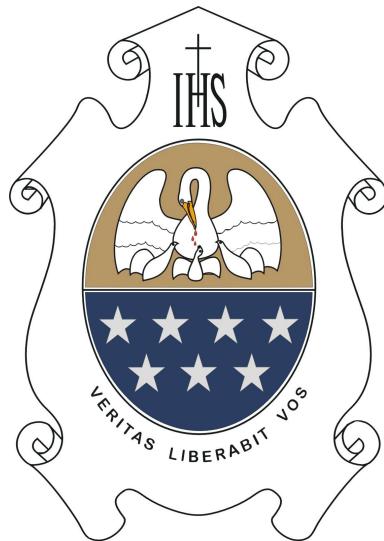


Universidad Católica de Córdoba

Facultad de Ingeniería

Ingeniería Electrónica



TRABAJO FINAL

“*OpenBraille*”
Un dispositivo libre
para impresión braille

Presentado por: Victor Hugo Rosales

Tutor: John Coppens
Córdoba, Argentina.

Agradecimientos

... a mis padres ...



Índice

1. Diagnóstico	10
2. Braille	12
2.1. Historia	12
2.2. Técnica	13
2.3. Braille escrito	17
2.3.1. Regleta/pauta y punzón	17
2.3.2. Maquina de escribir braille	18
2.3.3. Impresoras braille	19
3. Panorama General	23
4. Tecnologías a usar	25
4.1. Comunicación	25
4.2. Microcontrolador	27
4.3. Sistema Operativo	29
4.4. Compiladores	29
4.4.1. Driver	29
4.4.2. Firmware	29
4.5. Simulación y diseño	30
5. Módulos	31
5.1. Módulo de potencia	31
5.2. Modulo de procesamiento	34
5.2.1. Manejo de motores	39

5.3. Driver	41
6. USB	43
6.1. Descripción	44
6.1.1. Características	44
6.1.2. Arquitectura	44
6.1.3. Interconexión USB	45
6.1.4. <i>Host</i> USB	46
6.1.5. Dispositivo USB	47
6.2. Endpoints	49
6.3. Transferencias USB	50
6.3.1. Control	50
6.3.2. Bulk	50
6.4. Descriptores USB	52
6.5. Estados USB	53
6.6. Enumeración del bus USB	55
6.7. PIC18F4550 USB	56
6.7.1. Registros	56
6.7.2. USB RAM	58
6.7.3. Descriptores de Buffer y Tabla de Descriptores de Buffers	60
6.7.4. BDn	60
7. Impacto Ambiental	62
8. Impacto Social	63
9. Economía	65
9.1. Modelos de negocios	65
9.1.1. Financiación externa de empresas	65
9.1.2. Uso Interno	66
9.1.3. Mejor conocimiento aquí sin limitaciones	66
9.1.4. Mejor conocimiento aquí con limitaciones	67
9.1.5. Doble Licencia	67
9.1.6. Desarrollo no financiado	67
9.2. Posibles modelos particulares	68
9.2.1. Costos	68
9.2.2. Modelos de negocio	70
10. Conclusión	71
11. Anexo	73
11.1. Software Libre	73
11.1.1. Historia	73
11.1.2. GNU	73
11.1.3. GNU/Linux	74

11.1.4. Concepto de Software Libre	74
11.1.5. El Software Libre en la ingeniería	75
11.2. Mercado de impresoras braille	75
11.2.1. Index Braille	75
11.2.2. Viewplus	76
11.2.3. Enabling Technologies	76
11.3. USB firmware	77
11.3.1. Cabecera	77
11.3.2. Fuente USB	79
11.3.3. Fuente Main	95
11.4. USB Driver	102
11.4.1. Cabecera de funciones	102
11.4.2. Fuente de funciones	103
11.4.3. Cabecera de opciones	109
11.4.4. Fuente de opciones	109
11.4.5. Cabecera de manejador de errores	111
11.4.6. Fuente de manejador de errores	111
11.4.7. Fuente de archivo principal	113
11.5. Hardware	115
11.5.1. Driver motores paso a paso	115
11.5.2. Módulo de procesamiento	118
11.5.3. Bastidor de impresora	122



Índice de figuras

2.1. Celda braille.	13
2.2. Medidas de las celdas braille.	13
2.3. Hoja impresa con braille de ambos lados.	16
2.4. Relechas y punzones para escritura braille.	17
2.5. Primera maquina de escribir braille inventada por Frank Haven Hall en 1892.	18
2.6. Maquina de escribir braille actual del mercado.	19
2.7. Múltiples punzones deslizantes.	21
2.8. Corte cabezal IndexBraille. 1 - Alimentación de papel, 2 - Yunque de goma, 3 - Punzón o martillo	21
3.1. S.O, conexión e impresora	24
5.1. Motores paso a paso.	32
5.2. Motor unipolar paso a paso de 4 fases.	32
5.3. Circuito de potencia para una bobina de motor paso a paso unipolar. .	33
5.4. Simulación del circuito de potencia para una bobina de motor paso a paso unipolar.	33
5.5. Conexión ICSP (In-circuit Serial Programming)	34
5.6. Diseño de alto nivel.	35
5.7. Circuito opto acoplado para el percutor	35
5.8. Circuito opto acoplado para uno de los sensores	36
5.9. Canales de comunicación entre el driver y el firmware.	37
5.10. Canales de comunicación entre el driver y el firmware mediante endpoints.	37
5.11. Proceso de reconstrucción de datos	39

5.12. Puerto B del uC conectado a las señales de control de los motores paso a paso.	39
6.1. Cable USB.	45
6.2. Velocidad USB.	45
6.3. Topología del bus USB.	46
6.4. Endpoints de USB.	49
6.5. Transacción Bulk.	51
6.6. Descriptores USB.	52
6.7. Estados USB.	53
6.8. Arquitectura USB de la familia PIC18FX455/X550.	56
6.9. Stack FIFO del registro USTAT.	58
6.10. Mapeo de memoria <i>USB RAM</i>	59
6.11. Tabla de descriptores de Buffers <i>BDT</i>	60
9.1. Contribución de diversas empresas al kernel de linux.	69
11.1. Vista superior del driver de los motores paso a paso	115
11.2. Vista en 3D del driver de los motores paso a paso	115
11.3. Foto de la placa controladora de motores	116
11.4. Esquemático de la placa del driver de los motores paso a paso	117
11.5. Vista superior del la placa de procesamiento	118
11.6. Vista en 3D de la placa de procesamiento	118
11.7. Foto de la placa de procesamiento sin el microcontrolador	119
11.8. Foto de la placa de procesamiento con el microcontrolador	120
11.9. Esquemático de la placa del microcontrolador	121
11.10Foto del bastidor de una impresora matriz de punto antigua usado para hacer pruebas	122



Índice de tablas

2.1. Alfabeto castellano braille.	14
2.2. Mayúsculas braille.	14
2.3. Números braille.	15
4.1. Comparación PIC18F2550 y PIC18F4550	28
4.2. Comparación de precios PIC18F2550 y PIC18F4550	28
5.1. Bytes por puntos braille	37
5.2. Instrucciones de impresión	38
5.3. Byte de reporte	38
5.4. Secuencias de un paso para motores paso a paso unipolares de cuatro fases.	40
5.5. Secuencia de medio paso para motores paso a paso unipolares de cuatro fases.	40
6.1. Velocidades de USB	44
6.2. Campos Token	47
6.3. Campos Data	48
6.4. Campos Status	48
6.5. Estados USB.	54
9.1. Precios unitarios por cien unidades	68
11.1. Modelos de <i>INDEX BRAILLE</i>	75
11.2. Modelos de <i>Viewplus</i>	76
11.3. Modelos de <i>Enabling Technologies</i>	76



Prólogo

Durante el transcurso del trabajo se describe el desarrollo e implementación de la parte electrónica de una impresora braille. El objetivo final del mismo es proveer una posible solución de bajo costo, tanto de producción como de armado. Para ello, se estudiarán y analizarán todas aquellas alternativas disponibles que puedan satisfacer los requerimientos planteados.

La principal premisa con que se aborda el desarrollo de este trabajo es que mediante soluciones básicas y simples, es posible concretar diseños robustos y funcionales, siempre y cuando se hayan elegido las herramientas y condiciones correctas. Para lograr esto, gran parte del trabajo se centra en investigar alternativas de desarrollo no convencionales en la industria electrónica.

CAPÍTULO

1

Diagnóstico

En la actualidad existen muchas empresas¹ que fabrican dispositivos de impresión braille, pero debido a el tamaño reducido del nicho de mercado en el que se encuentran, sus precios suelen ser muy elevados y no están al alcance del ciudadano medio. Y al pertenecer, de una forma u otra, a un sector tecnológico, existe una gran competencia en cuanto al avance de sus tecnologías. Esto conlleva a que fabriquen dispositivos con muchas funcionalidades, prestaciones y de gran performance, dejando de lado diseños sencillos y meramente funcionales que harían al producto menos costoso.

Otro problema que presentan estos dispositivos, es que, a falta de estándares de impresión braille, cada fabricante provee su propia solución de software que suele ser un costo extra en algunas ocasiones. Por este mismo motivo el soporte que proveen suele limitarse a un único sistema operativo² forzando al usuario a comprar una licencia del mismo e incluso en muchos casos un único procesador de texto³.

Se encuentra también dicha industria embebida en modelos de desarrollo privativo, haciendo imposible al usuario final poder agregar cambios basándose en sus necesidades particulares. Si bien el modelo de desarrollo privativo es uno de los más usados en todas las industrias, existen varias que se encuentran, ya sea en etapas de exploración o producción, trabajando con modelos de *Código Abierto*⁴ o incluso *Software Libre*⁵, siendo esto un lujo que la industria de dispositivos de impresión braille no puede darse

¹Ver la página 75 del capítulo 11.2

²Normalmente Microsoft Windows.

³Normalmente Word de la suite Microsoft Office.

⁴Del inglés *open-source*

⁵Del inglés *Free Software*

debido mayormente a su tamaño.

Las problemáticas antes planteadas hacen que un posible mercado nacional de estas tecnologías sea prácticamente imposible, por lo que las impresoras braille deben ser adquiridas en el exterior o mediante un importador.

Todo esto conlleva a que los usuarios finales deben gastar una importante suma de dinero para poder realizar impresiones braille en su hogar, comprando un sistema operativo, una suite ofimática, y un dispositivo con prestaciones que exceden las necesidades del mismo.

CAPÍTULO

2

Braille

2.1. Historia

El sistema Braille es un método utilizado por personas ciegas para leer y escribir. Fue ideado en 1821 por el francés *Louis Braille*¹. Se basa en un método de comunicación desarrollado y perfeccionado por *Charles Barbier*², en respuesta a la demanda de Napoleón, de un código que los soldados pudieran usar para comunicarse en silencio y sin luz en la noche. Se lo llamó *Night writing*³. El sistema de Barbier era demasiado complejo para los soldados de aprender, y fue rechazada por los militares. En 1821, Barbier, visitó el Instituto Nacional para Ciegos, en París, donde conoció a *Louis Braille*, quién identificó el mayor defecto del código: el dedo de la mano humana no puede abarcar todo el símbolo sin moverse, y así no puede pasar rápidamente de un símbolo a otro. Su modificación fue utilizar una celda de 6 puntos (el sistema Braille) que revolucionó la comunicación escrita de los ciegos. Cada célula (o celda) braille o carácter se compone de seis posiciones de puntos, dispuestos en un rectángulo que contiene dos columnas de tres puntos cada uno. Un punto puede ser colocado en alguna de las seis posiciones para formar sesenta y cuatro (2^6) permutaciones, incluido el arreglo de puntos que no se coloca. Una permutación puede ser descrita nombrando las posiciones en que se disponen los puntos: Las posiciones están universalmente numerados de 1 a 3, de arriba a abajo, a la izquierda, y 4 a 6, de arriba a abajo, a la derecha como se muestra en la figura 2.1.

¹Véase - http://es.wikipedia.org/wiki/Luis_Braille

²Véase - http://en.wikipedia.org/wiki/Charles_Barbier

³Véase - http://en.wikipedia.org/wiki/Night_writing

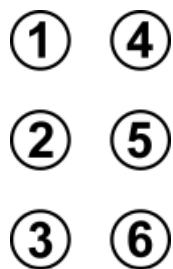


Figura 2.1: Celda braille.

2.2. Técnica

Las líneas horizontales de texto en Braille están separados por un espacio a fin de que los puntos de una línea puede ser diferenciada de la de texto en braille por encima y por debajo. La puntuación está representada por su propio conjunto de caracteres único. No existe una estandarización rigurosa para las distancias o medidas entre los puntos de las celdas braille, aunque si se ha generado un estándar implícito como muestra la figura 2.2.

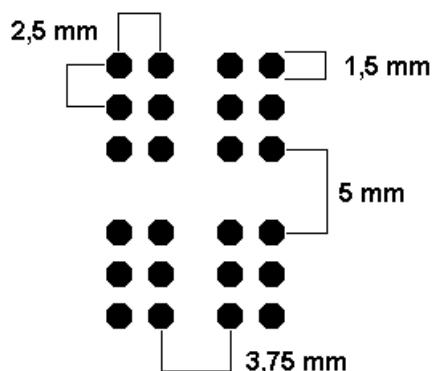


Figura 2.2: Medidas de las celdas braille.

La combinación de estos puntos generan el alfabeto braille y todos los símbolos, como se muestra en la tabla 2.1.

Debido a la limitación que impone la reducida cantidad de combinaciones que pueden generarse con éste sistema de seis puntos, *Louis Braille* propuso reservar algunas combinaciones⁴ que, actuando como prefijos o sufijos y en contexto, pueden cambiar el significado de los símbolos adyacentes. Los ejemplos más importantes son los de las letras mayúsculas (ver tabla 2.2) y la de los números (ver tabla 2.3).

⁴Éstos símbolos suelen variar dependiendo del idioma.

•	a	•••	n	•••	á
•	b	•••	ñ	•••	é
••	c	•••	o	••	í
••	d	•••	p	••	ó
••	e	•••	q	•••	ú
••	f	•••	r	••	,
••	g	•••	s	••	;
••	h	•••	t	••	:
••	i	•••	u	••	.
••	j	•••	v	•••	!
••	k	•••	w	•••	(
••	l	•••	x	•••)
••	m	•••	y	•••	*
		•••	z	•••	?
				•••	"

Tabla 2.1: Alfabeto castellano braille.

••	A
••	B
••	C
••	D
••	E
...	...

Tabla 2.2: Mayúsculas braille.

	1
	2
	3
	4
	5
	6
	7
	8
	9
	0

Tabla 2.3: Números braille.

Existen tres tipos de transcripción *braille*, conocidos como “Grado 1”, “Grado 2” y “Grado 3”. El *braille* de grado 1, es el ideado por *Louis Braille* y considerado oficial⁵ en la mayoría de los países. Los grados 2 y 3 son conocidos como *estenotipia*⁶ y tienen como finalidad economizar la cantidad de caracteres usados. Un ejemplo de esto es el usar un solo símbolo de número cuando lo que continua es todo un número como se ve a continuación:


(num) 1 2 3

O por ejemplo, si se quiere hacer saber que toda la palabra siguiente se encuentra en mayúscula, se anteponen dos símbolos de mayúscula como se ve a continuación:


(may)(may) H O L A

En la figura 2.3 se muestra una hoja impresa con braille y siendo leída con los dedos índice y anular.

⁵Esto depende exclusivamente del idioma y los países

⁶Véase - <http://es.wikipedia.org/wiki/Estenotipia>

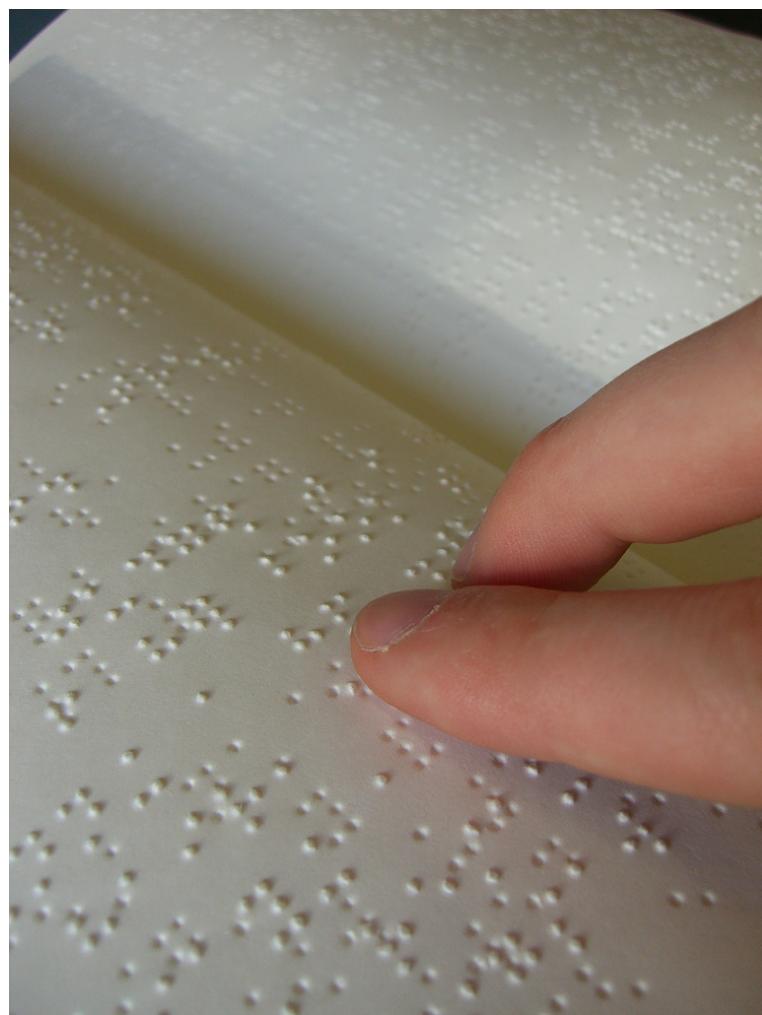


Figura 2.3: Hoja impresa con braille de ambos lados.

2.3. Braille escrito

Como se explicó en la sección anterior, el sistema braille escrito consta de celdas de dos columnas de tres puntos, cada uno de los cuales puede estar en relieve de la superficie en la que se está escribiendo. Es de imaginarse que existen muchas maneras de lograr esto. A continuación se explican los métodos más usados.

2.3.1. Regleta/pauta y punzón

Este método es uno de los más usados y quizá uno de los más económicos. Consta de dos plantillas de algún material firme⁷ unidas en un extremo mediante una bisagra. Una de las plantillas posee una matriz de celdas vacías con separaciones estándar entre ellas, la otra posee una matriz de celdas con huecos cóncavos siguiendo el estándar braille. Esta *regleta* se alimenta con papel⁸ entre ambas plantillas, permitiendo luego, mediante un punzón de mano, ir marcando el relieve de los puntos correspondientes a la letra braille que se desea escribir. La figura 2.4 muestra tres tipos de regletas y dos punzones.

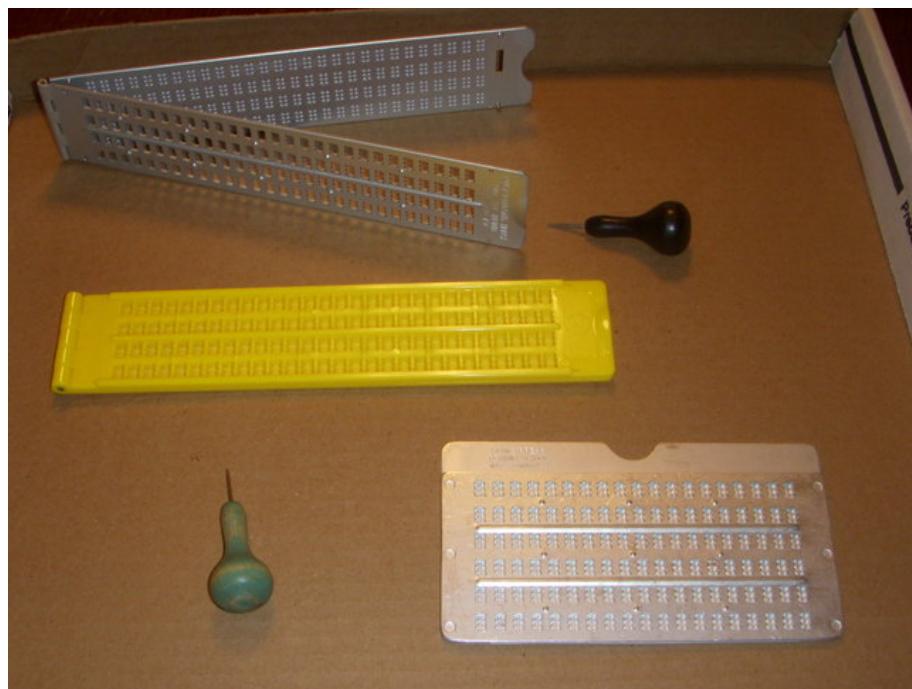


Figura 2.4: Regletas y punzones para escritura braille.

La principal desventaja de este método, es que requiere de una persona muy capacitada y que pueda pensar al revés⁹ a la hora de escribir, además de que se obtienen muy pocos caracteres por minuto.

⁷Normalmente de plástico o aluminio

⁸Este papel si bien no es necesario que sea especial, se usa algún tipo de papel grueso donde se puedan marcar puntos en relieve sin perforarlo.

⁹Esto se debe a que en las regletas las celdas se escriben del lado opuesto a donde se leerán

2.3.2. Maquina de escribir braille

En 1892 Frank Haven Hall inventó una maquina compleja completamente mecánica pero de funcionamiento muy sencillo para la escritura braille.

Hasta el día de hoy las actuales maquinas de escribir braille se basan en los mismos principios de funcionamientos que aquella que inventó Frank Haven Hall, pero la tecnología de fabricación se basa en materiales más livianos y resistentes que los de aquella época.

El mecanismo consiste de una parte móvil con movimientos horizontales donde se fija el papel el cual puede a su vez desplazarse hacia arriba o hacia abajo como en las maquinas de escribir comunes. Posee seis teclas que mueven un juego de vástagos que a su vez activan una serie de punzones que marcan la hoja. Estas seis teclas permiten escribir (o impactar) un carácter braille a la vez, y luego existe una séptima tecla¹⁰ que se encarga de desplazar la hoja hacia un lado permitiendo así escribir un carácter nuevo.

Se puede ver en la figura 2.5 la maquina original inventada por Frank Haven Hall, y en la figura 2.6 se muestra un modelo nuevo que se puede encontrar actualmente en el mercado.



Figura 2.5: Primera maquina de escribir braille inventada por Frank Haven Hall en 1892.

Estas maquinas, al igual que la regleta y el punzón, requieren una persona con conocimientos y práctica para su uso, pero poseen la ventaja de que se pueden lograr una mayor cantidad de caracteres por minuto. De hecho cuando Frank Haven Hall

¹⁰Esto sería similar a la barra espaciadora de las maquinas de escribir convencionales.



Figura 2.6: Maquina de escribir braille actual del mercado.

presentó su invento, impresionó a la audiencia con una velocidad de 58 palabras por minuto.

2.3.3. Impresoras braille

La primera¹¹ impresora braille¹² para papel normal fue fabricada en el Instituto de Tecnología de Masachusset (*MIT - Massachusetts Institute of Technology*)¹³ a finales de la década de los 60 y se llamó *BrailleEmboss* cuyo diseño estuvo a cargo de George Dalrymple¹⁴.

Esta impresora formaba parte de un proyecto más grande llamado DOTSYS¹⁵ cuyo objetivo era crear un programa para traducir texto en inglés a braille inglés estándar. Tenía la capacidad de producir una página braille de 38 celdas de ancho por 25 de largo cada 1.6 a 2 minutos. Soportaba diversos métodos de entrada de dato, entre ellos tarjetas perforadas, teclado o cintas de teletipo¹⁶. Debido al carácter mayormente investigativo de este proyecto solo se fabricaron una veintena de estas impresoras y su costo de producción es desconocido.

Existen actualmente en el mercado una amplia variedad de impresoras braille para cubrir la mayoría de las necesidades de los usuarios, desde impresoras personales hasta impresoras para grandes niveles de producción, con velocidades de impresión desde 10 hasta 400 caracteres por segundo y precios desde 3200 hasta más de 30000 dólares.

¹¹Véase - <http://www.duxburysystems.com/bthist.asp>

¹²El término correcto es *impactadora* debido a que proviene del inglés *embosser*

¹³Véase - <http://web.mit.edu/>

¹⁴Véase - http://www.ll.mit.edu/Retirees/Picnic05/index_04.html

¹⁵Véase - <http://www.dpawson.co.uk/braille/braille.html>

¹⁶Véase - <http://es.wikipedia.org/wiki/Teletipo>

Las impresoras braille comúnmente se especifican según:

- Velocidad medida en CPS (caracteres por segundo)
- Tamaño
- Peso
- Nivel de ruido en decibeles
- Tipo de papel que soporta
- Consumo de potencia eléctrica en estado inactivo
- Consumo de potencia eléctrica durante funcionamiento

Debido a su principio de funcionamiento de impresión por impacto el nivel de ruido es una de las características más importantes al igual que su peso proveniente mayormente de la fuente y los mecanismos de impacto.

La característica más atrayente de las impresoras braille es que pueden ser utilizadas por personas con muy poco conocimiento, ya que la parte técnica del braille es resuelta enteramente por software, entonces basta solo con escribir el texto que se desea imprimir en el idioma deseado y luego mediante software este es traducido y enviado a la impresora quien se encargara de imprimirlo.

Funcionamiento

La mayoría de las impresoras braille del mercado tienen un funcionamiento similar. Se escribe el texto que se desea imprimir en un editor de texto, ya sea común o bien uno provisto por el fabricante, una vez que se ha terminado el texto se necesita generar una especie de vista preliminar donde se pueden observar las modificaciones que agregará el programa para cumplir con las limitaciones del braille usando una tipografía braille. Luego el documento se *traduce* a un metalenguaje que contiene tanto el texto como caracteres adicionales inherentes al braille pero con una codificación particular que interpretará la impresora. La información codificada puede ser guardada para una posterior impresión o directamente puede enviarse a imprimir.

La forma en la que las impresoras marcan el papel depende, entre otras cosas, del fabricante, el mercado que cubre, la tecnología usada, la precisión, y otros factores. No obstante pueden diferenciarse dos grandes soluciones entre las más usadas; punzón único deslizante y múltiples punzones deslizantes.

En la actualidad la mayoría de las impresoras braille hacen uso del mecanismo de múltiples punzones deslizantes, ya que éste método permite una mayor velocidad de impresión (caracteres por segundo), pero como contrapartida esta tecnología encarece

enormemente el producto, ya que los punzones se fabrican a medida y son de alta precisión. En la figura 2.7 se puede observar un cabezal de trece punzones que es usado por las impresoras braille del fabricante *Index Braille*¹⁷.

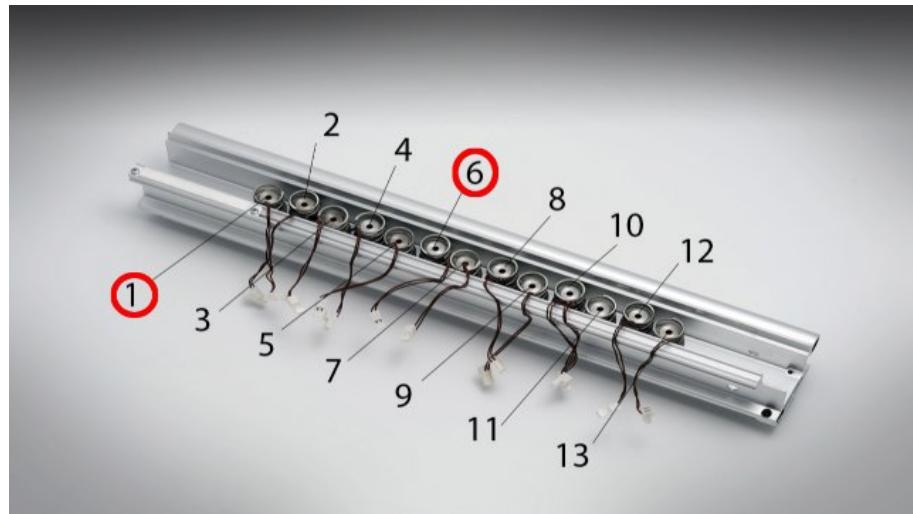


Figura 2.7: Múltiples punzones deslizantes.

El cabezal solo necesita un pequeño desplazamiento horizontal para cubrir el ancho completo de la hoja y generar una secuencia continua de tres filas de puntos. Este mecanismo también permite la impresión de braille en ambos lados de la hoja, intercalando punzones cóncavos y convexos como se aprecia en la figura 2.8.

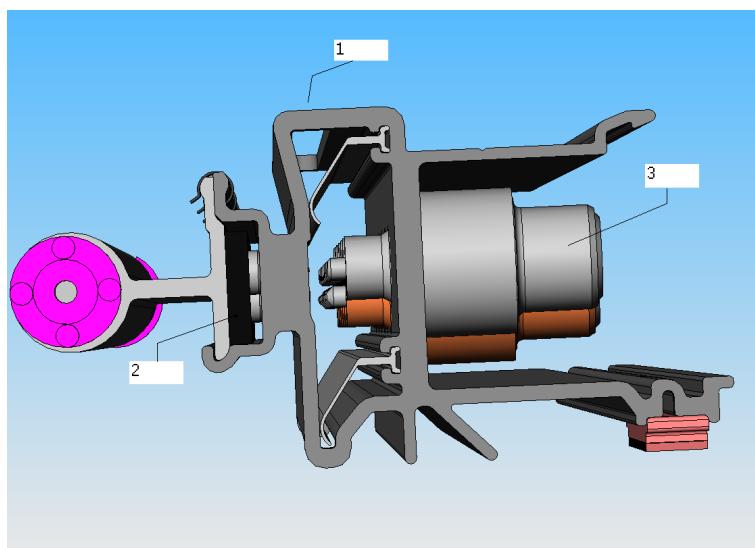


Figura 2.8: Corte cabezal IndexBraille. 1 - Alimentación de papel, 2 - Yunque de goma, 3 - Punzón o martillo

Los punzones grises poseen punta convexa, mientras que los marrones poseen punta

¹⁷Véase - <http://www.indexbraille.com/>

cóncava. Fijos al yunque de goma se encuentra una matriz de punzones que encastran perfectamente con sus respectivos punzones móviles, y es entre éstos que el papel es impactado generándose el relieve hacia un lado o hacia el otro.

El mecanismo de cabezal con un único punzón deslizante no es muy usado en la actualidad debido a que es muy lento, sin embargo es una de las soluciones más económicas. Consiste en un cabezal móvil con un único punzón, el cual se desplaza a lo ancho de la hoja golpeando cada vez que se desea marcar un punto. Una matriz con los puntos marcados en forma cóncava al otro lado del papel hace de guía para que el golpe del punzón marque el papel.

La sencillez de este mecanismo le otorga tanto robustez como versatilidad, junto a un diseño simple de bajo costo de producción y mantenimiento. Todas estas características hacen de este sistema el ideal para implementar un dispositivo económico y sencillo, razón por la cual será el usado en este trabajo.

CAPÍTULO

3

Panorama General

Lo primero a considerar es la forma de comunicación entre la computadora y la impresora, y si bien el diseño debe centrarse en mantener los costos de producción y construcción bajos, de nada sirve un dispositivo basado en tecnología en desuso o pronta a desaparecer. Es por ello que en vez de elegir métodos convencionales como lo pueden ser el protocolo serial R232¹ o paralelo² (típico de impresoras antiguas), se usará el protocolo serie de comunicación *USB*³. Éste estándar se encuentra en la mayoría de los equipos actuales y provee una versatilidad y flexibilidad como pocos.

Otro tema muy importante a considerar es el sistema operativo a usar en la computadora. Si bien puede pensarse como respuesta obvia el sistema operativo de Microsoft, Windows⁴ (en su versión más comercializada *XP*), este enfoque fuerza al potencial usuario a comprar una licencia⁵ solo para poder utilizar la impresora. Entonces en vez de optar por un sistema pago se elige el sistema operativo *GNU/Linux*⁶ del cual existen más de mil versiones distintas, la mayoría de ellas gratuitas. De esta forma el usuario puede obtener una copia de este sistema operativo gratuito y usar la impresora sin mayores restricciones.

En la figura 3.1 se muestra un esquema básico de lo dicho anteriormente. El esquema en su más alto nivel será usar una computadora con *GNU/Linux* como sistema

¹Véase - <http://es.wikipedia.org/wiki/RS-232>

²Véase - http://es.wikipedia.org/wiki/Puerto_paralelo

³El estándar se explica en detalle en la página 43 del capítulo 6.

⁴Véase - <http://www.microsoft.com/spain/windows/>

⁵Dichas licencias rondan los 100 US\$

⁶Véase la página 73 del capítulo 11.1

operativo y una conexión mediante *USB* al dispositivo.

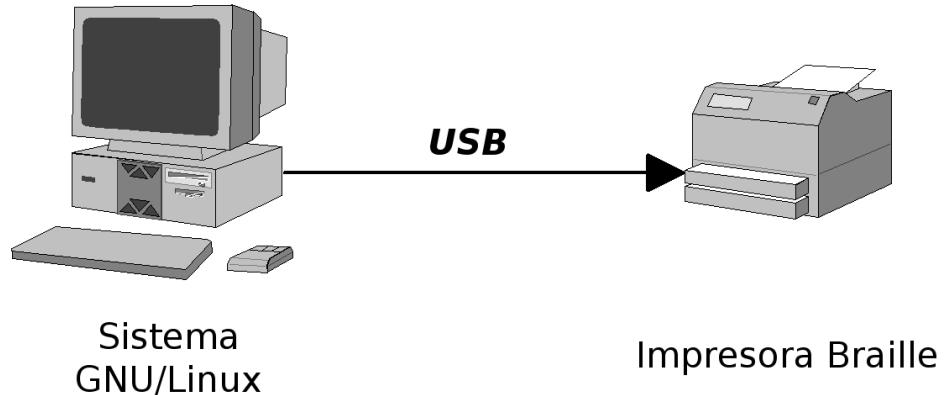


Figura 3.1: S.O, conexión e impresora

Todo el trabajo esta basado en este sencillo esquema. No obstante, y para proveer cierta flexibilidad, cada una de las partes interviniéntes es desglosada en módulos coherentes y finitos.

CAPÍTULO

4

Tecnologías a usar

Para cumplimentar el objetivo de mantener los costos de producción, y del dispositivo final bajos, las tecnologías a usar deben ser cuidadosamente elegidas entre aquellas disponibles en el mercado.

En esta sección se explican brevemente cada una de las tecnologías elegidas y se hace referencia a secciones más detalladas en el caso de que sea necesario.

4.1. Comunicación

Como se mencionó anteriormente es necesario hacer uso de algún método de comunicación entre la computadora y el dispositivo. Existen varias alternativas posibles para ello, a continuación se enumeran alguna de ellas.

- RS232¹
- Puerto paralelo (Centronics²)
- I2C³
- FireWire⁴
- USB⁵

¹Véase - <http://es.wikipedia.org/wiki/RS232>

²Véase - <http://es.wikipedia.org/wiki/Centronics>

³Véase - <http://es.wikipedia.org/wiki/I2C>

⁴Véase - <http://es.wikipedia.org/wiki/Firewire>

⁵Véase - <http://es.wikipedia.org/wiki/USB>

- BlueTooth⁶

Cada uno de los anteriores “métodos” de comunicación poseen diversas características, pero es el estándar USB el único que satisface las siguientes necesidades.

- Velocidad de transmisión
- Tecnología no pronta a desaparecer
- Buena documentación técnica
- Costos de implementación
- Disponibilidad en el mercado
- Flexibilidad a la hora de implementarla
- Facilidad de uso
- Familiarización por parte del usuario

Justamente por ello es el usado en este trabajo. Ya habiendo elegido el estándar USB para lograr la comunicación entre la computadora y el dispositivo, es necesario luego elegir la forma de interactuar con el estándar desde el punto de vista del software. Se presentan dos opciones claras para encarar el problema, hacer uso de las funcionalidades más bajas del sistema operativo, esto sería escribir un “driver”, o bien mediante algún tipo de *API*⁷.

Si bien linux provee todas las herramientas necesarias para escribir drivers (o módulos) USB, este método quitaría toda posibilidad de portar el código a otros sistemas operativos. En cambio existe una API portada a varios sistemas operativos que satisface todas las necesidades para controlar la comunicación USB entre el dispositivo y el kernel.

La API *libusb*⁸ consiste en un set de librerías de código abierto licenciadas bajo LGPL⁹ (GNU Lesser General Public License). Dichas librerías resuelven la comunicación USB pero a nivel de usuario. La API se encuentra en casi todas las distribuciones de GNU/Linux, y se presenta en forma de librería dinámica para usar con aplicaciones ya compiladas y un paquete extra para *development*¹⁰, con los archivos cabecera, programas para compilar, y la documentación completa de sus funcionalidades con algunos ejemplos extras.

La librería provee entre definiciones y estructuras, las siguientes funciones:

⁶Véase - <http://es.wikipedia.org/wiki/Bluetooth>

⁷Del inglés *Application Programming Interface*

⁸Véase - <http://www.libusb.org/>

⁹Véase - <http://www.gnu.org/copyleft/lesser.html>

¹⁰Del inglés desarrollo

```

/* usb.c */
usb_dev_handle *usb_open(struct usb_device *dev);
int usb_close(usb_dev_handle *dev);
int usb_get_string(usb_dev_handle *dev, int index, int langid, char *buf,
size_t buflen);
int usb_get_string_simple(usb_dev_handle *dev, int index, char *buf, size_t
buflen);

/* descriptors.c */
int usb_get_descriptor_by_endpoint(usb_dev_handle *udev, int ep, unsigned char
type, unsigned char index, void *buf, int size);
int usb_get_descriptor(usb_dev_handle *udev, unsigned char type, unsigned char
index, void *buf, int size);

/* <arch>.c */
int usb_bulk_write(usb_dev_handle *dev, int ep, const char *bytes, int size,
int timeout);
int usb_bulk_read(usb_dev_handle *dev, int ep, char *bytes, int size, int
timeout);
int usb_interrupt_write(usb_dev_handle *dev, int ep, const char *bytes,
int size, int timeout);
int usb_interrupt_read(usb_dev_handle *dev, int ep, char *bytes, int size, int
timeout);
int usb_control_msg(usb_dev_handle *dev, int requesttype, int request, int
value, int index, char *bytes, int size, int timeout);
int usb_set_configuration(usb_dev_handle *dev, int configuration);
int usb_claim_interface(usb_dev_handle *dev, int interface);
int usb_release_interface(usb_dev_handle *dev, int interface);
int usb_set_altinterface(usb_dev_handle *dev, int alternate);
int usb_resetep(usb_dev_handle *dev, unsigned int ep);
int usb_clear_halt(usb_dev_handle *dev, unsigned int ep);
int usb_reset(usb_dev_handle *dev);

```

Una particularidad interesante de esta API, es que no solo ha sido portada a varios sistemas operativos, sino que también ha sido portada a varios lenguajes de programación como java, perl, python y otros.

Todas las funciones de esta librería (para la versión estable 0.1) son síncronas, lo que significa que se debe esperar a que termine la operación para poder seguir. Por este motivo la mayoría de las funciones implementan un *timeout* en milisegundos.

Esta API satisface tanto el estándar USB 1.0 como el 2.0, es por ello que si el dispositivo respeta el estándar, establecer una comunicación a USB requiere solo la adición de algunas funciones al código fuente para; inicializar la comunicación, para buscar el dispositivo, para abrirlo y luego el programa en si.

Debido a la complejidad e importancia que tiene USB en este trabajo, se le dedica un capítulo entero¹¹ con toda la información técnica relevante para este trabajo.

4.2. Microcontrolador

El mercado de desarrollo electrónico Argentino tiene una tendencia a usar las soluciones de Microchip¹², y es por ello que existe mucha gente capacitada en los microcon-

¹¹Véase la página 43 del capítulo 6

¹²Véase - <http://www.microchip.com/>

troladores de esta empresa, y si bien sus productos no son los mejores del mercado internacional, sus precios son bastante competitivos al igual que las funcionalidades que prestan y la amplia gama de variedades que poseen.

Como el objetivo de este trabajo se enfoca en el mercado local y el bajo costo de producción, los productos de Microchip se presentan como la mejor alternativa.

El requerimiento primario para elegir el microcontrolador a usar es que posea soporte USB. La familia *18FXXXX*¹³ satisface este requerimiento. De esta familia los dos microcontroladores más comercializados¹⁴ en el país son el *18F2550* y el *18F4550*. La tabla 4.1 muestra las principales características de ambos microcontroladores.

	18F2550	18F4550
Program Memory Type	Flash	Flash
Program Memory (KB)	32	32
CPU Speed (MIPS)	12	12
RAM Bytes	2048	2048
Data EEPROM (bytes)	256	256
Digital Communication Peripherals	1-A/E/USART, 1-MSSP(SPI/I2C)	1-A/E/USART, 1-MSSP(SPI/I2C)
Capture/Compare/PWM Peripherals	2 CCP	CCP, 1 ECCP
Timers	1 x 8-bit, 3 x 16-bit	1 x 8-bit, 3 x 16-bit
ADC	10 ch, 10-bit	13 ch, 10-bit
Comparators	2	2
USB (ch, speed, compliance)	1, Full Speed, USB 2.0	1, Full Speed, USB 2.0
Temperature Range (C)	-40 to 85	-40 to 85
Operating Voltage Range (V)	2 to 5.5	2 to 5.5
Pin Count	28	40

Tabla 4.1: Comparación PIC18F2550 y PIC18F4550

Se observa que no existe gran diferencia entre ambos MCU¹⁵, al igual que sus precios como se observa en la tabla 4.2.

MCU	US\$
18F2550	2.39
18F4550	3.65

Tabla 4.2: Comparación de precios PIC18F2550 y PIC18F4550

16

Se puede concluir entonces que para éste trabajo el PIC18F2550 satisface en grandes rasgos los requerimientos mínimos. No obstante, y por cuestiones de comodidad¹⁷, se usará el PIC18F4550, realizando el diseño con vistas a migrar el MCU al PIC18F2550 en el futuro.

¹³Véase - http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2654

¹⁴Esto implica facilidad a la hora de conseguir en el mercado local.

¹⁵Del inglés *Micro Controller Unit*

¹⁷Esto es disponibilidad a la hora de realizar el proyecto.

4.3. Sistema Operativo

Como se mencionó anteriormente, el sistema operativo elegido es GNU/Linux, esto se debe principalmente a que de esta manera, para poder hacer uso de la impresora, el usuario no se vería obligado a comprar ningún tipo de software, ya que la inmensa mayoría de las versiones de GNU/Linux son completamente gratuitas.

Además de lo anterior, las mayoría de las distribuciones de GNU/Linux proveen una enorme cantidad de aplicaciones para desarrollo, tanto de software como de hardware, otorgando entonces, un ambiente amigable para el desarrollador.

4.4. Compiladores

Para el caso de los compiladores, son necesarios dos; uno para el driver y otro para el firmware.

4.4.1. Driver

Debido a haber elegido GNU/Linux como sistema operativo, la opción más evidente es usar GCC¹⁸ (GNU Compiler Collection). GCC es una colección de compiladores (C, C++, Objective-C, Fortran, Java, y Ada) que se encuentra en todas las distribuciones de GNU/Linux. Para este trabajo en particular, solo se usa GCC como compilador de ANSI C¹⁹ que se integra fácilmente con la API *libusb*.

4.4.2. Firmware

Para el caso del firmware (esto sería el programa alojado en el microcontrolador), el mismo fabricante del microcontrolador elegido provee una serie de compiladores para asembler y C. La desventaja de elegir esta solución es que solo soportan sistemas Windows²⁰, y si bien es posible instalarlo en sistemas GNU/Linux, esto implica esfuerzo extra e innecesario. Debido a la complejidad requerida para implementar el protocolo USB, el usar asembler no es una opción, y el compilador de C de Microchip luego del periodo de evaluación de 60 días, deshabilita ciertas opciones de optimización de código, generando programas más grandes. Esto último no es deseado debido al tamaño limitado de memoria que posee el PIC18F4550, y una licencia de dicho programa ronda los 500 dólares²¹.

¹⁸Véase - <http://gcc.gnu.org/>

¹⁹Véase - http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_C

²⁰Véase - http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en010014&redirects=c18

²¹Véase - <http://www.microchipdirect.com/ProductSearch.aspx?Keywords=SW006011>

SDCC²² (Small Device C Compiler), es un compilador ANSI C libre, con soporte para una amplia variedad de microcontroladores, entre ellos se encuentra toda la familia PIC18 de Microchip. Este compilador se encuentra disponible en la mayoría de las distribuciones GNU/Linux, aunque también ha sido portado para sistemas Windows. Una característica muy interesante de este compilador es el excelente soporte que sus creadores dan en linea mediante listas de correo.

4.5. Simulación y diseño

Existen varios programas disponibles para realizar diseños y simulaciones, en este caso, y aprovechando el haber elegido GNU/Linux como sistema operativo, se buscan alternativas “nativas” a este.

De los proyectos de simulación electrónica libres para GNU/Linux, la *suite gEDA*²³, es el más importante. Entre los programas que vienen con esta suite, se desataca *ngspice*²⁴, un simulador de circuitos que satisface completamente el estándar Spice3²⁵.

Si bien la suite gEDA posee programas para diseño de PCBs, se usa para este trabajo el programa de diseño libre KiCAD²⁶, ya que este permite definir y trabajar por proyectos.

²²Véase - <http://sdcc.sourceforge.net/>

²³Véase - <http://www.geda.org/>

²⁴Véase - <http://ngspice.sourceforge.net/>

²⁵<http://en.wikipedia.org/wiki/SPICE>

²⁶Véase - <http://kicad.sourceforge.net/>

CAPÍTULO

5

Módulos

Para mantener la simplicidad del diseño se utiliza un método modular, de tal manera de independizar las soluciones y focalizar los problemas. Es fácilmente divisible el diseño en dos grandes áreas; procesamiento y potencia.

El módulo de procesamiento se encarga principalmente en establecer la conexión con la computadora, procesar los datos y generar las señales de control. El módulo de potencia es el encargado de manejar los motores, sensores y el punzón.

El mantener ambos diseños independientes provee la flexibilidad necesaria para mantener bajos costos y sencillez de implementación.

5.1. Módulo de potencia

Como muestra la figura 5.6, solo se controlan dos motores, y para mejor precisión y mantener el diseño compatible con impresoras viejas matriz de punto, los motores *paso a paso* son la mejor elección.

Existen dos grandes tipos de motores paso a paso; *bipolares* y *unipolares*. La figura 5.1 muestra la disposición y conexión de los devanados de ambos tipos de motores. Claramente la desventaja que presentan los motores *bipolares* frente a los *unipolares* es que necesitan invertir el sentido de la corriente en sus bobinados, esto implica mayor complejidad en el circuito controlador. Y en contra partida la mayor ventaja de los motores *unipolares* radica (más allá de la simpleza del circuito controlador) en que

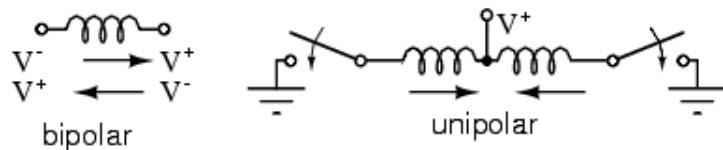


Figura 5.1: Motores paso a paso.

eran los más usados¹ en la antiguas impresoras matriz de punto, lo cual es un beneficio más que importante para este trabajo.

Para satisfacer los dos movimientos mecánicos necesarios (rotación del rodillo de papel y desplazamiento horizontal del cabezal) el diseño requiere controlar dos motores paso a paso unipolares individuales. Entre los motores paso a paso *unipolares*, los más usados son los de cuatro fases, más comúnmente denominados *motor de cinco cables*. La figura 5.2 muestra simbólicamente la distribución interna de los devanados de este tipo de motores.

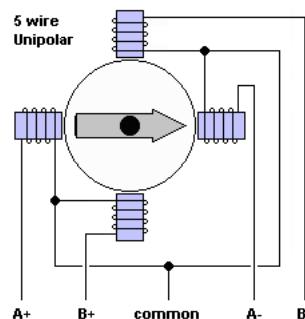


Figura 5.2: Motor unipolar paso a paso de 4 fases.

Para lograr la rotación, cada bobina debe ser energizada de forma independiente y siguiendo una secuencia particular. El circuito de la figura 5.3 satisface los requisitos mínimos para energizar una única bobina.

El divisor resistivo $R2\ R3$ es una resistencia variable, que junto con la resistencia de emisor de $1\ \Omega$ permiten controlar la corriente que circula por la bobina, de esta manera este circuito puede ser usado por diversos motores de diferentes impedancias. Debido a que el circuito de salida se conecta directamente a la bobina del motor, y esta puede ser de muy baja impedancia, el transistor de salida debe ser capaz de manejar corrientes altas, en este caso se hace uso de un *TIP122*. La figura 5.4 muestra las diferentes tensiones que actúan en el circuito.

Un motor paso a paso unipolar de cuatro fases tiene cuatro bobinas, para manejar los dos motores es necesario luego replicar ocho veces este circuito. El esquemático final se encuentra en el anexo en la página 117 del capítulo 11.5.1.

¹Y son más fáciles de encontrar en el mercado local

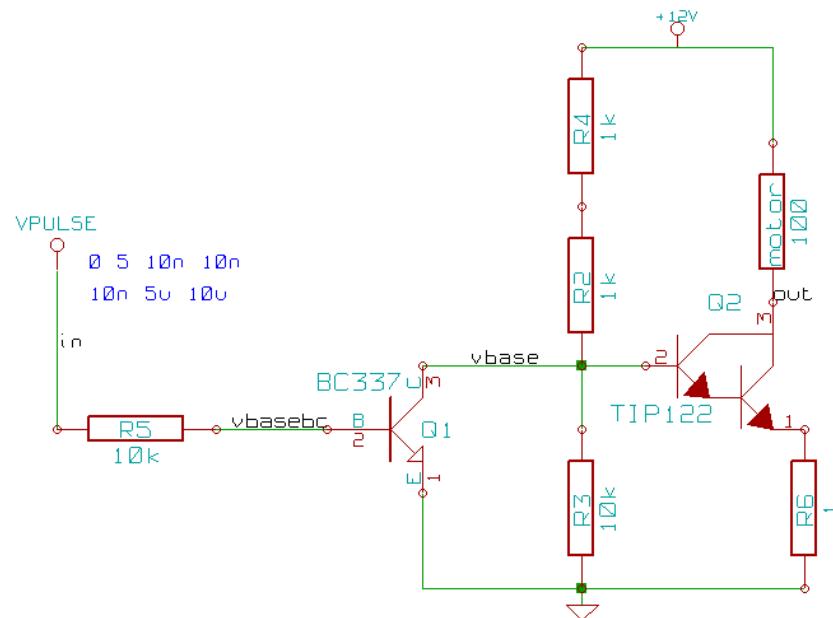


Figura 5.3: Circuito de potencia para una bobina de motor paso a paso unipolar.

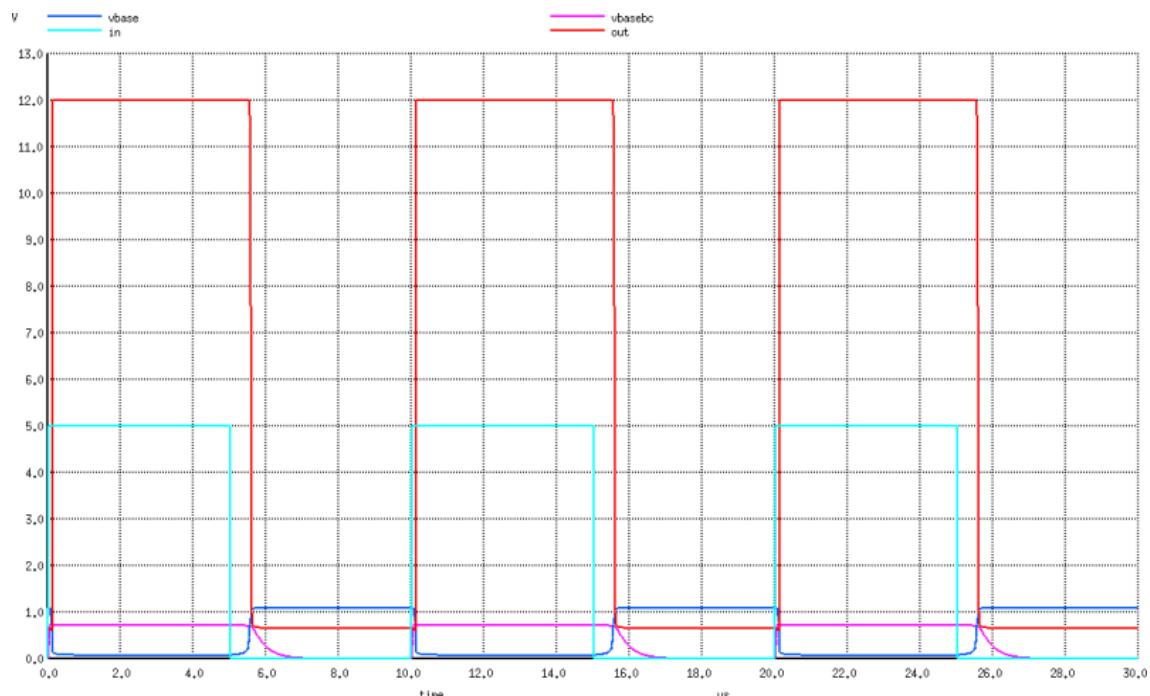


Figura 5.4: Simulación del circuito de potencia para una bobina de motor paso a paso unipolar.

5.2. Modulo de procesamiento

El microcontrolador PIC18F4550 posee un método de grabación en circuito² (In-Circuit Serial Programming). Esta prestación es implementada, no solo para dar dinamismo al desarrollo, sino también para permitir futuras actualizaciones de firmware en las placas de producción. Como se ve en la figura 5.5, la implementación de esta funcionalidad es bastante sencilla.

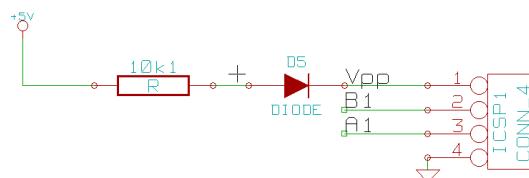


Figura 5.5: Conexión ICSP (In-circuit Serial Programming)

En cuanto a la velocidad de procesamiento, si bien el microcontrolador soporta cristales externos de hasta 20 MHz, se usa uno de 4 MHz, ya que son los más comúnmente usados y en algunos casos más baratos.

El microcontrolador posee varios registros de configuración que deben ser escritos para habilitar ciertas funcionalidades. La configuración usada se aprecia en el siguiente pedazo de código.

```
code char at 0x300000 CONFIG1L = 0x20; /* USB clk 96MHz PLL/2, PLL 4MHz */
code char at 0x300001 CONFIG1H = 0x0e; /* HSPLL */
code char at 0x300002 CONFIG2L = 0x20; /* USB regulator enable, PWRT On */
code char at 0x300003 CONFIG2H = 0x00; /* Watchdog OFF */
code char at 0x300004 CONFIG3L = 0xff; /* ***UNUSED*** */
code char at 0x300005 CONFIG3H = 0x81; /* MCLR enabled, PORTB digital */
code char at 0x300006 CONFIG4L = 0x80; /* all OFF except STVREN */
code char at 0x300007 CONFIG4H = 0xff; /* ***UNUSED*** */
code char at 0x300008 CONFIG5L = 0xff; /* No code read protection */
code char at 0x300009 CONFIG5H = 0xff; /* No data/boot read protection */
code char at 0x30000A CONFIG6L = 0xff; /* No code write protection */
code char at 0x30000B CONFIG6H = 0xff; /* No data/boot/table protection */
code char at 0x30000C CONFIG7L = 0xff; /* No table read protection */
code char at 0x30000D CONFIG7H = 0xff; /* No boot table protection */
```

Si bien no se encuentra dentro del alcance de este trabajo proveer una solución para la parte mecánica de la impresora, es necesario determinar con que debe interactuar la parte electrónica, esto es; cuales serán las entradas y cuales serán las salidas.

Como se determinó anteriormente el mecanismo a usar será de un único punzón deslizante, esto implica que se debe generar movimiento horizontal para el cabezal y movimiento vertical para el papel, es necesaria además una señal para manejar el punzón. Quedan entonces definidas las salidas. Por otra parte es necesario definir algún tipo de señal de entrada. Para mantener un diseño simple solo se usarán entonces una

²Véase - <http://ww1.microchip.com/downloads/en/DeviceDoc/30277d.pdf>

señal de presencia de papel y un fin de carrera para el cabezal.

En la figura 5.6 se ve el diseño general. Éste posee la particularidad de ser el usado por la mayoría de las antiguas impresoras matriz de punto, siendo entonces posible usar un mecanismo fabricado a medida, o bien adaptar una impresora vieja.

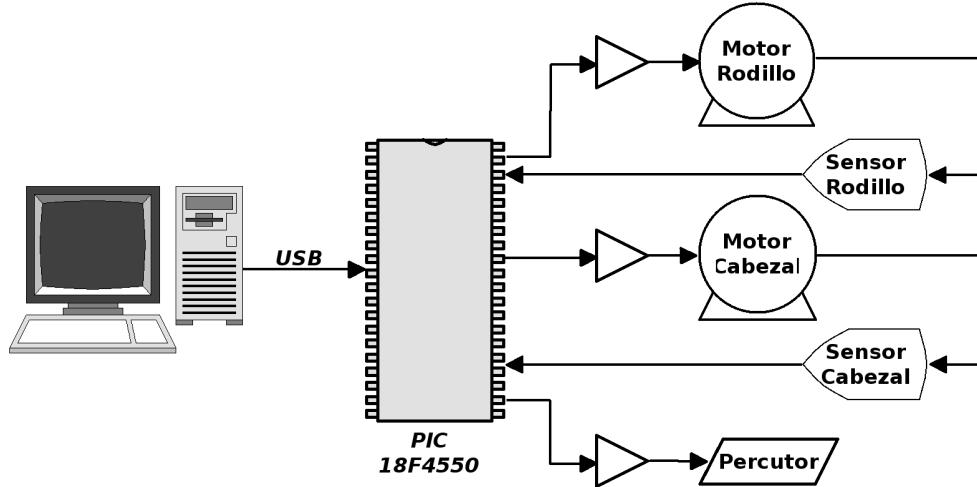


Figura 5.6: Diseño de alto nivel.

Se implementan a modo de protección circuitos opto acoplados para la entrada de los sensores y la salida del percutor. En la figura 5.7 se ve el circuito de la salida del percutor, y en la figura 5.8 el circuito de uno de los sensores. En ambos casos se ha agregado un LED para ayuda visual.

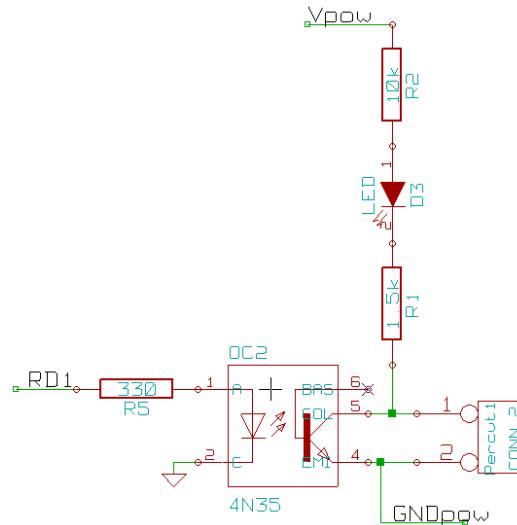


Figura 5.7: Circuito opto acoplado para el percutor

Para mantener los costos bajos, y teniendo en cuenta las limitaciones del microcon-

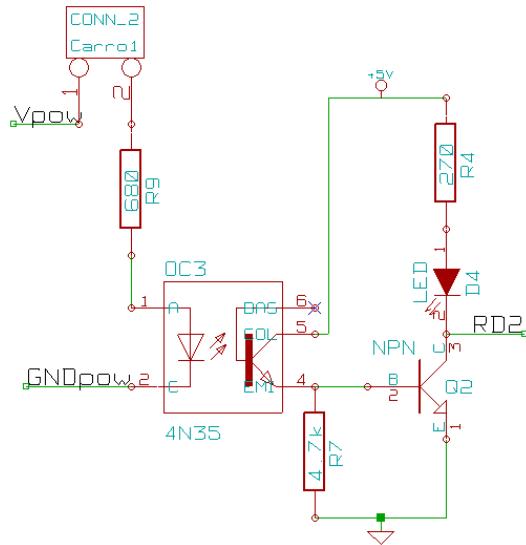


Figura 5.8: Circuito opto acoplado para uno de los sensores

trolador elegido, la mayor parte del procesamiento se lleva a cabo en la computadora, logrando de esta manera economizar espacio en el firmware y evitar la necesidad de hardware extra.

En su mayoría, las impresoras braille comerciales poseen integrado un sistema sintetizador de voz, con mensajes pregrabados en varios idiomas que se reproducen por cada acción que la impresora ejecuta. Esto, claro está, implica la necesidad de grandes cantidades de memoria y quizás un chip dedicado. Para evitar usar chips sintetizadores de voz y memorias, y teniendo en cuenta que la mayoría de las computadoras actuales poseen sistemas de reproducción de audio, éste trabajo queda a cargo de éstas. Al igual que el audio, existe gran cantidad de procesamiento a llevar a cabo para la conversión del texto a braille, y varias decisiones que usualmente toman las impresoras. Todo esto se resuelve del lado de la computadora. Entonces es posible ahora, hablar de driver (que será el programa ejecutándose en la computadora) y firmware (que será el programa ejecutándose en el microcontrolador) al referirnos a la computadora y a el dispositivo respectivamente. Para lograr esto, el firmware solo es capaz de realizar tareas básicas y sencillas, respondiendo a instrucciones que el driver envía y reportando los resultados obtenidos. La figura 5.9 muestra una representación de lo dicho anteriormente.

Para instanciar el modelo anterior haciendo uso del método de comunicación elegido³, se definen dos endpoints⁴ bidireccionales, uno dedicado exclusivamente al envío de datos (endpoint 1 o EP1) y otro para enviar instrucciones y recibir resultados (endpoint 2 o EP2). La figura 5.10 muestra la comunicación mediante endpoints.

³Esto es el estándar USB

⁴Véase la página 49 del capítulo 6.2

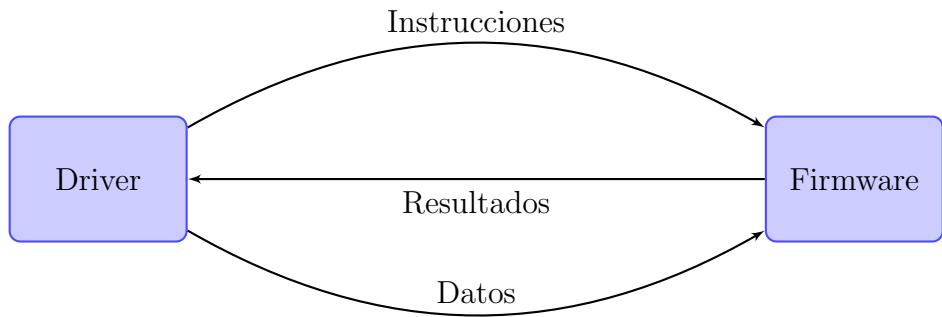


Figura 5.9: Canales de comunicación entre el driver y el firmware.

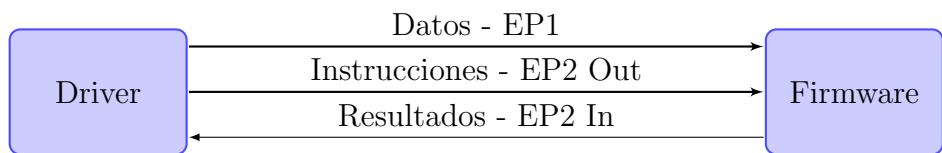


Figura 5.10: Canales de comunicación entre el driver y el firmware mediante endpoints.

Habiendo definido los canales de comunicación y sus respectiva funciones, es necesario determinar el tipo y forma de los datos y las instrucciones a manejar.

Datos

La cantidad de caracteres braille que entran en el ancho de una hoja A4, ronda los 30 dependiendo del tamaño de las sangrías. Para dejar un margen se eligen 28 caracteres. Teniendo en cuenta que los caracteres braille poseen dos columnas de puntos⁵ y que el dato mínimo que se puede enviar por USB es de un byte⁶, para completar una linea a lo ancho de una página de una fila de puntos se necesitan:

$$\begin{aligned} 28 \text{caracteres} * 2 \text{puntos} &= 56 \text{bits} \\ 56 \text{bits} / 8 &= 7 \text{bytes} \end{aligned}$$

••	••	••	••	••	••	••	...
..	
byte 1		byte 2		...			

Tabla 5.1: Bytes por puntos braille

Entonces el máximo numero de bytes a enviar por linea es de 7 bytes.

Para formar una fila completa de caracteres, es necesario realizar tres envíos, ya que los caracteres braille poseen tres filas de puntos⁷, lo cual genera 21 bytes ($7 \text{bytes} * 3$)

⁵Véase la página 13 del capítulo 2.1

⁶Véase la página 60 del capítulo 6.7.3

⁷Véase la página 13 del capítulo 2.1

por fila, y un total de 588 ($21\text{bytes} * 28\text{filas}$) bytes por página ya que en una hoja A4 se suelen imprimir 28 filas de caracteres.

Instrucciones

Para limitar la capacidad de funcionalidades del dispositivo solo se definen ocho instrucciones básicas descriptas en la tabla 5.2.

Instrucción	Byte	Descripción
RESET	0x01	Resetea el cabezal al punto de origen
PRINT	0x02	Imprime los datos
MOV_SHORT	0x03	Desplaza el cabezal entre puntos
MOV_LONG	0x04	Desplaza el cabezal entre caracteres
ROT_SHORT	0x05	Rota el rodillo entre lineas
ROT_LONG	0x06	Rota el rodillo entre caracteres
PULL_PAPER	0x07	Rota el rodillo hasta sacar el papel
STATUS	0x08	Pide el estado de los sensores

Tabla 5.2: Instrucciones de impresión

Este set de instrucciones proporcionan todas las funcionalidades necesarias para poder llevar a cabo un proceso de impresión normal.

Resultados

Se definen dos tipos de resultados distintos, uno que es devuelto cuando se envía la instrucción *STATUS* y otro que se genera siempre luego de un envío de datos que consta en el dato recreado por la impresora, esto es solo a modo de verificación tipo *echo*.

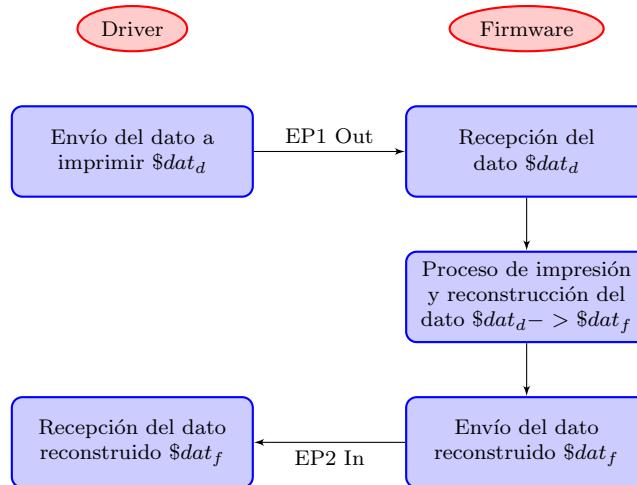
Ante una instrucción del tipo *STATUS*, el dispositivo lee los estados de los dos sensores y genera un byte de reporte descripto en la tabla 5.3.

Byte							
x	x	x	x	x	x	1/0	1/0
-	-	-	-	-	-	Sensor de cabezal	Sensor de papel

Tabla 5.3: Byte de reporte

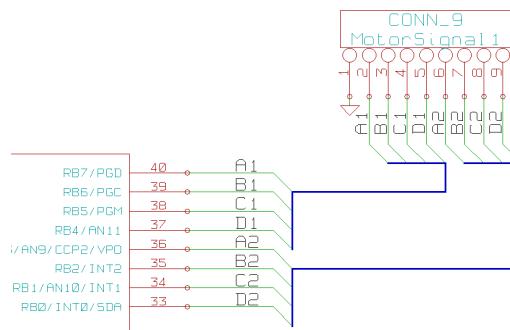
Entonces dependiendo del estado de los dos bits menos significativos del byte de reporte, el driver puede saber en que estado se encuentra la impresora.

El segundo tipo de resultado definido es el *echo*. Cuando el driver envía datos para imprimir, la impresora comienza el proceso de impresión y recrea los datos recibidos según lo que hace, luego este dato recreado se envía de vuelta al driver quien comprueba ambos datos y verifica que sean iguales, la figura 5.11 muestra un diagrama simplificado de este proceso.

**Figura 5.11:** Proceso de reconstrucción de datos

5.2.1. Manejo de motores

Como se explica en la página 31 del capítulo 5.1, se requieren dos motores, lo cual implica tener a disposición ocho señales de control. Para ello se hace uso del puerto B del microcontrolador, el cual posee la particularidad de estar provisto de resistencias de *pull-up* programables. De esta manera los dos motores pueden controlarse mediante la escritura de un único puerto. La figura 5.12 muestra la conexión del puerto B del microcontrolador a la ficha de señales de los motores.

**Figura 5.12:** Puerto B del uC conectado a las señales de control de los motores paso a paso.

Teniendo en cuenta la figura 5.2, y tomando el valor '1' como “energizado”, las secuencias de la tabla 5.4, indistintamente, efectúan la rotación de el motor en un paso completo.

Es posible combinar ambas secuencias para obtener incrementos de medio paso como se ve en la tabla 5.5, de esta manera se obtiene una mejor precisión en la rotación del motor.

Luego con el sencillo código que se muestra a continuación se logra manejar ambos motores independientemente.

Bobina A+	1	0	0	0
Bobina B+	0	0	1	0
Bobina A-	0	1	0	0
Bobina B-	0	0	0	1

Bobina A+	1	1	0	0
Bobina B+	0	0	1	1
Bobina A-	0	1	1	0
Bobina B-	1	0	0	1

Tiempo ...

Tabla 5.4: Secuencias de un paso para motores paso a paso unipolares de cuatro fases.

Bobina A+	1	1	0	0	0	0	0	1
Bobina B+	0	0	0	1	1	1	0	0
Bobina A-	0	1	1	1	0	0	0	0
Bobina B-	0	0	0	0	0	1	1	1

Tiempo ...

Tabla 5.5: Secuencia de medio paso para motores paso a paso unipolares de cuatro fases.

```


/**
 * move() -      Function to move both motors
 * @loops:        Number of loops of a complete sequence
 * @direction:   Direction to move (RIGHT/LEFT, UP/DOWN)
 * @motor:        Which motor to move (CAR/ROLLER)
 *
 * This function moves each motor in a defined direction and a specific
 * amount of turns depending on the parameters that are passed to it.
 */
void move(byte loops, byte direction, byte motor)
{
    byte stepsI[8] = {0x77, 0x33, 0xbb, 0x99, 0xdd, 0xcc, 0xee, 0x66};
    byte stepsD[8] = {0x66, 0xee, 0xcc, 0xdd, 0x99, 0xbb, 0x33, 0x77};
    byte val, i, loops_aux;

    if (direction)
        for (loops_aux = 0; loops_aux < loops; loops_aux++) {
            for (i = 0; i < 8; i++) {
                val = stepsI[i];
                if (motor) {
                    PORTB = (PORTB & 0xf0) | (val & 0x0f);
                    delay(50);
                } else {
                    PORTB = (PORTB & 0x0f) | (val & 0xf0);
                    delay(50);
                }
            }
        }
    else
        for (loops_aux = 0; loops_aux < loops; loops_aux++) {
            for (i = 0; i < 8; i++) {
                val = stepsD[i];
                if (motor) {
                    PORTB = (PORTB & 0xf0) | (val & 0x0f);
                    delay(50);
                } else {
                    PORTB = (PORTB & 0x0f) | (val & 0xf0);
                    delay(50);
                }
            }
        }
}


```

5.3. Driver

Para comenzar una comunicación USB con la API *libusb* es preciso primero descubrir el dispositivo, esto se logra con:

```


struct usb_bus *busses;

usb_init();
usb_find_busses();
usb_find_devices();

busses = usb_get_busses();


```

Luego de esto ya se poseen los *busses* USB del sistema. Es preciso luego iterar sobre cada uno de ellos para encontrar el dispositivo específico de la siguiente manera:

```


struct usb_bus *bus;
int c, i, a;


```

```

/* ... */

for (bus = busses; bus; bus = bus->next) {
    struct usb_device *dev;

    for (dev = bus->devices; dev; dev = dev->next) {
        /* Buscar el dispositivo por VendorID */
        if (dev->descriptor.idVendor == MY_ID) {
            /* Buscar el dispositivo por ProductID */
            if (dev->descriptor.idProduct==USBPRINTER){
                /* Abrir dispositivo */
                udev = usb_open(dev);
                /* Reclamar la interfaz */
                ret = usb_claim_interface(udev,0);

                /* Programa */
            }
        }
    }
}

```

Una limitación intrínseca de la librería es que se requiere abrir tantas instancias del dispositivo como interfaces se desee usar de él.

Luego de esto, ya se ha inicializado la comunicación con el dispositivo USB, resta solo hacer uso de las funciones que la API provee para comunicarse con el mismo; estas son:

```

int usb_bulk_write(usb_dev_handle *dev, int ep, const char *bytes, int size,
int timeout);
int usb_bulk_read(usb_dev_handle *dev, int ep, char *bytes, int size, int
timeout);

```

Estas dos funciones son la únicas necesarias para llevar a cabo las tareas específicas que debe cumplir el driver.

También como parte del trabajo se desarrolló un modulo para manejo de errores que hace uso de un pequeño sintetizador de voz⁸ para decir los mensajes por los parlantes de la computadora. Este modulo es capaz de manejar varios idiomas. El código del mismo se encuentra en el apéndice en la página 111 del capítulo 11.4.6.

⁸Véase - <http://espeak.sourceforge.net/>

CAPÍTULO

6

USB

El *Universal Serial Bus*(USB¹) es un bus serial estándar que hace de interfaz entre un dispositivo y una computadora. Dicha estandarización es llevada acabo por el *Fórum de Implementadores de USB* (USB Implementers Forum, USB-IF²). Actualmente el estándar se encuentra en su versión 3.0³, aunque la mayoría de las implementaciones comerciales solo soportan el estándar 2.0.

¹Véase - <http://www.usb.org>

²Véase - <http://www.usb.org/about>

³Véase - <http://www.usb.org/developers/docs/>

6.1. Descripción

La especificación USB provee una serie de atributos con los cuales se pueden implementar dispositivos según el precio/rendimiento deseado.

6.1.1. Características

Del amplio rango de características definidas en el estándar, es posible agruparlas en ocho categorías:

- Facilidad de uso para el usuario final
- Amplio rango de aplicaciones
- Ancho de banda isócrono
- Flexibilidad
- Robustez
- Sinergia con la industria de la PC
- Implementación de bajo costo
- De arquitectura actualizable

La combinación de estas características hacen a la versatilidad del estándar, ya que permiten implementar todo tipo de dispositivos según la carga, versatilidad, eficiencia, precio, velocidad y demás atributos que puedan inferir en un diseño.

Para la versión 2.0 de la especificación, el protocolo soporta tres velocidades de transmisión como se muestra en la tabla 6.1.

High Speed	480 Mbits/seg
Full Speed	12 Mbits/seg
Low Speed	1.5 Mbits/seg

Tabla 6.1: Velocidades de USB

6.1.2. Arquitectura

USB es un *bus* cableado que soporta conexiones entre un *host* y gran rango de periféricos. Dicho *bus* permite la conexión, configuración, uso y desconexión de un dispositivo mientras el *host* y demás periféricos están en funcionamiento.

Un sistema USB se define según tres grandes áreas:

- Interconexión USB
- Dispositivo USB
- *Host* USB

Características eléctricas

USB usa cuatro cables para alimentación y señal. La señal viaja a través de un par trenzado diferencial y se encuentra codificada con Non Return to Zero Inverted (NRZI). En la figura 6.1 se muestra un diagrama de un cable USB.

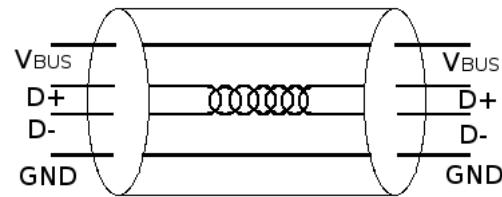


Figura 6.1: Cable USB.

Al ser un par diferencial, el host interpreta un *1* diferencial cuando D+ es 200 mV mayor que D- y un *0* diferencial cuando D- es 200 mV mayor que D+. Los dispositivos USB indican su velocidad mediante una resistencia de *pull-up* a 3.3 V. En el caso de un *full speed device* se coloca una resistencia de 1.5K en D+ a 3.3 V, y para *low speed device* se coloca una resistencia de 1.5K a 3.3 V, como se muestra en la figura 6.2. Algunos fabricantes suelen integrar estas resistencias de *pull-up* dentro de sus *chips* para ser activadas mediante software.

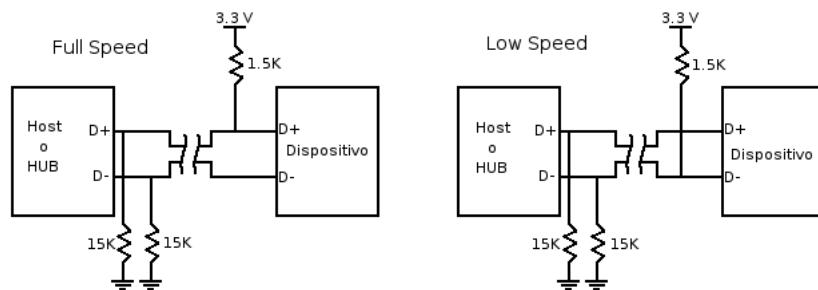


Figura 6.2: Velocidad USB.

6.1.3. Interconexión USB

La interconexión USB define la manera en la que los dispositivos USB se conectan entre si y con el *host*.

Topología del bus

USB usa una topología de estrella por niveles. Un *hub*⁴ está ubicado al centro de cada estrella. Las conexiones cableadas se dan entre el *host* y un *hub* o función, entre *hub* y *hub*, o entre un *hub* y una función.

Debido a cuestiones de latencia, el número máximo de niveles permitido es siete, incluida la raíz. La figura 6.3 muestra un esquema de la topología.

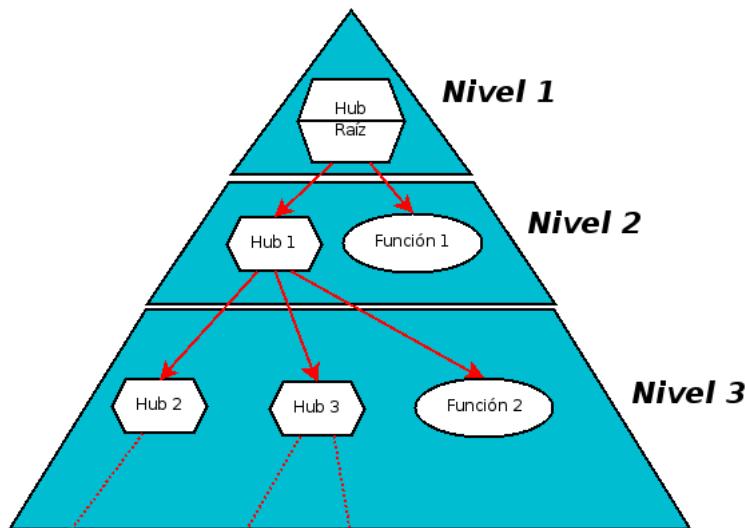


Figura 6.3: Topología del bus USB.

6.1.4. Host USB

La especificación admite un solo *Host* para un todo sistema USB. La interfaz USB del *host*, se la denomina *Host Controller*⁵. La implementación de un *Host Controller* puede ser una combinación de hardware, firmware o software. El *hub* raíz esta integrado en el sistema *host*, proveyendo puntos de acceso. En la versión 1.1 de la especificación, existían dos tipo de controladores de *hosts*; *Universal Host Controller Interface* (UHCI)⁶ y *Open Host Controller Interface* (OHCI)⁷. Luego con para la versión 2.0 de la especificación definió un tercer controlador; *Enhanced Host Controller Interface* (EHCI)⁸

⁴La palabra *hub* se traduce al castellano como *centro*, pero su traducción no sera usada en este documento por cuestiones prácticas.

⁵Al igual que con *host*, esta palabra se mantendrá en inglés.

⁶Desarrollada por Intel

⁷Desarrollada por Compaq, Microsoft y National Semiconductors

⁸Que luego pasaría a convertirse en el estándar más usado.

6.1.5. Dispositivo USB

Los dispositivos USB pueden ser dispositivos funcionales o bien *hubs*. En ambos casos deben ser capaces de entender el protocolo USB y responder a peticiones estándares USB además de cumplir su función intrínseca.

El protocolo USB define capas de abstracción, cada cual con una funcionalidad específica. Normalmente los *chips* manejan las capas más bajas, para abstraer al usuario de todo lo que ocurre en lo más bajo del protocolo.

Una transacción USB consiste de los siguientes paquetes:

- Token (paquete cabecera)
- Data (opcional)
- Status (handshaking)

Los paquetes USB tienen ciertos campos comunes:

- Sync: Es un campo de sincronismo y todos los paquetes deben comenzar con uno.
- PID: Es el identificador de paquete que informa el tipo de paquete que es.
- ADDR: Es la dirección a la cual esta asignado el paquete
- ENDP: Este campo de 4 bits permite direccionar hasta 16 *endpoints*.
- CRC: Lleva la información de un chequeo de redundancia cíclica de los datos
- EOP: Este campo indica la finalización del paquete.

Paquete *Token*

Los paquetes *token* son usados para indicar el tipo de transacción que se realizará. Existen tres tipos de paquetes *token*:

- IN - Informa al dispositivo USB que el host desea leer información
- OUT - Informa al dispositivo USB que el host desea escribir información
- Setup - Es usado para comenzar transferencias de control

Un paquete *token* esta formado por los siguientes campos mostrados en la tabla 6.2

Sync	PID	ADDR	ENDP	CRC5	EOP
------	-----	------	------	------	-----

Tabla 6.2: Campos Token

Sync	PID	DATA	CRC16	EOP
------	-----	------	-------	-----

Tabla 6.3: Campos Data

Paquete *Data*

Un paquete *data* esta formado por los siguientes campos mostrados en la tabla 6.3

Paquete *Status*

Los paquetes *status* son usados para indicar el tipo de estado de la transacción. Existen tres tipos de paquetes *status*:

- ACK - Reconocimiento de que el paquete fue recibido.
- NAK - Avisa de que el dispositivo no puede recibir ni enviar datos.
- STALL - Significa que el dispositivo necesita la intervención del host.

Un paquete *status* esta formado por los siguientes campos mostrados en la tabla 6.4.

Sync	PID	EOP
------	-----	-----

Tabla 6.4: Campos Status

6.2. Endpoints

Un aspecto muy importante del estándar USB es la forma en la que se logra la comunicación entre el host y el dispositivo. Un *endpoint* es un buffer único que define la punta extrema de un canal de comunicación. La identificación de un endpoint en particular se lleva a cabo mediante un campo de 4 bits permitiendo un máximo de 16 endpoints. Cada endpoint funciona como emisor o receptor de datos como se ve en la figura 6.4.

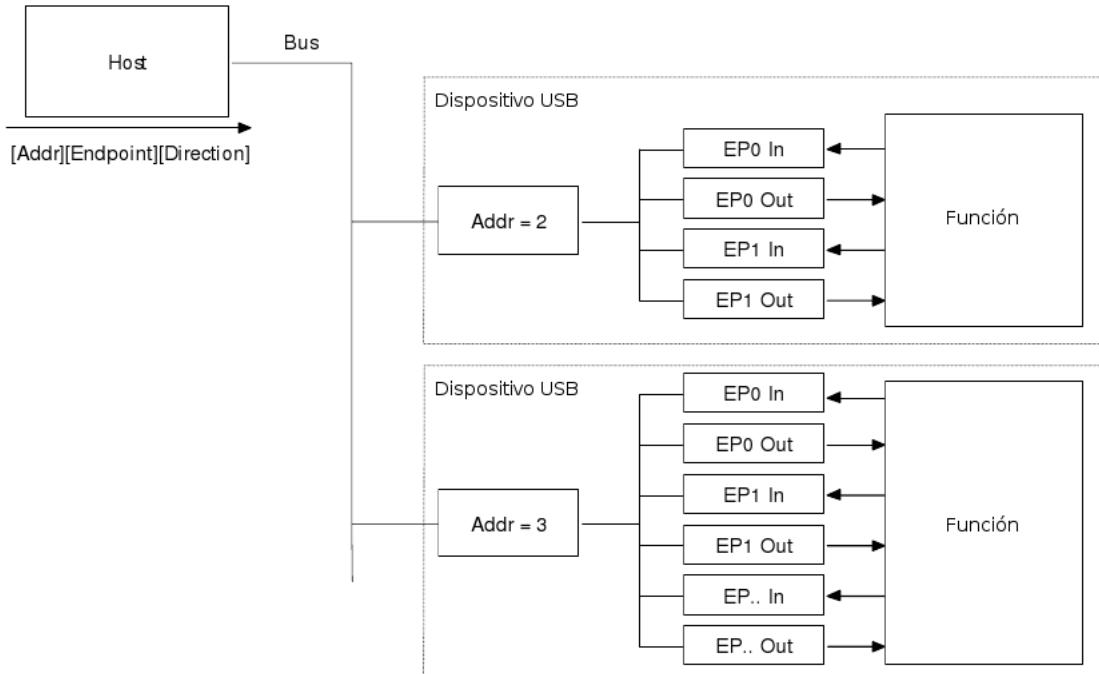


Figura 6.4: Endpoints de USB.

Entonces si por ejemplo el host emite un pedido de descripción de dispositivo (*device descriptor request*) el dispositivo va a determinar por medio de el campo *ADDR* si el pedido esta dirigido hacia él, y luego copiará el dato en el endpoint correspondiente al que indica el campo *ENDP*, que luego podrá ser leído por la aplicación embebida en el dispositivo.

El endpoint 0 debe existir en todos los dispositivos, ya que cumple la función específica de realizar el *handshaking* y todo el control en una comunicación USB.

6.3. Transferencias USB

La especificación USB define cuatro tipos de transferencia:

- Control
- Interrupt
- Isochronous
- Bulk

Debido a el alcance de éste trabajo solo se desarrollarán las transferencias *Control* y *Bulk*, que fueron las usadas durante el desarrollo del proyecto.

6.3.1. Control

Las transferencias de control son usadas para petición y reporte de estados o emisión de ordenes. Este tipo de transferencia posee tres etapas distintas:

- Setup: Durante esta etapa se realiza la petición de datos.
- Data: En esta etapa se realiza el intercambio de datos y puede consistir de varias transferencias *IN* o *OUT* según corresponda.
- Status: Esta etapa se realiza al finalizar la transferencia de control y determina el éxito o no de la operación.

6.3.2. Bulk

Las transferencias tipo *bulk*, son usadas para transmisión de grandes pedazos de datos. Éste tipo de transferencias provee corrección de datos con un campo *CRC* de 16 bits y un mecanismo de detección y re-trasmisión de errores para asegurar la integridad de los mismos.

Las transferencias bulk no poseen un ancho de banda asignado, por lo cual no aseguran latencia alguna.

Las transacciones bulk pueden ser de tipo *IN* o *OUT* como se ve en la figura 6.5.

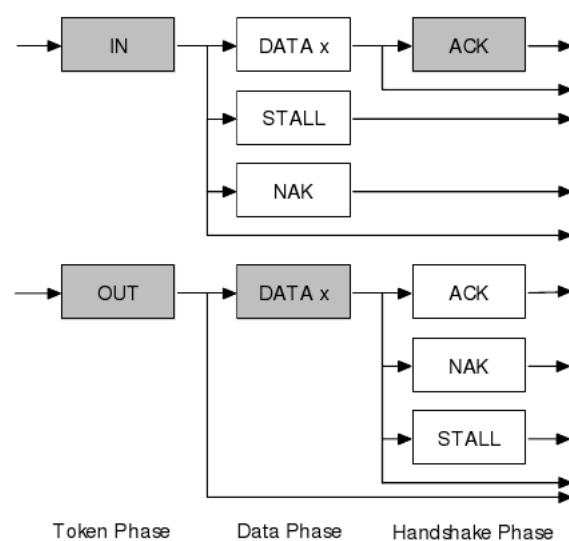


Figura 6.5: Transacción Bulk.

6.4. Descriptores USB

Los descriptores USB son estructuras de datos con formatos específicos, en los cuales se almacena todos los atributos del dispositivo. Cada descriptor comienza con un campo donde se almacena el tamaño de dicho descriptor, seguido inmediatamente por un campo que identifica el tipo de descriptor.

Entre los descriptores más comunes se encuentran:

- Device: Cada dispositivo tiene un solo descriptor *device*, y éstos poseen información básica sobre él. Entre otras cosas este descriptor tiene un número identificador del vendedor y el dispositivo en si. Tiene además un campo con la información de cuantas configuraciones soporta el dispositivo.
- Configuration: Este descriptor posee entre otras cosas el tipo de alimentación y el número de interfaces configuradas del dispositivo.
- Interface: Este descriptor puede ser visto como una cabecera con información sobre un grupo de endpoints.
- Endpoint: Este descriptor posee toda la información necesaria para caracterizar cada endpoint. Entre otras cosas aquí se define el tipo de transferencia que usa el endpoint (bulk, interrupt, isochronous, control), el tamaño de los paquetes, etc. El endpoint 0 siempre es considerado de control y debe estar configurado.
- String: Estos descriptores poseen información legible por humanos que pueden ser indexados por otros descriptores.

Los descriptores pueden ser ordenados jerárquicamente según se ve en la figura 6.6

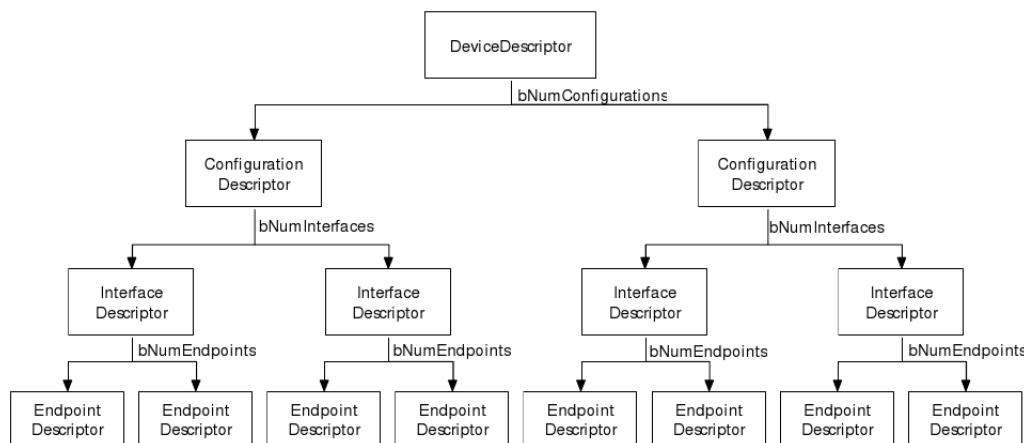


Figura 6.6: Descriptores USB.

6.5. Estados USB

Los dispositivos USB poseen varios estados definidos por los cuales pueden transicionar durante su uso. El diagrama de estados completo de un dispositivo USB puede verse en la figura 6.7.

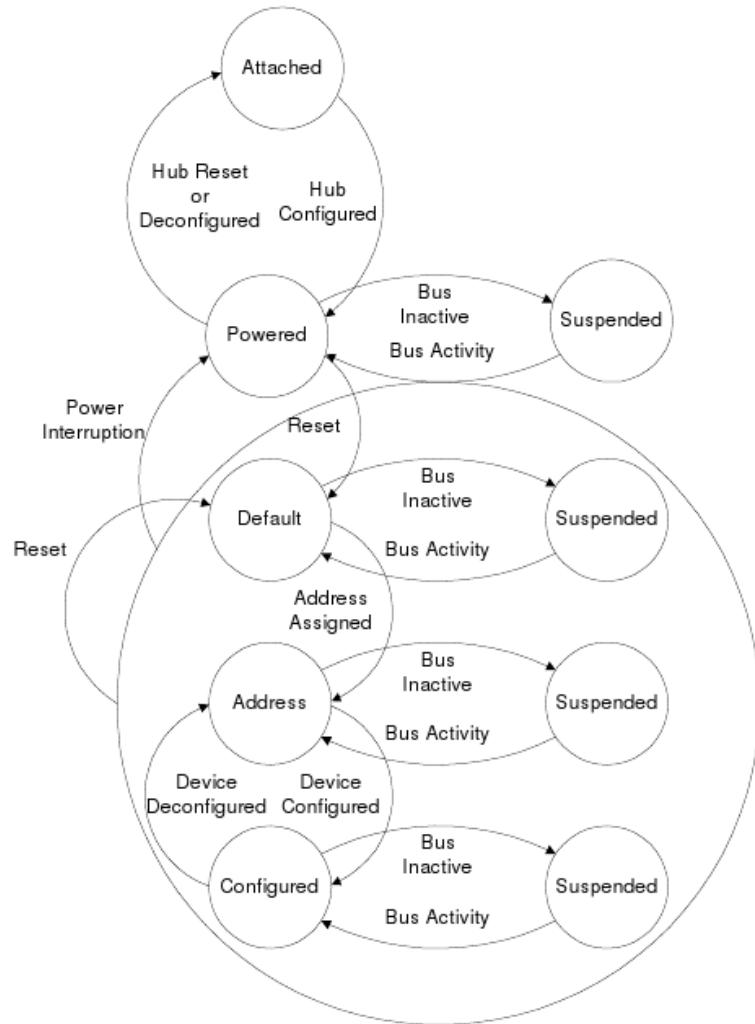


Figura 6.7: Estados USB.

En la tabla 6.5, se puede apreciar una breve explicación de cada estado

Attached	Powered	Default	Address	Configured	Suspended	Descripción
No	–	–	–	–	–	El dispositivo no está enchufado
Si	No	–	–	–	–	El dispositivo está enchufado pero no alimentado.
Si	Si	No	–	–	–	El dispositivo se encuentra enchufado y alimentado, pero no ha sido reseteado.
Si	Si	Si	No	–	–	El dispositivo se encuentra enchufado, alimentado y ha sido reseteado, pero no se le ha asignado una dirección única.
Si	Si	Si	Si	No	–	El dispositivo está enchufado, alimentado, ha sido reseteado y se le ha asignado una dirección única, pero no ha sido configurado.
Si	Si	Si	Si	Si	No	El dispositivo está enchufado, alimentado, ha sido reseteado, se le ha asignado una dirección única, ha sido configurado y no está suspendido. El host puede hacer uso del dispositivo ahora.
Si	Si	–	–	–	Si	El dispositivo se encuentra como mínimo enchufado y alimentado, pero no ha presentado actividad alguna en los últimos 3 ms, por lo que el dispositivo se encuentra suspendido y no puede ser usado por el host.

Tabla 6.5: Estados USB.

6.6. Enumeración del bus USB

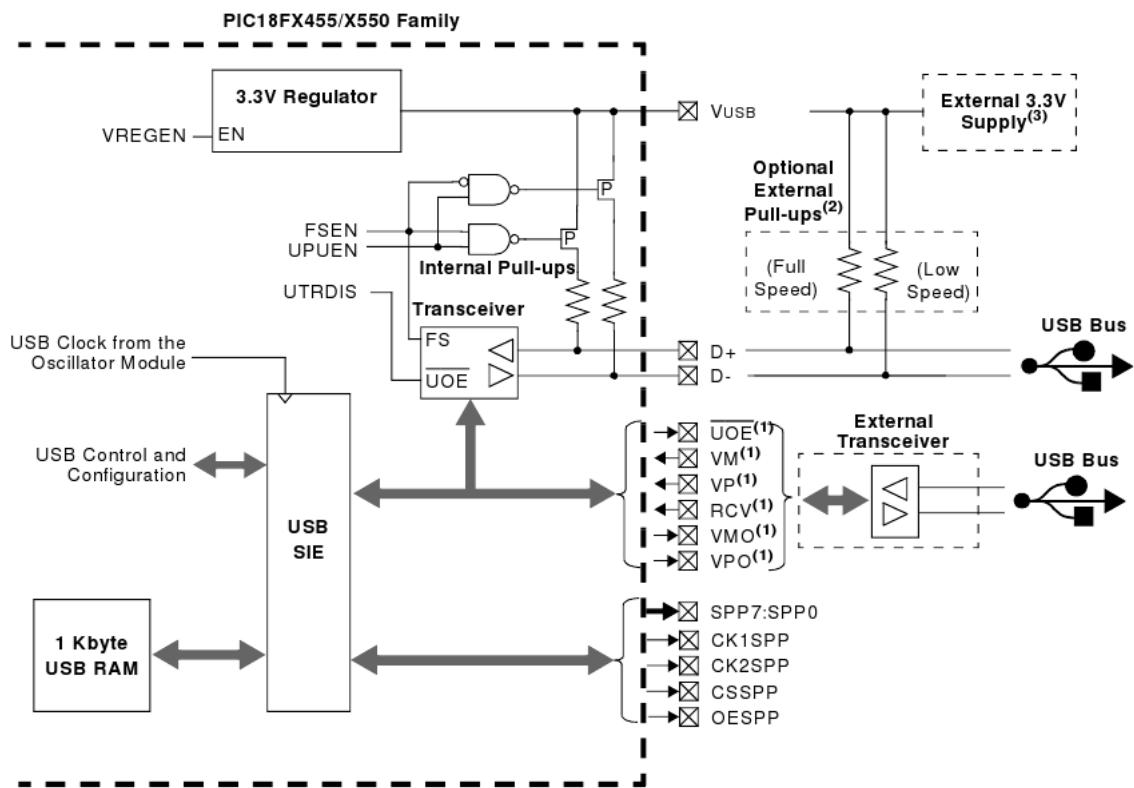
El proceso de enumeración determina si un dispositivo se ha conectado al bus, a que puerto, sus configuraciones, interfaces, etc. A continuación se listan los acciones que se llevan a cabo durante el proceso de enumeración.

1. El hub a el cual el dispositivo esta conectado informa al host del evento. En este paso el dispositivo se encuentra *POWERED* y el puerto al cual esta enchufado se encuentra deshabilitado.
2. El host determina la naturaleza del cambio consultándole al hub.
3. El host conoce ahora el puerto al cual se ha conectado el dispositivo y espera 100 ms para que se normalice su alimentación. El host habilita el puerto y envía una petición de *reset*.
4. El hub ejecuta el pedido de *reset* para ese puerto y luego el puerto queda habilitado. El dispositivo pasa a el estado *DEFAULT* y no puede consumir más de 100 mA. Todos sus registros han sidos reseteados y responde a la dirección por defecto.
5. El host asigna una dirección única al dispositivo y esta pasa al estado *ADDRESS*.
6. El host lee del dispositivo el tamaño de datos que puede recibir antes de que éste reciba la dirección.
7. El host lee toda la configuración de dispositivo.
8. Basado en la configuración obtenida, el host asigna un valor de configuración al dispositivo y éste pasa a el estado de *CONFIGURED*. Desde el punto de vista del dispositivo éste se encuentra listo para ser usado.

6.7. PIC18F4550 USB

La familia de microcontroladores PIC18FX455/X550 de Microchip provee conexión USB de tipo *full* y *high-speed*. Dicha comunicación se lleva acabo mediante lo que Microchip denomina *USB Serial Interface Engine (USB SIE)* o simplemente *SIE*. El *SIE* es capaz de interactuar con un transceptor tanto externo como interno. Y se provee también un regulador de voltaje de 3.3V interno completamente integrado para alimentar el transceptor.

En la figura 6.8 se muestra un diagrama interno de la arquitectura USB de ésta familia de microcontroladores.



Note 1: This signal is only available if the internal transceiver is disabled (UTRDIS = 1).

2: The internal pull-up resistors should be disabled (UPUEN = 0) if external pull-up resistors are used.

3: Do not enable the internal regulator when using an external 3.3V supply.

Figura 6.8: Arquitectura USB de la familia PIC18FX455/X550.

6.7.1. Registros

El modulo USB es controlado mediante 22 registros disponibles en el microcontrolador, la mayoría de los cuales son de lectura y escritura. A continuación se describirán en brevedad cada registro implicado en éste trabajo.

UCON

El registro *UCON*⁹ posee los bits necesarios para controlar el modulo USB durante las transferencias. De aquí se controlan:

- Activar o desactivar el periférico USB
- Reset del puntero del buffer Ping-Pong
- Control del modo *suspend*
- Activar o desactivar la transferencia de paquetes

UCFG

El registro *UCFG*¹⁰ contiene los bits necesarios para determinar el comportamiento del modulo USB. Dicho registro controla:

- Velocidad del bus (*Full/Low speed*)
- Activar o desactivar resistencias internas de Pull-Up
- Activar o desactivar el tranceptor interno
- Uso del buffer de ping-pong

USTAT

El registro *USTAT*¹¹ contiene la información específica de la última transferencia recibida. De éste registro se puede leer:

- Información de transacciones en el buffer de ping-pong
- Número de endpoint en el que se realizó la última transacción
- Dirección de la ultima transacción (*IN o OUT 'token'*)

Éste registro posee un stack del tipo *FIFO* de cuatro niveles que permite al *SIE* procesar otras transacciones en otros endpoints mientras el microcontrolador procesa la información que se encuentra actualmente en el registro *USTAT*, como se puede apreciar en la figura 6.9.

⁹Para más detalles referirse a la hoja de datos del dispositivo página 164

¹⁰Para más detalles referirse a la hoja de datos del dispositivo página 165

¹¹Para más detalles referirse a la hoja de datos del dispositivo página 168

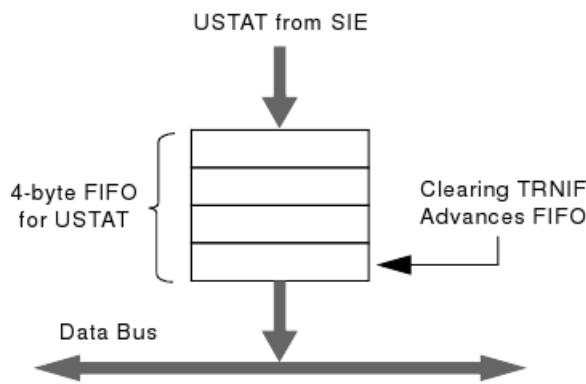


Figura 6.9: Stack FIFO del registro USTAT.

UEP_n

Los registros *UEP_n*¹² contiene la configuración de cada uno de los 16 endpoints. La letra 'n' representa el número del endpoint. Mediante estos registros se puede configurar, para cada endpoint individualmente, los siguientes parámetros:

- Activar o desactivar el *handshaking*
- Activar o desactivar las transferencias de control (*SETUP*)
- Activar o desactivar la dirección (*IN o OUT*)
- Poner el endpoint en modo *STALL*

UADDR

El registro *UADDR*¹³ contiene la dirección única que el periférico decodificará una vez activado. Es el microcontrolador quien debe encargarse de escribir la dirección obtenida durante el proceso de enumeración.

6.7.2. USB RAM

Los datos USB se mueven entre el núcleo del microcontrolador y el *SIE* mediante un segmento de memoria compartida denominado *USB RAM*. Este segmento de memoria de puerto dual se encuentra mapeado en espacio de memoria de datos normal ubicada en el Banco 4 en la dirección 7 (0x400 hasta 0x7FF) por un total de 1Kb como se observa en la figura 6.10.

De ésta memoria, el Banco 4 (400h a 4FFh), esta dedicada exclusivamente al control individual de cada endpoint, quedando disponible para uso de datos de USB la parte

¹²Para más detalles referirse a la hoja de datos del dispositivo página 169

¹³Para más detalles referirse a la hoja de datos del dispositivo página 170

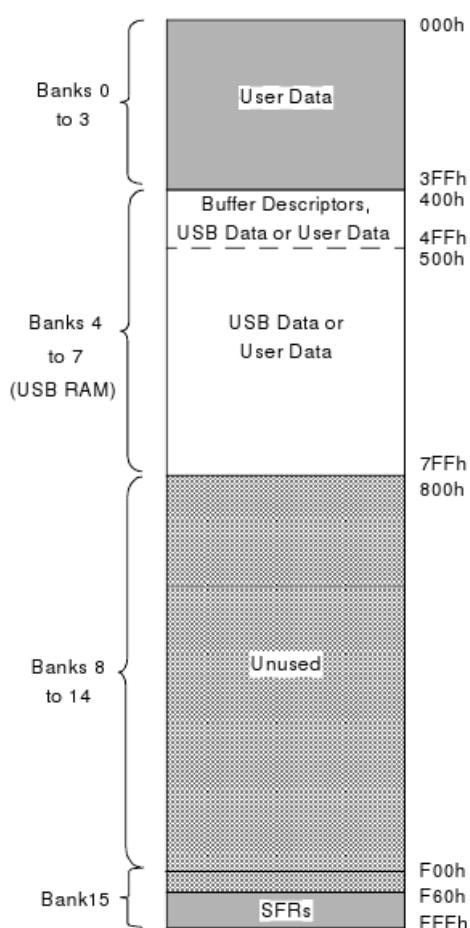


Figura 6.10: Mapeo de memoria *USB RAM*.

no usada del Banco 4 hasta el Banco 7.

Por estar mapeada en memoria de datos y siendo accesible tanto por el microcontrolador como por el *SIE*, la *USB RAM*, se maneja mediante un sistema de semáforos, asegurando de ésta manera que un solo dispositivo este accediendo al mismo segmento de memoria a la vez.

6.7.3. Descriptores de Buffer y Tabla de Descriptores de Buffers

Para proveerle cierta flexibilidad al usuario, el módulo USB hace uso de una estructura particular de datos donde se describen los *endpoints* del USB denominada *Buffer Descriptor Table (BDT)*.

La BDT se compone de *Buffer Descriptors (BDs)* los cuales, a su vez, se componen de 4 registros:

- BDnSTAT: Registro de estado del BD
- BDnCNT: Registro de tamaño de bloque de memoria del BD
- BDnADRH: Dirección alta del BD
- BDnADRL: Dirección baja del BD

Esta disposición se puede apreciar mejor en la figura 6.11.

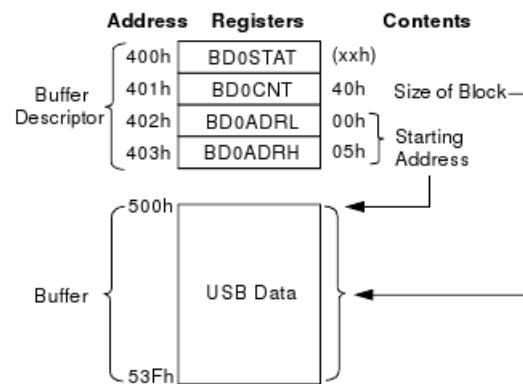


Figura 6.11: Tabla de descriptores de Buffers *BDT*.

6.7.4. BDn

Los *BDs* se componen de 4 registros (o bytes) por lo que cada *BDnSTAT* se encuentra siempre a $4n-1$ bytes de distancia de la dirección 400h.

Este registro tiene la particularidad de poseer un comportamiento que depende de su contexto.

Modo CPU

Cuando $UOWN = 0$ ($BDnSTAT< 7 >$), el microcontrolador posee el control sobre dicho BD . Del registro $BDnSTAT$ se destacan los siguientes bits:

- bit 7 - $UOWN$: Se encuentra en 0 cuando el BD le pertenece al microcontrolador
- bit 2 - $BSTALL$: Activar o desactivar el estado $STALL$
- bit 1-0 - $BC9:BC8$: Los dos bits más significantes del contador, quien determina el tamaño del los datos IN o OUT .

El registro $BDnCNT$ contiene los bits 7:0 que determinan el tamaño del buffer. De esta manera los buffers se limitan a un tamaño específico que será el que se puede recibir en una transferencia OUT . Mientras que en una transferencia IN se fija el tamaño del dato que se transmitirá al host.

En cuanto a los registros $BDnADRL$ y $BDnADRH$ poseen la dirección de USB RAM de los endpoints. El microcontrolador no posee ningún mecanismo intrínseco que valide dicha dirección, lo cual debería hacerse mediante software.

Modo SIE

Cuando $UOWN = 1$ ($BDnSTAT< 7 >$), el BD se encuentra bajo el control del SIE . Luego los bits de éste registro adquieran otro significado:

- bit 7 - $UOWN$: Se encuentra en 1 cuando el BD le pertenece al SIE
- bit 5-4 - $PID3:0$: Poseen el PID de la transacción (IN , OUT , $SETUP$)
- bit 1-0 - $BC9:BC8$: Los dos bits más significantes del contador, quien determina el tamaño del los datos IN o OUT .

El registro $BDnCNT$ contiene los bits 7:0 que determinan el tamaño del buffer. Es el SIE quien actualiza estos valores según la ultima transacción efectuada. En una transacción del tipo OUT , el SIE compara el tamaño del dato recibido si supera el que esta definido en este registro automáticamente se emite una señal del tipo NAK . Cuando se hace un IN , el SIE actualiza este registro con la cantidad de datos enviados. Con este método es posible luego hacer chequeos del resultado verdadero de las transacciones USB.

CAPÍTULO

7

Impacto Ambiental

El impacto ambiental de este trabajo es prácticamente imperceptible puesto que no posee en su diseño cantidades excesivas de materiales nocivos para el ambiente. No obstante, y debido a el objetivo de abaratar costos, aquellos materiales que podrían tener algún tipo de impacto ambiental, son usados en cantidades mínimas (esto podrían ser las placas PCB o los capacitores).

Los materiales necesarios para el armado del dispositivo fueron cuidadosamente elegidos de manera tal que sean los más comercializados en Argentina, evitando de esta manera posibles requerimientos particulares de importación y transporte.

Es importante también agregar que en el caso de que una empresa decida fabricar el trabajo aquí expuesto, no necesitaría maquinaria excesiva alguna, por ende no existiría requerimientos alguno en cuanto a sus instalaciones.

CAPÍTULO

8

Impacto Social

Todo el proyecto se encuentra bajo licencias libres, por lo que cualquier persona o empresa interesada en este puede descargarlo de internet y comenzar a usarlo. Esto implica que cualquier persona, empresa u organismo público o privado puede:

- Usarlo con fines educativos
- Usarlo con fines personales
- Usarlo con fines lucrativos

Y justamente debido a la licencia elegida, la misma debe ser usada en cualquier trabajo derivado, asegurando así la continuidad de la libertad del mismo.

Como se mencionó anteriormente, todas las herramientas de software usadas para el desarrollo son libres que, conjuntamente con la licencia del trabajo, generan una sucesión de eventos que favorecen económicamente a todos los participes de éste proceso. El desarrollador no necesita pagar por licencias de software en las herramientas necesarias para trabajar con este proyecto, ni tampoco pagar para obtener el código y los diseños aquí presentados, esto logra que no se transfieran ciertos costos al producto final más que los del trabajo explícito del desarrollador. Luego éste puede vender su trabajo a una empresa que productize el trabajo, quien habrá obtenido un producto de bajo costo y por ende podrá venderlo a precios razonables de igual manera. Finalmente el usuario final obtiene un producto barato que no requiere comprar software alguno para funcionar, economizando allí también.

En una primera instancia, y debido al mercado aun en un estado de cierta incertidumbre y con competencia prácticamente nula¹, productos basados en este proyecto sufrirían un impacto en el precio final, no obstante serían precios perfectamente afrontables por entidades públicas o privadas como escuelas, universidades y bibliotecas. Pero en el escenario hipotético en el que el mercado nacional crezca, los precios tenderían a bajar al punto de poder ser afrontados por personas de recursos limitados.

De una forma u otra, el solo hecho de incrementar la cantidad de impresoras braille en Argentina es una necesidad innegable, ya sea a nivel institucional u hogareño.

¹Refiriéndonos al mercado nacional

CAPÍTULO

9

Economía

Debido a la característica meramente investigativa de éste trabajo, solo se exponen en este capítulo posibles modelos de negocios alrededor del *Software Libre*, y algunos nichos de mercado que se encuentran siendo explotados en la actualidad.

9.1. Modelos de negocios

La organización FLOSSMetrics (Free / Libre and Open Source Software Metrics¹) , que forma parte de la Flossquality (Open source quality research²) , ambos financiados por la Comisión Europea³ , postula varias categorizaciones⁴ de modelos de negocios potenciales.

9.1.1. Financiación externa de empresas

⁵ Esta categoría contempla los grupos o compañías que desarrollan *Software Libre* siendo financiados por una organización externa. Normalmente es esta organización externa quien pone las guías y requerimientos para el proyecto. Esta categoría es luego desglosada en tres partes según el origen de la financiación.

¹Véase - <http://www.flossmetrics.org/>

²Véase - <http://www.flossquality.eu/>

³Véase - http://cordis.europa.eu/fetch?CALLER=PROJ_LIST&ACTION=D&RCN=79449

⁴Véase - http://smeguide.conecta.it/index.php/6._FLOSS-based_business_models

⁵En inglés *Externally funded ventures*

Financiación Pública

⁶ Se refiere a aquellos proyectos que son financiados por entidades públicas. Estos suelen ser proyectos científicos o desarrollo de estándares cuyo fin último no necesariamente es generar ganancias económicas si no más bien algún beneficio social.

Financiación “Necesidad de mejora”

⁷ Ésta es la situación donde una compañía le paga a un grupo u otra compañía para que mejore o adapte (y quizá más tarde de soporte) de un proyecto libre existente.

Fundación Indirecta

⁸ Muchas veces ciertas compañías financieras proyectan de *Software Libre* como estrategia de negocio, para obtener, de manera indirecta, algún rédito económico. El ejemplo más común es cuando un fabricante de *hardware* financia a alguien más (muchas veces son sus mismos empleados trabajando para proyectos de *Software Libre*) para que escriba los *drivers* para un sistema operativo libre (como lo puede ser GNU/Linux).

9.1.2. Uso Interno

⁹ En este caso, las empresas, optan por realizar un desarrollo propio (en vez, de quizá, optar por una solución propietaria ya existente) para solucionar algún problema en particular. Luego liberan el código para beneficiarse de las contribuciones que puede brindar la comunidad de *Software Libre*.

9.1.3. Mejor conocimiento aquí sin limitaciones

¹⁰ En este modelo, una empresa presta servicios como *consultor externo* para otra compañía sacando rédito mediante el alto nivel de conocimiento de su personal. Este tipo de modelos favorece la competencia del mercado, ya que aquella empresa que necesita consultoría, puede elegir siempre al mejor postor, obligando a quien provee el servicio a mantener precios competitivos y asegurar un excelente nivel técnico en sus empleados.

En la provincia de Córdoba existen varias empresas que proveen este tipo de servicios. A continuación se listan algunas de ellas.

Machinalis ¹¹ Machinalis provee desarrollo y consultaría de soluciones de calidad basadas en Software Libre. Desarrollo de aplicaciones web (Python, Django, Zope y PHP). Soluciones de CMS (Plone y Drupal). Desarrollo de aplicaciones a medida

⁶En inglés *Public funding*

⁷En inglés “*Needed improvement*” funding

⁸En inglés *Indirect funding*

⁹En inglés *Internal use*

¹⁰En inglés *Best knowledge here without constraints*

¹¹Véase - <http://www.machinalis.com/>

(Escritorio, Infraestructura, bajo nivel). Uso de metodologías ágiles, y experiencia con trabajo remoto (con clientes hispano y angloparlantes en todo el mundo).

Maram Sistemas¹² Soluciones basadas en Software Libre. Desarrollo de sistemas personalizados. Redes inalámbricas. Servidores web e intranet. Migraciones.

Menttes¹³ Productos y servicios informáticos de alta calidad basados en Software Libre.

9.1.4. Mejor conocimiento aquí con limitaciones

¹⁴ Para prevenir el “robo” de clientes, una empresa puede proveer servicios libres, pero mantener privativo (o con una licencia más restrictiva) a una pequeña parte clave código.

Este modelo se puede observar en empresas como Google¹⁵ la cual libera gran parte de sus productos con licencias libres exceptuando algunos productos claves que le otorgan ventaja competitiva.

9.1.5. Doble Licencia

¹⁶ Un modelo muy explotado en el negocio del *Software Libre* es el de doble licenciamiento. En este modelo la empresa que desarrolla *Software Libre* otorga licencias libres para los proyectos que vayan a continuar siendo libres (usualmente entregando el producto gratis) y una licencia distinta (usualmente paga) para aquellas otras empresas que no deseen redistribuir el código.

Los casos más emblemáticos de este modelo de negocios son empresas como *Sun*¹⁷ con su producto *MySQL*¹⁸, el cual es distribuido con una licencia libre (*GPL*) o bien con una licencia *comercial* para el caso en el que el comprador no desee redistribuir el código de su producto .

9.1.6. Desarrollo no financiado

¹⁹ Existen situaciones donde, si se da con suficiente énfasis un efecto de *networking*²⁰, requiere solo un esfuerzo mínimo de la organización el publicar su producto, ya que la mayoría del aporte lo obtiene de otras compañías que por diversos intereses asignan

¹²Véase - <http://maram.com.ar/>

¹³Véase - <http://www.menttes.com/>

¹⁴En inglés *Best knowledge here with constraints*

¹⁵Véase - <http://www.google.com/corporate/>

¹⁶En inglés *Dual licensing*

¹⁷Véase - <http://www.sun.com>

¹⁸ Véase - <http://www.sun.com/software/products/mysql/getit.jsp>

¹⁹En inglés *Unfunded developments*

²⁰En castellano *red*. Es cuando por motivos técnicos o sociales el proyecto obtiene una impacto importante en algunas comunidades de internet.

recursos dedicados a ese producto sin pretender un beneficio económico inmediato. Los casos más comunes son las distribuciones de GNU/Linux (como Debian²¹) o los sistemas operativos basados en *BSD*²² (como FreeBSD²³). Pero el ejemplo más ilustre de todos es el proyecto *linux*²⁴, el cual es desarrollado por un grupo muy reducido de personas, pero que recibe el aporte de miles de desarrolladores de alrededor del mundo, muchos de los cuales pertenecen a diversas empresas y son contratados para éste trabajo dedicado. La *Linux Foundation* publicó²⁵ un artículo²⁶ donde muestra el aporte de algunas de las empresas más importantes del mundo al proyecto linux como se puede observar en la figura 9.1.

²⁷

9.2. Posibles modelos particulares

No es el fin de éste proyecto obtener rédito económico alguno, sin embargo puede ser de interés de alguien sacar provecho del trabajo aquí presentado, para ello se analiza a continuación algunos aspectos económicos del mismo.

9.2.1. Costos

Los precios usados son obtenidos de (en caso de ser posible) sus respectivos fabricantes, o de grandes distribuidores en su defecto, asumiendo compras mayores de 100 unidades, suponiendo así que es una empresa en vista de producción en serie quien usará los datos aquí presentados.

Al ser un diseño sencillo, solo hay algunos elementos claves del mismo que poseen valor comparable al total. La tabla 9.1 muestra los precios unitarios de estos elementos.

Elemento	Precio (USD)
PIC18F4550	4.26
Ficha USB	0.52
TIP122	0.47
PCBs	2
Extras	3

Tabla 9.1: Precios unitarios por cien unidades

Luego por unos diez dolares es posible armar ambas placas, y gracias a haber elegido todo software libre para el desarrollo, no es necesario comprar ninguna herramienta

²¹<http://www.debian.org/>

²²<http://es.wikipedia.org/wiki/bsd>

²³<http://www.freebsd.org/>

²⁴<http://www.kernel.org/>

²⁵<http://www.linuxfoundation.org/publications/whowriteslinux.pdf>

²⁶Que se encuentra algo desactualizado debido a la rápida evolución de este tipo de proyectos.

²⁷Datos más actualizados pueden ser generados automáticamente usando los *scripts* disponibles en http://www.kernel.org/pub/linux/kernel/people/gregkh/kernel_history/.

Company Name	Number of Changes	Percent of Total
None	13,850	21.1%
Red Hat	7,897	12.0%
IBM	4,150	6.3%
Novell	4,021	6.1%
Intel	3,923	6.0%
Unknown	2,765	4.2%
Oracle	2,003	3.1%
Consultant	1,480	2.3%
Parallels	1,142	1.7%
Fujitsu	1,007	1.5%
Academia	992	1.5%
Analog Devices	889	1.4%
Renesas Technology	884	1.3%
SGI	755	1.2%
Movial	738	1.1%
Sun	639	1.0%
HP	628	1.0%
Freescale	613	0.9%
Marvell	601	0.9%
MontaVista	574	0.9%
AMD	552	0.8%
Nokia	549	0.8%
Vyatta	513	0.8%
Google	512	0.8%
Atheros Communications	494	0.8%
NTT	445	0.7%
linutronix	445	0.7%
XenSource	432	0.7%
Simtec	414	0.6%
Astaro	411	0.6%

Figura 9.1: Contribución de diversas empresas al kernel de linux.

que agregue costo extra al dispositivo.

9.2.2. Modelos de negocio

Las características particulares de este trabajo permite trabajar con varios modelos distintos.

Es importante notar que este trabajo se encuentra completamente liberado, por lo que cualquier interesado puede usarlo sin pagar nada, y todas las especificaciones del mismo también pueden obtenerse sin costo alguno. No obstante los siguientes modelos son perfectamente aplicables, y pueden ser transferidos durante todo la cadena del mercado.

Venta de soporte Quien posea el conocimiento técnico del desarrollo puede vender soporte, a algún potencial cliente que desee productizar el trabajo.

Venta de paquete de desarrollo Es posible empaquetar todo el trabajo de tal manera de entregarle al cliente una distribución de GNU/Linux con todas las herramientas necesarias instaladas para comenzar la productización sin tener que preocuparse por configurar el ambiente de desarrollo.

Venta de desarrollo particular Si el cliente desea alguna funcionalidad extra que actualmente no ese encuentra implementada o quizá portar el código a otro sistema operativo, el trabajo de desarrollar e implementar los requerimientos puede venderse.

Otro tema importante a notar es que la licencia usada (GPL v3) fuerza a que toda obra derivada de este trabajo mantenga la misma licencia y sea publicada, por lo que cualquier mejora hecha al mismo, ya sea pagada por un cliente privado o sencillamente implementada por un particular, siempre se hará pública.

CAPÍTULO

10

Conclusión

Ponerse como objetivo primario el reducir costos (sea cual fuere el motivo de ello), es sin duda alguna una de las dificultades, que de una forma u otra terminan manifestándose como técnicas, más arduas de un trabajo de ingeniería. Sin embargo las ganancias que se obtienen son invaluables.

Existen en la actualidad soluciones provistas por diversas empresas para casi cualquier problema de la ingeniería que requieren poco esfuerzo de aprendizaje e incluso, en muchos casos, sus costos traen aparejado un retorno de inversión principalmente al reducir el tiempo de desarrollo. No obstante la principal desventaja de este tipo de soluciones es que abstraen de manera significativa al ingeniero de las tecnologías y estándares subyacentes, y suelen condicionarlo e incluso generar dependencia.

El haber comenzado el trabajo sin ninguna inclinación hacia una metodología o herramienta en particular permitió hacer un análisis objetivo del mismo focalizándose en los objetivos finales y sin arrastrar condicionamientos o limitaciones durante el desarrollo. De esta manera fue posible evaluar las opciones y tomar decisiones confidentes.

La enorme cantidad de información recolectada para este trabajo fue tanto una gran fuente de ayuda técnica como de aprendizaje en general, puesto que todos los puntos tratados rozan tangencialmente diversas áreas de gran interés para el autor.

Durante el desarrollo del trabajo, cada uno de los objetivos propuestos fueron siendo superados mediante un proceso de investigación, análisis e implementación. Aunque en mucho casos la complejidad de ciertos tópicos y la escasez de información hicieron

del trabajo una tarea ardua, los mismos se re interpretaron como desafíos y fueron superados.

Si bien el resultado de este proyecto no es un producto terminado y listo para poner en producción, constituye un importante trabajo de investigación y provee todas la herramientas, información y las bases para continuar con una implementación específica que, de una forma u otra, cumpliría con los objetivos propuestos para este trabajo por un proceso transitivo.

CAPÍTULO

11

Anexo

11.1. Software Libre

El Software Libre, es aquel que le asegura cuatro libertades básicas al usuario. La libertad de usar el programa, de estudiar su funcionamiento, de compartirlo y la libertad de mejorarlo y distribuirlo.

11.1.1. Historia

El concepto de Software Libre (Free Software) comienza a gestarse en la década del 60/70. En ese entonces el software no era considerado un producto, sino más bien un añadido que formaba parte del hardware. Por esta razón las personas que trabajaban con software compartían sus códigos fuentes libremente. A finales de los 70, las compañías de software comenzaron a implementar licencias a sus programas con ciertas restricciones. Surgía entonces la necesidad de tener un sistema operativo libre donde correr aplicaciones libres.

11.1.2. GNU

En 1984, un estudiante del Instituto de Tecnología de Massachusetts (Massachusetts Institute of Technology - M.I.T), llamado Richard Stallman, comenzó a desarrollar el proyecto G.N.U (GNU's not UNIX). Éste, pretendía ser un reemplazo libre de los sistemas propietarios que estaban surgiendo en aquella época. Sus motivos¹ eran diversos, pero principalmente creía que era necesario desarrollar un entorno libre para las personas. Para llevar a cabo su proyecto se basó en UNIX, un sistema operativo portable

¹Véase "Manifiesto GNU" - <http://www.gnu.org/gnu/manifesto.es.html>

multiusuario y multitarea. Junto con la Colección de Compiladores GNU (GCC)², un editor de texto y todo un stack de software, comenzó a idear el proyecto GNU.

11.1.3. GNU/Linux

El proyecto GNU tomaba forma pero le faltaba un componente muy importante; el kernel. En 1991, un estudiante finlandés llamado Linus Torvalds, liberó un kernel basado en UNIX, que luego pasaría a formar parte de lo que hoy se conoce como GNU/Linux. GNU/Linux es un sistema operativo completo con un stack de software que satisface la mayoría de las necesidades de cualquier usuario, más un potente kernel multiplataforma. Lo que hace a GNU/Linux (y a la mayoría de los programas libres) versátiles, robustos y estables, es que se basan en estándares abiertos y libres, como lo son *SUS*³, *POSIX*⁴ o *IEEE*⁵.

11.1.4. Concepto de Software Libre

Luego del proyecto GNU, Richard Stallman fundó la Fundación para el Software Libre (Free Software Foundation) ⁶ quien se encarga (entre otras cosas) de mantener la definición del concepto⁷ de Software Libre.

“Free software” is a matter of liberty, not price. To understand the concept, you should think of “free” as in “free speech,” not as in “free beer.”

El Software Libre es una cuestión de libertad no de precio⁸. Para entender el concepto debe pensar libre (*free*) como en libre discurso no como cerveza gratis.

Para que un programa sea Software Libre, debe garantizarle cuatro libertades básicas al usuario:

Libertad 0 Libertad de ejecutar el programa con cualquier propósito

Libertad 1 Libertad de estudiar como funciona el programa y de adaptarlo a tus necesidades. *Acceso al código fuente es un pre-condición para esto*

Libertad 2 Libertad de redistribuir copias del mismo para poder ayudar a tu vecino.

Libertad 3 Libertad de mejorar el programa y publicar los cambios para que toda la comunidad se beneficie de ellos. *Acceso al código fuente es un pre-condición para esto*

²Previamente llamado -GNU C Compiler- puesto que solo compilaba código C. Versiones más recientes soporta varios lenguajes. Véase - <http://gcc.gnu.org>

³Single Unix Specification

⁴Véase - <http://en.wikipedia.org/wiki/Posix>

⁵Véase - <http://www.ieee.org>

⁶Véase - <http://www.fsf.org>

⁷Véase - <http://www.fsf.org/licensing/essays/free-sw.html>

⁸Ésta aclaración surge debido a que en inglés, *free* significa tanto *libre* como *gratis*.

11.1.5. El Software Libre en la ingeniería

Existen en la actualidad tecnologías o herramientas libres para lidiar con la mayoría de los problemas de la ingeniería. Pero que, debido a su propia naturaleza libre u *open source*, carecen de publicidad suficiente como para competir con sus alternativas *propietarias*.

No obstante su condición libre, la mayoría de estas tecnologías están a la altura de sus pares *propietarios*. Tanto es así que existen empresas que se dedican exclusivamente a este tipo de tecnologías⁹, y otras grandes empresas como Intel¹⁰, IBM¹¹ y Motorola¹², trabajan activamente en proyectos libres.

11.2. Mercado de impresoras braille

En las siguientes secciones se listan algunas empresas que fabrican dispositivos de impresión braille, con información sobre precios y velocidades de sus respectivos modelos.

11.2.1. Index Braille

Esta empresa posee cuatro modelos¹³ de impresoras braille. La tabla 11.1 muestra cada uno de ellos, sus precios y la cantidad de caracteres por segundos que pueden imprimir.

Modelo	Precio en dólares	Caracteres por segundo
Basic-D	3295	100
Everest	4495	100
4X4 PRO	5995	100
4Waves PRO	31500	300

Tabla 11.1: Modelos de *INDEX BRAILLE*

⁹Véase - <http://www.redhat.com/>

¹⁰Véase - <http://software.intel.com/sites/oss/>

¹¹Véase - <http://www.ibm.com/developerworksopensource/>

¹²Véase - <https://opensource.motorola.com/>

¹³Véase - <http://www.indexbraille.com/Products/Embossers.aspx>

11.2.2. Viewplus

Viewplus posee un gran variedad de modelos¹⁴ de impresoras braille. La tabla 11.2 muestra dichos modelos.

Modelo	Precio en dólares	Caracteres por segundo
Tiger Cub Jr.	3995	30
Tiger Cub	4995	50
Tiger Max	5995	60
Pro Gen II	9750	100
Emprint SpotDot	6995	50
Tiger Premier 80	9995	80
Tiger Premier 100	11995	100
Tiger Elite 150	17995	150
Tiger Elite 200	24995	200

Tabla 11.2: Modelos de *Viewplus*

11.2.3. Enabling Technologies

Los modelos de impresoras braille¹⁵ de *Enabling Technologies* se caracterizan por su portabilidad y no tanto por su velocidad. La tabla 11.3 muestra los modelos de impresoras de esta empresa.

Modelo	Precio en dólares	Caracteres por segundo
Romeo Attaché	1995	15
Romeo Attaché Pro	2195	15
Romeo 25	2495	25
Romeo Pro 50	2995	50
Thomas	3495	40
Thomas Pro 80	3795	40
TMarathon 100	16995	200

Tabla 11.3: Modelos de *Enabling Technologies*

¹⁴Véase - <http://www.viewplus.com/products/braille-embossers/braille-printers/>

¹⁵Véase - <http://www.brailler.com/>

11.3. USB firmware

11.3.1. Cabecera

```

/*
 * usb.h - The header file for usb.h.
 *
 * Copyright (C) 2009 Rosales Victor (todoesverso@gmail.com)
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

#ifndef USB_H
#define USB_H

/**
 * Convert pointers to fit PIC memory type
 */
#define PTR16(x) (((unsigned int)((unsigned long)x) & 0xFFFF))

/**
 * Define two new types of variables
 */
typedef unsigned char byte;
typedef unsigned int word;

/**
 * Separate words into 2 varialbes type byte
 */
#define LSB(x) (x & 0xFF)
#define MSB(x) ((x & 0xFF00) >> 8)

/**
 * Standard Request Codes USB 2.0 Spec Ref Table 9-4
 */
#define GET_STATUS 0
#define CLEARFEATURE 1
#define SETFEATURE 3
#define SET_ADDRESS 5
#define GET_DESCRIPTOR 6
#define SET_DESCRIPTOR 7
#define GET_CONFIGURATION 8
#define SET_CONFIGURATION 9
#define GET_INTERFACE 10
#define SET_INTERFACE 11
#define SYNCHFRAME 12

/**
 * Descriptors Types
 */
#define DEVICE_DESCRIPTOR 0x01
#define CONFIGURATION_DESCRIPTOR 0x02
#define STRING_DESCRIPTOR 0x03
#define INTERFACE_DESCRIPTOR 0x04
#define ENDPOINT_DESCRIPTOR 0x05

```

```

/*
 * Standard Feature Selectors
 */
#define DEVICE_REMOTE_WAKEUP      0x01
#define ENDPOINT_HALT             0x00

/*
 * Buffer Descriptor bit masks (from PIC datasheet)
 */
#define UOWN     0x80 /* USB Own Bit */          */
#define DTS      0x40 /* Data Toggle Synchronization Bit */    */
#define KEN      0x20 /* BD Keep Enable Bit */        */
#define INCDIS   0x10 /* Address Increment Disable Bit */    */
#define DTSEN   0x08 /* Data Toggle Synchronization Enable Bit */ */
#define BSTALL  0x04 /* Buffer Stall Enable Bit */       */
#define BC9     0x02 /* Byte count bit 9 */           */
#define BC8     0x01 /* Byte count bit 8 */            */

/*
 * Device states (USB spec Chap 9.1.1)
 */
#define DETACHED      0
#define ATTACHED      1
#define POWERED       2
#define DEFAULT        3
#define ADDRESS        4
#define CONFIGURED    5

/*
 * BDT - Buffer Descriptor Table
 * @stat:
 * @Cnt:
 * @ADDR:
 *
 */
typedef struct _BDT
{
    byte Stat;
    byte Cnt;
    word ADDR;
} BDT;

/*
 * Global Variables
 */
extern byte deviceState; /* Visible device states (from USB 2.0, chap 9.1.1) */
extern byte selfPowered;
extern byte remoteWakeups;
extern byte currentConfiguration;

extern volatile BDT at 0x0400 ep0Bo; /* Endpoint #0 BD Out */          */
extern volatile BDT at 0x0404 ep0Bi; /* Endpoint #0 BD In */           */
extern volatile BDT at 0x0408 ep1Bo; /* Endpoint #1 BD Out */          */
extern volatile BDT at 0x040C ep1Bi; /* Endpoint #1 BD In */           */
extern volatile BDT at 0x0410 ep2Bo; /* Endpoint #2 BD Out */          */
extern volatile BDT at 0x0414 ep2Bi; /* Endpoint #2 BD In */           */

/*
 * setupPacketStruct -
 *
 * Every device request starts with an 8 byte setup packet (USB 2.0, chap 9.3)
 * with a standard layout. The meaning of wValue and wIndex will
 * vary depending on the request type and specific request.
 */
typedef struct _setupPacketStruct {

```

```

    byte bmRequestType; /* D7: Direction , D6..5: Type , D4..0: Recipient      */
    byte bRequest;       /* Specific request                         */
    byte wValue0;        /* LSB of wValue                           */
    byte wValue1;        /* MSB of wValue                           */
    byte wIndex0;        /* LSB of wIndex                           */
    byte wIndex1;        /* MSB of wIndex                           */
    word wLength;         /* Number of bytes to transfer if a data stage   */
    byte extra[56];      /* Fill out to same size as Endpoint 0 max buffer */
} setupPacketStruct;

/**
 * Variable for Setup Packets
 */
extern volatile setupPacketStruct SetupPacket;

/**
 * Size of the buffer for endpoint 0
 */
#define E0SZ 64

/**
 * Size of data for BulkIN and BulkOut
 */
#define INPUT_BYTES      7
#define OUTPUT_BYTES     7

/**
 * IN/OUT Buffers
 * RxBuffer2 is only 1 byte long.
 */
extern volatile byte TxBUFFER[OUTPUT_BYTES];
extern volatile byte RxBuffer[INPUT_BYTES];
extern volatile byte TxBUFFER2[OUTPUT_BYTES];
extern volatile byte RxBuffer2;

/**
 * Pointers inPtr and outPtr are used to move data between buffers from user
 * memory to dual port buffers of the USB module
 */
extern byte *outPtr;
extern byte *inPtr;
extern unsigned int wCount; /* Total number of bytes to move */

// Funciones para uso del USB
/**
 * User functions to process USB transactions
 */
void EnableUSBModule(void);
void ProcessUSBTransactions(void);

/**
 * Functions to read and write bulk endpoints
 */
byte BulkOut(byte ep_num, byte *buffer, byte len);
byte BulkIn(byte ep_num, byte *buffer, byte len);

#endif /* USB_H */

```

11.3.2. Fuente USB

```

/* usb.c - The main functions for usb handle.
*
* Copyright (C) 2009 Rosales Victor (todoesverso@gmail.com)
*
```

```
* This program is free software: you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
*/
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program. If not, see <http://www.gnu.org/licenses/>.
*/
#include <pic18fregs.h>
#include <string.h>
#include <stdio.h>
#include "usb.h"

/**
 * Device and configuration descriptors. These are used as the
 * host enumerates the device and discovers what class of device
 * it is and what interfaces it supports.
*/
/**
 * Size in bytes of descriptors
*/
#define DEVICE_DESCRIPTOR_SIZE 0x12
#define CONFIG_HEADER_SIZE 0x09
#define CONFIG_DESCRIPTOR_SIZE 0x25
/**
 * The total size of the configuration descriptor
 * 0x09 + 0x09 + 0x07 + 0x07 = 0x20
*/
#define CFSZ CONFIG_HEADER_SIZE+CONFIG_DESCRIPTOR_SIZE

/**
 * Struct of Configuration Descriptor
 * @configHeader:
 * @Descriptor:
 *
 */
typedef struct _configStruct {
    byte configHeader[CONFIG_HEADER_SIZE];
    byte Descriptor[CONFIG_DESCRIPTOR_SIZE];
} ConfigStruct;

/**
 * Global Variables
*/
/**
 * Visible States (USB 2.0 Spec, chap 9.1.1)
*/
byte deviceState;
byte remoteWakeup;
byte deviceAddress;
byte selfPowered;
byte currentConfiguration;

/* Control Transfer Stages - see USB spec chapter 5 */
/* Start of a control transfer (followed by 0 or more data stages) */
#define SETUP_STAGE 0
#define DATA_OUT_STAGE 1 /* Data from host to device */
```

```

#define DATA_IN_STAGE 2 /* Data from device to host */  

#define STATUS_STAGE 3 /* Unused - if data I/O went ok, then back to setup */  
  

byte ctrlTransferStage; /* Holds the current stage in a control transfer */  

byte requestHandled; /* Set to 1 if request was understood and processed. */  
  

byte *outPtr; /* Data to send to the host */  

byte *inPtr; /* Data from the host */  

word wCount; /* Number of bytes of data */  

byte RxLen; /* # de bytes colocados dentro del buffer */  
  

/**  

 * Device Descriptor  

**/  

code byte deviceDescriptor[] = {  

    DEVICE_DESCRIPTOR_SIZE, 0x01, /* bLength, bDescriptorType */  

    0x00, 0x02, /* bcdUSB (LB), bcdUSB (HB) */  

    0x00, 0x00, /* bDeviceClass, bDeviceSubClass */  

    0x00, EOSZ, /* bDeviceProtocol, bMaxPacketSize */  

    0xD8, 0x04, /* idVendor (LB), idVendor (HB) */  

    0x31, 0x75, /* idProduct (LB), idProduct (HB) */  

    0x01, 0x00, /* bcdDevice (LB), bcdDevice (HB) */  

    0x01, 0x02, /* iManufacturer, iProduct */  

    0x00, 0x01 /* iSerialNumber (none), bNumConfigurations */  
};  
  

#define ISZ 7 /* wMaxPacketSize (low) of endpoint1IN */  

#define OSZ 7 /* wMaxPacketSize (low) of endpoint1OUT */  

#define OSZ2 1 /* wMaxPacketSize (low) of endpoint2OUT */  
  

/**  

 * Configuration Descriptor  

**/  

code ConfigStruct configDescriptor = {  

    {  

        /* Descriptor de configuración, contiene el descriptor de la clase */  

        CONFIG_HEADER_SIZE, 0x02, /* bLength, bDescriptorType (Configuration) */  

        CFSZ, 0x00, /* wTotalLength (low), wTotalLength (high) */  

        0x01, 0x01, /* bNumInterfaces, bConfigurationValue */  

        0x00, 0xA0, /* iConfiguration, bmAttributes () */  

        0x32, /* bMaxPower (100 mA) */  

    },  

    {  

        /* Interface Descriptor */  

        0x09, 0x04, /* bLength, bDescriptorType (Interface) */  

        0x00, 0x00, /* bInterfaceNumber, bAlternateSetting */  

        0x04, 0x07, /* bNumEndpoints, bInterfaceClass (Printer) */  

        0x01, 0x00, /* bInterfaceSubclass, bInterfaceProtocol */  

        0x00, /* iInterface */  

        /* EP1 IN */  

        0x07, 0x05, /* bLength, bDescriptorType (Endpoint) */  

        0x81, 0x02, /* bEndpointAddress, bmAttributes (Bulk) */  

        ISZ, 0x00, /* wMaxPacketSize (L), wMaxPacketSize (H) */  

        0x01, /* bInterval (1 millisecond) */  

        /* EP1 OUT */  

        0x07, 0x05, /* bLength, bDescriptorType (Endpoint) */  

        0x01, 0x02, /* bEndpointAddress, bmAttributes (Bulk) */  

        OSZ, 0x00, /* wMaxPacketSize (L), wMaxPacketSize (H) */  

        0x01, /* bInterval (1 millisecond) */  

        /* EP2 IN */  

        0x07, 0x05, /* bLength, bDescriptorType (Endpoint) */  

        0x82, 0x02, /* bEndpointAddress, bmAttributes (Bulk) */  

        ISZ, 0x00, /* wMaxPacketSize (L), wMaxPacketSize (H) */  

        0x01, /* bInterval (1 millisecond) */  

        /* EP2 OUT */  

        0x07, 0x05, /* bLength, bDescriptorType (Endpoint) */  

        0x02, 0x02, /* bEndpointAddress, bmAttributes (Bulk) */  

    }  
};  


```

```

    OSZ2, 0x00,           /* wMaxPacketSize (L), wMaxPacketSize (H)      */
    0x01,                 /* bInterval (1 millisecond)                      */
}
};

< /**
 * According to USB spec , the 0 index has the language code
 */
code byte stringDescriptor0[] = {
    0x04, STRING_DESCRIPTOR,          /* bLength, bDscType                           */
    0x09, 0x04,                     /* Establish wLANGID with 0x0409 (English ,USA) */
};

code byte stringDescriptor1[] = {
    0x1A, STRING_DESCRIPTOR,          /* bLength, bDscType                           */
    'E', 0x00, 'm', 0x00, 'b', 0x00, 'o', 0x00,
    's', 0x00, 's', 0x00, 'e', 0x00, 'r', 0x00,
    ' ', 0x00, ' ', 0x00,
    ' ', 0x00, ' ', 0x00,
};

code byte stringDescriptor2[] = {
    0x20, STRING_DESCRIPTOR,          /* bLength, bDscType                           */
    'U', 0x00, 'S', 0x00, 'B', 0x00, ' ', 0x00,
    'B', 0x00, 'r', 0x00, 'a', 0x00, 'i', 0x00,
    'l', 0x00, 'l', 0x00, 'e', 0x00, ' ', 0x00,
    'o', 0x00, '.', 0x00, '1', 0x00,
};

volatile BDT at 0x0400 ep0Bo; /* Endpoint #0 BD OUT */ 
volatile BDT at 0x0404 ep0Bi; /* Endpoint #0 BD IN  */
volatile BDT at 0x0408 ep1Bo; /* Endpoint #1 BD OUT */ 
volatile BDT at 0x040C ep1Bi; /* Endpoint #1 BD IN  */ 
volatile BDT at 0x0410 ep2Bo; /* Endpoint #2 BD OUT */ 
volatile BDT at 0x0414 ep2Bi; /* Endpoint #2 BD IN  */

/*
 * Put endpoint 0 buffers into dual port RAM
 */
#pragma udata usbram5 SetupPacket controlTransferBuffer
volatile setupPacketStruct SetupPacket;
volatile byte controlTransferBuffer[EOSZ];

< /**
 * Put I/O buffers into dual port USB RAM
 */
#pragma udata usbram5 RxBuffer TxBuffer
#pragma udata usbram6 RxBuffer2 TxBuffer2

< /**
 * Specific Buffers
 */
volatile byte RxBuffer[OSZ];
volatile byte TxBuffer[ISZ];
volatile byte RxBuffer2;
volatile byte TxBuffer2[ISZ];

< /**
 * Endpoints Initialization
 */
void InitEndpoint(void)
{
    /* Turn on both IN and OUT for this endpoints (EP1 & EP2) */
    UEP1 = 0x1E; /* See PIC datasheet , page 169 (USB E1 Control) */
    UEP2 = 0x1E; /* Same as above for EP2 */
    /*
     * Load EP1's BDT

```

```

        */
        ep1Bo.Cnt = sizeof(RxBuffer);
        ep1Bo.ADDR = PTR16(&RxBuffer);
        ep1Bo.Stat = UOWN | DTSEN;
        ep1Bi.ADDR = PTR16(&TxBuffer);
        ep1Bi.Stat = DTS;
        /**
         * Load de EP2's BDT
         */
        ep2Bo.Cnt = RxBuffer2;
        ep2Bo.ADDR = PTR16(&RxBuffer2);
        ep2Bo.Stat = UOWN | DTSEN;
        ep2Bi.ADDR = PTR16(&TxBuffer2);
        ep2Bi.Stat = DTS;
    }

/**
 * BulkIn() - Makes an IN and returns the amount of bytes transferred
 * @ep_num: Number of the endpoint to be used (only EP1 & EP2)
 * @buffer: Buffer of the data to be transferred
 * @len: Lenght of the bytes transferred
 *
 * The function checks if the BD is owned by the CPU returning 0 if so.
 * (PIC 18F4550 datasheet page 171 section 17.4.1.1 - Buffer Ownership)
 *
 * Send up to len bytes to the host. The actual number of bytes sent is returned
 * to the caller. If the send failed (usually because a send was attempted while
 * the SIE was busy processing the last request), then 0 is returned.
 */
byte BulkIn(byte ep_num, byte *buffer, byte len)
{
    byte i;
    /**
     * If selected EP1
     */
    if (ep_num == 1) {
    /**
     * If SIE owns the BD do not try to send anything and return 0.
     */
        if (ep1Bi.Stat & UOWN)
            return 0;
    /**
     * Truncate requests that are too large
     */
        if(len > ISZ)
            len = ISZ;
    /**
     * Copy data from user's buffer to dual-port ram buffer
     */
        for (i = 0; i < len; i++)
            TxBuffer[i] = buffer[i];
    /**
     * Toggle the data bit and give control to the SIE
     */
        ep1Bi.Cnt = len;
        if(ep1Bi.Stat & DTS)
            ep1Bi.Stat = UOWN | DTSEN;
        else
            ep1Bi.Stat = UOWN | DTS | DTSEN;

        return len;
    }
    /**
     * If selected EP2 (same as above)
     */
    else if (ep_num == 2) {
        if (ep2Bi.Stat & UOWN)

```

```

        return 0;

    if(len > ISZ)
        len = ISZ;

    for (i = 0; i < len; i++)
        TxBuffer2[i] = buffer[i];

    ep2Bi.Cnt = len;
    if(ep2Bi.Stat & DTS)
        ep2Bi.Stat = UOWN | DTSEN;
    else
        ep2Bi.Stat = UOWN | DTS | DTSEN;

    return len;
}

/**
 * In case of error (ep_num != 1|2) return 0
 */
return 0;
}

/**
 * BulkOut() - Reads 'len' bytes from the dual-port buffer
 * @ep_num: Number of the endpoint to be read
 * @buffer: Buffer to collect the data
 * @len: Length of the data received
 *
 * Actual number of bytes put into buffer is returned.
 * If there are fewer than len bytes, then only the available
 * bytes will be returned. Any bytes in the buffer beyond len
 * will be discarded.
 */
byte BulkOut(byte ep_num, byte *buffer, byte len)
{
    RxLen = 0;
    /**
     * If selected EP1
     */
    if (ep_num == 1) {
    /**
     * If the SIE doesn't own the output buffer descriptor,
     * then it is safe to pull data from it
     */
        if(!(ep1Bo.Stat & UOWN)) {
    /**
     * See if the host sent fewer bytes than we asked for.
     */
        if(len > ep1Bo.Cnt)
            len = ep1Bo.Cnt;
    /**
     * Copy data from dual-ram buffer to user's buffer
     */
        for (RxLen = 0; RxLen < len; RxLen++)
            buffer[RxLen] = RxBuffer[RxLen];
    /**
     * Resets the OUT buffer descriptor so the host can send more data
     */
        ep1Bo.Cnt = sizeof(RxBuffer);
        if (ep1Bo.Stat & DTS)
            ep1Bo.Stat = UOWN | DTSEN;
        else
            ep1Bo.Stat = UOWN | DTS | DTSEN;
    }
}
/**
 * The same that above but for EP2. In this case the 'len'

```

```

        * is hardcoded to '1' because only 1 byte is needed for
        * instructions for the printer.
        */
    else if (ep_num == 2) {
        if (!(ep2Bo.Stat & UOWN)) {
            if (len > ep2Bo.Cnt)
                len = ep2Bo.Cnt;

            buffer[0] = RxBuffer2;
            RxLen = 1;

            ep2Bo.Cnt = 1;
            if (ep2Bo.Stat & DTS)
                ep2Bo.Stat = UOWN | DTSEN;
            else
                ep2Bo.Stat = UOWN | DTS | DTSEN;
        }
    }
    /**
     * Returns the lenght of the data received
     */
    return RxLen;
}

/**
 * Start of code to process standard requests (USB spec chapter 9)
**/

/**
 * GetDescriptor(void) - Process Descriptors requests
 *
 *
 */
static void GetDescriptor(void)
{
    /**
     * If direction is device --> host
     */
    if (SetupPacket.bmRequestType == 0x80) {
        /**
         * MSB has descriptor type
         * LSB has descriptor Index
         */
        byte descriptorType = SetupPacket.wValue1;
        byte descriptorIndex = SetupPacket.wValue0;
        /**
         * If request for device
         */
        if (descriptorType == DEVICE_DESCRIPTOR) {
            requestHandled = 1;
            /**
             * Points to device descriptor's address
             */
            outPtr = (byte *) &deviceDescriptor;
            wCount = DEVICE_DESCRIPTOR_SIZE;
        }
        /**
         * If requested for descriptor's configuration
         */
        else if (descriptorType == CONFIGURATION_DESCRIPTOR) {
            requestHandled = 1;
            outPtr = (byte *) &configDescriptor;
            wCount = configDescriptor.configHeader[2];
            /** Note: SDCC may generate bad code with this ***/
        }
    }
}

```

```

        * If requested for descriptor's string
        */
    else if (descriptorType == STRING_DESCRIPTOR) {
        requestHandled = 1;
        if (descriptorIndex == 0)
            /* Language encoding */
            outPtr = (byte *) &stringDescriptor0;
        else if (descriptorIndex == 1)
            /* Author name */
            outPtr = (byte *) &stringDescriptor1;
        else
            /* Device name */
            outPtr = (byte *) &stringDescriptor2;
        wCount = *outPtr;
    } else {
        /**
         * A blinking LED may be used if error occur
         */
    }
}

// Solicitud GET_STATUS (los datos parecen estar contenidos en el paquete Setup)
/***
 * GetStatus(void) -
 *
 ***/
static void GetStatus(void)
{
    /**
     * Mask Off the Recipient bits
     *
     * From USB spec chapter 9.4.5
     *
     * bmRequestType
     * 10000000 -> Device
     * 10000001 -> Interface
     * 10000010 -> Endpoint
     */
    byte recipient = SetupPacket.bmRequestType & 0x1F;
    controlTransferBuffer[0] = 0;
    controlTransferBuffer[1] = 0;

    /**
     * Requested for Device
     */
    if (recipient == 0x00) {
        requestHandled = 1;
        if (selfPowered)
            /* Set SelfPowered bit */
            controlTransferBuffer[0] |= 0x01;
        if (remoteWakeup)
            /* Set RemoteWakeUp bit */
            controlTransferBuffer[0] |= 0x02;
    }
    /**
     * Requested for Interface
     */
    else if (recipient == 0x01)
        requestHandled = 1;
    /**
     * Requested for Endpoint
     */
    else if (recipient == 0x02) {
        byte endpointNum = SetupPacket.wIndex0 & 0x0F;
        byte endpointDir = SetupPacket.wIndex0 & 0x80;
        requestHandled = 1;
    }
}

```

```

/**
 * Endpoint descriptors are 8 bytes long, with each in and out
 * taking 4 bytes within the endpoint
 * (See PIC18F4550 'Buffer Descriptors and the Buffer Descriptor
 * Table' chapter 17.4)
 */
    inPtr = (byte *)&ep0Bo + (endpointNum * 8);

    if (endpointDir)
        inPtr += 4;

    if (*inPtr & BSTALL)
        controlTransferBuffer[0] = 0x01;
}
/***
 * If a request was handled (requestHandled) move OUT pointer to the
 * controlTransferBuffer adress.
***/
if (requestHandled) {
    outPtr = (byte *)controlTransferBuffer;
    wCount = 2;
}

/**
 * SetFeature( void ) -
 *
 */
static void SetFeature( void )
{
    byte recipient = SetupPacket.bmRequestType & 0x1F;
    byte feature = SetupPacket.wValue0;
    /**
     * Requested for Device
     */
    if (recipient == 0x00) {
        if (feature == DEVICE_REMOTE_WAKEUP) {
            requestHandled = 1;

            if (SetupPacket.bRequest == SET_FEATURE)
                remoteWakeUp = 1;
            else
                remoteWakeUp = 0;
        }
    }
    /**
     * Requested for Endpoint
     */
    else if (recipient == 0x02) {
        byte endpointNum = SetupPacket.wIndex0 & 0x0F;
        byte endpointDir = SetupPacket.wIndex0 & 0x80;

        if ((feature == ENDPOINT_HALT) && (endpointNum != 0)) {
            requestHandled = 1;
            inPtr = (byte *) &ep0Bo + (endpointNum * 8);

            if (endpointDir)
                inPtr += 4;

            if (SetupPacket.bRequest == SET_FEATURE)
                *inPtr = 0x84;
            else {
                if (endpointDir == 1)
                    *inPtr = 0x00;
                else
                    *inPtr = 0x88;
            }
        }
    }
}

```

```
        }

    }

/***
 * ProcessStandarRequest (void) -
 *
 ***/
void ProcessStandardRequest(void)
{
    /**
     * See USB 2.0 spec chapter 9.3
     */
    byte request = SetupPacket.bRequest;

    /**
     * Only attend Standar requests D6..5 == 00b
     */
    if ((SetupPacket.bmRequestType & 0x60) != 0x00)
        return;

    if (request == SET_ADDRESS) {
        /**
         * Set the address of the device. All future requests
         * will come to that address. Can't actually set UADDR
         * to the new address yet because the rest of the SET_ADDRESS
         * transaction uses address 0.
         */
        requestHandled = 1;
        deviceState = ADDRESS;
        deviceAddress = SetupPacket.wValue0;
    }

    else if (request == GET_DESCRIPTOR) {
        GetDescriptor();
    }

    else if (request == SET_CONFIGURATION) {
        requestHandled = 1;
        currentConfiguration = SetupPacket.wValue0;
        /**
         * TBD: ensure the new configuration value is one that
         * exists in the descriptor.
         */
        if (currentConfiguration == 0)
            /**
             * If configuration value is zero, device is put in
             * address state (USB 2.0 spec - 9.4.7)
             */
            deviceState = ADDRESS;
        else {
            deviceState = CONFIGURED;
            InitEndpoint();
        }
    }

    else if (request == GET_CONFIGURATION) {
        requestHandled = 1;
        outPtr = (byte*)&currentConfiguration;
        wCount = 1;
    }

    else if (request == GET_STATUS) {
        GetStatus();
    }

    else if ((request == CLEAR_FEATURE) || (request == SET_FEATURE)) {
```

```

        SetFeature();

    }

    else if (request == GET_INTERFACE) {
        /**
         * No support for alternate interfaces. Send
         * zero back to the host.
         */
        requestHandled = 1;
        controlTransferBuffer[0] = 0;
        outPtr = (byte *) &controlTransferBuffer;
        wCount = 1;
    }

    else if (request == SET_INTERFACE) {
        /**
         * No support for alternate interfaces - just ignore.
         */
        requestHandled = 1;
    }
    /* else if (request == SET_DESCRIPTOR) */      */
    /* else if (request == SYNCHFRAME) */          */
    /* else */                                     */
}

/**
 * InDataStage(void) - Data stage for a Control Transfer.
 *
 * Data stage for a Control Transfer that sends data to the host.
 */
void InDataStage(void)
{
    byte i;
    word bufferSize;
    /* Determine how many bytes are going to the host */
    if (wCount < EOSZ)
        bufferSize = wCount;
    else
        bufferSize = EOSZ;
    /**
     * Load the high two bits of the byte count into BC8:BC9
     */
    ep0Bi.Stat &= ~(BC8 | BC9); /* Delete BC8 and BC9 */
    ep0Bi.Stat |= (byte) ((bufferSize & 0x300) >> 8);
    ep0Bi.Cnt = (byte) (bufferSize & 0xFF);
    ep0Bi.ADDR = PTR16(&controlTransferBuffer);
    /**
     * Update the number of bytes that still need to be sent. Getting
     * all the data back to the host can take multiple transactions, so
     * we need to track how far along we are.
     */
    wCount = wCount - bufferSize;
    /**
     * Move data to the USB output buffer from wherever it sits now.
     */
    inPtr = (byte *)&controlTransferBuffer;

    for (i=0;i<bufferSize;i++)
        *inPtr++ = *outPtr++;
}

/**
 * OutDataStage(void) - Data stage for a Control Transfer
 *
 * Data stage for a Control Transfer that reads data from the host
 */
void OutDataStage(void)

```

```

{
    word i, bufferSize;

    bufferSize = ((0x03 & ep0Bo.Stat) << 8) | ep0Bo.Cnt;
    /**
     * Accumulate total number of bytes read
     */
    wCount = wCount + bufferSize;

    outPtr = (byte*)&controlTransferBuffer;

    for (i=0;i<bufferSize;i++)
        *inPtr++ = *outPtr++;
}

/**
 * SetupStage( void ) -
 *
 * Process the Setup stage of a control transfer. This code initializes the
 * flags that let the firmware know what to do during subsequent stages of
 * the transfer.
 */
void SetupStage( void )
{
    /**
     * Note: Microchip says to turn off the UOWN bit on the IN direction as
     * soon as possible after detecting that a SETUP has been received.
     */
    ep0Bi.Stat &= ~UOWN;
    ep0Bo.Stat &= ~UOWN;

    /* Initialize the transfer process */
    ctrlTransferStage = SETUP_STAGE;
    requestHandled = 0; /* Default is that request hasn't been handled */
    wCount = 0;           /* No bytes transferred */
    /**
     * See if this is a standard (as defined in USB chapter 9) request
     */
    ProcessStandardRequest();

    if (!requestHandled) {
        /**
         * If this service wasn't handled then stall endpoint 0
         */
        ep0Bo.Cnt = EOSZ;
        ep0Bo.ADDR = PTR16(&SetupPacket);
        ep0Bo.Stat = UOWN | BSTALL;
        ep0Bi.Stat = UOWN | BSTALL;
    }

    else if (SetupPacket.bmRequestType & 0x80) {
        /**
         * Direction: Device —> Host
         */
        if(SetupPacket.wLength < wCount)
            wCount = SetupPacket.wLength;

        InDataStage();
        ctrlTransferStage = DATA_IN_STAGE;
        /**
         * Reset the out buffer descriptor for endpoint 0
         */
        ep0Bo.Cnt = EOSZ;
        ep0Bo.ADDR = PTR16(&SetupPacket);
        ep0Bo.Stat = UOWN;
        /**
         * Set the in buffer descriptor on endpoint 0 to send data
        */
    }
}

```

```

        */
        ep0Bi.ADDR = PTR16(&controlTransferBuffer);
    /**
     * Give to SIE, DATA1 packet, enable data toggle checks
     */
        ep0Bi.Stat = UOWN | DTS | DTSEN;
    } else {
    /**
     * Direction: Host —> Device
     */
        ctrlTransferStage = DATA_OUT_STAGE;
    /**
     * Clear the input buffer descriptor
     */
        ep0Bi.Cnt = 0;
        ep0Bi.Stat = UOWN | DTS | DTSEN;
    /**
     * Set the out buffer descriptor on endpoint 0 to receive data
     */
        ep0Bo.Cnt = EOSZ;
        ep0Bo.ADDR = PTR16(&controlTransferBuffer);
    /**
     * Give to SIE, DATA1 packet, enable data toggle checks
     */
        ep0Bo.Stat = UOWN | DTS | DTSEN;
    }
    /**
     * Enable SIE token and packet processing
     */
    UC0Nbits.PKTDIS = 0;
}

/**
 * WaitForSetupStage(void) – Configures the buffer descriptor for EP0
 *
 * Configures the buffer descriptor for endpoint 0 so that it is waiting for
 * the status stage of a control transfer.
 */
void WaitForSetupStage(void)
{
    ctrlTransferStage = SETUP_STAGE;
    ep0Bo.Cnt = EOSZ;
    ep0Bo.ADDR = PTR16(&SetupPacket);
    ep0Bo.Stat = UOWN | DTSEN; /* Give to SIE, enable data toggle checks */
    ep0Bi.Stat = 0x00;          /* Give control to CPU */
}

/**
 * ProcessControlTransfer(void) –
 *
 * This is the starting point for processing a Control Transfer. The code directly
 * follows the sequence of transactions described in the USB spec chapter 5. The
 * only Control Pipe in this firmware is the Default Control Pipe (endpoint 0).
 * Control messages that have a different destination will be discarded.
 */
void ProcessControlTransfer(void)
{
    if (USTAT == 0) {
        /* Endpoint 0:OUT */                      */
        /* Pull PID from middle of BDOSTAT */      */
        byte PID = (ep0Bo.Stat & 0x3C) >> 2;

        if (PID == 0x0D)
        /**
         * SETUP PID – a transaction is starting
         */
            SetupStage();
}

```

```

        else if (ctrlTransferStage == DATA_OUT_STAGE) {
    /**
     * Complete the data stage so that all information has
     * passed from host to device before servicing it.
    */
        OutDataStage();
    /**
     * Turn control over to the SIE and toggle the data bit
    */
        if(ep0Bo.Stat & DTS)
            ep0Bo.Stat = UOWN | DTSEN;
        else
            ep0Bo.Stat = UOWN | DTS | DTSEN;
    } else {
    /**
     * Prepare for the Setup stage of a control transfer
    */
        WaitForSetupStage();
    }

else if (USTAT == 0x04) {
    /**
     * Endpoint 0:IN
    */
        if ((UADDR == 0) && (deviceState == ADDRESS)) {
            UADDR = SetupPacket.wValue0;

            if(UADDR == 0)
    /**
     * If we get a reset after a SET_ADDRESS, then we need
     * to drop back to the Default state.
    */
            deviceState = DEFAULT;
        }

if (ctrlTransferStage == DATA_IN_STAGE) {
    /**
     * Start (or continue) transmitting data
    */
        InDataStage();
    /**
     * Turn control over to the SIE and toggle the data bit
    */
        if(ep0Bi.Stat & DTS)
            ep0Bi.Stat = UOWN | DTSEN;
        else
            ep0Bi.Stat = UOWN | DTS | DTSEN;
    } else {
    /**
     * Prepare for the Setup stage of a control transfer
    */
        WaitForSetupStage();
    }
} /* else */

}

/**
 * EnableUSBModule( void ) -
 *
 */
void EnableUSBModule( void )
{
    if (UCONbits.USBEN == 0) {

```

```

        UCON = 0;
        UIE = 0;
        UCONbits.USBEN = 1;
        deviceState = ATTACHED;
    }
    /**
     * If we are attached and no single-ended zero is detected, then
     * we can move to the Powered state.
    */
    if ((deviceState == ATTACHED) && !UCONbits.SEO) {
        UIR = 0;
        UIE = 0;
        UIEbits.URSTIE = 1;
        UIEbits.IDLEIE = 1;
        deviceState = POWERED;
    }
}

/**
 * UnSuspend(void) -
 *
 */
void UnSuspend(void)
{
    UCONbits.SUSPND = 0;
    UIEbits.ACTVIE = 0;
    UIRbits.ACTVIF = 0;
}

/**
 * StartOfFrame(void) -
 *
 * Full speed devices get a Start Of Frame (SOF) packet every 1 millisecond.
 * Nothing is currently done with this interrupt (it is simply masked out).
 */
void StartOfFrame(void)
{
    /**
     * TBD: Add a callback routine to do something
    */
    UIRbits.SOFIF = 0;
}

/**
 * Stall(void) -
 *
 * This routine is called in response to the code stalling an endpoint.
 */
void Stall(void)
{
    if (UEP0bits.EPSTALL == 1) {
        /**
         * Prepare for the Setup stage of a control transfer
        */
        WaitForSetupStage();
        UEP0bits.EPSTALL = 0;
    }
    UIRbits.STALLIF = 0;
}

/**
 * BusReset(void) -
 *
 */
void BusReset()
{
}

```

```
    UEIR   = 0x00;
    UIR    = 0x00;
    UEIE   = 0x9f;
    UIE    = 0x7b;
    UADDR  = 0x00;
    /**
     * Set endpoint 0 as a control pipe
     */
    UEPO = 0x16;
    /**
     * Flush any pending transactions
     */
    while (UIRbits.TRNIF == 1)
        UIRbits.TRNIF = 0;
    /**
     * Enable packet processing
     */
    UCONbits.PKTDIS = 0;
    /**
     * Prepare for the Setup stage of a control transfer
     */
    WaitForSetupStage();
    remoteWakeups = 0;           /* Remote wakeup is off by default */
    selfPowered = 0;             /* Self powered is off by default */
    currentConfiguration = 0;    /* Clear active configuration */
    deviceState = DEFAULT;
}

/**
 * ProcessUSBTransactions(void) -
 *
 * Main entry point for USB tasks.
 * Checks interrupts, then checks for transactions.
 */
void ProcessUSBTransactions(void)
{
    /**
     * See if the device is connected yet.
     */
    if (deviceState == DETACHED)
        return;
    /**
     * If the USB became active then wake up from suspend
     */
    if (UIRbits.ACTVIF && UIEbits.ACTVIE)
        UnSuspend();
    /**
     * If we are supposed to be suspended, then don't try performing any
     * processing.
     */
    if (UCONbits.SUSPND == 1)
        return;
    /**
     * Process a bus reset
     */
    if (UIRbits.URSTIF && UIEbits.URSTIE)
        BusReset();
    /**
     * Process a suspend
     */
    if (UIRbits.IDLEIF && UIEbits.IDLEIE)
    /**
     * Process a SOF
     */
    if (UIRbits.SOFIF && UIEbits.SOFIE)
        StartOfFrame();
    /**

```

```

        * Process a Stall
        */
        if (UIRbits.STALLIF && UIEbits.STALLIE)
            Stall();
        /**
         * Process error - Clear errors
         * TBD: See where it came from.
         */
        if (UIRbits.UERRIF && UIEbits.UERRIE)
            UIRbits.UERRIF = 0;
        /**
         * Unless we have been reset by the host, no need to keep processing
         */
        if (deviceState < DEFAULT)
            return;
        /**
         * A transaction has finished. Try default processing on endpoint 0.
         */
        if (UIRbits.TRNIF && UIEbits.TRNIE) {
            ProcessControlTransfer();
        /**
         * Turn off interrupt
         */
            UIRbits.TRNIF = 0;
        }
    }

#if 0
// Test - put something into EEPROM
code at 0xF00000 word dataEEPROM [] =
{
    0, 1, 2, 3, 4, 5, 6, 7,
    '0', '1', '2', '3', '4', '5', '6', '7',
    '8', '9', 'a', 'b', 'c', 'd', 'e', 'f'
};
#endif

```

11.3.3. Fuente Main

```

/* main.c - The main program of the firmware.
 *
 * Copyright (C) 2008 Rosales Victor (todoesverso@gmail.com)
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

#include <pic18fregs.h>
#include <stdio.h>
#include "usb.h"

/***
 *
```

```

* Constant declaration
*
*/
#define NUMLINES 7

/**
 * Instruction codes from the host
 */
#define RESET      0x00
#define PRINT      0x01
#define MOV_SHORT  0x02
#define MOV_LONG   0x03
#define ROT_SHORT  0x04
#define ROT_LONG   0x05
#define PULLPAPER  0x06
#define STATUS     0x07

/**
 * Types of movements
 */
/* Horizontal */
#define SHORT_OUT 1
#define LONG_OUT  2

#define LEFT      1
#define RIGHT     0

/* Vertical */
#define SHORT_STEPS_OUT 4
#define LONG_STEPS_OUT  7

#define UP        1
#define DOWN     0

/**
 * Physical devices (PORT D)
 */
#define SENS_PAPER PORTDbits.RD3
#define SENS_CAR    PORTDbits.RD2
#define HAMMER     PORTDbits.RD1

#define CAR       1
#define ROLLER    0

/**
 * NOTA:
 *      There are problems related with GET_FEATURE
 *      when working under 48 MHZ
 *
 * Current:
 *      23 mA @ 16 MHz
 *      26 mA @ 24 MHz
 *      35 mA @ 48 MHz
 */
/* Fuses configuration */
#if defined(pic18f4550)
code char at 0x300000 CONFIG1L = 0x20; /* USB clk 96MHz PLL/2, PLL 4MHz */
code char at 0x300001 CONFIG1H = 0x0e; /* HSPLL */
code char at 0x300002 CONFIG2L = 0x20; /* USB regulator enable, PWRT On */
code char at 0x300003 CONFIG2H = 0x00; /* Watchdog OFF */
code char at 0x300004 CONFIG3L = 0xff; /* ***UNUSED*** */
code char at 0x300005 CONFIG3H = 0x81; /* MCLR enabled, PORTB digital */
code char at 0x300006 CONFIG4L = 0x80; /* all OFF except STVREN */
code char at 0x300007 CONFIG4H = 0xff; /* ***UNUSED*** */
code char at 0x300008 CONFIG5L = 0xff; /* No code read protection */
code char at 0x300009 CONFIG5H = 0xff; /* No data/boot read protection */
code char at 0x30000A CONFIG6L = 0xff; /* No code write protection */

```

```

code char at 0x30000B CONFIG6H = 0xff; /* No data/boot/table protection */
code char at 0x30000C CONFIG7L = 0xff; /* No table read protection      */
code char at 0x30000D CONFIG7H = 0xff; /* No boot table protection     */
#endif

/**
 * Buffers for Endpoint 1 (Data bus)
 */
volatile byte txBuffer[INPUT_BYTES];
volatile byte rxBuffer[OUTPUT_BYTES];
volatile byte echoVector[INPUT_BYTES];

/**
 * Buffers for Endpoint 2 (Instruction bus)
 */
volatile byte instruction;

/**
 * UserInit(void) - Initialize the PIC registers
 *
 * Here should be initialized all registers of the PIC.
 * This would be the state needed at boot time.
 */
void UserInit(void)
{
    ADCON0 = 0xFF;           /* Set RA4 as output */          */
    ADCON1 = 0x0F;           /* Set all I/O as digital */     */
    TRISB = 0x00;            /* Set all pins of PORTB as output */
    TRISD = 0xc;             /* Set pins 4,3 as in and 1,2 as out */
    INTCON = 0;              /* Turn off all interrupts */      */
    INTCON2 = 0;             /* Turn off all interrupts */      */
    PORTB = 0x00;            /* Start with motors turned off */ */
    HAMMER = 0;              /* Start with hammer turned off */  */
}

/**
 * delay() - Routine that generates a simple delay
 *
 * @ms:           Amount of cycles that will be executed
 *
 * This routine is not an accurate delay, it was implemented this way because
 * using the internal timer requiers more program space.
 */
void delay(byte ms)
{
    byte i;
    while (ms--)
        for (i = 0; i < 200; i++) /* Experimental value */
}

/**
 * motors_off(void) - Turns off both motors
 *
 * This function just turn off both motors by setting PORTB to 0
 */
void motors_off(void)
{
    PORTB = 0x00;
}

/**
 * move() - Function to move both motors
 * @loops:       Number of loops of a complete sequence
 * @direction:   Direction to move (RIGHT/LEFT, UP/DOWN)
 * @motor:       Which motor to move (CAR/ROLLER)
 *
 * This function moves each motor in a defined direction and an specific

```

```

 * amount of turns depending on the parameters that are passed to it.
 */
void move(byte loops, byte direction, byte motor)
{
    byte stepsI[8] =
        { 0x77, 0x33, 0xbb, 0x99, 0xdd, 0xcc, 0xee, 0x66 };
    byte stepsD[8] =
        { 0x66, 0xee, 0xcc, 0xdd, 0x99, 0xbb, 0x33, 0x77 };
    byte val, i, loops_aux;

    if (direction)
        for (loops_aux = 0; loops_aux < loops; loops_aux++) {
            for (i = 0; i < 8; i++) {
                val = stepsI[i];
                if (motor) {
                    PORTB =
                        (PORTB & 0xf0) | (val & 0x0f);
                    delay(50);
                } else {
                    PORTB =
                        (PORTB & 0x0f) | (val & 0xf0);
                    delay(50);
                }
            }
        } else
            for (loops_aux = 0; loops_aux < loops; loops_aux++) {
                for (i = 0; i < 8; i++) {
                    val = stepsD[i];
                    if (motor) {
                        PORTB =
                            (PORTB & 0xf0) | (val & 0x0f);
                        delay(50);
                    } else {
                        PORTB =
                            (PORTB & 0x0f) | (val & 0xf0);
                        delay(50);
                    }
                }
            }
    }

/**
 * mov_paper() -      Moves the paper up
 * @steps:           Amount of steps to move
 *
 * Moves the paper up according to the input 'steps'
 */
void mov_paper(byte steps)
{
    move(steps, UP, ROLLER);
}

/**
 * reset_car(void) -   Resets the car to the start position
 *
 * Resets the car to the beginning and inserts a left indentation
 */
void reset_car(void)
{
    while (SENS_CAR)
        move(1, LEFT, CAR);
    move(8, RIGHT, CAR); /* Indentation */
}

/**
 * hit(void) - Makes the hammer to hit the paper
 *

```

```

* Generates an impact with delays in the middle.
* This delays should be review when an actual hammer is in place.
*/
void hit(void)
{
    delay(100);
    HAMMER = 1;
    delay(100);
    HAMMER = 0;
    delay(100);
}

/**
 * check_bit() -           Checks the status of a bit in a byte
 * @byte_in:               The byte to check
 * @pos:                   The bit to check
 *
 * Receives a byte an an index for it.
 * Returns 1 if the bit is setted or 0 if is off.
 * (byte, [0-7])
 */
byte check_bit(byte byte_in, byte pos)
{
    byte mascara;
    mascara = 0x01;
    mascara = mascara << pos;
    if (byte_in & mascara)
        return 1;
    return 0;
}

/**
 * set_bit() -   Set a particual bit of a byte
 * @byte_in:     The byte to modify
 * @pos:         The particular bit to set.
 *
 * Used mainly to reconstruct the received byte, this function accepts a pointer
 * to a byte and the particular bit that want to be setted.
 */
void set_bit(byte * byte_in, byte pos)
{
    byte mask = 0x01, byte_aux = *byte_in;
    mask = mask << pos;
    *byte_in = byte_aux ^ mask;
    /**
     * This could be a simple | too, but ^ makes it better
     * because it would destroy the byte if something goes wrong.
     * Let's say that byte_in = 0x07 (0000 0111), and for some
     * reason we try to set the bit 2 the XOR would destroy the byte
     * byte_in = 0000 0111 ^ 0000 0100 = 0000 0011 = 0x03. This should
     * never happen because pos will increment, but you never know ;)
     */
}

/**
 * print_byte() -           Prints the byte revived
 * @p:                      Pointer to the byte that want to be printed
 *
 * The function receives a pointer to a byte and prints it.
 * It returns a control byte recreated in order to send it back to the host
 * to make controls.
 */
byte print_byte(byte * p)
{
    byte a, i, byte_in, byte_ctl = 0x00;
    byte_in = *p;
    for (i = 8; i > 0; i--) {

```

```

        if (check_bit(byte_in, i - 1)) {
            hit();
            set_bit(&byte_ctl, i - 1);      /*Recreate the byte */
        }
        a = (byte) i;
        /**
         * Movment of the car according to odd-even position of the
         * braille dot.
         */
        if (!(a & 1))    /* Checks parity (even = min, odd = max) */
            move(2, RIGHT, CAR);    /* Min separation */
        else
            move(4, RIGHT, CAR);    /* Max separation */
    }
    return byte_ctl;           /* Return control byte */
}

/**
 * print_line() -      Prints a complete line
 * @p:                  Pointer of the line to print
 * @e:                  Pinter to a recreated line
 *
 * Prints a complete line while recreates it in another pointer
 * to send back to the host.
 */
void print_line(byte * p, byte * e)
{
    byte width_b;
    reset_car();
    width_b = NUMLINES - 1;
    while (width_b) {
        *e = print_byte(p);
        p++;
        e++;
        width_b--;
    }
}

/**
 * USB(void) - Main function to process usb transactions
 *
 * This is the main function that process all usb transactions, it decides what
 * to do according to the instructions recived by the host.
 */
static void USB(void)
{
    byte rxCnt;
    /**
     * The code should present a clear distiction between
     * data direction
     */
    /**
     * Find out if an output request has arrived to EP2
     * *** HOST --> DEVICE ***
     */
    rxCnt = BulkOut(2, &instruction, 1);
    if (rxCnt == 0)
        return;

    /**
     * Interpret instructions recived from host
     */
    if (instruction == MOV_SHORT) {
        mov_paper(SHORT_STEPS_OUT);
    } else if (instruction == MOV_LONG) {
        mov_paper(LONG_STEPS_OUT);
    } else if (instruction == PRINT) {

```

```

        do {
            /* *** HOST --> DEVICE *** */
            rxCnt = BulkOut(1, rxBuffer, INPUT_BYTES);
        } while (rxCnt == 0);
        print_line(rxBuffer, echoVector);

        while (ep1Bi.Stat & UOWN)
            ProcessUSBTransactions();
        /**
         * We received the order to PRINT, so we did it .
         * Then we process usb transactions if any.
         * Now we are ready to send the data treated back
         * to the host so he can check if there where
         * any errors.
         */
        /* *** DEVICE --> HOST *** */
        do {
            rxCnt = BulkIn(1, echoVector, INPUT_BYTES);
        } while (rxCnt == 0);

    } else if (instruction == RESET)
        reset_car();

    while (ep1Bi.Stat & UOWN)
        ProcessUSBTransactions();

    motors_off();
}

/**
 * ProcessIO(void) -      Process IO requests
 *
 * This function just checks if a IO has been requested and calls
 * USB() if so.
 */
void ProcessIO(void)
{
    if ((deviceState < CONFIGURED) || (UCONbits.SUSPND == 1))
        return;
    USB();
}

/**
 * main(void) - Main entry point of the firmware
 *
 * This is the main entrance of the firmware and it creates the mail loop.
 */
void main(void)
{
    /**
     * Inits the PIC
     */
    UserInit();
    /**
     * Inits the USB
     *
     * Full-speed mode and sets pull-up internal resistances of PORTB
     * Starts the USB DEATTACHED, no wake ups, and no configured.
     * Configuring the USB is the job of the host.
     */
    UCFG = 0x14;
    deviceState = DETACHED;
    remoteWakeups = 0x00;
    currentConfiguration = 0x00;

    motors_off();
}

```

```

while (1) {
    /**
     * Make sure the USB is available
     */
    EnableUSBModule();
    /**
     * As soon as we get out of test mode (UTEYE)
     * we process USB transactions
     */
    if (UCFGbits.UTEYE != 1)
        ProcessUSBTransactions();
    /**
     * Now we can make our work
     */
    ProcessIO();
}
}

```

11.4. USB Driver

11.4.1. Cabecera de funciones

```

/* functions.h - The header for functions.c.
 *
 * Copyright (C) 2008 Rosales Victor (todoesverso@gmail.com)
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

#include "functions.c"

// Variables globales estaticas

// Cabecera de las funciones

//De uso general
void bprint(byte x);
void fill_buffer(FILE *braille, char *brailleIn);

//De codificacion
byte fill_byte(byte *ptr, byte mask);
void fill_line(byte *ptrOut, byte *ptrIn, int ancho);
byte rep_char(byte caract);
void code(FILE *ascii, FILE *braille);

//De manejo de USB
int start_usb(void);
void stop_usb(void);

int usb_discover(void);

```

```

void _usb_get_string_simple_Manuf(void);
void _usb_get_string_simple_Product(void);
void _usb_get_string_simple_SN(void);
void _usb_claim_interface(void);

```

11.4.2. Fuente de funciones

```

/*  function.c - The main functions used by the driver.
*
*  Copyright (C) 2008  Rosales Victor (todoesverso@gmail.com)
*
*  This program is free software: you can redistribute it and/or modify
*  it under the terms of the GNU General Public License as published by
*  the Free Software Foundation, either version 3 of the License, or
*  (at your option) any later version.
*
*  This program is distributed in the hope that it will be useful,
*  but WITHOUT ANY WARRANTY; without even the implied warranty of
*  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
*  GNU General Public License for more details.
*
*  You should have received a copy of the GNU General Public License
*  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
#include <stdio.h>
#include <usb.h>
#include <errno.h>
#include <time.h>
#include <ctype.h>

/**
 * Global definitions
 */
#define WIDTH 7
#define LONG 4*WIDTH
#define SIZE 840
#define TAM 3*WIDTH

/**
 * USB specific
 */
#define LIBRE 0x04d8 /* vendorId */
#define PRINTER 0x7531 /* deviceId */

/**
 * Specific USB global variables
 */
struct usb_bus *bus;
struct usb_device *dev;
usb_dev_handle *udev;
unsigned int WRITE, READ, WRITE2, READ2;

/**
 * Text (ASCII) files
 */
FILE *ascii, *braille;

/**
 * Global varialbes
 */
int cnt = 0;
typedef unsigned char byte;

```

```

/*
byte brAsciiTabla[] =
{
    /*b7 b6 b5 b4 b3 b2 b1 b0 "Sim"
     */
0x00,0x1b, /*(Espacio)          b7 b6 b5 b4 b3 b2 b1 b0 "Sim" */
0x04,0x17, /*0 0 0 0 0 1 0 0   " "
0x49,0x41, /*0 0 1 1 1 0 0 1   $" %
0x4b,0x02, /*0 0 1 1 1 0 1 1   & '
0x2f,0x1f, /*0 0 1 0 1 1 1 1   ( )
0x21,0x00, /*0 0 1 0 0 0 0 1   *
0x01,0x00, /*0 0 0 0 0 0 0 1   ,
0x11,0x12, /*0 0 0 1 0 0 0 1   .
0x07,0x08, /*0 0 0 0 0 1 1 1   0
0x0a,0x0c, /*0 0 0 0 1 0 1 0   2
0x0d,0x09, /*0 0 0 0 1 1 0 1   4
0x0e,0x0f, /*0 0 0 0 1 1 1 0   6
0x0b,0x06, /*0 0 0 0 1 0 1 1   8
0x25,0x05, /*0 0 1 0 0 1 0 1   :
0x29,0x4f, /*0 0 1 0 1 0 0 1   <
0x16,0x45, /*0 0 0 1 0 1 1 0   >
0x10,0x20, /*0 0 0 1 0 0 0 0   @
0x28,0x30, /*0 0 1 0 1 0 0 0   B
0x34,0x24, /*0 0 1 1 0 1 0 0   D
0x38,0x3c, /*0 0 1 1 1 0 0 0   F
0x2c,0x18, /*0 0 0 1 0 1 1 0   H
0x1c,0x22, /*0 0 0 1 1 1 0 0   J
0x2a,0x32, /*0 0 1 0 1 0 1 0   L
0x36,0x26, /*0 0 1 1 0 1 1 0   N
0x3a,0x3e, /*0 0 1 1 1 0 1 0   P
0x2e,0x1a, /*0 0 0 1 0 1 1 0   R
0x1e,0x23, /*0 0 0 1 1 1 1 0   T
0x2b,0x1d, /*0 0 0 1 0 1 0 1   V
0x33,0x37, /*0 0 0 1 1 0 0 1   X
0x27,0x19, /*0 0 0 1 0 0 1 1   Z
0x2d,0x3d, /*0 0 0 1 0 1 1 0   \
0x14,0x15 /*0 0 0 1 0 1 0 0   ^
                                         0 0 0 1 0 1 0 1   -
                                         ! # % , ) + - / 1 3 5 7 9 ; = ? A C E G I K M O Q S U W Y [ ] _ */

};

/*
***/

/* bprint() - Prints a nice formatted byte
 * @x:           Input byte
 *
 * Prints each bit of a byte to the STDOUT like
 *   0xFF = 1111 1111
 */
void bprint(byte x)
{
    int n;

    for(n = 0; n < 8; n++) {
        if((x & 0x80) != 0) {
            printf("1");
        } else {
            printf("0");
        }

        if (n == 3) {
            printf(" ");
        }
        x = x<<1;
    }

    printf("  ");
}

byte fill_byte(byte *ptr, byte mask)

```

```

{
    byte pkt = 0;
    switch(mask) {
        case 0x30:
            pkt = ((*ptr & mask) << 2) |
                  ((*ptr + 1) & mask) |
                  ((*ptr + 2) & mask) >> 2) |
                  ((*ptr + 3) & mask) >> 4);
            break;
        case 0x0c:
            pkt = ((*ptr & mask) << 4) |
                  ((*ptr + 1) & mask) << 2) |
                  ((*ptr + 2) & mask) |
                  ((*ptr + 3) & mask) >> 2);
            break;
        case 0x03:
            pkt = ((*ptr & mask) << 6) |
                  ((*ptr + 1) & mask) << 4) |
                  ((*ptr + 2) & mask) << 2) |
                  ((*ptr + 3) & mask);
            break;
        default: printf("Mask error");
    }
    /*ptr +=4;*/
    return pkt;
}

void fill_line(byte *ptrOut, byte *ptrIn, int width)
{
    int i,j;
    byte *ptrInA, *ptrOutA;
    ptrInA = ptrIn;
    ptrOutA = ptrOut + 21*cnt;
    byte mask[3] = { 0x30, 0x0c, 0x03 };

    for (i = 0; i < 3; i++) {
        ptrInA = ptrIn;
        /**
         * Fill line until 'width'
         */
        for (j = 0; j < width; j++) {
            *ptrOutA = fill_byte(ptrInA, mask[i]);
            ptrOutA++;
            ptrInA += 4;
        }
    }
}

/**
 * Unit function used in usb_discover()
 */
void _usb_get_string_simple_Manuf(void)
{
    int ret;
    char s[256];

    ret = usb_get_string_simple(udev,
                               dev->descriptor.iManufacturer,
                               s,
                               sizeof(s));

    if (ret > 0)
        printf("Manufacturer: %s\n", s);
    else
        printf("Could not read manufacturer\n");
}

```

```
/**  
 * Unit function used in usb_discover()  
 **/  
void _usb_get_string_simple_Product(void)  
{  
    int ret;  
    char s[256];  
  
    ret = usb_get_string_simple(udev,  
                               dev->descriptor.iProduct,  
                               s,  
                               sizeof(s));  
  
    if (ret > 0)  
        printf("Product: %s\n", s);  
    else  
        printf("Could not read product\n");  
}  
  
/**  
 * Unit function used in usb_discover()  
 **/  
void _usb_get_string_simple_SN(void)  
{  
    int ret;  
    char s[256];  
  
    ret = usb_get_string_simple(udev,  
                               dev->descriptor.iSerialNumber,  
                               s,  
                               sizeof(s));  
  
    if (ret > 0)  
        printf("Serial Number: %s\n", s);  
    else  
        printf("Could not read Serial Number\n");  
}  
  
/**  
 * Unit function used in usb_discover()  
 **/  
void _usb_claim_interface(void)  
{  
    int ret;  
  
    ret = usb_claim_interface(udev, 0);  
    if (ret >= 0)  
        printf("The interface responded.\n");  
    else  
        printf("Error opening interface.\n");  
}  
  
/**  
 * Main usb discovery function  
 **/  
int usb_discover(void)  
{  
    int sizein, sizeout, pts, sizein2, sizeout2;  
    char string[256];  
  
    printf("PIC found, dev-%s in the bus-%s\n",  
          dev->filename,  
          bus->dirname);  
  
    usb_detach_kernel_driver_np(udev, 0);  
    usb_set_configuration(udev, 1);
```

```

if (dev->descriptor.iManufacturer)
    _usb_get_string_simple_Manuf();

if (dev->descriptor.iProduct)
    _usb_get_string_simple_Product();

if (dev->descriptor.iSerialNumber)
    _usb_get_string_simple_SN();

_usb_claim_interface();

WRITE = dev->config[0].interface[0].
    altsetting[0].endpoint[1].
    bEndpointAddress;

READ = dev->config[0].interface[0].
    altsetting[0].endpoint[0].
    bEndpointAddress;

WRITE2 = dev->config[0].interface[0].
    altsetting[0].endpoint[3].
    bEndpointAddress;

READ2 = dev->config[0].interface[0].
    altsetting[0].endpoint[2].
    bEndpointAddress;

sizein = dev->config[0].interface[0].
    altsetting[0].endpoint[1].
    wMaxPacketSize;

sizeout = dev->config[0].interface[0].
    altsetting[0].endpoint[1].
    wMaxPacketSize;

pts = dev->config[0].interface[0].
    altsetting[0].endpoint[0].
    bInterval;

sizein2 = dev->config[0].interface[0].
    altsetting[0].endpoint[2].
    wMaxPacketSize;

sizeout2 = dev->config[0].interface[0].
    altsetting[0].endpoint[2].
    wMaxPacketSize;

printf("EP1\n");
printf("---\n");
printf("\tWRITE:\t%02xh\n", WRITE);
printf("\tREAD :\t%02xh\n", READ);
printf("\tSIZEIN:\t%d\n", sizein);
printf("\tSIZEOUT:\t%d\n", sizeout);
printf("\tE_PTS :\t%d\n\n", pts);

printf("EP2\n");
printf("---\n");
printf("\tWRITE:\t%02xh\n", WRITE2);
printf("\tREAD :\t%02xh\n", READ2);
printf("\tSIZEIN:\t%d\n", sizein2);
printf("\tSIZEOUT:\t%d\n", sizeout2);
}

/**
 * Main function that starts the usb device.

```

```

    */
int start_usb(void)
{
    int n, m, ret;

    usb_init();

    n = usb_find_busses();
    m = usb_find_devices();

    for (bus = usb_busses; bus; bus = bus->next) {
        for (dev = bus->devices; dev; dev = dev->next) {
            if (dev->descriptor.idVendor == LIBRE) {
                if (dev->descriptor.idProduct == PRINTER) {
                    udev = usb_open(dev);
                    if (udev)
                        usb_discover();
                } else {
                    continue;
                }
            }
        }
    }

    if (udev) {
        printf("Device Initialized\n");
        ret = 1;
    } else {
        printf("Device could not be initialized\n");
        ret = 0;
    }
    return ret;
}

/**
 * Function to stop usb device
 */
void stop_usb(void)
{
    usb_release_interface(udev,0);
    usb_close(udev);
}

/**
 * rep_char() - Replace a caracter with the braille table
 * @caract:      byte to replace
 *
 * Accepts an ASCII byte character and returns a replaced
 * byte according to the brAsciiTabla.
 */
byte rep_char(byte caract)
{
    byte charCod, retorna;

    charCod = caract - 0x20;
    retorna = brAsciiTabla[charCod];

    return retorna;
}

void code(FILE *ascii, FILE *braille)
{
    byte caract;

    while(!feof(ascii)) {
        caract = getc(ascii);

```

```

        if (!feof(ascii))
            putc(rep_char(caract), braille);
    }

void fill_buffer(FILE *braille, char *brailleIn)
{
    char caract;
    int i=0;

    while(!feof(braille)) {
        caract = getc(braille);

        if(!feof(braille))
            brailleIn[i] = caract;
        i++;
    }
}

```

11.4.3. Cabecera de opciones

```

/* options.h - The header for options.h.
 *
 * Copyright (C) 2008 Rosales Victor (todoesverso@gmail.com)
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

#include "options.c"

int getopt(char *option, char *value );

```

11.4.4. Fuente de opciones

```

/* options.c - The xml parser for options.
 *
 * Copyright (C) 2008 Rosales Victor (todoesverso@gmail.com)
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 */

```

```

* You should have received a copy of the GNU General Public License
* along with this program. If not, see <http://www.gnu.org/licenses/>.
*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <libxml/xmlmemory.h>
#include <libxml/parser.h>

/**
 * getOpt() - Config file parser
 * @option: Pointer to a string with the tag to look for
 * @value: Pointer to a string with the value in the tag
 *
 * This function receives a pointer to a string with the tag name on it and
 * parse the 'config.cfg' file and returns the pointer to a string with the
 * value of that tag.
 * It has some error handlers, such as 'no file', 'empty file', or 'wrong file',
 * this last one checks for a parent tag '<config>'. In any of those cases it
 * returns a null pointer.
 *
 * It handles only one indentation level.
 */
int getopt(char *option, char *value)
{
    xmlDocPtr doc;
    xmlNodePtr cur;
    xmlChar *key;
    doc = xmlParseFile("config.cfg");
    int ret_val = 0;

    if (doc == NULL) {
        fprintf(stderr, "Document not parsed successfully. \n");
        return 1;
    }

    cur = xmlDocGetRootElement(doc);

    if (cur == NULL) {
        fprintf(stderr, "Empty document\n");
        xmlFreeDoc(doc);
        return 1;
    }

    if (xmlStrcmp(cur->name, (const xmlChar *) "config")) {
        fprintf(stderr,
                "Document of the wrong type, root node != config");
        xmlFreeDoc(doc);
        return 1;
    }

    /**
     * If we arrived here it means that everything is OK
     * so we will look for the tag 'option' and give its value
     * to 'value'. If the value is NULL, the parameter value
     * will be 1 char long with '\0', and the return value will
     * be 1, that means an error occurred, else return 0 (succeeded)
     * so the programmers can make a choice of the default
     * value that option should have.
     */

    cur = cur->xmlChildrenNode;
    while (cur != NULL) {
        if ((!xmlStrcmp(cur->name, (const xmlChar *) option))) {
            key =
                xmlNodeListGetString(doc, cur->xmlChildrenNode,

```

```

        1);

    if (key == NULL) {
        value[0] = '\0';
        ret_val = 1;
    } else
        strcpy(value, (const char *) key);

    xmlFree(key);
    xmlFreeDoc(doc);
    return ret_val;
}
cur = cur->next;
}

return 1;
}

```

11.4.5. Cabecera de manejador de errores

```

/* errorsdrv.c - The error message deliver header.
 *
 * Copyright (C) 2008 Rosales Victor (todoesverso@gmail.com)
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

#include "errorsdrv.c"

int errors(int number);

```

11.4.6. Fuente de manejador de errores

```

/* errorsdrv.c - The error message deliver.
 *
 * Copyright (C) 2008 Rosales Victor (todoesverso@gmail.com)
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.

```

```

/*
#include "options.h"

/**
 * errors() - 'Says' an error
 * @number: Number of the error to 'say'
 *
 * Receives a number that matches a line in the file of errors messages that is
 * defined in the config file 'config.cfg' in the tag '<errors-msg>'. It also
 * gets the language from the config file in the tag '<errors-lang>'.
 *
 * It uses the program 'espeak', so it must be accessible through the PATH
 * variable.
 */
int errors(int number)
{
    FILE *filePtr;
    char file_name[512];
    char line[1024];
    char lang[10];
    int line_number = 0, defa = 0;
    char command[1280] = "espeak -s 120 -p 35 -v ";

    getopt("errors-msg", file_name);
    defa = getopt("errors-lang", lang);

    /**
     * If no language defined use spanish (es) as default
     */
    if (defa == 1)
        strncpy(lang, "es", 2);

    filePtr = fopen(file_name, "rt");

    if (filePtr == NULL) {
        printf("Error opening the file '%s'\n", file_name);
        return 1;
    }

    for (line_number = 0; line_number < number; line_number++)
        fgets(line, sizeof(line), filePtr);

    fclose(filePtr);

    /**
     * Make sure to remove the newline character ...
     */
    char *nlptr = strchr(line, '\n');
    if (nlptr)
        *nlptr = '\0'; /* ... and add the NULL end of the string
                        * otherwise a segfault will occur.
                        */
    strncat(command, lang, (strlen(lang) + 1));
    strncat(command, " ", 2);
    strncat(command, line, strlen(line));
    strncat(command, " ", 1);

    printf("Error: %s\n", line);
    system(command);
    return 0;
}

```

11.4.7. Fuente de archivo principal

```

/*
 * main.c - The main program.
 *
 * Copyright (C) 2008 Rosales Victor (todoesverso@gmail.com)
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

#include "functions.h"
#include "errorsdrv.h"
#include <assert.h>

#define TAM 3*WIDTH

#if 1
#define DBG(fmt, args...) do { \
    printf("%s:%d:%s(", __FILE__, __LINE__, __FUNCTION__); \
    printf(fmt, ##args); printf(")\n"); } while (0)
#else
#define DBG(fmt, args...) /* empty */
#endif

int main(int argc, char *argv[])
{
    int init, j = 0, i, j1 = 0, j2 = 0;
    FILE *ascii;
    char *brailleOut, brailleIn[64], car;
    char file_name[512];
    char value;
    char buff[] = { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06 }, rec, rec2;

    rec = 0x00;
    rec2 = 0x00;

    /* Init USB */
    init = start_usb();

    if (!init) {
        errors(1);
        exit(EXIT_FAILURE);
    }

    if (argc != 2)
        if (getOpt("file-name", file_name)) {
            errors(2);
            exit(1);
        }

    if (argv[1] != NULL)
        ascii = fopen(argv[1], "r");
    else
        ascii = fopen(file_name, "r");

    braille = fopen("/tmp/braille", "w");

```

```
/**  
 * Main loop sending and receiving data  
 * by using  
 * j = usb_bulk_write(udev, WRITE2, &buff[0], 1, 500);  
 * j1 = usb_bulk_read(udev, READ, &rec2, 1, 500);  
 */  
  
/* Stop USB only if it was initialized */  
if (init)  
    stop_usb();  
  
fclose(ascii);  
fclose(braille);  
system("rm /tmp/braille");  
  
return 0;  
}
```

11.5. Hardware

11.5.1. Driver motores paso a paso

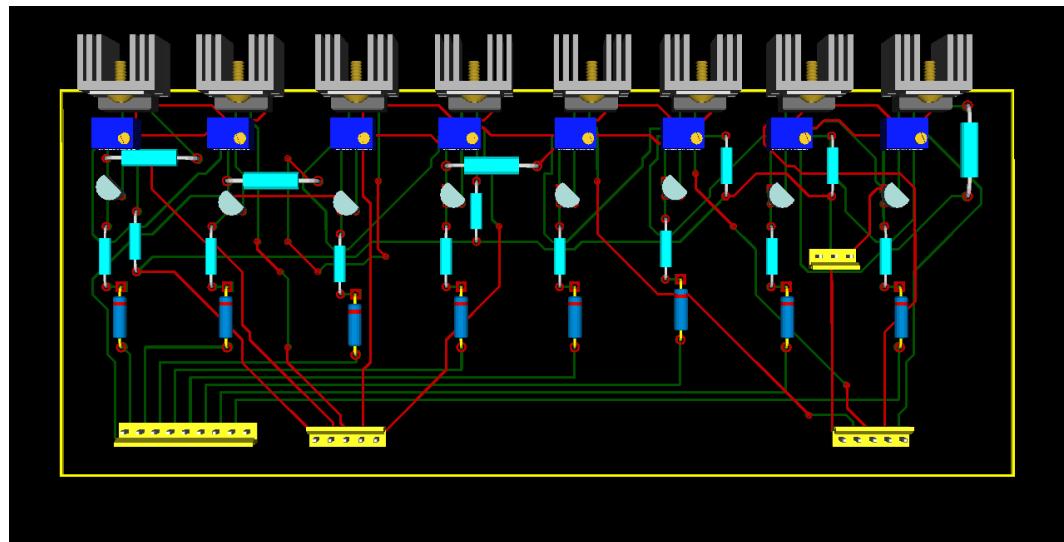


Figura 11.1: Vista superior del driver de los motores paso a paso

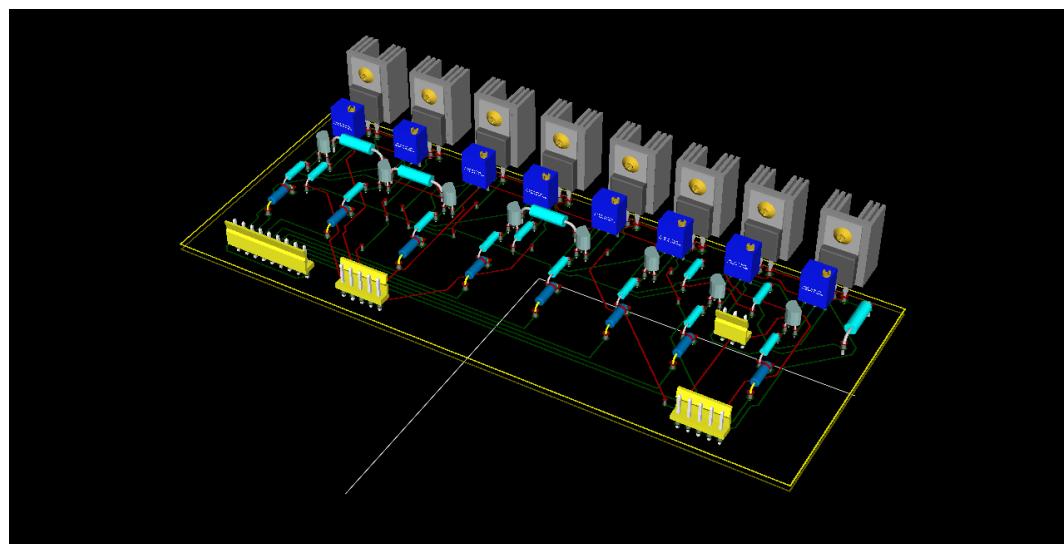


Figura 11.2: Vista en 3D del driver de los motores paso a paso



Figura 11.3: Foto de la placa controladora de motores

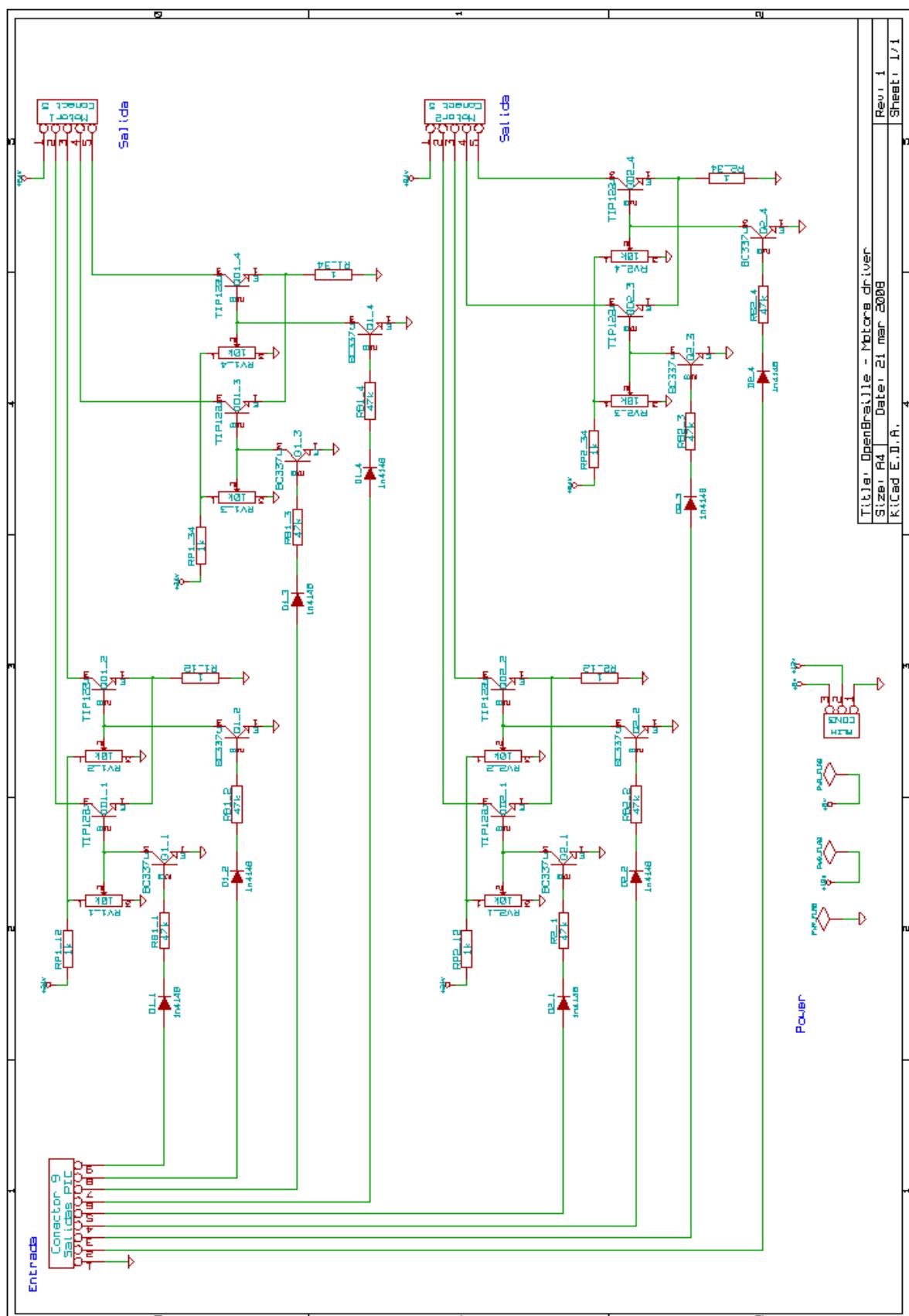


Figura 11.4: Esquemático de la placa del driver de los motores paso a paso

11.5.2. Módulo de procesamiento

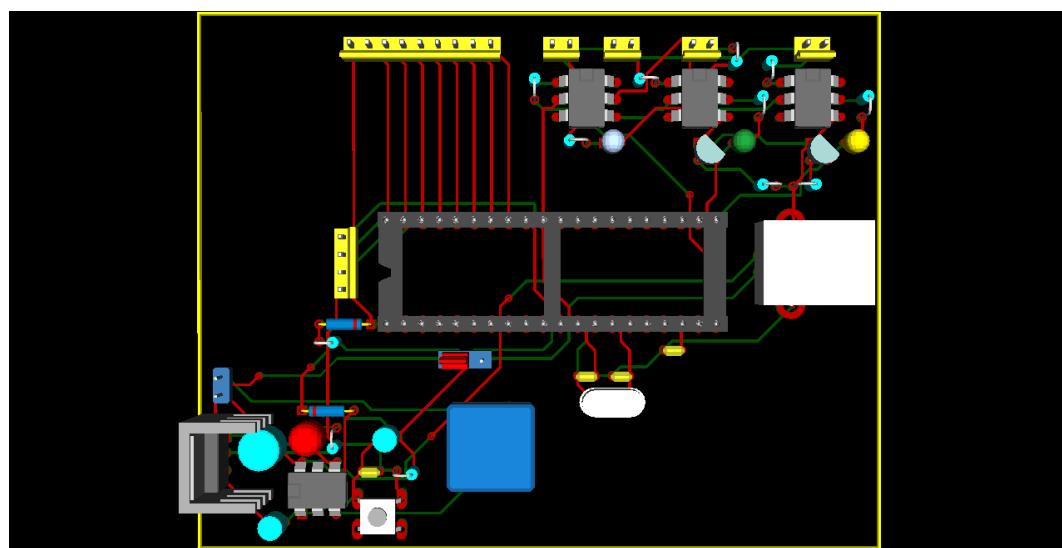


Figura 11.5: Vista superior del la placa de procesamiento

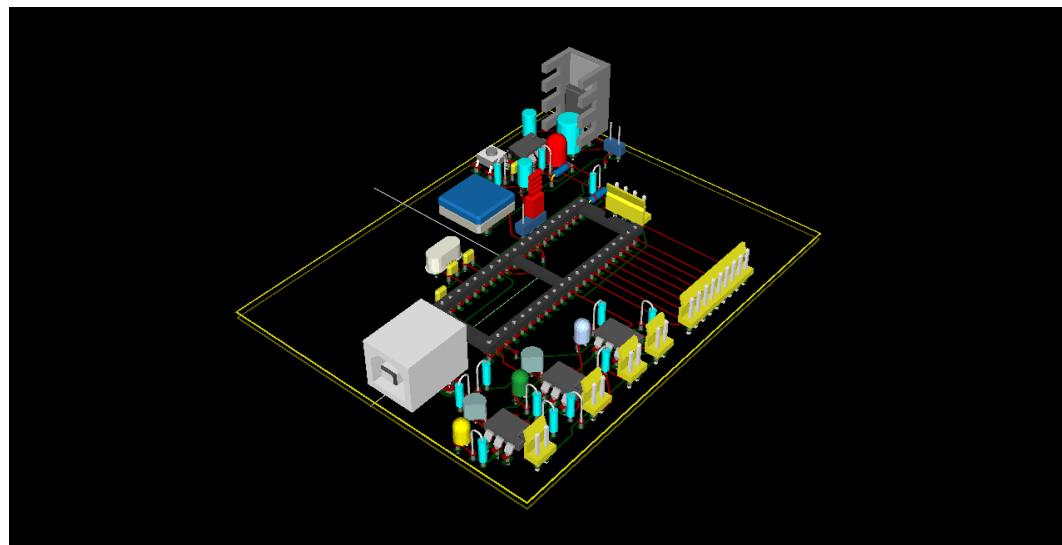


Figura 11.6: Vista en 3D de la placa de procesamiento

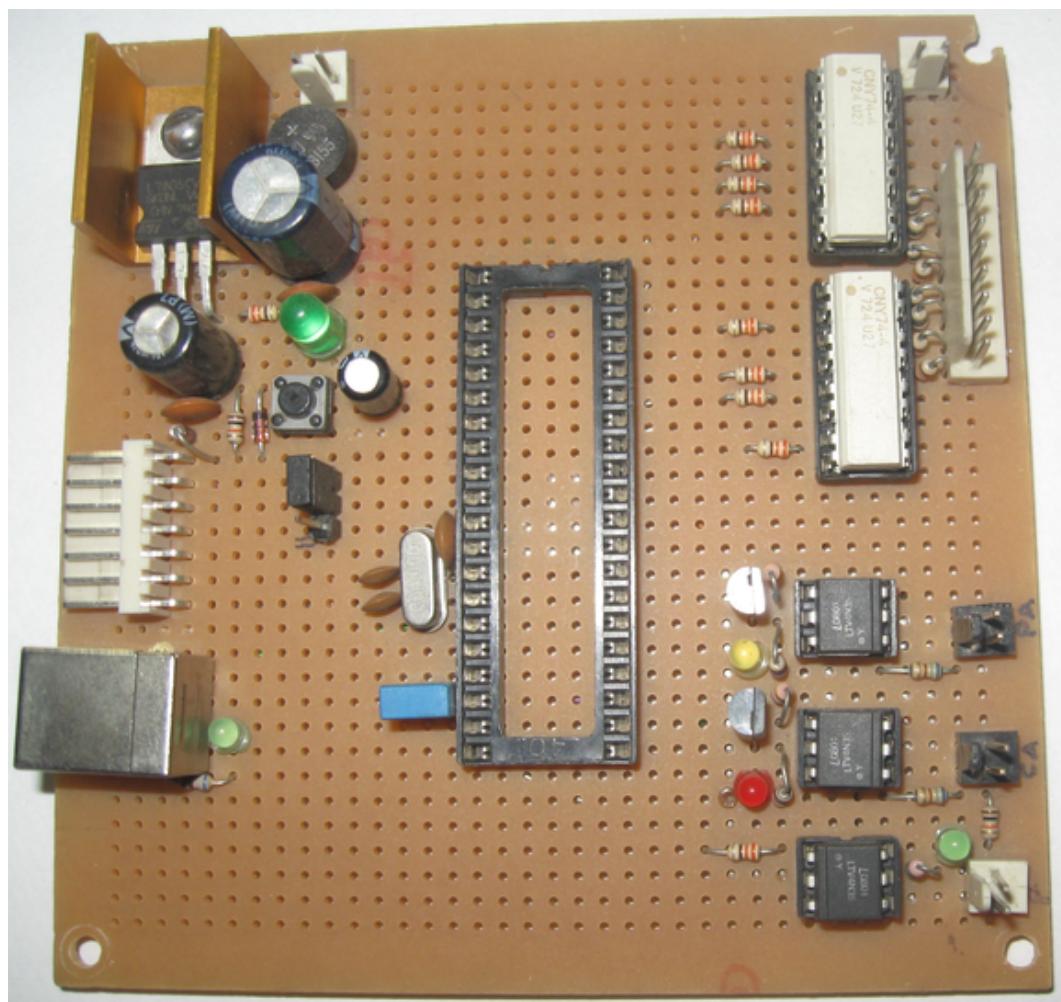


Figura 11.7: Foto de la placa de procesamiento sin el microcontrolador

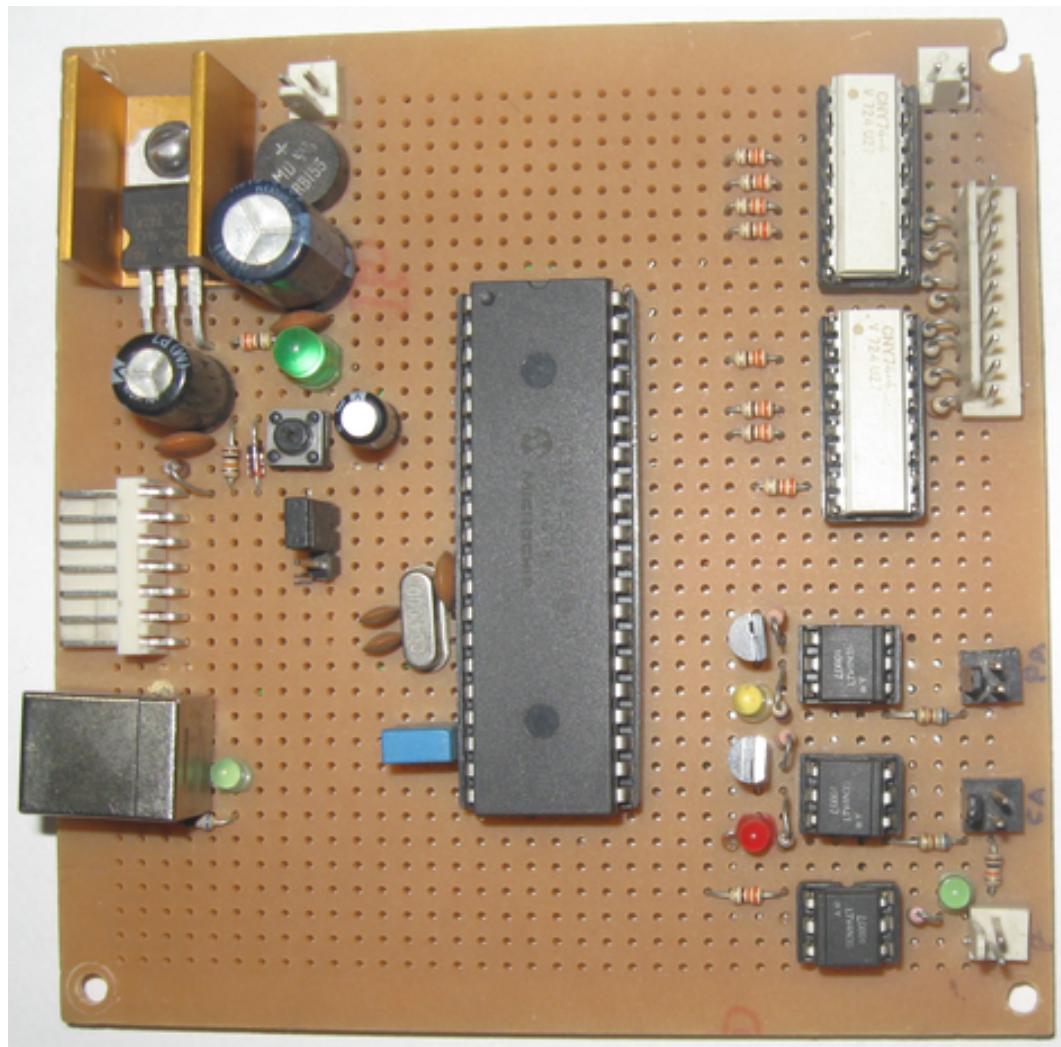


Figura 11.8: Foto de la placa de procesamiento con el microcontrolador

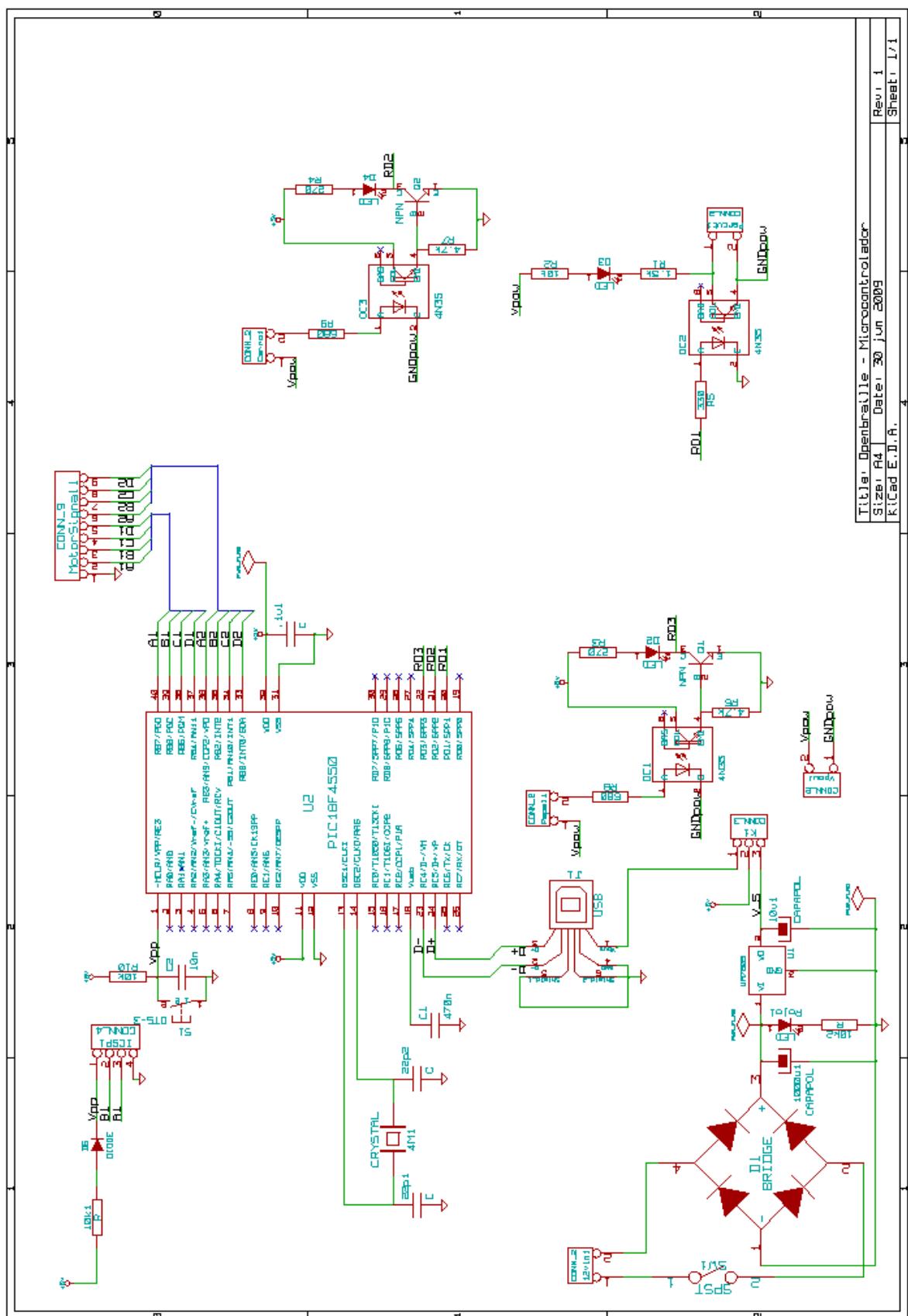


Figura 11.9: Esquemático de la placa del microcontrolador

11.5.3. Bastidor de impresora

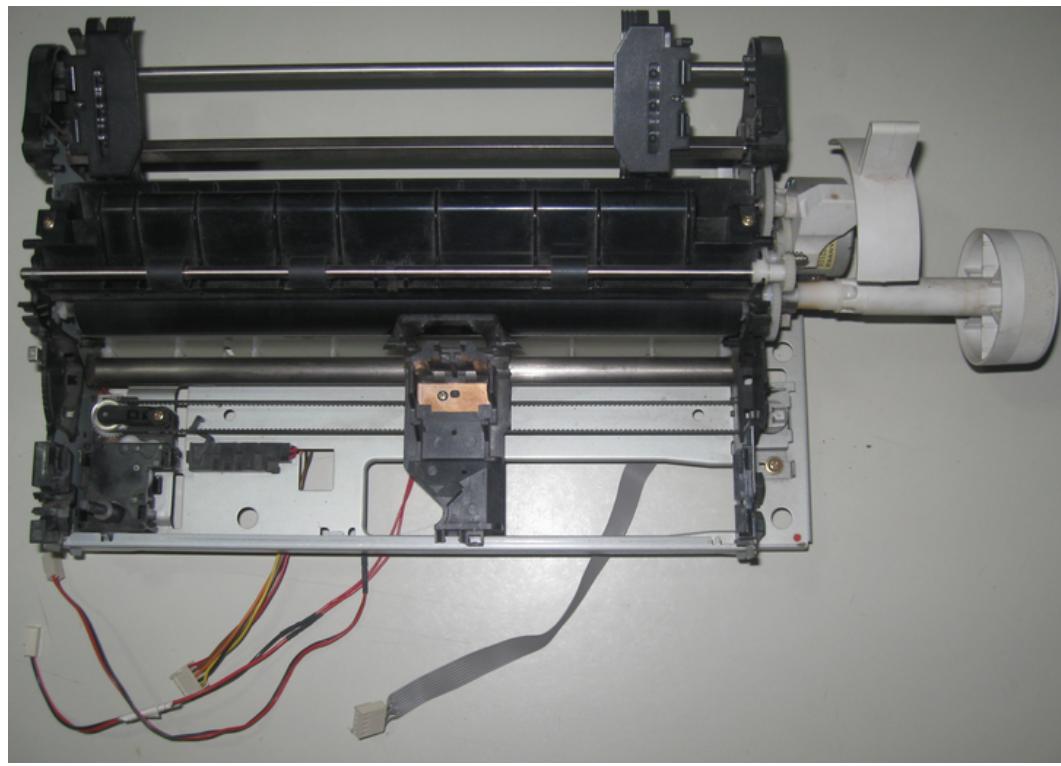


Figura 11.10: Foto del bastidor de una impresora matriz de punto antigua usado para hacer pruebas



Bibliografía

- [1] Brian W. Kernighan y Dennis M. Ritchie: *The C programming Language*. Editorial Prentice-Hall. 1988.
- [2] Herbert Schildt: *C Manual de referencia*. Segunda edición. Editorial Mc Graw-Hill. Madrid, 1990.
- [3] *SDCC Compiler User Guide*. 2009.
<http://sdcc.sourceforge.net/doc/sdccman.pdf>
- [4] usb.org: *Universal Serial Bus Revision 2.0 specification.*, 2000.
http://www.usb.org/developers/docs/usb_20_052709.zip
- [5] Craig Peacock: *USB in a Nutshell. Making Sense of the USB Standard*. beyond-logic.org, 2002.
<http://www.beyondlogic.org/usbnutshell/usb-in-a-nutshell.pdf>
- [6] Sullivan Joseph E.: *DOTSYS III: A Portable Program for Braille Translation*. 1975.
<http://www.eric.ed.gov/ERICWebPortal/contentdelivery/servlet/ERICServlet?accno=ED119734>
- [7] The Sensory Aids Evaluation and Development Center: *Accomplishments, Administrative Structure, and Activities of the Sensory Aids Evaluation and Development Center*. 1967
<http://www.eric.ed.gov/ERICWebPortal/contentdelivery/servlet/ERICServlet?accno=ED028578>