



# **ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL**

Facultad de Ingeniería en Electricidad y Computación

“Control mediante joystick de tarjeta AVR Butterfly  
(con microcontrolador ATmega169) mediante comunicación  
RS232 con tarjeta LPCXpresso controladora de motor BLDC”.

## **TESINA DE SEMINARIO**

Previo a la obtención del Título de:

## **INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES**

Presentado por:

Brian Israel Rosero Ortega  
Cinthia de los Angeles Castelo Barba

GUAYAQUIL – ECUADOR

AÑO 2012

## **AGRADECIMIENTO**

A Dios, por ser la guía de nuestro camino y fuente de esperanza, por colmarnos de bendiciones y brindarnos aliento y sabiduría.

A nuestros padres y hermanos por su apoyo incondicional. Por todos sus consejos que nos brindan cada día, y por ser nuestra fuente de motivación para alcanzar este logro.

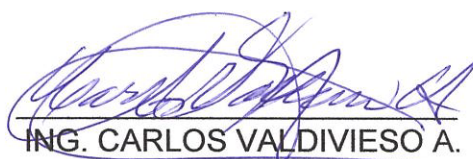
Al Ing. Carlos Valdivieso por brindarnos todo su conocimiento, profesionalismo en el desarrollo y la ejecución del proyecto.

## **DEDICATORIA**

A nuestros padres, pilar fundamental de nuestras vidas presentes durante todo este tiempo de aprendizaje y formación profesional.

A nuestros familiares y amigos que siempre han brindado su palabra de motivación y aliento para lograr alcanzar nuestras metas.

## TRIBUNAL DE SUSTENTACIÓN



ING. CARLOS VALDIVIESO A.

**PROFESOR DE SEMINARIO DE GRADUACIÓN**



ING. HUGO VILLAVICENCIO V.

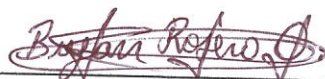
**PROFESOR DELEGADO POR LA UNIDAD ACADÉMICA**



## DECLARACIÓN EXPRESA

“La responsabilidad del contenido de esta Tesina, nos corresponde exclusivamente; y el patrimonio intelectual de la misma, a la Escuela Superior Politécnica del Litoral”.

(Reglamento de Graduación de la ESPOL)



Brian Israel Rosero Ortega



Cinthia de los Angeles Castelo Barba

## RESUMEN

El proyecto consiste en realizar una comunicación alámbrica RS232 entre una tarjeta LPCXpresso 1769 y el kit AVR Butterfly, los cuales deberán controlar los movimientos de un motor DC sin escobillas enviando señales digitales a la tarjeta LPCXpresso 1114 y la LPCXpresso base board.

Para realizar este proyecto debemos programar la tarjeta LPCXpresso 1769 y el microcontrolador Atmega 169 que se encuentra en el interior del kit AVR Butterfly, para esto usaremos lenguaje C en el software LPCXpresso y el AVR Studio 4 respectivamente los cuales son programas proporcionados por los fabricantes y en los cuales se debe manipular los registros respectivos de cada microcontrolador para lograr que las tarjetas realicen las funciones que necesitamos.

La tarjeta LPCXpresso cuenta con puertos de entrada y salida de datos y puertos para varios de tipos de comunicación, mientras que el kit AVR Butterfly es una pequeña tarjeta que cuenta con una pantalla LCD que nos permite visualizar datos, puertos de entrada y salida de datos, puertos para varios tipos de comunicación y un joystick que nos permite realizar movimientos en varias direcciones que nos ayudarán a controlar los movimientos del motor.

# ÍNDICE GENERAL

AGRADECIMIENTO .....	II
DEDICATORIA .....	III
TRIBUNAL DE SUSTENTACIÓN .....	IV
DECLARACIÓN EXPRESA .....	V
RESUMEN.....	VI
ÍNDICE GENERAL .....	VII
ÍNDICE DE FIGURAS.....	XI
ÍNDICE DE TABLAS .....	XIV
INTRODUCCIÓN.....	XV
CAPÍTULO 1	
1 GENERALIDADES .....	1
1.0 RESUMEN DEL CAPÍTULO .....	1
1.1 ANTECEDENTES.....	1
1.2 MOTIVACIÓN .....	3
1.3 IDENTIFICACIÓN DEL PROBLEMA.....	4
1.4 OBJETIVOS PRINCIPALES .....	5
1.5 LIMITACIONES .....	5
CAPÍTULO 2	
2 SUSTENTACIÓN TEÓRICA.....	7
2.0 RESUMEN DEL CAPÍTULO .....	7
2.1 HERRAMIENTAS DE SOFTWARE.....	9
2.1.1 AVR STUDIO 4 .....	9
2.1.2 WINAVR .....	11
2.1.3 LPCXPRESSO.....	12
2.1.3.1 CARACTERÍSTICAS PRINCIPALES .....	13

2.1.3.2 IDE.....	14
2.1.4 PROTEUS versión 7.7 .....	17
2.2 HERRAMIENTAS DE HARDWARE.....	18
2.2.1 KIT DE DESARROLLO AVR BUTTERFLY .....	18
2.2.1.1 CARACTERÍSTICAS DE LA TARJETA AVR BUTTERFLY .....	19
2.2.1.2 HARDWARE DE LA TARJETA AVR BUTTERFLY.....	20
2.2.1.3 FUNCIONAMIENTO DE LA TARJETA AVR BUTTERFLY .....	23
2.2.1.4 MICROCONTROLADORES DE ATMEL .....	25
2.2.1.4.1 CARACTERÍSTICAS DEL MICROCONTROLADOR ATMEGA169.....	27
2.2.1.4.2 DIAGRAMA DE BLOQUES DEL MICROCONTROLADOR AVR ATMEGA169.....	31
2.2.1.4.3 CONFIGURACIÓN DE PINES DEL MICROCONTROLADOR AVR ATMEGA169 .....	32
2.2.2 COMUNICACIÓN RS232.....	35
2.2.2.1 UART .....	36
2.2.2.1.1 INICIALIZACIÓN DE LA INTERFAZ UART .....	40
2.2.2.1.2 DESCRIPCIÓN DE LOS REGISTROS UART .....	45
2.2.3 TARJETA LPCXPRESSO .....	56
2.2.4 MOTORES DE CORRIENTE CONTINUA SIN ESCOBILLAS (BRUSHLESS).....	59
2.2.4.1 CARACTERÍSTICAS FUNDAMENTALES .....	60
2.2.4.1.1 Kv, CARACTERÍSTICAS BÁSICA DE UN MOTOR BLDC.....	61
2.2.4.2 CONSTRUCCIÓN.....	62
2.2.4.3 FUNCIONAMIENTO.....	63
2.2.4.4 VENTAJAS DE LOS MOTORES BLDC .....	64
2.2.4.4.1 DC SIN ESCOBILLAS VS DC CON ESCOBILLAS.....	66
2.2.4.5 DISPARO PARA LOS MOSFETS .....	67
CAPÍTULO 3	
3 TRABAJO REALIZADO .....	69



3.0 RESUMEN DEL CAPÍTULO .....	69
3.1 EJERCICIOS DESARROLLADOS .....	70
3.1.1 EJERCICIO 1.....	70
3.1.1.1 ESPECIFICACIÓN.....	70
3.1.1.2 LISTADO DE COMPONENTES .....	71
3.1.1.3 DIAGRAMA DE BLOQUES.....	71
3.1.1.4 DIAGRAMA DE BLOQUES DE LA CONEXIÓN ELÉCTRICA ...	72
3.1.1.5 DIAGRAMA DE FLUJO.....	73
3.1.1.6 CÓDIGO .....	74
3.1.1.7 IMÁGENES .....	76
3.1.1.8 RESULTADOS.....	77
3.1.2 EJERCICIO 2.....	77
3.1.2.1 ESPECIFICACIÓN.....	77
3.1.2.2 LISTADO DE COMPONENTES .....	78
3.1.2.3 DIAGRAMA DE BLOQUES.....	78
3.1.2.4 DIAGRAMA DE BLOQUES DE LA CONEXIÓN ELÉCTRICA ...	79
3.1.2.5 DIAGRAMA DE FLUJO.....	80
3.1.2.6 CÓDIGO .....	82
3.1.2.7 IMÁGENES .....	85
3.1.2.8 RESULTADOS.....	86
3.1.3 EJERCICIO 3.....	87
3.1.3.1 ESPECIFICACIÓN.....	87
3.1.3.2 LISTADO DE COMPONENTES .....	89
3.1.3.3 DIAGRAMA DE BLOQUES.....	89
3.1.3.4 DIAGRAMA DE BLOQUES DE LA CONEXIÓN ELÉCTRICA ...	90
3.1.3.5 DIAGRAMA DE FLUJO.....	91
3.1.3.6 CÓDIGO .....	92
3.1.3.7 IMÁGENES .....	99
3.1.3.8 RESULTADOS.....	99
3.1.4 EJERCICIO 4.....	100

3.1.4.1	ESPECIFICACIÓN .....	100
3.1.4.2	LISTADO DE COMPONENTES .....	100
3.1.4.3	DIAGRAMA DE BLOQUES .....	101
3.1.4.4	DIAGRAMA DE BLOQUES DE LA CONEXIÓN ELÉCTRICA..	102
3.1.4.5	DIAGRAMA DE FLUJO .....	103
3.1.4.6	CÓDIGO .....	105
3.1.4.7	IMÁGENES .....	111
3.1.4.8	RESULTADOS.....	114
3.2	PROYECTO COMPLETO .....	114
3.2.1	DIAGRAMA DE BLOQUES DEL PROYECTO .....	114
3.2.1.1	TRANSMISOR .....	115
3.2.1.1.1	AVR BUTTERFLY .....	115
3.2.1.1.2	DIAGRAMA DE FLUJO DEL TRANSMISOR.....	117
3.2.1.1.3	CÓDIGO DEL AVR BUTTERFLY .....	118
3.2.1.2	RECEPTOR.....	121
3.2.1.2.1	LPCXpresso .....	121
3.2.1.2.2	DIAGRAMA DE FLUJO DEL RECEPTOR.....	123
3.2.1.2.3	CÓDIGO DE LA TARJETA LPCXPRESSO .....	124
3.2.2	IMÁGENES DEL PROYECTO .....	126
CAPÍTULO 4		
4	PRUEBAS Y SIMULACIONES .....	133
4.0	RESUMEN DEL CAPÍTULO .....	133
4.1	SIMULACIÓN DEL TRANSMISOR .....	133
4.2	SIMULACIÓN DEL RECEPTOR.....	138
4.3	PRUEBAS DEL PROYECTO .....	138
4.4	RESULTADOS DE LA SIMULACIÓN .....	141
CONCLUSIONES		
RECOMENDACIONES		
BIBLIOGRAFÍA		
ANEXOS		

## ÍNDICE DE FIGURAS

Figura 2.1. Partes que conforman el desarrollo de LPCXPRESO .....	14
Figura 2.2. Ambiente de trabajo en el IDE de LPCXPRESSO.....	15
Figura 2.3. Tarjeta LPCXpresso.....	16
Figura 2.4. LPC – link.....	17
Figura 2.5. Kit AVR Butterfly.....	19
Figura 2.6. Parte Frontal del AVR Butterfly.....	22
Figura 2.7. Parte Posterior del AVR Butterfly.....	22
Figura 2.8. Menú Incluido en el AVR Butterfly.....	24
Figura 2.9. Entrada tipo Joystick .....	25
Figura 2.10. Diagrama de Bloques del Microcontrolador ATmega169.....	31
Figura 2.11. Distribución de Pines del ATmega169.....	32
Figura 2.12. Conector RS-232, DE-9 hembra.....	35
Figura 2.13. Diagrama de Bloque de la UART.....	37
Figura 2.14. Diagrama de Bloque de la Lógica de Generación de Reloj .....	39
Figura 2.15. Tarjeta LPCXpresso 1769.....	57
Figura 2.16. Motor BLDC.....	60
Figura 2.17. Despiece de Motor Brushless DC.....	62
Figura 2.18. Bobinado del motor BLDC en conexión estrella.....	64
Figura 2.19. Sensores de efecto Hall.....	64
Figura 2.20. Diagrama de Tiempo .....	66
Figura 2.21. Disparo de Mosfets.....	67

Figura 3.1. Diagrama de bloques del ejercicio 1.....	71
Figura 3.2. Conexión eléctrica del ejercicio 1.....	72
Figura 3.3. Diagrama de flujo del ejercicio 1.....	73
Figura 3.4. Encendido de leds desde el centro hacia los extremos .....	76
Figura 3.5. Diagrama de bloques del ejercicio 2.....	78
Figura 3.6. Conexión eléctrica del ejercicio 2.....	79
Figura 3.7. Diagrama de flujo del transmisor del ejercicio 2.....	80
Figura 3.8. Diagrama de flujo del receptor del ejercicio 2.....	81
Figura 3.9. Comunicación UART con las tarjetas LPCXpresso.....	85
Figura 3.10. Encendido de LED a través de comunicación UART con dos tarjetas LPCXpresso.....	86
Figura 3.11. Diagrama de conexión del joystick.....	88
Figura 3.12. Diagrama de bloques del ejercicio 3.....	89
Figura 3.13. Conexión eléctrica del ejercicio 3.....	90
Figura 3.14. Diagrama de flujo del ejercicio 3.....	91
Figura 3.15. Conexión de Leds del puerto D de la AVR Butterfly.....	99
Figura 3.16. Diagrama de bloques del ejercicio 4.....	101
Figura 3.17. Conexión eléctrica del ejercicio 4.....	102
Figura 3.18. Diagrama de flujo del transmisor del ejercicio 4.....	103
Figura 3.19. Diagrama de flujo del receptor del ejercicio 4.....	104
Figura 3.20. Presionando el botón UP .....	111
Figura 3.21. Presionando el botón DOWN.....	112
Figura 3.22. Presionando el botón RIGHT.....	112
Figura 3.23. Presionando el botón LEFT .....	113
Figura 3.24. Presionando el botón CENTRO.....	113

Figura 3.25. Diagrama de bloques del proyecto.....	115
Figura 3.26. Diagrama de flujo del transmisor del proyecto.....	117
Figura 3.27. Diagrama de flujo del receptor del proyecto.....	123
Figura 3.28. Tarjeta LPCXpresso Motor Control Board y LPCXpresso 1114.....	127
Figura 3.29. Motor BLDC.....	127
Figura 3.30. Proyecto Completo.....	128
Figura 3.31. Motor BLDC en movimiento al presionar UP .....	129
Figura 3.32. Motor BLDC en movimiento al presionar LEFT .....	130
Figura 3.33. Motor BLDC en movimiento al presionar RIGHT.....	131
Figura 3.34. Motor BLDC en movimiento al presionar DOWN.....	132
Figura 4.1. Mensaje de inicio en el LCD de la AVR Butterfly (ROSERO).....	134
Figura 4.2. Mensaje de inicio en el LCD de la AVR Butterfly (CASTELO).....	135
Figura 4.3. Transmisor mostrando la instrucción “CENTRO” en LCD.....	135
Figura 4.4. Transmisor mostrando la instrucción “RIGHT” en LCD.....	136
Figura 4.5. Transmisor mostrando la instrucción “UP” en LCD.....	136
Figura 4.6. Transmisor mostrando la instrucción “DOWN” en LCD.....	137
Figura 4.7. Visualización del caracter de transmisión “UP”.....	137
Figura 4.8. Instrucción UP mostrada en el osciloscopio virtual .....	138
Figura 4.9. Instrucción DOWN mostrada en el osciloscopio virtual.....	139
Figura 4.10. Instrucción LEFT mostrada en el osciloscopio virtual .....	139
Figura 4.11. Instrucción RIGNT mostrada en el osciloscopio virtual .....	140
Figura 4.12. Instrucción CENTRO mostrada en el osciloscopio virtual.....	140

## ÍNDICE DE TABLAS

Tabla 2.1. Familia de Microcontroladores de Atmel.....	26
Tabla 2.2. Registro para Datos de I/O.....	46
Tabla 2.3. Registro A del Uart.....	47
Tabla 2.4. Registro B del Uart .....	50
Tabla 2.5. Registro C del Uart.....	52
Tabla 2.6. Configuración del Bit UMSEL.....	53
Tabla 2.7. Configuración de los Bits UPM.....	53
Tabla 2.8. Configuración del Bit USBS.....	54
Tabla 2.9. Configuración de los Bits UCSZ.....	54
Tabla 2.10. Configuración del Bit UCPOL.....	55
Tabla 2.11. Registro Baud Rate del Uart.....	55
Tabla 2.12. Motor BLDC vs Motor con escobillas.....	66
Tabla 2.13. Códigos según orden de fase de conmutación.....	68
Tabla 3.1. Lista de componentes del ejercicio 1.....	71
Tabla 3.2. Lista de componentes del ejercicio 2.....	78
Tabla 3.3. Lista de componentes del ejercicio 3.....	89
Tabla 3.4. Lista de componentes del ejercicio 4.....	101
Tabla 3.5. Contenido del puntero *UART3BUFFER.....	122

# INTRODUCCIÓN

Para el desarrollo de este proyecto debemos programar nuestra tarjeta LPCXpresso 1769 usando el software LPCXpresso Versión 4.1.5 el cual posee librerías para el uso de los registros que nos permitirán realizar la comunicación con la tarjeta AVR Butterfly y además el envío de datos hacia la LPCXpresso 1114 para el control del motor BLDC.

La programación de la tarjeta AVR Butterfly se la realiza mediante el software AVR Studio 4 en el cual se deberán agregar las librerías para el uso de la LCD, el joystick y los puertos de comunicación debido a que inicialmente el software solo viene con las librerías básicas que nos permiten la entrada y salida de datos.

Para realizar la comunicación entre ambas tarjetas, debemos coincidir con ciertos parámetros como son el baud rate y los puertos físicos de comunicación, en este caso usaremos un baud rate de 9600 y los puertos uart en ambas tarjetas.

La tarjeta AVR Butterfly al realizar movimientos en su joystick deberá enviar datos por medio de su puerto uart en forma de caracteres, estos datos deberán ser recibidos por la tarjeta LPCXpresso 1769 los cuales los

interpretará y generara a la salida niveles de voltaje lógicos bajos, que a su vez llegaran a la tarjeta LPCXpresso 1114 para realizar los movimientos del motor.

Los capítulos en este texto se encuentran estructurados de la siguiente manera:

En el capítulo uno encontraremos una descripción detallada de los antecedentes del proyecto, una breve descripción del problema, aplicaciones que se podrían implementar, la motivación y ciertas limitaciones que tenemos al construir este proyecto.

En el capítulo dos hablaremos de los motores de corriente continua sin escobillas BLDC en donde detallaremos sus características, funcionamiento interno y aplicaciones, también mencionaremos sobre la tarjeta AVR Butterfly su programación, el funcionamiento del software y ciertas características importantes para este proyecto.

Además proporcionamos información sobre los registros, programación, software y librerías de la tarjeta LPCXpresso 1769.

En el capítulo tres detallamos cada uno de los ejemplos que realizamos para familiarizarnos con el funcionamiento de los microcontroladores como son sus registros de entrada o salida de datos y la comunicación entre las tarjetas hasta obtener el proyecto completo.

En el capítulo cuatro indicamos las simulaciones que hemos realizado previo a la implementación del proyecto, en donde indicamos datos de salida y visualización de caracteres a través de la tarjeta AVR Butterfly usando el software Proteus versión 7.7.



# **CAPÍTULO 1**

## **1 GENERALIDADES**

### **1.0 RESUMEN DEL CAPÍTULO**

En este capítulo tomaremos en cuenta los objetivos y factores iniciales para la realización de nuestro proyecto, así como también las expectativas y motivaciones que han causado en nosotros el llevar a cabo esta tesis de grado tomando en cuenta también los inconvenientes y limitaciones que se producen en la construcción del mismo.

### **1.1 ANTECEDENTES**

El desarrollo tecnológico en los últimos años ha sido de mucha importancia en el ámbito de la electrónica, la programación y las telecomunicaciones; lo cual ha permitido la creación de muchos dispositivos electrónicos de gran utilidad en la vida diaria y de objetos que en años anteriores parecían muy

lejanos a la realidad y un gran ejemplo de aquello es el desarrollo de los robots.

El avance en la robótica dio paso a que muchos dispositivos sean creados con el fin de que los seres humanos tengan herramientas que ayuden a llevar una vida más fácil, es así como aparecieron muchos mecanismos de control tanto en el ámbito industrial como en el hogar mismo mediante controladores programables, como son el control

de motores, control de procesos industriales automatizados, alarmas y un sin número de herramientas que se las usa a diario.

Por eso la industria en el Ecuador también ha tenido un gran desarrollo en sus sistemas de operación el cual ha llevado a que los trabajadores de estas empresas tengan conocimiento completo del manejo de los sistemas de control programados por software para de esta manera controlar, reparar e implementar nuevos sistemas y nuevos procesos que ayuden a crear nuevos y mejores productos para estas empresas.

Y es por eso que nuestro proyecto tiene como objetivo demostrar el control mediante el joystick de la tarjeta AVR Butterfly a través de la comunicación RS232 con la tarjeta LPCXpresso controladora de motor BLDC, que está basada en el kit de desarrollo Avr Butterfly, la cual permitirá verificar los recursos disponibles en la comunicación UART y desarrollar muchos

proyectos, el Kit de desarrollo consta de un micro-controlador integrado el Atmega169, de un Joystick que nos permite la interacción con la tarjeta LPCXpresso y de otra funciones muy óptimas para la realización de proyectos.

## **1.2 MOTIVACIÓN**

Crear un proyecto en el cual podamos poner en práctica muchos de los conocimientos adquiridos durante nuestra vida universitaria y que nos permite descubrir cómo trabajan los dispositivos electrónicos y de comunicaciones programables nos causa mucho entusiasmo, ya que sentimos que muchos conocimientos aprendidos durante largos años tienen sus frutos.

Sentir que somos capaces de construir un circuito electrónico aplicable y que podemos ponerlo en funcionamiento para que la sociedad pueda darle uso con mucha utilidad es muy gratificante.

Adquirir conocimientos de hardware y software modernos que están siendo usados por países desarrollados para de esta manera emplearlo en nuevas y modernas aplicaciones en la industria que ayuden a elevar la producción del país el cual sea de gran beneficio en la economía del país.

### **1.3 IDENTIFICACIÓN DEL PROBLEMA**

La ausencia de tecnología en nuestro país ha motivado a que profesionales y estudiantes especializados en electrónica y comunicaciones empiecen a utilizar instrumentos y dispositivos electrónicos que puedan contribuir con el desarrollo del país, brindando así mejoras en el estilo de vida para aquellas personas que pueden tener acceso a estas tecnologías y que tiene como propósito que en un futuro todas las personas puedan ser capaces de adquirir herramientas tecnológicas que hoy en día son muy necesarias.

Para la utilización de estas nuevas tecnologías se ha tenido que realizar muchos estudios y preparación académica para lograr entender de forma correcta el funcionamiento de todos estos dispositivos con el fin de sacar de ellos el mayor provecho, tal cual hemos realizado en la construcción de nuestro proyecto de grado.

Muchas de las empresas del país no cuentan con sistemas de producción modernos, lo cual hace que seamos incompetentes ante productores rivales de países mas dotados tecnológicamente y esto origina a que las probabilidades de venta sean bajas y que a nivel mundial nuestros productos sean poco requeridos, por eso es necesario implementar proyectos como el que hemos llevado a cabo para que luego sean aplicados en la industria y

que ayuden a tener empresas mejor dotadas tecnológicamente para lograr competir a la par con los países vecinos.

#### **1.4 OBJETIVOS PRINCIPALES**

- Crear un dispositivo capaz de ser de gran utilidad para la sociedad.
- Desarrollar nuestra capacidad de operar circuitos electrónicos programables.
- Aprender a usar herramientas de software para el manejo de las tarjetas electrónicas Avr Butterfly y LPCxpresso las cuales van a ser usadas en nuestro proyecto.
- Controlar mediante el joystick de la tarjeta Avr Butterfly el funcionamiento de un motor BLDC a través de comunicación RS232 entre la Avr Butterfly y la LPCxpresso.

#### **1.5 LIMITACIONES**

El desarrollo de nuevas tecnologías en el país es casi nulo, por lo que obtener dispositivos electrónicos de última tecnología es muy complicado para aquellas personas que quieren emprender y usar estas herramientas de actualidad.

Por ello es necesario importar todos estos instrumentos de países desarrollados que cuentan con normalidad con toda esta tecnología, lo cual es un tanto costoso y muchas veces es inaccesible para personas que no poseen altos recursos económicos.

La programación de los controladores de estos dispositivos electrónicos es compleja, ya que en muchas ocasiones es necesario usar un lenguaje ensamblador que es de alta dificultad o cuando se usan programas de alto nivel, no existe un algoritmo definido o una estructura básica para el desarrollo de estos programas, por lo cual es necesario recurrir a ejemplos creados por los fabricantes y esto conlleva a tener que entender sus librerías y programas para lograr construir un software acorde a nuestras necesidades.

Además nuestro proyecto es muy sensible en su manipulación ya que podemos inducirle estática y podemos dañarlo en su totalidad; contiene elementos muy delicados que con cualquier golpe pueden ser averiados parcial o completamente.

# **CAPÍTULO 2**

## **2 SUSTENTACIÓN TEÓRICA**

### **2.0 RESUMEN DEL CAPÍTULO**

En este capítulo detallamos cada una de las herramientas tanto de hardware y de software, las cuales trabajan en conjunto y que son necesarias para la construcción de nuestro proyecto.

A nivel de hardware usaremos 3 elementos principales, como son el kit de desarrollo AVR Butterfly, la tarjeta LPCXpresso, y los motores BLDC.

El kit AVR Butterfly consta principalmente de una tarjeta de desarrollo la cual esta compuesta de una pantalla LCD que ayuda a mostrar datos, un joystick y el microcontrolador Atmega 169 en el cual se ingresará un respectivo código para de esta manera darle la aplicación adecuada a la tarjeta.

También hablaremos de la tarjeta LPCXpresso, que posee un microcontrolador ARM Cortex-M3 que tiene registros el cual nos permite manejar entradas y salidas ya sean analógicas o digitales, además podemos realizar comunicaciones entre una o varias tarjetas que nos brindan un sin número de opciones dependiendo el tipo de aplicación que necesitamos.

El dispositivo final en la construcción del proyecto es un motor BLDC, es un tipo de motor especial que nos brinda una mayor eficiencia y que para aplicaciones industriales representa un gran ahorro a nivel de consumo eléctrico.

En software usaremos cuatro programas principales, los cuales nos ayudan a programar y a simular los circuitos con las tarjetas respectivas que mencionamos anteriormente.

Usaremos el software AVR Studio 4, que nos permite programar la tarjeta AVR Butterfly para de esa manera brindarle su respectiva aplicación, hay que destacar que este software trabaja en lenguaje ensamblador por lo que muchas veces es complicado realizar aplicaciones en esta plataforma; por eso hemos usado el software WIN AVR que nos permite trabajar en un ambiente más práctico, ya que trabaja con lenguajes de alto nivel como son lenguaje C, VHDL, visual Basic, Matlab, etc. Con este software logramos que



el AVR Studio 4 reconozca el lenguaje C, en el cual trabajaremos para el desarrollo del proyecto.

El programa LPCXpresso nos ayuda a programar en lenguaje C todos los registros que contiene nuestra tarjeta LPCXpresso para crear de esa manera la respectiva aplicación que necesitamos.

Por último hablaremos del simulador Proteus 7.7 en el cual simularemos el funcionamiento de la tarjeta AVR Butterfly, también veremos el comportamiento y funcionamiento de los motores BLDC que nos ayudara a verificar la construcción de los respectivos software previo a las pruebas físicas reales en las tarjetas de desarrollo.

## **2.1 HERRAMIENTAS DE SOFTWARE**

Para desarrollar el proyecto hicimos uso de cuatro tipos de software: AVR Studio 4, cuyo fin es la programación del ATmega169, WinAVR, LPCXpresso para la programación de dicha tarjeta y Proteus.

### **2.1.1 AVR STUDIO 4**

AVR Studio es un Entorno de Desarrollo Integrado (IDE) para escribir y depurar aplicaciones AVR en el entorno de Windows 9x/Me/NT/2000/XP.

AVR Studio4 soporta varias de las fases por las cuales se atraviesa al crear un nuevo producto basado en un microcontrolador AVR.

Las fases típicas son:

- ✓ El producto que debe crearse se define basándose en el conocimiento de la tarea que se quiere resolver y la entrada que tendrá posteriormente en el mercado.
- ✓ La especificación formal para el producto.
- ✓ A un equipo del proyecto, que consiste de una o más personas, se le asigna la tarea de crear el producto basándose en la especificación formal.
- ✓ El equipo del proyecto pasa por la secuencia normal de diseño, desarrollo, depuración, comprobación, planificación de producción, producción, prueba y embarque.

AVR Studio es actualizado continuamente y está disponible para descargarlo desde [www.atmel.com](http://www.atmel.com)

Los requerimientos mínimos del sistema que son necesarios para poder utilizar el AVR Studio 4 en una PC:

- Windows 2000/XP (o Windows NT 4.0 con Internet Explorer 5.0 o posterior).

- Windows 95/98/Me (con Internet Explorer 5.0 o posterior).
- Hardware Recomendado:
  - Procesador Intel Pentium de 200MHz o equivalente.
  - Resolución de Pantalla de 1024x768 (Resolución mínima de 800x600).
  - Memoria RAM de 64Mb.
  - Disco Duro con 50 Mb de espacio disponible.

### **2.1.2 WINAVR**

WinAVR es un conjunto de herramientas de desarrollo para microcontroladores RISC AVR de Atmel, basado en software de código abierto y compilado para funcionar en la plataforma Microsoft Windows.

WinAVR incluye las siguientes herramientas:

- avr-gcc, el compilador de línea de comandos para C y C++.
- avr-libc, la librería del compilador que es indispensable para avr-gcc.
- avr-as, el ensamblador.

- avrdude, la interfaz para programación.
- avarice, la interfaz para JTAG ICE.
- avr-gdb, el depurador.
- Programmers Notepad, el editor.
- MFile, generador de archivo makefile.

Para obtener gratuitamente WinAVR, es necesario visitar la página Web <http://sourceforge.net/projects/winavr> y descargar la última versión de este software.

### **2.1.3 LPCXPRESSO**

LPCXpresso <sup>™</sup> es una nueva plataforma de bajo costo de desarrollo disponible de NXP. Es compatible con microcontroladores basados en ARM LPC de NXP, la plataforma se compone de un simplificado programa basado en Eclipse IDE y de bajo costo, incluyen un depurador JTAG adjunto. Diseñado para la simplicidad y facilidad de uso, el LPCXpresso IDE (alimentado por el Código Rojo (code red)) proporciona a los ingenieros de software una manera rápida y fácil para desarrollar sus aplicaciones. Las tablas de destino LPCXpresso son desarrolladas en conjunto por empresas integradas, Code Red y NXP.

### 2.1.3.1 CARACTERÍSTICAS PRINCIPALES

- NXP crea plataformas de bajo costo desarrolladas para las familias de LPC.
- IDE basado en Eclipse con un muy bajo costo para las tarjetas.
- Las tarjetas vienen con un depurador JTAG integrado. No hay necesidad de un depurador por separado.
- Formas sencillas de actualizar el software que manejan las tarjetas (basados en code red)

Los usuarios pueden visualizar tres etapas, desde la evaluación hasta el desarrollo de productos. Durante la evaluación, características y periféricos de la tarjeta MCU se puede probar fácilmente con el área de creación de prototipos y con conexiones de fácil acceso en la tarjeta.

Como complemento de las tarjetas, también nos facilitan el uso de ejemplos de proyectos y constan de una práctica Guía de introducción. Para explorar rápidamente una prueba de concepto, los usuarios pueden obtener una tarjeta fuera de la base de la plataforma de artistas integrados que proporciona una gran variedad de interfaces y dispositivos IO. Por último, los usuarios pueden desarrollar sin problemas LPCXpresso su aplicación final mediante el uso de la LPC-Link 10-pin JTAG para conectar cualquier tablero personalizado JTAG-capaz. De esta manera, los usuarios pueden ahora

disfrutar de la misma interfaz de usuario desde la evaluación para el desarrollo de productos.



Figura 2.1. Partes que conforman el desarrollo de LPCXPRESSO (Ref. [2])

### 2.1.3.2 IDE

IDE de LPCXpresso (alimentado por Code Red) es un entorno de software altamente integrado de desarrollo para microcontroladores LPC de NXP, que incluye todas las herramientas necesarias para desarrollar soluciones de alta calidad de software de una manera efectiva en tiempo y costo. LPCXpresso está basado en Eclipse con muchas mejoras específicas de LPC. También cuenta con la última versión de la cadena de herramientas GNU estándar de la industria con una biblioteca propia optimizado C que proporciona herramientas profesionales de calidad a bajo costo. El IDE LPCXpresso puede construir un ejecutable de cualquier tamaño con la optimización de código completo, y es compatible con un límite de descarga de 128 KB después del registro.

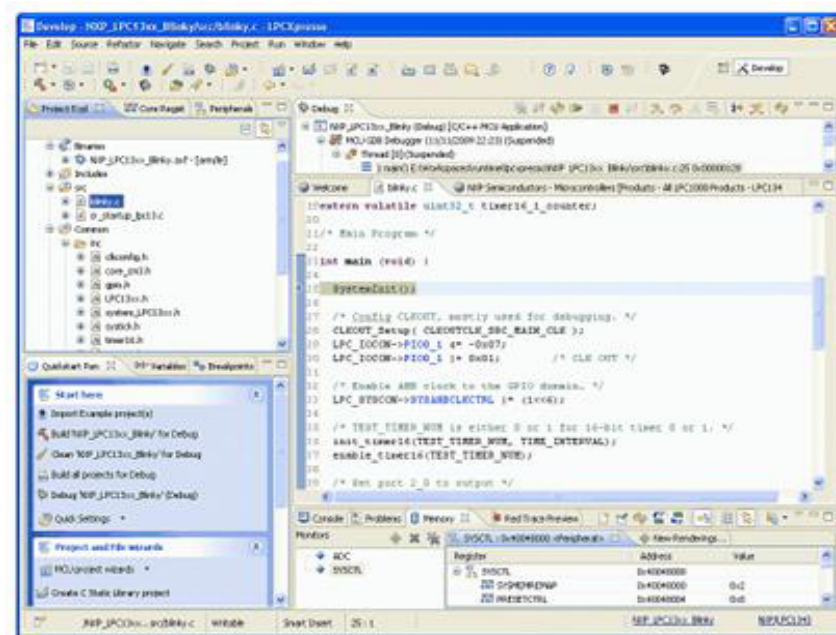


Figura 2.2. Ambiente de trabajo en el IDE de LPCXPRESSO (Ref.[2])

## Tarjetas de desarrollo

Las tarjetas LPCXpresso, desarrollados conjuntamente por NXP y Code Red, incluyen un depurador JTAG integrado, así que no hay necesidad de un depurador JTAG por separado. La porción de destino de la placa se puede conectar a las tarjetas de expansión para proporcionar una variedad de interfaces y dispositivos de E / S.

El depurador JTAG proporciona un USB de alta velocidad para JTAG / SWD interfaz para el IDE y puede ser conectado a dispositivos de depuración,

tales como un prototipo del cliente. Los usuarios también pueden usar el IDE LPCXpresso con el adaptador JTAG de Código Rojo.

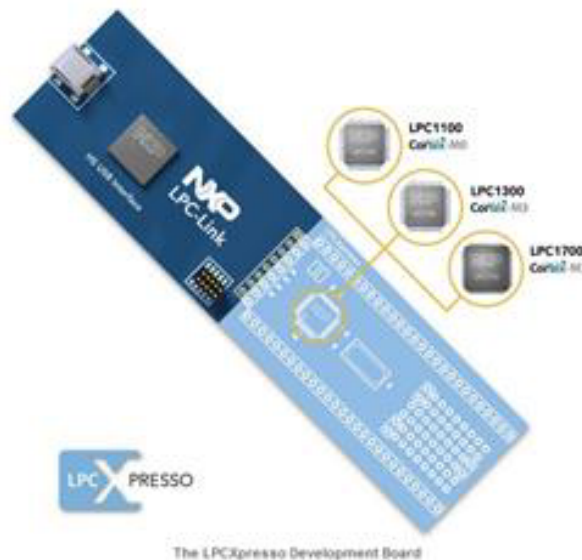


Figura 2.3. Tarjeta LPCXpresso (Ref. [2])

## LPC – Link

La parte JTAG / SWD de depuración de una tabla de LPCXpresso se llama el LPC-Link <sup>TM</sup>. La LPC-Link está equipado con una cabecera de 10 pines JTAG, y se conecta sin problemas a una tarjeta objetivo a través de USB (la interfaz USB y otras características de depuración son proporcionados por ARM9 basado en NXP LPC3154 MCU). Las huellas de corte entre el LPC-link y el objetivo será establecer el vínculo LPC-un solo depurador JTAG. Esto permite a la plataforma LPCXpresso para ser conectado a un destino



exterior y se utiliza para desarrollar para una amplia variedad de NXP Cortex-M0, Cortex-M3, y aplicaciones ARM7 / 9.



Figura 2.4. LPC – Link (Ref.[2])

#### 2.1.4 PROTEUS versión 7.7

Proteus versión 7.7 es un entorno completo diseñado para la realización de cualquier clase de proyecto que conste de elementos electrónicos en todas sus etapas: Diseño, Simulación y Construcción.

Consta de dos partes:

**ISIS.-** Permite diseñar y simular cualquier clase de proyecto electrónico para constatar su funcionamiento.

**Ares.-** Permite el diseño de ruteo del proyecto para ahora si plasmarlo en una placa electrónica. Esta herramienta te ayuda a buscar la mejor forma de cómo colocar los elementos y las líneas del circuito en la placa.

## **2.2 HERRAMIENTAS DE HARDWARE**

Los componentes físicos que utilizamos para la realización del proyecto fueron: el Kit AVR Butterfly, tarjeta LPCXpresso, interfaz uart y el motor BLDC

### **2.2.1 KIT DE DESARROLLO AVR BUTTERFLY**

El Kit AVR Butterfly se diseñó para demostrar los beneficios y las características importantes de los microcontroladores ATMEL.

El AVR Butterfly utiliza el microcontrolador AVR ATmega169V, que combina la Tecnología Flash con el más avanzado y versátil microcontrolador de 8 bits disponible. En la Figura 5 se puede observar el Kit AVR Butterfly.



Figura 2.5. Kit AVR Butterfly

### 2.2.1.1 CARACTERÍSTICAS DE LA TARJETA AVR BUTTERFLY

El Kit AVR Butterfly expone las siguientes características principales:

- La arquitectura AVR en general y la ATmega169 en particular.
- Diseño de bajo consumo de energía.
- El encapsulado tipo MLF.
- Periféricos:
  - Controlador LCD.
  - Memorias:
    - Flash, EEPROM, SRAM.
    - Data Flash externa.
- Interfaces de comunicación:

- UART, SPI, USI.
- Métodos de programación
  - Self-Programming/Bootloader, SPI, Paralelo, JTAG.
- Convertidor Analógico Digital (ADC).
- Timers/Counters:
  - Contador de Tiempo Real (RTC).
  - Modulación de Ancho de Pulso (PWM).

El AVR Butterfly está proyectado para el desarrollo de aplicaciones con el ATmega169 y además puede usarse como un módulo dentro de otros productos.

#### **2.2.1.2 HARDWARE DE LA TARJETA AVR BUTTERFLY**

Los siguientes recursos están disponibles en el Kit AVR Butterfly:

- Microcontrolador ATmega169V (en encapsulado tipo MLF).
- Pantalla tipo vidrio LCD de 120 segmentos, para demostrar las capacidades del controlador de LCD incluido dentro del ATmega169.
- Joystick de cinco direcciones, incluida la presión en el centro.
- Altavoz piezoeléctrico, para reproducir sonidos.
- Cristal de 32 KHz para el RTC.

- Memoria DataFlash de 4 Mbit, para el almacenar datos.
- Convertidor de nivel RS-232 e interfaz USART, para comunicarse con unidades fuera del Kit sin la necesidad de hardware adicional.
- Termistor de Coeficiente de Temperatura Negativo (NTC), para sensor y medir temperatura.
- Resistencia Dependiente de Luz (LDR), para sensor y medir intensidad luminosa.
- Acceso externo al canal 1 del ADC del ATmega169, para lectura de voltaje en el rango de 0 a 5 V.
- Emulación JTAG, para depuración.
- Interfaz USI, para una interfaz adicional de comunicación.
- Terminales externas para conectores tipo Header, para el acceso a periféricos.
- Batería de 3 V tipo botón (600mAh), para proveer de energía y permitir el funcionamiento del AVR Butterfly.
- Bootloader, para programación mediante la PC sin hardware especial.
- Compatibilidad con el Entorno de Desarrollo AVR Studio 4.

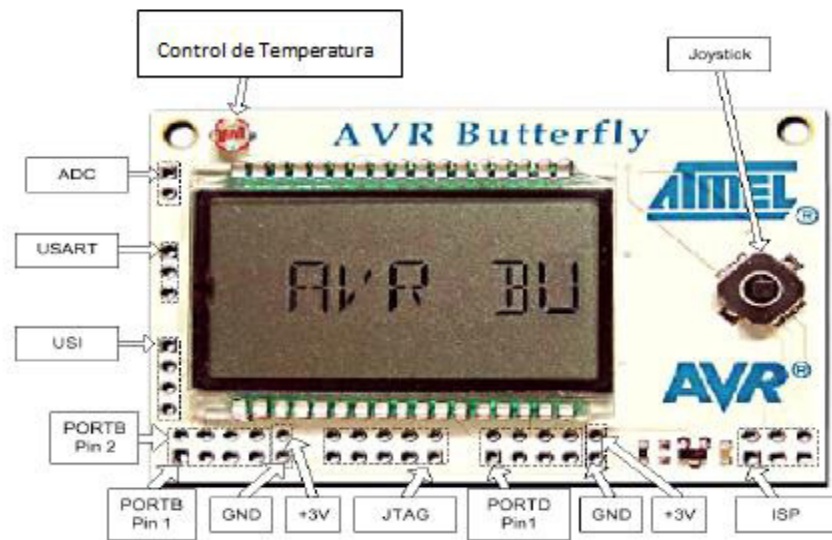


Figura 2.6. Parte Frontal del AVR Butterfly (Ref.[3])

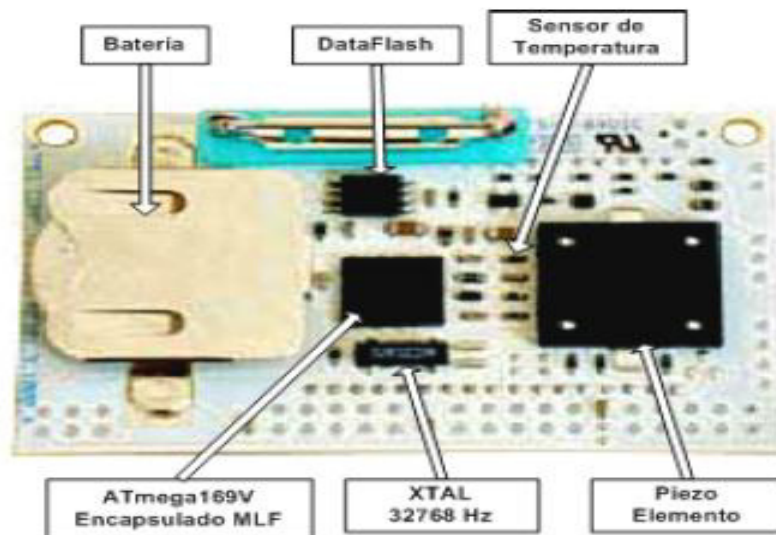


Figura 2.7. Parte Posterior del AVR Butterfly (Ref.[3])

### **2.2.1.3 FUNCIONAMIENTO DE LA TARJETA AVR BUTTERFLY**

#### **Menú**

El sistema de menú está creado para poder desplazarse de manera eficiente, entre los diferentes módulos de la aplicación.

La Figura 2.8 muestra el menú de la aplicación que viene con el AVR Butterfly.

Para moverse entre las alternativas del menú, se debe presionar el joystick hacia ARRIBA o hacia ABAJO. Para entrar en un submenú, se debe presionar el joystick hacia la DERECHA. Para salir de un submenú, se debe presionar el joystick hacia la IZQUIERDA. Para ingresar/ajustar un valor, se debe presionar ENTRAR. Por ejemplo, cuando aparece “Ajustar reloj” en el LCD, se debe presionar ENTRAR para ingresar a la función ajustar.

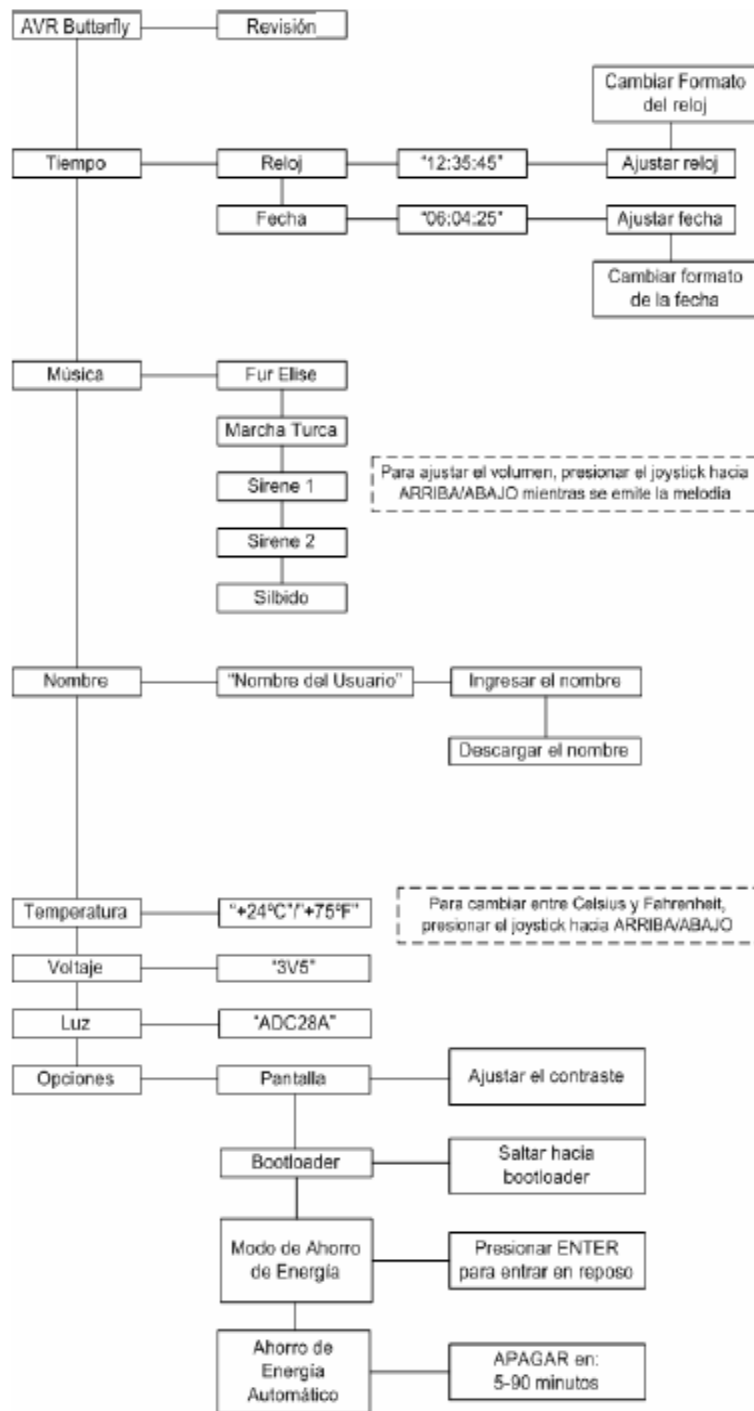


Figura 2.8. Menú Incluido en el AVR Butterfly (Ref.[4])



## JOYSTICK

Para operar el AVR Butterfly se emplea el joystick como una entrada para el usuario. Este opera en cinco direcciones, incluyendo presión en el centro; tal como se puede ver en la Figura 2.9.

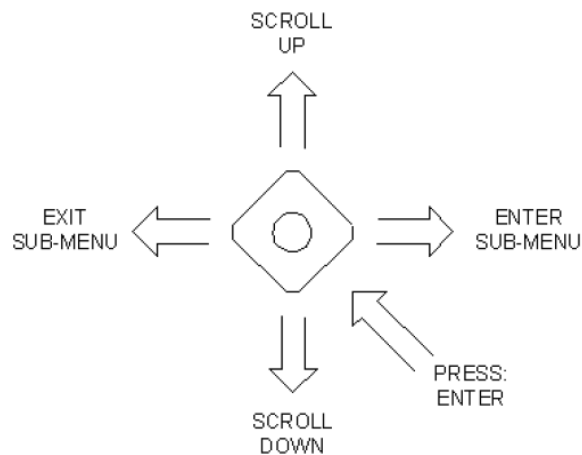


Figura 2.9. Entrada tipo Joystick (Ref.[4])

### 2.2.1.4 MICROCONTROLADORES DE ATMEL

La corporación ATMEL, fundada en 1984 es líder mundial en el diseño, fabricación y comercialización de semiconductores avanzados, con enfoque en microcontroladores; que incluyen lógica avanzada, memoria no volátil, circuitos integrados mezcladores de señal, componentes para radio frecuencia y sensores. Estas funciones se comercializan como productos estándar, productos estándar de aplicación específica (ASSP) o productos

para clientes específicos (ASIC), proporcionando una respuesta rápida y flexible a las necesidades de los clientes de Atmel.

Los chips Atmel se fabrican utilizando los más avanzados procesos de integración, que incluyen tecnologías BiCMOS, CMOS y Germanio de Silicio (SiGe).

Como se observa en la Tabla 2.1, los Socios/Clientes Estratégicos de Atmel son mundialmente los más importantes fabricantes de dispositivos de telecomunicaciones, computación, aplicaciones militares y sistemas de seguridad, etc.

Atmel AVR 8 bits				
Device	Flash (Kbytes)	EEPROM (Kbytes)	SRAM (bytes)	F.max (MHz)
ATmega128	128	4	4096	16
ATmega64	64	2	4096	16
ATmega32	32	1	2048	16
ATmega16	16	0,5	1024	16
ATmega162	16	0,5	1024	16
ATmega169	16	0,5	1024	16
ATmega8	8	0,5	1024	16
ATmega8515	8	0,5	512	16
ATmega8535	8	0,5	512	16

Tabla 2.1. Familia de Microcontroladores de Atmel (Ref.[5])

#### **2.2.1.4.1 CARACTERÍSTICAS DEL MICROCONTROLADOR ATMEGA169**

El ATmega169 es un Microcontrolador de 8 bits con arquitectura AVR RISC, este posee las siguientes características:

- Arquitectura RISC avanzada.
  - Conjunto de 130 instrucciones ejecutables en un solo ciclo de reloj.
  - 32 x 8 registros de trabajo de propósito general.
  - Rendimiento de hasta 16 MIPS a 16 MHz.
- Memoria no volátil para Programa y Datos.
  - Flash de 16 K bytes, auto-programable en el sistema.
  - Resistencia: 10 000 ciclos de Escritura/Borrado.
  - Sección Opcional para Código de Arranque con Lock Bits Independientes.
- Programación en el Sistema a través del Programa de Arranque residente en el chip.
  - Operación Real de Lectura Durante la Escritura.
  - EEPROM de 512 bytes.
  - Resistencia: 100 000 ciclos de Escritura/Borrado.
  - SRAM Interna de 1 Kbyte.
  - Bloqueo de Programación para Seguridad del Software.
- Interfaz JTAG (conforme el Standard IEEE 1149.1)

- Capacidad de Boundary-Scan Acorde al Standard JTAG.
  - Soporta Depuración On-chip.
  - Programación de la FLASH, EEPROM, Fusibles y Lock Bits a través de la Interfaz JTAG.
- Características de los Periféricos.
    - 6 puertos de I/O de 8-bits y 1 de 5-bits.
    - Controlador de LCD de 4 x 25 segmentos.
    - Dos Temporizadores/Contadores de 8 bits con Preajustador (Prescaler) Separado y Modo de Comparación.
    - Un Temporizador/Contador de 16 bits con Preajustador (Prescaler) Separado, Modo de Comparación y Modo de Captura.
    - Contador de Tiempo Real con Oscilador Separado.
    - Cuatro canales PWM.
    - Ocho canales ADC de 8 bits cada uno.
    - Interfaz Serial USART Programable.
    - Interfaz Serial SPI Maestro/Esclavo.
    - Interfaz Serial Universal con Detector de Condición de Inicio.

- WatchDog Timer Programable con Oscilador Separado incluido en el chip.
- Comparador Analógico.
- Interrupción y Salida del Modo de Sleep por Cambio en Pin.
- Características especiales del microcontrolador.
  - Reset al Encendido y Detección Brown-Out Programable.
  - Oscilador Interno Calibrable.
  - Fuentes de Interrupción Internas y Externas.
  - Cinco Modos de Sleep: Idle, ADC Noise Reduction, Power Save, Power-Down y Standby.
- Entradas/Salidas y Tipo de Encapsulado
  - 53 Líneas de I/O Programables.
  - 64 patillas en el encapsulado TQFP y 64 pads (para montaje superficial) en el encapsulado QFN/MLF.
- Rangos de Velocidad
  - ATmega169V: 0 – 4 MHz a 1.8 – 5.5 V, 0 – 8 MHz a 2.7 – 5.5 V.
  - ATmega169: 0 – 8 MHz a 2.7 – 5.5 V, 0 – 16 MHz a 4.5 – 5.5 V.

- Rango de Temperatura

- Desde  $-40^{\circ}\text{C}$  a  $85^{\circ}\text{C}$ .

- Consumo de Energía

- En el Modo Activo: 1 MHz, 1.8 V: 350  $\mu\text{A}$ .

- 32 KHz, 1.8 V: 20  $\mu\text{A}$  (incluyendo Oscilador).

- 32 KHz, 1.8 V: 40  $\mu\text{A}$  (incluyendo Oscilador y LCD).

- En el Modo Power-Down:

- 0.1  $\mu\text{A}$  a 1.8 V.

El AVR ATmega169 es compatible con un completo conjunto de programas y Herramientas de Desarrollo que incluye: Compiladores C, Ensambladores de Macro, Depurador/Simuladores de Programa, Emuladores de Circuito, Kits de Iniciación y Kits Evaluación.

## 2.2.1.4 DIAGRAMA DE BLOQUES DEL MICROCONTROLADOR AVR

### ATMEGA169

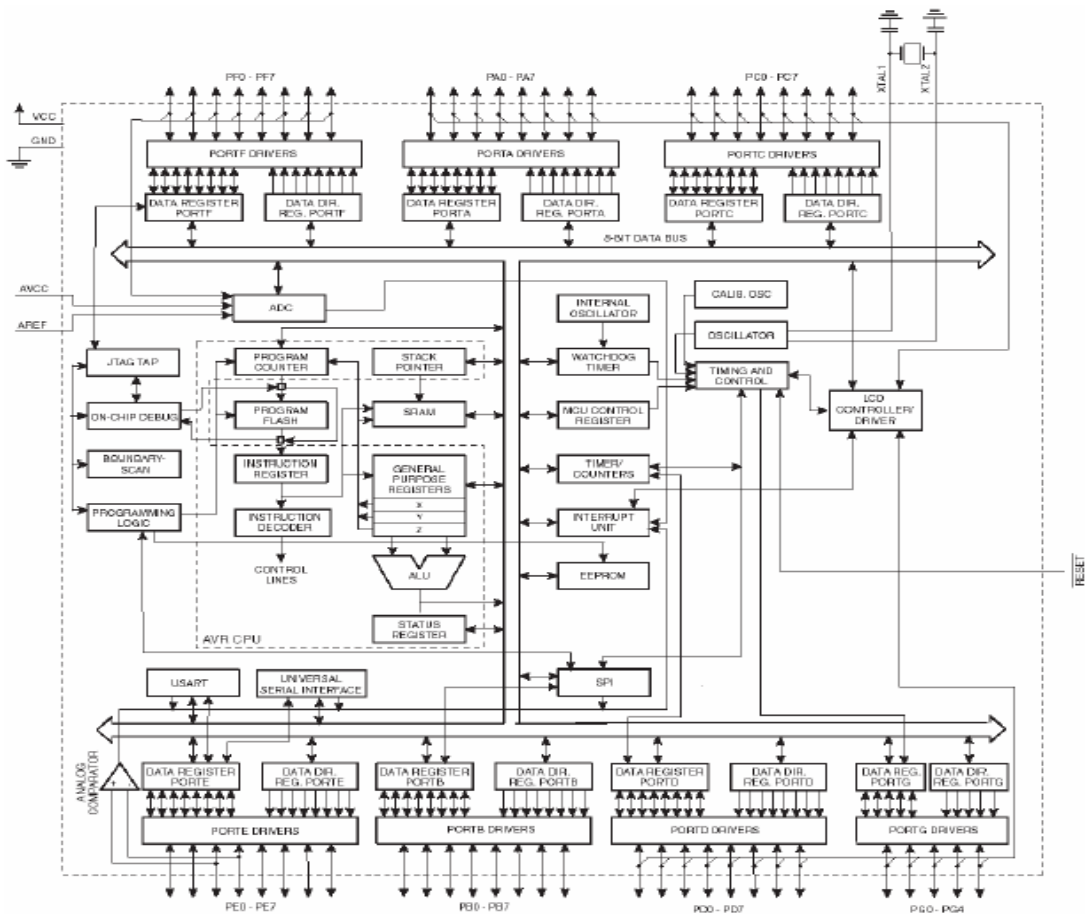


Figura 2.10. Diagrama de Bloques del Microcontrolador ATmega169 (Ref.[5])

El núcleo AVR combina un rico conjunto de instrucciones con 32 registros de Trabajo de Propósito General (General Purpose Working Registers). Todos los 32 registros están conectados directamente a la Unidad de Lógica Aritmética (ALU), permitiendo que se acceda a dos registros independientes en una sola instrucción ejecutada en un ciclo de reloj. La arquitectura da

como resultado código más eficiente mientras que el rendimiento alcanzado es diez veces más rápido que los convencionales microcontroladores CISC.

### 2.2.1.4.3 CONFIGURACIÓN DE PINES DEL MICROCONTROLADOR AVR ATMEGA169

En la Figura 2.11 se ilustra la distribución de pines del microcontrolador ATmega169V, en éste se aprecia el encapsulado MLF de 64 pines.

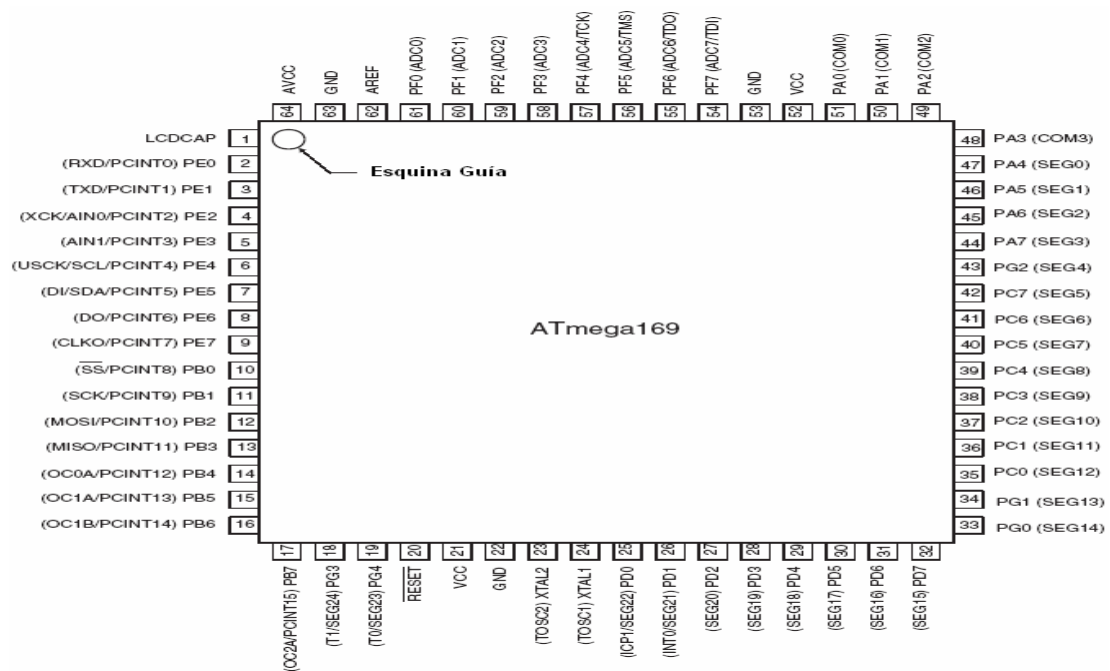


Figura 2.11. Distribución de Pines del ATmega169 (Ref.[6])

**VCC.** Voltaje de suministro digital.

**GND.** Tierra.



**Puerto A (PA7...PA0) y Puerto B (PB7...PB0).** El Puerto A y el puerto B son un puerto bi-direccional de I/O de 8-bits con resistencias pull-up internas (seleccionables para cada bit). Los buffers de salida del Puerto A y B tienen características de controlador simétricas, con capacidad alta tanto de fuente (source) como de sumidero (drain). Como entradas, si las resistencias pull-up están activadas los pines del Puerto A y B que estén externamente enganchados a un nivel bajo serán fuentes de corriente.

**Puerto C (PC7...PC0), Puerto D (PD7...PD0), Puerto E (PE7...PE0) y Puerto G (PG0...PG0).** Son Puertos bi-direccional de I/O de 8-bits con resistencias pull-up internas (seleccionables para cada bit). Las características de este puerto son las mismas que las descritas anteriormente para el puerto A.

**Puerto F (PF7...PF0).** El Puerto F sirve como entradas analógicas para el conversor A/D (Analógico a Digital). El Puerto F también sirve como un puerto bi-direccional de Entrada/Salida de 8 bits, si el conversor A/D no se está utilizando. Los buffers de salida del Puerto F tienen características de controlador simétricas, con capacidad alta tanto de fuente (source) como de sumidero (drain). Como entradas, si las resistencias pull-up están activadas los pines del Puerto F que estén externamente enganchados a un nivel bajo serán fuentes de corriente. Los pines del Puerto F están en alta impedancia cuando una condición de reset se activa, incluso si el reloj no está corriendo.

Si la interfaz JTAG es habilitada, las resistencias pull-up de los pines PF7 (TDI), PF5 (TMS) y PF4 (TCK) se activarán aún si ocurre un reset.

**RESET** . Entrada de reset.

**XTAL1**. Entrada al amplificador del oscilador y entrada al circuito de funcionamiento del reloj interno.

**XTAL2**. Salida del amplificador del oscilador.

**AVCC**. El Pin AVCC sirve de fuente de voltaje para el Puerto F y para el conversor A/D. Este debe ser conectado externamente a  $V_{CC}$ , aún si el ADC no se está usando. Si se está usando el ADC, el AVCC debe conectarse a  $V_{CC}$  a través de un filtro pasa bajo.

**AREF**. Éste es el Pin de Referencia Analógica para el conversor A/D.

**LCDCAP**. Un condensador externo (típicamente mayor a 470nF) debe conectarse al Pin LCDCAP. Este condensador actúa como un reservorio de energía para el LCD ( $V_{LCD}$ ). Una capacitancia grande reduce el rizo (ondulaciones) en el  $V_{LCD}$  pero incrementa el tiempo hasta el cual el VLCD alcanza su valor deseado.

## 2.2.2 COMUNICACIÓN RS232

RS232 (Recommended Standard 232, también conocido como Electronic Industries Alliance RS-232C) es una interfaz que designa una norma para el intercambio de una serie de datos binarios entre un DTE (Equipo terminal de datos) y un DCE (Equipo de Comunicación de datos), aunque existen otras en las que también se utiliza la interfaz RS-232.

En particular, existen ocasiones en que interesa conectar otro tipo de equipamientos, como pueden ser computadores. Evidentemente, en el caso de interconexión entre los mismos, se requerirá la conexión de un DTE (Data Terminal Equipment) con otro DTE. Para ello se utiliza una conexión entre los dos DTE sin usar módem, por ello se llama: null módem ó módem nulo.

El RS-232 consiste en un conector tipo DB-25 (de 25 pines), aunque es normal encontrar la versión de 9 pines (DE-9), más barato e incluso más extendido para cierto tipo de periféricos (como el ratón serie del PC).



Figura 2.12. Conector RS-232, DE-9 hembra (Ref.[7])

### 2.2.2.1 UART

El Transmisor y Receptor Serial Universal Asíncrono (Universal Asynchronous Serial Receiver and Transmitter UART) es un dispositivo de comunicación serial altamente flexible. Sus características principales se listan a continuación.

- Operación Full Duplex (Registros Independientes para Recepción y Transmisión Serial).
- Funcionamiento Asíncrono y Síncrono.
- Funcionamiento Master o Slave.
- Generador de Tasa de Baudio de Alta Resolución.
- Soporta Tramas Seriales con 5, 6, 7, 8 y 9 Bits de Datos y 1 ó 2 Bits de Parada.
- Generación de Paridad Par e Impar y Comprobación de Paridad Soportada por Hardware.
- Detección de Desbordamiento de Datos.
- Detección de Error de Trama.
- Filtraje de Ruido que Incluye Detección de Falso Bit de Inicio y Filtro Digital Pasabajo.
- Tres Interrupciones Separadas por los Eventos “TX Finalizada”, “Registro de Datos de TX Vacío” y “RX Finalizada”.
- Modo de Comunicación de Multi-procesador.

- Modo de Comunicación Asíncrono de Doble Velocidad.

Para habilitar el módulo UART el bit PRUSART0 en el “Power Reduction Register - PRR” debe escribirse con un cero.

En la Figura 2.13 se muestra un diagrama de bloque simplificado del Transmisor UART. Los Registros de I/O accesibles por la CPU, incluyendo los bits de I/O y los pines de I/O, se muestran en negrita.

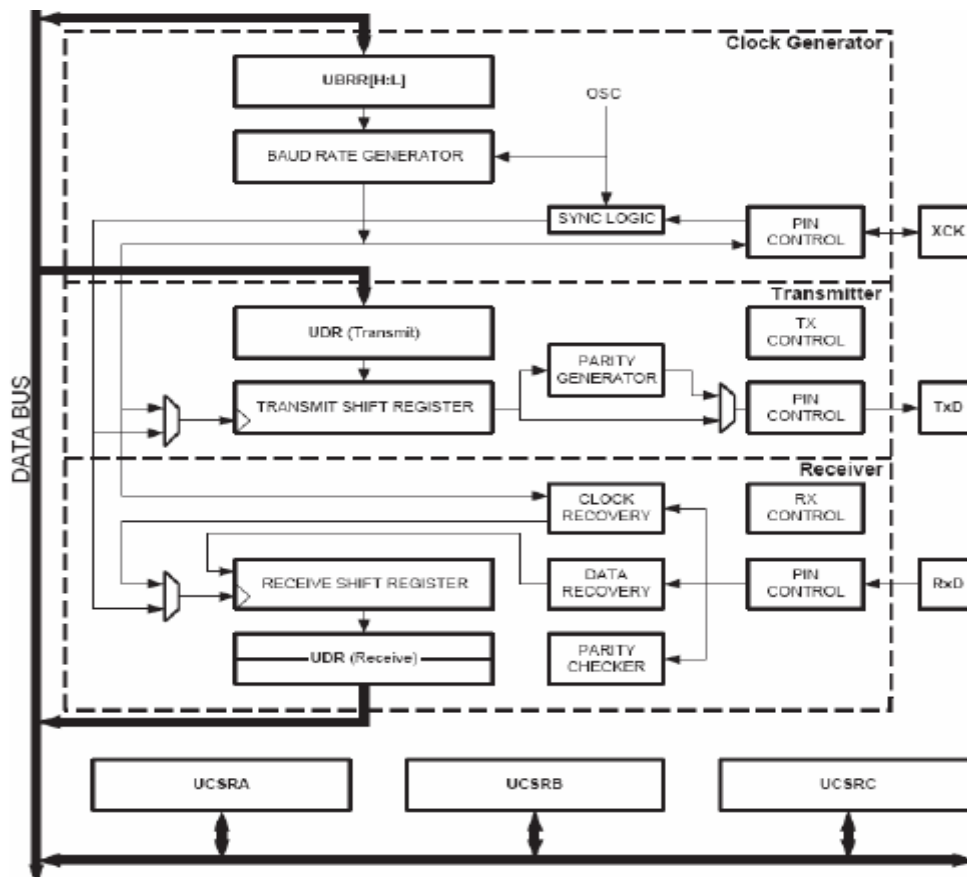


Figura 2.13. Diagrama de Bloque de la UART (Ref.[8])

Las zonas con líneas entre cortadas en el diagrama de bloque separa las tres partes principales de UART (listadas desde arriba): Generador de Reloj, Transmisor y Receptor. Los Registros de Control son compartidos por todas las unidades. La lógica de la Generación de Reloj consiste de la lógica de sincronización a causa de la entrada de reloj externa usada por la operación slave sincrónica, y el generador de tasa de baudio.

El pin XCK (Transfer Clock – Reloj para Transferencia) es únicamente usado por el modo de transferencia sincrónico. El transmisor consiste de un buffer de escritura sencillo, un Registro de Desplazamiento serial, Generador de Paridad y lógica de Control para manejar diferentes formatos de trama serial. El buffer de escritura permite la transferencia continua de datos sin ningún retraso entre tramas.

El Receptor es la parte más compleja del módulo USART debido a su reloj y unidades de recuperación de datos. Las unidades de recuperación se usan para recepción asincrónica de datos. Además de las unidades de recuperación, el Receptor incluye un Comprobador de Paridad, lógica de Control, un Registro de Desplazamiento y un buffer de recepción de dos niveles (UDR). El Receptor soporta los mismos formatos de trama que el Transmisor, y puede detectar Error de Trama, Desbordamiento de Datos y Errores de Paridad.

## Generación de Reloj

La lógica para la Generación de Reloj genera el reloj base para el Transmisor y Receptor. USART soporta cuatro modos de operación del reloj: modo asincrónico Normal, asincrónico de Velocidad Doble, síncrono Master y síncrono Slave. El bit UMSEL en el Registro C para el Control y Estado de USART (USART Control and Status Register C – UCSRC) selecciona entre la operación asincrónica y la síncrona.

La velocidad doble (únicamente modo asincrónico) se controla por medio del bit U2X que se encuentra en el Registro UCSRA. Al usar modo síncrono (UMSEL=1), el Registro de Dirección de Datos para el pin XCK (DDR\_XCK) controla si la fuente de reloj es interna (modo Master) o externa (modo Slave). El pin XCK está activo únicamente al usar el modo síncrono. En la Figura 2.14 se muestra el diagrama de bloques de la Lógica de Generación de Reloj.

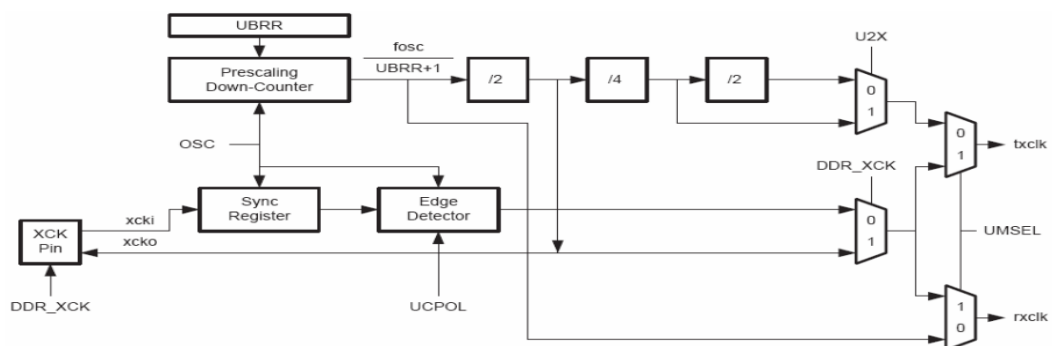


Figura 2.14. Diagrama de Bloque de la Lógica de Generación de Reloj (Ref.[8])

Descripción de las señales:

**txclk** Reloj del Transmisor (Señal Interna).

**rxclk** Reloj base del Receptor (Señal Interna).

**xcki** Entrada desde el pin XCK (Señal interna). Usada en el funcionamiento

sincrónico slave.

**xcko** Salida de Reloj para el pin XCK (Señal Interna). Usada en el

funcionamiento sincrónico Master.

**fosc** frecuencia del pin XTAL (Reloj del Sistema).

#### **2.2.2.1.1 INICIALIZACIÓN DE LA INTERFAZ UART**

El módulo USART tiene que ser inicializado antes de que cualquier comunicación pueda tener lugar. El proceso de inicialización normalmente consiste en configurar la baud rate, configurar el formato de la trama y dependiendo del uso habilitar el Transmisor o el Receptor. Cuando se hace la inicialización, el Indicador Global de Interrupción debe ser borrado (I = 0, todas las interrupciones deshabilitadas globalmente).



Antes de hacer una re-inicialización con cambios en la baud rate o en el formato de trama, hay que asegurarse que ninguna transmisión esté en proceso durante el período en que se haga cambios en los registros. El Indicador TXC puede usarse para chequear que el Transmisor ha finalizado todas las transferencias. El Indicador RXC puede usarse para chequear que no haya ningún dato sin leer en el buffer de recepción. Nótese que el Indicador TXC debe ser borrado antes de cada transmisión (antes de que el UDR sea escrito).

### **Transmisión de Datos – El Transmisor de UART**

El Transmisor USART es habilitado por el bit Habilitador de Transmisión (Transmit Enable – TXEN) en el registro UCSRB. Cuando el transmisor está habilitado, el USART anula el funcionamiento normal de puerto del pin TxD y le da la función de salida serial del Transmisor. La baud rate, el modo de funcionamiento y el formato de la trama deben fijarse una sola vez antes de que se haga cualquier transmisión. Si se usa funcionamiento sincrónico, el reloj en el pin XCK será anulado y usado como reloj de la transmisión.

### **Transmisión de Tramas de 5 a 8 Bits de Datos.**

Una transmisión de datos comienza cargando el buffer de transmisión con el dato que será transmitido. La CPU puede cargar el buffer de transmisión escribiendo en la localidad de I/O del UDR. El dato colocado en el buffer de

transmisión será movido hacia el Registro de Desplazamiento (Shift Register) cuando el Registro de Desplazamiento esté listo para enviar una nueva trama. El Registro de Desplazamiento se carga con datos nuevos si USART está en el estado inactivo (sin transmisión en proceso) ó inmediatamente luego de que se haya transmitido el último bit de parada de la trama previa. Cuando el Registro de Desplazamiento está cargado con datos nuevos, éste transferirá una trama completa a la velocidad de transmisión dada por el Registro de la Baud Rate (Baud Register), bit U2X o por XCK dependiendo del modo de funcionamiento.

### **El Indicadores e Interrupciones del Transmisor.**

El Transmisor USART tiene dos indicadores que revelan su estado: Registro de Datos Vacante (USART Data Register Empty – UDRE) y Transmisión Finalizada (Transmit Complete – TXC). Ambos indicadores pueden usarse para generar interrupciones.

El Indicador UDRE indica si el buffer de transmisión está listo para recibir datos nuevos. Este bit es puesto a uno cuando el buffer de transmisión está vacante, y a cero cuando el buffer de transmisión aún contiene datos para transmitir que aún no se han movido hacia dentro del Registro de Desplazamiento. Por compatibilidad con futuros dispositivos, se recomienda escribir siempre este bit a cero al escribir en el Registro UCSRA.

Cuando el bit Habilitador de la Interrupción por Registro de Datos Vacante (Data Register Empty Interrupt Enable – UDRIE) en el UCSRB está escrito a uno, la Interrupción por Registro de Datos Vacante de USART se ejecutará con tal que UDRE esté seteado (con tal de que las interrupciones estén habilitadas globalmente). El UDRE es puesto a cero al escribir en el UDR. Cuando se usa la transmisión de datos manejada por interrupción, la rutina de la interrupción por Registro de Datos Vacante debe escribir el dato nuevo en el UDR para poner en cero al UDRE ó deshabilitar la interrupción por Registro de Datos Vacante, de lo contrario una interrupción nueva ocurrirá una vez terminada la rutina de interrupción.

El bit Indicador de Transmisión Finalizada (TXC) es puesto a uno cuando la trama entera en el Registro de Desplazamiento para Transmisión (Transmit Shift Register) se ha desplazado hacia fuera y actualmente no hay nuevos datos presentes en el buffer de transmisión. El bit Indicador TXC es automáticamente borrado cuando se ejecuta una interrupción por transmisión finalizada. El TXC es útil en la interfaz de comunicación half-duplex (como el Standard RS-485), donde una aplicación de transmisión debe entrar al modo de recepción y liberar el bus de comunicación inmediatamente luego de finalizar la transmisión.

## **Recepción de Datos – El Receptor de UART**

El Receptor UART es habilitado por el bit Habilitador de Recepción (RXEN) en el registro UCSRB. Cuando el Receptor está habilitado, el USART anula el funcionamiento normal del pin RxD y le da la función de entrada serial del Receptor. La baud rate, el modo de funcionamiento y el formato de la trama deben ser fijados una sola vez antes de que se pueda hacer cualquier recepción. Si se usa funcionamiento sincrónico, el reloj en el pin XCK será anulado y usado como reloj de transferencia.

### **Recepción de Tramas de 5 a 8 Bits de Datos.**

El Receptor inicia la recepción de datos cuando detecta un bit de inicio válido. Cada bit que sigue al bit de inicio será muestreado a la velocidad de transmisión en baudios (baud rate) o con el reloj XCK, y desplazado hacia dentro del Registro de Desplazamiento para Recepción hasta que se reciba el primer bit de parada de una trama. Un segundo bit de parada será ignorado por el receptor. Al recibir el primer bit de parada se moverán los contenidos del Registro de Desplazamiento hacia el buffer de recepción. El buffer de recepción puede leerse entonces leyendo la localidad de I/O del UDR.

### **Indicador e Interrupción del evento Recepción Finalizada.**

El Indicador de Recepción Finalizada (RXC) indica que hay datos sin leer presentes en el buffer de recepción. Este indicador está a uno cuando existen datos sin leer en el buffer, y a cero cuando el buffer de recepción está vacío (no contiene datos sin leer). Si está deshabilitado el Receptor (RXEN=0), el buffer de recepción será vaciado y consecuentemente el bit RXC se volverá a cero.

Cuando el Habilitador de Interrupción por Recepción Finalizada (Receive Complete Interrupt Enable – RXCIE) en UCSRB está seteado, la Interrupción por Recepción USART Finalizada se ejecutará con tal de que el Indicador RXC esté seteado (con tal de que las interrupciones estén habilitadas globalmente). Cuando se usa la recepción de datos manejada por interrupción, la rutina de Recepción Finalizada debe leer el dato recibido desde el UDR para limpiar el Indicador RXC, de otro modo ocurrirá una nueva interrupción una vez terminada la rutina de interrupción.

#### **2.2.2.1.2 DESCRIPCIÓN DE LOS REGISTROS UART**

## Los Registros para Datos de I/O de UART – UDR

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDR (Read)
	TXB[7:0]								UDR (Write)
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Tabla 2.2. Registro para Datos de I/O (Ref.[9])

El Registro del Buffer de Datos de la Transmisión UART y el Registro del Buffer de Datos de la Recepción UART comparten la misma dirección de I/O llamada Registro de Datos UART o UDR. El Registro del Buffer de Datos de la Transmisión (TXB) será el destino para los datos escritos a la localidad del Registro UDR. Leer la localidad del Registro UDR retornará los contenidos del Registro del Buffer de Datos de Recepción (RXB).

Para caracteres de 5, 6 o 7 bits, los bits superiores no usados serán ignorados por el Transmisor y puestos a cero por el Receptor.

El buffer de transmisión únicamente puede ser escrito cuando el Indicador UDRE en el Registro UCSRA está puesto a uno (UDRE=1). Los datos escritos al UDR cuando el Indicador UDRE no está puesto en uno, serán ignorados por el Transmisor UART. Cuando se escriben datos al buffer de transmisión, y el Transmisor está habilitado, el Transmisor cargará los datos

dentro del Registro de Desplazamiento para Transmisión (Transmit Shift Register) cuando el Registro de Desplazamiento esté vacío. Entonces los datos serán transmitidos serialmente por el pin TxD.

El buffer de recepción consiste de un FIFO de dos niveles. El FIFO cambia su estado cuando el buffer de recepción es accedido. Debido a éste comportamiento del buffer de recepción, no se debe usar las instrucciones para Leer-Modificar-Escribir (SBI y CBI) en ésta localidad. Tenga cuidado al usar las instrucciones de prueba de bit (SBIC y SBIS), puesto que estas también cambiarán el estado del FIFO.

### El Registro A para el Control y Estado del UART – UCSRA

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDR (Read)
	TXB[7:0]								UDR (Write)
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Tabla 2.3. Registro A del Uart (Ref.[9])

#### Bit 7 – RXC: Recepción Finalizada.

Éste bit es “1” cuando hay datos sin leer en el buffer de recepción y es “0” cuando el buffer de recepción está vacío. Si el Receptor está deshabilitado, el buffer de recepción será vaciado y consecuentemente el bit RXC se

volverá cero. El Indicador RXC puede usarse para generar una interrupción por Recepción Finalizada.

#### **Bit 6 – TXC: Transmisión Finalizada.**

Este bit indicador es puesto a “1” cuando la trama entera, en el Registro de Desplazamiento para Transmisión, se ha desplazado hacia fuera y no hay nuevos datos actualmente presentes en el buffer de transmisión (UDR). El bit Indicador TXC es automáticamente puesto a “0” cuando se ejecuta una interrupción por transmisión finalizada. El Indicador TXC puede generar una interrupción por Transmisión Finalizada.

#### **Bit 5 – UDRE: Registro de Datos Vacante.**

El Indicador UDRE indica si el buffer de transmisión (UDR) está listo para recibir nuevos datos. Si el UDRE es “1”, el buffer está vacío, y por lo tanto, listo para ser escrito. El Indicador UDRE puede generar una interrupción por Registro de Datos Vacante.

El UDRE es “1” luego de un reset para indicar que el transmisor está listo.

#### **Bit 4 – FE: Error en la Trama.**

Este bit es “1” si el siguiente carácter en el buffer de recepción tiene un Error en la Trama recibida. Por ejemplo, cuando el primer bit de parada del siguiente carácter en el buffer de recepción es cero. Éste bit es válido hasta



que el buffer de recepción (UDR) se lee. EL bit FE es “0” cuando el bit de parada del dato recibido es uno. Siempre ponga éste bit a cero al escribir en el UCSRA.

### **Bit 3 – DOR: Desbordamiento de Datos.**

Este bit es puesto a “1” si se detecta condiciones de Desbordamiento de Datos. Un Desbordamiento de Datos ocurre cuando el buffer de recepción está lleno (dos caracteres), está un nuevo carácter esperando en el Registro de Desplazamiento para Recepción y un nuevo bit de inicio es detectado. Éste bit es válido hasta que el buffer de recepción (UDR) es leído. Siempre ponga este bit a cero al escribir en el UCSRA.

### **Bit 2 – UPE: Error de Paridad.**

Éste bit es puesto a “1” si el siguiente carácter en el buffer de recepción tiene un Error de Paridad cuando se lo recibió y la Verificación de Paridad fue habilitada en aquel punto (UPM1=1). Éste bit es válido hasta que el buffer de recepción (UDR) es leído. Siempre poner este bit a cero al escribir en el UCSRA.

### **Bit 1 – U2X: Doblar la Velocidad de Transmisión.**

Este bit únicamente tiene efecto en el funcionamiento asíncrono. Escriba este bit a cero al usar funcionamiento sincrónico.

Escribiendo éste bit a uno reducirá el divisor de la baud rate, de 16 a 8, doblando efectivamente la velocidad de transferencia para comunicación asincrónica.

### **Bit 0 – MPCM: Modo de Comunicación de Multi-procesador.**

Este bit habilita el modo de Comunicación Multi-procesador. Cuando el bit MPCM está escrito a uno se ignoran todas las tramas entrantes, recibidas por el Receptor USART, que no contienen información de dirección. El Transmisor no es afectado por la configuración MPCM.

### **El Registro B para Control y Estado del UART – UCSRB**

Bit	7	6	5	4	3	2	1	0	
	<b>RXCIE</b>	<b>TXCIE</b>	<b>UDRIE</b>	<b>RXEN</b>	<b>TXEN</b>	<b>UCSZ2</b>	<b>RXB8</b>	<b>TXB8</b>	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Tabla 2.4. Registro B del Uart (Ref.[9])

### **Bit 7 – RXCIE: Habilitar la Interrupción por RX Finalizada.**

Escribir éste bit a uno habilita la interrupción por el Indicador RXC. Una interrupción por Recepción UART

Finalizada se generará únicamente si el bit RXCIE es escrito a uno (RXCIE=1), el Indicador de la Interrupción Global en SREG está escrito a uno y el bit RXC en UCSRA se pone a uno.

**Bit 6 – TXCIE: Habilitar la Interrupción por TX Finalizada.**

Escribir este bit a uno habilita la interrupción por el Indicador TXC. Una interrupción por Transmisión UART Finalizada se generará únicamente si el bit TXCIE es escrito a uno, el Indicador de la Interrupción Global en SREG está escrito a uno y el bit TXC en UCSRA se pone a uno.

**Bit 5 – UDRIE: Habilitar la Interrupción por Registro de Datos Vacante.**

Escribir este bit a uno habilita la interrupción por el Indicador UDRE. Una interrupción por Registro de Datos Vacante se generará únicamente si el bit UDRIE es escrito a uno, el Indicador de la Interrupción Global en SREG está escrito a uno y el bit UDRE en el UCSRA se pone a uno.

**Bit 4 – RXEN: Habilitador del Receptor.**

Escribir este bit a uno habilita el Receptor USART. Cuando está habilitado, el Receptor anula el funcionamiento normal del puerto para el pin RxD. Al Deshabilitar el Receptor se vaciará el buffer de recepción, invalidando los Indicadores FE, DOR y UPE.

**Bit 3 – TXEN: Habilitador del Transmisor.**

Escribir este bit a uno habilita el Transmisor USART. Al habilitarlo, el Transmisor anulará el funcionamiento normal del puerto para el pin TxD. La desactivación del Transmisor (escribiendo TXEN=0) no se hará efectiva hasta que las transmisiones en proceso y pendientes se completen. Al deshabilitarlo, el Transmisor ya no anulará el puerto TxD.

**Bit 2 – UCSZ2: Tamaño de Carácter.** Los bits UCSZ2 combinados con el bit UCSZ1:0 del UCSRC fijan el número de bits de datos (Tamaño de Carácter) que el Receptor y el Transmisor usarán en una trama.

**Bit 1 – RXB8: Bit 8 de los Datos Recibidos.** RXB8 es el noveno bit de datos del carácter recibido al operar con tramas seriales de nueve bits de datos. Debe leerse antes de leer los bits bajos del UDR.

**Bit 0 – TXB8: Bit 8 de los Datos Transmitidos.** TXB8 es el noveno bit de datos del carácter a transmitirse al operar con tramas seriales de nueve bits de datos. Debe escribirse antes de escribir los bits bajos al UDR.

### El Registro C para el Control y Estado de UART – UCSRC

Bit	7	6	5	4	3	2	1	0	
	-	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	1	1	0	

Tabla 2.5. Registro C del Uart (Ref.[9])

**Bit 6 – UMSEL: Selección del Modo de UART.** Éste bit selecciona entre los modos de funcionamiento asincrónico y sincrónico.

UMSEL	Mode
0	Asynchronous Operation
1	Synchronous Operation

Tabla 2.6. Configuración del Bit UMSEL (Ref.[9])

**Bit 5:4 UPM1:0: Modo de Paridad.** Estos bits habilitan y fijan el tipo de generación y verificación de paridad. Ver Tabla 2.7. Si está habilitado, el transmisor generará automáticamente y enviará, dentro de cada trama, la paridad de los bits de datos transmitidos. El Receptor generará un valor de paridad para el dato entrante y lo comparará a la configuración del UPM0. Si se detectada una desigualdad, el Indicador UPE en el UCSRA se pondrá a uno.

UPM1	UPM0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

Tabla 2.7. Configuración de los Bits UPM (Ref.[9])

**Bit 3 – USBS: Selección del Bit de Parada.** Este bit selecciona el número de bits de parada a ser insertados por el Transmisor. Ver Tabla 2.8. El Receptor ignora esta configuración.

USBS	Stop Bit(s)
0	1-bit
1	2-bit

Tabla 2.8. Configuración del Bit USBS (Ref.[9])

**Bit 2:1 – UCSZ1:0: Tamaño del Carácter.** Los bits UCSZ1:0 combinados con el bit UCSZ2 del UCSRB fijarán el número de bits de datos (Tamaño del Carácter) que el Receptor y el Transmisor usarán en una trama. Ver Tabla 2.9.

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

Tabla 2.9. Configuración de los Bits UCSZ (Ref.[9])

**Bit 0 – UCPOL: Polaridad del Reloj.** Este bit se usa únicamente para el modo síncrono. Este bit debe ser escrito a cero cuando se usa el modo asíncrono. El bit UCPOL fija la relación entre el cambio de la salida de

datos, el muestreo de la entrada de datos y el reloj sincrónico (XCK). Ver Tabla 2.10.

UCPOL	Transmitted Data Changed (Output of TxD Pin)	Received Data Sampled (Input on RxD Pin)
0	Rising XCK Edge	Falling XCK Edge
1	Falling XCK Edge	Rising XCK Edge

Tabla 2.10. Configuración del Bit UCPOL (Ref.[9])

### Los Registros de la Baud Rate UART – UBRRL y UBRRH

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	UBRR[11:8]				UBRRH
	UBRR[7:0]								UBRRL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Tabla 2.11. Registro Baud Rate del Uart (Ref.[9])

**Bit 15:12 – Bits Reservados.** Estos bits son reservados para uso futuro. Por compatibilidad con dispositivos futuros, estos bits deben escribirse a cero cuando el UBRRH es escrito.

**Bit 11:0 – UBRR11:0: Registro de la Baud Rate de UART.** Este es un registro de 12 bits que contiene la baud rate de UART. El UBRRH contiene los cuatro bits más significativos, y el UBRRL contienen los ocho bits menos significativos de la baud rate UART. Las transmisiones en proceso se alterarán si la baud rate es cambiada. Escribir en el UBRRL activará una actualización inmediata del preajustador de la baud rate.

### **Configuración de la Baud Rate**

Para frecuencias estándar de cristal y resonador, las baud rate más comúnmente usadas por el funcionamiento asincrónico pueden ser generadas usando la configuración del UBRR. Las Tasas de Error superiores son aceptables, pero el Receptor tendrá menos resistencia al ruido cuando los índices de error sean altos, especialmente para tramas seriales grandes.

### **2.2.3 TARJETA LPCXpresso**

El LPCXpresso es un toolchain completo para evaluación y desarrollo con microcontroladores de NXP.

Está compuesto por:

- LPCXpresso IDE y "development tools"
- IDE basado en Eclipse



- Compiler y linker GNU
- GDB debugger
- LPCXpresso target board (stick)
- BaseBoard o hardware adicional (opcional)

El target board es un microcontrolador con todo lo necesario para encender y también una herramienta que incluye un programador y debugger.

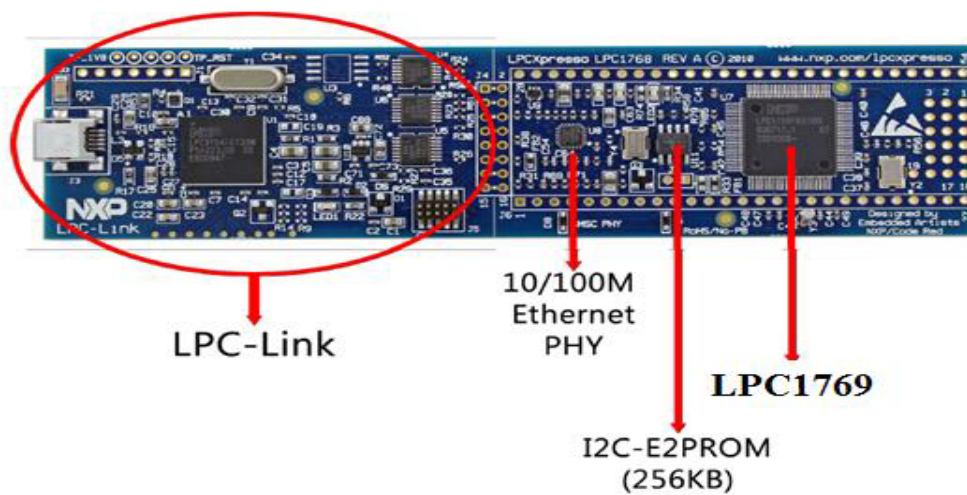


Figura 2.15. Tarjeta LPCXpresso 1769 (Ref.[13]).

## La placa contiene periféricos para desarrollo y experimentación:

### Generales:

- Socket for LPCXpresso and mbed module
- 50 pin expansion dual row pin/header list connector
- Battery powering (small coin battery)
- USB interface
- Reset pushbutton

### Digitales:

- RGB-LED (can be PWM controlled)
- 5-key joystick switch
- 2 pushbuttons, one for activating bootloader
- Rotary switch with quadrature encoding (timer capture)
- Temperature sensor with PWM output (timer capture)

### Analógicos:

- Trimming potentiometer input (analog input)
- PWM to analog LP-filtering (PWM output and analog input)
- Speaker output (PWM output)
- Oscilloscope probe inout stage

Serial - UART:

- USB-to-serial bridge, with automatic ISP activation
- RS422/485 interface
- Interface socket for XBee RF-module

#### **2.2.4 MOTORES DE CORRIENTE CONTINUA SIN ESCOBILLAS (BRUSHLESS)**

Los motores eléctricos sin escobillas se han venido utilizando desde hace años en la industria en general aplicándose en grandes motores servos, aire acondicionado, ventiladores etc., y su ventaja es que al estar libres de mantenimiento pueden durar muchos años. También se han venido utilizando en los aviones y barcos RC. Sin embargo hasta ahora no se ha dispuesto de una tecnología lo suficientemente pequeña y económica como para aplicarla a los automóviles de RC. Esto se debe a que los controles del motor son más exigentes en los automóviles, y a que en los barcos y aviones los frenos no tienen tanta importancia como en los automóviles.

### 2.2.4.1 CARACTERÍSTICAS FUNDAMENTALES



Figura 2.16. Motor BLDC (Ref. [10])

Los motores de corriente continua sin escobillas (BLDC) son uno de los tipos de motores que más popularidad ha ganado en los últimos años.

Su mecanismo se basa en sustituir la conmutación (cambio de polaridad) mecánica por otra electrónica sin contacto. En este caso, la espira sólo es impulsada cuando el polo es el correcto, y cuando no lo es, el sistema electrónico corta el suministro de corriente. Para detectar la posición de la espira del rotor se utiliza la detección de un campo magnético. Este sistema electrónico, además puede informar de la velocidad de giro, o si está parado, e incluso cortar la corriente si se detiene para que no se queme. Tienen la desventaja de que no giran al revés al cambiarles la polaridad (+ y -). Para hacer el cambio se deberían cruzar dos conductores del sistema electrónico.

La palabra brushless se puede traducir como "sin escobillas", las escobillas son los elementos que hacen contacto en el colector de un motor común. En

los motores de DC más pequeños, son de una aleación de cobre y en motores más grandes son de un compuesto a base de carbón.

Los motores BLDC tienen la característica de que no emplean escobillas en la conmutación para la transferencia de energía; en este caso, la conmutación se realiza electrónicamente. Esta propiedad elimina el gran problema que poseen los motores eléctricos convencionales con escobillas, los cuales producen rozamiento, disminuyen el rendimiento, desprenden calor, son ruidosos y requieren una sustitución periódica y por tanto, un mayor mantenimiento.

#### **2.2.4.1.1 Kv, CARACTERÍSTICAS BÁSICA DE UN MOTOR BLDC**

Esta constante Kv significa simplemente la cantidad de vueltas (RPM) que da el motor por cada voltio de continua aplicado al ESC (A máxima potencia). Es decir que si a un motor de 1100 Kv le aplicamos 11,1v funcionará a 12210 RPM como máximo (Con el ESC se puede disminuir). Esta es su velocidad nominal y nunca subirá más velocidad a no ser que aumentemos la diferencia de potencial (Voltaje).

### 2.2.4.2 CONSTRUCCIÓN

Los motores de corriente continua sin escobillas tienen una armadura estacionaria y una estructura rotatoria del campo, exactamente en forma opuesta a como están dispuestos esos elementos en los motores convencionales de CC. Esta construcción aumenta la rapidez de disipación del calor y reduce la inercia del rotor. Imanes permanentes suministran el flujo magnético para el campo. La corriente directa hacia la armadura se conmuta con un sistema electrónico llamado también inversor electrónico, en vez de las escobillas y las delgas.

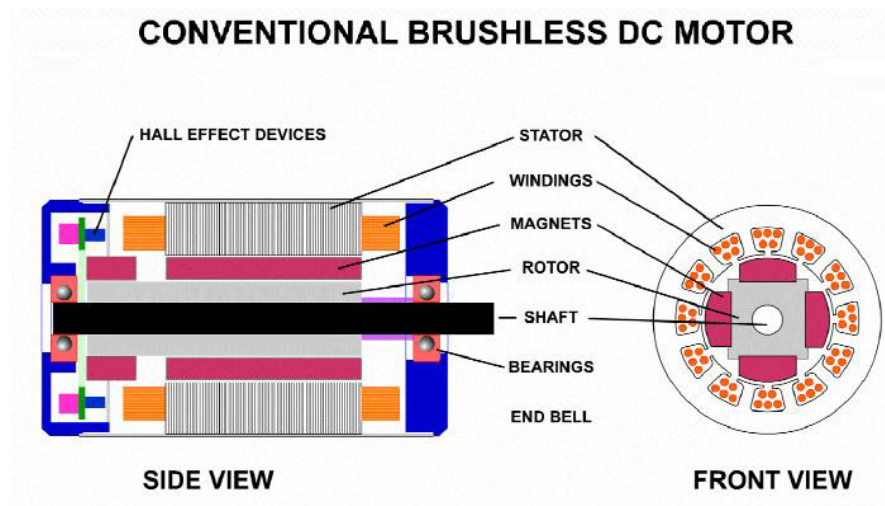


Figura 2.17. Despiece de Motor Brushless DC (Ref. [10])

La ubicación de sensado de los polos magnéticos en los motores DC sin escobillas normalmente se hace con sensores de efecto Hall, aunque existen

modelos que utilizan sensores ópticos, que funcionan de manera similar a los encoders.

#### **2.2.4.3 FUNCIONAMIENTO**

Se puede decir que los motores “sin escobillas” son como los motores “con escobillas” pero del revés. Es decir el rotor, la parte móvil, está compuesto por el eje y los imanes permanentes. En la carcasa o estator es donde se encuentra el bobinado del hilo conductor, que no se mueve. En los motores sin escobillas, la corriente eléctrica pasa por el hilo conductor que está bobinado en la carcasa y produce el campo electromagnético que hace girar a los imanes permanentes y por tanto al eje al que están unidos. Por ello ni las escobillas ni el conmutador son necesarios, ya que la corriente va al estator. Además, en los motores sin escobillas no existen las tres delgas que eran las que obligaban al rotor a moverse cualquiera que fuera su posición. Por ello en los motores sin escobillas es el variador electrónico el que controla en qué posición se encuentra el rotor para darle la corriente temporizada adecuada. Esto se realiza o mediante sensores instalados en el motor o a través de la respuesta que obtiene cuando envía una corriente lineal al motor.

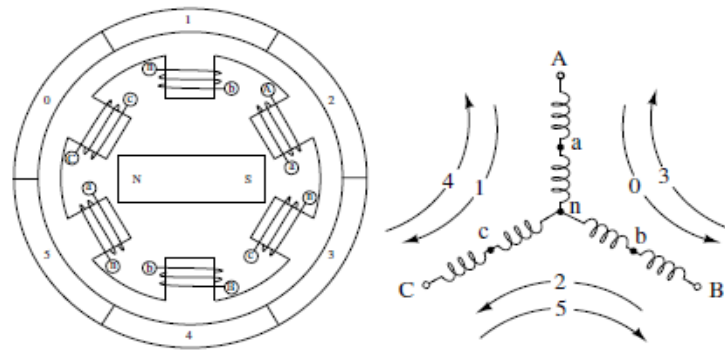


Figura 2.18. Bobinado del motor BLDC en conexión estrella (Ref. [10])

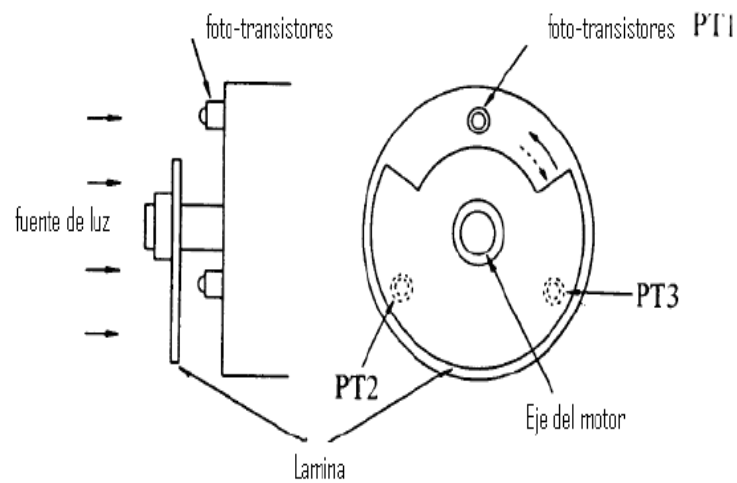


Figura 2.19. Sensores de efecto Hall (Ref. [11])

#### 2.2.4.4 VENTAJAS DE LOS MOTORES BLDC

Los motores BLDC tienen muchas ventajas frente a los motores DC con escobillas y frente a los motores de inducción. Algunas de estas ventajas son:



- Mejor relación velocidad-par motor
- Mayor respuesta dinámica
- Mayor eficiencia
- Mayor vida útil
- Menor ruido
- Mayor rango de velocidad.

Además, la relación par motor-tamaño es mucho mayor, lo que implica que se puedan emplear en aplicaciones donde se trabaje con un espacio reducido.

Por otra parte, los motores BLDC tienen dos desventajas, que son las siguientes:

- 1.- tienen un mayor coste
- 2.- requieren un control bastante más complejo.

### 2.2.4.4.1 DC SIN ESCOBILLAS VS DC CON ESCOBILLAS

	<b>Motor BLDC</b>	<b>Motor con escobillas</b>
Commutación	Commutación electrónica basada en sensores de posición de efecto Hall	Commutación por escobillas
Mantenimiento	Mínimo	Periódico
Durabilidad	Mayor	Menor
Curva Velocidad / par	Plana. Operación a todas las velocidades con la carga definida	Moderada. A altas velocidades la fricción de las escobillas se incrementa, reduciendo el par.
Eficiencia	Alta. Sin caída de tensión por las escobillas.	Moderada
Potencia de salida / Tamaño	Alta. Menor tamaño debido a mejores características térmicas porque los bobinados están en el estator, que al estar en la carcasa tiene una mejor disipación de calor.	Baja. El calor producido en la armadura es disipado en el interior aumentando la temperatura y limitando las características.
Inercia del rotor	Baja. Debido a los imanes permanentes en el rotor	Alta. Limita las características dinámicas.
Rango de velocidad	Alto. Sin limitaciones mecánicas impuestas por escobillas/conmutador.	Bajo. El límite lo imponen principalmente las escobillas.
Ruido eléctrico generado	Bajo	Arcos en las escobillas
Coste de construcción	Alto. Debido a los imanes permanentes	Bajo.
Control	Complejo y caro	Simple y barato.
Requisitos de control	Un controlador es requerido siempre para mantener el motor funcionando. El mismo puede usarse para variar la velocidad.	No se requiere control si no se requiere una variación de velocidad.

Tabla 2.12. Motor BLDC vs Motor con escobillas (Ref. [12])

### Diagrama de Tiempo del Motor BLDC

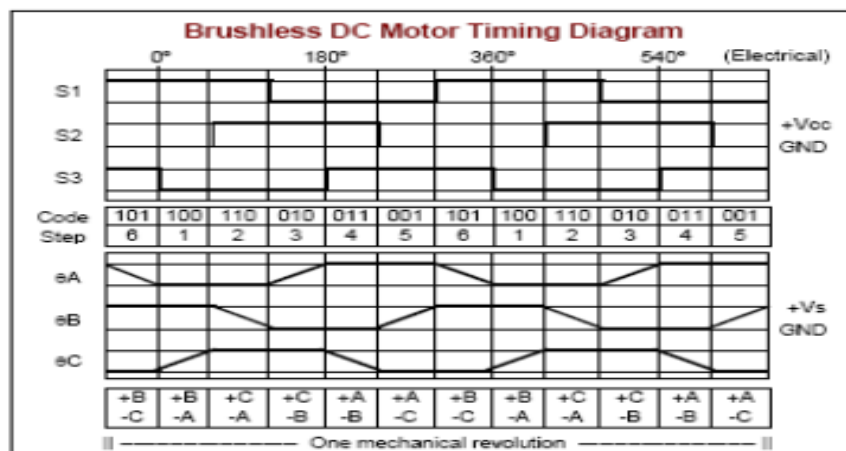


Figura 2.20. Diagrama de Tiempo (Ref.[11])

### 2.2.4.5 DISPARO PARA LOS MOSFETS

Aquí una breve explicación del funcionamiento de los puentes H con Mosfet's, explicando rápidamente cada una de sus partes. A pesar de la gran ventaja de los Mosfet's sobre los demás tipos de transistor, existen algunas desventajas en su uso. Como saben el MOSFET es un dispositivo que controla corriente con una entrada de voltaje. La mayoría de los Mosfet's (salvo Mosfet's especiales) usan voltajes base- surtidor de entre 10 a 12 o 15V para poder lograr su saturación, si se aplica una diferencia de voltaje menor pues el MOSFET se comportara como un transductor lineal, es decir la salida de corriente será proporcional al voltaje aplicado y esto no es lo que queremos, lo que necesitamos es el Mosfet como interruptor de potencia.

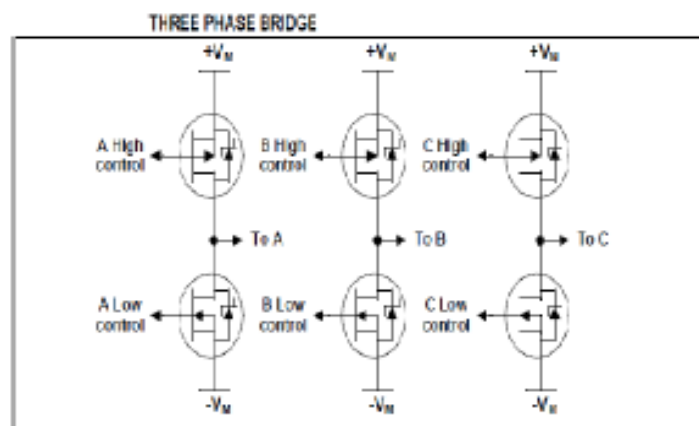


Figura 2.21. Disparo de Mosfets (Ref. [10])

**CW SENSOR AND DRIVE BITS BY PHASE ORDER**

Pin	RE2	RE1	RE0	RC5	RC4	RC3	RC2	RC1	RC0
Phase	Sensor C	Sensor B	Sensor A	C High Drive	C Low Drive	B High Drive	B Low Drive	A High Drive	A Low Drive
1	1	0	1	0	0	0	1	1	0
2	1	0	0	1	0	0	1	0	0
3	1	1	0	1	0	0	0	0	1
4	0	1	0	0	0	1	0	0	1
5	0	1	1	0	1	1	0	0	0
6	0	0	1	0	1	0	0	1	0

Tabla 2.13. Códigos según orden de fase de conmutación (Ref. [10])

# CAPÍTULO 3

## 3 TRABAJO REALIZADO

### 3.0 RESUMEN DEL CAPÍTULO

En este capítulo explicaremos de forma resumida los ejemplos desarrollados que nos han permitido llegar al circuito final para la composición y funcionamiento de nuestro proyecto, el cual está formado de tres partes básicas que son: Tarjeta Avr Butterfly, Tarjeta LPCxpresso 1769 y un motor BLDC.

Con estos elementos tratamos de construir un circuito capaz de controlar a través del joystick de la tarjeta Avr Butterfly el funcionamiento del motor BLDC que se encuentra conectado a los puertos de la tarjeta LPCxpresso.

Esto se llevará a cabo mediante comunicación RS232 que ejecutaremos entre la tarjeta Avr Butterfly y la LPCxpresso, para esto configuraremos

UART mediante software en el emisor (Avr Butterfly) y en el receptor (LPCxpresso) para que de esta manera el emisor envíe caracteres que representan el movimiento del joystick en forma de comandos y que luego en el receptor los interpretará para ser almacenados en una cadena de caracteres y será procesado por el programa según sea el movimiento que haya realizado el joystick para así ejecutar un movimiento determinado en el motor BLDC.

### **3.1 EJERCICIOS DESARROLLADOS**

**3.1.1 EJERCICIO 1:** Familiarización de los puertos de entrada y salida del GPIO.

#### **3.1.1.1 ESPECIFICACIÓN**

En este ejemplo aprenderemos a utilizar el registro GPIO, en este caso el GPIO2 para configurarlos como salida para el encendido de diodos Leds.

Además utilizaremos los registros FIOSET, FIOCLR y FIOPIN para setear o limpiar los puertos asignados y también configurarlos ya sea como entrada o salida.

Para realizar este trabajo configuraremos un pin de entrada, en el cual utilizaremos un switch que me permitirá cambiar la dirección de encendido

de los diodos Leds, los cuales se encenderán desde el centro hasta los extremos en forma consecutiva y al presionar el switch se encenderá en forma contraria.

### 3.1.1.2 LISTADO DE COMPONENTES

CANTIDAD	UNIDAD	DETALLE
1	u	Plataforma de Trabajo interactiva
1	u	Tarjeta LPCXpresso
8	u	Leds de propósito general
1	u	Switch
8	u	Resistencias de 330 $\Omega$ , 1/8 watt

Tabla 3.1. Lista de componentes del ejercicio 1

### 3.1.1.3 DIAGRAMA DE BLOQUES



Figura 3.1. Diagrama de bloques del ejercicio 1

### 3.1.1.4 DIAGRAMA DE BLOQUES DE LA CONEXIÓN ELÉCTRICA

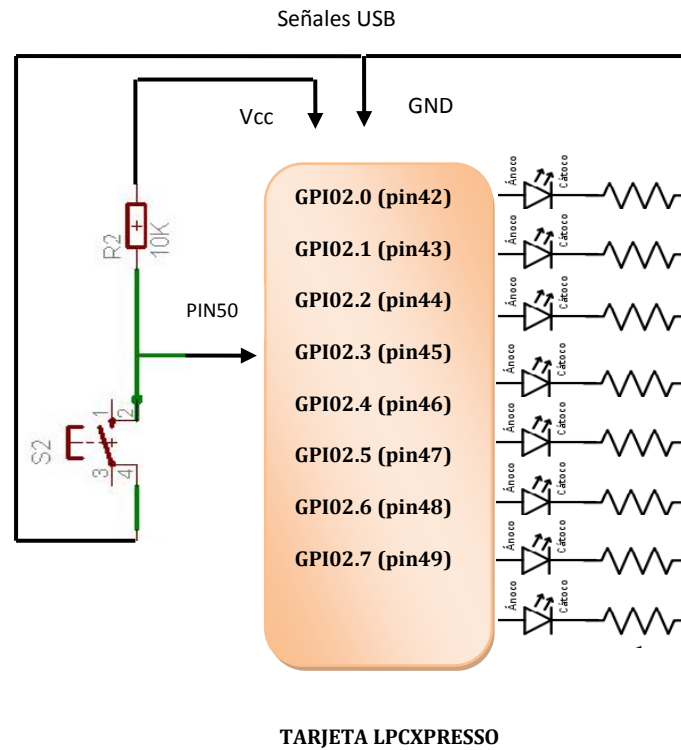


Figura 3.2 Conexión eléctrica del Ejercicio 1



### 3.1.1.5 DIAGRAMA DE FLUJO

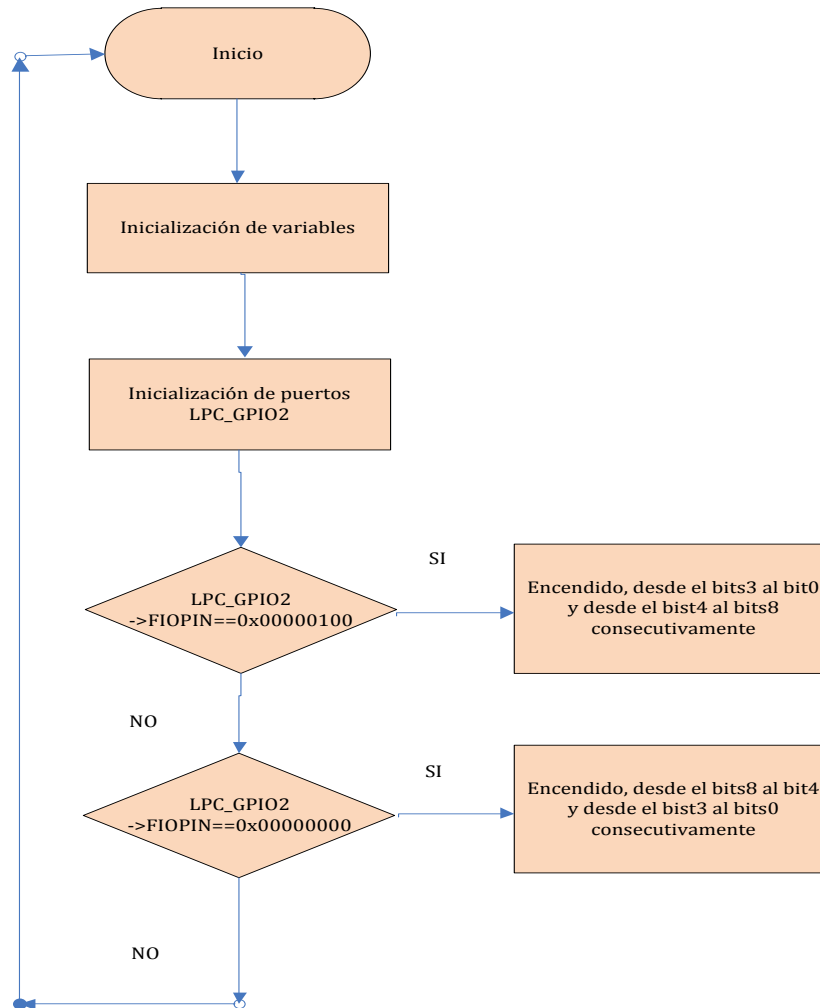


Figura 3.3 Diagrama de flujo del ejercicio 1

### 3.1.1.6 CÓDIGO

```

#include <cr_section_macros.h>

#include <NXP/crp.h>

// Variable to store CRP value in. Will be placed automatically
// by the linker when "Enable Code Read Protect" selected.
// See crp.h header for more information
__CRP const unsigned int CRP_WORD = CRP_NO_CRP ;

#include "lpc17xx.h"
#include "type.h"

int main (void)
{
    uint32_t i, j, k;

    /* SystemClockUpdate() updates the SystemFrequency variable */
    SystemClockUpdate();

    LPC_GPIO2->FIODIR = 0xFFFFFEFF;           /* P2.xx defined as
    Outputs */

    LPC_GPIO2->FIOCLR = 0xFFFFFFFF;          /* turn off all the LEDs
    */

    LPC_GPIO2->FIOMASK= 0x00000000;

```

```
while(1)
{
    k=4;
    if(LPC_GPIO2->FIOPIN==0x00000100){
        for(i = 7; i >3; i--)
        {
            LPC_GPIO2->FIOSET = 1 << k;

            LPC_GPIO2->FIOSET = 1 << (i-4);
            k++;
            for(j = 1000000; j > 0; j--);
        }

        LPC_GPIO2->FIOCLR = 0x000000FF;
        for(j = 1000000; j > 0; j--);
    }
}

if(LPC_GPIO2->FIOPIN==0x00000000){
    for(i = 0; i < 4; i++)
    {
        LPC_GPIO2->FIOSET = 1 << i;
        LPC_GPIO2->FIOSET = 1 << (7-i);
        for(j = 1000000; j > 0; j--);
    }

    LPC_GPIO2->FIOCLR = 0x000000FF;
```

```
for(j = 1000000; j > 0; j--);  
    }  
}  
}
```

### 3.1.1.7 IMÁGENES

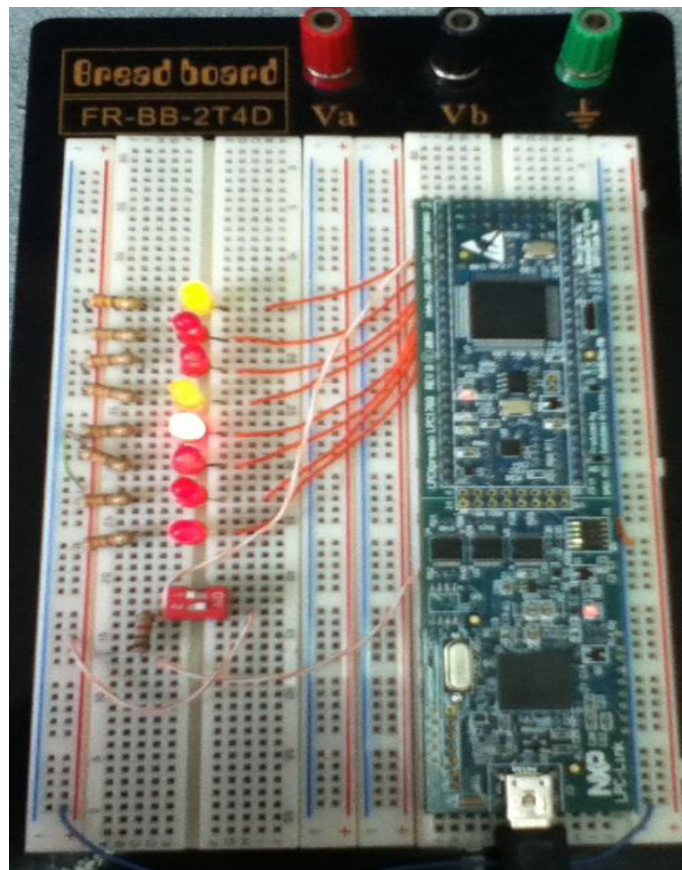


Figura 3.4. Encendido de leds desde el centro hacia los extremos.

### **3.1.1.8 RESULTADOS**

Con este ejemplo logramos familiarizarnos con el software LPCXpresso además aprendimos a usar los registros de la tarjeta especialmente el registro GPIO, el cual nos permite interactuar con cualquier tipo de dispositivo externo, ya sea con elementos de salida, elementos que nos proporcionan datos, o comunicaciones entre dos o mas tarjetas.

### **3.1.2 EJERCICIO 2: Comunicación Uart entre dos tarjetas LPCXpresso.**

#### **3.1.2.1 ESPECIFICACIÓN**

En este ejercicio mostraremos la comunicación entre dos tarjetas LPCXpresso mediante comunicación uart, para esto tomaremos una tarjeta LPCXpresso como emisor y otra como receptor.

En el emisor utilizaremos un switch como entrada la cual brindará una señal que será transmitida mediante la comunicación UART para luego ser adquirida por el receptor y encender un diodo Led.

Además utilizaremos los registros FIOSET, FIOCLR y FIOPIN para setear o limpiar los puertos asignados y configurarlos ya sea como entrada o salida. La variable UART3Buffer [BUFSIZE] nos permite almacenar y comparar el contenido bit por bit. Los registros: RBR contiene el siguiente caracter recibido para ser leído, THR

almacena el siguiente caracter a transmitir, IER contiene los bits de habilitación de interrupciones individuales de las 7 posibles interrupciones del UART y RLS es la Line Status Rx.

### 3.1.2.2 LISTADO DE COMPONENTES

CANTIDAD	UNIDAD	DETALLE
1	u	Plataforma de Trabajo interactiva
2	u	Tarjeta LPCXpresso
1	u	Leds de propósito general
1	u	Switch
1	u	Resistencias de 330 $\Omega$ , 1/8 watt
2	u	Resistencias de 1k $\Omega$

Tabla 3.2 Lista de componentes del ejercicio 2

### 3.1.2.3 DIAGRAMA DE BLOQUES

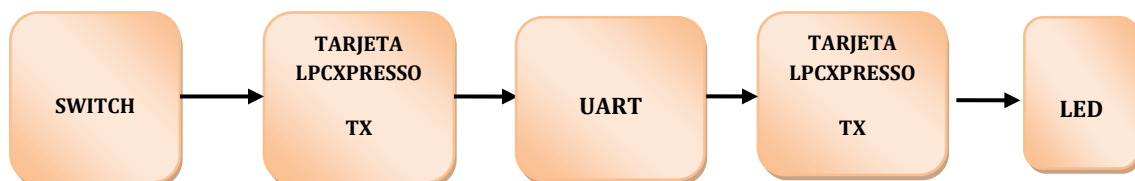


Figura 3.5 Diagrama de bloques del ejercicio 2

### 3.1.2.4 DIAGRAMA DE BLOQUES DE LA CONEXIÓN ELÉCTRICA

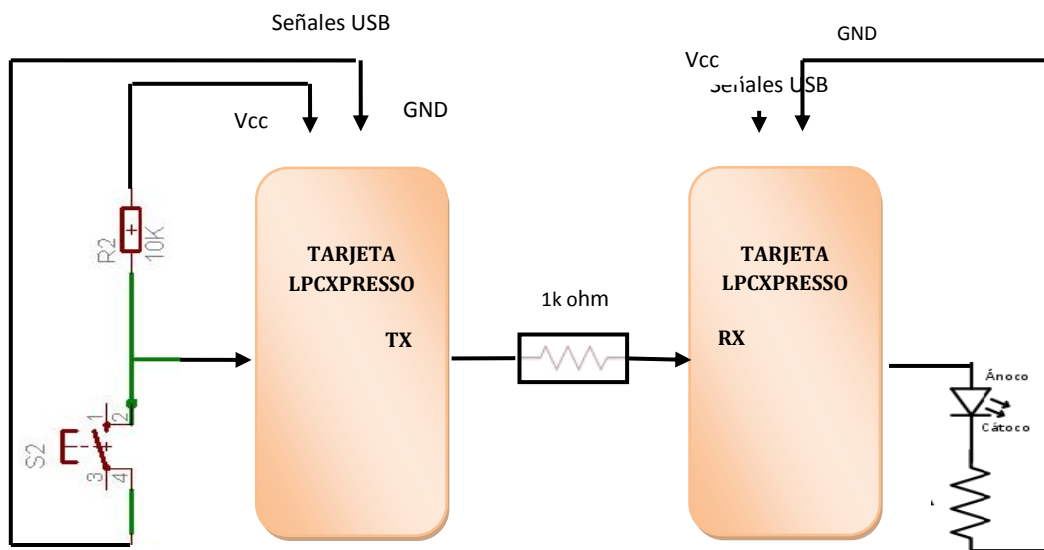


Figura 3.6 Conexión eléctrica del ejercicio 2

### 3.1.2.5 DIAGRAMA DE FLUJO

#### Comunicación UART del transmisor

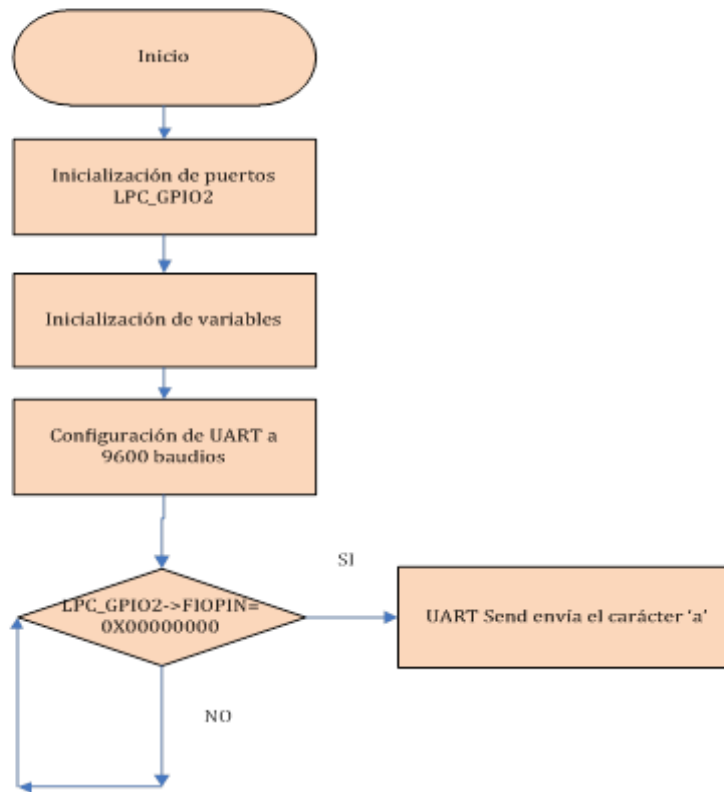


Figura 3.7. Diagrama de flujo del transmisor ejercicio 2



### Comunicación UART del receptor

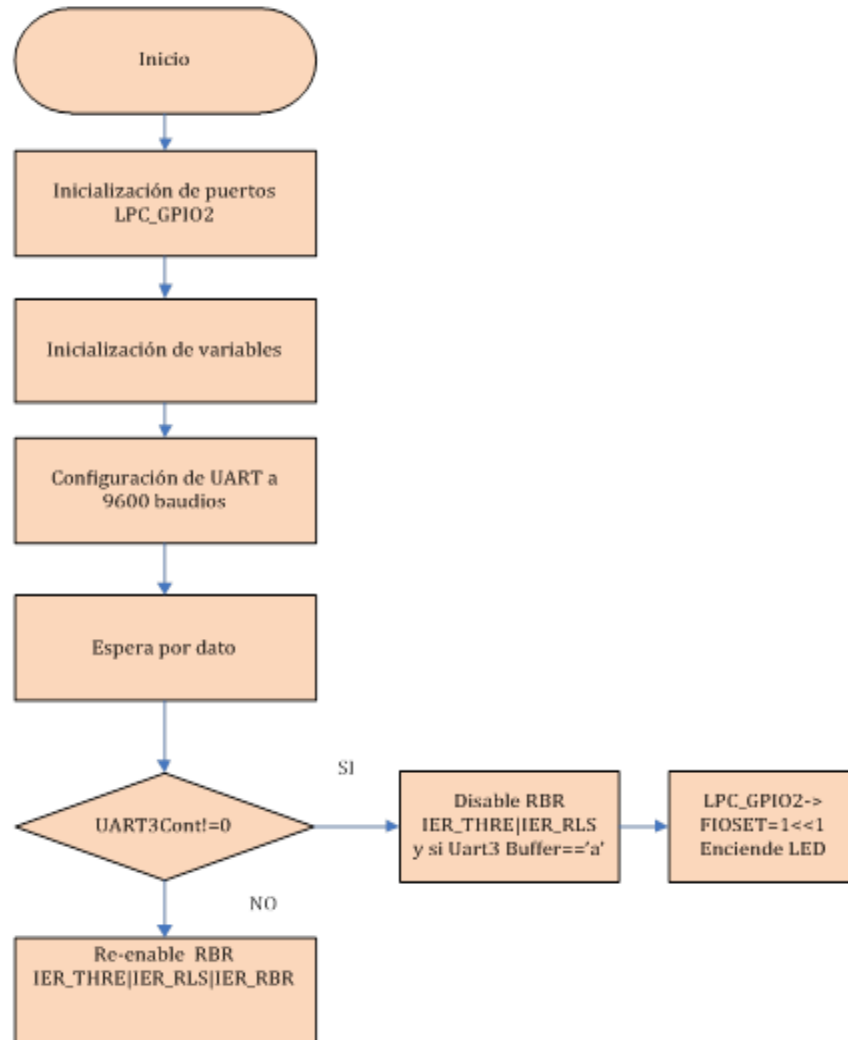


Figura 3.8. Diagrama de flujo del receptor ejercicio 2

### 3.1.2.6 CÓDIGO

#### Transmisor

```

#include <cr_section_macros.h>
#include <NXP/crp.h>
#include "lpc17xx.h"
#include "type.h"
#include "uart.h"
#include <string.h>
/*extern volatile uint32_t UART3Count;
extern volatile uint8_t UART3Buffer[BUFSIZE];*/
int main (void)
{
    const char* encender = 'a';
    uint32_t i, j, BOTON;
    UARTInit(3, 9600);    /* baud rate setting */

    LPC_GPIO2->FIODIR = 0xFFFFFFFF;    /* P2.xx defined as
Outputs */
    LPC_GPIO2->FIOCLR = 0xFFFFFFFF;    /* turn off all the LEDs
*/
    LPC_GPIO2->FIOMASK= 0x00000000;
    while(1)
    {

```

```

        if(LPC_GPIO2->FIOPIN==0x00000000 ){
            UARTSend(3, (uint8_t *)encender , strlen(encender) );
        }
    }
}

```

### Receptor

```

#include "LPC17xx.h"
#include "type.h"
#include "uart.h"
#include <string.h>

extern volatile uint32_t UART3Count;
extern volatile uint8_t UART3Buffer[BUFSIZE];

/*****

Main Function main()

This program has been test on LPCXpresso 1700.

*****/

int main (void)
{
    uint32_t i, j;
    LPC_GPIO2->FIODIR = 0xFFFFFFFF; /* P2.xx defined as Outputs */

```

```
LPC_GPIO2->FIOCLR = 0xFFFFFFFF;
UARTInit(3, 9600); /* baud rate setting */
/* Loop forever */
while (1)
{
    if ( UART3Count != 0 )
    {
        LPC_UART3->IER = IER_THRE | IER_RLS; /* Disable RBR */
        if(UART3Buffer=="a")
        {
            LPC_GPIO2->FIOSET = 1 << 1;
            for(j = 5000000; j > 0; j--);
            LPC_GPIO2->FIOCLR = 0x000000FF;
            for(j = 5000000; j > 0; j--);
        }
        UART3Count = 0;
        LPC_UART3->IER = IER_THRE | IER_RLS | IER_RBR;

        /* Re-enable RBR */
    }
}
}
```

### 3.1.2.7 IMÁGENES

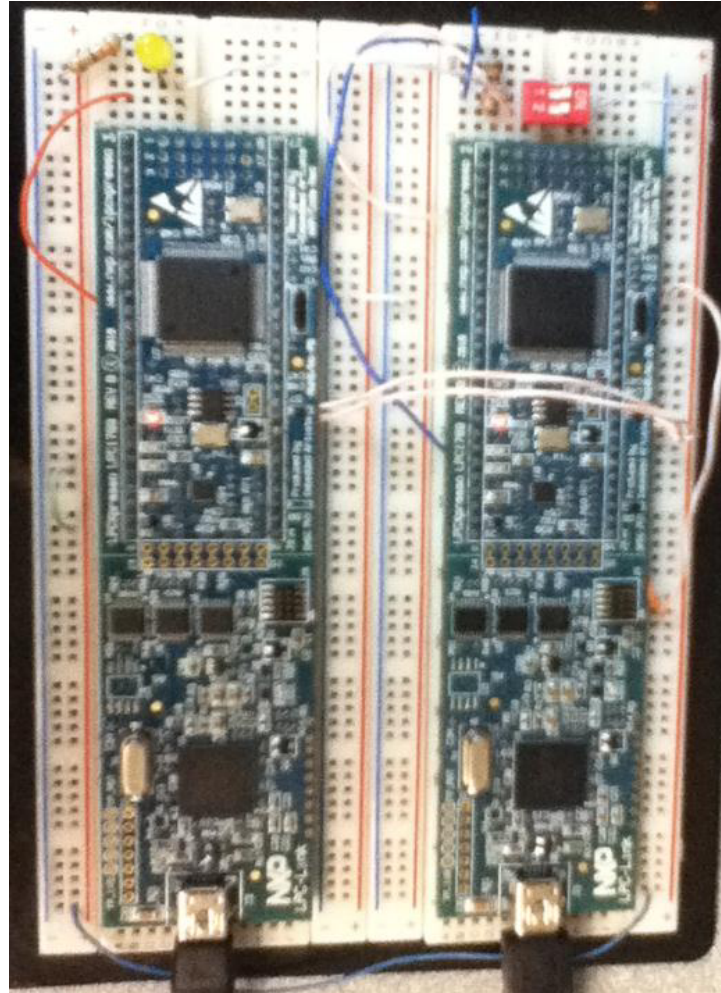


Figura 3.9. Comunicación UART con las tarjetas LPCXpresso.

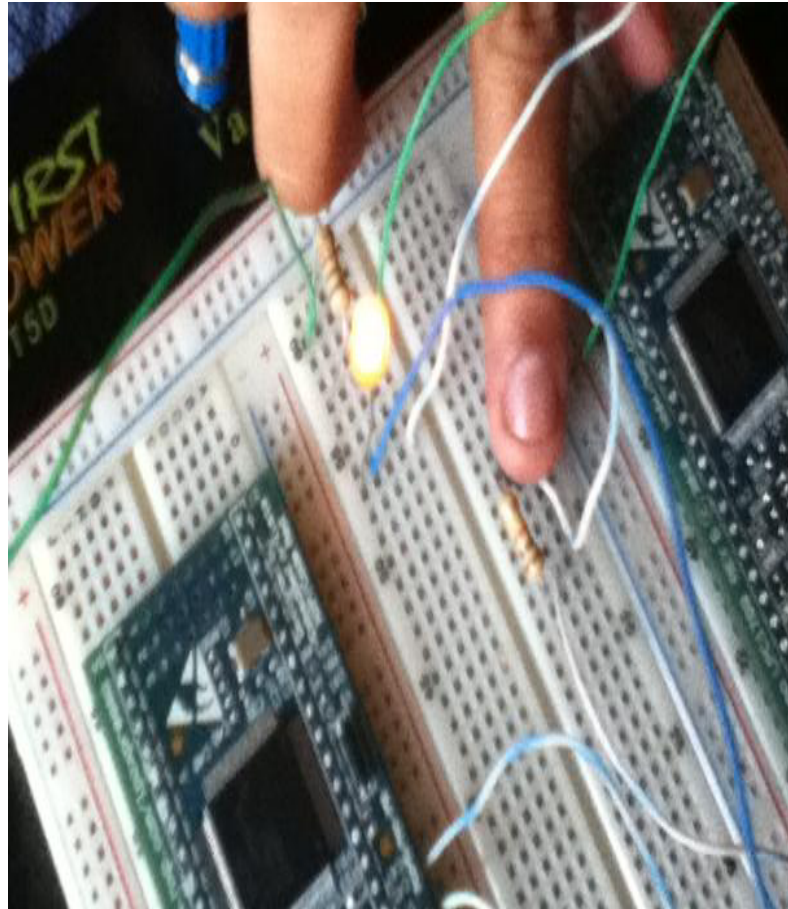


Figura 3.10. Encendido de LED a través de comunicación UART con dos tarjetas LPCXpresso.

### 3.1.2.8 RESULTADOS

Este ejemplo nos ayudó a comprender la comunicación RS232 usando dos tarjetas LPCXpresso que consiste en la transmisión y recepción de datos usando el código de programación uart que es necesario para este tipo de comunicación.

### **3.1.3 EJERCICIO 3: El Joystick y las Interrupciones Externas.**

#### **3.1.3.1 ESPECIFICACIÓN**

El programa permite validar la funcionalidad el Joystick de la AVR Butterfly utilizando las interrupciones externas del ATmega169 y emplear el joystick para según la manipulación de éste desplazar bits que se mostrarán en LEDs conectados al puerto D.

Al comenzar aparece en la pantalla Lcd el mensaje “ROSERO-CASTELO”, y posteriormente si interactuamos con el Joystick se transmite el dato que se genera de la siguiente manera:

- ARRIBA.- desplazará el dato actual en el puerto D, o sea la luz del LED, dos posiciones hacia la derecha.
- ABAJO.- desplazará el dato actual en el puerto D, o sea la luz del LED, dos posiciones hacia la izquierda.
- DERECHA.- desplazará el dato actual en el puerto D, o sea la luz del LED, una posición hacia la derecha.
- IZQUIERDA.- desplazará el dato actual en el puerto D, o sea la luz del LED, una posición hacia la izquierda.
- CENTRO.- borrará el contenido del puerto D y cargará un bit en el primer pin del mismo puerto (PIND0), lo que activará la luz del LED únicamente en aquel pin

El joystick con el que está equipado el AVR Butterfly está conectado como se indica en la Figura 3.11.

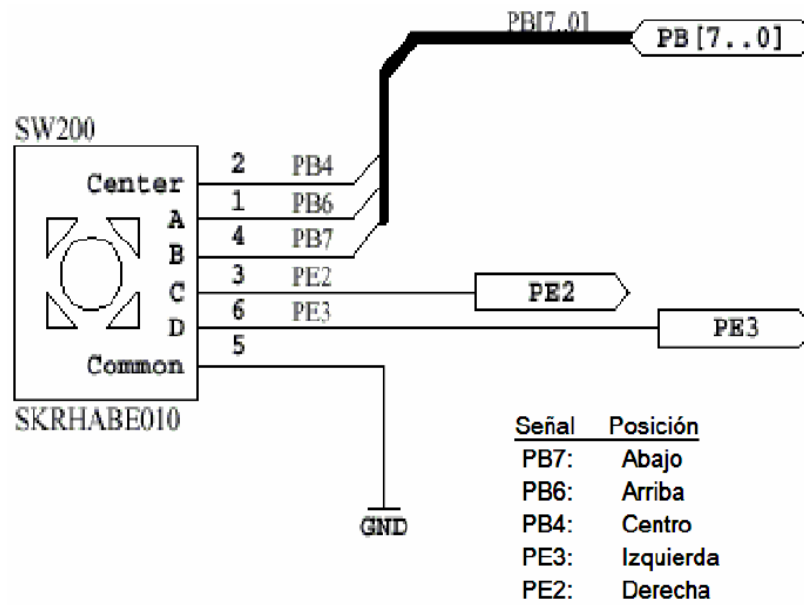


Figura 3.11. Diagrama de conexión del joystick



### 3.1.3.2 LISTADO DE COMPONENTES

CANTIDAD	UNIDAD	DETALLE
1	u	Plataforma de Trabajo interactiva
1	u	Tarjeta Electrónica AVR-BUTTERFLY
8	u	Leds de propósito general
2	u	Baterías AA (1.5 V)
1	u	Porta baterías
8	u	Resistencias de 330 $\Omega$ , 1/8 watt

Tabla 3.3 Lista de componentes del ejercicio 3

### 3.1.3.3 DIAGRAMA DE BLOQUES

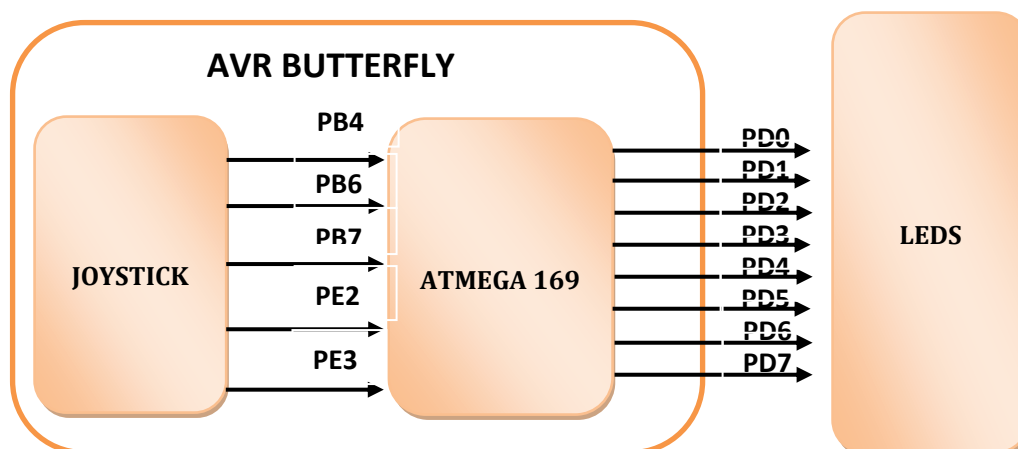


Figura 3.12. Diagrama de bloques del ejercicio 3

Como ilustra el diagrama de bloques, el Butterfly emplea los puertos B y E del ATmega169 para acceder al Joystick. En esta práctica también se empleará el Puerto D, accesible a conexiones externas en el AVR Butterfly, para mostrar la acción que se realizará en concordancia con la selección hecha con el joystick.

Para visualizar las acciones en el Puerto D, se conectará un LED a cada uno de los pines del puerto D.

### 3.1.3.4 DIAGRAMA DE BLOQUES DE LA CONEXIÓN ELÉCTRICA

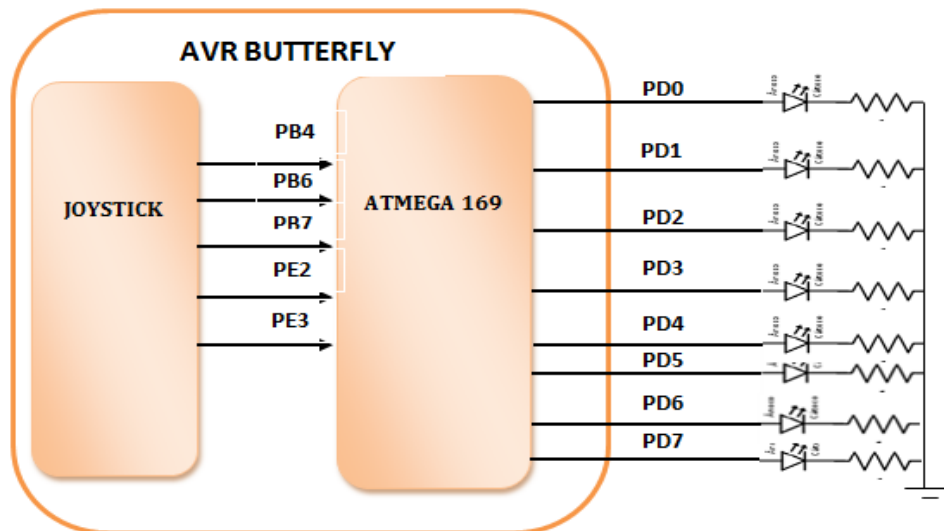


Figura 3.13. Conexión eléctrica del ejercicio 3

### 3.1.3.5 DIAGRAMA DE FLUJO

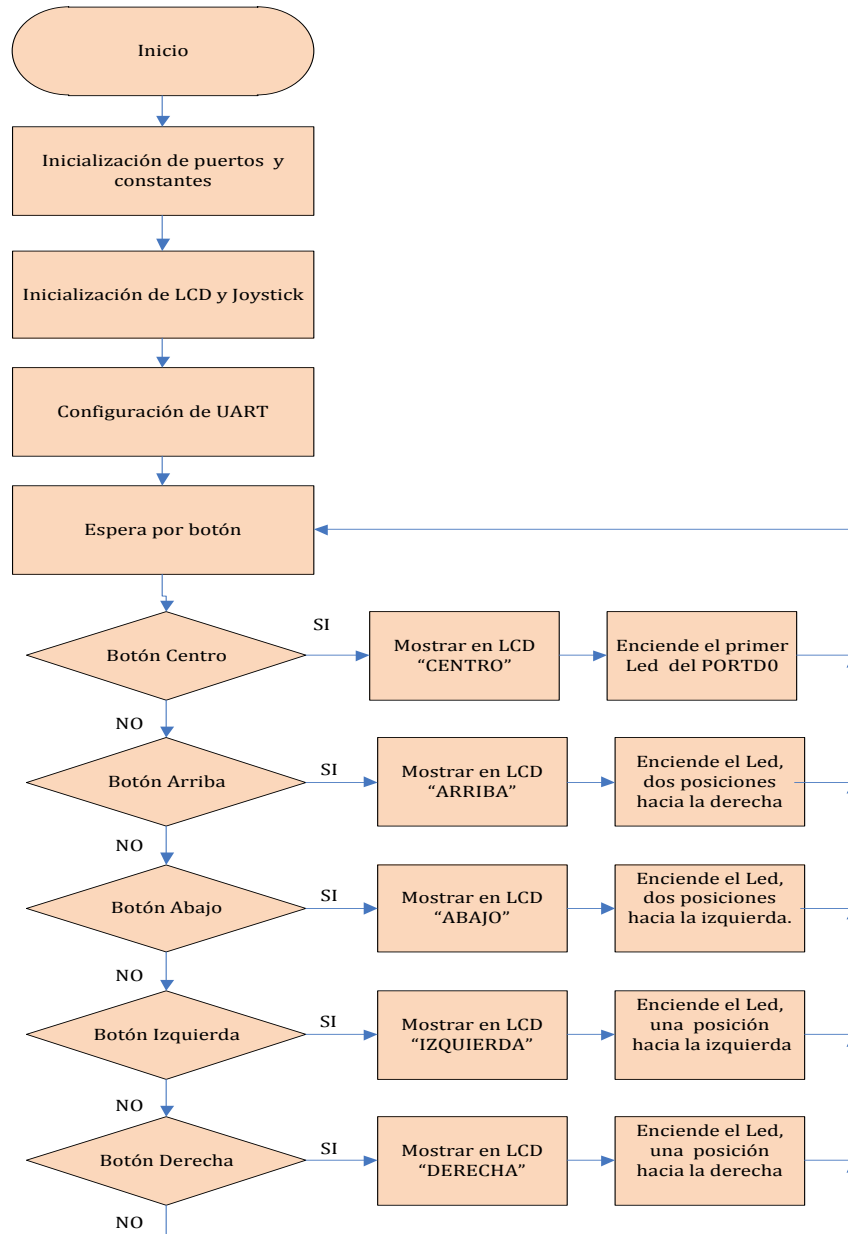


Figura 3.14. Diagrama de flujo del ejercicio 3

### 3.1.3.6 CÓDIGO

```
#include <avr/io.h>
```

```
#include <avr/interrupt.h>
```

```
#include <avr/signal.h>
```

```
/*
```

Pines del ATmega169 conectados con el Joystick:

```
-----
```

```
Bit   7 6 5 4 3 2 1 0
```

```
-----
```

```
PORTB B A O
```

```
PORTE D C
```

```
-----
```

```
PORTB | PORTE B A O D C => posición
```

```
-----
```

```
*/
```

```
#define MASCARA_PINB ((1<<PINB7)|(1<<PINB6)|(1<<PINB4))
```

```
#define MASCARA_PINE ((1<<PINE3)|(1<<PINE2))
```

```
#define ARRIBA 0
```

```
#define ABAJO 1
```

```
#define IZQUIERDA 2
```

```
#define DERECHA 3
#define CENTRO 4
#define NO_VALIDA 5
#define posicion_A 6 //ARRIBA
#define posicion_B 7 //ABAJO
#define posicion_C 2 //DERECHA
#define posicion_D 3 //IZQUIERDA
#define posicion_O 4 //CENTRO
#define VERDADERO 1
#define FALSO 0

volatile unsigned char SELECCION = 0;
volatile unsigned char SELECCION_VALIDA = 0;
void inicializar(void);
void manejar_interrupcion(void);
void obtener_seleccion(void);
int main(void)

{
inicializar();
sei();
while(1)
```

```
{  
SMCR = ((1<<SM1)|(1<<SE));  
}  
  
return 0;  
  
}  
void inicializar(void)  
{  
  
CLKPR = (1<<CLKPCE);  
CLKPR = (1<<CLKPS3);  
while(CLKPR & (1<<CLKPCE));  
DDRB |= 0xD0;  
PORTB |= MASCARA_PINB;  
DDRE |= 0x0C;  
PORTE |= MASCARA_PINE;  
DDRB = 0;//entrada  
PORTB = MASCARA_PINB;//habilitar PULL-UPs  
DDRE = 0;//entrada  
PORTE = MASCARA_PINE;//habilitar PULL-UPs  
PCMSK1 |= MASCARA_PINB;  
PCMSK0 |= MASCARA_PINE;
```

```
EIFR = ((1<<PCIF1)|(1<<PCIF0));
EIMSK = ((1<<PCIE1)|(1<<PCIE0));
DDRD = 0xFF;
PORTD = 0x00;

}

void manejar_interrupcion(void)
{

    unsigned char joystick;
    unsigned char seleccion;
    joystick = ((~PINB) & MASCARA_PINB);
    joystick |= ((~PINE) & MASCARA_PINE);
    if((joystick & (1<< posicion_A)))
        seleccion = ARRIBA;
    else if((joystick & (1<< posicion_B)))
        seleccion = ABAJO;
    else if((joystick & (1<< posicion_C)))
        seleccion = DERECHA;
    else if((joystick & (1<< posicion_D)))
        seleccion = IZQUIERDA;
    else if((joystick & (1<< posicion_O)))
        seleccion = CENTRO;
```

```
else seleccion = NO_VALIDA;
if(seleccion != NO_VALIDA)

{
if(!SELECCION_VALIDA)
{

SELECCION = seleccion;
SELECCION_VALIDA = VERDADERO;
}

}

EIFR = ((1<<PCIF1)|(1<<PCIF0));
obtener_seleccion();
}

void obtener_seleccion(void)
{
unsigned char seleccion;
cli();
if(SELECCION_VALIDA)
{
seleccion = SELECCION;
SELECCION_VALIDA = FALSO;
}
```



```
}  
else seleccion = NO_VALIDA;  
if(seleccion != NO_VALIDA)  
{  
  
switch(seleccion)  
  
{  
case ARRIBA:  
PORTD <<= 2;  
break;  
case ABAJO:  
PORTD >>= 2;  
break;  
case IZQUIERDA:  
PORTD <<= 1;  
break;  
case DERECHA:  
PORTD >>= 1;  
break;  
case CENTRO:  
PORTD = 1;  
default:
```

```
break;
}
}
sei();
}
SIGNAL(SIG_PIN_CHANGE0)
{
manejar_interrupcion();
}
SIGNAL(SIG_PIN_CHANGE1)
{
manejar_interrupcion();
}
```

### 3.1.3.7 IMÁGENES

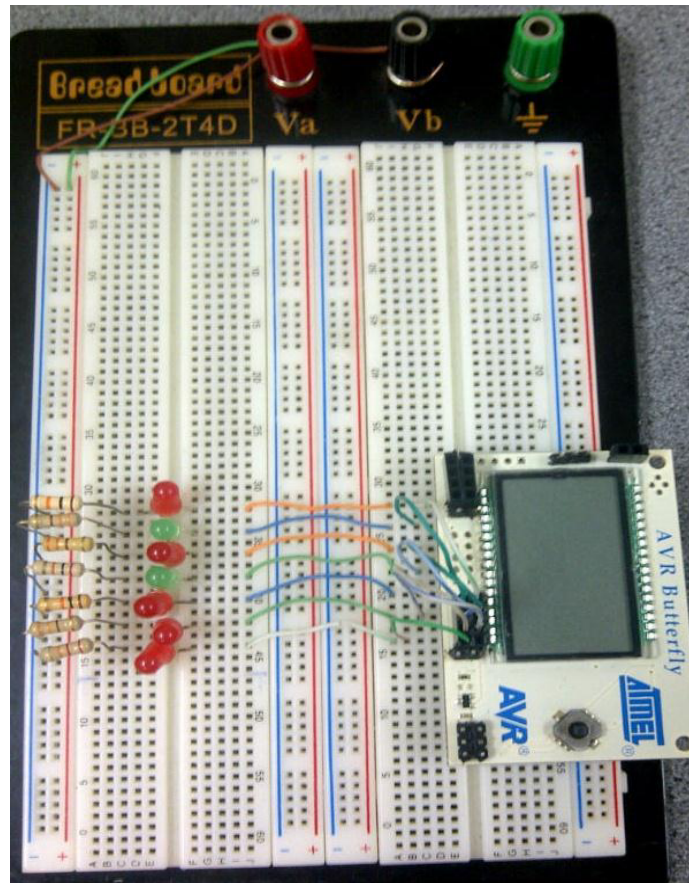


Figura 3.15. Conexión de Leds del puerto D de la AVR Butterfly.

### 3.1.3.8 RESULTADOS

Este ejemplo nos permitió conocer el ambiente de trabajo para la programación de la tarjeta AVR Butterfly como también el funcionamiento de su respectivo joystick haciendo uso de sus puertos como salida.

**3.1.4 EJERCICIO 4:** Control del joystick de tarjeta AVR Butterfly mediante comunicación RS232 con tarjeta LPCXpresso para el encendido de Leds.

#### **3.1.4.1 ESPECIFICACIÓN**

Este ejemplo se encuentra constituido por la etapa de transmisión y la etapa de la recepción.

La etapa de transmisión está constituida por el Kit AVR Butterfly que envía caracteres por el puerto uart dependiendo del movimiento que se ha realizado en el joystick y que servirá de condición para que la tarjeta LPCXpresso controle los motores.

En la etapa de recepción se encuentra la tarjeta LPCXpresso, la cual toma los caracteres que fueron enviados desde el transmisor a través de comunicación RS232 y el cual asignará un puerto específico para cada acción que realice el joystick en el transmisor para el respectivo encendido de diodos Leds.

#### **3.1.4.2 LISTADO DE COMPONENTES**

<b>CANTIDAD</b>	<b>UNIDAD</b>	<b>DETALLE</b>
1	U	Plataforma de Trabajo interactiva

1	U	Tarjeta Electrónica AVR-BUTTERFLY
1	U	Tarjeta LPCXpresso
8	U	Leds de propósito general
2	U	Baterías AA (1.5 V)
1	U	Porta baterías
8	U	Resistencias de 330 $\Omega$ , 1/8 watt

Tabla 3.4 Lista de componentes del ejercicio 4

### 3.1.4.3 DIAGRAMA DE BLOQUES



Figura 3.16. Diagrama de bloques del ejercicio 4

### 3.1.4.4 DIAGRAMA DE BLOQUES DE LA CONEXIÓN ELÉCTRICA

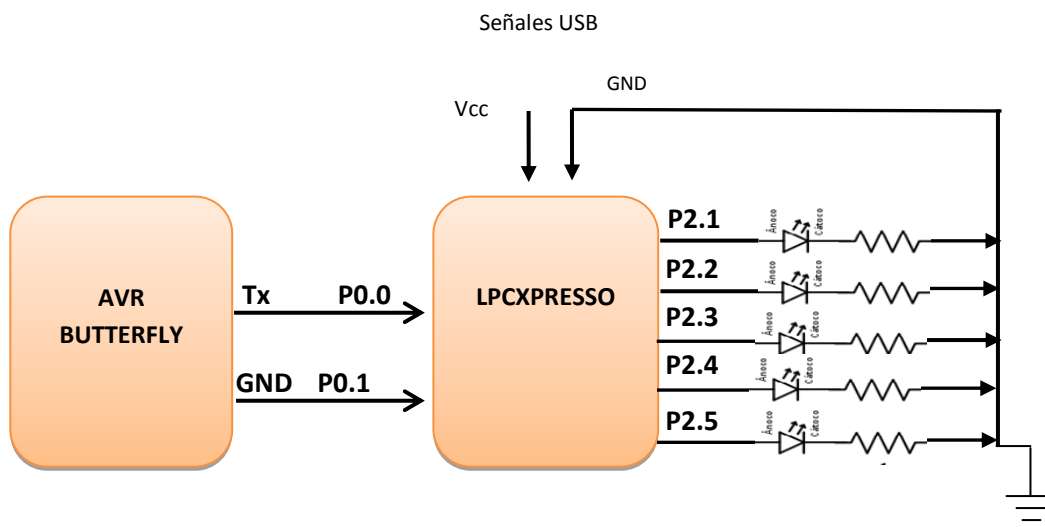


Figura 3.17. Conexión eléctrica del ejercicio 4

### 3.1.4.5 DIAGRAMA DE FLUJO

#### Transmisor

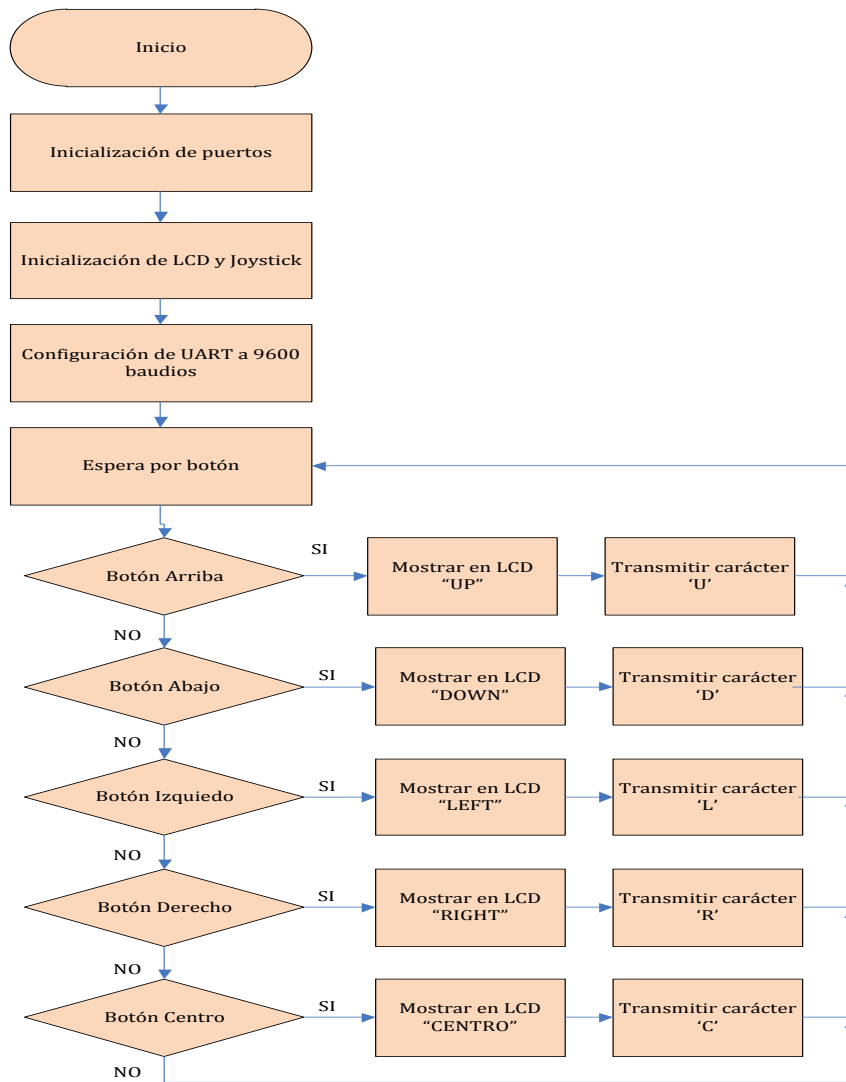


Figura 3.18. Diagrama de flujo del transmisor del ejercicio 4

## Receptor

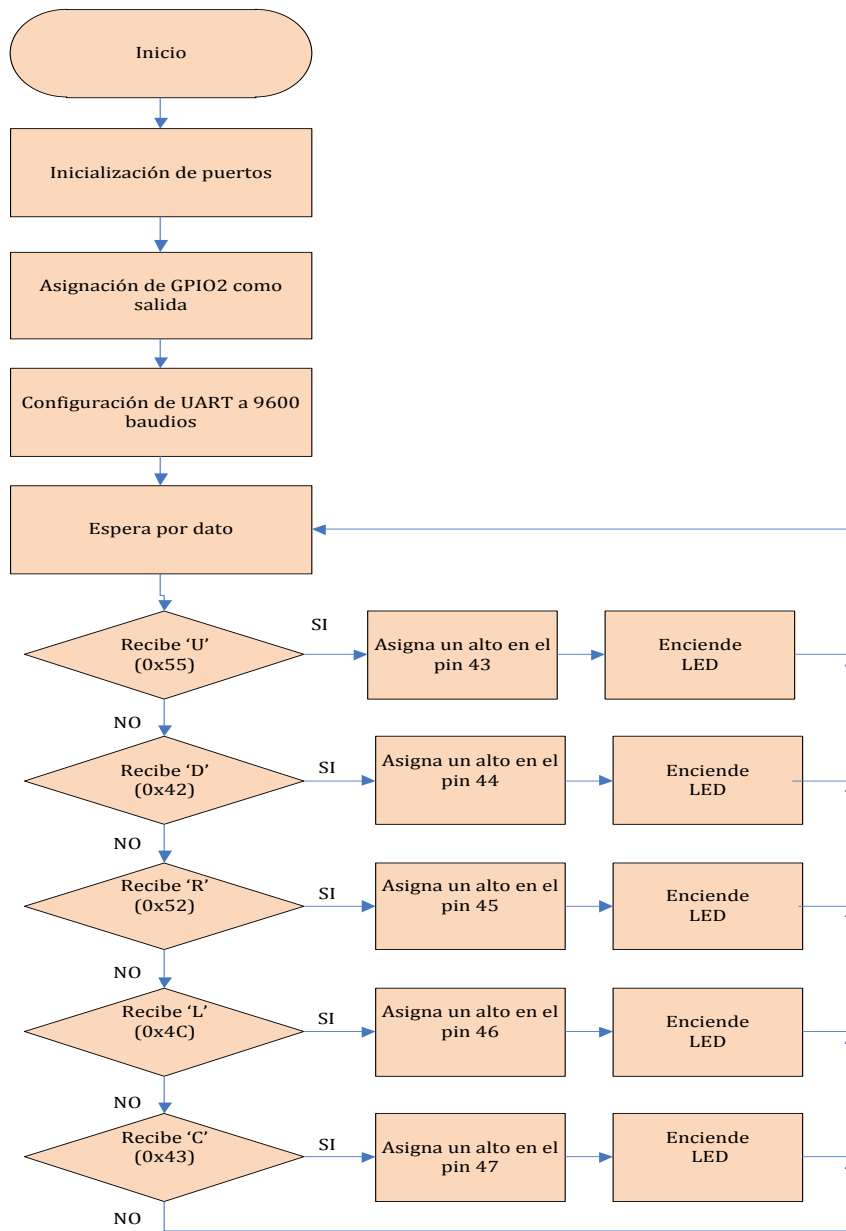


Figura 3.19. Diagrama de flujo del receptor del ejercicio 4



### 3.1.4.6 CÓDIGO

En la tarjeta AVR Butterfly

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <avr/delay.h>
#include <inttypes.h>
#include "mydefs.h"
#include "LCD_functions.h"
#include "LCD_driver.h"
#include "button.h"
#include "usart.h"
int main(void)
{
    PGM_P statetext = PSTR("ROSERO-CASTELO");
    uint8_t input;
    char *cadena;
    int i;

    // Disable Analog Comparator (power save)
    ACSR = (1<<ACD);
```

```
// Disable Digital input on PF0-2 (power save)
DIDR0 = (7<<ADC0D);

// Enable pullups
PORTB = (15<<PB0);
PORTE = (15<<PE4);

Button_Init();      // Initialize pin change interrupt on joystick
LCD_Init();         // initialize the LCD

CLKPR = (1<<CLKPCE); // set Clock Prescaler Change Enable
// set prescaler = 8, Inter RC 8Mhz / 8 = 1Mhz
CLKPR = (0<<CLKPS1) | (1<<CLKPS0);

USART_Init(25.04);
//sei();

while (1)
{
    if (statetext){
        LCD_puts_f(statetext, 1);
        LCD_Colon(0);
        statetext = NULL;
    }

    input = getkey(); // Read buttons
```

```
switch (input) {  
    case KEY_ENTER:  
        statetext = PSTR("CENTRO");  
        Usart_Tx('C');  
        break;  
    case KEY_NEXT:  
        statetext = PSTR("RIGHT");  
        Usart_Tx('R');  
        break;  
  
    case KEY_PREV:  
        statetext = PSTR("LEFT");  
        Usart_Tx('L');  
        break;  
    case KEY_PLUS;  
        statetext = PSTR("UP");  
        Usart_Tx('U');  
        break;  
    case KEY_MINUS:  
        statetext = PSTR("DOWN");  
        Usart_Tx('D');  
        break;
```

```

        default:
            break;
    }
}
return 0;
}

```

### En la tarjeta LPCXpresso

```

#include "LPC17xx.h"
#include "type.h"
#include "uart.h"
#include <string.h>

extern volatile uint32_t UART3Count;
extern volatile uint8_t UART3Buffer[BUFSIZE];

/*****

Main Function main()

This program has been test on LPCXpresso 1700.

*****/

int main (void)
{
    uint32_t i, j;

    LPC_GPIO2->FIODIR = 0xFFFFFFFF; /* P2.xx defined as Outputs */

```

```
LPC_GPIO2->FIOCLR = 0xFFFFFFFF;
UARTInit(3, 9600);           /* baud rate setting */
/* Loop forever */
while (1)
{
    if ( UART3Count != 0 )
    {
        LPC_UART3->IER = IER_THRE | IER_RLS; /* Disable RBR */
        if(*UART3Buffer==0x55){
            LPC_GPIO2->FIOSET = 1 << 1;
            for(j = 10000000; j > 0; j--);
            LPC_GPIO2->FIOCLR = 0x000000FF;
            for(j = 10000000; j > 0; j--);
        }
        if(*UART3Buffer==0x44){
            LPC_GPIO2->FIOSET = 1 << 2;
            for(j = 10000000; j > 0; j--);
            LPC_GPIO2->FIOCLR = 0x000000FF;
            for(j = 10000000; j > 0; j--);
        }
        if(*UART3Buffer==0x52){
            LPC_GPIO2->FIOSET = 1 << 3;
            for(j = 10000000; j > 0; j--);
        }
    }
}
```

```
LPC_GPIO2->FIOCLR = 0x000000FF;
for(j = 10000000; j > 0; j--);
    }
if(*UART3Buffer==0x4c){
    LPC_GPIO2->FIOSET = 1 << 4;
    for(j = 10000000; j > 0; j--);
    LPC_GPIO2->FIOCLR = 0x000000FF;
    for(j = 10000000; j > 0; j--);
        }
if(*UART3Buffer==0x43){
    LPC_GPIO2->FIOSET = 1 << 5;
    for(j = 10000000; j > 0; j--);
    LPC_GPIO2->FIOCLR = 0x000000FF;
    for(j = 10000000; j > 0; j--);
        }

UART3Count = 0;
LPC_UART3->IER = IER_THRE | IER_RLS | IER_RBR;
    /* Re-enable RBR */
}
}
}
```

### 3.1.4.7 IMÁGENES

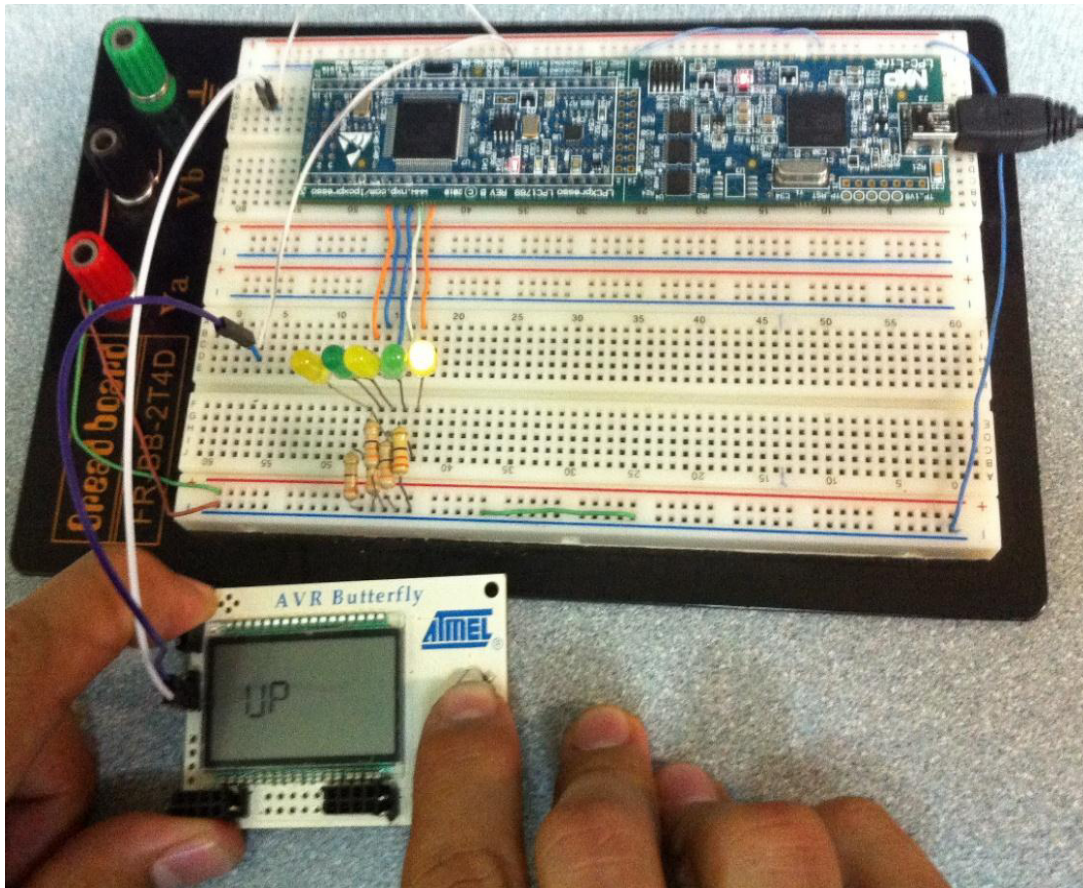


Figura 3.20. Presionando el botón UP

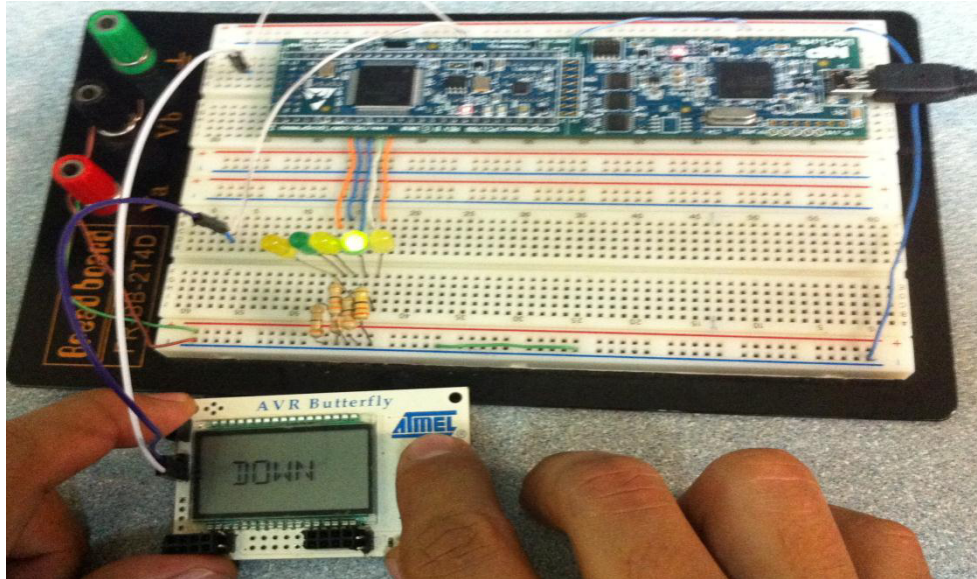


Figura 3.21. Presionando el botón DOWN

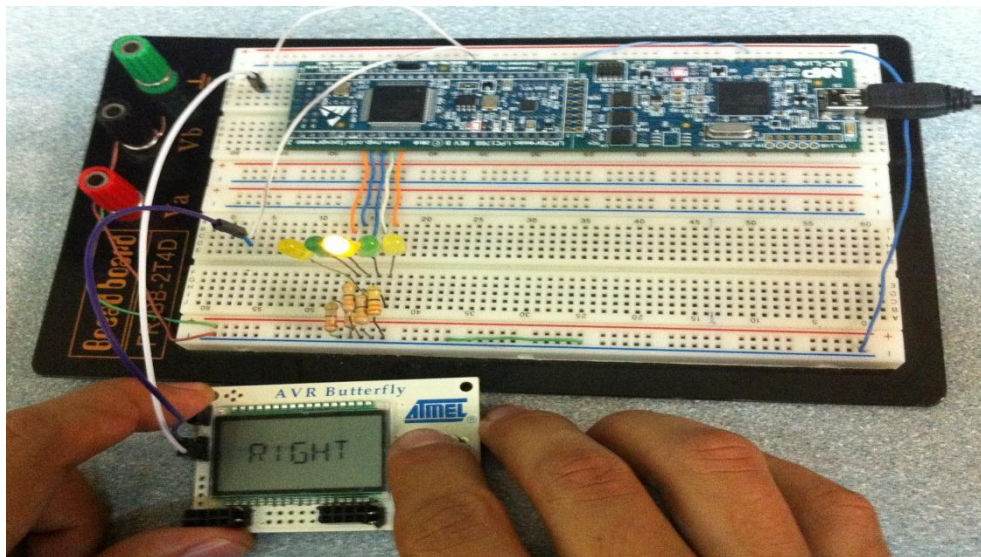


Figura 3.22. Presionando el botón RIGHT



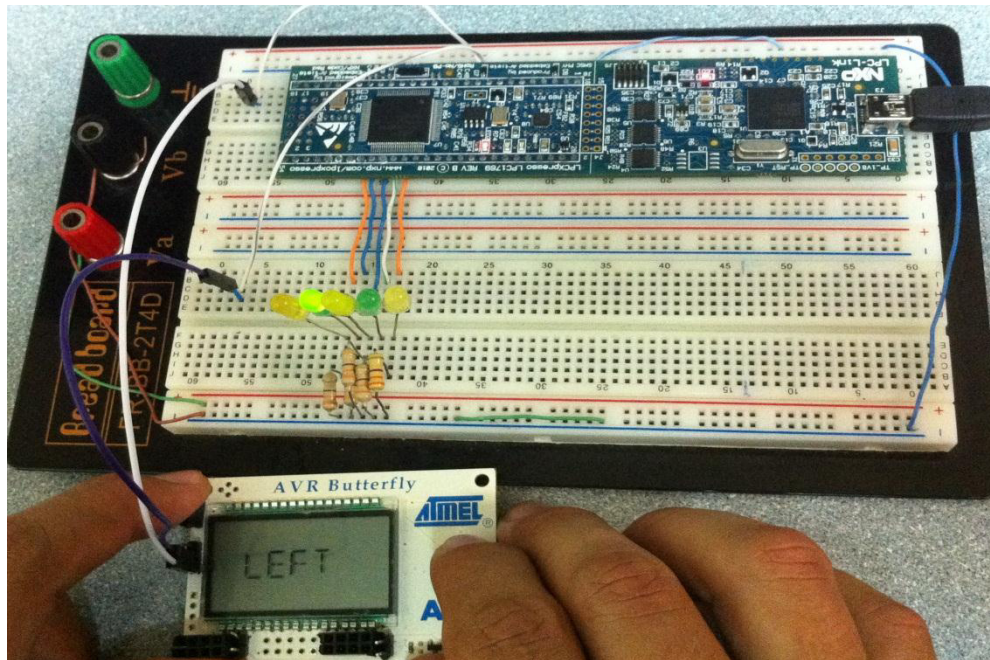


Figura 3.23. Presionando el botón LEFT

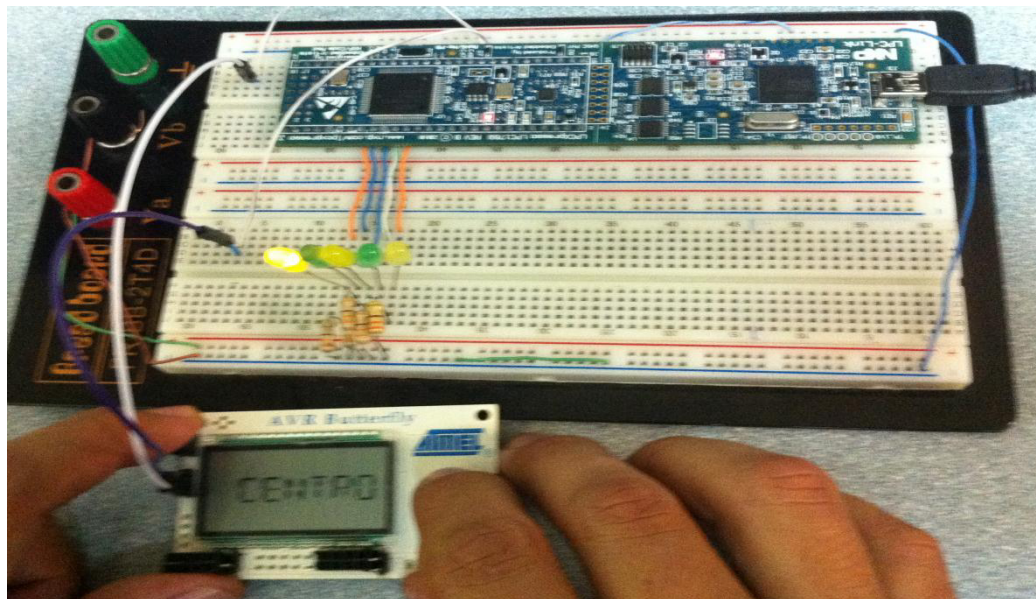


Figura 3.24. Presionando el botón CENTRO

### **3.1.4.8 RESULTADOS**

En este ejemplo logramos hacer la comunicación RS232 entre la tarjeta AVR Butterfly y la tarjeta LPCXpresso enviando instrucciones generadas desde el joystick y que serán enviadas mediante el puerto de comunicación uart de la AVR Butterfly para luego ser receptadas por la LPCXpresso que servirán de condiciones para el encendido de diodos leds a través del GPIO2 configurado como salida.

## **3.2 PROYECTO COMPLETO**

### **3.2.1 DIAGRAMA DE BLOQUES DEL PROYECTO**

El diagrama de bloques del sistema se encuentra constituido por la etapa del transmisor y la etapa del receptor.

La etapa de transmisión está constituida por el Kit AVR Butterfly que envía caracteres por el puerto uart dependiendo del movimiento que se ha realizado en el joystick y que servirá de condición para que la tarjeta LPCXpresso controle los motores.

En la etapa de recepción se encuentra la tarjeta LPCXpresso, la cual toma los caracteres que fueron enviados desde el transmisor a través de comunicación RS232 y el cual asignará un puerto específico para cada

acción que realice el joystick en el transmisor para el respectivo control de los motores BLDC.

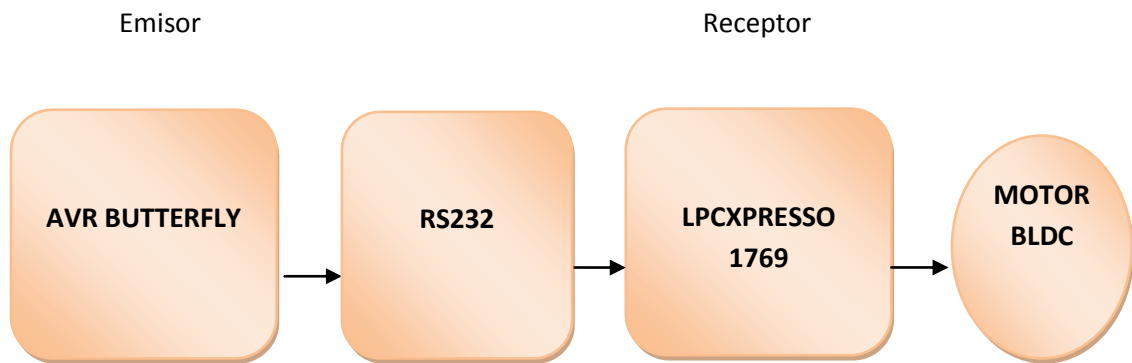


Figura 3.25 Diagrama de bloques del proyecto

### 3.2.1.1 TRANSMISOR

#### 3.2.1.1.1 AVR BUTTERFLY

Mediante el uso del Joystick del AVR Butterfly se usa las interrupciones generadas por las mismas por cambio de estado en lo cual se procede a enviar una trama por medio de la comunicación USART presente en el ATmega169 la cual contiene un caracter que indica la opción que ha sido realizada por el usuario de la siguiente manera:

- U: Up
- D: Down
- R: Right
- L: Left
- C: Centro

En la programación hemos configurado para que la AVR Butterfly envíe los caracteres a través de su puerto uart y además mostramos la opción que vamos a ejecutar por medio de la pantalla LCD para tener una mejor visualización de la tarea que se va a realizar, todo esto lo realizamos mediante el uso de funciones y librerías creadas por el fabricante para el uso de estas herramientas.

### 3.3.1.1.2 DIAGRAMA DE FLUJO DEL TRANSMISOR

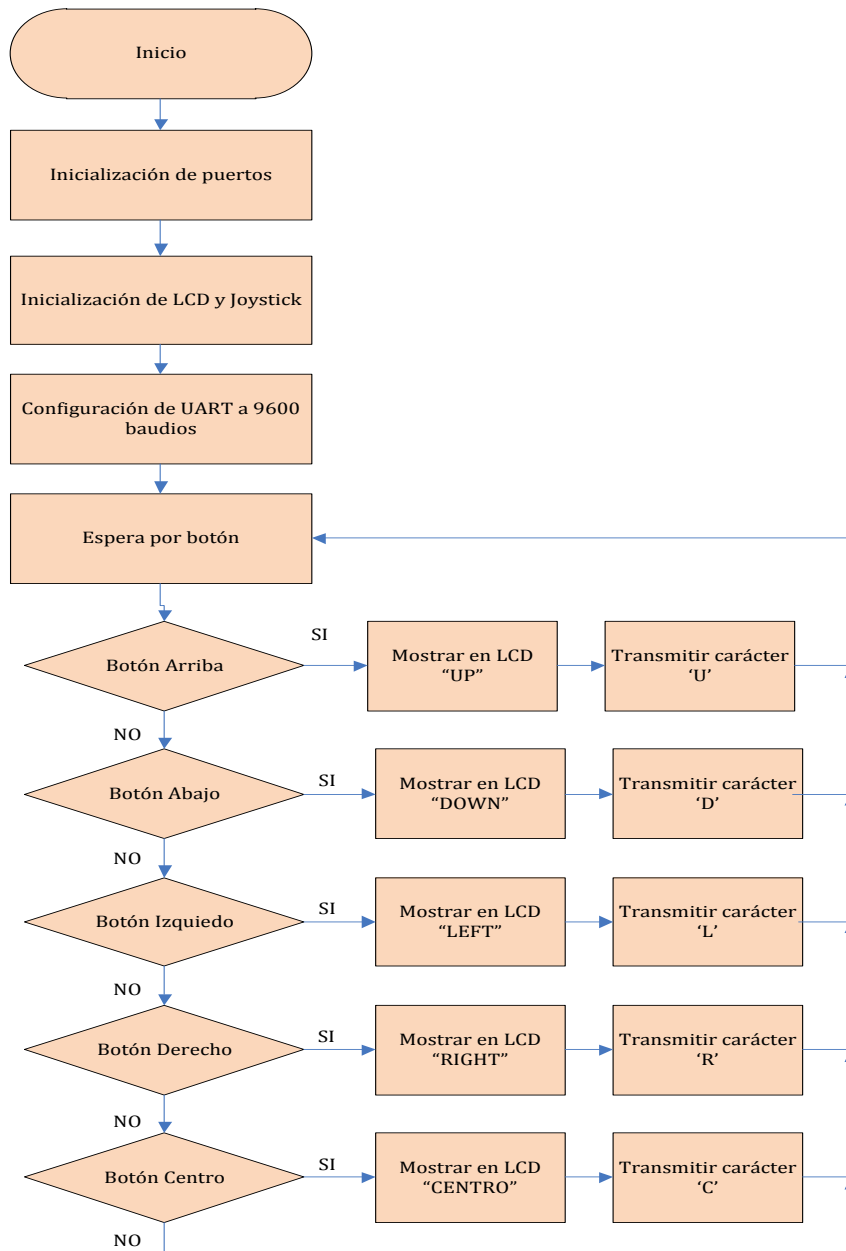


Figura 3.26 Diagrama de flujo del transmisor del proyecto

### 3.2.1.1.3 CÓDIGO DEL AVR BUTTERFLY

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <avr/delay.h>
#include <inttypes.h>
#include "mydefs.h"
#include "LCD_functions.h"
#include "LCD_driver.h"
#include "button.h"
#include "usart.h"

int main(void)
{
    PGM_P statetext = PSTR("ROSERO-CASTELO");
    uint8_t input;
    char *cadena;
    int i;

    // Disable Analog Comparator (power save)
    ACSR = (1<<ACD);
    // Disable Digital input on PF0-2 (power save)
```

```
DIDR0 = (7<<ADC0D);  
  
// Enable pullups  
PORTB = (15<<PB0);  
PORTE = (15<<PE4);  
  
Button_Init();      // Initialize pin change interrupt on joystick  
LCD_Init();         // initialize the LCD  
  
CLKPR = (1<<CLKPCE);    // set Clock Prescaler Change Enable  
  
// set prescaler = 8, Inter RC 8Mhz / 8 = 1Mhz  
CLKPR = (0<<CLKPS1) | (1<<CLKPS0);  
USART_Init(25.04);  
  
//sei();  
  
while (1)  
{  
    if (statetext){  
        LCD_puts_f(statetext, 1);  
        LCD_Colon(0);  
        statetext = NULL;  
    }  
  
    input = getkey();    // Read buttons  
    switch (input) {
```

```
case KEY_ENTER:  
    statetext = PSTR("CENTRO");  
    Usart_Tx('C');  
    break;
```

```
case KEY_NEXT:  
    statetext = PSTR("RIGHT");  
    Usart_Tx('R');  
break;
```

```
case KEY_PREV:  
    statetext = PSTR("LEFT"  
    Usart_Tx('L');  
    break;
```

```
case KEY_PLUS;  
    statetext = PSTR("UP");  
    Usart_Tx('U');  
    break;
```

```
case KEY_MINUS:  
    statetext = PSTR("DOWN");  
    Usart_Tx('D');  
    break;
```



```
        default:  
            break;  
    }  
}  
return 0;  
}
```

### **3.2.1.2 RECEPTOR**

#### **3.2.1.2.1 LPCXpresso**

Para la recepción de los caracteres hemos programado la tarjeta haciendo uso del registro Uart3 el cual usa el pin 9 para emisión Tx y el pin 10 para recepción Rx con los cuales nos ayudaremos para realizar la comunicación con la tarjeta AVR Butterfly.

En la programación primero inicializamos los puertos del registro GPIO2 y los configuramos como salidas, también hacemos uso del registro FIOSET para configurar los pines 43, 44, 45, 46 y 47 del GPIO2 de forma inicial como salidas en alto y luego por medio de la comunicación uart la tarjeta LPCXpresso esta lista para recibir los datos provenientes del transmisor los cuales servirán de condiciones para en lo posterior configurar los mismos puertos del GPIO2 como salidas en bajo para el manejo del motor según sea la instrucción enviada por el joystick de la tarjeta AVR Butterfly.

La recepción de los caracteres se la realiza asignando un puntero a la variable Uart3Buffer, la cual es un arreglo que almacena los caracteres que van llegando al receptor, de esta manera en vez de comparar si se ha recibido el caracter literalmente ('U', 'D', 'R', 'l', 'C') se compara el contenido que posee el puntero en el momento que recibe un carácter, así reconocemos que carácter llegó y realizamos una asignación como lo mostramos en la siguiente tabla.

<b>CARACTER</b>	<b>CONTENIDO DEL PUNTERO</b>	<b>ASIGNACIÓN</b>
'U'	0X55	Asigna un bajo en el pin 43
'D'	0x42	Asigna un bajo en el pin 44
'R'	0x52	Asigna un bajo en el pin 45
'L'	0x4c	Asigna un bajo en el pin 46
'C'	0x43	Asigna un bajo en el pin 47

Tabla 3.5. Contenido del puntero \*UART3BUFFER

De esta manera enviamos bajos a través de los pines indicados en la Tabla 3.5 anterior ya que el control del motor se encuentra realizado en lógica negativa.

### 3.2.1.2.2 DIAGRAMA DE FLUJO DEL RECEPTOR

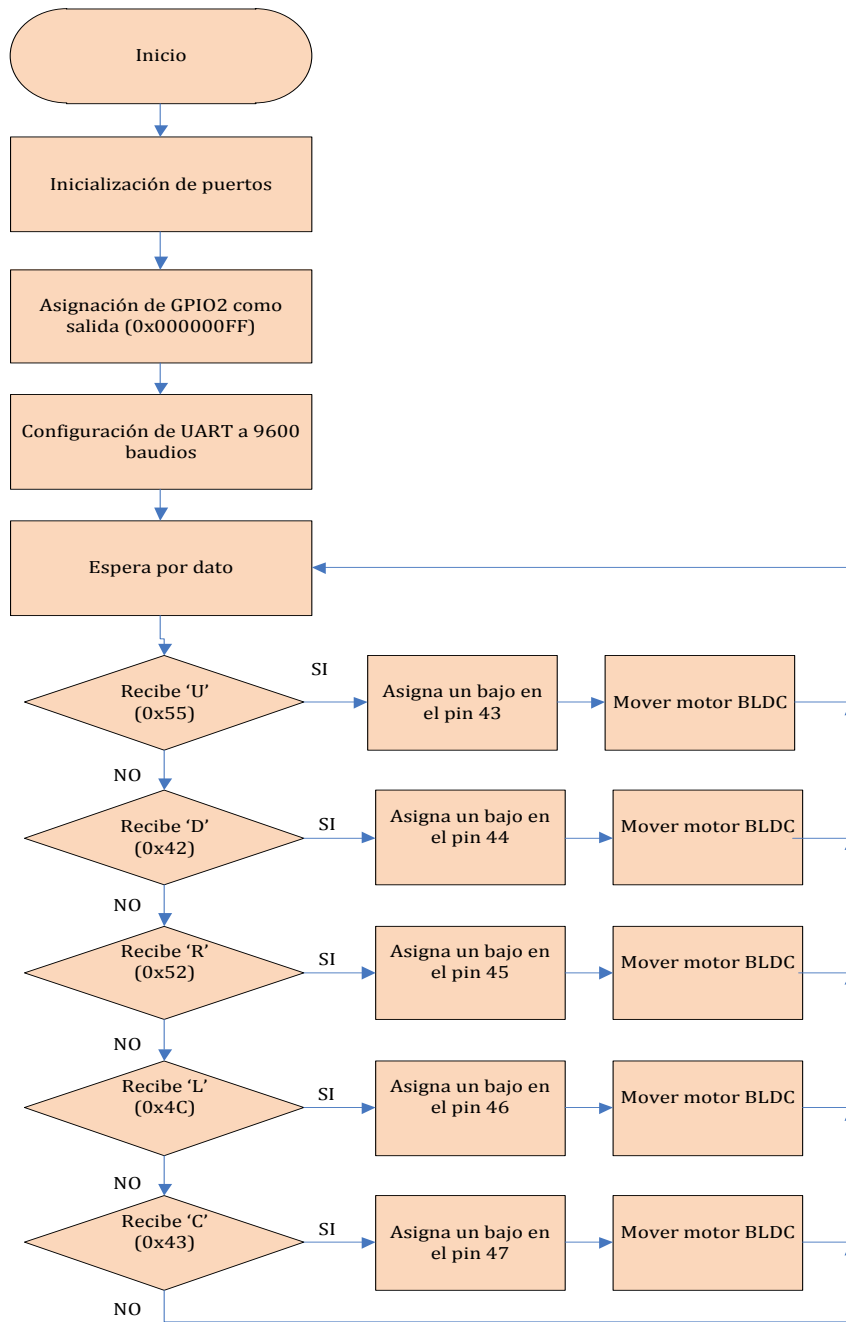


Figura 3.27. Diagrama de flujo del receptor del proyecto

### 3.2.1.2.3 CÓDIGO DE LA TARJETA LPCXpresso

```

#include "LPC17xx.h"

#include "type.h"

#include "uart.h"

#include <string.h>

extern volatile uint32_t UART3Count;

extern volatile uint8_t UART3Buffer[BUFSIZE];

/*****Main Function
main()

This program has been test on LPCXpresso 1700.

*****/

int main (void)
{
    uint32_t i, j;

    LPC_GPIO2->FIODIR = 0xFFFFFFFF; /* P2.xx defined as Outputs */
    LPC_GPIO2->FIOCLR = 0xFFFFFFFF;
    LPC_GPIO2->FIOSET = 0x000000FF;

    UARTInit(3, 9600);                /* baud rate setting */

    /* Loop forever */
    while (1)
    {

```

```
if ( UART3Count != 0 )
{
LPC_UART3->IER = IER_THRE | IER_RLS; /* Disable RBR */
if(*UART3Buffer==0x55)
{
LPC_GPIO2->FIOSET = 0x000000FF;
LPC_GPIO2->FIOCLR = 1 << 1;
}
if(*UART3Buffer==0x44)
{
LPC_GPIO2->FIOSET = 0x000000FF;
LPC_GPIO2->FIOCLR = 1 << 2;
}
if(*UART3Buffer==0x52)
{
LPC_GPIO2->FIOSET = 0x000000FF;
LPC_GPIO2->FIOCLR = 1 << 3;
}
if(*UART3Buffer==0x4c)
{
LPC_GPIO2->FIOSET = 0x000000FF;
LPC_GPIO2->FIOCLR = 1 << 4;
}
}
```

```
        if(*UART3Buffer==0x43)
        {
            LPC_GPIO2->FIOSET = 0x000000FF;
        }

        UART3Count = 0;

        LPC_UART3->IER = IER_THRE | IER_RLS | IER_RBR;    /* Re-enable
        RBR */

        }

    }

}
```

### 3.2.2 IMÁGENES DEL PROYECTO

En esta imagen observamos la tarjeta LPCXpresso Motor Control Board y la Tarjeta LPCXpresso 1114, la cual será la encargada de controlar los pulsos de PWM para el funcionamiento del motor BLDC.

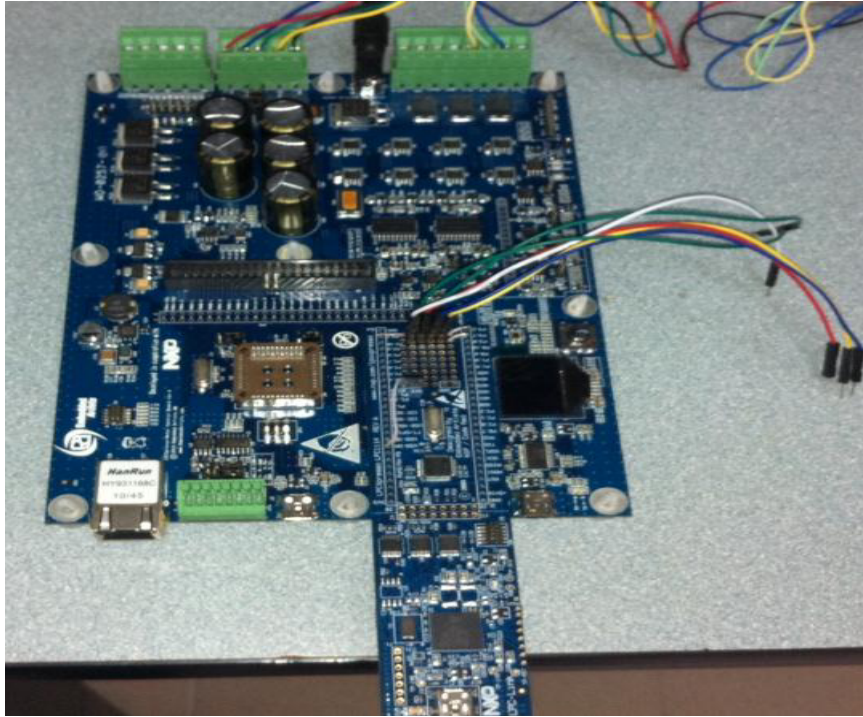


Figura 3.28. Tarjeta LPCXpresso Motor Control Board y LPCXpresso 1114.

Aquí observamos al Motor BLDC que será usado para realizar las pruebas de nuestro proyecto.



Figura 3.29. Motor BLDC

En esta imagen tenemos nuestro proyecto completo formado por la tarjeta LPCXpresso Motor Control Board, la tarjeta LPCXpresso1769 y la AVR Butterfly.

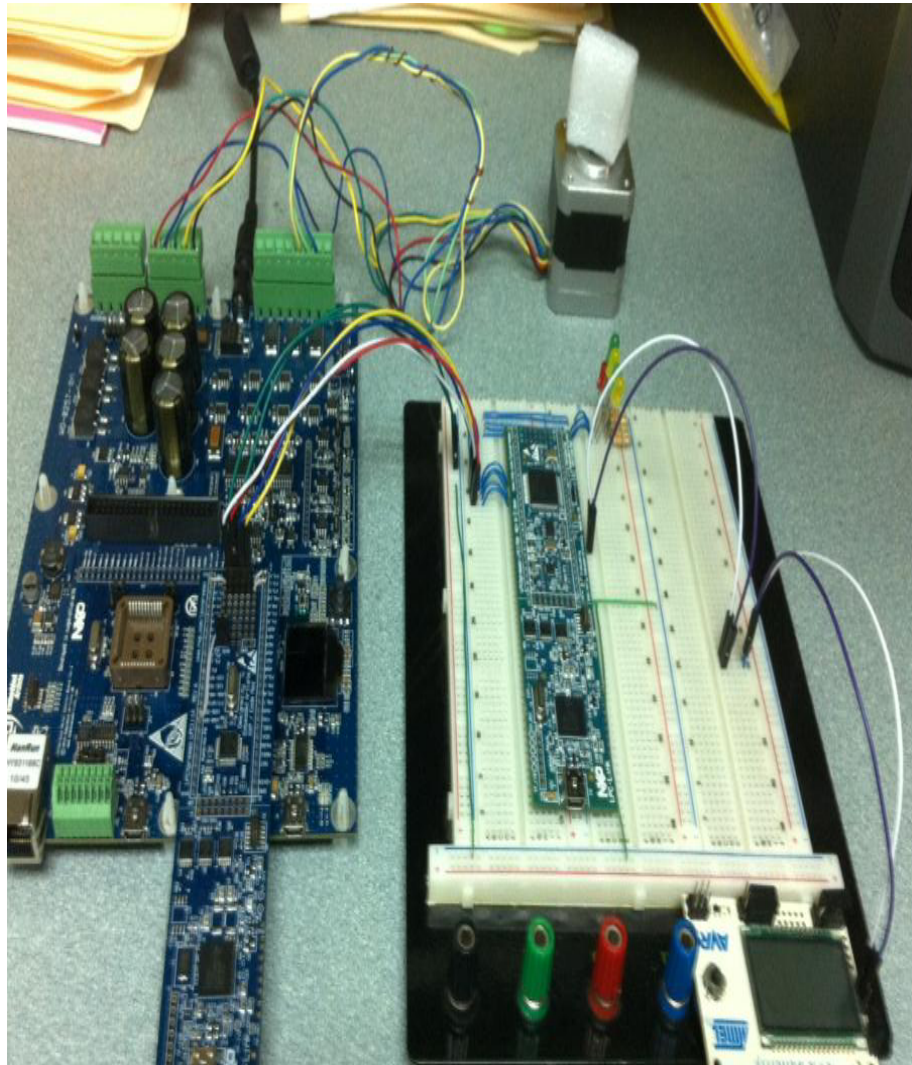


Figura 3.30. Proyecto completo.



Imagen del proyecto completo, enviando la señal "UP" desde la tarjeta AVR Butterfly, la cual servirá para aumentar la velocidad del motor BLDC.

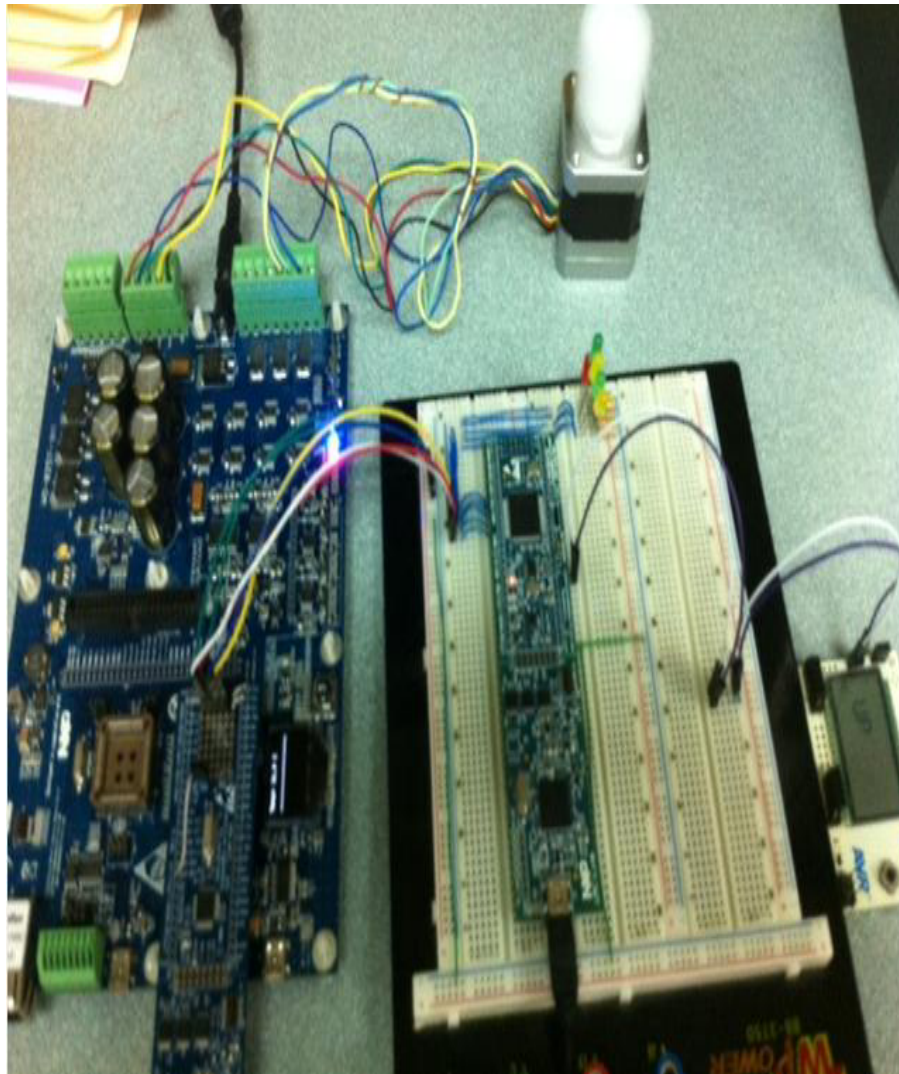


Figura 3.31. Motor BLDC en movimiento al presionar UP.

Imagen del proyecto completo, enviando la señal "LEFT" desde la tarjeta AVR Butterfly, la cual sirve para el encendido y el apagado del motor BLDC.

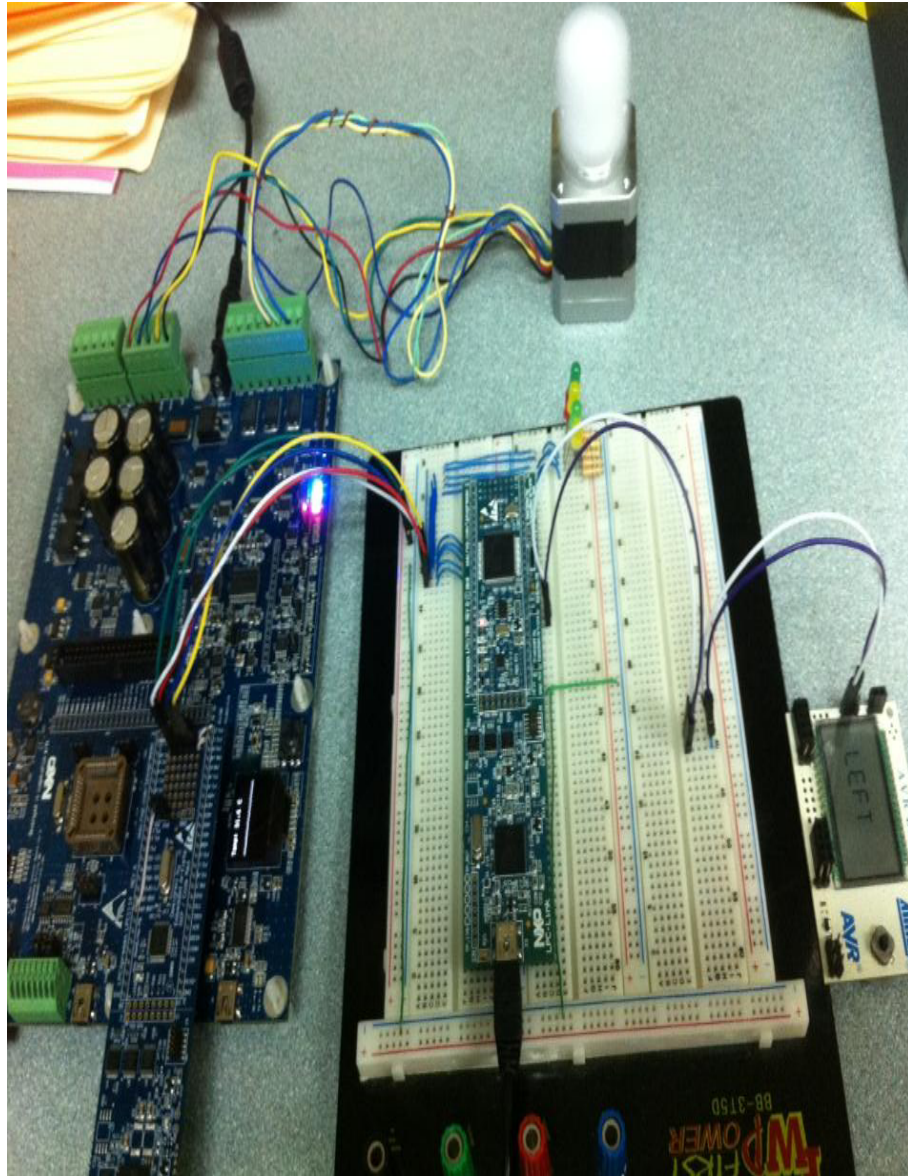


Figura 3.32. Motor BLDC en movimiento al presionar LEFT.

Imagen del proyecto completo, enviando la señal "RIGHT" desde la tarjeta AVR Butterfly, la cual servirá para cambiar la dirección de giro del motor BLDC.



Figura 3.33. Motor BLDC en movimiento al presionar RIGHT.

Imagen del proyecto completo, enviando la señal "DOWN" desde la tarjeta AVR Butterfly, la cual servirá para disminuir la velocidad del motor BLDC.

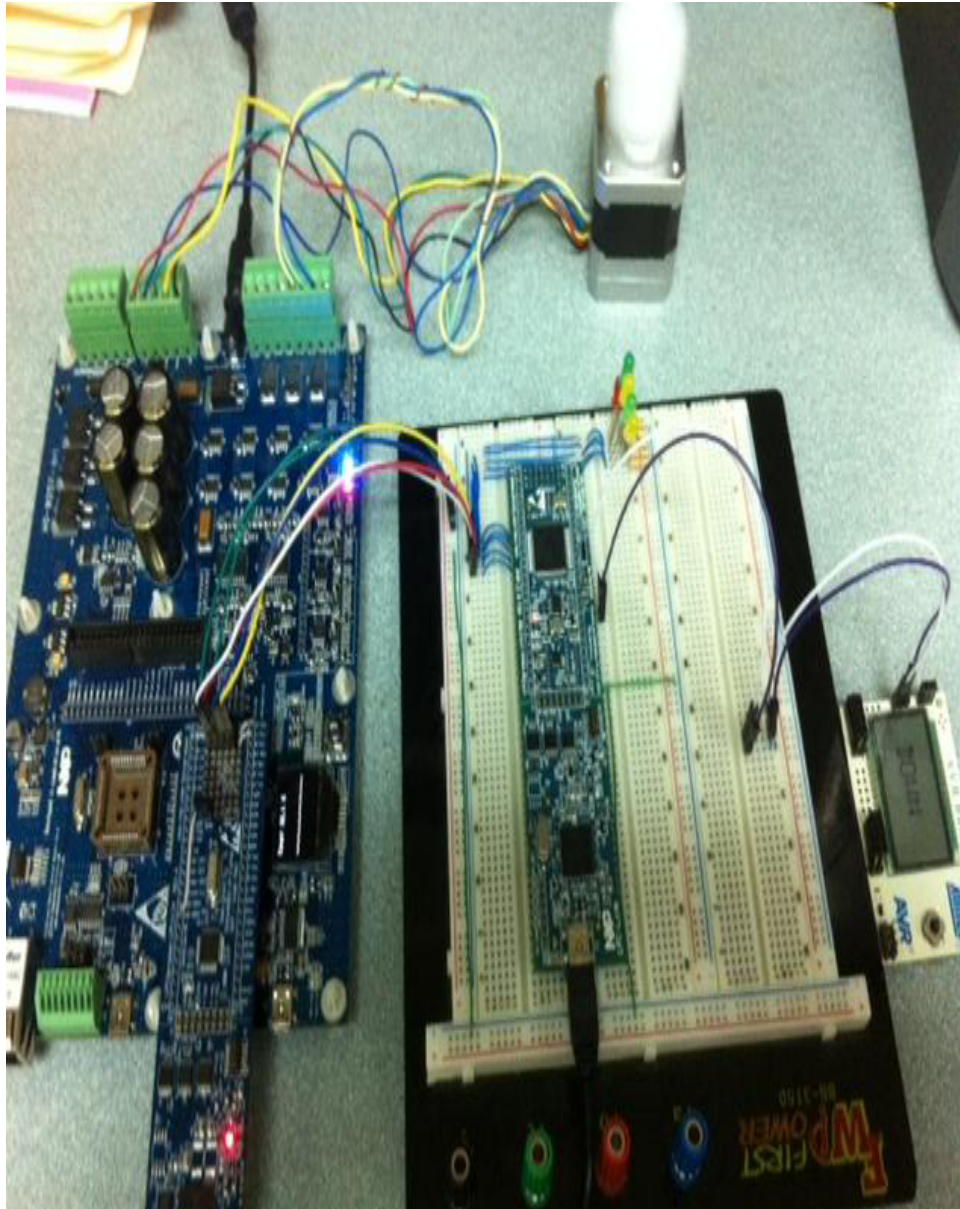


Figura 3.34. Motor BLDC en movimiento al presionar DOWN.

# **CAPÍTULO 4**

## **4 PRUEBAS Y SIMULACIONES**

### **4.0 RESUMEN DEL CAPÍTULO**

En este capítulo se presentan las simulaciones correspondientes del transmisor y del receptor que conforman el proyecto que se desarrolló, a su vez se presentan pruebas que fueron visualizadas en el osciloscopio virtual para la optimización del proyecto.

### **4.1 SIMULACIÓN DEL TRANSMISOR**

El transmisor está constituido por el ATMEGA169, el LCD y el PUSHBUTTON que representan el joystick del KIT AVR BUTTERFLY.

Al encender la AVR Butterfly en el LCD se muestra el mensaje “ROSERO-CASTELO”, luego se puede ejecutar las instrucciones respectivas de movimiento en el joystick el cual también se pueden visualizar en el LCD.

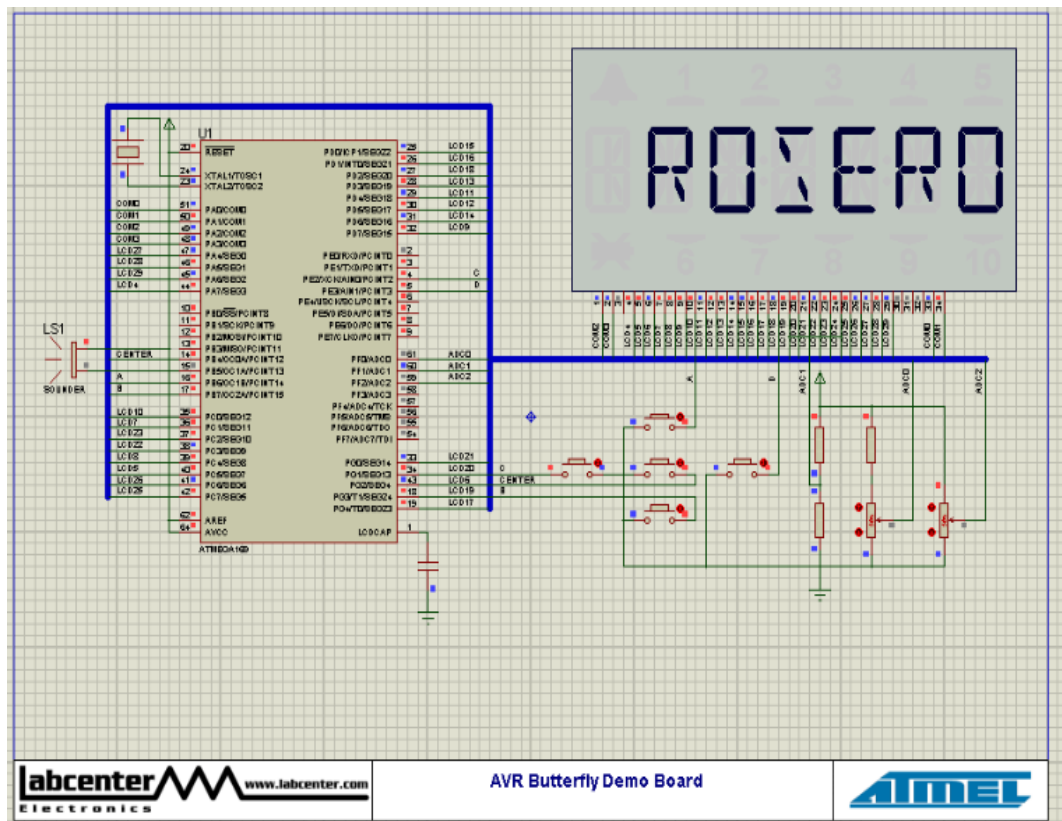


Figura 4.1. Mensaje de inicio en el LCD de la AVR Butterfly (ROSERO).

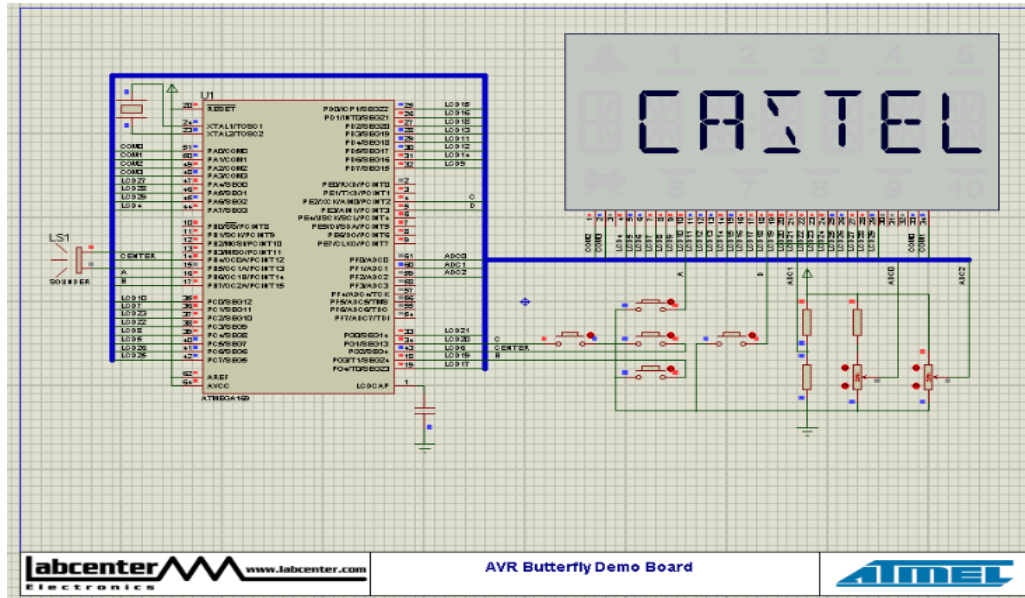


Figura 4.2. Mensaje de inicio en el LCD de la AVR Butterfly (CASTELO).

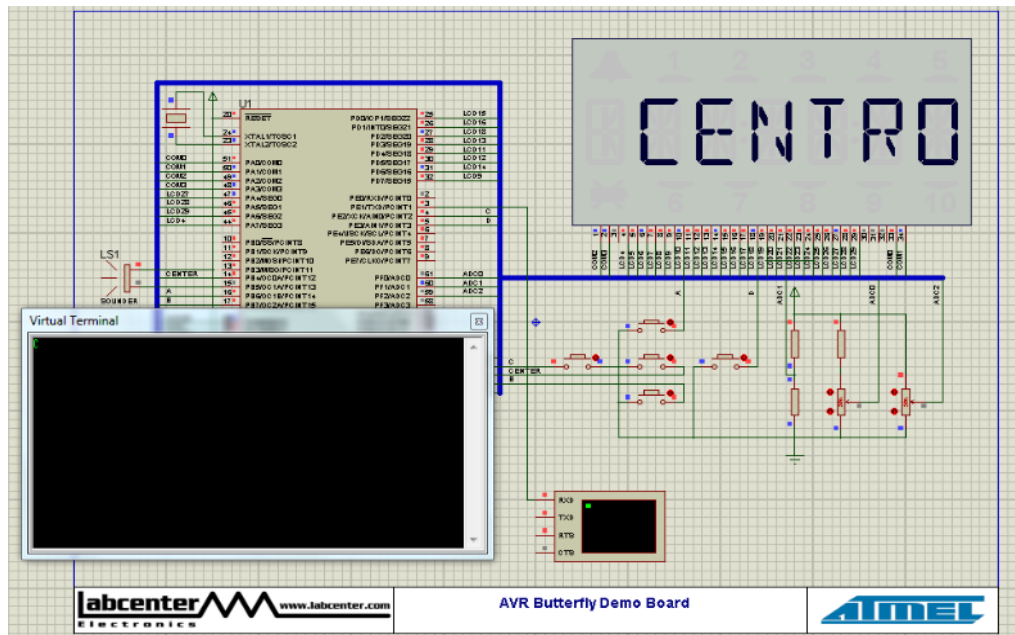


Figura 4.3. Transmisor mostrando la instrucción "CENTRO" en LCD.

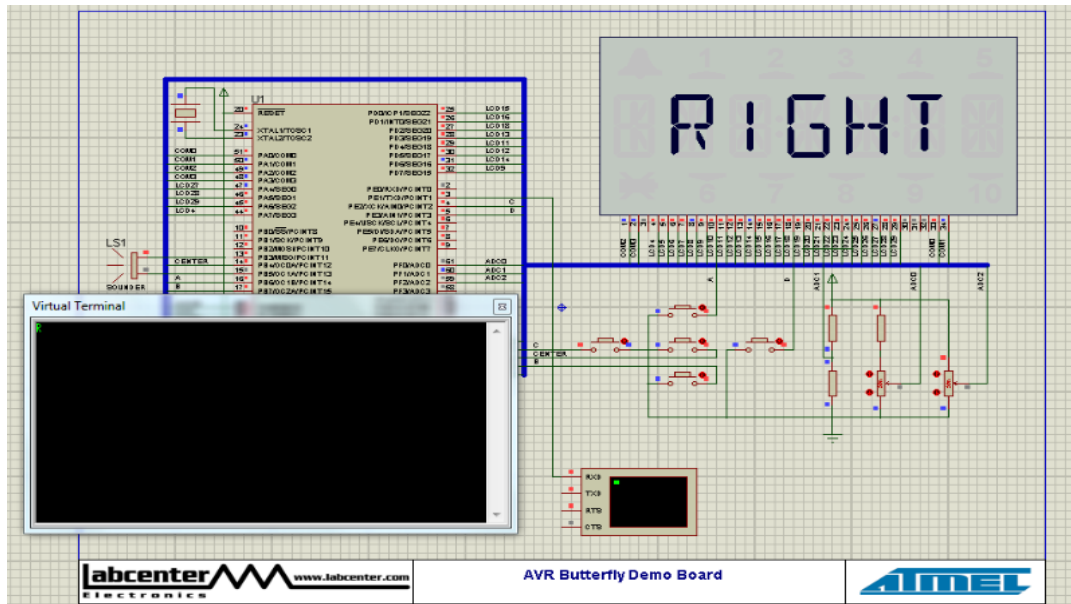


Figura 4.4. Transmisor mostrando la instrucción “RIGHT” en LCD.

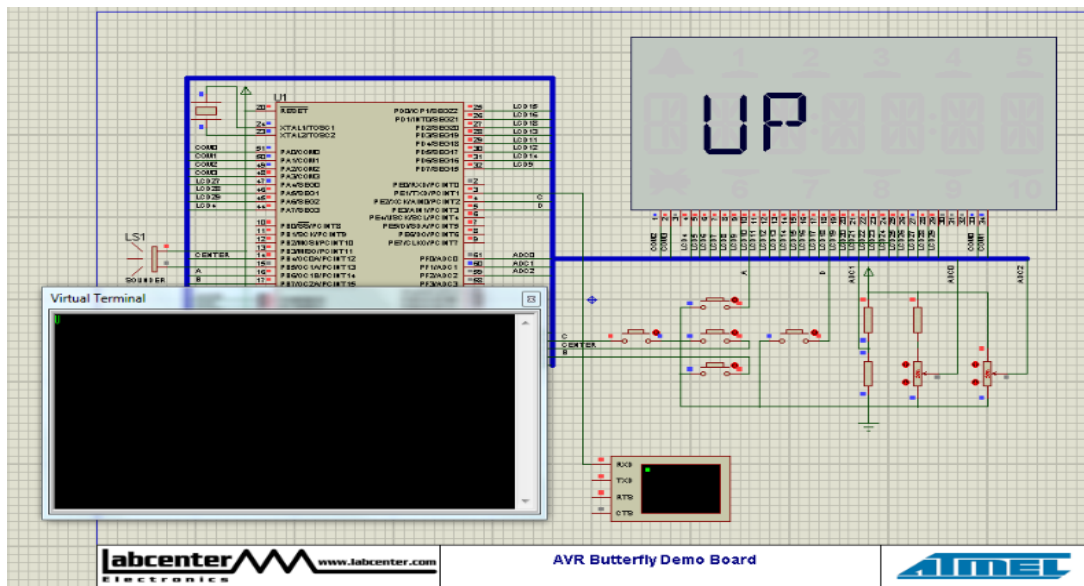


Figura 4.5. Transmisor mostrando la instrucción “UP” en LCD.



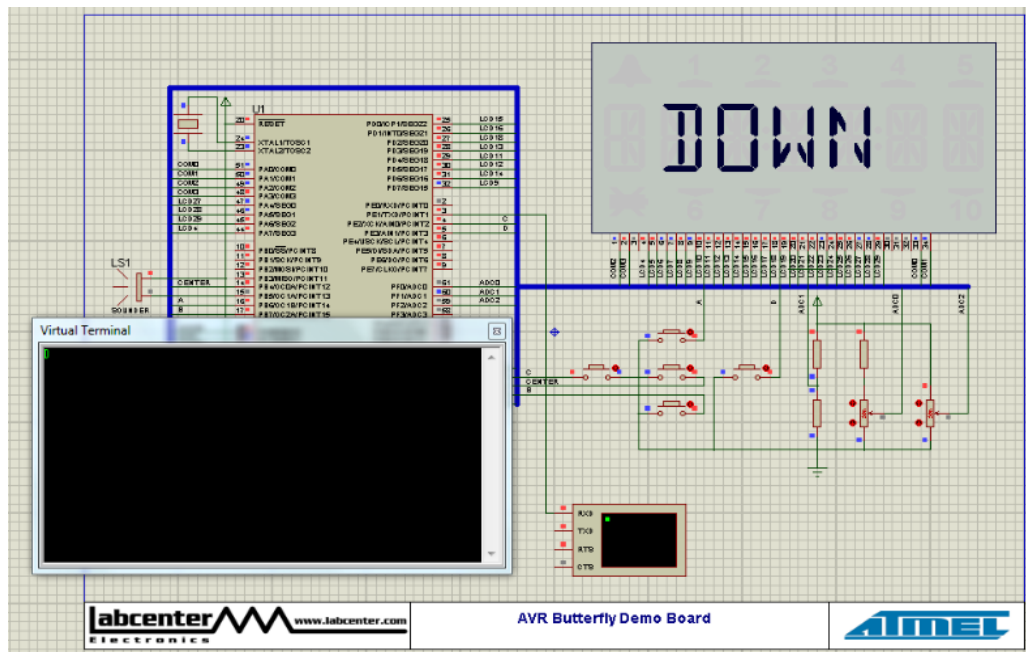


Figura 4.6. Transmisor mostrando la instrucción “DOWN” en LCD.

Para observar que caracter fue enviado, se usó la herramienta de Proteus Virtual Terminal y se configura a la velocidad de transmisión de datos.

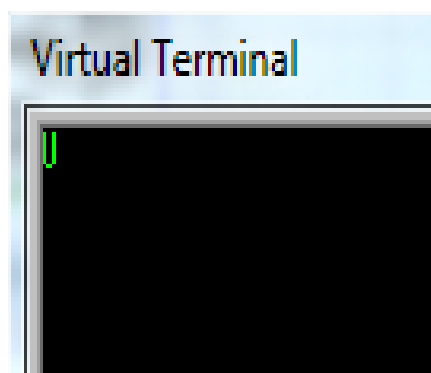


Figura 4.7. Visualización del caracter de transmisión “UP”

## 4.2 SIMULACIÓN DEL RECEPTOR

En el receptor no se pudo realizar las simulaciones debido a que ningún simulador presenta una tarjeta virtual LPCXpresso 1769, pero con el uso de algunos ejemplos se pudo comprobar el correcto funcionamiento para la elaboración del proyecto.

## 4.3 PRUEBAS DEL PROYECTO

También se puede simular los movimientos del joystick utilizando un osciloscopio virtual donde se observa los pulsos que son enviados al realizar la transmisión.

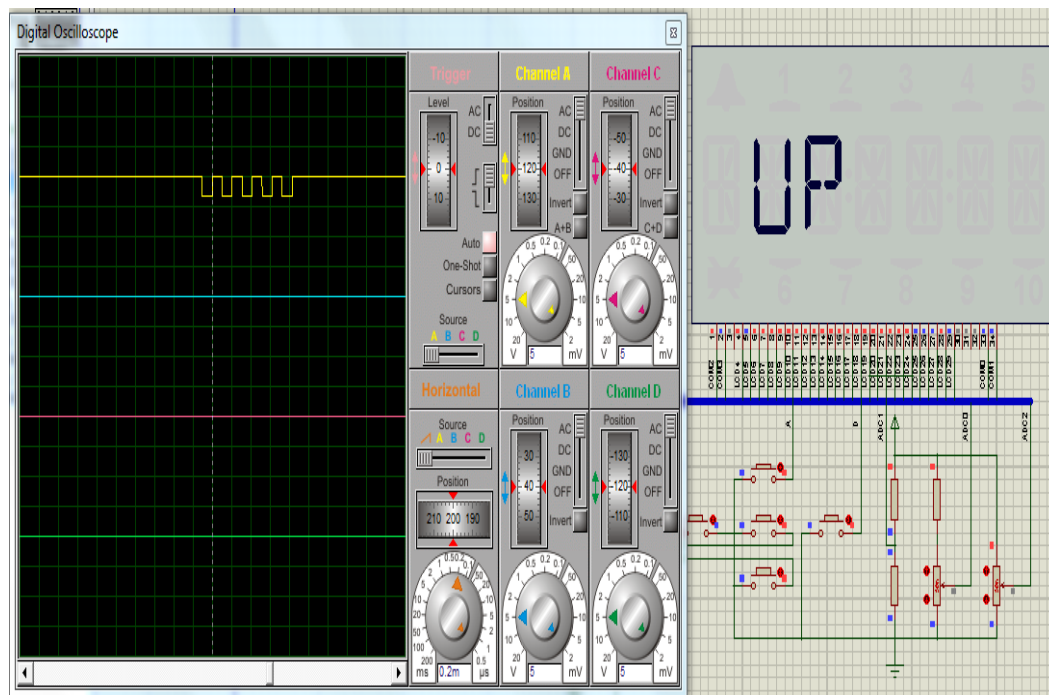


Figura 4.8. Instrucción UP mostrada en el osciloscopio virtual.

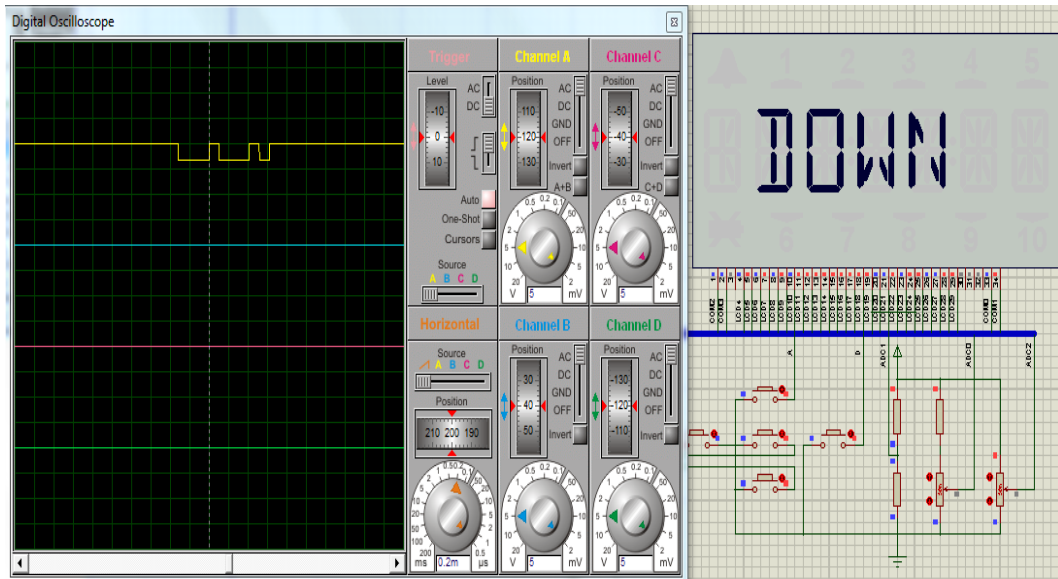


Figura 4.9. Instrucción DOWN mostrada en el osciloscopio virtual.

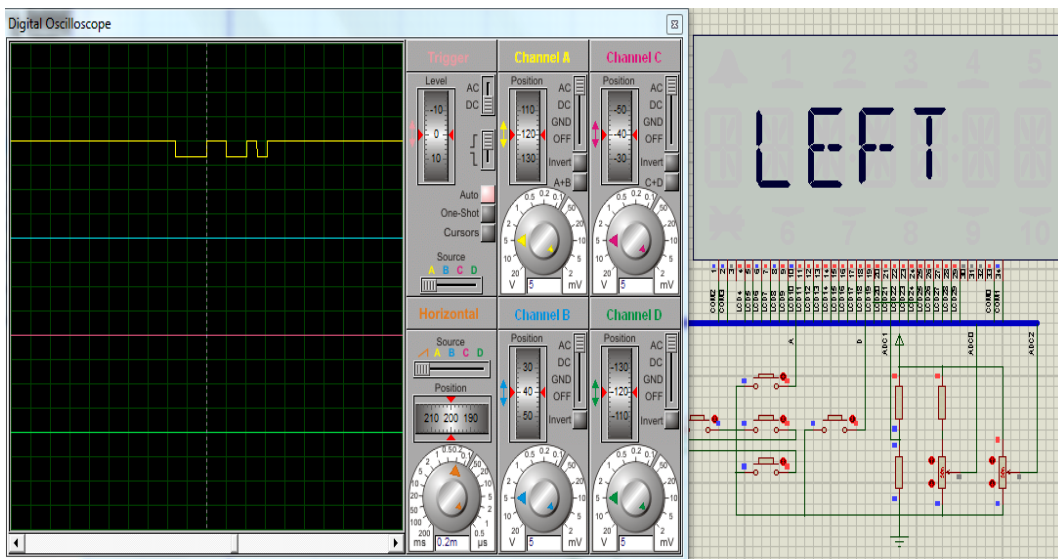


Figura 4.10. Instrucción LEFT mostrada en el osciloscopio virtual.

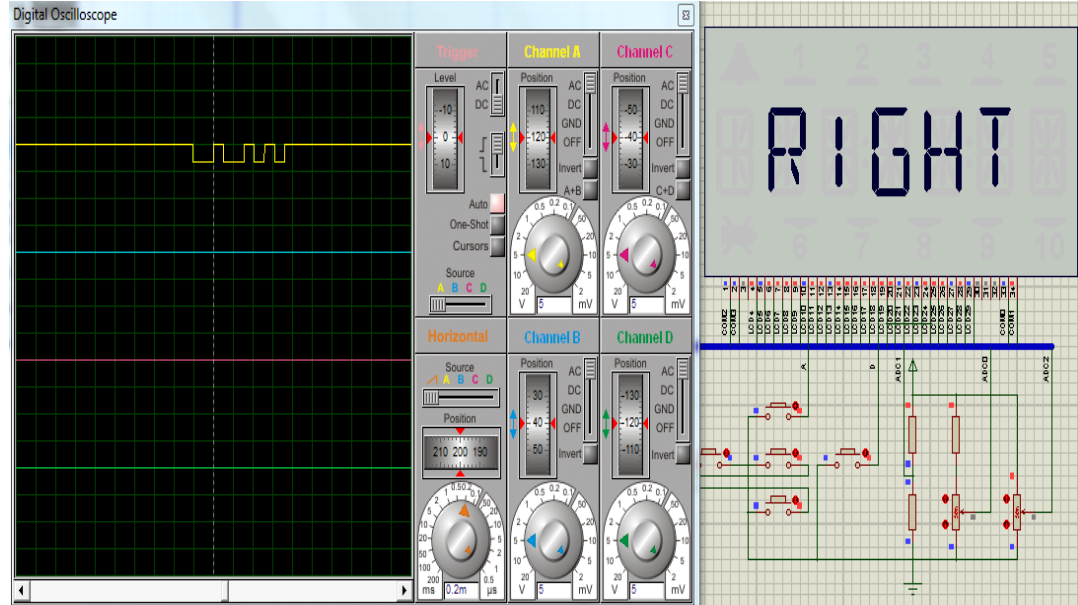


Figura 4.11. Instrucción RIGNT mostrada en el osciloscopio virtual.

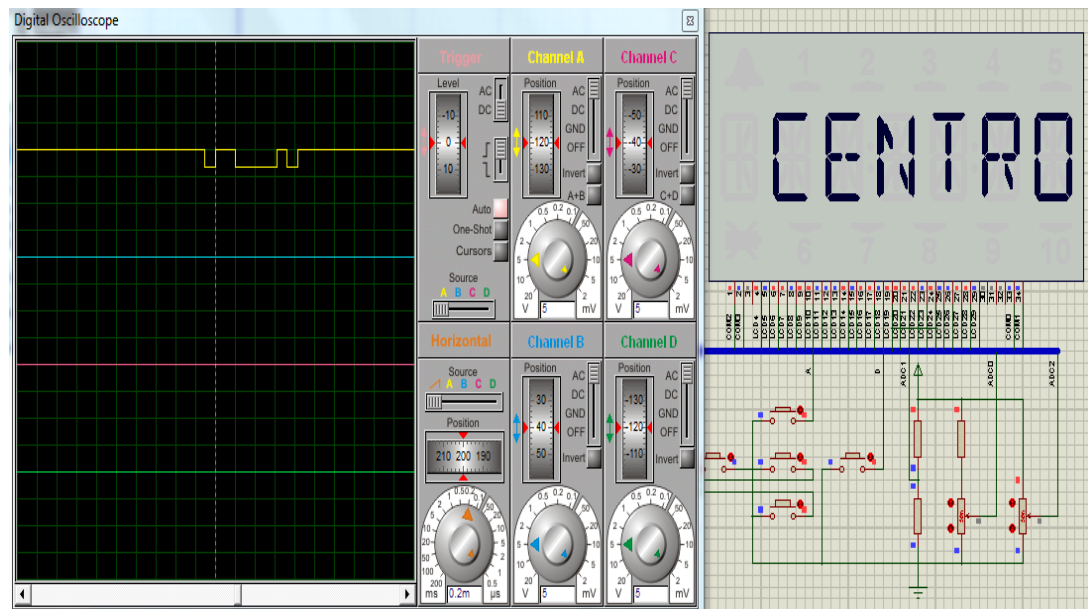


Figura 4.12. Instrucción CENTRO mostrada en el osciloscopio virtual.

#### **4.4 RESULTADOS DE LA SIMULACIÓN**

De los datos de simulación se pudo obtener una perspectiva del funcionamiento del proyecto, debido a que en cada una de las simulaciones se obtuvo el dato correspondiente desde el mensaje que se muestra en el LCD indicando que botón había sido presionado y el correspondiente carácter que se podía apreciar en el Virtual Terminal.

# CONCLUSIONES

La comunicación RS232 es muy importante debido a su característica principal de ser full dúplex lo cual es una ventaja en su construcción física, además es muy confiable debido a que la pérdida de datos es prácticamente nula y tiene la capacidad de ejecutarse en diferentes tarjetas con microcontroladores que poseen otros registros pero que logran comunicarse entre sí sin ningún problema y con alta fidelidad.

Verificamos la importancia que tienen las Tarjetas que incluyen Microcontroladores ya que gracias a ellos podemos realizar muchos circuitos aplicables y de gran utilidad en muchos ámbitos, en este caso logramos realizar un circuito en el cual se controla un motor BLDC usando

dos tarjetas con microcontroladores diferentes que se pudieron comunicar entre sí.

El uso de software para la programación de los microcontroladores nos proporciona gran utilidad debido a que podemos usar tarjetas como la AVR Butterfly que es programada en el software AVR Studio y debe ser elaborada en lenguaje assembler lo cual es complicado de realizar, pero existe un software adicional como el WinAVR que nos facilitará usar códigos en distintos lenguajes como C.

La tarjeta de desarrollo AVR Butterfly a pesar de ser de tamaño diminuto nos brinda muchas aplicaciones al usar sus puertos de comunicación ya sea de entrada o de salida, que nos permite operar en conjunto con otras tarjetas de desarrollo y que además nos puede mostrar datos de forma interactiva a través de su pantalla LCD, tal y como lo realizado en nuestro proyecto.

El uso de simuladores como Proteus, brinda la ventaja de poder realizar pruebas sin correr el riesgo de producir daños en las tarjetas reales y nos da la oportunidad de hacer cambios instantáneos de programación para corregir posibles errores que pudiéramos ocasionar.

# RECOMENDACIONES

En la programación del transmisor es necesario crear un proyecto en el cual se debe hacer uso de librerías ya establecidas para el manejo del joystick, de la pantalla LCD y también de su puerto de comunicación uart, ya que estos componentes de la tarjeta AVR Butterfly son indispensables para la construcción de nuestro proyecto.

Es importante verificar la forma de onda a la salida del puerto uart de la tarjeta AVR Butterfly para tener en cuenta que señal estamos enviando según sea el botón que presionemos en el joystick de la tarjeta AVR Butterfly puesto que eso nos proporciona una gran ayuda para verificar en el receptor que tipo de datos están llegando lo cual es bueno para saber si la comunicación es defectuosa o exitosa.



Tomar en cuenta que si bien es cierto que la comunicación RS232 puede realizarse mediante un solo cable ya que hemos configurado una interfaz asincrónica de canal simplex, es necesario contar con el cable de conexión a tierra entre la tarjeta LPCXpresso y el kit de desarrollo AVR Butterfly para una correcta comunicación entre ellas.

Para recibir los datos en la tarjeta LPCXpresso debemos hacer uso de un puntero, en este caso el arreglo Uart3Buffer el cual estará apuntando a la primera posición, la cual debemos tomar su contenido para verificar el valor hexadecimal y realizar la comparación respectiva en las condicionales existentes en el receptor.

Para la comunicación RS232 entre la tarjeta LPCXpresso y la AVR Butterfly es necesario definir el mismo valor de baud rate en ambas tarjetas para realizar una comunicación segura, pero hay que destacar que en la tarjeta AVR Butterfly no existe una función que permita ingresar el valor del baud rate, sino que hay un registro UBRR (Uart Baud Rate Register) el cual consta de dos valores un UBRR alto y un UBRR bajo, para lo cual existe tablas de conversión para hallar los valores de UBRR según sea el valor de baud rate que deseamos configurar.

# BIBLIOGRAFÍA

[1] <http://www.cursomicros.com/avr/proteus/cargar-el-programa.html>

Autor: cursomicros.com

Fecha de consulta: 10/02/12.

Temas consultados : Programación de Microcontrolador Atmega 169 de la tarjeta AVR Butterfly en el Software AVR Studio 4.

[2] <http://ics.nxp.com/lpcxpresso/~LPC1769/>

Autor: NXP (Fabricante LPCXpresso)

Fecha de consulta: 15/03/12.

Temas consultados : Configuración, características principales, software de programación y aplicaciones de la tarjeta LPCXpresso 1769

[3] [http://www.atmel.com/dyn/products/tools\\_card.asp?tool\\_id=3146](http://www.atmel.com/dyn/products/tools_card.asp?tool_id=3146)

Autor : Atmel Corporation

Fecha de consulta: 17/01/12.

Temas consultados : Productos de Atmel, aplicaciones, herramientas y Software de programación AVR Studio.

[4] [www.atmel.com/products/AVR/butterfly](http://www.atmel.com/products/AVR/butterfly)

Autor: Atmel Corporation

Fecha de consulta: 15/02/12.

Temas consultados : AVR Butterfly Evaluation, KitUserGuide.

[5] [www.atmel.com/dyn/resources/prod\\_documents/doc2514.pdf](http://www.atmel.com/dyn/resources/prod_documents/doc2514.pdf)

Autor: Atmel Corporation

Fecha de consulta: 28/03/12.

Temas consultados : ATMEGA 169 datasheet

[6] [moon-20.googlecode.com/files/ATmega164p\\_guide.pdf](http://moon-20.googlecode.com/files/ATmega164p_guide.pdf)

Autor: Atmel Corporation (traducción Andrés Mejía y Fernando Morales  
Estudiantes de la Escuela politécnica nacional, facultad de Control)

Fecha de consulta: 15/03/12.

Temas consultados: Registros del microcontrolador Atmega 169

[7] <http://es.wikipedia.org/wiki/RS-232>

Autor: Wikipedia

Fecha de consulta: 18/02/12.

Temas consultados: Comunicación RS-232

[8] <http://es.scribd.com/doc/53655717/287/Figura-130-Diagrama-de-bloques-de-la-UART>

Autor: Carles Vilella

Fecha de consulta: 12/03/12.

Temas consultados : Sistemas digitales y comunicaciones

[9] <http://www.atmel.com/Images/io.pdf>

Autor: Atmel Corporation

Fecha de consulta: 17/03/12.

Temas consultados : Entradas y salidas e interrupciones básicas.

[10] [www.dspace.espol.edu.ec/bitstream/123456789/11698/1/Control de velocidad por cambio de frecuencia.pdf](http://www.dspace.espol.edu.ec/bitstream/123456789/11698/1/Control%20de%20velocidad%20por%20cambio%20de%20frecuencia.pdf)

Autor: Freddy Rosero, Jorge Espinoza (Tesis de Graduación)

Fecha de consulta: 05/02/12.

Temas consultados : Control de velocidad de un motor trifásico usando microcontroladores.

[11] [www.javeriana.edu.co/biblos/tesis/ingenieria/tesis89.pdf](http://www.javeriana.edu.co/biblos/tesis/ingenieria/tesis89.pdf)

Autor: Carlos Escobar, José Martínez, German Tellez

Fecha de consulta: 18/03/12.

Temas consultados : Control de un motor Brushless.

[12] [http://books.google.com.ec/books/about/Permanent\\_magnet\\_and\\_brushless\\_DC\\_motors.html?id=2PJSAAAAMAAJ&redir\\_esc=y](http://books.google.com.ec/books/about/Permanent_magnet_and_brushless_DC_motors.html?id=2PJSAAAAMAAJ&redir_esc=y)

Autor: Takashi Kenjō, Shigenobu Nagamori

Fecha de consulta: 19/03/12.

Temas consultados : Características y funcionamiento de los Motores brushless

[13] [http://laboratorios.fi.uba.ar/lse/seminario/material-011/Sistemas\\_Embebidos-2011\\_2doC-](http://laboratorios.fi.uba.ar/lse/seminario/material-011/Sistemas_Embebidos-2011_2doC-Intro_a_LPCXpresso_y_repaso_lenguaje_C-Kharsansky.pdf)

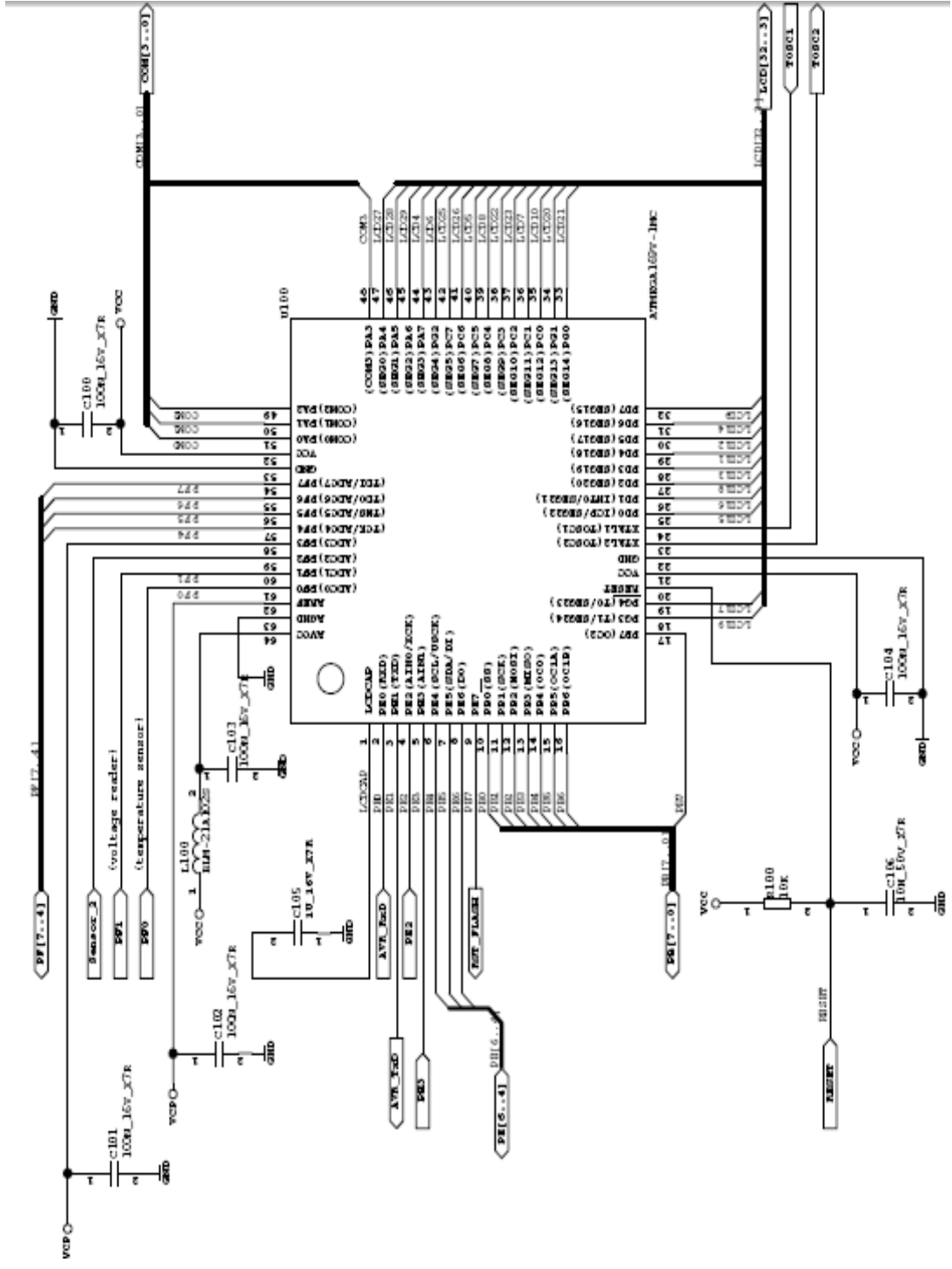
[Intro a LPCXpresso y repaso lenguaje C-Kharsansky.pdf](http://laboratorios.fi.uba.ar/lse/seminario/material-011/Sistemas_Embebidos-2011_2doC-Intro_a_LPCXpresso_y_repaso_lenguaje_C-Kharsansky.pdf)

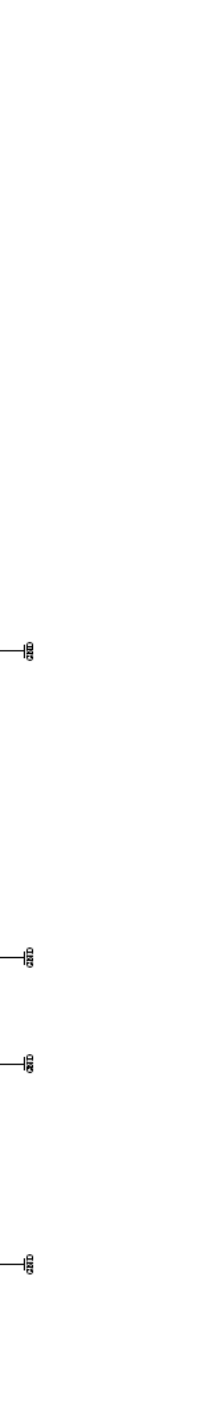
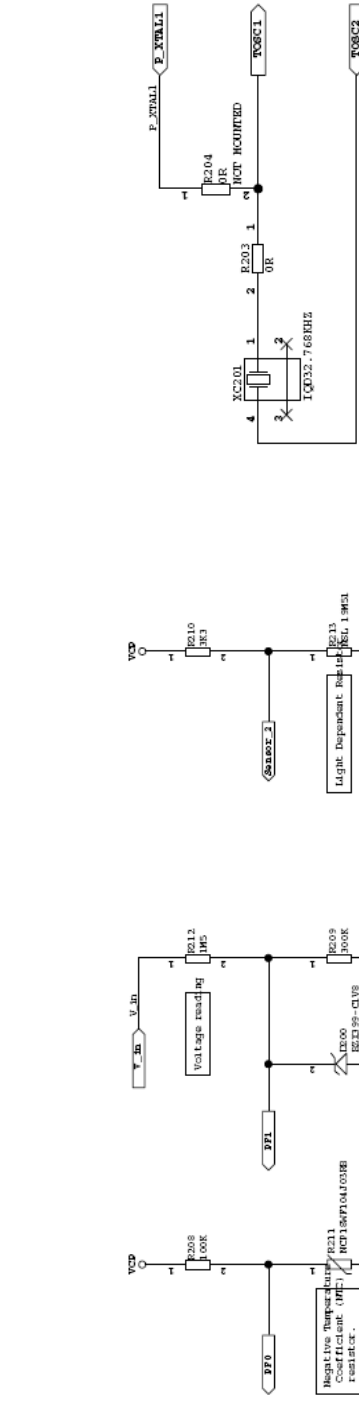
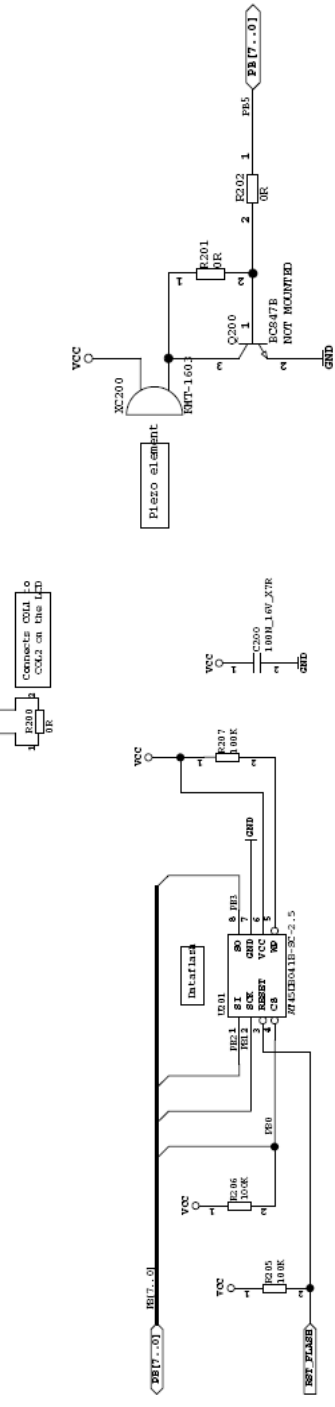
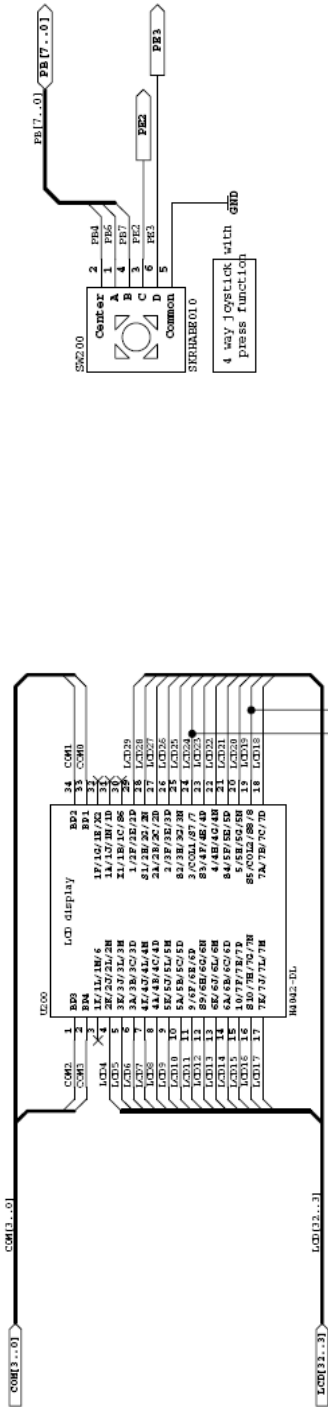
Autor: Alan Kharsansky

Fecha de consulta: 21/03/12.

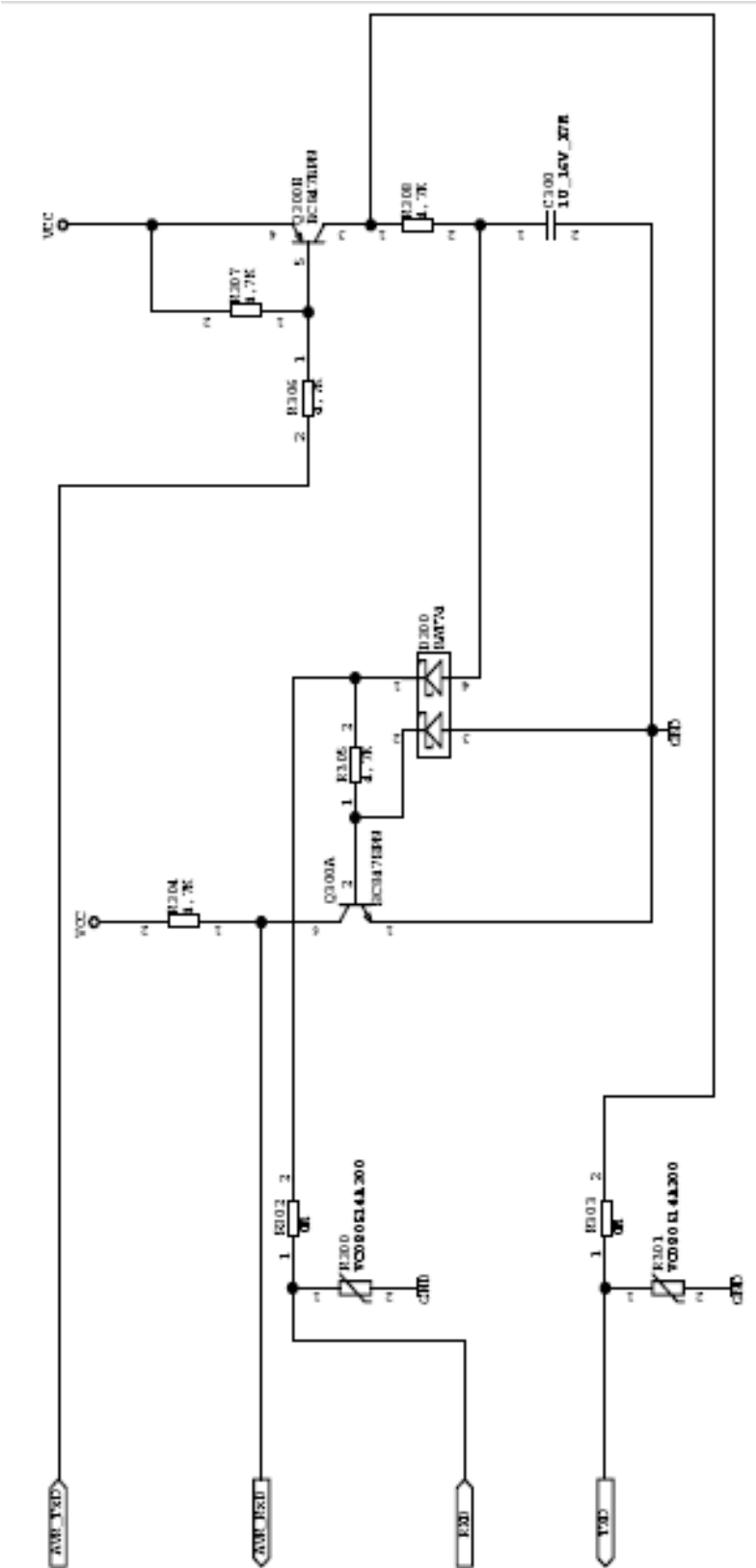
Temas consultados : Introducción a LPCXpresso y repaso de lenguaje C

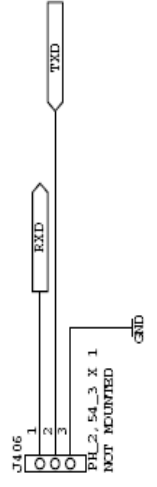
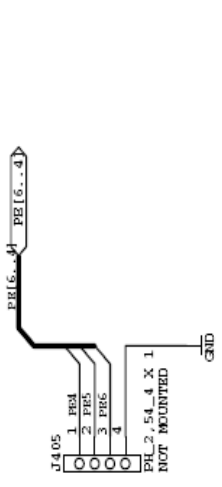
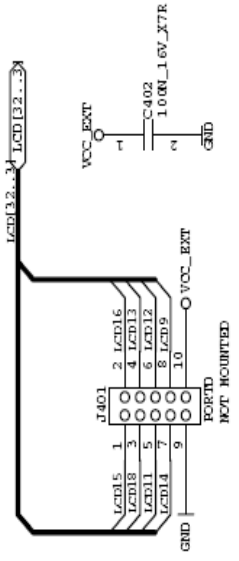
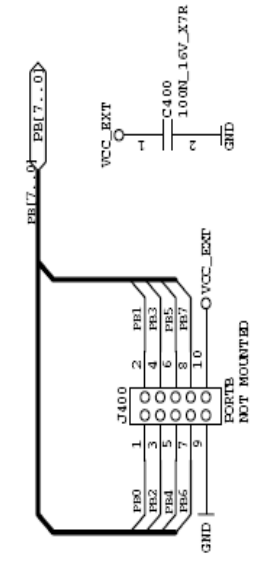
**ANEXO I**  
**DIAGRAMA ESQUEMÁTICO DEL MÓDULO**  
**AVR BUTTERFLY**





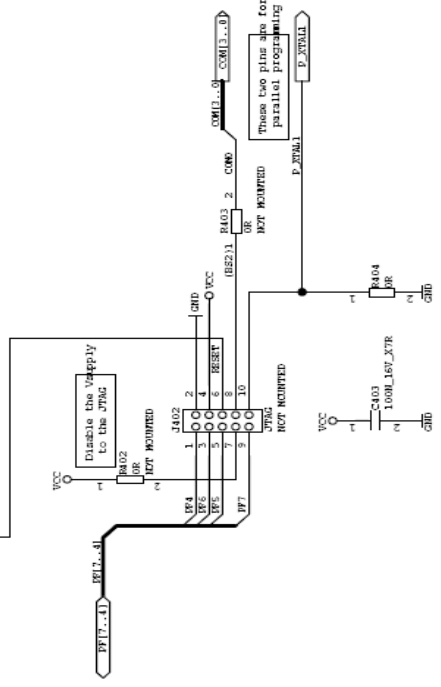
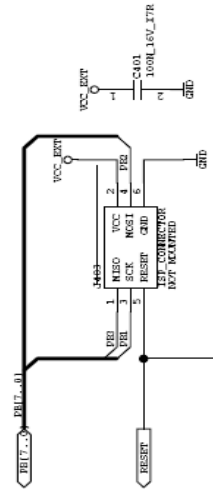
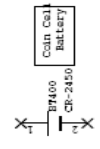
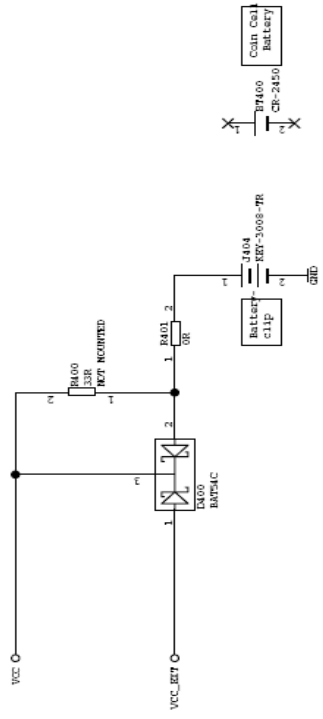






USI

UNFT P1D8



These two pins are for parallel programming

P\_PSW1

P\_PSW2

**ANEXO II**

**DIAGRAMA ESQUEMÁTICO DEL MICROCONTROLADOR**

**LPCXPRESSO 1769**



LPC-LINK side

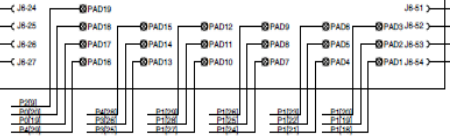

Expansion Connector  
(superset of mbed pinning)

mbed	LPCXpresso
GND	GND
VIN (4.5-14V)	VIN (4.5-5.5V)
VB (battery supply)	VB (battery supply)
nR (reset)	RESET_N
SPI-MOSI	PO.9 MOSI
SPI-MISO	PO.8 MISO
SPI-SCK	PO.7 SCK
GND	PO.6 SS/SL
UART1-TX / I2C1-SDA	PO.5 TXD0/SDA1
UART1-RX / I2C1-SCL	PO.1 RXD0/SCL1
SPI-MOSI	PO.18 MOSI
SPI-MISO	PO.17 MISO
SPI-SCK / UART2-TX	PO.15 TXD1/SCK0
UART2-RX	PO.16 RXD1/SS/SL0
AN0	PO.23 AD0.0
AN1	PO.24 AD0.1
AN2	PO.25 AD0.2
AN3 / ADUT	PO.26 AD0.3/ADUT
AN4	P1.30 AD0.4
AN5	P1.31 AD0.5
	PO.2
	PO.3
	PO.21
	PO.22
	PO.27
	PO.28
	PO.13

Dual row holes (2x27, 100 mil spacing)	
GND	J8-1
EXT_POWER	J8-2
VB	J8-3
RESET_N	J8-4
PO.9	J8-5
PO.8	J8-6
PO.7	J8-7
PO.6	J8-8
PO.5	J8-9
PO.1	J8-10
PO.18	J8-11
PO.17	J8-12
PO.15	J8-13
PO.16	J8-14
PO.23	J8-15
PO.24	J8-16
PO.25	J8-17
PO.26	J8-18
P1.30	J8-19
P1.31	J8-20
PO.2	J8-21
PO.3	J8-22
PO.21	J8-23
PO.22	J8-24
PO.27	J8-25
PO.28	J8-26
PO.13	J8-27

VCC_3V3V	J8-28
	J8-29
	J8-30
	J8-31
RD-	J8-32
RD+	J8-33
TD-	J8-34
TD+	J8-35
USB-D-	J8-36
USB-D+	J8-37
PO.4 CAN_RX0	J8-38
PO.5 CAN_TX0	J8-39
PO.10 TXD0/SDA0	J8-40
PO.11 RXD0/SCL0	J8-41
PWM1.1	J8-42
PWM1.2	J8-43
PWM1.3	J8-44
PWM1.4	J8-45
PWM1.5	J8-46
PWM1.6	J8-47
P2.8	J8-48
P2.7	J8-49
P2.8	J8-50
P2.10	J8-51
P2.11	J8-52
P2.12	J8-53
GND	J8-54

LPCXpresso	mbed
VOUT (3.3V out if full powered, else 3.3V input)	VOUT (3.3V out)
not used	VU (5.0V USB out)
not used	IF+
not used	IF-
RD-	RD- (Ethernet)
RD+	RD+ (Ethernet)
TD-	TD- (Ethernet)
TD+	TD+ (Ethernet)
USB-D-	D- (USB)
USB-D+	D+ (USB)
PO.4 CAN_RX0	CAN-RD
PO.5 CAN_TX0	CAN-TD
PO.10 TXD0/SDA0	UARTS-TX / I2C0-SDA
PO.11 RXD0/SCL0	UARTS-RX / I2C0-SCL
PWM1.1	PWMOUT0
PWM1.2	PWMOUT1
PWM1.3	PWMOUT2
PWM1.4	PWMOUT3
PWM1.5	PWMOUT4
PWM1.6	PWMOUT5
P2.8	
P2.7	
P2.8	
P2.10	
P2.11	
P2.12	
GND	

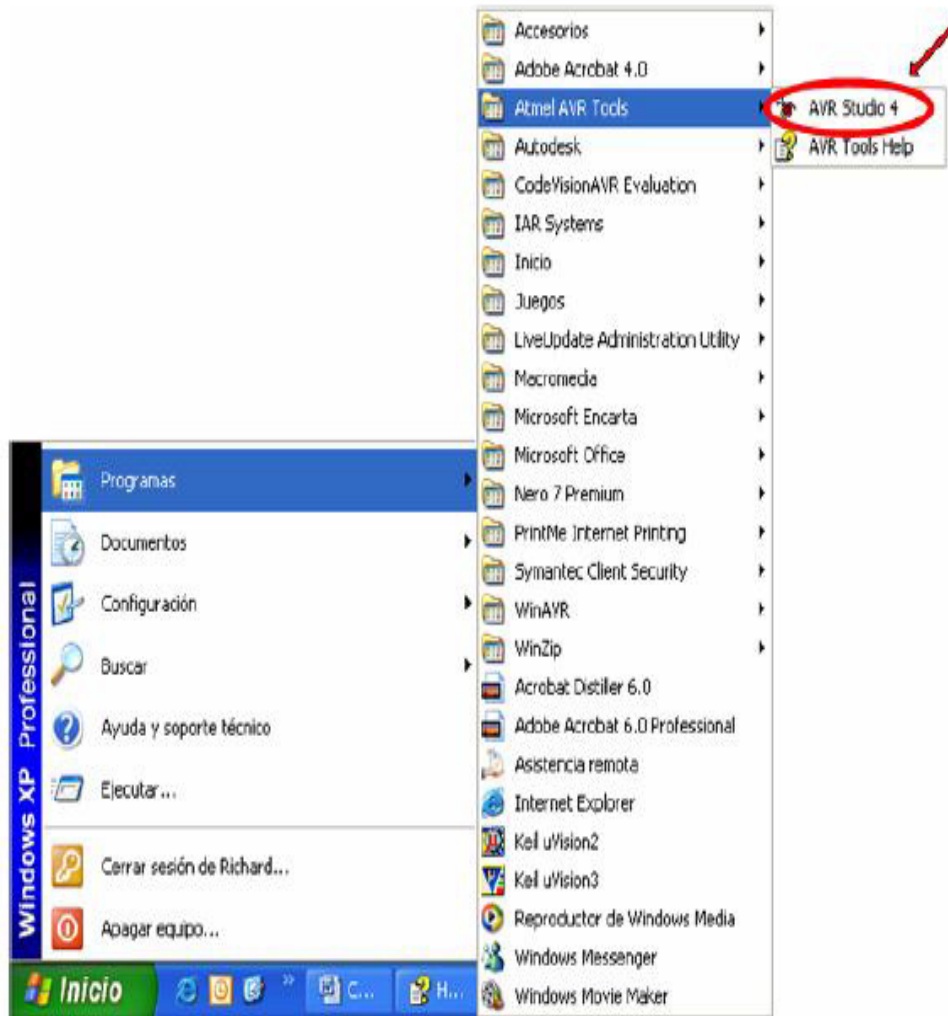
(C) Embedded Artists AB  
 TITLE: LPCXpresso LPC1114 rev 8  
 Document Number:  
 Date: 2010-10-19 14:19:36 Sheet: 7/7

## **ANEXO III**

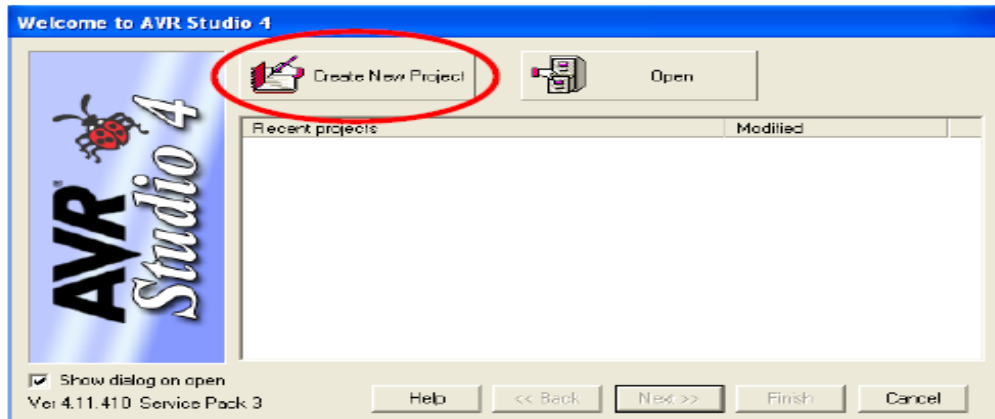
### **MANEJO DE SOFTWARE AVR STUDIO 4**

## GENERACIÓN DE PROYECTOS

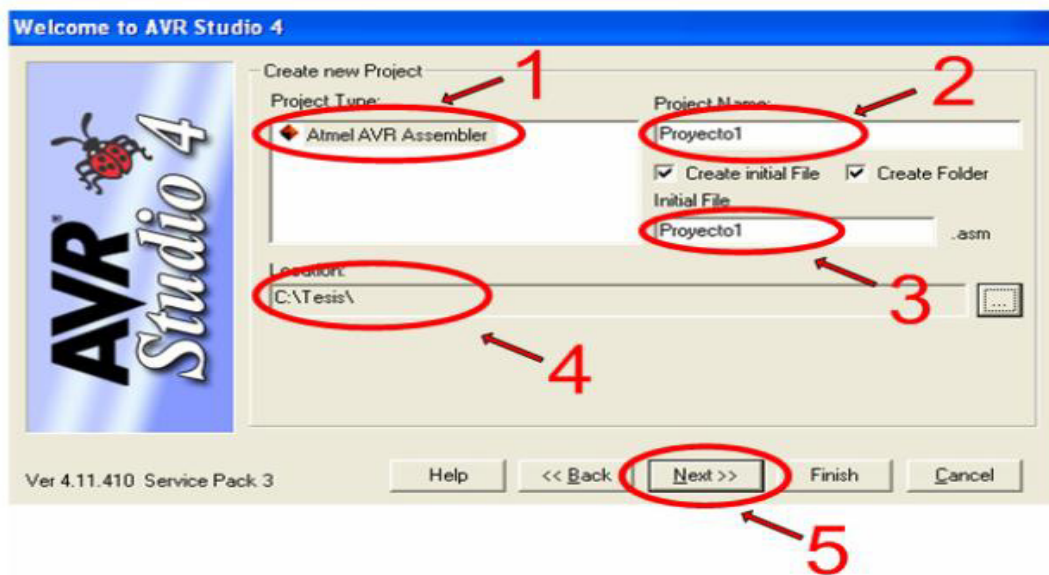
Se debe iniciar el AVR Studio 4 desde el menú [Inicio] [Programas] [Atmel AVR Tools], como se indica a continuación.



Después aparecerá la ventana para crear un nuevo proyecto.



El usuario debe configurar el tipo de proyecto que desea crear, el nombre del archivo y el lugar donde será guardado.



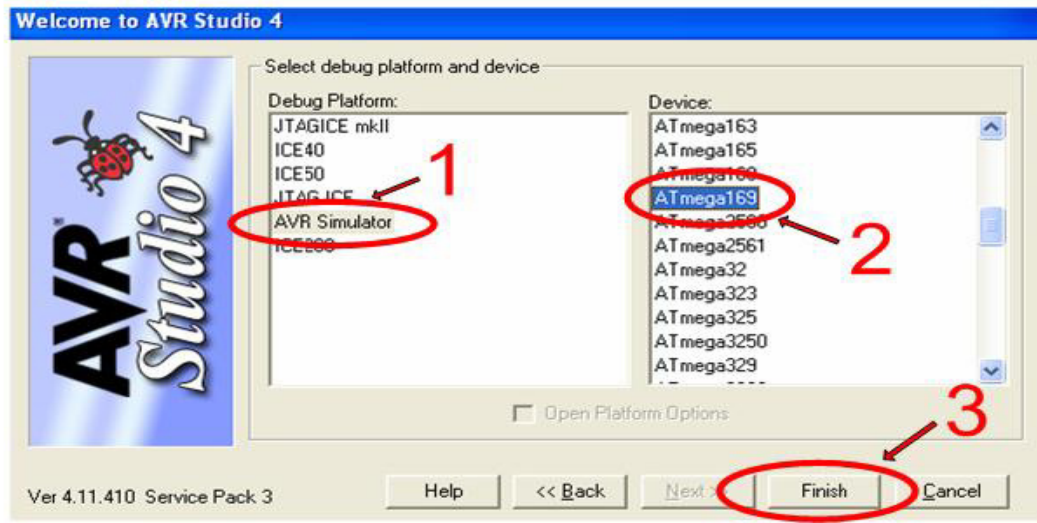
1. Todos los tipos de proyecto disponibles se listan en el cuadro Project type list. El usuario debe hacer clic en Atmel AVR Assembler para



dejar saber al AVR Studio que se quiere crear un programa en ensamblador.

2. Aquí se coloca el nombre del proyecto, puede ser cualquiera.
3. Aquí el usuario puede especificar si el AVR Studio debe crear automáticamente un archivo ensamblador. El nombre del archivo puede ser cualquiera.
4. Seleccionar la ruta donde se quiere guardar el proyecto y los archivos.
5. Verificar siempre una vez más y asegurarse de que las casillas de verificación estén seleccionadas. Una vez seguro, presionar el botón "Next>>".

Luego, siguiendo el orden en que aparecen las ventanas, el usuario debe escoger la plataforma de depuración y del Dispositivo para Simulación

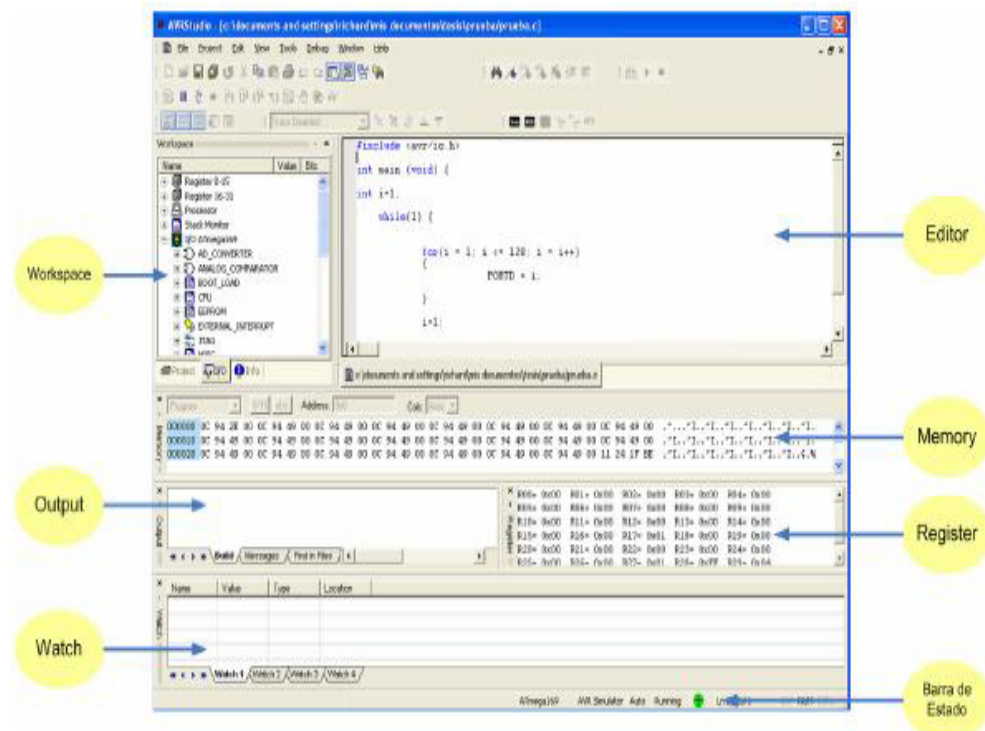


1. El AVR Studio 4 soporta un amplio rango de herramientas para emulación y depuración. Por conveniencia el usuario debe seleccionar la funcionalidad de simulación incluida, el AVR Simulator; ya que el resto de opciones requieren de H/W especial.
2. Aquí el usuario debe seleccionar el microcontrolador que utilizará en su aplicación, en nuestro caso ATmega169.
3. Verificar una vez más, luego presionar “Finish” para crear el proyecto e ir a editar archivo ensamblador (.asm). El AVR Studio iniciará y abrirá un archivo .asm.

## **DESCRIPCIÓN GENERAL DEL IDE**

Como se menciona anteriormente, el AVR Studio es un Entorno de Desarrollo Integrado (IDE). Éste tiene una arquitectura modular completamente nueva, que incluso permite interactuar con software de otros fabricantes.

AVR Studio 4 consiste de muchas ventanas y sub-módulos. Cada ventana apoya a las partes del trabajo que se intenta emprender, se puede observar las ventanas principales del IDE.

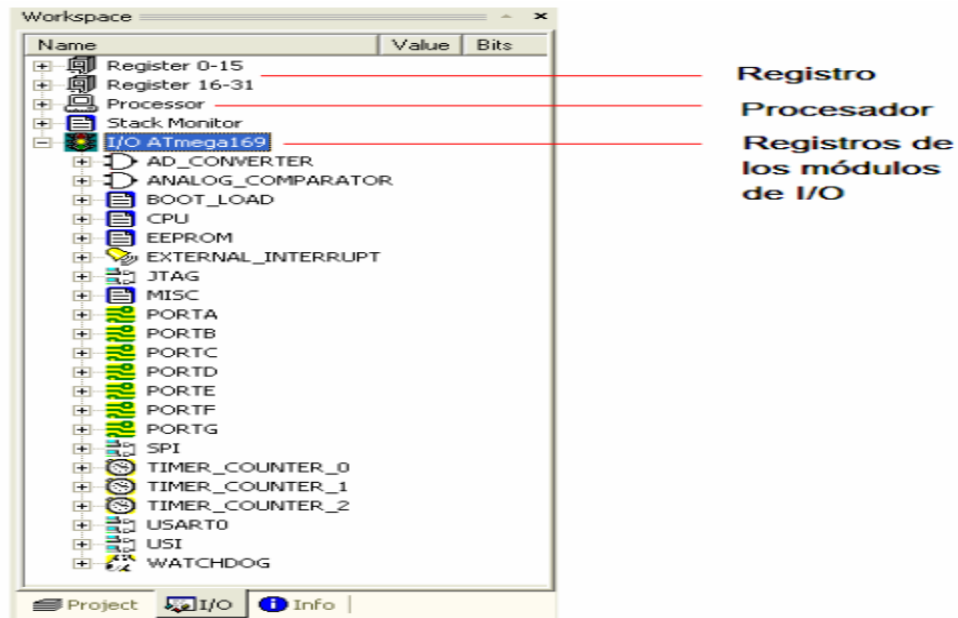


## Administración de Proyectos.

Toda elaboración de código dentro del AVR Studio se la realiza como proyectos de programación. Todos los proyectos de código tienen un archivo project que mantiene la información acerca del proyecto, de qué archivos consta, la estructuración del ensamblador, vistas personalizadas y así sucesivamente. El archivo project asegura que el proyecto sea el mismo cada vez que regrese a él, que todo esté correctamente organizado y que ensamble / compile.

## La ventana I/O

Esta ventana despliega información de los registros de I/O así como también del procesador y de los registros.



## DEPURACIÓN DE PROYECTOS

AVR Studio 4 puede orientarse hacia el Simulador AVR incorporado ó hacia un Emulador de Circuito AVR.

Independientemente de la plataforma que se esté ejecutado, el entorno de AVR Studio aparecerá idéntico. Al cambiar entre plataformas de depuración, todas las opciones del entorno se mantienen para la nueva plataforma.

Muchas plataformas tienen características únicas y consecuentemente aparecerán nuevas funcionalidades/ventanas.

### **Formato del Archivo Objeto**

Todas las sesiones de depuración necesitan que el usuario cargue un archivo objeto para depuración que sea soportado por el AVR Studio. Usualmente un archivo para depuración contiene información simbólica que no está incluida en un archivo simple.

La ventaja de esto es que, se puede generar código en otros Entornos de Desarrollo y con otros lenguajes por ejemplo C++. Pero la simulación se la puede hacer en AVR Studio.

Antes de depurar asegúrese de tener configurado su compilador /ensamblador para que genere un archivo para depuración con un formato de archivo objeto que son soportados por el AVR Studio.

Formato del Archivo Objeto	Archivo	Descripción
Intel hex Extendido	.hex	Este formato es producido habitualmente por la mayoría de los programas de desarrollo y su objetivo es liberar de comprobación. No incluye información adicional para depuración y por lo tanto no es un formato recomendado para la depuración. El archivo contiene únicamente datos para la memoria de programa. No están disponibles ejecución paso a paso del archivo fuente <b>ni tampoco watch simbólico</b> .
UBROF	.d90	UBROF es un formato de propiedad de IAR. El archivo resultante de la depuración contiene un conjunto completo de información de la depuración y símbolos para soportar todos los tipos de vigilantes. Soporta UBROF8 y las versiones anteriores. Este es el formato de salida predefinido del IAR EW 2.29 y sus versiones anteriores.
ELF/DWARF	.elf	La información para depuración del ELF/DWARF es un estándar abierto. El formato de depuración soporta un conjunto completo de información y símbolos para soportar todos los tipos de vigilantes. La versión de formato leído por el AVR Studio es DWARF2. Versiones AVR-GCC configuradas para obtener DWARF2 pueden generar este formato.
AVR COFF	.cof	COFF es un estándar abierto proyectado para fabricantes tercerizados creando herramientas que dan soporte con el AVR Studio. Con AVR Studio 4.09 el formato AVR COFF es extendido para soportar depuración total con ejecución paso a paso del archivo fuente y soporte completo de <b>watch</b> .
Formato del ensamblador AVR	.obj	El formato de archivo de salida AVR assembler contiene información del archivo fuente para ejecución paso a paso del archivo fuente y es únicamente un formato Atmel interno. El archivo .map es analizado para obtener un poco de información de <b>watch</b> .

## **ANEXO IV**

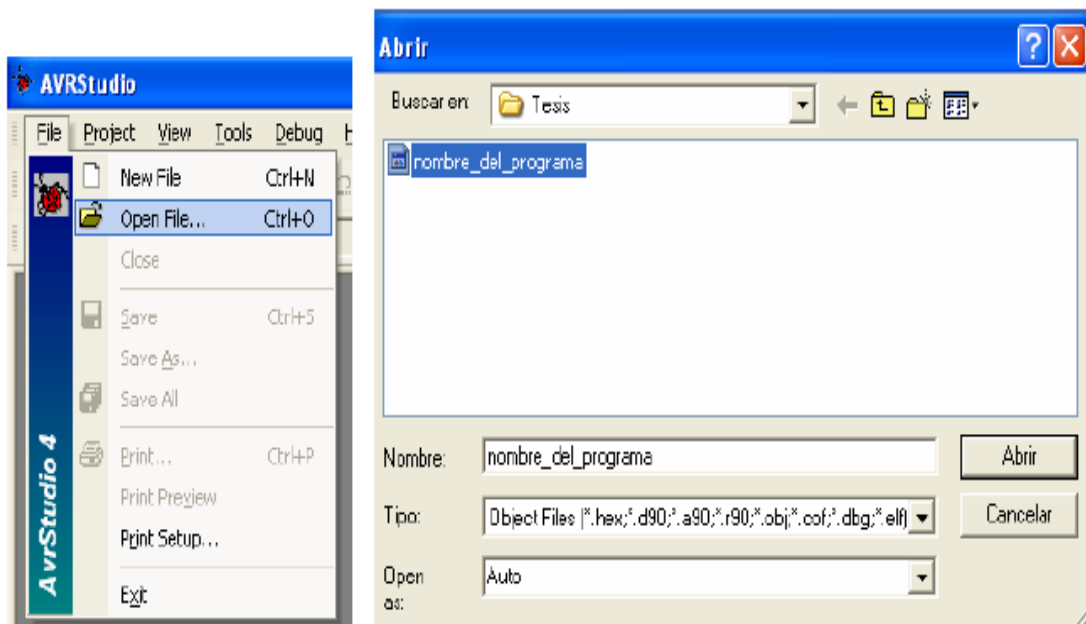
### **PROGRAMACIÓN DE LA AVR BUTTERFLY**



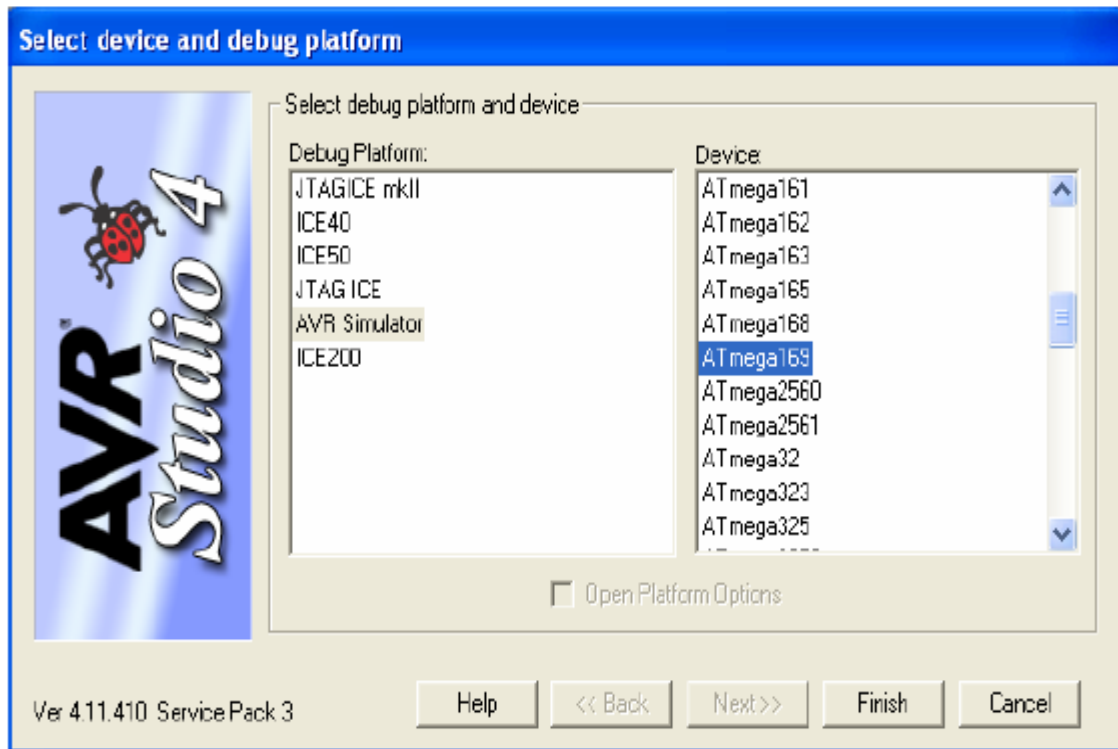
## PROGRAMACIÓN DE LA AVR BUTTERFLY

Los pasos fundamentales para la Programación del Kit AVR Butterfly son los siguientes:

- En la PC, localizar y ejecutar el AVR Studio.
- En el menú "File" del AVR Studio seleccionar "Open File", luego seleccionar el archivo con el que se desea programar al AVR Butterfly, por ejemplo: ...\`nombre_del_programa.cof`.

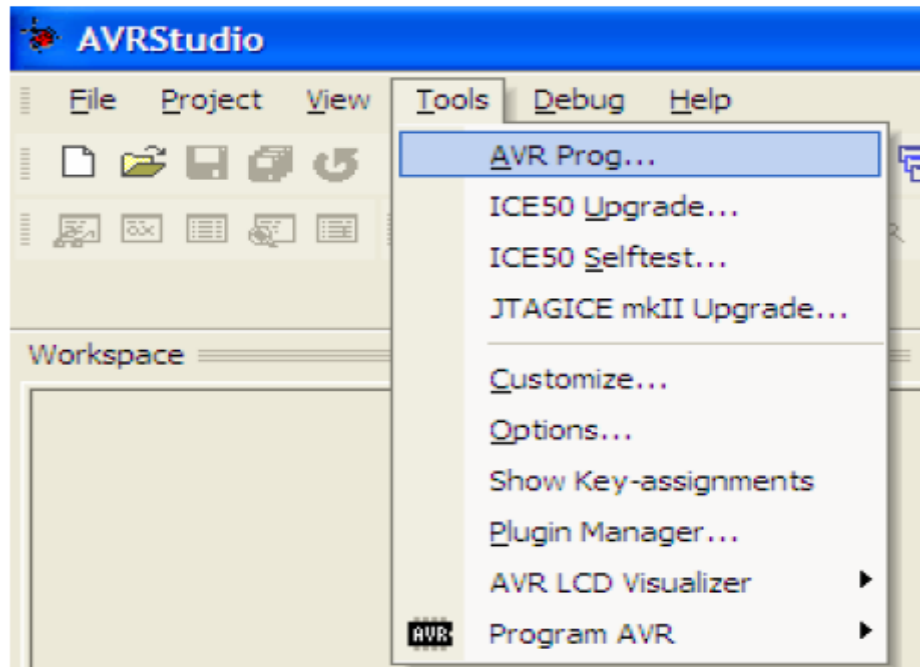


- Seleccionar el AVR Simulator y luego el Dispositivo ATmega169

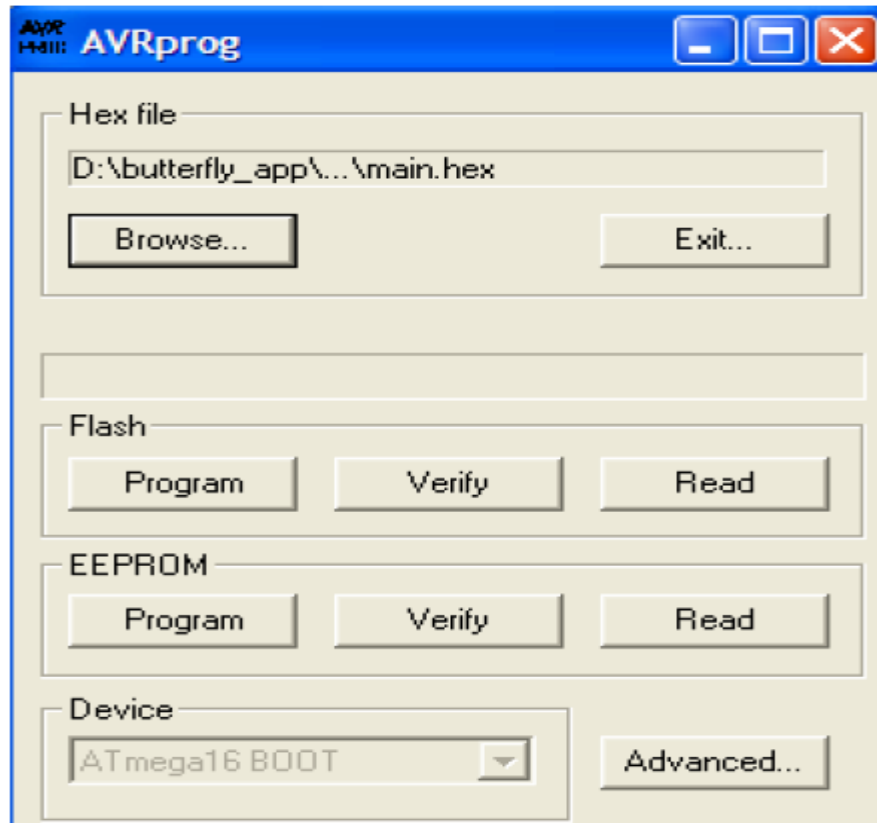


- Presionar "Finish".
- Conectar el cable serial entre la PC y el AVR Butterfly.
- Resetear el AVR Butterfly cortocircuitando los pines 5 y 6 en el conector J403, conector ISP, o quitando y aplicando nuevamente la fuente de alimentación.
- Luego de un reset, el microcontrolador ATmega169 comenzará desde la sección de Arranque. Nada se observará en el LCD mientras esté en la sección de Arranque. Entonces se deberá presionar ENTRAR

en el joystick y mantener esa posición; mientras tanto desde la PC en el AVR Studio, iniciar el AVR Prog.



- Una vez que se haya iniciado el AVR Prog, soltar el joystick del AVR Butterfly. Desde el AVR Prog, utilizar el botón “Browse” para buscar el archivo con la extensión \*.hex con el que desea actualizar al AVR Butterfly. Una vez localizado el archivo \*.hex, presionar el botón “Program”. Se notará que “Erasing Device”, “Programming” y “Verifying” se ponen en “OK”, de manera automática. Luego de actualizar la aplicación, presionar el botón “Exit” en el AVR Prog para salir del modo de programación del ATmega169.



Para que empiece a ejecutarse la nueva aplicación, resetear el AVR Butterfly cortocircuitando los pines 5 y 6 en el conector J403 y presionar el joystick hacia ARRIBA.