ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

"CONTROL MEDIANTE JOYSTICK DE TARJETA AVR BUTTERFLY (CON MICROCONTROLADOR ATMEGA169) MEDIANTE COMUNICACIÓN SPI CON TARJETA LPCXPRESSO CONTROLADORA DE MOTOR BLDC Y PRESENTACIÓN EN DISPLAY DE MENSAJES DE OPERACIÓN"

TESINA DE SEMINARIO

Previa la obtención del Título de:

INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

Presentado por:

Juan Andrés Maroto Lema

Lidia Isabel Mantilla Alarcón

GUAYAQUIL – ECUADOR

AÑO 2012

AGRADECIMIENTO

A Dios Todopoderoso.

A nuestros padres, hermanos y amigos.

A la Escuela Superior Politécnica del Litoral (ESPOL) y a nuestros profesores.

A nuestro director de tesis, el Ing. Carlos Valdivieso.

DEDICATORIA

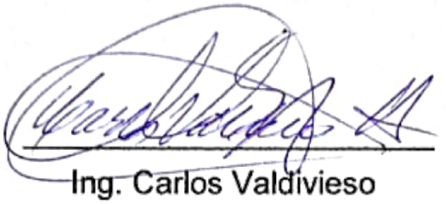
A Dios Todopoderoso, por habernos dado la sabiduría y la fortaleza para alcanzar nuestro sueño de ser ingenieros, y ser nuestro amparo cuando más necesitamos.

A nuestros padres, hermanos y amigos, por su cariño, apoyo, dedicación, por su continuo y afectuoso aliento, y por enseñarnos que la perseverancia y el esfuerzo son el camino para lograr objetivos.

A la Escuela Superior Politécnica del Litoral (ESPOL) por el apoyo brindado para la realización de este proyecto y a nuestros profesores quienes fueron pilar fundamental para el enriquecimiento de nuestros conocimientos.

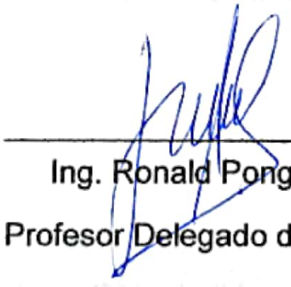
A nuestro director de tesis, el Ing. Carlos Valdivieso, por brindarnos la oportunidad de desarrollar este proyecto de tesis y ser nuestra guía para la elaboración del mismo.

TRIBUNAL DE SUSTENTACIÓN



Ing. Carlos Valdivieso

Profesor del Seminario de Graduación



Ing. Ronald Ponguillo

Profesor Delegado de la FIEC

DECLARACIÓN EXPRESA

"La responsabilidad del contenido de esta tesina, nos corresponde exclusivamente; y el patrimonio intelectual del mismo a la ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL".

(Reglamento de exámenes y títulos profesionales de la ESPOL)

Juan Andrés Maroto Lema

Lidia Isabel Mantilla Alarcón

RESUMEN

El objetivo principal de este proyecto es aplicar el uso de microcontroladores en el control de motores BLDC y, de esta manera, mostrar la utilidad, desarrollo y beneficios de emplear estos dispositivos electrónicos para el manejo de motores, con el fin de explicar de manera detallada las características de los motores BLDC, cómo se establece la comunicación SPI entre las tarjetas LPCXpresso1769 mediante la comunicación entre maestro y esclavo, mostrar la programación del microncontrolador LPCXpresso1769 y el AVR Butterfly, implementar el control de motores mediante el uso del joystick integrado al AVR y, por último, mostrar los resultados del control del motor BLDC por medio de un display, todo esto brindándole un enfoque teórico y práctico para, de esta manera, poder entender el funcionamiento del protocolo SPI, haciendo uso de varias herramientas que incluyen el software del LPCXpresso y el diseño en prototipo del proyecto.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Este proyecto tiene como objetivo el desarrollo y la implementación de un controlador de motor BLDC mediante un joystick que se encuentra incorporado en la tarjeta AVR Butterfly y la presentación del control de motor por medio de un display de mensajes, todo esto utilizando para su funcionamiento el protocolo de comunicación SPI (Serial Peripherical Interface), el cual es programado en la tarjeta LPCXpresso1769.

En el primer capítulo, se menciona una descripción general del proyecto, haciendo principal mención en el tema del protocolo de comunicación SPI, conceptos teóricos sobre el funcionamiento de este protocolo y su aplicación en el microcontrolador LPCXpresso por medio del cual realizaremos la comunicación entre dispositivos.

En el segundo capítulo, se da un detalle sobre las herramientas de hardware y se profundiza en el concepto, funcionamiento y uso de motores BLDC que son el objeto de manejo de las tarjetas a emplear en este proyecto.

El tercer capítulo, trata específicamente sobre el diseño del proyecto, una descripción del funcionamiento del código, diagrama de bloques, el diagrama de flujo, el algoritmo de la parte del control de motores; además se muestra el uso de las herramientas de software utilizadas: LPCXpresso y AVR

STUDIO4, los cuales son herramientas que permiten la programación en lenguaje C.

En el cuarto y último capítulo se muestran los diagramas esquemáticos de los ejercicios para llevar a cabo el proyecto y la lista de elementos utilizados para su implementación en prototipo.

CAPÍTULO 1

DESCRIPCIÓN GENERAL DEL PROYECTO

La tecnología ha avanzado tanto en nuestros tiempos actuales que gran cantidad de procesos han sido resumidos en microcontroladores, los cuales simplifican procesos y espacio físico, optimizando el funcionamiento en un sinnúmero de aplicaciones en diversos campos. Sin embargo, hasta en los circuitos más modernos, es necesario incorporar más de un chip en el diseño y es fundamental que los componentes sean capaces de comunicarse entre ellos. Es por esto que estos componentes requieren un sistema de comunicación que les permita comunicarse entre sí. Esta tesina hace referencia a la comunicación en serie, la cual será objeto de nuestro estudio en este texto.

Existen dos formas principales de comunicación, serie y paralelo. En diversas áreas, como telecomunicaciones y computación, la comunicación serie es el proceso de envío de datos de un bit por vez sobre un canal de comunicación o un bus de computadora, a diferencia de la comunicación paralela, donde todos los bits de cada símbolo (la más pequeña unidad de datos transmitida por vez) son enviados juntos.

La comunicación serie es utilizada en casi todas las comunicaciones y redes de computadoras debido a que los costos de los cables y las dificultades de sincronización hacen que la comunicación paralela sea poco práctica.

En la comunicación paralela, se emplean tantas conexiones como bits que necesitan ser enviados y recibidos. Los buses de comunicación paralelo son muy rápidos, sin embargo, llegan a ser extremadamente grandes y consumen invaluable espacio de pines que se utilizan en un microcontrolador, lo que implica que su programación resulte más compleja de realizar. Debido a estas razones, la comunicación serial es más utilizada que la comunicación en paralelo al conectar varios microcontroladores, ya que mediante este tipo de comunicación se ahorra el uso de pines y el manejo de su programación es más sencillo de realizar e interpretar al momento de implementar.

Un bus de comunicación serial funciona mediante el envío de ceros y unos de forma secuencial sobre uno o dos cables, este tipo de comunicación

permite la transmisión de los datos en ambos sentidos (transmisor – receptor y viceversa) y puede realizarse de manera simultánea. En la mayoría de los microcontroladores modernos es estándar tener un módulo de hardware que controla el reloj del Serial Peripheral Interface (SPI), el buffer receptor y el buffer transmisor.

El bus SPI se utiliza sobre todo entre los microcontroladores y sus dispositivos periféricos inmediatos. Se encuentra comúnmente en los teléfonos móviles, PDAs y otros dispositivos móviles que comunican datos entre la CPU, el teclado, la pantalla y los chips de memoria.

En el siguiente segmento referente a esta tesina explicaremos el desarrollo e implementación de un controlador mediante joystick para un motor BLDC y mostraremos el control de dirección del motor por medio de la LCD del AVR Butterfly. Explicaremos en detalle la realización del mismo de tal manera que sea de fácil entendimiento y comprensión.

1.1. Antecedentes

Imaginemos un motor muy eficiente que casi no malgastase la energía que le viene de las baterías en forma de calor. Esto supondría más potencia, más aceleración, mayor duración y menos desgaste de las baterías, posibilidad de mover mayores pesos y un menor mantenimiento del motor. Esto ya es casi una realidad con los motores sin escobillas. Se trata de motores eléctricos

que no usan escobillas, los cuales son mucho más eficientes y casi no requieren mantenimiento. Para ver las diferencias entre los motores con y sin escobillas, lo mejor es resumir el funcionamiento de estos dos tipos de motores eléctricos.

Los motores con escobillas están formados por dos partes fundamentales, una parte estática y otra móvil. La parte que no se mueve es la carcasa del motor, en cuyo interior (y unidos a ella) están situados dos imanes permanentes. A esta parte se ha denominado estator. En el interior de esa carcasa está la parte móvil (rotor) que también se denomina inducido. Está formado por un eje metálico con tres polos que sobresalen, y en cada polo está enrollado en forma de bobina el hilo conductor. Cuando la corriente eléctrica pase por la bobina, se producirá un campo magnético que interaccionará con el campo magnético que producen los imanes del estator. La consecuencia es una fuerza que hará que gire el inducido, y por tanto el eje del motor. De esta forma se transforma la energía eléctrica, que llega al motor, en energía mecánica.

La corriente llega al inducido a través de tres delgas que están situadas en su parte superior y forman lo que se denomina el conmutador o colector. A su vez las delgas reciben la electricidad de dos escobillas, que son unos contactos deslizantes que rozan las delgas mientras el rotor gira. Como una escobilla está conectada al polo positivo de la corriente y otra al negativo, la

corriente que transmiten a las tres delgas hace que el inducido gire. Son necesarias las tres delgas para garantizar que el rotor se mueva, ya que si solo hubiese dos, el rotor podría no iniciar el giro al quedar en posición perpendicular a las fuerzas magnéticas de los imanes. También hemos de saber que las escobillas se mantienen pegadas a las delgas mediante la presión de sendos muelles. El inducido gira dentro del estator sobre rodamientos.

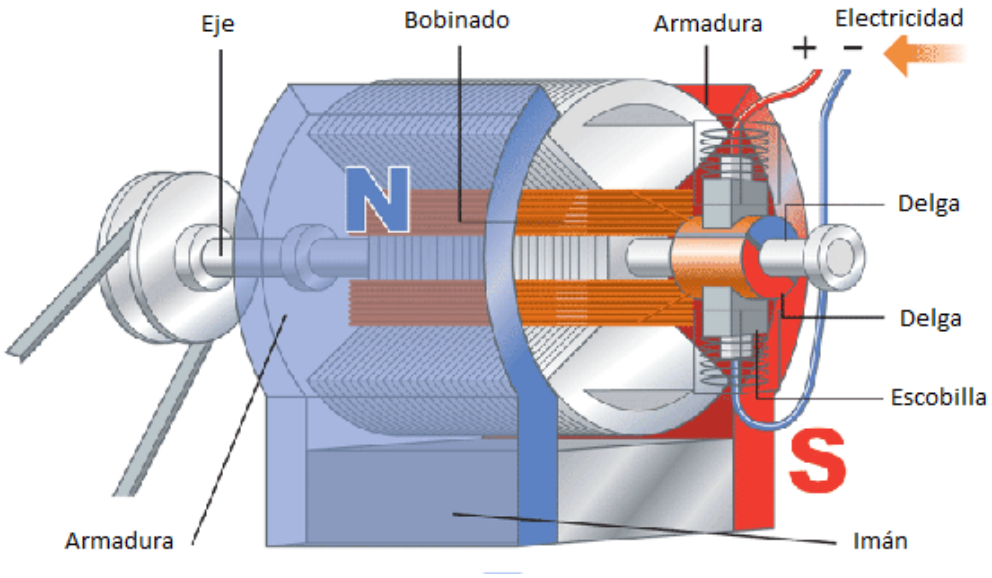


Figura 1.1. Motor con escobillas **[1]**

A modo de resumen, 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 mediante sensores instalados en el motor o a través de la respuesta que obtiene cuando envía una corriente lineal al motor. Debido a esto los variadores electrónicos de los motores sin escobillas han de ser mucho más complejos que los usados en motores con escobillas, ya que han de procesar la información del funcionamiento del motor en tiempo real.

Los motores eléctricos sin escobillas se han venido utilizando desde hace años en la industria en general, aplicándose en campos como la medicina, la industria, el aeromodelismo, 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.

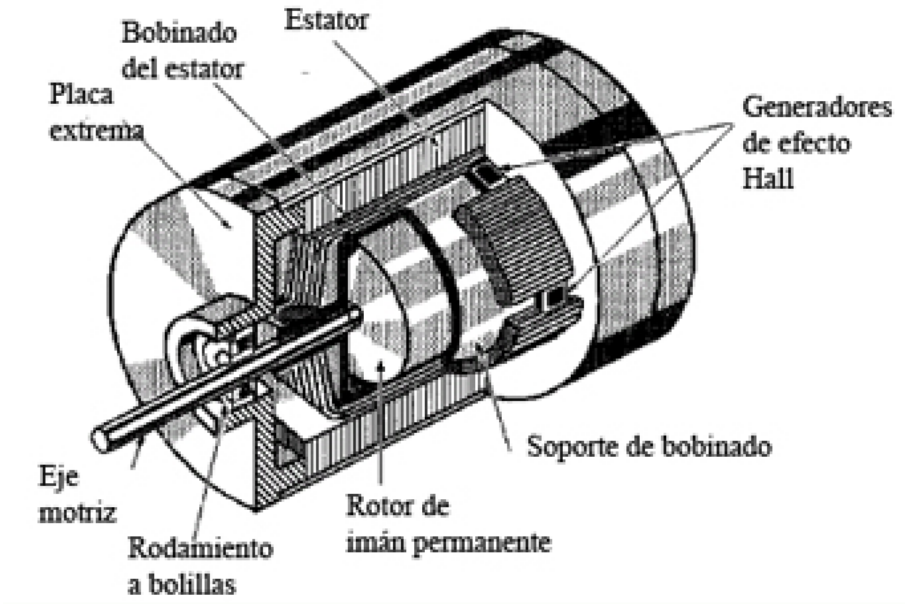


Figura 1.2. Vista en corte de un motor DC sin escobillas **[2]**

1.2. Motivaciones para el proyecto

La determinación de realizar el estudio e implementación de este proyecto de tesis está basado principalmente en adquirir mayor conocimiento acerca del manejo de microcontroladores y electrónica en general de los que ya hemos recibido durante el transcurso del estudio de la carrera tanto de forma teórica como práctica al momento de trabajar en los laboratorios, los cuales son aplicables para la carrera de Ingeniería en Electrónica y Telecomunicaciones, ya que estos conocimientos nos pueden ser de gran ayuda al momento de ejercer nuestra profesión como tal.

Con el afán de profundizar en el estudio de los microcontroladores y aprovechar todas las utilidades que los mismos nos pueden ofrecer en la

implementación de sistemas de control, tomamos la decisión de tomar este proyecto de tesis, el cual no solo es aplicable a nivel de electrónica sino también de protocolos de comunicación, en nuestro caso el uso del protocolo SPI, para establecer la comunicación entre las tarjetas, con el fin de trabajar con varios dispositivos a la vez y así realizar procesos de manera simultánea durante la implementación de nuestro proyecto, aprovechando tiempos de respuesta y todas las herramientas para el control de motores que los microcontroladores nos pueden ofrecer.

Entre nuestras motivaciones se encuentra como meta la obtención del título de Ingenieros en Electrónica y Telecomunicaciones y así conseguir el título de tercer nivel, el cual nos brinda mayores oportunidades laborales.

Este proyecto surge en base al diseño e implementación para el manejo y control de motores BLDC con sensores y su aplicación en el campo industrial, haciendo uso del protocolo de comunicación serial SPI (Serial Peripherical Interface) y de herramientas de hardware y software pertenecientes a LPCXpresso y Atmel Corporation (AVR Butterfly), con el apoyo del laboratorio de microcontroladores y el Ing. Carlos Valdivieso, quien nos proporcionó las herramientas necesarias para la elaboración de este proyecto de tesis.

1.3. Identificación del problema

La implementación de un proyecto de tesis mediante el uso de microcontroladores demanda algunas características fundamentales para permitir el correcto funcionamiento entre estos dispositivos, entre los cuales se encuentran la programación, la sincronía entre dispositivos y la compatibilidad entre los mismos.

Uno de los principales factores de dificultad al realizar este proyecto de tesis fue la comunicación por medio del protocolo SPI entre maestro y esclavo de las tarjetas LPCXpresso, ya que este tipo de comunicación requiere sincronización y el uso adecuado de ciertas librerías que vienen en el software del fabricante.

Otro problema que tuvimos fue la comunicación entre la tarjeta LPCXpresso y el AVR Butterfly, debido a que son dispositivos que trabajan con diferente software de programación por pertenecer a diferentes fabricantes.

Con el fin de desarrollar un sistema de control y manejo de motores BLDC, se busca realizar la interacción entre estos microcontroladores para aprovechar las ventajas y utilidades que los mismos nos ofrecen en el sector industrial, demostrando que el trabajo integrado de los mismos es factible, aunque tengan diferentes características de fábrica y se manejen con sistemas de software distintos.

1.4. Metas y Objetivos

El objetivo principal para la implementación de este proyecto de tesis es la aplicación de los microcontroladores para el manejo de motores BLDC mediante un joystick, integrado en la AVR Butterfly, que nos sirve como herramienta de control de los motores, todo esto es posible mediante las funciones que pueden realizar los microcontroladores y la comunicación SPI que se establece entre maestro – esclavo para dar las órdenes al motor.

También es importante destacar que en esta tesina se detalla y profundiza en el marco teórico sobre motores BLDC con y sin escobillas y el protocolo de comunicación SPI, para el control de los mismos, con el fin de emplear una nueva herramienta de desarrollo de microcontroladores como lo es LPCXpresso.

En resumen podemos decir que, para la realización de este proyecto, se requiere tener conocimientos en motores BLDC y en cómo se maneja su mecánica de funcionamiento; programar en el software LPCXpresso y conocer las características, registros y funciones que maneja el microcontrolador LPC1769; programar en el software AVR Studio 4 y conocer las características que emplea la tarjeta ATMEGA169; y comprender cómo se establece y realiza la comunicación SPI entre dispositivos mediante la asignación de un maestro y un esclavo, todo esto con el objetivo de explicar en una forma más amplia la aplicación de los microcontroladores en

diferentes campos, la extraordinaria forma en que reducen procesos y espacio físico en implementación de proyectos y su funcionamiento e interacción entre dispositivos de igual o diferente fabricante mediante protocolos de comunicación.

CAPÍTULO 2

FUNDAMENTO TEÓRICO

En este capítulo realizamos un análisis profundo sobre el funcionamiento y características de los elementos utilizados para la elaboración de este proyecto, entre ellos destacamos el protocolo de comunicación serial SPI (Serial Peripherical Interface) que es la base para realizar la transmisión de órdenes hacia el motor y el concepto y características de los motores BLDC, con y sin escobillas y de efecto Hall, lo cual es importante mencionar debido a que se debe entender claramente el funcionamiento mecánico y electromagnético de este tipo de motores para posteriormente realizar la programación en el microcontrolador LPC1769 (fabricante LPCXpresso) y el microcontrolador ATMEGA169, el cual se programa mediante el software AVR Studio 4.

2.1. Interfaz Periférica Serial – SPI

Como se mencionó anteriormente, el intercambio de información digital entre un microcontrolador y los periféricos puede establecerse de dos formas: serial o paralela. La comunicación paralela implica el envío simultáneo de la información, es decir, de varios bits al mismo tiempo, lo cual hace más complejo el manejo a nivel de costo e implementación ya que se requiere de mayor cableado y de mayor cantidad de terminales de E/S utilizadas en el microcontrolador, lo cual quita valioso espacio de uso en un microcontrolador. A diferencia de la comunicación paralela, la comunicación serial plantea justamente lo contrario, ya que en vez de enviar los bits del dato de manera simultánea, los envía por separado, es decir uno detrás del otro. En este capítulo se explicará de forma más detallada todo lo relacionado con la comunicación SPI (Serial Peripheral Interface) y su forma de establecer comunicación entre dos dispositivos, uno que actúa como maestro y otro que funciona como esclavo.

2.1.1. Definición

El SPI (Serial Peripheral Interface) es un subsistema de comunicación serial independiente, un estándar establecido por Motorola que emplea un bus de 4 líneas para interconectar dispositivos periféricos de baja y media velocidad como Eeprom, displays con pantalla de cristal líquido LCD, sistemas de

conversión analógico – digital (DAC y ADC), e incluso otros microcontroladores y dispositivos como los AVR. La comunicación se realiza basada en la interacción maestro – esclavo, donde el maestro llama al esclavo para establecer el proceso de trasmisión y recepción de la información. La comunicación SPI trabaja de manera full dúplex, esto quiere decir que puede enviar y recibir información de manera simultánea, elevando la transferencia de los datos. Con esta interfaz, usted tiene un dispositivo maestro que inicia y controla la comunicación, y uno o más esclavos que reciben y transmiten hacia el maestro.

Este protocolo permite establecer comunicación entre dispositivos básicamente mediante el uso de un maestro y un esclavo. El maestro es aquel que inicia la transferencia de datos sobre el bus y genera las señales de reloj (SCK) y control (MISO, MOSI). Un esclavo es un dispositivo el cual es controlado por el maestro. Cada esclavo es controlado sobre el bus a través de una línea selectora llamada Chip Select o Select Slave (SSEL), esto quiere decir que el esclavo es activado solamente cuando esta línea es seleccionada.

Cuando está configurado como maestro, la transferencia de datos puede ser tan alta en proporción a un medio de los ciclos del reloj, mientras que cuando se configura en modo esclavo la transferencia de datos puede ser tan rápida como la razón del reloj. A continuación se muestra una imagen sobre la

comunicación entre maestro – esclavo que puede ser establecida para este protocolo de comunicación.

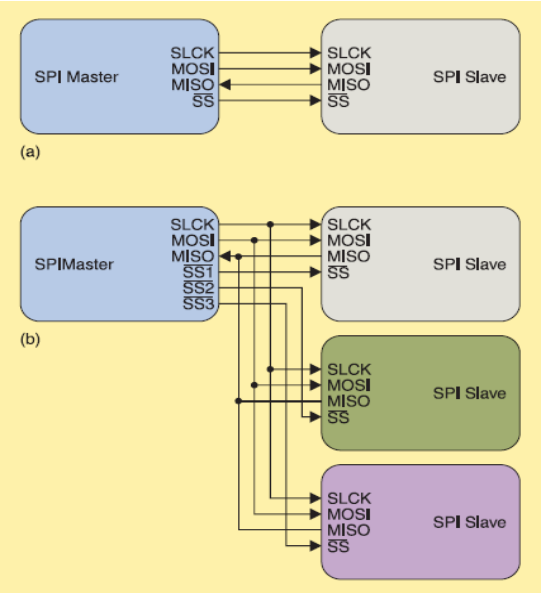


Figura 2.1.: a) Conexión entre un dispositivo maestro y un esclavo.
b) Conexión entre un dispositivo maestro y varios esclavos. **[3]**

Antes de poder comunicarse a través del protocolo SPI, tanto el maestro como el esclavo deben ponerse de acuerdo sobre algunos ajustes en la señal del reloj. Los detalles sobre cómo configurar esto en el AVR se discutirá más adelante.

2.1.2. Descripción de bits del SPI

Para establecer la comunicación SPI se utilizan cuatro señales (pines) para establecer comunicación mediante el protocolo SPI: MISO, MOSI, SCK y SS.

A continuación se muestra una breve descripción de la función que cumple cada señal:

MISO (Master In Out Slave): Se toma como pin de entrada el registro de desplazamiento del Maestro, y como pin de salida el registro del Esclavo.

MOSI (Master Out Slave In): Se toma como pin de salida el registro de desplazamiento del Maestro, y como pin de entrada el registro del Esclavo.

SCK (Serial Clock): En el Maestro, esta es la salida del generador de reloj. En el esclavo, es la señal de reloj de entrada.

SS (Slave Select): Sirve como selector de esclavo. Puesto que en una configuración de SPI puede tener varios esclavos, al mismo tiempo, usted necesita una manera de seleccionar el esclavo con el cual desea comunicarse. SS se utiliza para realizar esta función. Si SS se mantiene en un estado de alta, todos los pines esclavos SPI son las entradas normales, y no recibirá los datos entrantes de SPI. Por otro lado, si SS se mantiene en un estado bajo, el SPI se activa.

2.1.3. Características básicas del SPI

Entre las características más importantes de este protocolo de comunicación podemos citar las que se muestran a continuación:

- Puede transmitirse información desde un periférico maestro hacia uno o varios periféricos esclavos.
- La polaridad y fase para la transferencia de datos están relacionadas con la sincronización del reloj.
- La transferencia se realiza de manera full dúplex, es decir, que se puede enviar y recibir simultáneamente por el mismo canal.
- La transferencia de Datos puede ser LSB o MSB
- Se puede realizar la interrupción por bandera, también se puede usar el transfer polling para la sincronización de los datos.

2.2. Operación y Funcionamiento del SPI

Se inicializa el ciclo de comunicación configurando en el SPI Maestro en bajo el pin asignado para SS (Slave Select). El Maestro es quien se encarga de generar el pulso de reloj mediante el pin SCK para comenzar el intercambio de datos entre Maestro y Esclavo. Se habilita el pin MOSI para realizar la transferencia de datos desde el Maestro hacia el Esclavo, y se habilita el pin MISO para realizar el intercambio de datos desde el Esclavo hacia el Maestro.

El maestro transmite un bit de su SPDR al dispositivo esclavo en cada ciclo de reloj, esto quiere decir que para poder enviar un byte de datos se requiere que transcurran 8 ciclos de reloj. Luego de que se ha realizado el envío del byte completo, el Maestro debe enviar un alto al SS (Slave Select) para

establecer sincronía entre ambos. Siempre se debe configurar mediante software el manejo del SS, ya que el Maestro no tiene un manejo automático de la línea SS.

En un tiempo determinado, solo podrá existir un maestro sobre el bus. Cualquier dispositivo esclavo que no esté seleccionado, debe deshabilitarse poniendo un alto a través de la línea selectora (chip select).

El SS debe ser manejado por software antes de que la comunicación pueda empezar, esto se realiza escribiendo un byte en el registro SS del SPI, una vez que comienza el reloj del SPI, y el hardware cambia los 8 bits dentro del esclavo. Después de cambiar un Byte en el SS, el reloj del SPI para, habilitando el fin de la transmisión (SPIF).

Si la interrupción del SPI está habilitada (SPIE) en el registro SPCR, una interrupción es requerida. El maestro podría continuar al cambio del siguiente byte escribiendo dentro del SPDR, o señalizar el fin del paquete colocando en alto el Esclavo seleccionado, línea SS. Para este proyecto, no usaremos interrupciones, sino que trabajaremos en modo polling para la transferencia de los datos.

Cuando se configura como Esclavo, la interfaz ISP permanecerá durmiendo con MISO en tres-estados siempre y mientras el pin SS esté deshabilitado. En este estado, por el software se podría actualizar el contenido del registro SPDR, pero los datos no serán desplazados por la llegada del pulso de reloj

en el pin SCK hasta que el pin SS no sea habilitado ('0'). Será visto como un byte completamente desplazado en el fin de la transmisión cuando SPIF se habilite.

Si la interrupción SPI está habilitada, una interrupción es solicitada. El Esclavo podría continuar para colocar nuevos datos para ser enviados dentro del SPDR antes de seguir leyendo el dato que va llegando. El último byte que entra permanecerá en el buffer para luego usarse. Si SS es configurado como salida, esta salida es general por lo cual no afecta al sistema SPI. Por lo general, el SS es manejado desde el esclavo. Si está configurado como entrada, debe ser seteado en alto para asegurar la operación SPI del Maestro.

A continuación se detalla mediante figuras la transmisión de los bits desde el Maestro hacia el Esclavo:

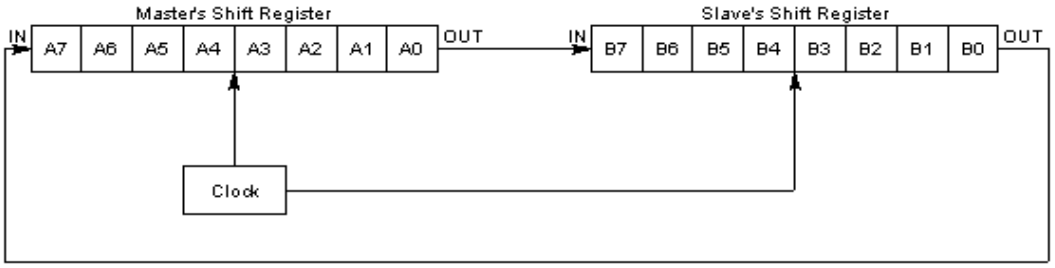


Figura 2.2.: Estado de los dispositivos antes de establecer la transferencia de datos.

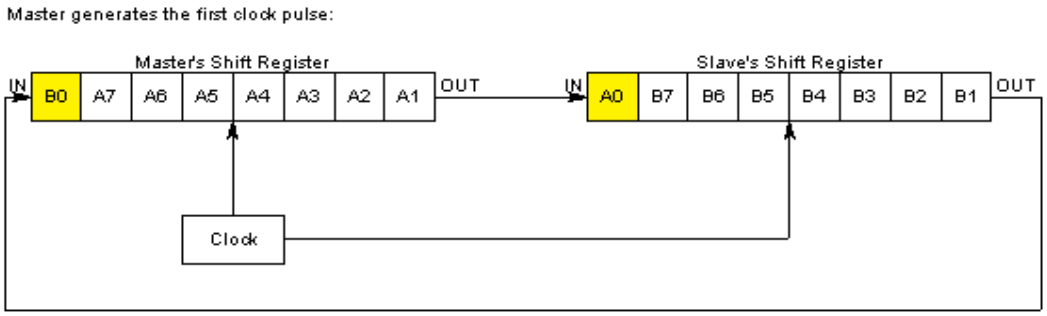


Figura 2.3.: El Maestro genera el primer pulso de reloj.

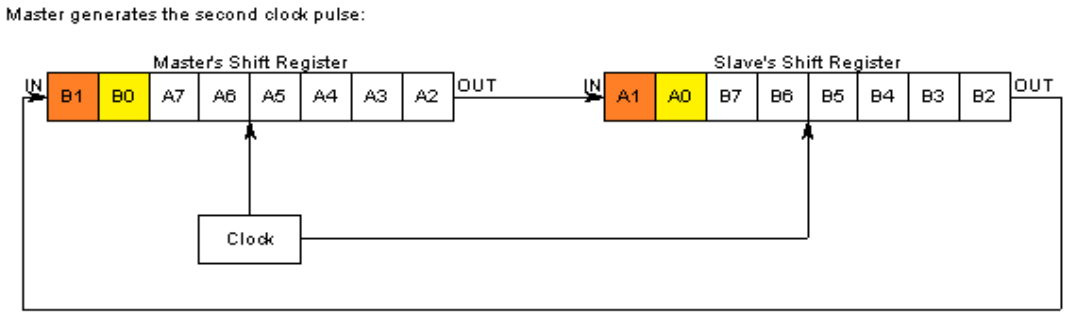


Figura 2.4.: El Maestro genera el segundo pulso de reloj.

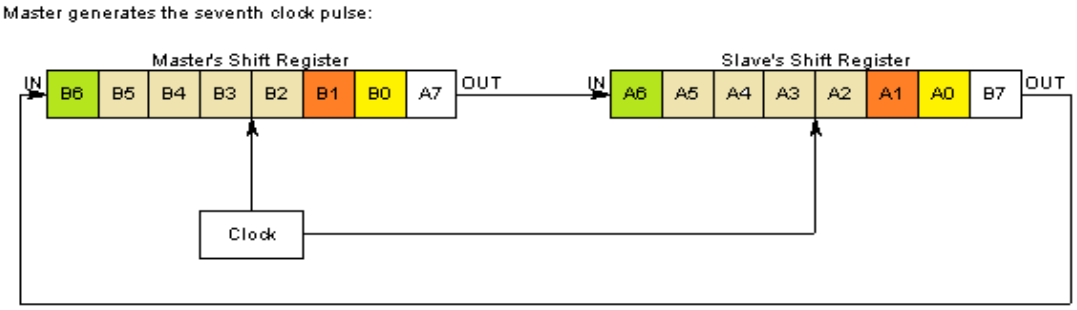


Figura 2.5.: El Maestro genera el séptimo pulso de reloj.

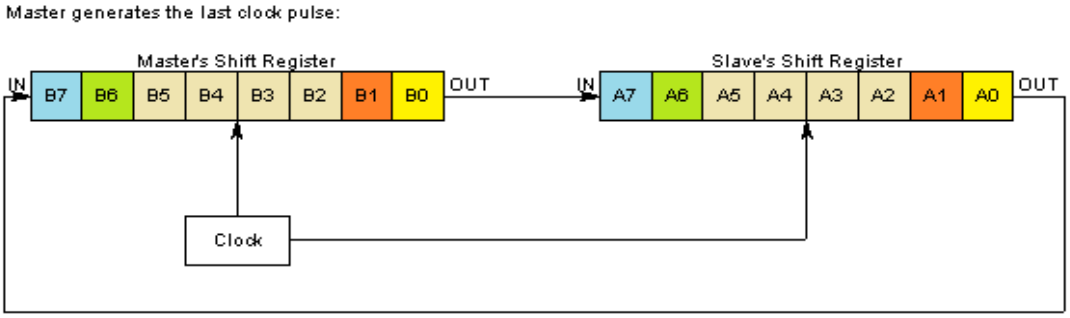


Figura 2.6.: El Maestro genera el último pulso de reloj.

2.3. Protocolo de comunicación SPI - AVR Butterfly, ATMega169

2.3.1. Registros del SPI

En este capítulo se describen cada uno de los registros que trabajan mediante protocolo SPI tanto en la AVR Butterfly como en la tarjeta LPC1769, así mismo se detalla la función de cada uno de los pines asignados en cada registro.

2.3.1.1. Registro de Control del SPI – SPCR

Bit	7	6	5	4	3	2	1	0	
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Valor inicial	0	0	0	0	0	0	0	0	

SPIE (Habilitación de la Interrupción por SPI): Este bit realiza la habilitación de la interrupción en SPI, si el bit SPIF (SPSR) y el SRGE son habilitados.

SPE (Habilitación del SPI): Cuando el bit SPE está en uno, el SPI es habilitado. Este bit debe estar habilitado para la ejecución de cualquier operación SPI.

DORD (Orden de los datos): Cuando el bit DORD está en uno, el LSB del dato se transmite primero, si el bit DORD es borrado o está en cero, el MSB del dato se transmite primero.

MSTR (Selector de Maestro/Esclavo): Este bit selecciona el modo maestro de SPI cuando está en uno, y el modo esclavo de SPI cuando es borrado o está en cero. Si el pin SS (Slave Select) es configurado como entrada y es habilitado (0) mientras MSTR está habilitado, MSTR será deshabilitado, y SPIF (en SPSR) se habilitará. El usuario tendrá que poner en uno MSTR para rehabilitar el modo maestro del SPI.

CPOL (Polaridad del reloj): Cuando este bit está en uno, SCK será activado en nivel alto cuando estamos en modo Idle, caso contrario será activado en bajo cuando estamos en modo Idle.

CPHA (Fase del reloj): Se encarga del control de la señal del reloj y de los tiempos de duración de cada ciclo de reloj. Si CPHA es igual a cero, los datos sobre la línea MOSI son detectados cada flanco de bajada y los datos sobre la línea MISO son detectados cada flanco de subida. Tanto el dispositivo Maestro como el dispositivo esclavo deben encontrarse sincronizados con la misma polaridad y fase de reloj.

SPR1, SPR0 (Selección de la frecuencia de clock 1 y 0 del SPI): Es el selector de la velocidad del reloj. Estos dos bits controlan la frecuencia de SCK del dispositivo configurado como maestro. SPR1 y SPR0 no tienen efecto sobre el esclavo.

La relación entre SCK y la frecuencia F_{osc} de clock del oscilador se muestra a continuación:

SPI2X	SPR1	SPR0	Frecuencia de SCLK
0	0	0	$F_{osc}/4$
0	0	1	$F_{osc}/16$
0	1	0	$F_{osc}/64$
0	1	1	$F_{osc}/128$
1	0	0	$F_{osc}/2$
1	0	1	$F_{osc}/8$
1	1	0	$F_{osc}/32$
1	1	1	$F_{osc}/64$

Tabla 2.1.: Relación entre SCK y la frecuencia de oscilación

2.3.1.2. Registro de Estado del SPI - SPSR

Bit	7	6	5	4	3	2	1	0	
	SPIF	WCOL	-	-	-	-	-	SPI2X	SPSR
Read/Write	R	R	R	R	R	R	R	R/W	
Valor inicial	0	0	0	0	0	0	0	0	

SPIF (Bandera de Interrupción SPI): Cuando una transferencia serial es completada, la bandera SPIF se habilita. Una interrupción es generada si SPIE (en SPCR) y SREG están habilitados. Si SS es una entrada y está en habilitada (0) cuando el SPI está en modo Maestro, este también habilitará la bandera SPIF. SPIF es deshabilitado por hardware cuando es ejecutada la correspondiente interrupción. Alternativamente, el bit SPIF es deshabilitado leyendo primero el registro de estado del SPI cuando SPIF está en uno.

WCOL (Bandera de colisión de escritura): Es habilitado si se produce una escritura durante la transferencia de datos. Es deshabilitado con la primera lectura del registro de estado SPI (SPDR).

SPI2X (Bit de velocidad doble del SPI): Cuando este bit está en alto la velocidad SPI (frecuencia SCK) será duplicada cuando el SPI esté en modo maestro. Esto significa que el periodo mínimo de SCK será de dos periodos de reloj del CPU. Cuando el SPI está configurado como esclavo, el SPI sólo está garantizado para trabajar a $F_{osc}/4$ o inferior.

2.3.1.3. Registro de Datos del SPI – SPDR

Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	SPDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Valor inicial	X	X	X	X	X	X	X	X	

El registro de datos del SPI es un registro de lectura y escritura, usado para la transmisión de datos entre el registro de archivos y el registro de cambio SPI. Escribiendo en el registro comienza la transmisión de datos. La lectura del registro prepara el búffer de recepción del registro de desplazamiento para ser leído.

2.3.2. Modos del Reloj

La mayoría de las interfaces SPI tienen 2 bits de configuración, llamados CPOL (Clock Polarity o Polaridad del Reloj) y CPHA (Clock Phase o Reloj de Fase). CPOL determina si el estado Idle de la línea de reloj está en bajo (0)

o si se encuentra en un estado alto (1). CPHA determina en qué flanco de reloj los datos son desplazados hacia dentro o hacia fuera.

Si CPHA está en bajo, los datos sobre la línea MOSI son detectados cada flanco de bajada y los datos sobre la línea MISO son detectados cada flanco de subida.

Cada bit tiene dos estados, lo cual permite 4 diferentes combinaciones, las cuales son incompatibles una con la otra. Por lo que si dos dispositivos SPI desean comunicarse entre sí, éstos deben tener la misma polaridad de reloj (CPOL) y la misma fase de reloj (CPHA). Existen cuatro modos de reloj definidos por el protocolo SPI, estos modos son: Modo 0, 1, 2 y 3.

Modo del reloj	**Estado de CPOL**	**Estado de CPHA**
Modo 0	CPOL = 0	CPHA = 0
Modo 1	CPOL = 0	CPHA = 1
Modo 2	CPOL = 1	CPHA = 0
Modo 3	CPOL = 1	CPHA = 1

Tabla 2.2.: Estado de CPOL y CPHA de acuerdo al modo de reloj.

Estos modos determinan el valor de la polaridad del reloj (CPOL) y el bit de fase del reloj (CPHA). La mayoría de los dispositivos SPI pueden soportar al menos 2 modos de los 4 antes mencionados. A continuación se realizará un breve análisis sobre los modos antes descritos.

El bit de polaridad del reloj determina el nivel del estado de Idle del reloj y el bit de fase de reloj determina qué flanco recibe un nuevo dato sobre el bus. El modo requerido para una determinada aplicación está dado por el dispositivo esclavo. La capacidad de multi-modo combinada con un simple registro de desplazamiento hace que el bus SPI sea muy versátil.

2.3.3. Polaridad del reloj (CPOL)

Si CPOL está en cero y ningún dato está siendo transferido (Estado Idle), el maestro mantiene la línea SCLK en bajo. Si CPOL está en uno, el maestro desocupa la línea SCLK en alto.

2.3.4. Fase del reloj (CPHA)

CPHA, junto con CPOL, controlan cuando los nuevos datos son colocados en el bus. Si CPHA está en uno, los datos son desplazados sobre la línea MOSI según lo determinado por el valor de CPOL.

Para CPHA en uno, y si CPOL es uno, los nuevos datos son colocados sobre la línea cuando el flanco de reloj es descendente y se leen cuando el flanco de reloj es ascendente.

Para CPHA en uno, y si CPOL es cero, los nuevos datos se ponen en la línea cuando el flanco de reloj es ascendente y se leen cuando el reloj tiene un flanco descendente.

Si CPHA está en cero, el reloj de cambio trabaja con la terminal Chip Select. Tan pronto como el terminal Chip Select se coloca en cero, los nuevos datos se ponen en la línea y, al primer flanco de reloj, se leen los datos. Si CPOL se activa en uno, el primer borde de reloj baja y los bits de datos se leen en cada flanco de bajada sobre la línea de reloj.

En conclusión, si CPHA es uno, la transferencia comienza en el segundo flanco de reloj. Si CPHA es cero, la transferencia comienza en el primer flanco de reloj y todas las transferencias que se realicen a continuación dentro del byte ocurrirán en cada flanco de reloj.

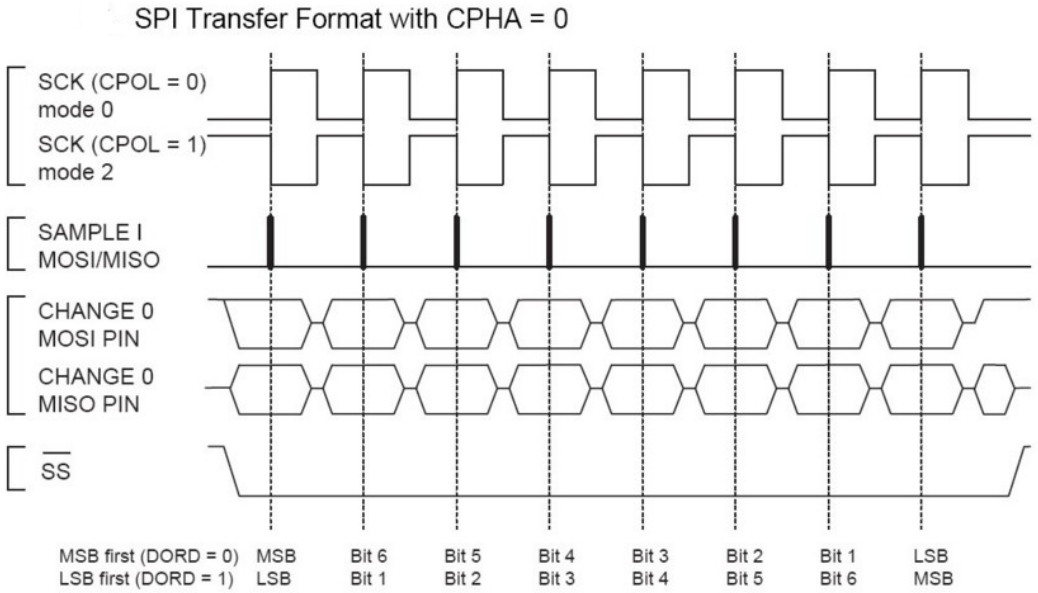


Figura 2.7.: Formato de transferencia de datos SPI cuando CPHA es igual a cero. **[4]**

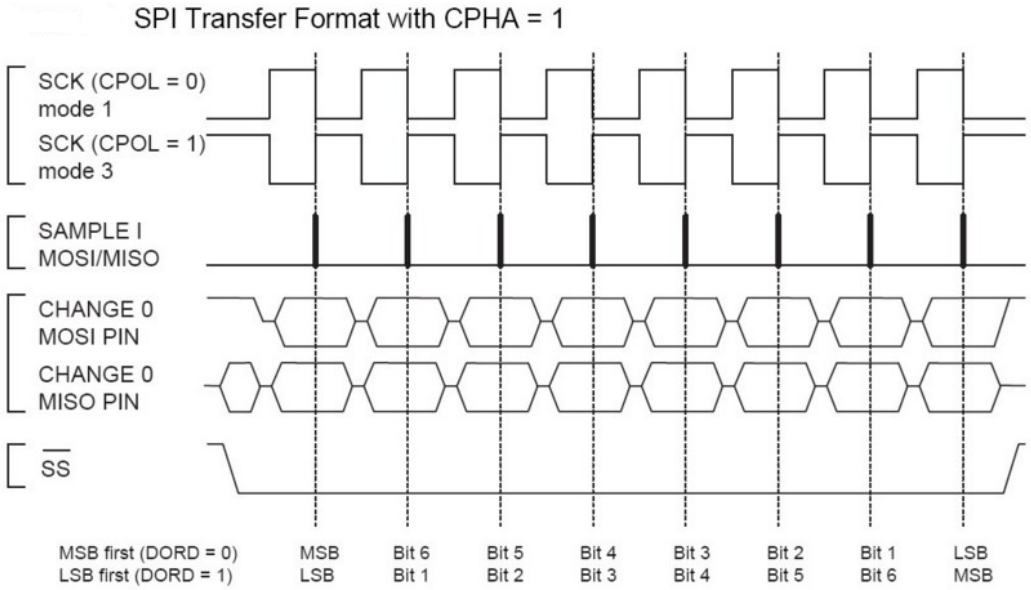


Figura 2.8.: Formato de transferencia de datos SPI cuando CPHA es igual a uno. **[5]**

2.4. Protocolo de comunicación SPI – LPCXpresso, LPC1769

2.4.1. Configuración básica del SPI

De manera similar a la configuración en el AVR, SPI está configurado usando los siguientes registros:

Power: En el registro PCONP, se coloca un alto en PCSPI. En reset, el SPI es habilitado (PCSPPI=1). El registro de control power está conformado por 32 bits que se encuentran configurados en el registro de periféricos PCONP – 0x400F C0C4. El registro PCONP permite deshabilitar funciones periféricas seleccionadas con el propósito de ahorrar energía, ayudando al ahorro de consumo de energía de la que depende el reloj y, de esta manera, dar un uso

óptimo a las funciones que se requiere para la programación de la tarjeta LPC1769.

Clock: En el registro PCLKSEL0, se coloca un alto en el bit PCLK_SPI. En modo maestro, el reloj debe tener un número mayor o igual a 8. La selección del reloj está conformada por 32 bits, los cuales se encuentran configurados en el registro de periféricos PCLKSEL0 – 0x400F C1A8. El registro PCLKSEL0 administra la selección del reloj de los timers, DAC y ADC, SPI, UART, CAN, etc., dependiendo de las funciones que se requiera utilizar para la programación de la tarjeta LPC1769.

Pines: Los pines del SPI son configurados usando PINSEL0 y PINSEL1, así como el registro PINMODE. PINSEL0 es usado para configurar el pin CLK del SPI. PINSEL1 es usado para configurar los pines SSEL, MISO y MOSI.

Interrupciones: La bandera de interrupción del SPI es habilitado usando el bit S0SPINT. SSP0 está destinado para ser utilizado como una alternativa para la interfaz SPI, por lo cual está incluido como un periférico legal. Solo uno de estos periféricos puede ser utilizado en cualquier tiempo.

2.4.2. Registros del SPI

El SPI contiene cinco registros: registro de control (S0SPCR), registro de estado (S0SPSR), registro de datos (S0SPDR), registro de contador de reloj (S0SPCCR) y bandera de interrupción (S0SPINT). Todos estos registros

tienen como valor de reset 0x00. En la tabla que se muestra a continuación se describen los cinco registros que se encuentran para protocolo de comunicación SPI en el LPC1769.

Nombre	Descripción	Acceso	Dirección de memoria
S0SPCR	**Registro de control SPI.** Este registro controla la operación del SPI.	R/W	0x4002 0000
S0SPSR	**Registro de estado SPI.** Este registro muestra el estado del SPI.	RO	0x4002 0004
S0SPDR	**Registro de datos SPI**. Este registro bidireccional permite transmitir y recibir datos para el SPI. La transmisión de datos es realizada por el SPI0 cuando se escribe sobre este registro. Los datos recibidos por el SPI0 pueden ser leídos desde este registro.	R/W	0x4002 0008
S0SPCCR	**Registro de contador de reloj SPI.** Este registro controla la frecuencia del maestro SCK0.	R/W	0x4002 000C
S0SPINT	**Bandera de interrupción SPI**. Este registro contiene la bandera de interrupción para la interfaz SPI.	R/W	0x4002 001C

Tabla 2.3.: Tabla de los registros que maneja el protocolo SPI en la LPC1769.

2.4.2.1. Registro de Control del SPI – S0SPCR

A continuación se muestra la tabla 2.4 que contiene la descripción de cada uno de los 32 pines asignados para el registro de control del SPI en la tarjeta LPC1769 y la función que desarrollan al colocar un alto o un bajo.

Bit	Símbolo	Valor	Descripción
1:0 - 31:12	-		Bits reservados. El usuario del software no puede escribir sobre estos bits.
2	Bit Enable	0	El controlador SPI envía y recibe 8 bits de datos por transferencia.

		1	El controlador SPI envía y recibe el número de bits seleccionados por bits 11:8
3	CPHA		El control de fase de reloj determina la relación entre los datos y el reloj en la transferencia SPI, y controla cuando una transferencia hacia el esclavo es definida como iniciada o terminada.
		0	Los datos son muestreados en el primer flanco de reloj del SCK. La transferencia comienza y termina con la activación y desactivación de la señal SSEL.
		1	Los datos son muestreados en el segundo flanco de reloj del SCK. La transferencia comienza con el primer flanco de reloj y termina con el último flanco de reloj cuando la señal SSEL se activa.
4	CPOL		Control de polaridad del reloj.
		0	SCK se activa en alto.
		1	SCK se activa en bajo.
5	MSTR		Selector de modo maestro.
		0	El SPI opera en modo esclavo.
		1	El SPI opera en modo maestro.
6	LSBF		Controla qué dirección de cada byte es desplazado cuando es transferido.
		0	El dato del SPI es transferido MSB (bit 7).
		1	El dato del SPI es transferido LSB (bit 0).
7	SPIE		Habilitador de interrupción serial periférica.
		0	Interrupciones SPI son inhibidas.
		1	Una interrupción por hardware es generada cada vez que los bits SPIF o MODF son activados.
11:8	BITS		Cuando el bit 2 de este registro es 1, éste controla el número de 0000 bits por transferencia.
		1000	8 bits por transferencia.
		1001	9 bits por transferencia.
		1010	10 bits por transferencia.
		1011	11 bits por transferencia.
		1100	12 bits por transferencia.
		1101	13 bits por transferencia.
		1110	14 bits por transferencia.
		1111	15 bits por transferencia.
		0000	16 bits por transferencia.

Tabla 2.4.: Tabla de registro de control del SPI para la tarjeta LPC1769.

2.4.2.2. Registro de Estado del SPI – S0SPSR

A continuación se muestra la tabla 2.5 que contiene la descripción de cada uno de los 32 pines asignados para el registro de estado del SPI en la tarjeta LPC1769 y la función que desarrollan al colocar un alto o un bajo.

Bit	Símbolo	Descripción
2:0 - 31:8	-	Bits reservados. El usuario del software no puede escribir sobre estos bits.
3	ABRT	Aborto del esclavo. Cuando es 1, este bit indica que un aborto del esclavo ha ocurrido. Este bit es encerado leyendo este registro.
4	MODF	Modo de falla. Cuando es 1, este bit indica que un error en el modo de falla ha ocurrido. Este bit es encerado leyendo este registro, entonces se escribe en el control de registro SPI0.
5	ROVR	Sobre lectura. Cuando es 1, este bit indica que una sobre lectura ha ocurrido. Este bit es encerado leyendo este registro.
6	WCOL	Colisión de escritura. Cuando es 1, este bit indica que una colisión de escritura ha ocurrido. Este bit es encerado leyendo este registro, entonces accede al registro de datos SPI.
7	SPIF	Bandera de transferencia completa SPI. Cuando es 1, este bit indica cuando una transferencia de datos SPI está completa. Cuando es maestro, este bit es seteado en el final del último ciclo de la transferencia. Cuando es esclavo, este bit es seteado en el último flanco de reloj del SCK. Este bit es encerado leyendo este registro, entonces accede al registro de datos SPI. Esta no es la bandera de interrupción del SPI. Esta bandera se encuentra en el registro SPINT.

Tabla 2.5.: Tabla de registro de estado del SPI para la tarjeta LPC1769.

2.4.2.3. Registro de Datos del SPI – S0SPDR

Este registro de datos bidireccional proporciona la transmisión y recepción de datos para el SPI. La transmisión de datos es dada al SPI escribiendo en este registro. Los datos recibidos por el SPI pueden ser leídos desde este registro. Cuando es usado como maestro, lo escrito en este registro comenzará una transferencia de datos SPI. Se bloqueará la escritura en este registro cuando comience una transferencia de datos, o cuando el estado de SPIF esté habilitado, por lo que el registro de estado SPI no podrá ser leído.

A continuación se muestra la tabla 2.6 que contiene la descripción de cada uno de los 32 pines asignados para el registro de datos del SPI en la tarjeta LPC1769 y la función que desarrollan al colocar un alto o un bajo.

Bit	Símbolo	Descripción
7:0	DataLow	Puerto bidireccional de datos SPI.
15:8	DataHigh	Si el bit 2 del SPCR es 1 y los bits 11:8 tienen más de 1000, algunos o todos estos bits contienen la transmisión y recepción adicional de bits. Cuando menos de 16 bits son seleccionados, el más significativo de estos bits leídos se lee como cero.
31:16	-	Bits reservados. El usuario del software no puede escribir sobre estos bits.

Tabla 2.6.: Tabla de registro de datos del SPI para la tarjeta LPC1769.

2.4.2.4. Registro Contador de Reloj del SPI – S0SPCCR

Este registro controla la frecuencia del SCK del maestro. El registro indica el número de ciclos de reloj de los periféricos del SPI.

En modo maestro, este registro debe ser un número mayor o igual a 8. La velocidad del SPI0 SCK puede ser calculada como: PCLK_SPI/SPCCR0. El reloj del periférico SPI es determinado por el registro PCLKSEL0 contenido por PCLK_SPI.

En modo esclavo, la velocidad del reloj SPI dada por el maestro no debe exceder de 1/8 del reloj seleccionado en el periférico SPI. El contenido del registro S0SPCCR no es relevante.

A continuación se muestra la tabla 2.7 que contiene la descripción de cada uno de los 32 pines asignados para el registro contador de reloj del SPI en la tarjeta LPC1769 y la función que desarrollan al colocar un alto o un bajo.

Bit	Símbolo	Descripción
7:0	Counter	Ubicación del contador de reloj SPI0
31:8	-	Bits reservados. El usuario del software no puede escribir sobre estos bits.

Tabla 2.7.: Tabla de registro contador de reloj del SPI para la tarjeta LPC1769.

2.4.2.5. Registro de Interrupción del SPI – S0SPINT

A continuación se muestra la tabla 2.8 que contiene la descripción de cada uno de los 32 pines asignados para el registro de interrupción del SPI en la tarjeta LPC1769 y la función que desarrollan al colocar un alto o un bajo.

Bit	Símbolo	Descripción
0	SPIF	Bandera de interrupción del SPI. Habilitada por la interfaz SPI se genera una interrupción. Encerada por escritura de un 1 en este bit. Este bit será habilitado una vez cuando

		SPIE = 1 y al menos SPIF y WCOL estén en 1. Sin embargo, sólo cuando el bit de interrupción del SPI está habilitado y la interrupción SPI0 es habilitada en el NVIC, la interrupción basada en SPI puede ser procesada por manejo de software de interrupción.
7:1	-	Bits reservados. El usuario del software no puede escribir sobre estos bits.
31:8	-	Bits reservados. El usuario del software no puede escribir sobre estos bits.

Tabla 2.8.: Tabla de registro de interrupción del SPI para la tarjeta LPC1769.

2.4.2.6. Registros de prueba del SPI – SPTCR y SPTSR

Adicionalmente, se tienen dos registros de prueba los cuales solamente funcionan como verificación. A continuación se muestran las tablas 2.9. y 2.10. Que contienen la descripción de cada uno de los 32 pines asignados tanto para el registro de control de prueba (SPTCR 0x4002 0010) como para el registro de estado de prueba (SPTSR 0x4002 0014) del SPI en la tarjeta LPC1769 y la función que desarrollan al colocar un alto o un bajo. Ambos registros son una replicación del registro de control o de estado descritos en esta unidad. La diferencia entre ellos es que una lectura de estos registros no comenzará la secuencia de eventos requeridos para limpiar el estado de estos bits ni realizar el control de los mismos.

Bit	Símbolo	Descripción
0	-	Bits reservados. El usuario del software no puede escribir sobre estos bits.
7:1	Test	Modo de prueba SPI. Cuando es 0, el SPI opera normalmente. Cuando es 1, SCK siempre estará habilitado, independientemente de la selección de modo maestro y la ubicación de los datos habilitados.
31:8	-	Bits reservados.

Tabla 2.9.: Tabla de registro de prueba de control del SPI para la tarjeta LPC1769.

Bit	Símbolo	Descripción
2:0	-	Bits reservados.
3	ABRT	Aborto del esclavo.
4	MODF	Modo de falla.
5	ROVR	Sobre lectura.
6	WCOL	Colisión de escritura.
7	SPIF	Bandera de transferencia completa de SPI.
31:8	-	Bits reservados.

Tabla 2.10.: Tabla de registro de prueba de estado del SPI para la tarjeta LPC1769.

2.4.3. Transferencia de datos SPI

De manera similar a la transferencia de datos que se estable mediante protocolo de comunicación SPI en la AVR Butterfly (fabricante ATMega), en la tarjeta LPCXpresso también se realiza la transferencia de datos empleando básicamente el uso de los bits de control SCK y CPHA. A continuación se expone cómo se establece la transferencia para la LPC1769.

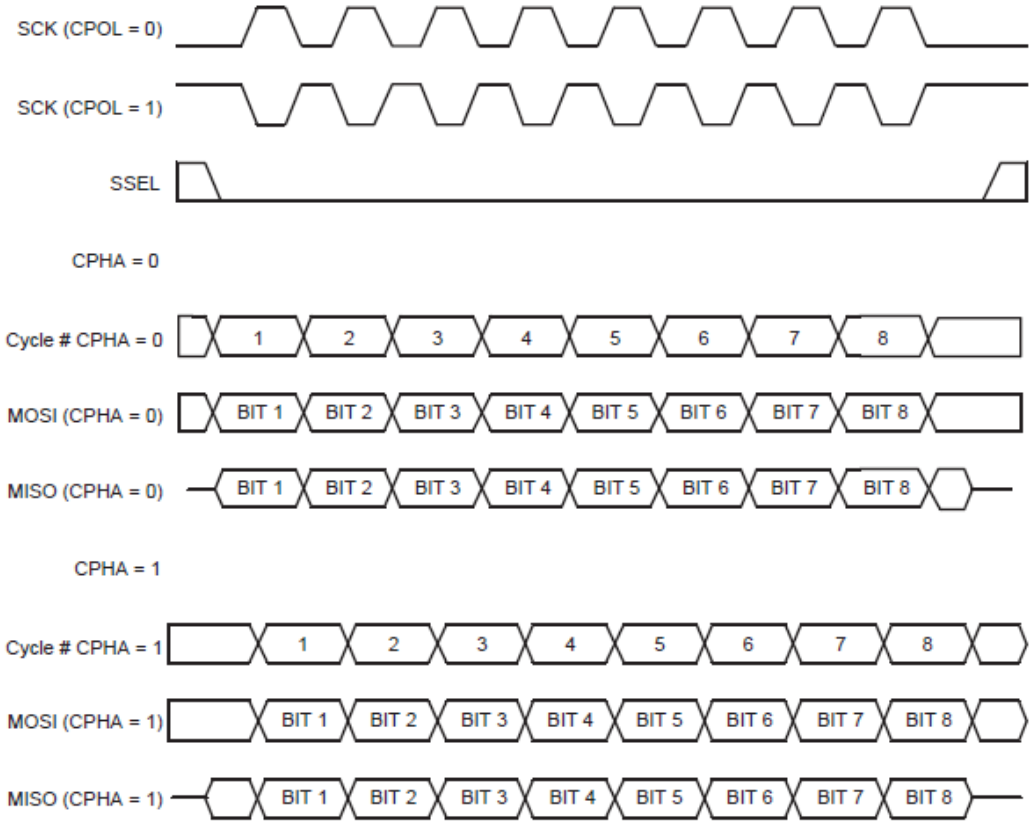


Figura 2.9.: Formato de transferencia de datos SPI cuando CPHA es igual a uno. **[6]**

La figura 2.9 es un diagrama de temporización que ilustra los cuatro formatos diferentes de transferencia de datos que se encuentran disponibles en los puertos del SPI. Este diagrama de tiempos ilustra solamente 8 bits de datos a transferir.

Lo primero que se debe observar en este diagrama de tiempos es que se divide en tres partes horizontales. La primera parte describe las señales SCK y SSEL. La segunda parte describe las señales de MOSI y MISO cuando el bit de control de fase de reloj (CPHA) en el registro de control del SPI es 0.

La tercera parte describe las señales de MOSI y MISO cuando la variable CPHA es 1.

En la primera parte del diagrama de temporización, hay que notar dos puntos. En primer lugar, el SPI se ilustra con el bit de control de polaridad del reloj (CPOL), seteado en 0 o 1. El segundo punto a destacar es la activación y desactivación de la señal SSEL. Cuando CPHA = 0, la señal SSEL siempre estará inactiva en la transferencia de datos. Esto no está garantizado cuando CPHA = 1 (la señal puede permanecer activa).

2.5. Ventajas del SPI

A continuación se exponen las ventajas de utilizar el protocolo SPI para la comunicación entre dispositivos:

- Comunicación Full Dúplex.
- Mayor velocidad de transmisión que con I²C o SMBus.
- Protocolo flexible en que se puede tener un control absoluto sobre los bits transmitidos.
- No está limitado a la transferencia de bloques de 8 bits.
- Elección del tamaño de la trama de bits, de su significado y propósito.
- Su implementación en hardware es extremadamente simple.

- Consume menos energía que I²C o que SMBus debido que posee menos circuitos (incluyendo las resistencias pull-up) y estos son más simples.
- No es necesario arbitraje o mecanismo de respuesta ante fallos.
- Los dispositivos clientes usan el reloj que envía el servidor, no necesitan por tanto su propio reloj.
- No es obligatorio implementar un transceptor (emisor y receptor), un dispositivo conectado puede configurarse para que solo envíe, sólo reciba o ambas cosas a la vez.
- Usa muchos menos terminales en cada chip/conector que una interfaz paralelo equivalente.
- Como mucho una única señal específica para cada cliente (señal SS), las demás señales pueden ser compartidas.

2.6. Desventajas del SPI

A continuación, también se exponen las desventajas de utilizar este protocolo en la transferencia de datos:

- Consume más pines de cada chip que I²C, incluso en la variante de 3 hilos.
- El direccionamiento se hace mediante líneas específicas (señalización fuera de banda) a diferencia de lo que ocurre en I²C

que se selecciona cada chip mediante una dirección de 7 bits que se envía por las mismas líneas del bus.

- No hay control de flujo por hardware.
- No hay señal de asentimiento. El servidor podría estar enviando información sin que estuviese conectado ningún cliente y no se daría cuenta de nada.
- No permite fácilmente tener varios servidores conectados al bus.
- Sólo funciona en las distancias cortas a diferencia de, por ejemplo, RS-232, RS-485, o Bus CAN.

2.7. Motores BLDC con sensores

El uso de motores de corriente continua sin escobillas (BLDC) está aumentando continuamente. La razón se debe a que los motores BLDC tienen un buen peso / tamaño de la ración de alimentación, un rendimiento de aceleración excelente, requiere poco o ningún mantenimiento y genera menos ruido acústico y eléctrico (cepillado) que los motores universales de corriente continua.

En un motor universal de CC (Corriente Continua), los cepillos controlan la conmutación al conectar físicamente las bobinas en el momento correcto. En los motores BLDC la conmutación es controlada por la electrónica. La electrónica puede tener entradas de sensor de posición que proporcionan

información acerca de cuándo conmutar o utilizar la fuerza contra electromotriz generada en las bobinas.

Los sensores de posición son los más utilizados en aplicaciones donde el torque inicial o par de arranque varía en gran medida o donde un alto par de arranque inicial es requerido. Los sensores de posición también se utilizan a menudo en aplicaciones donde se utiliza el motor para el posicionamiento.

Esta nota de aplicación describe el control de un motor BLDC con efecto Hall en los sensores de posición (o simplemente como sensores Hall). La implementación incluye la dirección y el control de velocidad en bucle abierto. Hablaremos sobre esta clase de motores más adelante.

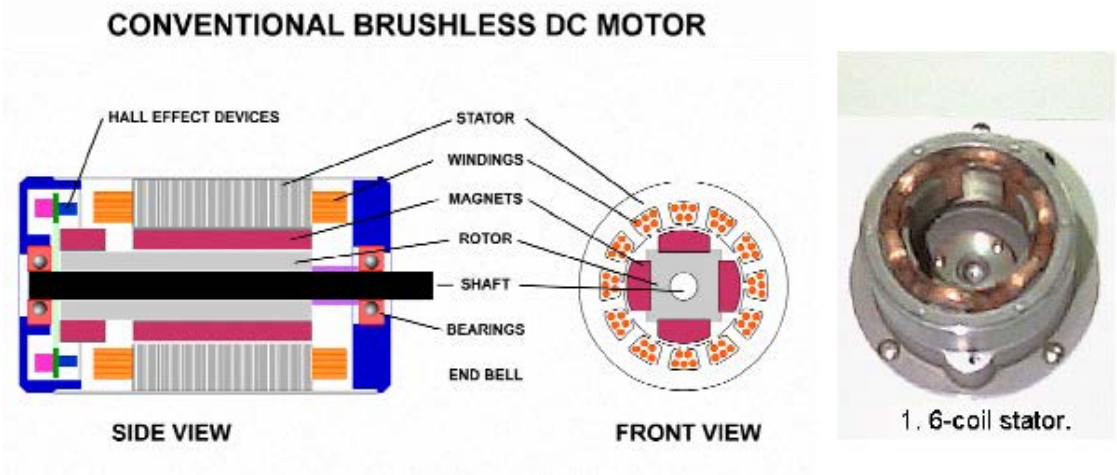


Figura 2.10.: Ejemplo de motor BLDC. **[7]**

El control de un motor BLDC es algo sencillo, y la mayoría de los microcontroladores pueden hacerlo. Sin embargo, en aplicaciones que requieren una alta tasa de revoluciones por minuto, por ejemplo, máquinas

textiles, autos a control remoto y controles industriales, es de alta prioridad el aumento en la frecuencia como la velocidad del motor.

Al aplicar un diseño de control de motores, los ingenieros pueden elegir entre funciones fijas de control de motores CI y microcontroladores. En muchos de los diseños, los ingenieros eligen microcontroladores ya que proporcionan la flexibilidad y la capacidad de integrar una variedad de otras funciones de la interfaz de usuario, como botones, interruptores y pantallas, a las funciones de comunicación, como UART (receptor asíncrono universal / transmisores) y SPI (Serial-periféricos interfaces).

Mediante microcontroladores se permite fácilmente conducir los motores múltiples a diferentes velocidades, allanando el camino para la reducción de gastos de lista de materiales, en aplicaciones que requieren múltiples motores que hoy en día utilizan múltiples microcontroladores dedicados.

Para entender por qué el control de motor BLDC es tan propenso a las interrupciones, hay que considerar cómo un microcontrolador lo controla. Un control electrónico de motor BLDC requiere para energizar las bobinas del estator de una secuencia particular. Para implementar esta secuencia, en la circuitería de control se debe conocer la posición del rotor. La mayoría de los motores contienen múltiples pares de polos en torno a la circunferencia del estator, que requieren múltiples ciclos por rotación. Este requisito se

incrementa la carga sobre el procesador y limita la velocidad máxima de rotación.

En la figura adjunta, se observa la conexión basada en topología de estrella que se mantiene en un motor BLDC básico. La rotación del mismo se determina por medio de códigos formados por los pulsos captados por los sensores que se encuentran en el motor, lo que genera el giro del mismo dependiendo de la polaridad de los imanes que se encuentran en las rutas por donde circula corriente y se produce el campo magnético inducido.

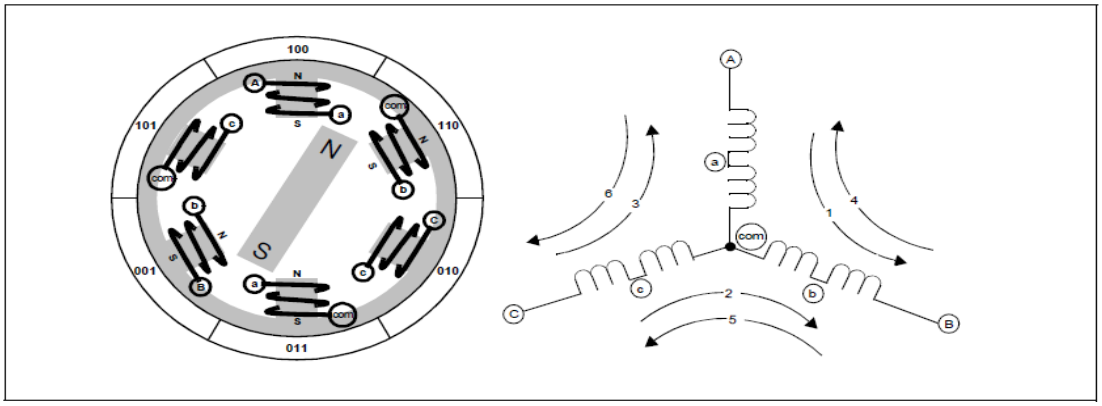


Figura 2.11.: Diagrama circuital de un motor BLDC con topología estrella. **[8]**

En este ejemplo hay tres circuitos electromagnéticos conectados a un punto común. Cada circuito se divide en el centro, permitiendo con ello que el rotor del imán permanente se pueda mover en medio del campo magnético inducido. Los motores con este tipo de topología son impulsados por la activación de 2 fases a la vez, mientras que la tercera fase se mantiene apagada o sin paso de corriente.

La alineación estática se muestra en la siguiente figura, la misma que se llevaría a cabo mediante la creación de un flujo de corriente eléctrica de la terminal A a la B, la misma que está señalada como paso 1 en la figura anterior.

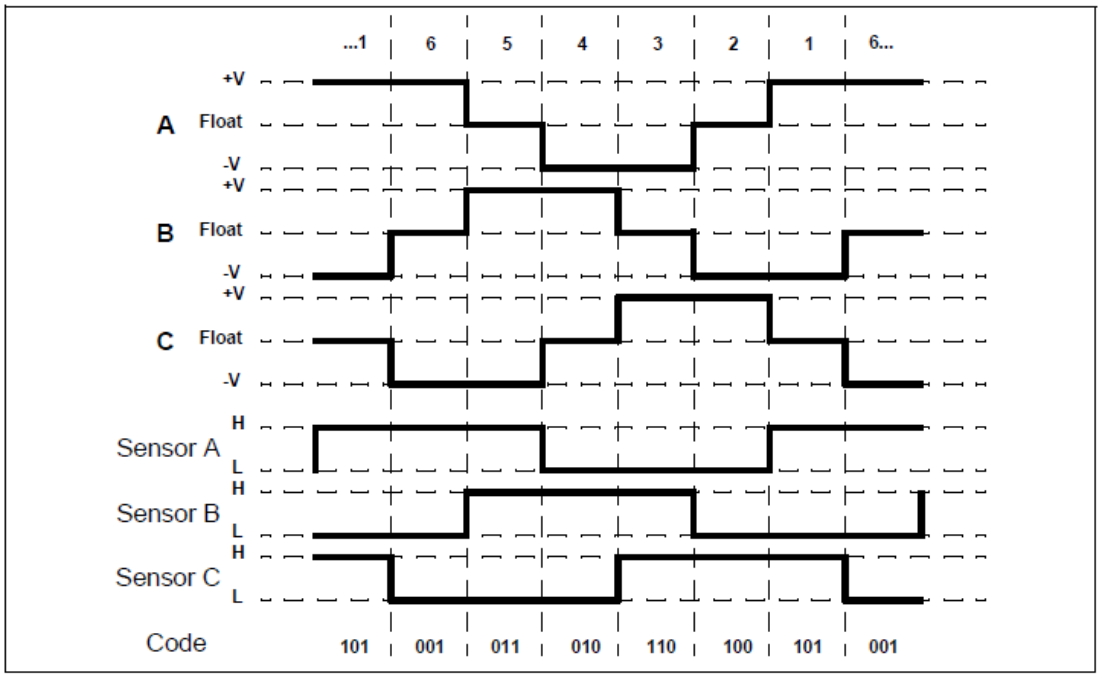


Figura 2.12.: Representación de las señales detectadas tanto a nivel de los sensores del motor como en las terminales del mismo, las que se observan en la conexión estrella en la Fig. 2.11 **[9]**

El rotor puede hacerse girar en sentido horario de 60 grados desde A hacia B. Al cambiar la ruta de la corriente, es decir que ahora fluya desde la terminal C a B, se observa que el control en A se mantiene en cero. Este esquema se mantiene en condiciones teóricas. En la práctica, el par máximo se obtiene cuando el permanente magnético del rotor neto es de 90 grados de distancia de la alineación con el estator.

La clave para la conmutación BLDC es detectar la posición del rotor, a continuación, se deben energizar las fases que producen la mayor cantidad de par motor. El rotor viaja 60 grados por paso de conmutación. El camino de corriente apropiado para el estator se activa cuando el rotor es de 120 grados a partir de la alineación con el estator magnético correspondiente de campo, y luego desactivados cuando el rotor es de 60 grados de alineación, momento en el cual el circuito siguiente se activa y se repite el proceso.

En la práctica, los motores BLDC tienen más de uno de los circuitos eléctricos que se muestran, conectados en paralelo a cada otro, y un permanente correspondiente multipolar del rotor magnético. Durante dos circuitos eléctricos hay dos revoluciones por revolución mecánica, por lo que para un circuito cada fase de conmutación eléctrica cubriría 30 grados de rotación mecánica.

En la siguiente tabla se adjuntan los resultados obtenidos mediante los pasos de la Figura 2.12.

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 2.11.: Tabla de resultados de los High y Low Drive detectados en las terminales del circuito del motor BLDC. **[10]**

En la tabla 2.11, se observa los resultados obtenidos mediante las gráficas de la Fig. 2.12. Tomando como ejemplo el paso 1, teníamos en la Fig. 2.11. que la corriente viaja desde A hacia B, produciendo un campo magnético el cual es inducido por las bobinas en este ramal de la conexión y se observa que, como toda la corriente viaja por ese ramal, la corriente que va hacia C es cero, por lo que esta terminal se encuentra en 0. El código detectado para este estado de la conexión es el 101, que se detalla en Sensor C, B y A. Como C se encuentra en cero, observamos que el resultado en High y Low Drive se mantiene en cero, en cambio en B como observamos que la polaridad que mantiene es negativa se observa un 1 en Low Drive y un 0 en High Drive, por consiguiente en A observamos que como tenemos una caída de potencial positivo que va hacia B el High Drive se mantiene en 1 y el Low Drive se mantiene en 0. Esta es la manera en que se interpreta la tabla descrita para los cambios producidos por inducción electromagnética dentro de los motores BLDC con sensor.

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
6	0	0	1	0	1	0	0	1	0
4	0	1	0	0	0	1	0	0	1
5	0	1	1	0	1	1	0	0	0
2	1	0	0	1	0	0	1	0	0
1	1	0	1	0	0	0	1	1	0
3	1	1	0	1	0	0	0	0	1

Tabla 2.12.: Tabla de resultados de los High y Low Drive detectados en las terminales del circuito del motor BLDC, ordenadas por la ubicación del sensor. **[11]**

En la tabla 2.12, se observa que se encuentran los mismos datos obtenidos en la tabla 2.11, solo que se encuentran ordenados de acuerdo al valor de código que se refleja por medio de los sensores A,B y C.

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
/6	0	0	1	1	0	0	0	0	1
/4	0	1	0	0	0	0	1	1	0
/5	0	1	1	1	0	0	1	0	0
/2	1	0	0	0	1	1	0	0	0
/1	1	0	1	0	0	1	0	0	1
/3	1	1	0	0	1	0	0	1	0

Tabla 2.13.: Tabla de resultados de los High y Low Drive detectados en las terminales del circuito del motor BLDC, con los valores de High y Low Drive invertidos. **[12]**

En la tabla 2.13, se observa nuevamente ordenados los cambios en el motor tomando como referencia los códigos organizados basados en los resultados obtenidos en los Sensores A,B y C. La diferencia es que en esta tabla se observa que los valores que se encontraban con un 1 o pulso alto en High Drive ahora tendrán el 1 o pulso alto en Low Drive y viceversa. Las terminales del circuito que se mantenían sin paso de corriente o con valor cero o bajo tanto en High como en Low Drive se mantienen en el valor de 0.

2.7.1. Características

- Menos de 5us de tiempo de respuesta en el cambio de salida del sensor.

- Máximo (teórico) de 1600k RPM.
- Apoyo para la regulación de bucle cerrado.
- UART, TWI y SPI disponible para la comunicación.
- No emplean escobillas en la conmutación para la transferencia de energía, por lo que realizan la conmutación electrónicamente
- Utiliza conmutación trapezoidal y conmutación sinusoidal para control de velocidad.

2.7.2. Ventajas

- Mejor relación velocidad – par motor.
- Mayor respuesta dinámica.
- Mayor eficiencia.
- Mayor vida útil.
- Menor ruido.
- Mayor rango de velocidad.
- Requieren menor mantenimiento ya que, al no usar escobillas, minimizan el rozamiento y producción de calor en el motor.
- Mejor rendimiento energético debido a que no se tiene caída de tensión ni pérdida de energía entre las escobillas y las delgas.
- Mayor fiabilidad de duración.
- Bajo costo de fabricación.
- Bajos niveles de vibración mecánica.

- Alta capacidad de carga (depende de la capacidad actual del suministro de voltaje).
- Pérdidas menores, menor generación de calor (importante dentro de dispositivos encapsulados).
- Posible operación de control de velocidad cerrada, velocidad constante.
- Puede operar bidireccionalmente.

2.7.3. Desventajas

- Mayor costo en comparación con otros motores DC.
- Requieren un control bastante más complejo.
- Se requiere agregar suministro de potencia DC para aplicaciones donde se requiere mayor arranque o torque del motor.

2.7.4. Motor sin escobillas (BLDC)

BLDC (corriente continua sin escobillas) han reemplazado a los motores de otros motores en aplicaciones que van desde equipos de aire acondicionado para coches teledirigidos, ofreciendo ventajas en la eficiencia, fiabilidad y rendimiento. El costo de los motores BLDC ha disminuido drásticamente en los últimos 10 años, haciendo que sean más accesibles.

Esta clase de motores de corriente continua sin escobillas son uno de los tipos de motores más conocidos en los últimos años debido a su gran utilidad en múltiples áreas. En la actualidad, los motores BLDC se emplean en campos como el aeroespacial, industrial, médico, automovilístico, y es debido a los grandes beneficios y potencia de estos motores los cuales son empleados mayoritariamente en el campo industrial siendo más eficientes que los motores sensorless.

Este tipo de motores carecen de colector y, en vez de funcionar en DC funcionan en AC. La mayoría se alimentan por medio de señales trifásicas las mismas que idealmente deben ser señales sinusoidales, pero que en la práctica se observan como pulsos o trapezoides, haciendo que la señal se observe como pulsos continuos o una señal continua con mucho componente de AC, sin embargo son catalogados como motores de funcionamiento DC porque, al igual que los motores universales, contienen imanes permanentes.

Estos imanes son atraídos por la polaridad de un campo magnético generado en las bobinas, las cuales, como se mencionaba anteriormente en este documento, reciben pulsos basados en un patrón específico de códigos. Si deseamos conseguir que el motor gire a mayor velocidad, simplemente aumentaríamos la frecuencia de los pulsos, lo que generaría que el campo magnético gire a mayor velocidad.

En el motor existen tres circuitos electromagnéticos conectados en un punto común, lo que se conoce como un circuito en conexión estrella, que es la topología establecida en la mayoría de los motores BLDC. Esto permite que el imán permanente del rotor se mueva por medio del campo magnético inducido.

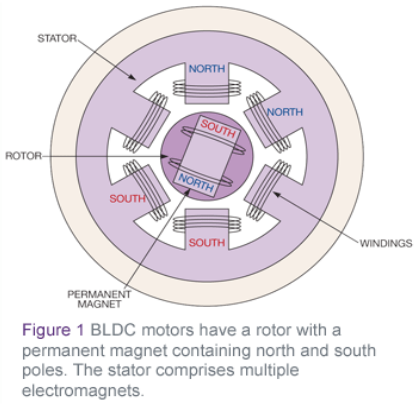


Figure 1 BLDC motors have a rotor with a permanent magnet containing north and south poles. The stator comprises multiple electromagnets.

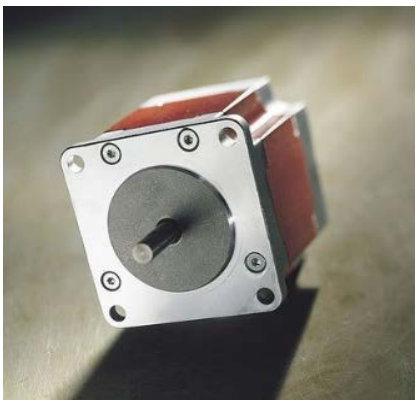


Figura 2.13.: Ejemplo de motor sin escobillas. **[13]**

2.7.5. Sensor de efecto Hall

Otros sistemas controlados por sensores sistemas utilizan sensores como los de efecto Hall integrados en el estator del motor, los cuales funcionan en base a campos electromagnéticos (fuerza electromotriz). Con los sistemas de efecto Hall basados en sensores, los polos magnéticos del rotor giran cerca de los sensores de efecto Hall, como resultado del suministro de una alta o una baja señal, lo que indica que el norte o el sur de los polos pasan cerca. La combinación exacta de las tres señales de los sensores Hall nos indica la

posición del rotor. Sucesivamente se energizan las bobinas del estator, manteniendo el giro del motor.

Los sensores de efecto Hall se utilizan para la medición de campos magnéticos que generan velocidades de rotación en el motor o para detectar la posición de un determinado elemento. La principal ventaja de estos es que pueden ofrecer datos fiables a cualquier velocidad de rotación, mientras que sus inconvenientes son su mayor complejidad y costo con respecto a sensores inductivos.

El modo de funcionamiento de los motores que utilizan sensores con efecto Hall se puede resumir de la siguiente manera: si fluye corriente por un sensor Hall y se aproxima a un campo magnético que fluye en dirección vertical al sensor, entonces el sensor crea un voltaje saliente proporcional al producto de la fuerza del campo magnético y de la corriente. Si se conoce el valor de la corriente, entonces se puede calcular la fuerza del campo magnético; si se crea el campo magnético por medio de corriente que circula por una bobina o un conductor, entonces se puede medir el valor de la corriente en el conductor o bobina. Si tanto la fuerza del campo magnético como la corriente son conocidas, entonces se puede usar el sensor Hall como detector de metales.

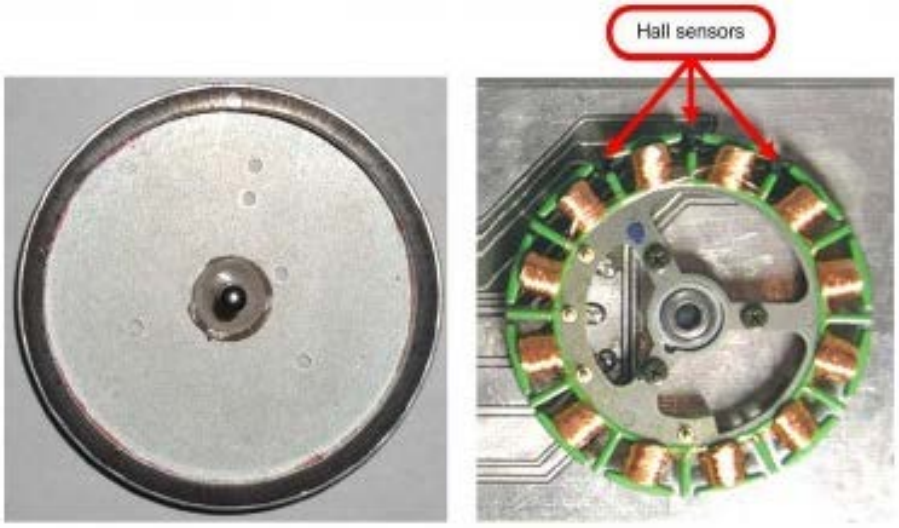


Figura 2.14.: Ejemplo de motor con sensor con efecto Hall. **[14]**

Un microcontrolador hace girar el motor BLDC por habilitar y deshabilitar dispositivos externos de potencia que proporcionan corriente a través de los bobinados del estator en la secuencia requerida. Se requiere una conmutación eficaz que energiza el momento de la corriente de arrollamiento en secuencia con la rotación del motor. Para el control de motores BLDC, el cambio de un sensor Hall de estado genera una interrupción, lo que indica que el motor ha girado en el estado de conmutación siguiente. Cuanto más rápido el motor está girando, mayor es la frecuencia que va a producir. Las interrupciones del estado de conmutación deben ser de alta prioridad y tener un servicio rápido para garantizar una rotación suave y para mantener el control de la velocidad del motor.

2.7.6. Diferencia entre Motor BLDC y Motor con Escobillas

A continuación se muestra mediante una tabla las diferencias entre los motores BLDC y los motores con escobillas, exponiendo los pro y contra de su aplicación.

	Motor BLDC	**Motor con escobillas**
Conmutación	Conmutación electrónica basada en sensores de posición de efecto Hall	Conmutación 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 2.14.: Tabla de comparación entre motor BLDC y motor con escobillas.

2.8. Herramientas de software

Las herramientas de software permiten el desarrollo de diferentes programas informáticos mediante el uso de lenguaje de programación u opciones establecidas en la herramienta a utilizar, ayudando a la implementación y diseño de proyectos.

Para la elaboración de este proyecto empleamos básicamente dos herramientas de software: AVR Studio 4, del fabricante ATMEL, para la programación del microcontrolador ATMega169 y el LPCXpresso 4, del fabricante LPCXpresso, para realizar la programación de la tarjeta LPC1769. Adicionalmente hicimos uso de la herramienta PROTEUS 7.7 Service Pack 2 para la simulación del proyecto.

En esta sección se dará una breve descripción de ambas herramientas de software para familiarizar al lector con estas interfaces de trabajo ya que son la base para la programación tanto del AVR Butterfly como del LPC1769.

2.8.1. AVR Studio 4

El AVR Studio 4 es una herramienta de software, perteneciente al fabricante ATMEL, la cual nos permite el desarrollo de proyectos que emplean el uso de la tarjeta AVR Butterfly, la misma que es programable mediante este software.

Es un entorno de desarrollo integrado (IDE), el cual nos permite escribir y depurar aplicaciones AVR en entornos Windows, proporcionando al programador una gestión de proyectos, herramientas, fuente de editor de archivos, simulador, ensamblador y front-end para C/C++, emulación en el chip, depuración, entre otras cosas lo que la convierte en una herramienta completa para la programación del ATMega169.

AVR Studio es una herramienta compatible con todas las herramientas de software de ATMEL AVR y cada versión contiene las últimas actualizaciones para las herramientas y el apoyo de los dispositivos AVR. AVR Studio 4 tiene una arquitectura modular que permite la interacción con herramientas de otros proveedores.

Figura 2.15.: Logo de la herramienta de software AVR Studio 4.

2.8.2. LPCXpresso 4

Esta herramienta de software, basada en 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 se basa en un grupo simplificado con muchas mejoras específicas, también cuenta con la versión actual de la cadena de herramientas GNU estándar de la industria con una biblioteca propia C optimizada.

El IDE del LPCXpresso puede construir un ejecutable de cualquier tamaño con la optimización de código completo y es compatible con un límite de descarga de 128kbps después del registro.

Diseñado para la simplicidad y facilidad de uso, el LPCXpresso IDE (alimentado por Red Code), proporciona al programador la facilidad y rapidez requerida para el desarrollo de aplicaciones.

Figura 2.16.: Logo de la herramienta de software LPCXpresso 4.

LPCXpresso proporciona un entorno programación en C de primera, con el formato de la sintaxis de color, origen, función de plegado y una amplia automatización de la gestión de proyectos. Las características incluyen: asistentes para crear proyectos para todos los microcontroladores soportados, generación automática de secuencia de comandos del vinculador incluyendo soporte para los mapas de memoria del microcontrolador, descarga directa cuando se depura, y apoyo para los microcontroladores de la familia NXP LPC, incluyendo otros más como Cortex-Mx y ARMx.

2.8.3. PROTEUS 7.7

PROTEUS es un programa que nos permite simular circuitos electrónicos complejos, integrando inclusive desarrollos realizados con microcontroladores de varios tipos, lo que lo convierte en una herramienta de alto desempeño con unas capacidades gráficas impresionantes.

PROTEUS en realidad es una herramienta que se divide en dos programas: ISIS y ARES. Con el primero tendremos un generador de circuitos reales, que funcionan de forma que podremos comprobar si el diseño que queremos implementar funcionará. Una vez comprobado y testeado con las herramientas incorporadas, ARES pasará a la acción de transformar el diseño virtual en algo que podamos implementar en un modelo real con todos los componentes que fueron colocamos en la primera parte y cuya simulación resultó exitosa.

PROTEUS incluye la función de generar pistas de cobre automáticamente si se le indica, simplemente, como van conectadas, aunque seguramente deberemos retocar algunos detalles en el diseño realizado por el ARES ya que este tipo de automatismos no siempre son precisos al momento de la colocación de pistas ya que las mismas suelen crearse sobrepuestas.

Figura 2.17.: Logo de la herramienta de software PROTEUS 7.7

2.9. Herramientas de hardware

Una herramienta de hardware se define como la parte tangible de un sistema. Esta herramienta es la parte física que contiene el software, lo que permite la interacción física con el usuario. El hardware está conformado por periféricos. Para la implementación de este proyecto empleamos diversos periféricos entre los cuales podemos destacar la computadora, el uso de dispositivos electrónicos como resistores, leds, displays, cableado, el motor BLDC y los microcontroladores.

En esta sección se menciona con detalle el uso de las dos herramientas de hardware más importantes para la implementación de este proyecto: la tarjeta AVR Butterfly de ATMega y la tarjeta LPC1769 de LPCXpresso.

2.9.1. AVR Butterfly

Pese a su reducido tamaño, la tarjeta AVR Butterfly tiene una gran capacidad de procesamiento de órdenes e información, un bajo consumo de energía y una gran capacidad de almacenamiento de datos.

El AVR Butterfly está compuesto, principalmente, por el microcontrolador ATMega169 y dispositivos periféricos como el joystick de control y puertos de comunicación con dispositivos externos. Entre las características que lo conforman tenemos que el encapsulado con el que trabaja es de tipo MLF (Micro Lead Frame); como periféricos utiliza el controlador de LDC, memorias flash, EEPROM, SRAM y memoria DataFlash externa. Esta tarjeta puede trabajar como Convertidor Analógico – Digital (ADC), contador en tiempo real (RTC) y como una herramienta de modulación de ancho de pulso (PWM).

El AVR Butterfly puede trabajar haciendo uso de protocolos de comunicación como SPI, UART e I2C. Esta tarjeta utiliza métodos de programación como Self-Programming/Bootloader, ISP, paralelo y JTAG.

Figura 2.18.: Vista de doble cara de la tarjeta AVR Butterfly, desarrollada por el fabricante ATMega.

2.9.1.1. Esquema funcional de la AVR Butterfly

En la figura 2.17 se muestra un esquema simplificado de los periféricos con los que cuenta la tarjeta AVR Butterfly y los componentes que conforman el microcontrolador ATMega169.

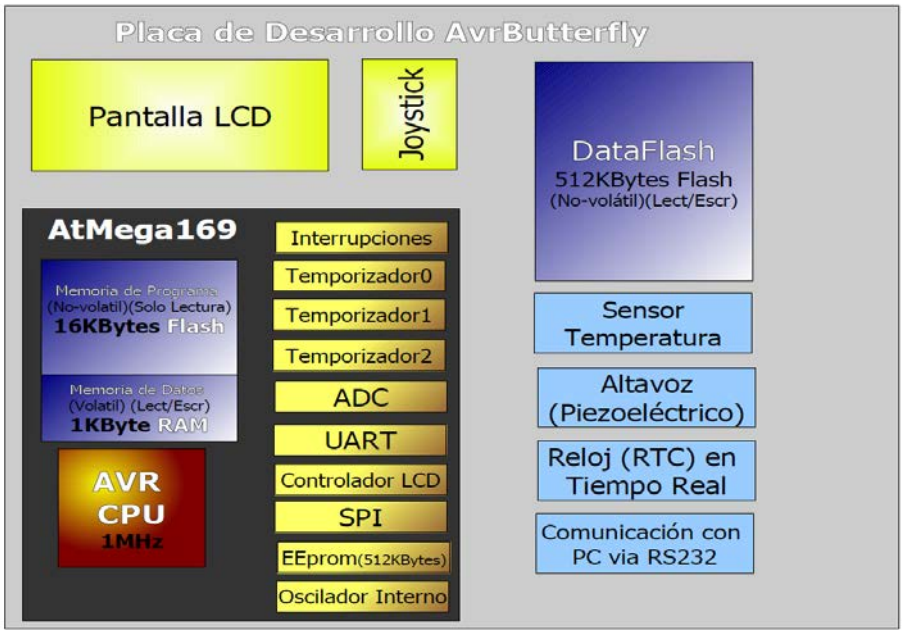


Figura 2.19.: Esquema simplificado de los periféricos internos y externos que conforman la tarjeta AVR Butterfly **[15]**

En la figura 2.17 se puede observar que la AVR Butterfly cuenta con una serie de periféricos internos y otros externos, para cada uno de ellos se requiere realizar unas rutinas de inicialización y otras dependiendo del uso que se les quiere dar.

En algunos casos va a ser necesario utilizar un periférico interno para poder utilizar un periférico externo, como para los casos que se describen a continuación:

Para utilizar el altavoz (piezoeléctrico) y generar un tono de sonido a una determinada frecuencia es necesario utilizar el Temporizador 1 como generador PWM.

El sensor de temperatura, una vez alimentado, genera una señal analógica que hay que capturar con el convertidor analógico digital (ADC) previamente configurado para uso del sensor.

Para conseguir la comunicación entre el microcontrolador y un ordenador PC con un puerto RS-232 es necesario configurar la UART y el reloj interno del sistema para conseguir la tasa de transferencia deseada para la comunicación, la cual no podría llevarse a cabo sin la circuitería elevadora de tensión de 3V a 12V con la que dispone la tarjeta AVR Butterfly.

Para tener una referencia temporal del Reloj en Tiempo Real (RTC) es necesario configurar el Temporizador 2, y utilizarlo con frecuencia para que genere interrupciones con la señal de tiempo real.

Para poder almacenar y restaurar datos del dispositivo de memoria DataFlash es necesario comunicarlo con el microcontrolador ATMega169 a través del periférico de comunicación Serial Peripheral Interface (SPI).

Para el correcto funcionamiento de la pantalla LCD, es necesario configurar el periférico Driver LCD, que genera las señales adecuadas para cada segmento LCD de dicha pantalla.

Para poder darle funcionalidad a la interfaz de entrada con el usuario (joystick), es necesario habilitar el periférico gestor de interrupciones, para que cada vez que se toque un botón se acceda a una rutina de interrupción asociada que la resuelva satisfactoriamente, esta aplicación nos será de gran utilidad para el manejo del motor BLDC.

Un esquema más detallado de la distribución de los periféricos internos sería el de la figura 2.18, en donde podemos ver que, además de los ya mencionados, se encuentran más periféricos, pero que no se pueden usar debido a que en la AVR Butterfly algunos de ellos tienen los pines de salida utilizados para otras funcionalidades, y por ello estos no se pueden emplear, como es el caso del comparador analógico (cuya salida está siendo utilizada por una parte del LCD). La distribución más relevante de los periféricos externos y de los puertos del microcontrolador se muestra en la siguiente figura.

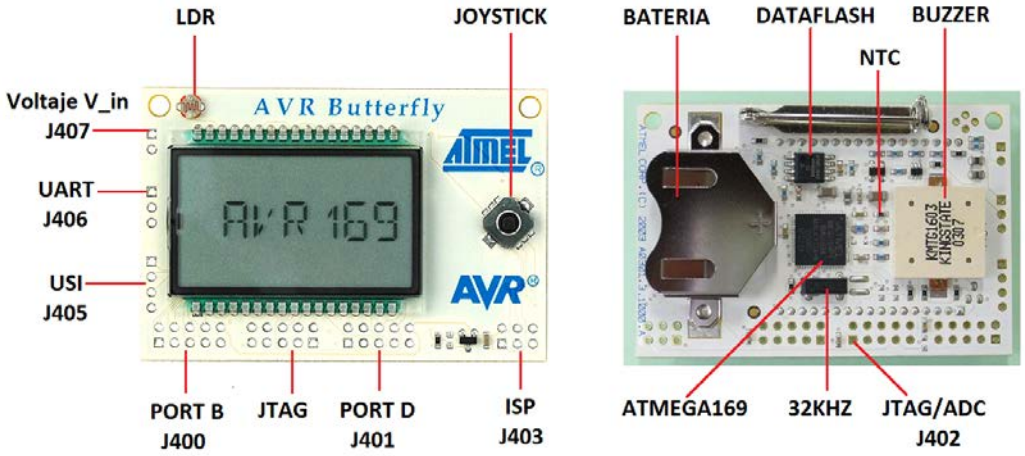


Figura 2.20.: Distribución de los periféricos externos en la tarjeta AVR Butterfly.

Entre los periféricos más importantes podemos destacar el NTC que es la resistencia negativa de temperatura o sensor de temperatura de la tarjeta, el LDR que es la resistencia dependiente de la intensidad de la luz, el Joystick de cinco posiciones de las cuales utilizaremos cuatro para manejo del motor BLDC, el oscilador de cuarzo de 32KHz para la generación de la onda del reloj en tiempo real, el buzzer (piezoeléctrico) o altavoz para la emisión de sonidos de un solo tono, la memoria DataFlash de 512Kbytes de capacidad de almacenamiento y el periférico de comunicación UART, cuyos pines de salida están directamente conectados al puerto RS232 del ordenador.

A continuación se muestra un diagrama esquemático con mayor detalle sobre las conexiones de los periféricos con el microcontrolador ATMega169.

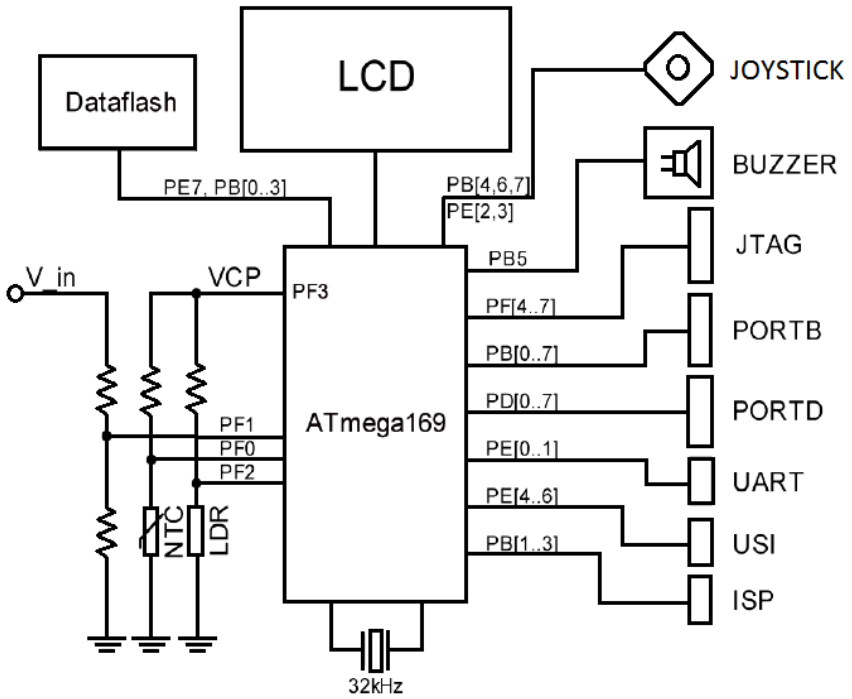


Figura 2.21.: Distribución de los periféricos externos en la tarjeta AVR Butterfly **[16]**

Como se puede observar, en la figura 2.19 se encuentra detallada la conexión de cada periférico hacia cada pin o pines del microcontrolador ATMega169.

En la figura 2.20 se ilustra la configuración interna del microcontrolador ATMega169. En esta gráfica se mostrará la conectividad entre los periféricos que posee el microcontrolador interno de la AVR Butterfly y la estructura interna del mismo, podemos observar con mayor claridad con qué puertos de E/S está relacionado cada periférico.

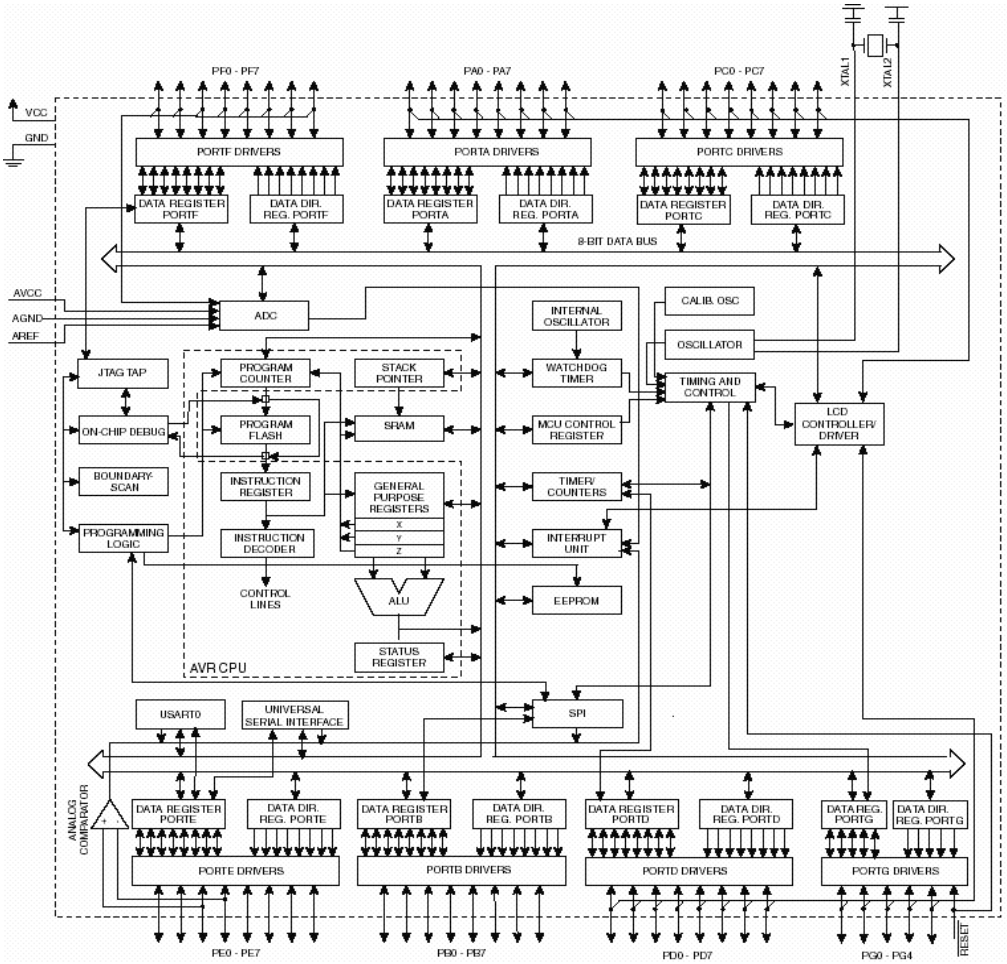


Figura 2.22.: Diagrama de bloques de la configuración interna del microcontrolador ATMega169 **[17]**

2.9.1.2. Capacidad de Almacenamiento de la AVR Butterfly

La tarjeta AVR Butterfly cuenta con dos circuitos integrados con capacidad de almacenamiento de datos. Estos dispositivos son: el microcontrolador ATMega169 y la memoria DataFlash. El microcontrolador ATMega169 cuenta, a su vez, con tres memorias integradas en su encapsulado: Una memoria flash auto-programable de 16Kbytes, una memoria SRAM de 1Kbyte y una memoria EEPROM de 512Kbytes.

La memoria no volátil Flash auto-programable de 16Kbytes tiene su capacidad destinada al almacenamiento del código de la aplicación que vayamos a implementar y de las constantes y cadenas de caracteres. Esta memoria hace las veces de memoria ROM ya que solamente puede escribirse en ella cuando el programa no está funcionando (debido a que lo que se almacena en ella es el código fuente del programa, el mismo que se carga mediante un programa cargador de arranque llamado Bootloader, el cual no se puede eliminar). Cabe mencionar que esta memoria es de solo lectura.

La memoria volátil SRAM de 1Kbyte y la memoria no volátil EEPROM de 512Kbytes tienen capacidad de almacenamiento para otras ejecuciones del sistema.

Debido a que las instrucciones de la tarjeta AVR Butterfly tienen una longitud de 16 y 32 bits, la memoria Flash del programa se organiza en dos bancos de 8Kbytes (uno de 8K y otro de 16K). Adicionalmente podemos decir que, por motivos de seguridad, el espacio de memoria del programa se divide en dos secciones, la de programa de arranque (boot) y la de programa de aplicación.

Además de este tipo de memorias internas, la tarjeta AVR Butterfly cuenta con un periférico de almacenamiento que es una memoria DataFlash de 512Kbytes de capacidad (4Mbits). Este periférico se puede describir como una memoria flash de acceso secuencial con una interfaz serial de comunicación SPI, lo que lo hace compatible con una variedad de dispositivos y aplicaciones de almacenamiento de datos, y que gracias a su doble buffer lo hace apto para aplicaciones que requieren altas tasas de velocidad en la transferencia de datos de entrada y salida, por ejemplo para aplicaciones de voz e imagen digital.

Sus 4.325.376 bits de memoria se organizan en 2048 páginas de memoria de 264 bytes cada una, además de la memoria principal, contiene dos buffers SRAM de datos, cada uno de ellos de 264bytes, gracias a los cuales es posible recibir datos al mismo tiempo que se reescribe una página en la memoria principal de la memoria DataFlash. A esta memoria se accede mediante una interfaz serie SPI de 5 líneas, que facilita el diseño en hardware e incrementa la fiabilidad del sistema, minimiza el ruido de

conmutación y reduce el tamaño del encapsulado. Además este dispositivo requiere simplemente de una fuente de 2.5 V a 3.6 V tanto para lectura como para escritura. Sin estas condiciones de voltaje, el microcontrolador puede funcionar pero la memoria DataFlash puede que no funcione adecuadamente.

2.9.1.3. Características de la AVR Butterfly

Entre las características de la tarjeta AVR Butterfly podemos citar las siguientes: tiene integrado el microcontrolador ATMega169 con encapsulado tipo MLF, posee una pantalla LCD de 120 segmentos la cual se utiliza en conjunto con el joystick de 5 direcciones para mostrar las capacidades de control del microcontrolador ATMega169, tiene un altavoz piezoeléctrico, el cual se utiliza para reproducir sonidos, tiene un cristal de cuarzo de 32KHz para el RTC, una memoria DataFlash de 4 Mb para el almacenamiento de datos, un convertidor de nivel RS-232 para establecer comunicación entre unidades fuera del kit sin necesidad de herramientas de hardware adicionales, un periférico NTC o termistor de coeficiente de temperatura que nos ayuda a medir y sensar temperatura, una resistencia dependiente de la luz o LDR que nos permite sensar y medir intensidad luminosa, emulación del JTAG para la depuración, una interfaz USI que nos sirve como interfaz adicional de comunicación, terminales externas para conectores tipo Header que nos permiten el acceso a periféricos, una batería de 3V que proporciona

energía para el funcionamiento de la tarjeta AVR Butterfly, el Bootloader que nos sirve para la programación mediante la PC y compatibilidad con la herramienta de software AVR Studio 4.

Adicionalmente, podemos destacar que la tarjeta AVR Butterfly trabaja con 16KBytes de sistema programable, memoria Flash de lectura y escritura, 512bytes de EEPROM, 1K de memoria SRAM y 54 registros de propósito general de los cuales 32 registros nos sirven para propósitos generales de trabajo.

2.9.2. Tarjeta LPCXpresso

El LPCXpresso es un toolchain completo que nos permite evaluar y desarrollar sobre microcontroladores de NXP, como por ejemplo el LPC1769, el cual es objeto de la presente unidad. Está compuesto por un IDE LPCXpresso (basado en Eclipse) y herramientas de desarrollo. Además cuenta con un compilador y linkeador GNU, un debugger GDB y una target board de LPCXpresso, la cual trabaja como una BaseBoard o hardware adicional del sistema.

El target board es un microcontrolador con todo lo necesario para encendido del sistema y también cuenta con una herramienta que tiene integrados un programador y un debugger. En éste se encuentra ensamblado el microcontrolador LPC1769, el cual es un sistema ARM Cortx-M3, con una

memoria Flash de 512Kb y 64Kb de memoria SRAM, puertos Ethernet y USB y que trabaja en modo On the Go.

A continuación se muestra una imagen de la tarjeta LPCXpresso, con el detalle de los componentes básicos que la conforman.

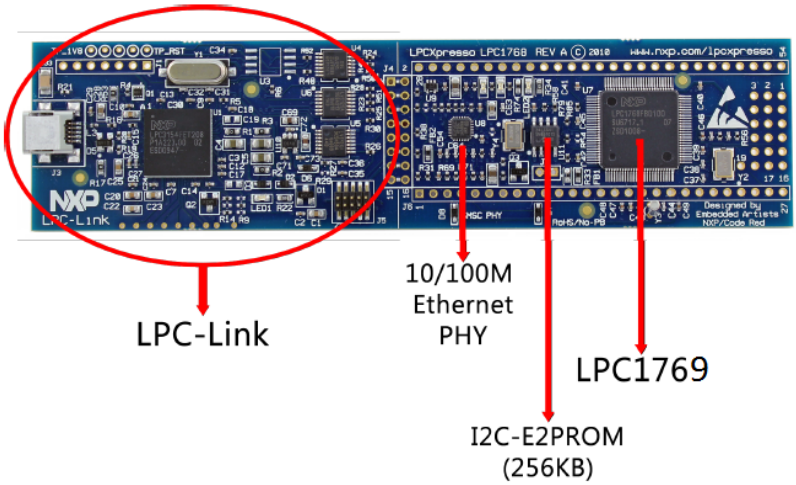


Figura 2.23.: Componentes básicos del kit de la tarjeta LCPXpresso **[18]**

2.9.2.1. Periféricos de la tarjeta LPCXpresso

La tarjeta LPCXpresso contiene periféricos entre los cuales podemos destacar los siguientes:

Entre los componentes periféricos generales podemos mencionar el socket para LPCXpresso y módulo MBED, 50 pines de expansión dual, el encendido de la batería y una interfaz USB.

Entre los componentes periféricos digitales encontramos el RGB-LED que puede ser controlado por PWM, 2 pulsadores de los cuales uno sirve para la activación del gestor de arranque, un interruptor giratorio con la codificación de cuadratura y un sensor de temperatura con salida PWM.

Entre los periféricos analógicos podemos mencionar un potenciómetro de ajuste de entrada analógica, un PWM de entrada y salida analógica y la salida PWM de los altavoces.

Entre los periféricos serie y UART podemos encontrar puerto USB con cambio a activación automática ISP, una interfaz RS232 y un socket para módulo RF.

2.9.2.2. Características de la tarjeta LPCXpresso

Entre las principales características y beneficios que ofrece la tarjeta LPCXpresso con el microcontrolador integrado LPC1769 podemos citar las siguientes: pertenecen a ARM Cortex-M3, que funciona a frecuencias de hasta 100MHz, en el caso de que trabaje con el micro LPC1769 trabaja con frecuencias de hasta 120MHz, tiene una unidad de protección de memoria, tiene el ARM Cortex-M3 integrado en el controlador de interrupciones anidadas vectoriales (CNTV), la memoria flash tiene hasta 512KB de capacidad de almacenamiento, por ende esto indica mayor rapidez de la memoria flash permitiendo una alta velocidad de operación de 120MHz con

cero estados de espera, En SPI e IAP trabaja a través de un chip gestor de arranque del software.

El chip SRAM incluye 16 o 32KB de memoria SRAM con un bus de datos de alto rendimiento. Estos bloques SRAM pueden ser utilizados para Ethernet, USB y memoria de DMA, así como para instrucciones de propósito general y almacenamiento de datos.

Emplea 8 canales de uso general DMA (GPDMA) en el AHB multicapa que se puede utilizar con protocolos de comunicación SSP, I2C, bus, UART, ADC y para transferencias de memoria a memoria.

Ofrece una matriz de conexión para cada maestro del AHB, estos pueden ser utilizados para uso general del controlador DMA, MAC Ethernet y la interfaz USB. Esta interconexión permite la comunicación sin retrasos.

Tiene interfaces seriales Ethernet MAC, RMII y un controlador dedicado DMA, tiene un USB 2.0 de alta velocidad, 4 entradas UART para la generación de fracción de velocidad de transmisión, FIFO interna y DMA. Una UART tiene el control de las E/S del módem.

Posee controlador CAN 2.0B para manejo de dos canales, control sincrónico SPI, comunicación serial full dúplex y una longitud de datos programable, además de poseer dos controladores SSP con capacidades de FIFO y multiprotocolo. Las interfaces del SSP pueden utilizarse con el controlador

GPDMA. Tiene tres interfaces mejoradas para bus de datos para uso de protocolo I2C. La interfaz de este bus I2C se puede utilizar con la GPDMA, la misma que es compatible con los datos de 3 y 4 hilos de transmisión y recepción, así como con el reloj del maestro SCK de entrada y salida.

Entre otros periféricos que podemos mencionar se encuentran de 70 a 100 paquetes de pines de uso general de E/S del GPIO con pines configurables para resistencias pull-up y pull-down. Al bloque de GPIO se accede a través del bus AHB multicapa para un acceso rápido y situado en la memoria de tal manera que soporta equipos bajo el estándar Cortex-M3. Además incluye 12 bits de ADC (Convertidor de Analógico Digital) con el aporte de multiplexación entre los ocho pines, las tasas de conversión son de hasta 200KHz. El ADC de 12 bits puede ser utilizado por el controlador GPDMA.

También se encuentran asignados 10 bits para convertidor digital analógico (DAC), los cuales incluyen un temporizador de conversión específico. Por último podemos indicar que tiene cuatro temporizadores para fines generales y contadores, con un total de ocho entradas de captura, cada bloque tiene un temporizador de entrada externa. También se permite el control de motor por PWM con soporte para control de motores de tres fases.

A continuación se ilustra el diagrama de bloques de la tarjeta LPCXpresso, donde se puede observar entre sus componentes como más destacados al microcontrolador LPC1769.

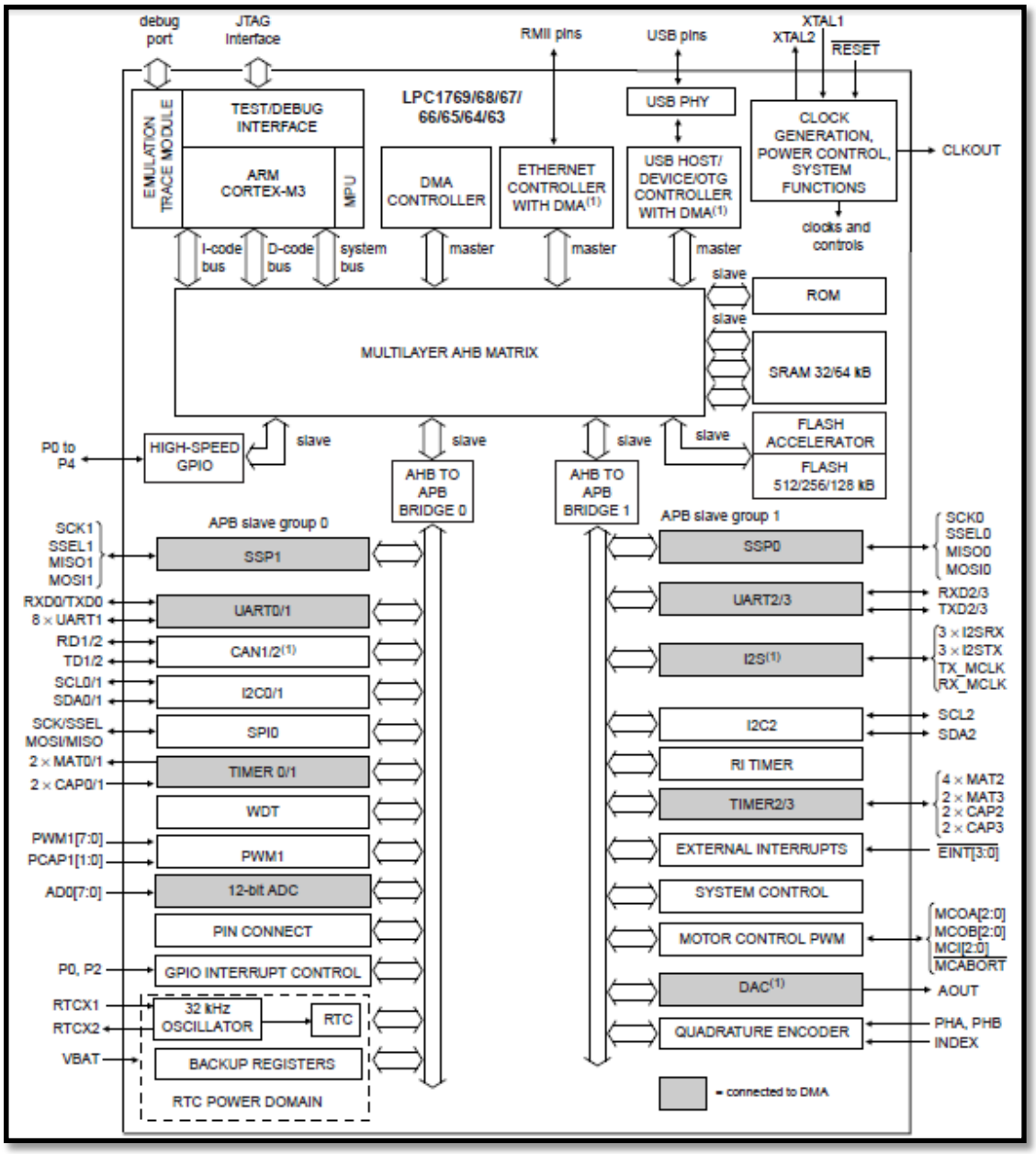


Figura 2.24.: Diagrama de bloques de la tarjeta LCPXpresso **[19]**

CAPÍTULO 3

EJERCICIOS PREVIOS Y REALIZACIÓN DEL PROYECTO

3.1. Introducción

En este capítulo analizaremos en forma detallada la elaboración de cada ejercicio realizado mediante el uso de protocolo de comunicación SPI, y la implementación del modo maestro – esclavo en la tarjeta LPCXpresso y la AVR Butterfly, además del desarrollo del tema central de este proyecto de tesis que está enfocado al control y manejo de motores BLDC mediante joystick utilizando el protocolo de comunicación serial SPI.

Para cada uno de los ejercicios se presentará una breve descripción de la funcionalidad del mismo, la asignación de master y Slave, la interacción entre las tarjetas LPCXpresso y AVR Butterfly, el respectivo diagrama de bloques y el algoritmo del sistema y, por último, el código fuente de la programación en

C, ya sea en la herramienta de software LPCXpresso o en el programa AVR Studio 4. Adicionalmente podemos mencionar que, para la elaboración de la programación master – Slave nos basamos en la programación del SSP que viene incluido en el kit de aplicaciones del software del LPCXpresso, el cual se encuentra contenido en el archivo NXP_LPCXpresso1769_MCB1700_2011-02-11.zip.

Los tres ejercicios adicionales al proyecto de tesis se han realizado variando la asignación del maestro – esclavo y la comunicación entre tarjetas de diferente fabricante, como se muestra a continuación:

- Comunicación SPI mediante asignación de tarjeta LPCXpresso como maestro y tarjeta LPCXpresso como esclavo, datos mostrados mediante display de 7 segmentos.
- Comunicación SPI mediante asignación de tarjeta AVR Butterfly como maestro y tarjeta LPCXpresso como esclavo, datos mostrados mediante display de 7 segmentos.
- Comunicación SPI mediante asignación de tarjeta LPCXpresso como maestro y tarjeta AVR Butterfly como esclavo, datos mostrados mediante display de 7 segmentos.

3.2. Contador de 0 a 9 mediante comunicación SPI de LPC a LPC

3.2.1. Descripción

El ejercicio consiste en realizar la presentación mediante display de 7 segmentos de un contador de 0 a 9, además de presentación en leds de los datos. La transmisión mediante comunicación serial SPI se va a realizar desde una tarjeta LPCXpresso, la cual está configurada como maestro, hacia otra tarjeta LPCXpresso, la cual se encuentra configurada como esclavo, y en esta última se encuentran conectados tanto el display de 7 segmentos como los leds que recibirán los datos de la tarjeta maestro. Para efectos de visualización, se ha colocado un display de 7 segmentos tanto en el puerto GPIO2 de la tarjeta maestro como en el puerto GPIO2 de la tarjeta esclavo, esto para poder apreciar el momento en que se corta la comunicación maestro – esclavo mediante un switch de conmutación y visualizar la pérdida de la transmisión, quedando en el esclavo el último dato mostrado al momento de cortar la comunicación entre ambos dispositivos.

3.2.2. Diagrama de bloques

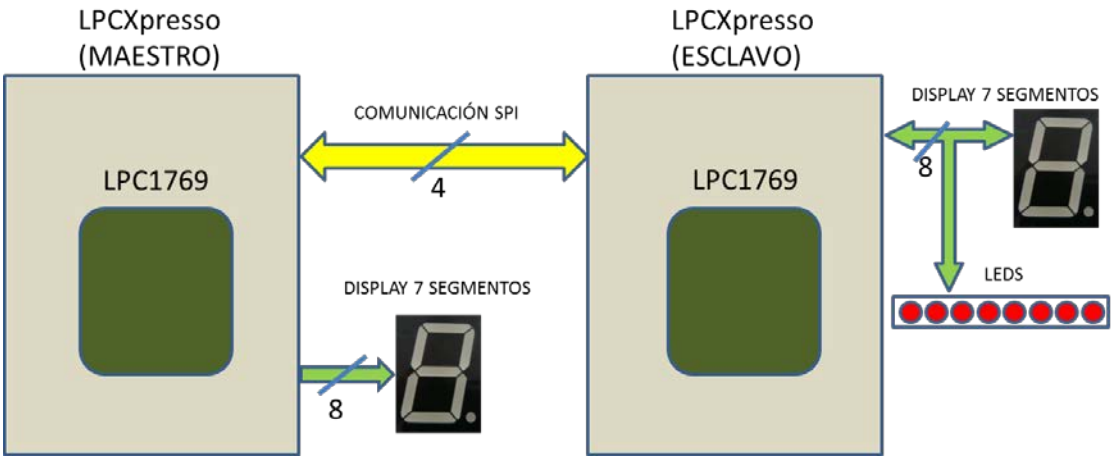


Figura 3.1.: Diagrama de bloques del Contador de 0 a 9 mediante comunicación SPI de LPC (Maestro) a LPC (Esclavo).

3.2.3. Diagrama de flujo

Diagrama de flujo del MAESTRO

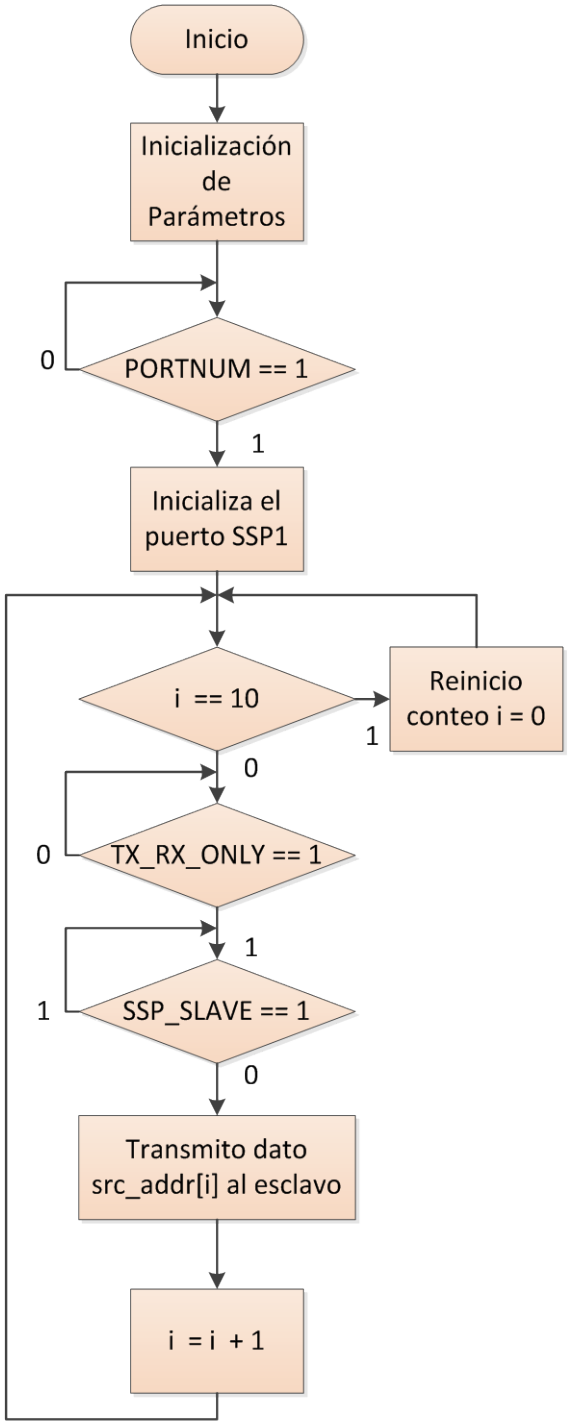


Figura 3.2.: Diagrama de Flujo del Contador de 0 a 9 mediante comunicación SPI del LPC (Maestro).

Diagrama de flujo del ESCLAVO

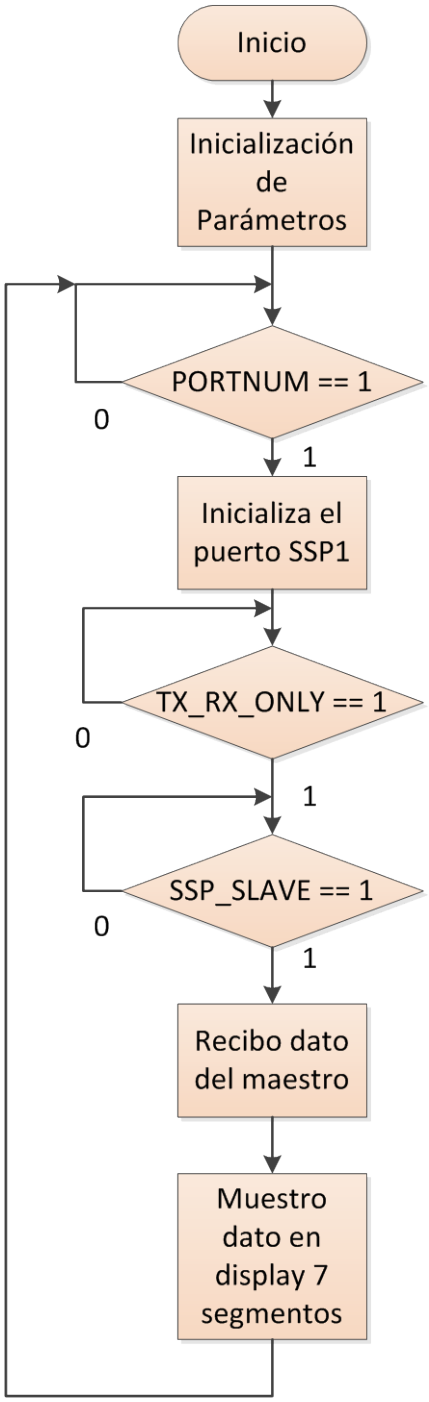


Figura 3.3.: Diagrama de Flujo del Contador de 0 a 9 mediante comunicación SPI del LPC (Esclavo).

3.2.4. Descripción del algoritmo

Descripción del algoritmo MAESTRO

1. Se realiza la inicialización de parámetros.

2. Se consulta si la variable PORTNUM == 1. En caso de que la condición sea verdadera, se va al siguiente paso.

3. Se inicializa el puerto SSP1 para habilitar la comunicación SPI en dicho puerto.

4. Se consulta si el valor de la posición del arreglo es mayor al tamaño del arreglo mismo. En caso de que la condición sea verdadera, se reinicia el conteo encerando i (i = 0). Caso contrario, se va al siguiente paso.

5. Se consulta si TX_RX_ONLY == 1. En caso de que la condición sea verdadera, se va al siguiente paso.

6. Se consulta si SSP_SLAVE ==1. En caso de que la condición sea falsa, se va al siguiente paso.

7. Se envía el dato desde el maestro hacia el esclavo por medio del arreglo src_addr[i].

8. Se incrementa el valor de i (i = i+1) para avanzar con el conteo y se regresa al paso 4.

Descripción del algoritmo ESCLAVO

1. Se realiza la inicialización de parámetros.

2. Se consulta si la variable PORTNUM == 1. En caso de que la condición sea verdadera, se va al siguiente paso.

3. Se inicializa el puerto SSP1 para habilitar la comunicación SPI en dicho puerto.

4. Se consulta si TX_RX_ONLY == 1. En caso de que la condición sea verdadera, se va al siguiente paso.

5. Se consulta si SSP_SLAVE ==1. En caso de que la condición sea falsa, se va al siguiente paso.

6. El esclavo recibe el dato proveniente del maestro.

7. El dato recibido por el esclavo es mostrado en el display de 7 segmentos. Se regresa al paso 2 para consultar si nuevamente se requiere establecer comunicación SPI entre ambos dispositivos.

3.2.5. Código fuente en C

Programa principal del MAESTRO

```
/*****************************************************************************
ESCUELA SUPERIOR POLITECNICA DEL LITORAL **************
**************       Microcontroladores Avanzados       *************
*****************************************************************
* Proyecto: Contador de 0 a 9 mediante comunicación SPI de LPC a LPC
*
* Descripción:
*
* Este archivo contiene la comunicación establecida entre tarjeta
* LCPXpresso maestro y LPCXpresso esclavo mediante protocolo  SPI,
* muestra de datos por medio de display de 7 segmentos. PROGRAMACION
* DEL MAESTRO. ***************************************************************
*
* Profersor: Ing. Carlos Valdivieso
* Alumnos: Andrés Maroto, Lidia Mantilla
* Fecha de creación: Abril, 2012
*
*****************************************************************
* Nombre del archivo: ssptest.c
* Proyecto realizado en base al proyecto NXP LPC17xx SSP example de
* NXP.
*****************************************************************/

#include <cr_section_macros.h> // Asignación de librerías
#include <NXP/crp.h>
/*Variable para almacenar el valor de entrada CRP.
Se realizará de forma automática cuando el "código de habilitación de lectura protegida" esté
seleccionado. Ver encabezado crp.h para mayor información*/

__CRP const unsigned int CRP_WORD = CRP_NO_CRP;

#include "LPC17xx.h"  //Definiciones del LPC13xx
```

```
#include "ssp.h"

/*Tenga cuidado con el número de puerto y el número de ubicación, ya que algunas
localidades de la ubicación pueden no existir en ese puerto.*/

#define PORT_NUM            1  //Asignación de constantes
#define LOCATION_NUM        0
#define LED     LPC_GPIO2

/*Asignación de arreglos para recepción y transmisión de los datos*/

uint8_t src_addr[SSP_BUFSIZE];
uint8_t dest_addr[SSP_BUFSIZE];

/*Asignación de valores a mostrar por cada led de configuración en el
display de 7 segmentos. */

static const uint8_t segmentLUT[10] =
{
                    //GFEDCBA(PTO)
                    (uint8_t) 0b01111110, //0
                    (uint8_t) 0b00001100, //1
                    (uint8_t) 0b10110110, //2
                    (uint8_t) 0b10011110, //3
                    (uint8_t) 0b11001100, //4
                    (uint8_t) 0b11011010, //5
                    (uint8_t) 0b11111010, //6
                    (uint8_t) 0b01001110, //7
                    (uint8_t) 0b11111110, //8
                    (uint8_t) 0b11011110, //9
};
/*****************************************************************************
PROGRAMA PRINCIPAL                  *
*****************************************************************************/

int main (void)
{
   uint32_t j, portnum = PORT_NUM;
   uint32_t i = 0;

   /*La función SystemClockUpdate() actualiza la variable
   SystemFrequency*/

   SystemClockUpdate();
   LED -> FIODIR = 0xFF;

   if ( portnum == 0 )
         SSP0Init();                        /* Inicializa el puerto SSP */
   else if ( portnum == 1 )
         SSP1Init();
   while ( 1 )
   {
```

```
            for(j = 10000000; j > 0; j--); // Retardo

            LED->FIOCLR = 0xFF;

            src_addr[0] = (uint8_t)segmentLUT[i]; // Dato a transmitir

            LED->FIOSET = src_addr[0];

            i=i+1; // Incremento de la posición del arreglo
            if (i==10) i=0;//Reinicio en el arreglo para reiniciar conteo


            #if TX_RX_ONLY
            /* Para la comunicación entre tarjetas, una tarjeta es
          establecida como maestro transmisor, la otra se establece
          como esclavo receptor.*/

            #if SSP_SLAVE
            /* Esclavo recibe datos del maestro */
                    SSPReceive( portnum, (uint8_t *)dest_addr, SSP_BUFSIZE );
                    for ( i = 0; i < SSP_BUFSIZE; i++ )
                    {
                    if ( src_addr[i] != dest_addr[i] )
                    {
                      while ( 1 ); /*Verificación de falla o error fatal en
                                          el programa */
                    }
                    }
            #else
                    /* Maestro transmite */
                    SSPSend( portnum, (uint8_t *)src_addr, 1);
            #endif
            #else
            /*TX_RX_ONLY = 0, es una prueba de bucle de retorno interno
              dentro del periférico SSP, sirve para comunicarse con una
              serie EEPROM.*/

            #if LOOPBACK_MODE
                    LoopbackTest( portnum, LOCATION_NUM );
            #else
                    SEEPROMTest( portnum, LOCATION_NUM );
            #endif
            #endif
            /*No salir del main() para una fácil depuración del
              programa.*/
      }
      return 0;
    }

/*****************************************************************************
FIN DEL PROGRAMA                    *
*****************************************************************************/
```

Programa principal del ESCLAVO

```
/*******************************************************************************
 ESCUELA SUPERIOR POLITECNICA DEL LITORAL *************
**************       Microcontroladores Avanzados       *************
*********************************************************************
* Proyecto: Contador de 0 a 9 mediante comunicación SPI de LPC a LPC
*
* Descripción:
*
* Este archivo contiene la comunicación establecida entre tarjeta
* LCPXpresso maestro y LPCXpresso esclavo mediante protocolo  SPI,
* muestra de datos por medio de display de 7 segmentos. PROGRAMACION
* DEL ESCLAVO. ***************************************************************************
*
* Profersor: Ing. Carlos Valdivieso
* Alumnos: Andrés Maroto, Lidia Mantilla
* Fecha de creación: Abril, 2012
*
*********************************************************************
* Nombre del archivo: ssptest.c
* Proyecto realizado en base al proyecto NXP LPC17xx SSP example de
* NXP.
*********************************************************************/

#include <cr_section_macros.h> // Asignación de librerías
#include <NXP/crp.h>
/*Variable para almacenar el valor de entrada CRP.
Se realizará de forma automática cuando el "código de habilitación de
lectura protegida" esté seleccionado. Ver encabezado crp.h para mayor
información*/

__CRP const unsigned int CRP_WORD = CRP_NO_CRP;

#include "LPC17xx.h" //Definiciones del LPC13xx
#include "ssp.h"

/*Tenga cuidado con el número de puerto y el número de ubicación, ya
que algunas localidades de la ubicación pueden no existir en ese
puerto.*/

#define PORT_NUM            1  //Asignación de constantes
#define LOCATION_NUM        0
#define LED LPC_GPIO2

/*Asignación de arreglos para recepción y transmisión de los datos*/

uint8_t src_addr[SSP_BUFSIZE];
uint8_t dest_addr[SSP_BUFSIZE];

/*********************************************************************
  PROGRAMA PRINCIPAL                  *
*********************************************************************/
```

```
int main (void)
{

  uint32_t portnum = PORT_NUM;

/*La función SystemClockUpdate() actualiza la variable
  SystemFrequency*/

  SystemClockUpdate();
  LED -> FIODIR = 0xFF;

  if ( portnum == 0 )
          SSP0Init();                          /* Inicializa el puerto SSP */
  else if ( portnum == 1 )
          //SSP1Init();
  while ( 1 )
  {
            SSP1Init();

            dest_addr[0] = 0;


           #if TX_RX_ONLY
           /* Para la comunicación entre tarjetas, una tarjeta es
         establecida como maestro transmisor, la otra se establece
         como esclavo receptor.*/

            #if SSP_SLAVE
       /* Esclavo recibe datos del maestro */
                    SSPReceive( portnum, (uint8_t *)dest_addr, 1);

                    LED -> FIOCLR = 0xFF;  //Encero valores en el display

                     if (dest_addr[0]!= 0xFF){
                             LED -> FIOSET =  dest_addr[0];
                     }
                    #else
                     /* Maestro transmite */
                     SSPSend( portnum, (uint8_t *)src_addr, SSP_BUFSIZE);
                    #endif
                    #else
                    /* TX_RX_ONLY = 0, es una prueba de bucle de retorno
                       Interno dentro del periférico SSP, sirve para
                       comunicarse con una serie EEPROM. */

                    #if LOOPBACK_MODE
                     LoopbackTest( portnum, LOCATION_NUM );
                    #else
                     SEEPROMTest( portnum, LOCATION_NUM );
                    #endif
                    #endif
                    /*No salir del main() para una fácil depuración del
```

```
                programa.*/
  }
  return 0;
}

/*************************************************************************
          FIN DEL PROGRAMA                    *
*************************************************************************/
```

3.3. Contador de 0 a 9 mediante comunicación SPI de AVR a LPC

3.3.1. Descripción

El ejercicio consiste en realizar la presentación mediante display de 7 segmentos de un contador de 0 a 9, además de presentación en leds de los datos. La transmisión mediante comunicación serial SPI se va a realizar desde una tarjeta AVR Butterfly, la cual está configurada como maestro, hacia una tarjeta LPCXpresso, la cual se encuentra configurada como esclavo, y en esta última se encuentran conectados tanto el display de 7 segmentos como los leds que recibirán los datos de la tarjeta maestro. Para efectos de visualización, se ha colocado un display de 7 segmentos tanto en el puerto GPIO2 de la tarjeta maestro como en el puerto GPIO2 de la tarjeta esclavo, esto para poder apreciar el momento en que se corta la comunicación maestro – esclavo mediante un switch de conmutación y visualizar la pérdida de la transmisión, quedando en el esclavo el último dato mostrado al momento de cortar la comunicación entre ambos dispositivos.

3.3.2. Diagrama de bloques

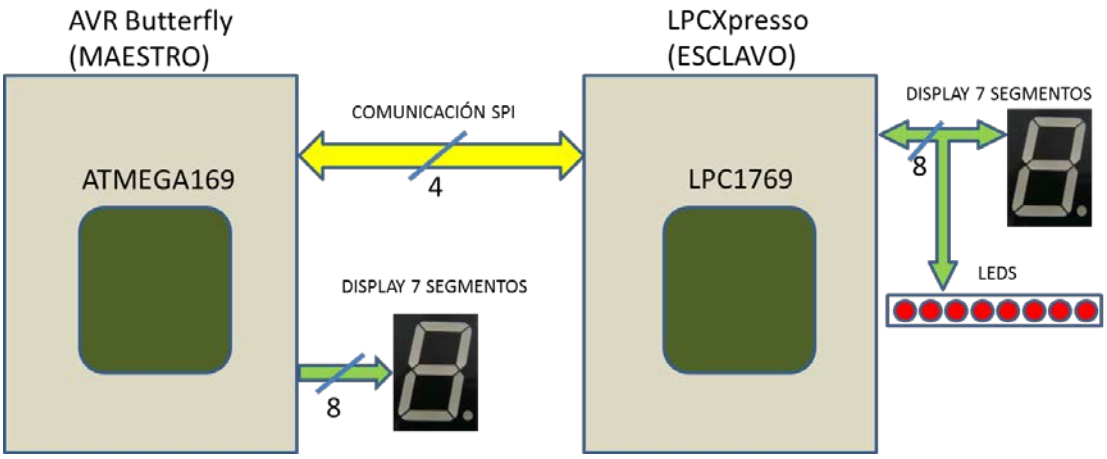


Figura 3.4.: Diagrama de Bloques del Contador de 0 a 9 mediante comunicación SPI de AVR (Maestro) a LPC (Esclavo).

3.3.3. Diagrama de flujo

Diagrama de flujo del MAESTRO

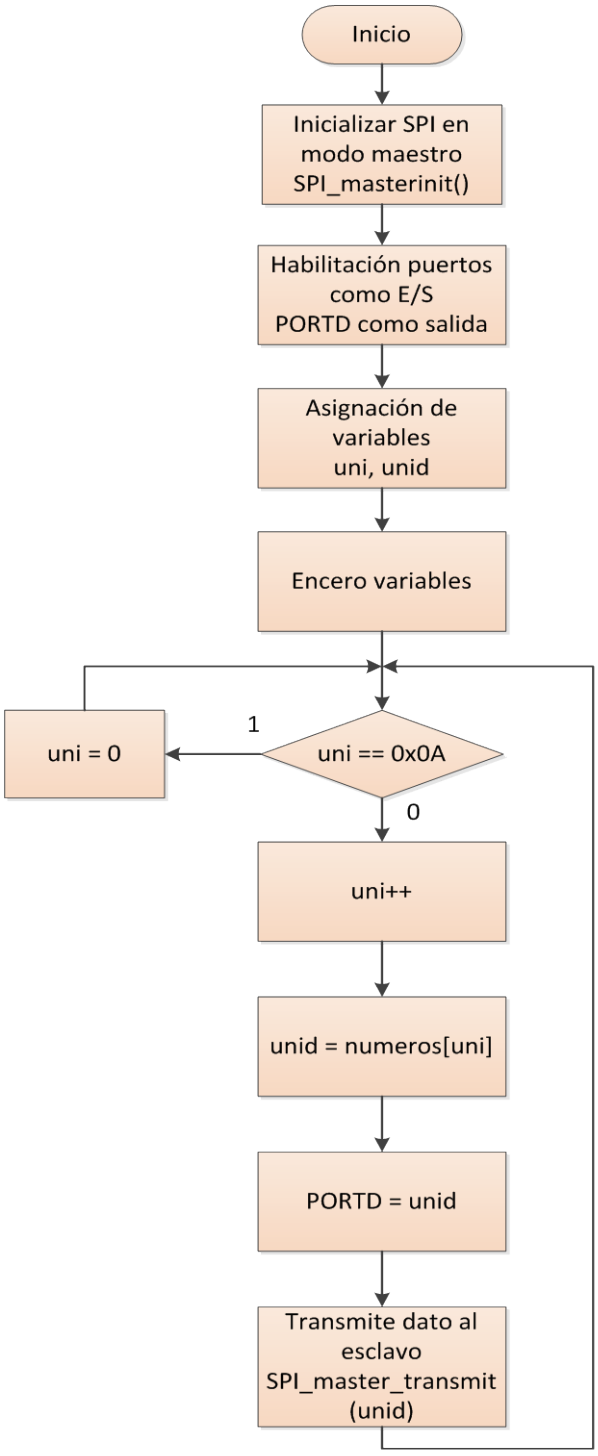


Figura 3.5: Diagrama de Flujo del Contador de 0 a 9 mediante comunicación SPI del AVR (Maestro).

Diagrama de flujo del ESCLAVO

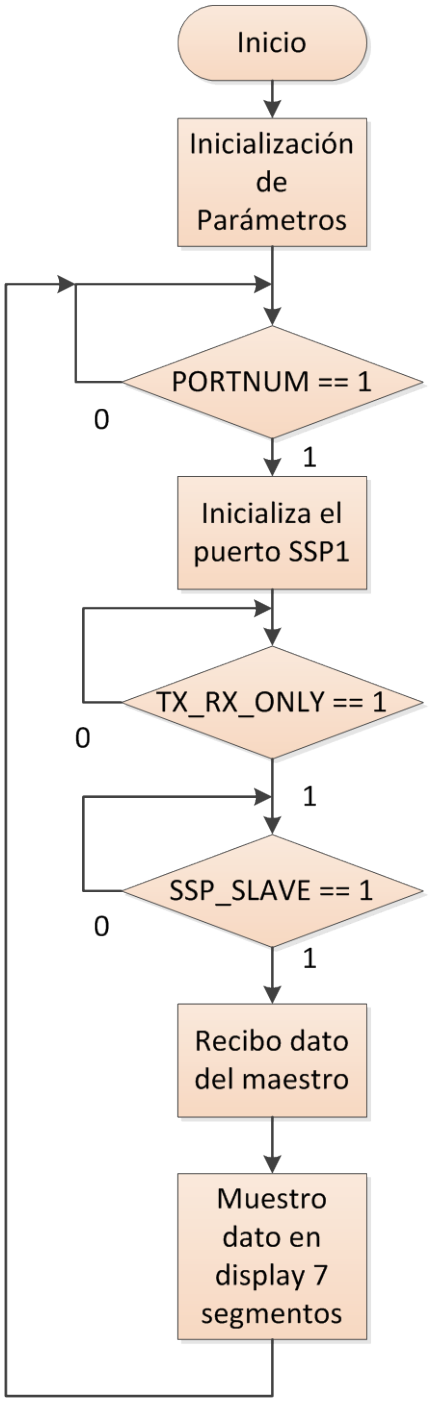


Figura 3.6: Diagrama de Flujo del Contador de 0 a 9 mediante comunicación SPI del LPC (Esclavo).

3.3.4. Descripción del algoritmo

Descripción del algoritmo MAESTRO

1. Se realiza la inicialización de SPI en modo maestro mediante la función SPI_MasterInit ().

2. Se habilita los puertos como entrada o salida. PORTD se configura como salida.

3. Se realiza la asignación de las variables, entre ellas uni y unid.

4. Se encera las variables a utilizar en el programa.

5. Se consulta si el valor de uni es igual a 10 (0x0A). En caso de que la condición sea verdadera, se reinicia el conteo encerando uni (uni = 0). Caso contrario, se va al siguiente paso.

6. Se incrementa el valor de uni en 1 (uni = uni+1 ó uni++).

7. Se asigna el valor del arreglo numeros[uni] a la variable unid. Este arreglo recibe el valor de la variable uni y envía el código correspondiente para encendido de leds en el display de 7 segmentos.

8. Se asigna el valor de unid en PORTD que es el puerto de comunicación con el esclavo.

9. Se envía el dato desde el maestro hacia el esclavo por medio de la función SPI_transmit(unid) y se regresa al paso 5.

Descripción del algoritmo ESCLAVO

1. Se realiza la inicialización de parámetros.

2. Se consulta si la variable PORTNUM == 1. En caso de que la condición sea verdadera, se va al siguiente paso.

3. Se inicializa el puerto SSP1 para habilitar la comunicación SPI en dicho puerto.

4. Se consulta si TX_RX_ONLY == 1. En caso de que la condición sea verdadera, se va al siguiente paso.

5. Se consulta si SSP_SLAVE ==1. En caso de que la condición sea falsa, se va al siguiente paso.

6. El esclavo recibe el dato proveniente del maestro.

7. El dato recibido por el esclavo es mostrado en el display de 7 segmentos. Se regresa al paso 2 para consultar si nuevamente se requiere establecer comunicación SPI entre ambos dispositivos.

3.3.5. Código fuente en C

Programa principal del MAESTRO

```
/*****************************************************************************
ESCUELA SUPERIOR POLITECNICA DEL LITORAL **************
**************       Microcontroladores Avanzados       **************
*********************************************************************
* Proyecto: Contador de 0 a 9 mediante comunicación SPI de AVR a LPC
*
* Descripción:
*
* Este archivo contiene la comunicación establecida entre tarjeta
* AVR Butterfly maestro y LPCXpresso esclavo mediante protocolo  SPI,
* muestra de datos por medio de display de 7 segmentos.
* PROGRAMACION DEL MAESTRO.
*********************************************************************
*
* Profersor: Ing. Carlos Valdivieso
* Alumnos: Andrés Maroto, Lidia Mantilla
* Fecha de creación: Abril, 2012
*
*********************************************************************
* Nombre del archivo: spi_master_lcd.c
*********************************************************************/

// Asignación de librerías
#include <avr/io.h>
#include <util/delay.h>
#include <avr/pgmspace.h>
#include <avr/signal.h>
#include <avr/interrupt.h>
```

```
// Tabla de conversión de binario a 7 segmentos
unsigned int numeros[] = {
                    0b01111110,
                    0b00001100,
                    0b10110110,
                    0b10011110,
                    0b11001100,
                    0b11011010,
                    0b11111010,
                    0b01001110,
                    0b11111110,
                    0b11011110,
                    };
// Variables globales:
unsigned int uni, dec;

void SPI_masterinit(void)
{
  DDRB=0x07;          // MOSI, SS y SCK como salida
                  // SPI enable, dispositivo MASTER, Fosc/2
  SPCR=(1<<SPE)|(1<<MSTR);
}

void SPI_master_transmit(unsigned char cdata)
{
 SPDR=cdata; // coloca dato a enviar en el registro SPDR
 while(!(SPSR & (1<<SPIF)));//espera mientras completa la transmisión
}
/*********************************************************************
*                    PROGRAMA PRINCIPAL                             *
*********************************************************************/
int main(void)
{
  SPI_masterinit();

  DDRB=0x0F;
  PORTB=0xF0;
  PORTB=0;
  DDRD=0xFF;  // PORTD como salida
  PORTD=0x00;

  uni=0;            // encera unidades
  unid=numeros[uni];  // conversión binario a 7 seg
  PORTD=unid;       // muestra en el PORTD del master
  SPI_master_transmit(unid);  // transmite información al esclavo

  while(1)
  {
     _delay_ms(500);
     _delay_ms(500);

      uni++;
```

```
    if (uni==0x0A)
      {
                uni=0;
      }
    unid=numeros[uni];  // conversión binario a 7 segmentos
    PORTD=unid;         // muestra en display
    SPI_master_transmit(unid);  // transmite dato
  }
}
/*****************************************************************
*                      FIN DEL PROGRAMA                          *
*****************************************************************/
```

Programa principal del ESCLAVO

```
/*****************************************************************
************** ESCUELA SUPERIOR POLITECNICA DEL LITORAL *************
**************       Microcontroladores Avanzados       *************
*****************************************************************
* Proyecto: Contador de 0 a 9 mediante comunicación SPI de AVR a LPC
*
* Descripción:
*
* Este archivo contiene la comunicación establecida entre tarjeta
* AVR Butterfly maestro y LPCXpresso esclavo mediante protocolo  SPI,
* muestra de datos por medio de display de 7 segmentos.
* PROGRAMACION DEL ESCLAVO.
*****************************************************************
*
* Profersor: Ing. Carlos Valdivieso
* Alumnos: Andrés Maroto, Lidia Mantilla
* Fecha de creación: Abril, 2012
*
*****************************************************************
* Nombre del archivo: ssptest.c
* Proyecto realizado en base al proyecto NXP LPC17xx SSP example de
* NXP.
*****************************************************************/

// Asignación de librerías

#include <cr_section_macros.h>
#include <NXP/crp.h>

/*Variable para almacenar el valor de entrada CRP.
Se realizará de forma automática cuando el "código de habilitación de
lectura protegida" esté seleccionado. Ver encabezado crp.h para mayor
información*/

__CRP const unsigned int CRP_WORD = CRP_NO_CRP;
```

```
#include "LPC17xx.h"  //Definiciones del LPC13xx
#include "ssp.h"

/*Tenga cuidado con el número de puerto y el número de ubicación, ya
que algunas localidades de la ubicación pueden no existir en ese
puerto.*/

#define PORT_NUM            1 // Asignación de constantes
#define LOCATION_NUM        0
#define LED LPC_GPIO2

/*Asignación de arreglos para recepción y transmisión de los datos*/

uint8_t src_addr[SSP_BUFSIZE];
uint8_t dest_addr[SSP_BUFSIZE];

/*****************************************************************
*                      PROGRAMA PRINCIPAL                       *
*****************************************************************/


int main (void)
{
  uint32_t portnum = PORT_NUM;

 /*La función SystemClockUpdate() actualiza la variable
  SystemFrequency*/

  SystemClockUpdate();
  LED -> FIODIR = 0xFF;

  if ( portnum == 0 )
        SSP0Init();           /* Inicializa el puerto SSP */
  else if ( portnum == 1 )
        SSP1Init();
  while ( 1 )
  {
          SSP1Init();

          dest_addr[0] = 0;


          #if TX_RX_ONLY
       /* Para la comunicación entre tarjetas, una tarjeta es
          establecida como maestro transmisor, la otra se
                establece como esclavo receptor.*/

          #if SSP_SLAVE
       /* Esclavo recibe datos del maestro */
            SSPReceive( portnum, (uint8_t *)dest_addr, 1);

            LED -> FIOCLR = 0xFF; // Encero valores en el display
```

```
                    if (dest_addr[0]!= 0xFF){
                            LED -> FIOSET =  dest_addr[0];
                    }
                  #else
                    /* Maestro transmite */
                    SSPSend( portnum, (uint8_t *)src_addr, SSP_BUFSIZE);
                  #endif
                  #else
                  /* TX_RX_ONLY = 0, es una prueba de bucle de retorno
              Interno dentro del periférico SSP, sirve para
              comunicarse con una serie EEPROM. */

                  #if LOOPBACK_MODE
                    LoopbackTest( portnum, LOCATION_NUM );
                  #else
                    SEEPROMTest( portnum, LOCATION_NUM );
                  #endif
                  #endif
                  /*No salir del main() para una fácil depuración del
          programa.*/

    }
    return 0;
}

/*****************************************************************
*                      FIN DEL PROGRAMA                         *
*****************************************************************/
```

3.4. Contador de 0 a 9 mediante comunicación SPI de LPC a AVR

3.4.1. Descripción

El ejercicio consiste en realizar la presentación mediante display de 7 segmentos de un contador de 0 a 9, además de presentación en leds de los datos. La transmisión mediante comunicación serial SPI se va a realizar desde una tarjeta LPCXpresso, la cual está configurada como maestro, hacia una tarjeta AVR Butterfly, la cual se encuentra configurada como esclavo, y en esta última se encuentran conectados tanto el display de 7 segmentos

como los leds que recibirán los datos de la tarjeta maestro. Para efectos de visualización, se ha colocado un display de 7 segmentos tanto en el puerto GPIO2 de la tarjeta maestro como en el puerto GPIO2 de la tarjeta esclavo, esto para poder apreciar el momento en que se corta la comunicación maestro – esclavo mediante un switch de conmutación y visualizar la pérdida de la transmisión, quedando en el esclavo el último dato mostrado al momento de cortar la comunicación entre ambos dispositivos.

3.4.2. Diagrama de bloques

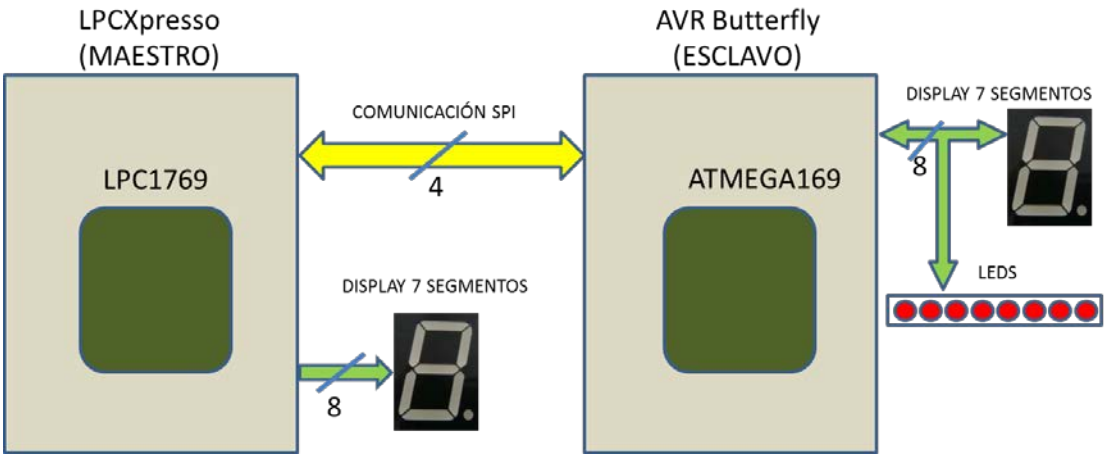


Figura 3.7: Diagrama de Bloques del Contador de 0 a 9 mediante comunicación SPI de LPC (Maestro) a AVR (Esclavo).

3.4.3. Diagrama de flujo

Diagrama de flujo del MAESTRO

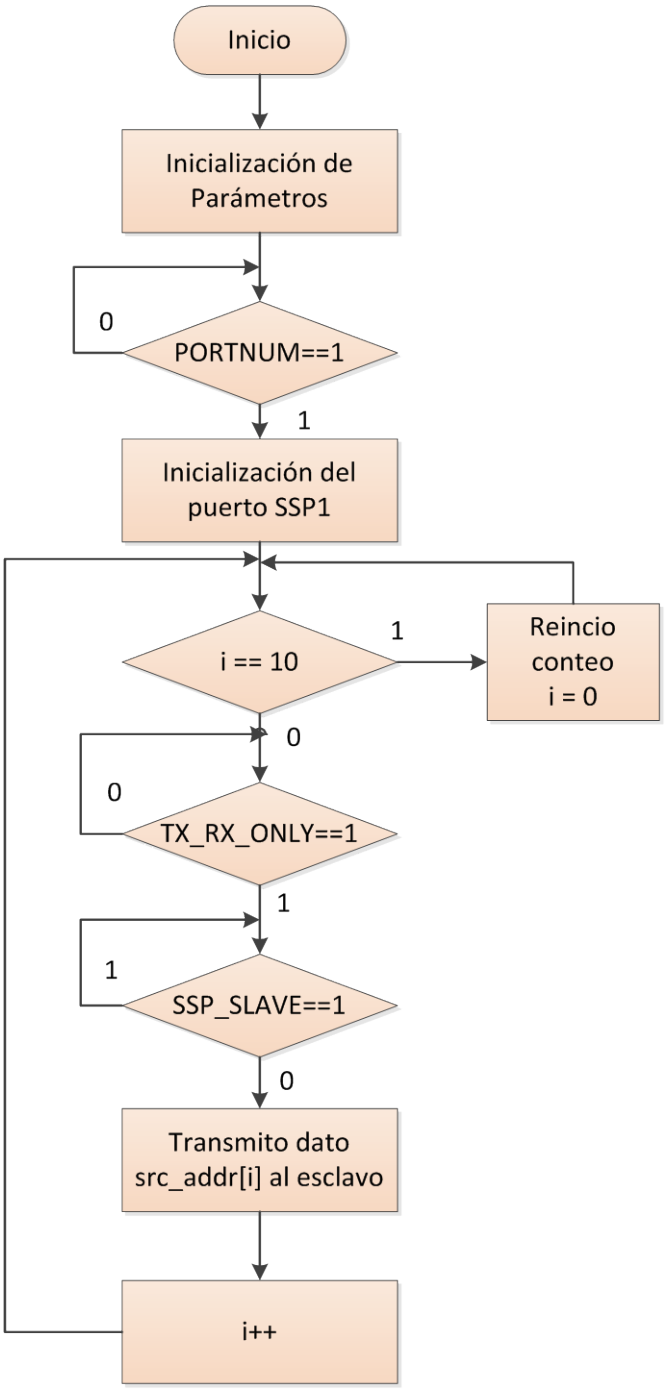


Figura 3.8: Diagrama de Flujo del Contador de 0 a 9 mediante comunicación SPI del LPC (Maestro).

Diagrama de flujo del ESCLAVO

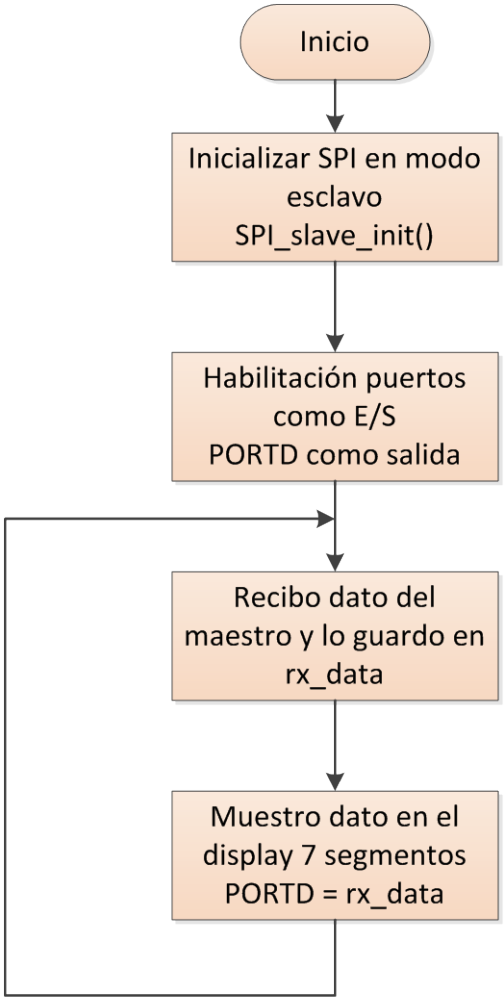


Figura 3.9: Diagrama de Flujo del Contador de 0 a 9 mediante comunicación SPI del AVR (Esclavo).

3.4.4. Descripción del algoritmo

Descripción del algoritmo MAESTRO

1. Se realiza la inicialización de parámetros.

2. Se consulta si la variable PORTNUM == 1. En caso de que la condición sea verdadera, se va al siguiente paso.

3. Se inicializa el puerto SSP1 para habilitar la comunicación SPI en dicho puerto.

4. Se consulta si el valor de la posición del arreglo es mayor al tamaño del arreglo mismo. En caso de que la condición sea verdadera, se reinicia el conteo encerando i (i = 0). Caso contrario, se va al siguiente paso.

5. Se consulta si TX_RX_ONLY == 1. En caso de que la condición sea verdadera, se va al siguiente paso.

6. Se consulta si SSP_SLAVE ==1. En caso de que la condición sea falsa, se va al siguiente paso.

7. Se envía el dato desde el maestro hacia el esclavo por medio del arreglo src_addr[i].

8. Se incrementa el valor de i (i = i+1) para avanzar con el conteo y se regresa al paso 4.

Descripción del algoritmo ESCLAVO

1. Se realiza la inicialización de SPI en modo esclavo mediante la función SPI_slave_init().

2. Se habilita los puertos como entrada o salida. PORTD se configura como salida.

3. Recibo dato del maestro por medio de la función SPI_slave_receive() y lo guardo en rx_data.

4. Muestro el dato recibido (rx_data) en el display de 7 segmentos, el cual se encuentra conectado en el PORTD y espero recibir un nuevo dato.

3.4.5. Código fuente en C

Programa principal del MAESTRO

```
/*****************************************************************
************** ESCUELA SUPERIOR POLITECNICA DEL LITORAL *************
**************       Microcontroladores Avanzados       **************
*****************************************************************
* Proyecto: Contador de 0 a 9 mediante comunicación SPI de LPC a AVR
*
* Descripción:
*
* Este archivo contiene la comunicación establecida entre tarjeta
* LPCXpresso maestro y AVR Butterfly esclavo mediante protocolo  SPI,
* muestra de datos por medio de display de 7 segmentos.
* PROGRAMACION DEL MAESTRO.
*****************************************************************
*
* Profersor: Ing. Carlos Valdivieso
* Alumnos: Andrés Maroto, Lidia Mantilla
* Fecha de creación: Abril, 2012
*
*****************************************************************
* Nombre del archivo: ssptest.c
* Proyecto realizado en base al proyecto NXP LPC17xx SSP example de
* NXP.
*****************************************************************/

// Asignación de librerías

  #include <cr_section_macros.h>
  #include <NXP/crp.h>

/*Variable para almacenar el valor de entrada CRP.
  se realizará de forma automática cuando el "código de habilitación
  de lectura protegida" esté seleccionado. Ver encabezado crp.h para
  mayor información*/

  __CRP const unsigned int CRP_WORD = CRP_NO_CRP ;

  #include "LPC17xx.h"   //Definiciones del LPC13xx
  #include "ssp.h"

 /*Tenga cuidado con el número de puerto y el número de ubicación,
   ya que algunas localidades de la ubicación pueden no existir en
   ese puerto.*/

  #define PORT_NUM            1 // Asignación de constantes
  #define LOCATION_NUM                0
  #define LED   LPC_GPIO2

 /*Asignación de arreglos para recepción y transmisión de los datos*/
```

```
 uint8_t src_addr[SSP_BUFSIZE];
 uint8_t dest_addr[SSP_BUFSIZE];

/*Asignación de valores a mostrar por cada led de configuración en
  el display de 7 segmentos. */

 static const uint8_t segmentLUT[10] =
 {
                    //GFEDCBA(PTO)
                    (uint8_t) 0b01111110, //0
                    (uint8_t) 0b00001100, //1
                    (uint8_t) 0b10110110, //2
                    (uint8_t) 0b10011110, //3
                    (uint8_t) 0b11001100, //4
                    (uint8_t) 0b11011010, //5
                    (uint8_t) 0b11111010, //6
                    (uint8_t) 0b01001110, //7
                    (uint8_t) 0b11111110, //8
                    (uint8_t) 0b11011110, //9
 };

/*****************************************************************
*                        PROGRAMA PRINCIPAL                      *
*****************************************************************/

int main (void)
 {

   uint32_t j, portnum = PORT_NUM;
   uint32_t i = 0;

   /*La función SystemClockUpdate() actualiza la variable
   SystemFrequency*/

   SystemClockUpdate();
   LED -> FIODIR = 0xFF;

   if ( portnum == 0 )
         SSP0Init();          /* Inicializa el puerto SSP */
   else if ( portnum == 1 )
         SSP1Init();
   while ( 1 )
   {

            for(j = 10000000; j > 0; j--); // Retardo

            LED->FIOCLR = 0xFF;

            src_addr[0] = (uint8_t)segmentLUT[i]; // Dato a transmitir

            LED->FIOSET=src_addr[0];

            i=i+1;  // Incremento de la posición del arreglo
```

```
        if (i==10) i=0;//Reinicio en el arreglo para reiniciar conteo


        #if TX_RX_ONLY
        /* Para la comunicación entre tarjetas, una tarjeta es
      establecida como maestro transmisor, la otra se establece
      como esclavo receptor.*/


        #if SSP_SLAVE
        /* Esclavo recibe datos del maestro */
          SSPReceive( portnum, (uint8_t *)dest_addr, SSP_BUFSIZE );
          for ( i = 0; i < SSP_BUFSIZE; i++ )
          {
                if ( src_addr[i] != dest_addr[i] )
                {
                  while ( 1 ); /*Verificación de falla o error fatal en
                    el programa */
                }
                }
        #else
                /* Maestro transmite */
                SSPSend( portnum, (uint8_t *)src_addr, 1);
        #endif
        #else
        /*TX_RX_ONLY = 0, es una prueba de bucle de retorno interno
          dentro del periférico SSP, sirve para comunicarse con una
      serie EEPROM.*/

        #if LOOPBACK_MODE
                LoopbackTest( portnum, LOCATION_NUM );
        #else
                SEEPROMTest( portnum, LOCATION_NUM );
        #endif
        #endif
        /*No salir del main() para una fácil depuración del
      programa.*/
    }
    return 0;
  }

/*********************************************************************
*                       FIN DEL PROGRAMA                            *
*********************************************************************/
```

Programa principal del ESCLAVO

```
/*********************************************************************
************** ESCUELA SUPERIOR POLITECNICA DEL LITORAL **************
**************       Microcontroladores Avanzados       **************
*********************************************************************
* Proyecto: Contador de 0 a 9 mediante comunicación SPI de LPC a AVR
*
* Descripción:
*
* Este archivo contiene la comunicación establecida entre tarjeta
* LPCXpresso maestro y AVR Butterfly esclavo mediante protocolo  SPI,
* muestra de datos por medio de display de 7 segmentos.
* PROGRAMACION DEL ESCLAVO.
*********************************************************************
*
* Profersor: Ing. Carlos Valdivieso
* Alumnos: Andrés Maroto, Lidia Mantilla
* Fecha de creación: Abril, 2012
*
*********************************************************************
* Nombre del archivo: cont_slave.c
*********************************************************************/

// Asignación de librerías
#include <avr/io.h>
#include <util/delay.h>


// Funciones:
void SPI_slave_init(void);
unsigned char SPI_slave_receive(void);

/*********************************************************************
*                      PROGRAMA PRINCIPAL                           *
*********************************************************************/

int main(void)
{
  unsigned char rx_data;

  SPI_slave_init();     // Inicializa la comunicación SPI

  DDRD=0xFF;            // PORTD como salida
  PORTD=0x00;

  while(1)
  {
    rx_data=SPI_slave_receive(); // recibe dato transmitido
    PORTD= rx_data;              // muestra número
  }

}
```

```
/*********************************************************************
*                        FIN DEL PROGRAMA                           *
*********************************************************************/

/*********************************************************************
*          Funciones para configurar AVR como esclavo               *
*********************************************************************/

void SPI_slave_init(void)
{
 DDRB=0x08;      // MISO como salida y el resto como entrada
 SPCR=(1<<SPE);  // SPI enable, dispositivo Slave
 //SPSR=(1<<SPI2X);
}

unsigned char SPI_slave_receive(void)
{
 while(!(SPSR & (1<<SPIF)));//Espera mientras recibe el dato completo
 return SPDR; // retorna dato recibido
}
```

3.5. Control de motor BLDC por medio de joystick utilizando comunicación SPI de AVR a LPC

3.5.1. Descripción

El ejercicio consiste en realizar el control del motor BLDC por medio del joystick incorporado en la tarjeta AVR. La transmisión mediante comunicación serial SPI se va a realizar desde una tarjeta AVR Butterfly, la cual está configurada como maestro, hacia una tarjeta LPCXpresso, la cual se encuentra configurada como esclavo, y a esta última se encuentra conectado el motor BLDC, el cual recibirá las órdenes de la tarjeta maestro. El control del joystick ha sido configurado de la siguiente manera: hacia arriba se incrementa la velocidad del motor, hacia abajo se disminuye la velocidad del motor, hacia la derecha se invierte el sentido de giro del motor

y hacia la izquierda o presionando el centro se realiza el encendido/apagado del motor. Para efectos de visualización, estas órdenes generadas por medio del joystick se irán presentando en la pantalla LCD incorporada en la tarjeta AVR Butterfly.

3.5.2. Diagrama de bloques

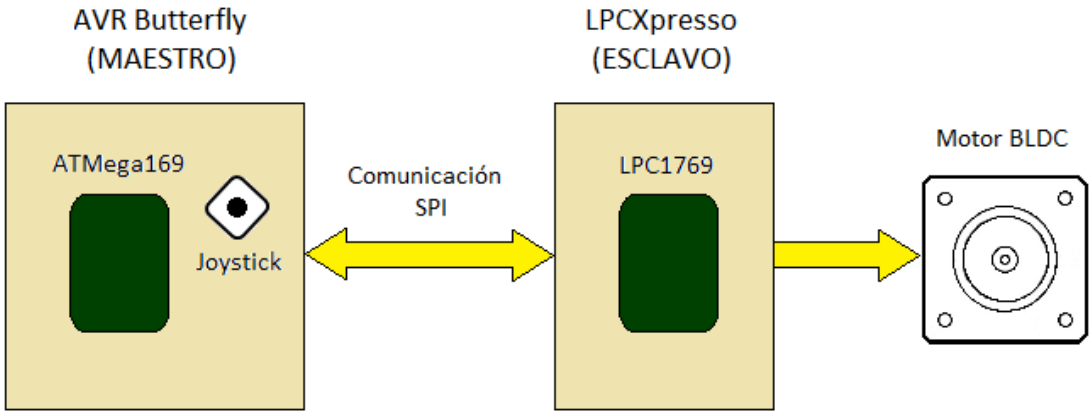


Figura 3.10: Diagrama de Bloques del Control de motor BLDC por medio del joystick utilizando comunicación SPI de AVR a LPC.

3.5.3. Diagrama de flujo

Diagrama de flujo del MAESTRO

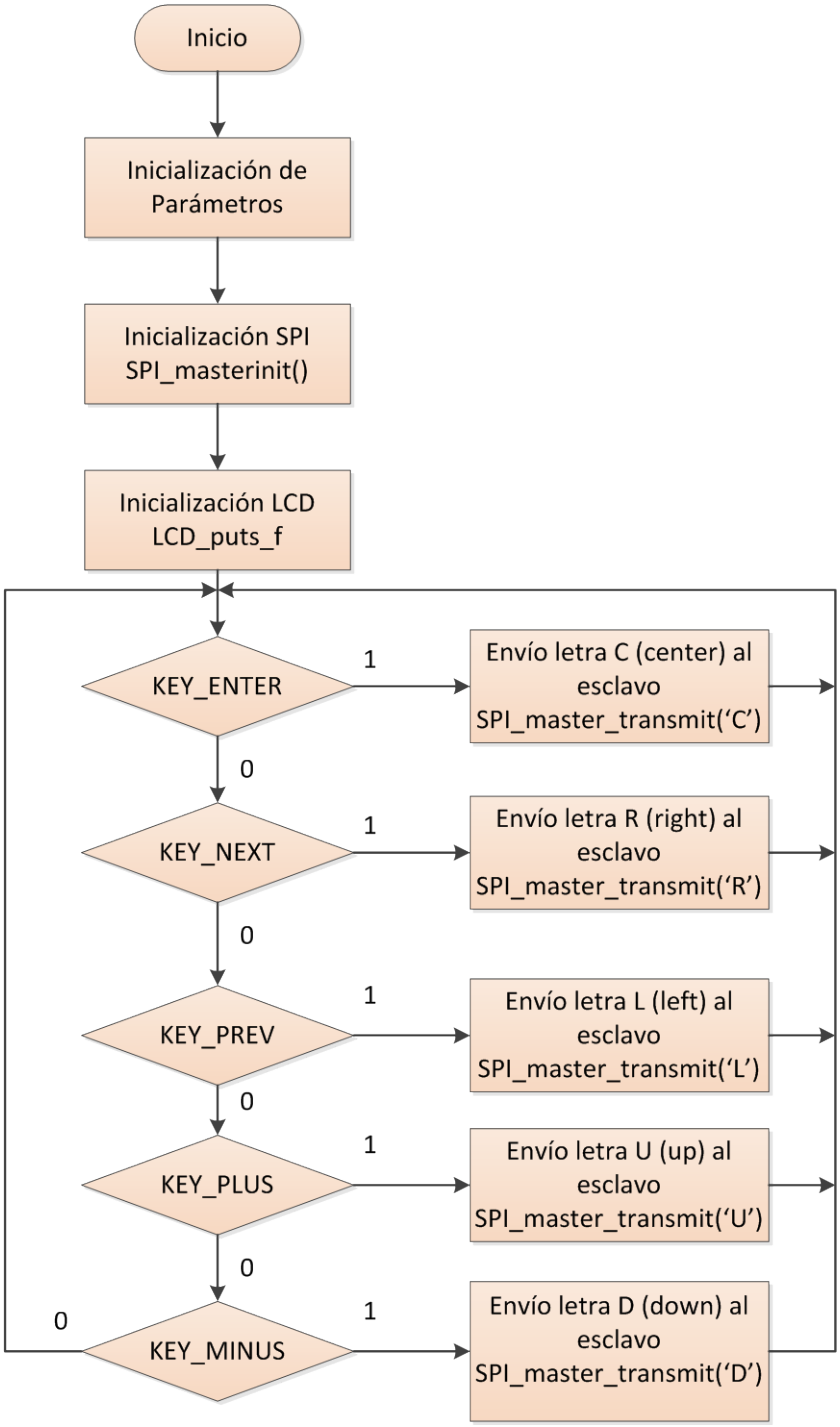


Figura 3.11: Diagrama de Flujo de la comunicación SPI del AVR (Maestro).

Diagrama de flujo del ESCLAVO

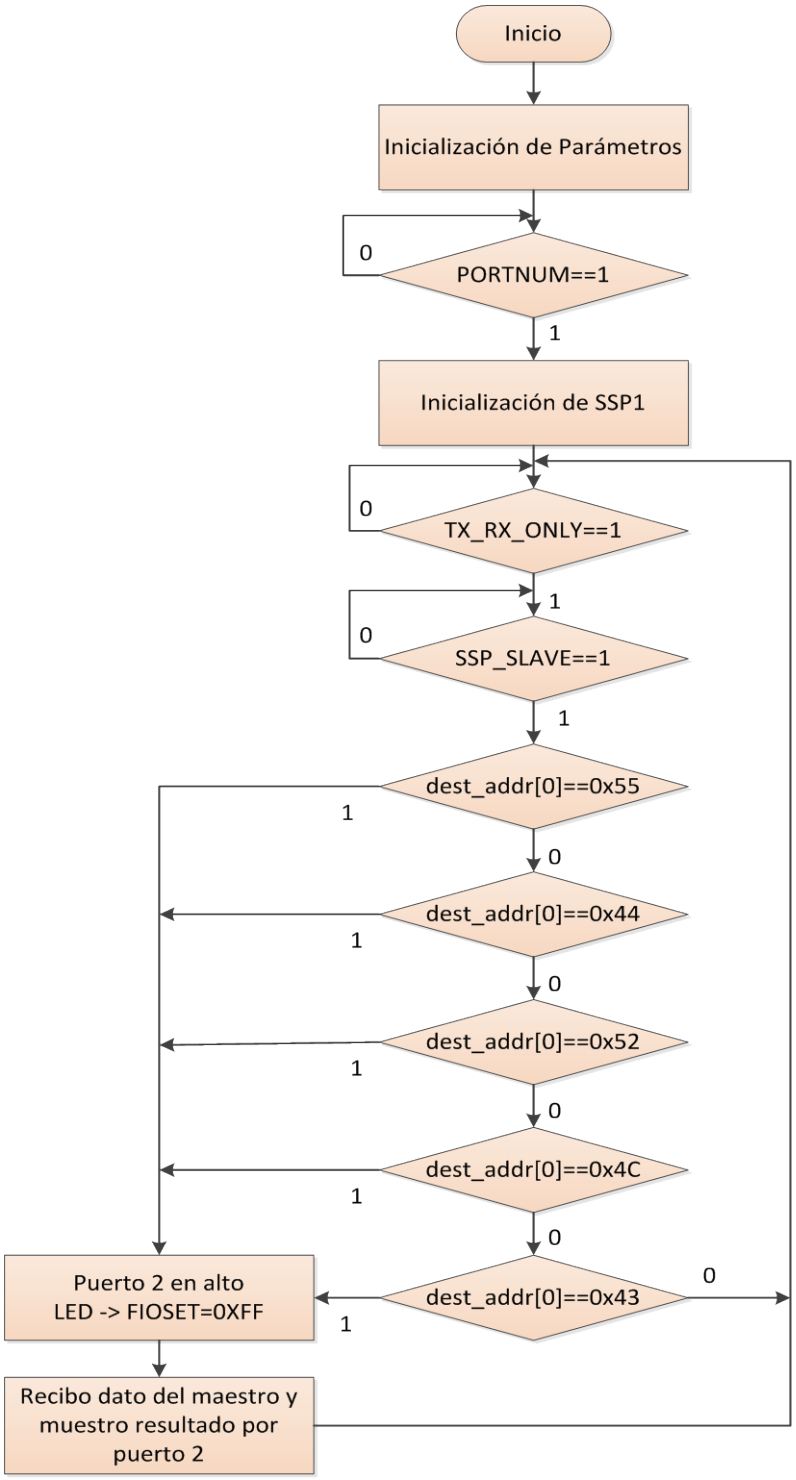


Figura 3.12: Diagrama de Flujo de la comunicación SPI de la LPC (Esclavo).

3.5.4. Descripción del algoritmo

Descripción del algoritmo MAESTRO

1. Se realiza la inicialización de parámetros.

2. Se habilita la comunicación SPI mediante la función SPI_masterinit().

3. Se habilita el puerto del LCD mediante la función LCD_puts_f.

4. Se ingresa al case del programa. En caso de que la opción KEY_ENTER sea elegida, se envía la letra C al esclavo haciendo uso de la función SPI_master_transmit(), esto quiere decir que el usuario se encuentra utilizando el joystick y ha elegido presionar el centro del mismo. Caso contrario, se pasa a la siguiente opción del case.

5. En caso de que la opción KEY_NEXT sea elegida, se envía la letra R al esclavo haciendo uso de la función SPI_master_transmit(), esto quiere decir que el usuario se encuentra utilizando el joystick y ha elegido mover el control del mismo hacia la derecha. Caso contrario, se pasa a la siguiente opción del case.

6. En caso de que la opción KEY_PREV sea elegida, se envía la letra L al esclavo haciendo uso de la función SPI_master_transmit(), esto quiere decir que el usuario se encuentra utilizando el joystick y ha elegido mover el control del mismo hacia la izquierda. Caso contrario, se pasa a la siguiente opción del case.

7. En caso de que la opción KEY_PLUS sea elegida, se envía la letra U al esclavo haciendo uso de la función SPI_master_transmit(), esto quiere decir que el usuario se encuentra utilizando el joystick y ha elegido mover el control del mismo hacia arriba. Caso contrario, se pasa a la siguiente opción del case.

8. En caso de que la opción KEY_MINUS sea elegida, se envía la letra D al esclavo haciendo uso de la función SPI_master_transmit(), esto quiere decir que el usuario se encuentra utilizando el joystick y ha elegido mover el control del mismo hacia abajo. Caso contrario, se retorna al inicio del case.

Descripción del algoritmo ESCLAVO

1. Se realiza la inicialización de parámetros.

2. Se consulta si la variable PORTNUM == 1. En caso de que la condición sea verdadera, se va al siguiente paso.
3. Se inicializa el puerto SSP1 para habilitar la comunicación SPI en dicho puerto.

4. Se consulta si TX_RX_ONLY == 1. En caso de que la condición sea verdadera, se va al siguiente paso.

5. Se consulta si SSP_SLAVE ==1. En caso de que la condición sea verdadera, se va al siguiente paso.

6. Se consulta si dest_addr[0]==0x55 (recibo letra U del maestro). En caso de que la condición sea verdadera, se ponen en alto los pines del puerto 2 mediante el comando LED-> FIOSET=0xFF, se recibe el dato enviado por el maestro y se muestra el resultado por el puerto 2. Caso contrario, se va al siguiente paso.

7. Se consulta si dest_addr[0]==0x44 (recibo letra D del maestro). En caso de que la condición sea verdadera, se ponen en alto los pines del puerto 2 mediante el comando LED-> FIOSET=0xFF, se recibe el dato enviado por el maestro y se muestra el resultado por el puerto 2. Caso contrario, se va al siguiente paso.

8. Se consulta si dest_addr[0]==0x52 (recibo letra R del maestro). En caso de que la condición sea verdadera, se ponen en alto los pines del puerto 2 mediante el comando LED-> FIOSET=0xFF, se recibe el dato enviado por el maestro y se muestra el resultado por el puerto 2. Caso contrario, se va al siguiente paso.

9. Se consulta si dest_addr[0]==0x4C (recibo letra L del maestro). En caso de que la condición sea verdadera, se ponen en alto los pines del puerto 2 mediante el comando LED-> FIOSET=0xFF, se recibe el dato enviado por el maestro y se muestra el resultado por el puerto 2. Caso contrario, se va al siguiente paso.

10. Se consulta si dest_addr[0]==0x43 (recibo letra C del maestro). En caso de que la condición sea verdadera, se ponen en alto los pines del puerto 2 mediante el comando LED-> FIOSET=0xFF, se recibe el dato enviado por el maestro y se muestra el resultado por el puerto 2. Caso contrario, se retorna al paso 4.

3.5.5. Código fuente en C

Programa principal del MAESTRO

```
/*********************************************************************
************** ESCUELA SUPERIOR POLITECNICA DEL LITORAL *************
**************       Microcontroladores Avanzados       *************
*********************************************************************
* Proyecto: Control de motor BLDC mediante comunicación SPI
* de AVR a LPC
*
* Descripción:
*
* Este archivo contiene la comunicación establecida entre tarjeta
* AVR Butterfly maestro y LPCXpresso esclavo mediante protocolo  SPI,
* para control de motor BLDC, muestra los controles por medio de LCD.
* PROGRAMACION DEL MAESTRO.
*********************************************************************
*
* Profersor: Ing. Carlos Valdivieso
* Alumnos: Andrés Maroto, Lidia Mantilla
* Fecha de creación: Abril, 2012
*
*********************************************************************
* Nombre del archivo: MasterPro.c
*********************************************************************/

// Asignación de librerías

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <avr/delay.h>
#include <inttypes.h>

#include <util/delay.h>
#include <avr/signal.h>

#include "mydefs.h"
#include "LCD_functions.h"
#include "LCD_driver.h"
#include "button.h"

void SPI_masterinit(void)
```

```
{
  DDRB=0x07;    // MOSI, SS y SCK como salida
                // SPI enable, dispositivo MASTER, Fosc/2
  SPCR=(1<<SPE)|(1<<MSTR);
}

void SPI_master_transmit(unsigned char cdata)
{
 SPDR=cdata; // coloca dato a enviar en el registro SPDR
 while(!(SPSR & (1<<SPIF)));//espera mientras completa la transmisión
}

/*********************************************************************
*                        PROGRAMA PRINCIPAL                          *
*********************************************************************/

int main(void)
{
   PGM_P statetext = PSTR("MANTILLA-MAROTO"); // Mensaje inicial

   uint8_t input;

   // Desactiva comparador análogo (power-save)
   ACSR = (1<<ACD);

   // Desactiva entradas digitales sobre PF0-2 (power-save)
   DIDR0 = (7<<ADC0D);

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

   Button_Init(); //Inicializa pin, cambios de interrupción joystick
   LCD_Init();    // Inicializa la LCD

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

   SPI_masterinit();

    while (1)
    {
        if (statetext)
            {
                LCD_puts_f(statetext, 1);
          LCD_Colon(0);
          statetext = NULL;
           }
           input = getkey();  // Lectura de los botones

           _delay_ms(500);
           _delay_ms(500);
```

```
        switch (input)
            {
          case KEY_ENTER:

                statetext = PSTR("CENTRO");
                //se envia el hexadecimal de la letra C
                //0x43
                SPI_master_transmit('C');
          break;

          case KEY_NEXT:

            statetext = PSTR("RIGHT");
                //se envia el hexadecimal de la letra R
                //0x52
                SPI_master_transmit('R');
              break;

          case KEY_PREV:

            statetext = PSTR("LEFT");
                //se envia el hexadecimal de la letra L
                //0x4c
                SPI_master_transmit('L');
          break;

          case KEY_PLUS:

            statetext = PSTR("UP");
                //se envia el hexadecimal de la letra U
                //0x55
                SPI_master_transmit('U');
                break;

          case KEY_MINUS:

            statetext = PSTR("DOWN");
                //se envia el hexadecimal de la letra D
                //0x44
                SPI_master_transmit('D');
          break;

          default:
          break;
        }
      }
     return 0;
}

/*****************************************************************
*                      FIN DEL PROGRAMA                         *
*****************************************************************/
```

Programa principal del ESCLAVO

```
/*****************************************************************
************** ESCUELA SUPERIOR POLITECNICA DEL LITORAL **************
**************       Microcontroladores Avanzados       **************
*****************************************************************
* Proyecto: Control de motor BLDC mediante comunicación SPI
* de AVR a LPC
*
* Descripción:
*
* Este archivo contiene la comunicación establecida entre tarjeta
* AVR Butterfly maestro y LPCXpresso esclavo mediante protocolo  SPI,
* para control de motor BLDC, muestra los controles por medio de LCD.
* PROGRAMACION DEL ESCLAVO.
*****************************************************************
*
* Profersor: Ing. Carlos Valdivieso
* Alumnos: Andrés Maroto, Lidia Mantilla
* Fecha de creación: Abril, 2012
*
*****************************************************************
* Nombre del archivo: ssptest.c
* Proyecto realizado en base al proyecto NXP LPC17xx SSP example de
* NXP.
*****************************************************************/

// Asignación de librerías
#include <cr_section_macros.h>
#include <NXP/crp.h>

/*Variable para almacenar el valor de entrada CRP.
Se realizará de forma automática cuando el "código de habilitación de
lectura protegida" esté seleccionado. Ver encabezado crp.h para mayor
información*/

__CRP const unsigned int CRP_WORD = CRP_NO_CRP;

#include "LPC17xx.h"  //Definiciones del LPC13xx
#include "ssp.h"

/*Tenga cuidado con el número de puerto y el número de ubicación, ya
que algunas localidades de la ubicación pueden no existir en ese
puerto.*/

#define PORT_NUM            1 // Asignación de constantes
#define LOCATION_NUM        0
#define LED LPC_GPIO2

/*Asignación de arreglos para recepción y transmisión de los datos*/

uint8_t src_addr[SSP_BUFSIZE];
uint8_t dest_addr[SSP_BUFSIZE];
```

```
/*****************************************************************
*                        PROGRAMA PRINCIPAL                      *
*****************************************************************/

int main (void)
{
  uint32_t portnum = PORT_NUM;
  uint32_t j = 0;

 /*La función SystemClockUpdate() actualiza la variable
  SystemFrequency*/

  SystemClockUpdate();
  LED -> FIODIR = 0xFF; // Define PORTD como salida

  while ( 1 )
  {
          if ( portnum == 0 )
        SSP0Init();                 /* Inicializa el puerto SSP0 */
          else if ( portnum == 1 )
             SSP1Init();                /* Inicializa el puerto SSP1 */

          dest_addr[0] = 0;

     #if TX_RX_ONLY
       /* Para la comunicación entre tarjetas, una tarjeta es
          establecida como maestro transmisor, la otra se
          establece como esclavo receptor.*/

          #if SSP_SLAVE
            /* Esclavo recibe datos del maestro */
            SSPReceive( portnum, (uint8_t *)dest_addr, 1);

            //LED -> FIOCLR =  dest_addr[0];
            //for(j = 2500000; j > 0; j--);
            LED -> FIOSET = 0xFF;

            if (dest_addr[0]!= 0xFF)
            {
                    if(dest_addr[0]==0x55)
                    {
                            LED -> FIOCLR = 1 << 0;
                            for(j = 2500000; j > 0; j--);
                            LED -> FIOSET = 0xFF;
                    }

                    if(dest_addr[0]==0x44)
                    {
                            LED -> FIOCLR = 1 << 1;
                            for(j = 2500000; j > 0; j--);
                            LED -> FIOSET = 0xFF;
                    }
```

```
                    if(dest_addr[0]==0x52)
                    {
                            LED -> FIOCLR = 1 << 2;
                            for(j = 2500000; j > 0; j--);
                            LED -> FIOSET = 0xFF;
                    }

                    if(dest_addr[0]==0x4c)
                    {
                            LED -> FIOCLR = 1 << 3;
                            for(j = 2500000; j > 0; j--);
                            LED -> FIOSET = 0xFF;
                    }

                    if(dest_addr[0]==0x43)
                    {
                            LED -> FIOCLR = 1 << 4;
                            for(j = 2500000; j > 0; j--);
                            LED -> FIOSET = 0xFF;
                    }
             }
            #else
              /* Maestro transmite */
              SSPSend( portnum, (uint8_t *)src_addr, SSP_BUFSIZE);
            #endif
            #else
            /* TX_RX_ONLY = 0, es una prueba de bucle de retorno
          Interno dentro del periférico SSP, sirve para
          comunicarse con una serie EEPROM. */

            #if LOOPBACK_MODE
              LoopbackTest( portnum, LOCATION_NUM );
            #else
              SEEPROMTest( portnum, LOCATION_NUM );
            #endif
            #endif
  }
  return 0;
}

/*****************************************************************
*                       FIN DEL PROGRAMA                  *   /
```

CAPÍTULO 4

DESARROLLO Y SIMULACIÓN DEL PROYECTO

La finalidad de este capitulo es de mostrar los ejercicios q se realizaron como prueba para llevar a cabo el proyecto. Estas pruebas eran de llevar a cabo la comunicación SPI entre dos hardware diferentes que son el AVR Butterfly y el LPC1769.

4.1. Descripción de los ejercicios.

4.1.1. Contador de 0 a 9 entre LPC - LPC

Descripción.

En este ejercicio, una tarjeta LPC 1769 funciona como el maestro de la comunicación serial SPI y otro LPC 1769 trabaja como esclavo.

Luego de haber sido cargados los códigos en cada LPC, el sistema empieza a funcionar como un contador de 0 a 9. Se configura el puerto dos de la tarjeta Esclavo como salida, donde se conecta un display de siete segmentos y otro display de 7 segmentos en la salida del puerto 2 del maestro, con el objetivo de observar que la transmisión de datos mediante comunicación SPI se realiza en forma correcta.

Lista de Materiales:

- Un Protoboard
- Dos LPC 1769.
- Cuatro resistencias de 1K Ω y dieciséis resistencias de 330 Ω.
- 2 Display de siete segmentos (cátodo común).
- Una batería de 3.6V
- Un switch de 4

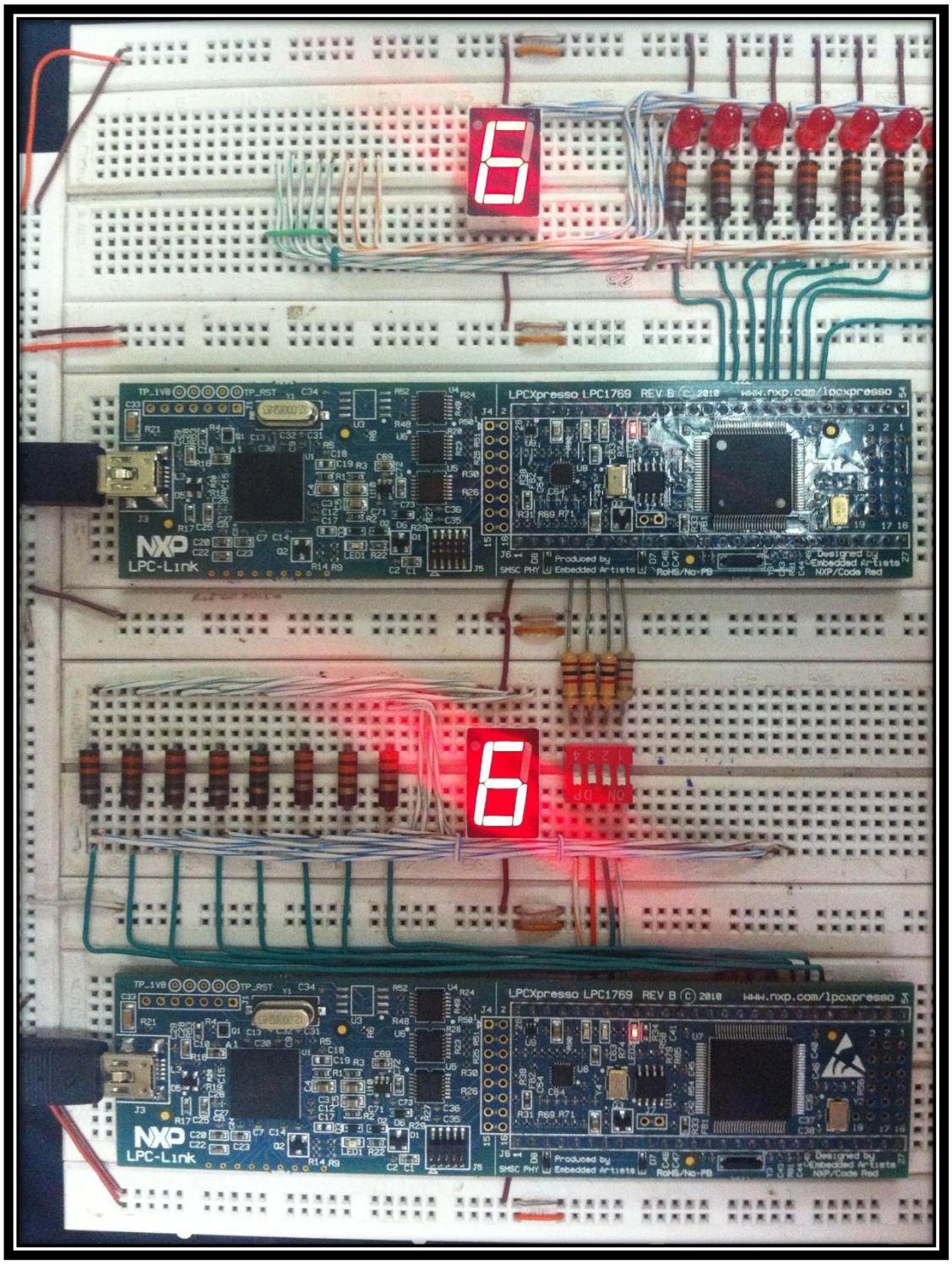


Figura 4.1 Implementación del contador de 0 a 9 entre LPC – LPC

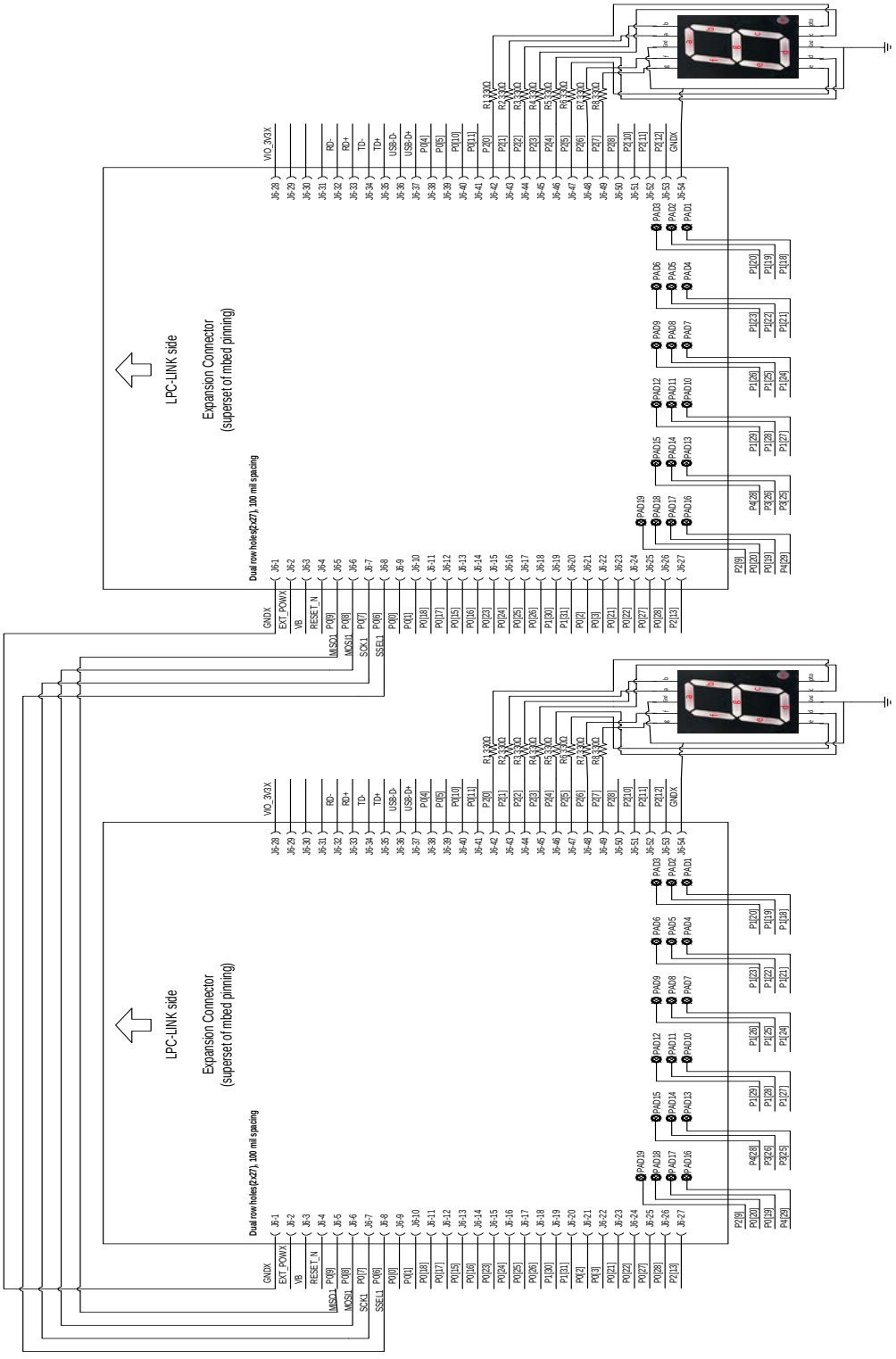


Figura 4.2 Diagrama de conexiones entre LPC – LPC.

Conclusión.

Podemos concluir que la comunicación por medio del protocolo SPI se puede realizar desde una tarjeta LPC maestro hacia una tarjeta LPC esclavo mediante la conexión de 4 puertos específicos en cada tarjeta (MOSI, MISO, SCK, SSEL), permitiendo así el envío y recepción de los datos. Para este ejemplo el maestro envió una secuencia numérica de 0 a 9 para ser mostrada por el esclavo por medio de un display de 7 segmentos y 7 leds. Si se pierde la comunicación con el maestro, ya sea por apagar la tarjeta maestro o desconectarla del esclavo, el esclavo se queda en el último valor recibido.

4.1.2. Contador de 0 a 9 entre AVR – LPC

Descripción

El AVR BUTTERFLY trabaja como maestro de la comunicación SPI y el LPC1769 trabaja como esclavo de la comunicación. Se configura en el puerto D del AVR Butterfly como salida, donde va conectado un display donde se observan de manera ascendente los números del 0 al 9. Tambien en el puerto dos de la tarjeta LPC1769 se configura como salida cuya finalidad es de conocer el dato que esta almacenado en el MASTER y comprobar que la comunicación SPI se realiza de manera correcta.

Lista de Materiales:

- Un Protoboard
- Un LPC 1769.
- Un AVR Butterfly
- Cuatro resistencias de 1K Ω y dieciséis resistencias de 330 Ω.
- 2 Display de siete segmentos (cátodo común).
- Una batería de 3.6V
- Un switch de 4

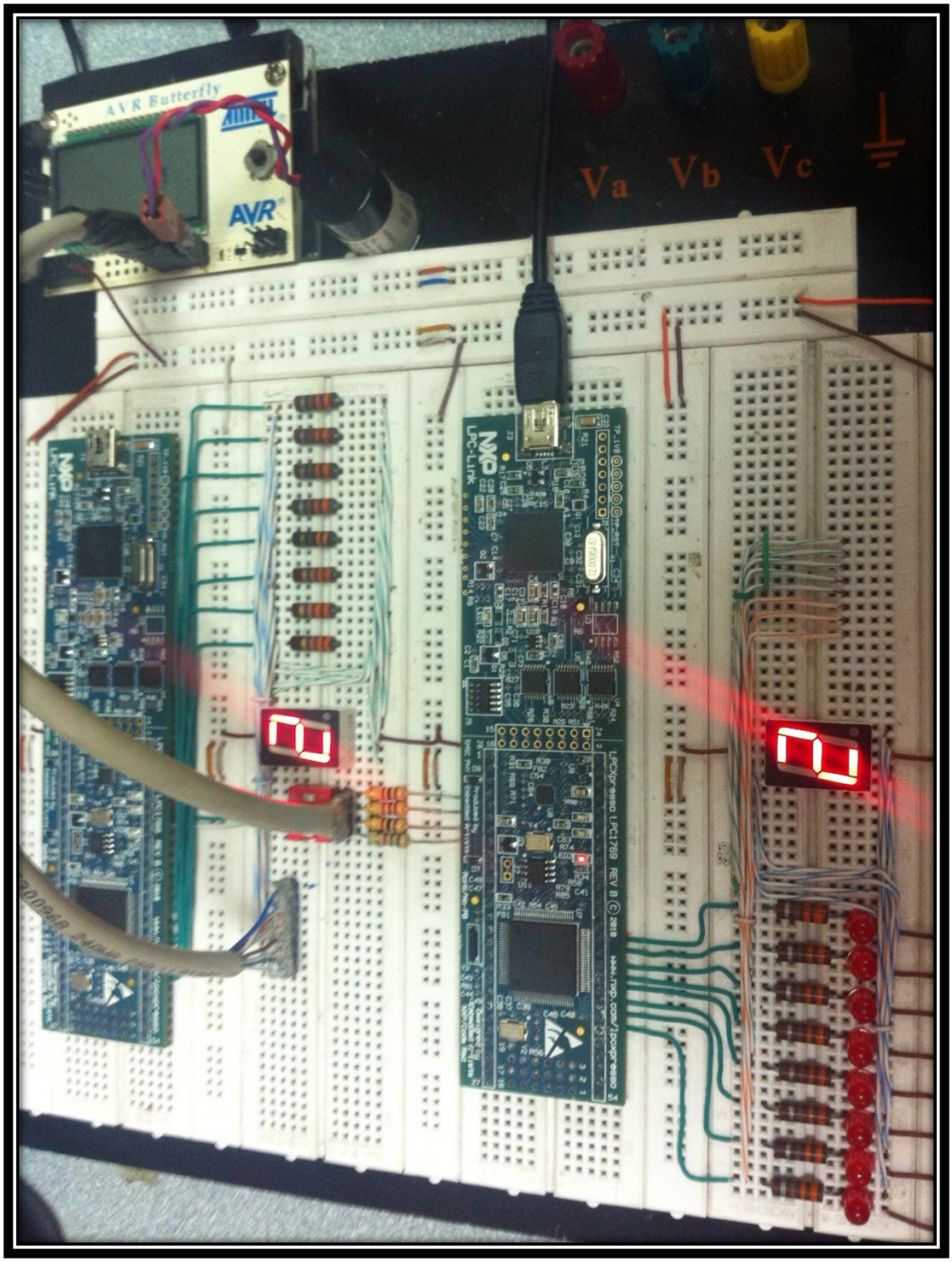


Figura 4.3 Implementación del contador de 0 a 9 entre AVR – LPC

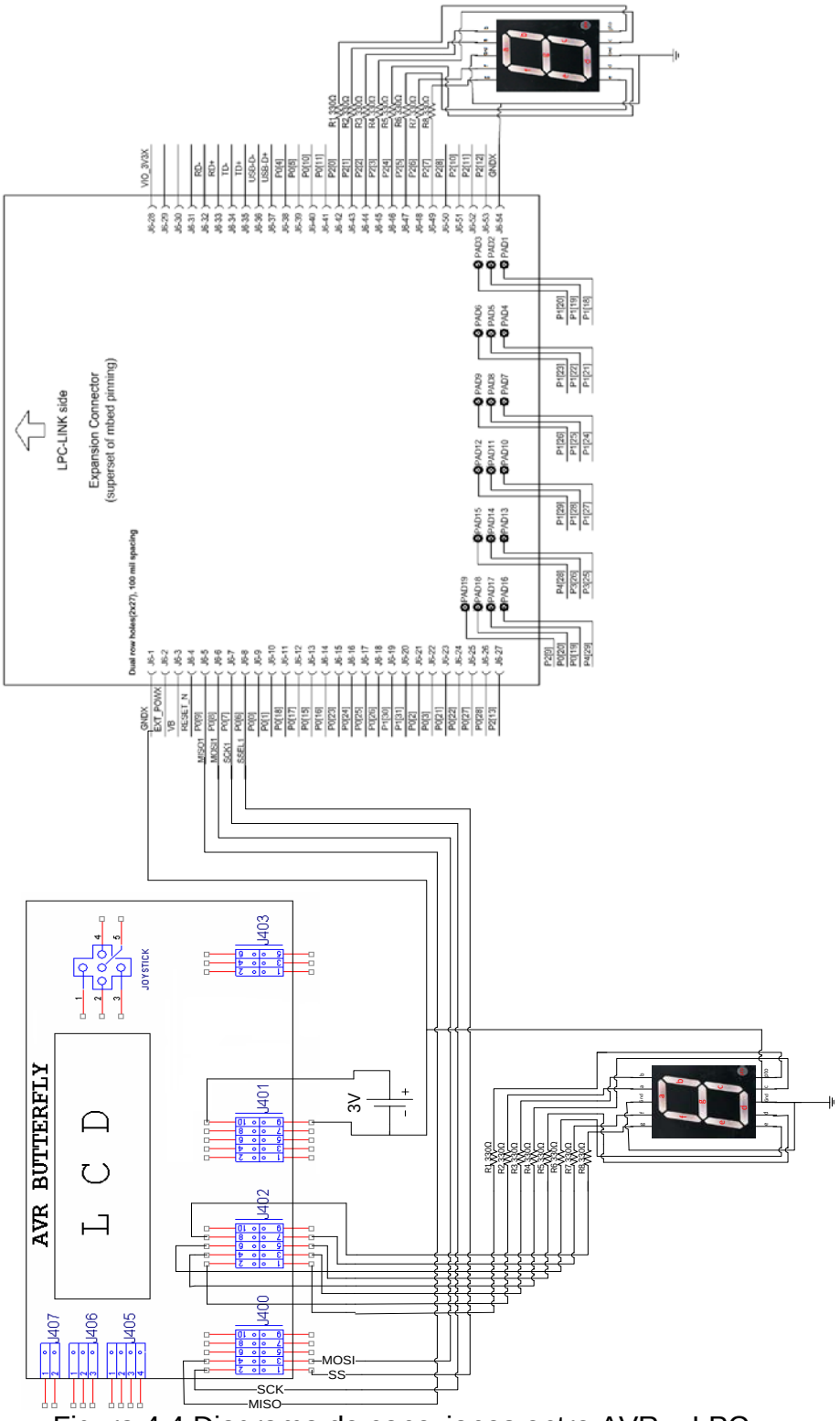


Figura 4.4 Diagrama de conexiones entre AVR – LPC

Conclusión.

Podemos concluir que la comunicación SPI se puede realizar entre tarjetas de diferentes fabricantes sin ningún problema. En este caso se emplea la AVR Butterfly como tarjeta maestro y la tarjeta LPCXpresso como esclavo, y la transmisión y recepción de datos se realiza mediante el PORTD de la tarjeta AVR y los 4 puertos específicos de la tarjeta LPCXpresso (MOSI, MISO, SCK, SSEL), por último se presentan los datos por medio del puerto GPIO2 (puerto 2) de la tarjeta LPCXpresso, al cual se encuentran conectados el display de 7 segmentos y los 7 leds.

4.1.3. Contador de 0 a 9 entre LPC – AVR

Descripción

El LPC1769 trabaja como maestro de la comunicación SPI y el AVR Butterfly trabaja como esclavo de la comunicación. Se configura en el puerto dos de la LPC como salida, donde va conectado un display donde se observan de manera ascendente los números del 0 al 9. También en el puerto D de la tarjeta AVR Butterfly se configura como salida cuya finalidad es de conocer el dato que esta almacenado en el MASTER y comprobar que la comunicación SPI se realiza de manera correcta.

Lista de Materiales:

- Un Protoboard
- Un LPC 1769.
- Un AVR Butterfly
- Cuatro resistencias de 1K Ω y dieciséis resistencias de 330 Ω.
- 2 Display de siete segmentos (cátodo común).
- Una batería de 3.6V
- Un switch de 4

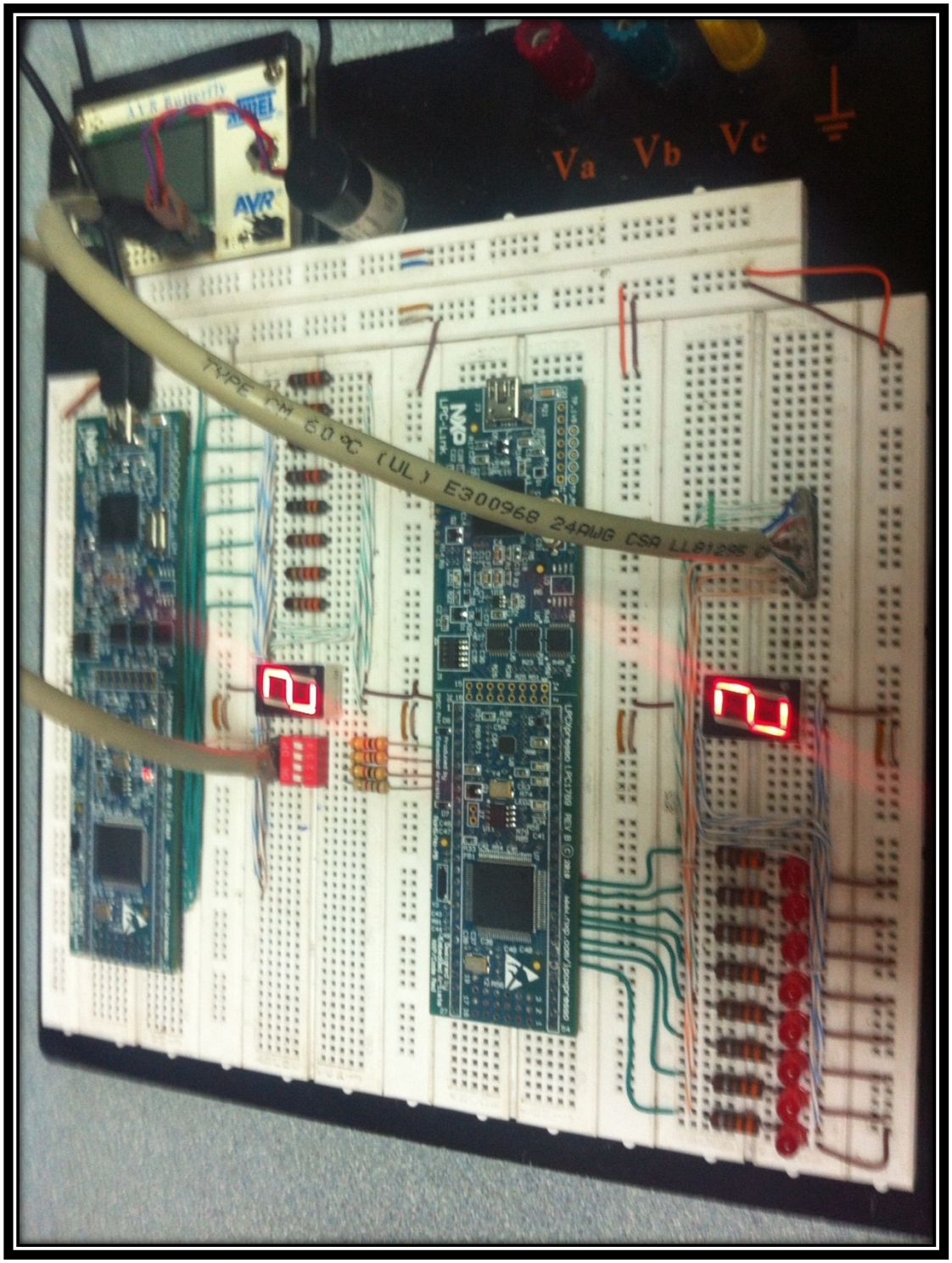


Figura 4.5 Implementación del contador de 0 a 9 entre LPC – AVR

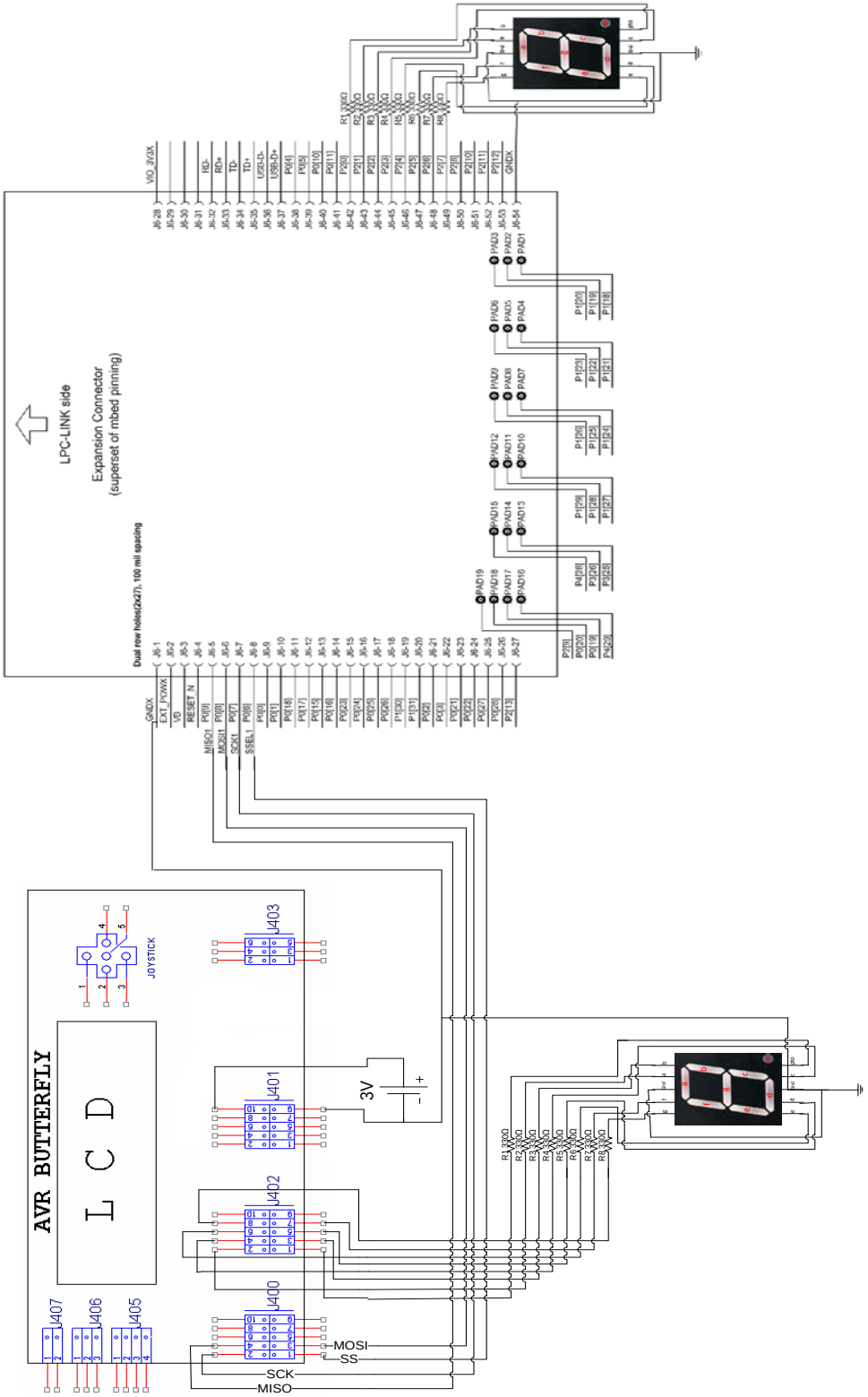


Figura 4.6 Diagrama de conexiones entre LPC – AVR

Conclusión.

Podemos concluir que la comunicación SPI se puede realizar entre tarjetas de diferentes fabricantes, independientemente de cuál asignemos como maestro o esclavo. En este caso utilizamos la tarjeta LPCXpresso como maestro y la AVR Butterfly como tarjeta esclavo, y la transmisión y recepción de datos se realiza mediante los 4 puertos específicos de la tarjeta LPCXpresso (MOSI, MISO, SCK, SSEL) y el PORTB de la AVR Butterfly, por último se presentan los datos por medio del puerto PORTD de la tarjeta AVR Butterfly, al cual se encuentran conectados el display de 7 segmentos y los 7 leds.

4.1.4. Control de motor BLDC entre AVR – LPC, mediante Joystick

Descripción

El AVR BUTTERFLY trabaja como maestro de la comunicación SPI, para nuestro controlador mediante joystick del motor BLDC. El LPC1769 trabaja como esclavo de la comunicación, en donde los pines del puerto dos son configurados como salidas y todos en estado alto.

El joystick utilizado del AVR Butterfly posee cinco movimientos que son: arriba, abajo, izquierda, derecha y centro. Para nuestro proyecto se configuró mediante programación, el envío de las 4 órdenes cuyo reconocimiento se realiza en un estado bajo durante un periodo de tiempo en la LPC1769

En la tabla 4.1 se muestra los pines configurados como salidas de la tarjeta LPC1769 y los pines configurados como entradas de la tarjeta LPC1114 para enviar y recibir respectivamente estas órdenes:

JOYSTICK (Movimiento)	**ORDENES**	**LPC1769**	**TARJETA GRANDE (LPCXpresso Motor Control Board)**
Arriba ≡ UP	**Aumentar Velocidad**	Port2 [3]	PIO3_3
Abajo ≡ DOWN	**Disminuir Velocidad**	Port2 [2]	PIO3_2
Derecha ≡ RIGHT	**Invertir Giro**	Port2 [1]	PIO3_1
Izquierda/centro ≡ LEFT/CENTRO	**ON/OFF**	Port2 [0]	PIO3_0

Tabla 4.1 Asignación de pines para el funcionamiento del motor BLDC.

Lista de Materiales:

- AVR Butterfly.

- LPC 1769.

- Cuatro resistencias de 1K Ω y cuatro resistencias de 330 Ω.

- Cuatro LEDs.

- Una batería de 3.6V

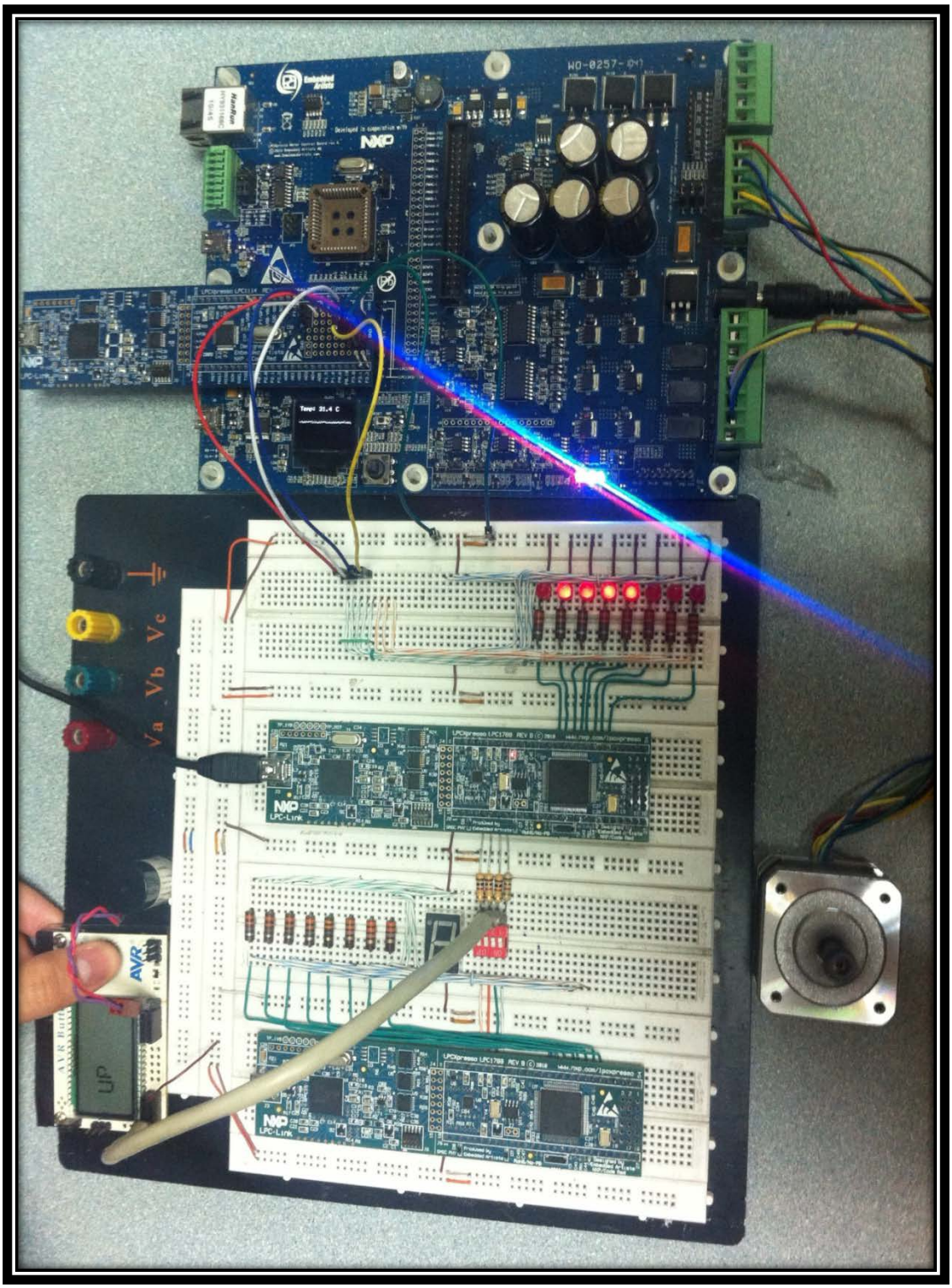


Figura 4.7 Movimiento del joystick hacia ARRIBA

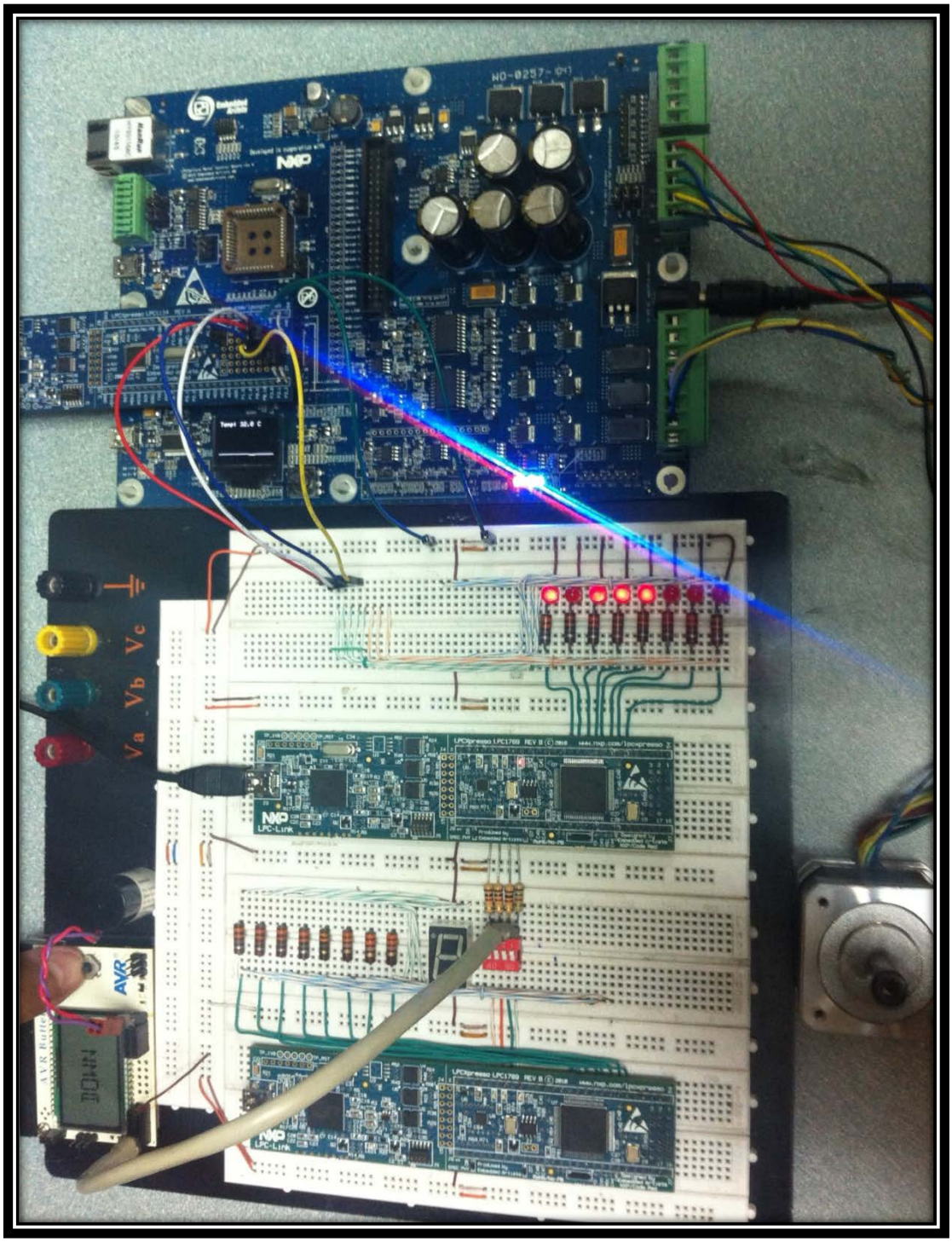


Figura 4.8 Movimiento del joystick hacia ABAJO

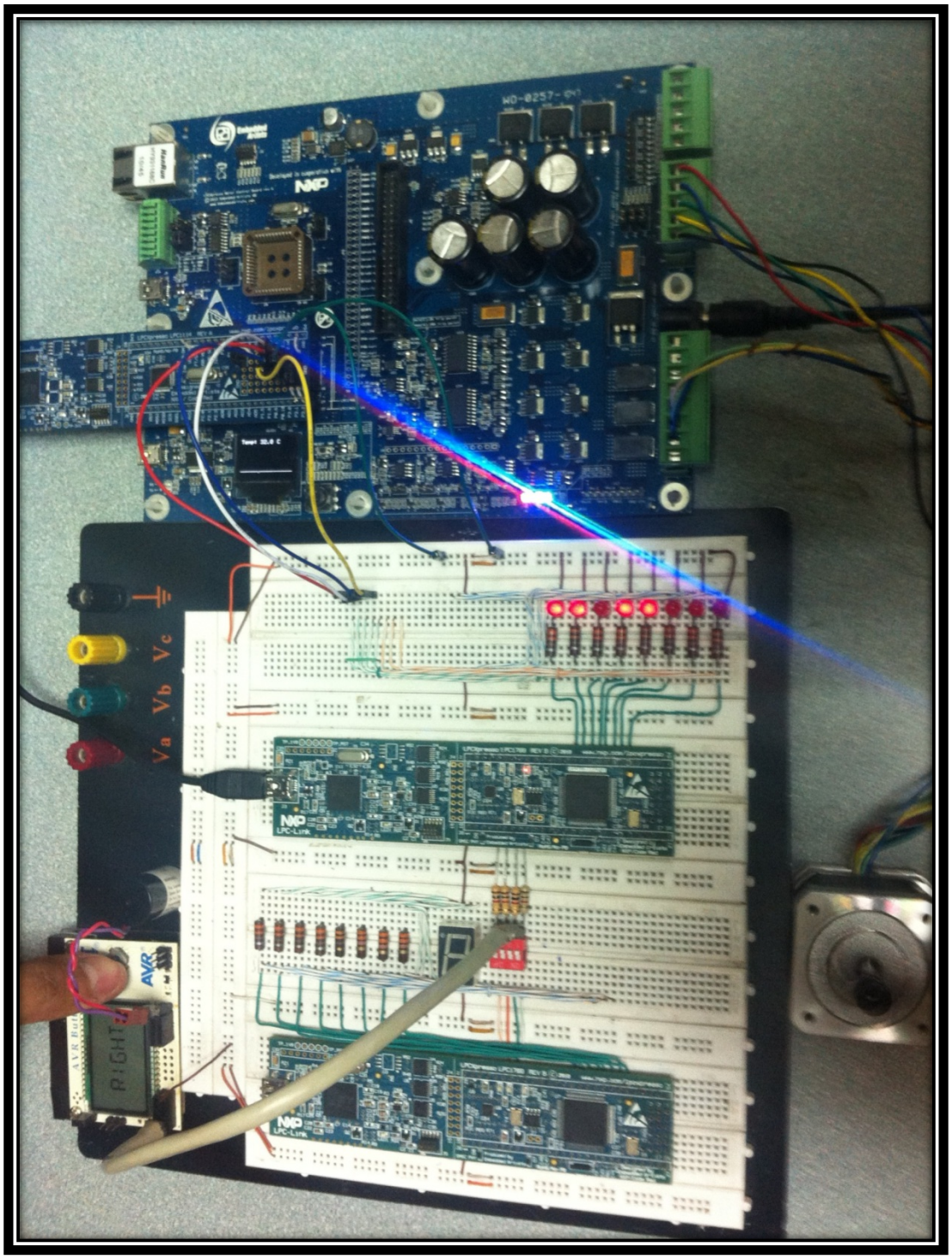


Figura 4.9 Movimiento del joystick hacia la DERECHA

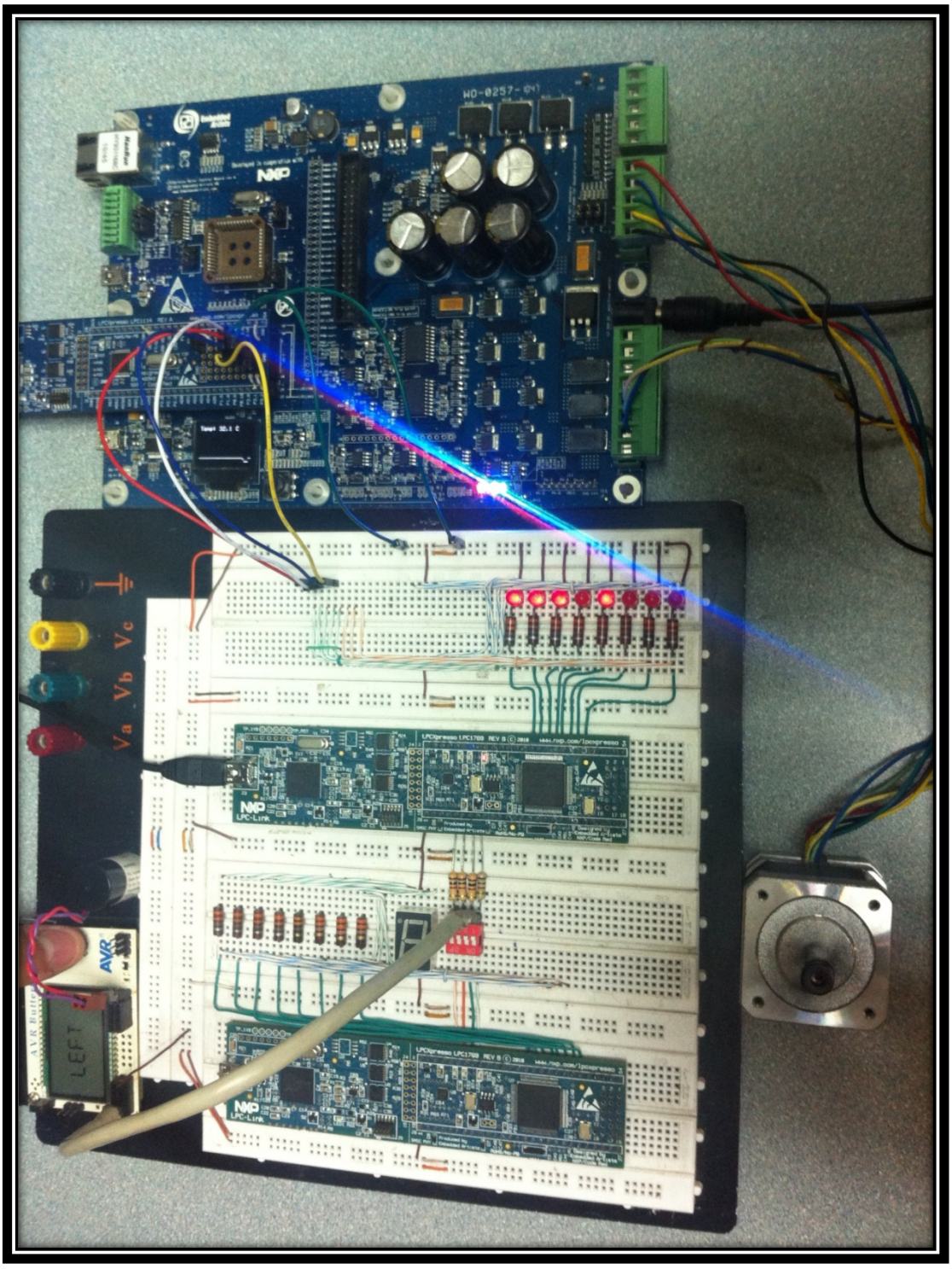


Figura 4.10 Movimiento del joystick hacia la IZQUIERDA/CENTRO

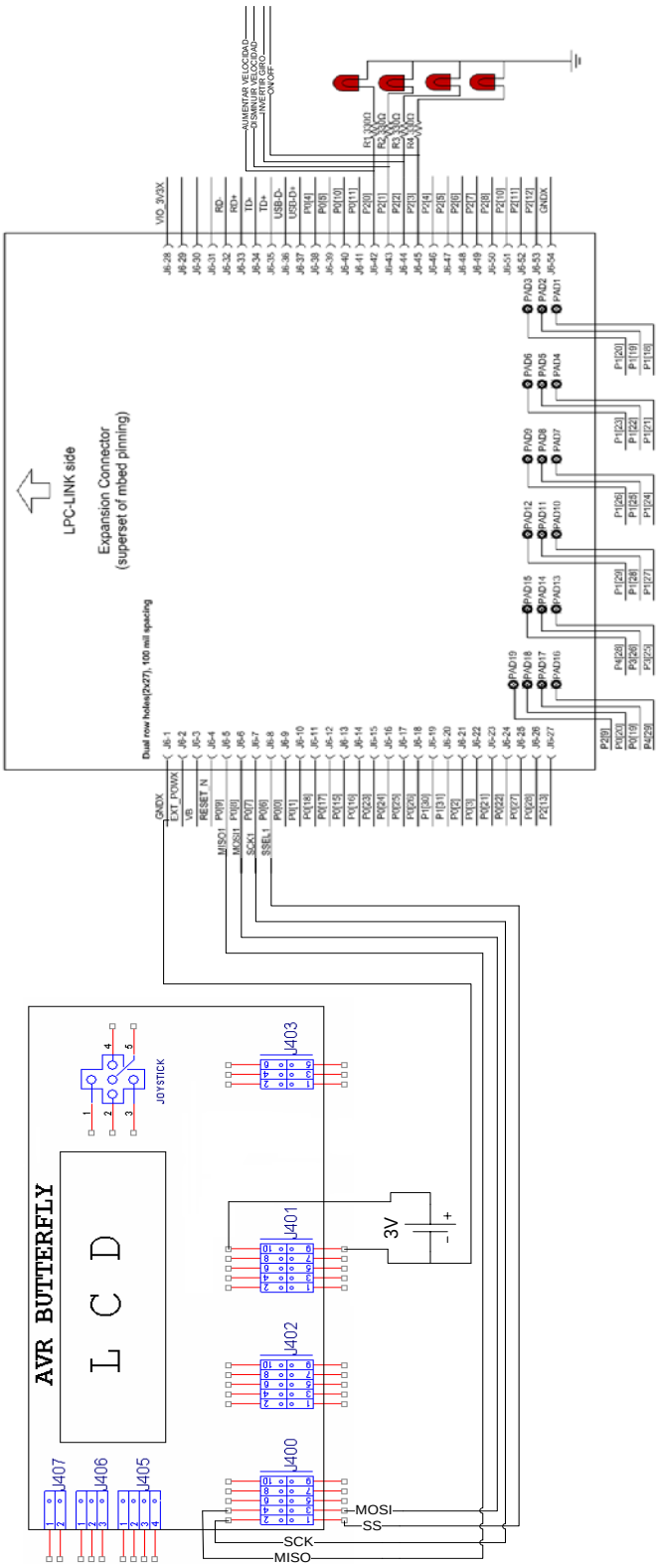


Figura 4.11 Diagrama de conexiones para el controlador de motor BLDC

Conclusión.

Podemos concluir que se puede realizar el control de un motor BLDC utilizando la tarjeta AVR Butterfly como maestro y la tarjeta LCPXpresso como esclavo por medio del protocolo de comunicación SPI, el cual nos permite la interacción entre ambas tarjetas. La tarjeta AVR Butterfly tiene un joystick de control integrado en su estructura, lo que la convierte en una tarjeta que permite realizar control, mientras que la LPCXpresso nos ayuda con la conexión hacia el motor debido a que es compatible con la tarjeta LPC1114 (también perteneciente al fabricante LPCXpresso), a la cual se puede conectar el motor BLDC.

CONCLUSIONES GENERALES

Basado en lo expuesto anteriormente en esta tesina podemos obtener las siguientes conclusiones:

1) Se utilizó el protocolo de comunicación serial sincrónica SPI para trasmisión – recepción mediante la asignación maestro – esclavo, lo cual nos permitió la comunicación entre tarjetas AVR Butterfly y LPCXpresso por medio de un mismo canal, optimizando el uso de los microcontroladores (ATMega169 y LPC1769), logrando así la interacción entre distintos elementos de diferentes fabricantes, reduciendo cableado y programación de pines.
2) Los microcontroladores de los fabricantes ATMega y LPCXpresso son de gran utilidad para el control de motores y otras funciones debido a que tienen gran capacidad de almacenamiento, poco retardo en sus tiempos de respuesta al momento de realizar el

envío y/o recepción de datos y su sencilla programación en comparación con algoritmos complejos que poseen otros tipos de microcontroladores, lo que los convierte en elementos de gran utilidad en diferentes campos de estudio.

3) La tarjeta AVR Butterfly, al poseer una pantalla LCD y un joystick integrados a su estructura, se convierte en un microcontrolador muy funcional para la operación de giro y velocidad de un motor BLDC, lo que nos permitió aprovechar espacio en la circuitería e implementación del proyecto puesto que en otros microcontroladores estos elementos no vienen incorporados, además se demostró que, pese a que se realizó la comunicación entre dos tarjetas de distinto fabricante, se pudo lograr la comunicación entre ellas, lo que las convierte en herramientas de amplia interacción con otros dispositivos similares.
4) Mediante la implementación de los ejercicios y el proyecto pudimos comprender y tener una visión más amplia acerca del funcionamiento de los microcontroladores ATMega169 y LPC1769 y los motores BLDC, así como la forma en que se realiza la comunicación entre dispositivos mediante protocolo SPI e identificar de manera clara la función que desempeñan el maestro y el esclavo en el dispositivo en el cual han sido configurados.

5) En general, la aplicación de microcontroladores en el uso industrial y otras áreas es un campo que se encuentra innovando y cambiando constantemente, y cada vez podemos observar nuevos modelos de microcontroladores con mayor capacidad, herramientas incorporadas y programación más sencilla. Por cumplir con todos estos parámetros, podemos concluir que los microcontroladores que han sido objeto de estudio en esta tesina son herramientas nuevas y con gran ventaja sobre los microcontroladores ya conocidos y estudiados en cursos anteriores durante nuestro proceso de desarrollo académico.

RECOMENDACIONES

Las recomendaciones que logramos obtener del proyecto son las expuestas a continuación:

1) Se requiere trabajar con una fuente de alimentación de 3V o más para que se pueda realizar exitosamente la comunicación SPI entre las tarjetas. Al trabajar con una fuente de menor voltaje, se puede cometer errores en la comunicación y el maestro podría estar enviando datos erróneos al esclavo, ejecutando mal la transmisión de los datos.

2) Al momento de cablear es importante verificar continuidad entre los puntos que están siendo conectados para de esta manera evitar fallos al momento de hacer funcionar nuestro proyecto, además se requiere utilizar una sola tierra de referencia y un solo punto Vcc (3V) para ahorro de espacio y evitar problemas con el voltaje aplicado a los pines de las tarjetas, las cuales son sensibles a las variaciones bruscas de voltaje.
3) Para la programación y uso de los pines del microcontrolador es necesario revisar el datasheet de cada uno de ellos y verificar qué pines se encuentran reservados por el fabricante, qué pines pueden ser usados como E/S y qué pines sirven para funciones específicas, además se requiere verificar que se encuentren correctamente conectados los 4 puertos que permiten la comunicación SPI entre maestro y esclavo para poder realizar la transmisión.

ANEXOS

Guía para programar el AVR Butterfly

Para programar el Kit AVR Butterfly mediante una PC, se recomienda al usuario seguir el procedimiento que se detalla a continuación:

1. Conectar el cable para la interfaz serial RS-232 entre la PC y el AVR Butterfly, tal como se indica en la Figura 1.

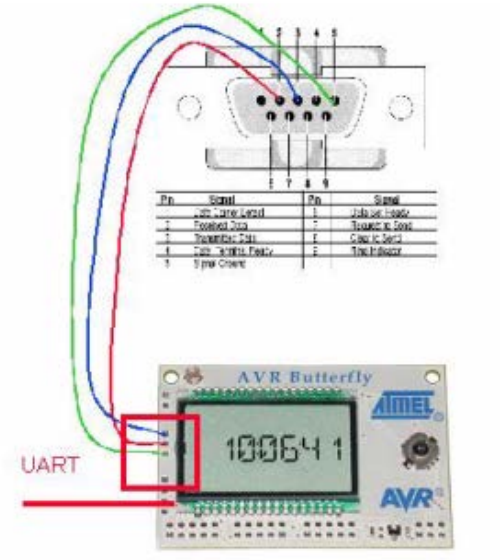


Figura 1.

2. En la PC, ejecutar AVR Studio 4.

3. Presionar el joystick del AVR Butterfly en el centro y mantenerlo en esa posición.

4. Energizar el AVR Butterfly con una fuente de voltaje de 3 V.

5. En AVR Studio 4, en la barra de herramientas, abrir el menú Tools. En este menú se visualiza la función AVRProg, tal como se observa en la Figura 2 entonces, seleccionar dicha función para que aparezca la ventana AVRprog que se observa en la Figura.

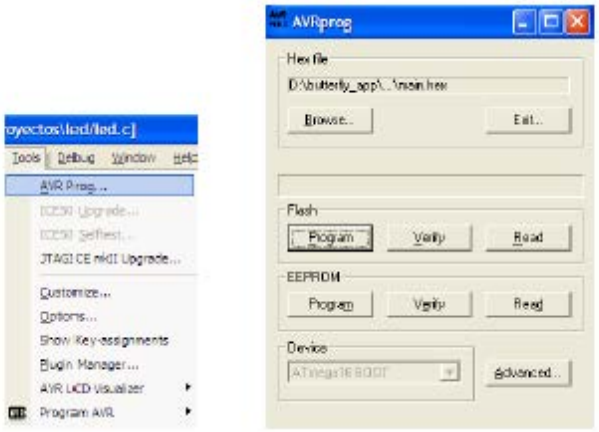


Figura 2.

6. Quitar la presión que se mantiene sobre el joystick del AVR Butterfly desde el paso 3.

7. Hacer clic en el botón Browse de la Ventana AVRprog, para localizar y cargar el archivo HEX generado con la compilación en el directorio del proyecto, tal como se aprecia en la Figura 3.

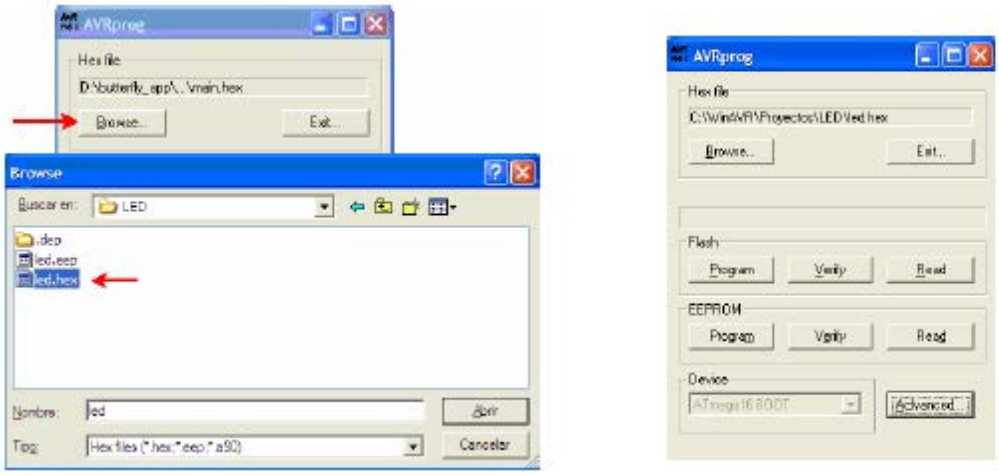


Figura 3.

8. Hacer clic en el botón Advanced de la ventana AVRprog, para acceder a las opciones para programar los Lock Bits, tal como se aprecia en la Figura 4. Hacer clic en el botón Close.

Figura 4.

9. Hacer clic en el botón Program de la ventana AVRprog, para programar el microcontrolador ATmega169V del AVR Butterfly y actualizarlo con la nueva aplicación. Entonces, se apreciará el proceso de grabación del microcontrolador tal como en la Figura 5.

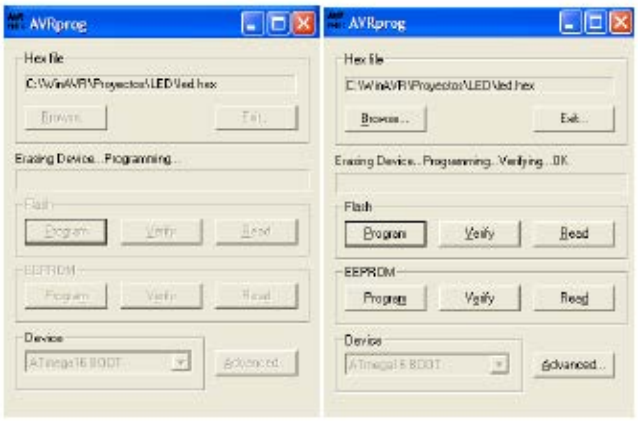


Figura 5.

10. Presionar el botón Exit de la ventana AVRprog, para salir del modo de programación.

Para que el AVR Butterfly empiece a funcionar, el usuario debe mover el Joystick hacia arriba haciendo que la MCU salte del sector de arranque (boot loader) hacia el sector de la aplicación (programa del usuario), y entonces se ejecute el programa escrito por el usuario. Desde aquí, el usuario puede evaluar el desempeño del software-hardware del AVR Butterfly.

BIBLIOGRAFÍA

[1] Motor con escobillas,

http://3.bp.blogspot.com/_9Ru4Uz9NNT4/SwmaTRuCCqI/AAAAAAAAAIk/rxd-huckAOQ/s1600/20070822klpingtcn_58.Ees.SCO.png,

Fecha de consulta febrero 2012

[2] Motor DC sin escobillas,

http://www.photomobiware.com/images/servo%20motor5.jpg,

Fecha de consulta febrero 2012

[3] Conexión entre un dispositivo maestro y un esclavo,

http://quierobits.com/wp-content/uploads/2011/02/spi_topology1.png,

Fecha de consulta marzo 2012

[4] Transferencia de datos SPI Modo0

http://avrhelp.mcselec.com/hardware_spi_cpha0.jpg,

Fecha de consulta marzo 2012

[5] Transferencia de datos SPI Modo1

http://avrhelp.mcselec.com/hardware_spi_cpha1.jpg,

Fecha de consulta marzo 2012

[6] Manual de Usuario del LPC1769,

http://www.nxp.com/documents/user_manual/UM10360.pdf,

Fecha de consulta marzo 2012

[7] Motor BLDC

http://www.dspace.espol.edu.ec/bitstream/123456789/14642/1/Control%20de%20velocidad%20por%20cambio%20de%20frecuencia.pdf,

Fecha de consulta marzo 2012

[8, 9, 10, 11, 12] AN857, Control de Motor DC

http://ww1.microchip.com/downloads/en/appnotes/00857a.pdf,

Fecha de consulta febrero 2012

[13] Motor sin escobillas,

http://www.edn.com/photo/284/284995-Hardware_controlled_brushless_dc_motors_ease_the_burden_on_CPUs_figure_1.gi,

Fecha de consulta febrero 2012

[14] Motor con sensor con efecto Hall,

http://www.psocdeveloper.com/fileadmin/cytest/BLDC2.jpg,

Fecha de consulta febrero 2012

[15] Esquema de los periféricos externos e internos del AVR Butterfly,

http://bibing.us.es/proyectos/abreproy/11175/fichero/vol1%252Frv_T4_Hardware_v03d.pdf, Fecha de consulta abril 2012

[16] Distribución de los periféricos externos en la tarjeta AVR Butterfly,

http://bibing.us.es/proyectos/abreproy/11175/fichero/vol1%252Frv_T4_Hardware_v03d.pdf, Fecha de consulta abril 2012

[17] Datasheet ATMega169,

http://www.atmel.com/Images/doc2514.pdf,

Fecha de consulta marzo 2012

[18] Introducción a LPCXpresso y repaso de lenguaje C – Seminario de sistemas embebidos – FIUBA Autor Alan Kharsansky Abril 2011,

http://laboratorios.fi.uba.ar/lse/seminario/material-2011/Sistemas_Embebidos-2011_2doC-Intro_a_LPCXpresso_y_repaso_lenguaje_C-Kharsansky.pdf

Fecha de consulta abril 2012

[19] Datasheet LPC familia 17xx, Fecha de consulta marzo 2012

[20] Barret S., Pack D., Atmel AVR Microcontroller Primer, Programming and Interfacing, Capítulo 2: Serial Peripheral Interface, pag 34-38, 2008,

Fecha de consulta marzo 2012

[21] López Chau, A., Microcontroladores AVR, Configuración total de periféricos, Capítulo SPI, primera edición, 2006,

Fecha de consulta marzo 2012

[22] Pardue Joe, C Programming for Microcontrollers, Capítulo 2: Quick Start Guide, pag 17-27, 2005.

Fecha de consulta marzo 2012

[23] Using SPI on an AVR

http://www.rocketnumbernine.com/2009/04/26/using-spi-on-an-avr-1

Fecha de consulta marzo 2012

[24] ATMEL, Hoja de Datos del microcontrolador ATMega169.

http://www.datasheetcatalog.org/datasheet/atmel/2514S.pdf

Fecha de consulta marzo 2012

[25] Kit de desarrollo AVR Butterfly, Desarrollo de guía de prácticas de laboratorio y tutoriales.

http://www3.espe.edu.ec:8700/handle/21000/424

Fecha de consulta marzo 2012

[26] TEXTO, Microcontroladores AVR, configuración total de periféricos, escrito por Asdrúbal López Chau.

http://books.google.com.ec/books?id=wSEjrFxjUr4C&pg=PA113&lpg=PA113&dq=REGISTROS+DEL+SPI+avr&source=bl&ots=cvtLw2_EWd&sig=hIK7QeUWK8cznNjW2SU9zxcZq30&hl=es&sa=X&ei=3dWJT-Buk4LxBL72lPcJ&ved=0CCAQ6AEwAA#v=onepage&q=REGISTROS%20DEL%20SPI%20avr&f=false

Fecha de consulta marzo 2012

[27] COMUNICACIÓN SERIAL SINCRONICA (SPI)

http://www.google.com.ec/url?sa=t&rct=j&q=comunicacion%20spi&source=web&cd=1&sqi=2&ved=0CCoQFjAA&url=http%3A%2F%2Fproyecto-test-hm1.googlecode.com%2Ffiles%2FCap_No_05.pdf&ei=XD1vT9PzNpOCtgfnrN28Bg&usg=AFQjCNEI_PptMuhuxCL6KZkJcCCnTjL4rA&cad=rja

Fecha de consulta marzo 2012

[28] Comunicacion Serial Sincrona (SPI) – TLC5615 DAC,

http://jealness.hostzi.com/?p=410

Fecha de consulta marzo 2012

[30] The Serial Peripheral Interface (SPI),

http://avrbeginners.net/architecture/spi/spi.html

Fecha de consulta abril 2012

[31] Curso de Redes de Microcontroladores PIC (Protocolo SPI),

http://www.i-micro.com/pdf/articulos/spi.pdf

Fecha de consulta abril 2012

[32] Introducción a los buses de comunicación,

http://www.olimex.cl/present.php?page=buses

Fecha de consulta abril 2012

[33] Using the SPI protocol,

http://avrhelp.mcselec.com/index.html?using_the_spi_protocol.htm

Fecha de consulta abril 2012

[34] REGISTROS DEL SPI,

http://www.sc.ehu.es/sbweb/webcentro/automatica/web_avr/archivos/Manual_AT90S8515/SPI/registros_spi_1.htm

Fecha de consulta abril 2012

[35] AVR STUDIO 4

http://www.gratisprogramas.org/descargar/avr-studio-4-19-avr-toolchain-4-19-hf/

Fecha de consulta abril 2012

[36] LPCXpresso 4: Herramientas de desarrollo de software,

http://www.code-red-tech.com/lpcxpresso.php

Fecha de consulta abril 2012

[37] Mini Tutorial LPCXpresso IDE, CodeRed,

http://laboratorios.fi.uba.ar/lse/seminario/material-2011/Sistemas_Embebidos-2011_2doC-Mini_Tutorial_CodeRed_LPCXpresso_IDE-Kharsansky.pdf

Fecha de consulta abril 2012

[38] Getting started with NXP LPCXpresso

http://ics.nxp.com/support/documents/microcontrollers/pdf/lpcxpresso.getting.started.pdf

Fecha de consulta abril 2012