ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

"ALMACENAMIENTO DE DATOS DE VIBRACIONES DE MOTOR BLDC PARA GRAFICACIÓN Y ANÁLISIS EN DISPLAYS DISPONIBLES EN TARJETA AVR BUTTERFLY Y EN TARJETA CONTROLADORA LPCXPRESSO"

TESINA DE SEMINARIO

Previa a la obtención del Título de:

INGENIERO EN ELECTRICIDAD
ESPECIALIZACIÓN ELECTRÓNICA Y AUTOMATIZACIÓN INDUSTRIAL

Presentado por:

FERNANDO ENRIQUE ORBEA GUERRERO
LUIS ALBERTO PATIÑO SÁNCHEZ

Guayaquil-Ecuador
2012

AGRADECIMIENTO

A Dios en primer lugar por darnos su bendición cada día de nuestras vidas, agradecemos a cada una de las personas que aportaron a nuestra formación humana, profesional y espiritual. Ayudándonos así a sobrellevar y pasar cualquier obstáculo que se nos ha presentado, durante todo este proceso.

A nuestros padres y familiares por su apoyo incondicional, a nuestros amigos por ser partícipes de nuestra trayectoria universitaria y de nuestra vida personal.

DEDICATORIA

A mi familia, maestros y amigos.

Fernando Orbea Guerrero

A mis padres por su apoyo incondicional en la formación de mi vida, a mi hermano por siempre guiarme y aconsejarme en mis decisiones, a mi abuelita por darme unos cuantos consejos sabios de su vida que me han ayudado en mi formación humana, a mis amigos por ayudarme y ser muy buenos amigos.

Luis A. Patiño Sánchez

TRIBUNAL DE SUSTENTACIÓN

_____

MSC. CARLOS VALDIVIESO

PROFESOR DEL SEMINARIO DE GRADUACIÓN

_____

MSC. RONALD PONGUILLO

PROFESOR DELEGADO POR LA UNIDAD ACADÉMICA

DECLARACIÓN EXPRESA

“La responsabilidad por los hechos, ideas y doctrinas expuestos en esta tesina, nos corresponde exclusivamente, y el patrimonio intelectual de la misma a la ESCUELA SUPERIOR POLITECNICA DEL LITORAL.”

(Reglamentos y exámenes y títulos profesionales de la ESPOL)

_____

FERNANDO ENRIQUE ORBEA GUERRERO

_____

LUIS ALBERTO PATINO SANCHEZ

RESUMEN

El presente documento corresponde al seminario de graduación “Microcontroladores Avanzados”, este proyecto consiste en el monitoreo de datos de vibraciones de motores BLDC haciendo uso de un sensor de impactos de sonido, quien será el encargado de obtener de los datos de vibración hacia el AVR Butterfly que actuará como maestro para luego ser transmitido mediante comunicación SPI (Serial Peripheral Interface) a la tarjeta LPC1769 utilizada como esclavo para posteriormente representar en un display el análisis de los datos obtenidos.

Algunos dispositivos solo pueden ser transmisores y otros solo receptores, generalmente un dispositivo que tramite datos también puede recibir. El bus SPI emplea un simple registro de desplazamiento para transmitir la información.

ÍNDICE GENERAL

ABREVIATURAS

µC	MCU Microcontrolador
GCC	GNU Compiler Collection
CC	Corriente Continua
CA	Corriente Alterna
PWM	Pulse Width Modulation (Modulación de Ancho de Pulso)
Vref	Voltaje de referencia
Vdd	Voltaje de alimentación +5V del microcontrolador
GND	Tierra
Vss	Voltaje de alimentación +0V del microcontrolador
Motor BLDC	Motor Brushless Direct Current (Motor sin Escobillas de Corriente Continúa)
ARM	Advanced RISC Machines
LCD	Liquid Crystal Display (Pantalla de Cristal Líquido)
TMR0	Módulo Temporizador 0
V	Voltios
Amp	Amperios
I2C	Inter-Integrated Circuit (Circuitos Inter-Integrados)
CAN	Controller Area Network (Red Area de Controlador)
RAM	Random Access Memory (Memoria de acceso aleatorio)
ROM	Read Only Memory (Memoria de sólo lectura)

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

La comunicación a usar en el proyecto será la Interfaz de Periférico Serie (SPI-Serial Peripheral Interface) es un bus de cuatro líneas, sobre el cual se transmiten paquetes de información de 8 bits. Cada una de estas cuatro líneas porta la información entre los diferentes dispositivos conectados al bus, ya sea esto en una comunicación simple de un maestro y un esclavo, como en la de un maestro y varios esclavos. Cada dispositivo conectado al bus puede actuar como transmisor y receptor al mismo tiempo, por lo que este tipo de comunicación serial es llamado también full dúplex. Dos de estas líneas trasfieren los datos en diferentes modalidades MISO (Master Input Slave Output) y MOSI (Master Output Slave Input) una en cada dirección y las otras dos líneas son la señal de reloj SCK (Signal Clock) y el selector de esclavo necesaria cuando en una comunicación tenemos varios esclavos SSEL (Slave Selector).

Para realizar este monitoreo de vibración haremos uso de un sensor de impactos de sonidos y de dos microcontroladores de familias diferentes comunicándolas mediante SPI.

CAPÍTULO 1

1. DESCRIPCIÓN GENERAL DEL PROYECTO

1.1. Antecedentes

Los motores de corriente continua sin escobillas o más conocidos como motores BLDC se han venido utilizando desde hace años en el área industrial en equipos tales como ventiladores, acondicionadores de aire, de bombeo, transportadores industriales, elevadores, llenadoras, tornos, fresadoras, etc.. Actualmente los motores BLDC son muy utilizados en el mercado del aeromodelismo por las prestaciones que entrega especialmente por su eficiencia.

Se trata de motores eléctricos que no usan escobillas, debido a esto no disipan la energía proveniente de las baterías en forma de calor. Esto supondría más potencia, más aceleración, mayor duración, y la posibilidad de mover mayores pesos.

1.2. Motivaciones para el proyecto

El poder desarrollar aplicaciones en microcontroladores de diferentes familias poniendo en práctica los conocimientos adquiridos a lo largo de nuestra vida académica es la principal motivación que nos llevó a realizar este proyecto, también el de estar actualizados con las nuevas implementaciones que han sido desarrolladas en estos microcontroladores.

1.3. Identificación del problema

En vista de que el uso de este tipo de motores ha tenido mucha acogida en distintas aplicaciones, se hace necesario monitorear variables que permitan entregar información sobre la integridad del motor, tales como las vibraciones del mismo, ya que al hacer esto, se podrán interpretar los datos recibidos, tomar medidas preventivas o a su vez correctivas según sea el caso, evitando el daño inesperado y por ende el paro prolongado en el proceso que se estuviere ejecutando.

1.4. Metas y Objetivos

El objetivo principal en este trabajo es el de introducir al estudio del lenguaje de programación que usan estas tarjetas, conocer sus características, realizar ejemplos demostrativos donde se involucre el protocolo de comunicación SPI, dándole un enfoque más específico y práctico para que de esta manera se pueda enlazar y comprender toda la teoría de

funcionamiento; haciendo uso de varias herramientas como el software LPCXpresso con la tarjeta de trabajo LPC1769 de NXP con un procesador ARM Cortex-M3 de 32-Bits y del AVR STUDIO 4 para poder programar el cerebro del kit de desarrollo AVR BUTTERFLY que consiste de un microcontrolador ATmega169.

CAPÍTULO 2

2. MARCO TEÓRICO

2.1. Motor BLDC

2.1.1 Funcionamiento de los motores CC con escobillas

Estos motores están formados por dos partes fundamentales, una parte estática y otra móvil. La parte que no se mueve es la carcasa del motor, en cuyo interior y unidos a ella están situados dos imanes permanentes. A esta parte se ha denominado estator.

En el interior de esa carcasa está la parte móvil (rotor) que también se denomina inducido. Está formado por un eje metálico con tres polos que sobresalen, y en cada polo está enrollado en forma de bobina el hilo conductor.

Cuando la corriente eléctrica pase por la bobina, se producirá un campo magnético que interaccionará con el campo magnético que producen los imanes del estator.

La consecuencia es una fuerza que hará que gire el inducido, y por tanto el eje del motor. De esta forma se transforma la energía eléctrica, que llega al motor, en energía mecánica [1].

La corriente llega al inducido a través de las delgas que están situadas en su parte superior y forman lo que se denomina el conmutador o colector. A su vez las delgas reciben la electricidad de las escobillas, que son unos contactos deslizantes que rozan las delgas mientras el rotor gira.

Como una escobilla está conectada al polo positivo de la corriente y otra al negativo, la corriente que transmiten a las delgas hace que el inducido gire. Son necesarias al menos tres delgas para garantizar que el rotor se mueva, ya que si solo hubiera dos, el rotor podría no iniciar el giro al quedar en posición perpendicular a las fuerzas magnéticas de los imanes.

También hemos de saber que las escobillas se mantienen pegadas a las delgas mediante la presión de sendos muelles. El inducido gira dentro del estator sobre rodamientos.

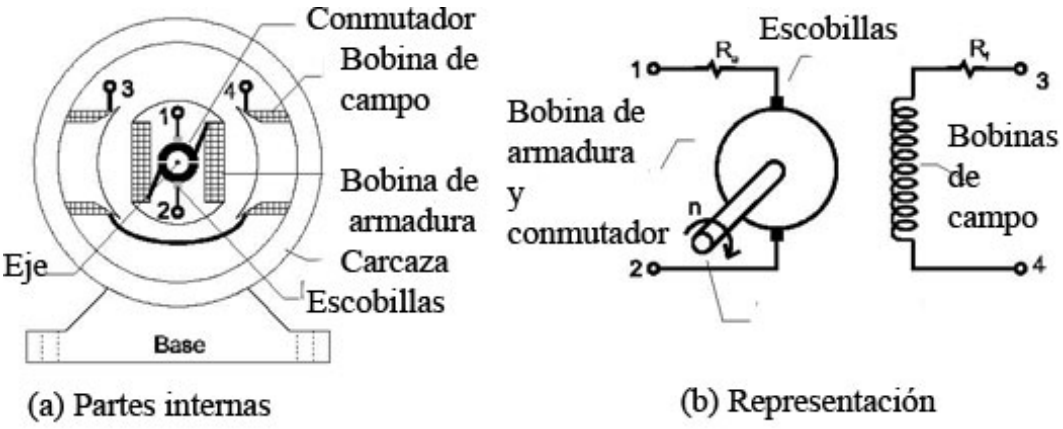

Figura 2. 1 Esquema sencillo del motor con escobillas [2]

2.1.2 Funcionamiento de los motores CC sin escobillas

A modo de resumen, se puede decir que los motores “sin escobillas” son como los motores “con escobillas” pero del revés. Es decir el rotor, la parte móvil, está compuesto por el eje y los imanes permanentes. En la carcasa o estator es donde se encuentra el bobinado del hilo conductor, que no se mueve.

La corriente eléctrica pasa por el hilo conductor que está bobinado en la carcasa y produce el campo electromagnético que hace girar a los imanes permanentes y por tanto al eje al que están unidos.

Por ello ni las escobillas ni el conmutador son necesarios, ya que la corriente va al estator. Además, no existen las delgas que eran las que obligaban al rotor a moverse cualquiera que fuera su posición.

El variador electrónico es el que controla en qué posición se encuentra el rotor para darle la corriente temporizada adecuada. Esto se realiza o mediante sensores instalados en el motor o a través de la respuesta que obtiene cuando envía una corriente lineal al motor. Debido a esto los variadores electrónicos de los motores sin escobillas han de ser mucho más complejos que los usados en motores con escobillas, ya que han de procesar la información del funcionamiento del motor a tiempo real[3].

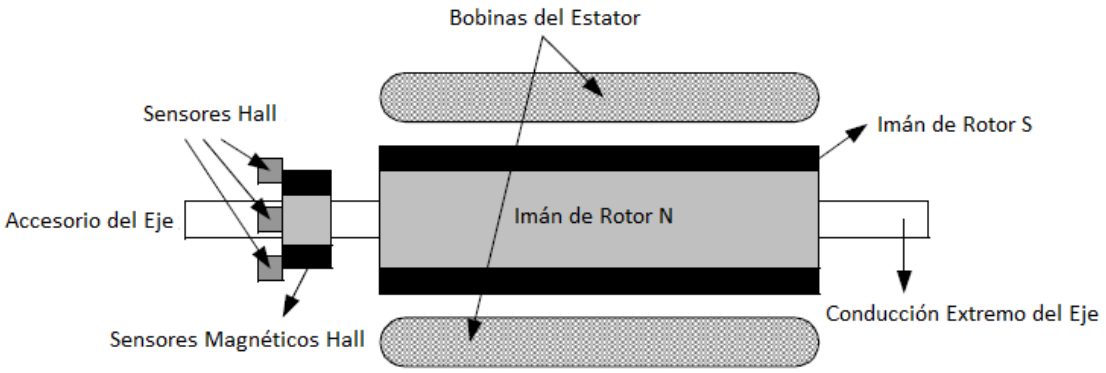

Figura 2. 2 Esquema sencillo del motor sin escobillas [4]

2.1.3 Comparación entre motores DC convencionales y sin escobillas

En los motores con escobillas la conmutación se hace mecánicamente a través del contacto entre el inducido y las escobillas. Este sistema es muy poco eficiente y el rozamiento y la resistencia eléctrica provocan que haya una gran pérdida de energía que se transforma en calor.

Además esto supone una limitación al número máximo de r.p.m. de los motores, ya que a elevadas r.p.m. las escobillas rebotarían. Para evitarlo serían necesarios muelles más rígidos y esto crearía a su vez más fricción y frenaría el giro[5].

En los motores sin escobillas la conmutación se controla de manera electrónica mediante el variador de velocidad, por lo que no se produce rozamiento mecánico ni pérdidas de energía y se consigue que los motores sin escobillas tengan una eficiencia muy superior, se habla de un 90%, frente a un 60% de los motores con escobillas.

El calentamiento del motor es mínimo y la masa que gira es menor, casi la mitad, por lo que aceleran más rápidamente. Gracias a esto pueden alcanzar muchas más r.p.m. y con mucho más par, hasta cuatro o cinco veces más que los motores con escobillas, y a la vez con un ahorro de energía de hasta el 30 %.

Esto se debe a que el rotor lleva los imanes, que son menos pesados en estos motores que el bobinado en los clásicos. Por ello el funcionamiento es además más suave al reducirse las vibraciones.

Al no haber chisporroteo eléctrico debido al roce de las escobillas con el conmutador, se eliminan las interferencias por el "ruido" eléctrico que podrían afectar al equipo de radio. Tampoco son necesarios ni los condensadores soldados al motor ni el diodo Schottky.

En los motores con escobillas hay que realizar mantenimientos periódicos; nuevas escobillas, muelles, desgaste del conmutador y mucho tiempo dedicado a las labores de mantenimiento. Por el contrario, en los motores sin escobillas el mantenimiento es mínimo, ya que al no tener ni escobillas ni conmutador, el mantenimiento es casi inexistente, solo sería necesario la limpieza y lubricado de los rodamientos. Adicionalmente, el concepto de "sin escobillas" permite fabricar motores totalmente cerrados, protegiéndolos del polvo[6].

Los motores sin escobillas son más sencillos y por ello más fiables: no llevan ni las escobillas, ni guías de escobillas, ni conmutador, ni muelles. Es decir menos pérdidas por culpa de fallos mecánicos del motor.

Por otra parte y gracias al control ejercido por el variador electrónico, que es de hecho un microprocesador digital, es posible regular el par y las r.p.m. e igualar las prestaciones de los motores.

Se podrían hacer carreras con motores de prestaciones iguales y se podría controlar en la inspección técnica. Además no habría que disponer de varios motores, solo sería necesario programar el número de r.p.m.

2.2. Interfaz Periférica Serial – SPI

La interfaz Periférica Serial (SPI, del inglés Serial Peripheral Interface) es un estándar de comunicación, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos.

El bus de interfaz de periféricos serie o bus SPI es un estándar para controlar casi cualquier dispositivo electrónico digital que acepte un flujo de bits serie regulado por un reloj, es decir que consiste en un subsistema de comunicaciones seriales independiente, que le permite al microcontrolador comunicarse síncronamente con los dispositivos periféricos, como los registros de corrimiento, drivers de display de cristal líquido LCD, subsistemas de conversión analógico-digital, e incluso otros microcontroladores[7].

Incluye una línea de reloj, dato entrante, dato saliente y un pin de selección, que conecta o desconecta la operación del dispositivo con el que uno desea comunicarse. Cada dispositivo puede actuar como transmisor y receptor al mismo tiempo, por lo que este tipo de comunicación serial es full dúplex.

SPI es un sistema de bajo coste para comunicaciones de corta distancia, como por ejemplo, entre pequeños procesadores y sus periféricos. El protocolo SPI necesita dos dispositivos para la comunicación. Uno de ellos es considerado como un maestro y otro como esclavo.

2.2.1 Ventajas del SPI

- Mayor velocidad de transmisión que con I²C o SMBus.
- Consume menos energía que I²C o que SMBus debido que posee menos circuitos (incluyendo las resistencias pull-up) y estos son más simples.
- Su implementación en hardware es extremadamente simple.
- Protocolo flexible en que se puede tener un control absoluto sobre los bits transmitidos.
- No está limitado a la transferencia de bloques de 8 bits.
- Elección del tamaño de la trama de bits, de su significado y propósito.
- Comunicación Full Dúplex.
- No es necesario arbitraje o mecanismo de respuesta ante fallos.
- Los dispositivos clientes usan el reloj que envía el servidor, no necesitan por tanto su propio reloj.
- No es obligatorio implementar un transceptor (emisor y receptor), un dispositivo conectado puede configurarse para que solo envíe, sólo reciba o ambas cosas a la vez.

- Usa muchos menos terminales en cada chip/conector que una interfaz paralelo equivalente.
- Como mucho una única señal específica para cada cliente (señal SS), las demás señales pueden ser compartidas.

2.2.2 Desventajas del SPI

- Consume más pines de cada chip que I²C, incluso en la variante de 3 hilos.
- El direccionamiento se hace mediante líneas específicas (señalización fuera de banda) a diferencia de lo que ocurre en I²C que se selecciona cada chip mediante una dirección de 7 bits que se envía por las mismas líneas del bus.
- Sólo funciona en las distancias cortas a diferencia de, por ejemplo, RS-232, RS-485, o Bus CAN.
- No permite fácilmente tener varios servidores conectados al bus.
- No hay control de flujo por hardware.
- No hay señal de asentimiento. El servidor podría estar enviando información sin que estuviese conectado ningún cliente y no se daría cuenta de nada[8].

2.2.3 Comunicación SPI entre un Maestro y un Esclavo

Un maestro 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 a través de una línea selectora llamada Chip Select o Select Slave, por lo tanto un esclavo es activado sólo cuando esta línea es seleccionada. Generalmente una línea de selección es dedicada para cada esclavo.

Cuando la configuración es como maestro, la transferencia de datos puede ser tan alta como una proporción de un medio de los ciclos del reloj. Cuando la configuración es como esclavo puede ser tan rápida como la razón del reloj.

Figura 2. 3 Conexión SPI entre el maestro y un esclavo

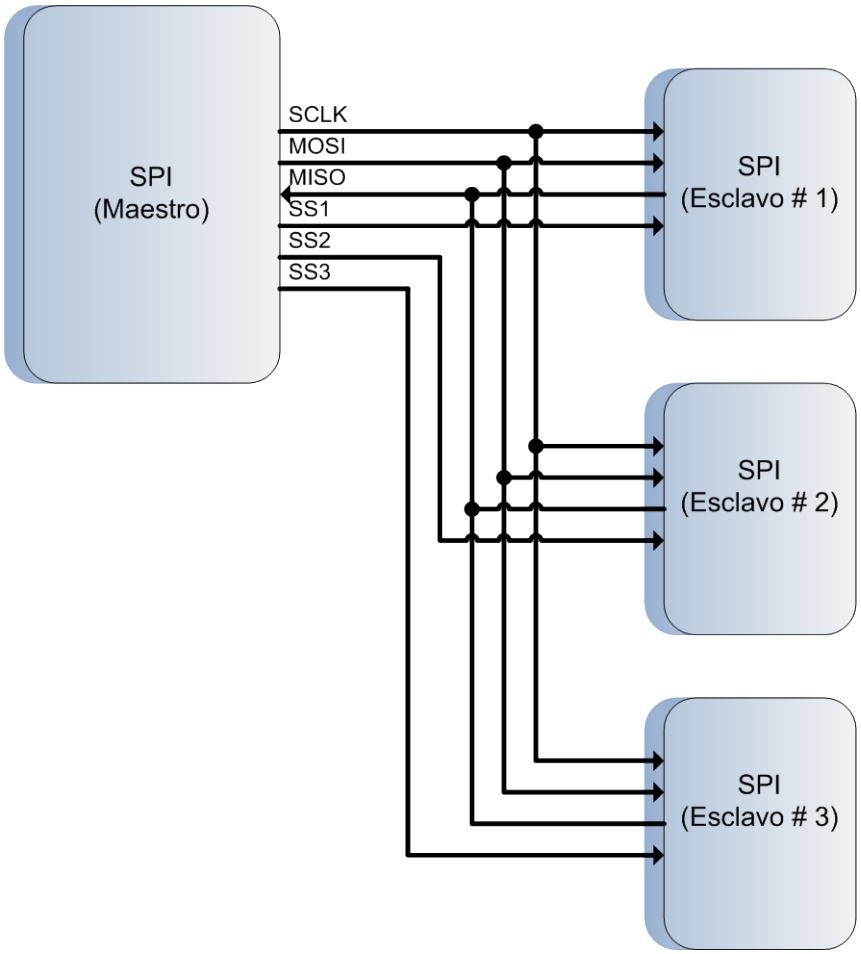

Figura 2. 4 Conexión SPI entre el maestro y varios esclavos

2.3. AVR Butterfly Atmega 169

2.4.1 Hardware

El Kit AVR Butterfly está diseñado para demostrar los beneficios y las principales características de los microcontroladores ATMEL.

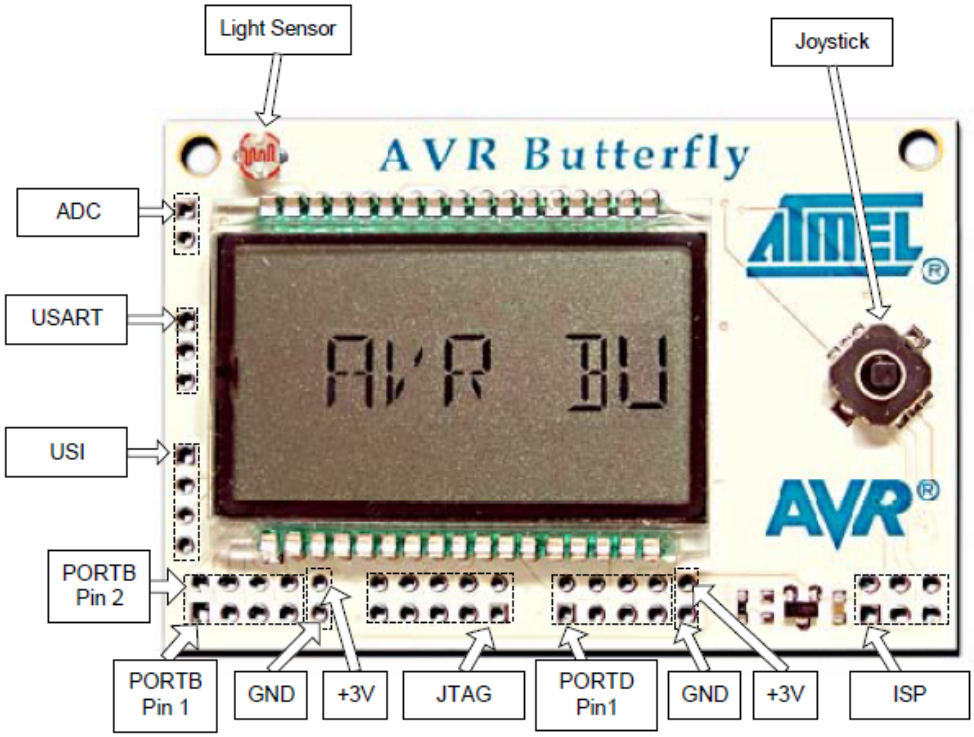

Figura 2. 5 Identificación AvrButterfly [9]

AVR Butterfly es un módulo de soporte que puede ser utilizado en numerosas aplicaciones. El AVR Butterfly contiene un microcontrolador ATMega169, el cual va a realizar el comando de las diferentes funciones de las que es capaz este kit. El AVR Butterfly presenta las siguientes características principales:

- La arquitectura AVR en general y la ATmega169 en particular.
- Diseño de bajo consumo de energía.
- El encapsulado tipo MLF.

- Periféricos:
  - Controlador LCD.
  - Memorias:
  - Flash, EEPROM, SRAM, Data Flash externa.
- Interfaces de comunicación:
  - UART, SPI, USI.
- Métodos de programación
  - Self-Programming/Bootloader, SPI, Paralelo, JTAG.
- Convertidor Analógico Digital (ADC).
- Timers/Counters:
  - Contador de Tiempo Real (RTC).
  - Modulación de Ancho de Pulso (PWM).

Este Kit puede reprogramarse de varias formas diferentes, incluyendo programación serial a través del puerto JTAG; pero se prefiere el uso del Bootloader precargado junto con el AVR Studio, para descargar nuevo código sin la necesidad de hardware especial.

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

Los siguientes recursos están disponibles en el AVR Butterfly:

- Microcontrolador ATmega169V (en encapsulado tipo MLF).
- Pantalla tipo vidrio LCD de 120 segmentos, para demostrar las capacidades del controlador de LCD incluido dentro del ATmega169.
- Joystick de cinco direcciones, incluida la presión en el centro.
- Altavoz piezoeléctrico, para reproducir sonidos.
- Cristal de 32 KHz para el RTC.
- Memoria Data Flash de 4 Mbit, para el almacenar datos.
- Convertidor de nivel RS-232 e interfaz USART, para comunicarse con unidades fuera del Kit sin la necesidad de hardware adicional.
- Termistor de Coeficiente de Temperatura Negativo (NTC), para sensar y medir temperatura.
- Resistencia Dependiente de Luz (LDR), para sensar y medir intensidad luminosa.
- Acceso externo al canal 1 del ADC del ATmega169, para lectura de voltaje en el rango de 0 a 5 V.
- Emulación JTAG, para depuración.
- Interfaz USI, para una interfaz adicional de comunicación.
- Terminales externas para conectores tipo Header, para el acceso a periféricos.
- Batería de 3 V tipo botón (600mAh), para proveer de energía y permitir el funcionamiento del AVR Butterfly.

- Bootloader, para programación mediante la PC sin hardware especial.
- Aplicación demostrativa pre programada.
- Compatibilidad con el Entorno de Desarrollo AVR Studio 4.

El ATmega 169 es un microcontrolador de baja potencia CMOS de 8 bits basado en el AVR mejorado de la arquitectura RISC. Mediante la ejecución de instrucciones de gran alcance en un solo ciclo de reloj, el ATmega 169 logra tasas de transferencia cerca de 1 MIPS por MHz que permite al diseñador del sistema optimizar el consumo de energía en comparación con la velocidad de procesamiento. El núcleo AVR combina un amplio conjunto de instrucciones con 32 registros de propósito general de trabajo.

Todos los 32 registros están conectados directamente a la unidad lógica aritmética (ALU), lo que permite dos registros independientes que se alcanzará en una sola instrucción ejecutada en un ciclo de reloj.

La arquitectura resultante es un código más eficiente mientras que alcanza rendimientos de hasta 10 veces más rápido que los convencionales microcontroladores CISC.

El ATmega 169 proporciona las siguientes características:

- 16k bytes de sistema programable.
- Flash con lectura y escritura mientras que las capacidades, 512 bytes de EEPROM, SRAM bytes 1K.
- 54 registros de propósito general.
- 32 registros de propósito general de trabajo.
- Controlador de LCD con la resistencia de step-up de tensión.
- Una serie UART programable, serie universal.
- Sistema de interrupción.
- Interfaz con el inicio de condición del detector.
- El modo Power down guarda el contenido del registro.

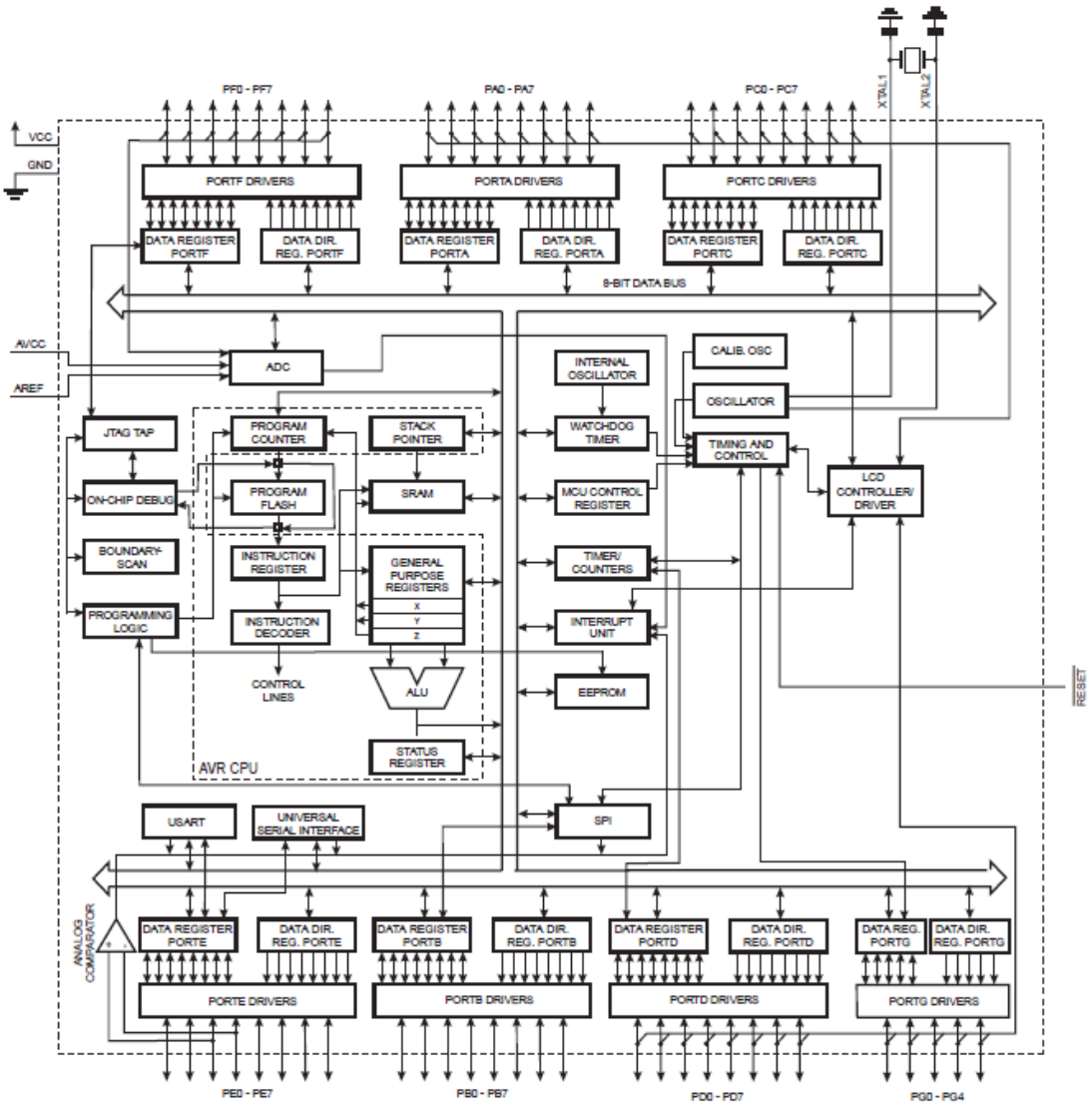

Figura 2. 6 Diagrama de Bloques del Atmega 169 [9]

2.4.2 Software

AVR Studio 4 posee un entorno muy acogedor con el usuario el cual ayuda con la visualización para la creación de programas en lenguaje *.asm (assembler) el cual es un lenguaje muy parecido al lenguaje de máquina que usan los micros y computadoras al momento de cargar al programa en el micro, incluye un editor, ensamblador, descargador de archivos *.hex y un emulador de microcontroladores.

Figura 2. 7 Ventana de presentación del AVR Studio 4.0

AVR Studio 4 es compatible con la gama completa de herramientas de ATMEL AVR y cada versión siempre contiene las últimas actualizaciones para las herramientas y el apoyo de los dispositivos AVR nuevos.

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 (GNU Compiler Collection).

AVR Studio 4 incorpora un depurador que permite el control de ejecución con fuente y nivel de instrucción, paso a paso y puntos de interrupción, el registro, la memoria y E/S puntos y configuración y gestión, y apoyo a la programación completa para los programadores independientes, además permite crear archivos assembler (asm) y archivos C [10].

2.4.3 Operación del SPI

El SPI Maestro inicializa el ciclo de comunicación cuando se coloca en bajo el Selector de Esclavo (SS). Maestro y Esclavo preparan los datos a ser enviados en sus respectivos registros de desplazamiento y el maestro genera el pulso del reloj en el pin SCK para el intercambio de datos. Los datos son siempre intercambiados desde el Maestro al Esclavo en MOSI, y desde el Esclavo al Maestro en MISO.

Cuando se configura un dispositivo como Maestro, la interfaz SPI no tendrá un control automático de la línea SS. Este debe ser manejado por software antes de que la comunicación pueda empezar; cuando esto es realizado,

escribiendo un byte en el registro de la SPI comienza el reloj de la SPI, y el hardware cambia los 8 bits dentro del Esclavo. Después de cambiar un Byte, el reloj del SPI para, habilitando el fin de la transmisión (SPIF).

El maestro transmite un bit de su SPDR al dispositivo esclavo en cada ciclo de reloj. Esto significa que para enviar un byte de datos, se necesitan 8 pulsos de reloj. Después de cada paquete de datos el Maestro debe sincronizar el esclavo llevando a 'alto' el selector de Esclavo, SS.

Si la interrupción del SPI está habilitada (SPIE) en el registro SPCR, una interrupción es requerida. El maestro podría continuar al cambio del siguiente byte escribiendo dentro del SPDR, o señalizar el fin del paquete colocando en alto el Esclavo seleccionado, línea SS. El último byte llegado se mantendrá en el registro Buffer para luego usarse.

Cuando se configura como Esclavo, la interfaz ISP permanecerá durmiendo con MISO en tres-estados siempre y mientras el pin SS esté deshabilitado. En este estado, por el software se podría actualizar el contenido del registro SPDR, pero los datos no serán desplazados por la llegada del pulso de reloj en el pin SCK hasta que el pin SS sea habilitado, luego de esto será visto como un byte completamente desplazado en el fin de la transmisión cuando SPIF se habilite.

Cuando el dispositivo trabaja como maestro, el usuario puede determinar la dirección del pin SS, si es configurado como salida, el pin es una salida general la cual no afecta el sistema SPI. Típicamente, el pin SS será manejado desde el Esclavo, si es como entrada, éste debe ser enviado en alto para asegurar la operación SPI del Master.

Si la interrupción SPI está habilitada, una interrupción es solicitada. El Esclavo podría continuar para colocar nuevos datos para ser enviados dentro del SPDR antes de seguir leyendo el dato que va llegando. El último byte que entra permanecerá en el buffer para luego usarse.

Nombre del pin	Función
SS	**Slave Select:** seleccionar al dispositivo como esclavo.
SCK	**SPI Clock:** señal de reloj para SPI.
MOSI	**Master Out Slave In:** pin de salida para el maestro y de entrada para el esclavo.
MISO	**Master In Slave Out:** pin de entrada para el maestro y de salida para el esclavo.

Tabla 2. 1 Función de los pines del SPI

En resumen, las principales características del SPI son:

- Operación Maestro-Esclavo
- Transferencia de Datos LSB o MSB
- Finalización de transmisión por Bandera de Interrupción
- Siete velocidades programables en los bits
- Transferencia de Datos síncrona tres-cables, bidireccional

- La protección de escritura contra colisión de bandera. (Write Collision Flag Protection).

2.4.4 Registros del SPI

- **Registro de Control del SPI – SPCR**

Bit	7	6	5	4	3	2	1	0
	SPIE	**SPE**	**DORD**	**MSTR**	**CPOL**	**CPHA**	**SPR1**	**SPR0**
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

Tabla 2. 2 Registro de control de SPI - SPCR

- 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 siguiente tabla:

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

Tabla 2. 3 Selección de frecuencia de la señal de reloj del SPI

- 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.

- 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 un 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 configurado con uno.

- 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**

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

Tabla 2. 4 Registro de Estado del SPI - SPSR

- 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**

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

Tabla 2. 5 Registro de Datos del SPI - SPSR

El Registro de Datos SPI 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.

2.4.5 Modos del Reloj

Toda la transferencia de los datos, es sincronizada por la línea de reloj de este bus. La mayoría de las interfaces SPI tienen 2 bits de configuración, llamados CPOL (Clock Polarity = Polaridad de Reloj) y CPHA (Clock Phase = Fase del Reloj). CPOL determina si el estado Idle de la línea de reloj esta en bajo (CPOL=0) o si se encuentra en un estado alto (CPOL=1). CPHA determina en que flanco de reloj los datos son desplazados hacia dentro o

hacia fuera (Si CPHA=0 los datos sobre la línea MOSI son detectados en cada flanco de bajada y los datos sobre la línea MISO son detectados cada flanco de subida).

Existen cuatro modos de reloj definidos para el protocolo SPI, estos son:

Modo de Reloj	Estado de CPOL	Estado de CPHA
A	1	1
B	0	1
C	1	0
D	0	0

Tabla 2. 6 Especificaciones de los Modos de reloj definidos en el SPI

Para un mejor entendimiento estos modos son ilustrados a continuación.

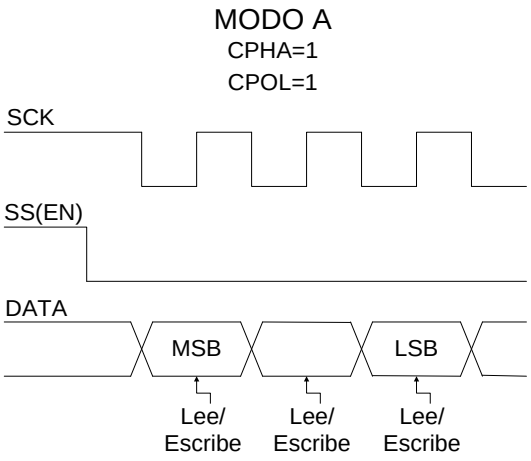

Figura 2. 8 Modo A del reloj definido para el Protocolo SPI

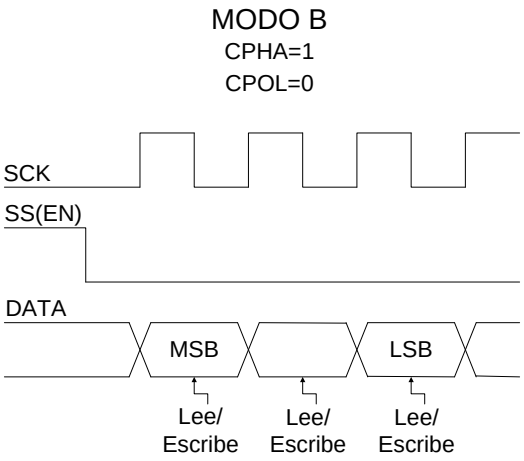

Figura 2. 9 Modo B del reloj definido para el Protocolo SPI

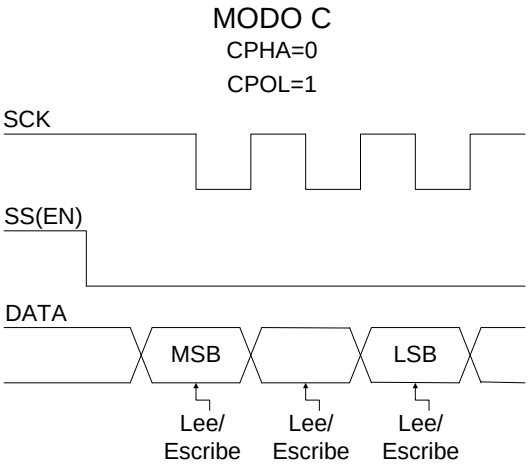

Figura 2. 10 Modo C del reloj definido para el Protocolo SPI

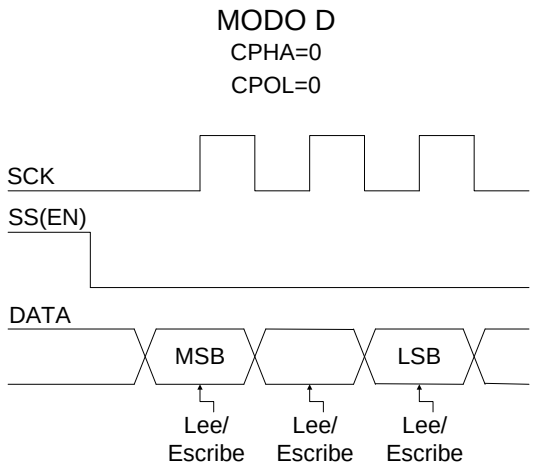

Figura 2. 11 Modo D del reloj definido para el Protocolo SPI

2.4. LPC1769

2.4.1 Hardware

El LPC1769 es un ARM Cortex-M3 basado en un microcontrolador para aplicaciones embebidas que requieren un alto nivel de integración y baja disipación de potencia.

El procesador ARM Cortex-M3 es la próxima generación de los CORE, que ofrece mejoras en el sistema de depuración, tales como modernización características y un mayor nivel de apoyo a la integración del bloque.

Las versiones de alta velocidad (LPC1769 y LPC1759) operan a frecuencias de hasta 120 MHz. Otras versiones operan a una frecuencia de hasta 100 MHz. EL CPU de la ARM Cortex-M3 incorpora una tubería de 3 etapas y

utiliza una arquitectura de hardware con instrucción local y buses de datos separados, así como un tercer autobús para los periféricos. El procesador ARM Cortex-M3 también incluye una unidad de captación previa interna que soporta procesos especulativos.

Los complementos periféricos de la LPC1769 incluyen:

- Hasta 512 KB de memoria flash, hasta 64 kB de memoria de datos
- Ethernet MAC
- Una interfaz USB que puede ser configurada como: host, dispositivo o OTG
- 8 canales de uso general controlados por DMA
- 4 UARTs
- 2 canales CAN
- 2 controladores de SSP
- interfaz SPI
- 3 interfaces I2C
- 2 plus de entrada y 2 interfaces de salida I2S
- 8 canales ADC de 12 bits
- 10 bits DAC
- Control de motor PWM
- Interfaz de codificador de cuadratura
- 4 temporizadores de uso general

- 6 de salida de propósito general PWM
- Una fuente ultra-low RTC con suministro de batería por separado, y hasta 70 pines I/O de propósito general [11].

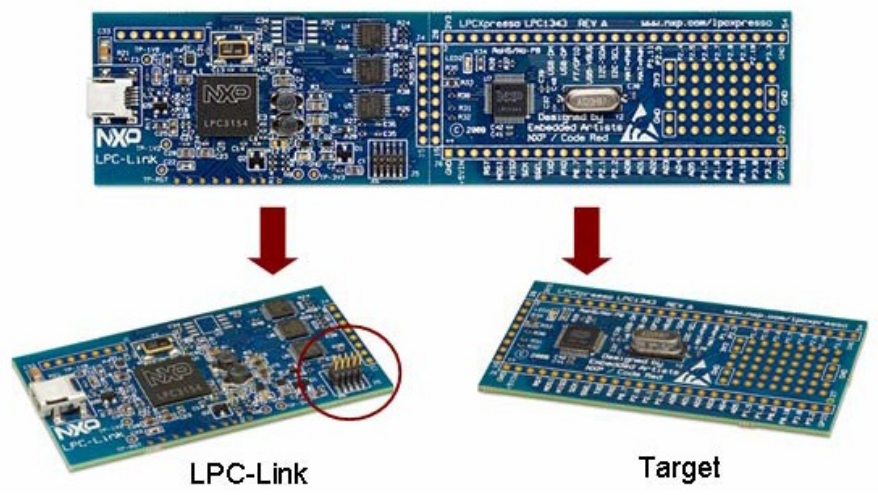

Figura 2. 12 Identificación Tarjeta LPC1769 [11]

Características:

- Los procesadores de ARM cortex-M3 corren a frecuencias de hasta 100 MHz (LPC1768/67/66/65/64/63) o de hasta 120 MHz (LPC1769). Una unidad de protección de Memoria (MPU) el apoyo a 8 regiones esta incluido.
- ARM Cortex-M3 contiene un controlador anidado de vector de interrupciones.

- Hasta 512 kB de memoria de programación flash en un chip. Mayor acelerador de memoria flash permite alta velocidad de operación de 120 MHz con cero estados de espera.
- Programación ISP e IAP a través de un chip gestor de arranque de software.
- El chip SRAM incluye:
  - De 32/16 KB de SRAM con un bus local de códigos/datos de alto rendimiento para el acceso al CPU.
  - Uno/dos bloques de memoria SRAM de 16kB con vías de acceso separadas para brindar un mayor rendimiento. Estos bloques SRAM puede ser utilizado para Ethernet, USB y memoria DMA, así como para instrucciones de prepósito general para el CPU y almacenamiento de datos.
- Ocho canales de propósito general controladas por DMA (GPDMA) sobre la matriz de multicapa AHB que puede ser usada con: SSP, I2S bus, UART, convertidores periféricos de analógico a digital y Digital a analógico, y para transferencias de memoria a memoria.
- La matriz de interconexión de multicapa AHB ofrece un autobús para cada maestro de AHB. Los maestros AHB incluyen EN la CPU, controladores de propósito general DMA, MAC Ethernet y la interfaz USB. Estas interconexiones permiten la comunicación con retardos no arbitrarios.

- Interfaz estándar JTAG test/debug para la compatibilidad con las herramientas existentes. Opción de puertos: Cable Serial debug y cable serial Trace.
- El módulo de emulación de seguimiento no permite intrusos, debido al seguimiento en tiempo real de la instrucción ejecutada.
- Cuatro modos de reducción de energía: Sleep, Deep-sleep, Power-down, and Deep power-down.
- Solo 3,3 V de voltaje de alimentación.
- Cuatro entradas de interrupción externa configuradas como sensibles. Todos los pines del puerto 0 y el puerto 2 pueden ser configuradas como fuentes de interrupción sensibles.
- Función de reloj de salida se puede reflejar en el reloj oscilador principal, el reloj del IRC, el reloj RTC, Reloj de la CPU y el reloj USB.
- Oscilador de cristal con un rango de operación de 1 MHz a 25 MHz.
- 4 MHz de oscilador interno RC, con el 1% de precisión, que opcionalmente se puede utilizar como un del reloj del sistema.
- Protección del CODE RED con diferentes niveles de seguridad.
- Interfaces serial: RMII, SSP, SPI, I2C, I2S, controladores DMA, UARTs, 2 canales controlados por CAN.

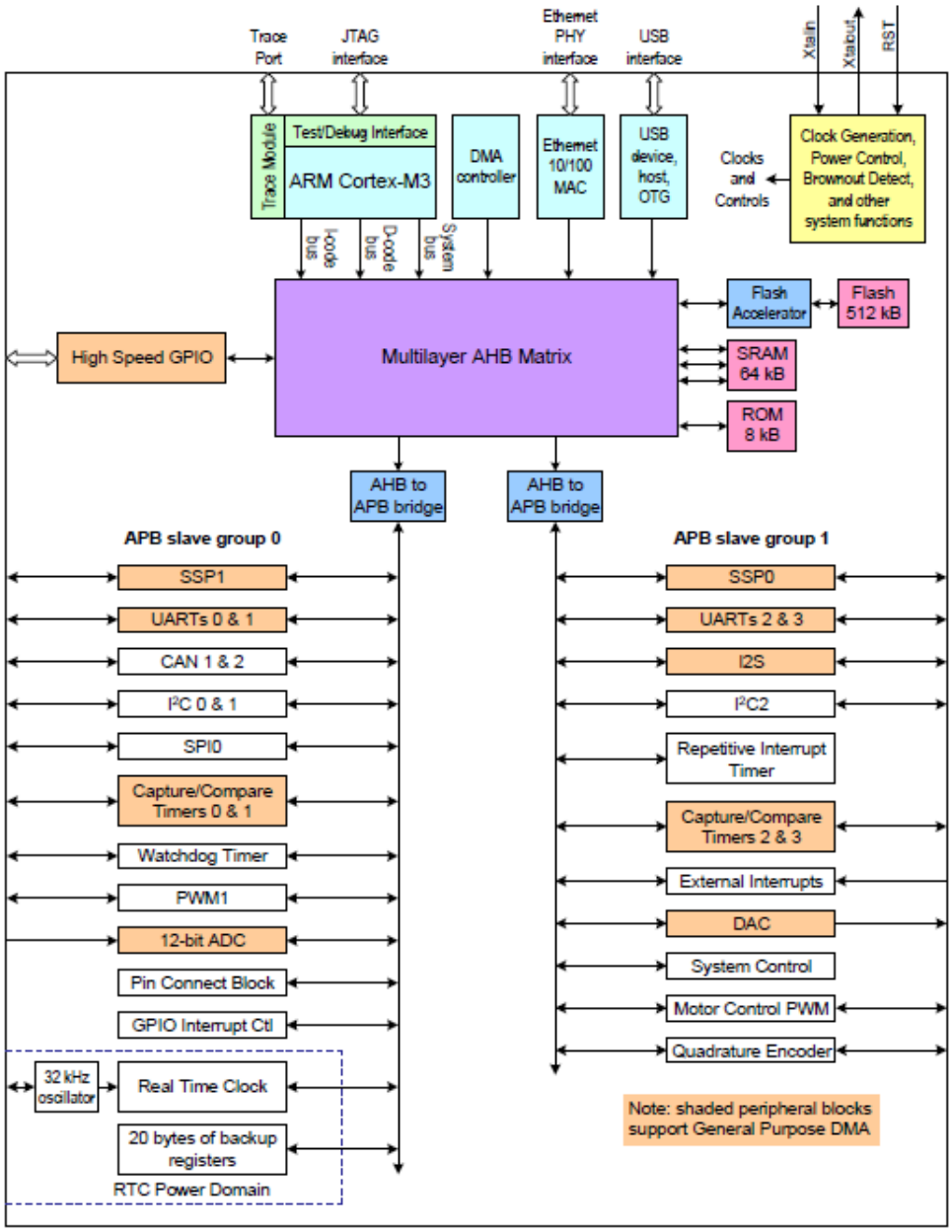

Figura 2. 13 Diagrama de bloques de la LPC1769 [11]

2.4.2 Software

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, de una manera efectiva en tiempo y costo.

Figura 2. 14 Ventana de presentación de LPCXpresso 4

LPCXpresso es una nueva plataforma de bajo costo de desarrollo disponible en NXP. 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.

LPCXpresso se basa en Eclipse, que permite muchas mejoras específicas en LPC. También cuenta con la versión actual de una cadena de herramientas de la industria estándar GNU, con una patente basada en librerías en C. El LPCXpresso IDE puede construir un ejecutable de cualquier tamaño con optimización del código completo, y soporta un límite de descarga de 128 KB después del registro.

2.4.3 Operación del SPI [12]

Las dos interfaces de SSP, SSP0 SSP1 y se configuran con los siguientes registros:

Encendido: En el registro PCONP, establezca el bit PCSSP0 para permitir SSP0 y el bit PCSSP1 para permitir SSP1 Tabla Anexo 2 Registro PCONP.

Nota: Si se reinicia, ambas interfaces de SSP están habilitados (PCSSP0/1=1).

Reloj: En PCLKSEL0 seleccione PCLK_SSP1, en PCLKSEL1 seleccione PCLK_SSP0 Tabla Anexo 3 Registro PCLKSEL0 y Tabla Anexo 4 Registro Pre-escalador del Reloj.

Pines: Seleccione las clavijas de SSP a través de los registros PinSel y los modos de clavijas través de los registros PinMode Tabla Anexo 5 Registros PinSel.

Registros del SPI

Las direcciones de registro de las direcciones de los controladores de SSP se muestran en la siguiente tabla.

Nombre Genérico	Descripción	Acceso	Valor de Reinicio	Nombre de Registro SSPn y Dirección
CR0	Registro de Control 0	L/E	0	SSP0CR0 - 0x40088000 SSP1CR0 - 0x40030000
CR1	Registro de Control 1	L/E	0	SSP0CR1 - 0x40088004 SSP1CR1 - 0x40030004
DR	Registro de dato	L/E	0	SSP0DR - 0x40088008 SSP1DR - 0x40030008
SR	Registro de estatus	L		SSP0SR - 0x4008800C SSP1SR - 0x4003000C
CPSR	Registro Pre-escalador del Reloj	L/E	0	SSP0CPSR- 0x40088010 SSP1CPSR- 0x40030010
IMSC	Registro mascara de interrupción y borrado	L/E	0	SSP0IMSC- 0x40088014 SSP1IMSC- 0x40030014
RIS	Registro del estado de interrupción	L/E		SSP0RIS - 0x40088018 SSP1RIS - 0x40030018
MIS	Registro de mascara de interrupción	L/E	0	SSP0MIS - 0x4008801C SSP1MIS - 0x4003001C
ICR	Registro borrado de interrupciones	L/E	NA	SSP0ICR - 0x40088020 SSP1ICR - 0x40030020
DMACR	Registro de control DMA	L/E	0	SSP0DMACR-0x40088024 SSP1DMACR-0x40030024

Tabla 2. 7 Registro de las direcciones de los controladores de SSP

Registro SSPnCR0

Este registro controla el funcionamiento básico del controlador de SSP.

Bit	Símbolo	Valor	Descripción	Valor de reinicio
3:0	DSS		Selecciona Tamaño del Dato	0000

		0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111	4 bits por transferencia. 5 bits por transferencia. 6 bits por transferencia. 7 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.	
2	FRF	00 01 10 11	Formato de Marco. SPI TI Microcable Esta combinación no existe	00
6	CPOL	0 1	Polaridad del reloj Mantiene bus del reloj bajo entre cuadros. Mantiene bus del reloj alto entre cuadros.	0
7	CPHA	0 1	Fase del reloj Captura los datos en serie en la primera transición de reloj. Captura los datos en serie en la segunda transición de reloj.	0
15:8	SCR		Velocidad de Reloj de Serie.	0x00
31:8	-		Reservado	NA

Tabla 2. 8 Registro de control de funcionamiento del controlador SSP

Registro SSPnCR1

Este registro controla ciertos aspectos del funcionamiento del controlador SSP.

Bit	Símbolo	Valor	Descripción	Valor de reinicio
0	LBM		Modo Realimentación	0000

		0 1	Durante operación normal La entrada serie se toma de la salida serie en lugar de la clavija de entrada serie.	
1	SSE	0 1	Habilitador SSP El controlador SSP es deshabilitado El controlador SSP interactuara con otros dispositivos con el bus en serie	00
2	MS	0 1	Modo Maestro/Esclavo El controlador SSP actúa como maestro El controlador SSP actúa como esclavo	0
3	SOD		Deshabilita la salida del esclavo	0
31:4	-		Reservado	NA

Tabla 2. 9 Registro SSPnCR1

Registro SSPnDR

El software puede escribir datos que deben transmitirse a este registro, y leer los datos que haya estado recibido.

Bit	Símbolo	Descripción	Valor de reinicio
15:0	DATO	**Escritura:** software puede escribir datos para ser enviados en un futuro cuadro a este registro cada vez que el bit de TNF en el registro de estado es 1, indicando que el FIFO Tx no se ha completado. **Lectura:** software puede leer datos de este registro cuando el bit de RNE en el registro de estado es 1, indicando que el FIFO Rx no está vacío.	0x0000
1:16	-	Reservado	NA

Tabla 2. 10 Registro SSPnDR

Registro SSPnSR

Este registro de sólo lectura refleja el estado actual del controlador de SSP.

Bit	Símbolo	Descripción	Valor de reinicio
0	TFE	El bit es 1 si la transmisión esta vacía	1
1	TNF	El bit es 0 si el transmisión esta llena	1
2	RNE	El bit es 0 si la recepción esta vacía	0
3	RFF	El bit es 1 si la recepción esta llena	0
4	BSY	El bit es 0 si el controlador SSP esta ocupado	0

31:5	-	Reservado	NA

Tabla 2. 11 Registro SSPnSR

Registro SSPnPSR

Este registro controla el factor por el cual el pre-escalador divide el SSP_PCLK SSP reloj periférico para dar el reloj pre-escalador que es, a su vez, dividido por el factor de SCR en SSPnCR0, para determinar el reloj de bit.

Registro SSPnIMSC

Este registro controla si cada una de las cuatro condiciones posibles de interrupción en el controlador de SSP están habilitadas. Tenga en cuenta que ARM se utiliza la palabra "enmascarado" en el sentido opuesto al clásico de la terminología informática, en el que "enmascarado" significaba "discapacitado". ARM se utiliza la palabra "enmascarado" en el sentido de "activado". Para evitar confusiones, no vamos a utilizar la palabra "enmascarado".

Registro SSPnRIS

Este registro de sólo lectura contiene un 1 para cada condición de interrupción que se afirma, independientemente de si o no la interrupción está habilitada en el SSPnIMSC.

Registro SSPnMIS

Este registro de sólo lectura contiene un 1 para cada condición de interrupción que se afirma y activado en el SSPnIMSC. Cuando se produce una interrupción del SSP, la rutina de servicio de interrupción debe leer este registro para determinar la causa (s) de la interrupción.

Registro SSPnICR

El software se puede escribir una o más una (s) a este registro de sólo escritura, para eliminar la condición de interrupción correspondiente (s) en el controlador de SSP. Tenga en cuenta que las otras dos condiciones de interrupción se puede borrar la escritura o la lectura de la FIFO su caso, o discapacitados en la limpieza de el bit correspondiente en SSPnIMSC.

Registro SSPnDMACR

El registro SSPnDMACR es el DMA de control de registro. Se trata de una lectura / escritura de registro.

2.5. Sensor de impacto de sonido [14]

El sensor de impacto de sonido proporciona un medio para agregar el control del ruido a su proyecto y responde a la fuerte ruidos, como una palmada en las manos. A través del micrófono incorporado, detecta cambios en nivel de decibelios, lo que desencadena un pulso de alta para ser enviado a través de la terminal de señal del sensor.

Este cambio puede ser leído por un pasador de E/S de cualquier microcontrolador Parallax. Este sensor también incluye un potenciómetro incorporado para un fácil ajuste del rango de detección del sensor.

Figura 2. 15 Sensor de Impacto de Sonido

2.5.1 Características

- Rango de detección hasta 3 metros de distancia.
- Incorpora un potenciómetro que proporciona un rango ajustable de la detección.
- Salida de un solo bit

- 3-pin SIP encabezado listo para protoboard o proyectos a través del agujero
- Capacidad de la resistencia en serie para ser compatible con el microcontrolador de la hélice y otros dispositivos de 3,3 V

2.5.2 Ideas de Aplicación

- Sistemas de Alarmas activados por ruidos.
- Accesorios animados de Vacaciones
- Los sistemas de vigilancia

2.5.3 Especificaciones Claves

- Requisitos de alimentación: 5 VDC.
- Comunicación: Bit único de alta/baja de salida.
- Dimensiones: 0.6 x 1.5 pulgadas (15 x 38 mm).
- Temperatura de funcionamientos: 32 a 158 °F (0 a +70 °C).

2.5.4 Definición de pines

Pin	Nombre	Función
1	GND	Tierra
2	5V	5 Voltios VDC
3	SIG	Señal del Pin

Tabla 2. 12 Definición de pines

2.5.5 Diagrama de Conexiones

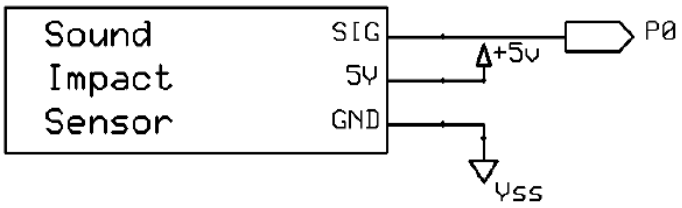

Figura 2. 16 Diagrama de Conexiones

2.5.6 Sensibilidad

El sensor de impacto de sonido tiene un alcance máximo de detección de 3 metros. Sin embargo, si usted va a utilizar este sensor en un área donde los factores ambientales pueden desencadenar lecturas falsas, el intervalo puede acortarse por ajustando el potenciómetro en la parte delantera de la tabla.

CAPÍTULO 3

3. EJERCICIOS PREVIOS Y REALIZACIÓN DEL PROYECTO

En esta etapa se describen los diferentes modos de operación de los elementos que conforman los ejercicios y su funcionamiento en conjunto para la aplicación implementada.

3.1. Comunicación SPI maestro-esclavo entre dos LPC1769.

3.1.1 Descripción

Una de las tarjetas LPC1769 trabaja como el maestro de la comunicación serial y muestra en un display de siete segmentos los números del 0 al 9 de manera cíclica, al mismo tiempo que se transmiten estos datos a la otra

tarjeta LPC1769, la cual trabaja como el esclavo de la comunicación mostrando en otro display el numero transmitido.

3.1.2 Diagrama de Bloques

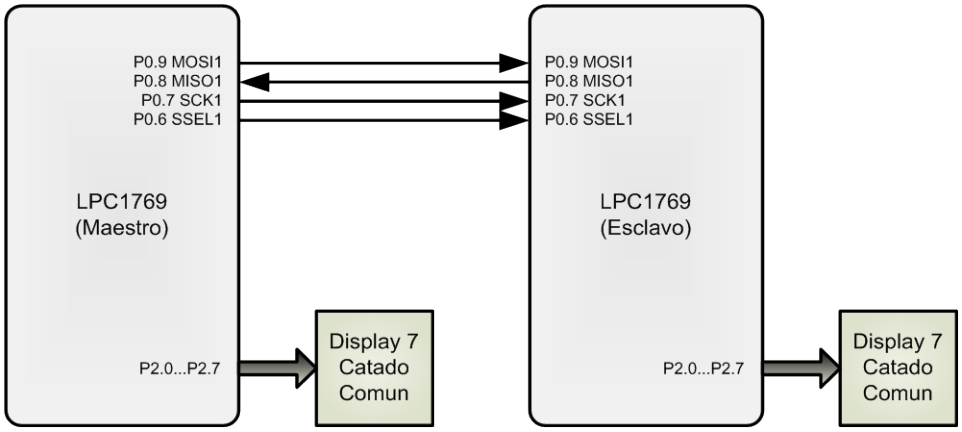

Figura 3. 1 Diagrama de bloques ejercicio 1

3.1.3 Diagrama de Flujo

Maestro

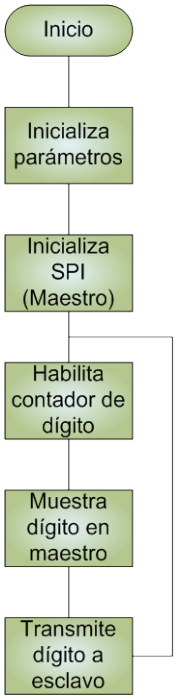

Figura 3. 2 Diagrama de Flujo Maestro Ejercicio 1

Esclavo

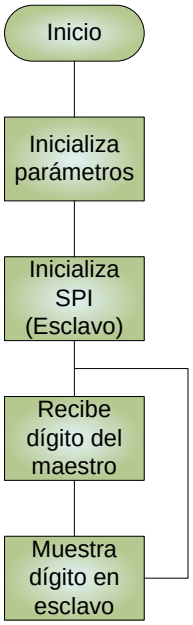

Figura 3. 3 Diagrama de Flujo Esclavo Ejercicio 1

3.1.4 Descripción del Algoritmo

Maestro

1. Se inicializan las variables y los puertos a utilizar.
2. Inicializamos el protocolo SPI.
3. Habilitamos el contador del digito que ira incrementando de 0 al 9 de manera cíclica.
4. Se muestra el digito transmitido en un display de 7 segmentos cátodo común.

5. Se procede a transmitir el digito hacia el esclavo y repetimos el ciclo.

Esclavo

1. Se inicializan las variables y los puertos a utilizar.
2. Inicializamos el protocolo SPI.
3. Recibimos el digito enviado por el maestro.
4. Se muestra el digito recibido en un display de 7 segmentos cátodo común.

3.1.5 Programa Principal del Controlador Maestro

```
/*****************************************************************
                    MICRO CONTROLADORES  AVANZADOS
 *****************************************************************
                        Comunicación Serial SPI
 *****************************************************************/
                          Maestro Ejercicio 1
*Nombre: Contador cíclico de 0 a 9
*Descripción: Código maestro genera, muestra en un display de 7
segmentos y transmite numeros del cero al nueve hacia el esclavo
 *****************************************************************/
  #include <cr_section_macros.h>
  #include <NXP/crp.h>
  __CRP const unsigned int CRP_WORD = CRP_NO_CRP ;
  #include "LPC17xx.h"/* LPC13xx definitions */
  #include "ssp.h"
  /*Tener cuidado con el número del Puerto
y la localización del número porque alguna de las direcciones podría
n no existir en ese puerto. */
  #define PORT_NUM            1
  #define LOCATION_NUM        0
  #define LED     LPC_GPIO2
  uint8_t src_addr[SSP_BUFSIZE];
  uint8_t dest_addr[SSP_BUFSIZE];
  static const uint8_t segmentLUT[10] =
  {
            //ABCDEFGP
            (uint8_t) 0b00111111,
            (uint8_t) 0b00000110,
            (uint8_t) 0b01011011,
```

```
            (uint8_t) 0b01001111,
            (uint8_t) 0b01100110,
            (uint8_t) 0b01101101,
            (uint8_t) 0b01111101,
            (uint8_t) 0b00000111,
            (uint8_t) 0b01111111,
            (uint8_t) 0b01100111,
  };
/*****************************************************************
                        Main Function  main()
*****************************************************************/
  int main (void)
  {
    volatile int timeKeeper=0;
    uint32_t j, portnum = PORT_NUM;
    uint32_t i = 0;

    /* Actualiza la variable de frecuencia del sistema */
    SystemClockUpdate();
    LED -> FIODIR = 0xFF;
    if ( portnum == 0 )
      SSP0Init(); /* initialize SSP port */
    else if ( portnum == 1 )
      SSP1Init();
    while ( 1 )
    {
      for(j = 10000000; j > 0; j--);
      LED->FIOCLR = 0xFF;
      src_addr[0] = (uint8_t)segmentLUT[i];
      LED->FIOSET=src_addr[0];
      i=i+1;
      if (i==10) i=0;

  #if TX_RX_ONLY

/* Para la comunicación interna de la tarjeta esta  deberá configura
rse como maestro o esclavo.*/
  #if SSP_SLAVE
    /* Esclavo recibe */
    SSPReceive( portnum, (uint8_t *)dest_addr, SSP_BUFSIZE );
    for ( i = 0; i < SSP_BUFSIZE; i++ )
    {
      if ( src_addr[i] != dest_addr[i] )
      {
        while ( 1 );/* Verification failure, fatal error */
      }
    }
  #else
    /* Maestro transmite */
    SSPSend( portnum, (uint8_t *)src_addr, 1);
  #endif
    return 0;
  }
/*****************************************************************
```

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

3.1.6 Programa Principal del Controlador Esclavo

```
/*****************************************************************
                    MICRO CONTROLADORES  AVANZADOS
 *****************************************************************
                        Comunicación Serial SPI
 *****************************************************************/
                          Esclavo Ejercicio 1
*Nombre: Contador cíclico de 0 a 9
*Descripción: Código maestro recibe y muestra en un display de 7
segmentos numeros del cero al nueve que son enviados por el maestro
 *****************************************************************/
#include <cr_section_macros.h>
#include <NXP/crp.h>
__CRP const unsigned int CRP_WORD = CRP_NO_CRP ;
#include "LPC17xx.h" /* LPC13xx definitions */
#include "ssp.h"
  /* Tener cuidado con el número del Puerto y la localización del
número porque alguna de las direcciones podrían no existir en ese
puerto. */
#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()
*****************************************************************/
int main (void)
{
  uint32_t portnum = PORT_NUM;
/* Actualiza la variable de frecuencia del sistema */
  SystemClockUpdate();
  LED -> FIODIR = 0xFF;
  if ( portnum == 0 )
      SSP0Init();            /* initialize SSP port */
  else if ( portnum == 1 )
  while ( 1 )
  {
        SSP1Init();
      dest_addr[0] = 0;
#if TX_RX_ONLY
   /* Para la comunicación interna de la tarjeta esta  deberá
configurarse como maestro o esclavo.*/
#if SSP_SLAVE
/* Esclavo recibe */
```

```
      SSPReceive( portnum, (uint8_t *)dest_addr, 1);
      LED -> FIOCLR = 0xFF;
       if (dest_addr[0]!= 0xFF){
      LED -> FIOSET =  dest_addr[0];
  }
#else
  /* Maestro transmite */
  SSPSend( portnum, (uint8_t *)src_addr, SSP_BUFSIZE);
#endif
        }
  return 0;
}
/******************************************************************************
                                End Of File
******************************************************************************/
```

3.1.7 Conclusiones

Por medio de la práctica en este ejercicio se ha podido establecer la comunicación SPI entre dos tarjetas LPC1769 transmitiendo de la tarjeta maestro a la tarjeta esclavo números del 0 al 9 de manera cíclica y mostrarla en un display de 7 segmentos.

3.2. Comunicación SPI entre LPC1769 y AVR Butterfly.

3.2.1 Descripción

El maestro LPC1769 muestra y envía mediante comunicación SPI, los números del 0 al 9 de manera cíclica al AVR Butterfly que esta definido como esclavo, los cuales se mostraran en un display de siete segmentos.

3.2.2 Diagrama de Bloques

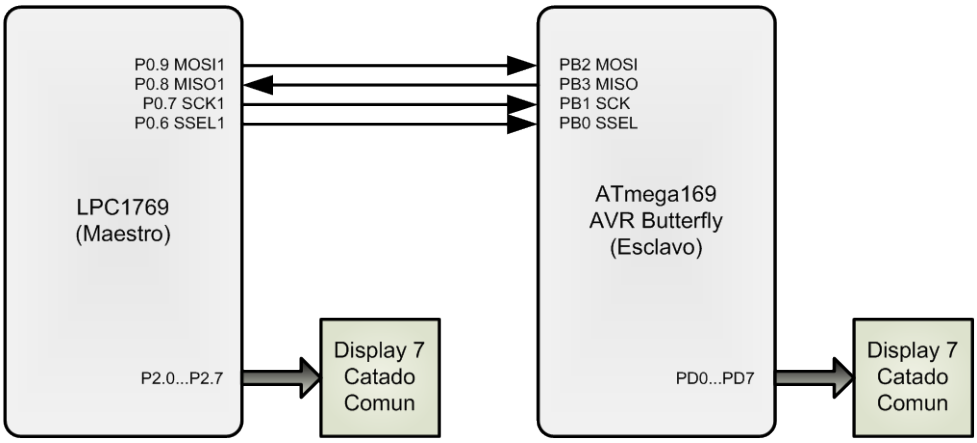

Figura 3. 4 Diagrama de bloques ejercicio 2

3.2.3 Diagrama de Flujo

Maestro

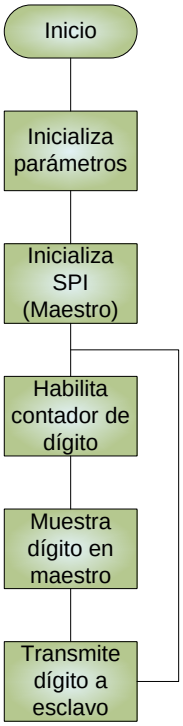

Figura 3. 5 Diagrama de Flujo Maestro Ejercicio 2

Esclavo

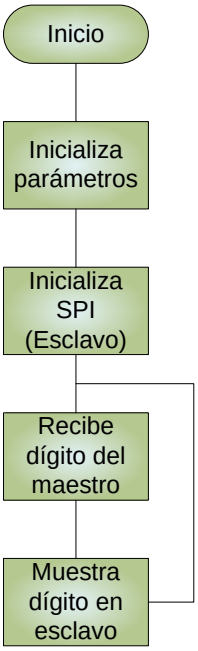

Figura 3. 6 Diagrama de Flujo Esclavo Ejercicio 2

3.2.4 Descripción del Algoritmo

Maestro

1. Se inicializan las variables y los puertos a utilizar.
2. Inicializamos el protocolo SPI.
3. Habilitamos el contador del digito que ira incrementando de 0 al 9 de manera cíclica.
4. Se muestra el digito transmitido en un display de 7 segmentos cátodo común.
5. Se procede a transmitir el digito hacia el esclavo y repetimos el ciclo.

Esclavo

1. Se inicializan las variables y los puertos a utilizar.
2. Inicializamos el protocolo SPI.
3. Recibimos el digito enviado por el maestro.
4. Se muestra el digito recibido en un display de 7 segmentos cátodo común.

3.2.5 Programa Principal del Controlador Maestro

```
/*****************************************************************
                    MICRO CONTROLADORES  AVANZADOS
 *****************************************************************
                        Comunicación Serial SPI
 *****************************************************************/
                           Maestro Ejercicio 2
*Nombre: Contador cíclico de 0 a 9
```

```
*Descripción: Código maestro genera, muestra en un display de 7
segmentos y transmite numeros del cero al nueve hacia el esclavo
 ****************************************************************/
  #include <cr_section_macros.h>
  #include <NXP/crp.h>
  __CRP const unsigned int CRP_WORD = CRP_NO_CRP ;
  #include "LPC17xx.h"/* LPC13xx definitions */
  #include "ssp.h"
  /*Tener cuidado con el número del Puerto
y la localización del número porque alguna de las direcciones podría
n no existir en ese puerto. */
  #define PORT_NUM           1
  #define LOCATION_NUM       0
  #define LED     LPC_GPIO2
  uint8_t src_addr[SSP_BUFSIZE];
  uint8_t dest_addr[SSP_BUFSIZE];
  static const uint8_t segmentLUT[10] =
  {
            //ABCDEFGP
            (uint8_t) 0b00111111,
            (uint8_t) 0b00000110,
            (uint8_t) 0b01011011,
            (uint8_t) 0b01001111,
            (uint8_t) 0b01100110,
            (uint8_t) 0b01101101,
            (uint8_t) 0b01111101,
            (uint8_t) 0b00000111,
            (uint8_t) 0b01111111,
            (uint8_t) 0b01100111,
  };
/*****************************************************************
                        Main Function  main()
*****************************************************************/
  int main (void)
  {
    volatile int timeKeeper=0;
    uint32_t j, portnum = PORT_NUM;
    uint32_t i = 0;

    /* Actualiza la variable de frecuencia del sistema */
    SystemClockUpdate();
    LED -> FIODIR = 0xFF;
    if ( portnum == 0 )
      SSP0Init(); /* initialize SSP port */
    else if ( portnum == 1 )
      SSP1Init();
    while ( 1 )
    {
      for(j = 10000000; j > 0; j--);
      LED->FIOCLR = 0xFF;
      src_addr[0] = (uint8_t)segmentLUT[i];
      LED->FIOSET=src_addr[0];
      i=i+1;
      if (i==10) i=0;
```

```
  #if TX_RX_ONLY

/* Para la comunicación interna de la tarjeta esta  deberá configura
rse como maestro o esclavo.*/
  #if SSP_SLAVE
    /* Esclavo recibe */
    SSPReceive( portnum, (uint8_t *)dest_addr, SSP_BUFSIZE );
    for ( i = 0; i < SSP_BUFSIZE; i++ )
    {
      if ( src_addr[i] != dest_addr[i] )
      {
        while ( 1 );/* Verification failure, fatal error */
      }
    }
  #else
    /* Maestro transmite */
    SSPSend( portnum, (uint8_t *)src_addr, 1);
  #endif
    return 0;
  }
/******************************************************************
                              End Of File
******************************************************************/
```

3.2.6 Programa Principal del Controlador Esclavo

```
/*****************************************************************************
                        MICRO CONTROLADORES  AVANZADOS
*****************************************************************************
                            Comunicación Serial SPI
*****************************************************************************/
                            Esclavo Ejercicio 2
*Nombre: Contador cíclico de 0 a 9
*Descripción: Código maestro recibe y muestra en un display de 7
segmentos numeros del cero al nueve que son enviados por el maestro
 ****************************************************************/
#include <avr/io.h>
#include <util/delay.h>
// Funciones:
void SPI_slave_init(void);
unsigned char SPI_slave_receive(void);
// Programa Principal:
int main(void)
{
  unsigned char rx_data;
  SPI_slave_init(); // inicializa SPI
  DDRD=0xFF;  // PORTD como salida
  PORTD=0x00;
  while(1)
  {
```

```
    rx_data=SPI_slave_receive(); // recibe dato transmitido
    PORTD= rx_data; // muestra número
  }
}
/*************************************************************************/
void SPI_slave_init(void)
{
  DDRB=0x08; // MISO como salida y el resto como entrada
  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
}
/*************************************************************************
                                End Of File
*************************************************************************/
```

3.2.7 Conclusiones

Por medio de la práctica se ha podido establecer la comunicación SPI entre una tarjeta LPC1769 como maestro y una tarjeta AVR butterfly como esclavo para transmitir números del 0 al 9 de manera cíclica y mostrarla en un display de 7 segmentos.

3.3. Comunicación SPI entre AVR Butterfly y LPC1769.

3.3.1 Descripción

El microcontrolador Atmega169 del AVR Butterfly funciona como el maestro de la comunicación SPI, muestra y envía los números del 0 al 9 de manera

cíclica a la tarjeta LPC1769 que esta definida como esclavo, los cuales se mostraran en un display de siete segmentos.

3.3.2 Diagrama de Bloques

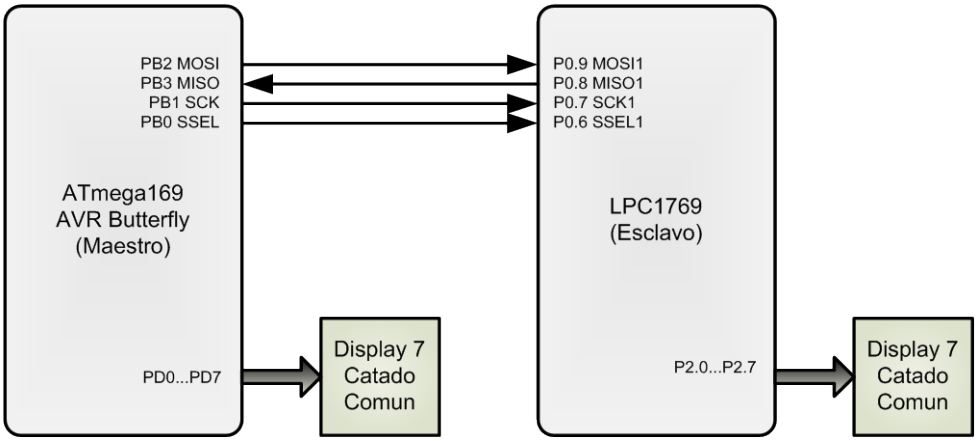

Figura 3. 7 Diagrama de bloques ejercicio 3

3.3.3 Diagrama de Flujo

Maestro

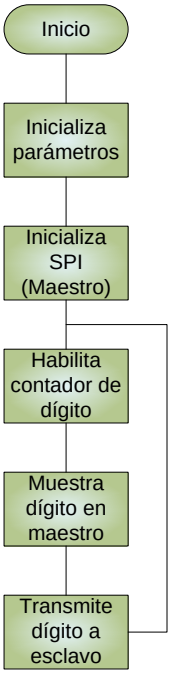

Figura 3. 8 Diagrama de Flujo Maestro Ejercicio 3

Esclavo

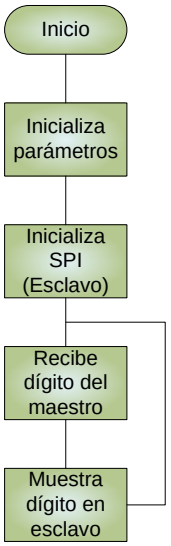

Figura 3. 9 Diagrama de Flujo Esclavo Ejercicio 3

3.3.4 Descripción del Algoritmo

Maestro

1. Se inicializan las variables y los puertos a utilizar.
2. Inicializamos el protocolo SPI.
3. Habilitamos el contador del digito que ira incrementando de 0 al 9 de manera cíclica.
4. Se muestra el digito transmitido en un display de 7 segmentos cátodo común.
5. Se procede a transmitir el digito hacia el esclavo y repetimos el ciclo.

Esclavo

1. Se inicializan las variables y los puertos a utilizar.
2. Inicializamos el protocolo SPI.
3. Recibimos el digito enviado por el maestro.
4. Se muestra el digito recibido en un display de 7 segmentos cátodo común.

3.3.5 Programa Principal del Controlador Maestro

```
/*****************************************************************************
                        MICRO CONTROLADORES  AVANZADOS
*****************************************************************************
```

```
                        Comunicación Serial SPI
*****************************************************************/
                         Maestro Ejercicio 3
*Nombre: Contador cíclico de 0 a 9
*Descripción: Código maestro genera, muestra en un display de 7
segmentos y transmite numeros del cero al nueve hacia el esclavo
 *****************************************************************/
#include <avr/io.h>
#include <util/delay.h>
#include <avr/pgmspace.h>
#include <avr/signal.h>
#include <avr/interrupt.h>
// tabla de conversión de binario a 7 segmentos
unsigned int numeros[] = {
       0b00111111,
       0b00000110,
       0b01011011,
       0b01001111,
       0b01100110,
       0b01101101,
       0b01111101,
       0b00000111,
       0b01111111,
       0b01100111,
              };
// Variables globales:
unsigned int uni, unid;
void SPI_masterinit(void)
{
  DDRB=0x07;    // MOSI, SS y SCK como salida
  // SPI enable, dispositivo MASTER, Fosc/4
  SPCR=(1<<SPE)|(1<<MSTR);
}
void SPI_master_transmit(unsigned char cdata)
{
 SPDR=cdata; // coloca dato a enviar en el registro SPDR
 while(!(SPSR & (1<<SPIF))); // espera mientras se completa la transmisión
}
int main(void)
{
  SPI_masterinit();
  DDRB=0x0F;
  PORTB=0xF0;
  PORTB=0;
  DDRD=0xFF;  // PORTD como salida
  PORTD=0x00;

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

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

```
   _delay_ms(500);
      uni++;
      if (uni==0x0A)
        {
                  uni=0;
         }
   }
      unid=numeros[uni];  // conversión binario
      PORTD=unid;         // muestra en display
      SPI_master_transmit(unid);  // transmite dato
 }
}
/*****************************************************************************
                                End Of File
*****************************************************************************/
```

3.3.6 Programa Principal del Controlador Esclavo

```
/*****************************************************************
                    MICRO CONTROLADORES  AVANZADOS
 *****************************************************************
                        Comunicación Serial SPI
 *****************************************************************/
                          Esclavo Ejercicio 3
*Nombre: Contador cíclico de 0 a 9
*Descripción: Código maestro recibe y muestra en un display de 7
segmentos numeros del cero al nueve que son enviados por el maestro
 *****************************************************************/
#include <cr_section_macros.h>
#include <NXP/crp.h>
__CRP const unsigned int CRP_WORD = CRP_NO_CRP ;
#include "LPC17xx.h" /* LPC13xx definitions */
#include "ssp.h"
  /* Tener cuidado con el número del Puerto y la localización del
número porque alguna de las direcciones podrían no existir en ese
puerto. */
#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()
*****************************************************************/
int main (void)
{
  uint32_t portnum = PORT_NUM;
/* Actualiza la variable de frecuencia del sistema */
  SystemClockUpdate();
```

```
  LED -> FIODIR = 0xFF;
  if ( portnum == 0 )
      SSP0Init();             /* initialize SSP port */
  else if ( portnum == 1 )
  while ( 1 )
  {
        SSP1Init();
      dest_addr[0] = 0;
#if TX_RX_ONLY
   /* Para la comunicación interna de la tarjeta esta  deberá
configurarse como maestro o esclavo.*/
#if SSP_SLAVE
/* Esclavo recibe */
      SSPReceive( portnum, (uint8_t *)dest_addr, 1);
      LED -> FIOCLR = 0xFF;
       if (dest_addr[0]!= 0xFF){
      LED -> FIOSET =  dest_addr[0];
  }
#else
  /* Maestro transmite */
  SSPSend( portnum, (uint8_t *)src_addr, SSP_BUFSIZE);
#endif
        }
  return 0;
}
/*****************************************************************
                            End Of File
*****************************************************************/
```

3.3.7 Conclusiones

Por medio de la práctica se ha podido establecer la comunicación SPI entre una tarjeta AVR Butterfly como maestro y una tarjeta LPC1769 como esclavo para transmitir números del 0 al 9 de manera cíclica y mostrarla en un display de 7 segmentos.

3.4. Adquisición datos de vibración de un motor BLDC utilizando comunicación SPI.

3.4.1 Descripción

El microcontrolador Atmega169 del AVR Butterfly funciona como el maestro de la comunicación SPI, a él se encuentra conectado el sensor de impactos de sonido que se encarga de sensar las vibraciones del motor BLDC para luego ser interpretada por la tarjeta AVR Butterfly y finalmente enviada hacia la tarjeta LPC1769 que esta definida como esclavo, el cual mostrara en un display de siete segmentos el numero de interrupciones que el sensor detecte.

3.4.2 Diagrama de Bloques

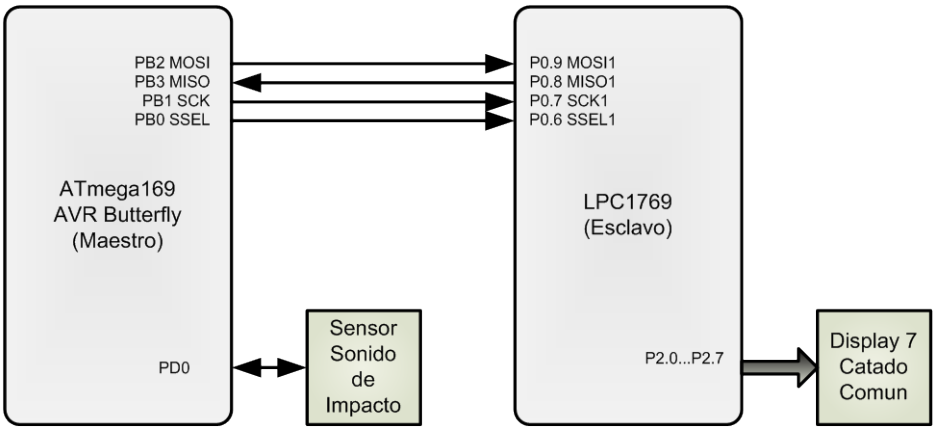

Figura 3. 10 Diagrama de bloques proyecto

3.4.3 Diagrama de Flujo

Maestro

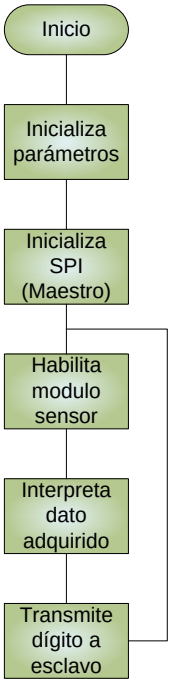

Figura 3. 11 Diagrama de Flujo Maestro Proyecto

Esclavo

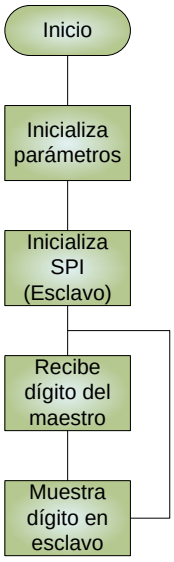

Figura 3. 12 Diagrama de Flujo Esclavo Proyecto

3.4.4 Descripción del Algoritmo

Maestro

1. Se inicializan las variables y los puertos a utilizar.
2. Inicializamos el protocolo SPI.
3. Habilitamos el contador del digito que ira incrementando de 0 al 9 de manera cíclica.
4. Se muestra el digito transmitido en un display de 7 segmentos cátodo común.
5. Se procede a transmitir el digito hacia el esclavo y repetimos el ciclo.

Esclavo

1. Se inicializan las variables y los puertos a utilizar.
2. Inicializamos el protocolo SPI.
3. Recibimos el digito enviado por el maestro.
4. Se muestra el digito recibido en un display de 7 segmentos cátodo común.

3.4.5 Programa Principal del Controlador Maestro

```
/*****************************************************************************
                          MICRO CONTROLADORES  AVANZADOS
*****************************************************************************
```

```
                        Comunicación Serial SPI
*************************************************************************/
                    Maestro Ejercicio del Proyecto
*Nombre: Adquisición de datos de vibración de un motor BLDC
*Descripción: El código maestro se encarga de tomar la señal del
sensor e interpretarla generando un número que va del cero al nueve
y enviarlo mediante comunicación SPI a la tarjeta esclavo
 *************************************************************************/
#include <avr/io.h>
#include <util/delay.h>
#include <avr/pgmspace.h>
#include <avr/signal.h>
#include <avr/interrupt.h>
#include <stdlib.h>

// tabla de conversión de binario a 7 segmentos
unsigned int numeros[] = {
       0b00111111,
       0b00000110,
       0b01011011,
       0b01001111,
       0b01100110,
       0b01101101,
       0b01111101,
       0b00000111,
       0b01111111,
       0b01100111,
              };

// Variables globales:
unsigned int uni, unid;


void SPI_masterinit(void)
{
  DDRB=0x07;    // MOSI, SS y SCK como salida
  // SPI enable, dispositivo MASTER, Fosc/2
  SPCR=(1<<SPE)|(1<<MSTR);
  //SPSR=(1<<SPI2X);
}
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
}
int main(void)
{
DDRD=0xFE;
SPI_masterinit();
while(1)
{
       if(PIND)
       {
              uni++;
              if (uni==0x0A)
```

```
        {
                uni=0;
        }
              unid=numeros[uni];  // conversión binario
              SPI_master_transmit(unid);  // transmite dato
              _delay_ms(500);
       }
}
}
/*****************************************************************************
                                End Of File
*****************************************************************************/
```

3.4.6 Programa Principal del Controlador Esclavo

```
/*****************************************************************
                    MICRO CONTROLADORES  AVANZADOS
 *****************************************************************
                        Comunicación Serial SPI
 *****************************************************************/
                    Esclavo Ejercicio del Proyecto
*Nombre: Adquisición de datos de vibración de un motor BLDC
*Descripción: Recibe los datos que le han sido transmitidos y los
muestra en un display de 7 segmentos.
 *****************************************************************/
#include <cr_section_macros.h>
#include <NXP/crp.h>
__CRP const unsigned int CRP_WORD = CRP_NO_CRP ;
#include "LPC17xx.h" /* LPC13xx definitions */
#include "ssp.h"
  /* Tener cuidado con el número del Puerto y la localización del
número porque alguna de las direcciones podrían no existir en ese
puerto. */
#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()
*****************************************************************/
int main (void)
{
  uint32_t portnum = PORT_NUM;
/* Actualiza la variable de frecuencia del sistema */
  SystemClockUpdate();
  LED -> FIODIR = 0xFF;
  if ( portnum == 0 )
      SSP0Init();             /* initialize SSP port */
```

```
  else if ( portnum == 1 )
  while ( 1 )
  {
        SSP1Init();
      dest_addr[0] = 0;
#if TX_RX_ONLY
   /* Para la comunicación interna de la tarjeta esta  deberá
configurarse como maestro o esclavo.*/
#if SSP_SLAVE
/* Esclavo recibe */
      SSPReceive( portnum, (uint8_t *)dest_addr, 1);
      LED -> FIOCLR = 0xFF;
       if (dest_addr[0]!= 0xFF){
      LED -> FIOSET =  dest_addr[0];
  }
#else
  /* Maestro transmite */
  SSPSend( portnum, (uint8_t *)src_addr, SSP_BUFSIZE);
#endif
        }
  return 0;
}
/******************************************************************
                              End Of File
******************************************************************/
```

3.3.1 Conclusiones

Por medio de la práctica se ha podido obtener información de la vibración de un motor BLDC usando un sensor de impactos de sonido, estos datos son tomados por la tarjeta AVR Butterfly que se encuentra funcionando como maestro y los transmite a la tarjeta LPC1769 que se encuentra como esclavo y finalmente es mostrado en un display de 7 segmentos para su visualización.

CAPÍTULO 4

4. IMPLEMENTACIÓN DE EJERCICIOS Y DEL PROYECTO

4.1. Plataforma del Proyecto

El conjunto de ejercicios que se especifican más adelante estará compuesto de varios elementos, muchos de los cuales se les da el mismo uso en todos los ejercicios y son detallados a continuación:

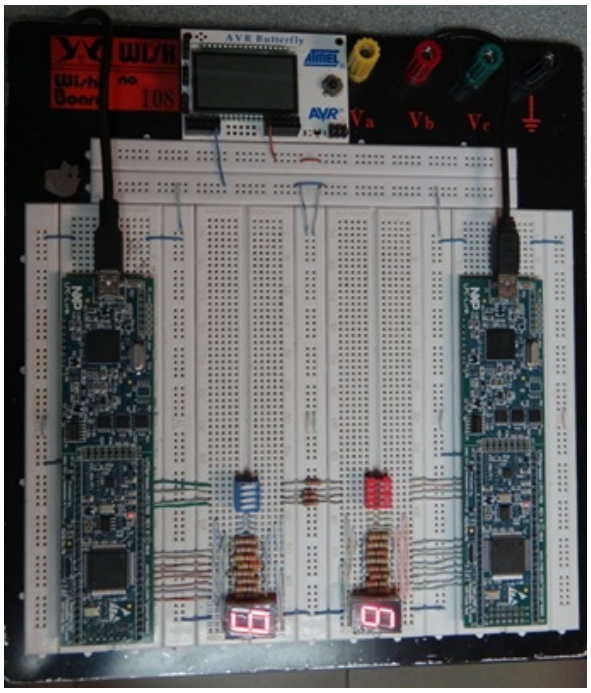

Figura 4. 1 Plataforma del proyecto

4.1.1 Protoboard

El Protoboard es una placa de uso genérico reutilizable o semipermanente, usado para construir prototipos de circuitos electrónicos con o sin soldadura.

Normalmente se utilizan para la realización de pruebas experimentales, como en nuestro caso.

Figura 4. 2 Protoboard de 4 regletas

Está compuesto por bloques de plástico perforados y numerosas láminas delgadas, de una aleación de cobre, estaño y fósforo, que unen dichas perforaciones, creando una serie de líneas de conducción paralelas. Las líneas se cortan en la parte central del bloque de plástico para garantizar que dispositivos en circuitos integrados tipo DIP (Dual Inline Packages) puedan ser insertados perpendicularmente a las líneas de conductores.

Debido a las características de capacitancia (de 2 a 30 pF por punto de contacto) y resistencia que suelen tener los protoboard están confinados a

trabajar a relativamente baja frecuencia (inferior a 10 ó 20 MHz, dependiendo del tipo y calidad de los componentes electrónicos utilizados).

Los demás componentes electrónicos pueden ser montados sobre perforaciones adyacentes que no compartan la tira o línea conductora, e interconectados a otros dispositivos usando cables, usualmente unifilares.

4.1.2 AVR Butterfly

El AVR Butterfly es un Kit de desarrollo, entrenamiento y aprendizaje de microcontroladores Atmel que contiene el microcontrolador Atmega169, en el cual se programarán los códigos de los proyectos a realizar.

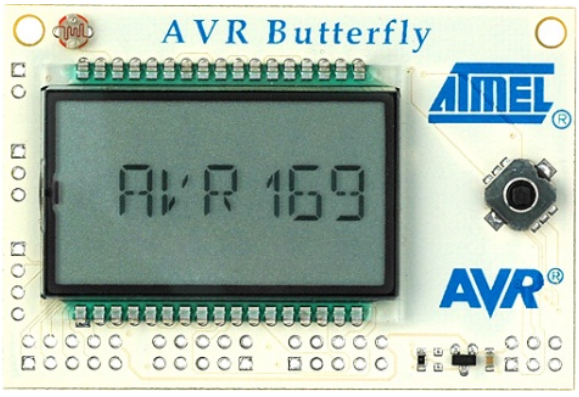

Figura 4. 3 AVR Butterfly ATmega 169

4.1.3 LPCXpresso LPC1769

El LPC1769 es un ARM Cortex-M3 basado en microcontrolador para aplicaciones embebidas requieren un alto nivel de integración y baja disipación de potencia. El procesador ARM Cortex-M3 es un núcleo de próxima generación que ofrece mejoras en el sistema de depuración, tales como modernización características y un mayor nivel de apoyo a la integración de bloques.

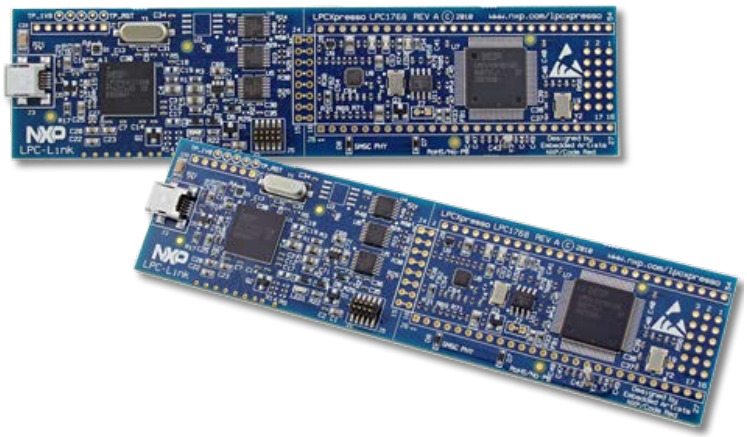

Figura 4. 4 Tarjeta LPCXpresso LPC1769

4.1.4 Pilas

Una pila eléctrica es un dispositivo que convierte energía química en energía eléctrica por un proceso químico transitorio. Cada uno de los proyectos utilizará una batería de litio de 3,6V para el del AVR Butterfly, ya que las tarjetas LPC1769 utilizaran la alimentación USB del computador.

Figura 4. 5 Batería de Litio ER14250/W 3.6V

4.2. Descripción de los Ejercicios

A continuación se describen detalladamente uno a uno los ejercicios del proyecto que involucran comunicaciones seriales SPI dedicado al trabajo con microcontroladores Atmel y LPCXpresso con aplicaciones específicas.

4.2.1 Comunicación SPI maestro-esclavo entre dos LPC1769.

- **Descripción:**

Una de las tarjetas LPC1769 trabaja como el maestro de la comunicación serial y muestra en un display de siete segmentos los números del 0 al 9 de manera cíclica, al mismo tiempo que se transmiten estos datos a la otra tarjeta LPC1769, la cual trabaja como el esclavo de la comunicación mostrando en otro display el numero transmitido.

- **Lista de Materiales:**

- Dos Displays de siete segmentos
- Dos tarjetas LPC1769.

- Cuatro resistencias de 1 K Ω.
- Dieciséis resistencias de 330 Ω.

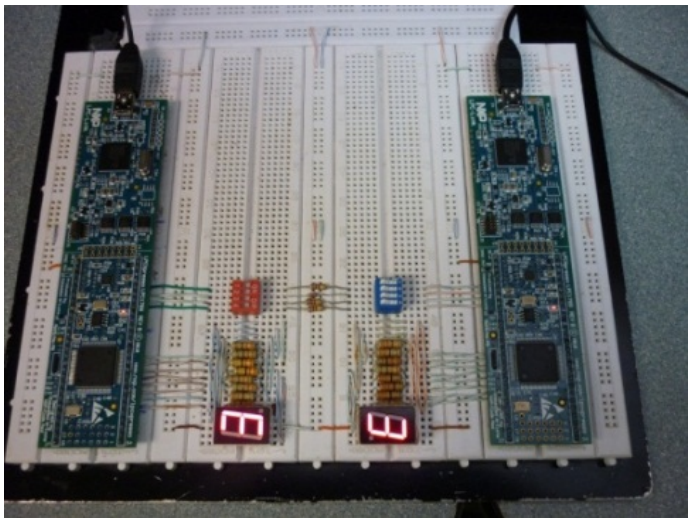

Figura 4. 6 Implementación ejercicio 1

4.2.2 Comunicación SPI entre LPC1769 y AVR Butterfly.

- **Descripción:**

El maestro LPC1769 muestra y envía mediante comunicación SPI, los números del 0 al 9 de manera cíclica al AVR Butterfly que esta definido como esclavo, los cuales se mostraran en un display de siete segmentos.

- **Lista de Materiales:**
- Una tarjeta LPC1769.
- Un AVR Butterfly.
- Cuatro resistencias de 1 K Ω.
- Dieciséis resistencias de 330 Ω.

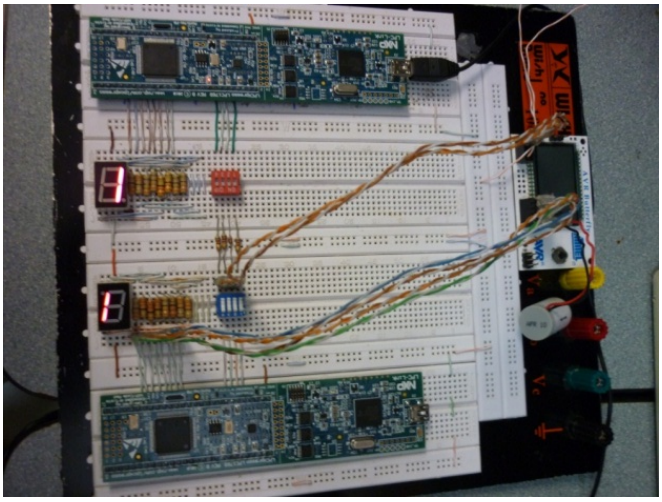

Figura 4. 7 Implementación ejercicio 2

4.2.3 Comunicación SPI entre AVR Butterfly y LPC1769.

- **Descripción:**

El microcontrolador Atmega169 del AVR Butterfly funciona como el maestro de la comunicación SPI, muestra y envía los números del 0 al 9 de manera cíclica a la tarjeta LPC1769 que esta definida como esclavo, los cuales se mostraran en un display de siete segmentos.

- **Lista de Materiales:**
- Una tarjeta LPC1769.
- Un AVR Butterfly.
- Cuatro resistencias de 1 K Ω.
- Dieciséis resistencias de 330 Ω.

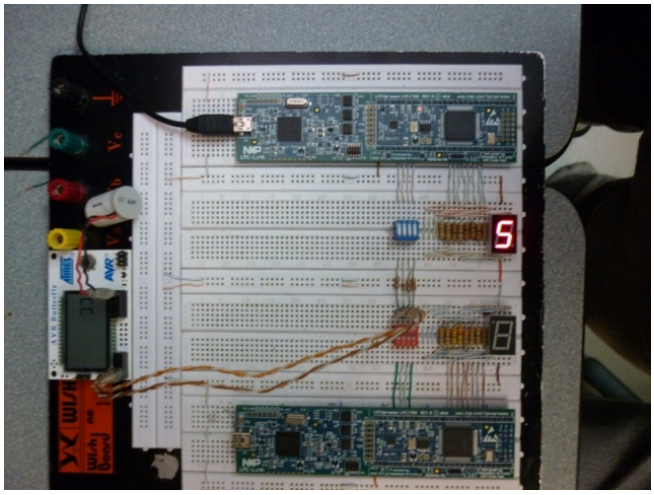

Figura 4. 8 Implementación ejercicio 3

4.2.4 Adquisición datos de vibración de un motor BLDC utilizando comunicación SPI.

- **Descripción:**

El microcontrolador Atmega169 del AVR Butterfly funciona como el maestro de la comunicación SPI, muestra y envía los números del 0 al 9 de manera cíclica a la tarjeta LPC1769 que esta definida como esclavo, los cuales se mostraran en un display de siete segmentos.

- **Lista de Materiales:**
- Una tarjeta LPC1769.
- Un AVR Butterfly.
- Cuatro resistencias de 1 K Ω.
- Dieciséis resistencias de 330 Ω.

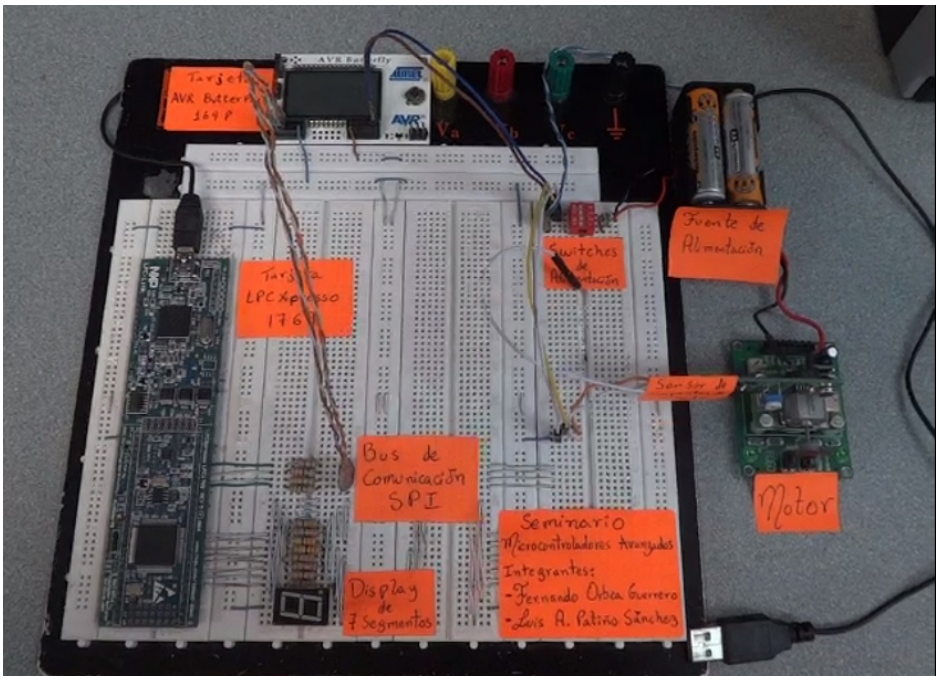

Figura 4. 9 Implementación del proyecto

CONCLUSIONES

1. En esta plataforma de trabajo se ha utilizado el kit de desarrollo AvrButterfly herramienta muy util para el desarrollo de aplicaciones realizadas en microcontroladores conjuntamente con la tarjeta LPC1769 que tiene prestaciones similares a la mencionada anteriormente, haciendo uso del protocolo de comunicación SPI que ambos microcontroladores lo tienen implementado, se ha podido transmitir datos entre estos dos dispositivos.

2. Se ha complementado la utilización de recursos, que estos microcontroladores ofrecen para obtener buenos resultados en los ejercicios de prueba detallados en este documento, que se han realizado con la finalidad de introducir e incentivar al nuevo programador, a realizar otras aplicaciones de intereses común, manteniendo el protocolo de comunicación SPI establecido.

3. Cabe recalcar que por medio de la práctica realizada en esta tesina se ha podido comprobar y demostrar la simplicidad, flexibilidad, entre

otras ventajas que ofrece este tipo de comunicación tanto para su ensamblaje como funcionamiento, además la posibilidad de administrar varios dispositivos, que han de configurarse en modo esclavo, cada cual con una referencia de selección diferente, para que sea identificado por el dispositivo maestro.

4. La adquisición de los datos de vibración del motor BLDC se la realizo haciendo uso del sensor de impacto de sonido de la familia Parallax, el cual envía un alto en la salida de la señal para indicar que se ha recibido un estimulo, se pudo calibrar el potenciómetro de sensibilidad de tal manera que se ajustara a los requerimientos del proyecto.

RECOMENDACIONES

1. Leer los manuales de usuario del kit de desarrollo AvrButterfly y de la tarjeta LPC1769, que podemos encontrar en el internet en las páginas web de sus respectivos fabricantes, poniendo énfasis en el funcionamiento del protocolo de comunicación SPI que es el usado en este proyecto.

2. Descargar el software que utilizan ambos microcontroladores, para conocer el entorno de programación, realizando pequeños ejemplos prácticos que permitan la familiarización y destreza del individuo en el uso de estos programas, luego aprender acerca de las librerías y módulos que se integran también en estos sistemas, y facilitar aun mas el desarrollo de alguna aplicación.

3. Tomar en cuenta el debido cuidado al manipular las tarjetas ya que son tecnología Cmos, susceptibles a la exposición de energía estática, la que podría presentarse en nuestro cuerpo al utilizar las manos, si es

este el caso utilizar pulseras aterrizadas que se encuentran de venta en cualquier electrónica para evitar el daño irreparable de las mismas.

4. Cuando realicemos el ajuste del potenciómetro que tiene el sensor de impacto de sonido, debemos hacerlo con mucho cuidado utilizando herramientas adecuadas ya que podríamos dañar dicho elemento y quedarnos sin la posibilidad de variar su sensibilidad. Colocar la entrada del sensor próximo a la fuente que se desea muestrear si que esto pueda causar daño alguno a ninguna de las partes.

BIBLIOGRAFÍA

[1]. Alfonso Perales, Motores Sin Escobillas

http://tecnica.carbi.net/newpage2.html

Fecha de consulta: 05/03/2012.

[2]. Sapiensman, Los Motores eléctricos

http://www.sapiensman.com/electrotecnia/motor_electrico.htm

Fecha de consulta: 05/03/2012

[3]. Joan Saball, Motores sin escobillas (Brushless)

http://es.scribd.com/doc/52981482/68/Motores-sin-escobillas-Brushless

Fecha de consulta: 05/03/2012

[4]. Microchip, Brushless DC (BLDC) Motor Fundamentals

http://ww1.microchip.com/downloads/en/AppNotes/00885a.pdf

Fecha de consulta: 07/03/2012

[5]. Tecnun, Sensor de Medición de la posición del acelerador

www.tecnun.es/automocion/proyectos/acelerador_electronico/MEMORIA.pdf

Fecha de consulta: 12/03/2012

[6]. ESPE, Diseño De Un Sistema De Control De Aceleración Electrónica

http://repositorio.espe.edu.ec/bitstream/21000/2935/1/T-ESPEL-0703.pdf

Fecha de consulta: 12/03/2012

[7]. Scribd, Serial Peripheral Interface

http://es.scribd.com/doc/80280302/Collection

Fecha de consulta: 12/03/2012

[8]. Wikipedia, Serial Peripheral Interface

http://es.wikipedia.org/wiki/Serial_Peripheral_Interface

Fecha de consulta: 13/03/2012

[9]. Atmel Corporation, AVR Butterfly guía para el usuario

http://www.atmel.com/Images/doc4271.pdf

Fecha de consulta: 05/04/2012

[10]. ESPE, Características del Kit AVR Butterfly en español

repositorio.espe.edu.ec/bitstream/21000/424/1/T-ESPE-014271.pdf

Fecha de consulta: 07/04/2012

[11]. LPCXpresso, Getting Starting LPC1769

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

Fecha de consulta: 09/04/2012

[12]. LPCXpresso, User Manual LPC1769

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

Fecha de consulta: 09/04/2012

[13]. LPCXpresso, Schematic LPC1769

http://ics.nxp.com/support/documents/microcontrollers/pdf/lpcxpresso.lpc1769.schematic.pdf

Fecha de consulta: 09/04/2012

[14]. Parallax, Sound Impact Sensor

http://www.parallax.com/Portals/0/Downloads/docs/prod/sens/29132-SoundImpactSesnor-v1.0.pdf

Fecha de consulta: 14/04/2012

ANEXOS

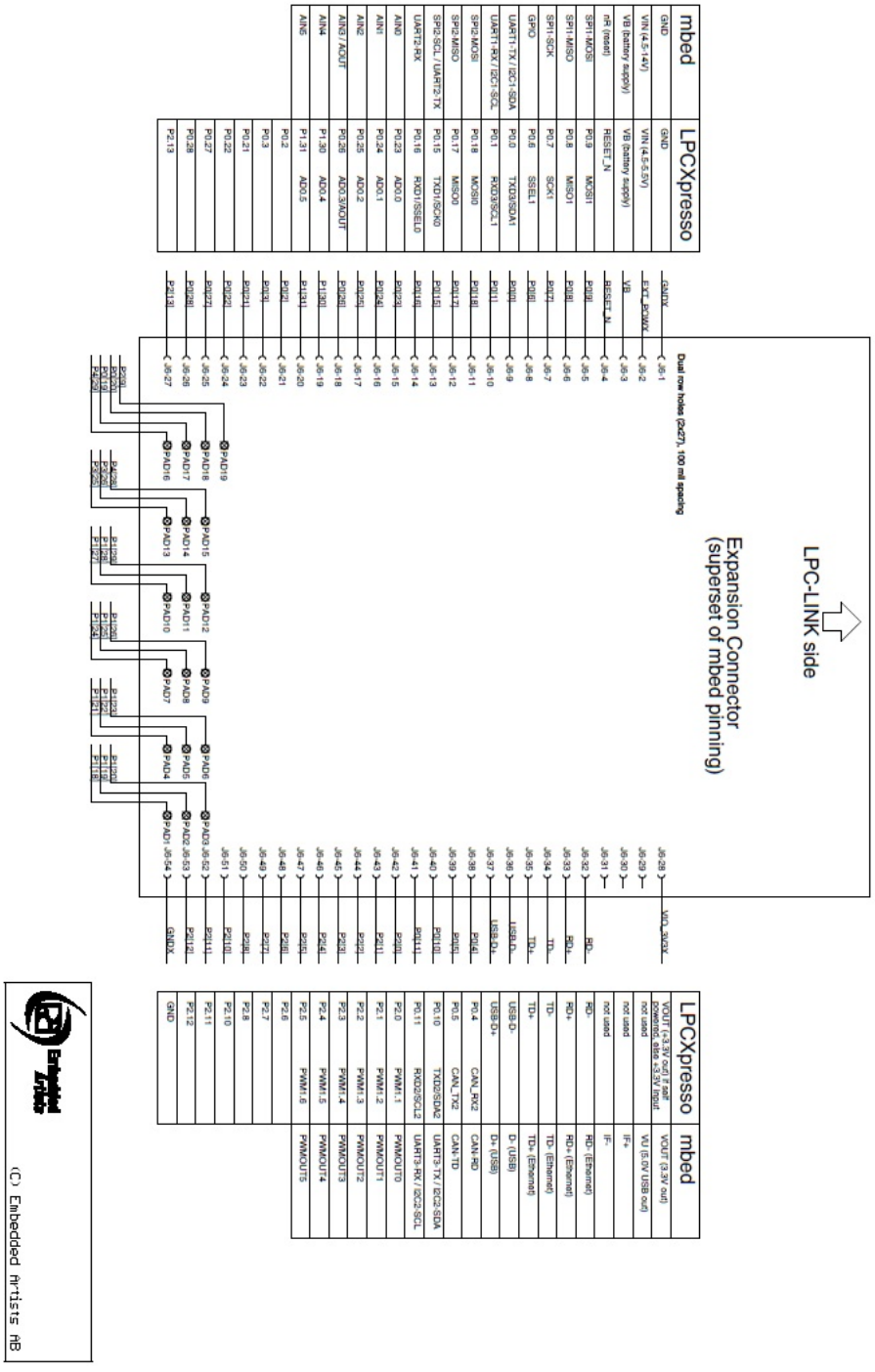

Tabla Anexo 1 Esquemático de pines de tarjeta LPC1769

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

Tabla Anexo 2 Registro PCONP

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

Tabla Anexo 3 Registro PCLKSEL0

Bit	Symbol	Description	Reset Value
7:0	CPSDVSR	This even value between 2 and 254, by which SSP_PCLK is divided to yield the prescaler output clock. Bit 0 always reads as 0.	0
31:8	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Tabla Anexo 4 Registro Pre-escalador del Reloj

Name	Description	Access	Reset Value[1]	Address
PINSEL0	Pin function select register 0.	R/W	0	0x4002 C000
PINSEL1	Pin function select register 1.	R/W	0	0x4002 C004
PINSEL2	Pin function select register 2.	R/W	0	0x4002 C008
PINSEL3	Pin function select register 3.	R/W	0	0x4002 C00C
PINSEL4	Pin function select register 4	R/W	0	0x4002 C010
PINSEL7	Pin function select register 7	R/W	0	0x4002 C01C
PINSEL8	Pin function select register 8	R/W	0	0x4002 C020
PINSEL9	Pin function select register 9	R/W	0	0x4002 C024
PINSEL10	Pin function select register 10	R/W	0	0x4002 C028
PINMODE0	Pin mode select register 0	R/W	0	0x4002 C040
PINMODE1	Pin mode select register 1	R/W	0	0x4002 C044
PINMODE2	Pin mode select register 2	R/W	0	0x4002 C048
PINMODE3	Pin mode select register 3.	R/W	0	0x4002 C04C
PINMODE4	Pin mode select register 4	R/W	0	0x4002 C050
PINMODE5	Pin mode select register 5	R/W	0	0x4002 C054
PINMODE6	Pin mode select register 6	R/W	0	0x4002 C058
PINMODE7	Pin mode select register 7	R/W	0	0x4002 C05C
PINMODE9	Pin mode select register 9	R/W	0	0x4002 C064
PINMODE_OD0	Open drain mode control register 0	R/W	0	0x4002 C068
PINMODE_OD1	Open drain mode control register 1	R/W	0	0x4002 C06C
PINMODE_OD2	Open drain mode control register 2	R/W	0	0x4002 C070
PINMODE_OD3	Open drain mode control register 3	R/W	0	0x4002 C074
PINMODE_OD4	Open drain mode control register 4	R/W	0	0x4002 C078
I2CPADCFG	I^2C Pin Configuration register	R/W	0	0x4002 C07C

Tabla Anexo 5 Registros PinSel