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."

TESINA DE SEMINARIO

Previa la obtención del Título de:

INGENIERO EN ELECTRICIDAD ESPECIALIZACIÓN ELECTRÓNICA Y AUTOMATIZACIÓN INDUSTRIAL.

INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES.

Presentado por:

Walter Andrés Orellana Rivera

Gian Carlo Banchón Parra

GUAYAQUIL – ECUADOR

AÑO 2012

AGRADECIMIENTO

A Dios el creador, dueño y salvador de mi vida; y a mis padres Walter y Mónica que fueron la instrucción, dirección y la ayuda incondicional para llegar a esta etapa de mi vida, además a mis Maestros de la ESPOL.

Walter Orellana Rivera.

A Dios por haberme permitido llegar hasta este punto y haberme dado salud para lograr mis objetivos. A mis padres, hermanas y profesores por haberme apoyado en todo momento, por sus consejos, sus valores, sus enseñanzas, por la motivación constante que me ha permitido ser una persona de bien.

Gian Banchón Parra.

DEDICATORIA

A quienes llevo en mi corazón: mi amado Dios, a mi mamá Mónica a mi papá Walter, a mi hermana Ivette; a mis mejores amigos Betty, Jonathan, León, Adriano y mi utopía una niña muy especial en mi vida M. Virginia V.

Walter Orellana Rivera.

A mis padres: Yolanda y Juan por ser el pilar fundamental en todo lo que soy, en toda mi educación, tanto académica, como de la vida, por su incondicional apoyo perfectamente mantenido a través del tiempo.

Gian Banchón Parra.

TRIBUNAL DE SUSTENTACIÓN

_____

Ing. Carlos Valdivieso

Profesor del Seminario de Graduación

_____

MSIS. Ignacio Marín García

Profesor Delegado por la Unidad Académica

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)

_____

Walter Andrés Orellana Rivera

_____

Gian Carlo Banchón Parra

RESUMEN

El proyecto realizado consiste en el control y manejo de un motor BLDC a través de un joystick, utilizando comunicación SPI (Serial Peripheral Interface) entre las tarjetas: AVR Butterfly y LPCXpresso. Para la comunicación SPI se usó la tarjeta AVR Butterfly como MAESTRO, mientras que la tarjeta LPCXpresso trabajó como ESCLAVO. La programación de los microcontroladores se realizó en lenguaje C, utilizando como software el AVR STUDIO4 para la programación de la AVR Butterfly (con microcontrolador ATmega169), y el software LPCXpresso4 para la programación de la LPCXpresso (con microcontrolador LPC1769).

ÍNDICE GENERAL

GLOSARIO

Algoritmo: conjunto de instrucciones o reglas bien definidas, ordenadas y finitas que permite realizar una actividad mediante pasos sucesivos

Atmega169: Microcontrolador que pertenece a la familia AVR Atmel; de baja potencia y presenta arquitectura RISC.

AVR Studio4: Software que sirve para la programación de cualquier microcontrolador AVR Atmel en leguaje C/C++.

Bootloader: Pequeño programa que grabado previamente en un área especial de la flash, la zona o área de booteo, nos permitirá la actualización de la flash. Es decir que ya no necesitarás un programador para volver a actualizar tus aplicaciones.

Comunicación serial: consiste en el envío de un bit de información de manera secuencial, esto es, un bit a la vez y a un ritmo acordado entre el emisor y el receptor.

Estator: Parte fija de una máquina rotativa, la cual alberga una parte móvil (rotor)

Data Frame (trama de datos): es un paquete de datos de longitud fija o variable, que ha sido codificado por un protocolo de comunicaciones en la capa de enlace de datos, para la transmisión digital sobre un enlace nodo-a-nodo.

Full Dúplex: Cualidad de los elementos que permiten la entrada y salida de datos de forma simultánea.

I²C: La metodología de comunicación de datos del bus I2C es en serie y sincrónica. Facilita la comunicación entre microcontroladores, memorias y otros dispositivos con cierto nivel de "inteligencia", sólo requiere de dos líneas de señal y un común o masa.

Idle: En un procesador o una tarea, se describe como idle cuando no está siendo utilizado

LPC1769: Microcontrolador basado en ARM Cortex-M3, para aplicaciones que requieren un alto nivel de integración y baja disipación de potencia.

LPCXpresso4: Software altamente integrado, desarrollado para trabajar con microcontroladores LPC de NXP en lenguaje C.

Microelectrónica: Tecnología mediante la cual se diseña dispositivos electrónicos empacados en grandes densidades en una pastilla única de semiconductor.

MISO: Señal utilizada en una comunicación SPI para transferir datos en serie desde una tarjeta esclavo a una tarjeta maestro.

MOSI: Señal utilizada en una comunicación SPI para transferir datos en serie desde una tarjeta maestro a una tarjeta esclavo.

NXP: Nombre de una nueva compañía fundada por Philips en 2006. Esta empresa desarrolla productos para las industrias automovilísticas, de identificación, comunicación móvil, el hogar, semiconductores y software.

Rotor: Componente que gira (rota) en una máquina eléctrica, sea ésta un motor o un generador eléctrico.

Síncrono: Tiene un intervalo de tiempo constante entre cada evento.

SS: Señal selectora utilizada en una comunicación SPI que activa al esclavo cada vez que es seleccionada.

Transistor: Dispositivo compuesto de un material semiconductor que amplifica una señal o abre o cierra un circuito.

ABREVIATURAS

AC: Corriente Alterna.

ALU: Unidad lógica aritmética.

AVR: Virtual Avanzado RISC. (Advanced Virtual RISC).

BLDC: Motores DC sin escobillas (Brushless DC).

CI: circuito integrado.

CMOS: Semiconductor complementario del óxido de metal (Complementary Metal Oxide Semiconductor.

DC: Corriente Directa

ECM: Motores de conmutación electrónica.

ESC: Controladores electrónicos de velocidad.

FOC: Control de Campo Orientado.

IDE: Electrónica Integrada de dispositivos. (Integrated device Electronics).

I²C: Inter-Circuitos Integrados (Inter-Integrated Circuit).

JTAG: Prueba conjunta del grupo de acción (Joint Test Action Group).

SS: Selector esclavo. (Slave select).

MISO: Salida del esclavo, entrada al maestro. (Master In Slave Out).

MOSI: Salida del maestro, entrada al esclavo. (Master Out Slave In).

NXP: Próxima Experiencia. (Next eXPerience).

RS232: Estándar Recomendado 232 (Recommended Standard 232.)

SCC: Código Controlador de Segmento del LCD (Segment Control Code)

SCK: Reloj serial. (Serial clock).

SMBus: Bus del sistema de gestión. (System Management Bus).

SPI: Interfaz Serial de periféricos (Serial Peripherical Interface).

SSP: Puerto Serial Sincrónico (Synchronous Serial Port).

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

El objetivo del trabajo fue el diseño e implementación de un control mediante Joystick de un motor BLDC (DC Sin Escobillas), usando el protocolo SPI (Interfaz Serial Periférica) entre la AVR Butterfly y la LPCxpresso. Para mostrar el funcionamiento de este protocolo de comunicación usamos y aprovechamos las diferentes características que nos ofrecen las tarjetas mencionadas anteriormente.

El primer capítulo describe los antecedentes que tienen las partes implementadas en el proyecto, que son motores BLDC y microcontroladores. La motivación que tuvimos para realizarlo, en base al problema de usar dispositivo de diferentes familias en la comunicación SPI. Además, se plantean los objetivos alcanzados en el proyecto.

El segundo capítulo indica las ventajas, desventajas y funcionamiento del protocolo SPI. Describe los registros SPI que manejan los microcontroladores ATmega169 y LPC1769 de las tarjetas AVR Butterfly y LPCXpresso, respectivamente. Y presenta una comparación entre motores DC con escobillas y DC sin escobillas.

El tercer capítulo presenta los ejercicios previos realizados para la implementación de nuestro proyecto final. Estos consisten en una serie de combinaciones (maestro-esclavo) entre las tarjetas usadas en el proyecto. Además, para cada ejercicio hay un diagrama de bloques y un diagrama de flujo con su respectiva descripción.

El cuarto capítulo explica la implementación de nuestro proyecto final, donde se configura la tarjeta AVR Butterfly y LPCXpresso como maestro y esclavo respectivamente. Se indica la conexión de los pines, al Kit del motor (con tarjeta LPCXpresso LPC1114), para controlar el motor BLDC. También presenta un diagrama de bloques y un diagrama de flujo con su respectiva descripción.

CAPÍTULO 1

GENERALIDADES

Parte importante del proyecto son los microcontroladores y los motores BLDC, de los cuales haremos mención a fin de entender por qué son utilizados como parte de estudio de este trabajo que está enfocado a la implementación de una comunicación SPI. Indicaremos los beneficios que nos ofrece un motor DC sin escobillas, además de dar a conocer el uso de nuevas tecnologías en microcontroladores como lo son el LPC1769 y el ATmega169.

1.1 ANTECEDENTES

El primer circuito integrado fue desarrollado en 1959 por el ingeniero Jack Kilby (1923-2005) [1] pocos meses después de haber sido contratado por la firma Texas

Instruments. Se trataba de un dispositivo de germanio que integraba seis transistores en una misma base semiconductora para formar un oscilador de rotación de fase, eso fue posible gracias a desarrollos en la fabricación de dispositivos semiconductores a mediados del siglo XX.

Siendo las características principales: el pequeño tamaño, los tiempos de conmutación mínimos, el consumo de energía moderado, la confiabilidad, la capacidad de producción en masa y la versatilidad.

En la actualidad los circuitos integrados siguen evolucionando, se fabrican en tamaños cada vez más pequeños, tienen mejor eficiencia y eficacia. Permitiendo así que mayor cantidad de elementos sean empaquetados (integrados) en un mismo chip. Entre los circuitos integrados más complejos y avanzados se encuentran los microprocesadores.

Para realizar comunicación entre circuitos integrados nos valemos de interfaces de comunicación como por ejemplo RS232, I²C, SPI, siendo esta última de nuestro interés. La interfaz serial de periféricos SPI es un estándar de enlace de datos

seriales, sincronizados por un reloj que operan en modo full dúplex. Los microcontroladores se comunican en modo maestro/esclavo donde el dispositivo maestro inicia el data frame (trama de datos).

Los avances en los microcontroladores han sido de ayuda para el avance en el área eléctrica, que se centra en mejorar la eficiencia de los motores eléctricos, con el uso de nuevas configuraciones de control. Los motores eléctricos con mejor acogida para cumplir con esta finalidad son los motores BLDC.

1.2 MOTIVACIÓN PARA EL PROYECTO.

La comunicación serial SPI, para las tarjetas AVRbuttefly y LPCXpresso, se han implementado entre dispositivos de la misma familia. La motivación se formó en el hecho de aportar la misma comunicación SPI, pero entre las diferentes familias (AVRbuttefly con LPCXpresso), siendo esto algo nuevo, económico y su implementación en hardware muy simple.

1.3 IDENTIFICACIÓN DEL PROBLEMA

En la comunicación serial el proceso de envío de datos es de un bit por ciclo, secuencialmente. Contrasta con la comunicación paralela, donde todos los bits de

cada símbolo son enviados juntos. La comunicación serie es utilizada en casi todas las comunicaciones, porque los costos de los cables y las dificultades de sincronización hacen a la comunicación paralela poco práctica.

El problema en comunicar dos tarjetas de diferente familia en una comunicación serial SPI, radica en la desincronización que se produce en el reloj de cada tarjeta. Una posible solución a esta problemática es la modificación de la frecuencia del reloj en cualquiera de los dos microcontroladores (LPC1769 y ATmega169).

1.4 OBJETIVOS DEL PROYECTO

Los objetivos generales y específicos del presente proyecto, se detallan a continuación.

1.4.1 Objetivo General

El objetivo principal de este proyecto es realizar el diseño de un sistema maestro – esclavo mediante una comunicación SPI entre las tarjetas AVR Butterfly y la tarjeta LPCXpresso para controlar un Motor BLDC mediante un Joystick.

1.4.2 Objetivos Específicos

- Usar herramientas para el diseño de interfaces, basados en microcontroladores avanzados

- Investigar los microcontroladores usados (LPC1769 y ATMega169) para el adecuado diseño de nuestro prototipo electrónico.

- Aplicar los conceptos sobre el protocolo de comunicación SPI al momento de diseñar el sistema Maestro – Esclavo

- Que la implementación del hardware sea lo más simple posible, de tal manera que se puedan realizar las combinaciones Maestro - esclavo entre las tarjetas AVR Butterfly y LPCXpresso.

CAPÍTULO 2

MARCO TEÓRICO

Este capítulo describe las características de los componentes que son implementados en este proyecto que son los motores BLDC, el microcontrolador LPC1769 (perteneciente a la LPCXpresso) y el microcontrolador ATMEGA169 (perteneciente a la AVR Butterfly). Se explicará de manera general el funcionamiento del protocolo SPI y de manera particular los registros, del SPI, de los microcontroladores de cada tarjeta. Además indicaremos las herramientas de software utilizadas; las cuales son LPCXpresso 4, y AVR Studio 4.

2.1 INTERFAZ SERIAL PERIFÉRICA: SPI

SPI (del inglés Serial Peripheral Interface) es un bus de comunicaciones, usado principalmente para la trasferencia de información entre circuitos integrados en equipos electrónicos.

SPI es capaz de controlar casi cualquier electrónica digital que acepte un flujo de bits serie regulados por un reloj. Esta interface se encuentra formada de 4 hilos (SCK, SS, MISO, MOSI), descritos en la Tabla 2.1, los cuales son la línea de reloj, dato entrante, dato saliente y un selector de chip.

Nombre	Tipo	Descripción
SCK	I/O	SERIAL CLOCK: La señal de reloj SPI (SCK) se utiliza para sincronizar la transferencia de datos a través de la interfaz SPI. El SPI es siempre guiado por el maestro y recibido por el esclavo.
SS	I	SELECTOR DE ESCLAVO: es una señal baja activa que indica que esclavo se encuentra seleccionado para participar en la transferencia de datos. Cada esclavo tiene su propia y única señal de entrada SS. Podría ser impulsada por un simple propósito general I/O bajo el control de software.
MISO	I/O	Maestro Entrada Esclavo Salida: es una señal unidireccional utilizada para transferir datos en serie desde un SPI esclavo a un SPI maestro.
MOSI	I/O	Maestro Salida Esclavo Entrada: es una señal unidireccional utilizada para transferir datos en serie de un SPI maestro a un SPI esclavo.

Tabla 2.1 Descripción de bits

Un maestro de sincronización es aquel que inicia la transferencia de información sobre el bus y genera las señales de reloj y control. Un esclavo es un dispositivo controlado por el maestro. Cada esclavo es controlado sobre el bus a través de una línea selectora por lo tanto el esclavo es activado solo cuando esta línea es seleccionada. Generalmente una línea de selección es dedicada para cada esclavo. Como se ilustra en la Figura 2.1.

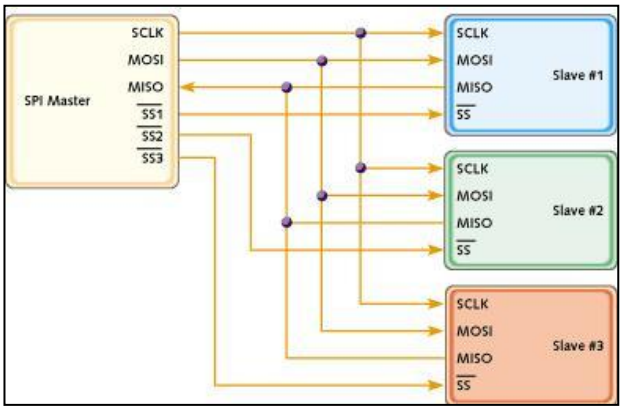

Figura 2.1 Comunicación SPI entre un Maestro y varios Esclavos. [2]

2.1.1 Ventajas del SPI

El protocolo SPI permite una comunicación Full Dúplex a una mayor velocidad de transmisión en comparación con I²C o SMBus. Es un protocolo flexible en que se puede tener un control absoluto sobre los bits transmitidos, por lo cual no está limitado a la transferencia de bloques de 8 bits y se puede elegir del tamaño de la trama de bits.

Su implementación en hardware es extremadamente simple ya que consume menos energía que I²C o que SMBus debido que posee menos circuitos. Los dispositivos clientes usan el reloj que envía el servidor, no necesitan por tanto su propio reloj, además necesitan como mucho una señal específica para cada cliente (señal SS), las demás señales pueden ser compartidas. 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.

2.1.2 Desventajas del SPI

El protocolo SPI presenta desventajas tales usar más pines en cada chip a diferencia de I²C que usa dos. El direccionamiento se hace mediante líneas específicas (señalización fuera de banda) a diferencia de lo que ocurre en I²C que selecciona cada chip mediante una dirección de 7 bits que se envía por las mismas líneas del bus.

SPI sólo funciona en las distancias cortas a diferencia de, por ejemplo, RS-232, RS-485, o Bus CAN. No posee señal de asentamiento, es decir, que el servidor podría estar enviando información sin que estuviese conectado algún cliente y no se daría cuenta de nada; tampoco permite fácilmente, tener varios servidores conectados al bus.

2.2 SPI DE LA LPCXPRESSO, LPC1769

Se presentará el protocolo SPI concerniente al microcontrolador LPC1769 de la tarjeta LPCXpresso, con la finalidad de conocer su funcionamiento.

2.2.1 Transferencia de Datos

La figura 2.2 es un diagrama de temporización que ilustra los cuatro formatos diferentes de transferencia de datos que están disponibles con el puerto SPI. Este diagrama de tiempos ilustra una transferencia de datos de 8 bits. Lo primero que observa en este diagrama de tiempos, es que se divide en tres partes horizontales.

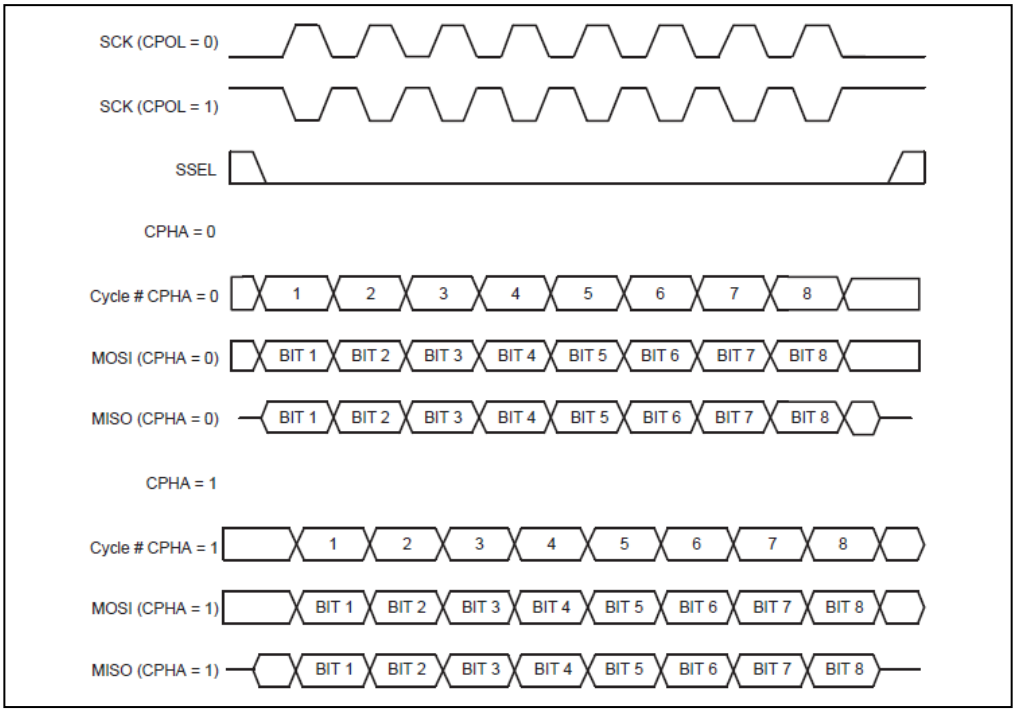

Figura 2.2 Formato de transferencia de datos SPI (CPHA = 0 and CPHA = 1) [3]

La parte superior describe la señal SCK (0 y 1) y la señal SSEL. En parte intermedia describe las señales MOSI y MISO cuando el bit de Control de Fase de Reloj (CPHA) del Registro de control SPI es 0. La parte inferior describe las señales MOSI y MISO cuando el bit de Control de Fase de Reloj (CPHA) del Registro de control SPI es 1.

En la parte superior del diagrama de temporización, note dos puntos. Primero, el SPI se ilustra con el bit de Control de Polaridad del Reloj (CPOL) en el Registro de Control de SPI ajustado con 0 y 1. El segundo punto nota la activación y desactivación de la señal SSEL. Cuando CPHA = 0, la señal SSEL siempre irá inactiva entre las transferencias de datos. Esto no está garantizado cuando CPHA = 1 donde la señal puede permanecer activa. Los datos y las relaciones de fase de reloj se muestran en resumen en la tabla 2.2.

CPOL y CPHA	**Cuando el primer bit del dato es conducido**	**Cuando todos los bits del dato son conducidos**	**Cuando el dato es muestreado**
CPOL=0 CPHA=0	Antes del primer flanco de subida SCK	Flanco de bajada SCK	Flanco de subida SCK
CPOL=0 CPHA=1	En el primer flanco de subida SCK	Flanco de subida SCK	Flanco de bajada SCK
CPOL=1 CPHA=0	Antes del primer flanco de bajada SCK	Flanco de subida SCK	Flanco de bajada SCK
CPOL=1 CPHA=1	En el primer flanco de bajada SCK	Flanco de bajada SCK	Flanco de subida SCK

Tabla 2.2 Datos del SPI relacionados para la fase de reloj. [3]

Definir cuando una transferencia se inicia y se detiene, depende de si un dispositivo es un maestro o esclavo, y del valor de la variable CPHA. Cuando un dispositivo es un maestro, el comienzo de una transferencia está indicado por el maestro que tiene un byte de datos que está listo para ser transmitido. En este punto, el maestro puede activar el reloj, y comenzar la transferencia. La transferencia termina cuando el último ciclo de reloj de la transferencia es completado.

Cuando un dispositivo es un esclavo y CPHA = 0, la transferencia se inicia cuando la señal SSEL se activa, y termina cuando SSEL se desactiva. Cuando un dispositivo es un esclavo, y CPHA = 1, la transferencia se inicia en el primer flanco del reloj cuando el esclavo es seleccionado, y termina en el último flanco de reloj donde el dato es muestreado.

2.2.2 Registros del SPI

El microcontrolador LPC1769 tiene cinco registros para el puerto SPI los cuales son S0SPCR, S0SPSR, S0SPDR, S0SPCCR, S0SPINT y se describen en síntesis en la tabla 2.3.

Nombre	Descripción	Acceso
S0SPCR	Registro de control SPI.- Este registro controla la operación del SPI.	R/W
S0SPSR	Registro de Estado SPI.- Este registro muestra el estado del SPI	R
S0SPDR	Registro de datos SPI.- Este registro bidireccional proporciona los datos de transmisión y recepción para el SPI. Los datos de transmisión se proporcionan al SPI0 por escrito a este registro. Los datos recibidos por el SPI0 se puede leer en este registro.	R/W
S0SPCCR	Registro de contador de reloj.- Este registro controla la frecuencia del maestro SCK0	R/W
S0SPINT	Banderas de interrupción.- Este registro contiene las banderas de interrupción para la interface SPI.	R/W

Tabla 2.3 Descripción de registros del SPI [3]

Registro de Control SPI.-

El registroS0SPCRcontrola el funcionamiento del SPI0. La configuración de los bits se muestra en la tabla 2.4. SPI0 (Serial Peripheral Interface 0) es la única Interfaz Serial que maneja la LPC1769.

Bit	Símbolo	Valor	Descripción	Valor de reinicio
1:0	-		Reservado, el software de usuario no escribe en los bits reservados: el valor se lee de un bit reservado no definido.	NA
2	BitEnable	0 1	El controlador de SPI envía y recibe 8 bits de datos por transferencia. El controlador de SPI envía y recibe el número de bits seleccionado por bits del 11:8	0
3	CPHA	0 1	El control de fase del reloj determina la relación entre los datos y el reloj en la transferencia SPI, y los controles cuando la transferencia de un esclavo se define como inicial y final. Los datos son muestreados en el primer flanco reloj SCK. Una transferencia se inicia y finaliza con la activación y desactivación de la señal SSEL. Los datos son muestreados en el segundo flanco de reloj SCK. Una transferencia se inicia en el primer flanco de reloj y termina con el último flanco de muestreo cuando la señal de SSEL se desactiva.	0
4	CPOL	0 1	Polaridad del controlador de reloj SCK es activa en alto. SCK es activa en bajo.	0
5	MSTR	0 1	Selector del modo MESTRO El SPI opera en el modo esclavo El SPI opera en el modo Maestro	0

Tabla 2.4 Registro de control SPI (S0SPCR) descripción de bit. [3]

Bit	Símbolo	Valor	Descripción	Valor de reinicio
6	LSBF		Primer LSB controla cual dirección de cada byte es desplazado cuando es transferido. El primer dato SPI que es transferido es el MSB (bit 7) El primer dato SPI que es transferido es el LSB (bit 0)	0
7	SPIE	0 1	Habilitador de interrupción SERIAL Periférica. Interrupción SPI son inhibidas. Una interrupción de hardware es generada cada vez que el bit SPIF o MODF son activados.	0
11:8	BITS	1000 1001 1010 1011 1100 1101 1110 1111 0000	Cuando el bit 2 de este registro es 1, este campo controla el número de bits por transferencia: 8 bits por transferencia. 9 bits por transferencia. 10 bits por transferencia. 11 bits por transferencia. 12 bits por transferencia. 13 bits por transferencia. 14 bits por transferencia. 15 bits por transferencia. 16 bits por transferencia.	0000
31:12	-		Reservado, el software de usuario no escribe en los bits reservados. El valor se lee desde un bit reservado no definido.	NA

Tabla 2.4 Registro de control SPI (S0SPCR) descripción de bit. [3]

Registro de Estado SPI.-

El registroS0SPSRmuestra el estado en el funcionamiento deSPI0. La configuración de los bits se muestra en la tabla 2.5.

Bit	Símbolo	Descripción	Valor de Reinicio
2:0	-	Reservado, el software de usuario no escribe en los bits reservados: el valor se lee de un bit reservado no definido.	NA
3	ABRT	Aborto del esclavo. Cuando este bit es 1 indica que un aborto de esclavo ha ocurrido. Este bit es limpiado por lectura de registro.	0
4	MODF	Modo fault. Cuando este bit es 1 indica que un Modo fault ha ocurrido. Este bit es limpiado por lectura del registro, la escritura depende del registro de control SPI0.	0
5	ROVR	Read over run. Cuando este bit es 1 indica que un read over run ha ocurrido. Este bit es limpiado por lectura de registro.	0
6	WCOL	Write Collision. Cuando este bit es 1 indica que un Write Collison ha ocurrido. Este bit es limpiado por lectura del registro, el acceso depende del registro de Datos SPI.	0
7	SPIF	Bandera de transferencia completa. Cuando el bit es 1 indica que la transferencia de un dato es completa. Cuando un maestro, este bit se pone al final de la última ciclo de transferencia. Cuando un esclavo, este bit se establece en el último flanco de datos de muestreo del SCK. Este bit es limpiado por la primera lectura de este registro, el acceso se da en el registro de Datos SPI. Nota: Esto no es una bandera de Interrupción. Esta bandera es encontrada en el registro SPINT.	0
31:8	-	Reservado, el software de usuario no escribe en los bits reservados: el valor se lee de un bit reservado no definido.	NA

Tabla 2.5 Registro de estado SPI (S0SPSR) descripción de bits. [6]

Registro de Datos SPI.-

Este registro bidireccional de datos proporciona los datos de transmisión y recepción para el SPI. La transmisión de datos se proporciona a la SPI a través de este registro. Los datos recibidos por el SPI pueden ser leídos desde este registro.

Cuando se utiliza como un maestro, una escritura en este registro se inicia con una transferencia de datos SPI. Los escritos en este registro son bloqueados cuando se inicia una transferencia de datos, o cuando el bit de estado SPIF es seteado, y el Registro de Estado SPI no ha sido leído.La configuración de los bits se muestra a continuación en la tabla 2.6

Bit	Símbolo	Descripción	Valor de Reinicio
7:0	Datos bajos	Puerto de dato bidireccional del SPI	0x00
15:8	Datos altos	Si el bit 2 del SPCR es 1 y en los bits del 11:8 existen otros más que el 1000, algunos o todos de estos bits contienen la transmisión y recepción adicional de bits. Cuando menos de 16 bits son seleccionados, el más significativo entre estos bits leídos son ceros.	0x00
31:16	-	Reservado, el software de usuario no escribe en los bits reservados: el valor se lee de un bit reservado no definido.	NA

Tabla 2.6 Registro de datos SPI (S0SPDR). [3]

Registro Contador de Reloj SPI.-

Registro Contador de Reloj SPI controla la frecuencia SCK de un maestro. El registro indica el número de ciclos de reloj SPI periféricos que componen un reloj SPI. En el modo maestro, este registro debe ser un número mayor o igual a 8. Violaciones de esto puede resultar en un comportamiento impredecible. En modo esclavo, la velocidad de reloj SPI facilitada por el maestro no debe exceder 1/8 del

reloj periférico SPI seleccionado en PCLK_SPI. El contenido del registro S0SPCCR no es relevante. La configuración de los bits se muestra en la Tabla 2.7.

Bit	Símbolo	Descripción	Valor de Reinicio
7:0	Contador	Estableciendo el contador de reloj SPI0	0x00
31:8	-	Reservado, el software de usuario no escribe en los bits reservados: el valor se lee de un bit reservado no definido.	NA

Tabla 2.7 Registro contador de reloj SPI (S0SPCCR) descripción de bit. [3]

Registro de Interrupción SPI.-

El registro de Interrupción SPI contiene las banderas de interrupción de la interfaz SPI0. La Bandera de interrupción se encarga de setear la interfaz SPI para generar una interrupción.La configuración de los bits se muestra en la tabla 2.8

Bit	Símbolo	Descripción	Valor de Reinicio
0	SPIF	Bandera de interrupción SPI. Setea la interfaz SPI para generar una interrupción. Limpiada por escrito de un 1 en este bit. Nota: este bit será seteado cada vez y cuando el SPIE=1y al menos el SPIF y el WCOL sean 1.Sin embargo, sólo cuando el bit de interrupción SPI es seteado y la interrupción SPI0 está habilitada en la NVIC, la interrupción basada en SPI puede ser procesada por manejo de software de interrupción.	0
7:1	-	Reservado, el software de usuario no escribe en los bits reservados. El valor se lee desde un bit reservado no definido.	NA
31:8	-	Reservado, el software de usuario no escribe en los bits reservados. El valor se lee desde un bit reservado no definido.	NA

Tabla 2.8 Registro de interrupción SPI (S0SPINT) descripción de bit. [3]

2.3 SPI DE LA AVR BUTTERFLY, ATmega169

Se presentará el protocolo SPI concerniente al microcontrolador ATmega169 de la tarjeta AVR Butterfly, con la finalidad de conocer su funcionamiento.

2.3.1 Modos del Reloj

Existen cuatro modos de reloj definidos por el protocolo SPI. Estos determinan el valor de la polaridad del reloj (CPOL = Clock Polarity) y el bit de fase del reloj (CPHA = Clock Phase). Estos modos son mostrados en la tabla A.8 del ANEXO A

La mayoría de los dispositivos SPI pueden soportar al menos 2 modos de los 4 antes mencionados. El bit de CPOL determina el nivel del estado de Idle del reloj y el bit CPHA determina qué flanco recibe un nuevo dato. El modo requerido, está dado por el dispositivo esclavo y son ilustrados en la Figura 2.3.

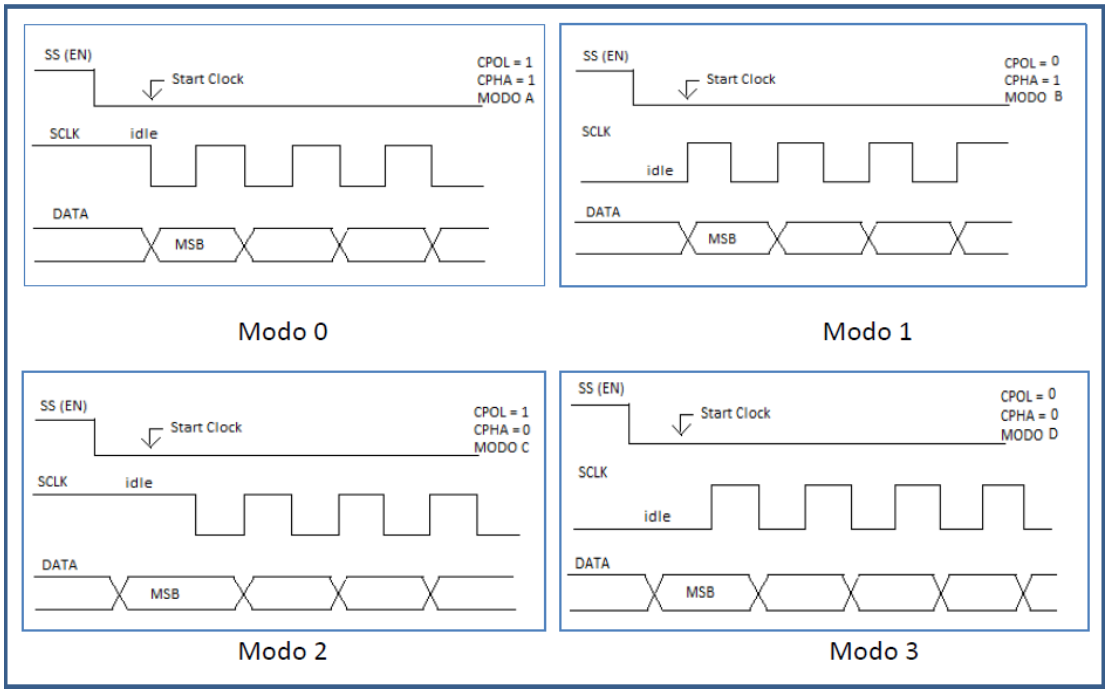

Figura 2.3 Modos de reloj del SPI. [4]

2.3.2 Registros del SPI

Los microcontroladores AVR contienen el maestro y el esclavo en un solo chip, por lo tanto, pueden trabajar como maestro y esclavo. Generalmente un microcontrolador AVR toma el papel de maestro y cualquier dispositivo conectado a él se comporta como esclavo (aunque los papeles pueden intercambiarse). Al igual que nuestra tarjeta LPCxpreso para comunicar un dispositivo mediante SPI con un tarjeta AVR Butterfly se utilizan cuatro pines: MISO, MOSI, SCK Y SS.

Registro de Control del SPI – SPCR.-

La figura 2.4 muestra la disposición de los bits para la configuración del registro SPCR, que se encarga del control de operación del SPI.

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	

Figura 2.4 Registro de Control del SPI. [4]

SPR1:0, Velocidad de la Señal de Reloj del SPI: estos bits junto con el bit SPI2X del registro SPSR deciden la frecuencia de la señal de reloj SCLK. La combinación de estos tres bits para seleccionar la frecuencia de reloj se muestra en la tabla A.9 del ANEXO A.

CPHA, Reloj de Fase: este bit permite adelantar o retrasar la señal de reloj SCLK con respecto a los datos provenientes del esclavo.

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. Si dos dispositivos SPI desean comunicarse entre sí, estos deben tener la misma Polaridad de Reloj (CPOL) y la misma Fase de Reloj (CPHA).

CPOL, Polaridad del reloj: el bit CPOL en 1 hace que SCLK se mantenga en alto cuando no se esté transmitiendo, CPOL en 0 hace que SCLK se mantenga en bajo cuando no hay transmisiones, entonces resumimos la funcionalidad del CPOL así:

- CPOL = 1 entonces SCK es Alto, en inactividad.
- CPOL = 0 entonces SCK es Bajo, en inactividad.

MSTR, Selección del Maestro/Esclavo: selecciona el modo Maestro SPI cuando se escribe uno en este bit, y el modo esclavo SPI cuando está escrito con cero lógico. Si SS es configurado como una entrada y es controlada en bajo mientras MSTR es uno, MSTR será limpiada, y SPIF en SPSR llegará a ser uno. El uso tendrá uno MSTR al re-habilitar el modo Maestro SPI.

DORD, Orden del Dato: cuando este bit es igual a uno el bit menos significativo LSB del dato se transmitirá primero, caso contrario, el primer bit en transmitirse será el más significativo MSB.

SPE, Habilitador del SPI: el sistema SPI es habilitado cuando este bit es puesto a 1, por lo tanto para realizar o habilitar cualquier operación del SPI este bit debe ser necesariamente 1.

SPIE, Habilitador de la Interrupción por SPI: Este bit causa la interrupción del SPI al ser ejecutado si el bit SPIF en el registro SPSR es uno y si las Interrupciones Globales son habilitadas con uno en el bit del SREG.

Registro de Estado del SPI – SPSR.-

Tal y como se puede ver en la figura 2.5 muestra la disposición de los bits para la configuración del registro SPSR, que se encarga de mostrar el estado del SPI.

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	

Figura 2.5 Registro de Estado del SPI. [4]

SPI2X, Bit para Doble Velocidad en SPI: Cuando este bit es escrito con uno lógico la velocidad del SPI (Frecuencia SCK) será duplicada cuando el SPI esté en Modo Maestro.

WCOL, Escritura de la Bandera de Interrupción: El bit WCOL es uno si el registro de Datos del SPI (SPDR) es escrito durante la transferencia.

SPIF, Bandera de Interrupción SPI: este bit se pone en uno automáticamente cuando una transferencia serial se completa. Si SS es una entrada y es controlada en bajo cuando está en Modo maestro SPI, esto también pone en uno la bandera SPIF.

Registro de Datos del SPI - SPDR.-

Tal y como se puede ver en la figura 2.6 en la que se muestra el Registro de Datos SPI, el cual es usado para la transferencia de datos entre el Registro Archivo y el de Cambio del SPI. Escribiendo en el registro inicializa la transmisión de datos. Leyendo el registro causa cambios al registro al recibir la lectura

MSB: Bit más significativo (Most significant bit).

LSB: Bit menos significativo (Less significant bit).

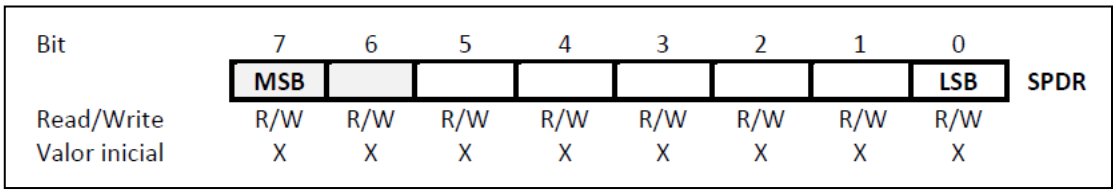

Figura 2.6 Registro de Datos del SPI. [4]

2.4 INTRODUCCIÓN A MOTORES BLDC

Los motores BLDC también conocidos como motores de conmutación electrónica (ECM) son utilizados frecuentemente en industrias, equipo de automatización industrial e instrumentación. Puede ser descrito como motores paso a paso , con imanes permanentes fijos y posiblemente más polos en el rotor.

Los motores BLDC no usan escobillas para la conmutación, sino que usan una conmutación electrónica. Este atributo eliminará la problemática que poseen los motores eléctricos convencionales con escobillas, los cuales disminuyen el rendimiento al producir rozamiento, ser ruidosos, y requerir un mayor mantenimiento.

Los motores BLDC tienen muchas ventajas, como: alta eficiencia, mejor velocidad, la respuesta dinámica alta, larga vida útil, funcionamiento silencioso y mayores rangos de velocidad, sobre los motores de corriente continua con escobilla y motores de inducción. Por otro lado, los motores BLDC tienen un par de desventajas, que son las siguientes: requieren un control bastante más complejo, tienen un mayor coste.

2.4.1 DC sin escobillas vs DC con escobilla

Las principales diferencias entre un motor DC sin escobilla y el que si tiene escobillas están dadas en 11 aspectos, los cuales mencionaremos a continuación:
Un motor DC con escobillas es un equipo de durabilidad y coste de construcción bajos, tiene una eficiencia moderada y su mantenimiento debe ser periódico. Este tipo de motor no requiere de control, aunque es simple y barato, si no se requiere una variación de velocidad. Su curva Par/Velocidad a altas velocidades, hace incrementar la fricción de las escobillas produciendo una reducción en el par y la inercia de su rotor es alta la cual limita su capacidad dinámica. Debido a que su conmutación es por escobillas, la potencia de salida es baja ya que se produce calor en la armadura y se disipa en el interior aumentando la temperatura.

Un motor DC sin escobillas (BLDC) es un equipo de durabilidad y coste de construcción altos porque usa imanes permanentes, tiene una eficiencia alta y su mantenimiento es mínimo. Este tipo de motor requiere de un control, aunque complejo y caro, para mantenerse funcionando y variar su velocidad. En cuanto a su curva Par/Velocidad este motor opera a todas las velocidades con la carga definida y la inercia de su rotor es baja por consecuencia de los imanes permanentes. Debido a que su conmutación es electrónica basada en sensores de posición de efecto Hall, la potencia de salida es alta ya que al tener el bobinado en el estator mejora su característica térmica.

CAPÍTULO 3

EJERCICIOS PREVIOS AL PROYECTO

El capítulo describe los tres ejercicios realizados, previos al proyecto. Cuyo fin consistió en la familiarización de la comunicación SPI, utilizando las tarjetas LPCXpresso y AVR Butterfly. Cada ejercicio contiene una explicación, la cual está formada por un diagrama de bloques, un diagrama de flujo con su respectiva descripción y la ilustración de la implementación. Los diagramas de conexiones de cada ejercicio se encuentran en el ANEXO E. Los ejemplos darán a conocer 3 combinaciones diferentes entre las tarjetas LPCxpresso y AVR Butterfly usando una comunicación SPI. Las cuales son las siguientes:

- LPCxpresso como maestro y LPCxpresso como esclavo.
- LPCxpresso como maestro y AVR Butterfly como esclavo.
- AVR Butterfly como maestro y LPCxpresso como esclavo.

Para la comunicación SPI de la LPC nos basamos en el ejemplo SPP que nos ofrece el software LPCxpresso 4 en el archivo **NXP_LPCXpresso1769_MCB1700_2011-02-11.zip.** [9].

3.1 EJERCICIO 1: CONTADOR DE 0 A 9 ENTRE LPC - LPC

La tarjeta LPCxpresso (con microcontrolador LPC1769), configurada como maestro, envía mediante comunicación SPI, una serie de datos a otra tarjeta LPCxpresso que está como esclavo; la cual enviará por su puerto 2 la respectiva combinación binaria para mostrar los números en el display de 7 segmentos, como se muestra en la Figura 3.1. Además también se mostrarán en un arreglo de LEDs.

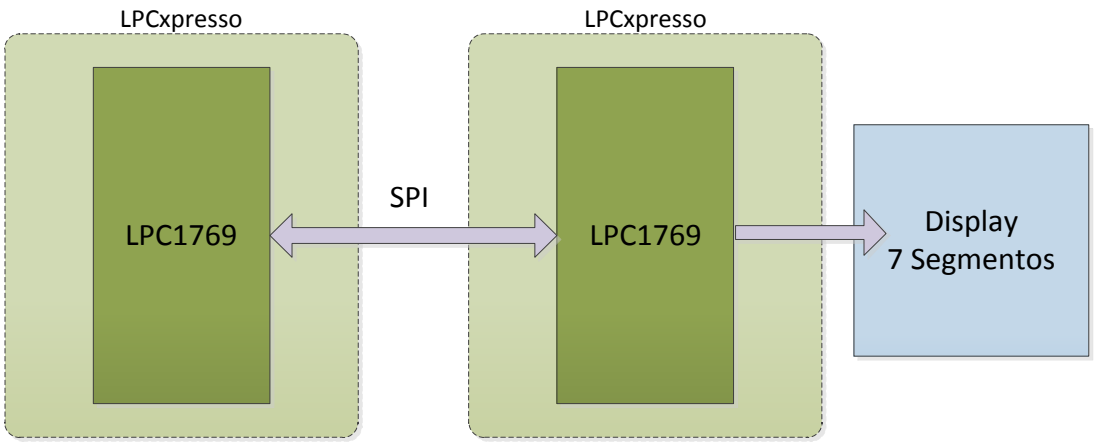

Figura 3.1 Diagrama de Bloques del Ejercicio 1

3.1.1 Descripción del Algoritmo

A continuación se explicará el algoritmo del programa principal para el controlador maestro (LPCXpresso) y el algoritmo del programa principal para el controlador esclavo (LPCXpresso).

Controlador Maestro

Para el control del dispositivo se inicializan los parámetros PORTNUM, LED, TX_RX_ONLY, SSP_SLAVE. y segmentLUT. Posteriormente preguntamos si la variable PORTNUM==1 para inicializar una de las interfaces SSP capaz de funcionar en SPI. Una vez seleccionada la interfaz se escoge el número del arreglo segmentLUT; preguntamos si TX_RX_ONLY==1 para entrar en modo de Transmisión-Recepción.

Finalizamos preguntando si la variable SSP_SLAVE==1 para indicar si trabaja como maestro y luego trasmitir el dato. Las instrucciones de este algoritmo son ilustradas en la Figura 3.2 y el código fuente se encuentra en el ANEXO D.

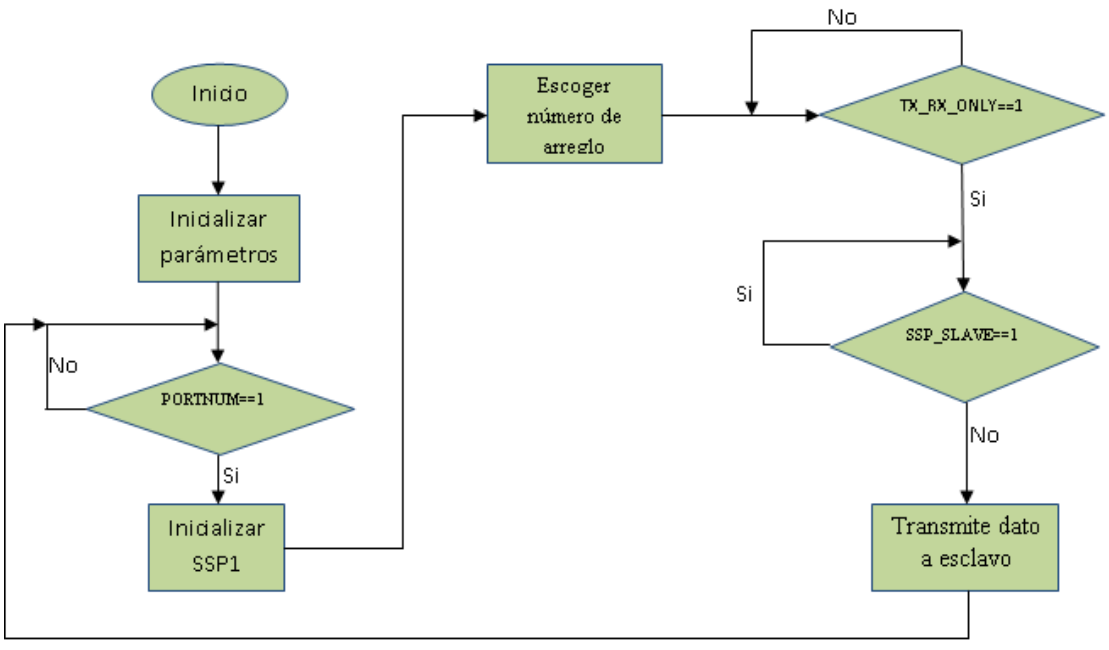

Figura 3.2 Diagrama de Flujo Controlador Maestro Ejercicio 1

Controlador Esclavo

Para el control del dispositivo se inicializan los parámetros PORTNUM, LED, TX_RX_ONLY, SSP_SLAVE. y segmentLUT. Seguido preguntamos si la variable PORTNUM==1 para inicializar una de las interfaces SSP capaz de funcionar en SPI. Una vez seleccionada la interfaz; preguntamos si TX_RX_ONLY==1 para entrar en modo de Transmisión-Recepción. Luego preguntamos si la variable SSP_SLAVE==1 para indicar si trabaja como esclavo. Para finalizar, el dato es recibido del maestro y lo muestra en el puerto 2 llamado LED. Las instrucciones de este algoritmo son ilustradas en la Figura 3.3 y el código fuente se encuentra en el ANEXO D.

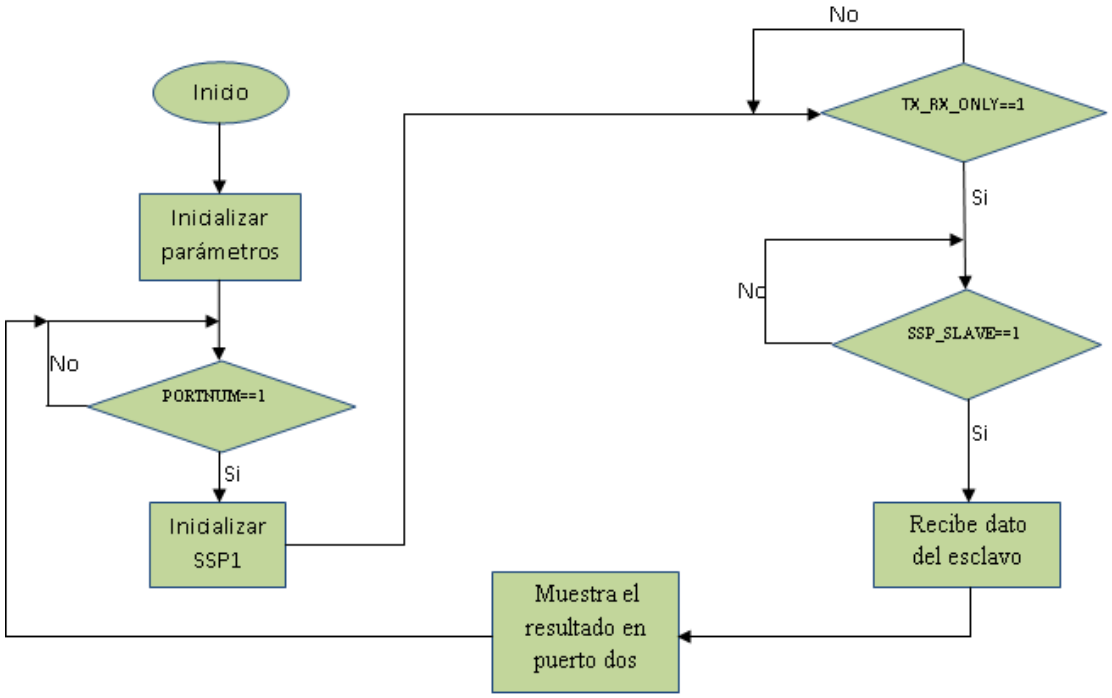

Figura 3.3 Diagrama de Flujo Controlador Esclavo Ejercicio 1

3.1.2 Implementación

Nuestro ejercicio consiste en un CONTADOR AUTOMÁTICO donde la comunicación entre la LPC Master y la LPC Esclavo se da a través de 4 hilos: SEL, SCLK, MISO, MOSI. Los datos viajan desde la LPC master hacia la LPC esclavo, en esta última se configura el puerto dos como salida, donde conectamos un arreglo de LEDs y un display de siete segmentos que servirán para mostrar los dígitos de 0 a 9, como se ilustra en la Figura 3.4. Con la implementación del ejercicio se demuestra que la comunicación SPI entre “LPCs” es un buen ensayo para la ejecución de nuestro proyecto final. Se recomienda en ambas tarjetas conectar en un punto común su pin de tierra.

Materiales:

Dos tarjetas LPCXpresso (LPC 1769), cuatro resistencias de 1K Ω, dieciséis resistencias de 330 Ω, un display de siete segmentos (cátodo común), ocho LEDs, dos Baterías AA – 1,5 V, un porta baterías AA.

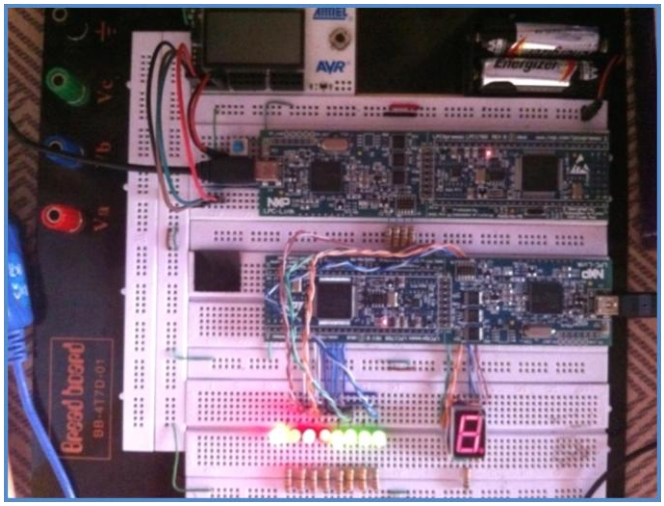

Figura 3.4 Implementación del contador de 0 a 9 entre LPC – LPC

3.2 EJERCICIO 2: CONTADOR DE 0 A 9 ENTRE LPC – AVR

El maestro, tarjeta LPCxpresso (con microcontrolador LPC1769), envía mediante comunicación SPI, una serie de datos a otra tarjeta AVR Butterfly (con microcontrolador Atmega169) que está como esclavo; la cual enviará por su puerto D la respectiva combinación binaria para mostrar los números en el display de 7 segmentos, como se muestra en la Figura 3.5. Además también se mostrarán en un arreglo de LEDs.

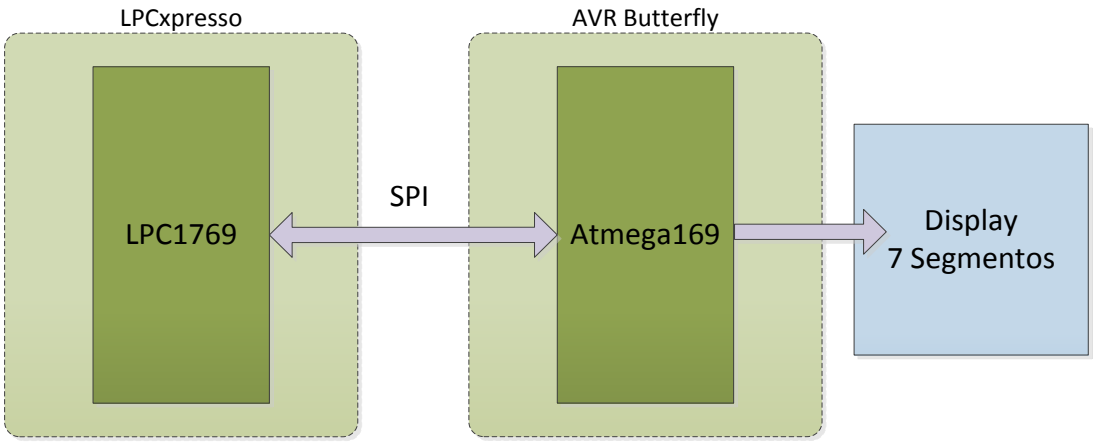

Figura 3.5 Diagrama de Bloques del Ejercicio 2

3.2.1 Descripción del Algoritmo

A continuación se explicará el algoritmo del programa principal para el controlador maestro (LPCXpresso) y el algoritmo del programa principal para el controlador esclavo (AVR Butterfly).

Controlador Maestro

Para el control del dispositivo se inicializan los parámetros PORTNUM, LED, TX_RX_ONLY, SSP_SLAVE. y segmentLUT. Seguido preguntamos si la variable PORTNUM==1 para inicializar una de las interfaces SSP capaz de funcionar en SPI. Una vez seleccionada la interfaz se escoge el número del arreglo segmentLUT; preguntamos si TX_RX_ONLY==1 para entrar en modo de Transmisión-Recepción. Finalizamos preguntando si la variable SSP_SLAVE==1 para indicar si trabaja como maestro y luego trasmitir el dato. Las instrucciones de este algoritmo son ilustradas en la Figura 3.6 y el código fuente se encuentra en el ANEXO D.

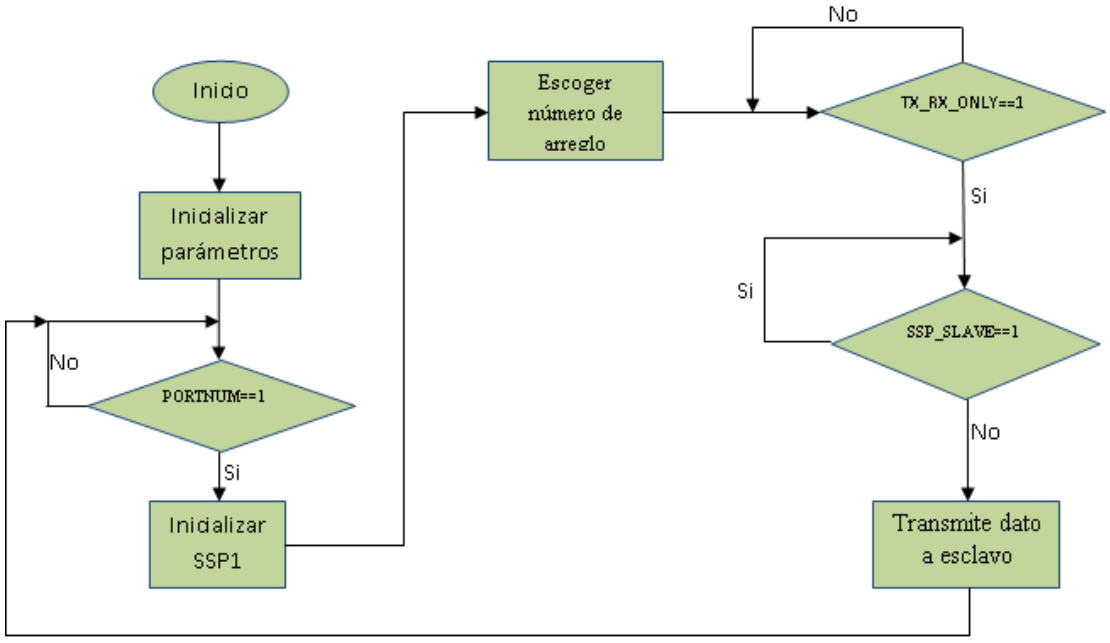

Figura 3.6 Diagrama de Flujo Controlador Maestro Ejercicio 2

Controlador Esclavo

Para el control del dispositivo se inicializa la comunicación SPI configurándola como esclavo, mediante la función SPI_slave_init(void), seguido configuramos el puerto D de la tarjeta como salida y se lo encera.

Finalmente se ingresa a un lazo while infinito donde se van a recibir los datos trasmitidos a través de la función SPI_slave_receive() para luego mostrarlos en el Puerto D. Las instrucciones de este algoritmo son ilustradas en la Figura 3.7 y el código fuente se encuentra en el ANEXO D.

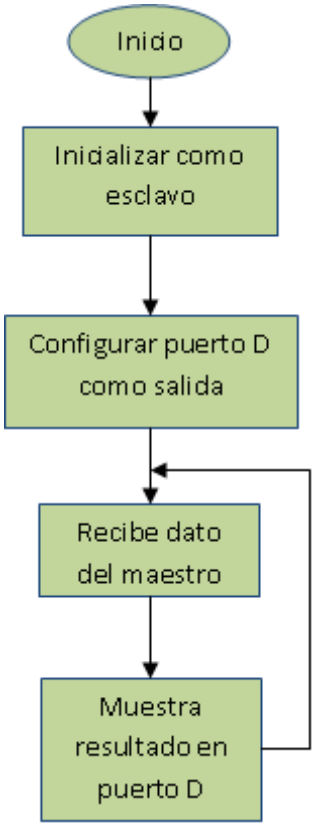

Figura 3.7 Diagrama de Flujo Controlador Esclavo Ejercicio 2

3.2.2 Implementación

Nuestro ejercicio consiste en un CONTADOR AUTOMÁTICO donde la comunicación SPI entre la LPC Master y el AVR Esclavo se da a través de 4 hilos: SEL, SCLK, MISO, MOSI. Los datos viajan desde la LPC master hacia el AVR esclavo, en este último está configurado el puerto D como salida, donde conectamos un display de siete segmentos que servirá para mostrar los dígitos de 0 a 9, como se ilustra en la Figura 3.8. Se recomienda en la función Void SSP1Init(Void), de nuestro ejemplo tomado del sotfware LPCXpresso 4 (SSP), realizar un cambio en el Registro Pre-escalador del reloj: CPSR = 0x2 por CPSR = 0xf.

Materiales:

Una tarjeta AVR Butterfly (ATmega169), tarjeta LPCXpresso (LPC 1769), cuatro resistencias de 1K Ω, dieciséis resistencias de 330 Ω, un display de siete segmentos (cátodo común), ocho LEDs, dos Baterías AA – 1,5 V, un porta baterías AA.

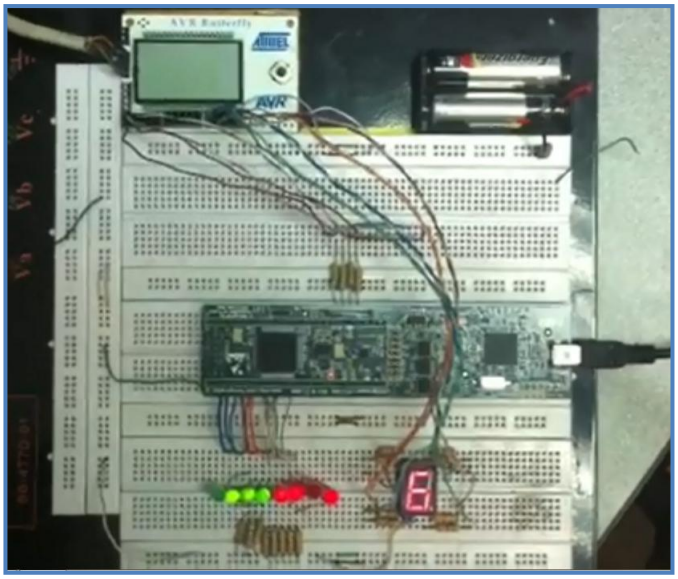

Figura 3.8 Implementación del contador de 0 a 9 entre LPC - AVR

3.3 EJERCICIO 3: CONTADOR DE 0 A 9 ENTRE AVR – LPC, MEDIANTE JOYSTICK

El maestro, tarjeta AVR Butterfly (con microcontrolador Atmega169), envía mediante comunicación SPI, una serie de datos a otra tarjeta LPCxpresso (con microcontrolador LPC1769) que está como esclavo. Este envío de datos es controlado por el Joystick de la AVR Butterfly, que con movimiento de arriba y abajo incrementará y disminuirá respectivamente el contador de uno en uno, y de izquierda a derecha incrementará y disminuirá de dos en dos. Los datos se mostrarán en el puerto 2 de la LPCxpresso en el display de 7 segmentos, como se muestra en la Figura 3.9. Además también se mostrarán en un arreglo de LEDs.

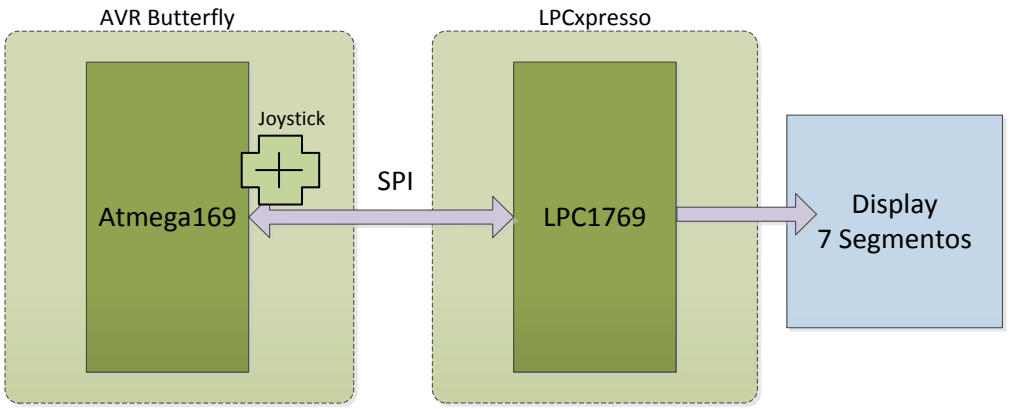

Figura 3.9 Diagrama de Bloques del Ejercicio 3

3.3.1 Descripción del Algoritmo

A continuación se explicará el algoritmo del programa principal para el controlador maestro (AVR Butterfly) y el algoritmo del programa principal para el controlador esclavo (LPCXpresso).

Controlador Maestro

Para el control del dispositivo se inicializan los parámetros y configuramos la tarjeta como maestro, a través de la función void inicializar(void). Seguido activamos globalmente las interrupciones por medio de la función sei(). Luego ingresa a un lazo while infinito y habilita modo sleep y entrar en el modo Power-Down, para que el microcontrolador consuma energía mínima y salga de éste modo únicamente cuando ocurra una interrupción. Si hay interrupciones en los pines que maneja el joystick pasa a identificar la selección hecha (es decir que movimiento del joystick generó la interrupción), para finalmente obtener la selección y enviar el dato al esclavo. Las instrucciones de este algoritmo son ilustradas en la Figura 3.10 y el código fuente se encuentra en el ANEXO D.

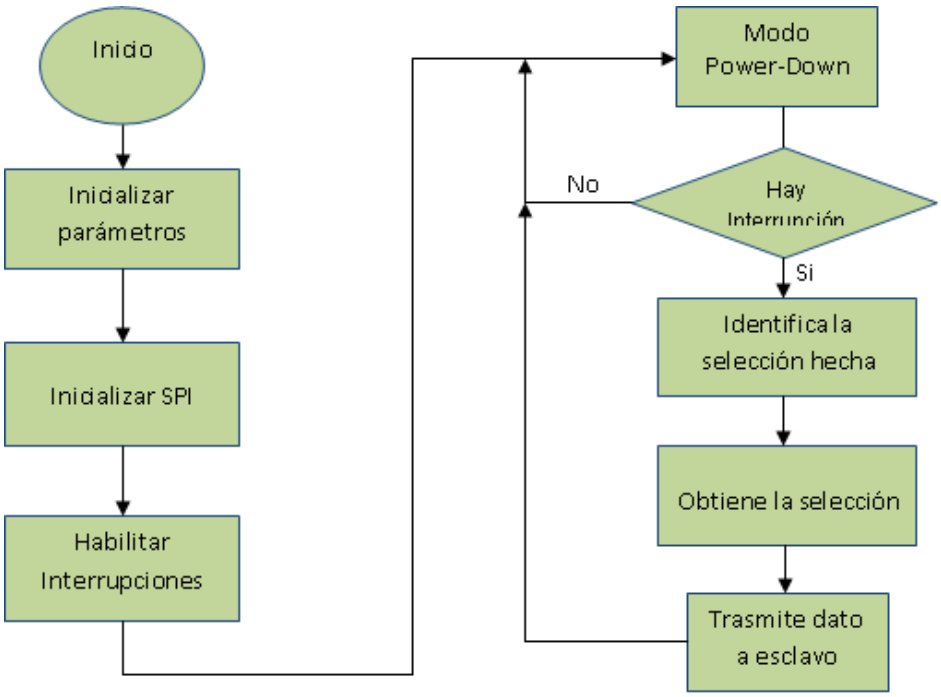

Figura 3.10 Diagrama de Flujo Controlador Maestro Ejercicio 3

Controlador Esclavo

Para el control del dispositivo se inicializan los parámetros PORTNUM, LED, TX_RX_ONLY, SSP_SLAVE. y segmentLUT. Seguido preguntamos si la variable PORTNUM==1 para inicializar una de las interfaces SSP capaz de funcionar en -SPI

Una vez seleccionada la interfaz; preguntamos si TX_RX_ONLY==1 para entrar en modo de Transmisión-Recepción. Luego preguntamos si la variable SSP_SLAVE==1 para indicar si trabaja como esclavo. Para finalizar el dato es recibido del maestro y lo muestra en el puerto 2 llamado LED. Las instrucciones de este algoritmo son ilustradas en la Figura 3.11 y el código fuente se encuentra en el ANEXO D.

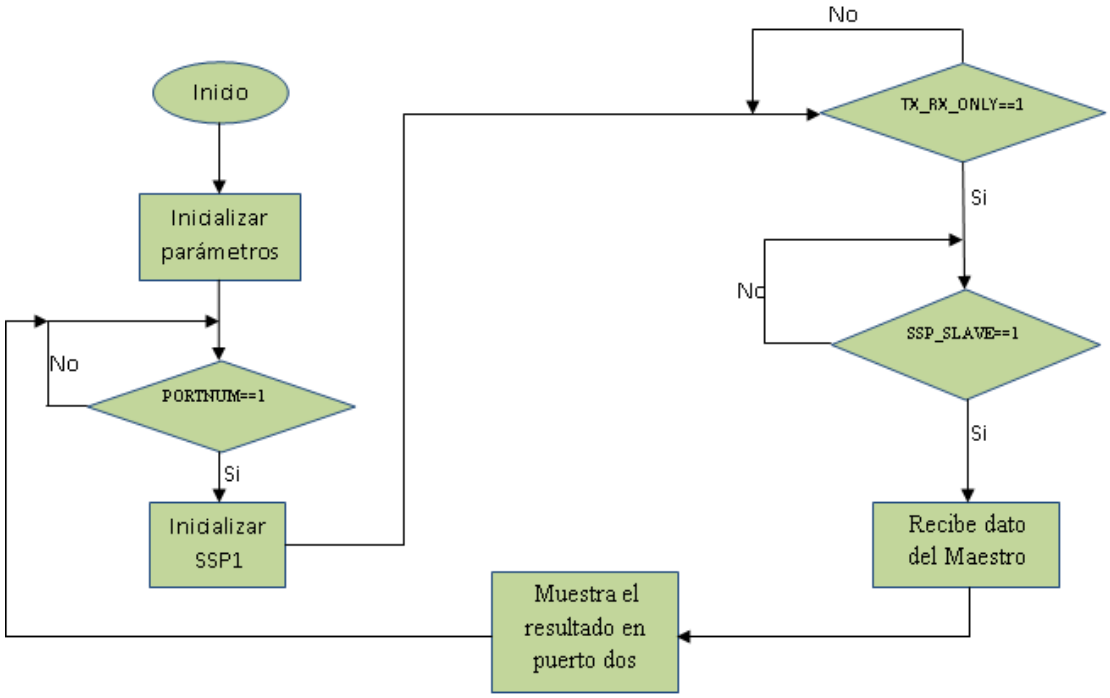

Figura 3.11 Diagrama de Flujo Controlador Esclavo Ejercicio 3

3.3.2 Implementación

Nuestro ejercicio consiste en un CONTADOR CONTROLADO POR JOYSTICK mediante comunicación SPI, donde el AVR funcionará como maestro y la LPC como esclavo. La comunicación se realiza a través de 4 hilos (SEL, SCLK, MISO, MOSI) que es por donde viajan los datos. En la LPC se configura el puerto dos como salida, donde conectaremos un arreglo de LEDs y un display de siete segmentos, como se ilustra en la Figura 3.12. El movimiento hacia arriba del joystick permite avanzar el conteo de 1 en 1; hacia abajo retrocede de 1 en 1; el movimiento hacia la izquierda avanza el conteo de 2 en 2 y hacia la derecha retrocede de 2 en 2.

Materiales:

Una tarjeta AVR Butterfly (ATmega169), tarjeta LPCXpresso (LPC 1769), cuatro resistencias de 1K Ω, dieciséis resistencias de 330 Ω, un display de siete segmentos (cátodo común), ocho LEDs, dos Baterías AA – 1,5 V, un porta baterías AA.

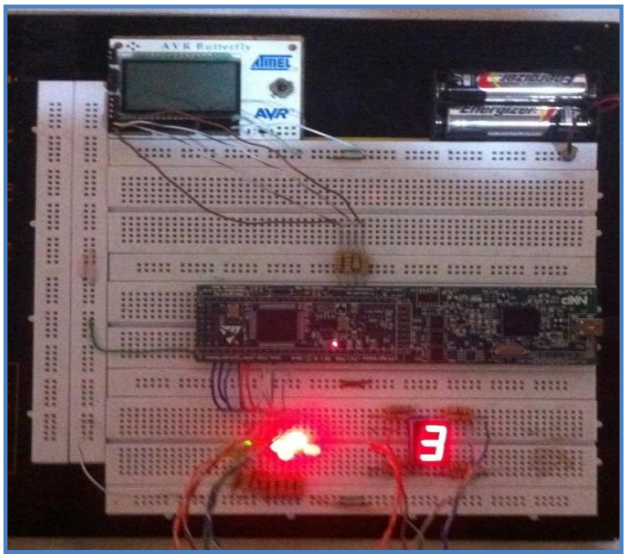

Figura 3.12 Implementación del contador de 0 a 9 entre AVR – LPC, mediante joystick

CAPÍTULO 4

IMPLEMENTACIÓN FINAL

El cuarto capítulo explica la implementación de nuestro proyecto final, que consiste en el control de un motor BLDC, mediante una comunicación SPI entre las tarjetas AVR Butterfly y la tarjeta LPCXpresso, a través de un Joystick.

El capítulo contiene como explicación de nuestro proyecto final, un diagrama de bloques, un diagrama de flujo con su respectiva descripción. Se mostrará la ilustración total de la implementación del proyecto y de los cuatro mandos hechos con el Joystick El diagrama de conexión se encuentran en el ANEXO E.

Para la comunicación SPI de la LPC, al igual que los ejercicios previos, nos basamos en el ejemplo SPP que nos ofrece el software LPCxpresso 4 en el archivo **NXP_LPCXpresso1769_MCB1700_2011-02-11.zip**; del cual indicaremos los cambios realizados en el código para lograr la comunicación entre las tarjetas.

4.1 IMPLEMENTACIÓN FINAL: CONTROL DE MOTOR BLDC ENTRE AVR – LPC, MEDIANTE JOYSTICK

El maestro, tarjeta AVR Butterfly (con microcontrolador Atmega169), envía mediante comunicación SPI, órdenes a la tarjeta LPCxpresso (con microcontrolador LPC1769) que está como esclavo.

El motor es controlado por el Joystick de la AVR Butterfly, que con movimiento de arriba y abajo incrementará y disminuirá la velocidad del motor respectivamente, el movimiento a la izquierda y presionar en el centro encenderá y apagará el motor, el movimiento a la derecha invertirá el giro. Todas estas órdenes se irán presentando en la LCD de la AVR Butterfly. Esas 4 órdenes saldrán en los cuatro primeros bits del puerto 2 de la LPCxpresso; que irán conectados al kit del motor (LPC1114) y mostradas en un arreglo de LEDs. Esta descripción es presentada en la Figura 4.1

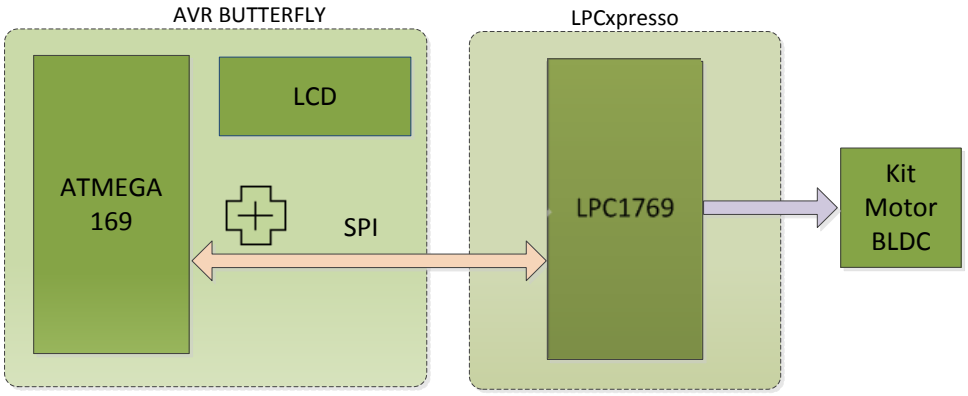

Figura 4.1 Diagrama de Bloques del Proyecto Final.

4.1.1 Descripción del Algoritmo

A continuación se explicará el algoritmo del programa principal para el controlador maestro (AVR Butterfly) y el algoritmo del programa principal para el controlador esclavo (LPCXpresso).

Controlador Maestro

Para el control del dispositivo se inicializan los parámetros y configuramos la tarjeta como maestro, a través de la función void inicializar(void). Seguido activamos globalmente las interrupciones por medio de la función sei(). Luego ingresa a un lazo while infinito y habilita modo sleep y entrar en el modo Power-Down, para que el microcontrolador consuma energía mínima y salga de éste modo únicamente cuando ocurra una interrupción. Si hay interrupciones en los pines que maneja el joystick pasa a identificar la selección hecha (es decir que movimiento del joystick generó la interrupción), a continuación obtiene la selección, muestra en la LCD y finalmente envía el dato al esclavo. Las instrucciones de este algoritmo son ilustradas en la Figura 4.2 y el código fuente se encuentra en el ANEXO D.

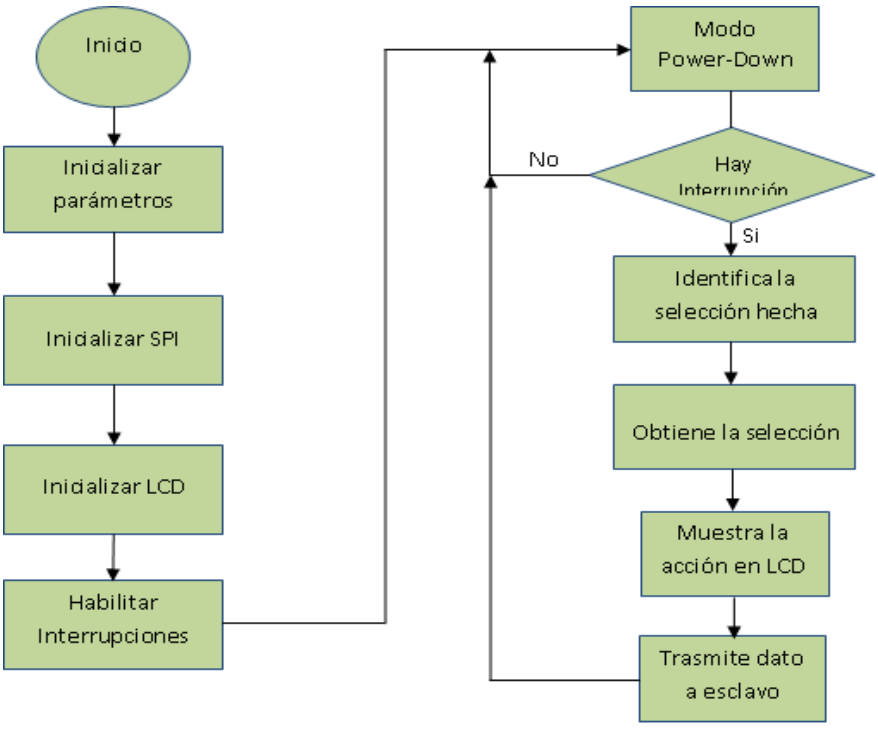

Figura 4.2 Diagrama de Flujo Controlador Maestro del Proyecto

Controlador Esclavo

Para el control del dispositivo se inicializan los parámetros PORTNUM, LED, TX_RX_ONLY, SSP_SLAVE. y segmentLUT. Seguido preguntamos si la variable PORTNUM==1 para inicializar una de las interfaces SSP capaz de funcionar en -SPI

Una vez seleccionada la interfaz; preguntamos si TX_RX_ONLY==1 para entrar en modo de Transmisión-Recepción. Luego preguntamos si la variable SSP_SLAVE==1 para indicar si trabaja como esclavo. Seguido el puerto 2 se lo configura en alto, ya que trabajamos con lógica negativa, y al momento de recibir el dato del maestro muestra el resultado en el puerto 2 con una señal en bajo. Las instrucciones de este algoritmo son ilustradas en la Figura 4.3 y el código fuente se encuentra en el ANEXO D.

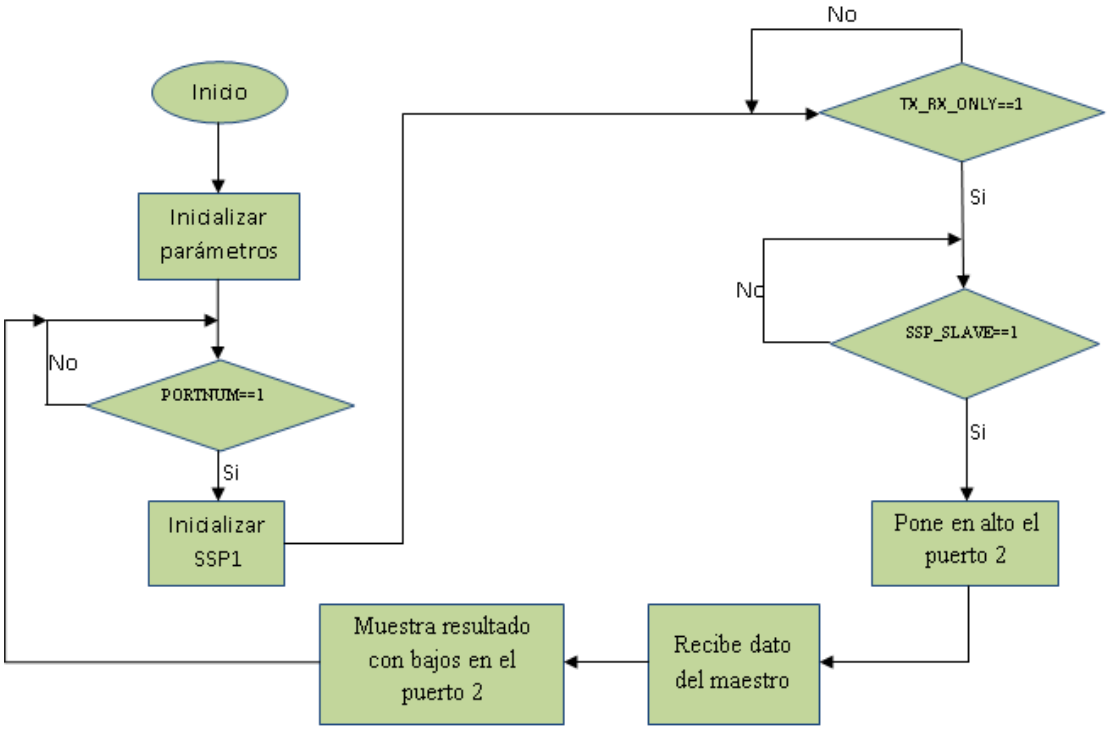

Figura 4.3 Diagrama de Flujo Controlador Esclavo del Proyecto

4.1.2 Aportes al código fuente

A continuación se muestran las líneas de código implementadas en nuestro proyecto final, en conjunto con funciones predeterminadas para el manejo de la tarjeta AVR Butterfly y la Tarjeta LPCXpresso como se muestra en la tabla 4.1 y tabla 4.2 respectivamente, con el fin de lograr la comunicación SPI y enviar las señales de control.

Nombre	Descripción
inicializar_LCD()	Borra el buffer de datos y la memoria del LCD, luego habilita la interface LCD, configura el nivel de contraste y fuente de reloj
escribir_caracter_en_LCD(char , char)	Verifica si el caracter está dentro del rango alfanumérico, busca el SCC correspondiente al caracter dentro de la tabla de caracteres LCD, le da la posición solicitada y copia el SCC en el buffer (registro temporal de datos)
borrar_LCD()	Borra lo que se está visualizando actualmente en el LCD por medio del borrado del contenido de la memoria del LCD y del contenido del buffer de datos SCC.
actualizar_LCD()	Actualiza la memoria LCD con los SCC presentes en el buffer.
inicializar()	Habilita y configura las interrupciones externas por cambio en bit y también para definir cuales pines de los puertos funcionarán como entradas o salidas
manejar_interrupcion()	Usada por las funciones gestoras de interrupción, en ésta se definirá que acciones se debe realizar cuando ocurre una interrupción
obtener_seleccion()	Deshabilita globalmente las interrupciones toma el valor correspondiente a la selección hecha con el joystick y de acuerdo a ésta realiza una de las acciones indicadas
SPI_master_transmit(unsigned char cdata)	Transmite el dato al esclavo

Tabla 4.1 Funciones para manejo de la AVR Butterfly [8]

Nombre	Descripción
SSP1Init()	Rutina de inicialización del puerto SSP
SSPReceive(uint32_t portnum, uint8_t *buf, uint32_t Length)	El módulo recibe un bloque de datos de la SSP, el segundo parámetro es la longitud de bloque.
SSPSend(uint32_t portnum, uint8_t *buf, uint32_t Length)	Envía un bloque de datos al puerto de SSP, el primer parámetro es el puntero de memoria, el segundo parámetro es la longitud de bloque.

Tabla 4.2 Funciones para manejo de la LPCXpresso [9]

Primero.

Se ilustra en la figura 4.4 las variables que se declararon para la selección del dato en la tabla ordenes [ ]. En ***dec*** se guardará el índice correspondiente al dato de la tabla y ***decd*** se guardará el dato seleccionado de la tabla.

```
unsigned char dec, decd;
```

1

Figura 4.4 Variables declaradas para selección del dato en tabla ordenes [ ], AVR

Segundo.

La tabla ***ordenes [ ]*** es la que contiene los datos binarios de cada uno de los mandos enviados a la tarjeta esclava, correspondientes a los movimientos del Joystick. Lo que logramos con esto es enviar un alto para manejar cualquiera de los 4 primeros pines de la tarjeta esclava

```
// Tabla del Dato a enviar según la orden del Joystick
unsigned char ordenes[] = {
0b00000001, //Derecha, Centro
0b00000010, //Izquierda
0b00000100, //Abajo
0b00001000, //Arriba
0b00000000,
};
```

2

Figura 4.5 Tabla de dato según la orden del Joystick, AVR

Tercero.

Para la activación de la comunicación SPI, maestro y ajuste del reloj, se maneja el registro de control del SPI (SPCR), pero antes se configura como salida los pines MOSI, SS, SCK respectivamente; lo cual logramos poniendo un 1 lógico en aquellos bits (00000111 = 07). Para el SPCR manejamos los siguientes bits, tal como se muestra en la figura 4.6:

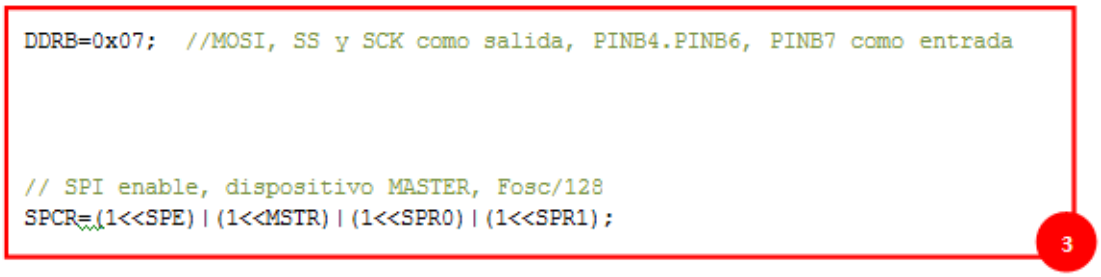

Figura 4.6 Inicialización de SPI y configuración como Maestro, AVR

SPE: al poner 1 lógico activamos la comunicación SPI.

MSTR: al poner 1 lógico activamos el modo maestro.

SPR1 y SPR0: al poner 1 lógico configuramos la velocidad de la Señal de Reloj del SPI.

Cuarto.

En nuestra función *obtener_seleccion ()* usamosun switch, en el cual tenemos 5 casos que son: case ARRIBA, case ABAJO, case IZQUIERDA, case DERECHA, case CENTRO. En todos los casos se maneja la misma secuencia para mostrar el mensaje en la LCD y enviar el dato al esclavo.

Empezamos llamando la función *borrar_LCD* y luego la función *actualizar_LCD* para asegurarnos no haya nada en la memoria. Procedemos a escribir letra por letra el mensaje, mediante la función *escribir_carcater_en_LCD*, dándole a cada una la posición correspondiente en la LCD.

Para el movimiento arriba del Joystick ***dec*** es igual a 3 entonces el valor de la tabla que toma ***decd*** es 00001000 en cual es enviado al esclavo por medio de la función *spi_master_trasmit ()* y mostrado en el puerto D como se ilustra en la figura 4.7.

```
case ARRIBA:
borrar_LCD();
actualizar_LCD();
escribir_caracter_en_LCD('S',0);
escribir_caracter_en_LCD('P',1);
escribir_caracter_en_LCD('E',2);
escribir_caracter_en_LCD('E',3);
escribir_caracter_en_LCD('D',4);
escribir_caracter_en_LCD('+',5);
actualizar_LCD();
dec=3;
decd=ordenes[dec]; //conversión binario a 7 segmentos
SPI_master_transmit(decd); // transmite dato
PORTD=decd; //Muestra numero en el PUERTO D
break;
```

Figura 4.7 Mensaje y envío de orden moviendo el Joystick hacia arriba, AVR

Para el movimiento arriba del Joystick ***dec*** es igual a 2 entonces el valor de la tabla que toma ***decd*** es 00000100 en cual es enviado al esclavo por medio de la función *spi_master_trasmit ()* y mostrado en el puerto D como se ilustra en la figura 4.8.

```
case ABAJO:
borrar_LCD();
actualizar_LCD();
escribir_caracter_en_LCD('S',0);
escribir_caracter_en_LCD('P',1);
escribir_caracter_en_LCD('E',2);
escribir_caracter_en_LCD('E',3);
escribir_caracter_en_LCD('D',4);
escribir_caracter_en_LCD('-',5);
actualizar_LCD();
dec=2;
decd=ordenes[dec]; //conversión binario a 7 segmentos
SPI_master_transmit(decd); // transmite dato
PORTD=decd; //Muestra numero en el PUERTO D
break;
```

5

Figura 4.8 Mensaje y envío de orden moviendo el Joystick hacia la abajo, AVR

Para el movimiento arriba del Joystick ***dec*** es igual a 1 entonces el valor de la tabla que toma ***decd*** es 00000010 en cual es enviado al esclavo por medio de la función *spi_master_trasmit ()* y mostrado en el puerto D como se ilustra en la figura 4.9.

```
case IZQUIERDA:
borrar_LCD();
actualizar_LCD();
escribir_caracter_en_LCD('I',0);
escribir_caracter_en_LCD('N',1);
escribir_caracter_en_LCD('V',2);
escribir_caracter_en_LCD('E',3);
escribir_caracter_en_LCD('R',4);
escribir_caracter_en_LCD('T',5);
actualizar_LCD();
dec=1;
decd=ordenes[dec]; //conversión binario a 7 segmentos
SPI_master_transmit(decd); // transmite dato
PORTD=decd; //Muestra numero en el PUERTO D
break;
```

6

Figura 4.9 Mensaje y envío de orden moviendo el Joystick hacia la izquierda, AVR

Para el movimiento arriba del Joystick ***dec*** es igual a 0 entonces el valor de la tabla que toma ***decd*** es 00000001 en cual es enviado al esclavo por medio de la función *spi_master_trasmit ()* y mostrado en el puerto D como se ilustra en la figura 4.10.

```
case DERECHA:
borrar_LCD();
actualizar_LCD();
escribir_caracter_en_LCD('O',0);
escribir_caracter_en_LCD('N',1);
escribir_caracter_en_LCD('-',2);
escribir_caracter_en_LCD('O',3);
escribir_caracter_en_LCD('F',4);
escribir_caracter_en_LCD('F',5);
actualizar_LCD();
dec=0;
decd=ordenes[dec]; //conversión binario a 7 segmentos
SPI_master_transmit(decd); // transmite dato
PORTD=decd; //Muestra numero en el PUERTO D
break;
```

7

Figura 4.10 Mensaje y envío de orden moviendo el Joystick hacia derecha, AVR

Para el movimiento arriba del Joystick ***dec*** es igual a 0 entonces el valor de la tabla que toma ***decd*** es 00000001 en cual es enviado al esclavo por medio de la función *spi_master_trasmit ()* y mostrado en el puerto D como se ilustra en la figura 4.11.

```
case CENTRO:
borrar_LCD();
actualizar_LCD();
escribir_caracter_en_LCD('O',0);
escribir_caracter_en_LCD('N',1);
escribir_caracter_en_LCD('-',2);
escribir_caracter_en_LCD('O',3);
escribir_caracter_en_LCD('F',4);
escribir_caracter_en_LCD('F',5);
actualizar_LCD();
dec=0;
decd=ordenes[dec]; //conversión binario a 7 segmentos
SPI_master_transmit(decd); // transmite dato
PORTD=decd; //Muestra numero en el PUERTO D
default:
break;
```

8

Figura 4.11 Mensaje y envío de orden presionando el Joystick en el centro, AVR

Quinto.

Definimos el puerto 2 de la LPC Xpresso con la variable *LED*, como se ilustra en la figura 4.12.

```
#define LED LPC_GPIO2  //Definimos el Puerto 2 como LED
```

9

Figura 4.12 Variable declarada para manejo del puerto 2, LPC

Sexto.

Definimos la variable *j*, como se ilustra en la figura 4.13, que usaremos como contador para realizar un retardo.

```
uint32_t j = 0; //Declaramos un contador j
```

10

Figura 4.13 Variable declara para ser usada como contador, LPC

Séptimo.

Con la ayuda de FIODIR declaramos todo el puerto D como salida poniendo 1 lógico en todos los bits, como se ilustra en la figura 4.14.

```
LED -> FIODIR = 0xFF; /*Define port D as output*/
```

11

Figura 4.14 Definición del puerto 2 como salida, LPC

Octavo.

Para el proyecto final, la LPC se usa como esclavo, por lo que el SSPSLAVE toma el valor de 1 previamente, como se indico en la descripción del algoritmo. Llamamos la función *SSPRecive* para recibir el dato enviado del maestro. A continuación manejamos el FIOSET y FIOCLR los cuales ponen en alto o bajo, respectivamente, cuando se les asigna el 1 lógico en sus bits, como se ilustra en la figura 4.15.

Enviamos todos los pines del puerto 2 (*LED*) a alto, mediante *FIOSET = 0xFF*, ya que el Kit del Motor funciona con bajos. Preguntamos si el dato que llega a *destaddr* es distinto de *0xFF*(para verificar cualquier error) y mientras lo sea realice la acción FIOCLR según el 1 lógico que tenga el dato guardado en *destaddr*; para luego de un retardo con un lazo for volvemos a poner en alto todos los pines del puerto D.

```
#if SSP_SLAVE
    /* Slave receive */
        SSPReceive( portnum, (uint8_t *)dest_addr, 1);

        LED -> FIOSET = 0xFF; //hacemos alto todos los pines del puerto 2
  if (dest_addr[0]!= 0xFF)
  {
        //hacemos bajo el pin correspondiente a la orden enviada por el maestro
        LED -> FIOCLR =  dest_addr[0];
        for(j = 2000000; j > 0; j--);//retardo
        LED -> FIOSET = 0xFF;//hacemos alto todos los pines del puerto 2
  }
#else
  /* Master transmit */
  SSPSend( portnum, (uint8_t *)src_addr, SSP_BUFSIZE);
#endif
```

Figura 4.15 Transmisión o Recepción de dato, LPC

4.1.3 Implementación

El AVR Butterfly, con microcontrolador Atmega169; la LPCXpresso con microcontrolador LPC1769; el Kit del motor con tarjeta LPCXpresso (LPC1114) y el motor BLDC, son los dispositivos usados, como se ilustra en la figura 4.16.

La comunicación SPI se realiza a través de 4 hilos (SEL, SCLK, MISO, MOSI) entre el AVR Butterfly (maestro) y la LPC1769 (esclavo), en este último se configura 4 pines del puerto dos como salidas para la conexión con el KIT del Motor, cuyo funcionamiento se lo obtiene enviando por estos pines un bajo para realizar los diferentes movimientos del motor BLDC.

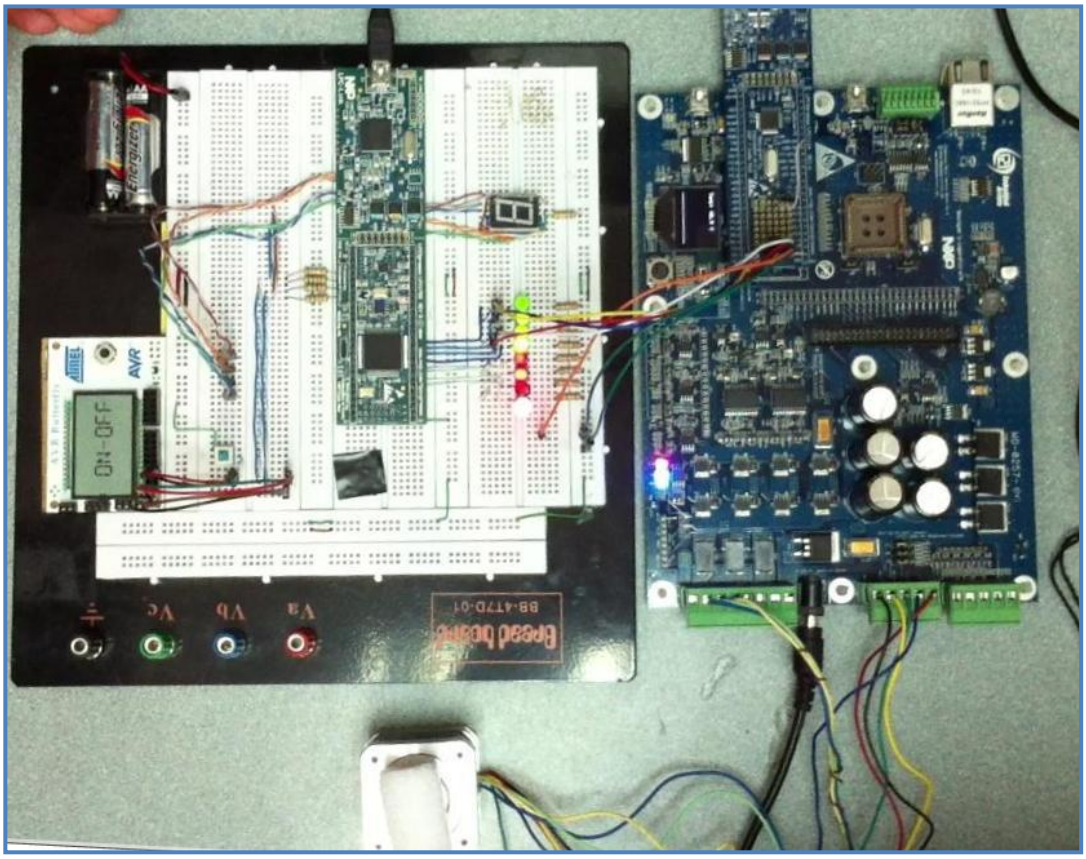

Figura 4.16 Implementación del controlador de motor BLDC mediante Joystick

Materiales:

Una tarjeta AVR Butterfly (ATmega169), tarjeta LPCXpresso (LPC 1769), tarjeta LPCXpresso (LPC 1114),cuatro resistencias de 1K Ω, dieciséis resistencias de 330 Ω, ocho LEDs, dos Baterías AA – 1,5 V, un porta baterías AA.

El joystick tiene 5 movimientos, los cuales fueron configurados para el envío de 4 órdenes, cuyos reconocimientos se realizan en un estado bajo, como se muestra en la tabla 4.1 y se ilustra en la figura 4.3

JOYSTICK (movimiento)	**ORDENES**	**LCD**	**LPC1769**	**LPC1114**
Arriba	Aumentar Velocidad	SPEED+	Port2 [3]	PIO3_3
Abajo	Disminuir Velocidad	SPEED-	Port2 [2]	PIO3_2
Derecha	Cambio de Giro	INVERT	Port2 [1]	PIO3_1
Izquierda o Centro	Arranque/Pare	ON-OFF	Port2 [0]	PIO2_4

Tabla 4.3 Asignación de pines para el funcionamiento del motor BLDC.

El aumento o disminución de la velocidad del motor es de 50 rpm cada vez que se mueve el Joystick ya sea Arriba o Abajo, el motor se mueve mediante un controlador PID y llega hasta una velocidad máxima de 4100 rpm y mínima de 600 rpm (es decir no se detiene).

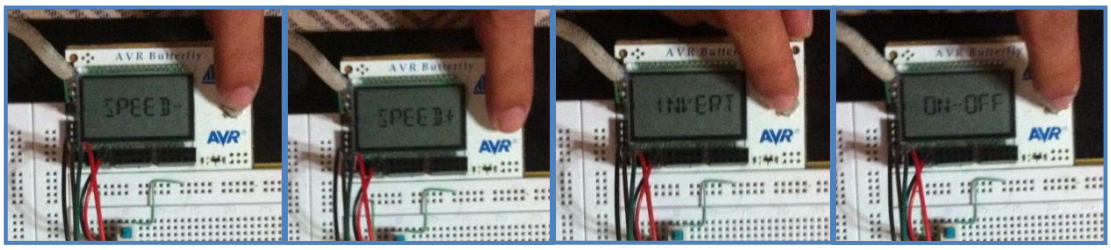

Figura 4.17 Órdenes enviadas por el Joystick.

CONCLUSIONES

1.- El protocolo SPI es una interfaz de comunicación que permite controlar varios periféricos pudiendo utilizar para todos ellos un mismo bus de datos. Esta comunicación nos permitió diseñar un sistema maestro – esclavo, cuyo fin consiste en el control del motor BLDC mediante un intercambio de datos utilizando comunicación SPI entre las dos tarjetas empleadas en nuestro proyecto (AVR - LPC).

2.- Esta nueva estrategia de control presentada, a través de modernos microcontroladores, nos brinda una característica destacable del sistema, debido al gran grado de integración que existe entre esta variedad de tarjetas. Las tarjetas de

control utilizadas en este proyecto resultan mucho más compactas que la de los sistemas de control basadas en componentes análogos. Facilitando así su manejo, instalación y programación.

3.- En cuanto a los resultados experimentales, estos han demostrado que la estructura de control propuesta, basada en el manejo mediante joystick de un motor BLDC, permite debido al gran grado de integración de las tarjetas y al fácil control del joystick, una total confiabilidad en el mando del motor.

4.- El desarrollo de este proyecto fue realizado en lenguaje C haciendo uso de las herramientas como el AVR Studio 4 que sirve para programar el micro controlador ATmega 169, y del LPCXpresso 4 que sirve para programar la tarjeta LPC 1769, convirtiéndose ambas herramientas en el corazón de nuestro proyecto. Siendo el lenguaje C el más óptimo en el desarrollo de software para microcontroladores, ya que permite segmentar el código fuente en varias funciones especializadas y archivos, permitiéndole al programador reusar las mismas funciones en otras aplicaciones.

RECOMENDACIONES

1.- Entre las cosas a tomar en cuenta cuando se trata de implementar una comunicación SPI. Una de estas es el cableado. Es importante asegurarse de que la entrada serie del microcontrolador esté conectada correctamente a la salida serie del microcontrolador con el que se realiza las comunicaciones.

2.- En lo que a programación se refiere se recomienda como primer paso realizar pruebas demo, con los muestras que ya vienen incorporados en cada software, que nos permitió como por ejemplo en particular, familiarizarnos con el software AVRSTUDIO 4 para la programación del AVR Butterfly, realizando pruebas como es prender la pantalla LCD y ver los comandos que tiene el Joystick.

3.- Gracias a la variedad de experimentos realizados acerca de la comunicación mediante el protocolo SPI entre diferentes combinaciones de dispositivos empleados, podemos recomendar como ingenieros el empleo de este protocolo ya que según nosotros es el más rápido, eficaz y sencillo, debido a que los comandos o instrucciones necesarios para operar el protocolo son específicos y relativamente simples, como también el empleo de cuatro líneas de comunicación que en realidad es lo más destacable en lo que al protocolo se refiere.

ANEXO A

CONFIGURACIÓN BÁSICA DEL SPI DE LA LPC1769 - MODOS Y FRECUENCIA DEL RELOJ DEL ATMega169

SPI DE LA LPC1769

1. **Power**: En el registro PCONP (tabla 2.1), seteamos el bit PCSPI.

   Nota: Al resetear, el SPI es habilitado (PCSPI=1).

Bit	Symbol	Description	Reset value
0	-	Reserved.	NA
1	PCTIM0	Timer/Counter 0 power/clock control bit.	1
2	PCTIM1	Timer/Counter 1 power/clock control bit.	1
3	PCUART0	UART0 power/clock control bit.	1
4	PCUART1	UART1 power/clock control bit.	1
5	-	Reserved.	NA
6	PCPWM1	PWM1 power/clock control bit.	1
7	PCI2C0	The I^2C0 interface power/clock control bit.	1
8	PCSPI	The SPI interface power/clock control bit.	1
9	PCRTC	The RTC power/clock control bit.	1
10	PCSSP1	The SSP 1 interface power/clock control bit.	1
11	-	Reserved.	NA
12	PCADC	A/D converter (ADC) power/clock control bit. **Note:** Clear the PDN bit in the AD0CR before clearing this bit, and set this bit before setting PDN.	0
13	PCCAN1	CAN Controller 1 power/clock control bit.	0
14	PCCAN2	CAN Controller 2 power/clock control bit.	0
15	PCGPIO	Power/clock control bit for IOCON, GPIO, and GPIO interrupts.	1
16	PCRIT	Repetitive Interrupt Timer power/clock control bit.	0
17	PCMCPWM	Motor Control PWM	0
18	PCQEI	Quadrature Encoder Interface power/clock control bit.	0
19	PCI2C1	The I^2C1 interface power/clock control bit.	1
20	-	Reserved.	NA
21	PCSSP0	The SSP0 interface power/clock control bit.	1
22	PCTIM2	Timer 2 power/clock control bit.	0
23	PCTIM3	Timer 3 power/clock control bit.	0
24	PCUART2	UART 2 power/clock control bit.	0
25	PCUART3	UART 3 power/clock control bit.	0
26	PCI2C2	I^2C interface 2 power/clock control bit.	1
27	PCI2S	I^2S interface power/clock control bit.	0
28	-	Reserved.	NA
29	PCGPDMA	GPDMA function power/clock control bit.	0
30	PCENET	Ethernet block power/clock control bit.	0
31	PCUSB	USB interface power/clock control bit.	0

Tabla A.1 Registro Control de Poder para Periféricos (PCONP).[3]

2. **Clock**: En el registro PCLKSEL0 (tabla 2.2), seteamos el bit PCLK_SPI. En el modo master, el reloj debe ser un número incluso mayor que o igual a 8.

Bit	Symbol	Description	Reset value
1:0	PCLK_WDT	Peripheral clock selection for WDT.	00
3:2	PCLK_TIMER0	Peripheral clock selection for TIMER0.	00
5:4	PCLK_TIMER1	Peripheral clock selection for TIMER1.	00
7:6	PCLK_UART0	Peripheral clock selection for UART0.	00
9:8	PCLK_UART1	Peripheral clock selection for UART1.	00
11:10	-	Reserved.	NA
13:12	PCLK_PWM1	Peripheral clock selection for PWM1.	00
15:14	PCLK_I2C0	Peripheral clock selection for I^2C0.	00
17:16	PCLK_SPI	Peripheral clock selection for SPI.	00
19:18	-	Reserved.	NA
21:20	PCLK_SSP1	Peripheral clock selection for SSP1.	00
23:22	PCLK_DAC	Peripheral clock selection for DAC.	00
25:24	PCLK_ADC	Peripheral clock selection for ADC.	00
27:26	PCLK_CAN1	Peripheral clock selection for CAN1.[1]	00
29:28	PCLK_CAN2	Peripheral clock selection for CAN2.[1]	00
31:30	PCLK_ACF	Peripheral clock selection for CAN acceptance filtering.[1]	00

[1] PCLK_CAN1 and PCLK_CAN2 must have the same PCLK divide value when the CAN function is used.

Tabla A.2 Registro Selección del Reloj Periférico 0 (PCLKSEL0). [3]

3. **Pines:** Los pines del SPI son configurados usando el PINSEL0 (tabla 2.3) y el PINSEL1 (tabla 2.4). Así como el registro pin MODE (tabla 2.5). El PINSEL0[31:30] es usado para configurar el pin SPI CLK. PINSEL1[1:0], PINSEL1[3:2] Y PINSEL1[5:4] son usados para configurar los pines SSEL, MISO, Y MOSI, respectivamente.

PINSEL0	Pin name	Function when 00	Function when 01	Function when 10	Function when 11	Reset value
1:0	P0.0	GPIO Port 0.0	RD1	TXD3	SDA1	00
3:2	P0.1	GPIO Port 0.1	TD1	RXD3	SCL1	00
5:4	P0.2	GPIO Port 0.2	TXD0	AD0.7	Reserved	00
7:6	P0.3	GPIO Port 0.3	RXD0	AD0.6	Reserved	00
9:8	P0.4[1]	GPIO Port 0.4	I2SRX_CLK	RD2	CAP2.0	00
11:10	P0.5[1]	GPIO Port 0.5	I2SRX_WS	TD2	CAP2.1	00
13:12	P0.6	GPIO Port 0.6	I2SRX_SDA	SSEL1	MAT2.0	00
15:14	P0.7	GPIO Port 0.7	I2STX_CLK	SCK1	MAT2.1	00
17:16	P0.8	GPIO Port 0.8	I2STX_WS	MISO1	MAT2.2	00
19:18	P0.9	GPIO Port 0.9	I2STX_SDA	MOSI1	MAT2.3	00
21:20	P0.10	GPIO Port 0.10	TXD2	SDA2	MAT3.0	00
23:22	P0.11	GPIO Port 0.11	RXD2	SCL2	MAT3.1	00
29:24	-	Reserved	Reserved	Reserved	Reserved	0
31:30	P0.15	GPIO Port 0.15	TXD1	SCK0	SCK	00

Tabla A.3 Registro de Selección de Función de PIN, 0 (PINSEL0). [3]

PINSEL1	Pin name	Function when 00	Function when 01	Function when 10	Function when 11	Reset value
1:0	P0.16	GPIO Port 0.16	RXD1	SSEL0	SSEL	00
3:2	P0.17	GPIO Port 0.17	CTS1	MISO0	MISO	00
5:4	P0.18	GPIO Port 0.18	DCD1	MOSI0	MOSI	00
7:6	P0.19[1]	GPIO Port 0.19	DSR1	Reserved	SDA1	00
9:8	P0.20[1]	GPIO Port 0.20	DTR1	Reserved	SCL1	00
11:10	P0.21[1]	GPIO Port 0.21	RI1	Reserved	RD1	00
13:12	P0.22	GPIO Port 0.22	RTS1	Reserved	TD1	00
15:14	P0.23[1]	GPIO Port 0.23	AD0.0	I2SRX_CLK	CAP3.0	00
17:16	P0.24[1]	GPIO Port 0.24	AD0.1	I2SRX_WS	CAP3.1	00
19:18	P0.25	GPIO Port 0.25	AD0.2	I2SRX_SDA	TXD3	00
21:20	P0.26	GPIO Port 0.26	AD0.3	AOUT	RXD3	00
23:22	P0.27[1][2]	GPIO Port 0.27	SDA0	USB_SDA	Reserved	00
25:24	P0.28[1][2]	GPIO Port 0.28	SCL0	USB_SCL	Reserved	00

Tabla A.4 Registro Selección de Función de PIN, 1 (PINSEL1). [3]

PINMODE0 to PINMODE9 Values	Function	Value after Reset
00	Pin has an on-chip pull-up resistor enabled.	00
01	Repeater mode (see text below).	
10	Pin has neither pull-up nor pull-down resistor enabled.	
11	Pin has an on-chip pull-down resistor enabled.	

Tabla A.5 Registro de Selección de Modo de PIN, Bits. [3]

4. **Interrupciones**: La bandera de interrupción del SPI es habilitada usando el bit SOSPINT[0] (tabla A.6). La bandera de interrupción del SPI debería ser habilitada en el NVIC (tabla A.7).

Bit	Symbol	Description	Reset Value
0	SPIF	SPI interrupt flag. Set by the SPI interface to generate an interrupt. Cleared by writing a 1 to this bit. **Note:** this bit will be set once when SPIE = 1 and at least one of SPIF and WCOL bits is 1. However, only when the SPI Interrupt bit is set and SPI0 Interrupt is enabled in the NVIC, SPI based interrupt can be processed by interrupt handling software.	0
7:1	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
31:8	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Tabla A.6 Registro de Interrupción SPI (S0SPINT). [3]

Interrupt ID	Exception Number	Vector Offset	Function	Flag(s)
0	16	0x40	WDT	Watchdog Interrupt (WDINT)
1	17	0x44	Timer 0	Match 0 - 1 (MR0, MR1) Capture 0 - 1 (CR0, CR1)
2	18	0x48	Timer 1	Match 0 - 2 (MR0, MR1, MR2) Capture 0 - 1 (CR0, CR1)
3	19	0x4C	Timer 2	Match 0-3 Capture 0-1
4	20	0x50	Timer 3	Match 0-3 Capture 0-1
5	21	0x54	UART0	Rx Line Status (RLS) Transmit Holding Register Empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI) End of Auto-Baud (ABEO) Auto-Baud Time-Out (ABTO)
6	22	0x58	UART1	Rx Line Status (RLS) Transmit Holding Register Empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI) Modem Control Change End of Auto-Baud (ABEO) Auto-Baud Time-Out (ABTO)
7	23	0x5C	UART 2	Rx Line Status (RLS) Transmit Holding Register Empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI) End of Auto-Baud (ABEO) Auto-Baud Time-Out (ABTO)
8	24	0x60	UART 3	Rx Line Status (RLS) Transmit Holding Register Empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI) End of Auto-Baud (ABEO) Auto-Baud Time-Out (ABTO)
9	25	0x64	PWM1	Match 0 - 6 of PWM1 Capture 0-1 of PWM1
10	26	0x68	I^2C0	SI (state change)
11	27	0x6C	I^2C1	SI (state change)
12	28	0x70	I^2C2	SI (state change)
13	29	0x74	SPI	SPI Interrupt Flag (SPIF) Mode Fault (MODF)

Tabla A.7 Conexión de la Fuente de interrupción del controlador del vector interrupción. [3]

Interrupt ID	Exception Number	Vector Offset	Function	Flag(s)
14	30	0x78	SSP0	Tx FIFO half empty of SSP0
				Rx FIFO half full of SSP0
				Rx Timeout of SSP0
				Rx Overrun of SSP0
15	31	0x7C	SSP 1	Tx FIFO half empty
				Rx FIFO half full
				Rx Timeout
				Rx Overrun
16	32	0x80	PLL0 (Main PLL)	PLL0 Lock (PLOCK0)
17	33	0x84	RTC	Counter Increment (RTCCIF)
				Alarm (RTCALF)
18	34	0x88	External Interrupt	External Interrupt 0 (EINT0)
19	35	0x8C	External Interrupt	External Interrupt 1 (EINT1)
20	36	0x90	External Interrupt	External Interrupt 2 (EINT2)
21	37	0x94	External Interrupt	External Interrupt 3 (EINT3).
				Note: EINT3 channel is shared with GPIO interrupts
22	38	0x98	ADC	A/D Converter end of conversion
23	39	0x9C	BOD	Brown Out detect
24	40	0xA0	USB	USB_INT_REQ_LP, USB_INT_REQ_HP, USB_INT_REQ_DMA
25	41	0xA4	CAN	CAN Common, CAN 0 Tx, CAN 0 Rx, CAN 1 Tx, CAN 1 Rx
26	42	0xA8	GPDMA	IntStatus of DMA channel 0, IntStatus of DMA channel 1
27	43	0xAC	I^2S	irq, dmareq1, dmareq2
28	44	0xB0	Ethernet	WakeupInt, SoftInt, TxDoneInt, TxFinishedInt, TxErrorInt, TxUnderrunInt, RxDoneInt, RxFinishedInt, RxErrorInt, RxOverrunInt.
29	45	0xB4	Repetitive Interrupt Timer	RITINT
30	46	0xB8	Motor Control PWM	IPER[2:0], IPW[2:0], ICAP[2:0], FES
31	47	0xBC	Quadrature Encoder	INX_Int, TIM_Int, VELC_Int, DIR_Int, ERR_Int, ENCLK_Int, POS0_Int, POS1_Int, POS2_Int, REV_Int, POS0REV_Int, POS1REV_Int, POS2REV_Int
32	48	0xC0	PLL1 (USB PLL)	PLL1 Lock (PLOCK1)
33	49	0xC4	USB Activity Interrupt	USB_NEED_CLK
34	50	0xC8	CAN Activity Interrupt	CAN1WAKE, CAN2WAKE

Tabla A.7 Conexión de la Fuente de interrupción del controlador del vector interrupción. [3]

Modos del reloj definidos por el protocolo SPI

	Modo 0	Modo 1	Modo 2	Modo 3
CPOL	1	0	1	0
CPHA	0	1	0	0

Tabla A.8 Selección de modo de reloj del SPI

Combinación de bits para seleccionar la frecuencia del reloj.

SPI2X	**SPR1**	**SPR0**	**Frecuencia de SCLK**
0	**0**	**0**	Fosc/4
0	**0**	**1**	Fosc/16
0	**1**	**0**	Fosc/64
0	**1**	**1**	Fosc/128
1	**0**	**0**	Fosc/2
1	**0**	**1**	Fosc/8
1	**1**	**0**	Fosc/32
1	**1**	**1**	Fosc/64

Tabla A.9 Selección de frecuencia de la señal de reloj del SPI [4]

ANEXO B

HERRAMIENTAS DE DISEÑO E IMPLEMENTACIÓN

HERRAMIENTAS DE SOFTWARE

Para desarrollar el proyecto hicimos uso de dos tipos de software: **AVR Studio 4**, cuyo fin es la programación del ATmega169 y **LPCXPRESSO 4** que nos servirá para la programación de la LPC1769.

AVR Studio 4

AVR Studio es un entorno de desarrollo integrado que incluye un editor, el ensamblador, descargador de archivos HEX y un emulador de microcontroladores. Este software permite escribir y depurar aplicaciones AVR en entornos de Windows, que proporciona una gestión de proyectos de herramientas, fuente de editor de archivos, simulador, ensamblador. Para programar el microcontrolador AVR Atmel usando el lenguaje de programación C, también necesitaremos de una herramienta extra dentro del AVR Studio, que es el WinAVR, el cual consiste en un compilador para AVR basado en GCC. Este aparece en AVR Studio como un plug-in.

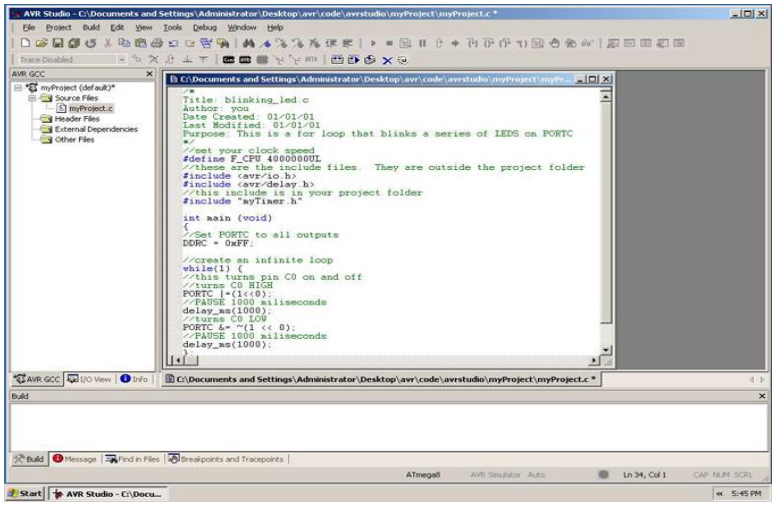

Figura B.1 Entorno gráfico AVR Studio 4

LPCXpresso 4

LPCXpresso (creado por Code Red) es un software altamente integrado desarrollado para trabajar con microcontroladores LPC de NXP, que incluye todas las herramientas necesarias para desarrollar soluciones de software de alta calidad

El software consiste de algunos aumentos para su mejora, como: IDE basado en Eclipse, un compilador de C basado en GNU, links, librerías, y un depurador GDB mejorado

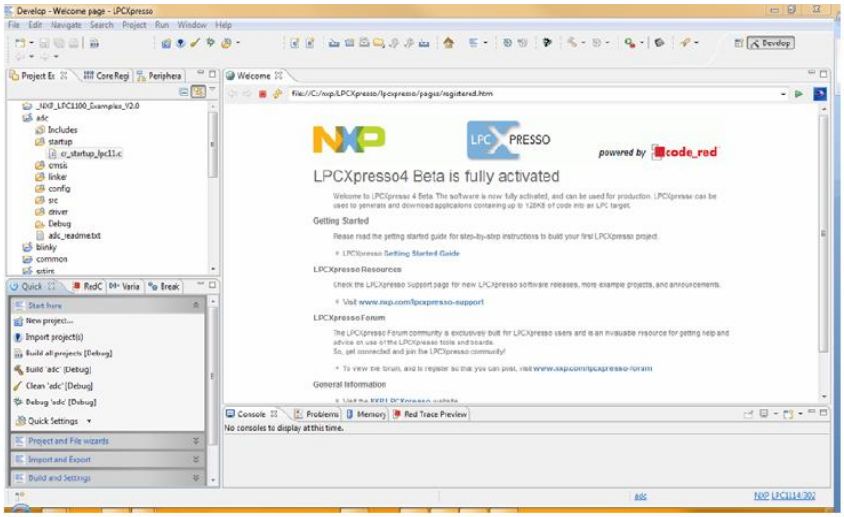

Figura B.2 Entorno gráfico LPCXpresso 4 IDE

HERRAMIENTAS DE HARDWARE

En lo que hace referencia a la parte física, el hardware es la implementación de las tarjetas LPCXpresso y AVR Butterfly, para que comiencen a trabajar como Esclavo y Maestro respectivamente. En este capítulo se muestra información de los componentes utilizados para la implementación del proyecto

AVR Butterfly (ATmega169)

AVR Butterfly es un módulo de soporte que puede ser utilizado en numerosas aplicaciones. El AVR Butterfly contiene un microcontrolador ATMega169.

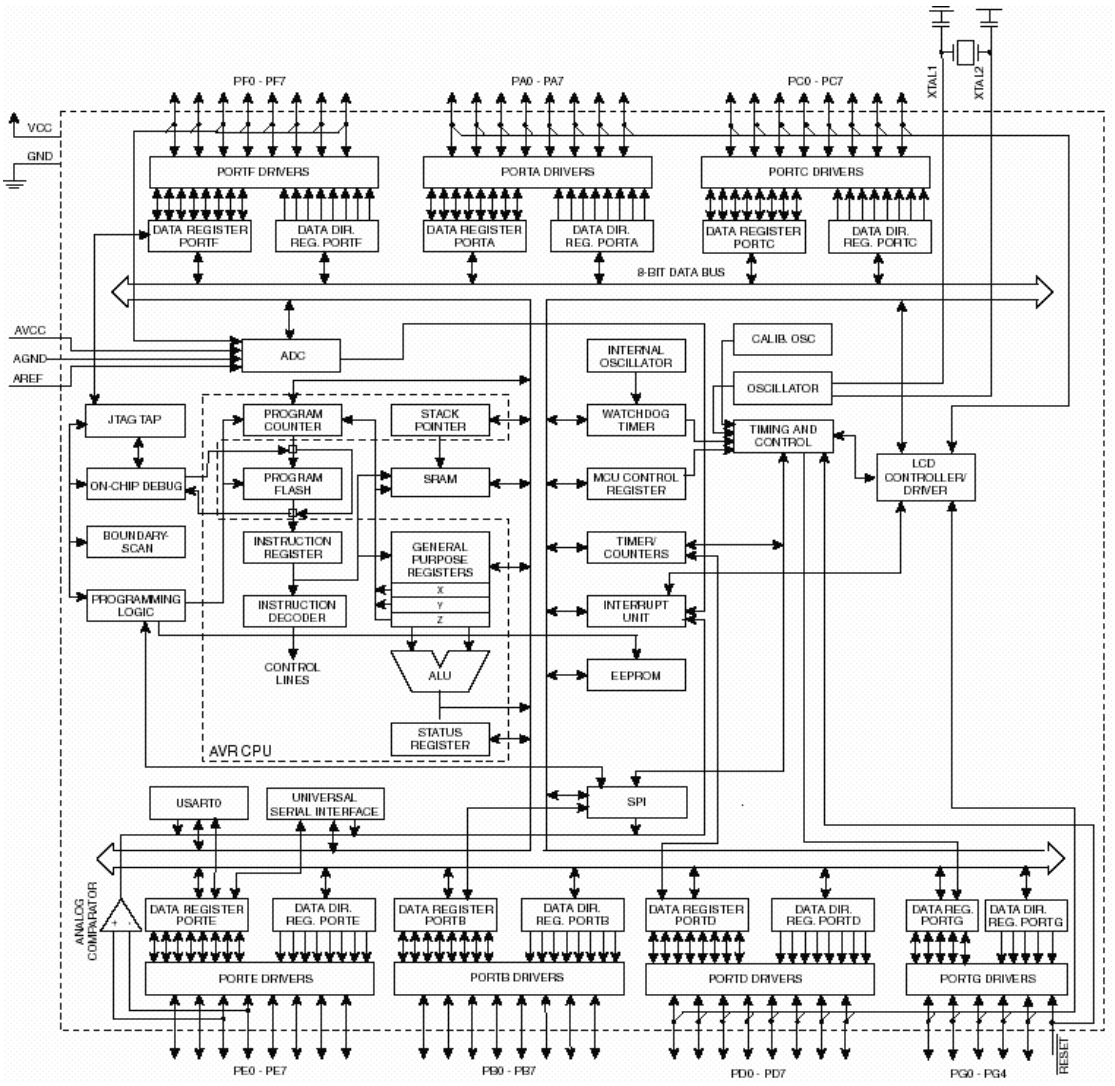

Figura B.3 Diagrama de Bloques del Atmega169 [6]

LPCXpresso (LPC1769)

La tarjeta LPCXpresso con microcontrolador LPC1769 sirve para creación de aplicaciones embebidas que requieren un alto nivel de integración y baja disipación de potencia

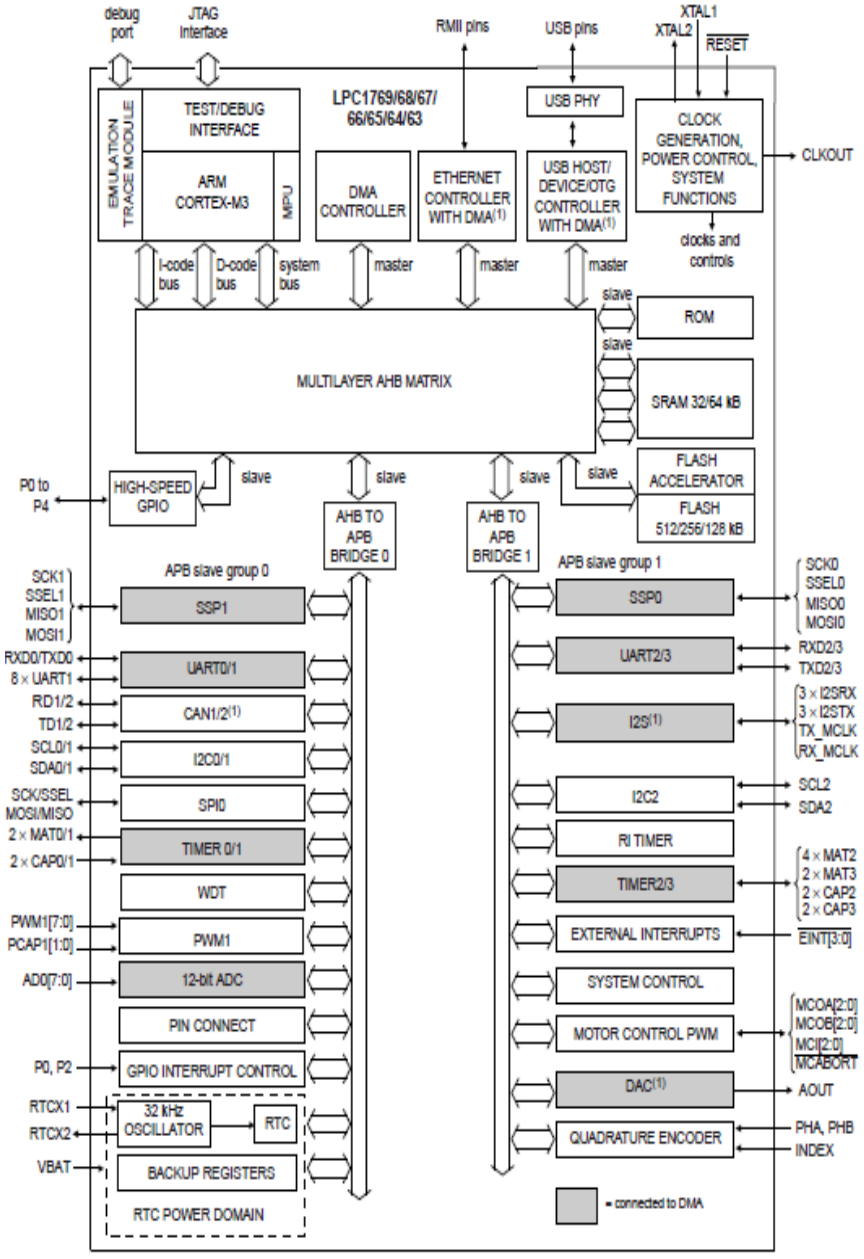

Figura B.4 Diagrama de bloques de la LPC1769 [7]

ANEXO C

GUIAS PARA PROGRAMAR TARJETAS

AVR BUTTERFLY

Para realizar la programación del AVR Butterfly con la PC, daremos a conocer al usuario el siguiente procedimiento:

1. Realizar la conexión del cable, para la interfaz serial RS-232 entre la PC y el AVR Butterfly, de la siguiente manera:

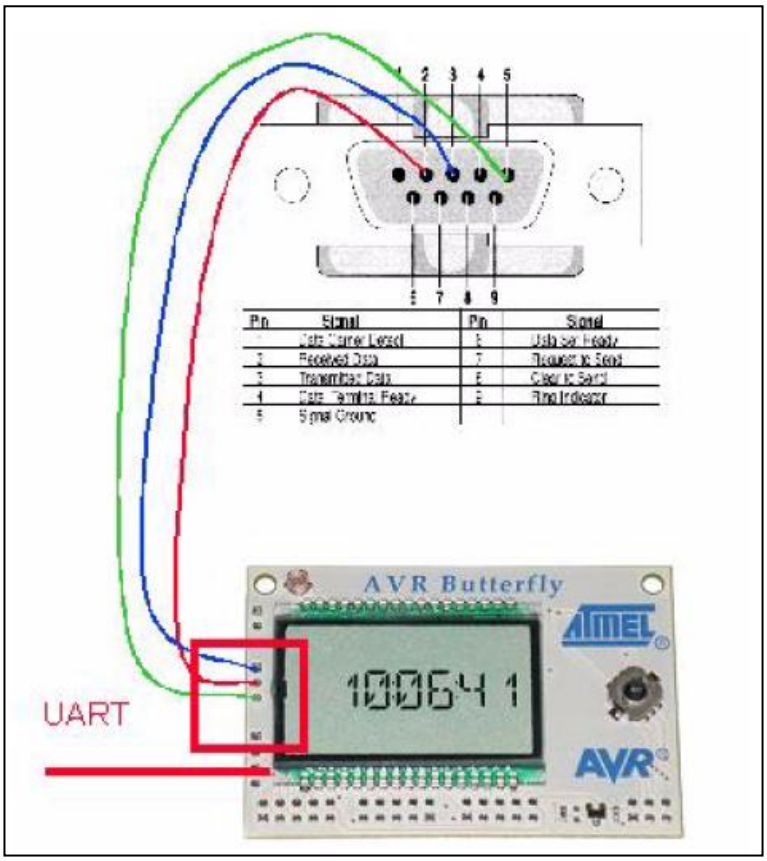

Figura C.1 Conexión para interfaz RS-232 entre PC y AVR

2. Ejecutar el software AVR Studio 4
3. Ejercer presión en el joystick del AVR Butterfly, en el centro y mantenerlo así.
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,

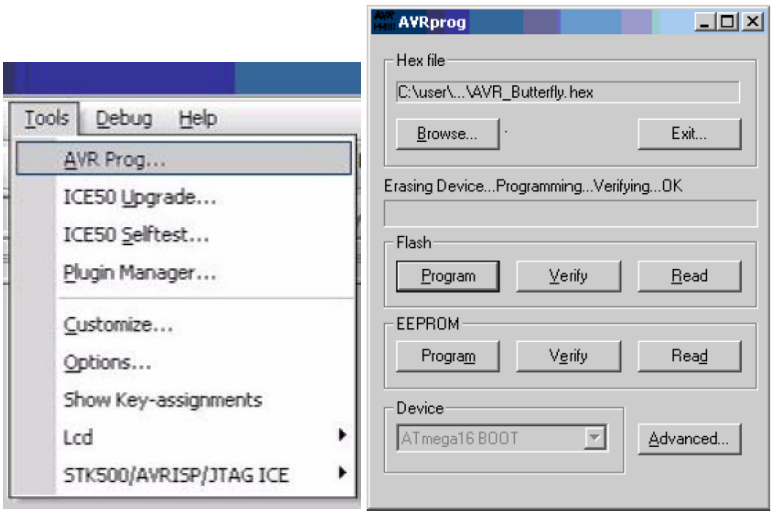

Figura C.2 ventana para cargar archivo .hex al AVR

6. Quitar la presión que se mantiene sobre el joystick del AVR Butterfly
7. Clic en el botón Browser de la Ventana AVRprog, para localizar y cargar el archivo HEX generado con la compilación en el directorio del proyecto.
8. clic en el botón Program de la ventana AVRprog, para programar el microcontrolador ATmega169V del AVR Butterfly y actualizarlo con la nueva aplicación.
9. Clic el botón Exit de la ventana AVRprog, para salir del modo de programación.

LPCXPRESSO.

Para realizar la programación del LCPxpresso con la PC, daremos a conocer al usuario el siguiente procedimiento.

En nuestro caso, lo hicimos a partir de un ejemplo, así como se va indicar a continuación.

1. Para comenzar el desarrollo, la LPCXpresso se puede conectar a un PC mediante un puerto USB 2.0 A / Mini-B del cable.

Figura C.3 Cable USB 2.0 A / Mini

2. Para trabajar con un ejemplo de los que ofrece NXP , seleccione “Import project(s)” desde el panel Quickstart en la esquina inferior izquierda de la pantalla.

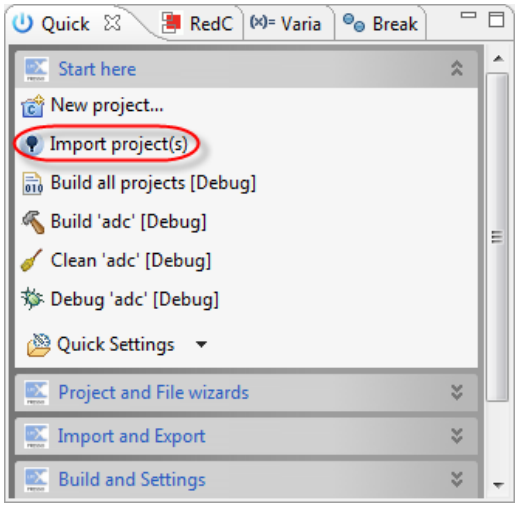

Figura C.4 Panel Quickstart del LPCXpresso IDE

3. Seleccionamos Browser en Proyect Archive (zid), para poder acceder al directorio donde se encuentran por default los ejemplos que ofrece el programa

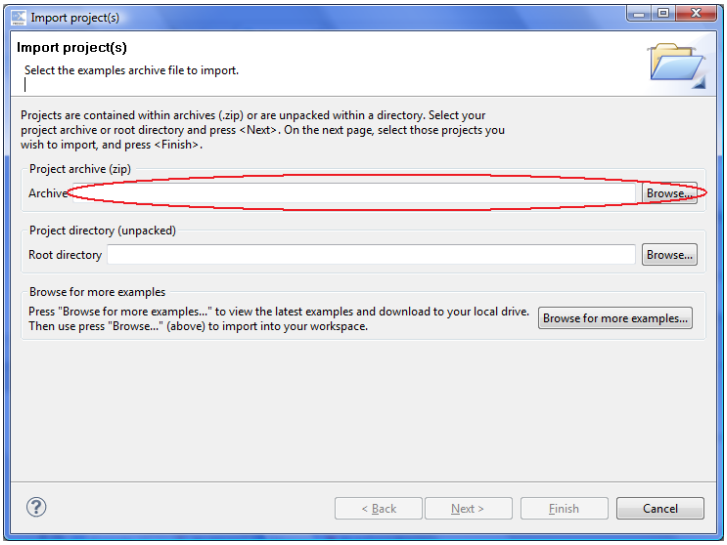

Figura C.5 Ventana para importar proyectos

4. Seguido cuando ya se haya importado el archivo damos NEXT y aparecerá una pantalla para seleccionar el ejemplo una vez hecho es damos

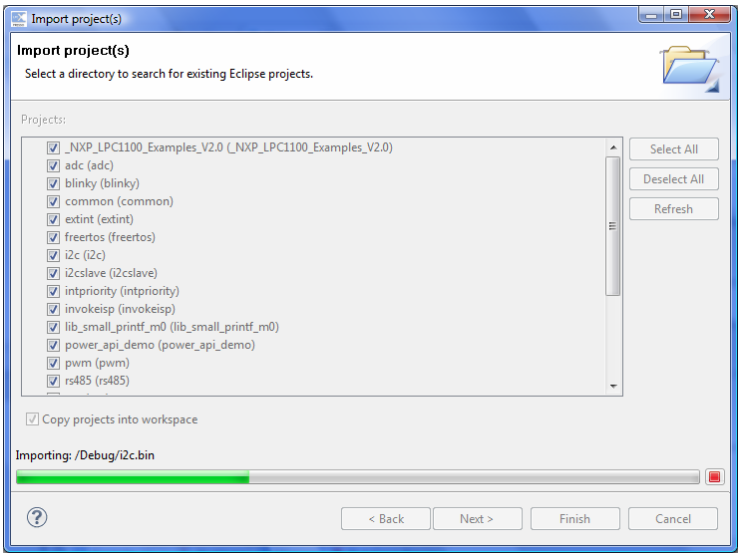

Figura C.6 Ventana para seleccionar proyectos existentes

5. Finalmente en el panel Quickstart damos clic en “Built” y luego en “Debug”. En LPCXpresso, cuando se empieza a depurar, el programa se descargará automáticamente a la tarjeta y es programado en la memoria flash.

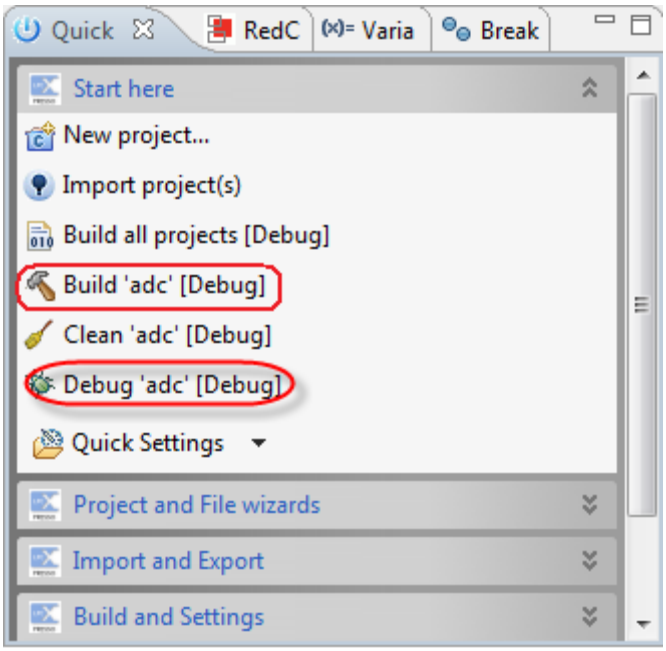

Figura C.5 Panel Quickstart del LPCXpresso IDE

6. Aquí se presenta la barra de herramientas para tener control sobre el código que se ejecuta.

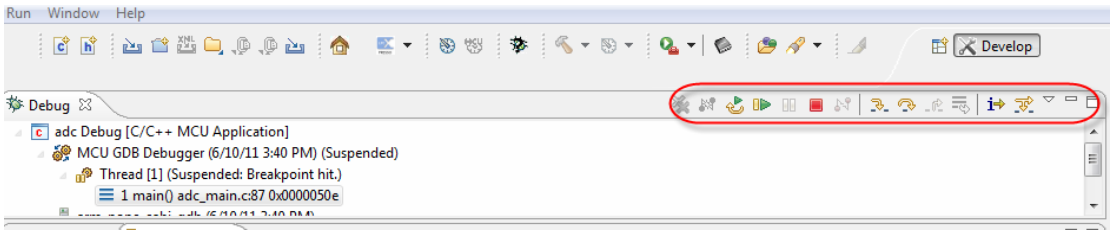

Figura C.6 Barra de herramientas de control de LPCXpresso IDE

ANEXO D

CÓDIGOS FUENTE PARA EJERCICIOS PREVIOS E IMPLEMENTACIÓN FINAL

EJERCICIOS PREVIOS

Ejercicio 1: Contador de 0 a 9 entre LPC - LPC

Programa principal del controlador Maestro

```
/*****************************************************************
 **********        MICROCONTROLADORES AVANZADOS         **********
 *****************************************************************
                     Comunicación Serial SPI
 *****************************************************************
                           Ejercicio # 1

 * Nombre: Contador de 0 a 9
 * Descripción:
   El maestro LPCxpresso envía el dato del esclavo LPCxpresso y  lo
muestra por el puerto 2 en un display 7 segmentos

 *****************************************************************
 ** Nombre del archivo: ssptest.c
 **
**   Project: NXP LPC17xx SSP example
 **   Description:
 **   This file contains SSP test modules, main entry, to test SSP APIs.
*****************************************************************/
#include<cr_section_macros.h>
#include<NXP/crp.h>

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

#include "LPC17xx.h"                        /* LPC13xx definitions */
#include "ssp.h"

/* Be careful with the port number and location number, because
some of the location may not exist in that port. */
#define PORT_NUM                    1
#define LOCATION_NUM          0
#define LED    LPC_GPIO2

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

//Tabla que contiene el codigo de los numeros de 0 a 9
staticconst uint8_t segmentLUT[10] =
{
               //PGFEDCBA
               (uint8_t) 0b00111111, // 0
               (uint8_t) 0b00000110, // 1
               (uint8_t) 0b01011011, // 2
               (uint8_t) 0b01001111, // 3
               (uint8_t) 0b01100110, // 4
               (uint8_t) 0b01101101, // 5
               (uint8_t) 0b01111101, // 6
               (uint8_t) 0b00000111, // 7
               (uint8_t) 0b01111111, // 8
               (uint8_t) 0b01100111, // 9
};
```

```
/*****************************************************************************
**   Main Function main ()
*****************************************************************************/
intmain (void)
{

uint32_t j, i=0, portnum = PORT_NUM;


/* SystemClockUpdate() updates the SystemFrequency variable */
SystemClockUpdate();
  LED -> FIODIR = 0xFF;
//seleccionamos que puerto SSP deseamos
if ( portnum == 0 )
        {SSP0Init();}                /* initialize SSP port */
elseif ( portnum == 1 )
        {SSP1Init();}

while ( 1 )
{
        for(j = 20000000; j > 0; j--); // retardo

                        LED->FIOCLR = 0xFF; //enceramos el puerto 2

                        //asignamos numero de la tabla a  source address
                        src_addr[0] = (uint8_t)segmentLUT[i];

                        //hacemos alto los pines para formar el nuemero
                        LED->FIOSET=src_addr[0];

                        i=i+1;
                        if (i==10) i=0;//repetimos el conteo

#if TX_RX_ONLY
/* For the inter-board communication, one board is set as
master transmit, the other is set to slave receive. */
#if SSP_SLAVE
                /* Slave receive */
                        SSPReceive( portnum, (uint8_t *)dest_addr, 1);

                        LED -> FIOCLR = 0xFF;//enceramos el puerto 2

                if (dest_addr[0]!= 0xFF)
                {
                        //hacemos alto los pines para formar el número
                        LED -> FIOSET =  dest_addr[0];
                  }
#else
/* Master transmit */
SSPSend( portnum, (uint8_t *)src_addr, 1);
#endif
#else
/* TX_RX_ONLY=0, it's either an internal loopback test
within SSP peripheral or communicate with a serial EEPROM. */
#if LOOPBACK_MODE
LoopbackTest( portnum, LOCATION_NUM );
#else
SEEPROMTest( portnum, LOCATION_NUM );
#endif                /* endif NOT LOOPBACK_MODE */
#endif                /* endif NOT TX_RX_ONLY */

}
return 0;
}
/*****************************************************************
                        End Of File
*****************************************************************/
```

Programa Principal del controlador Esclavo

```
/*****************************************************************
 **********        MICROCONTROLADORES AVANZADOS          **********
 *****************************************************************
                        Comunicación Serial SPI
 *****************************************************************
                            Ejercicio # 1

 * Nombre: Contador de 0 a 9
 * Descripción:
   El esclavo LPCxpresso recibe el dato del maestro LPCxpresso y  lo
muestra por el puerto 2 en un display 7 segmentos

 *****************************************************************
 ** Nombre del archivo: ssptest.c
 **
**   Project: NXP LPC17xx SSP example
 **   Description:
 **   This file contains SSP test modules, main entry, to test SSP APIs.
*****************************************************************/
#include<cr_section_macros.h>
#include<NXP/crp.h>

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

#include "LPC17xx.h"                        /* LPC13xx definitions */
#include "ssp.h"


/* Be careful with the port number and location number, because
some of the location may not exist in that port. */
#define PORT_NUM                     1
#define LOCATION_NUM          0
#define LED LPC_GPIO2

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


/*****************************************************************************
**   Main Function  main()
*****************************************************************************/
intmain (void)
{

uint32_t portnum = PORT_NUM;

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

  LED -> FIODIR = 0xFF; /*Define port D as output*/


while ( 1 )
  {
       if ( portnum == 0 )
              SSP0Init();                   /* initialize SSP0 port */
       elseif ( portnum == 1 )
              SSP1Init();                   /* initialize SSP1 port */

         dest_addr[0] = 0;

#if TX_RX_ONLY
```

```
 /* For the inter-board communication, one board is set as
master transmit, the other is set to slave receive. */
        #if SSP_SLAVE
        /* Slave receive */
                SSPReceive( portnum, (uint8_t *)dest_addr, 1);

                LED -> FIOCLR = 0xFF;//enceramos el puerto 2


        if (dest_addr[0]!= 0xFF)
          {
                //hacemos alto los pines para formar el numero (puerto 2)
                LED -> FIOSET =  dest_addr[0];
          }
        #else
        /* Master transmit */
        SSPSend( portnum, (uint8_t *)src_addr, SSP_BUFSIZE);
        #endif

#else
        /* TX_RX_ONLY=0, it's either an internal loopback test
        within SSP peripheral or communicate with a serial EEPROM. */
        #if LOOPBACK_MODE
        LoopbackTest( portnum, LOCATION_NUM );
        #else
        SEEPROMTest( portnum, LOCATION_NUM );
        #endif                  /* endif NOT LOOPBACK_MODE */
#endif  /* endif NOT TX_RX_ONLY */

}
return 0;
}

/*****************************************************************
                            End Of File
*****************************************************************/
```

Ejercicio2: Contador de 0 a 9 entre LPC – AVR

Programa principal del controlador Maestro

```
/*****************************************************************
 **********        MICROCONTROLADORES AVANZADOS         **********
 *****************************************************************
                      Comunicación Serial SPI
 *****************************************************************
                            Ejercicio # 2

 * Nombre: Contador de 0 a 9
 * Descripción:
   El maestro LPCxpresso envia el dato del esclavo LPCxpresso y  lo
muestra por el puerto 2 en un display 7 segmentos

 *****************************************************************
 ** Nombre del archivo: ssptest.c
 **
**   Project: NXP LPC17xx SSP example
 **   Description:
 **   This file contains SSP test modules, main entry, to test SSP APIs.
*****************************************************************/
/* Se realizo un cambio en el Registro Pre-escalador del reloj:
   SPSR =     0x2 por  SPSR = 0xf.
   Observacion de: Walter Orellana y Gian Banchon
*****************************************************************/

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

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

#include "LPC17xx.h"                        /* LPC13xx definitions */
#include "ssp.h"

/* Be careful with the port number and location number, because
some of the location may not exist in that port. */
#define PORT_NUM                    1
#define LOCATION_NUM          0
#define LED    LPC_GPIO2

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

//Tabla que contiene el codigo de los numeros de 0 a 9
staticconst uint8_t segmentLUT[10] =
{
               //PGFEDCBA
               (uint8_t) 0b00111111, // 0
               (uint8_t) 0b00000110, // 1
               (uint8_t) 0b01011011, // 2
               (uint8_t) 0b01001111, // 3
               (uint8_t) 0b01100110, // 4
               (uint8_t) 0b01101101, // 5
               (uint8_t) 0b01111101, // 6
               (uint8_t) 0b00000111, // 7
               (uint8_t) 0b01111111, // 8
               (uint8_t) 0b01100111, // 9
};
```

```
/*****************************************************************************
**   Main Function  main()
*****************************************************************************/
intmain (void)
{

uint32_t j, i=0, portnum = PORT_NUM;


  /* SystemClockUpdate() updates the SystemFrequency variable */
SystemClockUpdate();
  LED -> FIODIR = 0xFF;

  //seleccionamos que puerto SSP deseamos
if ( portnum == 0 )
        {SSP0Init();}                /* initialize SSP port */
elseif ( portnum == 1 )
        {SSP1Init();}

while ( 1 )
{
        for(j = 20000000; j > 0; j--); // retardo

                        LED->FIOCLR = 0xFF; //enceramos el puerto 2

                        //asignamos numero de la tabla a  source address
                        src_addr[0] = (uint8_t)segmentLUT[i];
                        //hacemos alto los pines para formar el numero (puerto 2)
                        LED->FIOSET=src_addr[0];

                        i=i+1;
                        if (i==10) i=0;//repetimos el conteo

#if TX_RX_ONLY
  /* For the inter-board communication, one board is set as
master transmit, the other is set to slave receive. */
#if SSP_SLAVE
                /* Slave receive */
                        SSPReceive( portnum, (uint8_t *)dest_addr, 1);

                        LED -> FIOCLR = 0xFF; //enceramos el puerto 2

                if (dest_addr[0]!= 0xFF)
                  {
                        //hacemos alto los pines para formar el numero (puerto 2)
                        LED -> FIOSET =  dest_addr[0];
                  }
#else
  /* Master transmit */
SSPSend( portnum, (uint8_t *)src_addr, 1);
#endif
#else
  /* TX_RX_ONLY=0, it's either an internal loopback test
within SSP peripheral or communicate with a serial EEPROM. */
#if LOOPBACK_MODE
LoopbackTest( portnum, LOCATION_NUM );
#else
SEEPROMTest( portnum, LOCATION_NUM );
#endif                /* endif NOT LOOPBACK_MODE */
#endif                /* endif NOT TX_RX_ONLY */

}
return 0;
}
/*****************************************************************
**                            End Of File
*****************************************************************/
```

Programa Principal del controlador Esclavo

```
/*****************************************************************
 **********        MICROCONTROLADORES AVANZADOS         **********
 *****************************************************************
                      Comunicación Serial SPI
 *****************************************************************
                            Ejercicio # 2
 * Nombre: Contador de 0 a 9
 * Descripción:
   El esclavoo recibe el dato del maestro LPC1769 y  lo
muestra por el display del AVR Butterfly.
 *****************************************************************
 ** Nombre del archivo: Contador_0a9_sclv.c
 *****************************************************************/
//contiene la definición de los registros y de sus respectivos bits,
#include <avr/io.h>
//contiene funciones para establecer períodos de retraso
#include <util/delay.h>
//contiene funciones para acceder datos almacenados en espacios de memoria de programa
#include <avr/pgmspace.h>
//contiene macros q permite escribir funciones gestoras d interrupción.
#include <avr/signal.h>
//habilita el uso de las funciones habilitadora global de interrupción
#include <avr/interrupt.h>

void SPI_slave_init(void);
unsigned char SPI_slave_receive(void);

int main(void)

{
        unsigned char rx_data;
        SPI_slave_init(); // inicializa SPI
        DDRD=0XFF;
        PORTD=0;

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

void SPI_slave_init(void)

{
DDRB=0x08; // MISO como salida
SPCR=(1<<SPE); // SPI enable, dispositivo Slave
}

unsigned char SPI_slave_receive(void)

{
while(!(SPSR & (1<<SPIF))); // espera mientras recibe el dato completo
return SPDR; // retorna dato recibido
}
```

Ejercicio 3: Contador de 0 a 9 entre AVR – LPC, mediante Joystick

Programa principal del controlador Maestro

```
/*****************************************************************
 **********        MICROCONTROLADORES AVANZADOS         *************
 *****************************************************************
                      Comunicación Serial SPI
 *****************************************************************
                            Ejercicio # 3

 * Nombre: Contador de 0 a 9
 * Descripción:
   El maestro recibe el dato del Joystick del AVR Butterfly y se lo
envía al esclavo.
 ****************************************************************
 ** Nombre del archivo: Contador_0a9.c
 ****************************************************************/
//contiene la definición de los registros y de sus respectivos bits,
#include <avr/io.h>
//contiene funciones para establecer períodos de retraso
#include <util/delay.h>
//contiene funciones para acceder datos almacenados en espacios de memoria de programa
#include <avr/pgmspace.h>
//contiene macros q permite escribir funciones gestoras d interrupción.
#include <avr/signal.h>
//habilita el uso de las funciones habilitadora global de interrupción
#include <avr/interrupt.h>

//Definir que pines contribuirán con las interrupciones
// en las variables MASCARA_PINB MASCARA_PIN
#define MASCARA_PINB ((1<<PINB7)|(1<<PINB6)|(1<<PINB4))
#define MASCARA_PINE ((1<<PINE3)|(1<<PINE2))

//Definir algunas constantes y variables globales que
//facilitarán la interpretación del código
#define ARRIBA 0
#define ABAJO 1
#define IZQUIERDA 2
#define DERECHA 3
#define CENTRO 4
#define NO_VALIDA 5
//según el diagrama de conexión del joystick
#define posicion_A 6 //ARRIBA
#define posicion_B 7 //ABAJO
#define posicion_C 2 //DERECHA
#define posicion_D 3 //IZQUIERDA
#define posicion_O 4 //CENTRO

#define VERDADERO 1
#define FALSO 0

volatile unsigned char SELECCION = 0;
volatile unsigned char SELECCION_VALIDA = 0;

unsigned char dec, decd;
```

```
// Tabla de conversión de binario a 7 segmentos

unsigned char numeros[] = {
0b00111111,
0b00000110,
0b01011011,
0b01001111,
0b01100110,
0b01101101,
0b01111101,
0b00000111,
0b01111111,
0b01100111,
};

//Funciones necesarias para utilizar el Joystick
void inicializar(void);
void manejar_interrupcion(void);
void obtener_seleccion(void);
void SPI_master_transmit(unsigned char cdata);


int main(void)
	{
	inicializar();
	sei();

	while(1)
		{
		SMCR = ((1<<SM1)|(1<<SE));
		}
		return 0;
	}
void inicializar(void)
	{

	DDRB=0x07;  //MOSI, SS y SCK como salida, PINB4.PINB6,PINB7como entrada
	PORTB = MASCARA_PINB; //habilitar PULL-UPs
	DDRE = 0;      //todos los pines del PUERTO E como entradas:
	PORTE = MASCARA_PINE; //habilitar PULL-UPs

	// SPI enable, dispositivo MASTER, Fosc/128
	SPCR=(1<<SPE)|(1<<MSTR)|(1<<SPR0)|(1<<SPR1);

	//Habilitar las interrupciones en los pines PINB7, PINB6 yPINB4
	PCMSK1 |= MASCARA_PINB;
	//Habilitar las interrupciones en los pines PINE3 y PINE2
	PCMSK0 |= MASCARA_PINE;

	//Borrar los indicadores de interrupción externa por cambio de nivel en pin
	EIFR = ((1<<PCIF1)|(1<<PCIF0));
	//Habilita interrupciones por cambio d nivel: puertos B (PCIE1) y E (PCIE0)
	EIMSK = ((1<<PCIE1)|(1<<PCIE0));

	DDRD = 0xFF;
	PORTD = 0x00;
	}
```

```
void manejar_interrupcion(void)
	{

	unsigned char joystick;
	unsigned char seleccion;

	//Identificar cual pin del puerto B presenta cambio de nivel
	joystick = ((~PINB) & MASCARA_PINB);
	//Identificar cual pin del puerto E presenta cambio de nivel
	joystick |= ((~PINE) & MASCARA_PINE);

	if((joystick & (1<<posicion_A)))
		seleccion = ARRIBA;
		else if((joystick & (1<<posicion_B)))
			seleccion = ABAJO;
		else if((joystick & (1<<posicion_C)))
			seleccion = DERECHA;
		else if((joystick & (1<<posicion_D)))
				seleccion = IZQUIERDA;
		else if((joystick & (1<<posicion_O)))
			seleccion = CENTRO;
		else seleccion = NO_VALIDA;

	if(seleccion != NO_VALIDA)
	{
		if(!SELECCION_VALIDA)
		{
			SELECCION = seleccion;
			SELECCION_VALIDA = VERDADERO;
		}
	}

	//Borrar los indicadores de interrupción correspondientes al cambio de nivel
	EIFR = ((1<<PCIF1)|(1<<PCIF0));
	obtener_seleccion();

	}

void obtener_seleccion(void)

	{
	unsigned char seleccion;

	cli(); //Deshabilitar globalmente todas las interrupciones

	if(SELECCION_VALIDA)
	{
		seleccion = SELECCION;
		SELECCION_VALIDA = FALSO;
	}
	else seleccion = NO_VALIDA;
		if(seleccion != NO_VALIDA)
		{
			switch(seleccion)
		{
				case ARRIBA:
				dec++;
				if(dec==0x0A) {dec=0;}
```

```
                                decd=numeros[dec]; //conversión binario a 7segmentos
                                SPI_master_transmit(decd); // transmite dato
                                PORTD=decd; //Muestra numero en el PUERTO D
                                break;

                                case ABAJO:
                                dec--;
                                if(dec==0xFF) {dec=9;}
                                decd=numeros[dec]; //conversión binario a 7 segmentos
                                SPI_master_transmit(decd); // transmite dato
                                PORTD=decd; //Muestra numero en el PUERTO D
                                break;

                                case IZQUIERDA:
                                dec=dec-2;
                                if(dec==0xFF) {dec=9;}
                                if(dec==0xFE) {dec=8;}
                                decd=numeros[dec]; //conversión binario a 7segmentos
                                SPI_master_transmit(decd); // transmite dato
                                PORTD=decd; //Muestra numero en el PUERTO D
                                break;

                                case DERECHA:
                                dec=dec+2;
                                if(dec==0x0A) {dec=0;}
                                if(dec==0x0B) {dec=1;}
                                decd=numeros[dec]; //conversión binario a 7segmentos
                                SPI_master_transmit(decd); // transmite dato
                                PORTD=decd; //Muestra numero en el PUERTO D
                                break;

                                case CENTRO:
                                dec=0;
                                decd=numeros[dec]; //conversión binario a 7segmentos
                                SPI_master_transmit(decd); // transmite dato
                                PORTD=decd; //Muestra numero en el PUERTO D
                                default:
                                break;
                        }
                }
                sei();//Habilitar globalmente las interrupciones
                }

//Las siguientes funciones son las gestoras de interrupciones
SIGNAL(SIG_PIN_CHANGE0)
                {
                manejar_interrupcion();
                }
SIGNAL(SIG_PIN_CHANGE1)
                {
                manejar_interrupcion();
                }
//Funcion que trasmite el dato al esclavo
void SPI_master_transmit(unsigned char cdata)
                {
                SPDR=cdata; // coloca dato a enviar en el registro SPDR
                while(!(SPSR & (1<<SPIF))); // espera mientras se completa la transmisión
                }
```

Programa Principal del controlador Esclavo

```
/*****************************************************************
 **********        MICROCONTROLADORES AVANZADOS          **********
 *****************************************************************
                      Comunicación Serial SPI
 *****************************************************************
                           Ejercicio # 3

 * Nombre: Contador de 0 a 9
 * Descripción:
   El esclavo LPCxpresso recibe el dato del maestro LPCxpresso y  lo
muestra por el puerto 2 en un display 7 segmentos

 *****************************************************************
 ** Nombre del archivo: ssptest.c
 **
**   Project: NXP LPC17xx SSP example
 **   Description:
 **   This file contains SSP test modules, main entry, to test SSP APIs.
*****************************************************************/

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

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

#include "LPC17xx.h"                        /* LPC13xx definitions */
#include "ssp.h"


/* Be careful with the port number and location number, because
some of the location may not exist in that port. */
#define PORT_NUM                     1
#define LOCATION_NUM          0
#define LED LPC_GPIO2

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




/*****************************************************************
**   Main Function  main()
*****************************************************************/
intmain (void)
{

uint32_t portnum = PORT_NUM;

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

  LED -> FIODIR = 0xFF; /*Define port D as output*/


while ( 1 )
  {
       if ( portnum == 0 )
              SSP0Init();                   /* initialize SSP0 port */
```

```
        elseif ( portnum == 1 )
                SSP1Init();                   /* initialize SSP1 port */

          dest_addr[0] = 0;

#if TX_RX_ONLY

        /* For the inter-board communication, one board is set as
        master transmit, the other is set to slave receive. */

        #ifSSP_SLAVE
            /* Slave receive */
                SSPReceive( portnum, (uint8_t *)dest_addr, 1);

                LED -> FIOCLR = 0xFF; //enceramos el puerto 2
        if (dest_addr[0]!= 0xFF)
          {
                //hacemos alto los pines para formar el numero (puerto 2)
                LED -> FIOSET =  dest_addr[0];
          }
        #else
          /* Master transmit */
        SSPSend( portnum, (uint8_t *)src_addr, SSP_BUFSIZE);
        #endif

#else
        /* TX_RX_ONLY=0, it's either an internal loopback test
        within SSP peripheral or communicate with a serial EEPROM. */
        #if LOOPBACK_MODE

        LoopbackTest( portnum, LOCATION_NUM );

        #else

        SEEPROMTest( portnum, LOCATION_NUM );

        #endif                /* endif NOT LOOPBACK_MODE */
#endif  /* endif NOT TX_RX_ONLY */

  }
return 0;
}

/*****************************************************************************
**                            End Of File
******************************************************************************/
```

PROYECTO FINAL

Control de motor BLDC entre AVR – LPC, mediante Joystick

Programa principal del controlador Maestro

```
/*****************************************************************
 **********        MICROCONTROLADORES AVANZADOS         **********
 *****************************************************************
                                Comunicación Serial SPI
 *****************************************************************
                                    Proyecto

 * Nombre: Control BLDC mediante Joystick
 * Descripción:
   El maestro recibe orden del Joystick del AVR Butterfly y se lo envía al esclavo.
 *****************************************************************
 ** Nombre del archivo: Master_Control_BLDC.c
 *****************************************************************/

//contiene la definición de los registros y de sus respectivos bits,
#include <avr/io.h>
//contiene funciones para establecer períodos de retraso
#include <util/delay.h>
//contiene funciones para acceder datos almacenados en espacios de memoria de programa
#include <avr/pgmspace.h>
//contiene macros q permite escribir funciones gestoras d interrupción.
#include <avr/signal.h>
//habilita el uso de las funciones habilitadora global de interrupción
#include <avr/interrupt.h>
//Definir que pines contribuirán con las interrupciones
// en las variables MASCARA_PINB MASCARA_PIN
//Definir que pines contribuirán con las interrupciones
// en las variables MASCARA_PINB MASCARA_PIN
#define MASCARA_PINB ((1<<PINB7)|(1<<PINB6)|(1<<PINB4))
#define MASCARA_PINE ((1<<PINE3)|(1<<PINE2))
//Definir algunas constantes y variables globales que
//facilitarán la interpretación del código
#define ARRIBA 0
#define ABAJO 1
#define IZQUIERDA 2
#define DERECHA 3
#define CENTRO 4
#define NO_VALIDA 5
//según el diagrama de conexión del joystick
#define posicion_A 6 //ARRIBA
#define posicion_B 7 //ABAJO
#define posicion_C 2 //DERECHA
#define posicion_D 3 //IZQUIERDA
#define posicion_O 4 //CENTRO

#define VERDADERO 1
#define FALSO 0
```

```
volatile unsigned char SELECCION = 0;
volatile unsigned char SELECCION_VALIDA = 0;

unsigned char dec, decd;

#define pLCDREG ((unsigned char *)(0xEC))
#define TAMANIO_DEL_REGISTRO_LCD 20
#define NUMERO_MAXIMO_DE_CARACTERES 36

char LCD_Data[TAMANIO_DEL_REGISTRO_LCD];
char memo_temp_texto[NUMERO_MAXIMO_DE_CARACTERES];
unsigned char ESCRITURA_DE_CADENA_HABILITADO = 0;
unsigned char LCD_INT_contador = 0;

//Funciones necesarias para utilizar el LCD
void inicializar_LCD(void);
void escribir_caracter_en_LCD(char , char );
void borrar_LCD(void);
void actualizar_LCD(void);

// Tabla del Dato a enviar según la orden del Joystick
unsigned char ordenes[] = {
0b00000001,//Derecha, Centro
0b00000010,//Izquierda
0b00000100,//Abajo
0b00001000,//Arriba
0b00000000,
};

//Funciones necesarias para utilizar el Joystick
void inicializar(void);
void manejar_interrupcion(void);
void obtener_seleccion(void);
void SPI_master_transmit(unsigned char cdata);

unsigned int tabla_de_caracteres_LCD[] PROGMEM =
{
0x0A51, // '*' (?)
0x2A80, // '+'
0x0000, // ',' (Sin definir)
0x0A00, // '-'
0x0A51, // '.' Signo de grados
0x0000, // '/' (Sin definir)
0x5559, // '0'
0x0118, // '1'
0x1e11, // '2
0x1b11, // '3
0x0b50, // '4
0x1b41, // '5
0x1f41, // '6
0x0111, // '7
0x1f51, // '8
0x1b51, // '9'
0x0000, // ':' (Sin definir)
0x0000, // ';' (Sin definir)
0x0000, // '<' (Sin definir)
0x0000, // '=' (Sin definir)
0x0000, // '>' (Sin definir)
```

```
0x0000, // '?' (Sin definir)
0x0000, // '@' (Sin definir)
0x0f51, // 'A' (+ 'a')
0x3991, // 'B' (+ 'b')
0x1441, // 'C' (+ 'c')
0x3191, // 'D' (+ 'd')
0x1e41, // 'E' (+ 'e')
0x0e41, // 'F' (+ 'f')
0x1d41, // 'G' (+ 'g')
0x0f50, // 'H' (+ 'h')
0x2080, // 'I' (+ 'i')
0x1510, // 'J' (+ 'j')
0x8648, // 'K' (+ 'k')
0x1440, // 'L' (+ 'l')
0x0578, // 'M' (+ 'm')
0x8570, // 'N' (+ 'n')
0x1551, // 'O' (+ 'o')
0x0e51, // 'P' (+ 'p')
0x9551, // 'Q' (+ 'q')
0x8e51, // 'R' (+ 'r')
0x9021, // 'S' (+ 's')
0x2081, // 'T' (+ 't')
0x1550, // 'U' (+ 'u')
0x4448, // 'V' (+ 'v')
0xc550, // 'W' (+ 'w')
0xc028, // 'X' (+ 'x')
0x2028, // 'Y' (+ 'y')
0x5009, // 'Z' (+ 'z')
0x0000, // '[' (Sin definir)
0x0000, // '\' (Sin definir)
0x0000, // ']' (Sin definir)
0x0000, // '^' (Sin definir)
0x0000, // '_'
};

int main(void)
        {

        inicializar();
        inicializar_LCD();
        sei();

        while(1)
                {
                SMCR = ((1<<SM1)|(1<<SE));
                }
                return 0;
        }

void inicializar(void)
        {

        DDRB=0x07;  //MOSI, SS y SCK como salida, PINB4.PINB6, PINB7 como entrada
        PORTB = MASCARA_PINB; //habilitar PULL-UPs
        DDRE = 0;       //todos los pines del PUERTO E como entradas:
        PORTE = MASCARA_PINE; //habilitar PULL-UPs

        // SPI enable, dispositivo MASTER, Fosc/128
        SPCR=(1<<SPE)|(1<<MSTR)|(1<<SPR0)|(1<<SPR1);
```

```
        //Habilitar las interrupciones en los pines PINB7, PINB6 y PINB4
        PCMSK1 |= MASCARA_PINB;
        //Habilitar las interrupciones en los pines PINE3 y PINE2
        PCMSK0 |= MASCARA_PINE;

        //Borrar los indicadores de interrupción externa por cambio de nivel en pin
        EIFR = ((1<<PCIF1)|(1<<PCIF0));
        //Habilita las interrupciones por cambio d nivel: puertos B (PCIE1) y E
(PCIE0)
        EIMSK = ((1<<PCIE1)|(1<<PCIE0));

        DDRD = 0xFF;
        PORTD = 0x00;
        }

void manejar_interrupcion(void)
        {
        unsigned char joystick;
        unsigned char seleccion;

        //Identificar cual pin del puerto B presenta cambio de nivel
        joystick = ((~PINB) & MASCARA_PINB);
        //Identificar cual pin del puerto E presenta cambio de nivel
        joystick |= ((~PINE) & MASCARA_PINE);

        if((joystick & (1<<posicion_A)))
                seleccion = ARRIBA;
                else if((joystick & (1<<posicion_B)))
                        seleccion = ABAJO;
                else if((joystick & (1<<posicion_C)))
                        seleccion = DERECHA;
                else if((joystick & (1<<posicion_D)))
                                seleccion = IZQUIERDA;
                else if((joystick & (1<<posicion_O)))
                        seleccion = CENTRO;
                else seleccion = NO_VALIDA;

        if(seleccion != NO_VALIDA)
        {
                if(!SELECCION_VALIDA)
                {
                        SELECCION = seleccion;
                        SELECCION_VALIDA = VERDADERO;
                }
        }
        //Borrar los indicadores de interrupción correspondientes al cambio de nivel
        EIFR = ((1<<PCIF1)|(1<<PCIF0));
        obtener_seleccion();

        }



void obtener_seleccion(void)
        {
        unsigned char seleccion;

        cli(); //Deshabilitar globalmente todas las interrupciones
```

```
if(SELECCION_VALIDA)
{
        seleccion = SELECCION;
        SELECCION_VALIDA = FALSO;
}
else seleccion = NO_VALIDA;

        if(seleccion != NO_VALIDA)
        {
                switch(seleccion)
        {
```

```
case ARRIBA:
borrar_LCD();
actualizar_LCD();
escribir_caracter_en_LCD('S',0);
escribir_caracter_en_LCD('P',1);
escribir_caracter_en_LCD('E',2);
escribir_caracter_en_LCD('E',3);
escribir_caracter_en_LCD('D',4);
escribir_caracter_en_LCD('+',5);
actualizar_LCD();
dec=3;
decd=ordenes[dec]; //conversión binario a 7segmentos
SPI_master_transmit(decd); // transmite dato
PORTD=decd; //Muestra numero en el PUERTO D
break;
```

4

```
case ABAJO:
borrar_LCD();
actualizar_LCD();
escribir_caracter_en_LCD('S',0);
escribir_caracter_en_LCD('P',1);
escribir_caracter_en_LCD('E',2);
escribir_caracter_en_LCD('E',3);
escribir_caracter_en_LCD('D',4);
escribir_caracter_en_LCD('-',5);
actualizar_LCD();
dec=2;
decd=ordenes[dec]; //conversión binario a 7segmentos
SPI_master_transmit(decd); //transmite dato
PORTD=decd; //Muestra numero en el PUERTO D
break;
```

5

```
case IZQUIERDA:
borrar_LCD();
actualizar_LCD();
escribir_caracter_en_LCD('I',0);
escribir_caracter_en_LCD('N',1);
escribir_caracter_en_LCD('V',2);
escribir_caracter_en_LCD('E',3);
escribir_caracter_en_LCD('R',4);
escribir_caracter_en_LCD('T',5);
actualizar_LCD();
dec=1;
decd=ordenes[dec]; //conversión binario a 7segmentos
SPI_master_transmit(decd); // transmite dato
PORTD=decd; //Muestra numero en el PUERTO D
break;
```

6

```
                case DERECHA:
                borrar_LCD();
                actualizar_LCD();
                escribir_caracter_en_LCD('O',0);
                escribir_caracter_en_LCD('N',1);
                escribir_caracter_en_LCD('-',2);
                escribir_caracter_en_LCD('O',3);
                escribir_caracter_en_LCD('F',4);
                escribir_caracter_en_LCD('F',5);
                actualizar_LCD();
                dec=0;
                decd=ordenes[dec]; //conversión binario a 7segmentos
                SPI_master_transmit(decd); // transmite dato
                PORTD=decd; //Muestra numero en el PUERTO D
                break;
```

7

```
                case CENTRO:
                borrar_LCD();
                actualizar_LCD();
                escribir_caracter_en_LCD('O',0);
                escribir_caracter_en_LCD('N',1);
                escribir_caracter_en_LCD('-',2);
                escribir_caracter_en_LCD('O',3);
                escribir_caracter_en_LCD('F',4);
                escribir_caracter_en_LCD('F',5);
                actualizar_LCD();
                dec=0;
                decd=ordenes[dec]; //conversión binario a 7segmentos
                SPI_master_transmit(decd); // transmite dato
                PORTD=decd; //Muestra numero en el PUERTO D
                default:
                break;
```

8

```
                }
        }
        sei();//Habilitar globalmente las interrupciones
        }

//Las siguientes funciones son las gestoras de interrupciones
SIGNAL(SIG_PIN_CHANGE0)
        {
        manejar_interrupcion();
        }

SIGNAL(SIG_PIN_CHANGE1)
        {
        manejar_interrupcion();
        }

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

void inicializar_LCD(void)
        {
        borrar_LCD();
        //Habilitar la interfaz LCD y configurar la forma de onda de baja potencia
```

```
        LCDCRA = (1<<LCDEN) | (1<<LCDAB);
        //Fijar el tiempo de control al 50% del CLK_LCD_PS y el nivel de contraste al
        100%:
        LCDCCR = (1<<LCDDC2)|(1<<LCDDC1)|(1<<LCDDC0)|(1<<LCDCC3)|(1<<LCDCC2)
        |(1<<LCDCC1)|(1<<LCDCC0);
        ASSR = (1<<AS2);
        //Seleccionar N=16 para determinar la velocidad d trama d 64 Hz
        LCDFRR = (0<<LCDPS0) | (1<<LCDCD1)|(1<<LCDCD0);
        //Seleccionar fuente de reloj externa asincrónica de 32768Hz
        LCDCRB = (1<<LCDCS)|(1<<LCDMUX1)|(1<<LCDMUX0)|(1<<LCDPM2)|(1<<LCDPM1)
        |(1<<LCDPM0);
        //Habilitar interrupción por inicio de trama LCD:
        LCDCRA |= (1<<LCDIE);
        }

void escribir_caracter_en_LCD(char c, char posicion)
        {
        unsigned int seg = 0x0000;
        char mascara, nibble;
        char *ptr;
        char i;
                if (posicion > 5) return;//Abortar si la posición no escorrecta

                //Verificar que el caracter sea un caracter alfanumérico
                if ((c >= '*') && (c <= 'z'))
                        {
                        //Si el caracter ‘c’ es una letra, convertirlo amayúsculas
                        if (c >= 'a') c &= ~0x20;
                                c -= '*';
                        seg = (unsigned int)
                pgm_read_word(&tabla_de_caracteres_LCD[(uint8_t)c]);
                        }
                if (posicion & 0x01) mascara = 0x0F;
                else mascara = 0xF0;
                        ptr = LCD_Data + (posicion >> 1);
                        for (i = 0; i < 4; i++)
                                {
                                nibble = seg & 0x000F;
                                seg>>= 4;
                                if (posicion & 0x01) nibble <<= 4;
                                *ptr = (*ptr & mascara) | nibble;
                                ptr += 5;
                                }
        }

void borrar_LCD(void)
        {
        unsigned char i=0;
        //Borrar el contenido de la matriz global memo_temp_texto[ ]
        for( i=0;i<NUMERO_MAXIMO_DE_CARACTERES;i++)
                memo_temp_texto[i]='\0';
        for (i = 0; i < TAMANIO_DEL_REGISTRO_LCD; i++)
                {
                //Borrar la pantalla borrando la memoria LCD
                *(pLCDREG + i) = 0x00;
                //Borrar el buffer de datos LCD
                *(LCD_Data+i) = 0x00;
                }
        //Actualizar el LCD para así borrar lo que haya en la pantalla
```

```
        actualizar_LCD();
        }

void actualizar_LCD(void)
        {
        ESCRITURA_DE_CADENA_HABILITADO = 0;
        for (char i = 0; i < TAMANIO_DEL_REGISTRO_LCD; i++)
                *(pLCDREG + i) = *(LCD_Data+i);
        }
//La función SIGNAL(SIG_LCD) es la gestora de la interrupción
//por el evento de inicio de una nueva trama LCD.
SIGNAL(SIG_LCD)
        {
        unsigned char letra=0;
        unsigned char i=0;
                if (ESCRITURA_DE_CADENA_HABILITADO==1)
                {
                for(i=0;(i<6);i++)
                {
                        if(!(memo_temp_texto[i+LCD_INT_contador]=='\0'))
                        {
                        letra = memo_temp_texto[i+LCD_INT_contador];
                        escribir_caracter_en_LCD(letra,i);
                        }
                        else
                        {
                        escribir_caracter_en_LCD(' ',i);
                        }
                        _delay_loop_2(20000);
                }
                if(LCD_INT_contador<NUMERO_MAXIMO_DE_CARACTERES)
                        LCD_INT_contador++;
                else
                        {
                        LCD_INT_contador=0;
                        ESCRITURA_DE_CADENA_HABILITADO = 0;
                        }
                }
                for (char i = 0; i < TAMANIO_DEL_REGISTRO_LCD; i++)
                        *(pLCDREG + i) = *(LCD_Data+i);
        }
```

Programa principal del controlador Esclavo

```
/*****************************************************************
 **********        MICROCONTROLADORES AVANZADOS          **********
 *****************************************************************
                      Comunicación Serial SPI
 *****************************************************************
                           Proyecto

 * Nombre: Control BLDC ESCLAVO
   El esclavo LPCxpresso recibe el dato del maestro LPCxpresso y  lo
muestra por el puerto 2

 *****************************************************************
 ** Nombre del archivo: ssptest.c
**
**   Project: NXP LPC17xx SSP example
 **   Description:
 **   This file contains SSP test modules, main entry, to test SSP APIs.
*****************************************************************/

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

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

#include "LPC17xx.h"                        /* LPC13xx definitions */
#include "ssp.h"

/* Be careful with the port number and location number, because
some of the location may not exist in that port. */
#define PORT_NUM                    1
#define LOCATION_NUM          0
#define LED LPC_GPIO2//Definimos el Puerto 2 como LED

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



/*****************************************************************
**   Main Function  main()
*****************************************************************/
intmain (void)
{
uint32_t portnum = PORT_NUM;
uint32_t j = 0;//Declaramos un contador j

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

  LED -> FIODIR = 0xFF; /*Define port D as output*/

while ( 1 )
  {
       if ( portnum == 0 )
              SSP0Init();                   /* initialize SSP0 port */
       elseif ( portnum == 1 )
              SSP1Init();                   /* initialize SSP1 port */

         dest_addr[0] = 0;
```

```
#if TX_RX_ONLY

 /* For the inter-board communication, one board is set as
master transmit, the other is set to slave receive. */
        #if SSP_SLAVE
        /* Slave receive */
                SSPReceive( portnum, (uint8_t *)dest_addr, 1);

                LED -> FIOSET = 0xFF; //hacemos alto todos los pines del puerto 2
        if (dest_addr[0]!= 0xFF)
          {
                //hacemos bajo el pin correspondientea la orden enviada por el maestro
                LED -> FIOCLR =  dest_addr[0];
                for(j = 2000000; j > 0; j--);//retardo
                LED -> FIOSET = 0xFF;//hacemos alto todos los pines del puerto 2
        }
        #else
        /* Master transmit */
        SSPSend( portnum, (uint8_t *)src_addr, SSP_BUFSIZE);
        #endif

#else
        /* TX_RX_ONLY=0, it's either an internal loopback test
        within SSP peripheral or communicate with a serial EEPROM. */
        #if LOOPBACK_MODE
        LoopbackTest( portnum, LOCATION_NUM );
        #else
        SEEPROMTest( portnum, LOCATION_NUM );
        #endif                  /* endif NOT LOOPBACK_MODE */
#endif  /* endif NOT TX_RX_ONLY */

  }
return 0;
}
/******************************************************************************
**                            End Of File
******************************************************************************/
```

12

ANEXO E

DIAGRAMA DE CONEXIONES PARA EJERCICIOS PREVIOS E IMPLEMENTACIÓN FINAL

Ejercicio 1: Diagrama de conexiones entre LPC – LPC

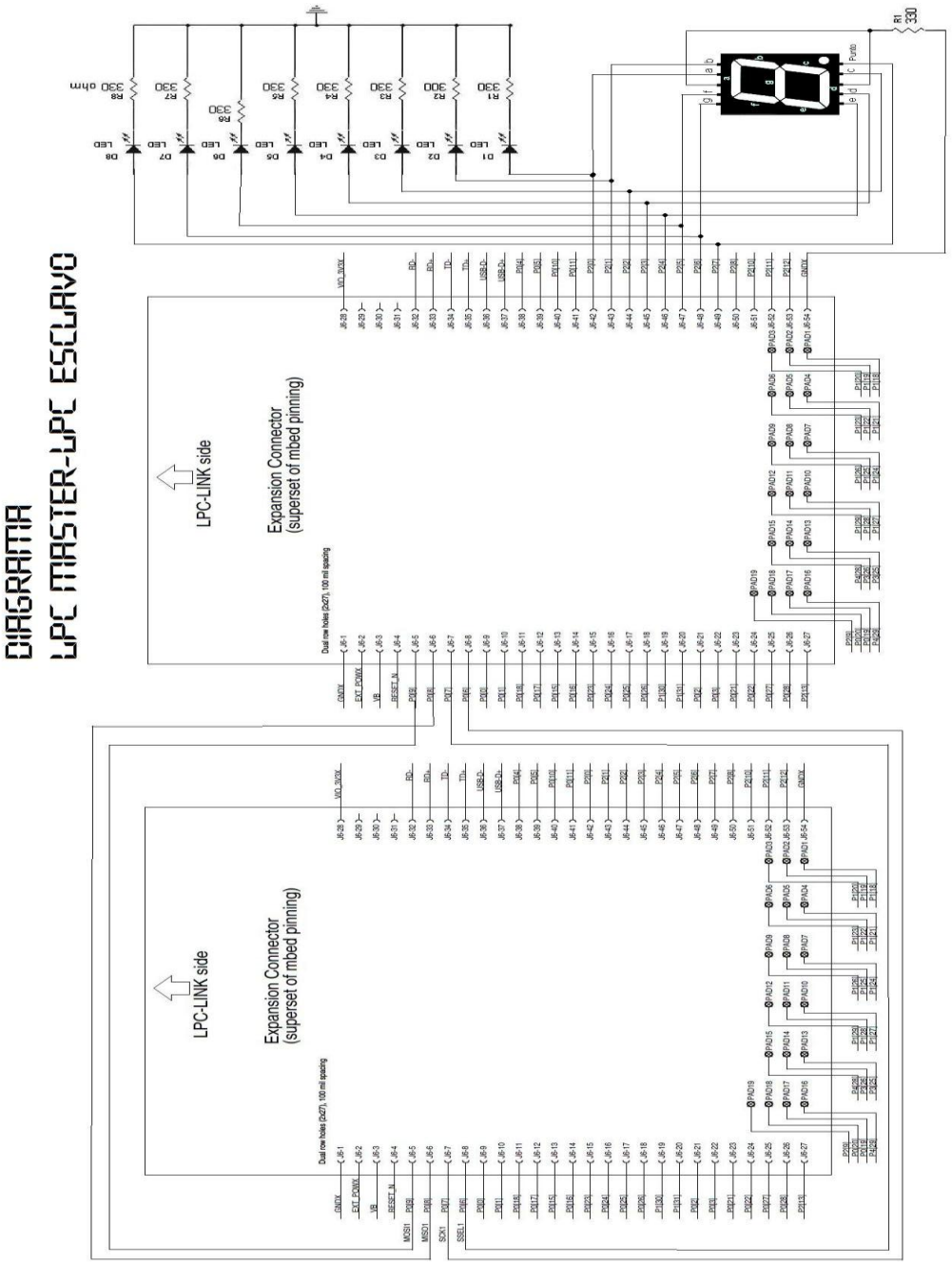

Figura E.1 Diagrama de conexiones entre LPC – LPC

Ejercicio 2: Contador de 0 a 9 entre LPC – AVR

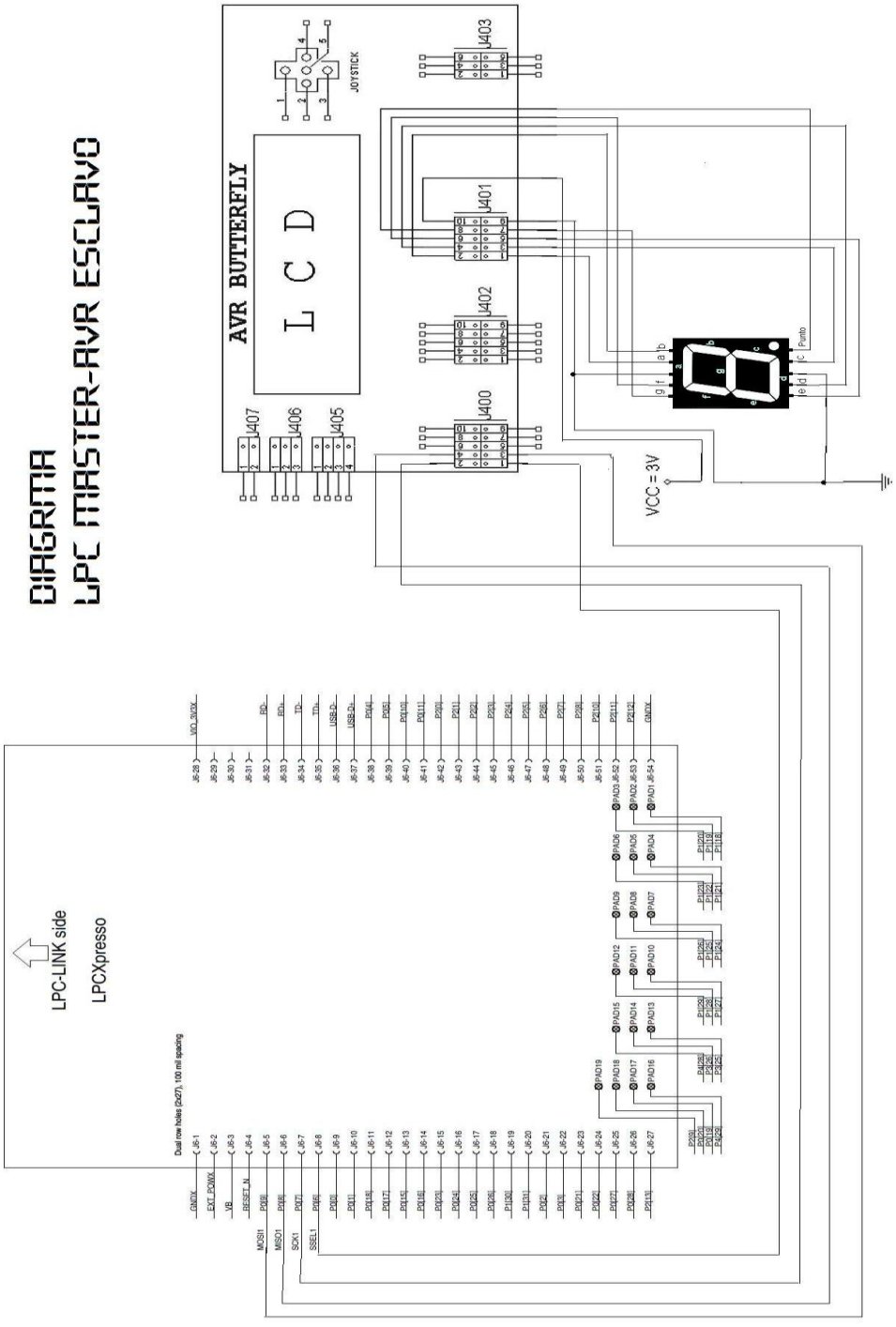

Figura E.2 Diagrama de conexiones entre LPC – AVR

Ejercicio 3: Contador de 0 a 9 entre AVR – LPC, mediante Joystick

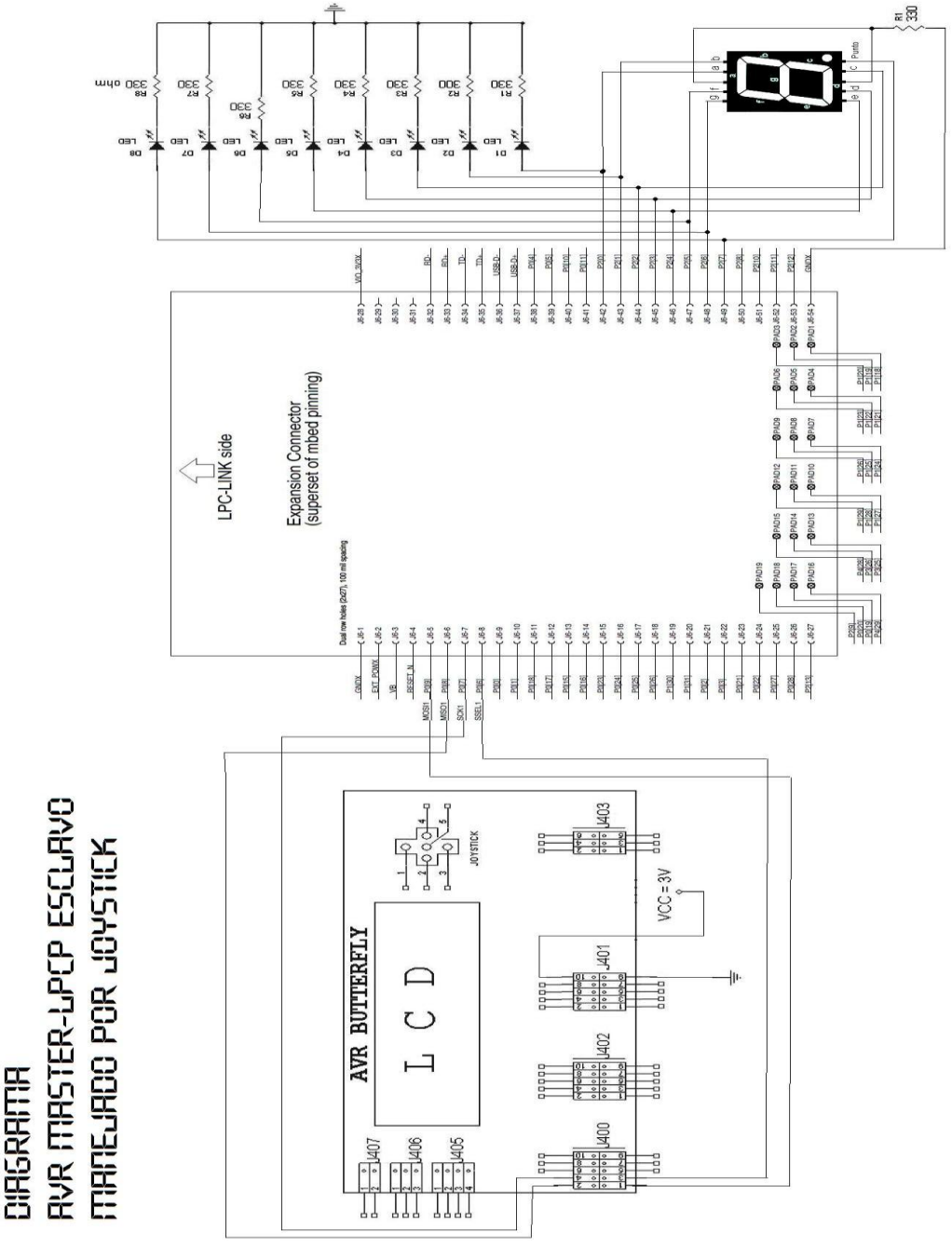

Figura E.3 Diagrama de conexiones entre AVR – LPC

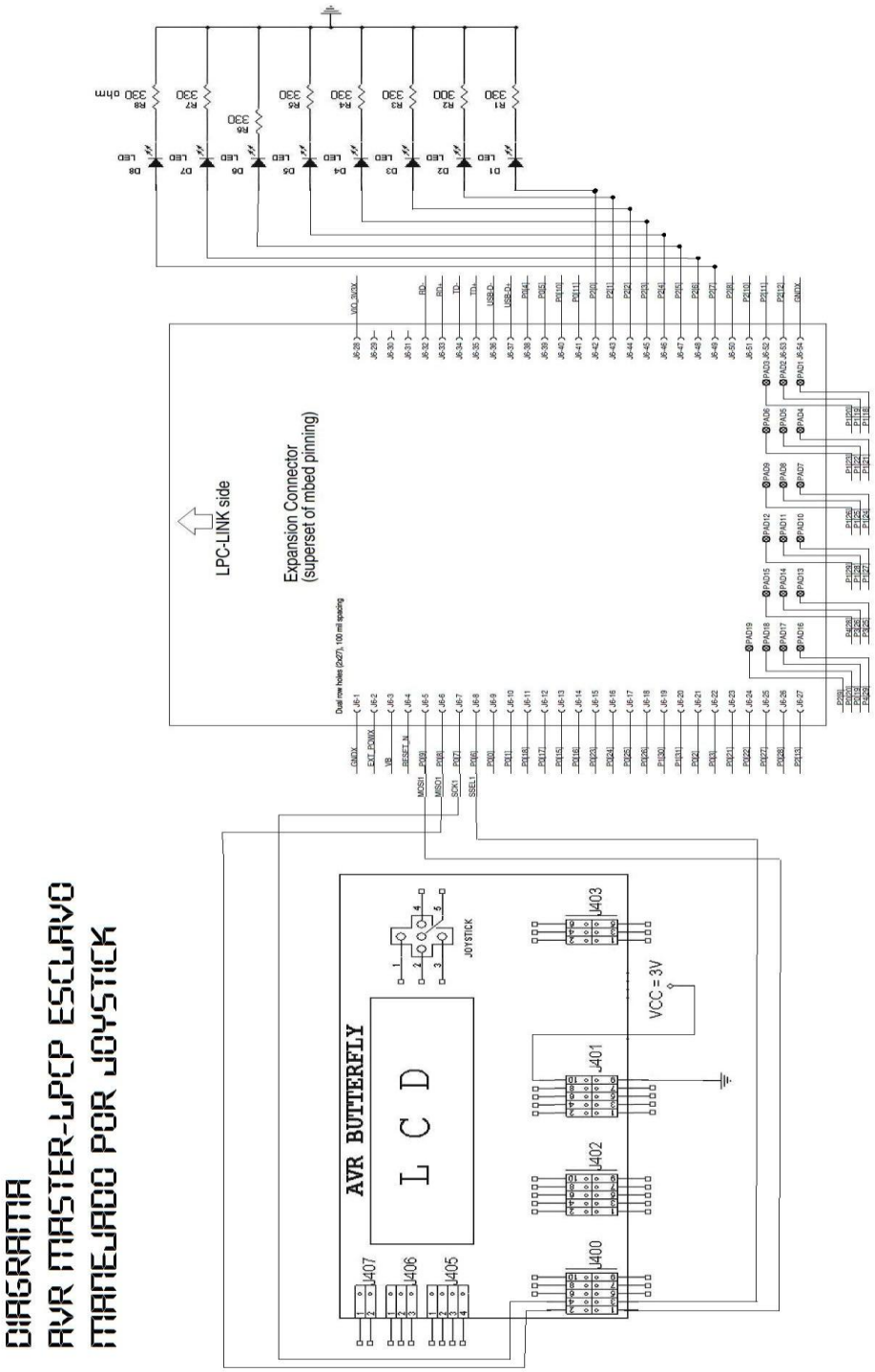

Figura E.4 Diagrama de conexiones para el controlador de motor BLDC

BIBLIOGRAFÍA

[1] Nobelprize.org, The History of the Integrated Circuit. (17/03/12) Página Internet: http://www.nobelprize.org/educational/physics/integrated_circuit/history/index.html

[2] Eric López Pérez, Ingeniería en Microcontroladores. (09/03/12) Página Internet: http://www.i-micro.com/pdf/articulos/spi.pdf

[3] User Manual LPC1769. (23/02/12) Página Internet: http://www.nxp.com/documents/user_manual/UM10360.pdf

[4] AVR Butterfly- Atmel Corporation. (25/03/12) Página Internet: http://www.ddrservice.net/files/Microcontrollers/atmel/atmega169.pdf

[5] Juanpere Tolrá, Roger, Técnicas de control para motores Brushless, Comparativa entre conmutación Trapezoidal, conmutación Sinusoidal y Control Vectorial, Universidad de Cataluña, Barcelona, España. (10/03/12)

[6] ATMEL corporation. (23/03/12) Página Internet: http://www.atmel.com/Images/doc4271.pdf

[7]NXP semiconductors. (04/04/12) Página Internet: http://www.nxp.com/documents/data_sheet/LPC1769_68_67_66_65_64_63.pdf

[8] Richard Leonel Guerrero Jumbo, Kit de desarrollo AVR Butterfly, desarrollo de guía de prácticas de laboratorio y tutoriales. (18/06/12)

[9] Software LPCXpresso 4, ejemplo. (17/04/12) Dirección en PC: C:\nxp\LPCXpresso_4.1.5_219\lpcxpresso\Examples\NXP\LPC1000\LPC17xx\NXP_LPCXpresso1769_MCB1700_2011-02-11.zip