Los parámetros mostrados en la Figura 3.8 son listados y descritos en la Tabla 3.7. La única diferencia entre los dos ciclos de borrado es un dato que es escrito en el sexto ciclo de la rutina.

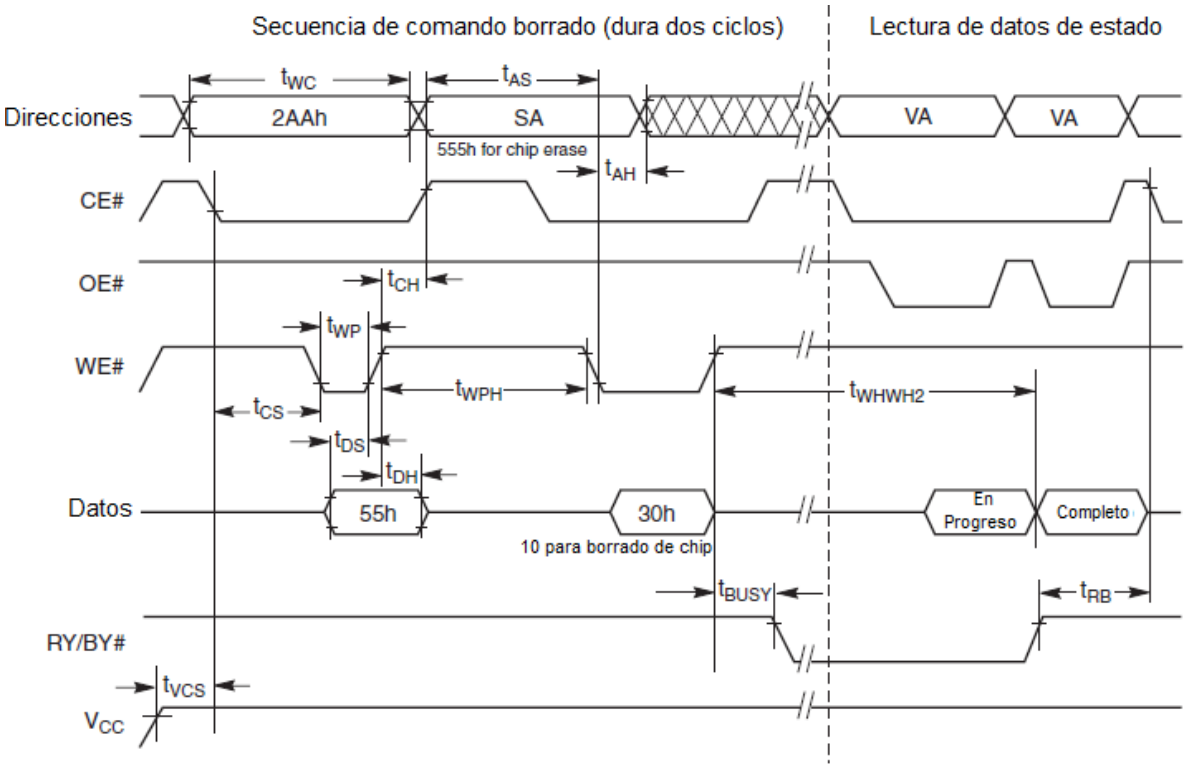

Figura 3.8. Diagrama de tiempos para ciclo de borrado de sector/chip.

III.1.1.5.2.5 Secuencia de comando borrado de sector

El borrado de sector es una operación de seis ciclos. La secuencia de comando borrado de sector es inicializada al escribir dos ciclos de desbloqueo, seguidos por un comando de establecimiento. Dos ciclos adicionales son escritos, y son entonces seguidos por la dirección del sector a ser borrado, y el comando borrado de sector. En la Tabla 3.5 son mostrados los requerimientos de datos y direcciones para la secuencia de comando borrado de sector.

El dispositivo no requiere que el sistema preprograme previo a borrar. El algoritmo de borrado embebido automáticamente programa y verifica la memoria

completa en busca de un patrón de ceros previo al borrado eléctrico. El sistema no es requerido proveer ninguna señal de control o tiempos durante estas operaciones. El correspondiente diagrama de tiempos es mostrado en la Figura 3.8, los parámetros son listados en la Tabla 3.7.

III.1.1.5.2.6 Desempeño borrado y programación

En la Tabla 3.8 se resume el desempeño de las operaciones utilizadas en este trabajo de investigación. Los valores típicos con las siguientes condiciones: 25°C, VCC = 3.0 V, 10,000 ciclos, patrón de datos 1 y 0s alternados. Los valores máximos bajo las siguientes condiciones: 90°C, 100,000 ciclos. En cuanto al borrado se considera que todas las localidades de memoria están programadas a 00h. En los tiempos de programado se excluye el tiempo en el que se ejecuta la secuencia de comando.

Parámetro	Típico	Máximo	Unidad
Tiempo de borrado de sector	0.5	3.5	Segundos
Tiempo de borrado de chip	64	128	Segundos
Tiempo de programación de chip	63	-	Segundos

Tabla 3.8. Tiempos de programación y borrado.

III.1.1.6 Pantalla de cristal liquido

La pantalla utilizada es del fabricante Crystalfontz America Inc. modelo CFAH1602B-TMC-JP, pantalla tipo caracteres, dimensiones 16 columnas 2 líneas.

III.1.1.6.1 Descripción de funcionamiento

El modulo LCD es construido en un controlador LSI, el controlador tiene dos registros de 8 bits, un registro de instrucción (IR) y un registro de datos (DR). El IR almacena los códigos de instrucción, como limpiado de pantalla y desplazamiento

de cursor, y la información de dirección para la RAM (DDRAM) de los datos de la pantalla y el generador de caracteres (CGRAM).

El IR solo puede ser escrito desde el MPU. El DR temporalmente almacena datos a ser escritos o leídos desde la DDRAM o CGRAM. Por medio de la señal selector de registro (RS), estos dos registros pueden ser seleccionados. En la Tabla 3.9 se muestran las operaciones de los registros.

RS	R/W	Operación
0	0	El IR escribe una operación interna
0	1	Lee la bandera de ocupado y contador de dirección
1	0	Escribe datos a la DDRAM o CGRAM (DR a DDRAM o CGRAM)
1	1	Lee datos de DDRAM o CGRAM (DDRAM o CGRAM a DR)

Tabla 3.9. Operaciones bus de datos de modulo LCD.

En la Tabla 3.10 se lista la tabla de instrucciones indicando el código, y el tiempo de ejecución. El símbolo – significa que el valor de tal pin carece de relevancia en la presente instrucción. El tiempo de ejecución es el lapso de tiempo que el microprocesador interno ocupa para ejecutar la instrucción. En la Tabla 3.10 solamente se han incluido los comandos utilizados.

Instrucción	Código de instrucción										Descripción	Tiempo de ejecución
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Limpiar pantalla	0	0	0	0	0	0	0	0	0	1	Escribe “00H” a la DDRAM y establece la dirección de la DDRAM a “00H” desde AC	1.53ms
Retorno a inicio	0	0	0	0	0	0	0	0	1	–	Establece la dirección de la DDRAM a “00H” desde AC y regresa el cursor a su posición original si ha sido	1.53ms

											desplazado	
Establecer modo de entrada	0	0	0	0	0	0	0	1	I/D	SH	Asignar dirección de movimiento del cursor y habilitar el desplazamiento de la pantalla completa	39µs
Control de apagado /encendido	0	0	0	0	0	0	1	D	C	B	Establecer el bit de control apagado/ encendido de la pantalla (D), cursor (C) y parpadeo del cursor (B)	39µs
Desplazamiento de cursor o pantalla	0	0	0	0	0	1	S/C	R/L	–	–	Establecer bit de control movimiento de cursor, desplazamiento de pantalla y dirección, sin cambiar los datos de la DDRAM	39µs
Conjunto de funciones	0	0	0	0	1	DL	N	F	–	–	Establecer longitud de datos de la interfaz (DL: 8bit/4bit), numero de líneas (N: 2-lineas/1-linea) y tipo de fuente (F: 5X11 puntos /5X8 puntos)	39µs
Establecer dirección de DDRAM	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Establecer dirección en el contador de dirección	39µs
Escribir datos a RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Escribir datos en la RAM interna (DDRAM/CGRAM)	43µs

Tabla 3.10. Conjunto de instrucciones del modulo LCD.

En la Figura 3.9 se muestra el correspondiente diagrama de tiempos que describe la manera en que se deben escribir los datos en el modulo LCD, a pesar

de que el fabricante proporciona también un diagrama de tiempos para la lectura este no será utilizado y por lo tanto no se incluye. En la Tabla 3.11 se muestran los valores para los símbolos utilizados en la Figura 3.9. Los pines RS, R/W y E son de control, mientras DB0-DB7 son de datos, también por medio de estos últimos se envían las direcciones de la RAM interna.

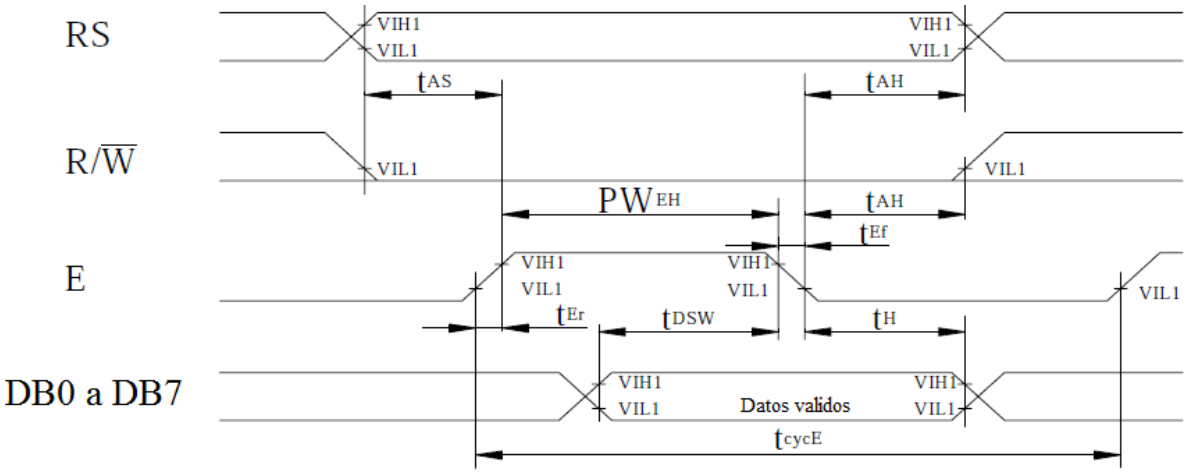

Figura 3.9. Diagrama de tiempos de escritura al modulo LCD.

Elemento	Símbolo	Min	Max	Unidad
Tiempo ciclo de habilitación	t_{cycE}	500	–	ns
Ancho de pulso de habilitación (nivel alto)	PW_{EH}	230	–	ns
Tiempo subida/baja habilitación	t_{Er}, t_{Ef}	–	20	ns
Tiempo de establecimiento de dirección (RS, R/W a E)	t_{AS}	40	–	ns
Tiempo de retención dirección	t_{AH}	10	–	ns
Tiempo de establecimiento de datos	t_{DSW}	80	–	ns
Tiempo de retención de datos	t_{H}	10	–	ns

Tabla 3.11. Tabla de símbolos correspondiente a diagrama de tiempos de la Figura 3.9.

En la Figura 3.10 se muestra la rutina de inicialización del modulo LCD para una interfaz de datos de 8 bits. Antes de leer o escribir cualquier información del modulo LCD la rutina de inicialización debe ser ejecutada. Una vez ejecutada la rutina de inicialización no es necesario proporcionar ninguna señal de control y el modulo queda en espera de secuencias de escritura o lectura. Entre cada bloque

es necesario ejecutar un comando de escritura, esto debido principalmente a la señal de habilitación.

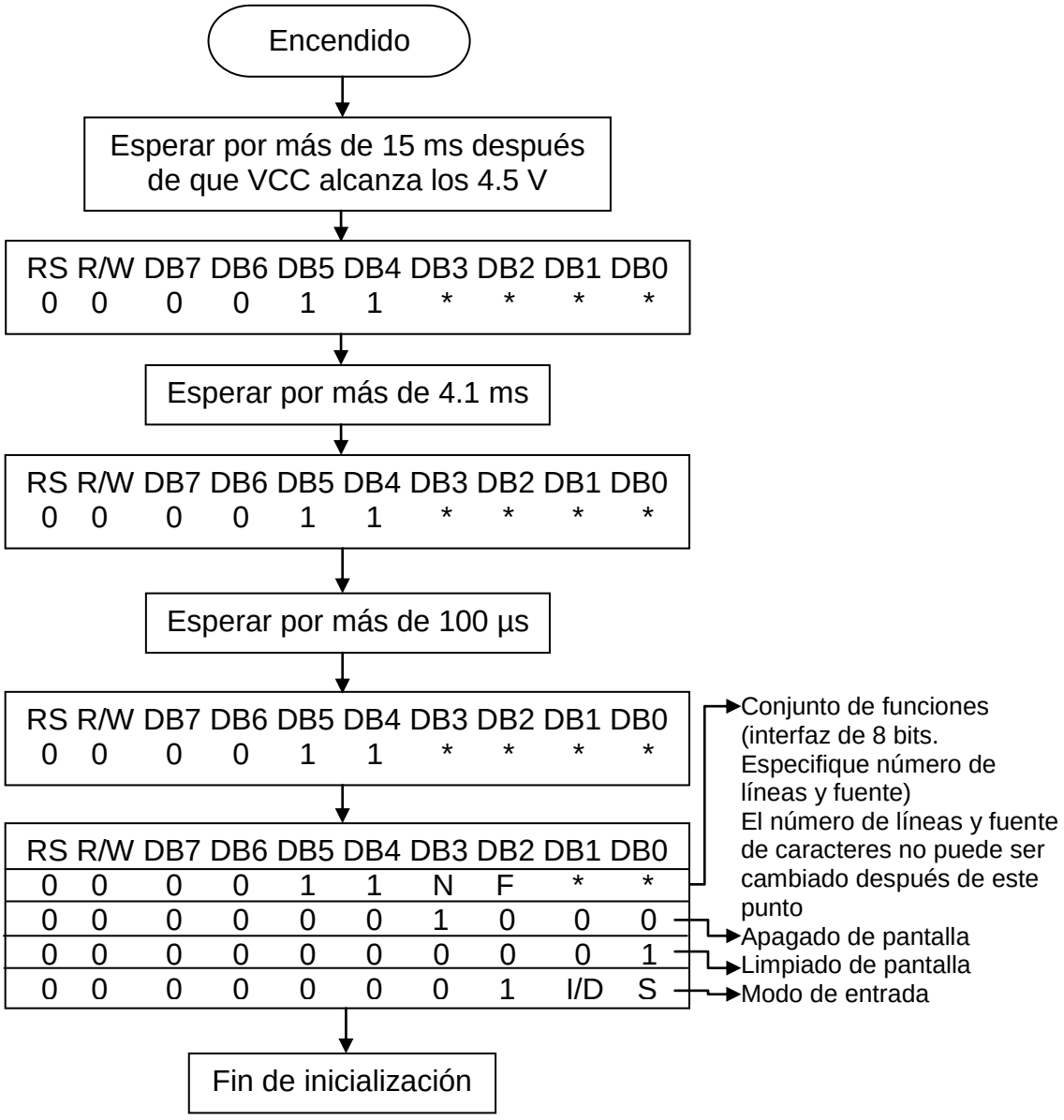

Figura 3.10. Diagrama de flujo de rutina de inicialización de modulo LCD.

III.1.2 Comunicación RS232

De acuerdo con Axelson (2007) los puertos seriales son ideales para comunicaciones entre sistemas embebidos, o entre sistemas embebidos y PCs. Los puertos seriales también pueden ser una buena opción cuando se necesitan cables bastante largos, una red básica entre PCs, sistemas embebidos, o una combinación.

El RS-232 está diseñado para manejar comunicaciones entre dos dispositivos con una distancia límite de entre 80 y 130 pies, dependiendo de la velocidad y tipo de cable. El RS-232 usa líneas desbalanceadas, o de único fin. Cada señal en la interfaz tiene una línea dedicada cuyo voltaje es referenciado a una tierra común.

Comúnmente, RS-232 se refiere a una interfaz serial que cumple con mucho del estándar TIA-232-F: Interfaz entre Equipamiento de Terminal de Datos. El nombre RS-232 se refiere a una edición previa del estándar. El actual editor del estándar es la Asociación de la Industria de las Telecomunicaciones (TIA, Telecommunications Industry Association). Mucha de la terminología del RS-232 refleja su origen como un estándar para comunicaciones entre una computadora terminal y un modem externo. PCs con módems e interfaces de red han hecho este tipo de conexiones terminales casi obsoletas.

Hoy día, un puerto RS-232 es mayormente utilizado para conectar una PC a un sistema embebido o para conectar dos sistemas embebidos. Mucha de la terminología, por tanto, no aplica a las aplicaciones modernas, pero la interfaz hardware se mantiene útil.

El estándar RS-232 llama la terminal final del enlace el equipamiento terminal de datos, o DTE (data terminal equipment). El modem final del enlace es el equipamiento terminador del circuito de datos, o DCE (data circuit - terminating

equipment). En la Tabla 3.12 se enumeran los pines de la interfaz física indicando las características de cada uno. El puerto serial de la PC y otras interfaces utilizan mayormente los nueve pines nombrados aquí, en el caso de este proyecto se utilizaran los pines 2,3 y 5

#Pin	Señal	Fuente	Tipo	Descripción
1	CD	DCE	Control	Detección de portador
2	RX	DCE	Datos	Recepción de datos
3	TX	DTE	Datos	Transmisión de datos
4	DTR	DTE	Control	Terminal de datos lista
5	SG	-	Control	Señal de tierra
6	DSR	DCE	Control	Conjunto de datos listo
7	RTS	DTE	Control	Requisición de envío
8	CTS	DCE	Control	Disponible para envío
9	RI	DCE	Control	Indicador de tono

Tabla 3.12. Señales del conector DB-9.

Los niveles lógicos RS-232 son definidos como voltajes positivos y negativos en vez los voltajes positivos de la lógica TTL y CMOS. A la salida de datos de un RS-232 (TX), 0 lógico es definido igual o mayor que +5V, y 1 lógico es definido igual o menor que -5V. En otras palabras, los datos usan lógica negativa, donde el voltaje más positivo es 0 lógico y el voltaje más negativo es 1 lógico. En la Tabla 3.13 se listan los voltajes y la señal lógica que representan, este utiliza voltajes positivos y negativos.

Parametro	Voltaje(V)
0 lógico o salida encendida	+5 A +15
1 lógico o salida apagada	-5 A -15
0 lógico o entrada encendida	+3 A +15
1 lógico o entrada apagada	-3 A -15

Tabla 3.13. Voltajes especificados por el estándar RS-232.

Un UART transmite datos en fragmentos usualmente llamados palabras. Cada palabra contiene un bit de inicio, bits de datos, un bit de paridad opcional, y

uno o más bits de stop. La mayoría de las UARTs soportan múltiples formatos de palabras. Un formato común es 8-N-1, donde el transmisor envía cada palabra como un bit de inicio, seguido por ocho bits de datos y un bit de paro. La transmisión de datos empezando con el bit 0 (el bit menos significativo, o LSb), el proceso se ilustra en la Figura 3.11, las transmisiones asíncronas requieren que cada dispositivo genere su propia señal de reloj.

La N en 8-N-1 indica que las palabras no contendrán bit de paridad. Para receptores que requieren tiempo extra para aceptar los datos recibidos, algunas UARTs permiten al transmisor extender el bit de paro a 1.5 o 2 bits. El propósito original del bit de paro más largo era para permitir a maquinas de teletipo mecánicas el establecer a un estado inactivo.

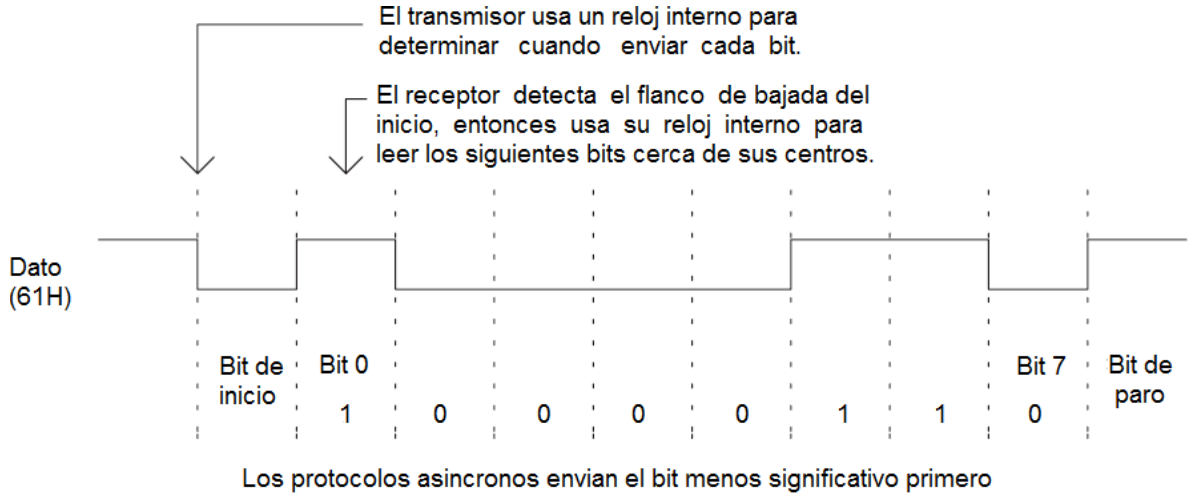

Figura 3.11. Forma de onda de la información a ser enviada en la comunicación serial RS-232.

III.1.3 Tarjeta de desarrollo FPGA

En la Figura 3.12 se muestra una fotografía del kit de desarrollo del fabricante Xilinx, que incluye un FPGA XC3S200, 8 interruptores de palanca, 3 puertos de expansión de 40 pines cada uno; y del que se utilizaran principalmente:

FPGA XC3S200 ~ 200,000 compuertas.

Transceptor RS-232 y conector DB9.

3 cabezales de expansión de 40-pines.

Oscilador de 50 MHz

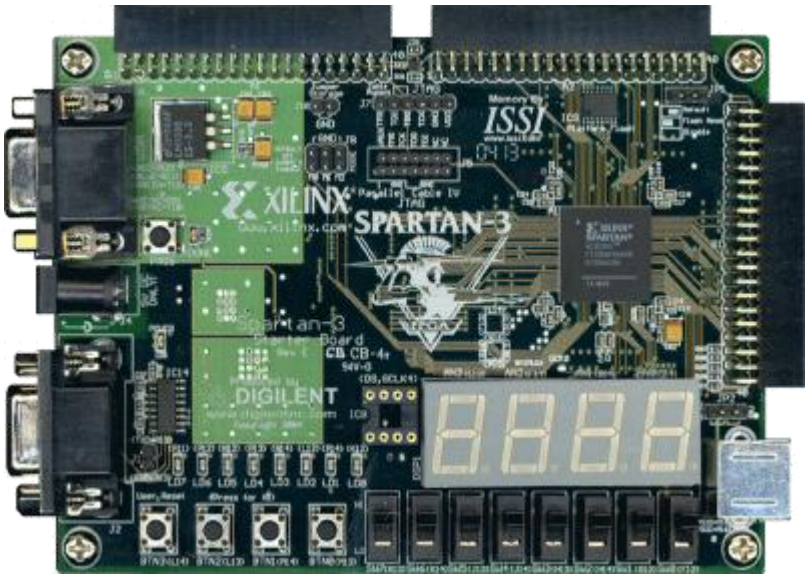

Figura 3.12. Tarjeta de desarrollo Spartan 3.

III.1.4 Sensor de temperatura

Sensor de temperatura Davis 7817

- Termistor de platino.
- Rango -45°c a 60°C.
- Precisión +/- 0.5°C.
- Resolución 1°C.
- Termistor de coeficiente negativo.
- Constante de tiempo: 20 segundos (en líquidos).
- Tipo de cable: 4 conductores, 26 AWG.
- Longitud de cable: 7.6 m.

En la Figura 3.15 se muestra el sensor de temperatura, el sensor es bastante susceptible a fuentes de calor en el cable. El fabricante recomienda su protección a luz del sol o cualquier otra fuente de calor para asegurar su correcto funcionamiento a fin de evitar lecturas erróneas. El sensor de temperatura es de coeficiente negativo, esto es, a mayor temperatura menor resistencia y viceversa, su curva de resistencia se muestra en la Figura 3.14, en la grafica se puede

observar la relación inversa que la resistencia tiene con la temperatura. Dicha curva de comportamiento es la correspondiente a la longitud del cable de 7.6 m, en caso de incrementar la longitud del cable se debe realizar los acondicionamientos de la señal correspondientes.

Figura 3.13. Sensor de temperatura con conector telefónico.

III.1.4.1 Respuesta de termistor

°F	°C	kΩ
-40	-40	328.400
-31	-35	237.700
-22	-30	173.900
-13	-25	128.500
-4	-20	95.890
5	-15	72.230
14	-10	54.890
23	-5	42.070
32	0	32.510
41	5	25.310
50	10	19.860
59	15	15.690
68	20	12.490
77	25	10.000
86	30	8.060
95	35	6.536
104	40	5.331
113	45	4.373
122	50	3.606
131	55	2.989
140	60	2.490

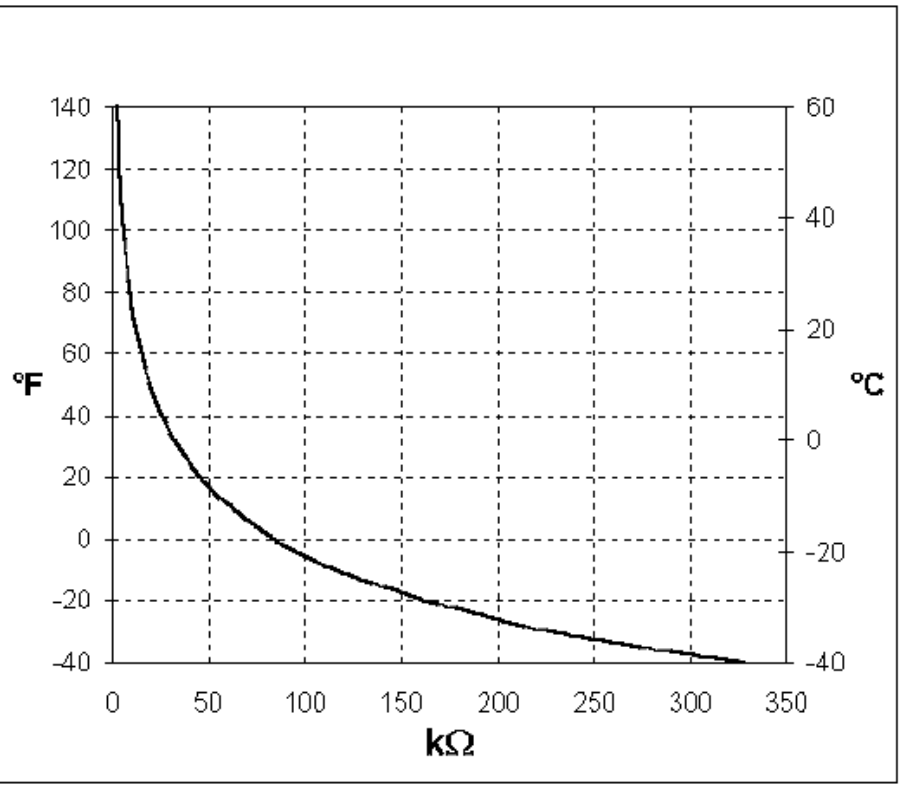

Figura 3.14. Respuesta de termistor 7817 Davis Instruments (2007).

III.1.5 Sensor de oxigeno disuelto

Sensor de oxigeno disuelto Global Water WQ401

- Salida: 4 – 20 mA.
- Rango
  - Saturación 0 – 100%.
  - 0 – 8 ppm.
- Compensación de temperatura a 25 °C.
- Precisión: ±0.5% a escala completa.
- Presión máxima: 40 PSI.
- Voltaje de operación: 10 – 36 VDC.
- Consumo de corriente: 15.5 mA + salida de sensor.
- Tiempo de calentamiento: mínimo 10 segundos
- Temperatura de operación: -40 a +55°C.
- Tamaño: 27.9 cm x 3.2 cm ø.

En la Figura 3.15 se muestra el sensor de oxigeno disuelto a utilizar. El elemento primario de medición debe de ser mantenido húmedo, de lo contrario sería necesario cambiarlo. El elemento primario de medición se ubica en la punta del sensor; la parte metálica es una cubierta protectora.

Figura 3.15. Sensor de oxigeno disuelto.

III.1.5.1 Conversión corriente-voltaje

El sensor de oxígeno es el modelo WQ401 de Global Water, que entrega una señal de corriente entre 4 y 20 mA para representar de 0 a 100% de oxigeno disuelto, en el caso de este sistema se utilizara una conversión a voltaje con un rango equivalente de 0.5 a 2.5 V. Se utiliza una resistencia de 125 ohms para convertir la corriente de salida a voltaje, la conexión de la resistencia es mostrada en la Figura 3.16.

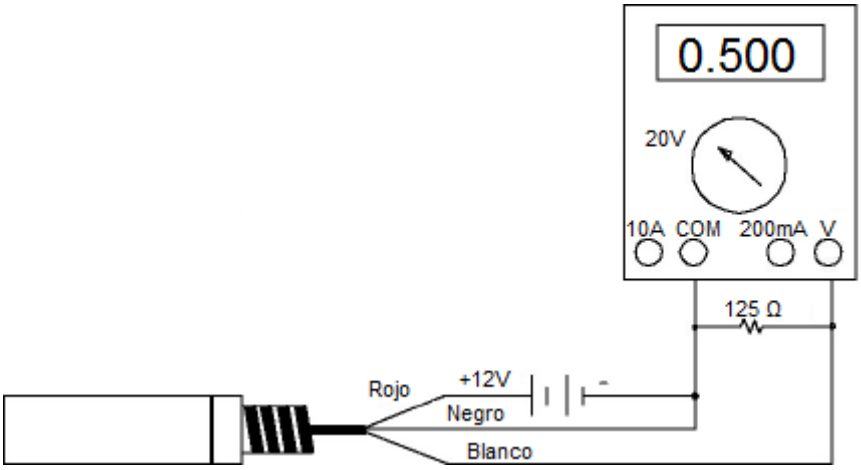

Figura 3.16. Conexión de resistencia para conversión de señal de corriente.

III.1.6 Convertidor analógico digital

Las principales características del convertidor son:

- Resolución 10 bits (expandible a 12 bits).
- 8 canales analógicos.
- Máximo 200 ksps (kilo samples per second – kilo muestras por segundo).
- Referencia, reloj de conversión y 8 x FIFO integrados.
- Diferencial/integral error de linealidad: $\pm$ 1lsb.
- Señal a ruido y radio de distorsión: 59 DB, f_i = 12-khz.
- Amplio y único rango de alimentación: 2.7 a 5.5 VDC.
- Rango analógico de entrada: 0-v a voltaje de alimentación con 500 KHz BW (bandwidth – ancho de banda).
- Corriente de operación: 1.0 mA a 3.3 V, 1.1 mA a 5.5 V con referencia externa.
- Auto-alternado de canal programable.

III.1.6.1 Modos de operación

El convertidor analógico digital (ADC) incorpora cuatro distintos modos de operación que son conversión de canal seleccionado, repetición de lectura de canal seleccionado, intercalado seleccionando una de las cuatro secuencias de lectura preprogramadas y repetición intercalada, se repite la lectura de los canales en el orden especificado por la secuencia de alternado seleccionada.

Se ha seleccionado el modo de operación conversión de canal seleccionado en el cual solamente se toma una lectura del canal seleccionado. Los diagramas de tiempos correspondientes al ciclo de escritura de configuración y ciclo de inicio, muestreo y lectura de conversión son mostrados en las Figuras 3.17 y 3.18 respectivamente. En ambos casos no se utiliza la señal de sincronización de cuadros.

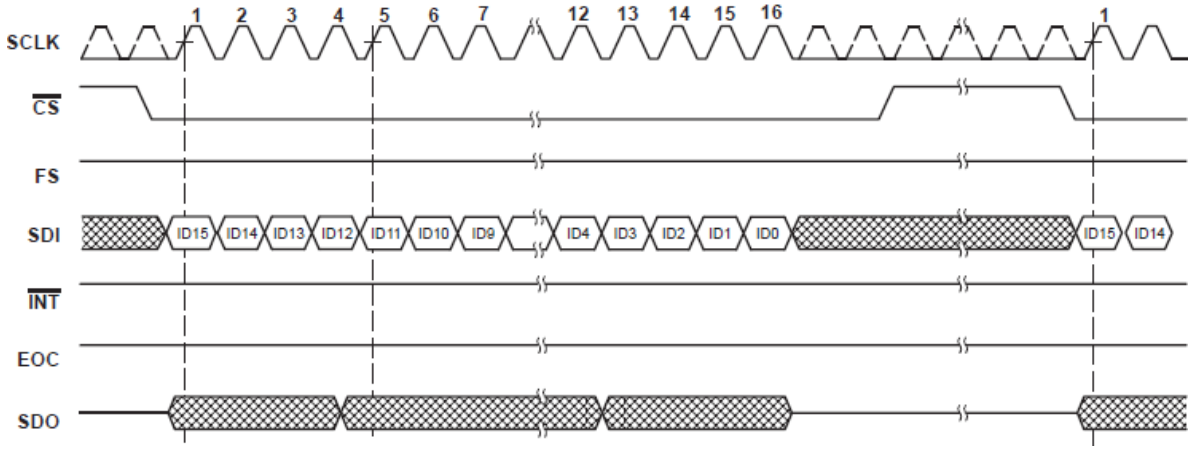

Figura 3.17. Diagrama de tiempos del ciclo de escritura de programación.

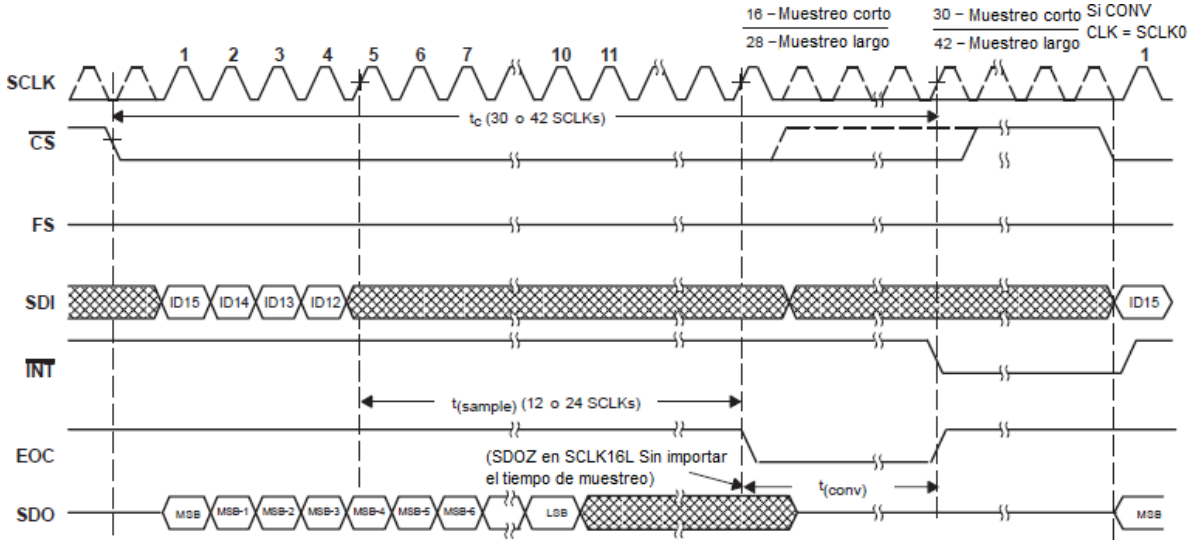

Figura 3.18. Diagrama de tiempos del modo de operación conversión de canal seleccionado.

III.1.7 Software de simulación

El software utilizado para la descripción, simulación de operación y verificación de sintaxis es el Active-HDL™ de la compañía Aldec versión 8.1 compilación 1864. Las librerías utilizadas son STD_LOGIC_1164, STD_LOGIC_ARITH y STD_LOGIC_UNSIGNED.

III.1.8 Software de síntesis

Para el FPGA del fabricante Altera, se utilizará el software Quartus II de la compañía Altera versión 7.2 compilación 151. Para el FPGA del fabricante Xilinx, se utilizará el software ISE versión 9.2i. Ambos software son proveídos gratuitamente por los fabricantes de los FPGA. Cada fabricante maneja su propio software de sintetizado, debido principalmente a la diferente organización de sus dispositivos programables; así como el hecho de que cada fabricante conoce su propia arquitectura y sabe cómo obtener el mejor desempeño u organización de un diseño electrónico a ser grabado en alguna de sus tecnologías programables.

III.1.9 Software para desarrollo de interfaz

El Lenguaje a utilizar es C#, con el fin de facilitar el desarrollo de la interfaz se hará uso de una IDE que es proporcionada gratuitamente por la compañía Microsoft llamada Microsoft Visual C# 2008 Express Edition en su versión 9.0.21022.8 RTM.

III.2 Metodología

III.2.1 Descripción general

El sistema será descrito en VHDL utilizando como herramientas de descripción conjuntamente el software Active-HDL y Quartus II, cuyas versiones

han sido indicadas en las secciones III.1.7 y III.1.8, respectivamente. El sistema en general será descrito a nivel de comportamiento. La descripción de hardware será en base a instanciación de componentes. Los componentes cuyo funcionamiento dependa de una señal de reloj, serán descritos por medio de maquinas de estados finitos.

Para desarrollar el sistema propuesto es necesario implementar varios bloques funcionales, entonces el primer paso para el desarrollo de este proyecto de investigación será la búsqueda de información, actividad que también se desempeñara a lo largo del desarrollo del proyecto. Seguido de ello, se diseñaran versiones tipo borrador del sistema en general; así como de cada uno de los subcomponentes que interconectados entre sí formarán el sistema completo. La versión borrador del diseño de más alto nivel es mostrado en la Figura 3.19.

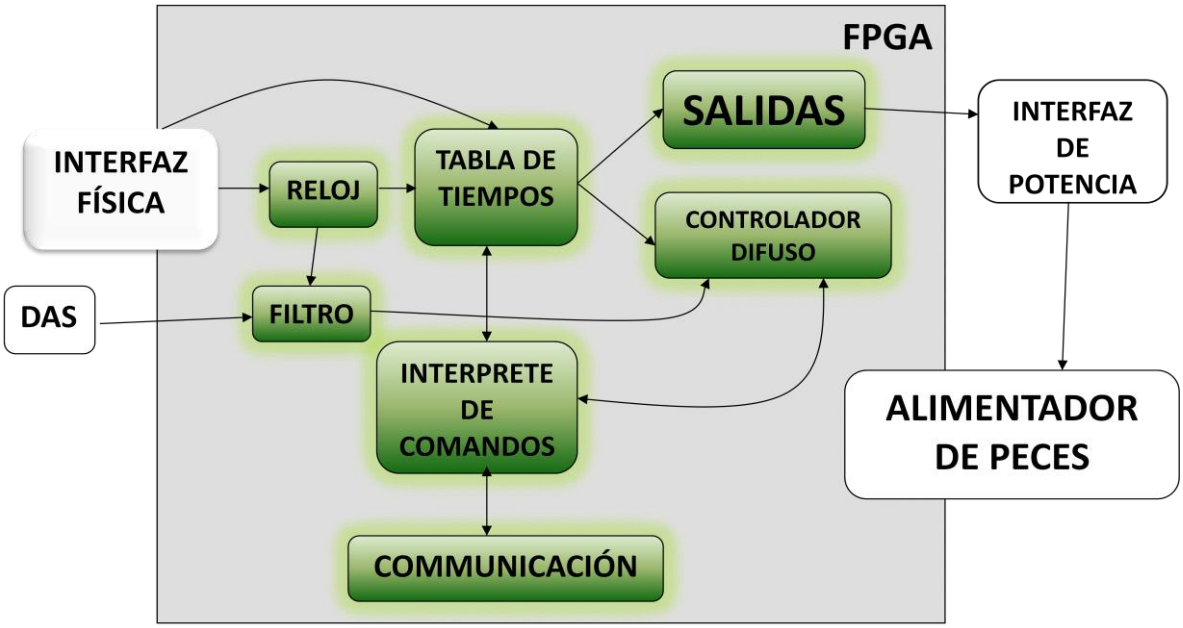

Figura 3.19. Diagrama general del diseño de más alto nivel.

III.2.2 Especificación de bloques funcionales

Cada subbloque mostrado en la Figura 3.19 se le llama bloque funcional. Cada uno de estos subbloques tiene una función específica. Los subbloques

funcionan de manera concurrente a la par con los otros subbloques que componen el sistema completo. El sistema de adquisición de datos (DAS) es uno de estos bloques funcionales, este bloque funcional será desarrollado externamente al FPGA y únicamente el FPGA adquirirá los datos binarios del DAS para procesarlos dentro de la FPGA.

El DAS se empleara para leer las variables que modelan la dinámica del controlador. Dichas variables son temperatura y oxigeno disuelto. El sistema de adquisición de datos tiene el esquema mostrado en la Figura 3.20. A pesar de que algunos componentes ya están integrados en los circuitos integrados a utilizar, son mostrados para enfatizar su importancia.

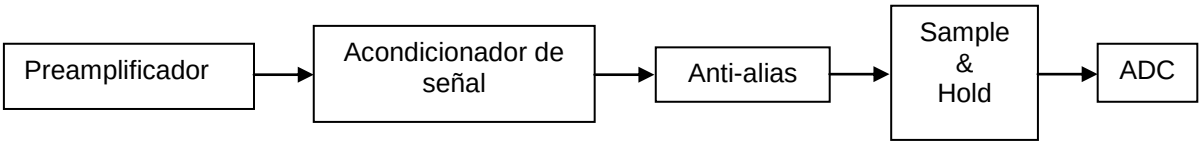

Figura 3.20. Sistema de adquisición de datos.

Las especificaciones de diseño del sistema de adquisición de datos son mostradas en la Tabla 3.14.

Especificación	Valor y Unidad
Resolución	10 bits
Número de canales analógicos	8
Periodo de muestreo	1 ms
Frecuencia de muestreo	1 KHz
Interfaz de configuración	Paralela

Tabla 3.14. Especificaciones de diseño del DAS.

El siguiente bloque funcional es el reloj. El reloj ha de mantener una referencia entre el controlador y el usuario con el fin especificar la hora de activación del alimentador. El reloj también será el bloque responsable de indicar hora del sistema a la tabla de horarios para que se efectúe la activación de las

salidas correspondientes dependiendo si está o no en rango, su diagrama es mostrado en la Figura 3.21.

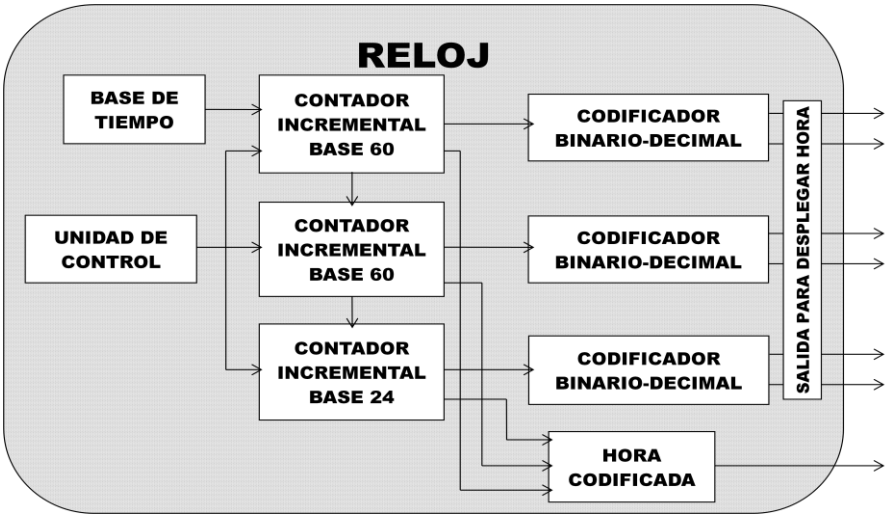

Figura 3.21. Diagrama de subcomponentes internos del reloj del sistema de control.

Como se muestra en la Figura 3.21 el reloj tiene siete salidas. Las primeras 6 (de arriba hacia abajo), son salidas de 4 bits cada una, que representan cada digito de la hora. La última salida (de arriba hacia abajo), representa la hora codificada utilizada por la tabla de horarios. Su longitud es de 17 bits.

Las especificaciones de diseño del reloj son mostradas en la Tabla 3.15.

Especificación	**Valor y unidad**
Resolución	Segundos
Formato	24 horas
Error de sincronización	±1 segundo
Precisión	1 segundo
Interfaz de configuracion	Paralela
Modos de operación	2

Tabla 3.15. Especificaciones de diseño del reloj.

El sistema de control necesita algún medio de almacenamiento para guardar los parámetros del control difuso, tiempos de activación, información relacionada a la activación como tipo de operación (por tiempo o control difuso),

etc. De entre los tipos de memoria, se requiere un tipo de memoria que retenga dicha información aun si el suministro eléctrico es removido.

El kit de desarrollo FPGA incluye principalmente tres tipos de memoria, estática, dinámica y flash. Tanto la memoria estática como la dinámica pierden la información almacenada al remover la fuente de alimentación del circuito. La memoria dinámica es económica, pero por el contrario requiere de ciclos de refrescado, para conservar la información que esta contiene, lo que complica su uso. La memoria estática es fácil de usar pero es muy cara y de bajas capacidades. La memoria Flash, en cuanto al control, tiene un nivel medio de complexidad, tiene un precio medio y puede retener la información aun al remover el suministro eléctrico de energía.

Debido a la complejidad de operación de la memoria Flash, la arquitectura general del sistema de activación por tiempos será verificada con la ayuda de memoria SSRAM. Una vez comprobada la correcta operación del sistema se procederá a desarrollar el controlador para la memoria Flash así como el desarrollo del controlador difuso.

La memoria Flash cumple con las mencionadas características además de ofrecer capacidades de almacenamiento superiores a otras tecnologías de almacenamiento de estado sólido. Debido a que en el chip de memoria Flash se guardara toda la información necesaria para la operación del sistema de control; es necesario controlar el acceso al chip de memoria entre los diferentes subcomponentes que requieran uso del chip para escritura o lectura.

El subcomponente encargado de controlar directamente el chip de memoria es llamado administrador de acceso a memoria y controla cualquier operación de escritura o lectura de la memoria. El administrador de acceso a memoria es solamente un subcomponente del componente encargado del escaneo continuo de la memoria. El mencionado escaneo de la memoria se hace en busca de

entradas de activación escritas en la memoria, y es llevado a cabo por un componente llamado Tabla de horarios. El procedimiento de operación de la tabla de horarios es: partiendo de una condición de restablecimiento a) genera incrementalmente la dirección de memoria b) proporciona dirección de memoria al administrador de acceso a memoria y comanda operación de lectura c) una vez completada la operación de lectura verifica los códigos de identificación de la entrada de activación leída en caso de ser por tiempos compara la hora del sistema con la hora de activación de la entrada de activación; en caso de correspondencia paso “d)” en caso de no correspondencia paso “e)” d) activa la salida correspondiente y regresa a paso “a)” e) verifica si coincide con tiempo de desactivación en caso de correspondencia desactiva salida correspondiente al finalizar la desactivación y en caso contrario (a la correspondencia) regresa a paso “a)”.

Si en el paso “c)” los códigos de identificación señalan la presencia de una entrada de activación por medio de cálculos difusos el componente operara de la siguiente manera: d) decodificación de conjuntos difusos, limites, base de reglas, maquina de inferencias; así como llenado de esqueleto de control difuso con dicha información decodificada e) inicio de cálculo por parte de controlador f) tabla de horarios en espera de cálculo de horario de apagado por parte del controlador difuso g) escritura de horario de apagado en memoria Flash h) regreso a paso “a)”.

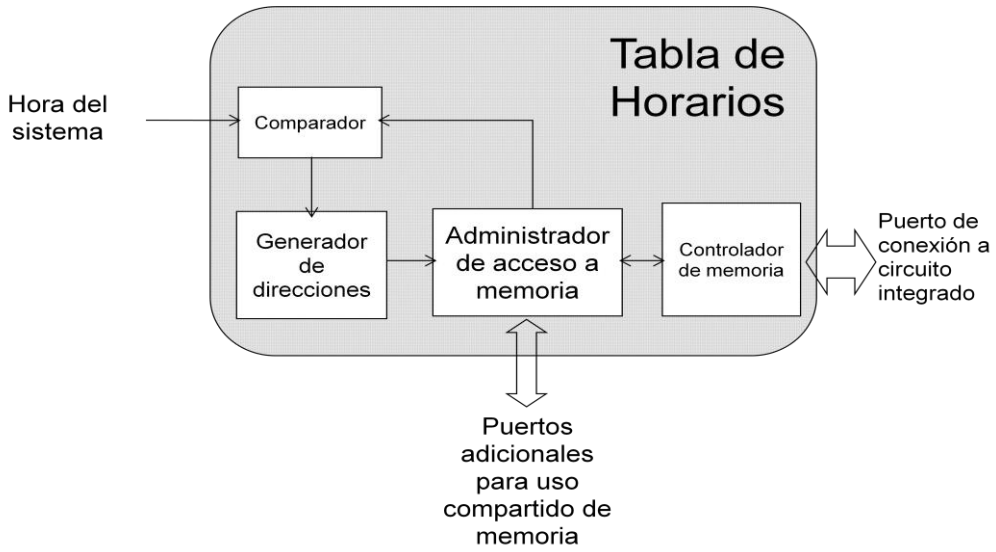

Figura 3.22. Diagrama de subcomponentes internos de la tabla de horarios.

En la Figura 3.22 se muestra el diagrama de subcomponentes del componente Tabla de horarios. El generador de direcciones es un contador incremental controlado por la máquina de estados de la tabla de horarios. Debido al uso de dos tipos de memoria distintos se deberán de crear dos administradores de acceso a memoria, dos tablas de horarios, dos controladores de memoria y se deberán de definir dos codificaciones de información; una para cada tipo de memoria.

Las codificaciones de memoria definidas de acuerdo a las características de cada chip son mostradas en las Tablas 3.16 y 3.17 para la memoria síncrona estática y Flash, respectivamente.

Byte 3			Byte 2			Byte 1			Byte 0		
Reservados (3 bits)			Numero de dispositivo (7 bits)		Hora (5 bits)		Minuto (6 bits)			Segundo (6 bits)	
Código de identificación (4 bits)		Inicio/Paro (1 bit)									
2^{31}	…	2^{24}	2^{23}	…	2^{16}	2^{15}	…	2^{8}	2^{7}	…	2^{0}

Tabla 3.16. Codificación de la información (SSRAM).

Word 1						Word 0					
Reservados (3 bits)		Numero de dispositivo (7 bits)		Hora (5 bits)		Minuto (6 bits)			Segundo (6 bits)		
Código de identificación (4 bits)		Inicio/Paro (1 bit)									
2^{15}	…	2^{8}	2^{7}	…	2^{0}	2^{15}	…	2^{8}	2^{7}	…	2^{0}

Tabla 3.17. Codificación de la información (Flash).

El componente de comunicación es el responsable de la comunicación entre la computadora y el controlador. Este bloque ha sido diseñado para trabajar a velocidad y configuración fija. Las especificaciones de diseño del bloque de comunicación son mostradas en la Tabla 3.18. El bloque de comunicación básicamente adecua la información sobre el canal serial como una trama de bits a una velocidad y codificación especifica.

Especificación	**Valor y unidad**
Velocidad	115200 bps
Paridad	Ninguna
Numero de bits de datos	8 bits
Numero de bits de paro	1 bit
Tipo de comunicación	Full-Duplex

Tabla 3.18. Especificaciones de diseño componente de comunicación.

El diagrama de subcomponentes internos del componente de comunicación es mostrado en la Figura 3.23. Principalmente se ilustra la base de tiempo que indica el momento para enviar cada bit, registros seriales para almacenar la trama serial recibida y a enviar, convertidores paralelo-serie y serie-paralelo para adecuar la señal recibida y la enviada. Así como su correspondiente unidad de control para indicar cuándo enviar los datos y verificar si existe algún dato en el canal de entrada. Esta unidad es directamente controlada por el componente intérprete de comandos.

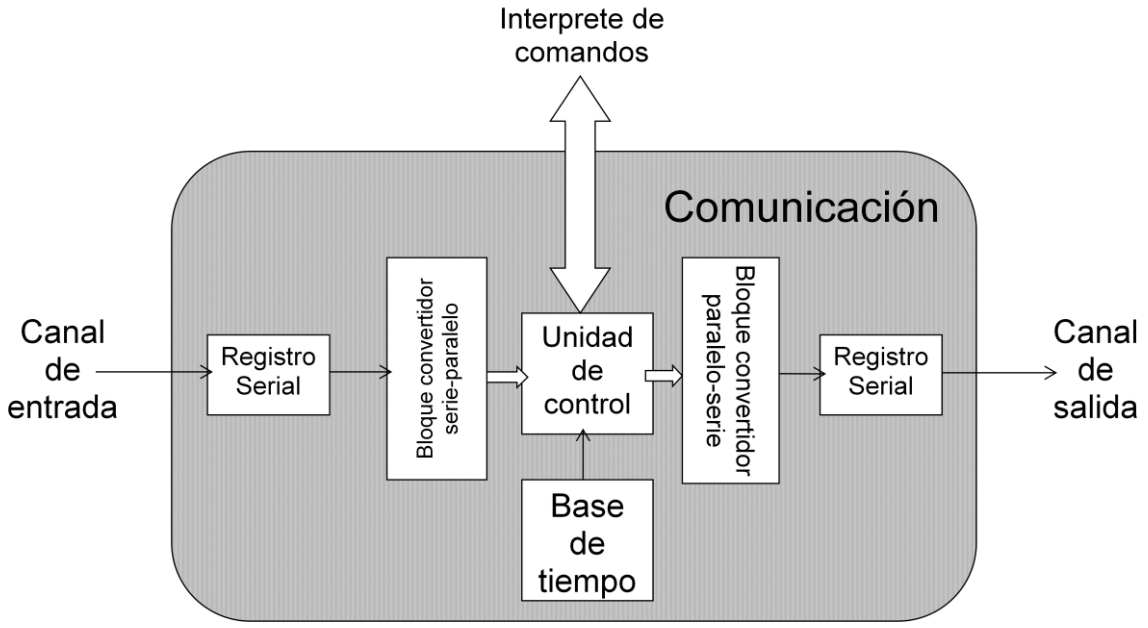

Figura 3.23. Diagrama de subcomponentes internos del componente de comunicación.

Debido al tiempo de desarrollo que demandaría la elaboración de una interfaz grafica de usuario embebida en el sistema, se hará uso de la computadora como interfaz grafica del sistema. Así entonces se hace necesario el desarrollo de un intérprete de comandos, mostrado en la Figura 3.24. El intérprete de comandos, como su nombre sugiere, controlara el sistema dependiendo del comando enviado desde el programador de horarios, comandos como establecer/borrar programación(es), establecer hora del sistema, configurar conjuntos difusos, obtener lecturas del DAS, etc.

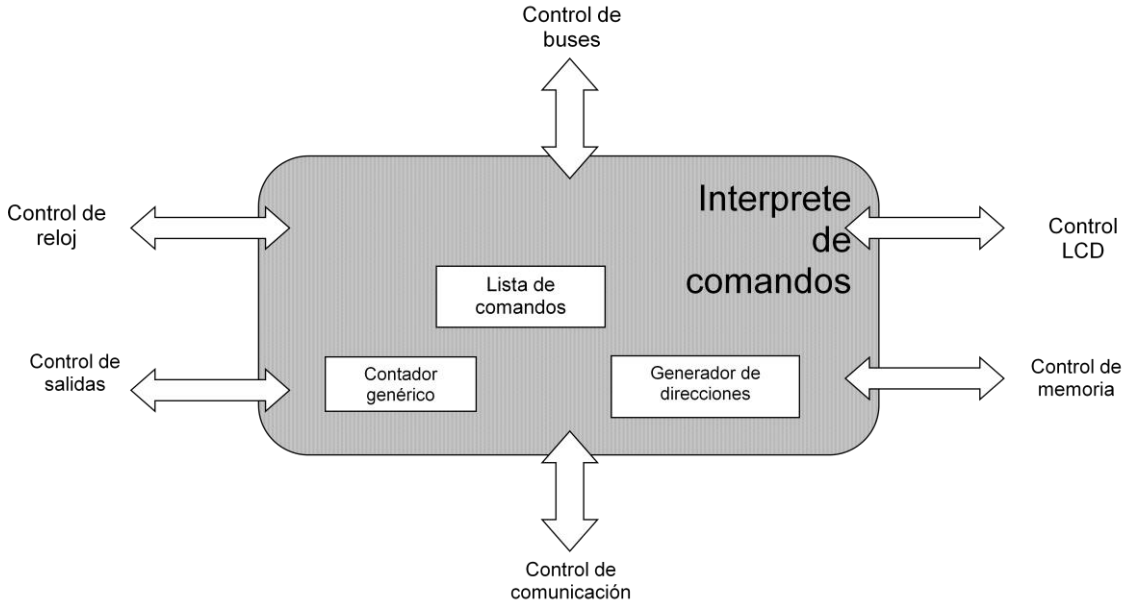

Figura 3.24. Diagrama de subcomponentes internos del intérprete de comandos.

El componente encargado de controlar el estado de las salidas es el mostrado en la Figura 3.19 como “Salidas”. El nombre designado al tal componente es controlador de salidas; su arquitectura es mostrada en la Figura 3.25. Éste tiene tres modos de operación para cada pin de salida.

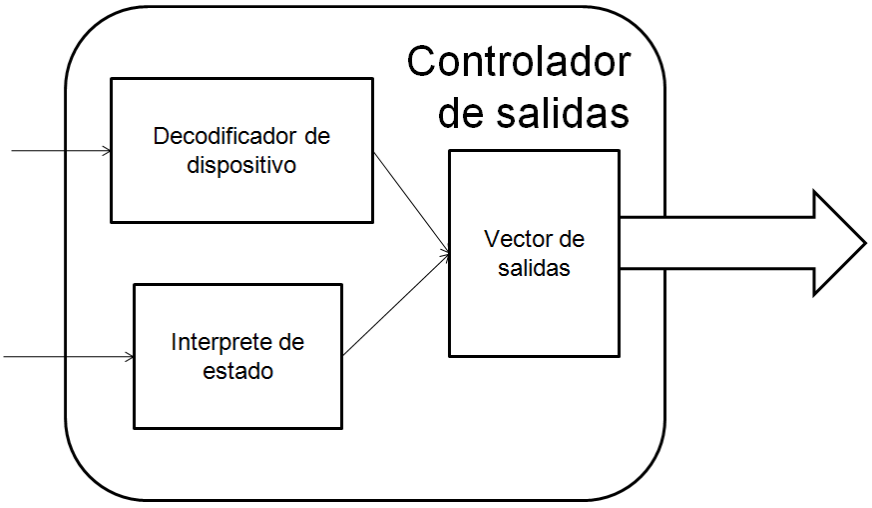

Figura 3.25. Diagrama de subcomponentes internos del controlador de salidas.

Las especificaciones de diseño son mostradas en la Tabla 3.19. El controlador de salidas, después de una condición de restablecimiento, inicia con todas las salidas apagadas por seguridad.

Especificación	**Valor y unidad**
Número máximo de salidas	128
Modos de operación	Mantener estado Apagar salida Activar salida
Cantidad de salidas modificable	Si
Limite de físico de salidas definido por codificación y cantidad de pines del FPGA	

Tabla 3.19. Especificaciones de diseño controlador de salidas.

El controlador difuso funcionará a base de los llamados componentes típicos de un controlador de su clase. Tales componentes son fuzzificador, base de reglas, conjuntos difusos, maquina de inferencias y defuzzificador; la estructura del controlador es mostrada en la Figura 3.26. El tanto la fuzzificador como el defuzzificador serán implementados por medio de tablas de verdad variables, cuyos valores de salida serán cargados desde la memoria Flash. El tamaño de dichas tablas de verdad será variable dependiendo del tamaño de los conjuntos difusos que apliquen a dicha operación de fuzzificacion o defuzzificacion. Los conjuntos difusos serán almacenados en la memoria Flash. Las especificaciones de diseño del controlador difuso son mostradas en la Tabla 3.20.

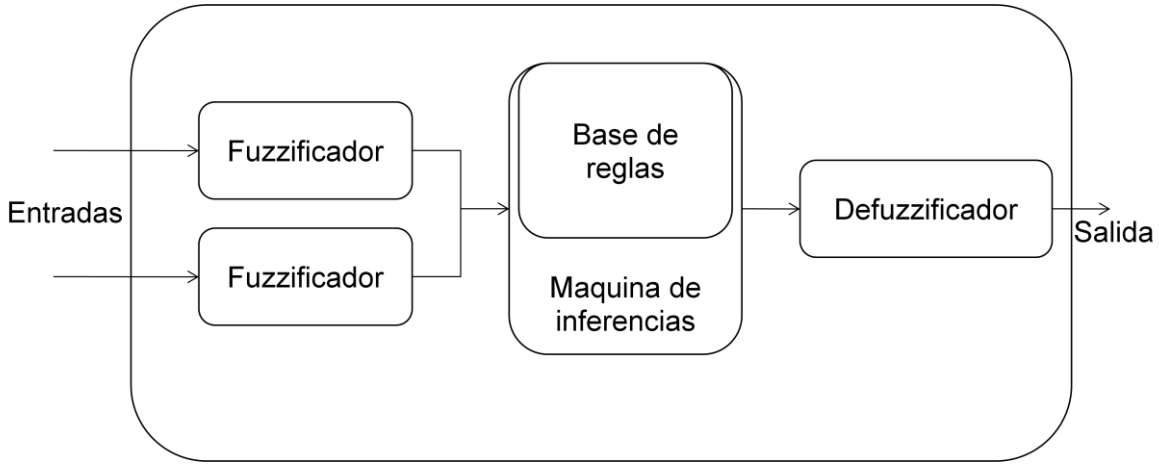

Figura 3.26. Diagrama de subcomponentes internos de controlador difuso implementado.

Especificación	Valor y unidad
Número de entradas	2
Número de salidas	1
Número máximo de conjuntos difusos por entrada/salida	5
Tipo de fuzzificadores soportados	Triangular y trapezoidal
Número máximo de reglas por controlador	25
Tipo de reglas	And lógico

Tabla 3.20. Especificaciones de diseño controlador difuso.

La codificación utilizada para almacenar los conjuntos difusos será la mostrada en la Tabla 3.21. Nótese que utilizara el mismo tamaño de la codificación para las programaciones por tiempos. F es una bandera que indica al sistema si es necesario calcular el tiempo de apagado o si éste ya ha sido calculado.

La tabla de valores de salida es cargada en las últimas direcciones de la memoria Flash. Una vez que la tabla de tiempos identifique una activación basada en el controlador difuso, éste escaneará las ultimas direcciones del chip de

memoria en busca de la LUT cuyo código de identificación corresponda con el especificado en la activación. Las limitantes de este diseño, por la codificación de la memoria, son de 256 controladores difusos y de 16 entradas analógicas. En el caso de esta implementación solamente se han utilizado 8 entradas.

Word 1																								Word 0											
Identificador								Controlador difuso (Parte alta)							Identificador de dispositivo											Hora Encendido					Minuto Encendido						Segundo Encendido					
2^{31}			2^{28}					2^{27}			2^{24}				2^{23}							2^{17}			2^{16}				2^{12}		2^{11}				2^{6}		2^{3}			2^{0}
Word 3																								Word 2											
Identificador								Controlador difuso (Parte baja)					F		Entrada 1					Entrada 2					Hora calculada apagado					Minuto calculado apagado						Segundo calculado apagado					
2^{31}			2^{28}					2^{27}			2^{24}		2^{23}		2^{22}			2^{20}		2^{19}			2^{18}		2^{16}				2^{12}		2^{11}				2^{6}		2^{3}			2^{0}

Tabla 3.21. Codificación de las activaciones por medio del controlador difuso.

III.2.3 Desarrollo del sistema

III.2.3.1 Panorama general

Cada uno de los bloques funcionales especificados en la sección III.2.2 será descrito en VHDL. El software utilizado en la fase de descripción es el especificado en la sección III.1.7 Active-HDL; de acuerdo con la sección II.8 lo primero que se escribe en todos y cada uno archivos de diseño es las librerías. Seguido de ello se declarará la entidad del diseño electrónico especificando los genéricos en caso de ser necesarios; los puertos indicando nombre, tipo de puerto y tipo de datos.

La declaración de la arquitectura se ubicará después de la declaración de la entidad. Previo a la descripción del cuerpo de la arquitectura se declararán los componentes en caso de ocupar alguno, en caso de hacer uso de alguna señal interna la instanciación de estas se hará entre el cuerpo de la arquitectura y la declaración del los componentes. Dentro del cuerpo de la arquitectura se instanciaran los componentes utilizados, y seguido de ello se listará la descripción de los procesos requeridos en caso de existir.

Todos los procesos secuenciales responderán al cambio del reloj en el flanco positivo, a excepción de la máquina de estados del convertidor analógico digital. La máquina de estados del convertidor analógico digital será el único diseño electrónico que responderá al flanco negativo de la señal de reloj, con el fin de evitar el uso de la señal de sincronización de cuadros (FS).

El diseño contendrá dos señales de reloj. La primera es la señal de reloj maestro, el cual será conectado a todos los componentes secuenciales. La frecuencia de la señal de reloj maestro será de 50 MHz. El controlador del ADC incluirá un componente que generará una señal de reloj de 5 MHz, dicha señal de reloj será utilizada única y exclusivamente por el controlador del ADC.

Los nombres de las entidades denotarán en la mayor medida de lo posible la función de cada componente. Para los nombres de las entidades se utilizarán letras mayúsculas para la primera letra de cada palabra que componga el nombre de la misma. Para los nombres de los puertos de cada entidad se utilizara la primera letra minúscula y la primera letra de cada subsiguiente palabra que le componga ira con mayúsculas. Se utilizará un nivel de indentación para cada subelemento de cualquier estructura.

Los archivos de diseño serán asociados a un diseño dentro de un espacio de trabajo. Los archivos de diseño llevarán el nombre de la entidad que describe cada uno de estos. Los archivos de diseño de las entidades de uso en más de un diseño, se incluirán en un diseño que contenga

III.2.3.2 Descripción del reloj

El reloj del sistema contará con una base de tiempo que generará un pulso de flanco positivo de una duración de 20 ns, dicho pulso será generado por cada segundo transcurrido. El pulso generado será conectado a la entrada del primer

contador incremental de base 60. Tanto el contador de base 60 como el de base 24, operarán en uno de 2 modos.

El primer modo espera el arribo de la señal de base de tiempo para incrementar su cuenta en 1 y en caso de llegar a la cuenta “Base - 1” generará un pulso de flanco de subida a la salida. El segundo modo cargara la cuenta de un puerto de entrada. Se instanciarán dos contadores base 60 y un contador base 24. Cada bloque contador tiene un puerto por el cual se externa la cuenta actual.

El primer contador (de arriba hacia abajo) mostrado en la Figura 3.21 llevará la cuenta de los segundos, y por base de tiempo, la señal de la base de tiempo del reloj. El segundo contador base 60 llevara como base de tiempo la salida del primer contador base 60; éste llevará la cuenta de los minutos. El tercer contador tendrá como base de tiempo la salida del contador de los minutos y contará las horas transcurridas.

La unidad de control (Figura 3.21) del reloj indicará a cada uno de los tres contadores su modo de operación. Solamente se podrá establecer un valor de cuenta a la vez. La condición de restablecimiento maestra establecerá todos los contadores en 0. El codificador binario-decimal tomara como entrada la cuenta de cada contador y la representara en dos salidas de 4 bits. Una de las salidas representara las unidades y otra las decenas.

El controlador del LCD será administrado por una maquina de estados que únicamente mostrara la hora del sistema. El controlador del LCD ejecuta la rutina de inicialización mostrada en la Figura 3.10 después de una condición de restablecimiento. Una vez inicializado el LCD se despliega la hora del sistema en formato 24 hrs.

III.2.3.3 Descripción de tabla de horarios

La tabla de horarios incorpora un generador de direcciones, éste proporciona al administrador de acceso a memoria la dirección de 22 bits. En el diseño final, el administrador de acceso a memoria se separará de la tabla de horarios y el controlador de la memoria será un componente interno del administrador de acceso a memoria.

La tabla de horarios decodificará la información de acuerdo a la codificación establecida en las Tablas 3.16, 3.17 y 3.21. La primera acción después de una condición de restablecimiento es, restablecer el generador de direcciones a 0. Seguido de ello se espera a que los buses de direcciones, datos y control sean asignados a la tabla de horarios por el administrador de acceso a memoria. La tabla de horarios únicamente puede leer información de la memoria Flash, quedando inhabilitadas las operaciones de programado y borrado sector/chip.

Una vez teniendo el control de lectura de la memoria flash se lee la primera dirección del chip de memoria y se decodifica la primera parte de la programación (Word 1 en Tabla 3.17, Word 1 en Tabla 3.21). Ya decodificada la información se incrementará en 1 la dirección del generador de direcciones, se comanda la lectura de la memoria y se decodificará la segunda parte de la programación de encendido (Word 0 en Tabla 3.17, Word 0 en Tabla 3.21). Nuevamente se incrementa la dirección del generador de direcciones, se ejecuta la lectura de la localidad de memoria y se decodifica la primera parte de la programación de apagado (Word 1 en Tabla 3.17, Word 3 en Tabla 3.21).

En el caso de las programaciones por tiempo la codificación utilizada para la programación de encendido y apagado son iguales solo cambiando la bandera inicio/paro, 0/1 respectivamente. Finalmente se incrementa la dirección del generador, se comanda la lectura y se decodifica la segunda parte de la programación de apagado. Si el código de identificación de la información es 0x1,

la programación será ejecutada con la rutina de tiempos; si el identificador es 0x2, se ejecutará como programación por control difuso.

Independientemente de si es por control difuso o por tiempos si la hora del sistema es mayor o igual a la hora de encendido de la programación de encendido se comandará al controlador de salidas el encendido de dicha salida. En caso de que sea por tiempos su apagado se comandará cuando la hora del sistema sea mayor que la hora de apagado.

Cuando se enciende alguna salida por controlador difuso, el sistema comanda la ejecución del algoritmo de asignación de tiempo de apagado. El controlador difuso tomará el control de programado del memoria Flash y una vez completado el proceso de cálculo éste grabará en dicha localidad el tiempo de apagado.

Al finalizar el proceso de evaluación de las programaciones de encendido y apagado, el generador de direcciones incrementara en 1 y volverá a decodificar las 4 palabras de la siguiente programación. Cuando la memoria haya sido escaneada por completo el proceso comenzara nuevamente a partir de la primera dirección de memoria.

III.2.3.4 Descripción de administrador de acceso a memoria

El administrador de acceso a memoria incluye al controlador de memoria como un componente. Su única función es recibir las peticiones de operación del intérprete de comandos, el controlador difuso y la tabla de horarios. Su principio de operación puede ser asociado a un multiplexor con prioridades para la selección de las peticiones. Éste tiene un puerto de datos, puerto de direcciones y puerto de control, de entrada, para cada uno de los tres dispositivos.

Por medio de una sentencia **if** verifica si el intérprete de comandos ha comandado alguna operación para la memoria Flash, en caso de no haber petición de operación verificara el estado de las peticiones del controlador difuso. En último caso si ninguno de los dos anteriores ha solicitado el uso de la memoria Flash el control de ésta será asignado a la tabla de horarios en el caso de que ésta lo requiera.

El control de la memoria lo mantendrá el dispositivo que haya solicitado el uso de ésta hasta que se haya concluido la operación requerida. Cuando ningún dispositivo requiera el uso de la memoria el administrador de acceso a memoria desconectara todos los buses y esperará hasta el arribo de alguna petición.

III.2.3.5 Descripción de comunicación

Como se ha especificado en la Tabla 3.18, la comunicación será bidireccional y al mismo tiempo (Full-duplex). El componente de comunicación se compone entonces de dos subcomponentes de operación independiente, el componente de recepción y el de transmisión. El componente de recepción estará esperando que el canal de entrada cambie su estado de 1 a 0 para empezar el proceso de recepción. Los datos serán interceptados a la mitad de cada bit que integre la trama de entrada.

La base de tiempo generará un pulso de flanco positivo cada 8680 ns con una duración de 20 ns, este pulso indicara al componente de recepción cuando interceptar cada bit de la trama de entrada. Así mismo el componente de transmisión hace uso de la instanciación de la base de tiempo para determinar en qué momento enviar cada uno de los bits de la trama de salida.

El componente de transmisión utilizará una señal para indicar el inicio de la transmisión y generará una señal que indique el fin de la transmisión. El componente de recepción generará una señal que indique cuando se está

recibiendo algún dato. A pesar de que se han integrado ambos componentes en un solo diseño se debe tener en cuenta que su operación es independiente.

III.2.3.6 Descripción del intérprete de comandos

El intérprete de comandos concentra los puertos de control de todos los componentes del sistema. Éste se encargará principalmente de configurar el sistema. Se implementará un nivel de stack para optimizar las operaciones de comunicación en modo recepción. Cualquier información que se envié por medio de la computadora al chip, llegará a este componente y se procesará para operar los componentes de la manera que corresponda.

Las operaciones actualmente previstas para la configuración del sistema son: establecer hora del sistema, programar memoria Flash, borrar sector, borrar chip, cambiar a modo manual el control de las salidas, leer programaciones guardadas en la memoria Flash y obtener lectura de entrada analógica.

Uno de los componentes que instanciara el intérprete de comandos es el decodificador de sector, este genera la dirección del sector a borrar en caso de ejecutar la operación borrado de sector.

III.2.3.7 Descripción del controlador de salidas

El controlador de salidas opera a base de dos entradas el número de dispositivo y el modificador de estado. Internamente el decodificador de dispositivo funciona a base de una LUT que tiene como entrada 7 bits y como salida 128 bits que indican la salida seleccionada. Al valor decodificado se le aplicará la operación **or** con el vector de salidas para encender una salida; y una operación and entre el negado del valor decodificado y el vector de salidas para apagar una salida. Finalmente el valor del vector de salidas se actualiza en el siguiente ciclo de reloj.

III.2.3.8 Descripción del controlador difuso

El controlador difuso será almacenado en la memoria Flash como una LUT. El controlador incorporará para la entrada de temperatura un rango de 20 a 30, con incrementos de 0.1. La entrada del oxigeno disuelto tiene un rango de 20 a 100, con incrementos de 0.1. La salida tendrá un rango de 2.5 a 4.5 con incrementos de de 0.1.

Las conjuntos difusos propuestos por Soto-Zarazúa (2010) para temperatura, oxigeno disuelto y porcentaje de alimentación son mostrados en las Figuras 3.27, 3.28 y 3.29, respectivamente. En la Tabla 3.22 se muestran las variables lingüísticas utilizadas, mientras la base de reglas es mostrada en la Tabla 3.23.

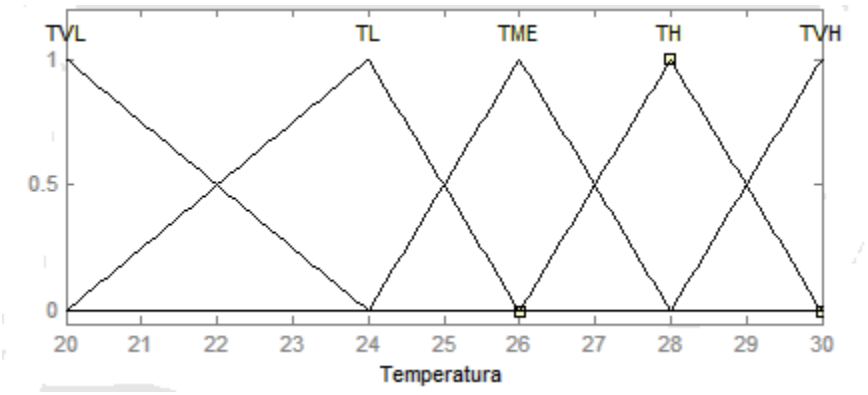

Figura 3.27. Conjuntos difusos para la entrada de temperatura.

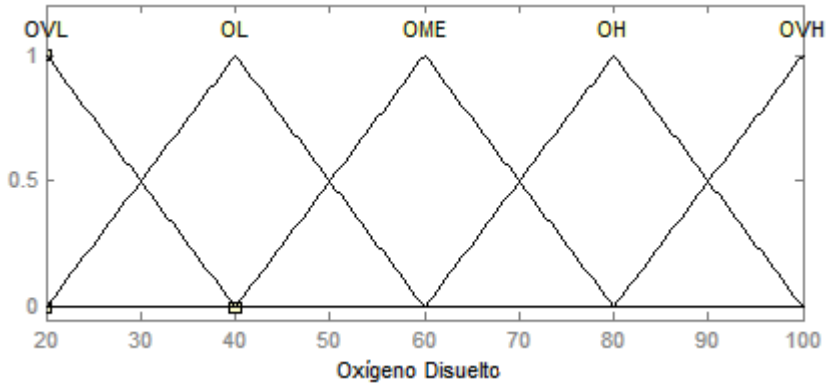

Figura 3.28. Conjuntos difusos para la entrada de oxígeno disuelto.

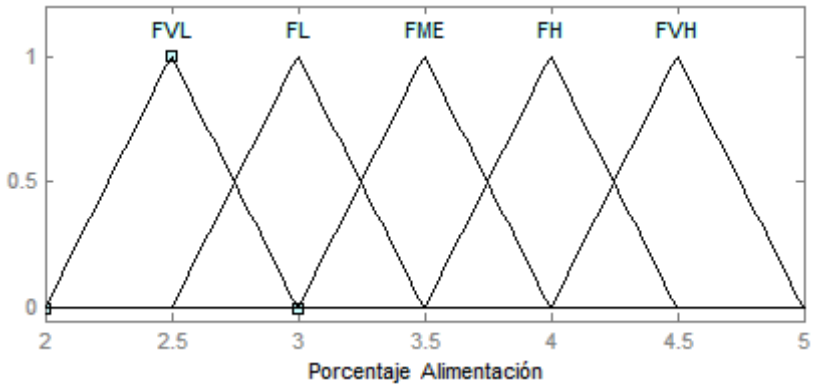

Figura 3.29. Conjuntos difusos para la salida de porcentaje de alimentación.

Temperatura (°C)	Oxígeno disuelto (% de saturación)	Porcentaje de alimentación (FP)
20(TVL)	20(OVL)	2.5(FVL)
24(TL)	40(OL)	3(FL)
26(TME)	60(OME)	3.5(FME)
28(TH)	80(OH)	4(FH)
30(TVH)	100(OVH)	4.5(FVH)

Tabla 3.22. Valores de las variables lingüísticas.

T/OD	**OVL**	**OL**	**OME**	**OH**	**OVH**
TVL	FVL	FVL	FVL	FVL	FVL
TL	FVL	FL	FL	FL	FL
TME	FVL	FL	FME	FME	FME
TH	FVL	FL	FME	FH	FH
TVH	FVL	FL	FME	FH	FVH

Tabla 3.23. Base de reglas.

Tales parámetros serán introducidos en el software de configuración del chip y este calculara con base en ellos la respuesta del sistema y la escribirá en la memoria Flash con un número de identificación de 8 bits. Dicho identificador será utilizado por las activaciones por control difuso para seleccionar el controlador adecuado.

III.2.4 Acondicionamiento de señales DAS

Para el sensor de temperatura se utilizará un puente de Wheatstone, esto para detectar con mayor facilidad los cambios de temperatura. Las resistencias utilizadas serán de 10 KΩ. La fórmula para calcular el voltaje del puente de Wheatstone es mostrada en la Ecuación 1

$$V_G = \left(\frac{R_x}{R_3+R_x} - \frac{R_2}{R_1+R_2}\right) V_S \qquad \text{Ecuación 1}$$

El voltaje calculado con la Ecuación 1 esta referenciado a 2.5 v para calcular el voltaje con referencia a tierra será necesario sumarle dicha referencia al voltaje calculado. El voltaje de salida del puente será acoplado al ADC por medio de un seguidor de voltaje. El filtro a utilizar a la entrada de las lecturas del ADC será un filtro promediador. Este guarda la última lectura y la promedia con la nueva. El resultado será contenido en registros y estará disponible después del primer ciclo de lectura.

IV. RESULTADOS Y DISCUSION

A continuación se presentarán las simulaciones de cada componente, describiendo aspectos generales de la simbología utilizada así como descripción de funcionamiento y modificaciones a los diseños propuestos en la sección de metodología. En la siguiente sección finalmente se mostrarán los modelos RTL de cada componente descrito en VHDL y se darán las conclusiones finales del proyecto de investigación.

IV.1 Simulación de componentes y operación

Las simulaciones de los diseños electrónicos han sido elaboradas por medio del software Active-HDL especificado en la sección III.1.7 del capítulo 3. Para todas las figuras de simulación se presentan diferentes símbolos correspondientes al tipo de pin simulado. El símbolo representa un pin de entrada; el símbolo representa un pin de salida, el símbolo representa una señal interna del componente y el símbolo indica un pin bidireccional. Para los pines que sean precedidos por el símbolo se tratara de una señal tipo std_logic_vector; aquellos que no sean precedidos por el mismo serán de tipo std_logic. La línea de color verde indica alta impedancia.

IV.1.1 Reloj del sistema

En la Figura 4.1 se muestra la simulación de uno de los contadores base 60 del reloj del sistema. El componente incorpora dos funciones que son contar y cargar cuenta. En la Figura 4.1 se muestra la operación de cuenta de 0 a 59 y su regreso a 0. También se muestra el puso enviado al contar 60 pulsos de entrada y la operación de cargado de cuenta.

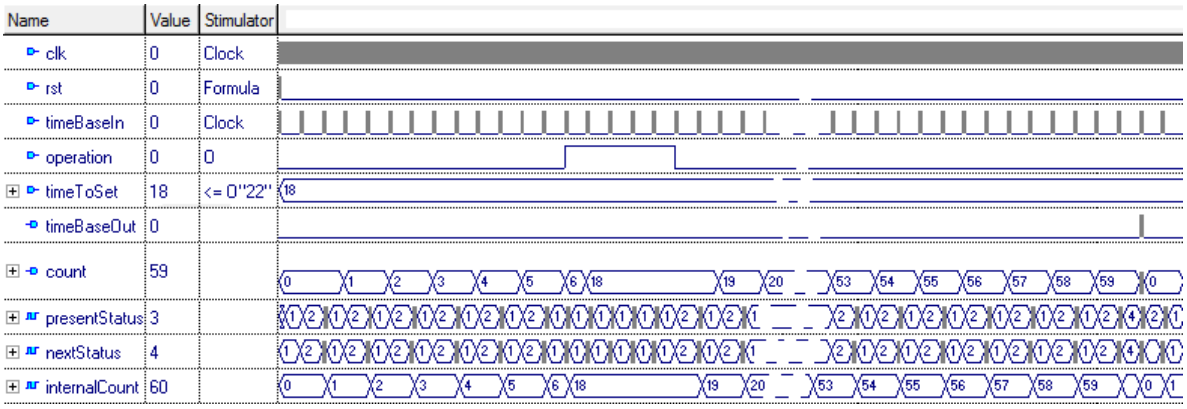

Figura 4.1. Simulación contador base 60.

En la Figura 4.1 la línea de tiempo ha sido omitida por carecer de importancia; de igual manera la simulación mostrada ha sido recortada después de la cuenta de 20 para ilustrar partes más importantes como la regresión de la cuenta y el pulso de salida. Una vez validado el componente por medio de la simulación se modifico el límite de cuenta a 23 para la cuenta de las horas y se interconectaron con la unidad de control del reloj.

El reloj fue probado directamente en la tarjeta descrita en la sección III.1.1, delegando las señales de control a los interruptores. El diseño del reloj ha presentado buenos resultados siendo capaz de contar correctamente segundos, minutos y horas con un error de sincronización máximo de ±1 segundos.

Los codificadores binario decimal que se muestran en la parte derecha de la Figura 3.21 son componentes meramente combinacionales, su simulación es mostrada en la Figura 4.2. El digito uno representa las decenas mientras el digito dos representa las unidades. El codificador binario decimal puede representar únicamente números entre 0 y 99. El rango de números a representar para desplegar la hora es de 0 a 59 para minutos y segundos, mientras para la hora es de 0 a 23.

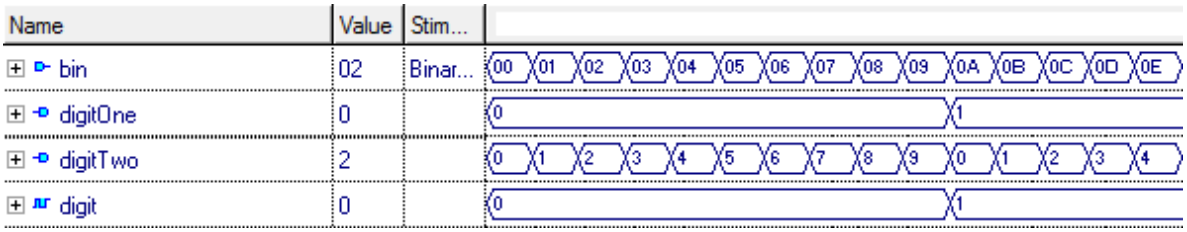

Figura 4.2. Simulación codificador binario decimal.

El componente que codifica la hora del sistema ilustrado en la Figura 3.21 como “Hora codificada” concatena los valores binarios que representan la hora, minuto y segundo en el orden mencionado; utilizando 5, 6 y 6 bits respectivamente. Finalmente la base de tiempo desempeña el comportamiento mostrado en la Figura 4.3. Este funciona a base de una señal de reloj de 50 MHz y emite un pulso (flanco positivo) por cada segundo transcurrido. La cota en azul indica el tiempo entre cada pulso.

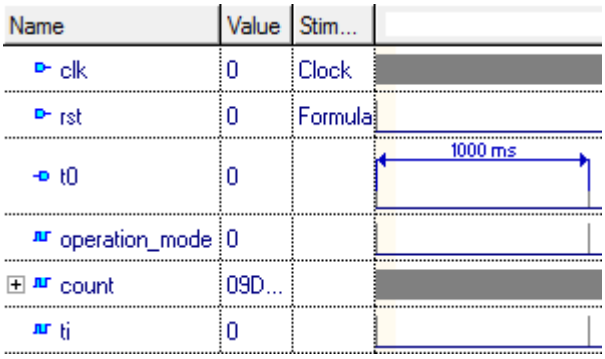

Figura 4.3. Simulación base de tiempo de 1 segundo.

Cabe remarcar que de la idea de diseño inicial del reloj no se hicieron modificaciones en su arquitectura.

IV.1.2 Tabla de horarios

De acuerdo al diseño de la tabla de horarios inicial se ha optado en la etapa de descripción subir en uno la jerarquía del “Administrador de acceso a memoria” como también embeber el “Controlador de memoria” dentro del “Administrador de acceso a memoria”.

Una vez aclarada dicha modificación de diseño, se ha desarrollado en primera instancia el controlador de la memoria SSRAM. Seguido de ello se ha descrito una rutina de lectura continua a través de la memoria a lo que se le ha referido en la sección III.2.2 como escaneo continuo de la memoria. Dicho escaneo, se encarga de leer la información de la memoria en busca de programaciones de activación. Tales programaciones son buscadas de acuerdo con el esquema de codificación mostrado en la Tabla 3.16 y comparadas con la hora del sistema en busca de correspondencia de encendido, apagado o inmutación de estado. Debido a que el bus de datos de la memoria SSRAM es de 32 bits es posible leer con sola operación de lectura la información relacionada con el encendido/apagado de salidas. El procedimiento de escaneo para la memoria SSRAM después de una condición de restablecimiento es: a) restablecer apuntador de memoria para leer a partir de la primera localidad de memoria b) esperar la correcta conexión de los buses por parte del administrador de acceso a memoria así como la selección de operación por parte del intérprete de comandos c) comandar al controlador de memoria la lectura d) esperar el arribo de los datos del chip de memoria e) decodificar y almacenar temporalmente (por medio de registros) el identificador de dispositivo, el identificador de información, la bandera modificador, la hora de encendido (hora:minuto:segundo), así como almacenar temporalmente la hora actual del sistema e incrementar la dirección de memoria f) comandar al controlador de memoria la lectura g) esperar el arribo de los datos del chip de memoria h) decodificar y almacenar temporalmente el identificador de dispositivo, el identificador de información, la bandera modificador, la hora de apagado (hora:minuto:segundo) e incrementar la dirección de memoria i) cotejar ambos identificadores de información para corroborar que se trata de una programación por tiempos e incrementar la dirección de la memoria j) en caso de que el identificador haya fallado en el cotejo del identificador de información regresa a paso “b)”, en caso contrario, si la hora actual del sistema es mayor o igual a la hora de encendido y menor a la hora de apagado ir a paso “k)” en caso contrario y si la hora actual del sistema es igual a la hora de apagado ir a paso “l)”

en caso contrario ir a paso “b)” k) comandar el encendido de la salida de acuerdo al identificador de dispositivo después regresa a paso “b)” l) comandar el apagado de la salida de acuerdo al identificador de dispositivo después regresa a paso “b)”.

La rutina es repetida indefinidamente y el único componente que puede interrumpir esta rutina es el intérprete de comandos a través de las señales de control de la Tabla de horarios. En el caso de la Tabla de horarios para la memoria Flash, es igualmente repetida indefinidamente y solamente interrumpida por el intérprete de comandos. La única diferencia entre las dos Tablas de horarios es que para la memoria Flash es necesario hacer 4 operaciones de lectura para procesar una programación por tiempos contra las 2 operaciones de lectura requeridas por la memoria SSRAM.

Lo anterior, debido únicamente al ancho del bus de datos de la memoria Flash que es de 16 bits. El procedimiento de escaneo para la memoria Flash después de una condición de restablecimiento es: a) restablecer apuntador de memoria para leer a partir de la primera localidad de memoria b) esperar la correcta conexión de los buses por parte del administrador de acceso a memoria así como la selección de operación por parte del intérprete de comandos c) comandar al controlador de memoria la lectura d) esperar el arribo de los datos del chip de memoria d) decodificar y almacenar temporalmente los primeros 16 bits de la hora de encendido e incrementar la dirección de la memoria e) comandar al controlador de memoria la lectura f) esperar el arribo de los datos del chip de memoria g) decodificar y almacenar temporalmente el bit más significativo de la hora de encendido, el identificador de dispositivo, el identificador de información y la bandera modificador, e incrementar la dirección de la memoria h) comandar al controlador de memoria la lectura i) esperar el arribo de los datos del chip de memoria j) decodificar y almacenar temporalmente los primeros 16 bits de la hora de apagado e incrementar la dirección de la memoria k) comandar al controlador de memoria la lectura l) esperar el arribo de los datos del chip de memoria m) decodificar y almacenar temporalmente el bit más significativo de la hora de

apagado, el identificador de dispositivo, el identificador de información y la bandera modificador n) cotejar ambos identificadores de información para corroborar que se trata de una programación por tiempos e incrementar la dirección de la memoria o) en caso de que el identificador haya fallado en el cotejo del identificador de información regresa a paso “b)”, en caso contrario, si la hora actual del sistema es mayor o igual a la hora de encendido y menor a la hora de apagado ir a paso “p)” en caso contrario y si la hora actual del sistema es igual a la hora de apagado ir a paso “q)” en caso contrario ir a paso “b)” p) comandar el encendido de la salida de acuerdo al identificador de dispositivo después regresa a paso “b)” q) comandar el apagado de la salida de acuerdo al identificador de dispositivo después regresa a paso “b)”.

El esquema de codificación acuerdo al cual se buscan las programaciones en la Tabla de horarios para la memoria Flash es el mostrado en la Tabla 3.17. Para ambas Tablas de horarios el incremento de dirección de memoria es por uno. Las dos Tablas de horarios han sido diseñadas con un puerto de conexión al Administrador de acceso a memoria; dicho puerto es de solo lectura.

IV.1.3 Administrador de acceso a memoria

Al igual que la Tabla de horarios cada tipo de memoria requiere su propio Administrador de acceso a memoria. El Administrador de acceso a memoria funciona a base de prioridades. El Administrador de acceso a memoria contiene 3 buses de datos, 3 buses de direcciones, 3 buses de control y un terceto adicional que se conecta directamente al controlador de la memoria.

El primer puerto de conexión (entiéndase por puerto de conexión un bus de datos, uno de control y uno de direcciones) ha sido designado al intérprete de comandos, el siguiente puerto es utilizado por el controlador difuso y el último por la Tabla de horarios.

Tanto el puerto de conexión del intérprete de comandos como el del controlador difuso incluyen en su bus de control la función de escribir y leer datos, mientras el puerto de la Tabla de horarios es de solo lectura. La mayor jerarquía es asociada al intérprete de comandos, seguido de este el controlador difuso y la menor jerarquía es asignada a la Tabla de horarios.

A través de los 3 puertos de conexión se pueden requerir operaciones de lectura o escritura al mismo tiempo y el Administrador de acceso a memoria decidirá en base a la mencionada jerarquía quien tiene el control de la memoria. Dentro de este componente se ha embebido el controlador de memoria como un componente interno.

Como parte del Administrador de acceso a memoria se han reservado los correspondientes pines de conexión del chip de memoria. Dichos pines del chip de memoria son conectados al Administrador de acceso a memoria y el Administrador de acceso a memoria los conecta internamente al controlador de memoria.

IV.1.3.1 Controlador de memoria SSRAM

Adicionalmente a las señales de control del chip de memoria que son manejadas directamente por el controlador, se han designado 2 señales de control general de operación del controlador de memoria estas son ejecución de operación y selección de operación. La selección de operación puede ser lectura (0b01), escritura (0b10) o desconexión (0b00 y 0b11); la señal de selección de operación es de 2 bits. La señal de ejecución de operación funciona por nivel (0 o 1) 0 es permanecer en desconexión y 1 es ejecutar la operación seleccionada.

IV.1.3.1.1 Operación de lectura

Con base en el diagrama de tiempos de la Figura 3.2 solamente se ha implementado el ciclo de lectura sencillo. El controlador opera de la siguiente

manera después de una condición de restablecimiento: a) pone en alta impedancia los buses de datos, dirección y bits de paridad del chip de memoria; y espera el cambio de nivel (a 1) de la señal de ejecución. Una vez activada la señal de ejecución, si la operación es 0b01 (lectura) ir a paso “b)”, si la operación es 0b10 ir a paso “g)” de la sección IV.3.1.2 y si la operación es 0b00 o 0b11 regresa a paso “a)” b) establece dirección de lectura, pone a 0 CE1 y CE3, pone a 1 CE2 c) pone en alta impedancia el bus de direcciones, CE1, CE2 y CE3 d) pone en 0 el habilitador de salida, CE1 y CE3, pone en 1 CE2, y retiene los datos leídos de la memoria en registros e) pone en cero CE1, CE2 y CE3 f) enciende bandera de finalización de lectura de datos y regresa a paso “a)”. En la Figura 4.4 se muestra el diagrama de tiempos obtenido de la simulación de esta rutina.

Con respecto a la Tabla 3.1 se han dado los siguientes tiempos para: t_{AS} = 10 ns, t_{AH} = 10 ns, t_{CES} = 10 ns, t_{CEH} = 10 ns, t_{KC} = 20 ns, t_{KH} = 10 ns, t_{KL} = 10 ns, t_{OEQ} = 0 ns, t_{WS} = 10 ns, t_{WH} = 10 ns, t_{KQ} = 0 ns, t_{KQLZ} = 0 ns, t_{OELZ} = 0 ns. Todos los tiempos se encuentran dentro de las cotas especificadas por el fabricante de la memoria, descritas en la Tabla 3.1 con respecto a la Figura 3.2.

En la Figura 4.4 se muestra el resultado de la simulación del controlador de la memoria SSRAM en modo lectura. Se indican en cotas azules los tiempos de cada evento. Los cambios de estado toman efecto en el flanco de bajada de la señal de reloj.

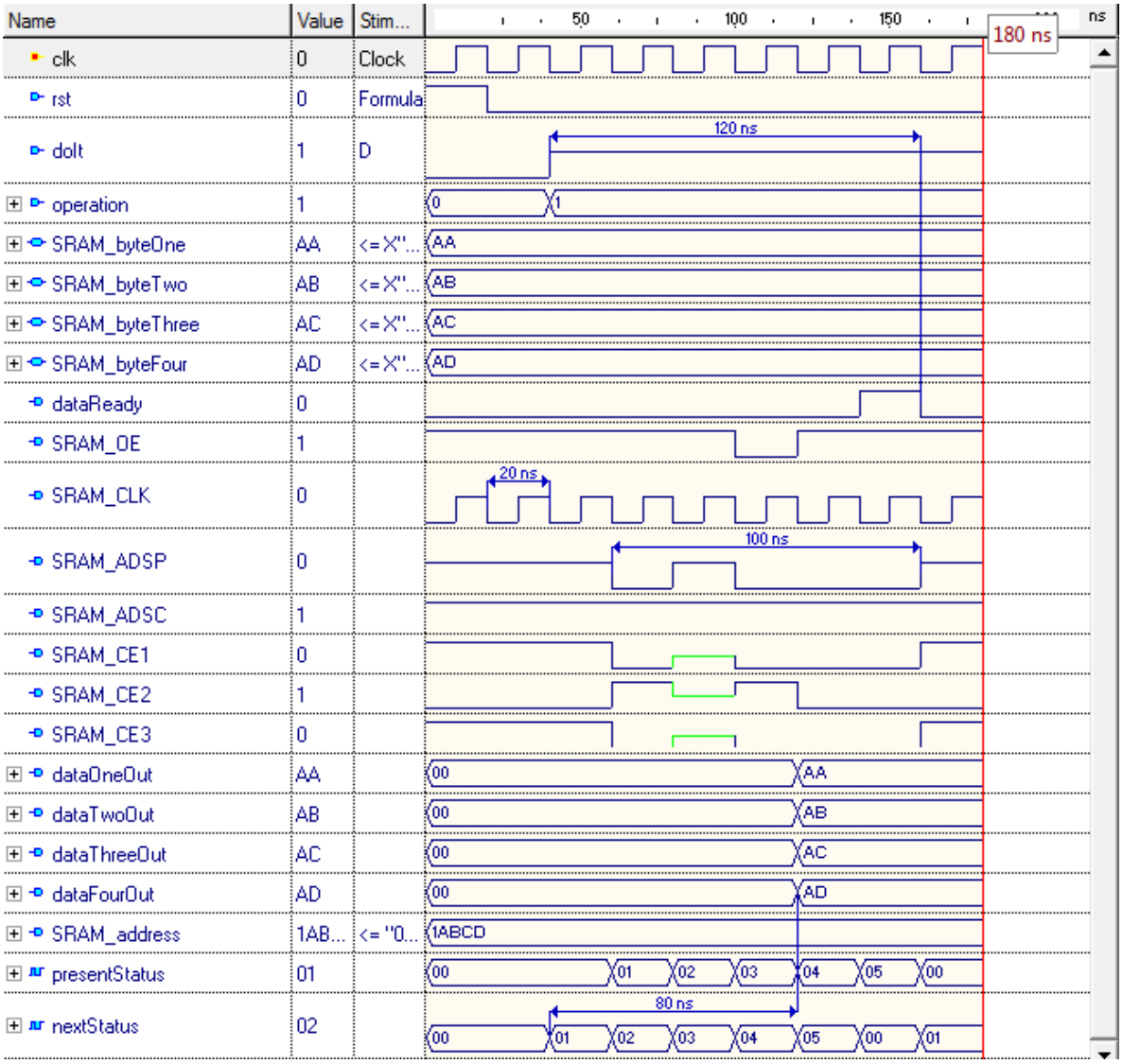

Figura 4.4. Simulación controlador de memoria SSRAM, rutina de lectura.

De acuerdo con lo anterior el tiempo máximo total de lectura del controlador descrito es de 100 ns por cuarteto de bytes, con un tiempo total de operación de 120 ns, y un retardo máximo de datos a salida de 80 ns.

La tasa total de lectura continua es de 31.7 MB por segundo. La capacidad de la memoria SSRAM es de 2 MB; esto implica que teóricamente la Tabla de horarios podrá ser leída 15.8 veces por segundo. La tasa de lectura proporciona margen suficiente para facilitar la prueba de la arquitectura del sistema.

IV.1.3.1.2 Operación de escritura

Solamente se ha implementado el ciclo de escritura sencillo. Para la operación de escritura el controlador se basa en el diagrama de tiempos de la Figura 3.3. El controlador parte del estado del inciso “a)” de la sección IV.1.3.1.1 la secuencia continua al paso g que es: g) establece la dirección de escritura, pone en 0 CE1 y CE3, pone en 1 CE2 h) establece los datos a escribir para cada byte, pone en 0 BWE, especifica los bytes a escribir por medio de BWA, BWB, BWC y BWD, y pone en alta impedancia el bus de direcciones i) pone en alta impedancia el bus de datos, pone a 0 CE1, CE2 y CE3, pone en 1 BWE j) enciende la bandera de finalización de escritura de datos y regresa al paso “a)” de la sección IV.3.1.1.

Con respecto a la Tabla 3.1 se han dado los siguientes tiempos para: t_{KC} = 20 ns, t_{KH} = 10 ns, t_{KL} = 10 ns, t_{AS} = 10 ns, t_{AH} = 10 ns, t_{WS} = 10 ns, t_{WH} = 10 ns, t_{CES} = 10 ns, t_{CEH} = 10 ns, t_{DS} = 10 ns, t_{DH} = 10 ns. Todos los tiempos se encuentran dentro de las cotas especificadas por el fabricante de la memoria, descritas en la Tabla 3.1 con respecto a la Figura 3.3.

En la Figura 4.5 se muestra el resultado de la simulación del controlador de la memoria SSRAM en modo escritura. Se indican en cotas azules los tiempos de cada evento. Los cambios de estado toman efecto en el flanco de bajada de la señal de reloj.

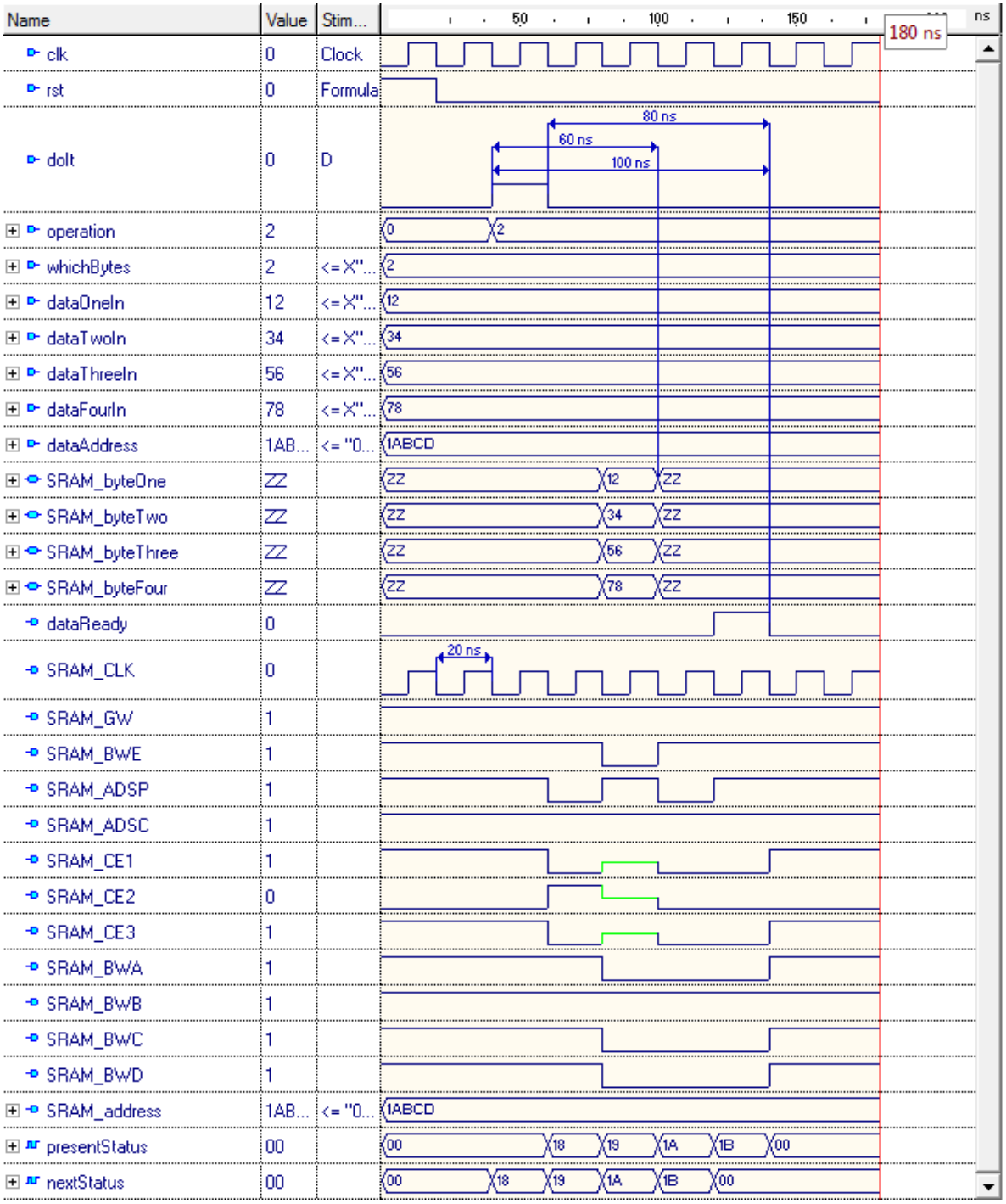

Figura 4.5. Simulación controlador de memoria SSRAM, rutina de escritura.

Los tiempos descritos a continuación son independientes del número de bytes escritos. El tiempo total de operación es de 100 ns, tiempo máximo de retención de datos 60 ns y tiempo total de escritura 80 ns. La tasa total máxima (teórica) de escritura es de 38.1 MB por segundo.

IV.1.3.2 Controlador de memoria Flash

A diferencia de la memoria SSRAM, la memoria Flash requiere de dos rutinas adicionales a la de lectura y escritura. En el caso de la memoria Flash el borrado es una operación controlada internamente y es por sectores; no siendo posible borrar un bit o byte individual. Aunado a ello la memoria Flash opera por medio de comandos, los comandos descritos en el controlador de la memoria Flash son lectura, programación de palabra, borrado de sector y borrado de chip.

De manera adicional, el chip de memoria requiere de tiempo después de una condición de restablecimiento. Dicho tiempo es distinto dependiendo de la condición de operación previa del chip de memoria a tal condición de restablecimiento. El chip de memoria también proporciona un comando de restablecimiento vía software (soft reset).

Todas las operaciones requieren la escritura de un comando o ciclo de comandos, especificados en la Tabla 3.5, el controlador ha sido diseñado para simplificar la operación de la memoria Flash. Análogamente al controlador de la memoria SSRAM, el controlador de la memoria Flash hace uso de 2 señales ejecución de operación y selección de operación, de 1 bit y 3 bits respectivamente.

Los códigos de operación para la señal selección de operación son 0o1 (leer), 0o2 (programar), 0o3 (restablecimiento via software), 0o4 (borrado de sector) y 0o5 (borrado de chip).

Las operaciones programar, borrar sector y borrar chip requieren de al menos dos ciclos de desbloqueo. La rutina de ejecución de las 3 operaciones comparte una parte de la máquina de estados para la ejecución de las instrucciones; esta parte es los primeros dos ciclos de la secuencia de comandos, mostrados en la Tabla 3.5.

Para las operaciones de borrado (de sector y chip) se comparte la secuencia en los ciclos tercero, cuarto y quinto. La reutilización de los ciclos de bus permitió reducir el tamaño de la máquina de estados para el controlador de la memoria Flash.

IV.1.3.2.1 Rutina de restablecimiento maestro (hard-reset)

En caso de que la condición de restablecimiento haya sido ejecutada fuera de la ejecución de un algoritmo embebido del chip de memoria el controlador aplica una señal de restablecimiento al chip de memoria de 500 ns y espera 100 ns antes de activar la bandera que indica la disponibilidad de la memoria flash.

Si la condición de restablecimiento ha sido ejecutada durante la ejecución de algún algoritmo embebido del chip de memoria, adicionalmente se espera que la señal RY/BY# se establezca en 1.

Los tiempos de aplicación de señal de restablecimiento (t_{RP}) y tiempo en alto de restablecer antes de leer (t_{RH}) se basan en las especificaciones del fabricante del chip de memoria y son mostrados en la Tabla 3.4. Se han designado los siguientes tiempos para: t_{RP} = 500 ns y t_{RH} = 100 ns; ambos estando dentro del margen especificado por el fabricante.

En la Figura 4.6 se muestran los resultados obtenidos de una condición de restablecimiento, las cotas en azul muestran los tiempos designados a cada pulso. En el caso de la condición de restablecimiento fuera de algoritmos embebidos la señal RY/BY# siempre estará en 1.

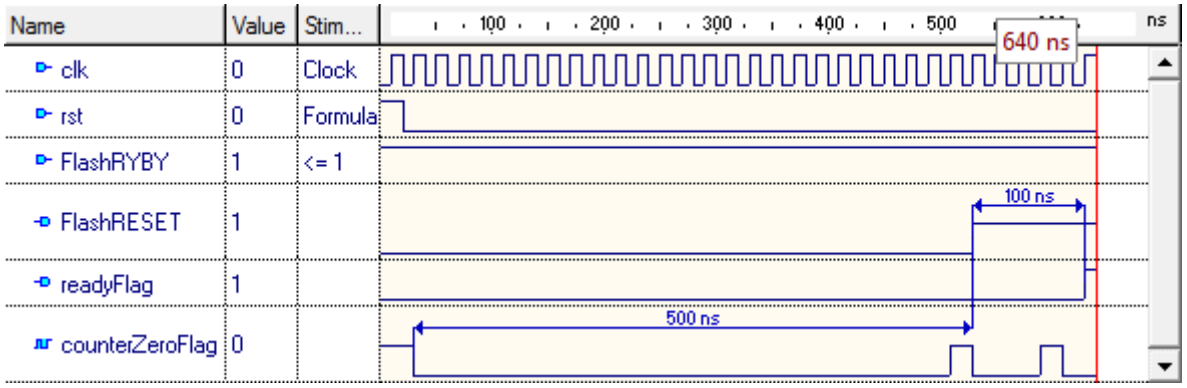

Figura 4.6. Simulación controlador de memoria Flash, rutina de restablecimiento.

IV.1.3.2.2 Rutina de lectura

La rutina de lectura requiere solamente de un ciclo de bus en el que se especifica la localidad de memoria a leer. Tal localidad es especificada por una dirección escrita en el bus de direcciones del chip de memoria. La longitud del bus de direcciones es de 22 bits.

El código utilizado para seleccionar la operación de lectura es 0o1. Antes de activar el bit de ejecución de operación es necesario proporcionar la dirección de lectura. En la Figura 4.7 se muestra el resultado de la simulación. Las cotas azules muestran los tiempos más relevantes.

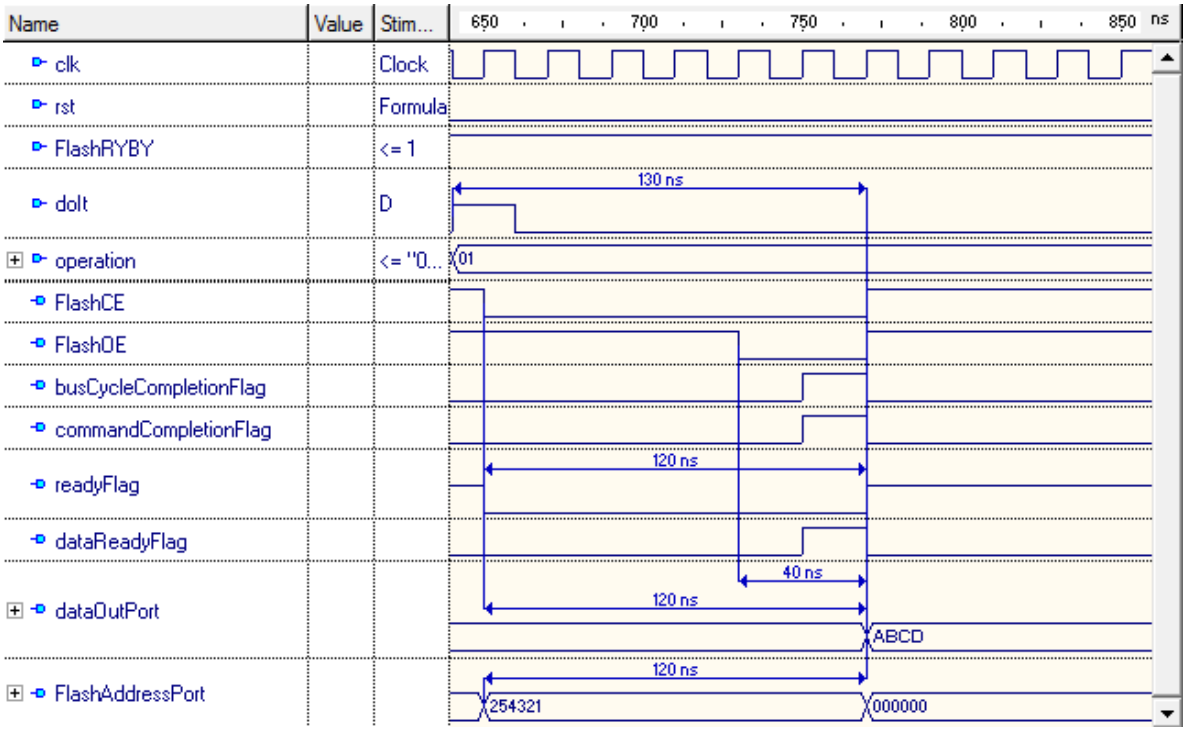

Figura 4.7. Simulación controlador de memoria Flash, rutina de lectura.

Con fines de simulación los buses de datos y direcciones han sido alimentados con señales arbitrarias. Los valores 0x254321 y 0xABCD ambos en notación hexadecimal, son asignados a los puertos de dirección y datos, respectivamente.

Los resultados de la simulación indican los siguientes tiempos para: t_{RC} = 120 ns, t_{ACC} = 120 ns, t_{OE} = 40 ns, t_{OEH} = 90 ns, t_{CE} = 120 ns, t_{OH} = 0 ns. Los tiempos logrados cumplen satisfactoriamente las especificaciones del fabricante mostradas en la Tabla y Figura 3.6.

El tiempo total de lectura es de 130 ns, la tasa total de lectura continua es de 7’692,307 palabras por segundo, o bien 14.6 MB por segundo. La memoria utilizada es de 8 MB, lo que implica que en modo de espera de comando, la Tabla de horarios escanea 1.8 veces la memoria por segundo. De manera teórica esto nos asegura que la memoria será escaneada al menos una vez por segundo y con ello se supone el correcto encendido/apagado de las salidas.

IV.1.3.2.3 Rutina de programación

Los primeros dos ciclos de la rutina de programación, mostrados en la Tabla 3.5, son de desbloqueo. El tercer ciclo indica, a la máquina de estados interna del chip de memoria, que se trata de una operación de programación; mientras el cuarto ciclo escribe la información a programar en los registros del chip de memoria.

Los ciclos de desbloqueo son compartidos con las rutinas de borrado (de sector y chip). El selector de operación debe mantener el código de programación hasta que la bandera de finalización de escritura haya cambiado su nivel a 1. El código utilizado para la operación de programación es 0o2.

En la Figura 4.8 se muestra el tiempo de programación, indicándolo con la cota en azul. El tiempo de programación es medido a partir del cambio de estado del bit de ejecución de operación hasta el cambio de estado de la bandera de finalización de operación de programación. Se ha simulado la respuesta de pin RY/BY# con una valor típico de 60 µs, de acuerdo a las especificaciones del fabricante.

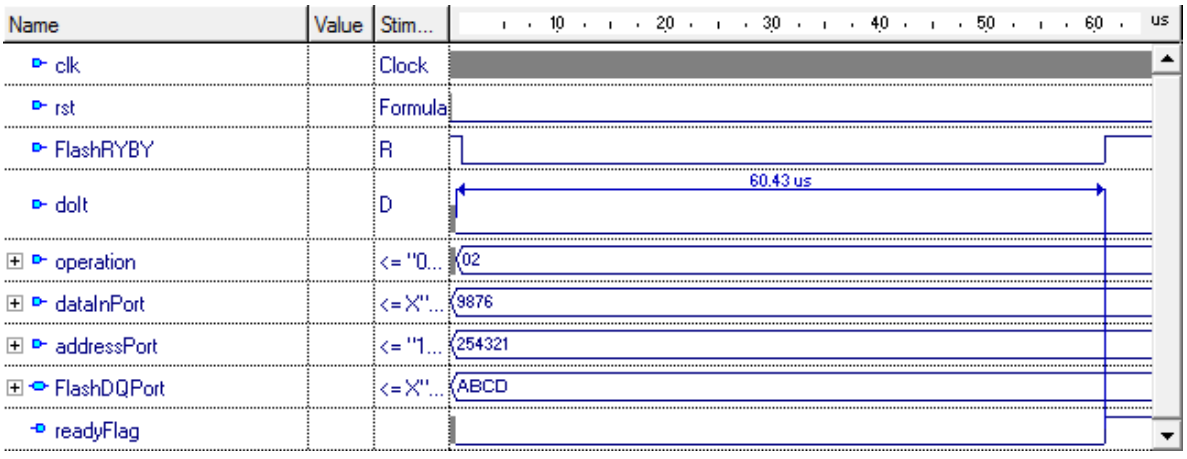

Figura 4.8. Simulación controlador de memoria Flash, rutina de programación – tiempo de operación.

Para la rutina de programación se ha verificado la correcta escritura de los 4 ciclos de bus. En la Figura 4.9 se pueden observar claramente los ciclos de bus en los que se escribe para el puerto de datos 0x00AA, 0x0055, 0x00A0 y 0x9876; el último de ellos siendo el dato a programar. Para el bus de direcciones se escribe 0x000555, 0x0002AA, 0x000555 y 0x254321; en este caso el ultimo siendo la dirección de la localidad de memoria donde se programarán los datos.

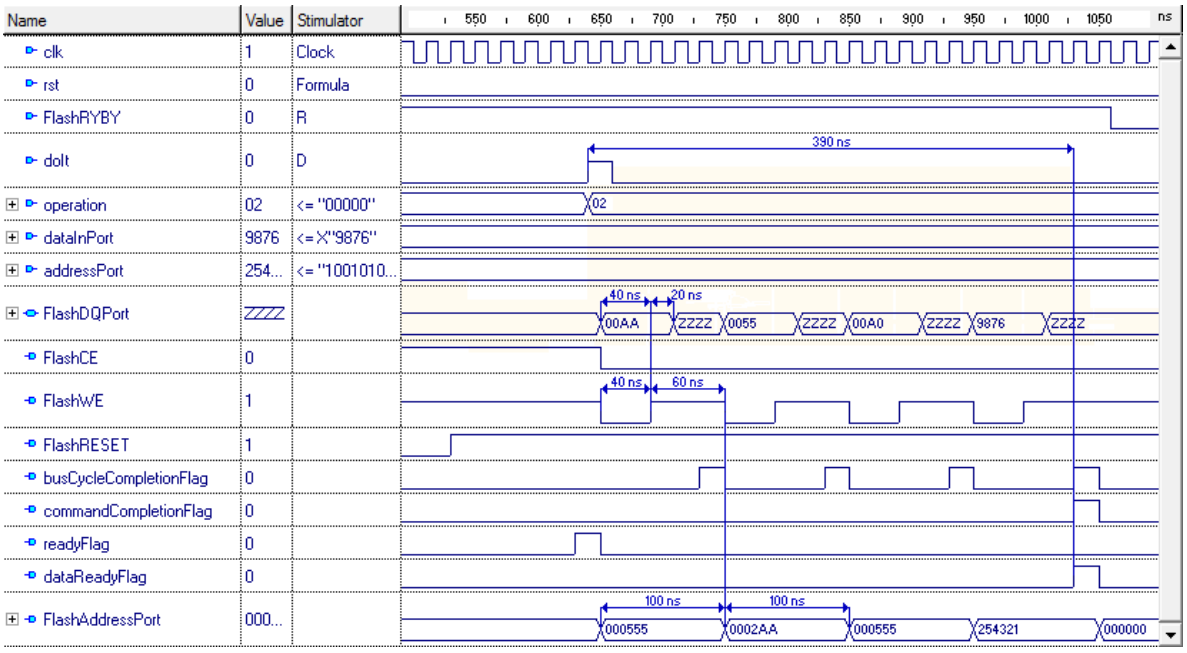

Figura 4.9. Simulación controlador de memoria Flash, rutina de programación.

Los resultados de la simulación indican los siguientes tiempos para: t_{WC} = 100 ns, t_{AS} = 0 ns, t_{AH} = 100 ns, t_{CH} = 0 ns, t_{WP} = 40 ns, t_{WPH} = 60 ns, tCS = 0 ns, tDS = 40 ns, tDH = 20 ns. Los tiempos logrados cumplen satisfactoriamente las especificaciones del fabricante mostradas en la Tabla y Figura 3.7.

El tiempo total de programado es de 60.43 µs (por palabra – 16 bits). La tasa total de programado es de 32.3 KB por segundo. La escritura completa de la memoria (8MB) tiene un tiempo mínimo teórico de programado de 254 segundos. Los tiempos calculados son respecto a una rutina de programado continua, y con respecto al controlador desarrollado y no al desempeño de la memoria. Los tiempos son a partir de la simulación, no han sido probados físicamente.

IV.1.3.2.4 Rutinas de borrado

Debido a que las rutinas de programado, borrado de chip/sector; comparten la misma rutina de ciclos de bus las especificaciones de tiempo no serán analizadas. Las especificaciones de tiempo logradas para la rutina de programado son iguales a las de las rutinas de borrado. En la Figura 4.10 se muestran los resultados de simulación para la rutina de borrado de chip.

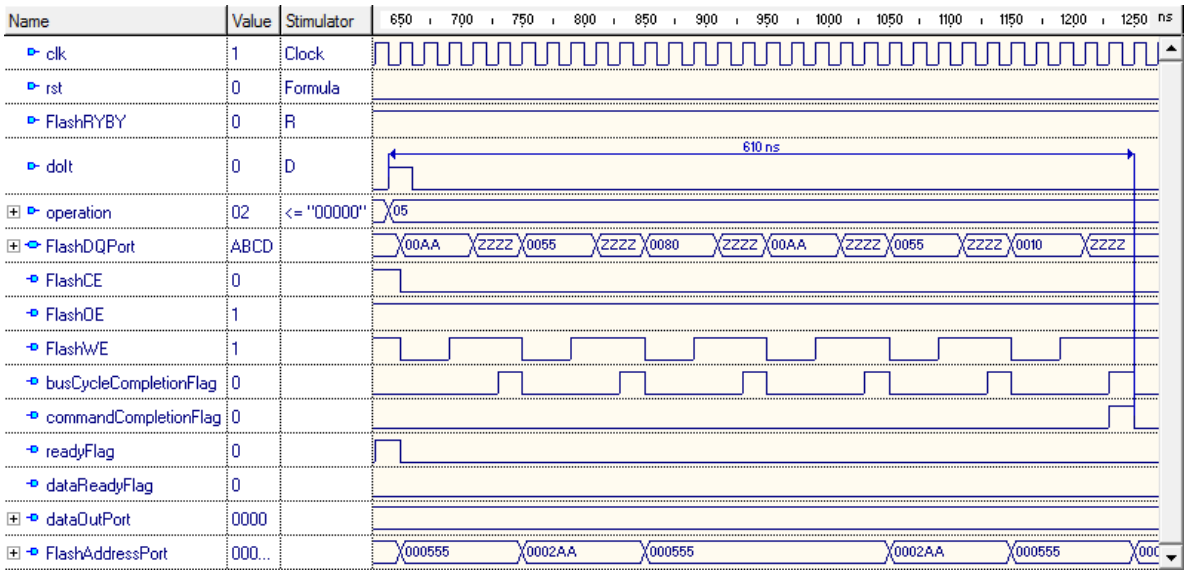

Figura 4.10. Simulación controlador de memoria Flash, rutina borrado de chip.

Tanto la rutina borrado chip como la rutina borrado de sector son iguales en los primeros 5 ciclos, es en el 6° ciclo de bus cuando en el puerto de datos se especifica 0x10 o 0x30 para borrado de chip y borrado de sector; respectivamente. En el caso de la rutina borrado de sector, es en el 6° ciclo de bus cuando se indica la dirección del sector a borrar. En la Figura 4.11 se muestra la simulación de la rutina borrado de sector. Los tiempos acotados son la duración de la ejecución de las rutinas, el tiempo de ejecución dependerá de las condiciones actuales de uso de acuerdo con las Tablas 3.7 y 3.8.

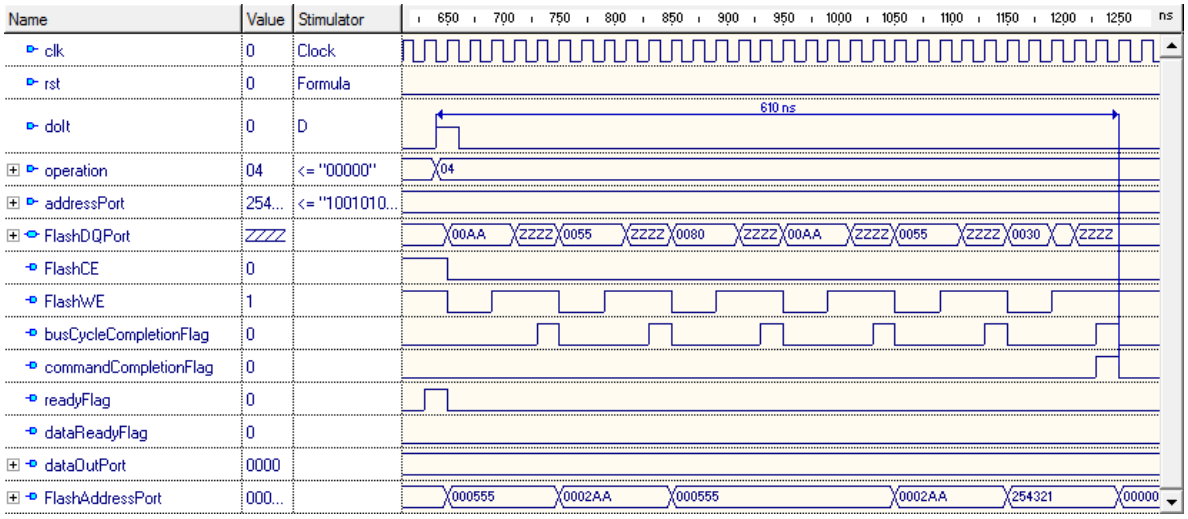

Figura 4.11. Simulación controlador de memoria Flash, rutina borrado de sector.

IV.2 Resultados de respuesta de sensores

Como se ha mostrado en la Figura 3.14, el sensor de temperatura no tiene una respuesta lineal. La etapa de acondicionamiento para este sensor a través del puente de Wheatstone ha mostrado la relación temperatura/voltaje mostrada en la Figura 4.12. La interpolación del comportamiento al rango completo del ADC se muestra en la Figura 4.13.

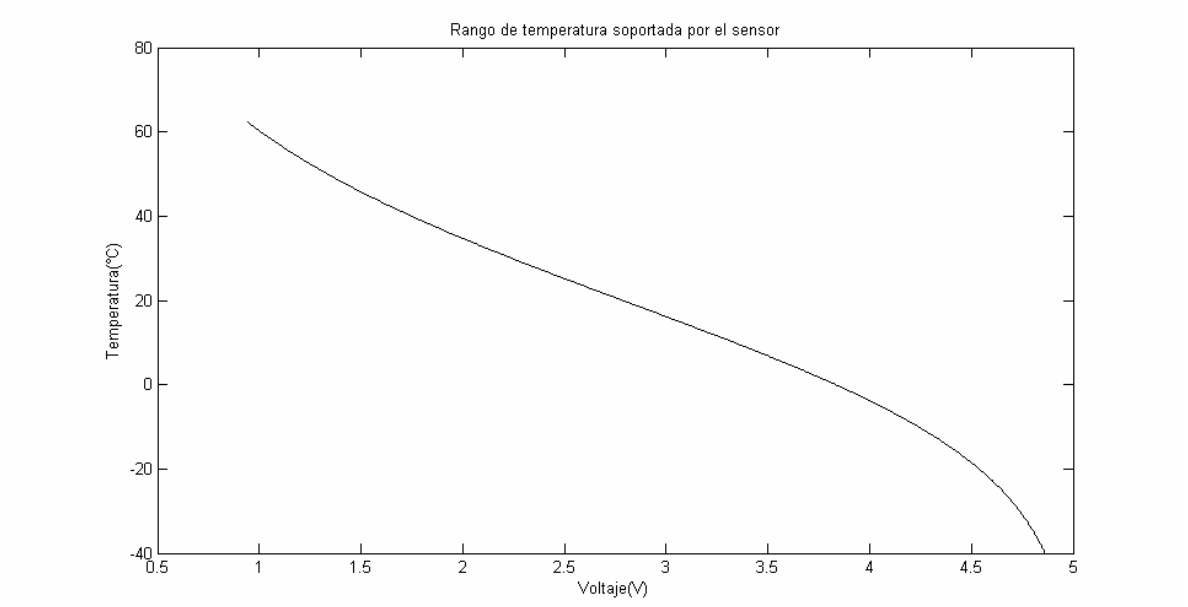

Figura 4.12. Respuesta del sensor en voltaje, después de puente Wheatstone.

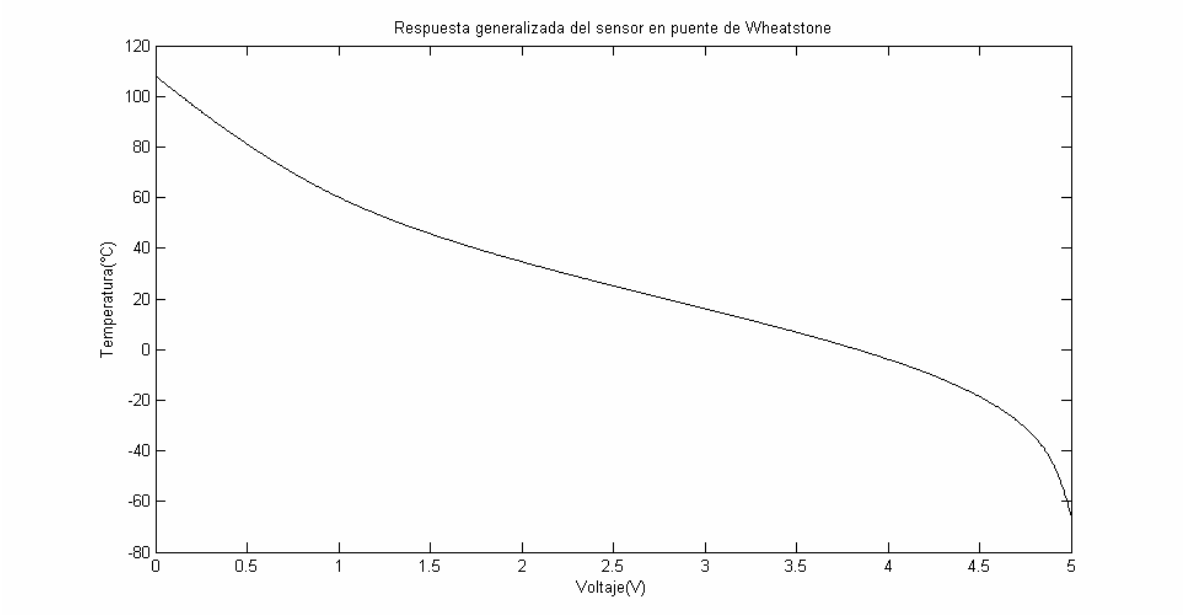

Figura 4.13. Respuesta del sensor en voltaje, ADC rango completo.

La respuesta de la Figura 4.13 ha sido interpolada por medio del método ‘spline’. El uso del rango completo del termistor de temperatura, ha llevado a no tener tanta precisión en el rango utilizado por el controlador 20 – 30. El sensor ha mostrado una precisión aceptable para la entrada del controlador difuso.

IV.3 Respuesta controlador difuso

En la Figura 4.14, se muestra la respuesta del controlador difuso obtenida por Soto-Zarazúa (2010). La respuesta mostrada en la Figura 4.14 incorpora un valor de porcentaje de alimentación para cualquier valor de temperatura y oxígeno disuelto. En el resultado del controlador implementado en la FPGA los valores de temperatura y oxigeno son limitados a una precisión de 0.1. A pesar de la discretización de las entradas el controlador ha mostrado una respuesta bastante favorable.

La correlación del controlador difuso propuesto y el implementado en FPGA es de R = 1.0. Sin embargo, la respuesta del controlador del FPGA no deja

de ser discreto y de precisión fija, en la Figura 4.15 se muestra la respuesta del controlador implementado en FPGA.

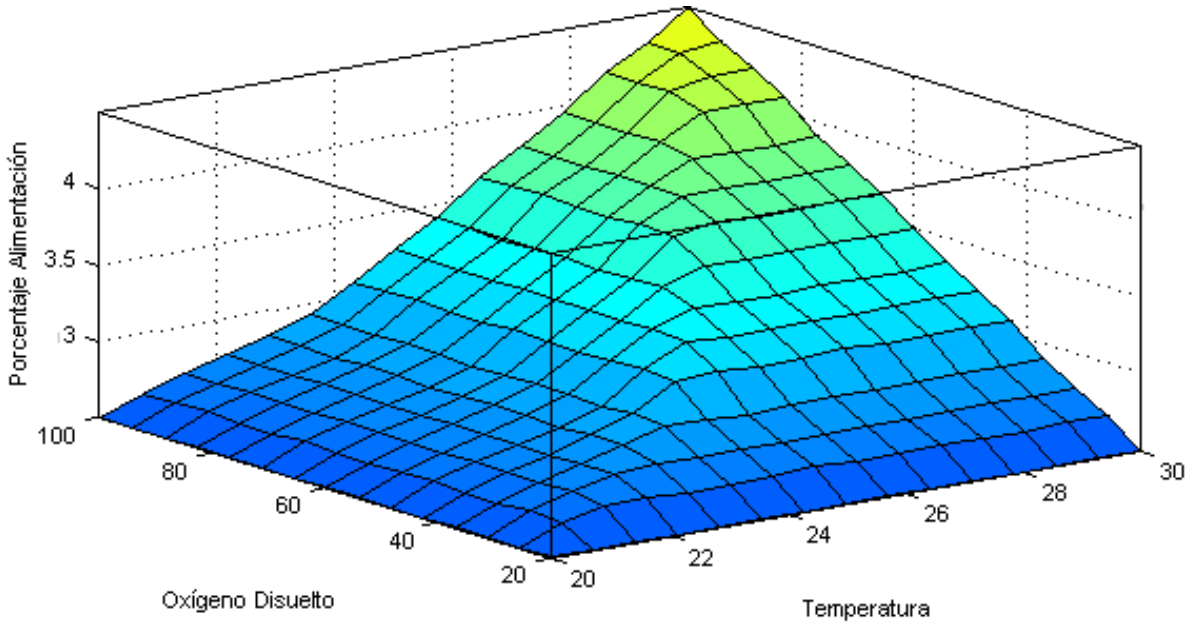

Figura 4.14. Respuesta del controlador propuesto por Soto-Zarazúa (2010).

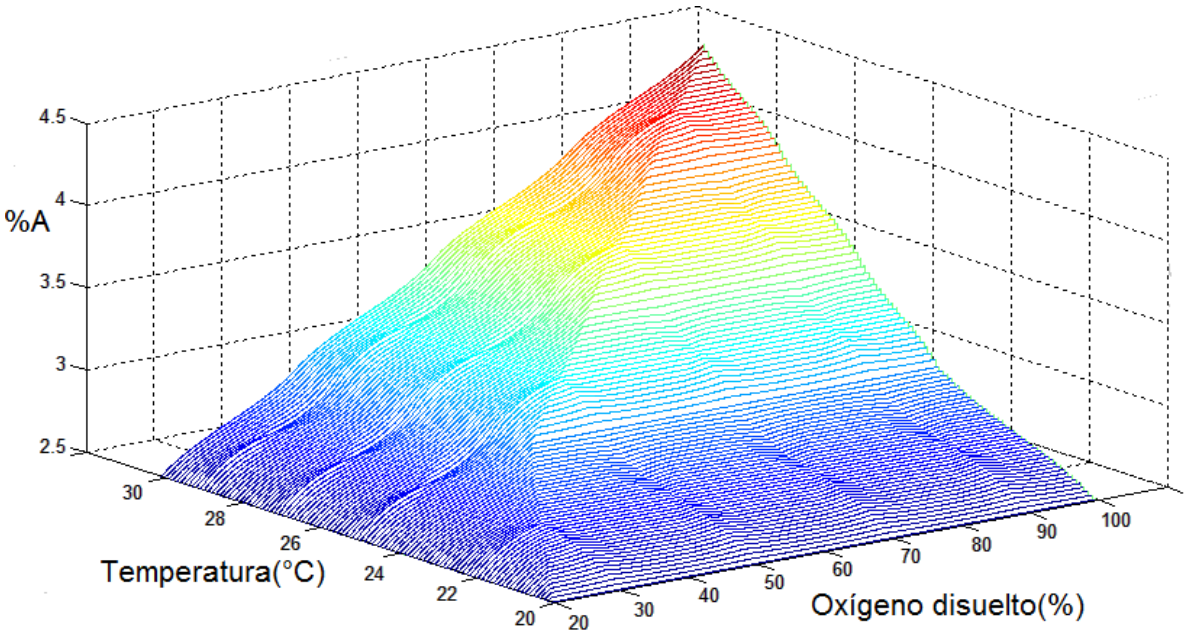

Figura 4.15. Respuesta del controlador implementado en FPGA.

LITERATURA CITADA

Axelson, J. 2007. Serial Port Complete: COM Ports, USB Virtual COM Ports, and Ports for Embedded Systems.

Brown, S. y Z. Vranesic. 2005. Fundamentals of Digital Logic with VHDL Design. Mc Graw Hill. Department of Electrical and Computer Engineering, University of Toronto.

Carlisle R. 2004. Scientific American Inventions and Discoveries. John Wiley & Sons, Inc. Hoboken, New Jersey. 342 p.

Chemali, H., A. Khellaf, A. Benhamadouche and A. Ballouti. 2006. A Fast Testable Digital Fuzzy Logic Controller. Asian J. Inform. Tech. 5:1464-1469.

Davis Instruments. 2007. External Temperature Sensor 7817 Specs Datasheet.

Islam S., N. Amin, M.S. Bhuyan, M. Zaman, B. Madon, M. Othman. 2007. FPGA Realization of Fuzzy Temperature Controller for Industrial Application. WSEAS TRANSACTIONS on SYSTEMS and CONTROL. Issue 10, Volume 2, 484-290.

Maxfield, C. 2004. The Design Warrior's Guide to FPGAs, Newnes, ISBN 0-7506-7604-3.

Masmoudi, M. S., W. Tsui, I. Song, F. Karray, M. Masmoudi, and N. Derbel. 2008. Implementation of a Real-Time FPGA-Based Intelligent Parallel Parking System, J. Adv. Comput. Intell. Intell. Informat. 12:348.

Mermoud, G., A. Upegui, C. A. Peña, E. Sanchez. 2005. A Dynamically-Reconfigurable FPGA Platform for Evolving Fuzzy Systems.

Mohammed, Y. A. y L. K. Hashim. 2007. Implementing Fuzzy Logic Controller Using VHDL. Engineering & Technology. Vol.25 No.9, 1049-1055.

Monroe, G. S. 2007. Robust Fuzzy Controllers Using FPGAs. NASA LaRC.

Programa Maestro Nacional de Tilapia, 2007, Semarnat.

Reznik, L. 1997. Fuzzy Controllers. Newnes. Victoria University of Technology, Melbourne, Australia. 205 p.

Skahill, K. 1996. VHDL for Programmable Logic, Addison-Wesley, ISBN 0-201-89573-0.

Smith, M. 1999. Application Specific Integrated Circuits, Addison-Wesley, ISBN 0-201-50022-1.

Soto-Zarazúa, G.M., E. Rico-García, R. Ocampo, R. G. Guevara-González, G. Herrera-Ruiz, in press. Fuzzy-logic-based feeder system for intensive tilapia production (Oreochromis niloticus). Aquacult. Int. DOI 10.1007/s10499-009-9251-9.

Stonham, T. J. 1988. Digital Logic Techniques: Principles and practice. Segunda edición, Van Nostrand Reinhold, UK, ISBN 0-278-00011-8.

The Institute of Electrical and Electronics Engineers, IEEE Standard 1364-2001, Verilog® Hardware Description Language, IEEE, USA, http://www.ieee.org.

The Institute of Electrical and Electronics Engineers, IEEE Standard 1076-2002, VHDL Language Reference Manual, IEEE, USA, http://www.ieee.org.

Tocci, R. J., N. S. Widmer, G. L. K. Moss. 2004. Digital Systems. Ninth Edition, Pearson Education International, USA, ISBN 0-13-121931-6.

Wilson, P. R. 2007. Design Recipes for FPGAs. Newnes, Elsevier. ISBN: 978-0-7506-6845-3

APENDICE