PLATAFORMA DE DESARROLLO PARA SISTEMAS EMBEBIDOS BASADA EN EL GAME BOY ADVANCE

Sergio Ricardo Banguero Torres

Mauricio Erazo Sierra

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERIA ELÉCTRICA ELECTRÓNICA Y TELECOMUNICACIONES

Bucaramanga, 2007

PLATAFORMA DE DESARROLLO PARA SISTEMAS EMBEBIDOS BASADA EN EL GAME BOY ADVANCE

Sergio Ricardo Banguero Torres

Mauricio Erazo Sierra

Trabajo de grado para optar por el título de Ingeniero Electrónico.

Director:

Msc. Jorge Hernando Ramón Suárez

Co-director:

Msc. Carlos Iván Camargo Bareño

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERIA ELÉCTRICA ELECTRÓNICA Y TELECOMUNICACIONES

Bucaramanga, Septiembre de 2007

Agradecimientos

Agradecemos especialmente a nuestras familias por su apoyo a lo largo de este camino.

A nuestros maestros que a lo largo de la carrera nos han brindado sus conocimientos; a Jorge Hernando Ramón Suárez, por compartir la idea de desarrollar este proyecto con nosotros, por su primordial soporte, colaboración y entusiasmo en el desarrollo del mismo; a Carlos Iván Camargo Bareño por su paciencia y por proporcionar elementos y conocimientos indispensables para la realización del proyecto y a Elkim Felipe Roa por su desinteresada colaboración.

RESUMEN

TITULO: PLATAFORMA DE DESARROLLO PARA SISTEMAS EMBEBIDOS BASADA EN EL GAME BOY ADVANCE[1]

AUTORES: BANGUERO TORRES, SERGIO RICARDO, y, ERAZO SIERRA, MAURICIO[2]

PALABRAS CLAVES: Sistema Embebido, Game Boy Advance, GBA, ARM, PCB.

DESCRIPCION:

Este trabajo de grado desarrolla una tarjeta de expansión con arquitectura de cartucho de video juego compatible con el Game Boy Advance (GBA). Esta tarjeta permite el acceso a los múltiples recursos que ofrece la consola añadiendo la posibilidad de lógica externa programable por medio de una FPGA, otorgando libertad al desarrollador para el uso de los mismos y abriendo la posibilidad de interconexión para interfaces con variables externas.

El diseño del sistema está basado en desarrollos anteriores y pretende mediante la práctica de la reproducción apropiarse de la tecnología que está contenida en este tipo de plataformas. El desarrollo del trabajo comienza detallando las especificaciones de la consola para luego comprender el funcionamiento a bajo nivel de la versión comercial más popular del sistema de expansión. Después de generado un diagrama de interconexión lógico y físico de los dispositivos pertinentes se prosigue al diseño y fabricación del circuito impreso (PCB) en el que se albergará el sistema, para finalmente desarrollar una aplicación sencilla que demostrativamente enseñe su funcionamiento tanto a nivel hardware como a nivel software, haciendo hincapié en las herramientas de desarrollo, todas de dominio público.

Con la ejecución de este proyecto se evidencia la capacidad de generación y producción por parte del estudiantado, de sistemas digitales de complejidad notable, con requerimientos de servicios internacionales para su desarrollo, permitiendo aportar a la Escuela de Ingenieías Eléctrica, Electrónica y Telecomunicaciones de la Universidad Industrial de Santander y a la región, nuevas tecnologías que cubran la necesidad latente de herramientas que ofrezcan la mayor cantidad de bienes al menor costo posible con versatilidad, de manera didáctica y suficiente documentación.

[1] Trabajo de grado.

[2] Facultad de ingenierías físico mecánicas. Escuela de ingenierías eléctrica, electrónica y de telecomunicaciones. Adolfo León Arenas Landinez

ABSTRACT

TITLE: EMBEDDED SYSTEMS DEVELOPMENT PLATFORM BASED ON THE GAME BOY ADVANCE[3]

AUTORS: BANGUERO TORRES, SERGIO RICARDO, and, ERAZO SIERRA, MAURICIO[4]

KEYWORDS: Embedded System, Game Boy Advance, GBA, ARM, PCB.

DESCRIPTION:

This work develops an expansion board with videogame architecture compatible with the Game Boy Advance (GBA). This board allows access to the multiple resources offered by the console adding the possibility of external programmable logic via FPGA, granting freedom to the developer for the use of those and opening the possibility of interconnection with external variables interfaces.

The design of the system is based in early developments and pretends by means of the practice of reproduction, to take control of the technology that is enclosed in this type of platforms. The work exposed begins by detailing the specifications of the videogame console in order to understand the low level operation of the most popular commercial version of the expansion set. After the generation of a logical and physical interconnection diagram of the relevant devices one continues to the design and fabrication of the printed circuit board (PCB) in which the system will be lodged, to finally, develop a simple application that demonstratively teaches its operation in hardware levels as well as software levels, making emphasis in the development tools, all of them of public domain.

The execution of this project demonstrates the capacity of generation and production of noticeable complex digital systems, with international services requirements for its development by part of the students; allowing to contribute to the School of Electrical, Electronics and Telecommunications Engineering, the Industrial University of Santander and the region, new technologies that cover the strong necessity of tools that offer the wider amount of goods at the lowest price with versatility, in a didactic way and with enough documentation.

[3] Degree Work.

[4] Faculty of Physical-Mechanical Engineering. Electronics Engineering. Adolfo León Arenas Landinez

Contenido general

página

Índice de tablas

Índice de figuras

Índice de abreviaturas

GBA	Game Boy Advance.
GBC	Game Boy Color
FPS	Frames Por Segundo
FPGA	Field Programmable Gate Array.
CPLD	Complex Programmable Logic Device.
SW	Software.
HW	Hardware.
GCC	GNU C/C++ Compiler.
ELF	Executable and Linkable File.
GDB	GNU Debugger
HDL	Hardware Description Language
RTL	Register Transfer Level
ISP	In-System Programming
VBA	Visual Boy Advance

Capítulo 1

Introducción

A nivel mundial es cada vez más clara la tendencia que tienen los productos electrónicos a volverse más compactos, portables y con menores consumos de energía. Reproductores de audio portátiles, PDA's, teléfonos celulares, aplicaciones industriales, médicas y otras que apuntan a ofrecer múltiples servicios dentro de un solo dispositivo, son ejemplos de *sistemas embebidos* que cumplen con estos requisitos.

A pesar del crecimiento de la demanda de sistemas embebidos a nivel regional, la oferta local en la mayoría de los casos se ve reducida a soluciones basadas en plataformas de 8-bits, que además de haber estado en el mercado por más de una década, presentan limitaciones inherentes y altos costos si se compara su relación precio/tecnología. La solución aparente apunta inicialmente a la adquisición de tecnologías modernas que proporcionen al diseñador ventajas tanto econónomicas como técnicas y que por supuesto extiendan el horizonte de recursos permitiendo ampliar considerablemente el panorama de soluciones a desarrollar. Lastimosamente, dicha tecnología debe ser importada, y por tal, el diseño de aplicaciones que estén basadas en la misma acarrea costos que en general corresponden a la adquisición de las herramientas necesarias como las plataformas de desarrollo, cables, software, y tiempo que se debe invertir para manejar la tecnología. Estos costos pueden llegar a ser considerados equivocadamente altos debido a nuestro poder adquisitivo y representan entre otras, una razón obvia de rechazo a la nueva tecnología.

El Game Boy es una popular consola de video juegos portátil fabricada por Nintendo, con un registro garantizado de durabilidad y confiabilidad, comprobado por los muchos modelos han aparecido a través de los años desde 1989. Gracias a su bajo costo y alta disposición de recursos, ha sido usada en varias ocasiones para aplicaciones diferentes a video juegos y entretenimiento, dentro de las que se destacan:

- Unidad para Electrocardiograma (US patent #5876351) [1]
- Osciloscopio Digital [2]
- Unidad de GPS [3]
- Reproductor MP3 [4]

La plataforma de desarrollo que plantea este trabajo pretende servir como anfitrión universal a diferentes aplicaciones haciendo uso de los múltiples recursos y de las facilidades de programación que ofrece la consola, junto con la posibilidad externa de lógica programable, otorgando libertad al diseñador para el uso de los recursos hardware/software disponibles y

abriendo un abanico de posibilidades de interconexión con diferentes dispositivos, sensores, etc, por medio de varios GPIOs[1] .

Con el presente trabajo de grado se suministrará una **"PLATAFORMA DE DESARROLLO PARA SISTEMAS EMBEBIDOS BASADA EN EL GAME BOY ADVANCE"** a la Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones de la Universidad Industrial de Santander, para cubrir la necesidad latente de una herramienta que ofrezca la mayor cantidad de servicios al menor costo posible, que tenga herramientas accesibles, que sea versátil, didáctica y que esté suficientemente documentada. De igual forma, este proyecto constituye un aporte significativo al desarrollo y fortalecimiento del Grupo de Investigación en Conectividad y Procesado de Señal (CPS) en el área de *Embedded Systems*.

1.1 Marco Teórico

1.1.1 ¿Qué es un sistema embebido?

Los sistemas computacionales están en todas partes. Probablemente no sea sorprendente que millones de sistemas computacionales sean fabricados cada año destinados para computadores de escritorio (computadores personales, o PC), estaciones de trabajo y servidores. Lo que puede ser sorprendente es que miles de millones de sistemas computacionales son construidos cada año con propósitos muy diferentes [5] : ellos estan "embebidos" dentro de dispositivos electrónicos más grandes, ejecutando repetitivamente una función particular, frecuentemente desconocida por el usuario del dispositivo. Crear una definición precisa de estos sistemas embebidos computacionales, o simplemente *sistemas embebidos*, no es una tarea fácil porque no existe ninguna definición estándar del mismo, por tanto nos podemos encontrar con las siguientes definiciones:

- Un sistema embebido es un computador dentro de un producto.
- Un sistema embebido es un sistema operativo ejecutándose en un microcontrolador de pocos recursos.
- Un sistema embebido es un artefacto (hardware + software) no suceptible de modificación del algoritmo que define su comportamiento.
- Un sistema embebido es un procesador, con sus elementos externos que desarrolla una función específica de manera autónoma.
- Un sistema embebido es una mezcla de hardware y software que constituye un componente dentro de un sistema más complejo y se espera que funcione sin intervención humana.

Así, consideraremos una definición que englobará a la mayoría de las definiciones anteriores:

> *Un sistema embebido es un sistema que usa un computador para realizar una función específica, pero ni es usado ni es percibido como un computador.*

Podemos entender estos sistemas examinando ejemplos comunes y características comunes. Los sistemas embebidos se encuentran en una variedad de dispositivos electrónicos comunes, como:

[1] Entradas-salidas de propósito general.

(a) **electrónica de consumo** – celulares, cámaras digitales, cámaras de video, video juegos portátiles, calculadoras y PDA's; (b) **aparatos domésticos** – hornos microondas, contestadoras, termostatos, seguridad doméstica, lavadoras, y sistemas de iluminación; (c) **aparatos de oficina** – máquinas de fax, copiadoras, impresoras y scanners; (d) **equipo de negocios** – cajas registradoras, sistemas de alarmas, lectores de tarjetas, scanners de productos, y cajeros automáticos; (e) **automóviles** – control de transmisión, control de velocidad, inyección de combustible, frenos antibloqueo, y suspensión activa. Se puede entonces decir que casi cualquier dispositivo que funciona con energía eléctrica tiene o tendrá muy pronto un sistema computacional embebido en él [5].
La figura 1–1 ilustra una disposición genérica, que es aplicable virtualmente a todo sistema embebido, resaltando sus componentes típicos.

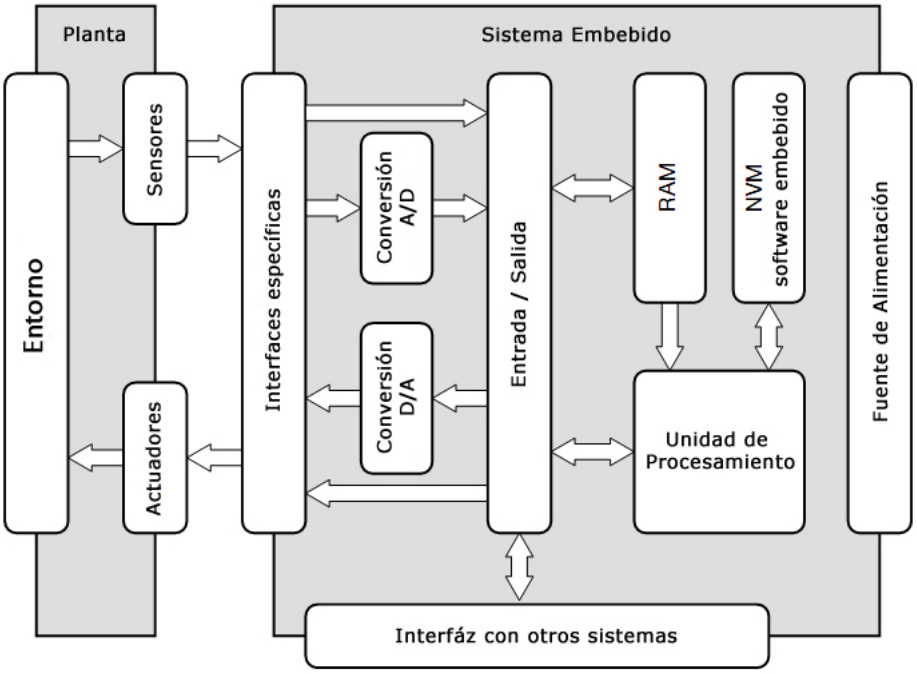


Figure 1–1: Esquema genérico de un sistema embebido
Modificado de [6].

Un sistema embebido interactúa con el mundo real, recibiendo señales a través de *sensores* y enviando señales de salida a *actuadores* que de alguna manera manipulan el *entorno*. El entorno de un sistema embebido, incluyendo los actuadores y los sensores, es normalmente referido como la *planta*. El *software embebido* del sistema es almacenado en cualquier tipo de NVM[2] . Normalmente es una ROM[3] , pero el software también puede estar almacenado en tarjetas flash, discos duros, CD-ROM, y descargado vía red o satélite. El software embebido es compilado para un procesador en particular, la *unidad de procesamiento*, que usualmente requiere una cierta cantidad de RAM[4] para operar. La unidad de procesamiento sólo puede procesar

[2] Memoria no volátil (Non-volatile memory)

[3] Memoria de solo lectura (Read Only Memory)

[4] Memoria de acceso aleatorio (Random Access Memory)

señales digitales mientras que el entorno posiblemente interactúa con señales análogas, al tiempo que conversiones *análogas-digitales* (bi-direccionales) tienen lugar. La unidad de procesamiento maneja todas las señales de entrada y salida (I/O) a través de una capa de I/O dedicada. El sistema embebido interactúa con la planta y posiblemente otros sistemas (embebidos) a través de *interfaces específicas* [6].

Los microprocesadores utilizados por estos sistemas embebidos pueden ser desde simples máquinas de 4-bits como los que se encuentran en tarjetas de felicitación o en un juguete de niño, hasta poderosos microprocesadores de 128-bits, DSP especializados y procesadores de redes. Algunos de los productos que incluyen estos chips pueden estar ejecutando pequeños códigos en *assembler* desde una ROM sin sistema operativo alguno mientras muchos otros ejecutan sistemas operativos en tiempo real (RTOS), complejas rutinas multihilo en C o C++ y variantes de sistemas operativos de escritorio basados en Linux y Windows.

1.1.2 Clasificación de los sistemas embebidos

Dentro de la definición expuesta anteriormente, una gran cantidad de dispositivos pueden ser considerados como sistemas embebidos. Una de las principales clasificaciones de sistemas embebidos sobresale si consideramos su interacción con el resto del entorno. Atendiendo a esta característica, podemos encontrar los siguientes sistemas:

- Sistemas reactivos: son aquellos sistemas que siempre interactúan con el exterior, de tal forma que la velocidad de operación del sistema es la velocidad del entorno exterior.
- Sistemas interactivos: son aquellos sistemas que siempre interactúan con el exterior, de tal forma que la velocidad de operación del sistema es la velocidad del propio sistema embebido.
- Sistemas transformacionales: son aquellos sistemas que no interactúan con el exterior, únicamente toman un bloque de datos de entrada y lo transforman en un bloque de datos de salida, que no es necesario en el entorno.

Dentro de los sistemas reactivos se puede incluir el sistema de control aéreo de un aeropuerto, ya que la velocidad del sistema dependerá de la velocidad con la que lleguen los datos de los diferentes aviones que se aterricen o despeguen del mismo. En cuanto a los sistemas interactivos, se puede incluir a cualquier tipo de máquina de videojuegos, ya que la velocidad del sistema depende del él mismo, y el exterior (es decir el usuario del videojuego) se debe adecuar a su velocidad. Por último, dentro de los sistemas transformacionales se encuentran los afiches de publicidad electrónicos, en los que no existe ningún tipo de interactividad excepto la entrada de datos iniciales y la salida de datos finales.

Aunque estos tres tipos de sistemas cumplen con la definición de sistema embebido, se suelen tomar como tal a los sistemas reactivos, ya que su auge surgió cuando se adaptaron estos diseños a este tipo de problemas. De hecho, los sistemas reactivos son comúnmente conocidos como sistemas de tiempo real.

1.1.3 El Game Boy Advance

El Game Boy Advance (GBA) (ver fig. 1–2) es la mejora realizada por Nintendo en el 2001 a su famosa consola de juego portátil Game Boy. Sus especificaciones técnicas se detallan en la tabla 1–1.

Los programas que ejecuta el GBA están generalmente almacenados en un “Game Pack” o un cartucho de juego. Un Game Pack consiste básicamente de una ROM y posiblemente *Cart*

Figure 1–2: Game Boy Advance

Table 1–1: Especificaciones Técnicas del GBA

CPU	ARM7TDMI 32-bits operando a 16.7MHz + CPU CISC 8-bits.
Memoria	32kB en chip (bus de 32-bits) + 256kB fuera del chip (bus de 16-bits) + 96kB (Memoria de video).
Pantalla	LCD Color, 240x160 pixels, 7.36cm (diagonal).
Sonido	4 canales análogos, bocina y conector audífonos.
Puertos de comunicación	Game Pack (cartucho de juegos), GameLink (serial).
Alimentación	3.3V.

RAM (en forma de SRAM, FLASH ROM, o EEPROM, usada principalmente para guardar información del juego). La memoria ROM es donde el código compilado y los datos son almacenados. A diferencia de los PC's de escritorio, estaciones de trabajo o servidores, no hay discos ni otras unidades, así que todo aquello que de otra manera se almacenaría como archivos separados debe ser compilado en la ROM del programa. Estas características combinadas con un bajo precio[5] y alta disponibilidad, hacen al Game Boy una plataforma ideal para gran variedad de aplicaciones diferentes a las que fue originalmente concebido para ejecutar, además, su diseño robusto permite buenos desempeños en ambientes hostiles donde otras computadoras portables podrían no sobrevivir por tanto tiempo.
A continuación se expondrán a un nivel más técnico las carácteristicas del GBA que se consideran fundamentales para el desarrollo de la tarjeta de expansión.

CPU

El Core ARM7TDMI es miembro de la familia ARM de microprocesadores de 32-bits de propósito general. La familia ARM ofrece alto desempeño a un bajo consumo de potencia y pequeño tamaño.

La Arquitectura ARM esta basada en los principios de la arquitectura RISC (Reduced Instruction Set Computer). El set de instrucciones RISC, y su mecanismo de decodificación relacionado son mucho más simples que aquellos de los diseños CISC (Complex Instruction Set Computer). Esta simplicidad proporciona [7]:

- Ejecución de mayor cantidad de instrucciones en menor tiempo
- Una excelente respuesta en tiempo real a las interrupciones
- Un procesador de macrocelda pequeña y económica

[5] Su precio en el mercado oscila actualmente entre menos de 10USD y 29 USD para consolas usadas en buen estado

ARM y THUMB. Los diseñadores del ARM decidieron que podían crear un set de instrucciones más denso (el set de instrucciones THUMB) eliminando algunas características y haciendo la codificación de la instrucción un poco menos regular, de manera tal que los dos sets de instrucciones pudieran trabajar en conjunto:

- **Estado ARM :** Usa por completo los 32-bits del set de instrucciones (*opcodes* de 32-bits).
  Ventajas: En este set de instrucciones cada *opcode* individual provee más funcionalidad, dando como resultado una ejecución más rápida cuando se usan buses de memoria de 32-bits, además permite accesar a todos los registros `r0-r15` (ver tabla 1–2) directamente.
  Desventajas: El código del programa ocupa mas espacio de memoria y no es tan rápido cuando se usa un sistema de memoria de 16-bits.

- **Estado THUMB:** Usa un set de instrucciones reducido a 16-bits (*opcodes* de 16-bits).
  Ventajas: Sin importar el tamaño del opcode, ambos estados están usando registros de 32-bits, permitiendo direccionamiento de memoria de 32-bits asi como operaciones aritmético-lógicas de 32-bits. Su ejecución es más rápida (alrededor de 160% cuando se usa un sistema de memoria con bus de 16-bits) y reduce el tamaño del código, recortando la carga de memoria en un un 65%.
  Desventajas: Sus *opcodes* no son tan multifuncionales como aquellos del estado ARM, siendo necesario en algunas ocasiones usar más de un opcode para encontrar un resultado similar a lo realizado por un solo *opcode* en el estado ARM, además, la mayoría de los *opcodes* permiten solamente usar directamente los registros `r0-r7` ver tabla (1–2).

Set de Registros. Existen 16 registros visibles para el usuario en cualquier momento, y 20 "*banked registers*" los cuales son intercambiados cuando la CPU cambia a algunos de los varios modos privilegiados. Los registros visibles en el modo usuario se muestran en la tabla 1–2:

Table 1–2: Registros visibles en modo usuario

Registro(s)	**Descripción**
`r0 - r12`	Registros de propósito general, para uso en operaciones comunes.
`r13 (SR)`	Registro **Stack Pointer**. Usado primordialmente para mantener la dirección del stack. Su valor inicial depende del modo del procesador en el que se encuentre.
`r14 (LR)`	**Link Register**. Usado primordialmente para guardar la dirección siguiente a una instrucción "`bl`" (branch and link).
`r15 (CP)`	El **Program Counter** o contador de programa. Debido a que el ARM7TDMI usa un pipeline de 3 estados, este registro siempre contiene una dirección que es 2 instrucciones adelante de la que se está siendo ejecutada. En el estado ARM (32-bits), está 8 bytes adelante, mientras que en el estado THUMB (16-bits) está 4 bytes adelante.
`CPSR`	El registro de estado actual del programa (Current Program Status Register). Contiene los bits de estado relevantes a la CPU. Cuando ocurre una excepción, el contenido del CPSR se almacena en el registro SPSR (Saved Program Status Register) del respectivo modo de excepción.

Arquitectura de la Memoria

El GBA como cualquier otro sistema computacional tiene un conjunto de memorias, cada una de ellas con su función o labor específica. A continuación se describen en detalle:

- **Internal Window RAM**
  Es la única memoria a la que se puede acceder directamente en el bus interno de 32-bits del *core* de la CPU porque de hecho esta construída dentro de la misma. Es la memoria mas rápida en el GBA y es también la única memoria a la que se puede acceder 32-bits a la vez. La velocidad y el bus de esta memoria la hacen ideal para ejecutar código ARM a toda velocidad. Desafortunadamente solamente hay 256kb (32kB).

- **External Window RAM**
  A pesar de que su nombre indica su naturaleza externa al *core*, esta memoria no está ubicada en el cartucho de juego (*Game Pack*) sino que se encuentra dentro del sistema GBA y por tanto está en el bus de datos de 16-bits. Su tamaño es mayor que la IWRAM (256kB), pero es mucho más lenta que ésta, pues cada acceso a memoria toma tres ciclos.

- **Memoria Gráfica**
  Tres secciones de memoria se encargan exclusivamente del video y de la pantalla:

  - **Video RAM**
    Aquí es donde todos los datos de los gráficos deben ser almacenados para poder ser mostrados en la pantalla. Esta memoria de video (VRAM) es una memoria de cero estados de espera o *zero wait-state* como la IWRAM, pero está sobre el bus 16-bits, de manera que se pueden mover datos a la mitad de la velocidad que en la IWRAM. La manera en que esta memoria es usada depende en gran parte del modo de video y otras características que la aplicación selecciona.

  - **Palette Memory**
    La mayoría de los modos de video del GBA usan paletas para especificar los colores que serán usados. El GBA tiene dos paletas independientes de 256 colores: una para imágenes de fondo y otra para *sprites*[6] . Cada una de estas paletas es posteriormente dividida en 16 paletas de 16 colores en algunos modos, permitiendo comprimir aún más los datos gráficos. El color 0 de cualquier paleta es definido como transparente sin importar el valor que est almacenado en la memoria de la paleta. Las paletas son usualmente actualizadas durante el período de borrado vertical (Vblank), momento en el cual la pantalla no está siendo dibujada.

  - **Object Atribute Memory**
    La memoria de atributo de objetos (OAM) es donde se almacenan los atributos, o descripciones, de qué, cómo y donde se debe mostrar el *sprite*. Los datos gráficos vienen de la VRAM, pero la posición del *sprite*, el tamaño y otra información viene de la OAM.

[6] Un *sprite* es un mapa de bits de forma arbitraria que puede ser movido alrededor de fondos complejos sin causar parpadeos o daño a la imagen de fondo.

El Sistema de Gráficos

El GBA tiene una pantalla TFT LCD a color de 240x160 pixeles y una tasa de refresco de exactamente 280,896 ciclos de CPU por frame, o alrededor de 59,73 Hz. La mayoría de programas del GBA necesitarán ser estructurados alrededor de esta tasa de refresco. Cada refresco consiste en un período de dibujo vertical (VDraw) de 160 *scanlines* seguido por un período de vacio (VBlank) de 68 *scanlines*. Cada una de estas *scanlines* consiste en un período de dibujo (HDraw) de 1004 ciclos seguido por un período de vacío (VBlank) de 228 ciclos. Durante los perodos HDraw y VDraw el hardware de gráficos procesa los datos del fondo y del objeto (*sprite*) y dibuja en la pantalla, mientras que los períodos HBlank y VBlank están abiertos para que el código del programa pueda modificar datos de fondos y objetos sin el riesgo de crear *artifacts*[7] gráficos [8].

Modos de Video. Lo que el GBA dibuja en la pantalla depende en gran parte del modo de video en uso (tambien llamado *modo de pantalla* o *modo gráfico*). El GBA tiene 6 de estos modos, algunos de los cuales son basados en mapa de bits y otros en *tiles* [8] . Los datos del fondo son manejados de manera diferente dependiendo del modo que esté habilitado. Los fondos pueden ser fondos de texto (basados en *tiles*), fondos de rotar-escalar (fondos basados en tiles que pueden ser transformados), o fondos de mapa de bits. La dirección y tamaño inicial de la memoria gráfica de *sprites* es también dependiente del modo de video (ver tablas 1–3 y 1–4).

Table 1–3: Modos de video basados en *tiles*

Modo 0	En este modo pueden ser mostradas 4 capas de fondos de texto. Todos los fondos 0-3 cuentan como fondos de texto y no pueden ser escalados o rotados.
Modo 1	Este modo es similar en muchos aspectos al modo 0, siendo su principal diferencia que solo son accesibles (0,1, y 2). Los fondos 0 y 1 son fondos de texto, mientras el fondo 2 es un fondo rotable / escalable.
Modo 2	Al igual que los modos 0 y 1, el modo 2 usa fondos con tiles. Usa los fondos 2 y 3, ambos rotables / escalables.

Table 1–4: Modos de video basados en mapa de bits

Modo 3	Este es un modo estándar de mapa de bits de 16-bit (sin paleta) 240x160. El mapa empieza en `0x06000000` y es de 76,800 (`0x12C00`) bytes de largo (ver tabla 1–5 para el formato de estos bytes).
Modo 4	Modo de mapa de bits de 8-bits (con paleta) a 240x160. La paleta se encuentra en `0x5000000`, y contiene 256 entradas de color de 16-bit.
Modo 5	Este es otro modo de mapa de bits de 16-bit, pero con una resolución de 160x128. Se muestra desde la esquina superior izquierda de la pantalla, pero puede ser movido usando registros de rotación y escalamiento para el modo 2. La ventaja de uso de este modo es presumiblemente el hecho de que existen dos frame buffers disponibles, y esto puede ser usado para realizar efectos de "paso de página" que no pueden ser usados en el modo 3.

[7] Un *artifact* o artefacto corresponde a un elemento no deseable encontrado en la imagen de un juego de video. Los artifacts pueden incluir pixeles coloreados incorrectamente, imágenes fantasma, o desenfoque, entre otros.

[8] Un "*tile*" es una pequeña sección de 8x8 en la pantalla.

Formato del Color. Todos los colores (con paleta y de mapa de bits) son representados como un valor de 16 bit, usando 5 bits para rojo, verde, y azul, y ignorando el bit 15. En el caso de la memoria de paleta, los pixeles en una imagen son representados como como indices de 8 ó 4 bits en la memoria de paleta (Palette Memory) empezando en `0x5000000` para fondos y `0x5000200` para *sprites*. Cada paleta es una tabla que consiste en 256 entradas de colores de 16-bit. En el caso de los fondos de mapa de bits en los modos 3 y 4, los pixeles son representados como los valores de color de 16-bits [8].

Table 1–5: Formato de Colores

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
X	B	B	B	B	B	G	G	G	G	G	R	R	R	R	R

0-4 (R)=Rojo
5-9 (G)=Verde
A-F (B)=Azul

Interfaces de Hardware

El GBA cuenta básicamente con dos interfaces para cargar juegos, el puerto Game Pack (paralelo) y el puerto GameLink (serial). El puerto Game Pack es el encargado de conectar al cartucho de juegos con el procesador y de esta manera ejecutar las aplicaciones que se encuentren almacenadas en la ROM del cartucho. El GameLink es un puerto serie que permite varios modos de comunicación pero en general está diseñado para conectar la consola con 2 o más unidades.

Puerto Game Pack. Esta interface multiplexa el bus de 16-bits de datos y el bus 24-bits de dirección (ver figura 1–3). Es esta multiplexación la que impide conectar a la consola dispositivos de memoria sin algún tipo de interfaz lógica. En el modo GBA el bus esta descrito como se muestra en la tabla 1–6.

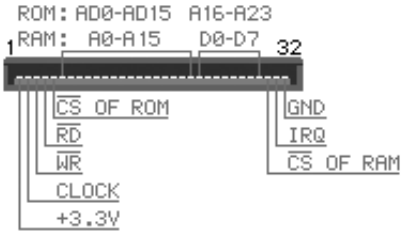


Figure 1–3: Puerto GamePack

El puerto Game Pack puede ser usado tanto para cartuchos del Game Boy Color (GBC) como para del GBA, gracias a que Nintendo ha buscado siempre la compatibilidad de sus consolas a medida que nuevos modelos salen al mercado. El GBA diferencia cartuchos de juego del GBC (8-bits) presionando o liberando un switch cuando un cartucho del GBA es insertado o cuando no hay cartucho. Él mecánicamente controla el nivel de voltaje, es decir selecciona entre 3.3V usados en el modo GBA o 5V usados en juegos antiguos de 8-bits.

Los cartuchos de juego del GBA son sistemas especiales que contienen una ROM estándar, SRAM opcional y lógica de direccionamiento y de demultiplexación del bus de dirección-datos. Para correr una aplicación independiente con el GBA es fundamental empezar por decodificar

Table 1–6: Bus del Game Pack - Conector de 32-bits

Pin	Nombre	Dir	Descripción
1	VDD	O	Alimentación 3.3V DC
2	PHI	O	Reloj del sistema (ninguno, 4.19MHz, 8.38MHz, 16.76MHz)
3	/WR	O	Selector de escritura
4	/RD	O	Selector de lectura
5	/CS	O	Chip Select de ROM
6-21	AD0-15	I/O	16-bits de dirección menos significativos y/o 16-bits datos-ROM[a]
22 - 29	A16- 23	I/O	8-bits de dirección más significativos ó 8-bits datos-SRAM[b]
30	/CS2	O	Chip Select de SRAM
31	/REQ	I	Solicitud de Interrupción (/IREQ) o de DMA (/DREQ)
32	GND	O	Tierra 0V

[a] Cuando se accede a la SRAM del Game Pack, los 16-bits de dirección salen del puerto por AD0-AD15, después 8-bits de datos son transferidos por A16-A23.

[b] Cuando se accede a la ROM del Game Pack, los 24-bits de dirección salen del puerto por AD0-AD23, después 16-bits de datos son transferidos por AD0-AD15.

el bus, y organizar estas señales de manera tal que se emule un cartucho original[9] . Aunque el método de multiplexación es información propietaria perteneciente a Nintendo, existen algunos aficionados al desarrollo en el GBA que han logrado descrifrarlo mediante ingenería inversa realizada a los cartuchos originales y así se han fabricado cartuchos "caseros" para correr aplicaciones varias (ver [9]). Estos trabajos fundamentaron las bases hardware para permitir lo que es ahora el desarrollo del GBA[10] y constituyen parte fundamental de la documentación realizada para este trabajo de grado.

Puerto GameLink. El puerto serial del GBA puede usarse en diferentes modos de comunicación, como lo son el modo normal, el modo multi-player, el modo UART, el modo JOY BUS, y el modo de propósito general. En el modo normal se puede intercambiar datos entre dos GBA's (o transferir data desde un GBA maestro a diferentes GBA esclavos de manera unidireccional), el modo multi-player permite intercambiar datos entre hasta 4 GBA's, el modo UART funciona muy parecido a una interfaz RS-232, el modo JOY Bus usa un protocolo estandarizado de Nintendo y finalmente el modo de propósito general permite usar el puerto "serial" como un puerto paralelo bidireccional de 4-bits.

[9] Los juegos originales tienen cierta información básica, como el logotipo de Nintendo, que es necesaria reproducir idénticamente para que la CPU empiece a ejecutar el código del programa.

[10] http://www.devrs.com/gba/, http://linux.gbadev.org/, http://www.gbadev.org/

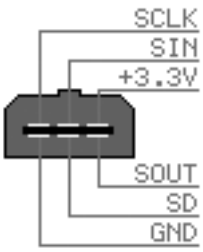


Figure 1–4: Puerto GameLink

Gracias a una función denominada MultiBoot, que está alojada en la BIOS del GBA, es posible descargar aplicaciones en el GBA cuando este está en modo multi-player esclavo, permitiendo de esta manera ejecutar códigos (sin necesidad de insertar cartucho) que se almacenan en la EWRAM. Este modo esta diseñado por Nintendo para permitir la opción de multijugador en algunos de sus juegos, pero puede ser usada también por desarrolladores para descargar aplicaciones pequeñas (hasta 256KB) sin necesidad de otro hardware que un cable especial[11] y un PC con un software[12] que contiene el protocolo de transferencia MultiBoot.

1.2 Organización del documento

El documento pretende seguir una secuencia lógica para el lector, motivo por el cual está organizado de manera conceptualmente cronológica, iniciando con una **introducción** que lo sitúa en el tópico de los sistemas emebebidos, describiendo de manera general el qué y el por qué del proyecto para finalmente enfocarse seriamente en la consola de videojuegos GBA en torno a la cual se desarrolla el mismo.

Capítulo 2
Descripción del sistema
Describe los temas relacionados con el diseño de la tarjeta de expansión, y abarca aspectos como los antecedentes y la descripción general del diseño. En este capítulo se trata al diseño como sistema y por tanto a sus componentes como subsistemas. Las descripciones son a lo sumo de carácter lógico y pretenden dar claridad en cuanto al desempeño de la tarjeta de expansión (MEXGBA) en sus diferentes actividades. Finalmente se exponen consideraciones relacionadas con el diseño físico del prototipo.

Capítulo 3
Configuración del Módulo de Expansión MEXGBA
Aborda aspectos relativos a la programación de la plataforma de desarrollo y las herramientas necesarias para este fin. Plantea los conceptos básicos sobre los requerimientos Hardware y Software y a su vez describe de manera sintetizada su funcionamiento. Finalmente se describe de manera concisa la programación del GBA y de la tarjeta de expansión.

Capítulo 4

[11] Para más información del cable multiboot (Xboo) consulte la referencia [10].

[12] http://nocash.emubase.de/gba-xboo.zip.

Aplicación
Exhibe las características de la aplicación realizada con la plataforma de desarrollo, siempre resaltando su naturaleza sencilla y demostrativa, detallando los aspectos más relevantes, como configuración de periféricos en la FPGA y la comunicación de estos con la CPU.

Capítulo 5
Conclusiones y observaciones
Se plantean las conclusiones y observaciones del trabajo realizado.

Capítulo 2

Descripción del prototipo

El **M**ódulo de **EX**pansión del **GBA** (**MEXGBA**) es un sistema basado en cartuchos originales y desarrollos anteriores. Constituye una herramienta para acceder a los recursos del GBA via Game Pack Bus. Está compuesto por una memoria flash, una FPGA y un CPLD que permiten descargar las aplicaciones y programar los periféricos externos necesarios.

2.1 Antecedentes

El desarrollo de aplicaciones de terceros en el Game Boy (GB) ha sido un tema que si bien no es de conocimiento público, ha tenido un avance considerable desde sus inicios (ver [2]) y poco a poco ha venido surgiendo sólida y hasta comercialmente. Existen gran cantidad de herramientas (ver [11], [12]) que conciernen, y que constituyen referencias fundamentales para los desarrollos que al respecto se realizan actualmente.

Las alternativas enfocadas al desarrollo usando consolas tipo GB (GB, GBC, GBA, GBASP) existen en gran variedad; desde modificación de cartuchos originales para sobreescribir en ellos, pasando por programación por el puerto GameLink de cartuchos caseros conectados al puerto Game Pack, hasta módulos totalmente programables de manera independiente del GBA; estos últimos son los que nos atañen y constituyen el soporte del diseño del MEXGBA.

2.1.1 XPORT

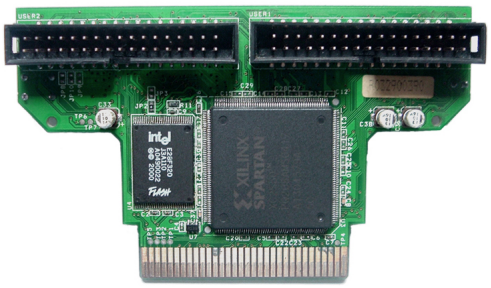


Figure 2–1: Xport de Charmedlabs

La tarjeta de expansión Xport (desarrollada por *Charmedlabs*[1]) convierte al GBA en una plataforma de desarrollo de sistemas embebidos. Su diseño consta de una FPGA de hasta 150.000 compuertas lógicas, 4Mbytes de memoria flash, 16Mbytes de RAM opcionales, 64 conectores de entrada-salida programables, programabilidad en sistema (ISP) y un puerto de comunicaciones y depuración de alta velocidad (300kbps) incorporado. Esta herramienta constituye la base teórica de este trabajo de grado.

2.2 Descripción del sistema

Una plataforma de desarrollo consiste de un conjunto de dispositivos que pueden ser configurados de diferentes maneras permitiendo la ejecución de gran variedad de aplicaciones. Normalmente están constituidas por un microprocesador y/o un microcontrolador, dispositivos reconfigurables como FPGA's y CPLD's, memorias y controladores de periféricos comunes como VGA, Ethernet, USB, etc.

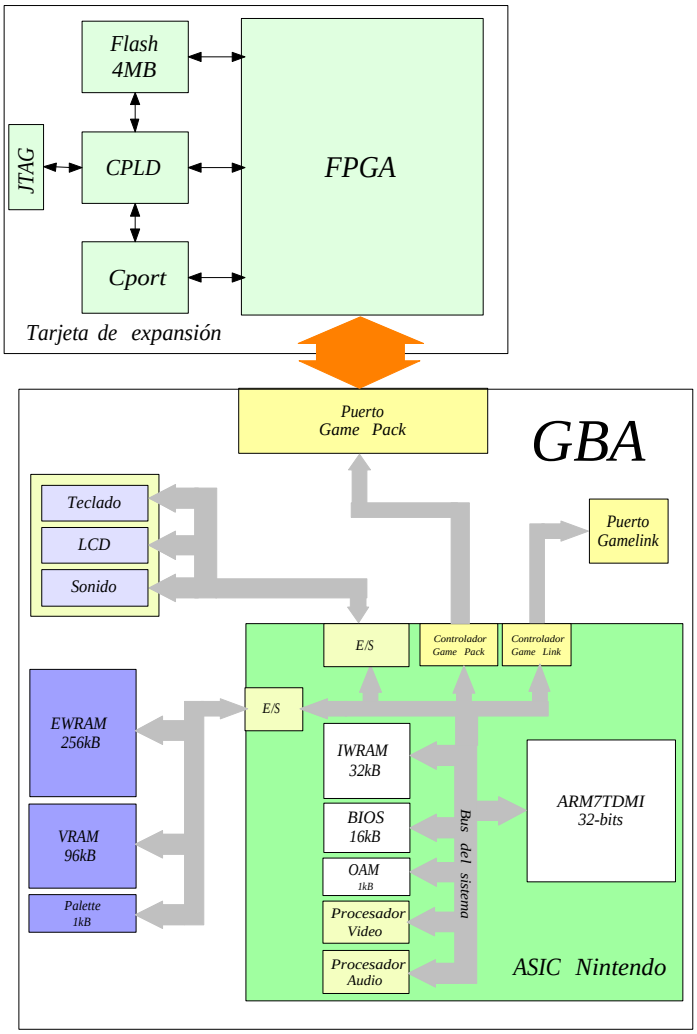


Figure 2–2: Diagrama de Bloques

[1] http://www.charmedlabs.com.

Es importante notar que la arquitectura del MEXGBA esta basada en la XPORT, eliminando únicamente la posibilidad de SRAM externa. Debido a que el objetivo fundamental del diseño del primer prototipo del MEXGBA es proveer una herramienta que permita ejecutar aplicaciones con el GBA, no existe inicialmente motivo para intentar mejorar de alguna manera un diseño robusto, y por el contrario, existe un enorme interés en estudiar a profundidad el mismo.

El hecho de basar la arquitectura en la XPORT conlleva a numerosas ventajas como el soporte técnico en foros, documentación específica, herramientas software para programación y configuraciones funcionales.

La plataforma de desarrollo está compuesta por el GBA y la tarjeta de expansión; quien a su vez consta de 5 componentes principales (ver fig 2–2), 2 dispositivos electrónicos de lógica programable (FPGA y CPLD), un dispositivo de memoria (FLASH) y 2 puertos de configuración (JTAG y CPORT).

La operación más realizada con la tarjeta de expansión es la lectura y escritura de la memoria flash, otras operaciones como programación de la FPGA o del CPLD son en algunos casos transparentes para desarrollador y serán abordadas más adelante.

2.2.1 La memoria flash

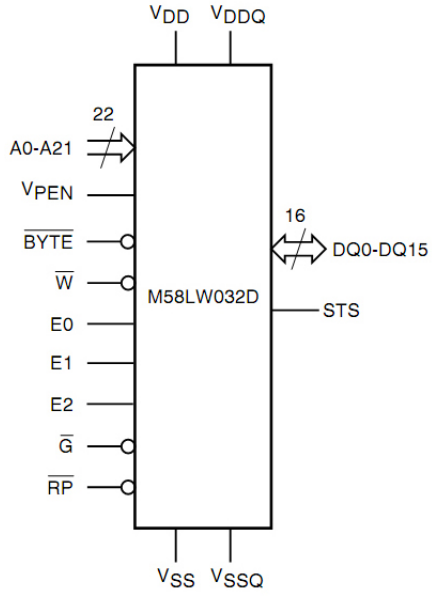


Figure 2–3: Memoria Flash

La memoria flash es un híbrido RAM-ROM que puede ser borrada y sobreescrita bajo control Software. La flash es una memoria no volátil programable en circuito (*in-circuit*)[2] segmentada en bloques denominados sectores. Cada sector puede ser borrado individualmente, y los datos en el pueden ser reescritos [13]. El MEXGBA usa una memoria de 32Mbit con la posibilidad de bus de datos de 8 o 16 bit (M58LW032D de ST, ver fig 2–3).

[2] La programación *in-circuit* se refiere a la disponibilidad del dispositivo para ser programado una vez ubicado en un sistema.

Table 2–1: Descripción de señales de la memoria flash

A0	Esta señal de entrada de dirección es usada para seleccionar el Byte superior o inferior en el modo X8. No es usado en el modo X16 (donde A1 es es el Bit menos significativo)
A1 -A21	Las entradas de direcciones son usadas para seleccionar las celdas para acceder en el arreglo de memoria durante las operaciones de lectura de bus, ya sea para leer o para programar datos. Durante las operaciones de escritura de bus ellas controlan los comandos enviados a la interfaz de comandos del controlador de Programación/Borrado. El dispositivo debe estar habilitado para seleccionar las direcciones.
$\overline{BYTE}$	El pin selector de organización Byte/Word es usado para seleccionar entre los buses de memoria de ancho x8 y x16. Cuando el pin tiene un nivel bajo, VIL, la memoria está en modo x8, cuando es nivel alto, VIH, la memoria está en modo x16
DQ0–DQ15	Las entradas/salidas de Datos realizan la interfaz con los datos almacenados en la dirección seleccionada durante la operación de lectura del bus. Durante las operaciones de escritura del bus representan los comandos enviados a la interfase de comandos del controlador de Programación/Borrado.
E0, E1, E2	Las entradas habilitadoras (Chip Enable), activan el control lógico de memoria, buffers de entrada, decodificadores y amplificadores de sensado.
$\overline{G}$	El Habilitador de salida transmite las salidas a través de los buffers de salida de datos durante una operación de lectura.
$\overline{RP}$	El pin de Reset/Power-Down puede ser usado para aplicar un Reset Hardware a la memoria.
STS	La señal STS es una salida de drenador abierto que puede ser usada para identificar el estatus del controlador de Programación/Borrado. Puede ser configurada en dos modos: Ready/Busy y Status.
V_{PEN}	La entrada habilitadora de Programación/Borrado es usada para proteger todos los bloques, previniendo que operaciones de programación o borrado afecten los datos.
$\overline{W}$	La entrada habilitadora de escritura, W̃, controla la escritura a la interfáz de comandos, direcciones de entrada y latches de datos.
V_{DD}	VDD provee la fuente de alimentación para el core interno del dispositivo de memoria. Es la fuente de alimentación principal para todas las operaciones (lectura, programación y borrado)
V_{DDQ}	VDDQ provee la fuente de alimentación para los pines I/O y habilita todas las salidas para ser alimentadas independientemente de VDD. VDDQ puede estar amarrado a VDD o puede usar una alimentación diferente.
V_{SS}	Tierra, es la referencia para la fuente de alimentación del core.
V_{SSQ}	Tierra de la circuitería de I/O alimentada por VDDQ.

La memoria usada (M58LW032D) es una memoria que funciona usando una fuente de alimentación sencilla a un bajo voltaje (2.7V a 3.6V). La memoria está dividida en 32 bloques o slots de 1Mbit (128KB) que pueden ser borrados independientemente, por tanto es posible preservar datos válidos mientras se eliminan datos viejos. Los comandos de programación y borrado están escritos en la interfaz de comandos de la memoria. Un controlador de programación/borrado dentro del IC simplifica el proceso de programación y borrado de la memoria realizando todas las operaciones especiales que son requeridas para actualizar los contenidos de memoria [14].

La memoria flash almacena los datos y código correspondientes a la aplicación, el procesador ARM del GBA ejecuta este código y produce resultados ya sea en alguno de sus periféricos incorporados como LCD, audio, o, simplemente los datos resultados de la ejecución del código son tomados del bus de datos por la FPGA para alimentar algún periférico externo según sea el caso.

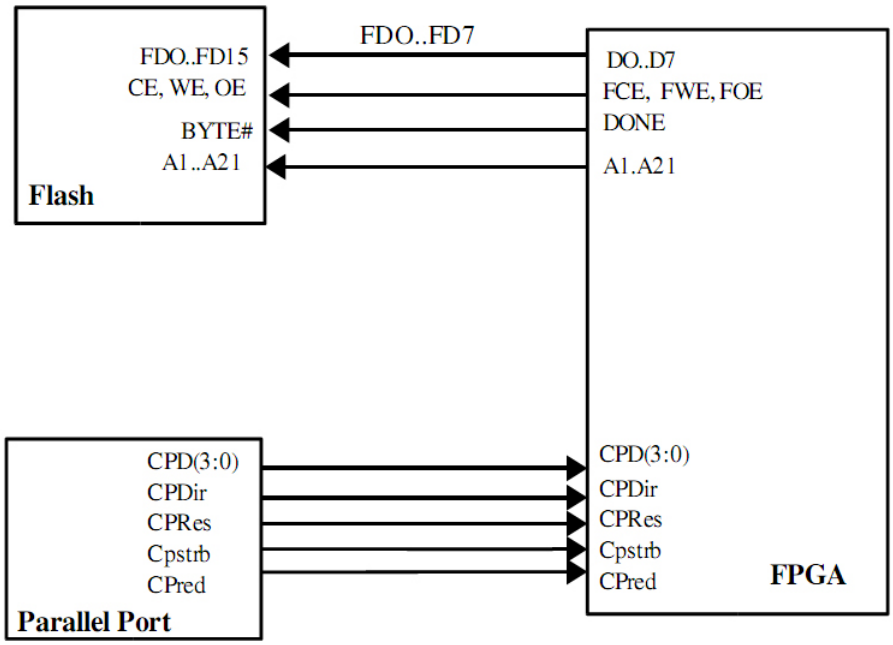


Figure 2–4: Escritura de la memoria flash

La **programación de la memoria flash** se realiza con la asistencia de la FPGA y el puerto CPORT (ver figura 2–4).

El puerto CPORT está conectado al puerto paralelo en el PC (ver sección B.3.1) y se maneja por medio del Software xpcomm[3] , la programación de la memoria flash entonces consiste en enviar los datos con las respectivas banderas correspondientes a los modos para escribir en la flash, la FPGA se encarga de organizar estos datos y enviarlos a la flash para su correcto almacenamiento.

Este proceso se realiza de manera automática y no requiere ayuda externa del desarrollador, quien debe solamente por medio de comandos del xpcomm enviar el archivo binario correspondiente al código fuente compilado.

La **lectura de la memoria flash** (ver fig 2–6) se realiza por medio de la FPGA quien debe decodificar el bus y separar las señales de dirección y de datos. Esta multiplexación se aplica a los 16 bits menos significativos del bus de dirección junto con el bus de datos de 16 bits (ver tabla 1–6). El acceso a dispositivos de memoria externos varía dependiendo del tipo de memoria externa, siendo el procedimiento de multiplexación anteriormente descrito aplicable para memorias tipo ROM como es el caso del MEXGBA. Este acceso puede ser secuencial o aleatorio siendo obligatorio que el primer acceso a memoria sea aleatorio.

Después del primer acceso aleatorio, es posible leer la flash por completo secuencialmente, este acceso secuencial comienza con un flanco bajo de la señal /CS de ROM quien indica el tipo de memoria a usar. La dirección de memoria a leer se envía por el bus

[3] xpcomm es la interfaz por medio de la cual se envían los datos por el puerto paralelo al Cport, ver capítulo 3, *Configuración del módulo de expansión*, sección 4.3.2 en la página 38.

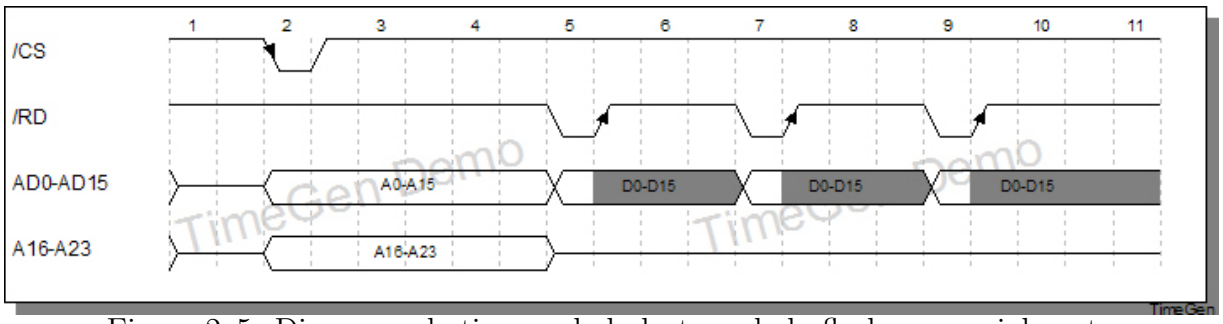


Figure 2–5: Diagrama de tiempo de la lectura de la flash secuencialmente

"AD0-AD15"+"A16-A23" que se convierten en el bus de dirección, los datos entonces se envían en el flanco positivo de la señal /RD por ADO-AD15 (ver fig 2–5).

El diagrama de la fig. 2–6 muestra de manera detallada el proceso de lectura de la flash. Es importante notar que solamente 8 de las 16 líneas de datos de la flash están conectadas a la FPGA y es por estas líneas es que se efectúa tanto la programación de la flash como la configuración en paralelo de la FPGA como se verá mas adelante. Las restantes 8 líneas van directamente al bus del Game Pack.

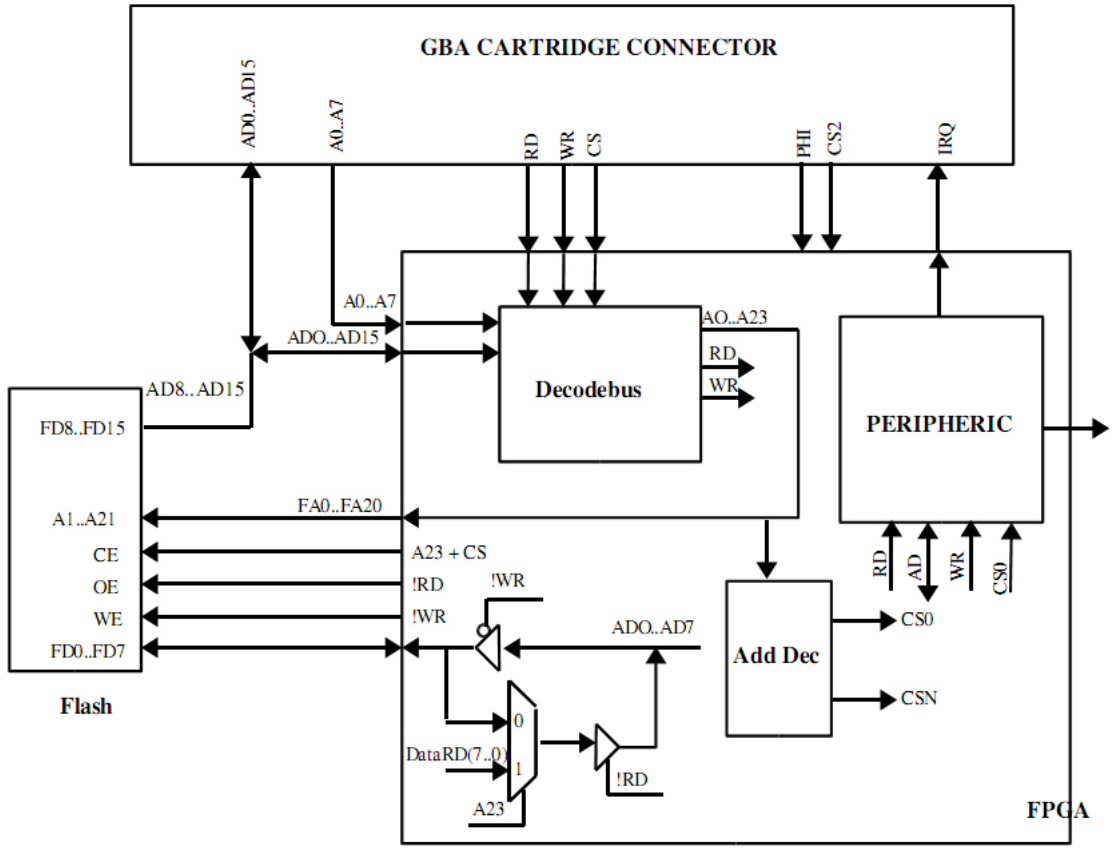


Figure 2–6: Lectura de la memoria flash

2.2.2 El CPLD

Un CPLD (Complex Programmable Logic Device) consiste típicamente de varios bloques lógicos programables además de una matriz de interconexiones programables (ver fig 2–7)

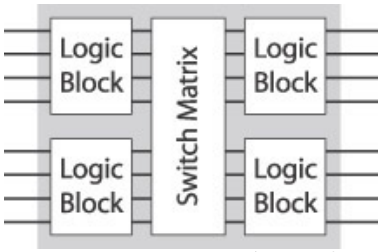


Figure 2–7: Diagrama genérico de un CPLD

El CPLD usado (XC9536XV de Xilinx) funciona a 2.5V y está diseñado para aplicaciones de sistemas computacionales de alto desempeño y bajo voltaje. Provee 800 compuertas usables con retardos de programación de 5ns.

El papel del CPLD dentro del sistema es servir de asistente para la programación, serial o paralela de la FPGA. Su programación se realiza por medio de la interfaz JTAG[4] .

El archivo de configuración del CPLD fue extraído de la tarjeta de expansión Xport a través del puerto JTAG, y por tal razón, los componentes más relevantes del módulo desarrollado tienen las mismas referencias.

El incurrir en nuevos elementos, como por ejemplo una FPGA más moderna implica necesariamente una reprogramación de este dispositivo, lo cual se traduce en una reinvención total del diseño con un alto nivel de complejidad en programación y sistemas digitales. Debido a que el objetivo fundamental que engloba este trabajo de grado se encuentra directamente relacionado con el aprendizaje obtenido de la práctica de la reproducción, el rediseño y optimización se considera una posibilidad solo para una etapa posterior.

2.2.3 La FPGA

La **FPGA** es el elemento lógico fundamental en la ejecución de las aplicaciones, el MEXGBA está equipado con una Spartan-II XC2S50 (50k compuertas) quien se encarga de la decodificación del bus, la programación de la FLASH y de la administración y control de periféricos externos.

Los dispositivos Spartan-II son configurados[5] descargando secuencialmente frames de datos que han sido concatenados en un archivo de configuración de 550,200 bits (para mayor información respecto a la configuración de la FPGA refierase al Anexo C.8 en la página 78).

La FPGA en el MEXGBA se configura en cada inicio del sistema en el modo de programación esclavo paralelo bajo la asistencia del CPLD en modo maestro (ver fig 2–8). En este proceso la FPGA descarga los datos almacenados en el *SLOT* 0 de la FLASH[6] quien contiene la configuración de operación por defecto, la cual se carga cuando el GBA

[4] JTAG es un estándar que provee acceso serial externo para pruebas en circuitos integrados, por medio de una interfaz de 4 ó 5 pines.

[5] La FPGA Soporta cuatro modos de configuración: esclavo serial, maestro serial, esclavo paralelo y boundary-scan.

[6] Un *slot* corresponde a un espacio de memoria de 128KBytes.

se enciende [15]. Esta programación es sin embargo transparente en la mayoría de los casos para el programador, ya que, se carga automáticamente en cada secuencia de energización de la tarjeta de expansión.

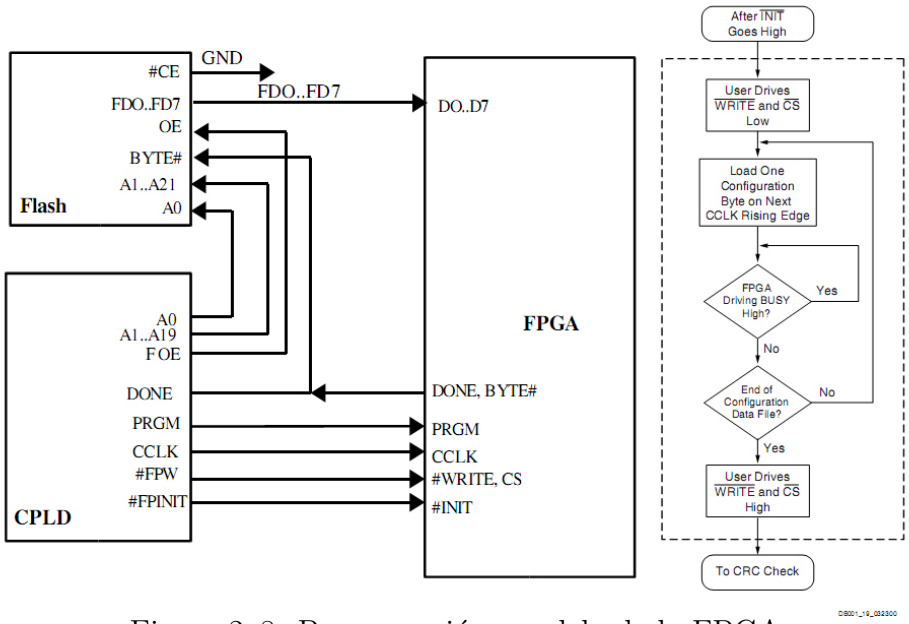


Figure 2–8: Programación paralela de la FPGA

La FPGA también tiene la posibilidad de programación en modo serial esclavo (ver fig 2–9) en la que directamente desde el xpcomm se envían las tramas correspondientes al archivo de configuración y con la asistencia del CPLD son transmitidas a la FPGA. Este modo de configuración es usado como modo de depuración mayormente pues al desenergizar el sistema la configuración se pierde.

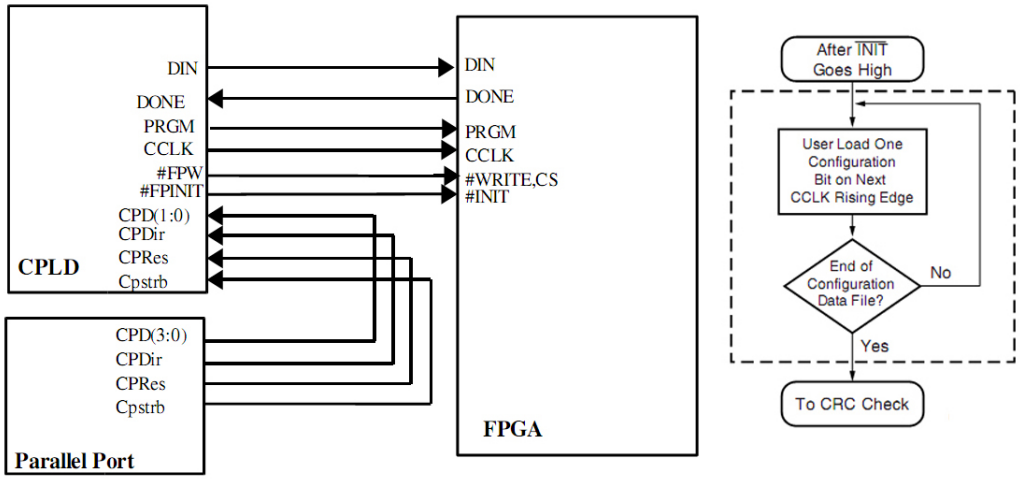


Figure 2–9: Programación serial de la FPGA

La tarea de demultiplexación requiere solo una pequeña fracción de la lógica de la FPGA. El resto de la lógica está disponible para aplicaciones particulares del usuario. Estas aplicaciones pueden usar la FPGA para acceder a dispositivos externos, coprocesamiento o ambos. Algunos ejemplos de coprocesamiento incluyen:

- Multiplicación, división.
- Filtrado FIR.
- Procesamiento de Imágenes.
- Procesador RISC de propósito general.

Es importante resaltar en este punto que este proyecto converge en desarrollos anteriores que hacen uso de dispositivos de lógica programable, permitiendo empalmar y continuar los mismos. Un ejemplo contundente, se encuentra en la aplicabilidad de diseños de controladores, funciones matemáticas y hasta microprocesadores [16] que han sido exitosamente creados y sintetizados por parte de estudiantes de la E3T[7] , permitiendo incluirlos en novedosos diseños que apunten hacia nuevos horizontes en el desarrollo acádemico y tecnológico del Alma Máter.

2.3 Esquemático

El diseño de interconexión entre los dispositivos que conforman la tarjeta de expansión está basado, como se mencionó anteriormente, en la tarjeta Xport. En las figuras 2–11 y 2–10 se muestra un esquema detallado.

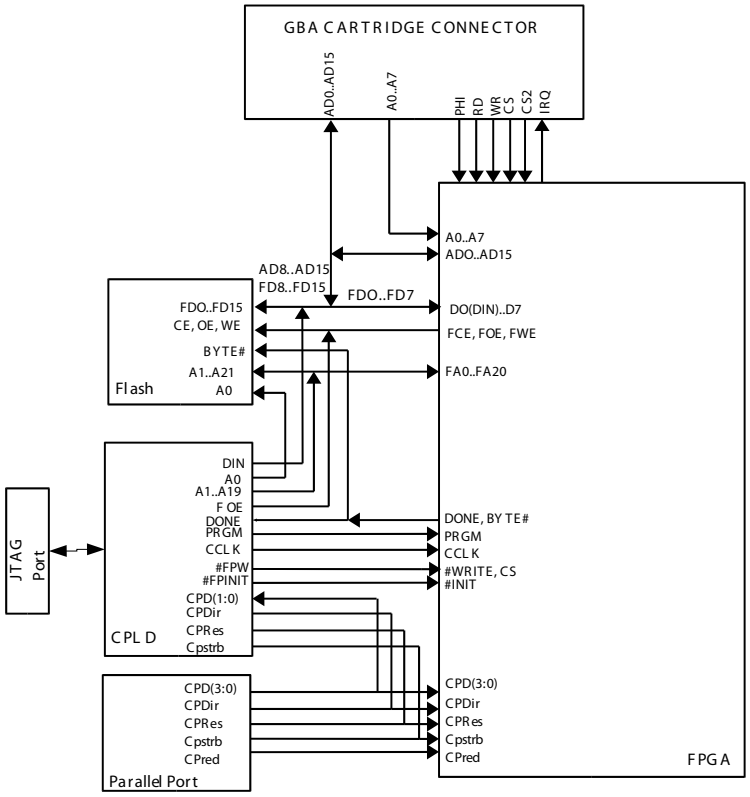


Figure 2–10: Diagrama Esquemático

[7] Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones.

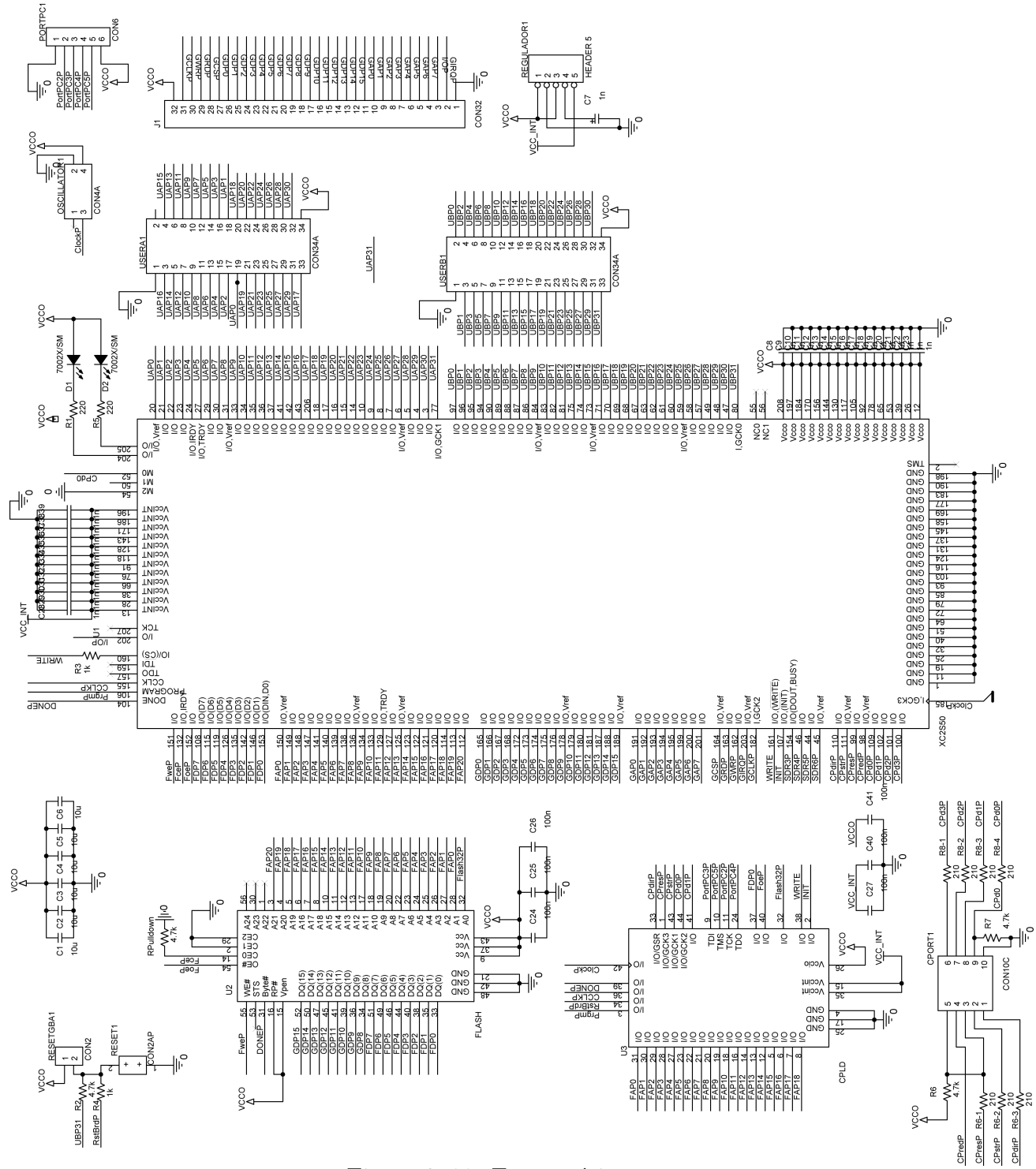


Figure 2–11: Esquemático

Capítulo 3

PCB

Del diseño y fabricación del circuito impreso (PCB) depende un gran porcentaje del éxito del proyecto electrónico que se esté desarrollando. En este aspecto se relacionan actividades como la creación de footprints, ubicación de los componentes, especificaciones técnicas de medidas y otras, que dejan entrever el nivel de complejidad que puede llegar a tener un diseño.
Este capítulo compila los aspectos y conocimientos más relevantes adquiridos durante el desarrollo del trabajo de grado en materia de PCB's.

3.1 Consideraciones

Parámetros que intervienen directamente en el diseño de un circuito impreso y que son de gran importancia para asegurar el correcto funcionamiento del prototipo fueron analizados en el proceso de diseño.

3.1.1 Dimensiones (*Board Outline*)

Este diseño involucra limitaciones físicas concernientes a las dimensiones y estructura de un cartucho de juegos que intervienen en parámetros de diseño tales como el ancho mínimo de pista, ancho mínimo de VIA[1] y grosor del material FR4.

Para empezar hay que cumplir con las dimensiones del conector *Game Pack* (ver fig. 3–1) para asegurar una conexión robusta. Estas dimensiones se refieren principalmente al recuadro que enmarca una sección de la imagen, es decir, al ancho (49.97mm), alto (13.98mm), y grosor (2mm). Dichas restricciones de tamaño involucran de inmediato una reducción del ancho de pista y de del tamaño de VIA, que de otra manera estarían ligadas a las dimensiones de los PADS de los dispositivos y al área que el diseñador considere necesaria usar. Por otra parte, el grosor del material dieléctrico FR4, influye directamente en la impedancia característica[2] del PCB.

[1] Una VIA es un agujero conductor que conecta dos capas.

[2] La impedancia característica es un parámetro de línea de transmisión que determina cómo las señales de transmisión son transmitidas o reflejadas en la línea.

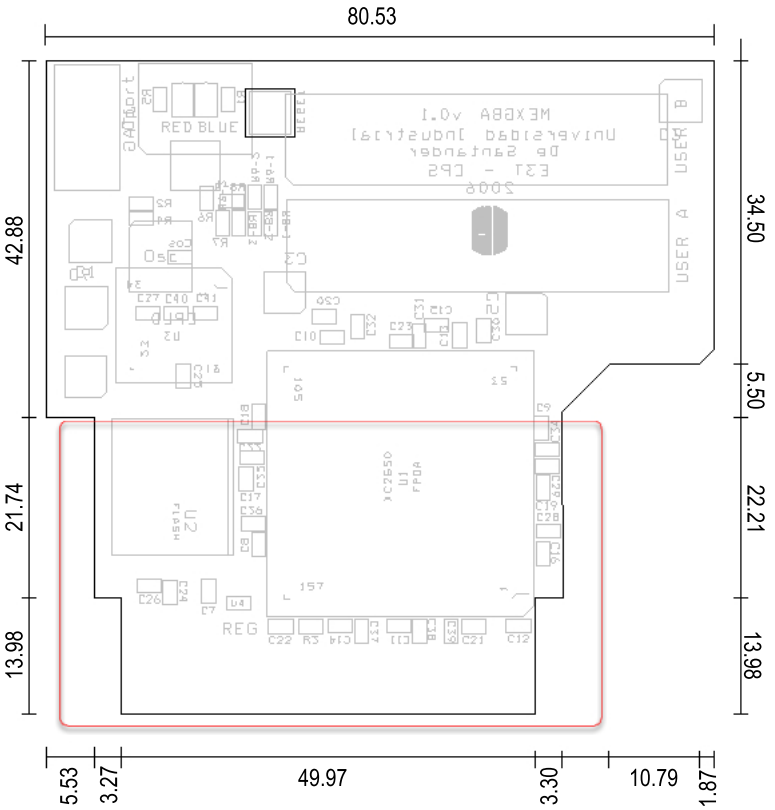


Figure 3–1: Dimensiones del PCB en mm

Es necesario aclarar que esta impedancia característica es un parámetro que depende del ancho de la pista (W), del grosor del cobre (T) sobre el dieléctrico, de la permitividad relativa del material diélectrico FR4 (ϵ_r), y del grosor de este material (H, para el caso de un PCB de dos capas), como se puede ver en la fig. 3–2.

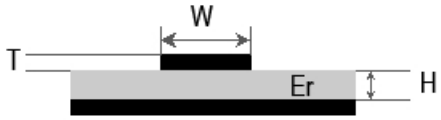


Figure 3–2: Parámetros de la impedancia característica

$$Z_o = \frac{87}{\sqrt{\epsilon_r + 1.41}} \ln\left(\frac{5.98H}{0.8W + T}\right) \tag{3.1}$$

En la ecuación[3] 3.1 se aprecia la manera como estos parámetros están relacionados. Con base en (3.1) se obtiene que la impedancia caraceterística del PCB realizado tiene un valor aproximado de 160 Ω (tomando W=0.16mm y ϵ_r=4.3), y no presenta un factor de riesgo

[3] De la norma IPC-D-317A (Eq. 5.32).

para el funcionamiento del sistema dada su frecuencia (50MHz).

3.1.2 Ubicación de los componentes

De la ubicación de los dispositivos dependen las longitudes máximas de las pistas y por tanto los retardos de señales, y efectos de antena por frecuencia. El *layout* diseñado para el MEXGBA intenta tener una alta densidad, evitando estos efectos y tratando de minimizar el área de la tarjeta como se puede ver en la figura 3–3.

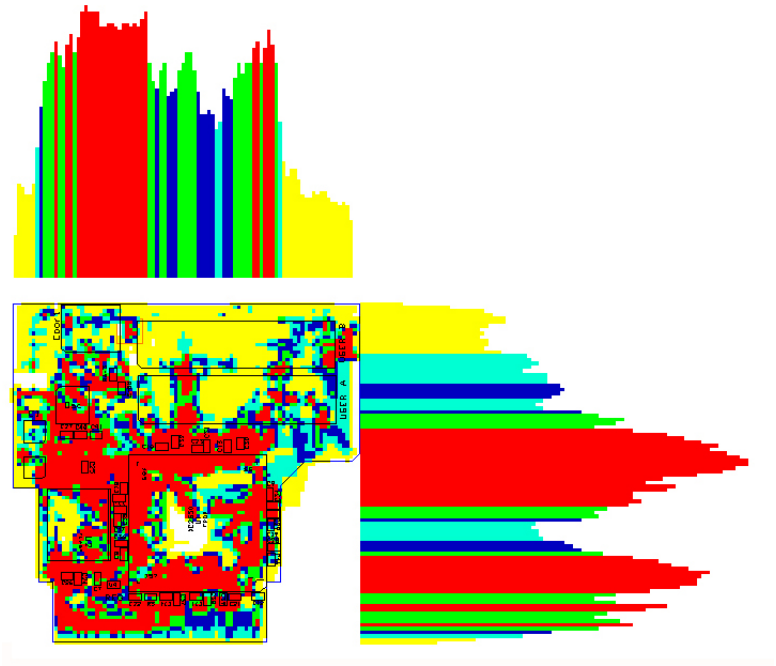


Figure 3–3: Gráfico de densidad del PCB

Por desarrollos anteriores de este tipo de tarjetas de expansión, se hace relevante disminuir al mínimo la distancia entre la FPGA y los pines de contacto con el puerto *Game Pack*, por tal motivo, se ubica la FPGA como elemento inicial y se parte desde esta para determinar la posición de los elementos. Es necesario resaltar también que la configuración espacial de la tarjeta Xport fue analizada en este proceso.

Efectos de *crosstalk* fueron tambien tenidos en cuenta. *Crosstalk* se refiere a la transferencia de energía desde una pista (fuente) a otra adyacente (víctima) (ver fig. 3–4). La intensidad del acoplamiento de señal entre las pistas se reduce disminuyendo la longitud del segmento que se encuentra adyacente, con separaciones mayores entre segmentos y mayor impedancia de línea, entre otros [17].

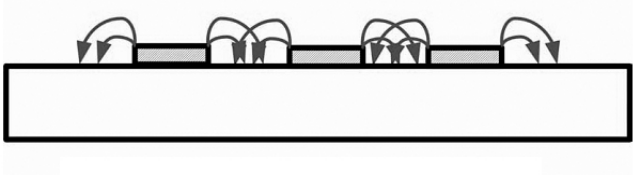


Figure 3–4: Crosstalk

Algunas reglas que se proponen para evitar este efecto consisten en dejar separaciones de hasta 3W (donde W es el ancho de la pista), entre pista y pista, afectando directamente

el área del diseño, parámetro que prima en muchos casos. Entonces, es "aceptable" admitir cierto *crosstalk* dependiendo de la aplicación y los márgenes de diseño.

Para el caso de este diseño se usó UltraCAD[4] , una herramienta que da un valor aproximado del coeficiente de acoplamiento entre pistas, dada una longitud de los segmentos adyacentes, separación entre los mismo y el *rise-time* del oscilador, entre otros parámetros. Este coeficiente obtenido (0.22 ± 0.07)[5] indica el nivel de tensión que se acopla entre pistas y fue calculado para el peor caso, es decir, el más alto rise-time (8ns) [18], la menor separación entre pistas (W=0.16mm) y un una longitud de segmento adyacente de 2cm, permitiendo asignar a este nivel de crosstalk el calificativo de aceptable.

3.1.3 Corriente

Dentro de las consideraciones eléctricas que afectan directamente parámetros de diseño del PCB se encuentra el consumo de corriente, específicamente, los niveles máximos que circularán por el circuito impreso. El consumo de corriente[6] del sistema varía de acuerdo al estado del mismo como es de suponerse, siendo más alto el consumo de arranque (312mA aprox.) que el consumo de operación normal (93mA aprox). El arranque presenta esta característica debido a la corriente POS[7] de la FPGA, que debe ser proporcionada en un periodo desde 2ms hasta 50ms, en el cual la FPGA se puede modelar como un elemento de baja impedancia[8] .

Estos datos de corriente conciernen a las especificaciones del sistema de alimentación y algunas características físicas de la tarjeta como el ancho de la pista (la norma aconseja un ancho de 0.381mm por un Amperio) y el grosor del cobre (norma IPC-2221 [20]).

3.1.4 Frecuencia

La frecuencia del sistema es otro aspecto que interviene en las consideraciones, debido a su naturaleza medianamente alta (50MHz), específicamente en las longitudes de pista máximas de la tarjeta y en la geometría y disposición de los planos y señales.

Para una frecuencia de 50MHz, se tiene una longitud de onda cercana a los 6m. Una longitud de onda de esta dimensión no presenta una amenaza[9] importante para el desempeño del sistema en el PCB. Sin embargo, se realizó un análisis de frecuencia en el PCB centrado en evitar ciertas configuraciones que pueden actúan como antenas de

---

[4] http://www.ultracad.com

[5] Los valores de este coeficiente son de carácter especulativo pues no se tiene en cuenta que ϵ_r del FR4 puede variar hasta en un 20%, además existen otros parámetros que el modelo del simulador UltraCAD no contempla, sin embargo, es un valor que permite hacer un análisis cualitativo.

[6] Basado en cálculos teóricos.

[7] *Power-On Surge*

[8] Para más información acerca de los requerimientos y características de encendido (*Power-On*) de los dispositivos Spartan-II de Xilinx refiérase a [19].

[9] Considerando el criterio de $\frac{\lambda}{n}$, donde el valor de n es normalmente 4.

lazo o *mono-polo*, por tanto se trabajó con especial atención evitando configuraciónes de *lazo* (cómunes en la distribución de las señales de alimentación) y *mono-polo* (presente en pistas de largas longitudes no terminadas o con terminación en un dispositivo de alta impedancia).

3.1.5 Ruteo

Otros aspectos que son tenidos en consideración generalmente en diseños digitales de alta frecuencia son la geometría de las pistas. A altas frecuencias, las corrientes que fluyen por las pistas no pueden encontrar discontinuidades en el camino, por tal motivo es importante para asegurar la integridad de la señal evitar conexiones y cruces de menos de 90 grados.

3.1.6 Software de ruteo

La herramienta CAD[10] a seleccionar para realizar el diseño del PCB debe ser en lo posible una herramienta amigable, efectiva y bien documentada. Dentro de las herramientas que se consideraron para esta etapa se encuentran Eagle Layout Editor, KiCAD y OrCAD Layout (*versión estudiantil*). Las dos primeras bajo licencias GPL pero con notables desventajas de customización con respecto a OrCAD Layout (*versión estudiantil*), quien finalmente fue escogida para realizar el ruteo.

Es meritorio mencionar en este punto que debido a las consideraciones físicas y eléctricas ya contempladas y a otros aspectos relevantes, el ruteo del sistema se realizó manualmente, pues los algoritmos ofrecidos por las herramientas CAD no contemplaban en su mayoría parámetros importantes y otras herramientas como que finalmente podrían considerarlos, como *Mentor PADS PCB*[11] , implicaban una curva de aprendizaje que finalmente duraría más tiempo que el ruteo manual.

3.2 Especificaciones

Con las consideraciones expuestas anteriormente se realizó el diseño del PCB que alojaría el sistema. El conducto regular en este caso consiste en conocer el fabricante y sus especificaciones y diseñar con base en la oferta de fabricantes, sin embargo, por ignorancia, este procedimiento fue originalmente planeado a la inversa, motivo que llevó a una busqueda exhaustiva de fabricantes que permitieran la ejecución el diseño ya realizado con mínimas modificaciones.

En la tabla 3–1 se pueden observar algunos de los parámetros más relevantes para la selección del fabricante.

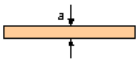
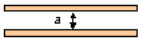
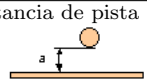
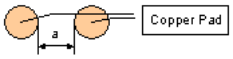
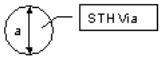
3.2.1 Fabricante

El fabricante debe ser escogido de acuerdo a las especificaciones de la tarjeta, procurando siempre obtener la mejor calidad al mejor precio y en el menor tiempo posible. El precio normalmente depende de los limites de los parámetros de diseño, algunos de los cuales son : numero de capas, grosor, peso del cobre, ancho de pista, separación entre

[10] *Computer Aided Design.*

[11] O algún otro producto de la suite Mentor.

Table 3–1: Especificaciones del PCB

Item	Mínimo
Ancho de pista 	a>0.16mm
Espacio entre pistas 	a>0.16mm
Distancia de pista a pad 	a>0.16mm
Distancia entre pads 	a>0.16mm
Anillo anular 	a>0.13mm
Diametro de Via STH 	a>0.44mm

cobre, tamaño mínimo de VIA y anillo anular. Estos parámetros son escogidos por el diseñador y pueden depender de otros parámetros o depender de relaciones entre los mismos, por ejemplo, las dimensiones limitadas de la tarjeta (aunque sea en una porción) limitan el tamaño mínimo de las pistas y por ende de los huecos.

La selección del fabricante se hizo de manera comparativa, haciendo un paralelo del parámetro tecnológia/precio, teniendo en cuenta fábricas en diferentes lugares. En Colombia, se revisaron las ofertas de Colcircuitos (Medellín), Electronicas AZ (Bucaramanga) y Microcircuitos (Bogotá), encontrando en general un alto costo y un bajo nivel tecnológico representado en especificaciones mínimas con umbrales altos (específicamente *annular ring*) si se contrastan con el diseño realizado. Fuera de Colombia se contemplaron opciones en Argentina (Ernesto Mayer S.A), Polonia (ELDOS Co. Ltd.), Australia (Entech), China (PCBcart), E.U.A y Cánada (Myropcb). En general todas las opciones presentaron inconvenientes con el tamaño de la VIA y con el Anillo anular que va relacionado, y siendo este párametro de importancia notable, se decidió por escoger al fabricante con la mejor tecnología, involucrando esto, un menor rediseño o en el mejor de los casos nulo. Afortunadamente, el fabricante seleccionado, Myropcb[12] presentó tambien la mejor relación precio/tecnología, además del mejor servicio al cliente, sin embargo cabe destacar que la opción argentina (Ernesto Mayer S.A) cuenta con muy buenos precios y especificaciones mínimas resaltables.

Si bien se escogió la mejor oferta, el fabricante incurrió en errores debido a la no actualización de datos, pues el diseño que entró a manufactura fue un diseño que no constituía la versión final enviada. Sin embargo, este error no afectó la funcionalidad del PCB. En la figura 3–5 se observa el diseño del pcb (se omiten los planos de tierra para poder observar el detalle de las conexiones).

[12] http://www.myropcb.com

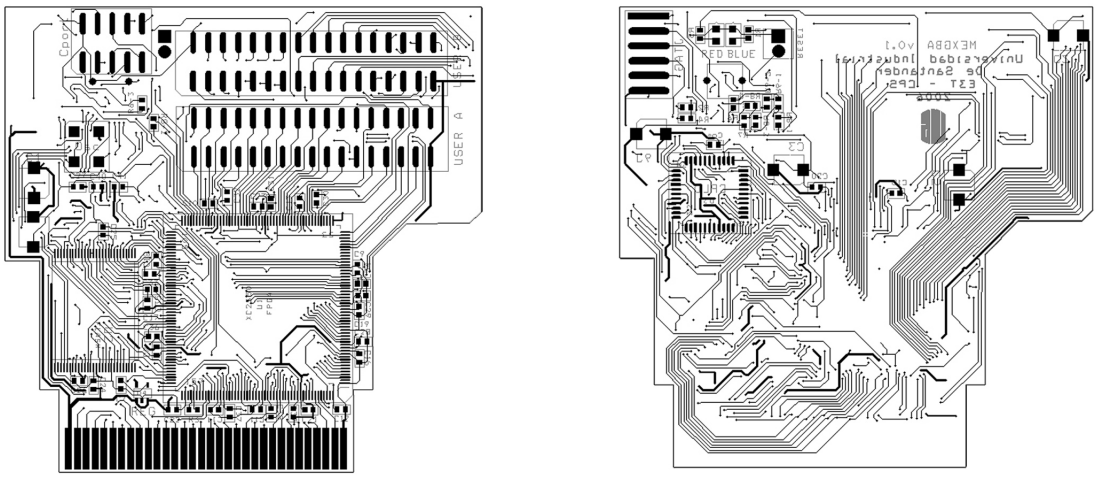


Figure 3–5: Diseño del PCB

3.2.2 Soldadura

Este proceso es de especial cuidado pues constituye la etapa definitiva del proyecto, y puede terminar con el trabajo de meses si se realiza sin las consideraciones necesarias. Es recomendable revisar las hojas técnicas de los dispositivos y del PCB para cerciorarse de la temperatura que es recomendada para esta labor.

Dentro de los diferentes métodos para realizar la actividad, se recomienda el uso de cautín o similar para circuitos integrados y el uso de flujo de aire caliente para elementos pasivos de pequeño tamaño. Esto debido a que el uso de flujo de aire caliente sobre circuitos integrados requiere considerable experiencia, además, existe mayor probabilidad de superar los límites de temperatura tanto del integrado como del PCB, contrario a la precisión que proporciona el cautín. Para soldar con flujo de aire caliente los elementos pasivos, se recomienda realizar este proceso antes de la soldadura de dispositivos integrados a fin de evitar exposiciones a temperaturas altas. Se deben soldar solamente aquellos componentes que no incomoden la posterior actividad de soldadura de circuitos integrados.

La limpieza juega un papel importante a la hora de asegurar una soldadura de buen nivel, por esto es necesario el uso de agentes limpiadores (como el isopropanol) antes y después del proceso, para eliminar impurezas y grasa de los contactos a soldar.

Flux

El Flux es la sustancia usada para ayudar a una soldadura limpia, y su función es eliminar los oxidos para asegurar una unión y conductividad adecuada. La mayoría de los flux para soldar dejan un rastro de residuos después de haber realizado su trabajo. Casi la totalidad de ellos generan humos tóxicos y se aconseja por esto trabajar en áreas ventiladas.

Existen gran cantidad de flux para soldadura y muy poca información al respecto, a continuación se presenta un resumen de los tipos de fluxes mas comunes:

- Flux de recina con colofonia

Son los mas viejos, los mas populares y aún hoy los más efectivos. Este flux es un compuesto orgánico que consiste principalmente en **colofonia**, una sustancia natural extraída de la savia de los árboles de pino y destilada para llevarla a estado liquido para fácil aplicabilidad. Los agentes activos en el flujo de recina son el ácido abiético y el ácido plicático; y es la reacción entre estos ácidos y los óxidos metálicos en los materiales de unión la que provee la limpieza requerida para la soldadura. Sin embargo son los que contienen el mayor porcentaje de residuos sólidos y los que requieren mas trabajo para una correcta limpieza.

- Flux No Clean

  Estos flux están hechos en su mayoría de recinas sintéticas y sus residuos no son corrosivos ni pegajosos. Fueron diseñados para ser dejados en la tarjeta en aplicaciones donde este tipo de residuos sea aceptable. Sin embargo se requiere su limpieza en aplicaciones de alta frecuencia.

- Flux soluble en agua

  Un flux soluble en agua es aquel cuya base está hecha de agua permitiendo ser limpiado fácilmente con un solvente basado en agua también. Tiene la desventaja que los solventes a base de agua y saponificadores pueden oxidar los contactos.

El proceso de soldadura realizado sobre la tarjeta de expansión presentó inconvenientes debidos al tipo de flux usado, ya que en realidad, este elemento no fue valorado apropiadamente. El uso de flux con alto contenido en residuos sólidos en soldadura de elementos superficiales constituye un riesgo toda vez que sus residuos se almacenan en lugares inalcanzables impidiendo la correcta limpieza de la tarjeta y almacenando polvo posiblemente conductor.

Finalmente, en la figura 3–6 se observa el resultado final del módulo de expansión y en la tabla 3–2 se condensan párametros relativos al diseño del PCB en pulgadas.

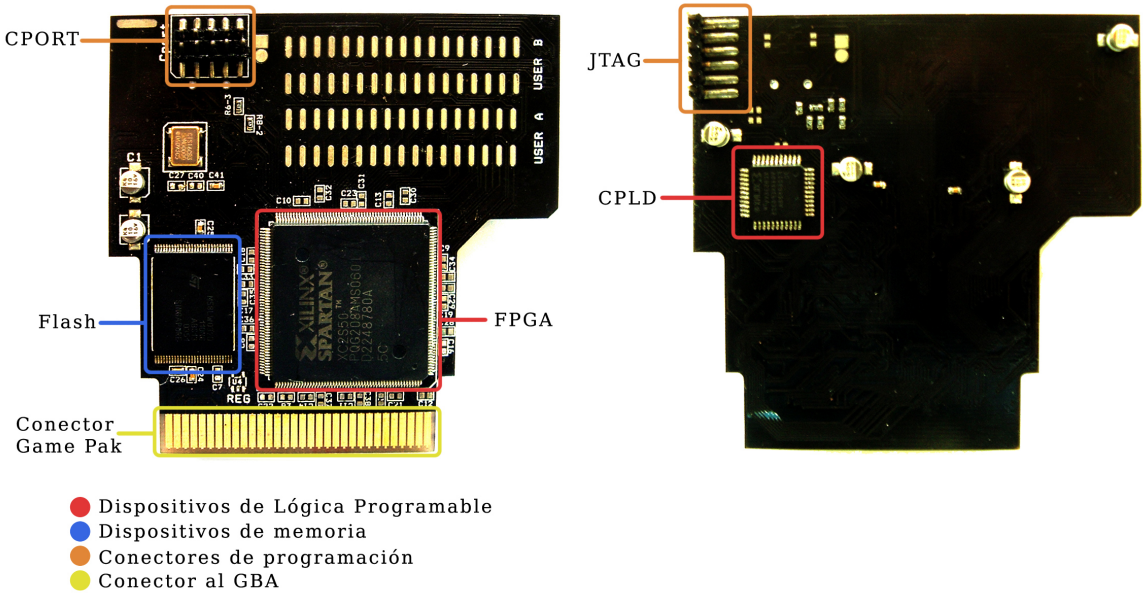


Figure 3–6: PCB fabricado y soldado

Table 3–2: Estadisticas del diseño

STATISTIC	TOTAL
Board Area	63.2
Equivalent IC's	38.8
Sq. inches per IC	1.63
# of pins	582
Layers	2
Design Rule Errors	0
Time Used	193:40:00
Vias	504
Test Points	0
Vias per Conn	1.21
Segments	4421
Connections	418
Nets	169
Components	99
Footprints	173
Padstacks	137
Obstacles	388
Theoretical Dist	706.3
Routed Dist	813.2

Capítulo 4

Configuración del Módulo de Expansión (MEXGBA)

Como en todos los sistemas embebidos, en el MEXGBA existe un aporte de Software (SW) y Hardware (HW) sobre la aplicación a desarrollar, cada uno de estos actua sobre un conjunto de dispositivos y requiere diferentes herramientas.

4.1 Requerimientos SW

Estos requerimientos se refieren al conjunto de herramientas necesarias para generar el contenido de la memoria de programa:

Compilador Los compiladores traducen programas estructurados a lenguaje de máquina o ensamblador. Los lenguajes de programación estructurados (C, C++, UML[1]) poseen instrucciones de alto nivel las cuales simplifican la programacion, así que cada instrucción de alto nivel puede resultar en varias decenas de instrucciones de máquina, sin embargo, la utilización de estos lenguajes de alto nivel reduce el tiempo de programación, facilita el re-uso y la portabilidad del código. Este proyecto fue enfocado a un desarrollo basado en herramientas GNU, las cuales son de gran confiabilidad además de ser de uso gratuito. El compilador escogido para este trabajo es el *gcc* (GNU project C/C++ compiler[2] .). El gcc genera un archivo en formato *ELF*[3] tras una serie de etapas de compilación (ver fig 4–1.).

El Compilador GNU se puede inicializar en cualquiera de estas etapas, proveyendo un archivo de entrada con la extensión requerida, como se muestra en la figura 4–1 [21].

1. **Preprocesamiento**

   La primera etapa del Compilador GNU convierte el archivo de entrada en una salida preprocesada. El procesamiento de un archivo convierte todos las declaraciones

[1] *Unified Modeling Language.*

[2] El compilador gcc es de hecho más que un compilador de C/C++, tiene soporte para otros lenguajes (Objective-C, Fortran, Java y Ada).

[3] Executable and Linkable Format.

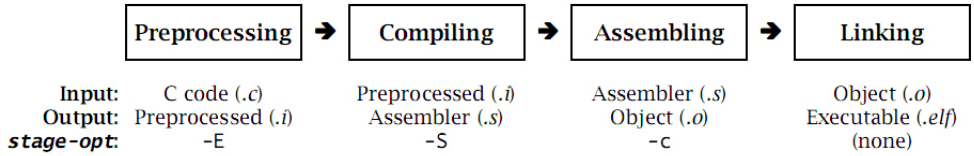


Figure 4–1: Etapas de compilación del gcc
Tomado de [21]

(como `#include`, `#define` y `#ifdef`) a verdadero código C. El archivo de entrada debe tener la extensión *.c* para implicar su naturaleza de código C.

2. **Compilación**

   La segunda etapa del compilador convierte la entrada preprocesada en un archivo de lenguaje ensamblador. Aquí es donde la mayoria del trabajo pesado de compilación en C es hecho.

3. **Ensamblado**

   La tercera etapa del compilador es convertir el código en lenguaje ensamblador en un objeto binario relocalizable de salida. El Compilador GNU llama al Ensamblador GNU (GNU Assembler), `arm-elf-as`[4] , para realizar el trabajo real de ensamblado.

4. **Enlazado**

   La etapa final del compilador consiste enlazar el archivo del objeto relocalizable en un archivo ejecutable de salida. El Compilador GNU usa al Enlazador GNU (GNU Linker),`arm-elf-ld`, para hacer el trabajo real.

El archivo de formato ELF, obtenido como resultado final de la compilación está dividido en las siguientes secciones (solo se muestran las más importantes) [22]:

.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 incluida en el archivo formato ELF no es necesaria para la ejecución de un programa, de hecho, solo son necesarias las secciones .data, .rodata y .text. Para "extraer" estas secciones el ejecutable se utilizan una serie de herramientas [5] que hacen parte de la cadena de herramientas GNU (GNU toolchain).

[4] Es necesario recordar que el procesador objetivo es de arquitectura ARM.

[5] http://sources.redhat.com/binutils/.

Depurador

El *gdb*[6] permite depurar el código ejecutando instrucción por instrucción y mostrando el valor que toman los registros y las variables internas. El GDB se puede usar en dos modos diferentes, con una interfaz gráfica llamada Insight y con la tradicional linea de comandos [23].

4.2 Requerimientos HW

Además de los componentes HW expuestos en el capítulo anterior se requieren herramientas que permitan generar el código que será sintetizado en los dispositivos de lógica programable. Estas *herramientas de síntesis* generalmente son provistas por el fabricante del dispositivo. En este caso en particular se usaron dispositivos Xilinx y por consiguiente las herramientas de síntesis contenidas en su paquete ISE (*Integrated Synthesis Environment*).

La sintetización lógica es el proceso de convertir una descripción de alto nivel del diseño en una representación optimizada del nivel de compuertas (ver fig 4–2), dada una librería de celdas estándar y ciertas condiciones de diseño. Una librería de celdas estándar puede tener celdas simples, como compuertas lógicas básicas *and*, *or*, y *nor*, o macro celdas, como sumadores, multiplexores, y flip-flops especiales [24].

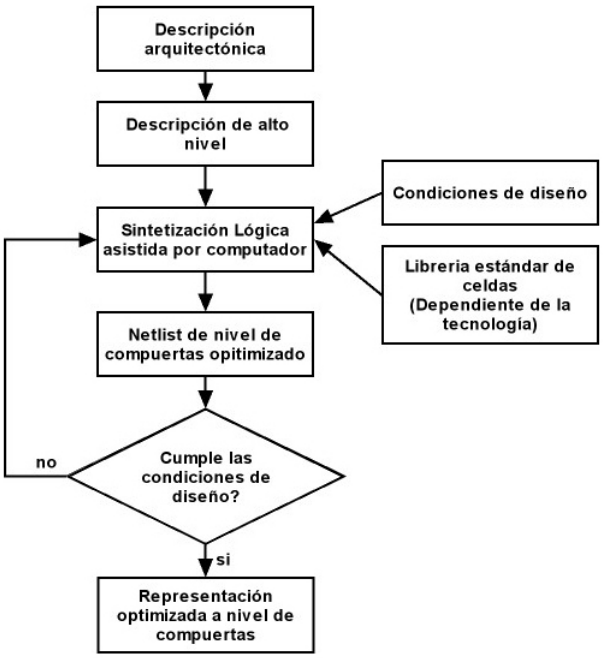


Figure 4–2: Sintetización lógica asistida por computador
Modificado de [24]

Para el propósito de síntesis lógica, los diseños son escritos en un lenguaje HDL (*Hardware Description Language*) a un nivel de transferencia de registros (RTL). El término RTL es usado por un estilo de descripción HDL que utiliza una combinación de funciones

[6] GNU Debugger.

del nivel de flujo de datos y del nivel comportacional. Verilog y VHDL son los 2 HDLs más populares usados para describir la funcionalidad a un nivel RTL. Cada uno de estos lenguajes presenta ciertas características fundamentales inherentes[7] .

4.3 Configuración de la plataforma

La plataforma conformada por MEXGBA y GBA constituye un sistema de desarrollo robusto desde cualquier punto de vista. La mezcla de procesamiento secuencial y concurrente de altas capacidades proporciona alta configurabilidad permitiendo desarrollar una gran gama de aplicaciones.

4.3.1 Generando el código para el GBA

Desde la perspectiva del programador, el sistema está compuesto por:

CPU	ARM7TDMI a 16.78Mhz
Memoria	De 8 a 11 diferentes áreas de memoria (dependiendo del *Game Pack*)
E/S	Funciones especiales de HW disponibles para el programador, principalmente concernientes a gráficos, sonido, DMA, timers, comunicación serial, entradas de teclado, e interrupciones

Los programas que corren en el GBA están usualmente contenidos en un Game Pack que consiste básicamente en una memoria ROM donde el código compilado y los datos son almacenados. Los programas pueden acceder a HW especializado para gráficas, sonido y otros periféricos entrada/salida a través de periféricos E/S mapeados en memoria (memory-mapped I/O). Esto proporciona un medio de comunicación con HW escribiendo y leyendo de direcciones específicas de memoria que están "mapeadas" a funciones internas de HW. Por ejemplo, escribir a la dirección `0x4000000` con el valor `0x100` indica al HW que debe "habilitar el Fondo 0 y el modo de gráficos 0".

Los lenguajes de programación C, C++, y ARM/Thumb ASM son los más comunes usados en el desarrollo en el GBA, principalmente porque son rápidos y relativamente de bajo nivel (existe un gran grado de correspondencia entre la estructura del lenguaje y la instrucción del lenguaje inherente en el set de la arquitectura). Miembros de la comunidad de desarrolladores del GBA han armado algunos kits de desarrollo que simplifican ampliamente la tarea de configurar un compilador C/C++ para el desarrollo en el GBA. Algunos de estos kits que son ampliamente conocidos son el DevkitAdvance, DevKitARM y el HAM [25].

Devkit-Advance

En el desarrollo de este proyecto se utilizó el DevkitAdvance. Este DevKit (kit de desarrollo) no usa propiedad intelectual de Nintendo. Permite crear archivos que funcionan con el GBA y con sus emuladores y está compuesto en su mayoria por las mismas herramientas que Nintendo proporciona a sus desarrolladores licenciados. Después de "instalado" (ver Anexo B, B.1.2 en la pág. 61) requiere, como mínimo, solamente 2

[7] Algunos desarrolladores prefieren Verilog por su parecido con C, por ejemplo.

comandos sin opciones especiales. Por ejemplo, suponiendo un archivo llamado `juego.c` con los siguientes contenidos [8] :

```
int main(void) { return 0; }
```

Se puede compilar con los siguientes comandos:

```
gcc -o juego.elf juego.c
objcopy -O binary juego.elf juego.bin
```

Para compilar programas escritos en C++ existen dos maneras. Una de ellas consiste en usar el compilador g++ en lugar del gcc. g++ es igual al gcc solamente que el envía opciones a los subprogramas que realizan el trabajo de compilación en C++. La segunda, consiste en añadir la opción `-lstdc++` a la linea de comando. Esto le dice al gcc que debe hacer un vínculo con la librería estándar de C++.

Visual Boy Advance

El Visual Boy Advance (VBA) es uno[9] de los muchos emuladores que emulan el GBA. Esta herramienta permite simular el código compilado proporcionando un método sencillo y eficiente de depuración. El uso del emulador se justifica por la rapidez que este proporciona (el probar el código en el emulador es un proceso mucho más rapido que el probarlo en el GBA).

Figure 4–3: Emulador Visual Boy Advance

Un emulador permite al desarrollador comprobar el funcionamiento de su código en aspectos relacionados con periféricos como pantalla, teclado, sonido, timers, etc. Periféricos externos al GBA conectados por medio de la tarjeta de expansión pueden ser simulados con datos estáticos y los códigos relacionados con estos pueden ser depurados con ayuda de la pantalla.

[8] Es importante notar que debe usarse **int main(void)** como punto de entrada.

[9] Entre los emuladores disponibles se encuentran el CowBite, no$gba y el Mappy.

Registros

El GBA está compuesto por diferentes dispositivos responsables de la creación de sonidos e imágenes que acompañan a la mayoría de los juegos. Existe un procesador de sonido, un procesador de video, dispositivos de memoria y el procesador principal; algunos de estos forman parte de un ASIC[10] (ver fig 2–2).

Cuando se escribe código C para describir eventos en el programa, se está controlando directamente el procesador del GBA. Pero, el GBA no trabaja solo y generalmente se debe tener algún nivel de control sobre lo que el resto del sistema está haciendo, esto se logra con el uso de registos de hardware.

Empezando en la dirección de memoria `0x4000000` y continuando por algunos bytes está el espacio de registros mapeados en memoria. Esto significa que si se escribe algún valor arbitrario en la dirección `0x4000000` entonces se estará escribiendo en un registro que tendrá algún efecto sobre el sistema [26].

Para información detallada referente a registros refiérase al Anexo A en la página 53.

Ejemplo. *El registro de control del display: REG_DISPCNT*

El registro REG_DISPCNT es un registro de 16-bit que reside en la dirección de memoria `0x400:0000`. Es responsable por el control básico de la pantalla y se encarga de encender o apagar capas de fondo, del control de la memoria de *sprites*, de establecer el modo de pantalla y de algunas otras cuestiones. Cada bit en un registro tiene un propósito específico, por ejemplo, habilitar bit 8 del REG_DISPCNT habilita el fondo `0` como se muestra en la tabla 4–1.

Table 4–1: Registro REG_DISPCNT en `0x400:0000`

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
OW	W1	W0	OBJ	BG3	BG2	BG1	BG0	B	OM	HB	DB	GB	MODO		

0-2 (MODO):Estos bits controlan el modo de pantalla. Este puede ser 0, 1 ,2 ,3, 4, ó 5.
3 (GB): Este es seteado por HW si un cartucho de GB o GBC es detectado.
4 (DB): Controla cuál buffer es el buffer activo en los modos 4 y 5.
5 (HB): Permite actualizar la OAM durante un HBlank.
6 (OM): Esta bandera determina el modo de mapeo usado para los sprites.
7 (B): Forza la pantalla a vacio causando que la pantalla se vuelva blanca.
8-C (BGx,OBJ): Estos bits habilitan los fondos y OBJ (sprites) respectivos.
D-F (OW,W1,W0): Estos bits habilitan las ventanas.

[10] Un ASIC se define como un circuito integrado de aplicación especfíca o *Application Specific Integrated Circuit* por sus siglas en ingés.

4.3.2 Configuración del módulo de expansión

El módulo de expansión contiene en la memoria flash el código compilado que el procesador del GBA va a leer y ejecutar, además, contiene la FPGA, en la cual estarán sintetizados los periféricos que el desarrollador en su aplicación requiera usar. La configuración de estos dispositivos se realiza por diferentes medios (diferentes puertos de programación y SW de programación).

Paraun mayor detalle de la configuración de arranque del módulo de expansión (todos los dispositivos en blanco), refiérase al *Anexo D en la pág. 86*.

Xpcomm

Charmedlabs [15], compañía creadora de la tarjeta Xport, creó una interfaz de programación (`xpcomm`) que permite comunicarse con los dispositivos en la tarjeta de expansión y configurarlos mediante diferentes comandos (ver tabla 4–2[11] .). Esta interfaz envía por el puerto paralelo del PC las tramas al puerto CPORT de la tarjeta de expansión. Para configurar la FPGA se pueden enviar cadenas de bits (.bit) generadas por herramientas de síntesis como Xilinix ISE o WebPACK. Para programar la flash Xpcomm acepta archivos de programa tanto en formato binario puro (.bin) y S-records (.srec). Estos achivos son usualmente generados por compiladores de C/C++ como *gcc* o *ARM SDT*.

Ya que el Xpcomm funciona sobre Cygwin, la compilación de los archivos C/C++ se puede realizar desde la misma consola utilizando makefiles[12] .

JTAG

En el estado inicial, el módulo de expansión carece de cualquier configuración. La flash está en blanco y por tanto no existe código que la FPGA pueda leer para configurarse. Es necesario entonces configurar inicialmente el CPLD quien es el actor principal para la programación. De los dos elementos lógicos programables del módulo de expansión, el CPLD es el único que se encuentra conectado a una interfaz JTAG[13] para *Boundary Scan*, ideal para programación en sistema (ISP). La configuración del CPLD por JTAG se puede realizar usando la herramienta iMPACT incluida dentro del paquete Xilinx ISE (ver *Anexo D en la página 86*).

GPIO

Los pines de propósito general (GPIO) son tal vez la manera más sencilla de conectarse con el mundo real. Estos pines se pueden configurar como salidas o entradas digitales y así usarse para leer o escribir información.

---

[11] Para una tabla de instrucciones más detallada refierase a C.4 en la página 72

[12] Un makefile es un archivo bashscript que contiene los comandos necesarios para ejecutar la tarea de compilación. Este archivo contiene las instrucciones de las las etapas de compilación que se deben realizar paso a paso (*ver B.1.1 en la pág. 58*).

[13] Una interfaz JTAG es una interfaz especial de cuatro o cinco pines agregados a un chip, diseñada para que multiples dispositivos en una tarjeta puedan tener sus lineas JTAG encadenadas en serie, de manera tal que puedan ser sometidos a pruebas con un solo puerto JTAG y de esta manera tener acceso a todos los dispositivos de la tarjeta.

Table 4–2: Algunas operaciones del Xpcomm

`-pf` <*archivo de programa*>	Programa la flash con el archivo de programa (binario o S-record). Nota, cuando se realiza el procedimiento de programación de la flash, Xpcomm realiza una suma de comprobación para asegurar la integridad de los datos
`-vf` <*archivo de programa*>	Verifica la flash con el archivo de programa especificado (binario o S-record).
`-pvf` <*archivo de programa*>	Programa y verifica la flash con archivo de programa especificado (binario o S-record).
`-rf` <*dump file*><*longitud*>	Lee y descarga los contenidos de la flash (*longitud* número de palabras) al archivo especificado *dump file*.
`-pc` <*archivo bitstream/directorio*>	Programa la configuración de la FPGA con el archivo *bitstream* especificado. Si un directorio es especificado, Xpcomm automáticamente escogerá el *bitstream* en el directorio que es compatible.
`-vc` <*archivo bitstream/directorio*>	Verifica la configuración de la FPGA con el archivo bitstream especificado. Si un directorio es especificado, Xpcomm automáticamente escogerá el bitstream en el directorio que es compatible
`-c` <*archivo bitstream/directorio*>	Programa la FPGA con el archivo *bitstream* especificado usando el modo esclavo externo. Esto no modifica los contenidos de la configuración no-volátil de la FPGA almacenados en flash. La configuración se perderá al desenergizar el sistema.
`-i`	Recupera y muestra información de la tarjeta de expansión, actualiza la configuración de la FPGA para la programación de la flash si es necesario
`-u`	Forza una actualización de la configuración de la FPGA para la programación de la flash.

Cada señal E/S tiene dos bits de control: un bit de dirección y un bit de datos. Poniendo el bit de dirección en 1 configura la señal de E/S correspondiente como salida. Poniendo el bit de dirección en 0 configura la señal de E/S como entrada. El bit de datos refleja la lógica de estado de la señal de E/S correspondiente sin importar si fue configurada como entrada o salida. Por conveniencia, estos bits son agrupados en registros de 16-bit. Los bits de dirección agrupados forman los "registros de dirección de datos" (data direction registers - DDRs) y los bits de datos los "registros de datos" (data registers - DRs). La tabla 4–3 detalla estos registros.

En la tabla 4–3, Pn corresponde a la enésima señal del conector (A ó B) (ver *Señales de los conectores GPIO y de la FPGA* en C.7, pág. 75).

Entonces, por ejemplo, para habilitar las señales PA0 a PA7 como salida y PA8 a PA15 como entrada, el registro DDR0 se debe configurar de la siguiente manera:

```
*((volatile unsigned short *)0x9ffc400) = 0x00ff;
```

Para configurar PA0 a PA3 como niveles altos y PA4 a PA7 como niveles bajos, se debe configurar DR0 como sigue:

```
*((volatile unsigned short *)0x9ffc408)=0x000f;
```

Table 4–3: Mapeo de registros del GPIO

Nombre	Dirección	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DDR0	0x9ffc400	PA15	PA14	PA13	PA12	PA11	PA10	PA9	PA8	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
DDR1	0x9ffc402	0	PA30	PA29	PA28	PA27	PA26	PA25	PA24	PA23	PA22	PA21	PA20	PA19	PA18	PA17	PA16
DDR2	0x9ffc404	PB15	PB14	PB13	PB12	PB11	PB10	PB9	PB8	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
DDR1	0x9ffc406	0	PB30	PB29	PB28	PB27	PB26	PB25	PB24	PB23	PB22	PB21	PB20	PB19	PB18	PB17	PB16
DR0	0x9ffc408	PA15	PA14	PA13	PA12	PA11	PA10	PA9	PA8	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
DR1	0x9ffc40a	0	PA30	PA29	PA28	PA27	PA26	PA25	PA24	PA23	PA22	PA21	PA20	PA19	PA18	PA17	PA16
DR2	0x9ffc40c	PB15	PB14	PB13	PB12	PB11	PB10	PB9	PB8	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
DR3	0x9ffc40e	0	PB30	PB29	PB28	PB27	PB26	PB25	PB24	PB23	PB22	PB21	PB20	PB19	PB18	PB17	PB16

Capítulo 5

Aplicación

Una tarea de diseño implica la consecución de un conjunto de subtareas, las cuales son conocidas comúnmente como flujo de diseño. En este flujo de diseño se recorren diferentes niveles de abstracción (ver tabla 5–2) los cuales hacen parte del diseño de cualquier sistema de circuitos electrónicos.

Table 5–1: Niveles de abstracción de circuitos electrónicos

Nivel de abstracción	**Comportamiento**	**Estructura**	**Físico**
Nivel de sistema	Procesos, Prestaciones	Procesadores, Memorias	Particiones básicas
Nivel algorítmico	Algoritmos	Unidades funcionales	Particiones básicas
Nivel de transferencia de registros	Transferencia de registros	Unidades funcionales, Registros	Macroceldas
Nivel lógico	Ecuaciones booleanas, Transiciones de estados	Puertas lógicas, Latches	Celdas básicas
Nivel de dispositivo	Ecuaciones eléctricas	Transistores, Conexiones	Metal, Polisilicio, Difusión

Según la tabla anterior, los sistemas embebidos se encuentran en el nivel de sistema. En este nivel, los componentes que forman al sistema son procesadores y memorias. No obstante, no se debe generalizar a los procesadores como procesadores de propósito general, sino como cualquier dispositivo capaz de procesar información, es decir, cualquier dispositivo de aplicación especifíca también estaría dentro de estos elementos. La tarea de diseño debe acabar en la descripción física del nivel de dispositivo, sin embargo, la descripción final no tiene por qué ser un único integrado monolítico (como sucede normalmente en microelectrónica), sino que puede ser una tarjeta impresa, o un dispositivo MCM (módulos multi-chip).

Un flujo de diseño bien establecido para los sistemas embebidos es el mostrado en la fig 5–1 en el cual se observa claramente una serie de pasos secuenciales, lo que implica que para avanzar en las etapas correspondientes hay que realizar una serie de subtareas.

En primer lugar se debe realizar un modelado del sistema completo. El modelo obtenido de esta subtarea servirá para fijar los diferentes algoritmos que están involucrados en la operación del sistema (ya sean algoritmos de control o de operación con datos).

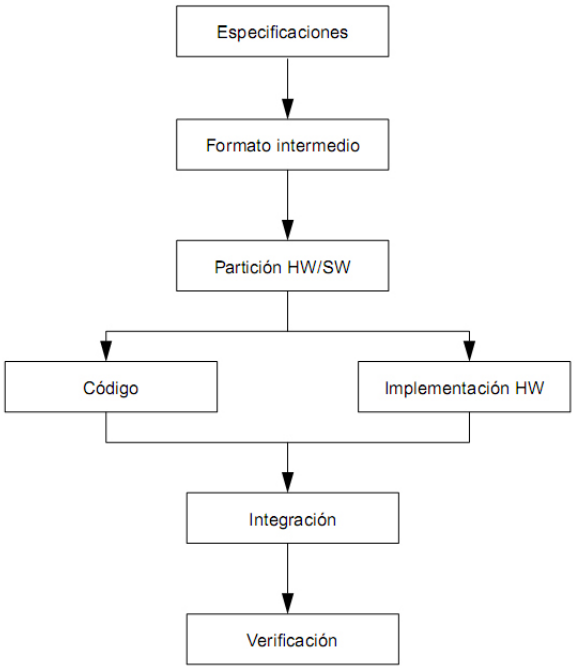


Figure 5–1: Flujo de diseño de un sistema embebido
Tomado de [27]

Un modelo formal debería contar con los siguientes componentes: una descripción funcional, un conjunto de propiedades, un conjunto de índices de desarrollo; y un conjunto de restricciones.

La siguiente subtarea es el particionado del sistema. Una vez que haya concluido, se tiene el comportamiento del sistema dividido en varias funciones de menor complejidad, con el propósito de abordar el problema completo como varios problemas desconectados entre sí. Se sigue con el particionado hardware/software, subtarea que determinará qué funciones se realizarán mediante dispositivos de aplicación específica (hardware) o mediante programación (software).

Una vez que se ha realizado la separación se debe realizar la tarea de síntesis de cada una de las partes para finalmente realizar una subtarea de *mapeado* que traducirá la descripción funcional en programas que pueden ser ejecutados en unos determinados procesadores o en una conexión de dispositivos hardware digitales y/o analógicos (dependiendo de la zona de particionado de la descripción funcional).

Aunque se ha tratado como un flujo secuencial de tareas, esta situación no se adapta a la realidad. Realmente, todas las subtareas están relacionadas entre sí, y no únicamente las adyacentes. Por ejemplo, el hecho de tener un determinado procesador con sus propias funciones (estaríamos hablando de la subtarea de *mapeado*), puede causar que el comportamiento global sea particionado en una serie de funciones (subtarea de particionado), que podrían cambiar si se dispusiese de otro tipo de procesador.

5.1 Descripción Funcional

La aplicación que se escogió para demostrar el funcionamiento de la plataforma es un robot seguidor de línea. Este robot debe avanzar siguiendo una línea negra de 18mm

de ancho ubicada de manera tal que evalúe la respuesta del sistema a diferentes tipos de trayectorias.

El sistema exhibe su simplicidad permitiendo claramente visualizar el funcionamiento de la plataforma usando una cantidad mínima de recursos.

5.1.1 Hardware

El móvil se basa en una arquitectura diferencial (fig 5–2), en esta arquitectura se sitúa un motor conectado a cada una de las ruedas tractoras, de forma que variando la velocidad de cada uno de los motores se consigue que el motor vaya recto, girando más o menos rápido. Una de las grandes ventajas de esta cinemática es que el robot puede girar sobre sí mismo, haciendo que las ruedas vayan a la misma velocidad pero en sentido contrario.

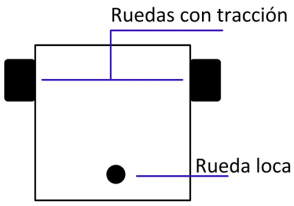


Figure 5–2: Arquitectura diferencial

Para la propulsión se usan motores DC acompañados de un juego de engranajes para aumentar el torque (fig 5–3).

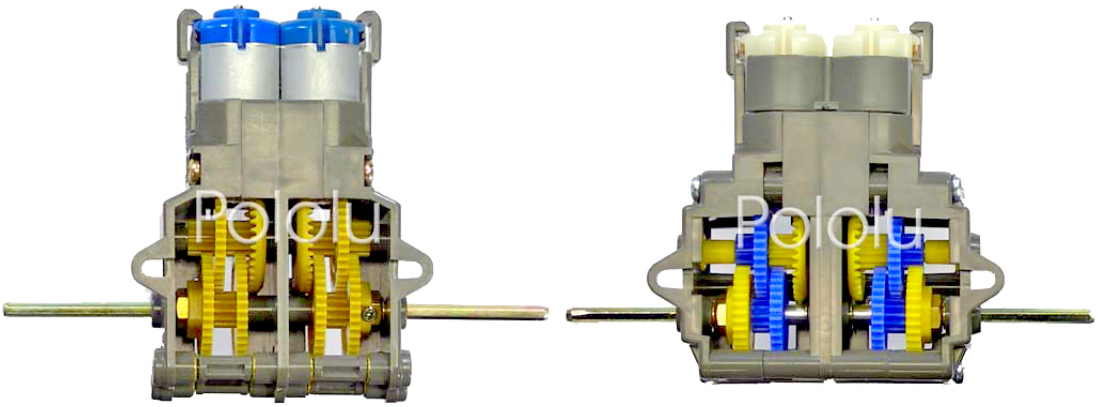


Figure 5–3: Caja reductora y motores

- Interfaz E/S:
  El módulo de expansión del GBA está provisto de un máximo de 62 señales distribuidas en los conectores USER A y USER B, las cuales son totalmente programables por medio de la interfaz GPIO (ver tabla 4–3), sintetizada en la FPGA. Este módulo provee al sistema con comunicación bidireccional al entorno, haciendo posible sensar y manipular actuadores.

  La aplicación seleccionada para el proyecto no requiere gran cantidad de cálculo computacional, así mismo no requiere las 62 señales de propósito general disponibles

en el sistema, por tal razón se usaron solo 15 señales de propósito general para los sensores y 10 para PWM[1] .

Los módulos del sistema están concebidos para que su funcionamiento sea sencillo y requiera mínima intervención del usuario. La FPGA se encarga de configurar y gestionar los periféricos de tal manera que todos los módulos cuenten con, un registro de *habilitación* o de *estado*, registros con los datos de cada módulo, valores de los puertos (para el módulo GPIO) y la modulación de cada canal (para el módulo PWM), logrando así que el usuario simplemente con habilitar el módulo pueda leer y escribir en dichas señales como si fueran una variable del sistema. La descripción del módulo GPIO se muestra a continuación:

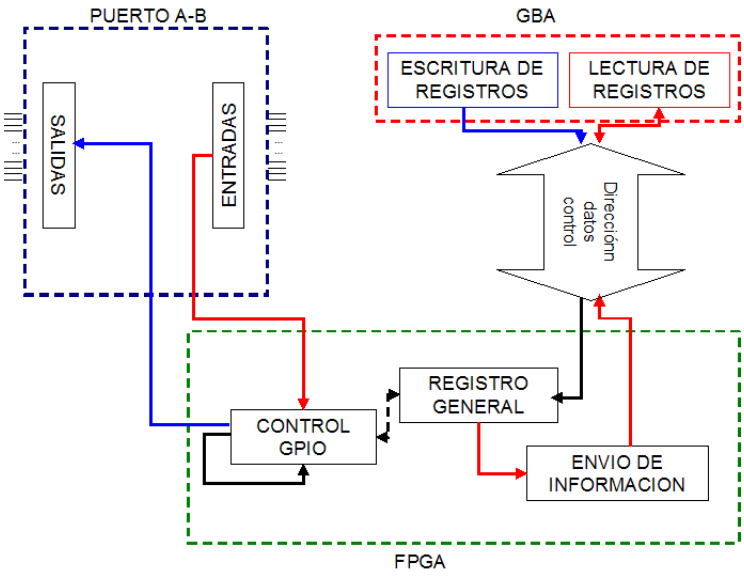


Figure 5–4: Diagrama de funcionamiento del módulo GPIO

Table 5–2: REGISTROS GPIO

REGISTRO	**DIRECCIÓN**
Registro de dirección	0x9ffc400
Datos	0x9ffc401

El registro general consta de dos partes, una contiene la información que indica la dirección (salida o entrada) de cada señal en el GPIO (Registro de dirección de datos o DDR), y otro registro que contiene la información que llevan dichas señales (Registro de datos o DR), cada uno de estos registros es de 16-bits.

- Sensores:
  El móvil toma las decisiones de la trayectoria a seguir mediante un arreglo de cuatro sensores QRD1114 ubicados en la parte frontal del mismo, como se muestra en la fig 5–5.

[1] PWM hace referencia a una técnica de modulación del pulso (*Pulse Width Modulation*) que se usa para varia el nivel de D.C. que es entregado a un dispositivo, en este caso motores.

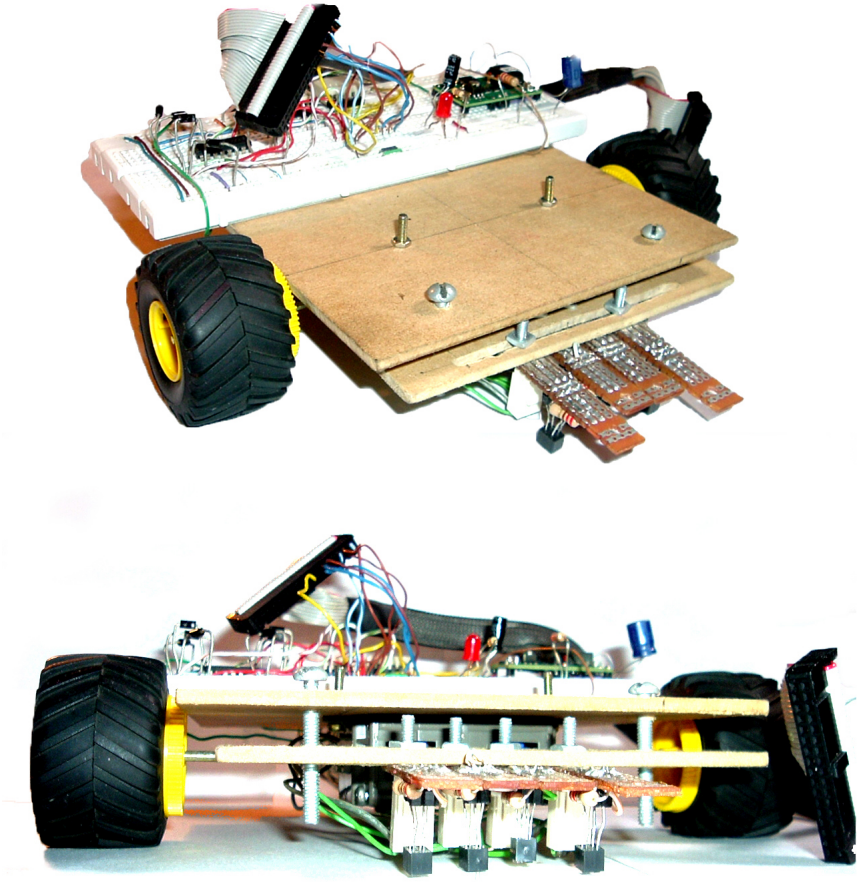


Figure 5–5: Móvil seguidor de linea

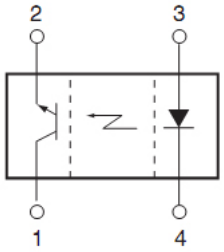


Figure 5–6: QRD1114

El QRD1114 es un sensor reflectivo de objetos, muy popular como sensor de línea y ampliamente usado en diseños de robots. Tiene un rango efectivo de aproximadamente 6mm. Está compuesto por un diodo emisor y un fototransistor. El diodo emite una señal infrarroja que rebota sobre la superficie (objeto) a censar, esta señal es captada por el fototransistor quien de acuerdo a la intensidad de la señal genera una corriente de activación que se verá representada en un voltaje de salida proporcional a la intensidad de señal infrarroja censada por el fototransistor (ver fig 5–6).

Los sensores se encuentran ubicados sobre una plataforma diseñada para proveer flexibilidad a esta ubicación, con el propósito de convertir al sistema en una herramienta más didáctica.

Para una topología de sensores como la que se muestra en la fig 5–5 se requiere una rutina de control (ver fig 5–8) sencilla que indicará al sistema que motor debe mover a que velocidad para seguir en la trayectoria.

- Señales de control: El móvil está provisto de dos motores que se encargan de ubicarlo en la trayectoria a seguir, estos actuadores son controlados por medio de señales PWM generadas en el módulo PWM sintetizado en la FPGA. La posibilidad que ofrece la FPGA cómo elemento alojador de módulos HW permite liberar de carga a la CPU y realizar táreas en concurrencia. El módulo PWM como ya se mencionó

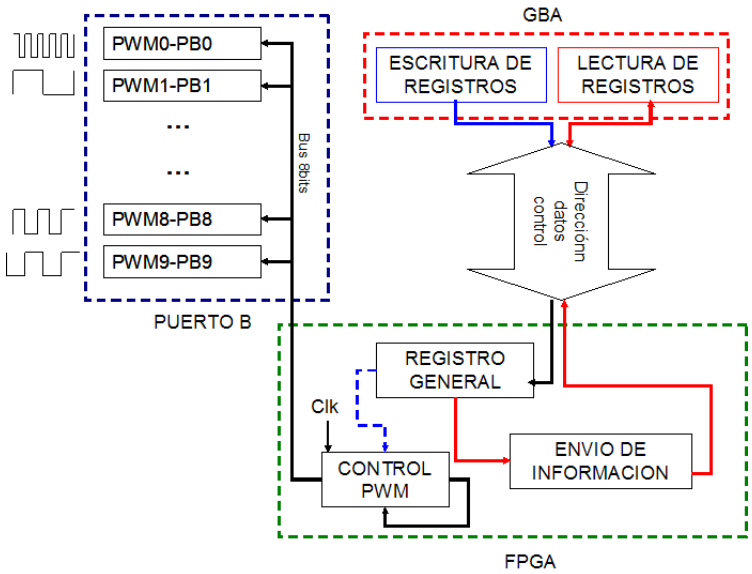


Figure 5–7: Diagrama de funcionamiento del módulo PWM

está compuesto por hardware que se encarga de su funcionamiento y por software que se encarga de su control, el diagrama del módulo se muestra en la figura 5–7.

Todos los módulos implementados en el sistema cuentan con una estructura similar. Dentro del hardware contienen un registro general que almacena la información pertinente para el funcionamiento de dicho módulo, un bloque encargado de enviar información al GBA y un módulo de control que estará realizando una tarea periódica para mantener actualizado el registro general y así poder responder rápidamente ante alguna petición del software.

En la tabla 5–3 se pueden detallar los registros mapeados del PWM.

- Etapa de Potencia: Para este módulo se decidió usar un IC conocido popularmente como Puente H (L293) quien proporcionará la potencia necesaria a las señales de control de los actuadores. Recordando que el objetivo de este trabajo no consiste en realizar una aplicación con restricciones estrechas de potencia, no entraremos en detalle de la justificación de la selección de los dispositivos que componen la misma y sugerimos al lector que estos sean tomados simplemente de manera funcional.

Table 5–3: REGISTROS PWM

REGISTRO	**DIRECCIÓN**
Habilitadores	0X9FFC800
Modulación PWM1-PWM0	0X9FFC801
Modulación PWM3-PWM2	0X9FFC802
Modulación PWM5-PWM4	0X9FFC803
Modulación PWM7-PWM6	0X9FFC804
Modulación PWM9-PWM8	0X9FFC805

- Condiciones Eléctricas: Para el funcionamiento del móvil se requieren dispositivos que deben ser energizados adecuadamente, sensores, actuadores, módulos de potencia, etc, cada uno necesita condiciones eléctricas específicas. La mayoría de estos no presentan condiciones de polarización que se consideren fuera de alcance, teniendo en cuenta que el móvil estará energizado por baterías recargables. Sin embargo los motores exigen un tipo de control de las baterías más específico pues requieren 1.2A c/u de corriente de arranque, por tal motivo se hace uso de un ISR (*Integrated Switching Regulator*), PTN78020 del fabricante *Texas Instruments*, quien actúa como un regulador de voltaje proveyendo una tensión constante de salida a una carga variable a una eficiencia aproximada del 96%.

5.2 Software

El móvil debe tomar decisiones de trayectoria de acuerdo a las señales proporcionadas por los sensores. Estas decisiones están modeladas en un algoritmo secuencial que debe contener todos los posibles casos para valores de las señales de entrada y sus respectivas acciones representadas en señales de salida.

En la figura 5–8 se muestra un diagrama que ilustra la simpleza del algoritmo al repartir carga computacional con los módulos sintetizados en la FPGA.

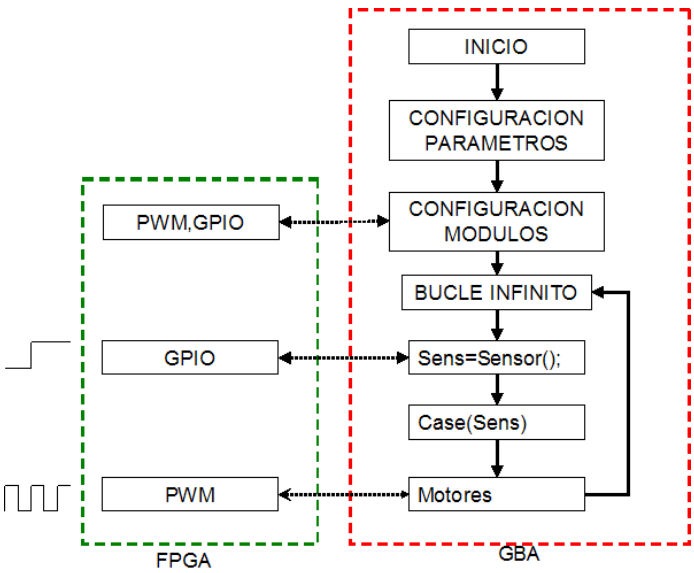


Figure 5–8: Diagrama general de funcionamiento del seguidor

En la figura 5–8 se omiten funciones[2] del algoritmo para la facilitar la comprensión al lector.

5.3 PCB

Para interconectar los módulos requeridos de la aplicación se desarrolló un circuito impreso (ver fig 5–9) que permite acceso a los otros puertos del GPIO que no tengan funciones asignadas. Este PCB reúne los elementos que se han descrito y conforma de una manera básica una etapa de potencia con salidas integradas para otras señales de propósito general, de manera tal que otros módulos puedan ser añadidos.

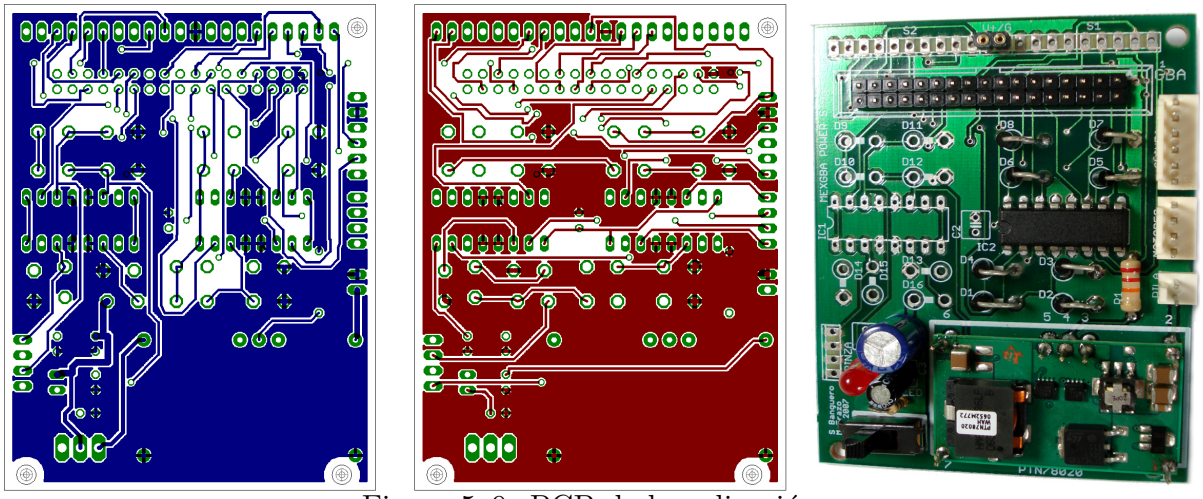


Figure 5–9: PCB de la aplicación

[2] Entre las funciones no mostradas están el timer con el cual se están generando retardos y la rutina de interrupción del timer.

Capítulo 6

Conclusiones y Observaciones

En este trabajo se desarrolló un módulo de expansión para la consola de video juegos portátil de Nintendo, **Game Boy Advance**, que permite total acceso a los dispositivos que la componen (CPU ARM 32bit, LCD color 240x160, teclado, sonido, etc.) actuando como un "cartucho de juego" con capacidad de almacenar hasta 4MB de código además de ofrecer la posibilidad de sintetización de hardware para complementar aplicaciones que requieran módulos periféricos. Esta herramienta constituye un aporte para el desarrollo y elaboración de prototipos para aplicaciones embebidas que requieran altos niveles de procesamiento dentro del grupo de Investigación CPS y por consiguiente de la E3T.

Los anteriores desarrollos realizados con consolas tipo GameBoy proveyeron una base sólida de información que facilitó la realización del módulo de expansión, especialmente, la tarjeta de expansión comercialmente conocida como Xport, desarrollada por Charmedlabs en E.U.A, quien constituye el modelo de sistema sobre el cual se desarrolló el módulo. Este módulo de expansión pretendió constituir desde su origen una herramienta de aprendizaje y apropiación de nuevas tecnologías que permitieran el acceso a nuevos saberes útiles a la sociedad.

El módulo de expansión permite el desarrollo de una amplia gama de aplicaciones a un bajo costo[1] , tanto económico como de aprendizaje, pues al estar basado en una consola de videojuegos popular, tiene el respaldo de cientos de desarrolladores aficionados que han construido una fuerte documentación y librerías de dominio público, permitiendo una programación de alto nivel.

El Game Boy Advance es un sistema avanzado de juego que incluye periféricos interesantes como LCD, Teclado y Sonido, entre otros. Estos abren la posibilidad a una programación interactiva del módulo, donde el sistema programado puede por ejemplo tener constantes reconfigurables en tiempo de ejecución.

[1] El costo neto de fabricación (sólo componentes) de la tarjeta con componentes es aproximadamente 195000 COP, asumiendo USD = 2200 COP

Para el diseño del circuito impreso (PCB) se tuvieron en cuenta los parámetros más relevantes correspondientes a diseños digitales de frecuencias medianamente altas, como lo son: dimensiones máximas de pistas y geometrías para evitar topologías de antena, consideraciones de corrientes y anchos de pista, ubicación de componentes y dimensiones de la tarjeta restringidas por el cartucho de juego del GBA, entre otros. Este diseño se realizó en el Software OrCAD Layout ruteado en su totalidad de manera manual con las especificaciones que se muestran en la tabla 6–1.

Table 6–1: Especificaciones mínimas del PCB en mm.

Min. distancia entre cobre	0.16
Min. tamaño annular ring	0.13
Min. tamaño hueco de via	0.31

La fabricación del circuito impreso (PCB) para el módulo de expansión demostró el alto costo y bajo nivel técnico que presenta la oferta nacional, motivo por el cual fue necesario consultar el panorama mundial con el propósito de obtener un precio justo con el nivel técnico requerido. Estas especificaciones fueron forzadas por las dimensiones del conector Game Pack del GBA además de las dimensiones y características de los dispositivos usados en el módulo. La opción escogida cumple con los requerimientos técnicos[2] , económicos y de tiempo.

Antes de seleccionar los parámetros mínimos del PCB hay que verificar las especificaciones del fabricante escógido teniendo en cuenta que estas varían de acuerdo a las características seleccionadas, algunas veces de manera mutuamente excluyente; por ejemplo en el caso de este trabajo, el grosor del material FR4 de la tarjeta (2mm) fue seleccionado especialmente para un ajuste perfecto en el conector Game Pack del GBA, la selección de este parámetro afecta el tamaño mínimo de los huecos pasantes (VIA STH) exigiendo un rediseño y rehubicación de los mismos, que se traduce en incremento de costos por horas de trabajo y en un notable retraso en el cronograma.

Aunque el prototipo presentó un inconveniente relacionado con el diseño esquemático (Una resistencia de pull-down para habilitación de la memoria flash) y otros inconvenientes menos significativos en el PCB (errores de comunicación con el fabricante) los verdaderos problemas para lograr el correcto funcionamiento de la tarjeta fueron debidos a la soldadura, pues fue una actividad que se realizó sin la suficiente documentación. El problema finalmente se ubicó en la sustancia usada para facilitar el proceso, comúnmente denominada flux. A pesar de que el flux realiza su trabajo de manera eficiente (eliminar los óxidos de la superficie de los metales para una unión eficiente) y a pesar de que una

[2] 71.24 USD por fabricación de 4 tarjetas en material FR4, grosor de 2mm, peso del cobre de 1oz., 582 vias y 32 golden fingers (añadir las características de la tabla 6–1) + 35 USD de envío

vez calentado es totalmente neutral (no conduce), puede dejar residuos pegajosos que absorben humedad y polvo potencialmente conductor. Estos residuos pegajosos en muchos casos son difíciles de eliminar si se encuentran debajo de los IC, incluso con largas jornadas expuestos a disolventes como el isopropanol, por tal motivo es recomendable usar flux de recinas sintéticas o "No Clean" quienes contienen en menor cantidad residuos sólidos de Colofonia (resina natural base del flux).

La tarea de compilar los algoritmos se simplifica en gran medida al utilizar el kit de desarrollo XportDevKit, el cual empleando makefiles ahorra gran cantidad de tiempo en dicha labor logrando que esta tarea se pueda realizar ejecutando el makefile. Así mismo el makefile se encarga de la tarea de programación, de esta manera en un solo paso se esta compilando y programando el código en el MEXGBA de una manera sencilla y rápida.

ANEXOS

Anexo A

Memoria del GBA

Table A–1: Mapa de Registros del GBA: Gráficos

Graphics Hardware Registers	
0x4000000	REG_DISPCNT (the display control register)
0x4000004	REG_DISPSTAT
0x4000006	LCY/REG_VCOUNT
	Background Registers
0x4000008	REG_BG0CNT
0x400000A	REG_BG1CNT
0x400000C	REG_BG2CNT
0x400000E	REG_BG3CNT
0x4000010	REG_BG0HOFS Horizontal scroll co-ordinate for BG0 (Write Only)
0x4000012	REG_BG0VOFS Vertical scroll co-ordinate for BG0 (Write Only)
0x4000014	REG_BG1HOFS Horizontal scroll co-ordinate for BG1 (Write Only)
0x4000016	REG_BG1VOFS Vertical scroll co-ordinate for BG1 (Write Only)
0x4000018	REG_BG2HOFS Horizontal scroll co-ordinate for BG2 (Write Only)
0x400001A	REG_BG2VOFS Vertical scroll co-ordinate for BG2 (Write Only)
0x400001C	REG_BG3HOFS Horizontal scroll co-ordinate for BG3 (Write Only)
0x400001E	REG_BG3VOFS Vertical scroll co-ordinate for BG3 (Write Only)
	Background Rotation / Scaling Registers (Write Only)
0x4000020	REG_BG2PA (BG2 Read Source Pixel X Increment)(Write Only)
0x4000030	REG_BG3PA (BG3 Read Source Pixel X Increment) (Write Only)
0x4000022	REG_BG2PB (BG2 Write Destination Pixel X Increment) (Write Only)
0x4000032	REG_BG3PB (BG3 Write Destination Pixel X Increment) (Write Only)
0x4000024	REG_BG2PC (BG2 Read Source Pixel Y Increment) (Write Only)
0x4000034	REG_BG3PC (BG3 Read Source Pixel Y Increment) (Write Only)
0x4000026	REG_BG2PD (BG2 Write Destination Pixel Y Increment) (Write Only)
0x4000036	REG_BG3PD (BG3 Write Destination Pixel Y Increment) (Write Only)
0x4000026	REG_BG2PD (BG2 Write Destination Pixel Y Increment) (Write Only)
0x4000036	REG_BG3PD (BG3 Write Destination Pixel Y Increment) (Write Only)
	Windowing Registers
0x4000040	REG_WIN0H (Window 0 X Coordinates) (Write Only)
0x4000042	REG_WIN1H (Window 1 X Coordinates)(Write Only)
0x4000044	REG_WIN0V (Window 0 Y Coordinates) (Write Only)
0x4000046	REG_WIN1V (Window 1 Y Coordinates)(Write Only)
0x4000048	REG_WININ (Inside Window Settings)
0x400004A	REG_WINOUT (Outside Window and Sprite Window)
	Effects Registers
0x400004C	REG_MOSAIC (Write Only)
0x4000050	REG_BLDMOD
0x4000052	REG_COLEV (Write Only)
0x4000054	REG_COLEY (Write Only)

Table A–2: Mapa de Registros del GBA: Sonido

Sound Controls	
0x04000060	REG_SOUND1CNT_L (Sound 1 Sweep control)
0x04000062	REG_SOUND1CNT_H (Sound 1 Length, wave duty and envelope control)
0x04000064	REG_SOUND1CNT_X (Sound 1 Frequency, reset and loop control)
0x04000068	REG_SOUND2CNT_L (Sound 2 Length, wave duty and envelope control)
0x0400006C	REG_SOUND2CNT_H (Sound 2 Frequency, reset and loop control)
0x04000070	REG_SOUND3CNT_L (Sound 3 Enable and wave ram bank control)
0x04000072	REG_SOUND3CNT_H (Sound 3 Sound length and output level control)
0x04000074	REG_SOUND3CNT_X (Sound 3 Frequency, reset and loop control)
0x04000078	REG_SOUND4CNT_L (Sound 4 Length, output level and envelope control)
0x0400007C	REG_SOUND4CNT_H (Sound 4 Noise parameters, reset and loop control)
0x04000080	REG_SOUNDCNT_L (Sound 1-4 Output level and Stereo control)
0x04000082	REG_SOUNDCNT_H (Direct Sound control and Sound 1-4 output ratio)
0x04000084	REG_SOUNDCNT_X (Master sound enable and Sound 1-4 play status)
0x04000088	REG_SOUNDBIAS (Sound bias and Amplitude resolution control)
0x04000090	REG_WAVE_RAM0_L (Sound 3 samples 0-3)
0x04000092	REG_WAVE_RAM0_H (Sound 3 samples 4-7)
0x04000094	REG_WAVE_RAM1_L (Sound 3 samples 8-11)
0x04000096	REG_WAVE_RAM1_H (Sound 3 samples 12-15)
0x04000098	REG_WAVE_RAM2_L (Sound 3 samples 16-19)
0x0400009A	REG_WAVE_RAM2_H (Sound 3 samples 20-23)
0x0400009C	REG_WAVE_RAM3_L (Sound 3 samples 23-27)
0x0400009E	REG_WAVE_RAM3_H (Sound 3 samples 28-31)
0x040000A0	REG_FIFO_A_L (Direct Sound channel A samples 0-1)(Write Only)
0x040000A2	REG_FIFO_A_H (Direct Sound channel A samples 2-3)(Write Only)
0x040000A4	REG_FIFO_B_L (Direct Sound channel B samples 0-1)(Write Only)
0x040000A6	REG_FIFO_B_H (Direct Sound channel B samples 2-3)(Write Only)

Table A–3: Mapa de Registros del GBA: Comunicación Serial

Serial Communication Registers	
0x4000120	REG_SCD0
0x4000122	REG_SCD1
0x4000124	REG_SCD2
0x4000126	REG_SCD3
0x4000128	REG_SCCNT_L (Serial Communication channel control register)
0x400012A	REG_SCCNT_H (Serial Communication Source Register)

Table A–4: Mapa de Registros del GBA: Interrupciones

Interrupt Registers	
0x4000200	REG_IE (Interrupt Enable Register)
0x4000202	REG_IF (Interrupt Flags Regster)
0x4000204	REG_WSCNT (Wait State Control)
0x4000208	REG_IME (Interrupt Master Enable)
0x4000300	REG_HALTCNT_L (First Boot/Debug Control)
0x4000301	REG_HALTCNT_H (Low Power Mode Control)
0x4000800	REG_IMC_L (Internal Memory Control)
0x4000802	REG_IMC_H (Internal Memory Control)

Table A–5: Mapa de Registros del GBA: DMA

DMA Registers	
	DMA Source Registers (Write Only)
0x40000B0	REG_DMA0SAD (DMA0 Source Address)(Write Only)
0x40000BC	REG_DMA1SAD (DMA1 Source Address)
0x40000C8	REG_DMA2SAD (DMA2 Source Address)
0x40000D4	REG_DMA3SAD (DMA3 Source Address)
	DMA Destination Registers (Write Only)
0x40000B4	REG_DMA0DAD (DMA0 Destination Address)
0x40000C0	REG_DMA1DAD (DMA1 Destination Address)
0x40000CC	REG_DMA2DAD (DMA2 Destination Address)
0x40000D8	REG_DMA3DAD (DMA3 Destination Address)(Write Only)
	DMA Count Registers (Write Only)
0x40000B8	REG_DMA0CNT_L (DMA0 Count Register)
0x40000C4	REG_DMA1CNT_L (DMA1 Count Register)
0x40000D0	REG_DMA2CNT_L (DMA2 Count Register)
0x40000DC	REG_DMA3CNT_L (DMA3 Count Register)
	DMA Control Registers
0x40000BA	REG_DMA0CNT_H (DMA0 Control Register)
0x40000C6	REG_DMA1CNT_H (DMA1 Control Register)
0x40000D2	REG_DMA2CNT_H (DMA2 Control Register)
0x40000DE	REG_DMA3CNT_H (DMA3 Control Register)

Table A–6: Mapa de Registros del GBA: Timers

Timer Registers	
0x4000100	REG_TM0D (Timer 0 Data)
0x4000104	REG_TM1D (Timer 1 Data)
0x4000108	REG_TM2D (Timer 2 Data)
0x400010C	REG_TM3D (Timer 3 Data)
0x4000102	REG_TM0CNT (Timer 0 Control)
0x4000106	REG_TM1CNT (Timer 1 Control)
0x400010A	REG_TM2CNT (Timer 2 Control)
0x400010E	REG_TM3CNT (Timer 3 Control)

Table A–7: Mapa de Registros del GBA: Teclado

Keypad Input and Control Registers	
0x4000130	REG_KEY (The input register)(Read Only)
0x4000132	REG_P1CNT (Key Control Register)
0x4000134	REG_RCNT
0x4000140	REG_JOYCNT (JOY BUS Control Register)
0x4000150	REG_JOYRE_L
0x4000152	REG_JOYRE_H
0x4000154	REG_JOYTR_L
0x4000156	REG_JOYTR_H
0x4000158	REG_JOYSTAT (JOY BUS Receive Status Register)

Table A–8: Mapa de Memoria del Game Boy Advance

Tomado de [9]

Address	**Type of memory**			**Buswidth**
0x0FFFFFFF			Image Area	
0x0E00FFFF 0x0E000000	Game Pack RAM (0-64 kB)		either Flash Memory or SRAM	8
0x0DFFFFFF Wait State 2 (Register) 0x0C000000	Game Pack ROM 32MB (mirror)	→ 	DACS Flash Memory (1 Mbit) Mask ROM (255 Mbit)	16
0x0BFFFFFF Wait State 1 (Register) 0x0A000000	Game Pack ROM 32MB (mirror)	→ 	DACS Flash Memory (1 Mbit) Mask ROM (255 Mbit)	16
0x09FFFFFF Wait State 0 (Register) 0x08000000	Game Pack ROM 32MB (mirror)	→ 	DACS Flash Memory (1 Mbit) Mask ROM (255 Mbit)	16
			Image Area	
0x070003FF 0x07000000	OAM 1kB			32
			Image Area	
0x06017FFF 0x06000000	VRAM 96kB			16
			Image Area	
0x050003FF 0x05000000	Palette RAM 1kB			16
			Image Area	
0x04000000	IO, Register			32
			Image Area	
0x03007FFF 0x03000000	CPU Internal Window RAM (32 kB)			32
			Image Area	
0x0203FFFF 0x02000000	CPU External Window RAM (256 kB)			16
			Unused Area	
0x00003FFF 0x00000000	System ROM (16kB)		Systems calls are here	32

Anexo B

Herramientas de Desarrollo

B.1 Herramientas del GBA

Como todas las consolas de videojuegos, el GBA tiene una variedad de herramientas destinadas al desarrollo de aplicaciones, fundamentalmente juegos. Aunque Nintendo proporciona un set de estas herramientas a los desarrolladores licensiados, los desarrolladores extraoficiales, o aficionados, cuentan con una extensa gama para desarrollar de manera eficiente. En esta sección se presentarán fundamentalmente aquellas de dominio público y mayor popularidad.

B.1.1 Herramientas GNU

Introducción al gcc

El Compilador GNU (gcc) hace parte de la suite de herramientas GNU, y es el compilador de C usado para compilar el código de la aplicación a ejecutar en la plataforma. Este compilador está extensamente documentado en *Using the GNU Compiler Collection* [28]. Esta introducción se restringe a describir el uso del compilador y de ninguna manera intenta enseñar cómo programar en C.

Invocando el Compilador

Para usar el Compilador GNU para el ARM se debe ejecutar el programa `arm-elf-gcc`. Esto puede realizarse con una línea de comando similar a la siguiente:

`arm-elf-gcc` [*stage-opt*] [*other-opts*] `-mcpu=arm7tdmi` *in-file* `-o` *out-file*

Se debe reemplazar *stage-opt* con una de las opciones de estado que se listarán más adelante, [*other-opts*] con cualquier otra opción necesaria, *in-file* con el nombre del archivo de entrada, y *out-file* con el nombre del archivo de salida. Los nombres del archivo de entrada como el de salida deben tener las extensiones apropiadas. Hay que notar que *no* se deben incluir las barras "[" y "]", estas indican solamente que [*stage-opt*] y [*other-opts*] son opcionales y pueden omitirse.

Aunque lo anterior puede sonar algo complicado, en la práctica se usan solamente "invocaciones estándar" sin modificación. Estas están listadas en el siguiente resúmen con breves explicaciones.

Para convertir código fuente C a un archivo de objeto binario:

`arm-elf-gcc -c -02 -g -mcpu=arm7tdmi` *archivo*`.c -o` *archivo*`.o`

Para convertir múltiples archivos de objetos binarios en un archivo ejecutable (caso general):

`arm-elf-ld` *archivo1*`.o` *archivo2*`.o ... -o` *archivo*`.elf`

Para convertir código fuente C en un archivo ejecutable (para uso solamente con Insight[1]):

`arm-elf-gcc -02 -g -mcpu=arm7tdmi` *archivo*`.c -o` *archivo*`.elf`

Para convertir código fuente C en código fuente en lenguaje assembler:

`arm-elf-gcc -S -fverbose-asm -mcpu=arm7tdmi` *archivo*`.c -o` *archivo*`.s`

Etapas de compilación

El Compilador GNU pasa por 4 diferentes etapas para convertir el programa en C en un archivo ejecutable: El *pre-procesa* el archivo de código, luego lo *compila*, lo *ensambla*, y finalmente lo *vincula* (ver sección 4.1 en la página 32).

Usar el Compilador GNU para crear un ejecutable *no* es lo mismo que usar el Linker GNU, `arm-elf-ld` independientemente. El Compilador GNU automáticamente vincula un número estándar de librerías del sistema en el ejecutable. Estas librerías permiten al programa interactuar con un sistema operativo, para usar funciones de librerías estándar de C, para usar algunas características y operaciones (como la división), etc. Si se desea ver exactamente cuales liberías están siendo vinculadas directamente en el ejecutable, se debe poner la bandera *verbose* `-v` al compilador.

Esto tiene serias implicaciones para sistemas embebidos ya que dichos sistemas usualmente no tienen un sistema operativo. Esto significa que el vincular librerías del sistema es casi siempre inoficioso.

Makefile.

Un makefile es un archivo de texto que indica le indica a `make`[2] como armar un archivo binario a partir de muchos archivos separados. Hay que notar que no siempre se trata de un archivo ejecutable, los makefiles pueden ser usados para automatizar muchas otras tareas, como preprocesar una gran cantidad de archivos gráficos para incluirlos a la ROM del

[1] Insight es la interfaz gráfica del Depurador GNU gdb.

[2] make es una herramienta de generación o automatización de código muy usada en los sistemas operativos tipo Linux/Unix. Por defecto lee las instrucciones para generar el programa u otra acción del archivo makefile.

GBA, etc. Pero el uso más común de un makefile es compilar un ejecutable. Usualmente es solamente llamado "Makefile" en el directorio fuente, sin extensión. Cuando se teclea "`make`" en la consola, buscará por un archivo llamado makefile y empezará a compilar el proyecto.

Suponiendo que se tiene un solo archivo de C, main.c, y se quiere compilar para la ROM del GBA. En la línea de comandos el proceso sería así:

```
gcc -c -mthumb -mthumb-interwork -o myRom.elf main.c
objcopy -O binary myRom.elf myRom.gba
```

La línea 1 toma main.c, lo compila y lo vincula a un archivo binario myRom.elf. La línea 2 toma el archivo elf y lo convierte en el archivo binario final para ejecutar en el GBA. Si el proyecto consistiera en solo un archivo .c y unos cuantos encabezados, eso es todo lo que se necesita. Pero si se quiere realizar una aplicación de mayor complejidad en el GBA, es muy probable que se termine con un directorio (o directorios) de fuente, gráficas y archivos de sonido, y el copiar y repetir la misma línea de comandos una y otra vez para compilar se convertirá en una tarea abrumadora. Es claro que un makefile manejaría esto más facilmente:

```
CFLAGS= -c -O2
THUMBMODEL = -mthumb -mthumb-interwork

all: myRom.gba
myRom.gba: myRom.elf
           objcopy -O binary myRom.elf myRom.gba
myRom.elf: main.o
           $(CC) -o myRom.elf main.o
main.o: main.c
        $(CC) \$(CFLAGS) \$(THUMBMODEL) main.c
```

`CFLAGS = -c -O2`	Define una variable llamada CFLAGS que tiene parámetros comunes que se le pasan al gcc.
`THUMBMODEL = -mthumb -mthumb-interwork`	Define otra variable llamada THUMBODEL la cual es usada para decirle al gcc que modelo cpu/link utilizar.
`all:  myRom.gba`	Aquí se define la primera regla del archivo binario final. "all" informa a make como será el archivo final.
`objcopy -O binary myRom.elf myRom.gba`	Aqui se le informa a make que hacer para armar el archivo, en este caso, myRom.gba. Nótese la alineación de la linea, make requiere espacios tabulares antes del comando para entender que es un comando a ejecutar.
`myRom.elf:  main.o`	Otra regla, Esta informa a make como armar el archivo .elf.
`$(CC) -o myRom.elf main.o`	El comando para armarr el archivo (myRom.elf de main.o).
`main.o:  main.c`	La última regla para construir main.o. Se debe notar que $(CC) es una variable predefinida que make provee para el nombre real del compilador ejecutable en el sistema.
`$(CC) $(CFLAGS) $(THUMBMODEL) main.c`	Este es el comando que construye main.o de main.c. Aparecen las dos variables que se definieron al comienzo del programa. Cuando se usan en la linea de comandos, se deben introducir entre $(). En make, esto se traduce a: `gcc -c -O2 -mthumb-interwork main.c`

Introducción al gdb

El Depurador GNU hace parte de la suite de herramientas del Compilador GNU. Este poderoso depurador permite ejecutar programas bajo condiciones controladas. Esto conlleva a ver exactamente el funcionamiento interno del programa, ayudando a remover cualquier problema (*bug*) que pueda está presente. El Depurador GNU esta extensamente documentado en el *GNU Debugger Manual*. **Invocando el Depurador**

El Depurador GNU puede ejecutarse en dos modos diferentes: con una interfaz gráfica llamada *Insight*, y con la tradicional interfaz de línea de comandos. La interfaz gráfica hace las tareas de depuración mucho más fáciles, mientras la interfaz de línea de comandos permite mayor control sobre tareas complicadas.

La interfaz gráfica Para iniciar el modo de interfaz gráfica se debe introducir la siguiente línea de comandos en el shell de Unix:

```
arm-elf-insigh archivo.elf
```

Naturalmente, se debe reemplazar el `archivo.elf` con el nombre del archivo ejecutable. Nótese el *ampersand* "**&**" al final de la línea de comandos, esto hace que el programa `arm-elf-insight` se ejecute como una tarea de fondo.

Al ejecutar estos comandos la siguiente ventana debe aparecer:

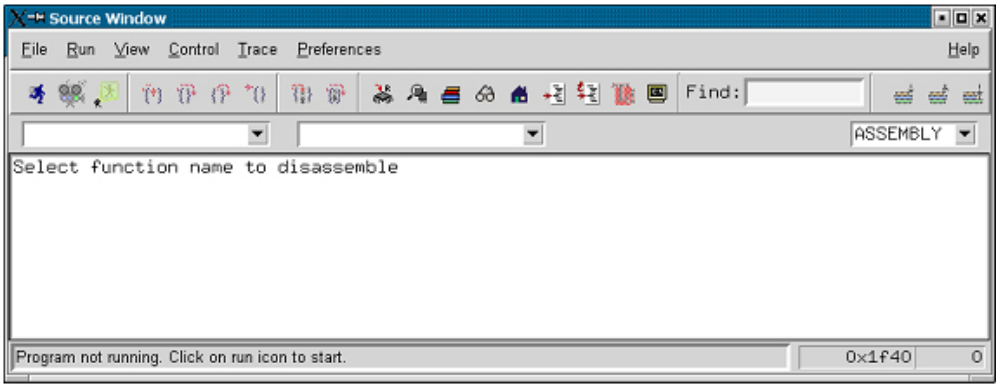


Figure B–1: *Source Window* de la interfaz Insight
Tomado de [29]

Inicialmente es aconsejable ignorar las otras ventanas que tambien aparecerán. Una de las primeras cosas que se debe hacer es abrir la ventana de la consola de depuración[3] , esto proporciona una interfaz completa de la línea de comandos del Depurador GNU, la cual se necesitará para tareas más avanzadas.

Antes de iniciar la tarea de depurar el ejecutable ARM, es necesario descargar el ejecutable a una tarjeta HW o a un simulador. A esto se le denomina *conectando y descargando al target (objetivo).*

Si el ejecutable se va a descargar en una tarjeta HW es necesario configurar los parámetros necesarios, si no, se puede seleccionar el Simulador como *target*. El simulador

[3] Esto se hace seleccionado View >>Console desde el menú principal.

es un programa que pretende ser una tarjeta HW real (Simula el microcontrolador ARM y algunas partes HW que están presentes en la tarjeta real). Habiendo escogido el target y descargado el programa, finalmente se procede a ejecutarlo.

Como en cualquier otro depurador, la interfaz gráfica del gdb ofrece las opciones básicas de depuración como la ubicación de breakpoints, la ejecución paso a paso de instrucciones (tanto ASM como C), la visualización de los registros y visualización de la memoria entre otros.

La interfaz de línea de comandos

El Depurador GNU es un poderoso depurador y el aprender a usarlo puede ser un tanto complicado. A continuación (ver tabla B–1) se muestran los comandos más usados:

Table B–1: Comandos más usados en el gdb

h	Proporciona ayuda con los comandos.
q	Sale del depurador.
r	Corre el programa desde el inicio.
b *ubicación*	Establece un breakpoint en **ubicación**, quien puede ser una etiqueta, un numero de línea en el **c**ódigo fuente o *dirección.
i b	Muestra la información de los breakpoints del programa.
d ***num***	Elimina el breakpoint. *num*
en ***num***	Habilita el breakpoint. *num*
dis ***num***	Deshabilita el breakpoint. *num*
c	Continúa ejecutando el programa hasta que encuentra un breakpoint o el final.
u *ubicación*	Ejecuta hasta que el programa alcanza *ubicación* (*ubicación* tiene la misma sintáxis que en el comando **b**).
s	Paso a través de la siguiente línea de código.
n	Paso a través de la siguiente línea, tratando a los llamados de funcion como si fueran una línea sencilla.
si	Paso a través de la siguiente instrucción en lenguaje asm.
ni	Igual que **si**, pero no sigue a través de funciones o saltos hacia atrás.
i r	Muestra todos los registros del procesador ARM.
x/*nfu dirección*	Examina la memoria en la *dirección*. Los formatos mas úsados *f* son **x** para hexadecimal, **d** para decimal con signo, **u** para decimal sin signo, **t** para binario y **i** para instrucciones ARM. Los tamaños de de unidad *u* más úsados son **b** para bytes, **h** para medias palabras y **w** para palabras.
p ***expresión***	Imprime el valor de *expresión* (que puede ser cualquier expresión complicada de C que se desee).
p/*f expresión*	Imprime el valor de *expresión* usando el formato *f* (ver el comando **x** para una lista).

B.1.2 DevKit Advance

El DevKit Advance es un GBA SDK (System Development Kit) basado en GCC que viene compilado con el *Socrates Gameboy Advance Development Enviroment* (SGADE), una librería de código génerico para la plataforma GBA, bajo licencia open source. En pocas palabras el DevKit Advance es una toolchain GNU con unos cuantos *parches* para hacerla funcionar particularmente en el GBA. (ver sección 4.3.1 en la página 35)

Instalación

El DevKit Advance puede ser instalado utilizado en Linux, Windows y Mac OS X. Charmedlabs, La compañia creadora de la tarjeta de expansión en la cual se basa este

proyecto, provee en su sitio web los instaladores[4] para Windowns y Linux. En el entorno Windows básicamente se necesitan el cygwin[5] , y el Devkit que es denominado *Xport DevKit*. Este Xport DevKit contiene una versión del ya mencionado DevKit Advance, además de una libreria de C para uso en sistemas embebidos (*newlib*), una colección de herramientas de programación para manipulación de código objeto (*binutils* -ver página 33) y el Insight. En Linux, el archivo descargado es un precompilado de estas mismas herramientas siendo las diferencias apreciadas solo de tipo interfaz.

B.1.3 Otros SDK

Existen varios entornos de desarrollo para el GBA como el el DevKitadv, la mayoría de ellos son de carácter gratuito, a continuación una breve introducción sobre los más populares.

- **DevKitARM o DevKitPRO**:

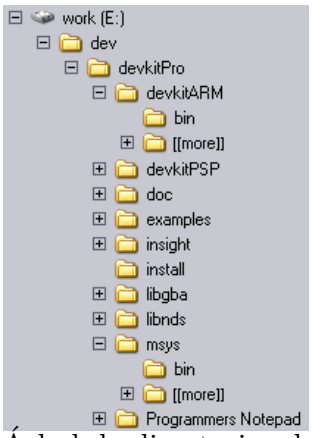


Figure B–2: Árbol de directorios del DevKitPro

  Este kit se ha convertido en el toolchain estándar y recomendado. Es actualizado regularmente, rápido de compilar, se puede instalar y actualizar él mismo y viene con códigos de ejemplo. Su instalación se puede realizar de dos maneras, manualmente o usando el instalador. Usando el instalador requiere solamente descargar el *devkitPRO installer*, quien se encargará por su cuenta de la instalación y descargas de otros paquetes requeridos. DevKitPro contiene toolchains además para PSP e incluso GameCube (ver fig B–2).

- **HAM**: HAM es un paquete de distribución que incluye todas las herramientas (incluyendo compilador C/C++) necesarias para escribir, compilar, vincular, y ejecutar programas del GBA. El término distribución viene del hecho de que HAM usa el compilador open source GCC (quien tiene licencia GNU), así como el ensamblador de ARM7 que fue distribuido al dominio publico por el fabricante del chip, ARM. La distribución HAM incluye las siguientes herramientas:

[4] www.charmedlabs.com >>Downloads >>Xport >>Software.

[5] Cygwin es una colección de herramientas gratuitas de software que permite a varias versiones de Microsoft Windows actuar como un sistema UNIX.

- HAM GBA SDK
- Librerias de código HAM
- Entorno de desarrollo Visual HAM
- Editor de Mapas PE
- Emulador VisualBoyAdvance

Uno de los aspectos que destacan del HAM es que esta distribución es fácil de instalar y usar [8].

B.2 Emuladores

El siguiente paso en el desarrollo de software es simularlo para comprobar el funcionamiento, dada la condición en la que se está programando para un *target* externo (GBA) solo existen dos opciones, o probarlo en la consola directamente o emularlo si es posible. La emulación constituye una herramienta de gran ayuda en los casos en los que no es necesario probar una interfaz realizada con el GPIO. Mediante esta herramienta es posible depurar visual y rapidamente errores cometidos en aspectos gráficos, o de interfaz básicos como teclado o sonido. Algunos de los emuladores más populares para el GBA son el Visual Boy Advance(ver sección), BoycottAdvance, NO$GBA, BatGBA, etc.

B.3 Xport DevKit

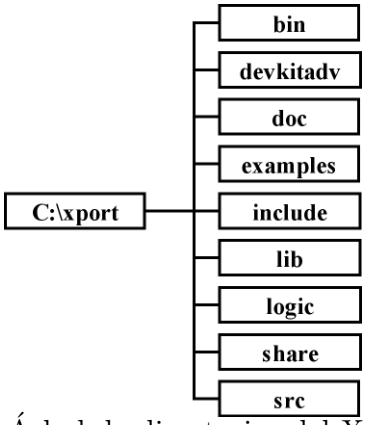


Figure B–3: Árbol de directorios del Xport DevKit

El Xport DevKit mencionado anteriormente se convierte en la herramienta fundamental de desarrollo para la plataforma construida. Contiene el compilador, depurador, programador y ejemplos para empezar a entender el funcionamiento del sistema.

Trás su instalación, crea un árbol de directorios (ver fig B–3) que será descrito de manera general a continuación:

- **bin**
  Aquí reside el xpcomm, programa que realiza la interfaz entre el PC y la tarjeta de expansión por medio del puerto paralelo, además se encuentra una librería (WinIO)que permite al xpcomm manejar sin problema los puertos E/S del sistema.

- **devkitadv**
  En este directorio se encuentra instalado el DevKit Advance ya mencionado anteriormente, incluye el compilador para C/C++, depurador, librerías, etc.

- **doc**
  En este directorio se encuentran documentos relacionados con la tarjeta de expansión Xport. Incluye un manual de instrucciones con la tabla de comandos del xpcomm así como la descripción básica del funcionamiento del dispositvo y el tutorial de configuración de la FPGA mediante el paquete ISE de Xilinx.

- **examples**
  En este directorio residen más de 10 ejemplos que vienen listos para cargar a la plataforma. La estructura de cada ejemplo consta de una carpeta con su nombre, en donde se encuentra el código para el procesador escrito en C ó C++, dentro de esta misma carpeta se encuentra otra carpeta denominada *logic* en donde se encuentra el código en verilog para la FPGA. Esta carpeta *logic* solo existe en aquellos ejemplos que lo requieren.

- **include**
  En este directorio residen archivos "cabeceras" para incluir en otros programas. Algunas de las cabeceras importantes que trae el Xport Devkit son la `gba.h` que contiene la información de todos los registros del GBA y la `textdisp.h` que es esencial para la impresión de textos en la LCD.

- **logic**
  En este directorio se almacena las configuraciones lógicas que charmedlabs ha desarrollado. Algunas de ellas son de carácter opcional, otras obligatorias a la hora de compilar. Se divide en 2 subdirectorios: lib y src.

  - **lib**
    Aquí se almacenan todas las configuraciones lógicas para la FPGA que no quieren o no se pueden mostrar en forma de código, los archivos que se encuentran en este directorio son principalmente archivos que tienen restricciones, como el código de demultiplexación del bus (`decodebus.ngc`), y otros que hacen parte fundamental del funcionamiento de la plataforma como `cport.ngc`, o simplemente configuraciones que son opcionales como `sdramcont.ngc` y `btuart.ngc`. Es importante mencionar que estos archivos son incluidos como librerías a un programa "principal" como se verá mas adelante.

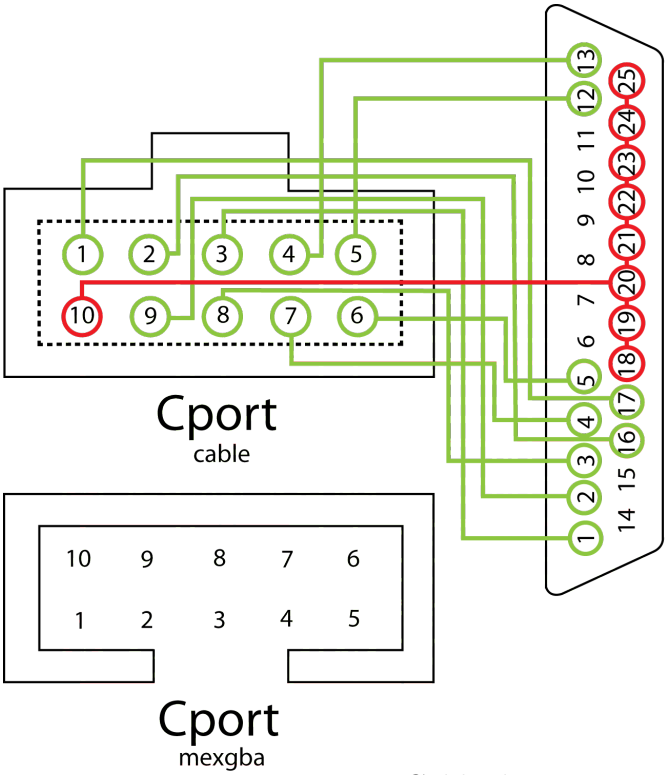
  - **src**
    Aquí se encuentran todos los demás archivos de lógica que no tienen restricciones. Dentro de los archivos más importantes se encuentra `primary.v`, quien es la columna vertebral del código básico y quien contiene al decodebus. Es importante notar que algunos de ellos están incompletos (`pwm.v`) y otros están vinculados a las configuraciones encontradas en el directorio lib (`decodebus.v sdramcont.v`). Los archivos que se encuentran en este directorio son archivos de código en verilog.

- **src**
  Aquí se encuentran los códigos fuentes del xpcomm, waveutil, librerias del gba y crt0. El waveutil es una utilidad que permite convertir sonidos wav a un formato que reconozca el gba, las librerías del gba son básicamente las mismas que se encuentran

en el directorio lib y el crt0 es un conjunto de rutinas de inicio de ejecución que es requerido para compilar usando el gcc y otras herramientas GNU.

B.3.1 Cable Cport

Una Herramienta de definitiva importancia es el cable que conecta el puerto paralelo al puerto cport de la tarjeta de expansión. Este cable representa una herramienta física para la configuración. Cabe notar que el cable de la tarjeta de expansión realizada (MEXGBA) difiere del cable de la tarjeta de expansión XPORT.



CPORT	DB25
1	17
2	16
3	1
4	13
5	12
6	5
7	4
8	3
9	2
10	18-25

Figure B–4: Cable de Programación CPORT

Anexo C

Manual de usuario

MEXGBA v1.0

C.1 Incio rápido

Este Manual le guiará a través de los pasos necesarios para comenzar a utilizar la plataforma.
Para comenzar a utilizar la plataforma es necesario reunir todas las herramientas:

- Un computador corriendo Windows 9x/Me/NT/2000/XP con puerto paralelo, 250MB disponibles de disco duro y privilegios de administrador
- CD de instalación de las herramientas (incluye Xport DevKit y Cygwin entre otros)
- MEXGBA v1.0
- Cable CPORT-Paralelo
- Game Boy Advance (GBA) o GBA SP

1. Se debe insertar la tarjeta MEXGBA en el puerto de juegos GamePak del GBA. Debe ponerse especial atención en la orientación de la tarjeta. los pines de programación y GPIO (User Ports) deben quedar orientados hacia la parte trasera del GBA.

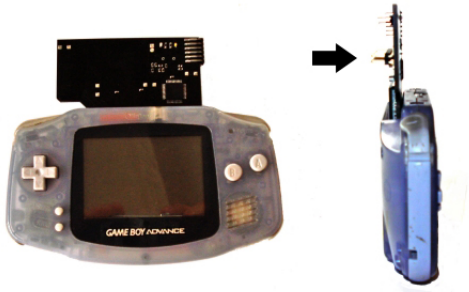


Figure C–1: MEXGBA + GBA

2. Para verificar el funcionamiento del MEXGBA se debe encender el GBA. Se debe observar el texto “MEXGBA en funcionamiento” y el parpadeo de los LEDs azul y rojo despues de que aparezca el logo de nintendo.

Figure C–2: Sistema en funcionamiento

3. Inserte el CD con el Software de instalación en la unidad de CD-ROM de su PC[1] y ejecute los instaladores de Cygwin, Xport DevKit y eCos. Con esto se instalarán las herramientas necesarias para compilación además de ejemplos.

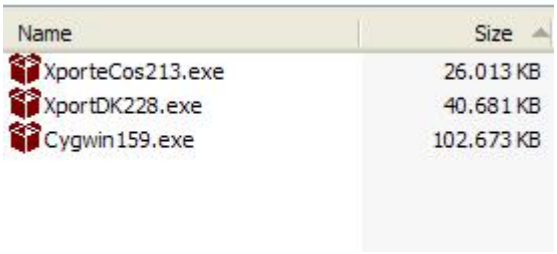


Figure C–3: Instaladores del DevkitXport y Cygwin

4. Conecte el conector paralelo (DB25) a el puerto paralelo de su PC.

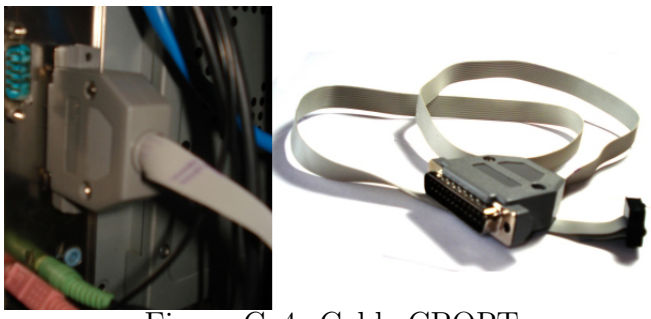


Figure C–4: Cable CPORT

5. Con el GBA apagado, conecte el otro terminal del cable CPORT al conector CPORT de 10 pines de la tarjeta MEXGBA. Debe ponerse especial atención en la orientación del conector como se muestra en la figura.

[1] O descarguelos desde http://www.charmedlabs.com

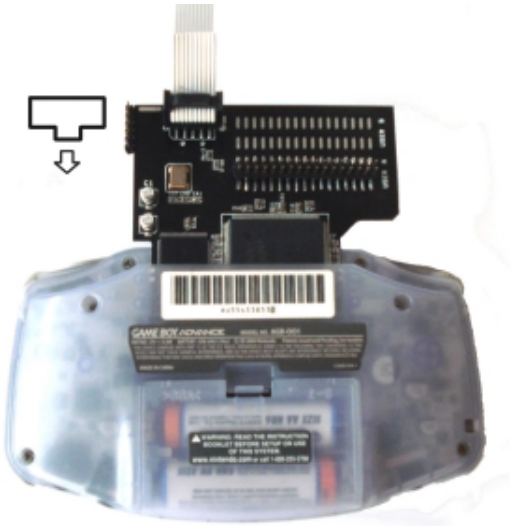


Figure C–5: MEXGBA + Cable CPORT

6. Abra el "Xport Shell". Esto lo puede hacer desde el escritorio o desde el menú de inicio (Inicio>Programas>Xport). Cambie al directorio helloworld_c entrando el comando "cd examples/helloworld_c"

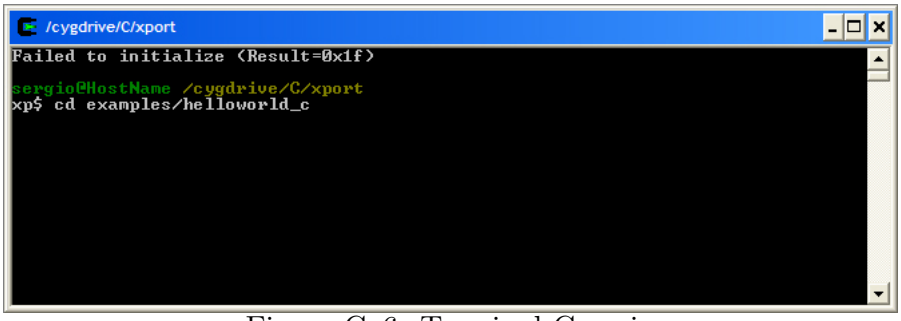


Figure C–6: Terminal Cygwin

7. Estando en ese directorio, ejecute "make upload". Esto configurará la lógica y la flash del MEXGBA, pero antes de hacer esto, pedirá que se encienda el GBA. Este procedimiento de encendido del GBA es necesario antes de cualquier operación de programación.

8. Después de haber realizado la programación es necesario Apagar >Encender el GBA de nuevo para correr el programa. Después de la secuencia del logo de Nintendo, se debe ver el texto "Hello world!".

9. Sientase en la libertar de modificar el código main.c a su gusto y recompilelo / descarguelo corriendo "make upload" de nuevo.

Si todo funcionó tal cual fué descrito, su MEXGBA está probado para su uso. Si ud experimentó problemas, refierase a la sección de *Solución de problemas* (C.6, pág 73) al final de este manual.

C.2 Descripción del MEXGBA v1.0

La Figura C–7 muestra el Sistema MEXGBA v1.0 completo incluyendo al MEXGBA y al GBA.

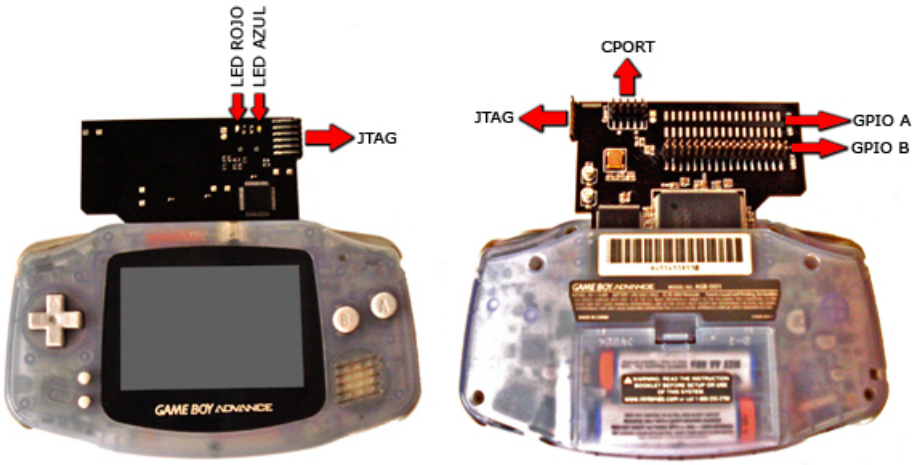


Figure C–7: Sistema MEXGBA v1.0

Led Rojo Led programable por el usuario
Led Azul Led programable por el usuario
JTAG Usado para la programación del CPLD (No está conectado a la FPGA)
Cport Usado para la programación y comunicación con el MEXGBA. Se conecta al puerto paralelo del PC usando el cable de programación CPORT.
Conectores USER (GPIO) Usados para hacer interfaz con sistemas externos. Refierase a la sección *Señales del conector GPIO* para más información

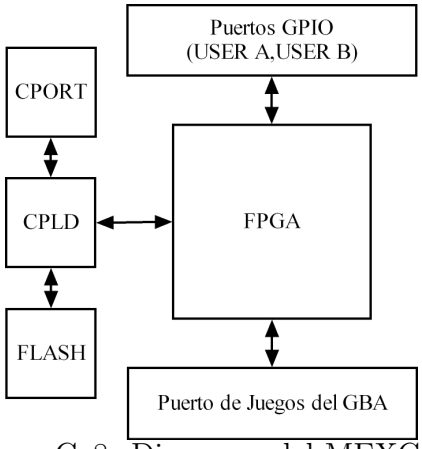


Figure C–8: Diagrama del MEXGBA

La figura C–8 muestra un diagrama de bloques generalizado para la tarjeta de expansión MEXGBA v1.0, que consiste básicamente en una FPGA, una memoria flash, y componentes de soporte.

Como se muestra en el diagrama, la FPGA es el componente central, esta topología

provee la mayor flexibilidad debido a que la FPGA es totalmente programable. El dispositivo de memoria flash almacena el codigo que es ejecutado por el GBA, y debido a que la FPGA utiliza celdas RAM volátiles para almacenar su configuración lógica, la memoria flash tambien es usad para almacenar la configuración lógica de la FPGA.
El CPLD asiste en el proceso de configuración en el encendido que es requierido para programar la FPGA cuando el sistema es energizado. En la FPGA se almacena la lógica que permite la interfaz entre el código almacenado en la flash y el procesador del GBA, además tiene espacio suficiente para albergar configuraciones lógicas de periféricos externos a ser conectados al MEXGBA.
Para aplicaciones que requieren I/O físicas, existen 64 señales disponibles a través de los conectores de expansión USER A y USER B. El usar la FPGA para estas aplicaciones que requieren interfaces externas puede requerir la escritura o modificación de las configuraciones lógicas. Es recomendado que estas configuraciones sean escritas en HDL (Hardware Description Language) como Verilog o VHDL (ver *Configuración de la FPGA*).

C.3 Xpcomm

El Xpcomm es la aplicación utilizada pra programar las configuraciones tanto de la flash como de la FPGA, así como para comunicación con el GBA a través del CPORT.
Para programar la configuración de la FPGA, Xpcomm acepta formatos .bit generados por Software para Síntesis cmo Xilinx ISE. Para programar la flash, Xpcomm acepta formatos .bin y .srec. Estos archivos son usualmente generados por compiladores de C/C++ como el gcc.
Cuando se ejecuta el Xpcomm este detectará el formato del archivo y de acuerdo a esto programará el MEXGBA correctamente.
Además Xpcomm puede ser guiado para ejecutar operaciones especificas con comandos extra, estos comandos se listan en la tabla C–2.
Las operaciones se ejecutan en el orden especificado. Por ejemplo,

```
xpcomm -vc redgreen.bit -rf out.bin 0x200000 -pf redgreen.bin
```

verificará la configuración de la fpga con redgreen.bit, luego almacenará las dos primeras megapalabras (4 megabytes) de la flash a out.bin, y finalmente programará la flash con los contenidos de redgreen.bin.
Xpcomm tambien soporta las propiedades detalladas en la tabla C–3.
Las propiedades pueden aparecer en cualquier orden en la línea de comando y aplicarse a diferentes operaciones especificadas. Por ejemplo,

```
xpcomm gpio.bit -portaddr 0x3bc gpio.bin -debug 2
```

Comunicará el puerto paralelo a través de la ubicación 0x3bc y imprimirá mensajes de depuración de nivel 2 para todas las operaciones.

C.4 Especificaciones Eléctricas

Table C–1: Características DC

Descripción	**Valor**
Voltaje máximo de alimentación	3.3V
Voltaje mínimo de alimentación	3.0V
Corriente máxima de alimentación	200mA[3]
Corriente nominal de alimentación	100mA[4]
Fuente de corriente por salida [1]	24mA
Sumidero de corriente por salida [1]	-24mA
Corriente de escape por entrada [2]	+/- 10 uA
Nivel alto de voltaje de salida [1]	3.3V max
Nivel bajo de voltaje de salida [1]	0.0V min
Nivel alto de voltaje de entrada	1.7V min, 5.5V max
Nivel bajo de voltaje de entrada	0.0V min, 1.0V max
Capacitancia de entrada por entrada [2]	20pF

[1] Para pines de I/O programables configurados como salida.

[2] Para pines de I/O programables configurados como entrada.

[3] Corriente máxima total disponible para el sistema externo asumendo que el GBA usa dos baterias AA como alimentación

[4] Se recomienda que la corriente total requerida por la carga del sistema externo no exceda 100mA.

Table C–2: Operaciones del Xpcomm

`-pf` <*archivo de programa*>	Programa la flash con el archivo de programa (binario o S-record). Nota, cuando se realiza el procedimiento de programación de la flash, Xpcomm realiza una suma de comprobación para asegurar la integridad de los datos
`-vf` <*archivo de programa*>	Verific la flash con el archivo de programa especificado (binario o S-record).
`-pvf` <*archivo de programa*>	Programa y verifica la flash con archivo de programa especificado (binario o S-record).
`-rf` <*dump file*><*longitud*>	Lee y descarga los contenidos de la flash (*longitud* numero de palabras) al archivo especificado *dump file*.
`-pc` <*archivo bitstream/directorio*>	Programa la configuración de la FPGA con el archivo bitstream especificado. Si un directorio es especificado, Xpcomm automáticamente escogerá el bitstream en el directorio que es compatible.
`-vc` <*archivo bitstream/directorio*>	Verifica la configuración de la FPGA con el archivo bitstream especificado. Si un directorio es especificado, Xpcomm automáticamente escogerá el bitstream en el directorio que es compatible
`-c` <*archivo bitstream/directorio*>	Programa la FPGA con el archivo bitstream especificado usando el modo esclavo externo. Esto no modifica los contenidos de la configuración no-volátil de la FPGA almacenados en flash. La configuración se perderá al desenergizar el sistema.
`-i`	Recupera y muestra información de la tarjeta de expansión, actualiza la configuración de la FPGA para la programación de la flash si es necesario
`-u`	Forza una actualización de la configuración de la FPGA para la programación de la flash.
`-reset`	Resetea el GBA y ejecuta el programa contenido en flash. Para esto el MEXGBA debe tener instalado el plug y el pulsador para la señal de reset, además el GBA debe tener la señal automática de reset instalada.
`-startup`	Resetea el estado del puerto paralelo para que el la configuración lógica del slot 0 sea usada.
`-console`	Corre una consola tty a través del CPORT.
`-rpc`	Corre el Servidor de Procedimiento de llamado remoto (Remote Procedure Call) a través del CPORT.
`-execute`	Igual que reset
`-pause` <*milisegundos*>	Pausa entre las operaciones por el numero especificado de milisegundos.
`-loop` <*iteraciones*>	Especifica el numero de veces que se debe repetir una secuencia de comandos.
`-time`	Imprime el tiempo transcurrido despues de que todas las operaciones fueron completadas.
`version`	Imprime la versión del Xpcomm

Table C–3: Propiedades del Xpcomm

`-resetauto`	Especifica al Xpcomm que la señal de reset automático está instalada
`-portaddr` <*dirección*>	(Aplica solamente a Windows NT, 2000 y XP). Especifica la dirección del puerto paralelo. La dirección por defecto es 0x378.
`-portnum` <*valor*>	(Aplica solamente a Windows NT, 2000 y XP) Especifica el numero de dispositivo del puerto paralelo (Numero LPT). Por defecto, LPT1 es usado. Para especificar LPT2, por ejemplo, debería escribirse -portnum 2 en la línea de comandos.
`-debug`<*debug level*>	Especifica el número relativo de mensajes de depuración que son enviados a la consola. Hay tres modos soportados: (0) solo mensajes críticos, (1) (por defecto) mensajes considerados importantes al usuario normal, y (2) información resumida que puede ser útil para depuración.
`delay`<*delay value*>	Especifica el retardo que debe ser usado cuando se escriba o lea el puerto paralelo. Es recomendado que no exceda 500.
`readdelay`<*delay value*>	Similar a delay, pero aplica solo a operaciones de lectura desde el puerto paralelo.
`writedelay`<*delay value*>	Similar a delay, pero aplica solo a operaciones de escritura al puerto paralelo.

C.5 Señal automática de reset

Como se ha mencionado antes esta señal permite efectuar reset del sistema desde la consola, ahorrando así tiempo y desconcentraciones.
El MEXGBA trae acondicionado el espacio para instalar una señal automática de reset (ver fig C–9)

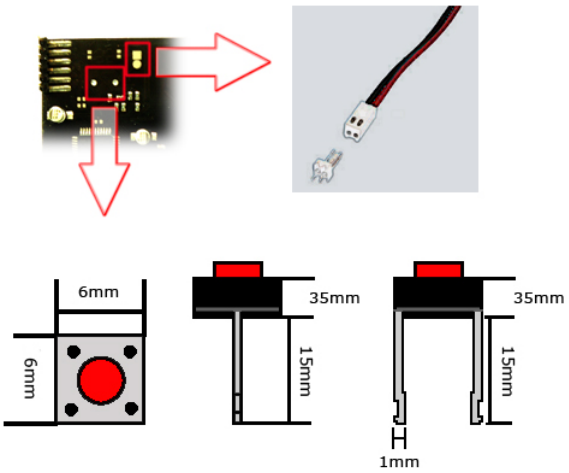


Figure C–9: Señal automática de reset

La instalación de esta señal requiere modificaciones del GBA. Las instrucciones para tal propósito se encuentran en la página web del desarrollador Jeff Frohwein:

http://www.devrs.com/gba/files/gbadevfaqs.php#ResetButton

C.6 Solución de Problemas

1. **El GBA inicia sin el logo de Nintendo y el programa no corre**
   Esto sucede normalmente debido a un estado incorrecto del pureto paralelo, causando que sea el slot 1 y no el slot 0 de la memoria flash quien sea cargado como configuración lógica de la FPGA en el ciclo de arranque. Ejecutando

   ```
   xpcomm -startup
   ```

   se reteará el puerto paralelo de manera que el slot 0 sea seleccionado.

2. **Cuando se ejecuta el Xpcomm, sale el error "Failed to initialize" y se cierra**
   Este error es causado normalmente por que otra copia del Xpcomm está ejecutandose. Es necesario para corregirlo revisar la lista de procesos y terminar todos los procesos del Xpcomm.

   Este error tambien puede ser causado por otro programa o dispositivo que está usando el puerto paralelo

3. **Fallan las operaciones de programación o las comunicaciones con el CPORT**
   Los problemas de comunicación o programación pueden ser causados por una configuración incompatible del puerto paralelo en su PC.

   El Xpcomm no es compatible con el modo EEP (enhanced parallel port) del puerto paralelo. Puede ser necesario reiniciar su PC y cambiar el modo del puerto paralelo a Bidireccional o ECP (esto se hace desde la pantalla de configuración de la BIOS). El modo bidireccional tambien puede ser llamado como PS2.

   Otro error que puede ser común se refiere al aspecto de programación cuando se intenta enviar un archivo de configuración para la flash y al final de la secuencia se interrumpe con el mensaje "Failed". Si se revisa la información de la flash que está contenida en el CPLD (esto se hace mediante el comando xpcomm -i), se debe notar que la información de la flash está corrupta. La solución consiste en ejecutar

   ```
   xpcomm -u
   ```

   que actualizará la configuración de la flash que concierne a la programación de la FPGA.

C.7 Señales de los conectores GPIO y de la FPGA

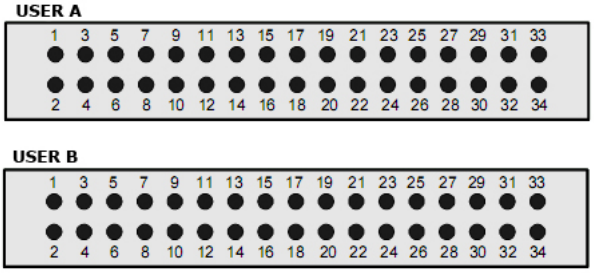


Figure C–10: Conectores USER A y USER B (vista superior)

Table C–4: Pinout de los GPIOs USER A y B

GPIO USER A		
Pin	Señal	Descripción
1	GND	Tierra (0V)
2	CLKINA	Reloj o entrada genérica
3	PA16	I/O programable
4	PA15	I/O programable
5	PA14	I/O programable
6	PA13	I/O programable
7	PA12	I/O programable
8	PA11	I/O programable
9	PA10	I/O programable
10	PA9	I/O programable
11	PA8	I/O programable
12	PA7	I/O programable
13	PA6	I/O programable
14	PA5	I/O programable
15	PA4	I/O programable
16	PA3	I/O programable
17	PA2	I/O programable
18	PA1	I/O programable
19	PA0	I/O programable
20	PA18	I/O programable
21	PA19	I/O programable
22	PA20	I/O programable
23	PA21	I/O programable
24	PA22	I/O programable
25	PA23	I/O programable
26	PA24	I/O programable
27	PA25	I/O programable
28	PA26	I/O programable
29	PA27	I/O programable
30	PA28	I/O programable
31	PA29	I/O programable
32	PA30	I/O programable
33	PA17	I/O programable
34	Vcc	3.3V

GPIO USER B		
Pin	Señal	Descripción
1	GND	Tierra (0V)
2	PB0	I/O programable
3	PB1	I/O programable
4	PB2	I/O programable
5	PB3	I/O programable
6	PB4	I/O programable
7	PB5	I/O programable
8	PB6	I/O programable
9	PB7	I/O programable
10	PB8	I/O programable
11	PB9	I/O programable
12	PB10	I/O programable
13	PB11	I/O programable
14	PB12	I/O programable
15	PB13	I/O programable
16	PB14	I/O programable
17	PB15	I/O programable
18	PB16	I/O programable
19	PB17	I/O programable
20	PB18	I/O programable
21	PB19	I/O programable
22	PB20	I/O programable
23	PB21	I/O programable
24	PB22	I/O programable
25	PB23	I/O programable
26	PB24	I/O programable
27	PB25	I/O programable
28	PB26	I/O programable
29	PB27	I/O programable
30	PB28	I/O programable
31	PB29	I/O programable
32	PB30	I/O programable
33	CLKINB	Reloj o entrada genérica
34	Vcc	3.3V

Table C–5: Señales de la FPGA

Pin	**Señal**	**Descripción**
165	CartData0	Bus datos/dirección GBA
166	CartData1	Bus datos/dirección GBA
167	CartData2	Bus datos/dirección GBA
168	CartData3	Bus datos/dirección GBA
172	CartData4	Bus datos/dirección GBA
173	CartData5	Bus datos/dirección GBA
174	CartData6	Bus datos/dirección GBA
175	CartData7	Bus datos/dirección GBA
176	CartData8	Bus datos/dirección GBA
178	CartData9	Bus datos/dirección GBA
179	CartData10	Bus datos/dirección GBA
180	CartData11	Bus datos/dirección GBA
181	CartData12	Bus datos/dirección GBA
187	CartData13	Bus datos/dirección GBA
188	CartData14	Bus datos/dirección GBA
189	CartData15	Bus datos/dirección GBA
191	CartAddr0	Bus dirección GBA
192	CartAddr1	Bus dirección GBA
193	CartAddr2	Bus dirección GBA
194	CartAddr3	Bus dirección GBA
195	CartAddr4	Bus dirección GBA
199	CartAddr5	Bus dirección GBA
200	CartAddr6	Bus dirección GBA
201	CartAddr7	Bus dirección GBA
164	CartCs	Chip select del cartucho
163	CartRd	Lectura
162	CartWr	Escritura
203	CartIReq	Petición de interrupción
182	CartClk	Salida de reloj
150	FAddr0	Dirección Flash
149	Faddr1	Dirección Flash
148	FAddr2	Dirección Flash
147	FAddr3	Dirección Flash
141	FAddr4	Dirección Flash
140	FAddr5	Dirección Flash
139	FAddr6	Dirección Flash
138	FAddr7	Dirección Flash
136	FAddr8	Dirección Flash
134	FAddr9	Dirección Flash
133	FAddr10	Dirección Flash
129	FAddr11	Dirección Flash
127	FAddr12	Dirección Flash
125	FAddr13	Dirección Flash
123	FAddr14	Dirección Flash
122	FAddr15	Dirección Flash
121	FAddr16	Dirección Flash
120	FAddr17	Dirección Flash
114	FAddr18	Dirección Flash
113	FAddr19	Dirección Flash
112	FAddr20	Dirección Flash
45, 96	FAddr21	Dirección Flash
66	FAddr22	Dirección Flash
153	FData0	Datos Flash
146	FData1	Datos Flash
142	FData2	Datos Flash
135	FData3	Datos Flash
126	FData4	Datos Flash
119	FData5	Datos Flash
115	FData6	Datos Flash
108	FData7	Datos Flash
152	Foe	Habilitador de salida de la flash
132	Fce	Habilitador flash
151	Fwe	Escritura Flash
185	Clk	Reloj de 50MHz
204	LED Azul	control LED (0V=encendido)
205	LED Rojo	control LED (0V=encendido)
20	PA0	I/O programable
21	PA1	I/O programable
22	PA2	I/O programable
23	PA3	I/O programable
24	PA4	I/O programable
27	PA5	I/O programable
29	PA6	I/O programable
30	PA7	I/O programable
31	PA8	I/O programable
33	PA9	I/O programable
34	PA10	I/O programable
35	PA11	I/O programable
36	PA12	I/O programable
37	PA13	I/O programable
41	PA14	I/O programable
42	PA15	I/O programable
43	PA16	I/O programable
206	PA17	I/O programable
18	PA18	I/O programable
17	PA19	I/O programable
16	PA20	I/O programable
15	PA21	I/O programable
14	PA22	I/O programable
10	PA23	I/O programable
9	PA24	I/O programable
8	PA25	I/O programable
7	PA26	I/O programable
6	PA27	I/O programable
5	PA28	I/O programable
4	PA29	I/O programable
3	PA30	I/O programable
77	CLKINA	Reloj o entrada genérica
97	PB0	I/O programable
96	PB1	I/O programable
95	PB2	I/O programable
94	PB3	I/O programable
90	PB4	I/O programable

89	PB5	I/O programable
88	PB6	I/O programable
87	PB7	I/O programable
86	PB8	I/O programable
84	PB9	I/O programable
83	PB10	I/O programable
82	PB11	I/O programable
81	PB12	I/O programable
75	PB13	I/O programable
74	PB14	I/O programable
73	PB15	I/O programable
71	PB16	I/O programable
70	PB17	I/O programable
69	PB18	I/O programable
68	PB19	I/O programable
67	PB20	I/O programable
63	PB21	I/O programable
62	PB22	I/O programable
61	PB23	I/O programable
60	PB24	I/O programable
59	PB25	I/O programable
58	PB26	I/O programable
57	PB27	I/O programable
49	PB28	I/O programable
48	PB29	I/O programable
47	PB30	I/O programable
	NC	
80	CLKINB	Reloj o entrada genérica, botón reset
162	NC	
161	NC	
107	NC	
154	NC	
46	NC	
44	NC	
45	NC	
110	CPDir	Cport data direction
111	CPStrobe	Cport data strobe
99	CPReset	Cport reset
98	Cpready	Cport data ready
109	CPData0	Cport data
102	CPData1	Cport data
101	CPData2	Cport data
100	CPData3	Cport data

C.8 Configuración de la FPGA

El desarrollo de la configuración lógica de la FPGA se realiza utilizando el software ISE proporcionado por Xilinx. Este WebPACK se puede descargar desde la página de Xilinx de manera gratuita:

http://www.xilinx.com/xlnx/xil_entry2.jsp?sMode=login&group=swreg4&SWR_PRODUCT_ID=WP91

Desde esta página se debe registrar como usuario para despues proceder a descargar. Es aconsejable escojer la opción "Xilinx WebInstall", la cual instala completamente el software y además aplica los últimos parches automáticamente. Durante la instalación se debe escoger los módulos a ser instalados, de los cuales solo es necesario el módulo "FPGA Design Envionment". Se recomienda no instalar los drivers MultiLINX ya que esta comprobado que pueden afectar severamente el funcionamiento de los puertos USB de su PC. Como todo este manual, las configuraciones indicadas se basan en el paquete de herramientas provistas por Charmedlabs, fabricante de la tarjeta de expansión Xport.

C.8.1 La configuración "RedGreen1"

Para propósitos ilustrativos, Charmedlabs creó una configuración simple llamada "RedGreen1" que permite a un programa corriendo desde el GBA controlar los LEDs que se encuentran en la tarjeta de expansión. Cabe aclarar que estos LEDs que en la tarjeta de expansión Xport son rojo y verde, son azul y rojo en el MEXBGA.
Después de creada la configuración se escribirá un programa de prueba en C++ que ilumine los LEDs cuando se presionen los botones A y B del GBA. Este ejemplo se encuentra en el directorio examples/redgreen1 dentro de la carpeta xport. Pero para propósitos ilustrativos se creará un proyecto nuevo desde cero y se copiará código del ejemplo cuando sea necesario.

Se necesitará:

- PC con software Xport instalado (XportDevkit, Cygwin, etc)
- Software Xilinx WebPack
- MEXGBA 1.0, GBA, Cable CPORT.

Creando y sintetizando RedGreen

Inicie el WebPACK con el "Project Navigator" (Inicio→Programas→Xilinx ISE). El nuevo proyecto se crea seleccionando "New Project" desde el menu"File". La figura C–11 muestra como debe lucir la ventana de "New Project". En el cuadro de "Project Name" introduzca "myredgreen1", por ejemplo. En el cuadro de "Project Location", es aconsejable introducir la dirección de la carpeta examples (c: xport examples myredgreen1 logic). Nótese que se crea otra carpeta dentro del ejemplo para separar el código C++ del codigo HDL asociado con la configuración lógica. Las opciones que deben ser seleccionadas dentro de la configuración lógica son las que siguen:

- Device Family: Spartan2
- Device XC2S50

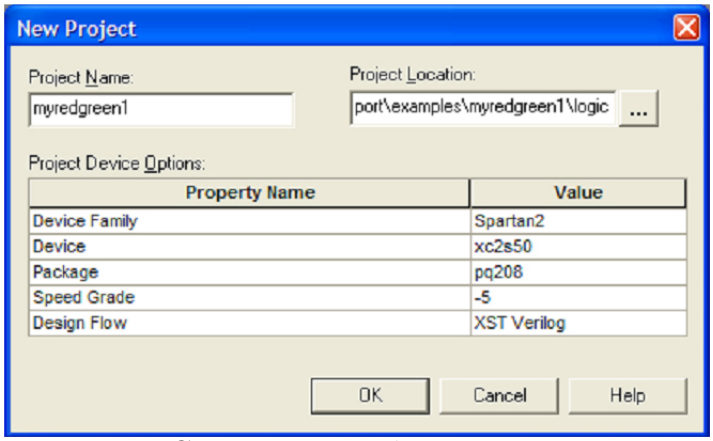


Figure C–11: Ventana de Nuevo Proyecto

- Speed Grade: -5
- Design Flow: XST Verilog

Copie el código fuente del ejemplo redgreen1 existente (c: xport examples redgreen1 logic redgreen1.v) a la carpeta creada para este ejemplo. Ahora agregue ese código al nuevo proyecto seleccionando "Add Source" desde el menu Source. Seleccione el archivo redgreen1.v que acaba de copiar y luego clic en "Open".

Después de que el archivo de código ha sido añadido, el panel de "Sources in project" debe lucir como en la figura C–12. Desde aquí se peude observar el código haciendo doble clic

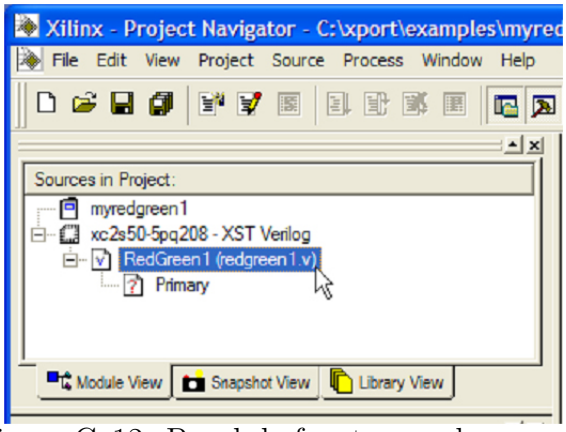


Figure C–12: Panel de fuentes en el proyecto

en redgreen1.v (resaltado en la fig C–12). Si nunca ha visto un código en Verilog, notará que tiene una sintaxis muy parecida a la de el lenguaje C. Podrá darse cuenta tambien debido al tamaño del archivo que lo que sea que redgreen1.v haga, no es muy complejo.

La primera parte del archivo declara lo que se llama el "módulo secundario". Este módulo secundario es un código de aplicación específica que implementa la funcionalidad que se necesita en la aplicación. En este caso estamos interesados en prender y apagar LEDs, de manera tal que el código es relativamente simple. La declaración del módulo secundario simplemente define la lista de los puertos que se usarán. Estos puertos corresponden directamente a pines físicos de I/O en la FPGA de la Xport (ver tabla C–5)

```
module RedGreen1 (CartData, CartAddr, FData, FAddr, CartCs, CartRd, CartWr, CartIReq,
                  FCe, FOe, FWe, PA, PB, ClkInA, ClkInB, GreenLED, RedLED, RCs, Clk);
```

La siguiente sección genera la *instancia* de lo que se denomina "Módulo Primario"

```
Primary InstPrimary(.CartData(CartData), .CartAddr(CartAddr), .CartCs(CartCs),
                    .CartRd(CartdRd), .CartWr(CartWr), .CartIReq(CartIReq),
                    .FData(FData), .FAddr(FAddr), .FCe(FCe), .FOe(FOe), .FWe(FWe),
                    .Addr(Addr), .Rd(Rd), .Wr(Wr), .SecDataRd(DataRd), .Clk(Clk));
```

El módulo primario contiene todas las funciones primarias que la mayoría de las configuraciones lógicas probablemente necesitarán. Dentro de este módulo, por ejemplo, se encuentra el código que demultiplexa el bus del cartucho del GBA en señales de datos y dirección usables. El módulo primario que se estará usando tambien contiene lógica para las comunicaciones con el CPORT, reset sincrono de encendido, identificación de configuración, y control de LEDs (pretenderemos por ahora que este no existe). Debido a que el ejemplo es sencillo, se necesitará solamente la lógica de demultiplexación contenida en el módulo primario. Este módulo es incluido usando una declaración `include` en la parte superior de redgreen1.v. El código fuente del mismo puede encontrarse en C:\xport\logic\src\primary.v. Una de las ideas detrás de este módulo es que es posible modificarlo basandose en las funcionalidades que se consideren primaria para su(s) aplicacion(es). Pero sin importar si es un módulo primario modificado o el original, es necesario hacer la *instancia* o el código del GBA no podrá ejecutarse debido a la falta de demultiplexación.

Note que la *instanciación* usa notación de puntos para pasar las señales. Esto previene errores en la mala asignación de señales. Las señales del Bus Game Pack que el módulo primario provee demultiplexadas están detalladas en la tabla C–6.

Table C–6: Señales del bus en primary

Señal	Dirección[a]	Descripción
Addr	Salida	Bus de dirección de 24-bit demultiplexado del puerto del cartucho del GBA. El puerto del cartucho es mapeado desde 0x8000000 hasta 0x9ffffff (32Mbytes). Accesos que ocurran fuera de este rango no son presentados al cartucho y por tanto no son vistos por la lógica de la FPGA. Cada dirección corresponde a una posición de memoria de 16-bit, y esta es la razón por la cual hay solo 24-bit de dirección (en vez de 25) para expandir a 32Mbyte el espacio de direcciones del cartucho
Wr	Salida	Nivel alto cuando el GBA está escribiendo dentro del rango de direcciones 0x8000 a 0x9ffffff
CartData	Salida	Datos de 16-bit del GBA. CartData es valida mientras Wr esté en lógica alta
Rd	Salida	Nivel alto cuando el GBA está leyendo dentro del rango de direcciones 0x8000 a 0x9ffffff
SecDataRd	Entrada	Datos de 16-bit al GBA. Los datos presentados en este puerto deben ser validos cuando la señal Rd esté en lógica alta.

[a] Dirección con respecto al módulo Primario. Por ejemplo, Addr es una *salida* del módulo

Después de la instanciación de el módulo primario, el resto del archivo redgreen1.v es dedicado al código espécifico de la aplicación. Para controlar los LEDs desde el código de

C++, es necesario crear un registro (LEDReg) en el espacio de direcciones del cartucho. Cada bit del registro corresponderá al estado de un LED (1 ó 0, encendido o apagado). Debido a que solo hay dos leds (verde y rojo en la xport, azul y rojo en el MEXGBA), son necesarios solo 2 bits para el registro. es necesario crear la señal LEDEn, la cual será usada para mapear la ubicación del registro en el espacio de direcciones del cartucho.

```
assign LEDEn = Addr[23:8]=16'hffe2;
```

Aquí se escogió 0xffe200. Para calcular la dirección que será usada por el software del GBA, se toma este valor, se multiplica por 2 y se le suma 0x8000000 para obtener 0x9ffc400. Esta dirección será usada por el software de ejemplo en C++ para controlar los LEDs.

A continuación se escriben las operaciones al registro de LEDs (LEDReg)

```
always @(negedge Wr)
   begin
   if (LEDEn)
       LEDReg <= CartData[1:0];
   end
```

Esta serie de instrucciones implica esperar hasta el flanco negativo de la señal Wr para luego guardar los datos del GBA en el registro LEDReg, pero si la dirección es válida (LEDEn es 1).
La lectura del estado del registro LEDReg no es absolutamente necesaria (el único objetivo es controlar los LEDs), pero es una opción interesante. Para el manejo de operaciones de lectura al registro LEDReg, se deben presentar simplemente los contenidos de LEDReg a DataRd cuando LEDEn esté activo. Cabe anotar que DataRd es instanciado en el módulo primario como la señal SecDataRd (ver tabla C–6).

```
always @(LEDEn or LEDReg)
   begin
   if (LEDEn)
       DataRd = {14'b00000000000000, LEDReg[1:0]};
   else
       DataRd = 16'hxxxx;
   end
```

Aquí, se implementó lo que podría ser el inicio de un multiplexador lógico para DataRd. Aunque el multiplexar una señal (LEDReg) en DataRd no resulta en la instanciación de un multiplexador lógico, si permite que futuras señales sean multiplexadas fácilmente. Note que el multiplexador no es dependiente de la señal Rd, lo cual es de esperar. El módulo Primario toma el control de la sincronización de la señal Rd.
Finalmente, se puede asignar las señales de los LEDs a los estados correspondientes de LEDReg. Debido a que las señales de los LEDs son activas bajas, se inverten los bits del registro.

```
assign RedLED = ~LEDReg[0];
assign GreenLED = ~LEDReg[1];
```

Sintetizando

Para sintetizar el diseño es necesario indicarle al sintetizador de Xilinx donde encontrar el código del módulo primario (Primary.v). Para hacer esto, se deben cambiar las preferencias de propiedades a avanzadas. Desde el menu "Edit" seleccione "Preferences", del cuadro de dialogo seleccione el tab "Processes" y "Advanced" para el "Property Display Level" como se muestra en la figura C–13. ISE recordará estas preferencias y no será necesario volver a realizar este procedimiento en el futuro.

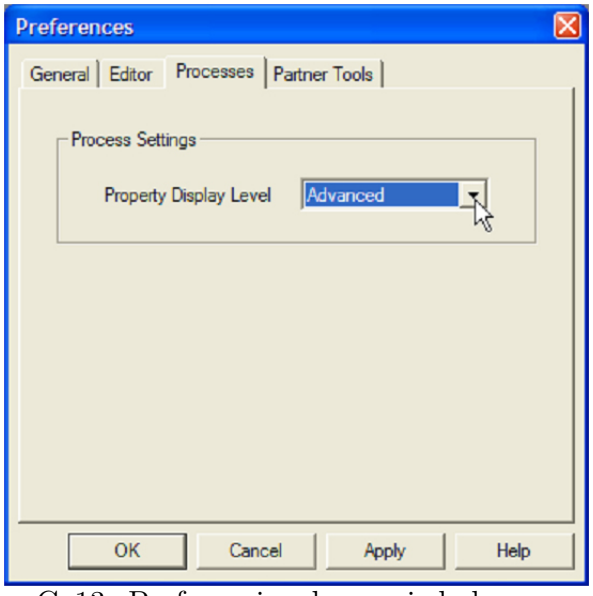


Figure C–13: Preferencias de propiedades avanzadas

A continuación, desde panel "Processes for Current SSource" (ver fig C–14) se debe hacer clic derecho en "'Synthesize" y seleccionar "Properties..." para llamar al cuadro de dialogo de "Process Properties" como se muestra en la figura C–15.

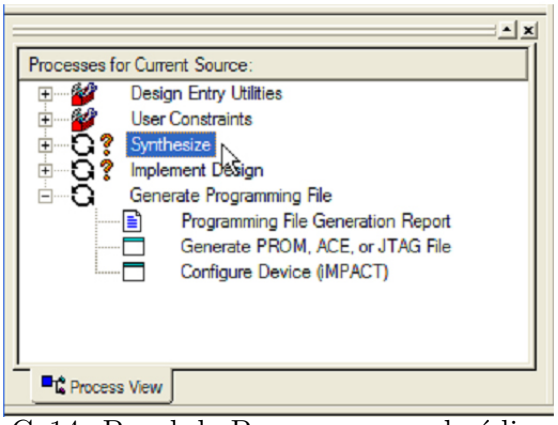


Figure C–14: Panel de Procesos para el código actual

Descienda hasta la opción "Verilog Include Directories" y haga clic en el botón como se muestra en la figura C–15. Desde el explorador de archivos, seleccione el directorio

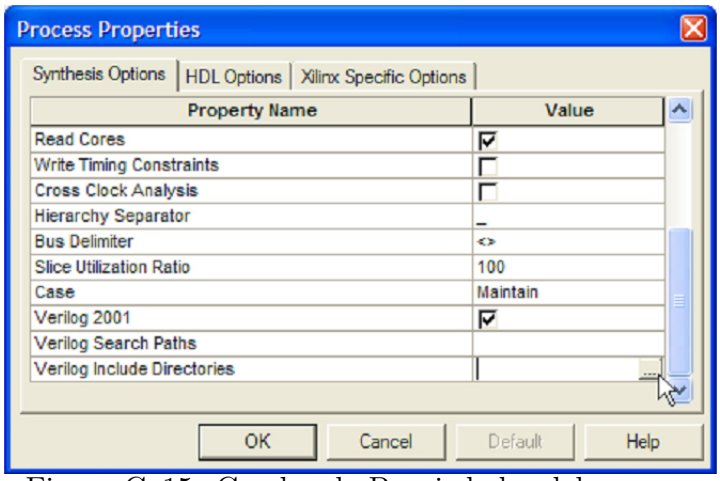


Figure C–15: Cuadro de Propiedades del proceso

logic src (típicamente ubicado en C: xport logic src) y haga clic en OK. Ahora haga doble clic en "Synthesize" en el panel "Processes for the Current Source" (figura C–14). Esto ejecutará el proceso de sintetizado. Cuando finalice el panel de procesos se verá como en la figura C–16. El signo de admiración "!" amarillo indica que uno o más advertencias

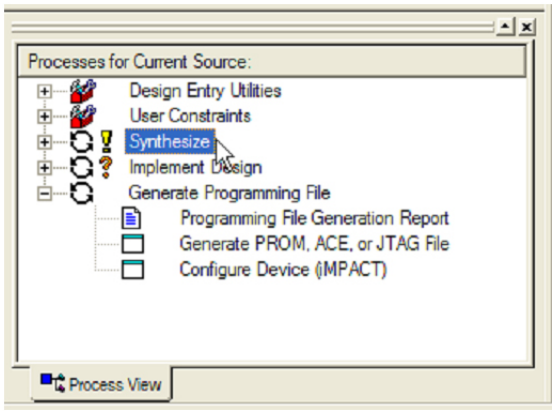


Figure C–16: Sintesis Completada

fueron generadas. Las advertencias en este caso pueden ser ignoradas.

Implementación

Después de haber sintetizado el proyecto, se puede llevar el diseño a la FPGA del MEXGBA (este proceso se conoce como implementación). Pero antes de hacerlo se deben añadir el archivo de asignación/restricción de pines al proyecto. Seleccione "Add Source" del menú "Project" y en el explorador de archivos que aparece seleccione el archivo xport.ucf que se encuentra en el directorio logic src (c: xport logic src xport.ucf) y haga clic en "Open" (ver figura C–17).

Después de haber añadido el archivo de restricciones, este será resaltado en el panel "Sources in Project". Re seleccione el archivo redgreen1.v como se muestra en la figura C–18.

Tambien es necesario indicarle a la utilidad de implementación donde se encuentran los módulos precompilados o "macros". Por ejemplo, el módulo de demultiplexación del cartucho del GBA es provisto en forma de macro. Haga clic derecho en "Implement

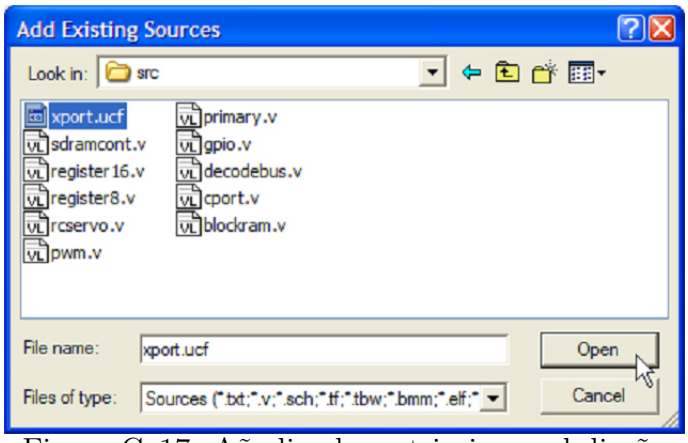


Figure C–17: Añadiendo restricciones al diseño

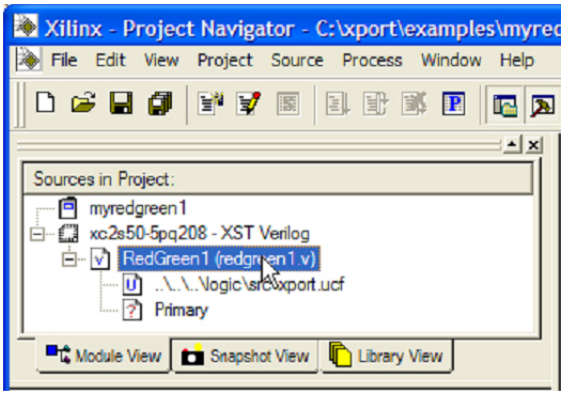


Figure C–18: Reseleccionando redgreen1.v

Design" en el panel "Processes for Current Source" (ver figura C–19) y seleccione "Properties. . . " para entrar al cuadro de "Process Properties".

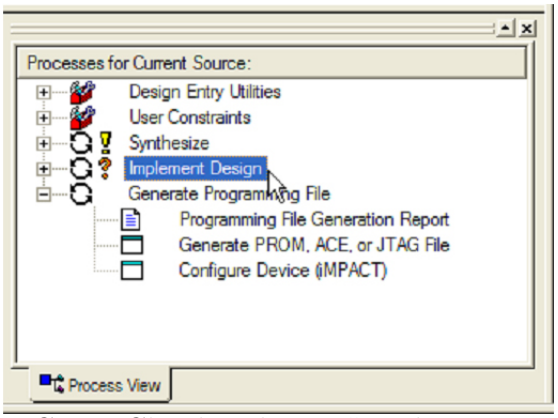


Figure C–19: Clic derecho en "Implement Design"

Encuentre la opción de "Macro Search Path" y haga clic en el botón como es muestra en la figura C–20. Desde el explorador de archivos, busque el directorio logic lib (c: xport logic lib) y luego haga clic en OK.

Luego, encuentre la opción "Allow Unmatched LOC Constraints" en el cuadro de "Process Properties" y haga clic en esta. Clic en OK para cerrar el cuadro. Para finalmente

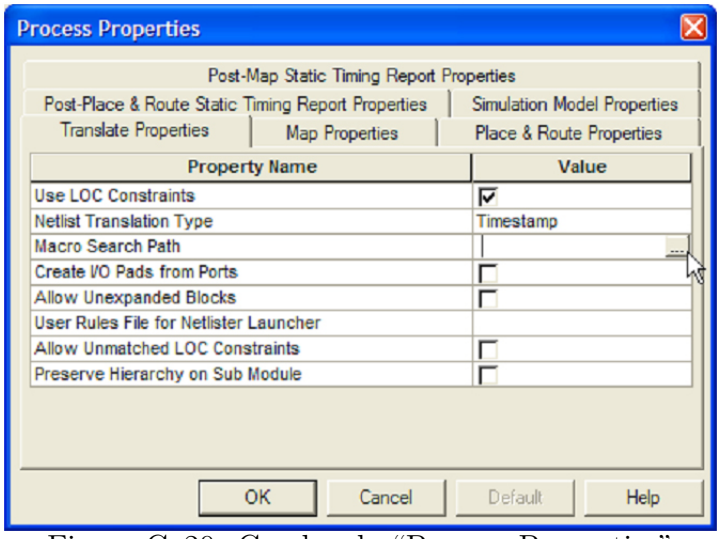


Figure C–20: Cuadro de "Process Properties"

implementar el diseño se debe hacer clic en "Implement Design" en el panel de "Processes for the Current Source". De nuevo, el signo de admiración "!" amarillo indica que una o más advertencias fueron generadas. Estas advertencias pueden ser ignoradas.

Lo único que resta por hacer ahora es generar el archivo de programación. Diríjase al final de la ventana "Processes for Current Source" y haga doble clic en "Generate Programming File" (figura C–21). Esto generará el archivo redgreen1.bit, el cual puede ser enviado al MEXGBA.

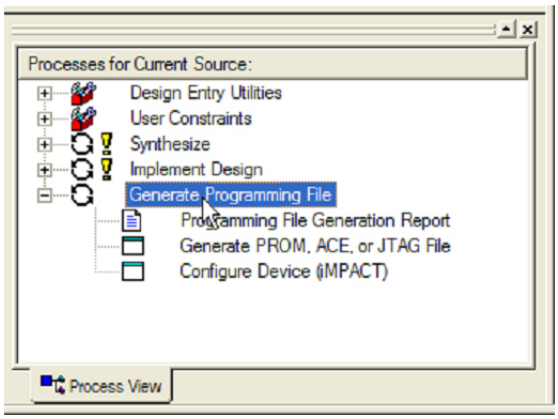


Figure C–21: Generando el archivo de programación

Este manual está basado en los documentos "Xport User Guide"[15] y "Xport Custom Configuration Tutorial"[29] proporcionados por Charmedlabs como herramientas didácticas para el uso de la tarjeta de expansión Xport.

Anexo D

Configuración inicial y otras

En el estado inicial, ningún dispositivo de la tarjeta de expansión esta configurado. Este apéndice describe en detalle los pasos necesarios para configurar el módulo exitosamente. Además contiene información acerca de otras configuraciones para acceder a la EWRAM y descargar la BIOS del GBA.

D.1 Configuración del CPLD

El CPLD es el encargado de gestionar la programación de la FPGA en cada energización del sistema y su configuración se realiza mediante la interfaz JTAG. Esta configuración se realiza con el software iMPACT, que hace parte del ISE WebPACK proporcionado por Xilinx, fabricante del dispositivo.

El archivo de configuración puede ser descargado de otro MEXGBA funcional de la siguiente manera:

1. Abra el programa iMPACT y espere el cuadro de diálogo que se muestra en la fig. D–1.

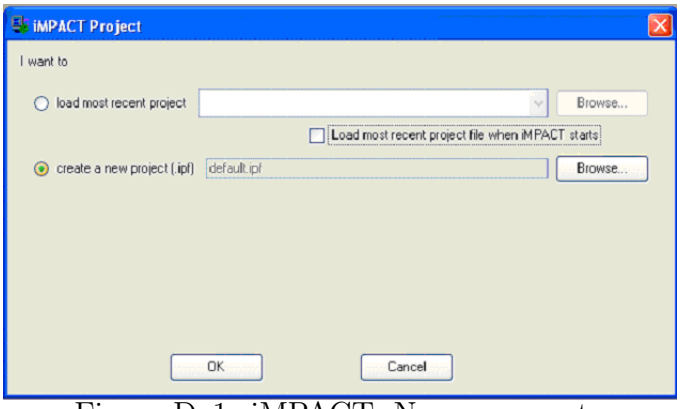


Figure D–1: iMPACT: Nuevo proyecto

2. Cree un nuevo proyecto y a continuación seleccione la opción "*Configure devices using Boundary-Scan (JTAG) / Automatically connect to a cable and identify Boundary-Scan chain*", como se muestra en la fig. D–2.

   Antes de continuar con el siguiente paso, se debe conectar el cable JTAG en el conector del MEXGBA, asegurandose que el módulo de expansión esté conectado

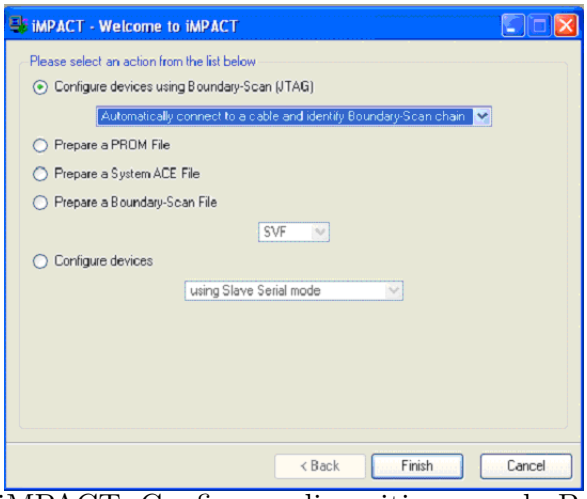


Figure D–2: iMPACT: Configurar dispositivo usando Boundary-Scan

al GBA y que este último se encuentra encendido[1] . Una vez conectado el cable y cumplidos los requisitos ya mencionados, se prosigue a finalizar.

3. En el siguiente cuadro de diálogo se pedirá el archivo de configuración con extensión .JED a programar, pero como en este caso se pretende "leer" se omite esta opción y se da clic en cancelar.

4. Como se observa, el software iMPACT ha detectado exitosamente el dispositivo (XC9536XV), entonces el paso a seguir consiste en extraer el archivo de configuración del mismo, para esto, se selecciona el CPLD que aparece en la pantalla y sobre el cuadro izquierdo se hace doble clic en la opción "*Readback*" como se muestra en la fig. D–3.

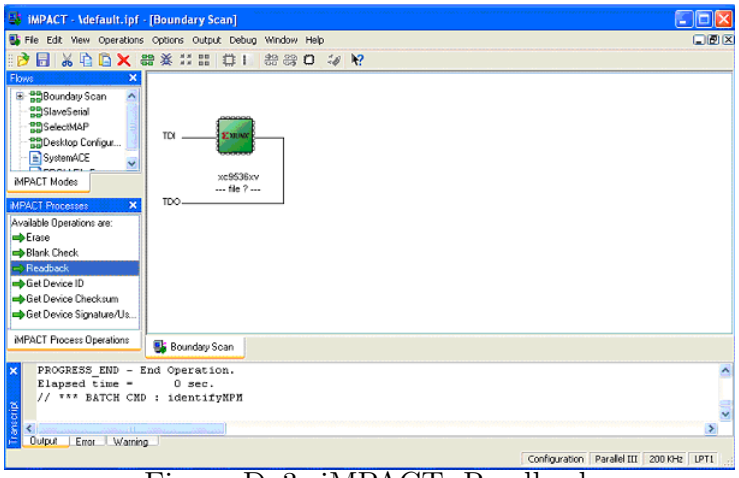


Figure D–3: iMPACT: Readback

[1] Es necesario encender el GBA para polarizar el CPLD.

5. A continuación debe aparecer un cuadro de diálogo que permita seleccionar el directorio en el cual se desea guardar el archivo a extraer.

Para cargar este archivo en el otro MEXGBA se sigue el mismo procedimiento hasta el tercer paso, en el cual se debe especificar el archivo a "escribir" en el CPLD.

D.2 Cargar el *slot* 1 en la memoria flash

Después de haber programado el CPLD, se prosigue a cargar en el *slot* 1 de memoria flash el código que corresponde a la programación de la misma. La flash se programa por intermedio de la FPGA (ver fig. 2–4) y para esta operación la FPGA debe conocer la manera de enviar las tramas y la temporización de las señales de control. El *slot* 1 contiene la información que corresponde a la configuración para la programación de la flash y es indispensable para empezar a ejecutar programas con el GBA.

La programación del *slot* 1 de la flash se realiza por medio del xpcomm, por tal motivo es necesario tener conectado el MEXGBA al GBA, y el cable CPORT al puerto paralelo. Una vez conectado todo se procede a usar el comando de actualización del *slot* 1, abriendo el Xport Shell y ejecutando "`xpcomm -u`" (ver fig. D–4).

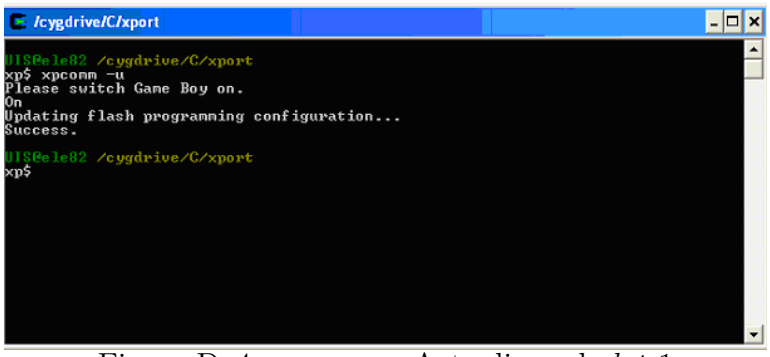


Figure D–4: xpcomm: Actualizar el *slot* 1

De esta manera el MEXGBA queda listo para recibir el código que el usuario desee ejecutar en el GBA.

D.3 El *slot* 0 de la memoria flash

El slot 0 de la memoria flash contiene la configuración lógica usada por el código de usuario, y en caso tal que no sea especificada, se programa por defecto la configuración lógica básica que corresponde a la demultiplexación de bus que se encuentra representada en el módulo primary, cualquier otra configuración siempre será compilada vinculando el módulo primary quien contiene al módulo decodebus.

Esta programación se explica en detalle en el *Anexo C, Manual de usuario, en la pág 66.*

D.4 Cargar un programa en la EWRAM

Para realizar esta operación ser requiere del cable Multiboot y del software Xboo, este software hace uso del puerto paralelo del computador y por tal se hace necesario tener privilegios de administrador y en algunos casos de otro programa adicional (UserPort) que permita una acceso de bajo nivel a este puerto.

El Xboo y el UserPort pueden ser descargados de:
http://www.work.de/nocash/gba-xboo.zip
Los pasos a seguir se listan a continuación:

1. Extraiga los archivos de los zip en cualquier directorio.

2. Copie el archivo UserPort.sys a C:\WINDOWS\system32\drivers.

3. Ejecute el programa UserPort.exe, seleccione los parámetros por defecto, clic en START, luego en EXIT.

4. Copie el ROM que quiere descargar al directorio del Xboo.

5. Para el Xboo con interfaz gráfica añada el archivo haciendo clic en el logo del GBA. Para el Xboo en DOS, abra una consola de DOS, ubíquese en el directorio del Xboo, y ejecute la siguiente linea de comando:

   Xboo /378 rom.gba,

Usualmente para win NT el puerto es /378 y para win 98 es /278.

D.5 Lectura de la BIOS del GBA

La BIOS del GBA es probablemente una de las características menos comprendidas de esta consola. La BIOS (*Basic Input/Output System*) se encuentra en el bus del sistema (32-bits) y tiene un tamaño de 16kB. En el GBA el código almacenado en la BIOS puede ser usado para tres propósitos:

- Secuencia de introducción

  Se encarga de verificar que el cartucho es original, comparando el código correspondiente al logo de Nintendo y posteriormente mostrandolo en pantalla cada vez que se inicia el sistema.

- Funciones SWI

  Contiene funciones funciones de matemática, inicialización, descompresión y otras que pueden ser usadas por el software del GBA.

- Función Multiboot

  Permite iniciar el GBA desde otros computadores o consolas inclusive sin tener un cartucho insertado.

En este punto cabe resaltar que la memoria BIOS sólo puede ser leída si el contador de programa (PC) apunta a una dirección del espacio de memoria que le corresponde (`0x00000000 - 00003FFFF`).

Para descargar la BIOS a un archivo exiten varios métodos, siendo el más sencillo mediante el uso del cable Multiboot con el software Xboo mencionado anteriormente.

Se ejecuta el comando:

1. Xboo /378 /b rom.gba

2. El programa rom.gba debe tener una cabecera valida, el proceso para descargar la BIOS del gba es sencillo, primero se sube la cabecera del rom al gba y procede a descargar la BIOS al archivo “xboo.tmp”.

Anexo E

Código de la aplicación

En este Apéndice se muestra el código en C++ con el cual se programó el GBA con la intención de ilustrar la cómo se configuran los periféricos y su uso.

```
\texttt{
#include "../../include/textdisp.h"
#include "../../include/xport.h"
#include "../../include/gba.h"

CTextDisp td(TDM_LCD_AND_CPORT);

//configuraci\'on de las direcciones de los perif\'ericos

#define PWM_ADDR 0X9FFC800
#define PWM_REG_NUM 1
#define PWM_REG(i) *((volatile unsigned short *)PWM_ADDR+i)
#define PWM_DDR(i) PWM_REG(i)

#define GPIO_NUM 15
#define GPIO_ADDR 0x9ffc400
#define GPIO_REG_NUM (GPIO_NUM+15)/16
#define GPIO_REG(i) *((volatile unsigned short *)GPIO_ADDR+i)
#define GPIO_DDR(i) GPIO_REG(i)
#define GPIO_DATA(i) GPIO_REG(i+GPIO_REG_NUM)

#define pwm_alto 255
#define pwm_medio 127
#define pwm_bajo 63
#define t_retardo 200
#define gir90der 430
#define gir90izq 410
```

```
volatile short  sensor_pas;
volatile unsigned  ban;

extern "C"
{ void _gba_interrupt(void); }

void _gba_interrupt(void) //gertion de la interrupcion del timer.
{
if (GBA_REG_IF & GBA_INT_TIMER0)
{
GBA_REG_IF = GBA_INT_TIMER0;  // reset interrupt flag
ban=ban+1;
}
}

void adelante(int izq, int der) {
PWM_DDR(0)=(izq<<8); //motor der
PWM_DDR(1)=(der<<8); //motor izq
}
void derecha(void){
PWM_DDR(0)=(pwm_alto<<8);
PWM_DDR(1)=pwm_alto;
}
void izquierda(void){
PWM_DDR(0)=pwm_alto;
PWM_DDR(1)=(pwm_alto<<8);
}
void parar(void){
PWM_DDR(0)=0;
PWM_DDR(1)=0;
}
void atras(void){
PWM_DDR(0)=pwm_bajo;
PWM_DDR(1)=pwm_bajo;
}
int sensor(void){
return (GPIO_DATA(0)&15);
}

void retardo(unsigned time){  //retardo en ms
int k;
ban=0;
for(k=0;k<time;k++)
{ GBA_REG_TM0D = 0xfff0;  //cuenta 1ms
GBA_REG_TM0CNT = 0xc3;
GBA_REG_IF = 0xffff;  // clear all flags
```

```
GBA_REG_IE = GBA_INT_TIMER0;  // enable timer interrupt
GBA_REG_IME = 1;  // set master interrupt enable
while(ban<time/2) {}
XP_REG_LED=1;
while(ban<time) {}
XP_REG_LED=2;  // LED feedback
GBA_REG_TM0CNT = 0x000;
}
}
void linea_s(void){
td.Printf("dato %d\n",sensor() );
switch (sensor())
{
case 0x0: //desalineado, corregir suave
if(sensor_pas==sensor())
atras();
break;

case 0x1: //salido de la linea
retardo(129);
derecha();
break;

case 0x2: //desalineado..ir a la derecha
adelante(pwm_alto,pwm_medio);
break;

case 0x3: //desalineado
adelante(pwm_alto,0);
break;

case 0x4: // desalineado , izq
adelante(pwm_medio,pwm_alto);
break;

case 0x5: derecha();
break;

case 0x6: //alineado
adelante(pwm_alto,pwm_alto);
break;

case 0x7: // alineado, giro a la deracha
derecha();
break;
```

```
case 0x8: //salido de la linea
izquierda();
break;

case 0x9: //???
izquierda();
break;

case 0xa: //desalineado , cruce a la izq
izquierda();
break;

case 0xb: //???
derecha();
break;

case 0xc: //desalineado, izq
adelante(pwm_bajo,pwm_alto)
break;

case 0xd: //???
izquierda();
break;

case 0xe: //desalineado, cruce a la izq
izquierda();
break;

case 0xf: //alineado, llego a una T
derecha();
break;
}
sensor_pas=sensor();
}

int main(void){
GPIO_DDR(0)=0; //se habilitan los 15 GPIO's como salidas.
PWM_DDR(0)=0xf; //se habilitan los 4 primeros canales del PWM.

while(1)
{
linea_s();

}}
```

Bibliografía

[1] Marcel Cremmel. Game boy ecg. *Elektor Electronics*, 10:32–26, 2006.

[2] Steve Willis. Game boy digital sampling osciloscope. *Elektor Electronics*, 10,11:34–39, 12–15, 2000.

[3] Mikhail Sharonov. *GBA GPS*.
http://www.msh-tools.com/GBA/gbagps.html, 2005.

[4] SongPro. *Songpro - The world's first module MP3 player*.
http://www.dmgice.com/hardware/yaksden.htm.

[5] Tony Givargis Frank Vahid. *Embedded system design: A unified Hardware/Software approach*. University of California, primera edition, 1999.

[6] Edwin Notenboom Bart Broekman. *Testing Embedded Software*. Addison-Wesley, primera edition, 2003.

[7] ARM. *ARM7TDMI Technical Reference Manual*. ARM, tercera edition, 2001.

[8] Jonathan S. Harbour. *Programming the Nintendo Game Boy Advance: The Unofficial Guide*.
http://www.jharbour.com, 2003.

[9] Reiner Ziegler. *Game Boy Advance Development*.
http://www.ziegler.desaign.de/GBA/gba.htm, 2006.

[10] Reiner Ziegler. *GBA Link Port*.
http://www.ziegler.desaign.de/GBA/GBAlinkport.pdf, 2006.

[11] Tim Brolin. *A Homebrew 8Mbit GBA Flashcart.* http://www.brolinembedded.se/projects/flashcart/, 2004.

[12] Thorsten Godau. *The DL9SEC GBA flashcart.* http://www.dl9sec.de/gbacart/gbacart.htm, 2005.

[13] Michael Barr. *Embedded Systems Dictionary.* CPM, primera edition, Junio 2003. http://www.netrino.com/Publications/Glossary/index.php.

[14] STMicroelectronics. *M58LW032D Datasheet.* STMicroelectronics, 2004.

[15] Charmedlabs. *Xport 2.0 User Guide.* http://www.charmedlabs.com, 2003.

[16] Sandra M. Ovallos G. *Diseño de un microprocesador usando el lenguaje de descripción de hardware VHDL.* UIS, primera edition, 2006.

[17] Barry Olney. *EMC Design for High Speed PCB's.* http://www.icd.com.au/articles/emc.html, 2005.

[18] CTS Communication Components Inc. *CB3LV Datasheet.* CTS, 2000.

[19] Kim Goldblatt. Power-on requeriments for the spartan-ii and the spartan-iie families, 2001.

[20] Institute for Interconnecting and Packaging Electronics Circuits. *IPC 2221 Generic Standard on Prited Board Design.* IPC, 1998.

[21] NS Wales University. *An Introduction to the GNU Compiler.* http://www.zap.org.au/elec2041-cdrom/unsw/common/compiler-intro.pdf, 2003.

[22] Carlos Iván Camargo. *Implementación de sistemas digitales complejos usando sistemas embebidos.* Universidad Nacional de Colombia.

[23] NS Wales University. *An Introduction to the GNU Debugger*. http://www.zap.org.au/elec2041-cdrom/unsw/common/gdb-intro.pdf, 2003.

[24] Samir Palnitkar. *Verilog HDL: A Guide to Digital Design and Synhtesis*. Prentice Hall, segunda edition, 2003.

[25] Tom Happ. *Cowbite Virtual Hardware Specifications*. http://www.cs.rit.edu/t̃jh8300/CowBite/CowBiteSpec.htm, 2002.

[26] *The Nintendo Reverse Engineering Project*. http://www.thepernproject.com, 2005.

[27] Raúl Jiménez. *Diseño de sistemas empotrados - Apuntes de Clase*. http://www.uhu.es/raul.jimenez/, 2005.

[28] Inc. Free Software Fundation. *A GNU Manual:Using the GNU Compiler Collection (GCC)*. FSF, 2005.

[29] Charmedlabs. *Xport Custom Configuration Tutorial*. http://www.charmedlabs.com, 2003.