

**Metodologías de Diseño para Sistemas Embebidos con
Aplicaciones Industriales**
Código: 808187 / 20201003404,

INFORME TECNICO

Carlos Iván Camargo Bareño

Departamento de Ingeniería Eléctrica y Electrónica
Universidad Nacional de Colombia

23 de agosto de 2005

INFORME FINAL

Metodologías de Diseño para Sistemas Embebidos con Aplicaciones Industriales

AUTHOR: C. Camargo

E-MAIL: cicamargoba@unal.edu.co

Copyright ©2004, 2005 Universidad Nacional de Colombia

<http://www.unal.edu.co>

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, version 1.2, with no invariant sections, no front-cover texts, and no back-cover texts. A copy of the license is included in the end.

This document is distributed in the hope that it will be useful, but without any warranty; without even the implied warranty of merchantability or fitness for a particular purpose.

Published by the Universidad Nacional de Colombia

Índice general

1. Sinopsis Divulgativa	1
2. Resumen Técnico	3
2.1. Introducción	3
2.2. Sistemas Embebidos	3
2.3. Metodología de Diseño Típica para Sistemas Embebidos	4
2.4. Arquitectura de un Sistema Embebido	5
2.5. Flujo de Diseño Software y Hardware	5
2.5.1. Herramientas de Desarrollo para Hardware	5
2.5.2. Herramientas de desarrollo Software	6
3. Cuadros de Resultados	7
3.1. Objetivo General	7
3.1.1. Resultado Esperado	7
3.1.2. Resultados Obtenidos	7
3.2. Objetivos Específicos	8
3.2.1. Resultado esperado	8
3.2.2. Resultados Obtenidos	8
4. Impacto del Proyecto	11
4.1. Impacto a nivel Académico	11
4.2. Generación de Conocimiento	11
4.3. Impacto a nivel Industrial	11
4.4. Fortalecimiento de la Investigación	12

5. Conclusiones	13
A. Tarjeta de Desarrollo para el SoC AT91	15
B. Tarjeta de Desarrollo para el GBA de Nintendo	17
C. Tarjeta de Desarrollo para el SoC LH79520 de Sharp	19
D. Tarjeta de Desarrollo el CPLD XC9572XL	21
E. Equipo de Laboratorio basado en el GBA de Nintendo	23
F. Horno de Reflujo Para Montaje de ICs de Montaje Superficial	25
G. Implementación de Sistemas Digitales Complejos Utilizando Sistemas Embebidos	27
H. Control Adaptativo Embebido	39
I. Metodología de Implementación de Autómatas Celulares en FPGAs	47

Capítulo 1

Sinopsis Divulgativa

El campo de aplicación de los sistemas digitales se expande de una manera muy rápida, en la actualidad los seres humanos se ven en la necesidad de interactuar con decenas de estos dispositivos, los cuales, son parte de aparatos, herramientas o vestuario de uso diario. Esta revolución digital ha sido originada por los avances de la industria de los Semiconductores, la cual produce dispositivos cada vez más poderosos, económicos y veloces. Por otro lado, el desarrollo de herramientas Software de libre distribución (compiladores, sistemas operativos de tiempo real, depuradores, simuladores) permiten el desarrollo de aplicaciones de forma rápida y económica.

Desafortunadamente esta no es la situación Colombiana, lo cual se ha convertido en un consumidor de tecnología, mucha de la cual podría ser diseñada localmente. Uno de los responsables de esta situación es la industria, que no creen en la capacidad de diseño de los profesionales Colombianos, y reduce las labores de los centros de Investigación y Desarrollo (industriales) a una búsqueda y compra por Internet. Compartiendo esta responsabilidad se encuentran las Universidades, las cuales continúan con programas académicos desactualizados que no reflejan la realidad de la industria electrónica actual.

El objetivo principal de este proyecto es el estudio de Metodologías de diseño que permitan utilizar los avances tecnológicos actuales para dar soluciones a problemas industriales locales. Adicionalmente el resultado de este estudio debe reflejarse en el contenido de los cursos del área de los Sistemas Digitales.

Capítulo 2

Resumen Técnico

2.1. Introducción

Con el pasar del tiempo los sistemas digitales van creciendo en complejidad, así como las exigencias de desempeño y funcionamiento. La incursión de los sistemas digitales en aplicaciones vitales, en las cuales un error resultaría en la pérdida de vidas humanas exige que estos sistemas sean muy confiables (tolerantes a fallos) y presenten un desempeño muy alto. El desarrollo de procesadores de 32, 64 bits de bajo costo unido a la disponibilidad de herramientas software como librerías, drivers y sistemas operativos de tiempo real hace que la realización de este tipo de sistemas no sea una tarea imposible. Este proyecto suministra los primeros pasos en el estudio de los sistemas embebidos (SE) y su intención es mostrar la potencialidad de estos como solución a problemas industriales.

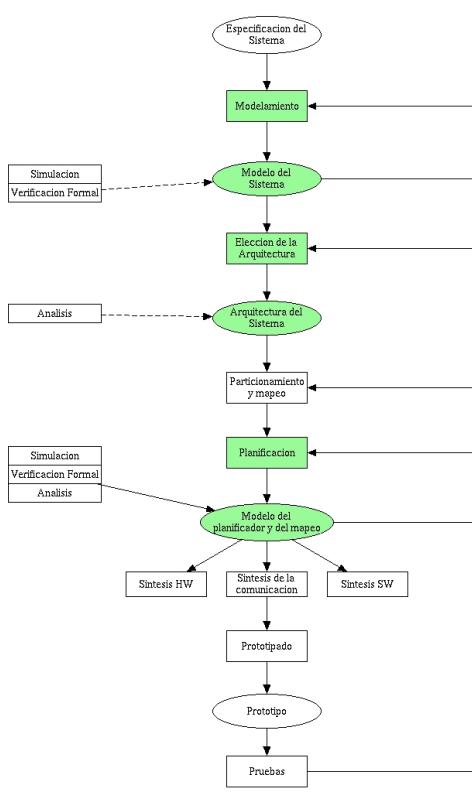
2.2. Sistemas Embebidos

Un sistema embebido (SE) es una máquina computacional que emplea una combinación de Hardware y Software para realizar una función específica. Es parte de un sistema más grande que puede no ser un computador y trabaja en un ambiente reactivo con restricciones temporales. En este tipo de sistemas, el software se utiliza para proporcionar flexibilidad y funcionalidad, mientras que el Hardware (Procesadores, ASICs, Memorias...) es utilizado para proporcionar desempeño y en algunas ocasiones seguridad. Las principales Características de un sistema Embebido son:

- Realiza una función única o un pequeño grupo de funciones relacionadas.
- De alto desempeño y con fuertes restricciones temporales
- Sistemas reactivos con características de tiempo real.
- Sistemas heterogéneos (Con componentes SW y HW).

2.3. Metodología de Diseño Típica para Sistemas Embebidos

La siguiente figura resume los pasos de una metodología de Diseño clásica para Sistemas Embebidos, la cual se siguió en el estudio realizado por este proyecto:

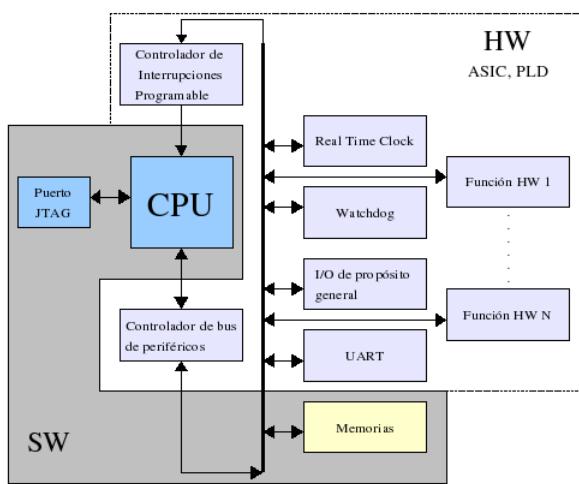


- Especificación y Modelamiento:** Descripción algorítmica del sistema. Por lo general estos algoritmos son validados en una descripción ejecutable, es decir, un programa escrito en un lenguaje de alto nivel1 que modela el comportamiento del sistema, en el cual se pueden validar de forma fácil y rápida dichos algoritmos.
- Particionamiento HW-SW:** Implementación de las unidades funcionales en SW (ejecutándose en un procesador) o en HW (implementado en un PLD). Debido a que los procesadores ejecutan instrucciones de forma secuencial, si la ejecución de una función compromete el cumplimiento de las restricciones temporales (la función emplea muchos ciclos de máquina) esta debe ser implementada en HW con el fin de aprovechar sus características concurrentes La partición óptima es aquella que cumple con las restricciones del diseño a costo mínimo

- Planificación:** En este paso se decide el tiempo y el orden en el que se ejecutan las funciones. Esto es importante cuando se comparten recursos Hardware. Los Sistemas Operativos de Tiempo Real (RTOS) proporcionan mecanismos que permiten administrar los recursos de forma eficiente.
- Síntesis:** Proceso de llevar una especificación de un nivel de abstracción alto (común en las primeras etapas del flujo de Diseño) a uno bajo. Este paso es necesario para poder llegar a la implementación física la cual es el nivel de abstracción más bajo.
- Prototipado y Pruebas:** Implementación física del sistema y pruebas. La importancia de las pruebas al nivel física radica en que durante las anteriores etapas de diseño se trabaja con simulaciones, en las cuales se modelan los dispositivos reales.

2.4. Arquitectura de un Sistema Embebido

Como se mencionó anteriormente un Sistema Embebido es un sistema heterogéneo compuesto por componentes SW y HW, lo cual da posibilidades de diferentes implementaciones:



1. Procesador - FPGA: Existen dos Circuitos Integrados (ICs): Un microcontrolador y una FPGA, este tipo de estructura es común en sistemas que requieren gran desempeño.
2. Sistem on Chip (SoC): El componente HW se encuentra implementado en el mismo IC que el procesador.
3. FPGA: El procesador es implementado en una FPGA junto con los componentes HW.

Implementación	Ventajas	Desventajas
FPGA - uP	Flexibilidad: Se puede implementar cualquier función HW en la FPGA	La interconexión reduce la máxima velocidad de operación. Costo Elevado
SoC	Económico: Elimina la necesidad de implementar las funciones HW ya que se encuentran en el SoC.	Funciones HW limitadas.
FPGA	Se tiene control total sobre los recursos HW. Económico	El procesador es sintetizado, por lo que existen mayores retardos asociados a las interconexiones que en el procesador normal.

2.5. Flujo de Diseño Software y Hardware

En esta sección se resumirán los resultados obtenidos en el estudio de herramientas de desarrollo para funciones Hardware y Software.

2.5.1. Herramientas de Desarrollo para Hardware

Después de un estudio que explora características como precio de los dispositivos, disponibilidad de herramientas de libre distribución y capacidad de las mismas, se llegó a la conclusión que la mejor opción es la utilización de PLDs de Xilinx.

Por otro lado, el Departamento de Ingeniería Eléctrica y Electrónica ingresó al programa universitario de Xilinx <http://www.xilinx.com/univ>. A través de este programa Xilinx donó a la Universidad una serie de herramientas Hardware y Software que están siendo utilizadas en los cursos de Sistemas Digitales y en la realización de trabajos de grado del Departamento.

2.5.2. Herramientas de desarrollo Software

Existen innumerables herramientas comerciales que permiten el fácil desarrollo de aplicaciones, sin embargo, estas son muy costosas y soportan un determinado tipo de dispositivos. Debido a la limitación de recursos económicos de las Universidades públicas y de las medianas y pequeñas industrias, la mejor opción es la utilización de herramientas de libre distribución. El proyecto GNU¹ (GNU is Not Unix) tiene como objetivo el desarrollo de un sistema operativo Unix de libre distribución, en la actualidad variaciones de este sistema operativo que utiliza el kernel de linux es utilizado en una gran variedad de aplicaciones. Además de la gran ventaja de ser un software gratis, existen miles de desarrolladores explorando, depurando y optimizando el código y en los foros de discusión se obtiene soporte gratuito. Las principales herramientas GNU utilizadas en el desarrollo de Sistemas Embebidos son:

1. Cadena de Herramientas (Toolchain):

- Compilador de C, C++ (GCC): La utilización del lenguaje ensamblador como herramienta de desarrollo ha disminuido de forma considerable, debido a la falta de portabilidad. En la actualidad se utilizan librerías escritas en C y C++, lo cual permite reducir el tiempo de diseño. GCC permite ser cross-compilado² para una gran cantidad de Procesadores entre los que se encuentran: ARM, ALPHA, MIPS, MOTOROLA, PPC, SH SPARC, INTEL x86. Lo cual permite migrar funciones o librerías entre arquitecturas de forma rápida.
- Librerías (GLIBC): Soporte de funciones básicas del sistema operativo.
- Utilidades Binarias (Binutils): Utilidades para manipular los archivos ejecutables obtenidos del proceso de compilación.
- Depurador (GDB): Herramienta que permite depurar el código y detectar errores.

2. Sistema Operativo: Sin lugar a dudas el sistema operativo que se debe utilizar para aplicaciones en Sistemas Embebidos es Linux. El cual soporta una gran cantidad de dispositivos Hardware como: Controladores de video, sonido, USB, buses, seriales, etc. Sin embargo, en este proyecto también se realizó el estudio de el sistema operativo eCos³, el cual debido a su reducido tamaño permite ser utilizado en sistemas con restricciones de memoria.

¹<http://www.gnu.org>

²Compilador que produce código para una arquitectura diferente a en la que se está ejecutando

³<http://sources.redhat.com/ecos>

Capítulo 3

Cuadros de Resultados

3.1. Objetivo General

Desarrollar un metodología para el diseño de Sistemas Embebidos basada en un arquitectura *software-hardware* que permita implementar de forma fácil y rápida sistemas digitales complejos dedicados a aplicaciones industriales.

3.1.1. Resultado Esperado

Investigar, entender y apropiar las técnicas de diseño modernas para aplicarlas al entorno académico e industrial Colombiano.

3.1.2. Resultados Obtenidos

- Se logró profundizar en el estudio de las técnicas modernas de desarrollo de sistemas digitales.
- Se diseñaron y construyeron herramientas de desarrollo que fueron utilizadas en la realización de proyectos de grado del Departamento de Ingeniería Eléctrica y Electrónica.
- Colaboración con el **Centro Internacional de Física** para el desarrollo del proyecto CIF - IDU "BMS"(Bridge Monitoring System).
- Creación de 4 tesis de pregrado en el Departamento de Ingeniería Eléctrica y Electrónica.
- Diseño e implementación de aplicaciones que permitieron validar la metodología.
- Publicación de tres artículos en eventos internacionales.
- Modificación del programa de las materias del área de sistemas digitales para actualizarlas al estado del arte.

3.2. Objetivos Específicos

1. Formular una metodología para el diseño e implementación de sistemas embebidos.
2. Generar herramientas software y hardware (plataforma de desarrollo) que faciliten la implementación de aplicaciones industriales sobre sistemas embebidos.
3. Generar con base en la metodología desarrollada equipo de laboratorio que sea susceptible a utilizar sistemas embebidos. Dentro de dicho equipo se apuntará a realizar alguno(s) de los siguientes ítems.
 - Equipo básico de laboratorio: multímetro, osciloscopio.
 - Equipo para el Laboratorio de Control del DIEE: Dispositivos capaces de captar la información proveniente de un sensor/actuador, procesarla e implementar estrategias de supervisión y control.
4. Creación de un curso de sistemas embebidos para los programas de post-grado y pregrado pertenecientes al Departamento de Ingeniería Eléctrica y Electrónica.

3.2.1. Resultado esperado

Los objetivos específicos anteriormente citados ayudan al cumplimiento del objetivo general. Estos objetivos están dirigidos a la implementación física de dispositivos siguiendo una metodología específica.

3.2.2. Resultados Obtenidos

A continuación se listan los resultados obtenidos durante el desarrollo de los objetivos específicos.

Metodología de Diseño

La metodología de diseño propuesta se basa en metodologías de diseño existentes, y es presentada en el Capítulo 2 de este informe.

Herramientas de Desarrollo Software y Hardware

- Diseño de Sistemas Embebidos con el ARM7TDMI¹ y eCos SoC. (Ver Apéndice A)
- Diseño de un Sistema de Desarrollo para ser utilizado con el GameBoy Advance de Nintendo². (Ver Apéndice B)

¹<http://www.atmel.com>

²<http://www.nintendo.com/systemsgba>

- Diseño de un Sistema de Desarrollo del LH79520 de Sharp³. (**Ver Apéndice C**)
- Diseño de un sistema de Desarrollo para CPLDs, para ser utilizado en el curso de electrónica Digital I del Departamento de Ingeniería Eléctrica y Electrónica. (**Ver Apéndice D**)

Aplicaciones

Para validar la metodología, el software y las plataformas de Desarrollo, se realizaron las siguientes aplicaciones:

- Diseño de un Osciloscopio, Multímetro y Analizador lógico sobre el GBA. (**Ver Apéndice E**)
- Diseño de un Horno de reflujo para Montaje de ICs de montaje superficial. (**Ver Apéndice F**)
- Linux Embebido sobre FPGAs.
- Cooperación con el *Centro Internacional de Física* en la realización del proyecto CIF - IDU "BMS"(Bridge Monitoring System).

Artículos

Se realizaron las siguientes publicaciones:

- Título: Implementación de Sistemas Digitales Complejos Utilizando Sistemas Embebidos.
Publicado en: Memorias del XI Workshop de Iberchip ISBN 959-261-105-X.
Año: 2005. (**Ver Apéndice G**)
- Título: Control Adaptativo Embebido
Publicado en: Memorias del XI workshop de Iberchip ISBN 959-261-105-X
Año = 2005. (**Ver Apéndice H**)
- Título: Metodología de Implementación de Autómatas Celulares en FPGAs.
Publicado en: Memorias del X Workshop de Iberchip ISBN 959-261-105-X.
Año: 2004 (**Ver Apéndice I**)

Proyectos de Grado Generados por el proyecto

En la realización de las tareas mencionadas anteriormente participaron estudiantes del programa de Ingeniería Electrónica, con la realización de los Siguientes trabajos de Grado:

³<http://www.sharpsama.com>

- D. Mendez, D. Ovalle, *Diseño e Implementación de equipo básico de laboratorio utilizando Sistemas Embebidos y FPAAs*
- M. Poveda, E. Poveda, *Sistema de Soldadura Para Montaje Superficial*
- M. Sepúlveda, S. Bolivar, *Metodología de Implementación de Autómatas Celulares en FPGAs*
- F. Pedraza, *Control Adaptativo Embebido*

Creación de un curso de Sistemas Embebidos

Este Objetivo específico está en proceso de desarrollo, inicialmente se deseaba crear una línea de investigación en el área y ofrecer una materia electiva. Sin embargo, el director de este proyecto: Carlos Iván Camargo Bareño, quien dictaría esta electiva, se encuentra actualmente en comisión de estudios, realizando sus estudios de doctorado. Razón por la cual, no se ha ofrecido la asignatura en la actualidad.

Capítulo 4

Impacto del Proyecto

4.1. Impacto a nivel Académico

Con la realización de este proyecto se logró cuestionar sobre la actualidad de los programas de las materias del área de Sistemas Digitales del Departamento de Ingeniería Eléctrica y Electrónica. Se encontró que el programa de Electrónica Digital I se encuentra muy desactualizado y su contenido no ha variado mucho en los últimos 10 años, lo cual no refleja los grandes avances obtenidos en esta área a nivel mundial. Para solucionar este problema se diseño y construyó una tarjeta de desarrollo económica que permitirá a los estudiantes apropiar nuevas metodologías de diseño.

Por otro lado, se descubrió que los estudiantes de los programas de Ingeniería Eléctrica y Electrónica tienen dificultades en el momento de realizar algoritmos que den solución a problemas reales. Esto llevó a la re-estructuración de los cursos de programación en estos programas.

4.2. Generación de Conocimiento

Gracias a la realización de este proyecto se trabajó con tecnologías de punta y se comenzó el estudio en temas como: Sistemas Operativos, Tiempo Real, Metodologías de Diseño, Co-Diseño, interfaces HW-SW, internet Embebido, intenet como medio medio de comunicación y control de procesos industriales, programación a alto nivel.

4.3. Impacto a nivel Industrial

Como resultado de la aplicación de los conocimientos generados en este proyecto en el entorno académico, se obtendrá a futuro un Ingeniero Electrónico que sea capaz de diseñar Sistemas Digitales de forma rápida y económica, lo cual generará la creación de nuevas líneas de trabajo en las industrias Colombianas y se reducirá la dependencia tecnológica actual.

4.4. Fortalecimiento de la Investigación

El estudio realizado en este proyecto continuará en la tesis de Doctorado de su director, el profesor Carlos Iván Camargo Bareño, quien, en la actualidad se encuentra desarrollando estudios en Hardware Reconfigurable y Hardware Bio-inspirado. Por lo tanto, este proyecto generó la realización de forma directa cuatro tesis de pregrado y una tesis de doctorado.

Adicionalmente el Departamento de Ingeniería Eléctrica y Electrónica ingresó al programa Universitario de Xilinx. Los beneficios obtenidos al ingresar a este programa son: Donación de herramientas de desarrollo y Donación de Software. En la actualidad el curso de Sistemas Digitales II utiliza software donado por Xilinx, la cual garantiza la actualización gratuita de dichas herramientas, lo cual se convierte en una herramienta muy valiosa para la enseñanza de nuevas tecnologías.

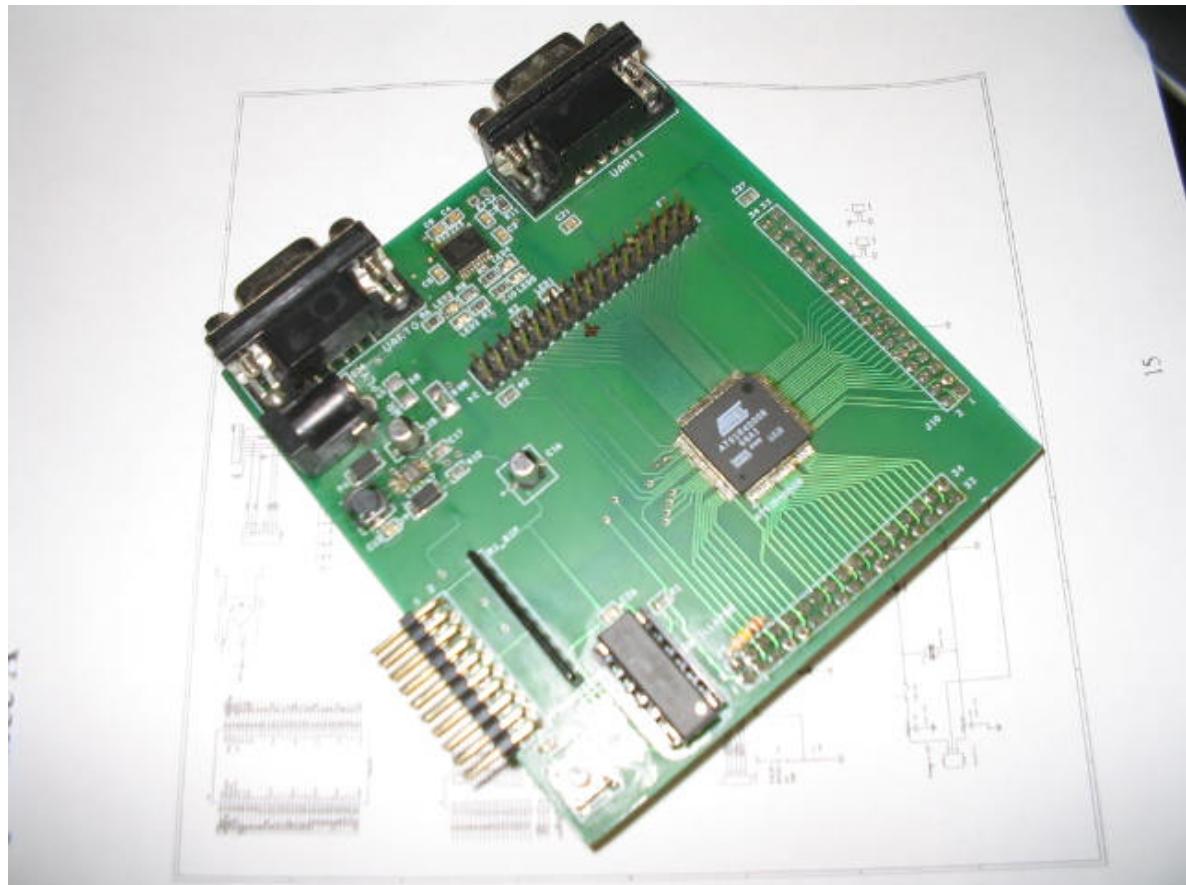
Capítulo 5

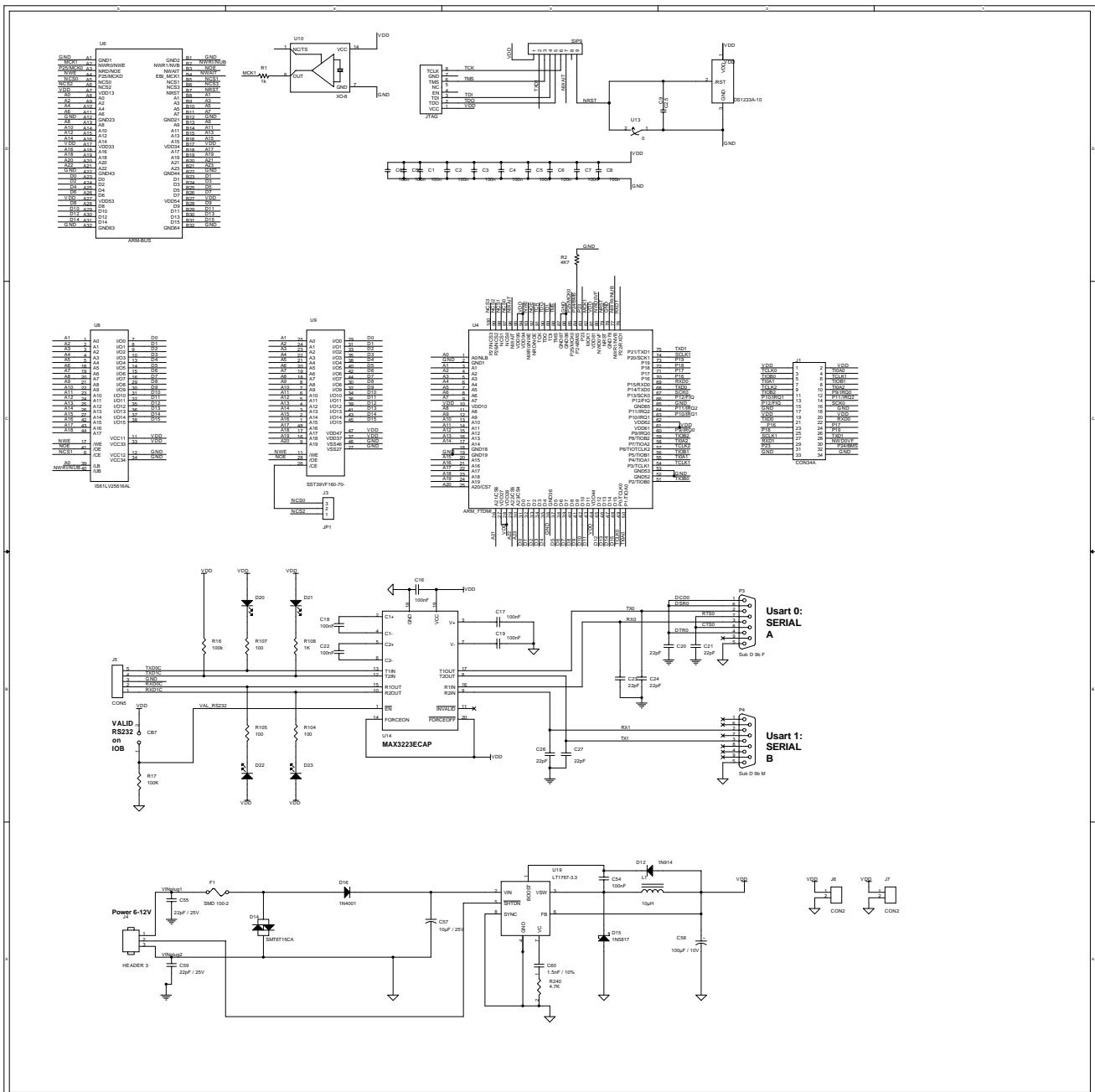
Conclusiones

- Con la realización de este proyecto se exploraron nuevas metodologías de diseño de sistemas Digitales.
- El área de los sistemas digitales es un área en continuo desarrollo y su campo de acción crece muy rápidamente.
- Los programas de las asignaturas del área de Sistemas Digitales del Departamento de Ingeniería Eléctrica y Electrónica no reflejan el estado actual de la tecnología digital.
- Los avances en la industria de los semiconductores permiten obtener dispositivos cada vez más poderosos, veloces y económicos
- El desarrollo de software de libre distribución permite el desarrollo de aplicaciones embebidas de bajo costo.
- El sistema operativo Linux se perfila como el RTOS dominante en la implementación de aplicaciones de Sistemas Embebidos.
- Es necesario replantear el contenido de las asignaturas de programación de computadores, ya que se detectan fallas en el pensamiento algorítmico de los estudiantes.
- La tendencia del diseño de sistemas digitales requiere que los ingenieros Electrónicos dominen los lenguajes de programación de alto nivel como el C++, Java, XML.
- Es necesaria la creación de una o dos asignaturas de sistemas embebidos donde se enseñe la arquitectura de los Sistemas Operativos Unix.
- La metodología de diseño seguida en la realización de las aplicaciones de este proyecto permite a las industrias dar solución a problemas de automatización de forma económica.

Apéndice A

Tarjeta de Desarrollo para el SoC AT91

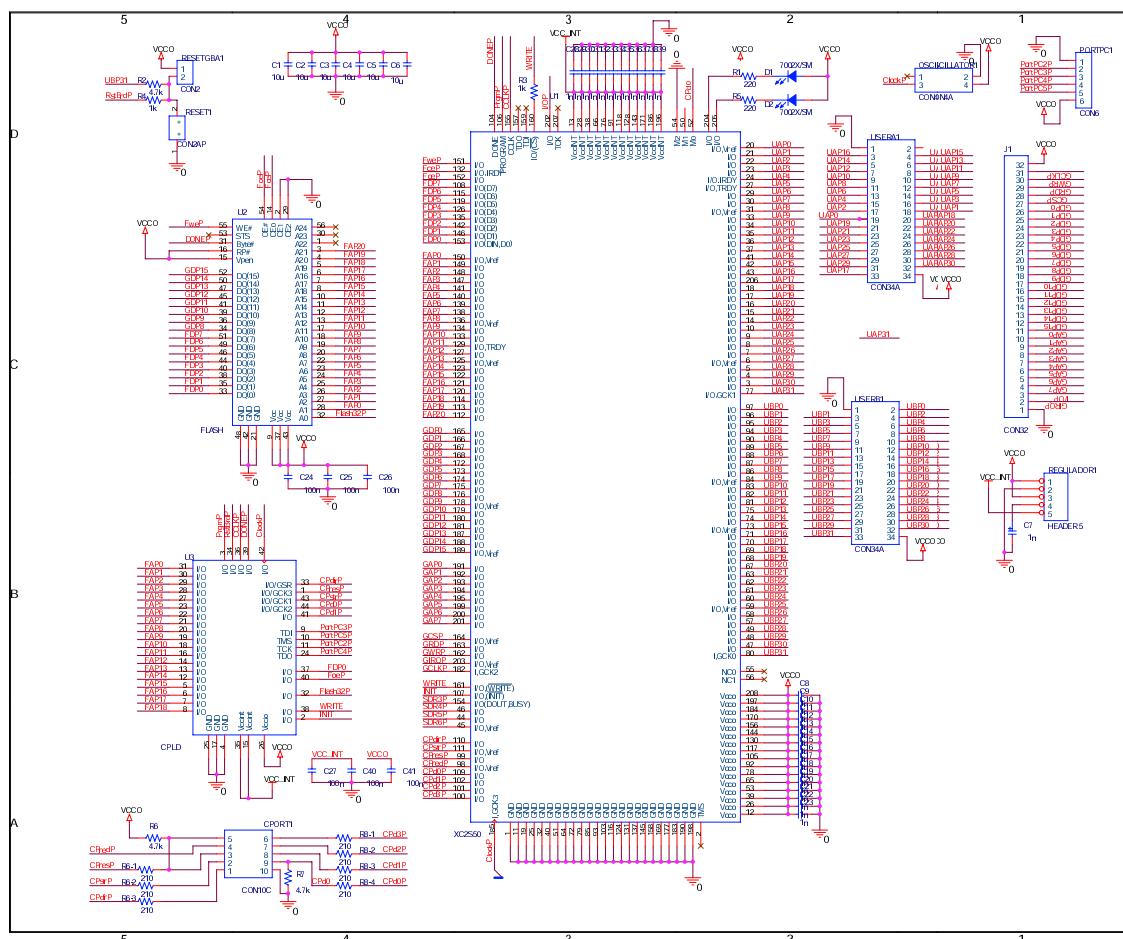




Apéndice B

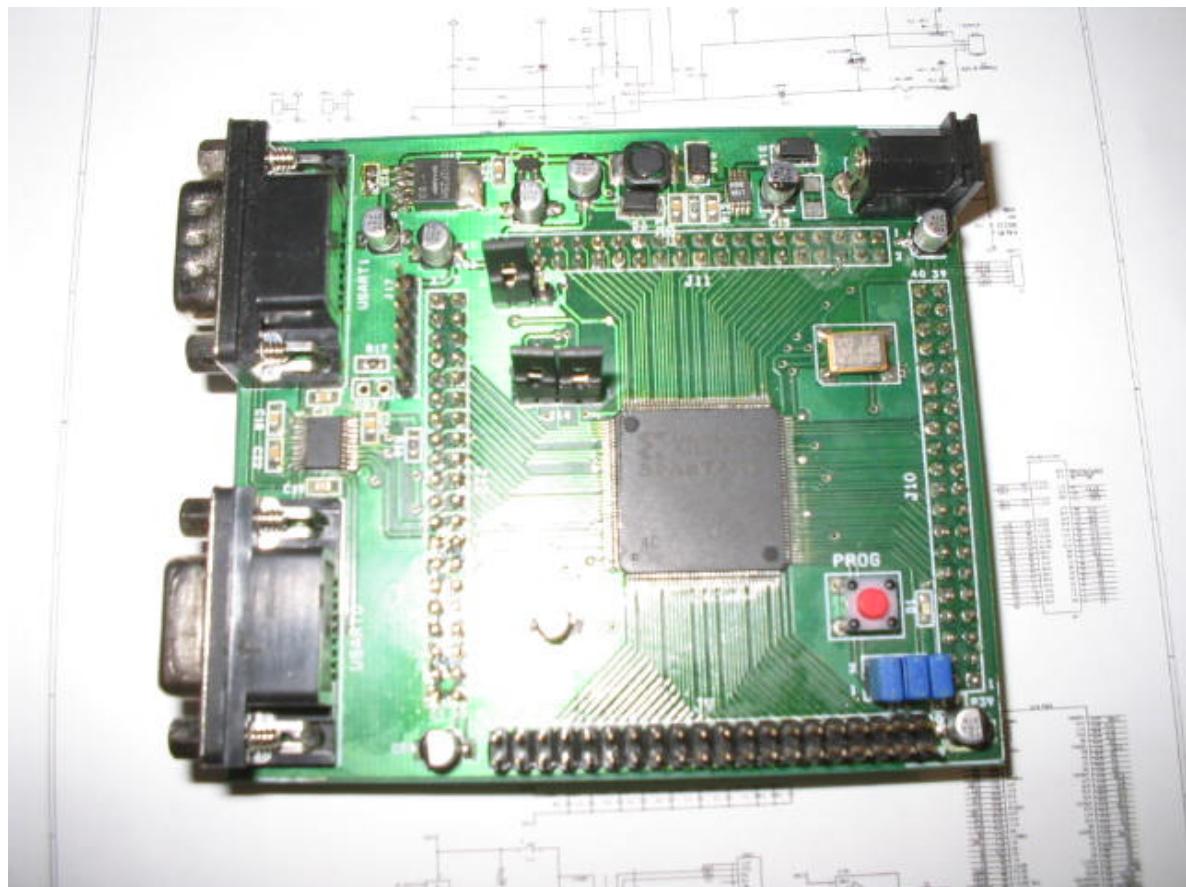
Tarjeta de Desarrollo para el GBA de Nintendo

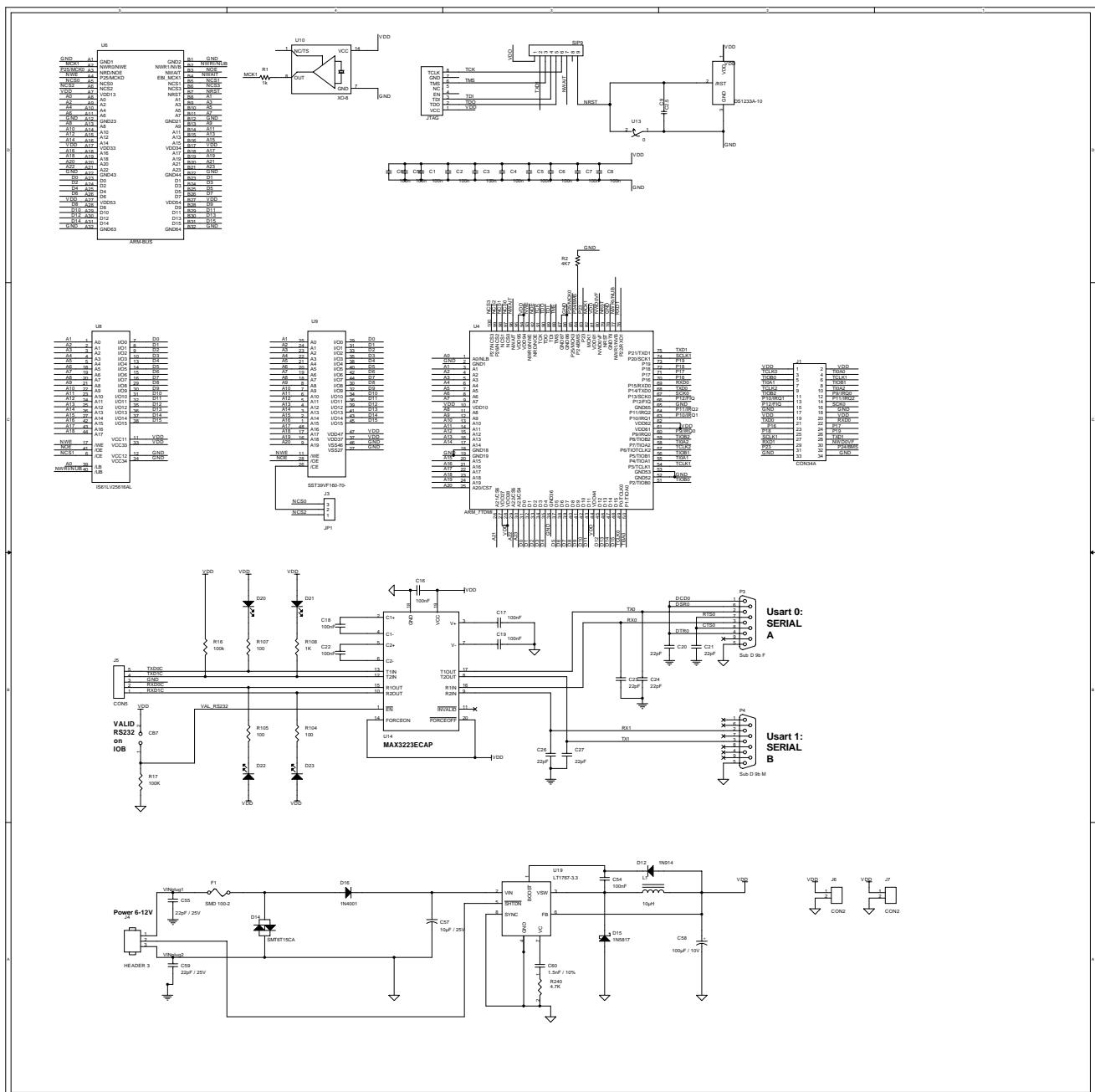




Apéndice C

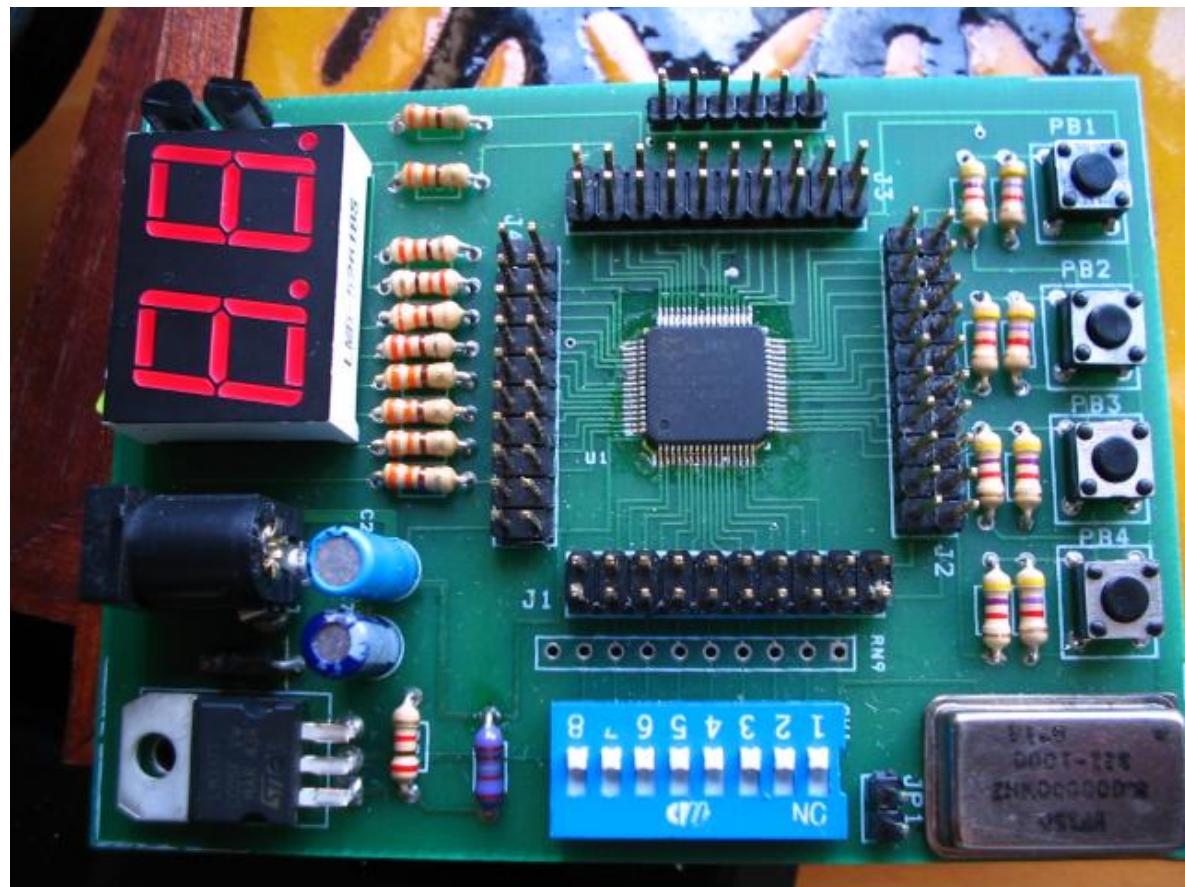
Tarjeta de Desarrollo para el SoC LH79520 de Sharp

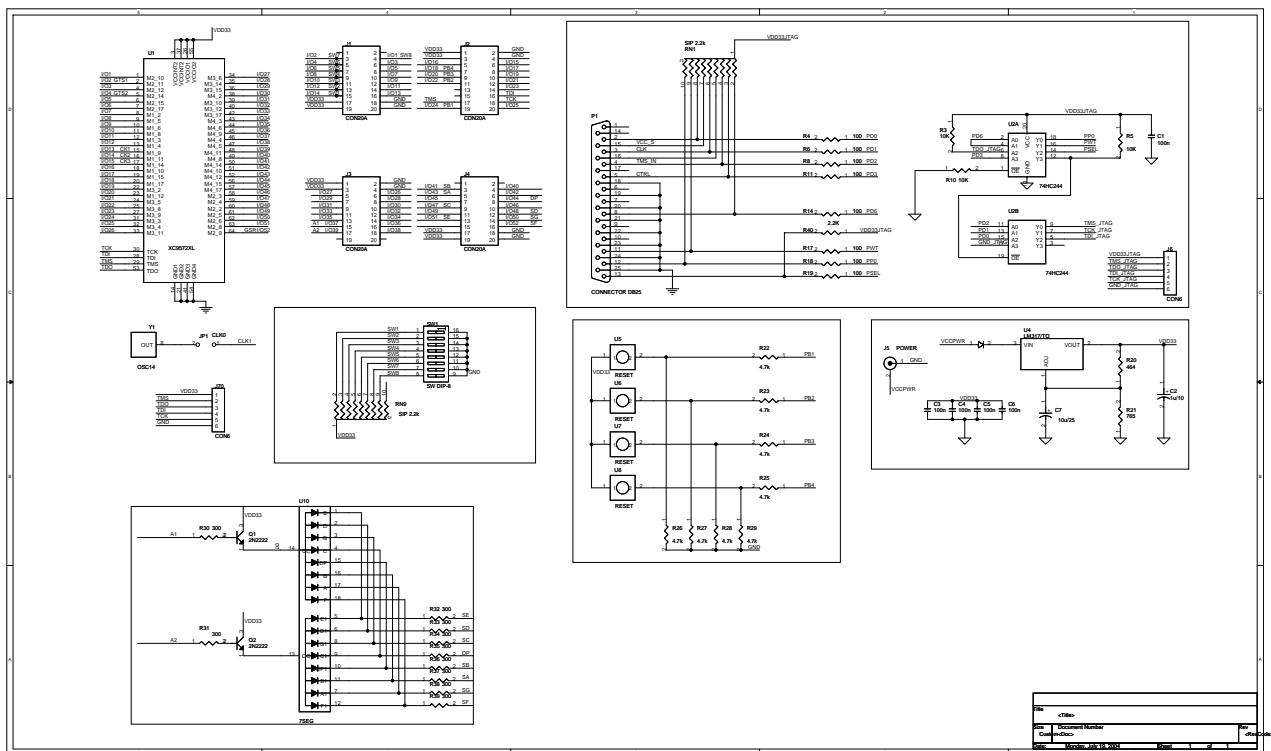




Apéndice D

Tarjeta de Desarrollo el CPLD XC9572XL

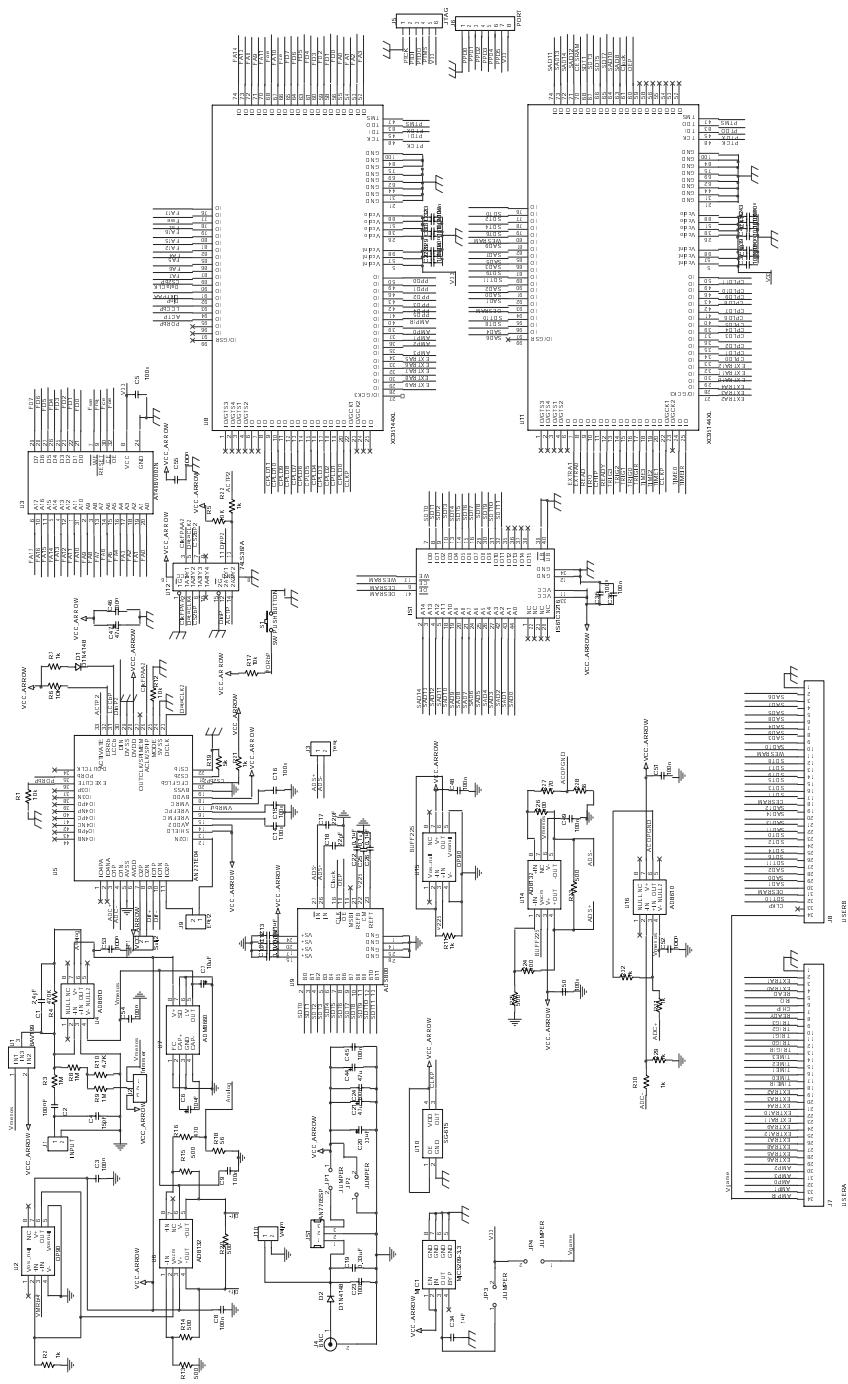




Apéndice E

Equipo de Laboratorio basado en el GBA de Nintendo

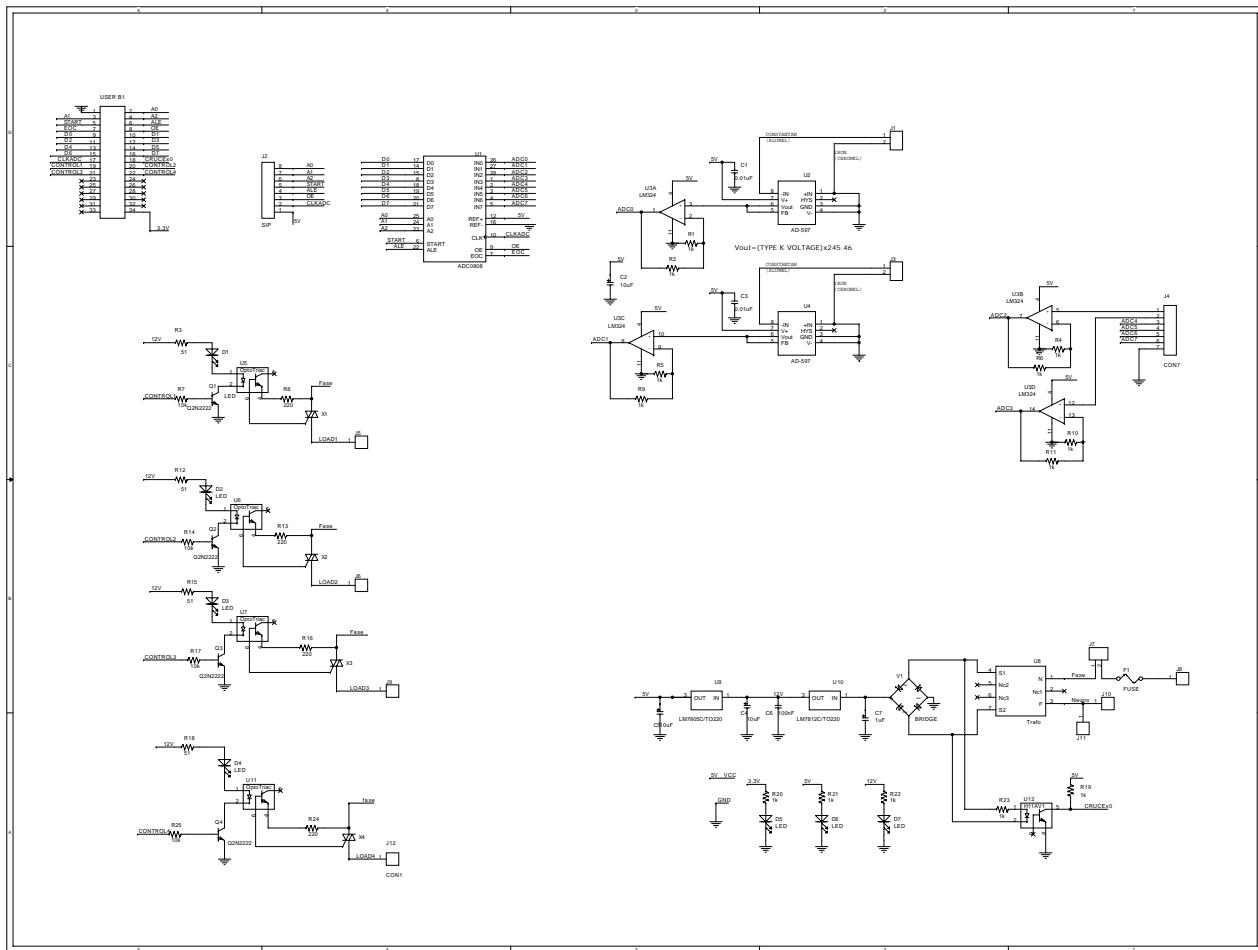




Apéndice F

Horno de Reflujo Para Montaje de ICs de Montaje Superficial





Apéndice G

Implementación de Sistemas Digitales Complejos Utilizando Sistemas Embebidos

Publicado en: Memorias del XI Workshop de Iberchip ISBN 959-261-105-X.

IMPLEMENTACION DE SISTEMAS DIGITALES COMPLEJOS UTILIZANDO SISTEMAS EMBEBIDOS

*C. Camargo**,

* Profesor del Departamento de Ingeniería Eléctrica & Electrónica
Universidad Nacional de Colombia, Bogotá
{ccamargo}@gmun.unal.edu.co

RESUMEN

Con el pasar del tiempo los sistemas digitales van creciendo en complejidad, así como las exigencias de desempeño y funcionamiento. La incurción de los sistemas digitales en aplicaciones vitales, en las cuales un error resultaría en la pérdida de vidas humanas exige que estos sistemas sean muy confiables (tolerantes a fallos) y presenten un desempeño muy alto. El desarrollo de procesadores de 32, 64 bits de bajo costo unido a la disponibilidad de herramientas software como librerías, drivers y sistemas operativos de tiempo real hace que la realización de este tipo de sistemas no sea una tarea imposible. Este artículo suministra los primeros pasos en el estudio de los sistemas embebidos (SE) y su intención es mostrar la potencialidad de estos. Adicionalmente se abordan los pasos que se deben llevar a cabo en el diseño de los SE y se realiza una explicación de las herramientas utilizadas .

IMPLEMENTACION DE SISTEMAS DIGITALES COMPLEJOS UTILIZANDO SISTEMAS EMBEBIDOS

C. Camargo^{}, D. Ovalle, D. Mendez*

+ Estudiantes Ingeniería Electrónica

* Profesor del Departamento de Ingeniería Eléctrica & Electrónica

Universidad Nacional de Colombia, Bogotá

{cicamargoba, dmendezc, daovallea}@gmun.unal.edu.co

RESUMEN

Con el pasar del tiempo los sistemas digitales van creciendo en complejidad, así como las exigencias de desempeño y funcionamiento. La incursión de los sistemas digitales en aplicaciones vitales, en las cuales un error resultaría en la pérdida de vidas humanas, exige que estos sistemas sean muy confiables (tolerantes a fallos) y presenten un desempeño muy alto. El desarrollo de procesadores de 32, 64 bits de bajo costo unido a la disponibilidad de herramientas software como librerías, drivers y sistemas operativos de tiempo real, hace que la realización de este tipo de sistemas no sea una tarea imposible.

1. INTRODUCCION

En algunos países en vía de desarrollo, aún se utilizan procesadores de 8 o 12 bits y el lenguaje ensamblador como plataforma de desarrollo para un amplio número de aplicaciones. Debido a las limitaciones de esta plataforma (velocidad, ancho del bus de datos, periféricos, número de pines, herramientas de desarrollo de alto nivel, librerías), no es posible realizar aplicaciones complejas que requieran procesamiento con características de tiempo real o multitarea. Además el uso de metodologías de diseño anticuadas crea una dependencia tecnológica y hace que el tiempo de desarrollo dependa fuertemente de la experiencia del diseñador.

El continuo avance de los sistemas software y de los componentes hardware presentan nuevas oportunidades para la implementación de sistemas embebidos en múltiples campos tales como: Control, multimedia, aplicaciones de red e instrumentación biomédica. En la actualidad se dispone de una gran variedad de dispositivos HW (FPGAs, CPLDs, IPs,

ICs) de bajo costo, esto unido al desarrollo de Sistemas Operativos de Tiempo Real (como linux [1], eCos [2], etc) de libre distribución, librerías que permiten el control de dispositivos Hardware como LCDs, Interfaces USB, RS232, de red, librerías matemáticas, científicas, etc. y el desarrollo de compiladores que permiten generar el contenido de la memoria de programa de un gran número de procesadores a partir de un lenguaje de alto nivel como el C, C++ o Java, permite el diseño de sistemas complejos, reduciendo el costo del diseño, el tiempo de desarrollo y en algunos casos la dependencia tecnológica.

Este artículo suministra los primeros pasos en el estudio de los sistemas embebidos y su intención es mostrar la potencialidad de estos. La discusión se divide en 3 partes: la primera es un resumido marco teórico donde se enumeran las características de los sistemas embebidos y las tareas que deben ser llevadas a cabo en su diseño, la segunda parte hace una descripción de las herramientas hardware y software necesarias para la implementación de sistemas embebidos, y por último se muestran algunas aplicaciones realizadas en la Universidad Nacional de Colombia.

2. SISTEMAS EMBEBIDOS

Un sistema embebido (SE) es una máquina computacional que emplea una combinación de Hardware y Software para realizar una función específica. Es parte de un sistema más grande que puede no ser un computador y trabaja en un ambiente reactivo con restricciones temporales. En este tipo de sistemas el software se utiliza para proporcionar flexibilidad y funcionalidad. El Hardware (Procesadores, ASICs, Memorias...) es utilizado para proporcionar desempeño y en algunas ocasiones seguridad.

2.1 Características

- Realiza una función única o un pequeño grupo de funciones relacionadas, no es un sistema de propósito general.
- De alto desempeño y con fuertes restricciones temporales
- El costo, consumo de potencia y la confiabilidad son atributos que afectan el diseño.
- Sistemas reactivos con características de tiempo real.

2.2 Principales tareas a desarrollar en el diseño de Sistemas Embebidos:

A continuación se enumeran las tareas que deben ser realizadas en el diseño de un sistema embebido, (adicionalmente se deben seguir los pasos de la metodología de diseño de sistemas digitales).

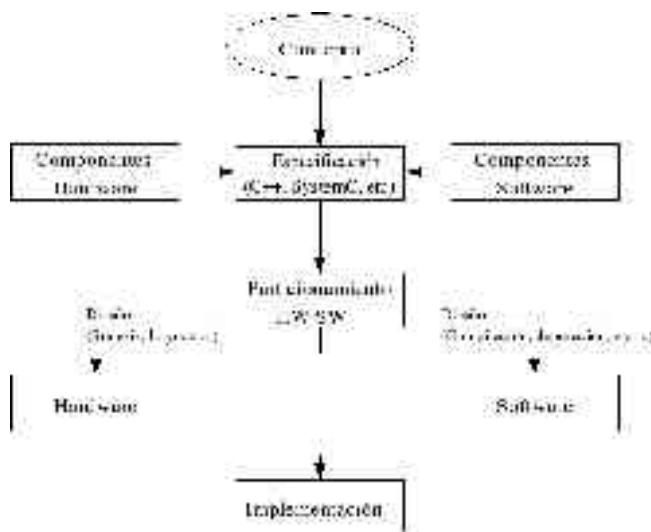


Figura 1. Flujo de Diseño de los Sistemas Embebidos.

2.3.1.Especificación y Modelamiento: Descripción algorítmica del sistema. Por lo general estos algoritmos son validados en una descripción ejecutable, es decir, un programa escrito en un lenguaje de alto nivel¹ que modela el comportamiento del sistema, en el cual se pueden validar de forma fácil y rápida dichos algoritmos.

¹Ejemplos de este tipo de lenguajes son el C++, UML, SystemC, HandelC

2.3.2.Particionamiento: División del algoritmo en componentes funcionales más pequeñas.

2.3.3.Particionamiento HW-SW: Implementación de las unidades funcionales del punto anterior en SW (Código ejecutándose en un procesador) o en HW (Implementado en un PLD). El particionamiento ideal es aquel en el que todas las funciones son implementadas en SW, (esto debido a que la implementación fué validada utilizando un lenguaje de alto nivel, el cual puede ser el mismo lenguaje utilizado por el compilador), pero debido a que los procesadores ejecutan instrucciones de forma secuencial, si la ejecución de una función compromete el cumplimiento de las restricciones temporales (es decir, la función emplea muchos ciclos de máquina) esta debe ser implementada en HW con el fin de aprovechar sus características concurrentes (de aquí la necesidad de las funciones HW en sistemas de alto desempeño). La partición óptima es aquella que cumple con las restricciones del diseño a costo mínimo².

Una vez se ha generado una partición óptima se debe validar la solución, este paso es algo complejo ya que involucra la simulación de componentes SW y HW. En la actualidad existen herramientas especializadas que permiten realizar esta co-simulación como: Ptolemy [3], SID [4], Seamless [5], etc.

2.3.6.Planificación: En este paso se decide el tiempo en el que se ejecuta una determinada función. Esto es importante cuando se comparten recursos Hardware. Los Sistemas Operativos de Tiempo Real (RTOS) proporcionan mecanismos que permiten administrar los recursos de forma eficiente.

2.3.7.Implementación: En este paso se baja en el nivel de abstracción y se llevan las funciones a software que corre dentro de un procesador y a una combinación de hardware custom, semicustom o IP dentro de un PLD.

²El costo SW se traduce en el número de ciclos de reloj que tarda una determinada función en ejecutarse, el Costo HW indica el área dentro del IC ocupada por la misma función .

3. REQUERIMIENTOS HW Y SW.

Debido a que los sistemas embebidos son una mezcla de componentes SW y HW es necesario contar con herramientas especializadas para cada caso. Uno de los objetivos de este artículo es mostrar las herramientas gratuitas que pueden ser utilizadas para el Diseño de un SE. Por lo tanto no se hablará de las herramientas comerciales.

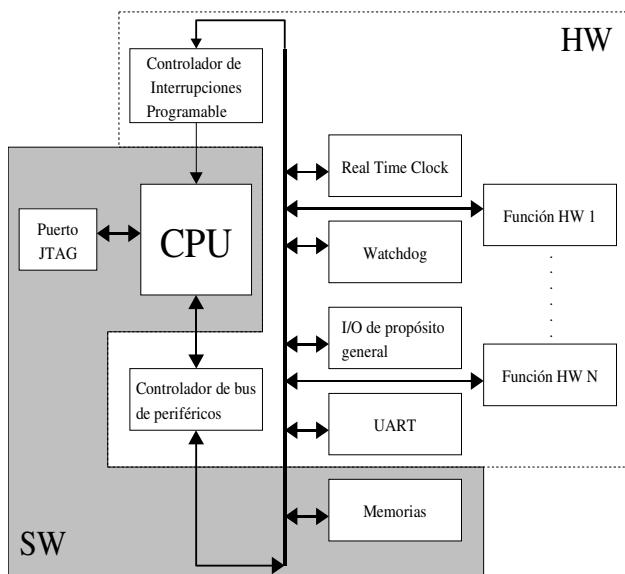


Figura 2. Componentes de un SE típico.

3.1 Requerimientos HW

Procesador: Las tareas SW se ejecutan sobre procesadores. Sus características como ancho de los buses de datos y direcciones, arquitectura, velocidad varían dependiendo de la aplicación. Sin embargo, es indispensable que tenga la capacidad de realizar operaciones JTAG que permitan realizar funciones de depuración y proporcione una ruta de acceso a los periféricos.

Memorias: Se debe disponer de dos tipos de memorias:

Memoria no volátil, que permite el almacenamiento de los programas, datos constantes y además suministre una forma de iniciar el sistema. En la actualidad se utilizan memorias Flash como medio de almacenamiento debido a su bajo costo y gran capacidad de almacenamiento, además estas memorias incorporan circuitos de programación lo cual elimina el uso de programadores especiales y la necesidad de niveles de voltaje elevados, estas memorias presentan como

inconveniente la baja velocidad de lectura y escritura, sin embargo esta desventaja puede ser eliminada haciendo un arranque tipo ROM-RAM, este arranque es similar al de los computadores de escritorio, los programas residen en una unidad de almacenamiento, y en tiempo de ejecución son copiados a la memoria RAM desde donde son ejecutados.

Memoria volátil: Utilizada para almacenar las variables y estructuras de datos necesarios para la ejecución de los programas.

Periféricos: Algunos sistemas operativos requieren ciertos periféricos para su funcionamiento como por ejemplo: RTC (Reloj de Tiempo Real), PIC(Controlador de Interrupciones Programable) y un puerto serie como herramienta de diagnóstico. En la actualidad, los procesadores incluyen ciertos periféricos que ayudan a la implementación de las aplicaciones más comunes, como por ejemplo:

- Controladores de LCDs.
- Tarjetas de Red
- Timers
- USARTs
- Codificadores de Audio
- Controlador de Touch screen
- Puertos I2C, SPI, IR.

Además de los periféricos que requiere el sistema operativo se encuentran los que requiere la aplicación.

Existen tres opciones para la implementación de un SE:

1. Utilizando un procesador con puerto JTAG, una FPGA y memorias externas.
2. Utilizando un SoC [6], [7] el cual posee una gran variedad de periféricos y memorias de los dos tipos en el mismo chip. Las funciones HW propias del sistema deben ser implementadas en una FPGA o en un CPLD.
3. En la actualidad las empresas desarrolladoras de PLDs como Xilinx [8] y Altera [9] proporcionan otra alternativa: En lugar de utilizar un procesador y una FPGA que implementa las funciones HW, se implementa el procesador dentro de la FPGA, esto reduce el tamaño de la tarjeta y aumenta el desempeño y flexibilidad del sistema. Las memorias deben ser externas.

3.2 Requerimientos SW

Comercialmente existen muchos procesadores y FPGAs que cumplen con los requerimientos anteriores, sin embargo, es indispensable contar con las herramientas adecuadas para poder realizar diseños sobre el procesador elegido. A continuación se listan las herramientas necesarias para generar el contenido de la memoria de programa,

Compilador: Aunque el lenguaje ensamblador ayuda a optimizar el tiempo de ejecución de funciones críticas, no es recomendable utilizarlo como herramienta de programación, debido a que el tiempo requerido para implementar un diseño complejo puede ser muy largo (dependiendo de la experiencia del programador), además el lenguaje ensamblador crea una dependencia tecnológica ya que está unido a un determinado procesador.

La utilización de lenguajes de alto nivel como el C++ o el UML, reduce el tiempo de programación, facilita el re-uso y la portabilidad del código. Al utilizar un lenguaje standard como el C o el C++ como herramienta de programación se puede reciclar el código en diferentes proyectos que involucren diferentes procesadores, lo único con lo que se debe contar es con un compilador que permita generar el contenido de la memoria de programa para las diferentes plataformas.

El compilador GCC [8](GNU project C and C++ compiler) de uso gratuito ha sido utilizado para el desarrollo de un gran número de aplicaciones comerciales y aprobado ser muy confiable. Por lo tanto uno de los puntos que deben ser tomados en cuenta a la hora de elegir el procesador es seleccionar uno de los que se encuentran soportados por gcc, en la actualidad gec soporta los siguientes procesadores: ALPHA, ARM, MIPS, M68K, SPARC, PPC, DSP16XX, H8300, INTEL, MMIX, BLAZE y otros.

GCC genera un archivo en formato ELF (Executable and Linking Format, este formato está dividido en las siguientes secciones (solo se muestran las más importantes):

.bss	Datos sin inicializar.
.data .data1	Datos inicializados
.debug	Información para depuración simbólica
.dynamic	Información para enlace dinámico
.dynstr	Strings necesarios para enlace dinámico

.rodata .rodata1 Datos de tipo solo lectura.
.text Instrucciones ejecutables.

Binutils: La mayoría de esta información no es necesaria para la ejecución de un programa, solo las secciones .data .rodata y .text, para “extraer” estas secciones del ejecutable se utilizan una serie de herramientas (binutils [9]) que hacen parte de la cadena de herramientas GNU.

Depurador: EL GDB [10] (GNU Debugger) permite depurar el código ejecutando instrucción por instrucción y mostrando el valor que toman los registros y las variables internas. Una ventaja muy importante de GDB es que permite hacer depuración dentro del chip, es decir ejecuta las instrucciones dentro del procesador, lee el valor de los registros y variables y retorna esta información, esto es importante ya que no simula el comportamiento del procesador sino que la información obtenida es real. GDB puede comunicarse con los procesadores de diversas formas siendo las más populares el puerto serie y el JTAG.

Para poder realizar este tipo de depuración, GDB permite descargar el programa a la memoria RAM del procesador. Otra herramienta que puede ser utilizada para esto es ARMTOP [11]

Programación de la memoria no volátil: La distribución oficial de GDB no puede programar las memorias no volátiles, sin embargo proporciona un canal por el cual se puede realizar dicha programación. Se puede cargar en RAM un programa que programe esta memoria.

JTAGER[2], es una herramienta que proporciona un puente entre la interfaz JTAG del procesador y el PC, permitiendo programar estas memorias vía JTAG.

Herramientas de Síntesis: Como herramienta de síntesis puede utilizarse cualquier herramienta comercial. Las cuales se encuentran en versiones gratuitas.

RTOS: El sistema operativo es muy importante dentro del diseño de un SE, ya que él debe proporcionar las siguientes funciones:

Secuencia de arranque, parada y re-arranque
Control de interrupciones
Control de excepciones

Control de temporizadores
 Programador de tareas
 Elementos de sincronización
 Mecanismos de arbitraje de recursos
 Gestión de la memoria
 Trazado de tareas y servicios
 Manejo de dispositivos hardware
 Herramientas de autodiagnóstico

4. APLICACIONES DESARROLLADAS EN LA UNIVERSIDAD NACIONAL DE COLOMBIA.

4.1 Plataforma de Desarrollo basada en el GBA.

El obstáculo más grande que debemos sortear cuando se quieren hacer diseños utilizando sistemas embebidos es el alto costo de las herramientas de desarrollo. Al realizar una búsqueda de plataformas disponibles comercialmente encontramos al GAMEBOY ADVANCE (GBA) de Nintendo, este “Juguete” posee las siguientes características:

Procesador: ARM7TDMI 32bit RISC CPU, 16.78MHz.
Memoria Interna:
BIOS ROM 16 KBytes
Work RAM 288 KBytes (32K in-chip + 256K on-board)
VRAM 96 KBytes
Palette RAM 1 KByte (256 BG colors, 256 OBJ colors)
Display: 240x160 pixels (TFT color LCD display)
Sonido
Análogo 4 channel CGB compatible
Digital 2 DMA sound channels
Output Built-in speaker, or stereo headphones
Gamepad 4 Direction Keys, 6 Buttons
Puerto serie Varios modos de transferencia (Asíncrono, síncrono)
Memoria Externa: 32MB ROM o flash ROM + max 64K SRAM

Como puede verse, esta plataforma cuenta con todas las funcionalidades de las plataformas comerciales y su principal ventaja es el bajo costo (USD 30). Como herramienta de enseñanza el GBA es muy útil a la hora de mostrar conceptos relacionados con los RTOS (Multitarea, mecanismos de

sincronización, planificador de tareas, etc), ya que posee un display que permite imprimir resultados de la ejecución de un programa. Debido a que el procesador es un ARM7, se pueden utilizar las herramientas GNU como entorno de desarrollo. Además existen programas gratuitos [12], [13] que emulan el funcionamiento del dispositivo.

Como se sabe, el GBA funciona con “cartuchos” de juegos. El conector donde se insertan estos juegos proporciona los buses de Datos, Direcciones y Control (Rd, Wr, IRQ). Un cartucho está formado por: una memoria no volátil, la cual almacena el “ejecutable” del juego, una memoria RAM y una interfaz con los buses. Si al momento de energizar el GBA se detecta la presencia de un cartucho, el procesador inicia la ejecución del juego. Si no, el procesador espera que se cargue un programa en la memoria RAM interna. La carga de este programa se hace através del puerto serie del GBA utilizando el software xboo [14].

El sistema operativo utilizado en las aplicaciones es eCos [15]. En la figura 3, se muestran los componentes de este RTOS. Se eligió eCos debido a que la compañía Charmedlabs [17] lo adaptó al GBA.

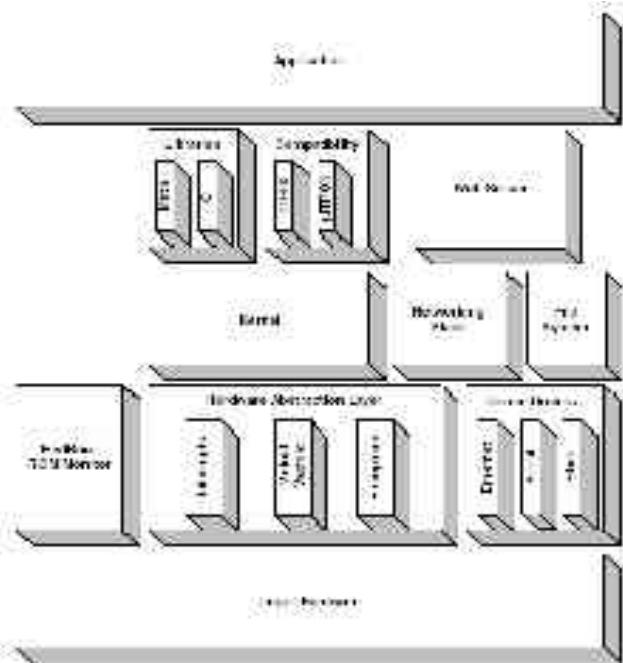


Figura 3. Componentes de eCos[16].

Debido a que las aplicaciones deben ser autónomas, se requiere que el programa esté almacenado en un medio no volátil. Además, como se dijo anteriormente un SE puede requerir funciones HW, las cuales se implementan en un PLD. Para cumplir con estos requerimientos se construyó una plataforma basada en la herramienta comercial XPORT [17]. El diagrama de bloques de nuestra tarjeta se muestra en la siguiente figura:

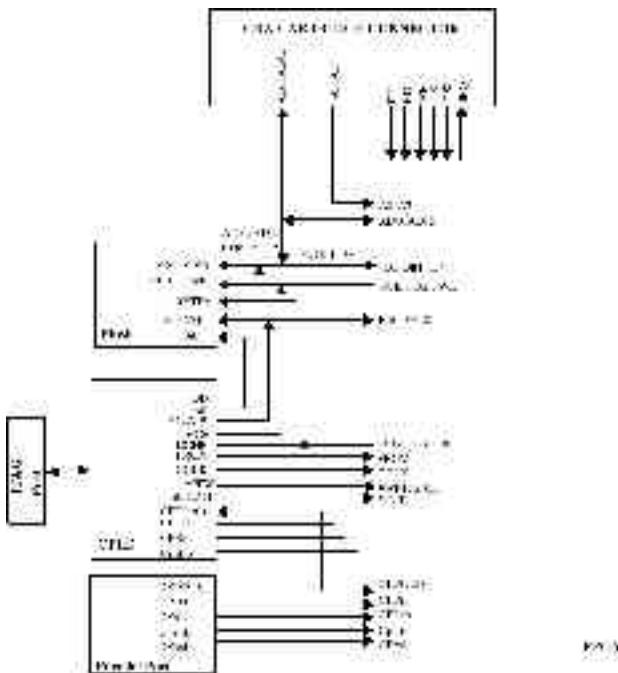


Figura 4. Diagrama de Bloques de Tarjeta de desarrollo.

Debido a que la FPGA es volátil se dispone de un mecanismo que realiza la configuración de la misma cada vez que se energiza el sistema. Este mecanismo está formado por el puerto JTAG (puerto paralelo del Computador), el CPLD (encargado de la comunicación con el PC) y la memoria Flash. La Memoria Flash realiza tres funciones: Guardar la información de configuración de la FPGA, Almacenar el programa que debe ser ejecutado por el GBA y por último almacenar una segunda configuración de la FPGA necesaria para establecer un medio de programación de la flash (Esta configuración solo se utiliza cuando se programa la Flash).

En la Figura 5 se muestra el Diagrama de bloques de la implementación de una aplicación utilizando la tarjeta de desarrollo. Como puede observarse, la FPGA proporciona un camino de conexión entre el GBA y la FLASH, además

proporciona los recursos HW para implementar periféricos propios de la aplicación.

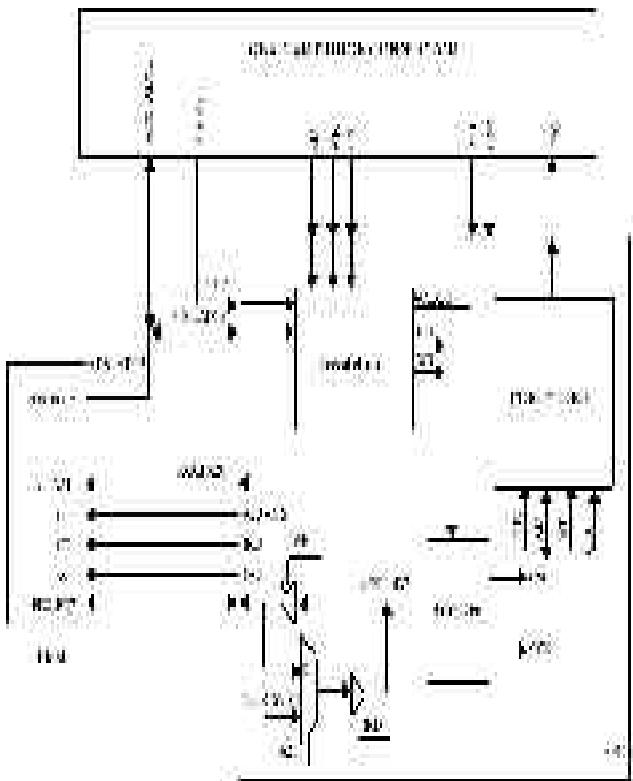


Figura 5. Diagrama de Bloques de una aplicación típica utilizando la tarjeta de desarrollo.

4.2. Tarjeta de desarrollo basada en un procesador AT91.

A pesar de los excelentes resultados arrojados por nuestro sistema de desarrollo basado en el GBA, esta no es una alternativa que se pueda utilizar en ambientes industriales aislados, en los cuales no existe la intervención humana. Para poder realizar aplicaciones comerciales en las que se requiera la utilización de un SE, se diseñó una tarjeta de desarrollo basada en un procesador ARM comercial de ATMEL el AT91. Esta tarjeta de desarrollo fue diseñada de tal forma que pudiera reemplazar al GBA en aplicaciones en las que no fuera necesario utilizar un dispositivo de video y pudiera adaptarse perfectamente a la tarjeta de desarrollo XPORT. La siguiente figura muestra una fotografía de la tarjeta.

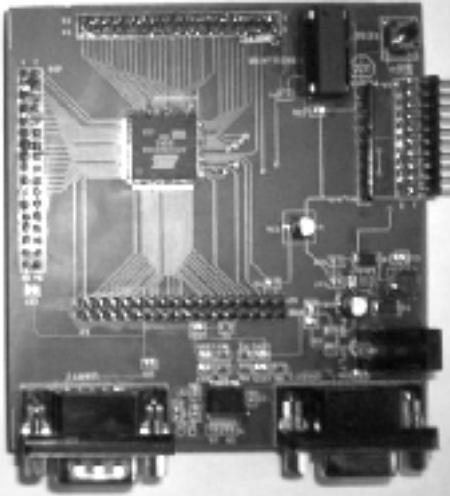


Figura 6. Tarjeta de desarrollo basada en un procesador AT91 de ATMEL.

Esta tarjeta solo posee los componentes mínimos que permitan depurar aplicaciones, es decir, el procesador (con una memoria RAM interna de 256 kB), el puerto de programación JTAG y puertos seriales de diagnóstico y aplicación.

4.3 Sistema de vigilancia con transmisión de imágenes vía Internet

En esta aplicación existe una cámara conectada a la red ethernet a través de un GBA, el GBA se encarga de recibir la información de la imagen y publicarla en un servidor WEB.

Para la realización de este proyecto se trabajó con el circuito integrado CS8900 [18] y se utilizó el stack lwip (lightweight TCP/IP stack) de eCos[15]. El stack lwip fue diseñado para ser usado en sistemas embebidos y posee un footprint de memoria muy bajo en comparación con el stack BSD.

Se eligió este chip y este stack debido a que están soportados bajo eCos, lo cual reduce el tiempo de desarrollo, el driver de eCos para el CS8900 proporciona las siguientes funciones:

Inicializa el controlador: (cs8900a_init)

Comienza la interfaz (cs8900a_start)

Detiene la Interfaz (cs8900a_stop)

Determina si es posible enviar otro paquete:

(cs8900a_can_send)

Controla la interface (cs8900a_control)

Envía (cs8900a_send)

Recibe (cs8900a_receive)

Envía un paquete a un protocolo de nivel superior
(cs8900a_deliver)

La figura 7, muestra una fotografía del adaptador de red diseñado e implementado.



Figura 7. Tarjeta de red con el IC CS8900.

4.4 Osciloscopio de un Canal.

Este proyecto explora la capacidad de nuestra plataforma para desarrollar tareas de tiempo REAL, para esto se implementaron dos aplicaciones basadas en la siguiente arquitectura:

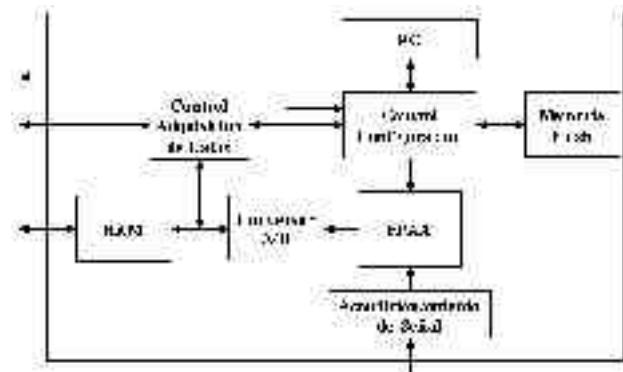


Figura 8. Diagrama de Bloques de la tarjeta de adquisición de datos.

El módulo principal es un FPAA [19] (Field Programmable Analog Arrays), dispositivo que puede ser configurado parcialmente de forma digital en tiempo de ejecución, para implementar una gran variedad de aplicaciones análogas como filtros, amplificadores, controladores, etc. Esto nos permitirá tener una plataforma de adquisición de datos muy flexible para dejar abierta cualquier otra aplicación que se desee desarrollar.

Debido a que la FPAA es volátil se debe proporcionar un mecanismo de configuración automática similar al de la tarjeta XPORT. Para este fin se tiene el bloque *Control Configuración*. La información de la configuración es estática para el caso del osciloscopio y dinámica para el caso del PID auto-sintonizado.

Una vez implementado el circuito análogo dentro del FPAA, se necesita convertir estos datos a formato digital para su posterior transmisión al GBA. Para esta tarea se utiliza un conversor Análogo-Digital

Estos datos que convierte el conversor no podrán ser enviados inmediatamente al GBA, ya que la velocidad de procesamiento de este disminuirá la rápida adquisición de las señales, y obviamente disminuye el ancho de banda que se podrá graficar. Debido a esto la adquisición debe hacerse por Hardware, para lo cual se cuenta con un módulo *Control Adquisición de Datos*. Este bloque se implementó nuevamente por medio de un CPLD, el cual se encarga de llenar una memoria RAM con una cantidad definida de datos que arroja el conversor, dependiendo de los parámetros de Trigger y Tiempo con los que se cuente. Una vez adquiridos los datos, se enviará una interrupción al procesador del GBA, para que éste los tome de la memoria RAM. De esta forma no importa la velocidad con la que el GBA procese los datos, ya que la adquisición ha ocurrido con anterioridad a alta velocidad, logrando capturar señales de mayor frecuencia. La figura 9 muestra una fotografía de la tarjeta de adquisición de datos.

5. TRABAJO FUTURO

El siguiente paso en nuestro estudio es la implementación de aplicaciones utilizando un sistema operativo más poderoso como linux, para esto ya se desarrollaron dos tarjetas, una basada en un procesador ARM720, el cual es básicamente un ARM7 + MMU, este procesador posee los periféricos

necesarios para implementar aplicaciones gráficas con lo cual se puede cargar un sistema manejador de ventanas como por ejemplo microwindows. La otra tarjeta es una plataforma de desarrollo para microblaze, posee una SPARTAN III de 400 mil compuertas y se implementarán aplicaciones realizadas con uClinux[20].



Figura 9. Tarjeta de adquisición de datos.

Además como parte del trabajo Doctoral de Carlos Camargo se estudiará la implementación de sistemas tolerantes a fallos utilizando sistemas inmunes artificiales y hardware reconfigurable. Utilizando claro los conocimientos adquiridos en este trabajo

6. CONCLUSIONES

Los SE proporcionan una herramienta muy poderosa para la implementación de sistemas digitales con de alto desempeño.

Las herramientas de libre distribución proporcionan una base de desarrollo sólida y económica.

Los RTOS de libre distribución como eCos, linux y uClinux, facilitan la implementación de aplicaciones, ya que soportan una gran variedad de dispositivos Hardware y poseen librerías muy desarrolladas para aplicaciones en SE.

La adopción de esta tecnología es necesaria en países como los nuestros, (en vía de desarrollo) ya que nos pone al mismo nivel que países más desarrollados.

El GBA es un excelente herramienta para la enseñanza de SE.

7. REFERENCIAS

- [1] ARM Linux Project <http://www.arm.linux.org.uk/>
- [2] Proyecto eCos: <http://sources.redhat.com/ecos>
- [3] Proyecto ptolrmy: <http://ptolemy.eecs.berkeley.edu/>
- [4] Proyecto SID: <http://sources.redhat.com/sid>
- [5]
http://www.mentor.com/products/fv/hwsw_coverage/index.cfm
- [6] <http://www.sharpsma.com>
- [7] <http://www.cirrus.com>
- [8] <http://www.xilinx.com>
- [9] <http://sources.redhat.com/binutils/>
- [10] <http://sources.redhat.com/gdb/>
- [11] <http://hri.sourceforge.net/tools/armtool.html>
- [12] <http://www.nintendo.com/systemsgba>
- [13] <http://vba.ngemu.com/>
- [14] <http://www.work.de/nocash/gba-xboo.htm>
- [15] <http://sources.redhat.com/ecos>
- [16] A. J. Massa, Embedded software development with eCOs, Prentice Hall.
- [17] <http://www.charmedlabs.com>
- [18] <http://www.cirrus.com/en/products/pro/detail/P46.html>
- [19] <http://www.anadigm.com>
- [20] <http://www.uclinux.org>

Apéndice H

Control Adaptativo Embebido

Publicado en: Memorias del XI Workshop de Iberchip ISBN 959-261-105-X.

CONTROL ADAPTATIVO EMBEBIDO

F. Pedraza[%], F. Segura, C⁺. Camargo *A. Gauthier[†]

{f-pedraz, fsegura, agauthie}@uniandes.edu.co, cicamargoba@unal.edu.co

* Profesor del Departamento de Ingeniería Eléctrica & Electrónica Universidad Nacional de Colombia, Bogotá

† Profesores del Departamento de Ingeniería Eléctrica & Electrónica Universidad de los Andes, Bogotá

[%] Estudiante de Ingeniería Electrónica Universidad de los Andes, Bogotá.

RESUMEN

En este documento se presenta información relevante acerca del control adaptativo y su implementación sobre plataformas embebidas. El sistema desarrollado en este proyecto presenta ventajas de versatilidad, siendo capaz de manejar diversos esquemas de control de procesos *SISO*; y de portabilidad, utilizando elementos de fácil manipulación. Como logros más destacados alcanzados en este trabajo, se destacan: obtención de un regulador auto-ajustable (*STR*) para sistemas de hasta quinto orden; desarrollo de librerías en control, identificación e interfaz para el sistema operativo de tiempo real *eCos*. Se realizaron modelos en *MATLAB* que permiten realizar pruebas *a priori* de las capacidades del sistema. Por las simulaciones realizadas, se observó la necesidad de introducir mejoras al algoritmo de control que permitan el manejo de señales de prueba generadas dentro del dispositivo. En general, este proyecto ha abierto la puerta para la implementación de diferentes esquemas de control adaptativo y su posterior verificación de manera experimental.

CONTROL ADAPTATIVO EMBEBIDO

F. Pedraza[%], F. Segura, C⁺. Camargo *A. Gauthier[†]

{f-pedraz, fsegura, agauthie}@uniandes.edu.co, cicamargoba@unal.edu.co

* Profesor del Departamento de Ingeniería Eléctrica & Electrónica Universidad Nacional de Colombia, Bogotá

† Profesores del Departamento de Ingeniería Eléctrica & Electrónica Universidad de los Andes, Bogotá

[%] Estudiante de Ingeniería Electrónica Universidad de los Andes, Bogotá.

RESUMEN

En este documento se presenta información relevante acerca del control adaptativo y su implementación sobre plataformas embebidas. El sistema desarrollado en este proyecto presenta ventajas de versatilidad, siendo capaz de manejar diversos esquemas de control de procesos SISO; y de portabilidad, utilizando elementos de fácil manipulación.

1. INTRODUCCION

El desarrollo del control adaptativo ha sido necesario para el manejo de procesos cuyas dinámicas cambian continuamente y cuyas variaciones no pueden ser previstas. En esos casos los esquemas de control tradicional no han sido capaces de presentar soluciones adecuadas.

A pesar de las ventajas que representa contar con controladores adaptativos, existen algunas limitaciones que han impedido el desarrollo e implementación de estos en procesos reales. Entre estas, se encuentra la demora que implica el cálculo que se debe realizar para obtener los parámetros del controlador. Una de las soluciones planteadas, es la de contar con arrancadores que lleven al proceso a puntos de estabilidad, desde los cuales sea posible activar métodos de adaptación que optimicen la respuesta del sistema. Sin embargo, un reto mayor se presenta al no contar con este tipo de arrancador para el ajuste rápido en la primera instancia del control; para esto se hace necesario el desarrollo de sistemas que permitan a los controladores auto ajustarse de forma eficaz pero de forma eficiente; llegando a obtener los resultados deseados.

En este proyecto se presenta una arquitectura que permite la implementación de diferentes esquemas de control adaptativo, así como posibles combinaciones entre estos, haciendo posible el planteamiento y puesta a punto de soluciones rápidas, capaces de superar las demandas impuestas por la sociedad actual (precio, tiempo de salida al mercado, versatilidad, etc.).

2. CONTROL ADAPTATIVO

Existen muchos procesos en los cuales la dinámica y/o las características de las perturbaciones cambian continuamente, haciendo necesario el desarrollo de controladores adaptables, capaces de ajustar sus parámetros de forma continua. Estos controladores tienen una estructura especial, caracterizada por dos ciclos esenciales; por un lado, el ciclo de realimentación normal, en el cual se encuentra el controlador (de parámetros ajustables), y el proceso o planta; y por el otro, se encuentra el ciclo encargado del ajuste de los parámetros del controlador. Es en este último ciclo en el que se diferencian realmente los esquemas de control adaptativo. Algunos de los cuales son: Programación de Ganancia, Sistema Adaptativo por Modelo de Referencia (*MRAS*), Regulador Auto-Ajustable (*STR*) y Control Dual o Estocástico.

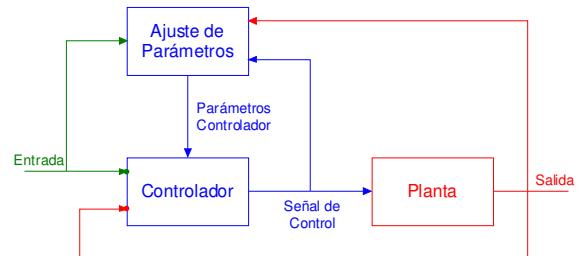


Fig. 1. Diagrama de Bloques de un Sistema Adaptativo Generalizado

2.1 Regulador Auto-Ajustable (*STR*):

En este esquema de control adaptativo, los parámetros del proceso son estimados en línea (recursivamente), para con éstos, ajustar los parámetros del controlador. El ciclo de ajuste, entonces, puede representarse separado en dos bloques; uno, encargado de la identificación, y otro, encargado del diseño.

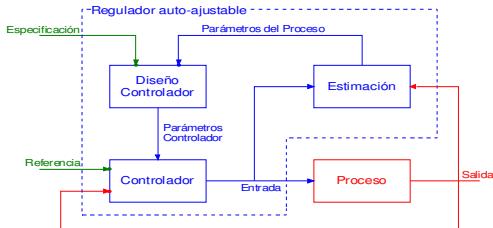


Fig. 2. Diagrama de Bloques de un *STR*

3. IDENTIFICACIÓN DE SISTEMAS

En la estructura de los *STR* (Figura 2.) se evidencia la importancia que tiene el bloque de Identificación, ya que de él, depende que el ajuste del controlador se realice de forma adecuada. Para lograr que los parámetros del proceso sean estimados correctamente, es necesario seleccionar la estructura del modelo que mejor se ajuste a las características reales del proceso.

La representación matemática del modelo, puede ser escrita como

$$\begin{aligned} y(t) &= -a_1 y(t-1) - a_2 y(t-2) - \dots - a_n y(t-n) \\ &\quad + b_0 u(t-d_0) + \dots + b_m u(t-d_0-m) \\ &= \varphi^T(t-1) \end{aligned} \quad (1)$$

donde:

$$\begin{aligned} \theta^T &= \begin{pmatrix} a_1 & a_2 & \dots & a_n & b_0 & \dots & b_m \end{pmatrix} \\ \varphi^T(t-1) &= \begin{pmatrix} -y(t-1) & \dots & -y(t-n) & u(t-d_0) \end{pmatrix} \end{aligned}$$

3.1. Estimador Recursivo por mínimos Cuadrados con Factor de Olvido

Es utilizado para la estimación recursiva de los parámetros en λ , y está dado por el siguiente conjunto de ecuaciones:

$$\begin{aligned} \hat{\theta}(t) &= \hat{\theta}(t-1) + K(t) \epsilon(t) \\ \epsilon(t) &= y(t) - \varphi^T(t-1) \hat{\theta}(t-1) \\ K(t) &= P(t-1) \varphi(t-1) (\lambda + \varphi^T(t-1) P(t-1)) \\ P(t) &= (I - K(t) \varphi^T(t-1)) \frac{P(t-1)}{\lambda} \end{aligned} \quad (2)$$

donde:

$0 < \lambda \leq 1$, es el factor de olvido. A medida que este se acerca a 1, la estimación de los parámetros pierde sensibilidad, pero gana estabilidad.

$P(t)$ es la matriz proporcional a la covarianza. Al inicializarla con valores muy altos, la estimación inicial va a converger rápidamente.

La identificación por medio de este método, se utiliza para procesos cuyos cambios se realizan de forma lenta.

3.2. Ejemplo: Motor DC.

Como prueba, se utilizó el modelo de velocidad angular de un motor DC. Con función de transferencia es:

$$\frac{\omega}{V} = \frac{K}{(Js+b)(Ls+R)+K^2} \quad (3)$$

donde:

J [kg.m²/s²]: Momento de inercia del rotor.

b [Nms]: Coiciente de amortiguación del sistema mecánico.

K [Nm/Amp]: Constante de fuerza electromecánica.

R [ohm]: Resistencia eléctrica.

L [H]: Inductancia eléctrica.

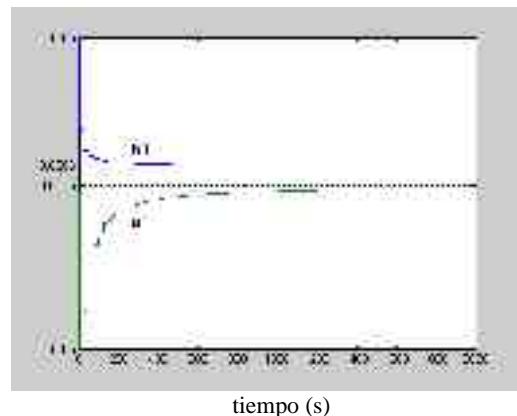
V [V]: Voltaje de entrada.

[rad/s]: Velocidad angular del eje.

Se utilizó un motor con los siguientes valores: $J=0.01$, $b=0.1$, $K=0.01$, $R=1$, $L=0.5$, y con un tiempo de muestreo de 0.5 segundos, se obtuvo la siguiente función de transferencia discreta:

$$\frac{y(t)}{u(t)} = \frac{0.05417z + 0.008598}{z^2 - 0.3742z + 0.002479} \quad (4)$$

De donde se obtiene: $b1=0.05417$, $b2=0.008598$, $a1=-0.3742$ y $a2=0.002479$.

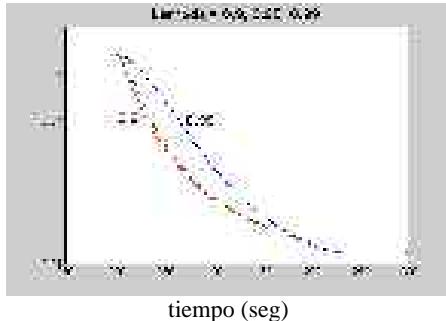


Gráf. 1. Valor numérico de la estimación de los parámetros a_1 y b_1 del motor como función del tiempo transcurrido.

Es posible apreciar en la Gráfica 1 que la convergencia es relativamente lenta, ya que se estabiliza aproximadamente en 1000s. Si cambiamos los valores de inicialización,

aumentando $P(0)$ o disminuyendo λ , la estimación se realizará mucho más rápido.

Ahora, si se introduce un cambio iniciando en $t=1000$ s hasta $t=1200$ s. Se obtiene el siguiente resultado



Gráf. 2. Valor numérico de b_1 en función del tiempo transcurrido. Para $J(t < 1000) = 0.001$, $J(1000 \leq t < 1200) = 0.001 + 0.0005t$, $J(2000 < t) = 0.11$

En la Gráfica 2 se observa como el valor de λ , afecta la respuesta del algoritmo (2). Vemos sin embargo, que debido a que la variación de J es lenta, la estabilidad del sistema se conserva.

En el dispositivo desarrollado se inicializan las variables de forma manual, aunque es posible mantenerlos con datos preestablecidos. Esto permite utilizar el conocimiento que se tenga del proceso para mejorar la respuesta del controlador.

4. MDPP¹ CON CANCELACION DE CEROS

Se escoge un controlador lineal R, S, T del tipo mostrado en la Figura 3.

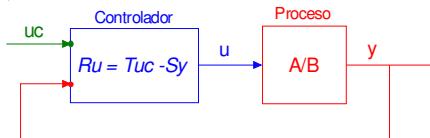


Fig. 3. Controlador Lineal Generalizado

Donde: $\frac{A}{B}$ representa la función de transferencia discreta

del proceso,

$$A(q)y(t) = B(q)u(t) \quad (5)$$

$$A(q) = q^n + a_1 q^{n-1} + \dots + a_n$$

$$B(q) = b_0 q^m + b_1 q^{m-1} + \dots + b_m$$

El sistema de lazo cerrado es:

$$y(t) = \frac{BT}{AR+BS} u_c(t) \quad (6)$$

$$u(t) = \frac{AT}{AR+BS} u_c(t) \quad (7)$$

Para sintonizar el controlador, deben encontrarse R y S , tal que, el denominador de la Ecuación (6) tenga las características deseadas.

4.1 Diseño del controlador

Para realizar el diseño debe tenerse la función de transferencia de lazo cerrado deseada, expresada por los polinomios A_m y B_m , y además un polinomio de observación A_0 . Estas son introducidas por el usuario al momento de iniciar el programa de control.

Teniendo entonces la ecuación:

$$A_m y(t) = B_m u_c(t) \quad (8)$$

Es posible entonces iniciar el diseño del controlador.

Primero, es necesario verificar las siguientes condiciones de compatibilidad:

$$\deg A_m = \deg A, \quad \deg B_m = \deg B,$$

$$\deg A_0 = \deg A - \deg B^+ - 1$$

$$B_m = B^- B_m'$$

En el caso del dispositivo desarrollado, esta verificación se realiza internamente para evitar posibles errores por parte del usuario; permitiéndole únicamente el ingreso de valores disponibles para estos polinomios.

El polinomio numerador en la función de transferencia de lazo abierto debe factorizarse de la siguiente manera, $B = B^+ B^-$ con B^+ mónica.

Se soluciona la ecuación de Diophantine, teniendo en cuenta que $\deg S < \deg A$

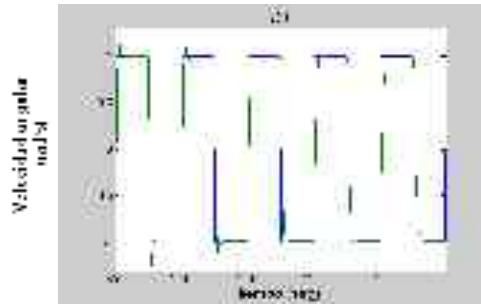
$$AR' + B^- S = A_0 A_m \quad (9)$$

Y se obtienen los valores de los polinomios del controlador

$R = R' B^+$ y $T = A_0 B_m'$. Con éstos se calcula la señal de control.

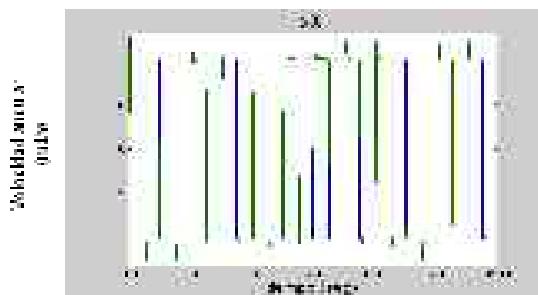
¹Minimum Degree Pole Placement

Todo el proceso del diseño del controlador se realiza utilizando los datos provenientes de la identificación, asumiendo que los parámetros estimados por ésta son los reales. Antes de llegar a obtener la respuesta deseada, el sistema pasa por un estado transitorio, en el cual, la señal de control oscila entre los valores máximo y mínimo permitidos. Para probar el controlador se utilizó la función de transferencia del motor DC expresada en la Ecuación (3).



Gráf. 3. Velocidad angular del motor DC en función del tiempo $A_m=\{1 -1 0.4\}$ Para $J(t<1000)=0.001$, $J(1000<=t<1200)=0.001+0.0005t$, $J(2000<t)=0.11$

En la Gráfica 3² se aprecia como el comportamiento de la respuesta del sistema en lazo cerrado se ve afectado por la variación de J . La respuesta que se observa entre 900 y 1000 s es la deseada. Ahora bien; si se utiliza el modulo de identificación para generar los parámetros de A y B necesarios para el diseño, obtenemos la Gráfica 4. En la cual, se observa un período de transición mientras los estimadores de los parámetros del modelo se estabilizan. Luego de este tiempo, la respuesta que se obtiene del proceso se mantiene igual a la deseada. Aquí se evidencian claramente las ventajas del control adaptativo; que permitió, a pesar de los cambios en el proceso, mantener “constante” la respuesta de lazo cerrado.



Gráf. 4. Velocidad angular en función del tiempo. Se utilizó el algoritmo completo para control *STR*

² Las variables del algoritmo de identificación utilizado para obtener la Gráfica 3, son: $\theta^T = \begin{pmatrix} 0.02 & 0.1 & 0 & 0 \end{pmatrix}$, $P_{1,1}(0)=P_{2,2}(0)=P_{3,3}(0)=P_{4,4}(0)=100$ y $\lambda=1$

Las variables del algoritmo de identificación utilizado para obtener la Gráfica 3, son:

$$\theta^T = \begin{pmatrix} 0.02 & 0.1 & 0 & 0 \end{pmatrix}, \\ P_{1,1}(0)=P_{2,2}(0)=P_{3,3}(0)=P_{4,4}(0)=100 \text{ y} \\ \lambda=1.$$

5. SISTEMAS EMBEBIDOS³

Uno de los principales impulsos para el desarrollo del control adaptativo, provino de los rápidos avances tecnológicos en los microprocesadores, que permitieron implementar las leyes asociadas a este tipo de control.

El siguiente paso es entonces la utilización de sistemas embebidos como plataforma para este tipo de desarrollo. Permitiendo probar al máximo las capacidades de controladores adaptativos especializados. La Arquitectura escogida para implementación del controlador adaptativo se encuentra conformada por la tarjeta XPORT 2.0 [12] y el GameBoy Advanced (GBA) de Nintendo [13], y cuyo diagrama de bloques se presenta en la Figura 4..

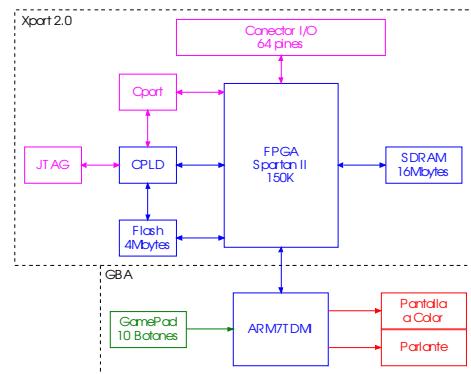


Fig. 4. Diagrama de Bloques Xport 2.0 + GBA

5.1. Tarjeta de expansión

Esta tarjeta fue diseñada para brindar la capacidad de trabajar como un sistema de control adaptativo completo, es capaz de recibir información por parte del usuario e interactuar con este; y lo más importante, permite utilizar las diferentes técnicas de control adaptativo con el fin de buscar las soluciones más eficientes al momento de controlar cualquier proceso.

³ sistemas electrónicos basados en el diseño conjunto de Hardware (HW) y Software (SW), desarrollados para cumplir tareas específicas.

5.2. Interfaz con en usuario

El sistema de control desarrollado en este proyecto (*STR*) no es ciento por ciento autónomo; ya que es necesario que le sean introducidos ciertos parámetros que permitan el funcionamiento tanto del bloque de identificación como el de control. Es posible sin embargo, que algunos de éstos, sean utilizados en valores predeterminados, dependiendo del tipo de proceso que se desea controlar y de la respuesta que se desea obtener del sistema como tal.

5.3. Interfaz con el proceso

El algoritmo de identificación *LSE* por un lado necesita tener los datos de la entrada “*ut*” y de la salida del proceso “*y*”; y el de control por el otro, requiere además de la referencia “*uc*”. Para la obtención de estos datos, se utilizó un conversor análogo-digital de 8 canales con una precisión de 8 bits por dato.



Fig. 5. Plataforma Culminada.

6. RESULTADOS Y CONCLUSIONES

- Se desarrolló e implementó una tarjeta de expansión, que puede ser utilizada junto al GBA y el XPORT para probar sistemas de control discreto tanto adaptativos como tradicionales.
- Se implementó por completo un controlador *STR* con capacidad para manejar procesos hasta de quinto orden.
- Existen otras aplicaciones de las que se puede tomar provecho, como lo son: interfaz GBA-teclado, visualización de señales análogas (7 en total), librerías para control *STR*, salida análoga desde el GBA, etc.
- Se diseñaron en MATLAB librerías destinadas al control adaptativo *STR*; tanto para la identificación como para diseño de controladores lineales.
- Existen algunos problemas debidos a la cuantización de las señales, que resultan más evidentes a medida que se disminuye el factor de olvido.

- El tiempo de muestreo debe encontrarse por encima de 0.1s. Los cálculos que se realizan en el procesador, toman demasiados ciclos de máquina. A pesar de que se realizó una optimización de las operaciones, no fue posible disminuir este tiempo. Para aumentar disminuir el tiempo de muestreo, se deben realizar algunas operaciones en Hardware dentro de la FPGA.
- Del tiempo requerido para el procesamiento, la identificación utiliza aproximadamente el 70%.

8. REFERENCIAS

- [1] Astrom K. J. and B. Wittenmark, 1995. *Adaptive Control*, 2^a edición. Addison-Wesley.
- [2] Sastry S. and M. Bodson, 1989. *Adaptive Control: stability, convergence and robustness*. Englewood Cliffs, N.J.: Prentice-Hall.
- [3] Ljung L., 1998. *System Identification: Theory for the User*. 2^a edición. Prentice-Hall.
- [4] Ogata K., 1994. *Discrete-Time Control Systems*. 2^a edición. Pearson Education.
- [5] Vélez C. M., 2003. *Control Adaptativo aplicado a un Péndulo Invertido*. Universidad EAFIT. Medellín.
- [6] Rodríguez, Rafael. *Sistema Operativo en Tiempo Real (RTOS) para MicroPP-ARM*.
- [7] Charmed Labs., 2003. *Xport 2.0 User Guide*. Versión 1.4. Disponible en: www.charmedlabs.com.
- [8] Charmed Labs., *Xport 2.0 Custum Configuration Tutorial*. Versión 2.2. Disponible en: www.charmedlabs.com.
- [9] University of Michigan., 1997. *Control Tutorials for Matlab*. Disponible en:
<http://www.engin.umich.edu/group/ctm/examples/examples.html>
- [10] Programa para el curso de *Adaptive Control*. Lund Institute of Technology. Disponible en:
<http://www.control.lth.se/~kursar/>
- [11] A course in *Neural Networks for System Identification and Control*. Technical University of Denmark. Disponible en:
<http://www.iau.dtu.dk/nnbook/Files/course5.pdf>
- [12] www.charmedlabs.com
- [13] www.nintendo.com

Apéndice I

Metodología de Implementación de Autómatas Celulares en FPGAs

Publicado en: Memorias del X Workshop de Iberchip ISBN 959-261-105-X.

METODOLOGIA DE IMPLEMENTACION DE AUTOMATAS CELULARES EN FPGA

J. Sepúlveda[†], C. Camargo^{}, S. Bolívar[†]*

[†] Estudiantes Ingeniería Electrónica

^{*} Profesor Departamento de Ingeniería Eléctrica & Electrónica
Universidad Nacional de Colombia, Bogotá
{marthase, ccamargo}@gmun.unal.edu.co

ABSTRACT

The cellular automata are dynamic systems inspired in nature developed models. This structure is capable of data processing and has the potential of showing a complex behavior, considering simple performance's rules. In this article, the method developed in order to solve a problem (autonomous navigation of a robot) applying this type of bioinspired system implemented in a FPGA is presented.

RESUMEN

Los Autómatas celulares son sistemas dinámicos inspirados en los modelos desarrollados por la naturaleza. Esta estructura es capaz de procesar información y de exhibir comportamientos complejos, teniendo en cuenta unas reglas de conducta simples. En este artículo, se presenta la metodología desarrollada para resolver un problema (navegación autónoma de un robot) empleando este tipo de sistema bioinspirado implementado en un FPGA.

METODOLOGIA DE IMPLEMENTACION DE AUTOMATAS CELULARES EN FPGA

J. Sepúlveda[†], C. Camargo^{}, S. Bolívar[†]*

[†] Estudiantes Ingeniería Electrónica

^{*} Profesor Departamento de Ingeniería Eléctrica & Electrónica
Universidad Nacional de Colombia, Bogotá
{marthase, ccamargo}@gmun.unal.edu.co

RESUMEN

Los Autómatas celulares son sistemas dinámicos inspirados en los modelos desarrollados por la naturaleza. Esta estructura es capaz de procesar información y de exhibir comportamientos complejos, teniendo en cuenta unas reglas de conducta simples. En este artículo, se presenta la metodología desarrollada para resolver un problema (navegación autónoma de un robot) empleando este tipo de sistema bioinspirado implementado en un FPGA.

1. INTRODUCCION

Motivados por el comportamiento de los organismos vivos, se desarrolló el concepto del autómata celular. Se emplean con el fin de describir un sistema complejo sin la necesidad de resolver ecuaciones complejas, en lugar de esto, la solución emerge debido a la interacción de individuos que siguen unas reglas simples [1,2,3]. Debido a la facilidad de comprensión, modelamiento y simulación, este tipo de sistemas se utilizan para generar simulaciones de sismos [4], modelar vida artificial [5,6], modelar evolución de un sistema biológico (hardware evolutivo) [7], para generar secuencias de números aleatorios [8] etc.

Los Autómatas Celulares son de naturaleza paralela y pueden emplearse en los sistemas de cómputo de alto desempeño. Los dispositivos lógicos reconfigurables y los lenguajes de descripción de Hardware se presentan como una buena alternativa para la implementación de sistemas digitales complejos. Por tal motivo resulta de gran interés plantear una metodología de diseño eficiente y capaz de aprovechar de una mejor forma las características y ventajas de los Autómatas celulares.

El artículo se divide en cuatro secciones. La primera describe el principio de funcionamiento de los autómatas

celulares. En la segunda sección se describe un problema a resolver empleando el autómata celular. En la tercera sección se realiza la descripción de alto nivel del problema. En la cuarta parte se plantea la arquitectura final de la implementación del sistema de navegación y por último se generan conclusiones.

2. AUTOMATAS CELULARES

La idea del autómata celular surge como un intento de crear un modelo matemático abstracto de la reproducción biológica [9].

En un sentido estricto, los autómatas celulares son un arreglo n-dimensional de unidades llamadas células, que poseen una cantidad de estados finita y que están interconectadas entre sí de una manera conocida [10]. El estado actual de la célula cambia según una regla, (para el sistema en general o para cada célula de la estructura) que define la norma por la cual se manipula la interacción con las células interconectadas (que conforman el vecindario) a la célula que está siendo evaluada. La regla puede definirse de tres formas: como tabla de verdad (método no totalístico), que indica todos los posibles estados del vecindario y el valor que tomará la célula evaluada, el método totalístico (toma en cuenta la cantidad de células con estado uno lógico en un vecindario) y el método semi-totalístico (tienen en cuenta configuraciones especiales del vecindario, creando plantillas) [11]. Si el vecindario de la célula evaluada no presenta cambios, ésta deberá mantener su estado.

Por las características anteriormente descritas, éste sistema es dinámico complejo, en el cual el espacio y el tiempo son discretos. Este modelo computacional puede ser categorizado como complejo, ya que es un sistema compuesto de elementos simples que interactúan de una manera definida y poseen un comportamiento emergente complejo [1,2,3].

La actualización de una estructura celular, se realiza teniendo en cuenta el estado de los vecinos y la regla que se ha definido.

Para lograr una mejor comprensión, se muestra un ejemplo de un autómata celular en la figura 1. Este es un arreglo unidimensional de 12 células que conforman un autómata binario (las células pueden tener dos estados cero y uno lógico, blanco o negro). Cada célula está interconectada con sus vecinos inmediatos de la izquierda y la derecha.

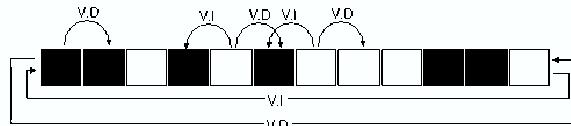


Figura 1. Autómata unidimensional binario

Para la primera célula, su estado siguiente dependerá del estado de la segunda y de la última (sus vecinos de la izquierda y la derecha), el estado siguiente de la segunda célula dependerá entonces de la primera y la tercera, y así sucesivamente. Después de la evolución (un paso discreto de tiempo) todo el autómata cambiará hacia su siguiente estado.

La evolución del autómata consiste en la evaluación de la regla de cada célula y el posterior cambio de estado de las células que conforman al sistema cuando transcurre un “cierto” tiempo (el tiempo se considera discreto). Si, por ejemplo, definimos una regla para el autómata de la figura 1, la cual hace que la célula sobreviva (se encuentra en 1 lógico y conserva éste estado) o nazca (cambia de estado 0 a 1 lógico) cuando sus vecinos tienen el mismo estado y muera (cambia de estado 1 a 0 lógico) en caso contrario. Esta regla consiste, básicamente, en realizar la operación lógica NOR exclusiva (la cual da como resultado un nivel lógico bajo cuando una entrada es el complemento de la otra). La evolución de esta regla para el autómata de la figura 1, se muestra en la figura 2.

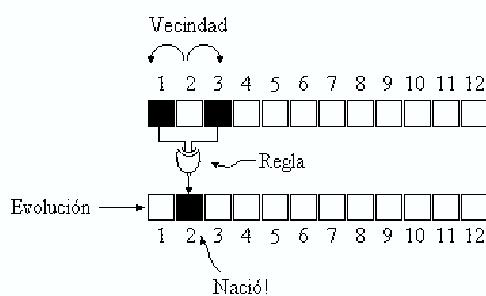


Figura 2. Representación de la evolución de un autómata unidimensional binario con la regla XNOR

Con la evolución del espacio celular, se procesan los datos, lo cual genera patrones que pueden ser interpretados y acondicionados para modelar un sistema.

Para emplear este sistema en la resolución de un problema se debe tener en cuenta los dos elementos característicos de un autómata celular: Reglas y conexiones entre los elementos. Modificando estas características, es posible modelar diferentes sistemas.

3. DESCRIPCIÓN Y MODELAMIENTO DEL SISTEMA

Como ejemplo de aplicación de los autómatas celulares, se propone realizar un control de navegación de un robot móvil (capacidad de moverse satisfactoriamente y sin intervención humana dentro de ambientes que no han sido específicamente dispuestos) que posee cuatro sensores y dos motores (servomotores). Ver figura 3.

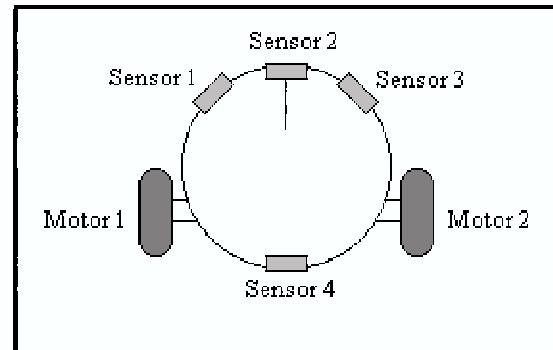


Figura 3. Vehículo controlado con el autómata celular. El robot tiene cuatro sensores y dos motores.

Cada sensor (ver Figura 3) produce un voltaje entre 0 y 3 voltios, este voltaje representa la distancia a un determinado objeto, cuando no encuentra ningún obstáculo la salida del sensor es igual a cero. Si encuentra un obstáculo dentro de su umbral más próximo la lectura va a ser igual a tres voltios.

Para realizar la navegación del robot se debe tener en cuenta que si el robot no detecta obstáculos en su camino, debe seguir en esa dirección empleando la máxima velocidad, que corresponde a una frecuencia determinada. Por otra parte si detecta algún obstáculo la velocidad de un motor disminuirá proporcionalmente a la distancia y girará debido a la diferencia de velocidades de sus motores. En la figura 4 se muestra el diagrama de flujo del control de navegación del robot.

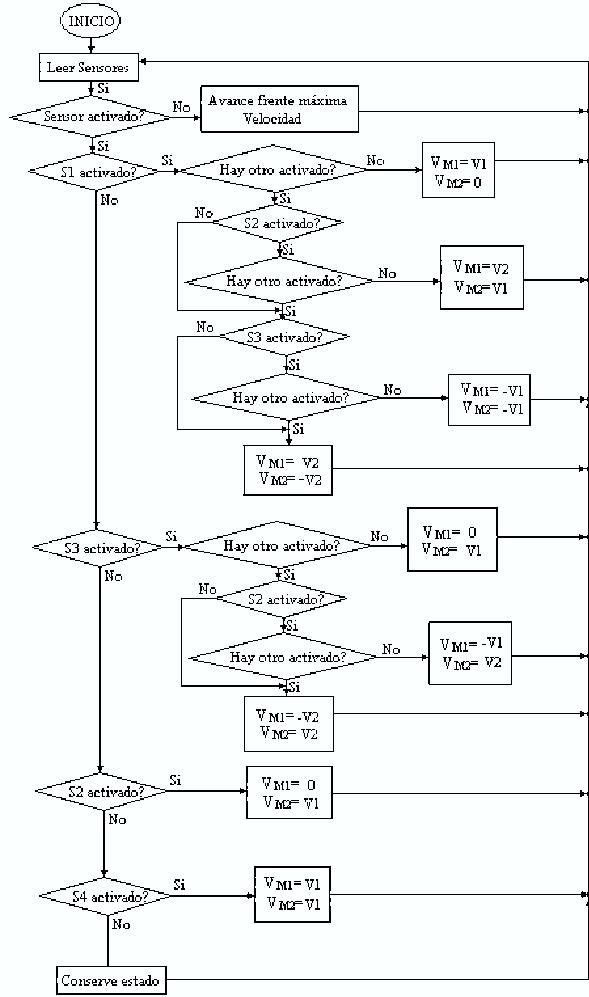


Figura 4. Diagrama de flujo del sistema de navegación del robot de la figura 3.

V2 y V1 representan velocidades de los motores (V2 es mayor que V1). Con las combinaciones de estas velocidades es posible representar gran variedad de giros hacia la izquierda y derecha. El signo negativo hace referencia al sentido del giro.

Una vez descrito el problema, se debe modelar teniendo en cuenta las características del autómata celular de tal forma que se pueda encontrar una representación óptima. Este modelo se puede obtener describiendo el sistema en términos de ecuaciones de diferencias, caso en el cual se puede emplear un autómata en donde cada célula represente el estado de la función en un tiempo t . Si el sistema es muy complejo de modelar de esta forma, se debe describir y simular para hallar la mejor representación.

Para implementar el control de navegación, se emplea un autómata para crear un espacio bidimensional capaz de representar el entorno en el cual navega el robot de tal forma que se pueda hallar una regla que permita al autómata evolucionar y evitar obstáculos. Primero se debe discretizar el espacio en el cual se desenvuelve, de forma que se mapee el ambiente [12]. Para tal fin, el sistema sensorial del agente, se codificó de forma que los sensores 1 y 3 detectan un obstáculo cuando éste se encuentra a una distancia menor de 10 cm, mientras que los sensores 2 y 4 tienen un alcance de 30 centímetros, este rango se encuentra dividido en cuatro partes, como se muestra en la tabla 1.

Rango (cm)	Zona
4,0 - 7,5	I
7,5 - 15	II
15,0 - 22,5	III
22,5 - 30	IV

Tabla 1. codificación de los sensores

4. GENERACIÓN DE LA REGLA

Una vez analizado el sistema se deben seleccionar las características del espacio celular. Es decir, se especifica el número y forma de las células que van a formar parte de la población del autómata celular. Para esto se emplean herramientas de simulación las cuales ayudan a determinar la forma de las células, la regla del sistema y la interconexión de los elementos.

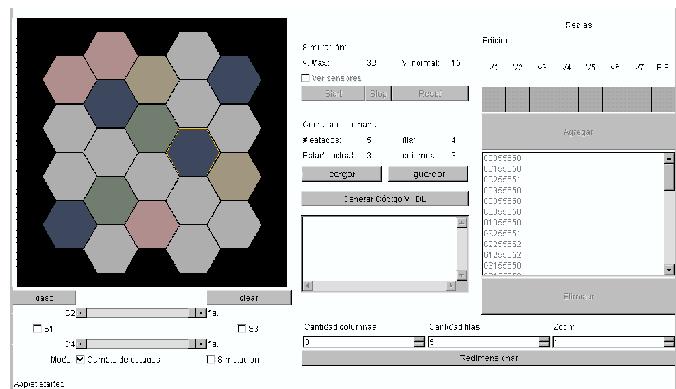


Figura 5. Ventana del simulador del autómata celular. Se utilizó el lenguaje de programación Java para la implementación del simulador, debido a su independencia frente a sistemas operativos y facilidad de manejo de interfaces gráficas, necesarias en este caso (Figura 5).

El simulador permite la edición del autómata: número de estados, reglas (en forma de tabla de verdad), conexión y la codificación de los sensores. Tiene la opción de simular el procesamiento del autómata y el movimiento del robot en una arena (Figura 6). Cuando el robot detecta los obstáculos modifica su movimiento siguiendo el resultado de la evolución del autómata.

Los autómatas son sistemas paralelos. Debido a que la simulación es de naturaleza secuencial se emplearon registros temporales que almacenaban el estado siguiente del autómata procesado. Después de un ciclo de evolución, el autómata virtual actualiza el estado de todas las células, simulando de esta forma el paralelismo.

Este simulador es capaz de generar el código en VHDL de las células que conforman la estructura celular.

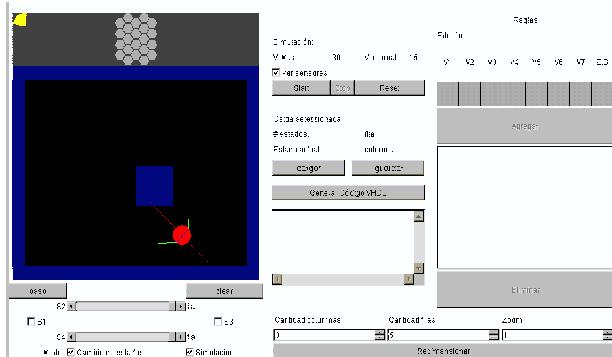


Figura 6. Ventana del autómata donde se muestra la arena donde navega el robot.

Se eligió un espacio bidimensional de células hexagonales debido a que se probó que este tipo de arreglo optimiza el espacio (número de células empleadas) y procesamiento de los datos (no emplea información redundante). En el espacio de células cuadradas, la información codificada resultaba redundante y se empleaban 40 células para lograr la misma resolución del sistema. Al cambiar la forma de la célula, no sólo se redujo el número de células a la mitad, sino también el vecindario de las mismas (de ocho células a seis). Todos estos cambios logran que el espacio (número de compuertas) empleado para la implementación del autómata celular disminuya, obteniendo los mismos resultados. La figura 7 muestra la codificación del espacio celular construido. Los sensores S1 y S3 (de umbral) están representados por una única célula y son parte del vecindario directo de las células que representan a los motores M1 y M2. Para expresar el valor de los sensores S2 y S4 se empleo la representación de la Tabla 1, por lo que se emplean 8 células para codificar el valor de cada uno de estos sensores.

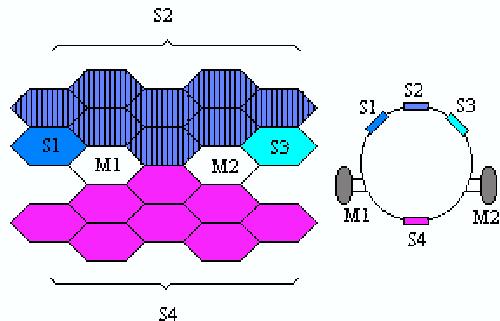


Figura 7. Codificación del espacio celular.

La figura 8 muestra en la estructura celular la codificación por zonas de los sensores S2 y S4. Se implementaron células de dos (IV), tres (S1, S3), cuatro (I, II, III) y cinco estados (M1 y M2) con reglas diferentes (autómata celular no uniforme). Dentro del sistema celular se pueden identificar cuatro tipos de células según las características de su vecindario:

Tipo 1: Son las células que poseen un vecino solamente. Dentro de este tipo se distinguen las células S1 y S3 que son realimentadas.

Tipo2: Estas células interaccionan con otras dos células.

Tipo 3: Este tipo de células no poseen vecindario y dependen únicamente del estado de la codificación de los sensores.

Tipo 4: Pertenece a este grupo las células que tienen un vecindario conformado por otras cinco células.

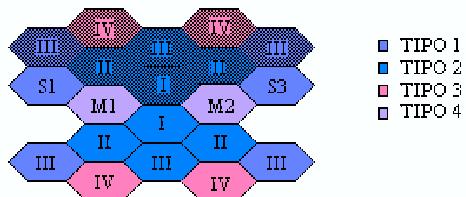


Figura 8. Representación de los tipos de células

Un ejemplo de la codificación de los sensores en el autómata celular se muestra en la figura 9. El obstáculo es detectado por el sensor infrarrojo ubicado en la parte frontal del robot (S2), la distancia a éste es de 25 centímetros, lo cual corresponde, según la tabla 1 a la zona IV, por lo cual las células que representan la zona IV del sensor S2 se activan. En la figura se observa el mapeo que realiza el robot del entorno sobre el autómata celular y la representación interna del obstáculo.

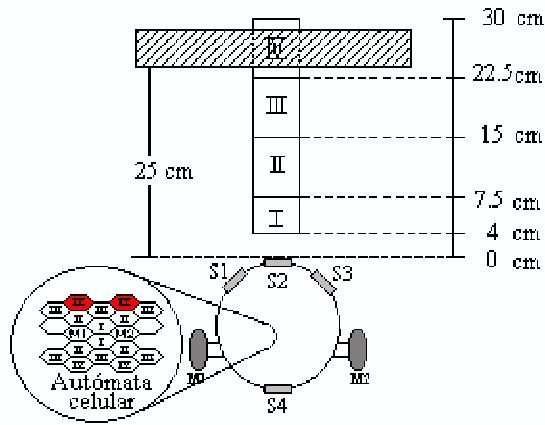


Figura 9. Representación interna de un obstáculo.

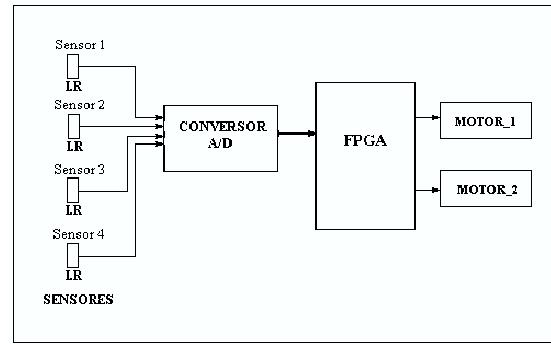


Figura 10. Esquema general de la implementación del sistema de navegación.

La figura 10 muestra la arquitectura de la implementación final, la cual está formada por un conversor A/D que se encarga de codificar las lecturas de los sensores en cadenas de cuatro bits para ser procesados por el FPGA. El FPGA contiene el autómata celular, se encarga de procesar y dar un resultado que se asigna a los dos motores.

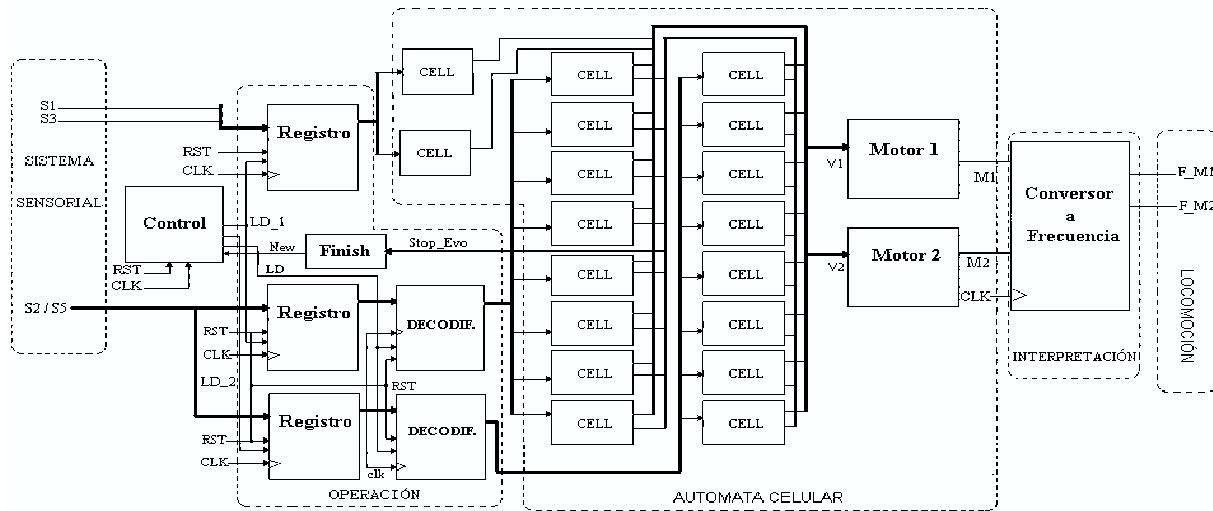


Figura 11. Implementación del sistema de navegación

5. IMPLEMENTACION DEL AUTOMATA CELULAR EN FPGA

La figura 11 muestra el diagrama de bloques de la implementación del control de navegación.

La estructura básica del sistema de navegación del robot consiste en un ciclo que inicia con el sistema sensorial del robot (lecturas de los cuatro sensores del robot), luego se

realiza el proceso de operación, encargado de la interacción entre el robot y los sensores (fija el estado inicial del autómata celular). Una vez fijado el estado actual del robot, el autómata celular planea la ruta a seguir teniendo en cuenta el objetivo de la misión (evitar obstáculos) y asignar un valor, el cual, mediante el proceso de interpretación se convierte en una señal que controla los motores. Una vez se ha tomado la decisión respecto a la ruta a seguir y se le asigna a los motores la frecuencia determinada el ciclo se repite.

En detalle, la lectura de los sensores es codificada y enviada al FPGA, en donde el bloque CONTROL se encarga de manipular la carga de estos datos. Cada vez que recibe una lectura del sistema sensorial completo, éste módulo habilita al DECODIFICADOR el cual inicializa las células del autómata celular. Luego el bloque CONTROL se encarga de comenzar el proceso de evolución de los diferentes elementos del autómata. Una vez se ha encontrado la respuesta, el bloque CONVERSOR FRECUENCIA genera las diferentes frecuencias que manipulan los motores del robot. El bloque FINISH recibe información de los estados de las células y se activa cuando el autómata no cambia de estado (se ha finalizado el proceso). Este envía una señal al módulo CONTROL para que detenga la evolución y el proceso se vuelve a repetir cargando en los bloques REGISTRO la información de los sensores.

La Figura 12 muestra el algoritmo general, como diagrama de flujo, correspondiente a la implementación del autómata celular en el FPGA. En éste se distinguen tres etapas: lectura de sensores, Evolución del autómata celular y transmisión de datos.

5.1 Bloque CONTROL

Este bloque se encarga de implementar el algoritmo general presentado en la figura 12, controla los procesos de adquisición y almacenamiento de las lecturas de los sensores, así como de la inicialización y evolución de las células. La Figura 13 es el diagrama de la máquina de estados diseñada para implementar el control.

Una vez se activa el reset global del sistema, el control inicializa los demás bloques asignando a las señales de los registros (LD_1, LD_2), decodificadores (LD), células (Evo_env, Rst_Cell, Stara_evo, LD_Reg) y del conversor A/D (Start, ADD, ALE, OE) un nivel lógico bajo. El control continúa en este estado a menos que la señal DIV (proveniente del módulo FINISH) se active, lo cual indica que la información anterior se ha procesado y se ha transmitido la información a los motores. Esta señal

permite que se inicie de nuevo el ciclo de captura, procesamiento y transmisión de datos. Ver Figura 13.

El Control activa la señal ALE, START, OE, LD_1 y LD_2, lo cual permite la codificación de las lecturas de los sensores y su posterior almacenamiento en los registros.

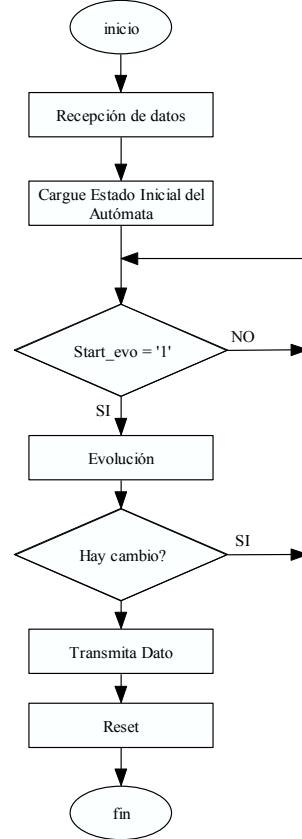


Figura 12 Diagrama de Flujo de la implementación del Autómata Celular en el FPGA.

Una vez capturados los datos, se activa la señal LD, que permite que los bloques DECODIFICADOR reciban los datos del registro y se codifiquen debidamente para que se inicialice el estado de las células y se mapee el entorno en el cual se desenvuelve el robot. Una vez realizada la codificación, por medio de la señal EVO_ENV se carga la configuración del autómata y con la señal START_EVO se comienza el proceso de evolución y culmina cuando el autómata no cambia más de estado, inicializándose de nuevo el ciclo. Una vez realizado este procedimiento, el control queda en espera de la recepción de otro dato para repetir el ciclo.

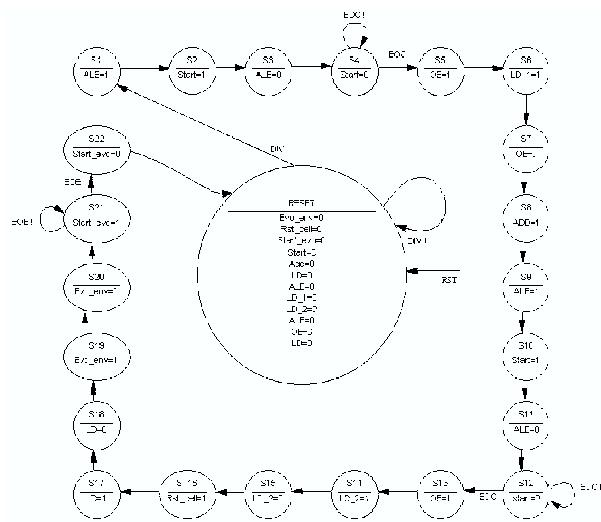


Figura 13. Máquina de estado empleada para el bloque Control.

5.3 Células

Este bloque se encarga de procesar el valor de las lecturas de los sensores según la regla implementada y del vecindario de cada célula del autómata celular. La figura 14 muestra el diagrama de bloques general de la célula

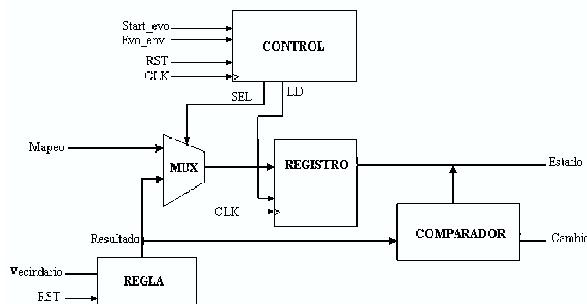


Figura 14 Diagrama general de la célula

Regla: Implementa la regla que se encontró empleando el simulador. Este depende del vecindario y para algunas células, puede depender del estado actual de la célula evaluada (Células tipo I).

Control: Este módulo se encarga de manejar del sincronismo al interior de la célula (carga de registros) e implementar el diagrama de flujo de la figura 15.

Multiplexor: Se emplea para poder seleccionar entre el estado generado del mapeo de los sensores (dato

proveniente del DECODIFICADOR) y el generado por la interacción con el vecindario celular.

Comparador: Se encarga de comparar el valor de la evolución anterior con la actual, para de esa forma determinar cuando la célula llega a un estado estático.

La diferencia de la implementación de cada tipo de célula, radica en el bloque regla y el vecindario. Una vez establecido el valor que deben tomar los motores, se emplea el bloque DIVISOR para generar la frecuencia correcta y transmitirla a los motores para que logren la navegación.

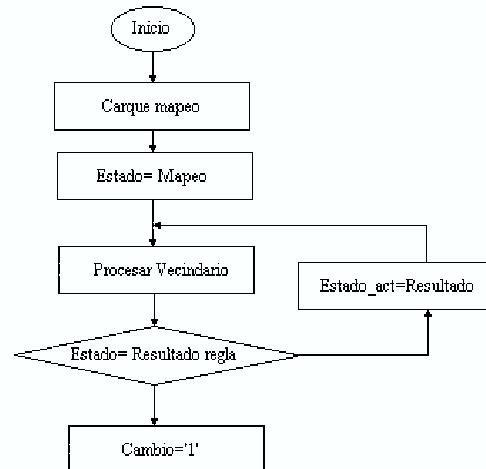


Figura 15 Diagrama de flujo de la célula

6. CONCLUSIONES

El sistema de navegación del robot se implementó no solo sobre el FPGA sino también se probó el autómata celular en un microcontrolador Motorola MC68HC908JK3. El algoritmo de implementación del autómata celular empleado se muestra en la figura 16.

Al comparar los dos sistemas de navegación se obtuvieron los siguientes resultados:

- El tiempo de procesamiento empleado por el FPGA es de 41 ciclos de reloj, mientras que en la implementación del microcontrolador se necesitaban 6419 ciclos de reloj para realizar tres evoluciones del autómata celular.

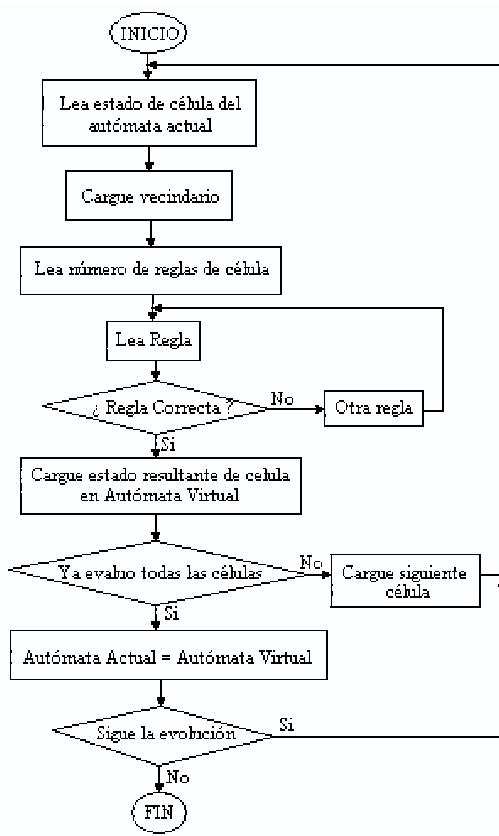


Figura 16. Algoritmo de implementación del Autómata Celular sobre el microcontrolador.

- En la implementación del sistema de navegación en el microcontrolador se necesitaban dos autómatas celulares, uno real y uno virtual, para conservar intacto el primero y realizar adecuadamente el proceso de evolución, almacenando los resultados en el autómata virtual.

A pesar de la diferencia de tiempo de procesamiento de datos, el desempeño del sistema de navegación no se ve afectado, pues aun empleando el microcontrolador, el robot es capaz de detectar obstáculos móviles. En conclusión, debido a la baja velocidad del movimiento del robot, la implementación del sistema de navegación no permite observar la potencialidad del procesamiento paralelo de información, ya que se pueden obtener los mismos resultados utilizando un microcontrolador en esta aplicación.

Sin embargo se mostró la ventaja de utilizar FPGAs para la implementación de un Autómata Celular ya que se aprovecha la característica de procesamiento paralelo de

información de éste sistema, obteniendo resultados en muy pocos ciclos de reloj. Además de disminuir el tiempo requerido para el cálculo del estado del AC, el sistema en la FPGA puede funcionar a una frecuencia de 50 MHz, mientras que el microcontrolador tiene una frecuencia de operación máxima de 8 MHz.

Por otro lado se propuso una metodología general para resolver un problema empleando los autómatas celulares. Se mostró como se puede obtener un comportamiento complejo formulando reglas simples, las cuales se emplean para describir la interacción de las células del autómata.

En general la metodología propuesta se puede resumir en el diagrama de flujo de la figura 17.

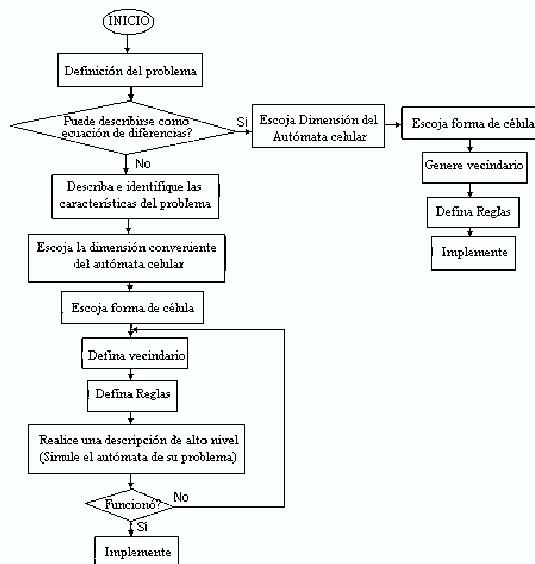


Figura 17 Metodología de diseño empleando autómatas celulares para la solución de un problema.

7. REFERENCIAS

- [1] Gwynne, P., G. Page, G.: “Microarrays devices”. Online : science-education.nih.gov/newsnapshots/TOC_Chips/toc_chips.html
- [2] M. Mitchell, J. Crutch, P. Hraber.: “Dynamics Computation and the Edge of Chaos A Re Examination”. G. Cowan, D. Pines, D. Melzner editors Integrative Themes Santa Fe Institute Studies in the Sciences of Complexity. Proceedings Volume 19. Reading, MA Addison-Wesley.

[3] W. Hordijk, J. Crutch, M. Mitchell.: “*Embedded Particle Computation in Evolved Cellular Automata*”. Pre-proceedings of Physics and Computation.

[4] J. Contreras.: “*FBF, un programa para la simulación de fallamiento en 2D y construcción de secciones estructurales*”. Online: <http://www.cicese.mx/~juanc>.

[5] A. De Souza.: “ Simulações Computacionais e Vida Artificial”.

[6] G. Volpato, C. Júnior, M. Nagel1.: ”Vida Artificial”

[7] K. Tomoyoshi, T. Maruyama, Y. Yamaguchi, A. Konagaya.: “A Cellular Automata System with Reconfigurable Hardware: - Towards a Whole Cell Simulation -.

[8] P. Castillo, C. Alberth, P. Privat, R. Juan. “Autómatas Celulares”. Online: <http://www.spc.org.pe/boletines/2002/html/11/articulos/article2.htm>

[9] R. Das, J. Crutch, M. Mitchell, J. Hanson.: “*Evolving Globally Synchronized Cellular Automata*”. Proceedings of the Sixth International Conference on Genetic Algorithms. April 5, 1995.

[10] W. Hordijk, J. Crutch, M. Mitchell.: “*Mechanisms of Emergent Computation in Cellular Automata*”

[11] A. Wuensche.: “*Discrete Dynamics Lab: Tools for investigating cellular automata and discrete dynamical networks*”.

[12] C. Behring, M. Bracho, M. Castro, J. Moreno.: ”*An Algorithm for Robot Path Planning with Cellular Automata*”.