ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

"Banco de pruebas para comunicaciones seriales SPI dedicado al trabajo con microcontroladores Atmel con aplicaciones específicas debidamente documentadas.

TESINA DE SEMINARIO

Previa la obtención del Título de:

INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

Presentado por:

Gema Stefanía Solórzano Aguilar

Raúl Antonio Serrano Mena

GUAYAQUIL – ECUADOR

AÑO 2012

AGRADECIMIENTO

A Dios.

A la familia.

A todas las personas que apoyaron en el desarrollo de este trabajo.

A todos quienes fomentan el desarrollo tecnológico en Ecuador.

DEDICATORIA

A Dios por ser nuestro creador, amparo y fortaleza, cuando más lo necesitamos, y por hacer palpable su amor a través de cada uno de los que nos rodeó.

A nuestros padres, amigos, parejas y profesores, que sin esperar nada a cambio, han sido pilares en nuestro camino y así, forman parte de este logro que nos abre puertas inimaginables en nuestro desarrollo profesional.

TRIBUNAL DE SUSTENTACIÓN

Ing. Carlos Valdivieso

Profesor de Seminario de Graduación

Ing. Hugo Villavicencio V.

Delegado del Decano

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)

Gema Stefanía Solórzano Aguilar

Raúl Antonio Serrano Mena

RESUMEN

El principal objetivo de este trabajo es el implementar las técnicas aprendidas en nuestra vida académica con respecto al uso de los microcontroladores, sus características, programación y optimización para poder realizar con la ayuda de algunos elementos, varios proyectos que involucren el protocolo SPI (Serial Peripherical Interface) como tema principal. Dándoles un enfoque más específico y práctico para de esta manera poder enlazar y comprender toda la teoría del funcionamiento de este protocolo; haciendo uso de varias herramientas como el software AVR STUDIO 4 para poder programar el cerebro del kit de desarrollo BUTTERFLY que consiste de un microcontrolador ATmega169.

ÍNDICE GENERAL

Contenido

ÍNDICE DE FIGURAS

INTRODUCCIÓN

El objetivo del proyecto es desarrollar e implementar un banco de ejercicios claves que permitan comprender toda la teoría y el funcionamiento referente al protocolo de comunicación SPI (Serial Peripherical Interface). Tratando de darle un uso adecuado y diversificado a los ejercicios, aprovechando las diferentes herramientas que dispone el AVR Butterfly y demás elementos; para de este modo facilitar la comprensión y entendimiento de esta interfaz de comunicación aplicado a la lectura y la enseñanza.

En el primer capítulo, se menciona una descripción general del proyecto, sus partes, la teoría detrás del funcionamiento del protocolo y los diferentes elementos que deben configurarse en el microcontrolador principal que contiene el AVR Butterfly.

En el segundo capítulo, se da un detalle sobre las herramientas de hardware: AVR Butterfly y de las herramientas de software: AVR STUDIO4 con su compilador AVR GCC que permite usar archivos en C y la herramienta de simulación PROTEUS.

El tercer capítulo, trata del diseño e implementación del proyecto, una breve descripción y una lista de materiales de los elementos que van a contener cada uno de los ejercicios planteados en el proyecto.

En el cuarto y último capítulo se detalla, desarrolla e implementa de manera independiente cada ejercicio propuesto en el proyecto; incluyendo el diagrama de bloques, diagrama de flujo, descripción del algoritmo y simulaciones.

Capítulo 1

Descripción General del Proyecto

En los circuitos más modernos, es inevitable que sea necesario incorporar más de un solo chip en el diseño. Por ello, también es esencial que los diversos componentes sean capaces de comunicarse entre sí.

Hay dos formas principales de comunicación, serie y paralelo. En una implementación paralela, hay tantas conexiones como bits que necesitan ser enviados y recibidos. Los buses de comunicación paralelo son muy rápidos, sin embargo, llegan a ser extremadamente grandes y consumen muchos de los invaluables pines que necesita un microcontrolador.

Por la razón de consumo de espacio y de pines, la comunicación serial es preferible a paralelo al conectar dos o más chips, usando solo de dos a cuatro pines. Un bus de comunicación serial funciona mediante el envío de ceros y unos de forma secuencial sobre uno o dos cables. En la mayoría de los microcontroladores modernos es estándar tener un módulo de hardware que controla el reloj del Serial Peripheral Interface (SPI), el buffer receptor y el buffer transmisor.

En el siguiente trabajo desarrollaremos e implementaremos un banco de ejercicios muy intuitivos que faciliten el entendimiento y comprensión de este tipo de interfaz aplicado a la lectura y la enseñanza del mismo.

1. Interfaz Periférica Serial – SPI de los Microcontroladores AVR

La interfaz Periférica Serial (SPI, del inglés Serial Peripheral Interface) es 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's de cristal líquido LCD, subsistemas de conversión analógico-digital, e incluso otros microcontroladores. 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.

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 1.1.: Conexión entre un dispositivo maestro y un esclavo.

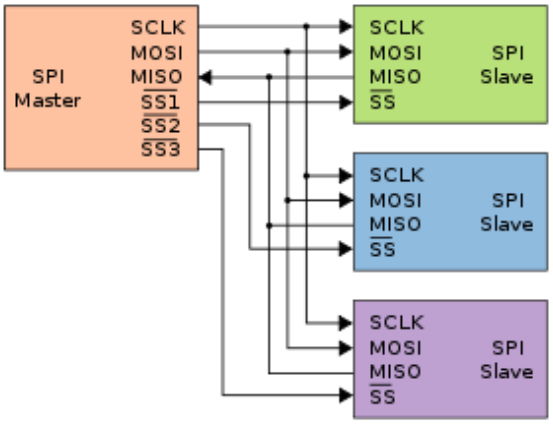


Figura 1.2.: Conexión entre un dispositivo maestro y varios esclavos.

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).

Para comunicar un dispositivo mediante SPI con un microcontrolador AVR se utilizan cuatro pines de este último: MISO, MOSI, SCK Y SS. La siguiente tabla explica la funcionalidad de estos pines:

Nombre del pin	Función
SS	Slave Select: seleccionar al dispositivo como esclavo.
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.
SCK	SPI clock: señal de reloj para SPI.

Tabla 1.1.: Función de los pines de SPI

En resumen, las principales características del SPI son:

- Transferencia de Datos síncrona tres-cables, bidireccional
- Operación Maestro-Esclavo
- Transferencia de Datos LSB o MSB
- Siete velocidades programables en los bits
- Finalización de transmisión por Bandera de Interrupción
- Write Collision Flag Protection
- Despertar desde Modo Idle

1.1. 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.

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.

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).

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

Si la interrupción SPI está habilitada, una interrupción es solicitada. El Esclavo podría continuar para colocar nuevos datos para ser enviados dentro del SPDR antes de

seguir leyendo el dato que va llegando. El último byte que entra permanecerá en el buffer para luego usarse.

Cuando el dispositivo trabaja como maestro, el usuario puede determinar la dirección del pin SS. Si SS 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 a alto para asegurar la operación SPI del Master.

1.2. 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**	**SPCR**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Valor inicial	0	0	0	0	0	0	0	0	

- SPIE, 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.

- SPE, Habilitador del SPI: el sistema SPI es habilitado cuando este bit es puesto a 1. Este bit sería uno para habilitar cualquier operación 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.

- 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.

- 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.

- 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).

- 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 1.2.: Selección de frecuencia de la señal de reloj del SPI

- **Registro de Estado del SPI - SPSR**

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

- SPIF, Bandera de Interrupción SPI: 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.

- 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.

- 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.

- **Registro de Datos del SPI - SPDR**

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

El Registro de Datos 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.

1.3. Modos del Reloj

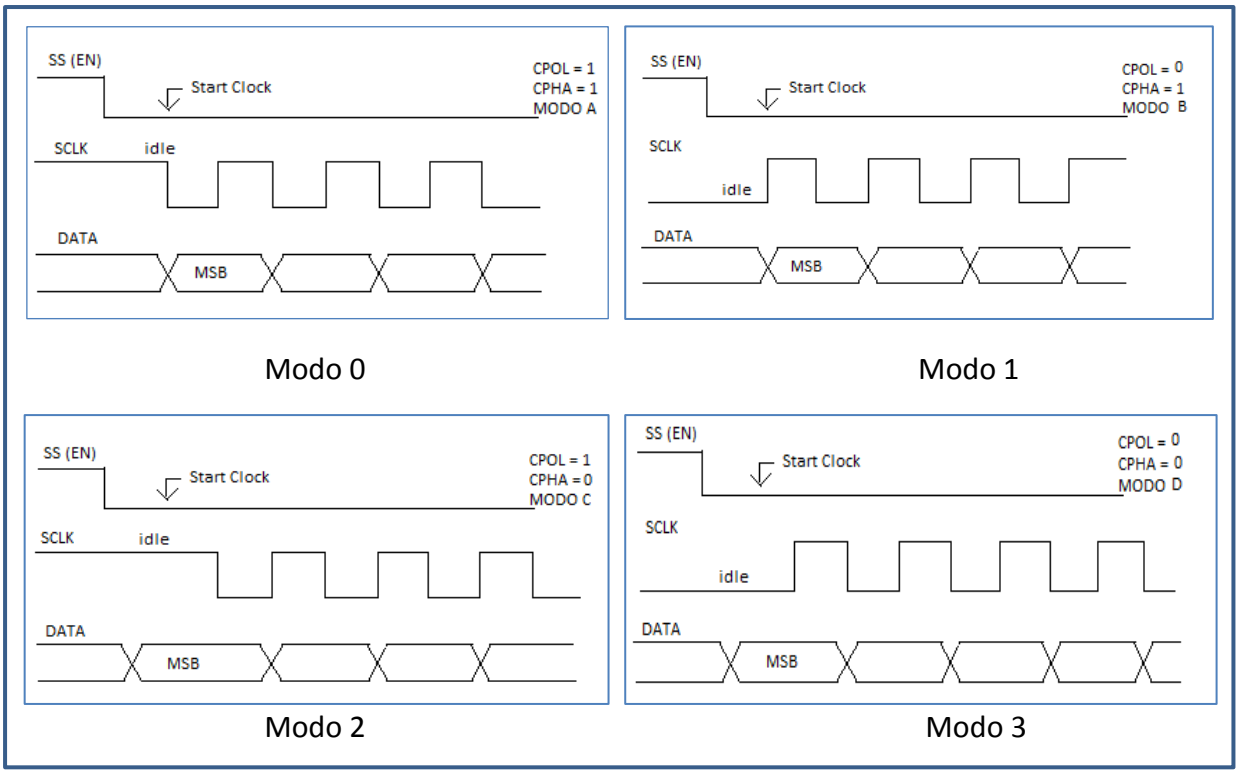
Existen cuatro modos de reloj definidos por el protocolo SPI. Estos determinan el valor de la polaridad del reloj (CPOL = Clock Polarity) y el bit de fase del reloj (CPHA = Clock Phase). Estos modos son:

- Modo 0: CPOL = 1 CPHA = 1
- Modo 1: CPOL = 0 CPHA = 1
- Modo 2: CPOL = 1 CPHA = 0
- Modo 3: CPOL = 0 CPHA = 0

La mayoría de los dispositivos SPI pueden soportar al menos 2 modos de los 4 antes mencionados.

El bit de Polaridad del reloj determina el nivel del estado de Idle del reloj y el bit de Fase de reloj determina qué flanco recibe un nuevo dato. El modo requerido para una determinada aplicación, está dado por el dispositivo esclavo.

Los diferentes modos son ilustrados a continuación.



1.4. Ventajas del SPI

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

1.5. 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.
- No hay control de flujo por hardware.
- No hay señal de asentimiento. El servidor podría estar enviando información sin que estuviese conectado ningún cliente y no se daría cuenta de nada.
- No permite fácilmente tener varios servidores conectados al bus.
- Sólo funciona en las distancias cortas a diferencia de, por ejemplo, RS-232, RS-485, o Bus CAN.

Capítulo 2

Fundamento Teórico

2. Requerimientos para la aplicación del Proyecto

El proyecto se lo puede dividir básicamente en dos partes esenciales: Software y Hardware, ambos indispensables para la elaboración del proyecto.

El software que se utilizará para la programación de los microcontroladores requeridos en el proyecto es el AVR Studio 4, el cual es un Entorno de Desarrollo Integrado para escribir y depurar aplicaciones AVR en el entorno de Windows y posee dos compiladores para los lenguajes Assembler y C usados para la creación de los códigos.

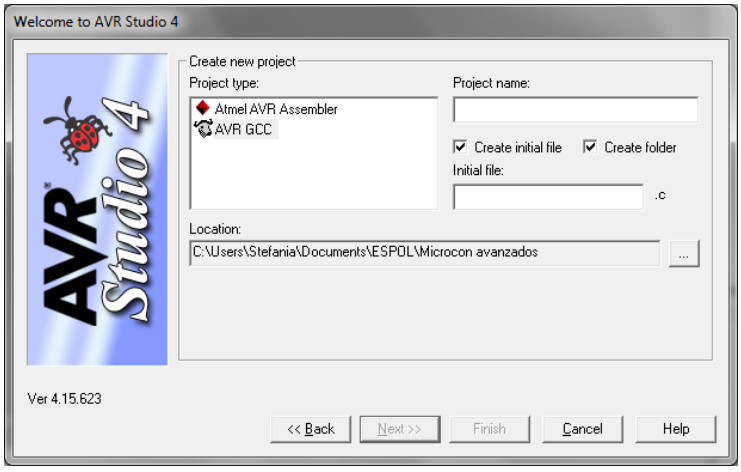


Figura 2.1.: Programa AVR Studio 4

También se utilizará el software de simulación PROTEUS 7.7 Service Pack 2, el cual permite implementar en forma simulada los códigos hechos en lenguaje C con los integrados y sus conexiones.

Figura 2.2.: Software Proteus

El hardware que se utilizará para desarrollar el proyecto principalmente es el AVR Butterfly, el cual es un Kit de desarrollo, entrenamiento y aprendizaje de microcontroladores Atmel, que contiene: un microcontrolador ATmega169V, LCD, Joystick, altavoz, cristal de 32 KHz, DataFlash de 4 Mbit, convertidor de nivel RS-232, interfaz USART, interfaz USI, sensor de temperatura, sensor de luz, ADC, conectores para acceso a periféricos, y Batería de 3 V.

Figura 2.3.: Vistas frontal y posterior del AVR Butterfly

2.1. Herramientas de Software

Es el conjunto de herramientas que permiten al programador desarrollar programas informáticos, usando diferentes alternativas y lenguajes de programación, de una manera práctica. Incluye entre otros: Editores de texto, Compiladores, Intérpretes, Enlazadores, Depuradores, Simuladores.

2.1.1. AVR STUDIO 4

El AVR Studio 4 es un software que pertenece a la familia ATMEL y proporciona herramientas para la administración de proyectos, edición de archivo fuente, simulación del chip e interfaz para emulación In-circuit para la poderosa familia RISC de microcontroladores AVR de 8 bits.

AVR Studio 4 consiste de muchas ventanas y sub-módulos donde cada uno apoya a las partes del trabajo que se intenta realizar.

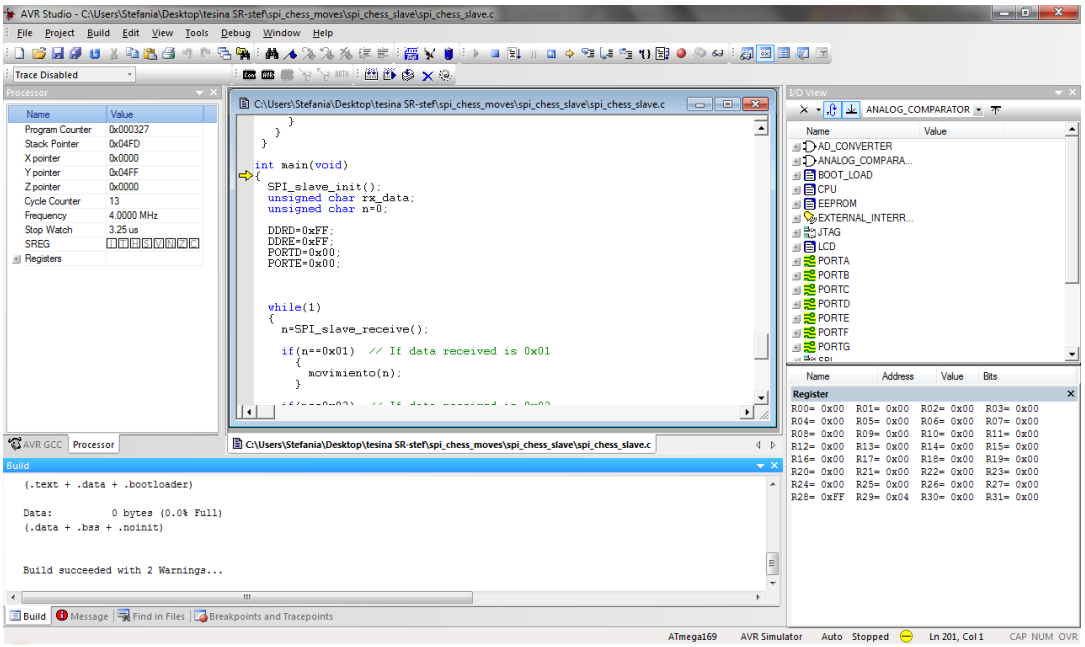


Figura 2.4.: Entorno de Desarrollo Integrado AVR Studio 4

Toda creación de código dentro del AVR Studio 4 se la realiza como proyectos de programación. Todos los proyectos de código tienen un archivo project que mantiene la información acerca del proyecto, de qué archivos consta, la estructuración del ensamblador, vistas personalizadas y así sucesivamente. El archivo project asegura que el proyecto sea el mismo cada vez que regrese a él, que todo esté adecuadamente organizado y que ensamble / compile.

El compilador es un programa que se encarga de traducir los programas escritos por el programador en lenguaje de alto nivel (entendible por el ser humano) a un lenguaje de bajo nivel que es el comprensible por la máquina y que, de esta manera, permite que pueda ser ejecutado por la computadora. Sería la transformación del código fuente a un lenguaje máquina o código objeto.

De esta manera un programador puede diseñar un programa en un lenguaje mucho más cercano a cómo piensa un ser humano, para luego compilarlo a un programa más manejable por una computadora.

Figura 2.5.: Pasos de la compilación.

Un compilador no es un programa que funciona de manera aislada, sino que necesita de otros programas para conseguir su objetivo: obtener un programa ejecutable a

partir de un programa fuente en un lenguaje de alto nivel. Algunos de esos programas son el preprocesador, el linker, el depurador y el ensamblador.

2.1.2. PROTEUS

PROTEUS es una herramienta software que permite la simulación de circuitos electrónicos con microcontroladores. Sus reconocidas prestaciones lo han convertido en el más popular simulador software para microcontroladores.

Esta herramienta permite simular circuitos electrónicos complejos integrando inclusive desarrollos realizados con microcontroladores de varios tipos, en una herramienta de alto desempeño con unas capacidades graficas impresionantes.

Presenta una filosofía de trabajo semejante al SPICE, arrastrando componentes de una barra e incrustándolos en la aplicación, es muy sencillo de manejar y presenta una interfaz gráfica amigable para un mejor manejo de las herramientas proporcionadas por el Proteus.

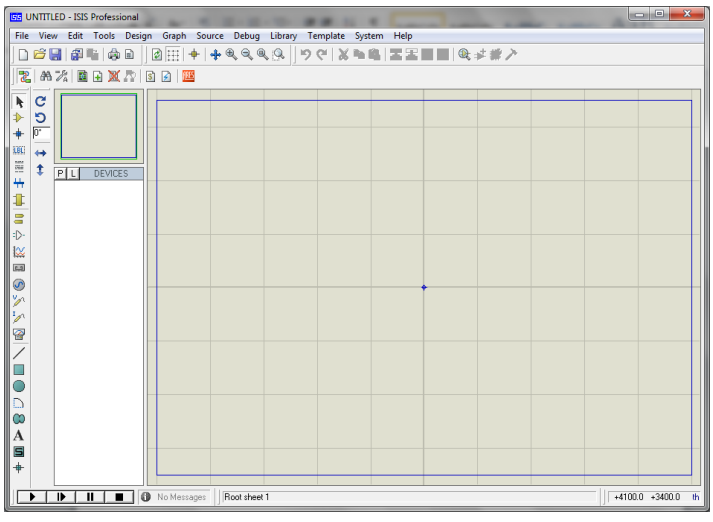


Figura 2.6.: Interfaz Gráfica Proteus

2.2. Herramientas de Hardware

Las herramientas de hardware corresponden a todas las partes tangibles de un sistema informático: sus componentes eléctricos, electrónicos, electromecánicos y mecánicos; sus cables, gabinetes o cajas, periféricos de todo tipo y cualquier otro elemento físico involucrado.

2.2.1. AVR Butterfly

El Kit AVR Butterfly se diseñó para demostrar los beneficios y las características importantes de los microcontroladores ATMEL. Este kit utiliza el microcontrolador AVR ATmega169V, que combina la Tecnología Flash con el más avanzado y versátil microcontrolador de 8 bits disponible (7).

El AVR Butterfly expone las siguientes características principales:

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

        Flash, EEPROM, SRAM.

        DataFlash externa.
- Interfaces de comunicación:
    - UART, SPI, USI.
- Métodos de programación

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

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

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 DataFlash de 4 Mbit, para el almacenar datos.
- Convertidor de nivel RS-232 e interfaz USART, para comunicarse con unidades fuera del Kit sin la necesidad de hardware adicional.
- Termistor de Coeficiente de Temperatura Negativo (NTC), para 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 preprogramada.
- Compatibilidad con el Entorno de Desarrollo AVR Studio 4.

En las Figuras 2.7 y 2.8 se observa el Hardware disponible en el AVR Butterfly.

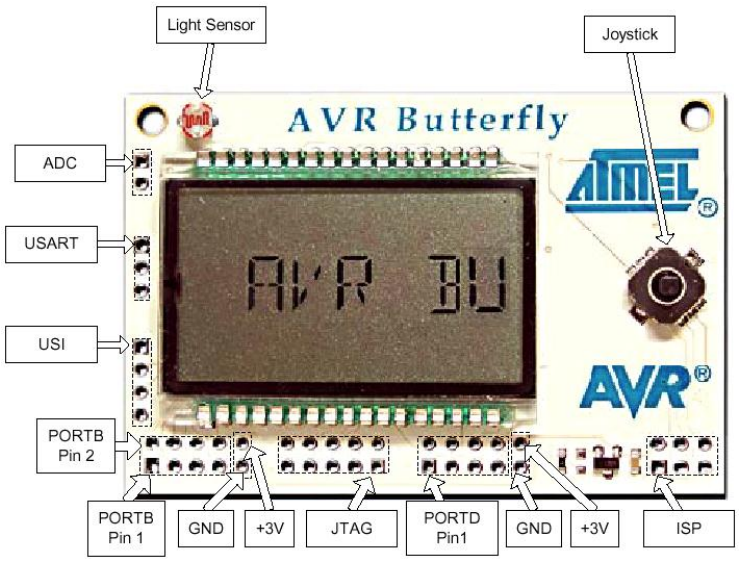


Figura 2.7.: Hardware disponible – Parte frontal

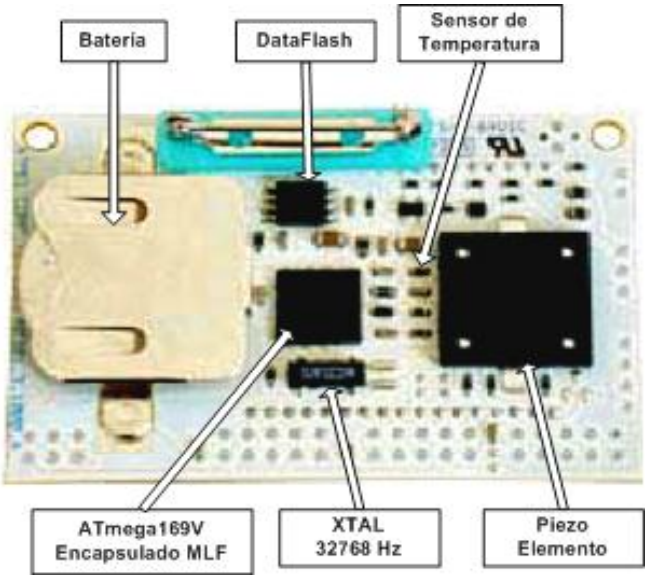


Figura 2.8.: Hardware disponible – Parte posterior

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 (3).

El Atmega169 es un microcontrolador CMOS de bajo consumo de 8-bits basado en la arquitectura RISC de los AVR. Mediante la ejecución de instrucciones de gran alcance en un solo ciclo de reloj, el ATmega169 logra rendimientos aproximados a 1 MIPS por MHz, permitiendo al sistema diseñado optimizar el consumo de energía frente a la velocidad de procesamiento.

El Atmega169 ofrece las siguientes características: 16K bytes de programación ISP, Flash con capacidades de Leer mientras se Escribe, 512 bytes de EEPROM, 1K byte de SRAM, 53 líneas I/O de propósito general, 32 registros de propósito general, interfaz JTAG, un completo controlador de LCD con voltaje interno, tres Temporizadores/Contadores con modos de comparación, interrupciones internas y

externas, programación serial USART, Interfaz Serial Universal con detector condicional de arranque, ADC de 10-bits, Watchdog Timer programable con oscilador interno, puerto serial SPI, y 5 modos de ahorro de energía seleccionable por software (6).

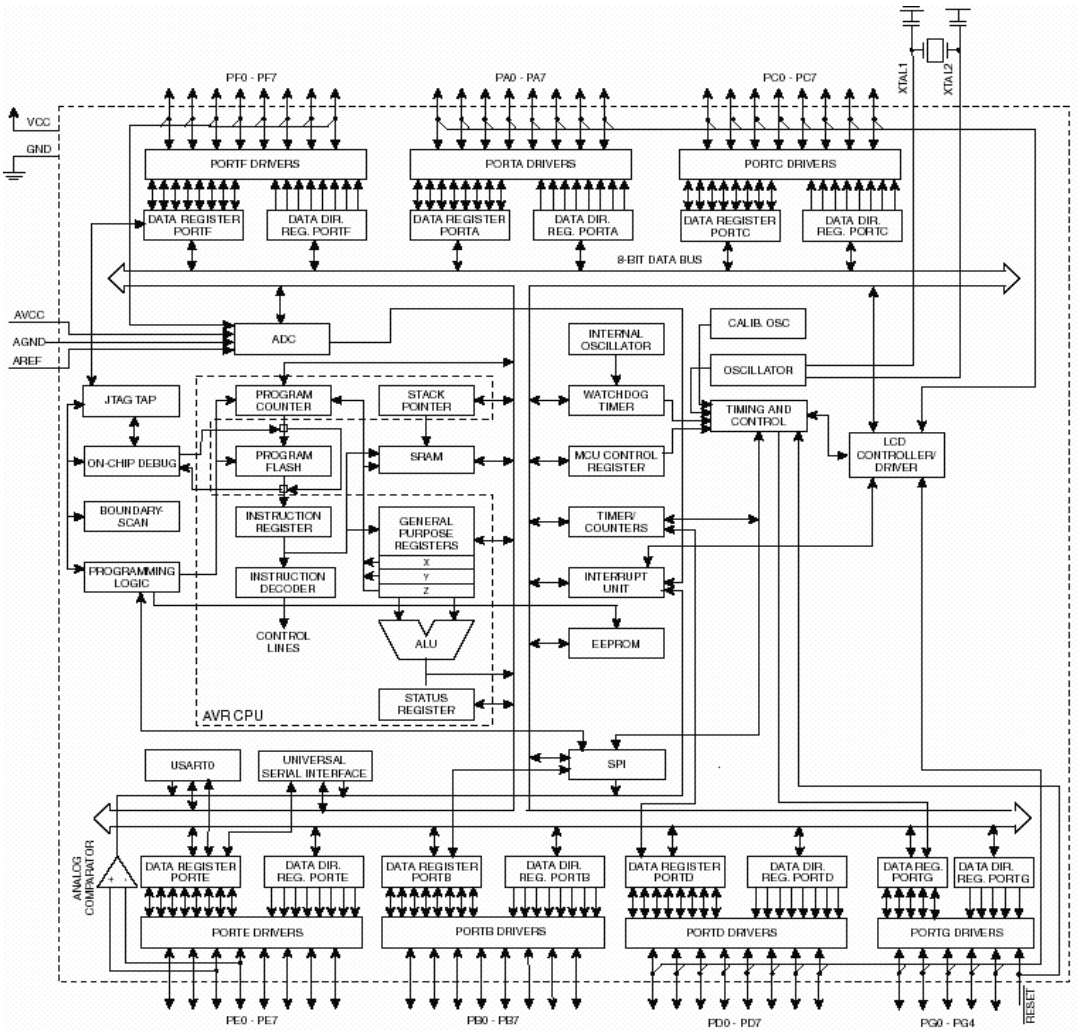


Figura 2.9.: Diagrama de Bloques del Atmega169

Capítulo 3

Diseño e Implementación del Proyecto

3. 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:

- **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 3.1.: 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.

- **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.

El AVR Butterfly cuenta también con controladores de LCD, joystick, interfaces seriales, EEPROM, dataflash, etc. que permiten la interacción del usuario con el programa, dando como resultado un proyecto dinámico.

- **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 4 pilas AA (1.50 V), es decir, se trabajará con una fuente de alimentación de 6V para el del AVR Butterfly.

Una pila AA es el tamaño estándar de una pila. Son por lo general utilizadas en dispositivos electrónicos portátiles. Está formada por una sola celda electroquímica. El voltaje en los terminales y la capacidad depende de la reacción química en la celda. Algunas celdas recargables se fabrican con este tamaño. Una pila AA mide 50 mm de longitud y 14 mm de diámetro.

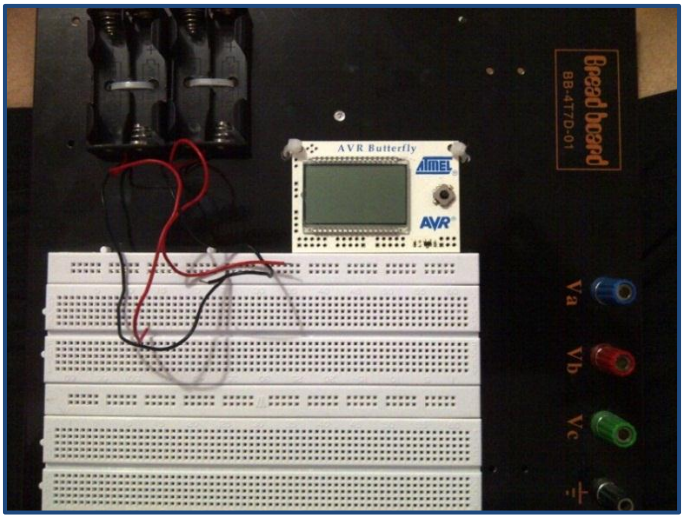


Figura 3.2.: Plataforma del Proyecto.

3.1. 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 con aplicaciones específicas.

3.1.1. Rotación de Leds.

- **Descripción:**

El Atmega169 del AVR Butterfly trabaja como el maestro de la comunicación serial y realiza la rotación de 8 leds conectados en el Puerto D del microcontrolador hacia la izquierda y luego hacia la derecha, al mismo tiempo que se transmiten estos datos al registro de desplazamiento 74HC595, el cual trabaja como el esclavo de la comunicación y realiza la misma rotación de leds conectados a su puerto de salida.

- **Lista de Materiales:**

- Registro de Desplazamiento 74HC595.
- Ocho leds rojos.
- Ocho leds azules.
- Dieciséis resistencias de 330 Ω.

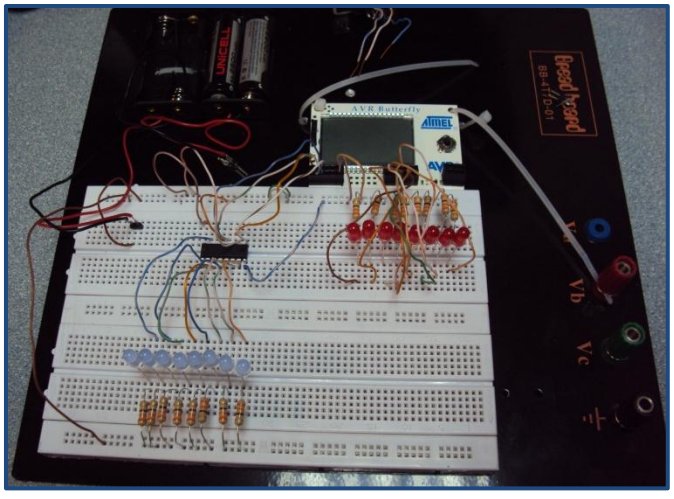


Figura 3.3.: Implementación del Ejercicio 1 en Proteus

3.1.2. Símbolos en Matriz de Leds 8x8.

- **Descripción:**

El maestro AVR Butterfly envía mediante comunicación SPI, una serie de datos a dos registros de desplazamiento 74HC595 que funcionan como esclavos, y que representan los diferentes símbolos a mostrarse en la matriz de leds de 8x8.

El microcontrolador del Butterfly utiliza la interrupción por desbordamiento del Timer 0 para mandar los datos cada 2ms.

Cada vez que se genera la interrupción se envían 16 bits: los primeros 8 bits al primer esclavo (74HC595) y que representan el símbolo a mostrar, y los siguientes 8 que se envían al segundo esclavo y que representan el número de columna de la matriz donde debe llegar el dato. La matriz de leds es conectada al segundo esclavo de la comunicación.

- **Lista de Materiales:**

- Dos Registros de Desplazamiento 74HC595.
- Una matriz de leds 8x8.
- Tres resistencias de 10K Ω.
- Ocho resistencias de 330 Ω.

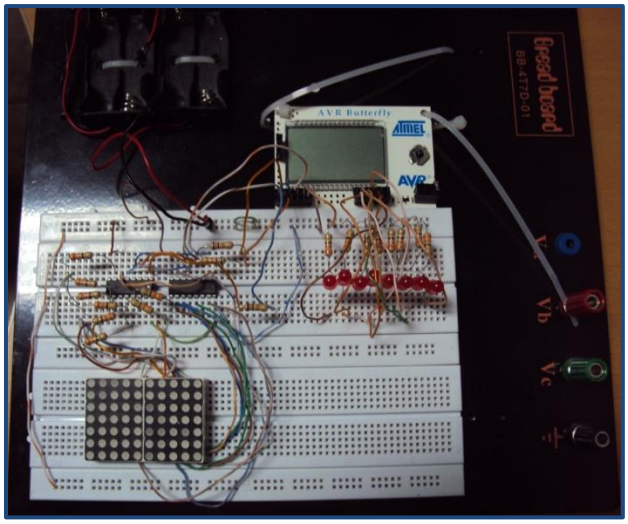


Figura 3.4.: Implementación del Ejercicio 2 en Proteus

3.1.3. Movimiento de las piezas de Ajedrez.

- **Descripción:**

El microcontrolador Atmega169 del AVR Butterfly funciona como el maestro de la comunicación SPI y recibe como entrada la lectura de 6 botoneras conectadas al Puerto F y que corresponden a las piezas de ajedrez: Peón, Caballo, Alfil, Torre, Reina y Rey. Cuando se presiona una de las botoneras, el maestro le envía esta información al esclavo y muestra el nombre de la pieza escogida por el LCD.

Un segundo microcontrolador Atmega169 trabaja como el esclavo de la comunicación y recibe la información dela botonera seleccionada. Según esta información, simula los movimientos de la pieza de ajedrez respectiva en el tablero, representado por una matriz de leds de 3x3 que se conectan al Puerto D del microcontrolador.

- **Lista de Materiales:**

- Microcontrolador Atmega169.
- Seis botoneras.
- Nueve leds rojos.
- Seis resistencias de 1K Ω.
- Nueve resistencias de 330 Ω.

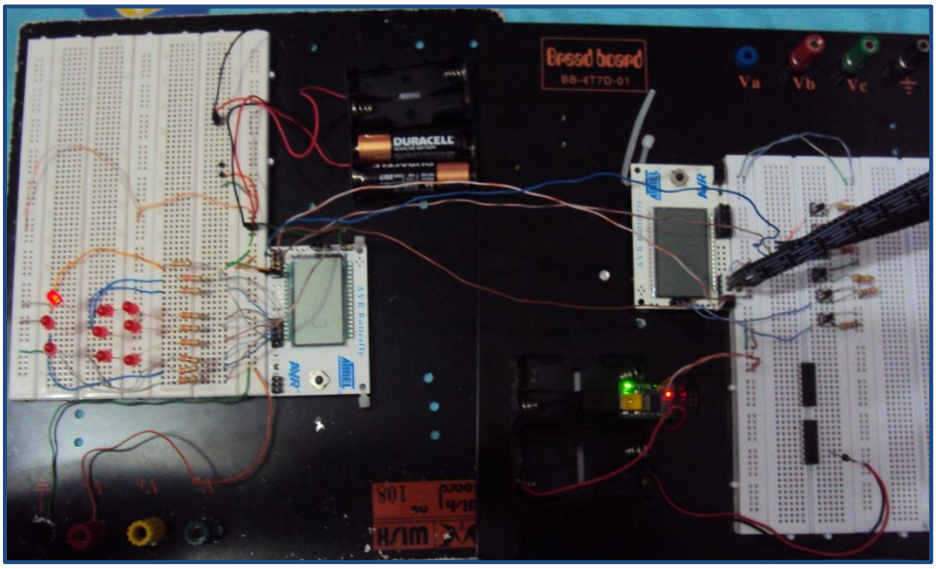


Figura 3.5.: Implementación del Ejercicio 3 en Proteus

3.1.4. Contador de dos Dígitos.

- **Descripción:**

Dependiendo de la botonera seleccionada en el Puerto B del AVR Butterfly, el cual funciona como el maestro del SPI, se incrementa o decrementa el contador de dos dígitos en una o dos unidades de la siguiente manera:

- INC1 = Incrementa 1 unidad
- DEC1 = Decrementa 1 unidad
- DEC2 = Decrementa 2 unidades
- INC2 = Incrementa 2 unidades

El dígito correspondiente a las unidades se muestra en un display conectado al Puerto D del Butterfly. El esclavo de la comunicación SPI, que en este caso es un microcontrolador Atmega169, recibe el dato del dígito correspondiente a las decenas y lo muestra en un display conectado a su Puerto D.

- **Lista de Materiales:**

- Microcontrolador Atmega169.
- Dos displays de 7 segmentos.
- Catorce resistencias de 330 Ω.

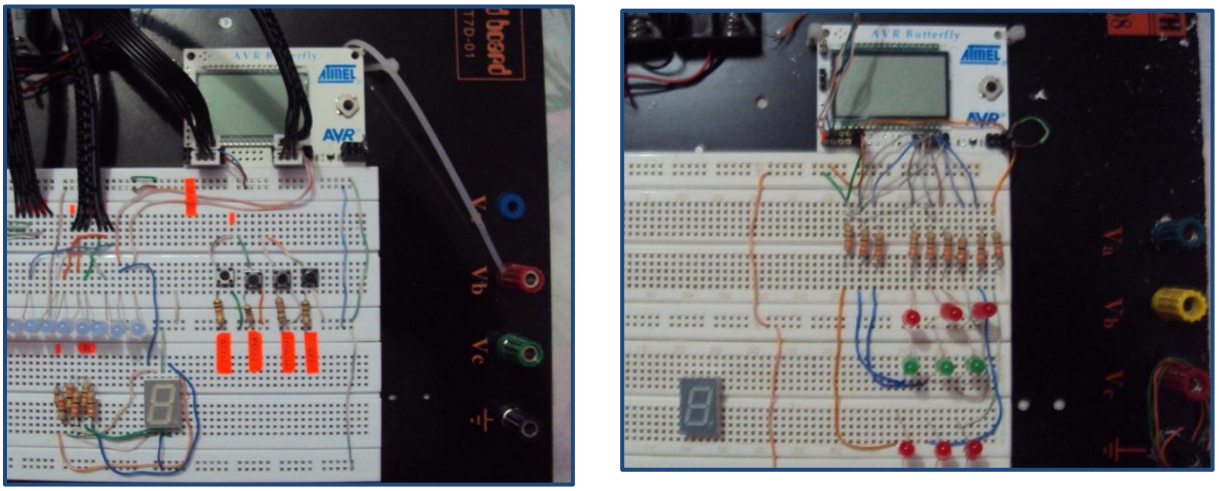


Figura 3.6.: Implementación del Ejercicio 4 en Proteus

3.1.5. Mensaje en Matriz de Leds 8x8.

- **Descripción:**

El maestro AVR Butterfly envía mediante comunicación SPI, una serie de datos a dos registros de desplazamiento 74HC595 que funcionan como esclavos, y que representan un mensaje. Este mensaje se va desplazando a través de la matriz de leds de 8x8 hasta que se logra visualizar completamente, para luego volverlo a mostrar desde su principio indefinidamente.

El maestro envía 16 bits de información: los primeros 8 bits al primer esclavo (74HC595) que representan el mensaje a mostrar, y los siguientes 8 bits al segundo esclavo que representan el número de columna de la matriz donde debe llegar el dato. La matriz de leds es conectada al segundo esclavo de la comunicación.

- **Lista de Materiales:**

- Dos Registros de Desplazamiento 74HC595.
- Una matriz de leds 8x8.
- Dos resistencias de 10K Ω.
- Ocho resistencias de 330 Ω.

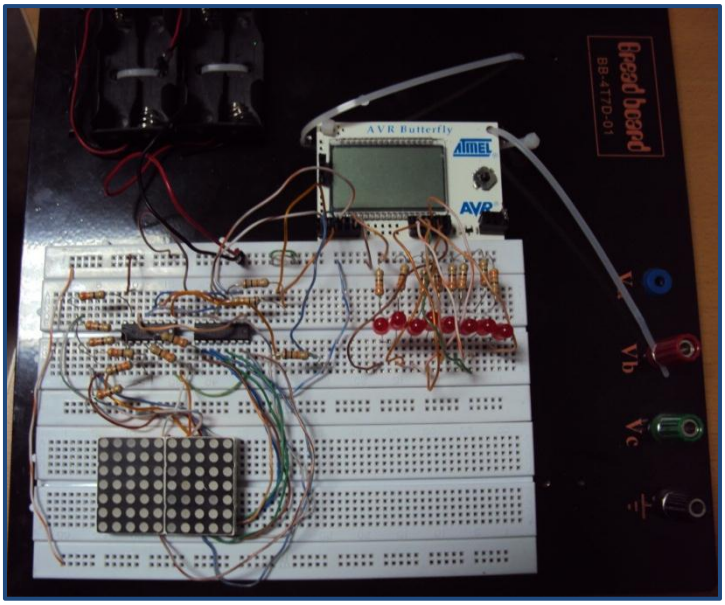


Figura 3.7.: Implementación del Ejercicio 5 en Proteus

Capítulo 4

Desarrollo y Simulación del Proyecto

4.1. Desarrollo de los Ejercicios

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.

4.1.1. Rotación de Leds.

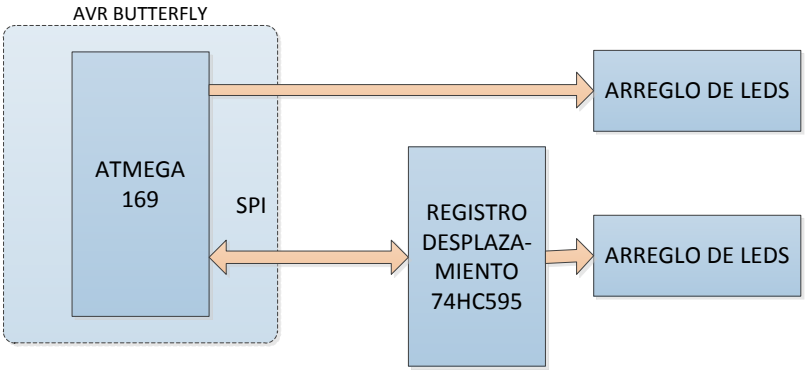
- Descripción

El Atmega169 del AVR Butterfly trabaja como el maestro de la comunicación serial y realiza la rotación de 8 leds conectados en el Puerto D del microcontrolador hacia la izquierda y luego hacia la derecha, al mismo tiempo que se transmiten estos datos al registro de desplazamiento 74HC595, el cual trabaja como el esclavo de la comunicación y realiza la misma rotación de leds conectados a su puerto de salida.

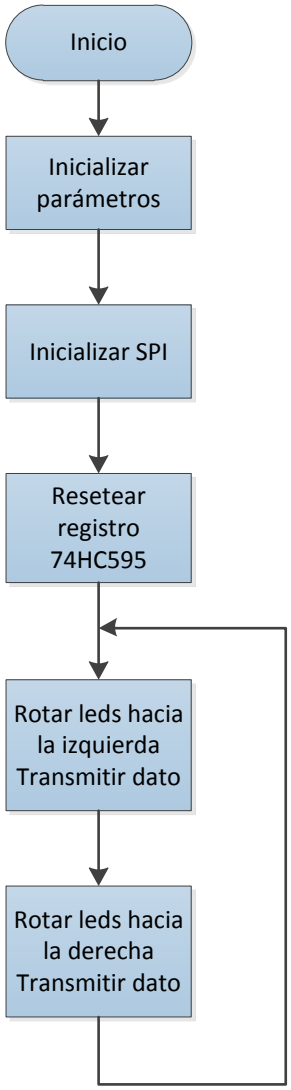
- Diagrama de Bloques

El AVR Butterfly es el maestro de la comunicación SPI, el cual muestra en un Puerto de salida la rotación de luces de leds. El registro de desplazamiento 74HC595 que

trabaja como esclavo muestra la misma rotación de leds que el maestro le envió mediante el SPI.



- Diagrama de Flujo



- Descripción del Algoritmo

1. Se inicializan las variables y los puertos a utilizar.
2. Inicializamos el protocolo SPI.
3. Se inicializa el registro de desplazamientos 74HC595 con un RESET.
4. El maestro ATMEGA169 rota el valor a mostrar y lo envía por las 3 vías SPI al esclavo 74HC595 para que muestre la rotación en sus salidas, esto lo hace hasta completar la rotación hacia la izquierda de todos los LEDs.
5. Concluida la rotación hacia la izquierda prosigue a hacer lo mismo que en el punto 4 pero ahora hacia la derecha.
6. Se repiten los pasos 4 y 5 indefinidamente debido a que se encuentran en un lazo WHILE.

- Programa Principal del Controlador Maestro

```
/*************************************************************
 **********       MICROCONTROLADORES AVANZADOS        **********
 *************************************************************
              Comunicación Serial SPI
 *************************************************************
                   Ejercicio # 1
 * Nombre: Rotación de Leds
 * Descripción:
   El AVR Butterfly trabaja como maestro y muestra por el
   Puerto D la rotación de 8 leds que le envía al registro
   74HC595, el cual trabaja como esclavo del SPI.

 *************************************************************
 ** Nombre del archivo: leds.c
 *************************************************************/

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

#define SPI_PORT PORTB
#define SPI_DDR  DDRB
#define SPI_CS   PB0
```

```
// Funciones:
void SPI_init (void)
{
 // Setear MOSI y SCK como salidas, los demás como entradas
 SPI_DDR = (1<<PB2)|(1<<PB1)|(1<<PB0);
 // Deshabilitar esclavo (RCK Low)
 SPI_PORT &= ~(1<<SPI_CS);
 // Habilitar SPI, Master, setear clock rate fck/2 (máximo)
 SPCR = (1<<SPE)|(1<<MSTR);
 SPSR = (1<<SPI2X);
}

unsigned char SPI_WriteRead(unsigned char dataout)
{
 unsigned char datain;

 SPDR = dataout; // empieza transmisión (MOSI)

 while(!(SPSR & (1<<SPIF))); // espera que se complete la transmisión
  datain = SPDR;  // retorna valor recibido


 SPI_PORT |= (1<<SPI_CS);  // habilita al esclavo
 _delay_us(1);           // espera 1 useg
 SPI_PORT &= ~(1<<SPI_CS); // deshabilita al esclavo

 return datain;  // retorna valor del serial (MISO)
}


// Programa Principal:
int main(void)
{
 unsigned char cnt;

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

 SPI_init(); // inicializa el SPI
 cnt=SPI_WriteRead(0);  // Resetea el registro 74HC595

 // rotación de leds hacia la izquierda
 for(;;) {
  cnt=1;
  while(cnt) {
           cnt=cnt<<1;   // rotación
           PORTD=SPI_WriteRead(cnt); // transmisión
           _delay_ms(100);
          }
          cnt=0x80;
          // rotación de leds hacia la derecha
```

```
          while(cnt) {
            cnt=cnt>>1;   // rotación
            PORTD=SPI_WriteRead(cnt); // transmisión
            _delay_ms(100);
          }

 }

 return 0;

}
```

4.1.2. Símbolos en Matriz de Leds 8x8.

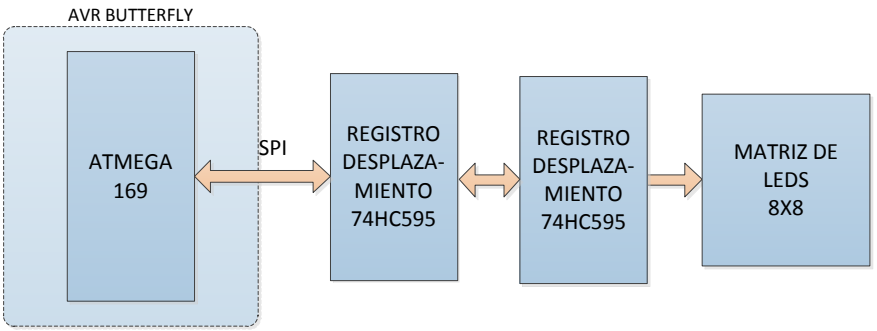
- Descripción

El maestro AVR Butterfly envía mediante comunicación SPI, una serie de datos a dos registros de desplazamiento 74HC595 que funcionan como esclavos, y que representan los diferentes símbolos a mostrarse en la matriz de leds de 8x8.

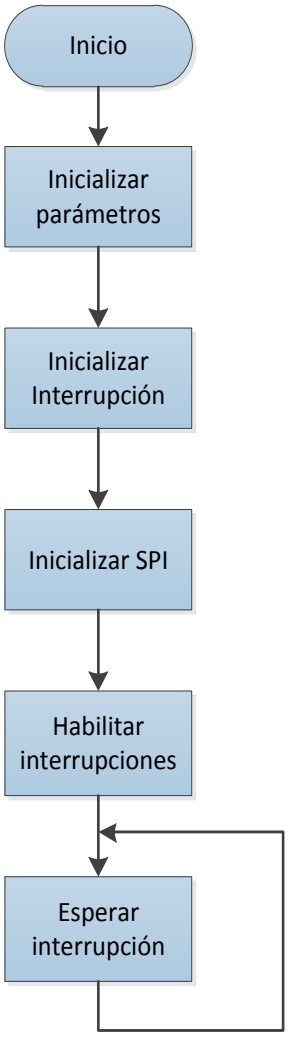
El microcontrolador del Butterfly utiliza la interrupción por desbordamiento del Timer 0 para mandar los datos cada 2ms. Cada vez que se genera la interrupción se envían 16 bits: los primeros 8 bits al primer esclavo (74HC595) y que representan el símbolo a mostrar, y los siguientes 8 que se envían al segundo esclavo y que representan el número de columna de la matriz donde debe llegar el dato. La matriz de leds es conectada al segundo esclavo de la comunicación.

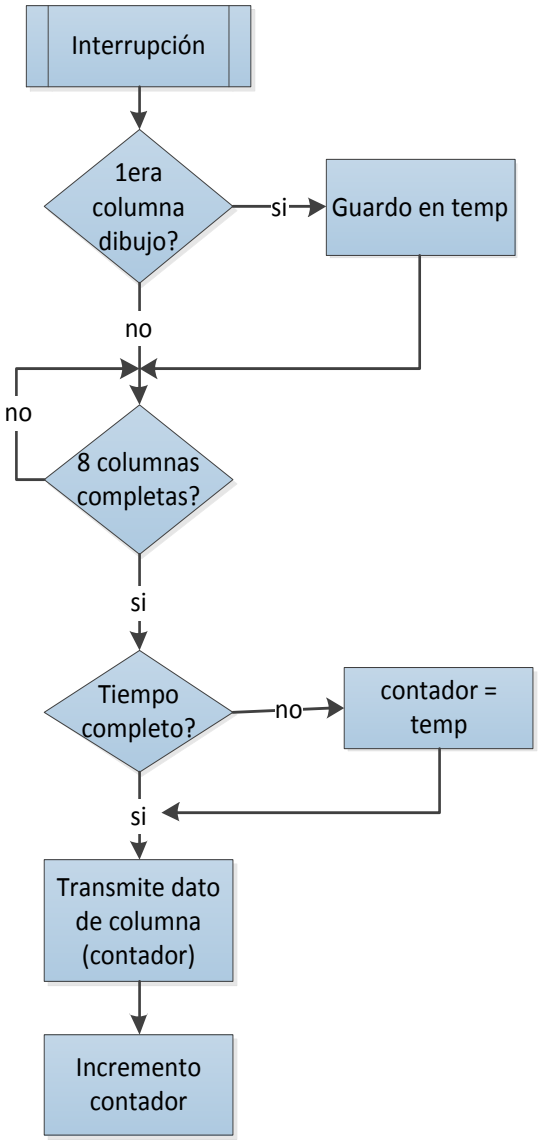
- Diagrama de Bloques

El AVR Butterfly envía los datos que representan a los símbolos a dos registros 74HC595 que son los esclavos de la comunicación SPI, y éstos muestran el símbolo en la matriz de leds de 8x8.



- Diagrama de Flujo





- Descripción del Algoritmo

1. Se inicializan las variables y los puertos a utilizar.
2. Se inicializa la interrupción por desbordamiento de Timer.
3. Se inicializa la comunicación SPI.
4. Se habilitan las interrupciones globales.
5. Espera indefinidamente por la interrupción.

Interrupción:

1. Si se llega al inicio del siguiente símbolo se guarda temporalmente el dato de la primera columna en una variable.
2. Si todavía no se llega al siguiente símbolo, se espera por la demostración completa del símbolo actual.
3. Si se muestran las 8 columnas correspondientes a un símbolo, se pregunta por el tiempo que debe permanecer en pantalla. Si no se ha completado el tiempo, se mantiene guardado el dato del siguiente símbolo en la variable temporal. Si se completa el tiempo, se regresa el dato al contador del programa para que muestre el siguiente dibujo.
4. Se transmite dato al esclavo para mostrar símbolo en la matriz de leds.
5. Se incrementa el contador para obtener el siguiente dato del símbolo.

- Programa Principal del Controlador

```
/*****************************************************************
**********      MICROCONTROLADORES AVANZADOS      **********
*****************************************************************
            Comunicación Serial SPI
*****************************************************************
                Ejercicio # 2
* Nombre: Símbolos en Matriz de Leds 8x8
```

```
 * Descripción:
   El esclavo muestra una serie de símbolos en una matriz de Leds
   de 8x8 que le envía el maestro AVR Butterfly mediante SPI.

 ***************************************************************
 ** Nombre del archivo: matriz.c
 ***************************************************************/

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

// variables globales
  int temp = 0;
  int cont = 0;
  int cont2 = 0;
  int cont3 = 0;
  int num = 10;
  int contador = 0;

// Símbolos:
const char matriz[] = {

0x10, 0x20, 0x40, 0x80, 0x80, 0x02, 0x02, 0x02, //  upper sinewave
0x08, 0x04, 0x02, 0x01, 0x01, 0x01, 0x01, 0x01, //  lower sinewave
0x99, 0x24, 0x42, 0x99, 0x99, 0x42, 0x24, 0x99, //  round pattern
0xFF, 0x81, 0xBD, 0xA5, 0xA5, 0xBD, 0x81, 0xFF, //  box pattern inner
0x55, 0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55, 0xAA, //  fine checkers
0x33, 0x33, 0xCC, 0xCC, 0x33, 0x33, 0xCC, 0xCC, //  big checkers
0x42, 0xC3, 0x24, 0x18, 0x18, 0x24, 0xC3, 0x42, //  X pattern
0xFD, 0x85, 0xB5, 0xA5, 0xA5, 0xBD, 0x81, 0xFF, //  sprial
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, //  8x8 block
0xFF, 0x7E, 0x7E, 0x7E, 0x7E, 0x42, 0x24, 0x18, //  bullet Right
0x18, 0x18, 0x3C, 0x66, 0x66, 0x3C, 0x18, 0x18, //  chain link
0x78, 0x78, 0x18, 0xFF, 0xFF, 0x0C, 0x3C, 0x3C, //  cactus
0xF2, 0x82, 0x12, 0x3A, 0x10, 0xC0, 0xC4, 0x0E, //  tetris
0x7F, 0x84, 0xA7, 0x84, 0xA7, 0x84, 0x7F, 0x00, //  pacman ghost
0x3C, 0x42, 0x81, 0xA1, 0x89, 0x95, 0xA5, 0x42, //  pacman
0x07, 0x2F, 0x1C, 0x3E, 0x3C, 0x30, 0x30, 0x30, //  gun
0x5A, 0x99, 0x00, 0x18, 0x18, 0x00, 0x18, 0x18, //  gun bullets
0x82, 0x41, 0x82, 0x41, 0x82, 0x41, 0x82, 0x41, //  checkers edges
0x00, 0x01, 0x06, 0x7E, 0xDF, 0x7E, 0x06, 0x01, //  rocket
0x04, 0x0F, 0x1F, 0x3C, 0x3C, 0x1F, 0x0F, 0x04, //  house
0xFF, 0x00, 0xFF, 0x00, 0xFF, 0x00, 0xFF, 0x00, //  vertical stripes
0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, 0x55, //  horizontal strips
0x49, 0x92, 0x24, 0x49, 0x92, 0x24, 0x49, 0x92, //  diag strips left
0x92, 0x49, 0x24, 0x92, 0x49, 0x24, 0x92, 0x49, //  diag strips rights
0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, 0x18, //  arrow shaft
0x18, 0x18, 0x3C, 0x5A, 0x99, 0x3C, 0x42, 0x81, //  arrow tail
0x18, 0x3C, 0x7E, 0xFF, 0x18, 0x18, 0x18, 0x18, //  arrow head
0x81, 0x42, 0x24, 0x18, 0x81, 0x42, 0x24, 0x18, //  carrots L
0x18, 0x24, 0x42, 0x81, 0x18, 0x24, 0x42, 0x81, //  carrots R
```

```
0x81, 0x42, 0x24, 0x99, 0x5A, 0x3C, 0x18, 0x18, //  tail good
0x18, 0x18, 0x18, 0x18, 0xFF, 0x7E, 0x3C, 0x18, //  head good

};

// Funciones:
void spi_init(void) {
  // configuración del SPI
  // dispositivo Master, modo de reloj= 3, f_sck = f_osc / 4
  // deshabilitado SPI interrupt, orden del dato = MSB primero
  SPCR = _BV(SPE) | _BV(MSTR) | _BV(CPOL) | _BV(CPHA);
}

void spi_begin(void) {
  // setear /SS = bajo para seleccionarlo como esclavo.
  PORTB &= 0b11111110;
}

void spi_end(void) {

  PORTB |= 0b00100000;  // apagar LEDs /OE = 1,
  PORTB |= 0b00000001; // setear /SS = 1,
  PORTB &= 0b11011111; // prender LEDs /OE = 0
}

char spi_send(char cData) {
  SPDR = cData;

  while(!(SPSR & (1<<SPIF))); // esperar que se complete la transmisión
  return SPDR;  // retorna dato del esclavo SPI
}

// Interrupción:
ISR (TIMER0_OVF_vect)
{
   char msj, col;
   int linea;

  if ((cont%8)==0)
    temp=contador;        // guarda primera columna del dibujo

  if (cont2==8)
  {
    contador=temp - 8;  // muestra las 8 columnas del dibujo
              cont2=0;
    cont3++;

    if (cont3==num)
      {
      contador=temp;    // muestra todo el dibujo determinado tiempo
      cont3=0;
```

```
    cont=0;
    }
  }

  linea = contador& 0x07; // selecciona columna de LED (0 .. 7)


  contador = contador& 0xFF;
  msj = matriz[contador];
  col = ~ (1 << linea);  // columna seleccionada en bajo, las demás en alto

   // SPI transferencia (16 bits)
  spi_begin();
  spi_send(msj);
  spi_send(col);
  spi_end();

  cont++;
  cont2++;
  contador++;
}


// Programa Principal:
int main( void )
{
  PORTB = 0;         // seteando PORT B;
  DDRB  = 0b00100111; // salidas = MOSI, SCK, /SS, PB5(/OE)

  contador = 0;   // puntero para la tabla del mensaje

  TCNT0 = 0;    // valor inicial del timer0
  TCCR0A = 2;    // preescalador por clk/8 (1 count = 8uS)
  TIMSK0 = _BV(TOIE0);  // habilita timer0 overflow interrupt
  // interrupción del timer cada 8uS * 256 = 2048uS (approx. 2mS)

  spi_init(); // inicializa SPI
  sei(); // habilita interrupciones
  while(1){

  }

}
```

4.1.3. Movimiento de las piezas de Ajedrez.

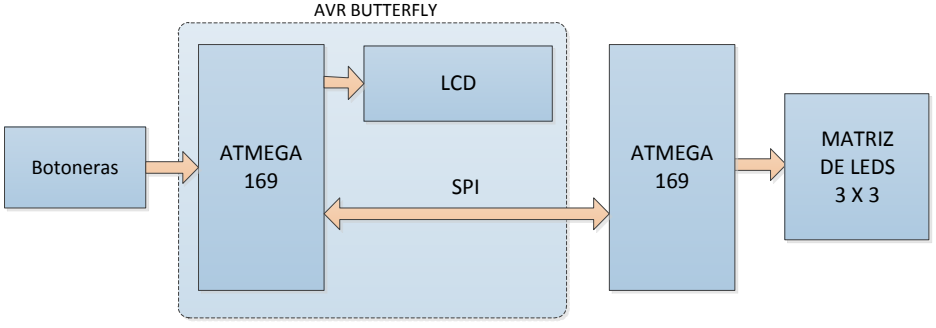
- Descripción

El microcontrolador Atmega169 del AVR Butterfly funciona como el maestro de la comunicación SPI y recibe como entrada la lectura de 6 botoneras conectadas al Puerto F y que corresponden a las piezas de ajedrez: Peón, Caballo, Alfil, Torre, Reina y Rey. Cuando se presiona una de las botoneras, el maestro le envía esta información al esclavo y muestra el nombre de la pieza escogida por el LCD.

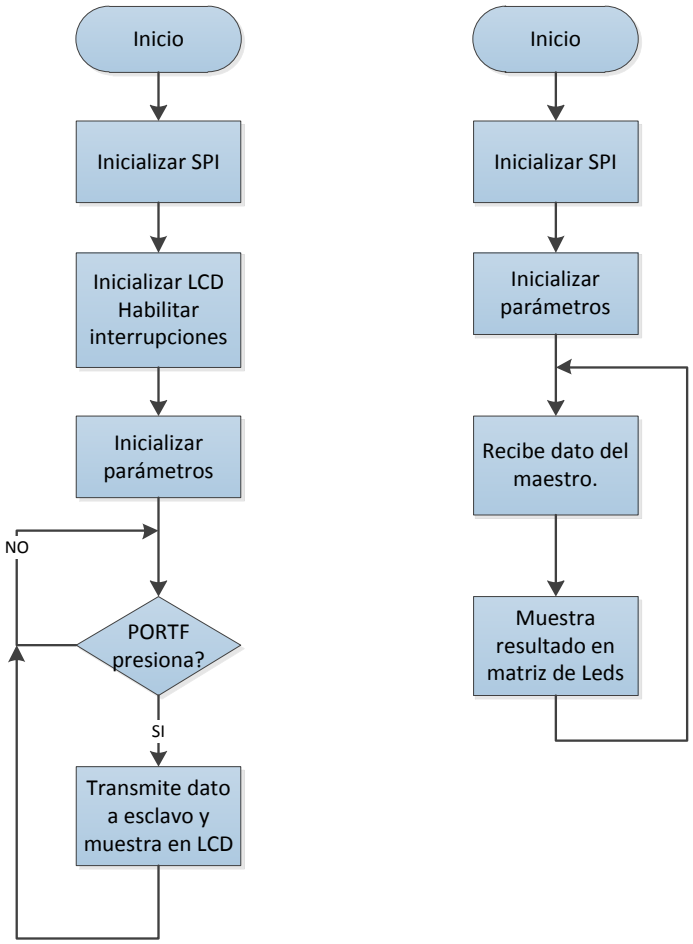
Un segundo microcontrolador Atmega169 trabaja como el esclavo de la comunicación y recibe la información de la botonera seleccionada. Según esta información, simula los movimientos de la pieza de ajedrez respectiva en el tablero, representado por una matriz de leds de 3x3 que se conectan al Puerto D del microcontrolador.

- Diagrama de Bloques

Se puede observar que el AVR Butterfly es el encargado de recibir los datos obtenidos de las botoneras y mostrar los mensajes en el LCD, y mediante la comunicación full-dúplex SPI que realiza con el otro microcontrolador muestra el resultado en la matriz de leds.



- Diagrama de Flujo



MAESTRO **ESCLAVO**

- Descripción del Algoritmo

MAESTRO:

1. Se inicializa la comunicación SPI.
2. Se configura el LCD y se habilitan las interrupciones globales.
3. Se inicializan las variables y los puertos a utilizar.
4. Espera por la selección de una de las botoneras conectadas al Puerto F.
5. Si se presiona una botonera transmite la información al esclavo y muestra el nombre del elemento seleccionado por el LCD.

ESCLAVO:

1. Se inicializa la comunicación SPI.
2. Se inicializan las variables y los puertos a utilizar.
3. Espera por la transmisión de información indefinidamente.
4. Cuando recibe el dato del maestro, muestra el resultado en la matriz de leds que representa el tablero.

- Programa Principal del controlador Maestro

```
/*****************************************************************
 **********       MICROCONTROLADORES AVANZADOS        **********
 *****************************************************************
              Comunicación Serial SPI
 *****************************************************************
                   Ejercicio # 3
 * Nombre: Movimiento de piezas de ajedrez
 * Descripción:
  El maestro recibe el dato de la pieza escogida y se lo
  envía al esclavo. Además muestra el nombre en el LCD.

 *****************************************************************
 ** Nombre del archivo: spi_master_lcd.c
 *****************************************************************/

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

#define pLCDREG ((unsigned char *)(0xEC))
#define TAMANIO_DEL_REGISTRO_LCD 20
#define NUMERO_MAXIMO_DE_CARACTERES 9

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;

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 // '_'
};

void SPI_masterinit(void)
{
  DDRB=0x07;    // MOSI, SS y SCK como salida
  // SPI enable, dispositivo MASTER, Fosc/128
  SPCR=(1<<SPE)|(1<<MSTR)|(1<<SPR0)|(1<<SPR1);

}

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)
{
  // configuración del LCD
  borrar_LCD();
  LCDCRA = (1<<LCDEN) | (1<<LCDAB);
  LCDCCR = (1<<LCDDC2)|(1<<LCDDC1)|(1<<LCDDC0)|(1<<LCDCC3)|(1<<LCDCC2)
  |(1<<LCDCC1)|(1<<LCDCC0);
  ASSR = (1<<AS2);
  LCDFRR = (0<<LCDPS0) | (1<<LCDCD1)|(1<<LCDCD0);
  LCDCRB = (1<<LCDCS)|(1<<LCDMUX1)|(1<<LCDMUX0)|(1<<LCDPM2)|(1<<LCDPM1)
  |(1<<LCDPM0);
  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;

  if ((c >= '*') && (c <= 'z'))
  {
    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 escribir_palabras_en_LCD(char *palabra, int caracter)
{
  unsigned char i=0;

  for( i=0;i<caracter;i++)
  memo_temp_texto[i]='\0';
  LCD_INT_contador = 0;
  ESCRITURA_DE_CADENA_HABILITADO = 1;
  for( i=0;(i<caracter)&&(*palabra!='\0');i++,palabra++)
  memo_temp_texto[i]=*palabra;
}

void borrar_LCD(void)
{
  unsigned char i=0;

  for( i=0;i<NUMERO_MAXIMO_DE_CARACTERES;i++)
  memo_temp_texto[i]='\0';

  for (i = 0; i < TAMANIO_DEL_REGISTRO_LCD; i++)
  {
    *(pLCDREG + i) = 0x00;
    *(LCD_Data+i) = 0x00;
  }
  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);
}

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_ms(100);
  }

  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);
}


int main(void)
{

 SPI_masterinit();
 inicializar_LCD();
 sei();

 char peon[6]="PEON\0", caballo[9]="CABALLO\0", alfil[7]="ALFIL\0";
 char torre[7]="TORRE\0", reina[7]="REINA\0", rey[5]="REY\0";

 int j=0, k=0;

 DDRB=0x0F;
 DDRE=0X00;
 PORTB=0xF0;
 PORTE=0XFF;
 PORTB=0;
 PORTE=0;
```

```
while(1)
{

        j=PINB & 0XF0; // tomar dato de botonera seleccionada
        k=PINE & 7;

        if(j==0x10) // si es el peón
          _delay_ms(500)
         {
                  while (j==0x10)
                  {
                  j=PINB; // espera que se suelte la botonera
                  }
                    borrar_LCD();
                  actualizar_LCD();
           escribir_palabras_en_LCD(peon,6); // muestra en el LCD
           SPI_master_transmit(0x01); // transmite dato
         }

        if(j==0x20) // si es el caballo
         _delay_ms(500);
         {
           while (j==0x20)
                  {
                  j=PINB;
                  }
                  borrar_LCD();
                  actualizar_LCD();
         escribir_palabras_en_LCD(caballo,9);
         SPI_master_transmit(0x02);

         }

        if(j==0x40) // si es el alfil
         _delay_ms(500);
         {
           while (j==0x40)
                  {
                  j=PINB;
                  }
           SPI_master_transmit(0x03);
           borrar_LCD();
          actualizar_LCD();
           escribir_palabras_en_LCD(alfil,7);

         }

        if(j==0x80) // si es la torre
         _delay_ms(500);
         {
           while (j==0x80)
```

```
                {
                j=PINB;
                }
    SPI_master_transmit(0x04);
                borrar_LCD();
                actualizar_LCD();
    escribir_palabras_en_LCD(torre,7);

   }

  if(k==0x01) // si es la reina
    _delay_ms(500);
    {
     while (k==0x01)
                {
                k=PINE;
                }
     SPI_master_transmit(0x05);
                borrar_LCD();
                actualizar_LCD();
     escribir_palabras_en_LCD(reina,7);

    }

  if(k==0x04) // si es el rey
    _delay_ms(500);
    {
     while (k==0x04)
                {
                k=PINE;
                }
     SPI_master_transmit(0x6);
                 borrar_LCD();
                actualizar_LCD();
     escribir_palabras_en_LCD(rey,5);        }
 }
}
```

- Programa Principal del controlador Esclavo

```
/*****************************************************************
 **********       MICROCONTROLADORES AVANZADOS        **********
 *****************************************************************
             Comunicación Serial SPI
 *****************************************************************
                 Ejercicio # 3
 * Nombre: Movimiento de piezas de ajedrez
 * Descripción:
  El esclavo recibe el dato de la pieza escogida del maestro
```

y muestra en un arreglo de leds (tablero) el movimiento.

```
 ***************************************************************
 ** Nombre del archivo: spi_chess_slave.c
 ***************************************************************/

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

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
}

void movimiento(unsigned char n)
 {
   // movimientos de cada una de las piezas
   switch (n)
   {
     case 0x01:
            {
              PORTD=0x40;
              _delay_ms(300);
              PORTD=0x08;
              _delay_ms(300);
              PORTD=0;
              _delay_ms(300);
              PORTD=0x40;
              _delay_ms(300);
              PORTD=0x01;
              _delay_ms(300);
              PORTD=0;
              _delay_ms(300);
              PORTD=0x40;
              _delay_ms(300);
              PORTD=0x10;
              _delay_ms(300);
              PORTD=0x00;
              break;
    }

            case 0x02:
            {
              PORTD=0x40;
```

```
          _delay_ms(300);
          PORTD=0x08;
          _delay_ms(300);
          PORTD=0x01;
          _delay_ms(300);
          PORTD=0x02;
          _delay_ms(300);
          PORTD=0;
          _delay_ms(300);
          PORTD=0x40;
          _delay_ms(300);
          PORTD=0x80;
          _delay_ms(300);
          PORTD=0x00;
          PORTE=0x01;
          _delay_ms(300);
          PORTE=0x00;
          PORTD=0x20;
          _delay_ms(300);
          PORTD=0x00;
          break;
}

        case 0x03:
         {
          PORTD=0x40;
          _delay_ms(300);
          PORTD=0x10;
          _delay_ms(300);
          PORTD=0x04;
          _delay_ms(300);
          PORTD=0;
          _delay_ms(300);
          PORTE=1;
          _delay_ms(300);
          PORTE=0;
          PORTD=0x10;
          _delay_ms(300);
          PORTD=0x01;
          _delay_ms(300);
          PORTD=0x00;
          break;
}

         case 0x04:
         {
          PORTD=0x40;
          _delay_ms(300);
          PORTD=0x08;
          _delay_ms(300);
          PORTD=0x01;
```

```
  _delay_ms(300);
  PORTD=0;
  _delay_ms(300);
  PORTD=0X40;
  _delay_ms(300);
  PORTD=0X80;
  _delay_ms(300);
  PORTD=0;
  PORTE=1;
  _delay_ms(300);
  PORTD=0;
  PORTE=0;
  break;

}

        case 0x05:
        {
  PORTD=0x40;
  _delay_ms(300);
  PORTD=0x08;
  _delay_ms(300);
  PORTD=0x01;
  _delay_ms(300);
  PORTD=0;
  _delay_ms(300);
  PORTD=0x40;
  _delay_ms(300);
  PORTD=0x10;
  _delay_ms(300);
  PORTD=0x04;
  _delay_ms(300);
  PORTD=0;
  _delay_ms(300);
  PORTD=0X40;
  _delay_ms(300);
  PORTD=0X80;
  _delay_ms(300);
  PORTD=0;
  PORTE=1;
  _delay_ms(300);
  PORTD=0;
  PORTE=0;
  break;
}

        case 0x06:
        {
  PORTD=0x10;
  _delay_ms(300);
  PORTD=0x40;
```

```
            _delay_ms(300);
            PORTD=0;
            _delay_ms(300);
            PORTD=0X10;
            _delay_ms(300);
            PORTD=0X80;
            _delay_ms(300);
            PORTD=0;
            _delay_ms(300);
            PORTD=0x10;
            _delay_ms(300);
            PORTD=0;
            PORTE=1;
            _delay_ms(300);
            PORTD=0;
            PORTE=0;
            _delay_ms(300);
            PORTD=0x10;
            _delay_ms(300);
            PORTD=0x20;
            _delay_ms(300);
            PORTD=0;
            _delay_ms(300);
            PORTD=0X10;
            _delay_ms(300);
            PORTD=0X04;
            _delay_ms(300);
            PORTD=0;
            _delay_ms(300);
            PORTD=0X10;
            _delay_ms(300);
            PORTD=0X02;
            _delay_ms(300);
            PORTD=0;
            _delay_ms(300);
            PORTD=0X10;
            _delay_ms(300);
            PORTD=0X01;
            _delay_ms(300);
            PORTD=0;
            _delay_ms(300);
            PORTD=0X10;
            _delay_ms(300);
            PORTD=0X08;
            _delay_ms(300);
            PORTD=0;
            break;
    }
  }
}
```

```
int main(void)
{
 SPI_slave_init();
 unsigned char rx_data;
 unsigned char n=0;

 DDRD=0xFF;
 DDRE=0xFF;
 PORTD=0x00;
 PORTE=0x00;



 while(1)
 {
  n=SPI_slave_receive();

        if(n==0x01)  // si dato recibido es 0x01
         {
          movimiento(n); // realiza el movimiento del peón
         }
        if(n==0x02)  // si dato recibido es 0x02
         {
          movimiento(n); // realiza el movimiento del caballo
         }
        if(n==0x03)  // si dato recibido es 0x03
         {
          movimiento(n); // realiza el movimiento del alfil
         }
        if(n==0x04)  // si dato recibido es 0x04
         {
          movimiento(n); // realiza el movimiento de la torre
         }
        if(n==0x05)  // si dato recibido es 0x05
         {
          movimiento(n); // realiza el movimiento de la reina
         }
        if(n==0x6)  // si dato recibido es 0x06
         {
          movimiento(n); // realiza el movimiento del rey
         }
 }
}
```

4.1.4. Contador de dos dígitos.

- Descripción

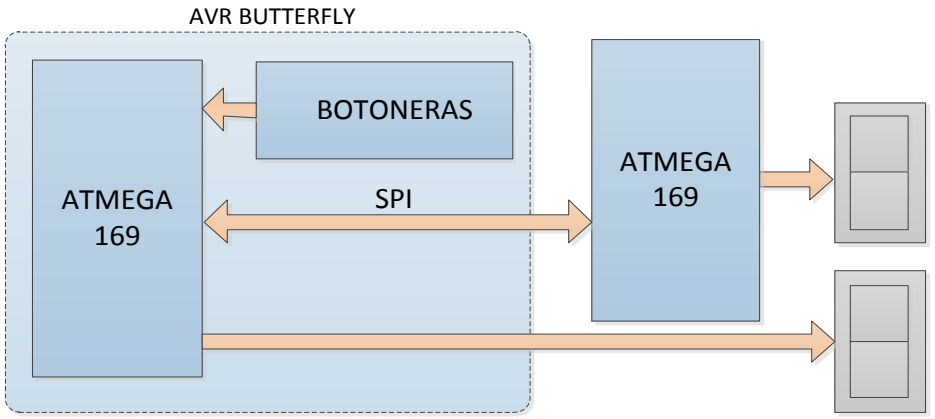
Dependiendo de la botonera seleccionada en el Puerto B del AVR Butterfly, el cual funciona como el maestro del SPI, se incrementa o decrementa el contador de dos dígitos en una o dos unidades de la siguiente manera:

- INC1 = Incrementa 1 unidad
- DEC1 = Decrementa 1 unidad
- DEC2 = Decrementa 2 unidades
- INC2 = Incrementa 2 unidades

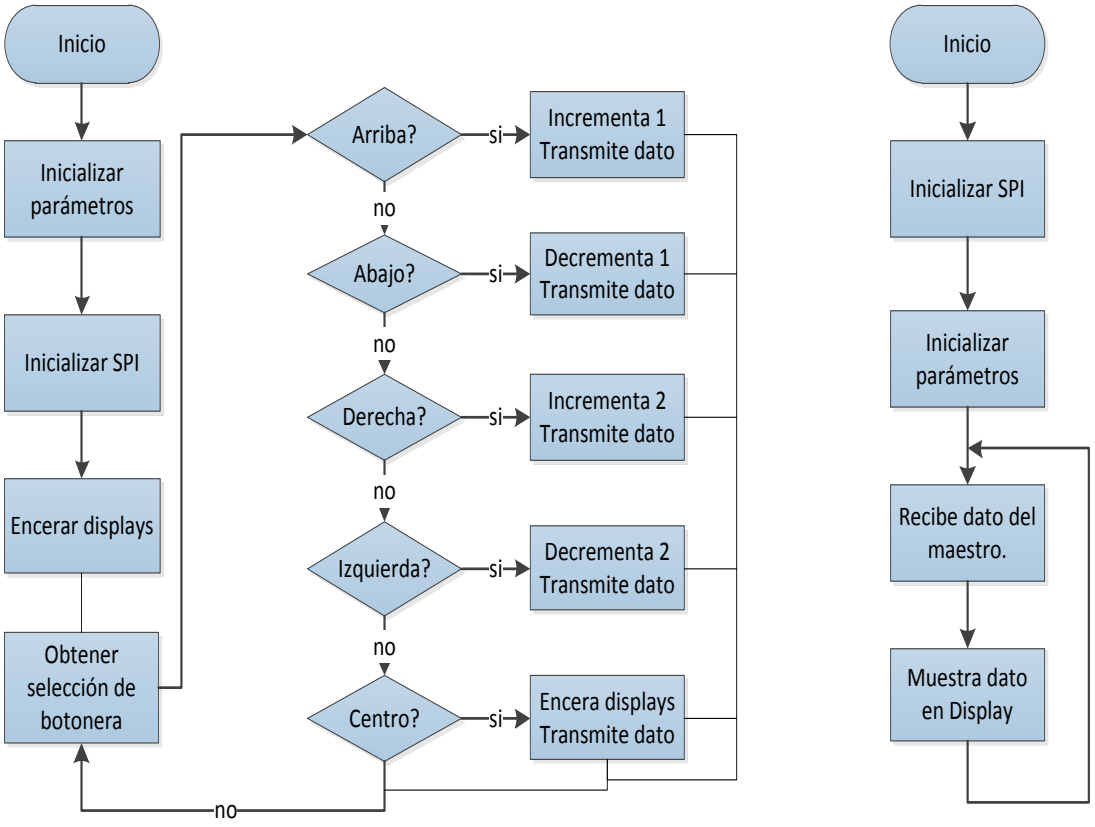
El dígito correspondiente a las unidades se muestra en un display conectado al Puerto D del Butterfly. El esclavo de la comunicación SPI, que en este caso es un microcontrolador Atmega169, recibe el dato del dígito correspondiente a las decenas y lo muestra en un display conectado a su Puerto D.

- Diagrama de Bloques

El AVR Butterfly recibe la información del joystick y según esto, realiza el incremento o decremento del contador mostrando por uno de sus puertos de salida un dígito. El otro dígito lo muestra el esclavo después de haber recibido el dato.



- Diagrama de Flujo



MAESTRO **ESCLAVO**

- Descripción del Algoritmo

MAESTRO:

1. Se inicializan las variables y los puertos a utilizar.
2. Se inicializa la comunicación SPI.
3. Encera las variables y muestra displays en cero.
4. Espera por la selección de una de las botoneras conectadas al Puerto B.
5. Se obtiene selección de la botonera presionada y confirma que la selección sea válida.
6. Dependiendo de la selección, incrementa o decrementa el contador de dos dígitos.
7. Muestra el dígito de las unidades en un display conectado al Puerto D.
8. Envía dígito de las decenas por comunicación SPI al esclavo.
9. Se repite procedimiento desde el paso 4.

ESCLAVO:

1. Se inicializa la comunicación SPI.
2. Se inicializan las variables y los puertos a utilizar.
3. Espera por la transmisión de información indefinidamente.
4. Cuando recibe la información del maestro, muestra el dato en un display conectado a uno de sus puertos de salida.

- Programa Principal del controlador Maestro

```
/*****************************************************************
 **********      MICROCONTROLADORES AVANZADOS       **********
 *****************************************************************
              Comunicación Serial SPI
 *****************************************************************
                  Ejercicio # 3
 * Nombre: Movimiento de piezas de ajedrez
 * Descripción:
  El maestro recibe el dato de la pieza escogida y se lo
  envía al esclavo. Además muestra el nombre en el LCD.

 *****************************************************************
 ** Nombre del archivo: spi_master_lcd.c
 *****************************************************************/

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


// Variables globales:
unsigned char uni, dec, unid, decd;


// Funciones:

void SPI_masterinit(void)
{
  DDRB=0x07;    // MOSI, SS y SCK como salida
  // SPI enable, dispositivo MASTER, Fosc/128
```

```
 SPCR=(1<<SPE)|(1<<MSTR)|(1<<SPR0)|(1<<SPR1);
 }


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();
 //inicializar_LCD();
 //sei();


 int j=0;

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

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


 while(1)
 {

        j=PINB & 0XF0; // tomar dato de botonera seleccionada

        switch(j) // realiza conteo dependiendo de la selección
  {
   case 0x10:  // incrementa en 1 unidad
         {
                _delay_ms(500);
                while (j==0x10)
                {
                j=PINB;

                }
   uni++;
```

```
if (uni==0x0A)
 {   dec++;
     uni=0;
 }
unid=numeros[uni];  // conversión binario
decd=numeros[dec];  // a 7 segmentos
PORTD=unid;        // muestra en display
SPI_master_transmit(decd);  // transmite dato
break;

case 0x20:   // decrementa en 1 unidad
{
                _delay_ms(500);
                while (j==0x20)
                {
                j=PINB;
                }
if (uni==0 & dec==0)  // valida números negativos
 { uni=0;
   dec=0;          // encera
   unid=numeros[uni]; // conversión
        decd=numeros[dec];
   PORTD=unid;      // muestra cero
        SPI_master_transmit(decd); // transmisión
   break;}
uni--;    // decrementa 1
if (uni==0xFF)
 {   dec--;
     uni=9;
 }
unid=numeros[uni]; // conversión
decd=numeros[dec];
PORTD=unid;      // muestra número
SPI_master_transmit(decd);
break;


case 0x40:   // decrementa en 2 unidades
 {
                _delay_ms(500);
                while (j==0x40)
                {
                j=PINB;
                }
        if (uni==1 & dec==0 || uni==0 & dec==0) // valida números negativos
 { uni=0;
   dec=0;     // encera
   unid=numeros[uni]; // conversión
   decd=numeros[dec];
   PORTD=unid;       // muestra número
   SPI_master_transmit(decd);
```

```
        break;}
      uni=uni-2;  //decrementa 2
    if (uni==0xFF)
     {   dec--;
        uni=9;
     }
    if (uni==0xFE)
     {   dec--;
        uni=8;
     }
    unid=numeros[uni];  // conversión
    decd=numeros[dec];
    PORTD=unid;        // muestra número
    SPI_master_transmit(decd);  // transmisión
    break;



    case 0x80:   // incrementa en 2 unidades
    {
                    _delay_ms(500);
                    while (j==0x80)
                    {
                    j=PINB;
                    }
          uni=uni+2;
    if (uni==0x0A)
     {   dec++;
        uni=0;
     }
    if (uni==0x0B)
     {   dec++;
        uni=1;
     }
    unid=numeros[uni];  // conversión
    decd=numeros[dec];
    PORTD=unid;        // muestra número
    SPI_master_transmit(decd); // transmisión
    break;

          default:
    break;
   }

  }

}
}
}
}
```

```
}
```

- Programa Principal del controlador Esclavo

```
/*****************************************************************
 **********       MICROCONTROLADORES AVANZADOS        **********
 *****************************************************************
                Comunicación Serial SPI
 *****************************************************************
                    Ejercicio # 4
 * Nombre: Contador de dos Dígitos
 * Descripción:
  El esclavo de la comunicación SPI recibe el dígito correspon-
  diente a las decenas y lo muestra en un display conectado al
  PortD .

 *****************************************************************
 ** Nombre del archivo: cont_slave.c
 *****************************************************************/

#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
  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
}
```

4.1.5. Mensaje en Matriz de Leds de 8x8.

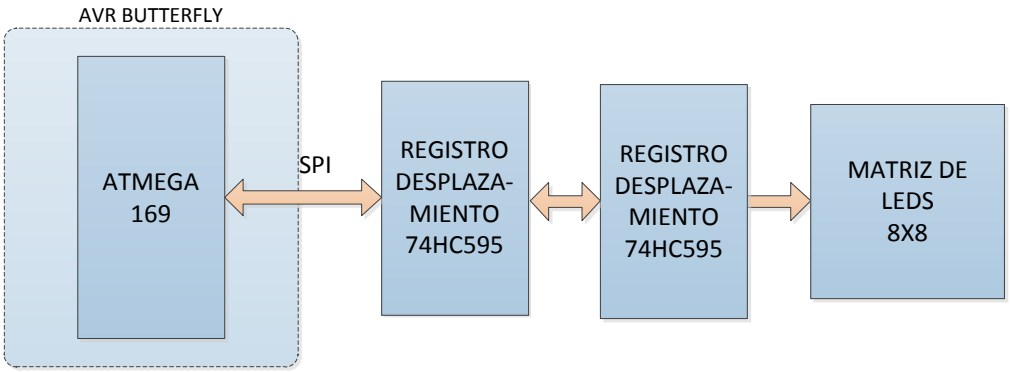
- Descripción

El maestro AVR Butterfly envía mediante comunicación SPI, una serie de datos a dos registros de desplazamiento 74HC595 que funcionan como esclavos, y que representan un mensaje. Este mensaje se va desplazando a través de la matriz de leds de 8x8 hasta que se logra visualizar completamente, para luego volverlo a mostrar desde su principio indefinidamente.

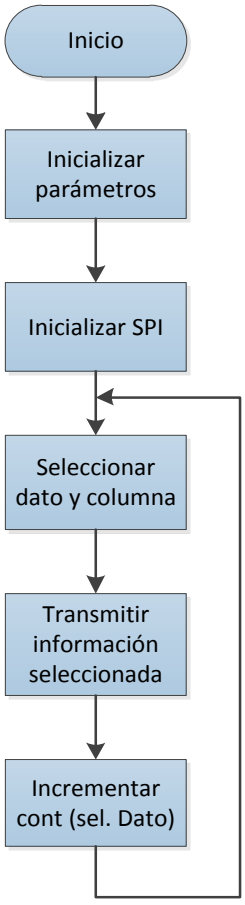
El maestro envía 16 bits de información: los primeros 8 bits al primer esclavo (74HC595) que representan el mensaje a mostrar, y los siguientes 8 bits al segundo esclavo que representan el número de columna de la matriz donde debe llegar el dato. La matriz de leds es conectada al segundo esclavo de la comunicación.

- Diagrama de Bloques

El AVR Butterfly recibe la información del joystick y según esto, realiza el incremento o decremento del contador mostrando por uno de sus puertos de salida un dígito. El otro dígito lo muestra el esclavo después de haber recibido el dato.



- Diagrama de Flujo



- Descripción del Algoritmo

1. Se inicializan las variables y los puertos a utilizar.
2. Se inicializa la comunicación SPI.
3. Se selecciona dato y columna a mostrar en la matriz.

4. Se transmite la información seleccionada mediante la comunicación SPI.
5. Se incrementa el contador, que es el que selecciona el dato a mostrar.
6. Se repiten los pasos desde el 3 indefinidamente debido a que se encuentran en un lazo While.

- Programa Principal del controlador

```
/*****************************************************************
 **********      MICROCONTROLADORES AVANZADOS       **********
 *****************************************************************
              Comunicación Serial SPI
 *****************************************************************
                  Ejercicio # 5
 * Nombre: Mensaje Móvil en Matriz de Leds 8x8
 * Descripción:
  El esclavo muestra un mensaje móvil en una matriz de Leds de
  8x8 que le envía el maestro AVR Butterfly mediante SPI.

 *****************************************************************
 ** Nombre del archivo: matriz_msj.c
 *****************************************************************/

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>


char mensaje[] = {
        // Mensaje: MICROCONTROLADORES Ɵ
0x00,0x7E,0x60,0x18,0x06,0x18,0x60,0x7E,
0x7E,0x60,0x18,0x06,0x18,0x60,0x7E,0x00,
0x60,0x18,0x06,0x18,0x60,0x7E,0x00,0x7E,
0x18,0x06,0x18,0x60,0x7E,0x00,0x7E,0x00,
0x06,0x18,0x60,0x7E,0x00,0x7E,0x00,0x3C,
0x18,0x60,0x7E,0x00,0x7E,0x00,0x3C,0x42,
0x60,0x7E,0x00,0x7E,0x00,0x3C,0x42,0x42,
0x7E,0x00,0x7E,0x00,0x3C,0x42,0x42,0x42,
0x00,0x7E,0x00,0x3C,0x42,0x42,0x42,0x24,
0x7E,0x00,0x3C,0x42,0x42,0x42,0x24,0x00,
0x00,0x3C,0x42,0x42,0x42,0x24,0x00,0x7E,
0x3C,0x42,0x42,0x42,0x24,0x00,0x7E,0x48,
0x42,0x42,0x42,0x24,0x00,0x7E,0x48,0x48,
0x42,0x42,0x24,0x00,0x7E,0x48,0x48,0x48,
0x42,0x24,0x00,0x7E,0x48,0x48,0x48,0x36,
0x24,0x00,0x7E,0x48,0x48,0x48,0x36,0x00,
0x00,0x7E,0x48,0x48,0x48,0x36,0x00,0x3C,
0x7E,0x48,0x48,0x48,0x36,0x00,0x3C,0x42,
```

```
0x48,0x48,0x48,0x36,0x00,0x3C,0x42,0x42,
0x48,0x48,0x36,0x00,0x3C,0x42,0x42,0x42,
0x48,0x36,0x00,0x3C,0x42,0x42,0x42,0x3C,
0x36,0x00,0x3C,0x42,0x42,0x42,0x3C,0x00,
0x00,0x3C,0x42,0x42,0x42,0x3C,0x00,0x3C,
0x3C,0x42,0x42,0x42,0x3C,0x00,0x3C,0x42,
0x42,0x42,0x3C,0x00,0x3C,0x42,0x42,0x42,
0x42,0x3C,0x00,0x3C,0x42,0x42,0x42,0x24,
0x3C,0x00,0x3C,0x42,0x42,0x42,0x24,0x00,
0x00,0x3C,0x42,0x42,0x42,0x24,0x00,0x3C,
0x3C,0x42,0x42,0x42,0x24,0x00,0x3C,0x42,
0x42,0x42,0x42,0x24,0x00,0x3C,0x42,0x42,
0x42,0x42,0x24,0x00,0x3C,0x42,0x42,0x42,
0x42,0x24,0x00,0x3C,0x42,0x42,0x42,0x3C,
0x24,0x00,0x3C,0x42,0x42,0x42,0x3C,0x00,
0x00,0x3C,0x42,0x42,0x42,0x3C,0x00,0x7E,
0x3C,0x42,0x42,0x42,0x3C,0x00,0x7E,0x60,
0x42,0x42,0x42,0x3C,0x00,0x7E,0x60,0x18,
0x42,0x42,0x3C,0x00,0x7E,0x60,0x18,0x06,
0x42,0x3C,0x00,0x7E,0x60,0x18,0x06,0x7E,
0x3C,0x00,0x7E,0x60,0x18,0x06,0x7E,0x40,
0x00,0x7E,0x60,0x18,0x06,0x7E,0x40,0x40,
0x7E,0x60,0x18,0x06,0x7E,0x40,0x40,0x7E,
0x60,0x18,0x06,0x7E,0x40,0x40,0x7E,0x40,
0x18,0x06,0x7E,0x40,0x40,0x7E,0x40,0x40,
0x06,0x7E,0x40,0x40,0x7E,0x40,0x40,0x00,
0x7E,0x40,0x40,0x7E,0x40,0x40,0x00,0x7E,
0x40,0x40,0x7E,0x40,0x40,0x00,0x7E,0x48,
0x40,0x7E,0x40,0x40,0x00,0x7E,0x48,0x48,
0x7E,0x40,0x40,0x00,0x7E,0x48,0x48,0x48,
0x40,0x40,0x00,0x7E,0x48,0x48,0x48,0x36,
0x40,0x00,0x7E,0x48,0x48,0x48,0x36,0x00,
0x00,0x7E,0x48,0x48,0x48,0x36,0x00,0x3C,
0x7E,0x48,0x48,0x48,0x36,0x00,0x3C,0x42,
0x48,0x48,0x48,0x36,0x00,0x3C,0x42,0x42,
0x48,0x48,0x36,0x00,0x3C,0x42,0x42,0x42,
0x48,0x36,0x00,0x3C,0x42,0x42,0x42,0x3C,
0x36,0x00,0x3C,0x42,0x42,0x42,0x3C,0x00,
0x00,0x3C,0x42,0x42,0x42,0x3C,0x00,0x7E,
0x3C,0x42,0x42,0x42,0x3C,0x00,0x7E,0x02,
0x42,0x42,0x42,0x3C,0x00,0x7E,0x02,0x02,
0x42,0x42,0x3C,0x00,0x7E,0x02,0x02,0x00,
0x42,0x3C,0x00,0x7E,0x02,0x02,0x00,0x06,
0x3C,0x00,0x7E,0x02,0x02,0x00,0x06,0x18,
0x00,0x7E,0x02,0x02,0x00,0x06,0x18,0x68,
0x7E,0x02,0x02,0x00,0x06,0x18,0x68,0x18,
0x02,0x02,0x00,0x06,0x18,0x68,0x18,0x06,
0x02,0x00,0x06,0x18,0x68,0x18,0x06,0x00,
0x00,0x06,0x18,0x68,0x18,0x06,0x00,0x7E,
0x06,0x18,0x68,0x18,0x06,0x00,0x7E,0x42,
0x18,0x68,0x18,0x06,0x00,0x7E,0x42,0x42,
```

```
0x68,0x18,0x06,0x00,0x7E,0x42,0x42,0x42,
0x18,0x06,0x00,0x7E,0x42,0x42,0x42,0x3C,
0x06,0x00,0x7E,0x42,0x42,0x42,0x3C,0x00,
0x00,0x7E,0x42,0x42,0x42,0x3C,0x00,0x3C,
0x7E,0x42,0x42,0x42,0x3C,0x00,0x3C,0x42,
0x42,0x42,0x42,0x3C,0x00,0x3C,0x42,0x42,
0x42,0x42,0x3C,0x00,0x3C,0x42,0x42,0x42,
0x42,0x3C,0x00,0x3C,0x42,0x42,0x42,0x3C,
0x3C,0x00,0x3C,0x42,0x42,0x42,0x3C,0x00,
0x00,0x3C,0x42,0x42,0x42,0x3C,0x00,0x7E,
0x3C,0x42,0x42,0x42,0x3C,0x00,0x7E,0x48,
0x42,0x42,0x42,0x3C,0x00,0x7E,0x48,0x48,
0x42,0x42,0x3C,0x00,0x7E,0x48,0x48,0x48,
0x42,0x3C,0x00,0x7E,0x48,0x48,0x48,0x36,
0x3C,0x00,0x7E,0x48,0x48,0x48,0x36,0x00,
0x00,0x7E,0x48,0x48,0x48,0x36,0x00,0x7E,
0x7E,0x48,0x48,0x48,0x36,0x00,0x7E,0x52,
0x48,0x48,0x48,0x36,0x00,0x7E,0x52,0x52,
0x48,0x48,0x36,0x00,0x7E,0x52,0x52,0x42,
0x48,0x36,0x00,0x7E,0x52,0x52,0x42,0x00,
0x36,0x00,0x7E,0x52,0x52,0x42,0x00,0x24,
0x00,0x7E,0x52,0x52,0x42,0x00,0x24,0x52,
0x7E,0x52,0x52,0x42,0x00,0x24,0x52,0x4A,
0x52,0x52,0x42,0x00,0x24,0x52,0x4A,0x24,
0x52,0x42,0x00,0x24,0x52,0x4A,0x24,0X00,
0x42,0x00,0x24,0x52,0x4A,0x24,0X00,0X00,
0x00,0x24,0x52,0x4A,0x24,0X00,0X00,0X00,
0x24,0x52,0x4A,0x24,0X00,0X00,0X00,0x00,
0x52,0x4A,0x24,0X00,0X00,0X00,0x00,0x00,
0x4A,0x24,0X00,0X00,0X00,0x00,0x00,0x00,
0x24,0X00,0X00,0X00,0x00,0x00,0x00,0x00,
0x3C,0x42,0x81,0xA1,0x89,0x95,0xA5,0x42,
0x3C,0x42,0x81,0xA1,0x89,0x95,0xA5,0x42,
0x3C,0x42,0x81,0xA1,0x89,0x95,0xA5,0x42,
0x3C,0x42,0x81,0xA1,0x89,0x95,0xA5,0x42,
0x3C,0x42,0x81,0xA1,0x89,0x95,0xA5,0x42,
0x3C,0x42,0x81,0xA1,0x89,0x95,0xA5,0x42,

};


// Funciones:
void spi_init(void) {
  // configuración del SPI
  // dispositivo Master, modo de reloj= 3, f_sck = f_osc / 4
  // deshabilitado SPI interrupt, orden del dato = MSB primero
  SPCR = _BV(SPE) | _BV(MSTR) | _BV(CPOL) | _BV(CPHA);

}

void spi_begin(void) {
```

```
  // setear /SS = bajo para seleccionarlo como esclavo.
  PORTB &= 0b11111110;
}

void spi_end(void) {

  PORTB |= 0b00100000;  // apagar LEDs /OE = 1,
  PORTB |= 0b00000001;  // setear /SS = 1,
  PORTB &= 0b11011111;  // prender LEDs /OE = 0
}

char spi_send(char cData) {
  SPDR = cData;

  while(!(SPSR & (1<<SPIF))); // esperar que se complete la transmisión
  return SPDR;  // retorna dato del esclavo SPI
}

// Programa Principal:
int main( void )
{
  char msj, col;
  int linea, cont;

  PORTB = 0;         // setting port B;
  DDRB  = 0b00100111; // salidas = MOSI, SCK, /SS, PB5(/OE)

  cont = 0;
  spi_init();   // inicializa SPI

  while(1){

  linea = cont& 0x07; // selecciona columna de LED (0 .. 7)

  msj = mensaje[cont];
  col = ~ (1 << linea);  // columna seleccionada en bajo, las demás en alto

   // SPI transferencia (16 bits)
  spi_begin();
  spi_send(msj);
  spi_send(col);
  spi_end();

  if(cont==0b1101010110)  // si llegué al final,
    cont=0x00;         // ir al inicio del mensaje

  cont++;
  _delay_ms(15);
  }

}
```

4.2. Simulación de los Ejercicios

En esta etapa se pueden observar las simulaciones de cada uno de los ejercicios que conforman este proyecto y que se realizaron previos a la implementación del proyecto en físico y que nos ayudó a tener una mejor idea de sus modos de operación y correctos funcionamientos.

4.2.1. Rotación de Leds.

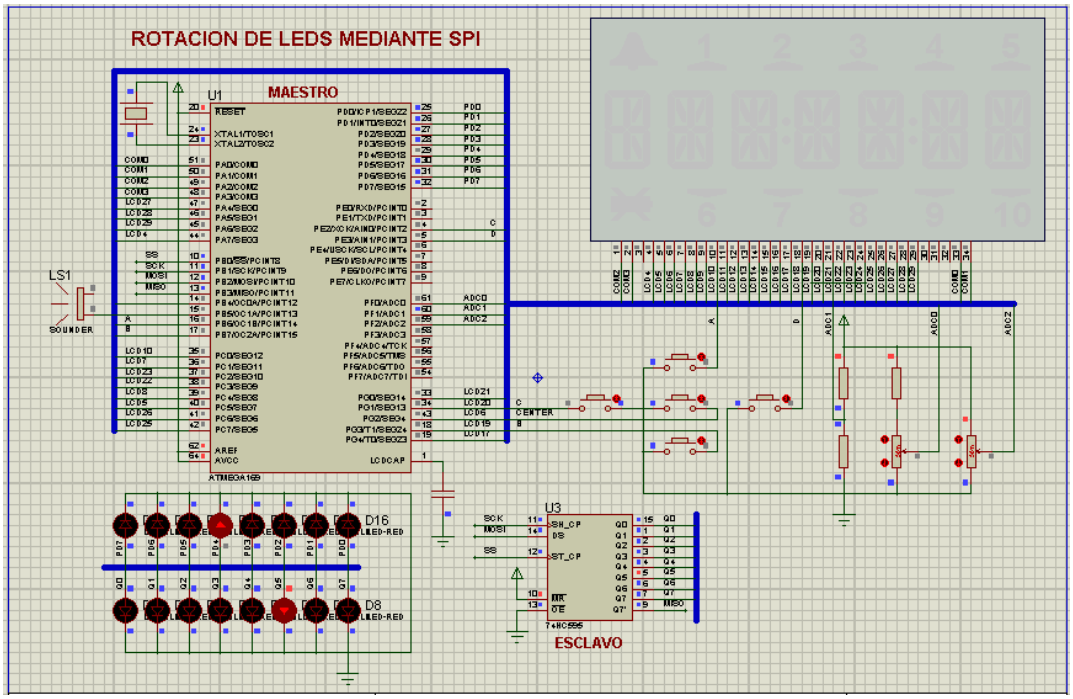


Figura 4.1.: Simulación del Ejercicio 1

4.2.2. Símbolos en Matriz de Leds 8x8.

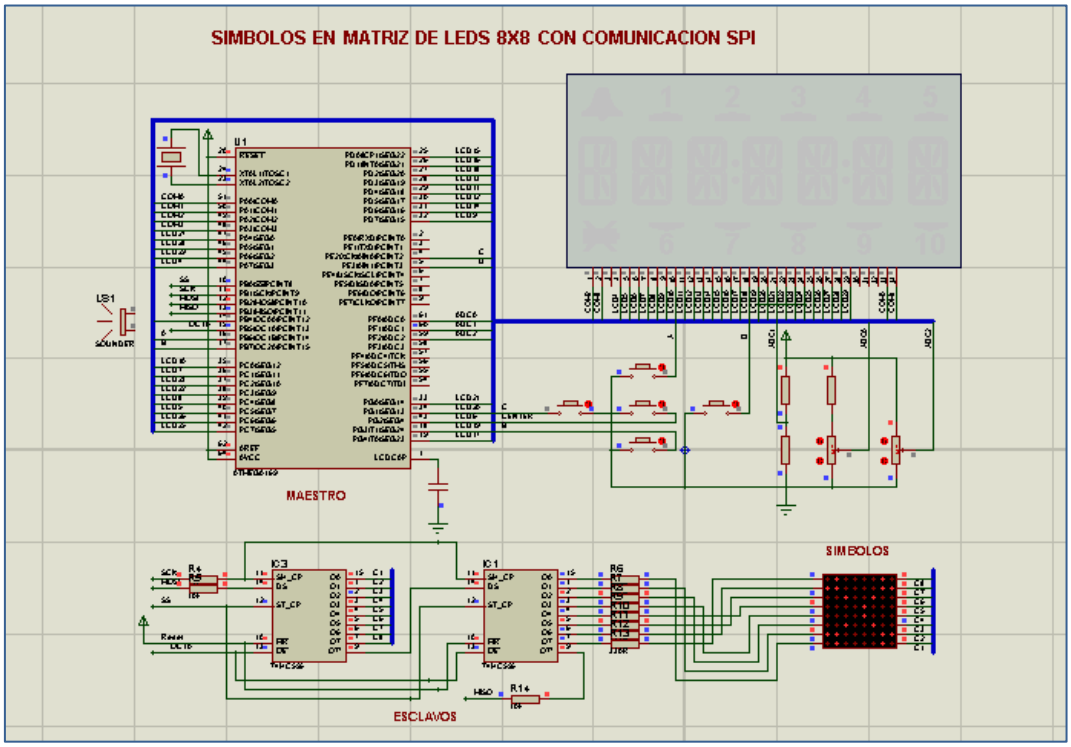


Figura 4.2.: Simulación del Ejercicio 2

4.2.3. Movimiento de las piezas de Ajedrez.

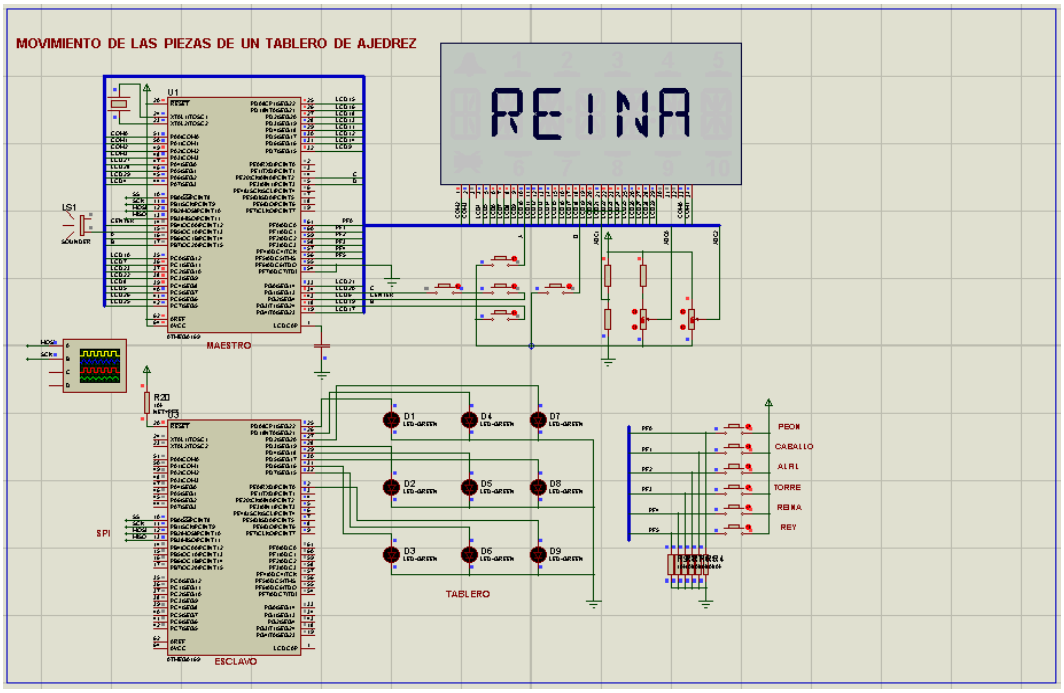


Figura 4.3.: Simulación I del Ejercicio 3

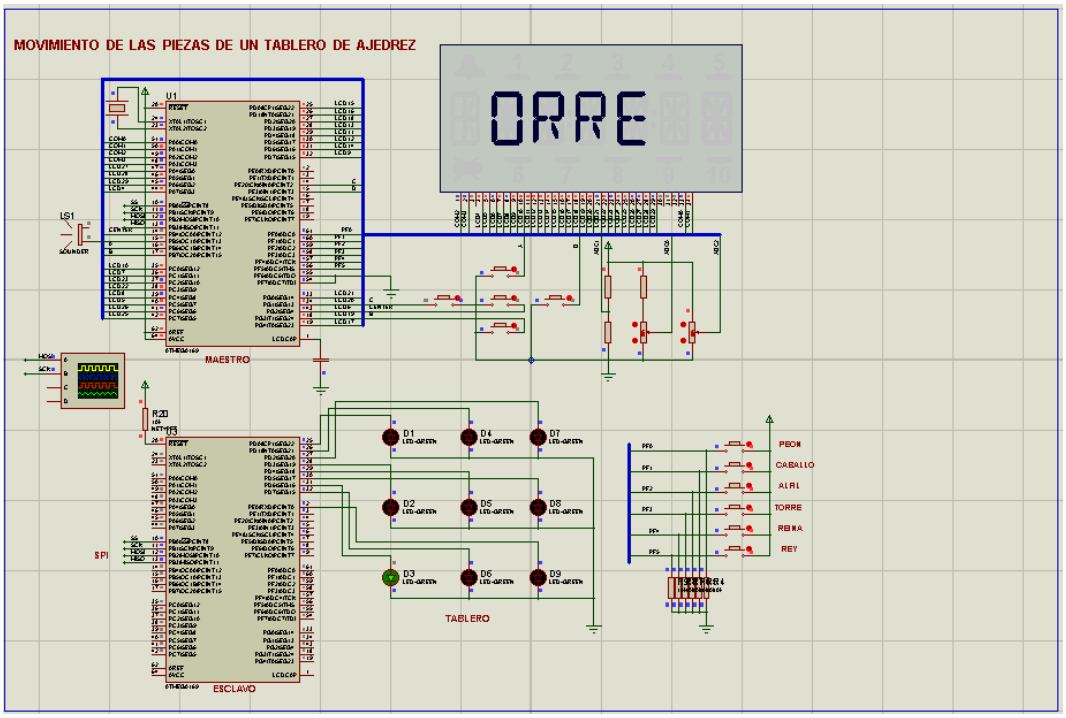


Figura 4.4.: Simulación II del Ejercicio 3

4.2.4. Contador de dos dígitos.

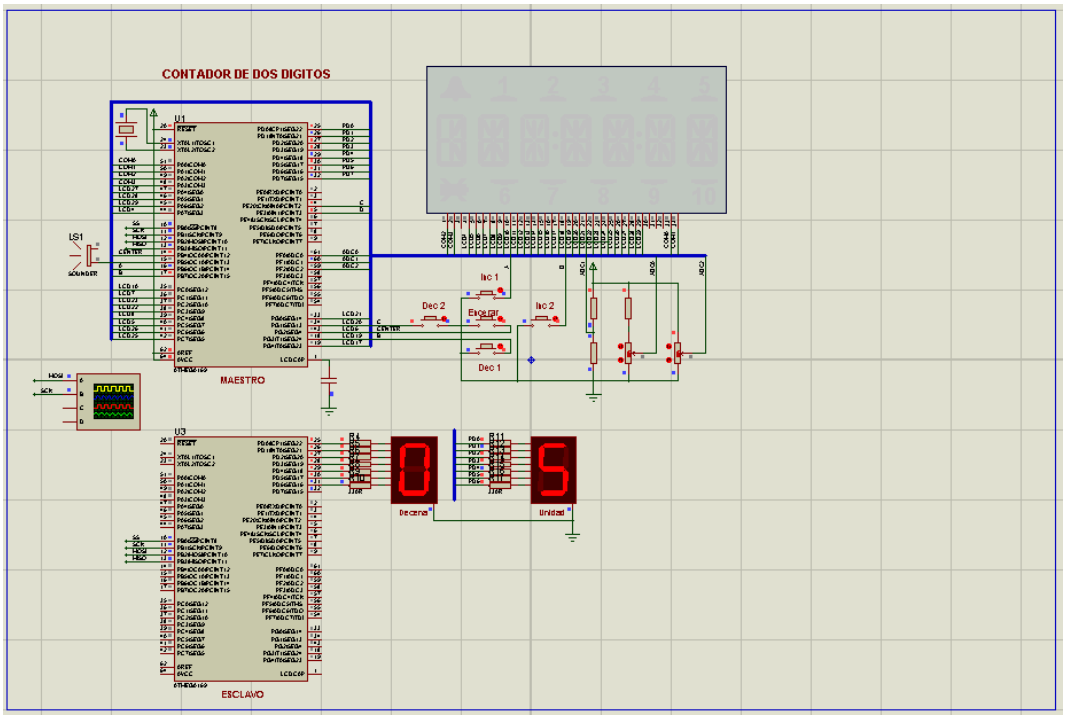


Figura 4.5.: Simulación I del Ejercicio 4

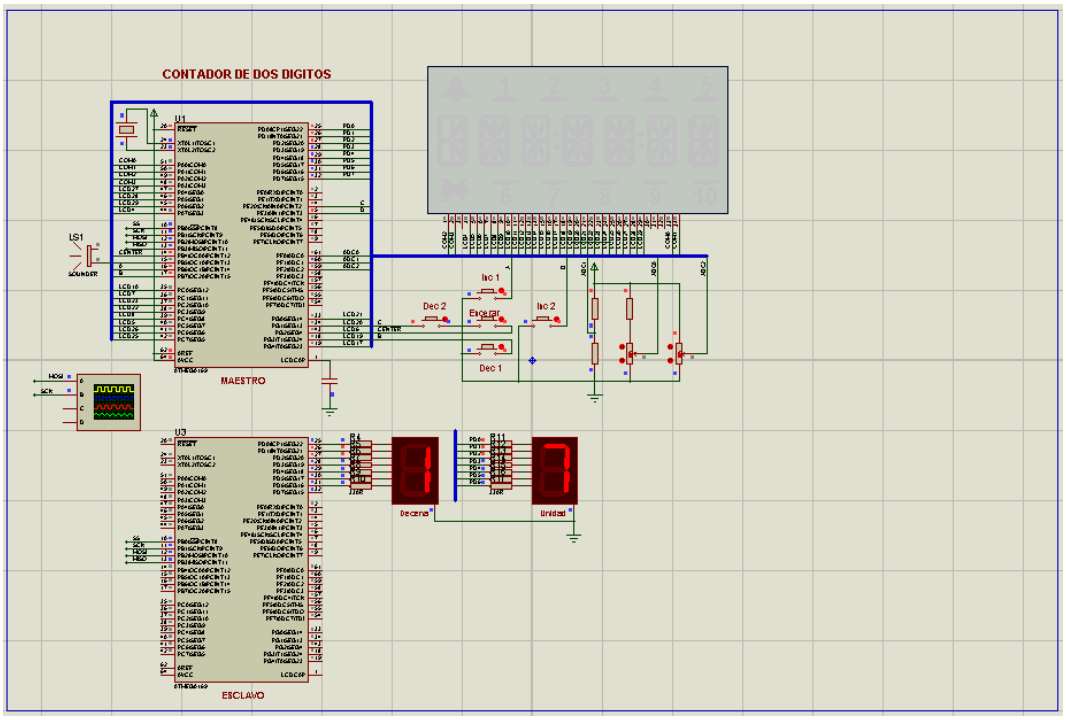


Figura 4.6.: Simulación II del Ejercicio 4

4.2.5. Mensaje en Matriz de Leds de 8x8.

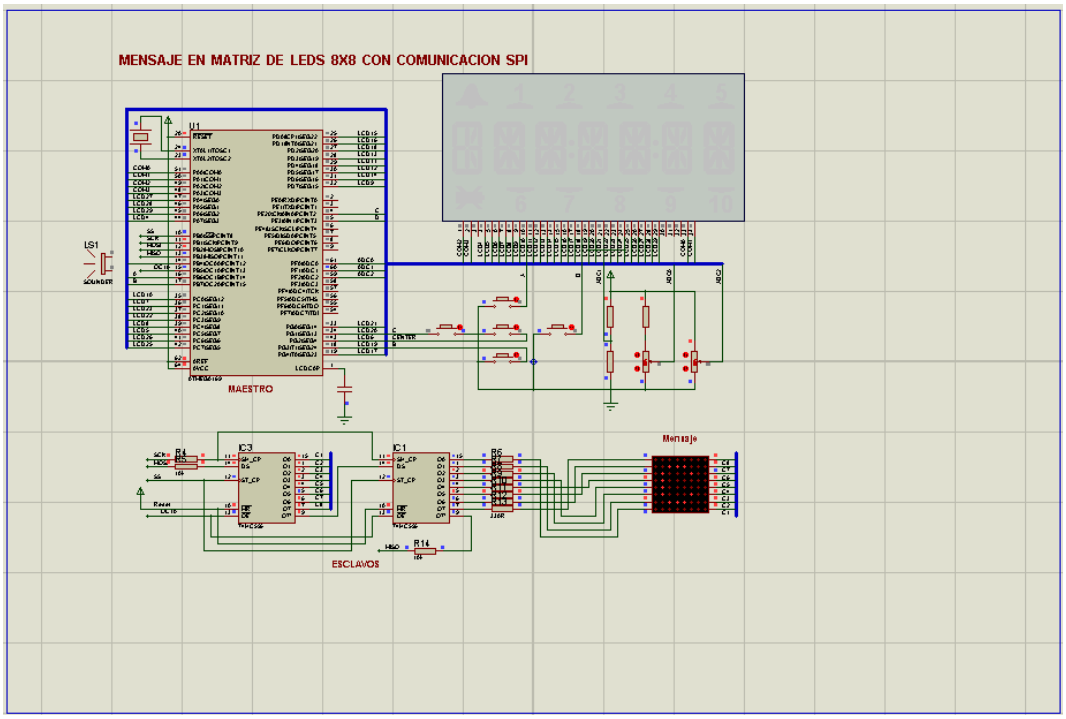


Figura 4.7.: Simulación I del Ejercicio 5

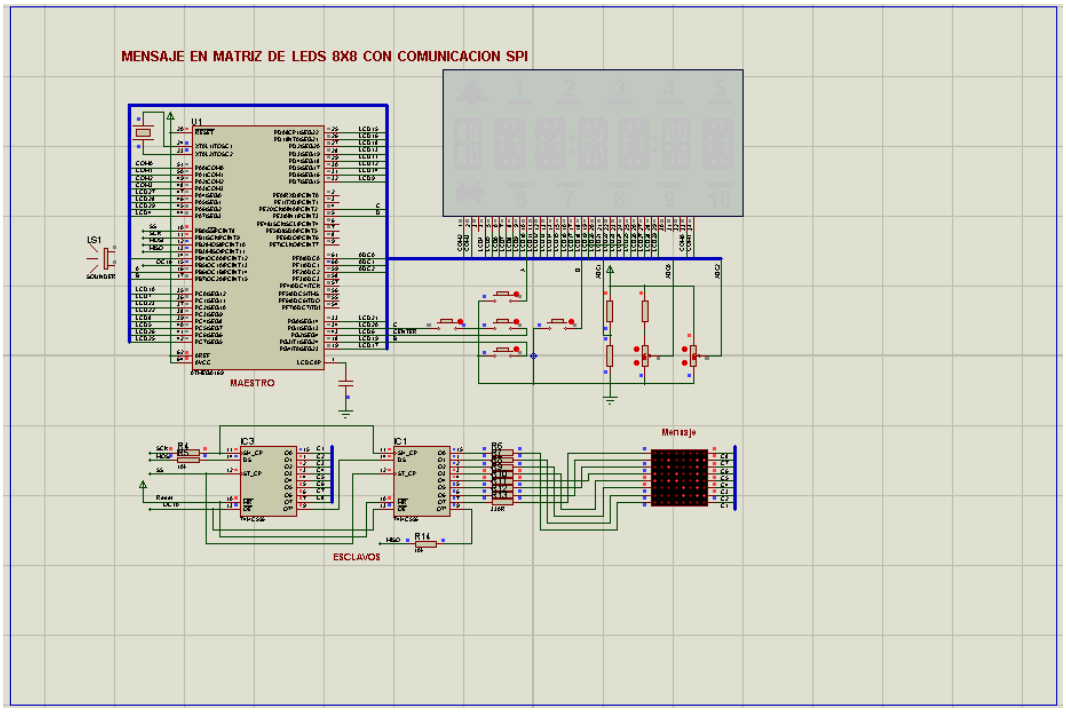


Figura 4.8.: Simulación II del Ejercicio 5

Conclusiones

Las conclusiones que logramos obtener del proyecto son las siguientes:

1) En el siguiente trabajo se hizo uso de un protocolo de comunicación de Maestro-Esclavo sincrónico, el mismo que nos permitió controlar varios periféricos pudiendo utilizar para todos ellos un mismo bus de datos. Ahorrándonos varios puertos y pines del microcontrolador para poder usarlos de manera mucho más efectiva en las aplicaciones en las que estemos trabajando.
2) La programación del conjunto de ejercicios se simplificó en gran medida debido a que los comandos o instrucciones necesarios para operar el protocolo son específicos y relativamente simples. Bastó entender el funcionamiento de los mismos y ponerlos en práctica para así desarrollar cada uno de los ejercicios.
3) Ya en la implementación del proyecto, las cuatro líneas de comunicación que se requieren en la comunicación sincrónica fueron en realidad lo más destacable en lo que al protocolo se refiere y al desarrollo de los ejercicios; ya que serían lo que cualquier ingeniero hubiera implementado si se encontrara en la necesidad de diseñar un protocolo rápido, eficaz y sencillo.
4) Con el desarrollo de los ejercicios y su implementación práctica pudimos darnos cuenta de las ventajas y desventajas mencionadas a lo largo del

trabajo que implica el uso de este protocolo para la comunicación entre los diferentes dispositivos. Como por ejemplo: el protocolo SPI permite una comunicación Full Duplex y una alta velocidad de transmisión con una implementación en hardware extremadamente simple, pero solo funciona en las distancias cortas. No hay control de flujo por hardware y no hay señal de asentamiento; es decir el maestro podría estar enviando información sin que estuviese conectado ningún esclavo. Podríamos concluir así entonces que todo dependerá de la aplicación que estemos desarrollando y el uso que le queramos dar en la aplicación.

Recomendaciones

Las recomendaciones que logramos obtener del proyecto son las siguientes:

1) Hay algunas cosas a tener en cuenta cuando se trata de poner en práctica una comunicación SPI. Una de estas cuestiones es el cableado. Es importante asegurarse de que la entrada serie en el dispositivo está conectada correctamente a la salida serie del microcontrolador con el que las comunicaciones se están produciendo.
2) Por otra parte, muchos dispositivos externos, tales como EEPROM tienen pines que puede desactivar el dispositivo de forma selectiva (para su uso en sistemas más grandes), es importante tomar nota de los pines que no están siendo utilizados así como de los pines que están en uso.
3) Para poder obtener una excelente comunicación SPI entre los microcontroladores utilizados es importante trabajar con una alimentación de 5V o más, ya que al trabajar con los 3V que generan las pilas no se establecía de manera correcta la comunicación y se enviaban datos erróneos.

ANEXOS

Guía para programar el AVR Butterfly

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

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

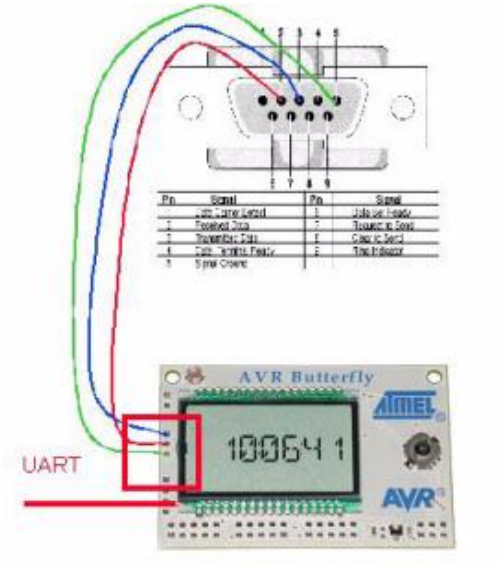


Figura 1.

2. En la PC, ejecutar AVR Studio 4.

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

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

5. En AVR Studio 4, en la barra de herramientas, abrir el menú Tools. En este menú se visualiza la función AVRProg, tal como se observa en la Figura 2 entonces,

seleccionar dicha función para que aparezca la ventana AVRprog que se observa en la Figura.

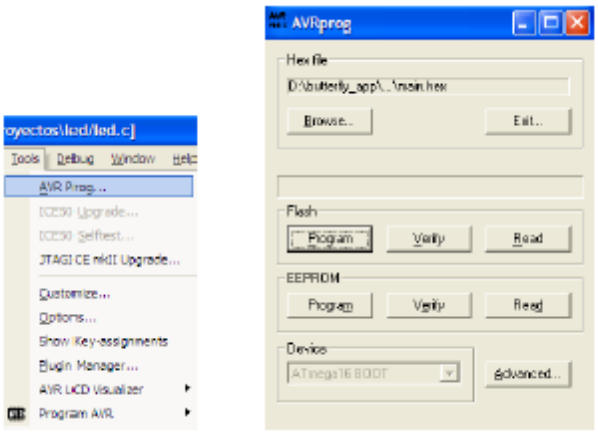


Figura 2.

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

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

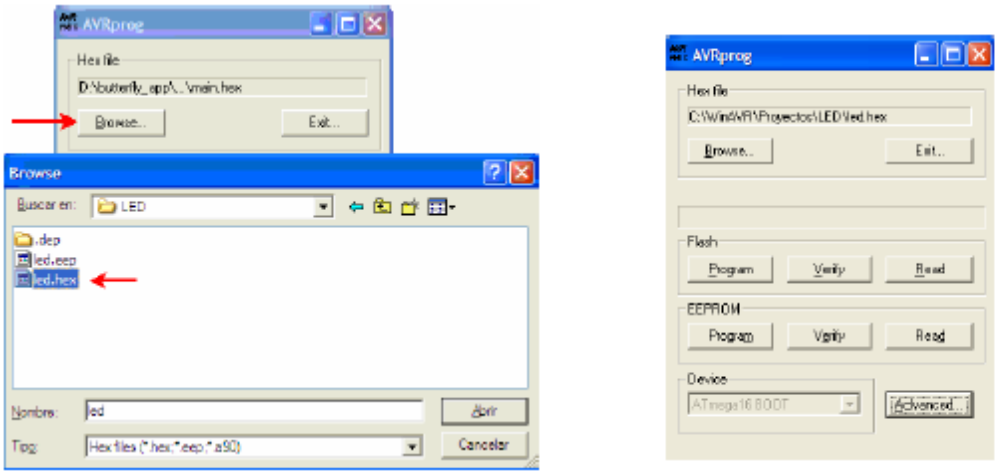


Figura 3.

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

Figura 4.

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

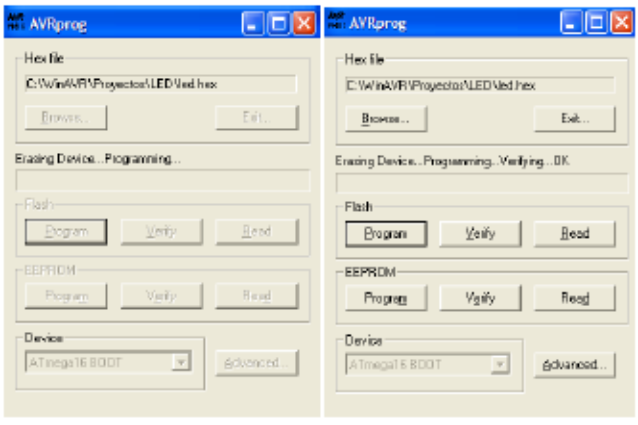


Figura 5.

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

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

Bibliografía

1. Barret S., Pack D., Atmel AVR Microcontroller Primer, Programming and Interfacing, Capítulo 2: Serial Peripheral Interface, pag 34-38, 2008.
   Fecha de Consulta: 10/Octubre/2011

2. López Chau, A., Microcontroladores AVR, Configuración total de periféricos, Capítulo SPI, primera edición, 2006.
   Fecha de Consulta: 29/Septiembre/2011

3. Pardue Joe, C Programming for Microcontrollers, Capítulo 2: Quick Start Guide, pag 17-27, 2005.
   Fecha de Consulta: 10/Octubre/2011

4. Topic 18c: Serial Peripheral Interface
   http://ww1.microchip.com/downloads/en/DeviceDoc/70243b.pdf
   Fecha de Consulta: 16/Septiembre/2011

5. Using SPI on an AVR
   http://www.rocketnumbernine.com/2009/04/26/using-spi-on-an-avr-1
   Fecha de Consulta: 17/Septiembre/2011

6. ATMEL, Hoja de Datos del microcontrolador ATMega169.

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

   Fecha de Consulta: 18/Septiembre/2011

7. KIT DE DESARROLLO AVR BUTTERFLY, DESARROLLO DE GUÍA DE PRÁCTICAS DE LABORATORIO Y TUTORIALES.

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

   Fecha de Consulta: 18/Septiembre/2011