ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

"ALMACENAMIENTO DE DATOS DE TEMPERATURA DE MOTOR BLDC PARA GRAFICACIÓN Y ANÁLISIS EN DISPLAYS DISPONIBLES EN TARJETA AVR BUTTERFLY Y EN TARJETA CONTROLADORA LPCXPRESSO Y PRESENTACIÓN DE MENSAJES DE OPERACIÓN"

TESINA DE SEMINARIO

Previa la obtención del Título de:

INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

Presentado por:

Pamela Puente Amador

Antonio Delgado Neira

GUAYAQUIL – ECUADOR

AÑO 2012

AGRADECIMIENTO

A Dios por guiar nuestro camino y mostrarnos esa luz de esperanza y dotarnos siempre de inteligencia y a todas aquellas personas que de una u otra forma, colaboraron o participaron en la realización de esta investigación

DEDICATORIA

A nuestros padres por ser el pilar fundamental en nuestras vidas, por habernos brindado la mejor educación posible y ese apoyo incondicional en todas las etapas de aprendizaje.

TRIBUNAL DE SUSTENTACIÓN

_____

Ing. Carlos Valdivieso A.

PROFESOR DEL SEMINARIO DE GRADUACIÓN

_____

Ing. Ronald Ponguillo

PROFESOR DELEGADO DE LA UNIDAD ACADÉMICA

DECLARACIÓN EXPRESA

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

(Reglamento de Graduación de la ESPOL)

Pamela Puente Amador **Antonio Delgado Neira**

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Nuestro proyecto persigue hacer el análisis de temperatura de los motores bldc (brushless), que desde este momento en adelante solo lo mencionaremos como los motores bldc para referirnos a ellos, este análisis comienza con la medición de temperatura usando un transductor activo para monitoreo en tiempo real, seguido del almacenamiento de los datos obtenidos en memoria externa usando el microcontrolador AVR Butterfly y por ultimo mostrar mensajes de operación con otro microcontrolador de 32bits LPCXPRESSO que estará conectado mediante comunicación serial con el AVR Butterfly.

Este proceso lo hemos dividido en cinco partes: Medición de temperatura del Motor BLDC, procesamiento digital de datos de temperatura, monitoreo en tiempo real (Graficación de datos), comunicación serial entre los 2 microcontroladores y mostrar mensajes de operación.

CAPÍTULO 1

DESCRIPCIÓN GENERAL DEL PROYECTO

1.1 ANTECEDENTES

El motor bldc o motor sin escobillas es ampliamente usado en una gran variedad de aplicaciones y entornos, debido entre otros motivos a su alto rendimiento, a la fiabilidad y ausencia de escobillas.

Actualmente, los motores BLDC se emplean en sectores industriales tales como: Automovilístico, Aeroespacial, equipos de automatización e instrumentación, aeromodelismo y automodelismo ya que como los motores BLDC tienen la característica de que no emplean escobillas en la conmutación para la transferencia de energía eliminan el gran problema que poseen los motores eléctricos convencionales con escobillas que producen rozamiento, disminuyen

el rendimiento, desprenden calor, son ruidosos y requieren una sustitución periódica de sus carbones por tanto un mayor mantenimiento.

En cuanto a la medición de temperatura de motores, es muy importante ya que la durabilidad del motor y su desempeño depende del esfuerzo o carga a la que está sometida y este a su vez se manifiesta mediante calor, nuestro proyecto está enfocado en controlar la temperatura del motor y mantenerlo en un rango de temperatura adecuado para su funcionamiento.

1.2 OBJETIVOS

OBJETIVO GENERAL

- Adquirir la destreza necesaria en las diversas etapas diseño e implementación de equipos electrónicos con microcontroladores avanzados.

OBJETIVOS ESPECÍFICOS

- Implementar un sensor de temperatura para un motor BLDC.
- Medición correcta de temperatura y digitalización de la misma para procesamiento y almacenamiento de dichos datos.

- Dominar el manejo correcto de los Microcontroladores LPCXPRESSO de 32 bits para propósitos generales.
- Manejar correctamente el microcontrolador AVR Butterfly y realizar la comunicación serial con microcontrolador LPCEXPRESSO para envió de datos.
- Lograr realizar un sistema efectivo y funcional que permita mostrar el desempeño y monitoreo de los motores BLDC en lo referente a la temperatura debido a la velocidad y a la carga a la que se someta.

1.3 IDENTIFICACIÓN DEL PROBLEMA

El presente trabajo, se trata sobre medición, monitoreo de temperatura de los motores bldc y evaluar el desempeño de dichos motores en tiempo real para posteriormente almacenar los datos medidos, analizarlos y tener registro del desempeño de estos motores.

El problema a solucionar consiste en medir con precisión y fidelidad la temperatura de los motores usando transductores activos para ese efecto y poder digitalizarlos mediante los módulos convertidores ADC del Microcontrolador AVR Butterfly de Atmel y esos datos digitalizados luego parametrizarlos en el rango de temperatura correspondiente y almacenarlos al mismo tiempo que se los muestra en una pantalla LCD incluida en el

Microcontrolador AVR Butterfly, estos datos se los enviaran mediante comunicación serial al Microcontrolador LPCXPRESSO 1769 para evaluar el rango de temperatura en que está trabajando el motor y mostrar mensajes de operación indicando el estado del motor.

1.4 LIMITACIONES

Unas de las limitaciones en este trabajo es tener todas las herramientas disponibles instaladas en una PC y tener que hacer uso de ellas para poder realizar diferentes pruebas y poder depurar hasta lograr nuestros objetivos.

Es importante también tener en cuenta las limitaciones en cuanto a la cantidad de equipos disponibles para la pruebas ya que al ser equipos importados si por algún motivo se descompone uno de estos volver a importarlos retrasaría el trabajo de investigación de este proyecto.

El LPCXPRESSO es relativamente un nuevo microcontrolador, esto hace que la investigación del uso y programación de este microcontrolador sea más exhaustiva debido a que no hay mucha información específica disponible en internet.

CAPÍTULO 2

FUNDAMENTO TEÓRICO

2.1 RESUMEN

En este capítulo haremos una breve descripción teórica para poder entender el funcionamiento de los elementos utilizados en este proyecto, como vamos a implementar un sensor de temperatura para un motor BLDC comenzaremos con la descripción de cada una de las herramientas que nos han ayudado a lograr nuestro objetivo.

Además de los microcontroladores utilizados que son las tarjetas AVR Butterfly y LPCXpresso también va a ser descrita con mayor detalle la comunicación que empleamos entre estos, se denomina comunicación serial UART o USART, esto nos va a permitir mandar datos a la LPCXpresso y mostrar mensajes de

operación, si sobrepasa la temperatura normal o si se encuentra en el rango adecuado de este modo se podrán realizar las acciones correspondientes a tiempo, estos datos no los proporcionara un transductor activo el cual también explicaremos como funciona a continuación, este dispositivo es el que se encontrara midiendo la temperatura constantemente, esto nos permitirá detectar cualquier cambio considerable de temperatura, mostrar y almacenar los datos en el AVR Butterfly.

2.2 TARJETA LPCXPRESSO 1769

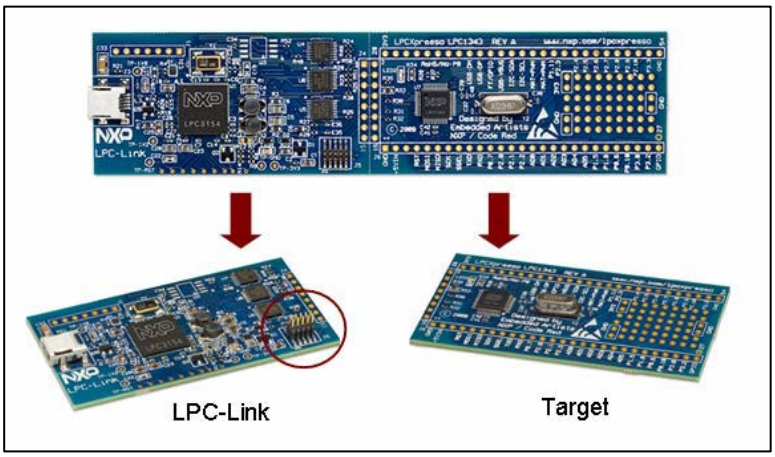


Figura 2.1 Tarjeta LPCXpresso 1769 **[2]**

El LPC1769 mostrado en la Figura 2.1 es un microcontrolador basado en la arquitectura ARM Cortex-M3 para aplicaciones embebidas que requieren un alto nivel de integración y baja disipación de potencia. El procesador ARM Cortex-M3 es un núcleo de nueva generación que ofrece mejoras en el sistema de

depuración, tales como modernas características de depuración y un mayor nivel de encapsulación de sus bloques.

Las versiones de alta velocidad (LPC1769 y LPC1759) operan a velocidades de hasta 120 MHz. Otras versiones pueden operar a 100 MHz. La ARMCortex-M3 de la CPU incorpora 3 etapas y utiliza una arquitectura de Harvard con instrucción local separada y buses de datos, así como un tercer autobús para los periféricos. La arquitectura ARMCortex-M3 de la CPU también incluye una unidad de procesamiento interno que soporta variantes.

El complemento periférico de la LPC17xx incluye hasta 512 KB de memoria flash,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 controlador DMA, 4 UARTs, 2 canales CAN, 2 controladores de SSP, interfaz SPI, 3 interfaces I2C, dos de entrada más de I2S 2-salidainterfaz, 8 canales de ADC de 12 bits, 10 bits DAC, el motor de control PWM, codificador de cuadratura, 4 temporizadores de uso general, de 6 de salida de propósito general PWM, ultra bajo de energía RTC con suministro de la batería por separado, y hasta 70 pines de propósito general I / O. **[1]**

2.2.1 CARACTERÍSTICAS

- ARM Cortex-M3, que funciona a frecuencias de hasta 100Hz. (LPC1768/67/66/65/64/63) o de hasta 120 MHz (LPC1769).
- Una unidad de protección de memoria (MPU) dando soporte a ocho regiones.
- ARM Cortex-M3 integrado en el vector de interrupciones (NVIC).
- Hasta 512 kB en un chip de memoria de programación flash.
- Permite alta velocidad de operación de 120 MHz con cero estados de espera.
- In-System Programming (ISP) y en programación de aplicaciones (IAP) a través de un chip gestor de arranque del software.
- El chip de SRAM incluye:
  - 32/16 KB de SRAM de la CPU con código local / bus de datos de alto rendimiento de la CPU.
  - 2 / 1 16 KB de SRAM con bloques de las vías de acceso independientes para un mayor rendimiento. Estos bloques SRAM puede ser utilizado para Ethernet, USB y memoria de DMA, así como para la instrucción general de CPU de propósito y almacenamiento de datos.

- Ocho canales de uso general controlador DMA (GPDMA) en la matriz multicapa AHB que se puede utilizar con SSP, I2S bus, UART, de analógico a digital y Digital a analógico convertidor de los periféricos, las señales del temporizador de partido, y transferencias de memoria.
- Matriz multicapa de interconexión AHB ofrece un autobús para cada maestro de AHB.
- AHB maestros incluyen DMA, MAC Ethernet y la interfaz USB. Esta interconexión permite la comunicación sin retrasos de arbitraje.
- Split bus APB permite un alto rendimiento con pocos puestos entre la CPU y el DMA.
- Interfaces seriales:
- Ethernet MAC con RMII interfaz y controlador DMA dedicado.
- USB 2.0 de alta velocidad del dispositivo / Host / OTG controlador DMA dedicado y PHY en el chip para el dispositivo, el host, y las funciones de OTG.
- Cuatro UART con la generación de fracción velocidad de transmisión, FIFO interna, y con DMA.
- Una UART tiene el control del módem E / S y RS-485/EIA 485 apoyo, una UART cuenta con el apoyo IrDA.

- CAN 2.0B controlador con dos canales.
- SPI del controlador sincrónico, comunicación serial, full dúplex y la longitud de datos programable.
- Dos controladores SSP con capacidades de FIFO y multi-protocolo. Las interfaces de SSP puede utilizarse con el controlador GPDMA.
- Tres interfaces mejoradas bus I2C, uno con una salida de colector abierto con soporte a todo.
- Especificación de bus I2C y el modo rápido, con velocidades de datos de 1Mbit/s, dos con los pines estándar del puerto. Las mejoras incluyen el reconocimiento de dirección múltiple.
- I2S (Inter-IC de sonido) de la interfaz de entrada de audio digital o la salida, con una tasa fraccional controlada. La interfaz de bus I2S se puede utilizar con la GPDMA. La interfaz de bus I2S es compatible con los datos de 3 hilos y 4 hilos de transmisión y recepción, así como reloj maestro de entrada / salida.

➢ Otros periféricos:

- 70 (100 paquete de pines) de uso general de I /O (GPIO) con pines configurable pull-up/down resistencias. Todos los GPIO soportan un nuevo

y modo configurable de colector abierto. El bloque de GPIO se accede a través del bus AHB multicapa para un acceso rápido y situado en la memoria de tal manera que soporta Cortex-M3 y el controlador de Propósito General DMA.

- 12 bits de conversión analógico a digital (ADC) con el aporte de multiplexación entre los ocho pines, las tasas de conversión de hasta 200 kHz, y los registros de resultados múltiples. El ADC de 12 bits puede ser utilizado con el controlador GPDMA.
- 10-bit digital a analógico (DAC) con temporizador de conversión específico y soporte DMA.
- Cuatro temporizadores para fines generales y contadores, con un total de ocho entradas de captura y diez comparadores de resultados. Cada bloque tiene un temporizador de entrada externa. contadores de tiempo específico pueden ser seleccionados para generar peticiones de DMA.
- Un motor de control PWM con soporte para control de motor de tres fases.
- Interfaz de codificador de cuadratura que puede controlar un codificador de cuadratura externo.
- Un estándar PWM / temporizador de bloque con entrada externa.

- WatchDog Timer (WDT). El WDT puede ser ajustado desde el oscilador RC interno, el oscilador RTC, o el reloj APB.
- Temporizador ARM Cortex-M3 del sistema, incluyendo una opción de entrada de reloj externo.
- Cada periférico tiene su propio divisor de reloj de ahorro de energía.
- Integrado PMU (Unidad de Administración de Energía) se ajusta automáticamente a los reguladores internos reducir al mínimo el consumo de energía durante el sueño, sueño profundo, al apagar, y los modos de sueño profundo.

➢ Única fuente de alimentación de 3,3 V (2,4 V a 3,6 V).**[2]**

2.3 AVR BUTTERFLY KIT

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

El AVR Butterfly utiliza el microcontrolador AVR ATmega169, que combina la Tecnología Flash con el más avanzado y versátil microcontrolador de 8 bits disponible. En la Figura 2.2 se puede apreciar el Kit AVR Butterfly. **[3]**

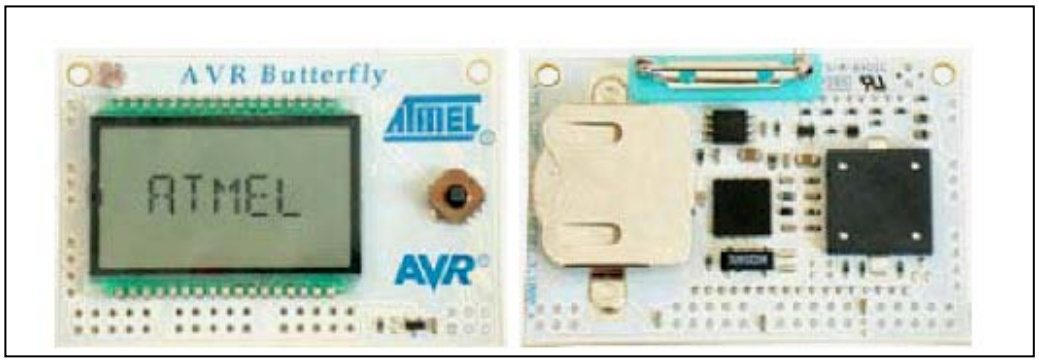


Figura 2.2 Kit AVR Butterfly **[3]**

2.3.1 CARACTERÍSTICAS

El Kit AVR Butterfly expone las siguientes características principales:

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

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

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

2.3.2 HARDWARE DISPONIBLE

Los siguientes recursos están disponibles en el Kit AVR Butterfly:

- Microcontrolador ATmega169V (en encapsulado tipo MLF).
- Pantalla tipo vidrio LCD de 120 segmentos
- 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 0-5 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 2.3 y 2.4 se observa el Hardware disponible en el AVR Butterfly.

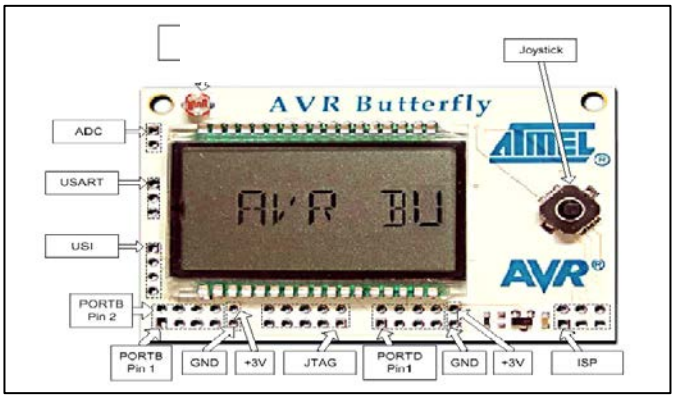


Figura 2.3 Parte Frontal **[4]**

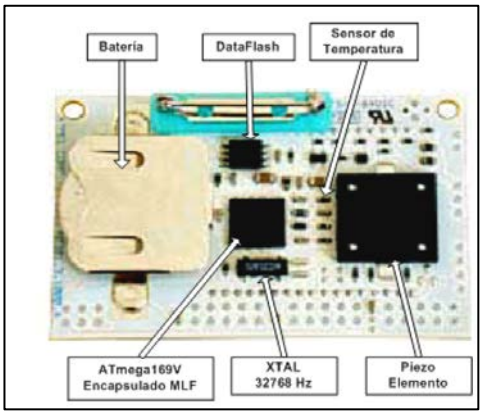


Figura 2.4 Parte Posterior **[4]**

2.4 HERRAMIENTAS DE PROGRAMACIÓN

2.4.1AVR STUDIO 4

AVR Studio es un Entorno de Desarrollo Integrado (IDE) para escribir y depurar aplicaciones AVR en el entorno de Windows 9x/Me/NT/2000/XP/7.

AVR Studio 4 soporta varias de las fases por las cuales se atraviesa al crear un nuevo producto basado en un microcontrolador AVR. Las fases típicas son:

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

AVR Studio apoya al diseñador en el diseño, desarrollo, depuración y parte de la comprobación del proceso. AVR Studio es actualizado continuamente y está disponible para descargarlo desde www.atmel.com.

A continuación se lista brevemente los requerimientos mínimos del sistema, necesarios para poder utilizar el AVR Studio 4 en una PC:

- Windows 7
- Windows 2000/XP (o Windows NT 4.0 con Internet Explorer 5.0 o posterior).
- Windows 95/98/Me (con Internet Explorer 5.0 o posterior).
- Hardware Recomendado:
  - Procesador Intel Pentium de 200MHz o equivalente.
  - Resolución de Pantalla de 1024x768 (Resolución mínima de 800x600).
  - Memoria RAM de 64Mb.
  - Disco Duro con 50 Mb de espacio disponible.

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

AVR Studio 4 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. **[5]**

AVR Studio 4 consiste de muchas ventanas y sub-módulos. Cada ventana apoya a las partes del trabajo que se intenta emprender. En la Figura 2.5 se puede apreciar las ventanas principales del IDE.

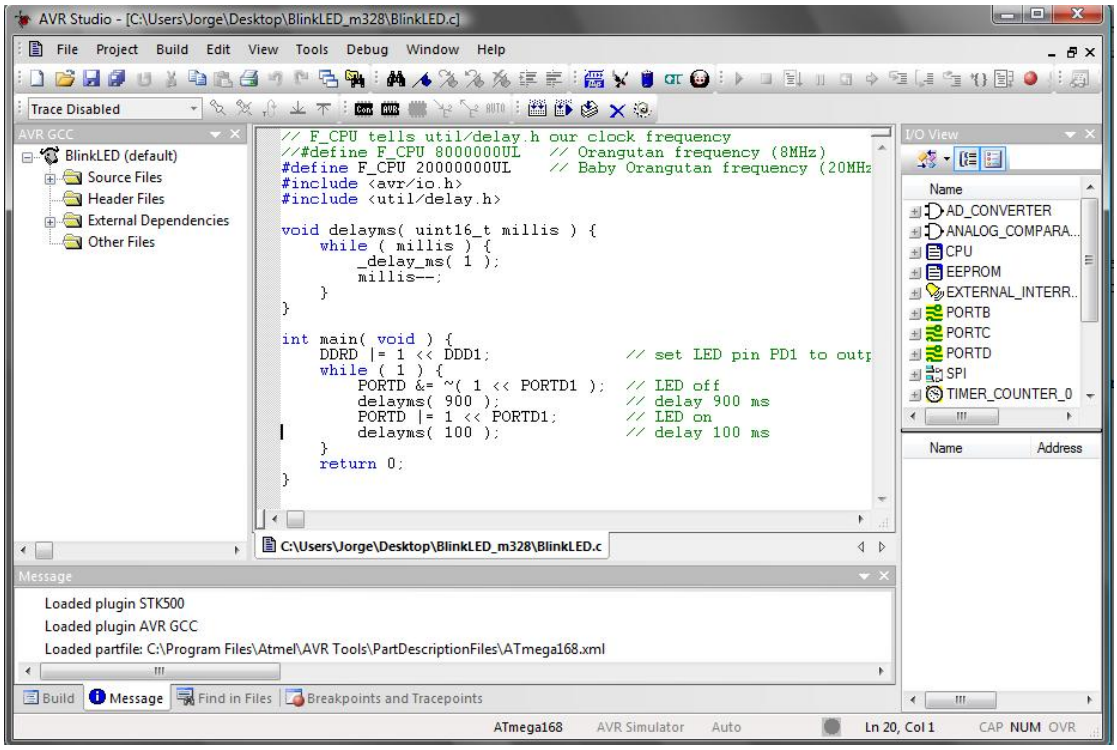


Figura 2.5 Ambiente programación AVR Studio

WIN AVR

La distribución Win Avr, que es una recopilación de programas de software libre diseñados para facilitar las tareas de programación y desarrollo de los microcontroladores Avr. Dicha distribución Win Avr incorpora además del compilador gcc de consola, un editor de texto especialmente diseñado para ayudar al programador y hacer el código más legible mediante su resaltado con colores.

El programador vía puerto serie mprog.exe, que permite transferir el programa compilado, que se encuentra en un archivo llamado main.hex, a la memoria flash del microcontrolador utilizando únicamente un cable de tres líneas.

En pocas palabras Win AVR es un conjunto de herramientas de desarrollo para microcontroladores RISC AVR de Atmel, basado en software de código abierto y compilado para funcionar en la plataforma Microsoft Windows. Win AVR incluye las siguientes herramientas:

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

  −avr-as, el ensamblador.
- avrdude, la interfaz para programación.
- avarice, la interfaz para JTAG ICE.
- avr-gdb, el depurador.

- Programmers Notepad, el editor.
- MFile, generador de archivo makefile.

La distribución Win Avr surge como una iniciativa para impulsar el desarrollo de software libre en el campo del desarrollo de los microcontroladores Avr de Atmel, a la vez de facilitar a sus potenciales usuarios el conocer la existencia de tales herramientas, y no sólo las más famosas, y también facilitar su actualización. Es por ello que dicho paquete de software incorpora no sólo dichas herramientas, sino además unos ficheros de ayuda y de descripción de cada una de las utilidades. Por supuesto, el soporte técnico es mayor según la herramienta tenga más importancia.

2.4.2 LPCXpresso

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

Está compuesto por:

- LPCXpresso IDE y "development tools"
- IDE basado en Eclipse
- compiler y linker GNU
- GDB debugger
- LPCXpresso target board (stick)
- BaseBoard o hardware adicional(opcional)

Eclipse utiliza algunos conceptos que no siempre son comunes a otros entornos de desarrollo por lo que vamos a ver algunos de ellos.

- **Workspace:** Es el contenedor de nuestros proyectos. Estos proyectos pueden ser aplicaciones y/o bibliotecas. También almacena todas las configuraciones del entorno por lo que se puede mover muy fácilmente de computadora en computadora.
- **Proyecto:** Este puede ser de dos tipos. Biblioteca estática o una aplicación ejecutable. Contiene archivos de código fuente (.c), encabezados (.h) y cualquier otro archivo que se desee.

En general utilizaremos el workspace para intercambiar proyectos (en el sentido convencional de la palabra) ya que el mismo incluirá todas las bibliotecas necesarias.

Los proyectos pueden ser de dos tipos:

- **Aplicaciones:** Se compilan y se pueden descargar directamente al target.
- **Bibliotecas estáticas:** Se pueden compilar, pero para usarlas, un proyecto de tipo aplicación debe hacer llamadas a las funciones que

este contiene. Es decir, no puede tener un main(). Este tipo de proyectos no se puede descargar por si solo al microcontrolador.

A continuación en la Figura 2.6 se muestra el IDE que utilizaremos para la programación de nuestra LPCXpresso.

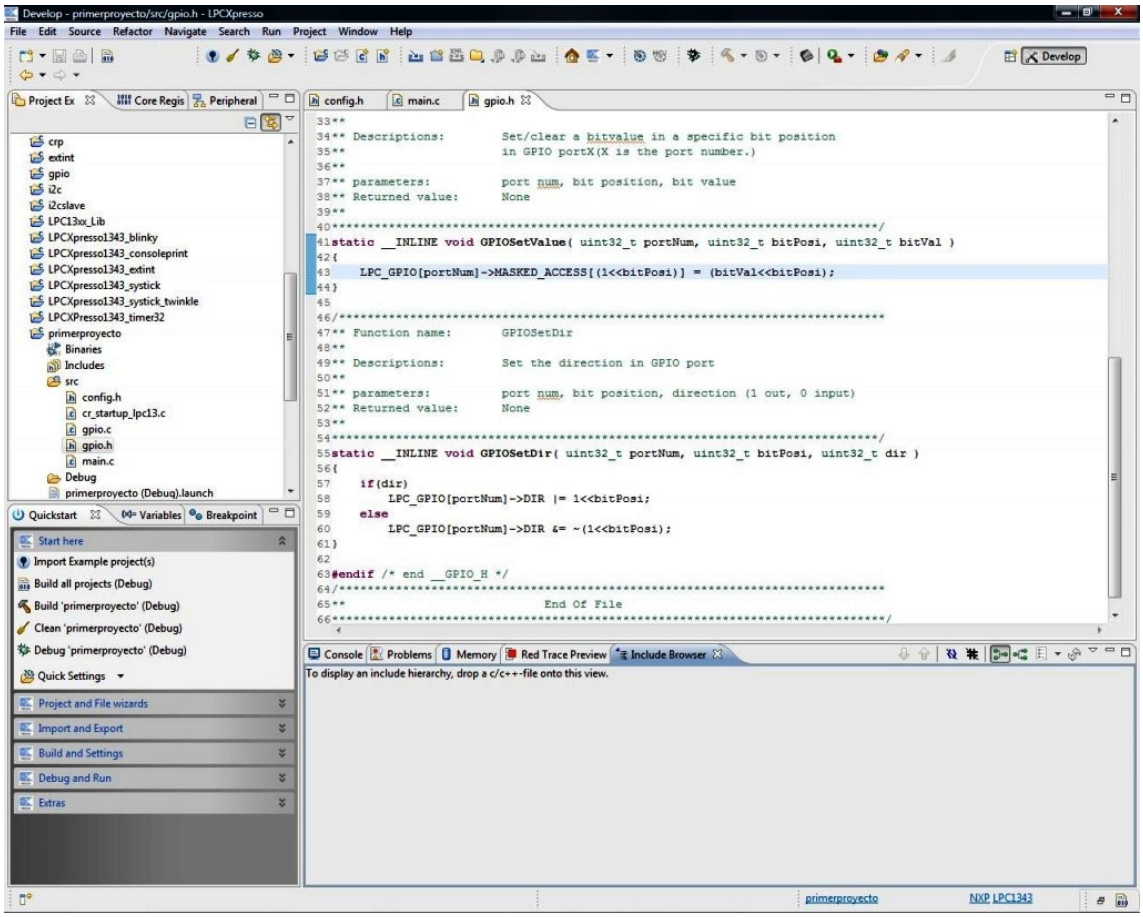


Figura 2.6 Ambiente programación LPCXpresso

2.5 COMUNICACIÓN UART O USART

La comunicación serial es un protocolo muy común (no hay que confundirlo con el Bus Serial de Comunicación, o USB) para comunicación entre dispositivos que se incluye de manera estándar en prácticamente cualquier computadora. La mayoría de las computadoras incluyen dos puertos seriales RS-232. La comunicación serial es también un protocolo común utilizado por varios dispositivos para instrumentación; existen varios dispositivos compatibles con GPIB que incluyen un puerto RS-232. Además, la comunicación serial puede ser utilizada para adquisición de datos si se usa en conjunto con un dispositivo remoto de muestreo.

El concepto de comunicación serial es sencillo. El puerto serial envía y recibe bytes de información un bit a la vez. Aun y cuando esto es más lento que la comunicación en paralelo, que permite la transmisión de un byte completo por vez, este método de comunicación es más sencillo y puede alcanzar mayores distancias. Por ejemplo, la especificación *IEEE 488*para la comunicación en paralelo determina que el largo del cable para el equipo no puede ser mayor a 20 metros, con no más de 2 metros entre cualesquier dos dispositivos; por el otro lado, utilizando comunicación serial el largo del cable puede llegar a los 1200 metros.

Típicamente, la comunicación serial se utiliza para transmitir datos en formato ASCII. Para realizar la comunicación se utilizan 3 líneas de transmisión: (1) Tierra (o referencia), (2) Transmitir, (3) Recibir. Debido a que la transmisión es asincrónica, es posible enviar datos por un línea mientras se reciben datos por otra. Existen otras líneas disponibles para realizar *handshaking*, o intercambio de pulsos de sincronización, pero no son requeridas. Las características más importantes de la comunicación serial son la velocidad de transmisión, los bits de datos, los bits de parada, y la paridad. Para que dos puertos se puedan comunicar, es necesario que las características sean iguales. **[10]**

a. **Velocidad de transmisión *(baudrate)*:** Indica el número de bits por segundo que se transfieren, y se mide en baudios (*bauds*). Por ejemplo, 300 baudios representa 300 bits por segundo. Cuando se hace referencia a los ciclos de reloj se está hablando de la velocidad de transmisión. Por ejemplo, si el protocolo hace una llamada a 4800 ciclos de reloj, entonces el reloj está corriendo a 4800 Hz, lo que significa que el puerto serial está muestreando las líneas de transmisión a 4800 Hz. Las velocidades de transmisión más comunes para las líneas telefónicas son de 14400, 28800, y 33600. Es posible tener velocidades más altas, pero se reduciría la distancia máxima posible entre los dispositivos. Las altas velocidades se

utilizan cuando los dispositivos se encuentran uno junto al otro, como es el caso de dispositivos GPIB.

b. **Bits de datos:** Se refiere a la cantidad de bits en la transmisión. Cuando la computadora envía un paquete de información, el tamaño de ese paquete no necesariamente será de 8 bits. Las cantidades más comunes de bits por paquete son 5, 7 y 8 bits. El número de bits que se envía depende en el tipo de información que se transfiere. Por ejemplo, el ASCII estándar tiene un rango de 0 a 127, es decir, utiliza 7 bits; para ASCII extendido es de 0 a 255, lo que utiliza 8 bits. Si el tipo de datos que se está transfiriendo es texto simple (ASCII estándar), entonces es suficiente con utilizar 7 bits por paquete para la comunicación. Un paquete se refiere a una transferencia de byte, incluyendo los bits de inicio/parada, bits de datos, y paridad. Debido a que el número actual de bits depende en el protocolo que se seleccione, el término paquete se usar para referirse a todos los casos.

c. **Bits de parada:** Usado para indicar el fin de la comunicación de un solo paquete. Los valores típicos son 1, 1.5 o 2 bits. Debido a la manera como se transfiere la información a través de las líneas de comunicación y que

cada dispositivo tiene su propio reloj, es posible que los dos dispositivos no estén sincronizados. Por lo tanto, los bits de parada no sólo indican el fin de la transmisión sino además dan un margen de tolerancia para esa diferencia de los relojes. Mientras más bits de parada se usen, mayor será la tolerancia a la sincronía de los relojes, sin embargo la transmisión será más lenta.

d. **Paridad:** Es una forma sencilla de verificar si hay errores en la transmisión serial. Existen cuatro tipos de paridad: par, impar, marcada y espaciada. La opción de no usar paridad alguna también está disponible. Para paridad par e impar, el puerto serial fijará el bit de paridad (el último bit después de los bits de datos) a un valor para asegurarse que la transmisión tenga un número par o impar de bits en estado alto lógico. Por ejemplo, si la información a transmitir es 011 y la paridad es par, el bit de paridad sería 0 para mantener el número de bits en estado alto lógico como par. Si la paridad seleccionada fuera impar, entonces el bit de paridad sería 1, para tener 3 bits en estado alto lógico. La paridad marcada y espaciada en realidad no verifican el estado de los bits de datos; simplemente fija el bit de paridad en estado lógico alto para la marcada, y en estado lógico bajo para la espaciada. Esto permite al dispositivo

receptor conocer de antemano el estado de un bit, lo que serviría para determinar si hay ruido que esté afectando de manera negativa la transmisión de los datos, o si los relojes de los dispositivos no están sincronizados.

2.5.1 PROGRAMACIÓN AVR BUTTERFLY

El AVR Butterfly tiene incluido un convertidor de nivel para la interfaz RS-232. Esto significa que no se necesita de hardware especial para reprogramar al AVR Butterfly utilizando la característica self-programming del ATmega169. A continuación se explica brevemente la distribución de los pines y como se debe realizar el cableado para la comunicación serial entre el AVR Butterfly y la PC. [8]

2.5.2 DISTRIBUCIÓN DE PINES ENTRE EL AVR Y LA PC

La comunicación con la PC requiere de tres líneas: TXD, RXD y GND. TXD es la línea para transmitir datos desde la PC hacia el AVR Butterfly, RXD es la línea para recepción de datos enviados desde el AVR Butterfly hacia la PC y GND es

la tierra común. En la Figura2.7 se observa la distribución de los pines para la comunicación serial, a la izquierda los pines del AVR Butterfly y a la derecha los pines del conector DB9 de la PC.

AVR Butterfly UART	COM2
Pin 1 (RXD)	Pin 3
Pin 2 (TXD)	Pin 2
Pin 3 (GND)	Pin 5

Figura 2.7 Distribución de pines, AVR Butterfly Vs. PC

En la Figura 2.8 se observa cómo se debe hacer el cableado para la comunicación, a través de la interfaz serial RS-232, entre el AVR Butterfly y la PC. A la izquierda se aprecia un conector DB9 hembra soldado a los cables que se conectan a la interfaz USART del AVR Butterfly (derecha).

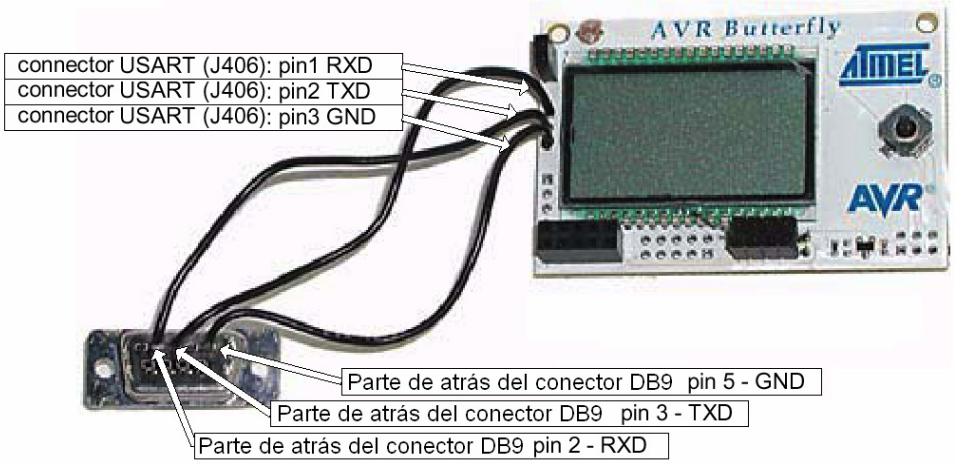


Figura 2.8 Conexiones para interfaz USART del AVR Butterfly **[4]**

2.5.3 COMUNICACIÓN UART EN TARJETA LPCXPRESSO 1769

Sus principales características son:

- Los tamaños de datos son de 5, 6, 7, y 8 bits.
- Generación y comprobación de la paridad.
- Uno o dos bits de parada.
- 16 bytes de recepción y transmisión FIFO.
- Generador de velocidad de transmisión, que incluye un divisor de tasa fraccional con una gran versatilidad.
- Compatible con DMA para transmitir y recibir.
- Capacidad de Auto-Baud.
- Multiprocesador con modos de direccionamiento.
- ModoIrDA para apoyar la comunicación por infrarrojos
- Soporte para el software de control de flujo.

2.5.4 CONEXIÓN PINES LPCXpresso 1769

Los pines a utilizar se muestran en la figura 2.9 son los que utilizaremos para el envío y recepción de datos en la tarjeta LPCXpresso.

Pin	Type	Description
RXD1	Input	**Serial Input.** Serial receive data.
TXD1	Output	**Serial Output.** Serial transmit data.

Figura 2.9Pines para comunicación UART

La ubicación de estos pines se encuentra mostrada en la figura 2.10.

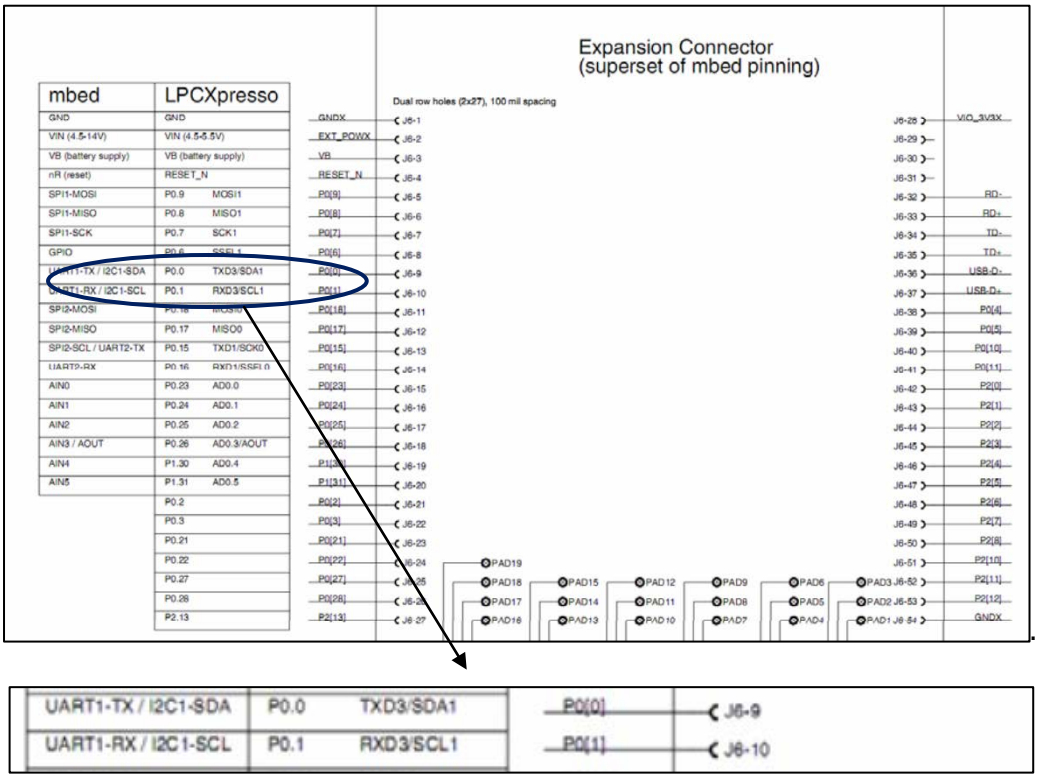


Figura 2.10 Pines LPCXpresso 1769

2.6 TRANSDUCTORES

2.6.1 DEFINICIÓN

El término transductor ha sido aplicado a dispositivos, o combinaciones de dispositivos, que convierten señales, o energía, de una forma física a otra forma. Más específicamente, en sistemas de medición, un transductor se define como un dispositivo que provee una salida usable, en respuesta a una medida especificada.

La medida es "una cantidad física, propiedad o condición, la cual es medida" y la salida es una "cantidad eléctrica, producida por un transductor, que es función de la medida". **[6]**

2.6.2 ELEMENTOS

Si bien los transductores actuales suelen estar integrados en una sola pieza, se pueden distinguir, en general, tres etapas en la generación de la salida eléctrica en respuesta a la medida física.

1. Sensor: es un elemento que responde directamente a la medida.
2. *Transductor* propiamente dicho: es el elemento en el que se transduce la señal física en una salida eléctrica.
3. *Circuito de acondicionamiento y procesamiento de la señal*: es un circuito, eléctrico o electrónico, que le da formato a la señal entregada

por el transductor. Su principal función es linealizarla salida y estandarizarla dentro de los límites de la aplicación.

4. El circuito acondicionador puede estar colocado dentro de la empaquetadura del transductor, o totalmente separado. Si el transductor consiste en varios módulos, las interconexiones provistas por el usuario en parte del sistema de medida y el correcto cableado, aislado y puesta a tierra, son esenciales para conseguir las condiciones de trabajo especificadas.

En general, se suele tratar al sensor y al transductor como un mismo componente, pero el circuito de acondicionamiento de la señal presenta algunas particularidades realmente importantes, por lo que se abarcará ahora este punto.

Circuito de acondicionamiento y procesamiento de la señal

Las señales que tienen que ser traducidas del mundo físico al mundo eléctrico, pueden ser de las formas más extrañas, consecuencia de que la señal eléctrica obtenida es función de la variación de un proceso físico, químico, atmosférico, etc, que se está midiendo.

Se hace entonces necesario imponerle a la señal eléctrica, antes de insertarla en el circuito de medida (ó en un SAD), una serie de condiciones que hagan favorable su manejo dentro de un circuito eléctrico. El circuito acondicionador de

la señal puede ser eléctrico o electrónico, y provee una variedad de funciones, como por ejemplo:

- Generación de la excitación o voltaje y frecuencia, de *referencia*.
- Generación de la señal de salida, típicamente por un circuito puente o un circuito potenciométrico.
- Acondicionamiento de la señal, esto es, amplificación de las salidas de bajo nivel y adaptación de los niveles de salida de tensión (o corriente) a un *rango standard*.
- Supresión de ruido, filtrado y aislación respecto a tierra.
- Conversión de señales, como AC/DC o A/D (cuando incluye el conversor A/D).
- Procesamiento de señales, como *linealización* de salidas intrínsecamente no lineales.

2.6.3 SEÑALES DE SALIDA

El nivel y rango de la señal de salida debe estar comprendido dentro de límites muy precisos para asegurar la compatibilidad con el resto del sistema. Estos rangos se determinan, generalmente, por el tipo de fenómeno que se está midiendo, y por el tipo de transductor que se utiliza. Las salidas utilizadas son:

De corriente: 0...5, 0...10, 0...20 y 4...20 mA

De tensión: 0...+5, 0...+10, -5...+5 y -10...+10 V

La salida de 4 .. 20 mA es particularmente usada, considerándose como la salida universal. Su importancia es tal que a aquellos transductores que tengan este rango de salida se los distingue como transmisores.

2.6.4 DETECCIÓN DE VARIACIÓN DE SEÑALES PEQUEÑAS

Como las variaciones que se deben medir suelen ser muy pequeñas, para detectarlas eléctricamente es usual utilizar conexiones en forma de potenciómetros o de puente de Wheatstone. Estos circuitos pueden ser de deflexión o nulos:

- *De deflexión*: la salida del puente o del potenciómetro pasa directamente al circuito acondicionador.
- *Nulos*: la salida del puente o potenciómetro es empleada para ajustar otro elemento del circuito de manera que anule al puente, y recién luego pasa al circuito acondicionador.

2.6.5 CARACTERÍSTICAS DE FUNCIONAMIENTO

Cada tipo de transductor tiene una relación *ideal* medida-salida, descripta por una ecuación teórica o por una representación numérica o gráfica. Esta característica ideal de transferencia puede ser en muchos casos lineal, en cuyo caso la pendiente de la recta es la relación de transferencia o *función*

transferencia de ese transductor. Si bien resulta prácticamente imposible construir transductores cuya relación medida-salida sea perfectamente lineal, en la práctica se manejan transductores cuya alinealidad es menor al 0,25%, pudiéndose considerar en ese caso la respuesta del transductor como lineal.

En el caso de una característica no lineal, la razón de transferencia es usada algunas veces para describir el transductor en un pequeño rango de entradas. Es decir, se limita a usar el transductor solo en aquellas zonas de su rango de funcionamiento donde la respuesta sea lineal. Cuando la alinealidad del transductor hace imposible su utilización en tales condiciones, entonces se debe *linealizar* dicha señal. La alinealidad de un transductor puede provenir tanto del elemento sensor, como también de la configuración utilizada para excitarlo o extraer su señal. La *linealización* de las señales se puede hacer, básicamente, de dos maneras:

- Uso de procesamiento en computadora (Linealización por Software).
- Uso de redes o circuitos (Linealización por Hardware).

2.6.6 LINEALIZACIÓN POR SOFTWARE

La señal recogida por el sensor es digitalizada y leída por una PC, o por un instrumento portátil (que contiene un microprocesador o un microcontrolador). Para corregir la alinealidad por software es preciso que la función transferencia (FT) esté perfectamente definida. Cuando esta FT está normalizada, y se

presenta en forma de tablas, la PC puede consultar las tablas (almacenadas en su memoria) y así realizar la linealización.

Así, la operación ejecutada es una *aproximación* de los valores medidos a la curva teórica de la FT. La linealización por software tiene algunas limitaciones importantes a saber:

- La FT debe definirse con un orden de magnitud superior a la precisión deseada.
- Para digitalizar es necesario amplificar la señal del sensor hasta, al menos, 100 mV a fondo de escala.
- Es necesaria una gran cantidad de circuitos de montaje complejo, sólo justificado si los comparten un cierto número de canales o si se necesita una precisión muy rigurosa.

2.6.7 LINEALIZACIÓN POR HARDWARE

Se utiliza cuando los sensores presentan una dispersión en sus características que harían necesario reprogramar la memoria de la PC que contiene la tabla de conversión entre la característica lineal y la real o entrar nuevos parámetros para el algoritmo linealizador.

Se trata de linealizar la respuesta del sensor con una *red pasiva*, al menos en un rango limitado. Existen diversos métodos, pero son complicados y engorrosos.

También se puede realizar algún tipo de procesamiento analógico a la señal de salida del transductor. Existen circuitos integrados (utilizando AO) que cumplen con una variedad de funciones simples tales como logaritmo, multiplicación o cociente y es posible combinarlos de manera de obtener funciones más complejas. También se puede utilizar una aproximación lineal por tramos con tantas secciones (cada una con su AO y red de resistores de precisión) como precisión deseemos. Pero los circuitos de complican rápidamente y además son muy sensibles a la calidad de los componentes.

2.6.8 INDICACIONES PARA SELECCIONAR Y EMPLEAR LOS TRANSDUCTORES

Cuando hay que elegir un transductor en especial, se deben considerar los siguientes puntos para determinar su capacidad para una medición en particular:

- ***Rango*****:** el rango del transductor debe ser lo suficientemente grande tal que abarque todas las magnitudes esperadas de la cantidad a ser medida.
- ***Sensibilidad*****:** para obtener datos significativos, el transductor debe producir una señal de salida suficiente por unidad de entrada de medida.

- ***Efectos de carga*:** como los transductores siempre consumirán algo de energía del efecto físico que se está probando, debe determinarse si se puede despreciar esta absorción o si se pueden aplicar factores de corrección para compensar las lecturas por pérdidas.
- ***Respuesta a la frecuencia*:** el transductor debe ser capaz de responder a la velocidad máxima de cambio en el efecto que se está observando.
- ***Formato de salida eléctrica*:** la forma eléctrica de la salida del transductor debe tener un valor que lo haga compatible con el resto del sistema de medición.
- ***Impedancia de salida*:** la impedancia de salida del transductor debe tener un valor que lo haga compatible con las siguientes etapas eléctricas del sistema.
- ***Requerimiento de potencia*:** los transductores pasivos necesitan de excitación externa. Entonces, si se deben emplear transductores pasivos, es necesario asegurar que haya disponibles fuentes de poder eléctricas adecuadas para operarlos.
- ***Medio físico*:** el transductor seleccionado debe poder resistir las condiciones ambientales a las que estará sujeto mientras se efectúe la prueba. Parámetros tales como temperatura, humedad y substancias químicas corrosivas podrían dañar algunos transductores y a otros no.

- ***Errores*:** los errores inherentes a la operación del mismo transductor o aquellos errores originados por las condiciones del ambiente en la medición, deben ser lo suficientemente pequeños o controlables para que permitan tomar datos significativos. Una vez que el transductor está elegido e instalado, se deben seguir las siguientes recomendaciones para aumentar la exactitud de las mediciones:
- ***Calibración del transductor*:** se debe calibrar la salida del transductor tomando algún estándar conocido al emplearlo en las condiciones reales de la prueba. Esta calibración se debe llevar a cabo con regularidad a medida que se haga la medición.
- Se deben monitorear en forma continua los cambios en las condiciones ambientales del transductor. Si se sigue este procedimiento, los datos medidos podrán corregirse posteriormente para tomar en cuenta cualquier cambio en las condiciones ambientales.
- Controlando artificialmente el ambiente de la medición, se pueden reducir errores posibles del transductor. Un control artificial del ambiente incluye, por ejemplo, el encerramiento del transductor en una caja de temperatura controlada y el aislamiento del dispositivo con respecto a golpes y vibraciones externas.

2.6.9 ERRORES

En la práctica, la respuesta de un transductor real se aparta de la ideal y en consecuencia el transductor real indicará un valor de medida que contendrá una componente de error. A este error contribuyen:

- *Errores estáticos*: es la diferencia entre la curva función transferencia teórica y la curva de calibración determinada en condiciones estacionarias.
- *Errores dinámicos*: son inherentes al funcionamiento del circuito electrónico (respuesta en frecuencia, respuesta temporal, función transferencia, etc.).

2.7TRANSDUCTOR LM35

En nuestro proyecto usaremos el transductor LM35 que nos ayudara a medir la temperatura del motor y por hacer el análisis en base de los datos adquiridos, ya que este transductor es del tipo activo, es decir necesita polarización y tiene una salida de voltaje lineal con respecto a la temperatura y escalado en grados Celsius, lo que nos da mas facilidad en la medición y simplicidad en la adquisición de estos datos medidos.

El LM35 es un sensor de temperatura con una precisión calibrada de 1ºC. Puede medir temperaturas en el rango que abarca desde -55º a + 150ºC. La salida es muy lineal y cada grado centígrado equivale a 10 mV en la salida.

2.7.1 CARACTERÍSTICAS

Sus características más relevantes son:

- Precisión de ~1,5ºC (peor caso), 0.5ºC garantizados a 25ºC.
- No linealidad de ~0,5ºC (peor caso).
- Baja corriente de alimentación (60uA).
- Amplio rango de funcionamiento (desde -55º a + 150ºC).
- Bajo costo.
- Baja impedancia de salida.

Su tensión de salida es proporcional ala temperatura, en la escala Celsius. No necesita calibración externa y es de bajo costo. Funciona en el rango de alimentación comprendido entre 4 y 30 voltios.

Como ventaja adicional, el LM35 no requiere de circuitos adicionales para su calibración externa cuando se desea obtener una precisión del orden de ±0.25 ºC a temperatura ambiente, y ±0.75 ºC en un rango de temperatura desde 55 a 150 ºC.

La baja impedancia de salida, su salida lineal y su precisa calibración inherente hace posible una fácil instalación en un circuito de control.

Debido a su baja corriente de alimentación (60uA), se produce un efecto de autocalentamiento reducido, menos de 0.1 ºC en situación de aire estacionario.

2.7.2 CONEXIÓN PINES LM35

La Figura 2.11nos muestra la configuración para los pines del LM35

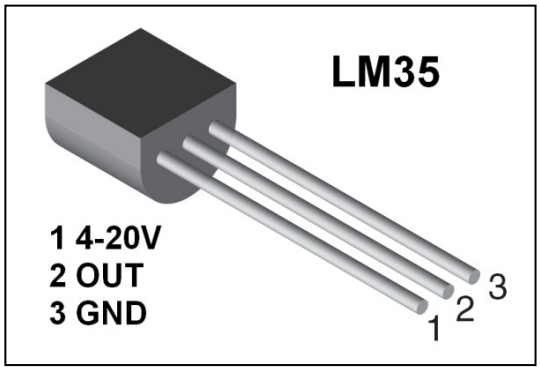


Figura 2.11Pines LM35

2.8 MOTORES BLDC

El motor brushless es ampliamente usado en una gran variedad de aplicaciones y entornos, debido, entre otros motivos a su alta densidad de potencia, la fiabilidad y ausencia de escobillas. La alta densidad de potencia permite obtener accionamientos de bajo volumen y peso.

Los bobinados de un motor brushless constan normalmente de tres fases con una separación de 120º entre ellas. A diferencia de los motores brushed convencionales donde la conmutación entre sus fases se realiza internamente de forma mecánica, en los motores brushless las corrientes y voltajes aplicados a cada uno de los bobinados del motor deben ser controlados independientemente mediante una conmutación electrónica. El dispositivo encargado de realizar esta tarea se denomina controlador de motor.

Existen dos grandes familias de controladores de motor diferenciadas principalmente en la utilización (sensored) o no (sensorless) de algún sensor para determinar la posición del rotor.

Actualmente, los motores BLDC se emplean en sectores industriales tales como: Automóvil, Aeroespacial, Consumo, Médico, equipos de automatización e instrumentación.

Como los motores BLDC tienen la característica de que no emplean escobillas en la conmutación para la transferencia de energía eliminan el gran problema que poseen los motores eléctricos convencionales con escobillas, los cuales producen rozamiento, disminuyen el rendimiento, desprenden calor, son ruidosos y requieren una sustitución periódica y, por tanto, un mayor mantenimiento.

2.8.1 ¿CÓMO FUNCIONAN?

Es simple, en vez de funcionar en DC funcionan en AC, la mayoría se alimentan con una señal trifásica, esta señal idealmente debería ser sinusoidal, pero en la práctica son pulsos, haciendo que la señal sea un continua pulsante o bien una continua con mucho componente de AC sin embargo se los clasifica como de DC porque al igual que los motores comunes tienen imanes permanentes. Estos imanes son atraídos por la polaridad de un campo magnético generado en las bobinas, las cuales como decíamos reciben pulsos en un patrón especifico. Si queremos que el motor gire más rápido, simplemente hacemos girar el campo magnético secuencial a mayor velocidad. O lo que sería lo mismo a aumentar la frecuencia de los pulsos. En el motor existen tres circuitos electromagnéticos conectados en un punto común. Cada circuito electromagnético se divide en el centro, permitiendo así el imán permanente del rotor a moverse en el medio del

campo magnético inducido. La mayoría de los motores BLDC tienen un bobinado trifásico con topología de conexión en estrella.

2.8.2 TIPOS DE MOTORES SIN ESCOBILLAS

Los motores BLDC se fabrican de dos tipos:

- Inrunner
- Outrunner.

Inrunner: Desarrollan mayor velocidad y suelen ser mas pequeños, entregan su torque máximo a muy altas revoluciones por minuto, por lo que se usan siempre con engranajes reductores. En estos motores el elemento móvil es el eje, sobre el cual se encuentran instalados los imanes permanentes.

Outrunner: Desarrollan su torque máximo a velocidades más bajas, por lo que usualmente no necesitan reducción, y se pueden acoplar directamente a una hélice. En estos los imanes permanentes están instalados en la carcasa externa del motor, que en este caso es la que gira y el bobinándose encuentra fijado al eje.

Estos motores trabajan por medio de variadores, también llamados controladores de velocidad (electronic speed controler o ESC), que transforman la corriente continua de las baterías en una tensión alterna trifásica y la

alimentan a los bobinados en cierta secuencia dependiendo de la posición del rotor. Para manejar los motores se precisa el conocimiento de la posición del rotor en cada momento, para lo cual se utilizan dos técnicas básicamente, dependiendo de la existencia o no de sensores en el motor, lo que los divide en dos familias: con sensores (sensored) y sin sensores (sensorless).

Sensored: Disponen de sensores de efecto hall o de encoders que indican la posición del rotor. Es habitual que tengan 3 sensores separados 120 grados, uno para cada bobinado del motor.

Sensorless: No tienen sensores; la posición se determina mediante la medición del efecto de la fuerza contra electromotriz sobre las bobinas. [7]

2.8.3 TÉCNICAS DE CONTROL

Las técnicas de control para motores brushless se pueden clasificar según el algoritmo de conmutación implementado. Las más utilizadas actualmente son:

- Conmutación trapezoidal (también llamada 6-steps mode o basada en sensores hall)
- Conmutación sinusoidal
- Control vectorial (Field Oriented Control).

Estas técnicas tienen básicamente como objetivo estimar la excitación óptima de cada una de las fases del motor y se diferencian principalmente por su complejidad de implementación, que se traduce en un incremento de prestaciones.

A continuación se describen algunas de las características más relevantes de cada técnica de control. **[9]**

CONTROL BASADO EN CONMUTACIÓN TRAPEZOIDAL

Uno de los métodos más simples de control de motores brushless es el llamado conmutación trapezoidal o 6-steps mode. En este esquema se controla la corriente que circula por los terminales del motor como se muestra en la Figura 2.12, excitando un par simultáneamente y manteniendo el tercer terminal desconectado. Sucesivamente se va alternando el par de terminales a excitar hasta completar las seis combinaciones posibles.

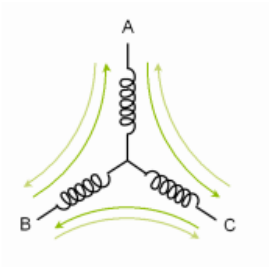


Figura 2.12 Esquema caminos de circulación de corriente

Tres sensores de efecto hall situados en el motor son utilizados para proporcionar la posición aproximada del rotor al controlador y que éste pueda determinar el próximo par de terminales a excitar. La Figura2.13 muestra el diagrama de bloques de un controlador trapezoidal típico con lazo cerrado de corriente.

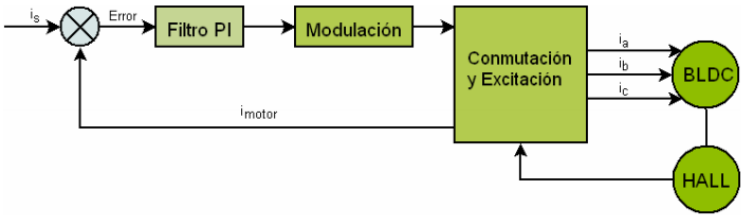


Figura 2.13 Esquema de un controlador con conmutación trapezoidal

La corriente que circula por el par de terminales activos es comparada con la corriente deseada y el error resultante es aplicado a un Filtro PI (Proporcional - Integrador). La salida de este filtro intenta corregir la desviación y por tanto minimizar el error. Con esta técnica se consigue mantener constante la corriente que circula por cualquiera de los bobinados del motor. Existen distintas técnicas de modulación orientadas a la generación de señales de excitación para motores Brushless mediante las cuales, se puede aumentar la eficiencia del sistema. Debido a que en todo momento las corrientes de dos bobinados son iguales en magnitud y la tercera siempre es nula, el vector de corrientes del

estator o resultado de la suma vectorial de las corrientes que circulan por las bobinas, sólo puede apuntar a 6 direcciones discretas como se muestra en la Figura 2.14.

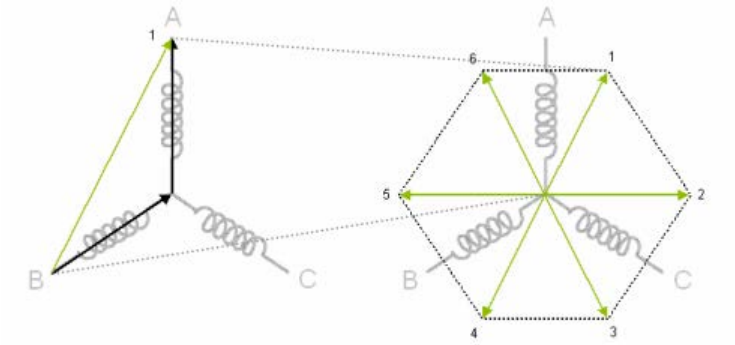


Figura 2.14 Ejemplo cálculo del vector de corrientes del estator

Dado que el vector de corrientes sólo puede apuntar en seis direcciones se produce una desalineación entre éstas y la posición real del rotor. En el peor de los casos, es decir cuando el rotor se encuentre en la posición intermedia de uno de los 6 sectores, la desalineación puede llegar a ser a ser de 30 grados. Esta desalineación genera un rizado en el par del motor como se muestra en la Figura 2.15 de aproximadamente el 15% (1-cos30º) a una frecuencia seis veces la velocidad de rotación del motor.

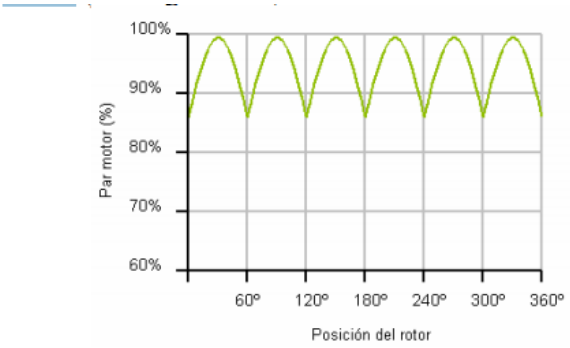


Figura 2.15 Rizado del par motor

Este rizado dificulta el control de motores brushless. En aplicaciones que demanden movimientos a baja velocidad se hace especialmente notable provocando una disminución en la precisión de dichos movimientos. Además puede ocasionar desgaste mecánico, vibraciones o ruido audible reduciendo las prestaciones y el tiempo de vida del motor.

No obstante gracias a su fácil implementación, esta técnica de conmutación viene siendo muy utilizada desde el inicio de los motores brushless especialmente en aplicaciones de bajo coste.

CONTROL BASADO EN CONMUTACIÓN SINUSOIDAL

La conmutación sinusoidal es vista como un control más avanzado y exacto que el trapezoidal, ya que intenta controlar la posición del rotor continuamente. Esta continuidad se consigue aplicando simultáneamente tres corrientes sinusoidales

desfasadas 120º a los tres bobinados del motor. La fase de estas corrientes se escoge de forma que el vector de corrientes resultante siempre esté en cuadratura con la orientación del rotor y tenga un valor constante. Como consecuencia de este procedimiento se obtiene un par más preciso y sin el rizado típico de la conmutación trapezoidal. No obstante, para poder generar dicha modulación sinusoidal es necesaria una medida precisa de la posición del rotor.

Debido a que los sensores de efecto hall solo proporcionan una posición aproximada es necesario el uso de otro dispositivo que aporte mayor precisión angular como puede ser un encoder. La siguiente Figura2.16 muestra el diagrama de bloques típico de un controlador con conmutación sinusoidal.

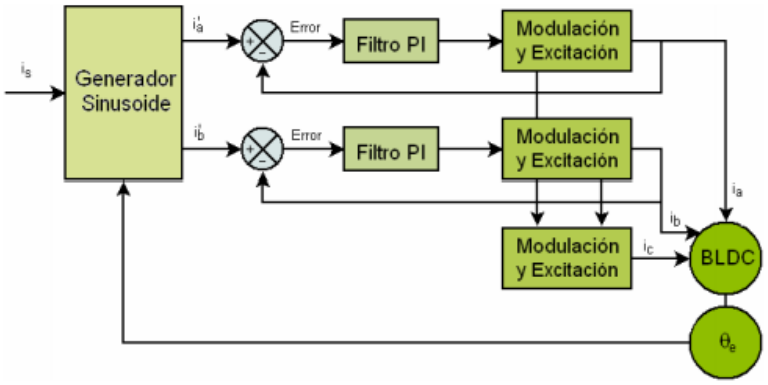


Figura 2.16 Esquema de un controlador con conmutación sinusoidal

Según la ley de Kirchoff mostrada en la ecuación 2.1 la suma de dos de las tres corrientes entrantes a un nodo es igual al valor negativo de la tercera:

$$\left(i_a + i_b = -i_c\right) \tag{2.1}$$

Por tanto, controlando dos de las corrientes aplicadas al motor se controla implícitamente la tercera o lo que es lo mismo la tercera corriente no puede ser controlada de forma independiente. En el caso que nos aplica la ecuación 2.2:

$$\begin{aligned} i_a &= i_s \times \sin\left(\theta_e\right) \\ i_b &= i_s \times \sin\left(\theta_e - 120°\right) \\ i_c &= i_s \times \sin\left(\theta_e - 240°\right) = -\left(i_a + i_b\right) \end{aligned} \tag{2.2}$$

(2.2)

Gracias a la información de la posición del rotor proporcionada por el encoder se sintetizan las dos sinusoidales deseadas, normalmente mediante el uso de una LUT(Look-up table). Éstas son comparadas con las medidas de las corrientes que circulan por el motor y el error resultante aplicado a dos Filtros PI que intentan corregir las desviaciones. La salida de los filtros es utilizada como entrada del generador de excitación que en la mayoría de los casos incorpora un modulador PWM. La conmutación sinusoidal soluciona los problemas de eficiencia que presenta la conmutación trapezoidal. Sin embargo, presenta problemas a altas velocidades de rotación del motor debido a la limitación frecuencial del bucle de corriente (Filtro PI). A mayor velocidad de rotación, mayor error y por tanto mayor desalineación entre el vector de corrientes y la

dirección de cuadratura del rotor. Este hecho provoca una progresiva disminución del par motor como se muestra en la Figura 2.17.

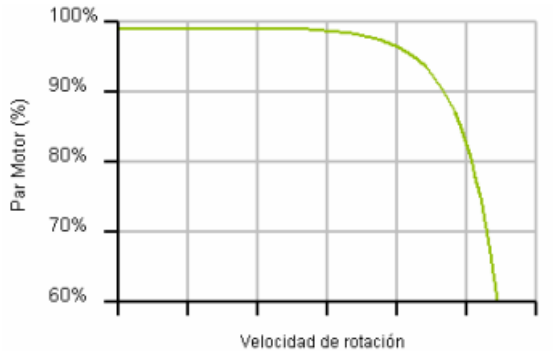


Figura 2.17 Par motor en función de la velocidad de rotación

Para mantener el par constante se necesita aumentar la corriente que circula por el motor provocando una disminución de la eficiencia. Este deterioro de la eficiencia aumenta al incrementarse la velocidad hasta llegar a un punto en el que el desfase entre el vector de corrientes y la dirección de cuadratura puede llegar a 90º produciendo un par motor completamente nulo.

CONTROL VECTORIAL

El control vectorial es el más complejo y el que requiere mayor potencia de cálculo de las tres técnicas. A su vez también es la que mejor control proporciona.

El problema principal que presenta la conmutación sinusoidal es que intenta controlar directamente las corrientes que circulan por el motor, las cuales son intrínsecamente variantes en el tiempo. Al aumentar la velocidad del motor, y por tanto la frecuencia de las corrientes, empiezan a aparecer problemas. El control vectorial o Field Oriented Control (FOC) soluciona el problema controlando el vector de corrientes directamente en un espacio de referencia ortogonal y rotacional, llamado espacio D-Q (DirectQuadrature).

Dicho espacio de referencia está normalmente alineado con en el rotor de forma que permite que el control del flujo y del par del motor se realice de forma independiente. La componente directa permite controlar el flujo y la componente en cuadratura el par.

Debido a que el vector de corrientes en el espacio de referencia D-Q es estático los filtros PI trabajan en continua y se eliminan por tanto los problemas frecuenciales de la conmutación sinusoidal.

Para poder realizar este control es necesario transformar matemáticamente las medidas de las tres corrientes referidas al espacio estático de las bobinas del motor al espacio rotacional D-Q. Aunque esta transformación puede implementarse en un único paso educacionalmente se divide en dos transformaciones.

- **Transformada de Clarke** - Transformación de un sistema de 3-fases equiespaciados (a,b,c) a uno de 2-fases ortogonales (α, β).
- **Transformada de Park** - Trasformación de un sistema ortogonal estacionario (α, β) a uno rotacional (d, q).

Al igual que en la conmutación sinusoidal es importante conocer la posición del rotor con exactitud. Un error en la estimación de ésta provocará que la componente directa y la componente cuadratura no estén totalmente desacopladas.

Una vez aplicadas las dos transformaciones el control del motor se simplifica considerablemente. Dos Filtros PI son utilizados para controlar la componente directa y la cuadratura de forma independiente. La componente en cuadratura es la única que proporciona par útil, por tanto, la referencia de la componente directa suele fijarse a cero. De esta forma se fuerza al vector de corrientes a situarse en la dirección de la componente de cuadratura maximizando la eficiencia del sistema. Posteriormente se realizan las transformadas inversas para regresar al espacio estacionario de las bobinas y se aplica la excitación correspondiente a cada una de las fases mediante modulación. El diagrama de bloques del control vectorial se muestra en la Figura 2.18.

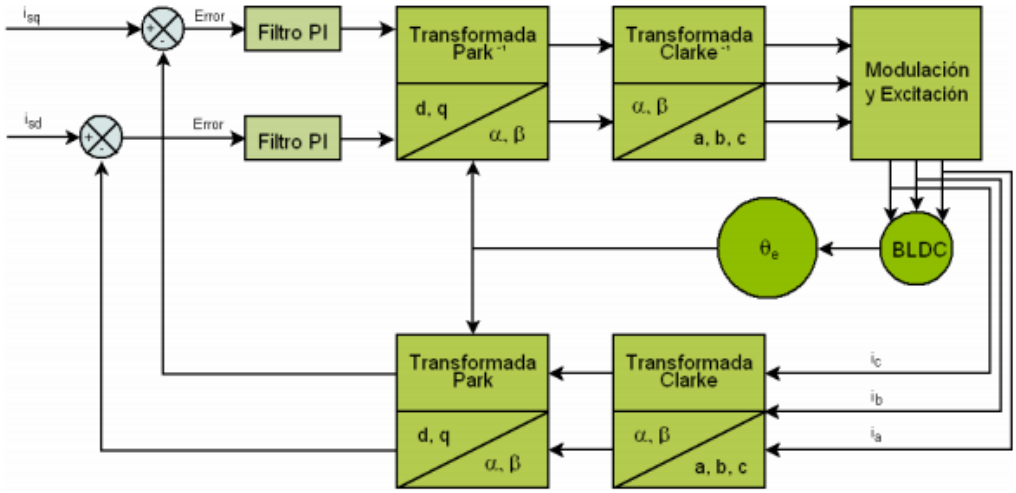


Figura 2.18 Esquema de controlador con control vectorial

Este tipo de control mantiene las mismas características de par que la conmutación sinusoidal pero eliminando la limitación frecuencial.

CAPÍTULO 3

DESARROLLO DEL PROYECTO

3.1 RESUMEN

En este capítulo mostraremos el desarrollo de nuestro proyecto, los pasos generales que utilizaremos para llevarlo a cabo con el fin de entender el funcionamiento de cada uno de los instrumentos usados y luego mostrar cómo han sido combinados para alcanzar el objetivo propuesto, para esto empezaremos con la programación de la AVR butterfly, programando la interfaz que interactúa con el usuario y le permitirá escoger las acciones a realizar luego adquisición de datos desde el sensor de temperatura a través del puerto de conversión analógico digital ADC, graficación de los datos que serán mostrados en el AVR Butterfly, envío mediante comunicación serial y por ultimo almacenamiento en memoria.

Para una mayor comprensión dividiremos nuestro proyecto en varias partes que serán mostradas como ejercicios individuales, así mostraremos como se puede avanzar paso por paso en la realización de este proyecto.

3.2 EJERCICIO 1: INTERFAZ DEL USUARIO EN AVR BUTTERFLY

Empezaremos programando la interfaz del usuario. Tendremos presente que a disposición del usuario está el Joystick con 5 opciones de movimiento, es decir 5 botones de los cuales usaremos solo 2, uno para mostrar la temperatura que está sensando y el otro para ver los datos guardados.

3.2.1 DIAGRAMA DE FLUJO

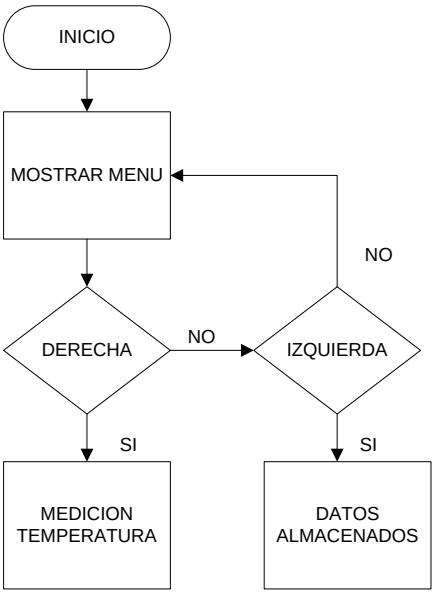


Figura 3.1 Secuencia Interfaz del usuario

ALGORITMO

1. Para dar inicio al programa, mostraremos un mensaje de inicialización

2. Mostrará en pantalla si desea medir temperatura presiones el joystick a la derecha, si desea ir a los datos almacenados presione a la izquierda.

3. Luego esperara a que se presiones uno de los 2 botones, si se presiona otro botón, simplemente el programa no hará nada.

3.2.2 CÓDIGO FUENTE

```
//************************LIBRERIAS A USAR************************
#include <avr/io.h>
#include <avr/delay.h>
#include <inttypes.h>
#include <avr/signal.h>
//************************LIBRERIAS LCD*****************************
#include "LCD_functions.h"
#include "LCD_driver.h"
#include "button.h"
//************************PROGRAMA PRINCIPAL**********************
int main(void)
{
 PGM_P statetext = PSTR("medicion temp DR datos IZ");
```

```
 // Enable pullups
PORTB = (15<<PB0);
   PORTE = (15<<PE4);
Button_Init();              // Initialize pin change interrupt on joystick
LCD_Init();                 // initialize the LCD
   CLKPR = (1<<CLKPCE);        // set Clock Prescaler Change Enable
    // set prescaler = 8, Inter RC 8Mhz / 8 = 1Mhz
CLKPR = (0<<CLKPS1) | (1<<CLKPS0);
while (1)
   {
         if (statetext){
                  LCD_puts_f(statetext, 1);
                  LCD_Colon(0);
                  statetext = NULL;
                   }
input = getkey();           // Read button
switch (input) {
//************************HACIA LA DERECHA************************
case KEY_NEXT:
         do{
```

```
                        if (statetext){
                                LCD_puts_f(statetext, 1);
                                LCD_Colon(0);
                                statetext = NULL;
                                        }
                        input = getkey();
                }while(input!=KEY_PREV);
break;
//**********************HACIA LA IZQUIERDA***********************
case KEY_PREV:
statetext = PSTR("DatosGuardados");
break;
   }
return 0;
}
//************************FIN DEL PROGRAMA************************
```

3.2.3 CONCLUSIÓN

Este ejercicio nos permitió aprender a mostrar mensajes a través de la pantalla LCD que posee la tarjeta AVR Butterfly y como crear un submenú al que podemos acceder fácilmente con la ayuda del joystick integrado en la misma tarjeta, todo esto es parte de la interfaz gráfica que será útil para la interacción del usuario.

3.3 EJERCICIO 2: ADQUISICIÓN DE DATOS DE TEMPERATURA Y GRAFICACIÓN DE LOS DATOS EN AVR BUTTERLFY.

Para la medición de temperatura usamos el convertidor analógico digital para adquirir los datos del sensor LM35 que nos proporciona 10mv por cada grado centígrado. Para esto tenemos que configurar los módulos de ADC de la AVR butterfly.

3.3.1 CONFIGURACIÓN DEL ADC

Para configurar primeramente tenemos que tener en cuenta que registro es necesario modificar.

El registro es el ADCSRA este registro tiene los bit para habilitar el modulo de conversión ADEN y los bits necesarios para establecer la frecuencia del ADC,

en este caso con ADPS1 y ADPS0 en 1 estamos fijando una frecuencia de 1MHz / 8 = 125kHz

La función ADC_init(); nos permite configurar este registro de esta manera de una manera más sencilla solo necesitamos llamar a esta función.

3.3.2 DIAGRAMA DE FLUJO

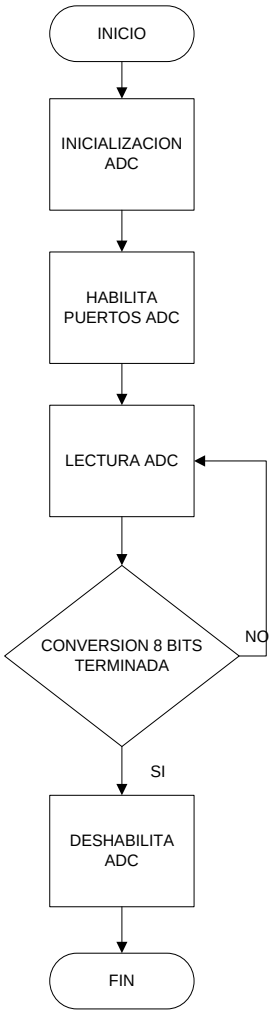


Figura 3.2 Secuencia conversión ADC

ALGORITMO

1. Inicializamos ADC con ayuda del registro ADCSRA
2. Habilitamos los puertos como salidas para leer el ADC
3. Leemos el ADC y esperamos que termine la conversión a 8 bits
4. Deshabilitamos la lectura ADC.

3.3.3 GRAFICACIÓN DE LOS DATOS

Una vez que tenemos los datos de temperatura ya listos para usar, mostramos estos datos mediante la pantalla LCD que tiene la AVR butterfly.

3.3.4 DIAGRAMA DE FLUJO

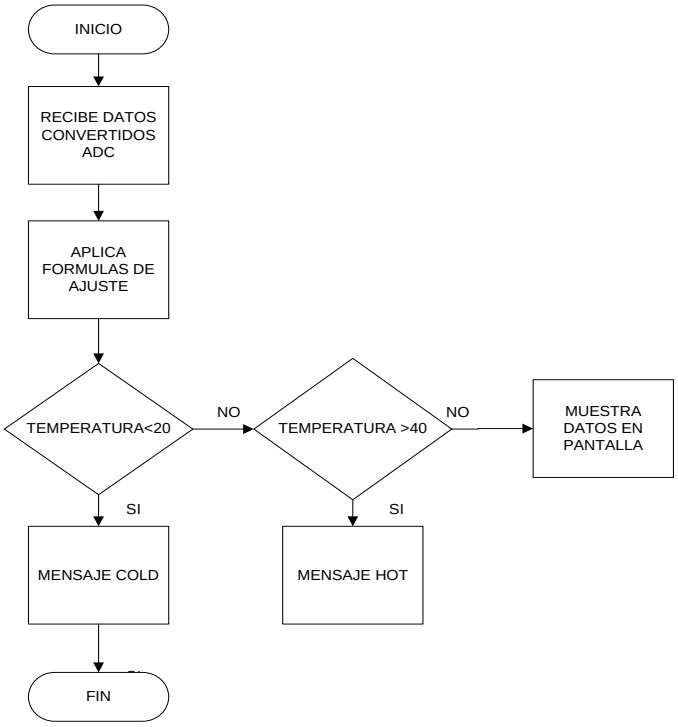


Figura 3.3 Secuencia Graficacion en AVR Butterfly

ALGORITMO

1. Recibir Datos después de la conversión ADC
2. Aplicar las fórmulas de ajuste correspondientes
3. Si se encuentra entre el rango de 20 a 40 grados mostrar en pantalla del AVR butterfly la temperatura caso contrario mostrar mensajes HOT y COLD si es menor o mayor respectivamente.

3.3.5 CÓDIGO FUENTE

```
//************************LIBRERIAS A USAR************************
#include <avr/io.h>
#include <avr/delay.h>
#include <inttypes.h>
#include <avr/signal.h>
//************************LIBRERIAS LCD************************
#include "LCD_functions.h"
#include "LCD_driver.h"
#include "button.h"
//************************INICIALIZACION ADC************************
voidADC_init(char input)
{
ADMUX=input;
```

```
ADCSRA = (1<<ADEN) | (1<<ADPS1) | (1<<ADPS0);
input=ADC_read();
}
//*************************LECTURA ADC*************************
intADC_read(void)
{
        chari;
        intADC_temp;
        intADCr = 0;
        // Enable the VCP (VC-peripheral)
        PORTF=(1<<PF3); Habilita puertos
        DDRF=(1<<DDF3);
        ADCSRA |= (1<<ADSC); // do single conversion
        // wait for conversion done, ADIF flag active
        while(!(ADCSRA & 0x10));
//************************CONVERSION ADC*************************
// do the ADC conversion 8 times for better accuracy
        for(i=0;i<8;i++)
        {       ADCSRA |= (1<<ADSC); // do single conversion
                // wait for conversion done, ADIF flag active
```

```
                while(!(ADCSRA & 0x10));
                        ADC_temp = ADCL; // read out ADCL register
                        ADC_temp += (ADCH << 8); // read out ADCH register
                        // accumulate result (8 samples) for later averaging
                        ADCr += ADC_temp;
        }
//**********************DESHABILTA ADC          **************************
        ADCr = ADCr>> 3; // average the 8 samples
        PORTF=(0<<PF3); Disable ADC
        DDRF=(0<<DDF3);
        returnADCr;
}
//**********************PROGRAMA PRINCIPAL**********************
int main(void)
{
    PGM_P statetext = PSTR("medicion temp DR datos IZ");
uint8_t input;

inti;
int temp;
chartempC;

// Enable pullups
PORTB = (15<<PB0);
  PORTE = (15<<PE4);
          DDRF=0x00;
```

```
Button_Init();            // Initialize pin change interrupt on joystick
ADC_init(1);
LCD_Init();               // initialize the LCD

   CLKPR = (1<<CLKPCE);        // set Clock Prescaler Change Enable
    // set prescaler = 8, Inter RC 8Mhz / 8 = 1Mhz
    CLKPR = (0<<CLKPS1) | (1<<CLKPS0);
USART_Init(25.04);
  //sei();

while (1)
   {
                if (statetext){

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

input = getkey();          // Read buttons

switch (input) {

case KEY_NEXT:
                        do{
                                if (statetext){
                                LCD_puts_f(statetext, 1);
                LCD_Colon(0);
                statetext = NULL;
                                }

                        temp=ADC_read();
                        temp=temp/2;
                        temp=temp/1.6;
                        if(temp<20){statetext = PSTR("COLD");}
                        if(temp>40){statetext = PSTR("HOT");}
                        if(temp>19 && temp<41){
                        statetext= muestraDato(temp);
                        _delay_ms(1000);
                                                }
```

```
                        input = getkey();
                        }while(input!=KEY_PREV);
        break;
return 0;
}
//********************FUNCION QUE IMPRIME DATOS*******************

intmuestraDato(intdato){
        PGM_P statetext;

        switch (dato) {
                case 20:
                        statetext = PSTR("20c");
                        break;
                case 21:
                        statetext = PSTR("21c");
                        break;
                case 22:
                        statetext = PSTR("22c");
                        break;
                case 23:
                        statetext = PSTR("23c");
                        break;
                case 24:
                        statetext = PSTR("24c");
                        break;
                case 25:
                        statetext = PSTR("25c");
                        break;
                case 26:
                        statetext = PSTR("26c");
                        break;
                case 27:
                        statetext = PSTR("27c");
                        break;
                case 28:
                        statetext = PSTR("28c");
                        break;
                case 29:
                        statetext = PSTR("29c");
```

```
                break;
            case 30:
                statetext = PSTR("30c");
                break;
            case 31:
                statetext = PSTR("31c");
                break;
            case 32:
                statetext = PSTR("32c");
                break;
            case 33:
                statetext = PSTR("33c");
                break;
            case 34:
                statetext = PSTR("34c");
                break;
            case 35:
                statetext = PSTR("35c");
                break;
            case 36:
                statetext = PSTR("36c");
                break;
            case 37:
                statetext = PSTR("37c");
                break;
            case 38:
                statetext = PSTR("38c");
                break;
            case 39:
                statetext = PSTR("39c");
                break;
            case 40:
                statetext = PSTR("40c");
                break;
            default:
                break;
        }
        returnstatetext;
}
        //**********************FIN DEL PROGRAMA**************************
```

3.3.6 CONCLUSIÓN

Este ejercicio es muy útil ya que nos permite aprender dos cosas importantes primero como se deben configurar los registros en el AVR Butterfly para que convierta los datos recibidos por el canal analógico a digital y luego de esto nos lo muestre en la pantalla LCD de la misma tarjeta, cabe recalcar que para ser mostrados necesitan tener un ajuste para escalarlo en el rango que deseamos.

3.4 AJUSTE DE LOS DATOS ADQUIRIDOS

Como mencionamos antes, el sensor nos da una lectura de 10mv/°c y la resolución que nos muestra al ADC de la butterfly es de 10 bits por lo tanto tenemos que escalar el dato obtenido para mostrarlo en un rango de 0 a 99 °C.

Como el ADC lee datos de 0 a 5V esto nos da una resolución en 10 bits de 0 a 1111111111, es decir en decimal de 0 a 1023, Siendo 5v 1023, entonces como no usaremos todo el rango de 0 a 5v, por que el sensor no provee tal magnitud de voltaje y necesitamos escalarlo en un rango de 0.0 a 0.99mV a un rango de 0 a 99°C para tener una lectura exacta de 1°c por cada 10mV que varía el sensor.

Para esto el dato obtenido de la conversión ADC lo guardaremos en la variable ADC obtenida y la dividiremos para 2.6, de esa manera reduciríamos 2.6 veces el máximo valor, pero como no usaremos el máximo valor por que el sensor

nunca llegara a 5 volt, sin embargo la pendiente da una relación lineal unitaria es decir:

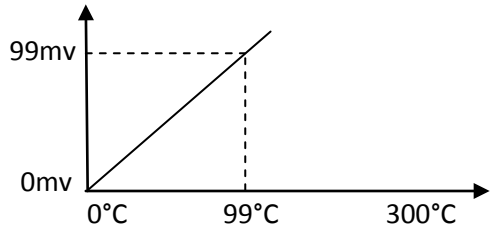


Figura 3.4Relacion Voltios-Temperatura

FÓRMULAS NECESARIAS PARA EL AJUSTE

temp=ADC_read();

temp=temp/2;

temp=temp/1.6;

3.5 EJERCICIO 3: TRANSMISIÓN UART EN AVR BUTTERFLY

Con los datos obtenidos de temperatura procederemos a mandar mediante comunicación serial códigos que nos ayudaran en la parte receptora en este caso la tarjeta LPCXpresso a mostrar mensajes de operación.

3.5.1 DIAGRAMA DE FLUJO

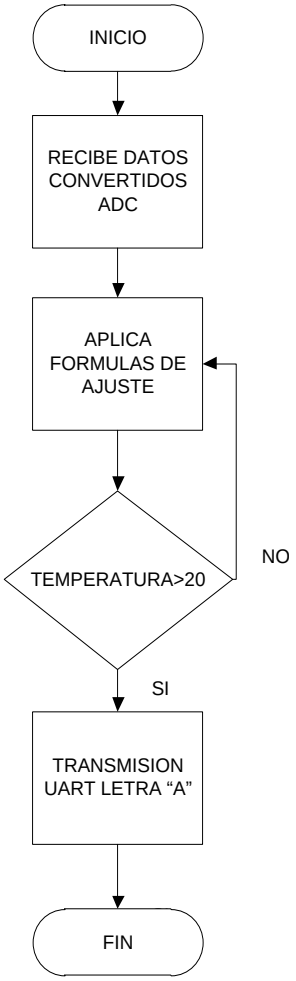


Figura 3.5Secuencia Transmisión UART

ALGORITMO

1. Recibir Datos después de la conversión ADC
2. Aplicar las fórmulas de ajuste correspondientes
3. Si la temperatura se encuentra menor a 20 ºC se transmite mediante comunicación serial a la LPCXpresso la letra “A”.

3.5.2 CÓDIGO FUENTE

```
//************************LIBRERIAS A USAR************************

#include <avr/io.h>

#include <avr/delay.h>

#include <inttypes.h>

#include <avr/signal.h>

//************************LIBRERIAS LCD*****************************

#include "LCD_functions.h"

#include "LCD_driver.h"

#include "button.h"

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

int main(void)
{
   PGM_P statetext = PSTR("medicion temp DR datos IZ");
uint8_t input;

inti;
```

```
int temp;

  // Enable pullups
PORTB = (15<<PB0);
  PORTE = (15<<PE4);
  DDRF=0x00;

Button_Init();            // Initialize pin change interrupt on joystick
ADC_init(1);
LCD_Init();               // initialize the LCD
  CLKPR = (1<<CLKPCE);       // set Clock Prescaler Change Enable
   // set prescaler = 8, Inter RC 8Mhz / 8 = 1Mhz
   CLKPR = (0<<CLKPS1) | (1<<CLKPS0);
USART_Init(25.04);
  //sei();

while (1)
   {
                if (statetext){

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

input = getkey();          // Read buttons

switch (input) {
case KEY_NEXT:

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

                        temp=ADC_read();
                        temp=temp/2;
                        temp=temp/1.6;
```

```
                        if(temp>20){
                        _delay_ms(1000);
                        Usart_Tx('A');
                        }
                        input = getkey();
                        }while(input!=KEY_PREV);
break;
}
return 0;
}
//**********************FIN DEL PROGRAMA*************************
```

3.5.3 CONCLUSIÓN

Este ejercicio es muy sencillo pero muestra claramente cómo se puede transmitir datos usando la comunicación serial llamada UART o USART, la cual nos permite transmitir ya sea un código o en nuestro caso un solo caracter dependiendo de la seguridad o el medio en el que se va a transmitir ya que si usamos módulos inalámbricos podría existir interferencia.

3.6 EJERCICIO 4: ALMACENAR DATOS EN MEMORIA BUTTERFLY

Para este ejercicio usamos la memoria de la AVR butterfly para guardar los datos de la medición de temperatura, para lograr nuestro objetivo podemos usar la memoria externa del dispositivo o la propia memoria interna del microcontrolador, guardando en un arreglo o Array de datos, declarando el

arreglo como una variable con [n] datos, en este caso usaremos la memoria interna DatosGuardados[n].

3.**6.1 DIAGRAMA DE FLUJO**

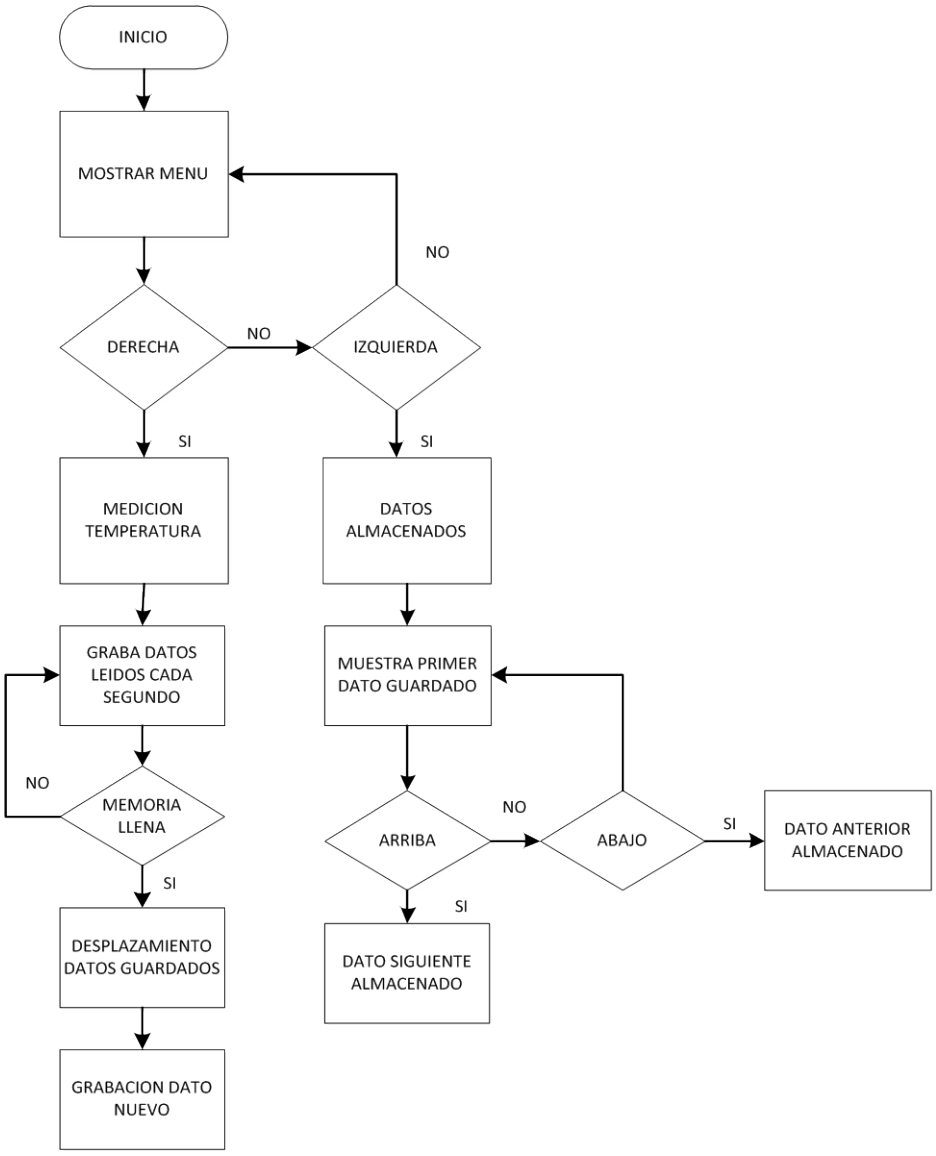


Figura 3.6Secuencia Almacenamiento Datos

ALGORITMO

1. Seleccionar una opción del menú
2. Si se presiona el joystick hacia la derecha comenzaremos con la medición de temperatura, estos datos van a ser almacenados automáticamente cada segundo, si se presiona hacia la izquierda entraremos a los datos almacenados, en este caso las ultimas 15 mediciones, las cuales podemos observar con las teclas arriba y abajo para avanzar o retroceder en los 15 datos almacenados.

3.6.2 CÓDIGO FUENTE

```
//**********************LIBRERIAS A USAR**************************
#include <avr/io.h>
#include <avr/delay.h>
#include <inttypes.h>
#include <avr/signal.h>
#include "button.h"
//*************************LIBRERIAS LCD*************************
#include "LCD_functions.h"
#include "LCD_driver.h"
//********************FUNCION QUE GRABA DATOS********************
```

```
voidgrabaDato(intdato,inti){

        int k=0;

        if(i<16){datoGuardado[i]=dato;}

        if(i<256){

        for(k=16;k>0;k--){

        datoGuardado[k]=datoGuardado[k-1];
        }
        datoGuardado[0]=dato;
        }
//*********************PROGRAMA PRINCIPAL************************

intADC_read(void);
intmuestraDato(intdato);
voidgrabaDato(intdato,inti);
voidADC_init(char input);
intdatoGuardado[15];
int j;
intmain(void)
{
PGM_P statetext = PSTR("medirdrmemoriaiz");
uint8_t input;

inti=0;
int temp;
// Enablepullups
PORTB = (15<<PB0);
  PORTE = (15<<PE4);
DDRF=0x00;

Button_Init();          // Initialize pin change interrupt on joystick
ADC_init(1);
LCD_Init();             // initialize the LCD

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

```
   CLKPR = (0<<CLKPS1) | (1<<CLKPS0);
USART_Init(25.04);
  //sei();

while (1)
   {
                if (statetext){

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

input = getkey();          // Read buttons

switch (input) {
case KEY_ENTER:
        break;
//*************OPCION MEDIR Y GUARDARTEMPERATURA**************
case KEY_NEXT:
statetext = PSTR("medicion Temp");
                        i=i+1;
                        if(i>1){
                        i=0;
                        j=0;
                        do{
                                if (statetext){
                                LCD_puts_f(statetext, 1);
                LCD_Colon(0);
                statetext = NULL;
                                }

                        temp=ADC_read();
                        temp=temp/2;
                        temp=temp/1.6;
                        if(temp<20){statetext = PSTR("cold");grabaDato(-1,j);
                        j=j+1;_delay_ms(3000);}
                        if(temp>40){statetext = PSTR("hot");grabaDato(100,j);
                        j=j+1;_delay_ms(3000);}
                        if(temp>19 && temp<41){
```

```
                                statetext= muestraDato(temp);
                                grabaDato(temp,j);
                                j=j+1;
                        if(j>300){j=16;}
                                _delay_ms(3000);
                        }
                        input = getkey();
                        }while(input!=KEY_PREV);
}
                                break;
//******************OPCION VER DATOS GUARDADOS******************

case KEY_PREV:
statetext = PSTR("datosguardados");
                        i=i+1;
                        if(i>2){
                        i=0;
                        do{
                                if (statetext){
                                LCD_puts_f(statetext, 1);
                LCD_Colon(0);
                statetext = NULL;
                                }
                        input = getkey();
                        if(input==KEY_PLUS){i=i+1;}
                        if(i>=16){i=i-1;}
                        if(input==KEY_MINUS){i=i-1;}
                        if(i<0){i=i+1;}
                        statetext=muestraDato(datoGuardado[i]);
}while(input!=KEY_NEXT);
                        i=0;
                        }
break;
default:
break;
   }
return 0;
}

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

3.6.3 CONCLUSIÓN

Con este ejercicio podemos concluir que la tarjeta AVR Butterfly posee una memoria interna que podemos hacer uso para almacenar una gran cantidad de datos y luego mostrarlos como en este caso que se almacenaron 15 valores de temperatura, todo esto es posible mediante la creación de un simple arreglo del tamaño deseado.

3.6 EJERCICIO 5: RECEPCIÓN DATOS MEDIANTE UART A TARJETA LPCXPRESSO Y ENCENDIDO LEDS

En este ejercicio mostraremos el manejo de la tarjeta LPCXpresso para la recepción datos mediante comunicación serial UART desde la AVR Butterfly y como basándonos en estos datos mostramos en que rango se encuentra la temperatura medida.

3.6.1 DIAGRAMA DE FLUJO

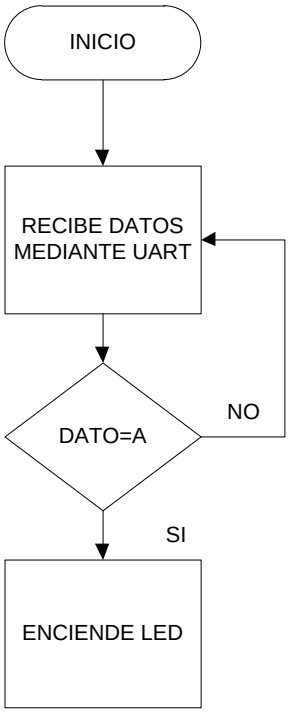


Figura 3.7Secuencia Encendido LED

ALGORITMO

1. Recibir datos mediante UART desde AVR Butterfly
2. Si el dato recibido coincide con la letra “A” será encendido un led.

3.6.2 CÓDIGO FUENTE

```
//************************LIBRERIAS A USAR************************
#include "LPC17xx.h"
#include "type.h"
#include "uart.h"
#include <string.h>
//***********************PROGRAMA PRINCIPAL************************

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

int main (void)
{
	uint32_ti, j;
	LPC_GPIO2->FIODIR = 0xFFFFFFFF;		/* P2.xx defined as
Outputs */
	LPC_GPIO2->FIOCLR = 0xFFFFFFFF;
	LPC_GPIO2->FIOSET = 0x0000FFFF;
	UARTInit(3, 9600);  /* baud rate setting */

	while (1)
	{
		if ( UART3Count != 0 )
		{
			LPC_UART3->IER = IER_THRE | IER_RLS;	/*	Disable
RBR */

			if(*UART3Buffer==0x52){
				LPC_GPIO2->FIOCLR = 0x0000000F;
				 LPC_GPIO2->FIOSET = 1 << 1;
					}
			UART3Count = 0;
		LPC_UART3->IER = IER_THRE | IER_RLS | IER_RBR;	/*Re-
enable RBR */
		}
	}
}
//************************FIN DEL PROGRAMA************************
```

3.6.3 CONCLUSIÓN

Este ejercicio nos permite concluir que para leer datos recibidos por comunicación serial UART en la tarjeta LPCXpresso es necesario configurar primero los registros correspondientes y en base a estos datos recibidos podemos mandar cualquier tipo de orden con una simple comparación en el Buffer del Uart que se está utilizando, esta deberá estar en hexadecimal.

3.7 PROYECTO TERMINADO

Todos los ejercicios mencionados anteriormente ayudaron a la culminación de nuestro proyecto que consiste en medir la temperatura de un motor, mostrar estos datos en la AVR Butterfly además de permitir almacenarlos y poder acceder a ellos, luego enviar datos mediante comunicación serial a la tarjeta LPCXpresso y mostrar mensajes de operación en base a la temperatura medida.

3.7.2 DIAGRAMA DE BLOQUES

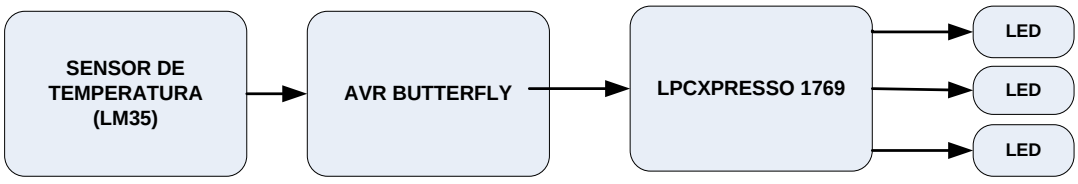


Figura 3.8Diagrama Bloques proyecto final

3.7.1DIAGRAMA DE FLUJO PARA TARJETA AVR BUTTERFLY

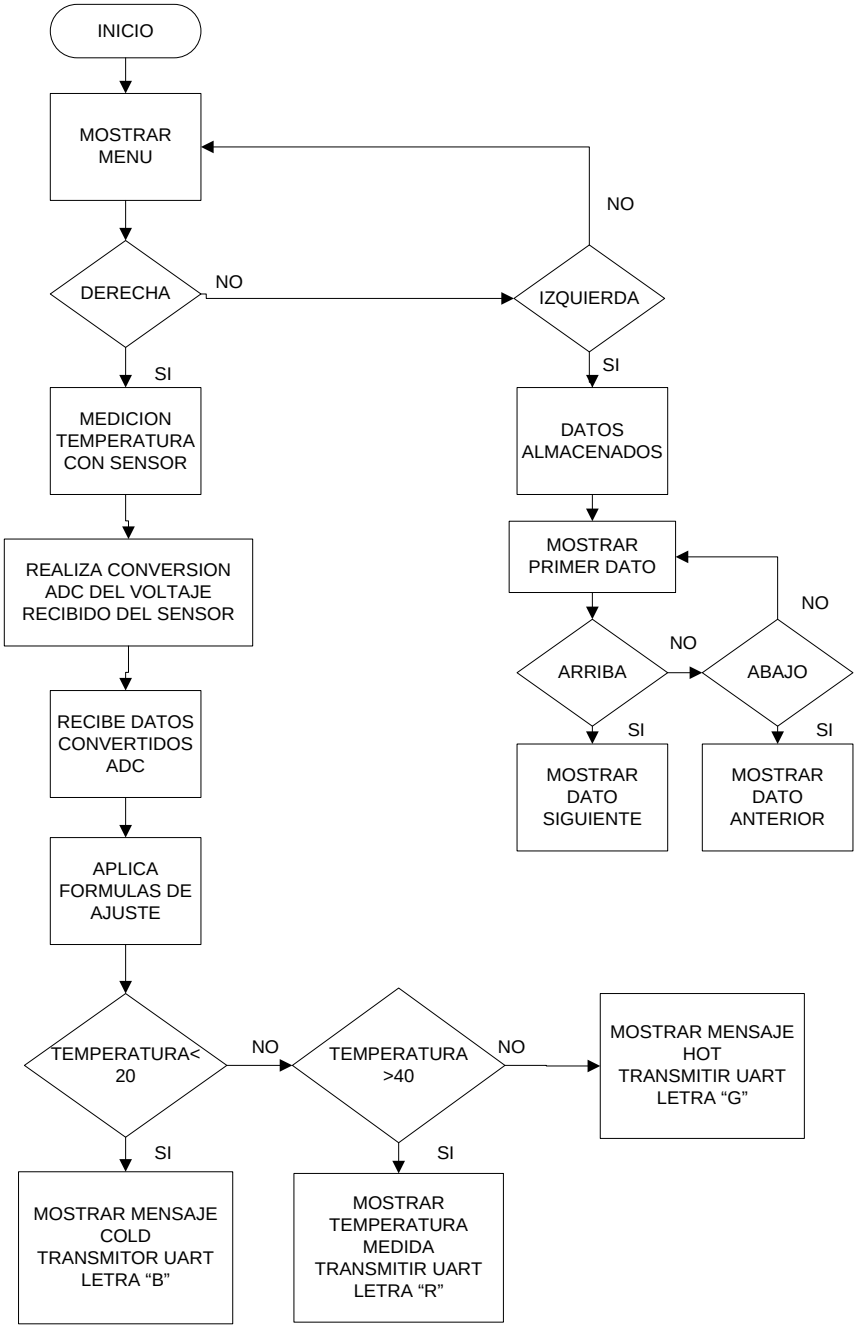


Figura 3.9Funcionamiento proyecto final

ALGORITMO

1. Mostrar menú principal, consta de dos opciones Medición de Temperatura presionar el joystick hacia la derecha y para ver datos guardados presionarlo hacia la derecha.

2. Si escogemos la opción izquierda procederemos a medir datos mediante nuestro sensor de temperatura (LM35) los cuales van a ser enviados al AVR Butterfly, este va a convertir estos datos analógicos a digital y con su respectiva relación voltios-centígrados se ajusta el valor para ser mostrados en la pantalla, si este valor es menor a 20 grados se imprimirá el mensaje COLD y se transmitirá mediante UART la letra “B”, si se encuentra entre 20 y 40 grados imprimirá la temperatura medida y se transmitirá mediante UART la letra “G” y si supera los 40 grados se imprimirá el mensaje HOT y se transmitirá mediante UART la letra “R”.

3. Si escogemos la opción derecha accederemos a los datos guardados, en este caso serán los últimos 15 medidos, tenemos la opción de mover hacia arriba para ver el dato siguiente y hacia abajo para ver el dato anterior.

3.7.2 CÓDIGO FUENTE AVR BUTTERFLY

```
//************************LIBRERIAS A USAR************************

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <avr/delay.h>
#include <inttypes.h>0
#include <avr/signal.h>
#include <stdlib.h>
#include "mydefs.h"
#include "LCD_functions.h"
#include "LCD_driver.h"
#include "button.h"
#include "usart.h"

//*****************DECLARACION DE FUNCIONES**********************
intADC_read(void);
intmuestraDato(intdato);
voidgrabaDato(intdato,inti);
voidADC_init(char input);
intdatoGuardado[15];
int j;

//**********************PROGRAMA PRINCIPAL************************
intmain(void)
{
PGM_P statetext = PSTR("medirdrmemoriaiz");
uint8_t input;
inti=0;
int temp;
chartempC;

// Enablepullups
PORTB = (15<<PB0);
  PORTE = (15<<PE4);
  DDRF=0x00;
```

```
Button_Init();            // Initialize pin change interrupt on joystick
ADC_init(1);
LCD_Init();               // initialize the LCD

   CLKPR = (1<<CLKPCE);        // set Clock Prescaler Change Enable
    // set prescaler = 8, Inter RC 8Mhz / 8 = 1Mhz
    CLKPR = (0<<CLKPS1) | (1<<CLKPS0);
USART_Init(25.04);

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


input = getkey();          // Read buttons

switch (input) {
case KEY_ENTER:

break;

//***********************OPCION DERECHA**************************
case KEY_NEXT:
statetext = PSTR("medicion Temp");
                        i=i+1;
                        if(i>1){
                        i=0;j=0;
                        do{
                                if (statetext){
                                LCD_puts_f(statetext, 1);
                LCD_Colon(0);
                statetext = NULL;
                                }

                        temp=ADC_read();
                        temp=temp/2;
```

```
                        temp=temp/1.6;
                        if(temp<20){Usart_Tx('B');statetext                    =
PSTR("cold");grabaDato(-1,j);
                        j=j+1;_delay_ms(3000);}
                        if(temp>40){Usart_Tx('R');statetext                    =
PSTR("hot");grabaDato(100,j);
                        j=j+1;_delay_ms(3000);}
                        if(temp>19 && temp<41){
                        statetext= muestraDato(temp);
                        grabaDato(temp,j);
                        j=j+1;
                        if(j>300){j=16;}
                        _delay_ms(3000);
                        Usart_Tx('G');
                        }
                        input = getkey();
                        }while(input!=KEY_PREV);
}
        break;

//************************OPCION IZQUIERDA************************
case KEY_PREV:
statetext = PSTR("datosguardados");
                        i=i+1;
                        if(i>2){
                        i=0;
                        do{
                                if (statetext){
                                LCD_puts_f(statetext, 1);
                LCD_Colon(0);
                statetext = NULL;
                                }
                        input = getkey();
                        if(input==KEY_PLUS){i=i+1;}
                        if(i>=16){i=i-1;}
                        if(input==KEY_MINUS){i=i-1;}
                        if(i<0){i=i+1;}
                        statetext=muestraDato(datoGuardado[i]);
}while(input!=KEY_NEXT);
                        i=0;
```

```
                        }
break;
        }
   }
return 0;
}
//**********************FUNCIONES CREADAS************************
intADC_read(void)
{
        chari;
        intADC_temp;
        // mtint ADC = 0 ;
        intADCr = 0;
        // To save power, the voltage over the LDR and the NTC is
        // turned off when not used. T is is done by controlling the
        // voltage from an I/O-pin (PORTF3)

        PORTF=(1<<PF3); //sbi(PORTF, PF3); // Enable the VCP (VC-peripheral)
        DDRF=(1<<DDF3); //sbi(DDRF, DDF3); // sbi(DDRF, PORTF3);

        //do a dummy readout first
        ADCSRA |= (1<<ADSC); // do single conversion
        // wait for conversion done, ADIF flag active
        while(!(ADCSRA & 0x10));
        // do the ADC conversion 8 times for better accuracy
        for(i=0;i<8;i++)
        {
                ADCSRA |= (1<<ADSC); // do single conversion

                // wait for conversion done, ADIF flag active
                while(!(ADCSRA & 0x10));

                        ADC_temp = ADCL; // read out ADCL register
                        ADC_temp += (ADCH << 8); // read out ADCH register
                        // accumulate result (8 samples) for later averaging
                        ADCr += ADC_temp;
        }

        ADCr = ADCr>> 3; // average the 8 samples
        PORTF=(0<<PF3);//cbi(PORTF,PF3); // disable the VCP
```

```
        DDRF=(0<<DDF3);//cbi(DDRF,DDF3); // mtcbi(DDRF, PORTF3);
        returnADCr;
}
voidADC_init(char input)
{
ADMUX=input;
ADCSRA = (1<<ADEN) | (1<<ADPS1) | (1<<ADPS0);
input=ADC_read();
}

voidgrabaDato(intdato,inti){

        int k=0;
        if(i<16){datoGuardado[i]=dato;}
        if(i<256){
        for(k=16;k>0;k--){
        datoGuardado[k]=datoGuardado[k-1];
        }
        datoGuardado[0]=dato;
        }
}
intmuestraDato(intdato){
        PGM_P statetext;
        switch (dato) {
                case -1:
                        statetext = PSTR("min");
                        break;
                case 100:
                        statetext = PSTR("max");
                        break;
                case 20:
                        statetext = PSTR("20c");
                        break;
                case 21:
                        statetext = PSTR("21c");
                        break;
                case 22:
                        statetext = PSTR("22c");
                        break;
                case 23:
```

```
            statetext = PSTR("23c");
            break;
case 24:
            statetext = PSTR("24c");
            break;
case 25:
            statetext = PSTR("25c");
            break;
case 26:
            statetext = PSTR("26c");
            break;
case 27:
            statetext = PSTR("27c");
            break;
case 28:
            statetext = PSTR("28c");
            break;
case 29:
            statetext = PSTR("29c");
            break;
case 30:
            statetext = PSTR("30c");
            break;
case 31:
            statetext = PSTR("31c");
            break;
case 32:
            statetext = PSTR("32c");
            break;
case 33:
            statetext = PSTR("33c");
            break;
case 34:
            statetext = PSTR("34c");
            break;
case 35:
            statetext = PSTR("35c");
            break;
case 36:
            statetext = PSTR("36c");
```

```
                    break;
                case 37:
                    statetext = PSTR("37c");
                    break;
                case 38:
                    statetext = PSTR("38c");
                    break;
                case 39:
                    statetext = PSTR("39c");
                    break;
                case 40:
                    statetext = PSTR("40c");
                    break;
                default:
                    break;
        }
        returnstatetext;
}
//**********************FIN DEL PROGRAMA*************************
```

3.7.3 DIAGRAMA DE FLUJO PARA LPCXPRESSO

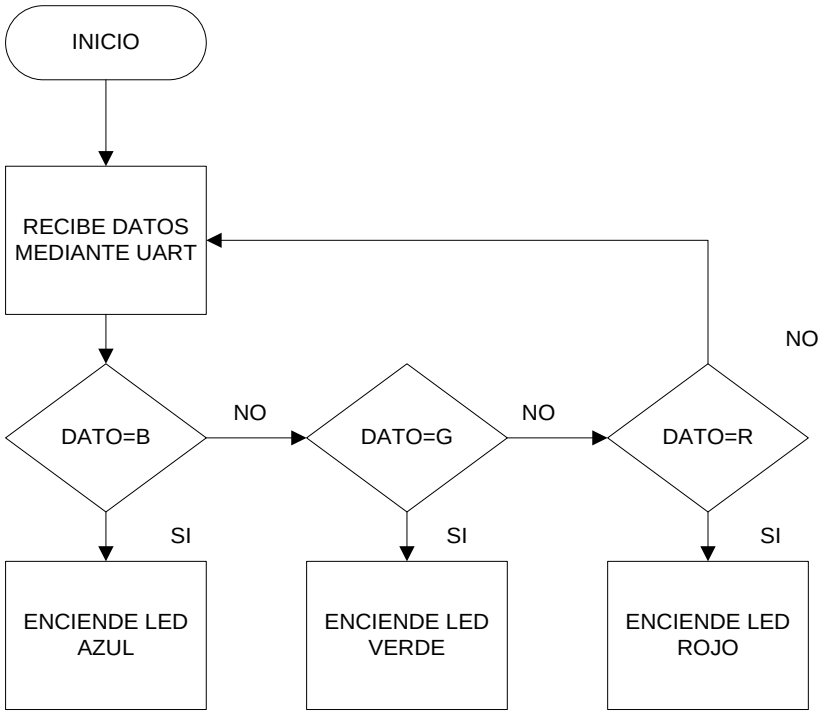


Figura 3.10Funcionamiento receptor

ALGORITMO

1. Recibir datos mediante UART desde AVR Butterfly

2. Si el dato recibido coincide con la letra “B” será encendido un led color azul que indicara que se encuentra en estado COLD y se mostrara en el display la temperatura máxima en este estado, si el dato recibido es la letra “G” será encendido un led color verde que indicara que está funcionando a la temperatura normal se mostrara en el display donde comienza este rango, si la

letra recibida es “R” será encendido un led color rojo que indicara que la temperatura esta pasándose de lo normal establecido y el display mostrara ese valor.

3.7.4 CÓDIGO FUENTE LPCXPRESSO

```
******************************************************************************/
#include "LPC17xx.h"
#include "type.h"
#include "uart.h"
#include <string.h>

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

/******************************************************************************
**   Main Function  main()
This program has been test on LPCXpresso 1700.
******************************************************************************/
int main (void)
{
	uint32_t i, j;

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


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


	/* Loop forever */
	while (1)
	{
		if ( UART3Count != 0 )
		{
```

```
                LPC_UART3->IER = IER_THRE | IER_RLS;
            /* Disable RBR */

                if(*UART3Buffer==0x42){
                    LPC_GPIO2->FIOCLR = 0x000000FF;
                    LPC_GPIO2->FIOSET = 1 << 1;

                                }
                if(*UART3Buffer==0x52){
                    LPC_GPIO2->FIOCLR = 0x000000FF;
                    LPC_GPIO2->FIOSET = 1 << 2;

                                }
                if(*UART3Buffer==0x47){
                    LPC_GPIO2->FIOCLR = 0x000000FF;
                    LPC_GPIO2->FIOSET = 1 << 3;

                                }


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

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

CAPÍTULO 4

SIMULACIONES

4.1 RESUMEN

A continuación mostramos los resultados obtenidos en cada uno de los ejercicios propuestos en el capítulo anterior que son los pasos para poder cumplir con los objetivos de este tema de proyecto.

En cada ejercicio se mostrara su simulación con su debida explicación de cómo se realiza la misma y luego los resultados de la implementación real con una breve descripción del esquema realizado y finalmente comentaremos los desafíos enfrentados para la realización de cada ejercicio.

4.2 SIMULACIÓN EJERCICIO 1

La figura 4.1 muestra la primera opción de nuestro menú en el AVR Butterfly en este caso es “MEDIR DR” que significa que para medir la temperatura hay que mover el joystick hacia la derecha.

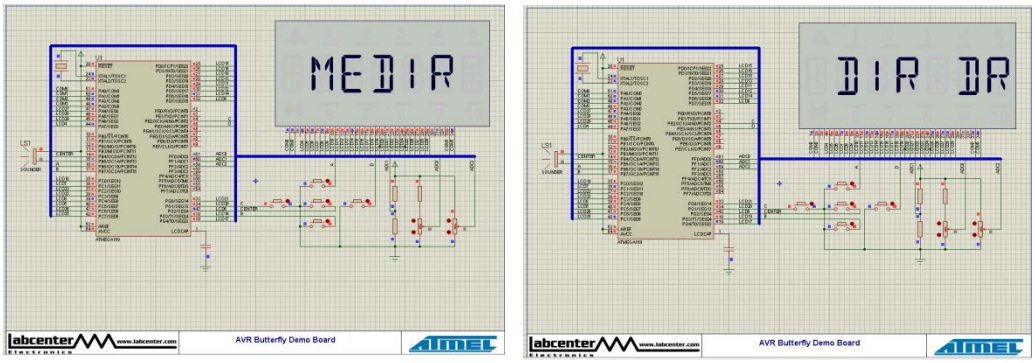


Figura 4.1 Interfaz menú opción 1 en AVR Butterfly

La figura 4.2 muestra la segunda opción de nuestro menú en el AVR Butterfly en este caso es “MEMORIA IZ” que significa que para acceder a la memoria y ver los datos guardados hay que mover el joystick hacia la izquierda.

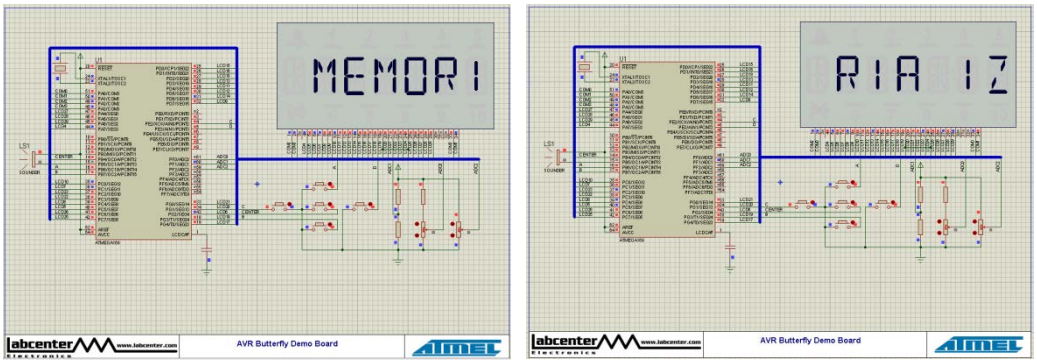


Figura 4.2 Interfaz menú opción 2 en AVR Butterfly

4.3SIMULACIÓN EJERCICIO 2

La temperatura medida con el sensor LM35 es mostrada en la pantalla si se encuentra en el rango de 20 a 40 grados centígrados como se muestra en la figura 4.3.

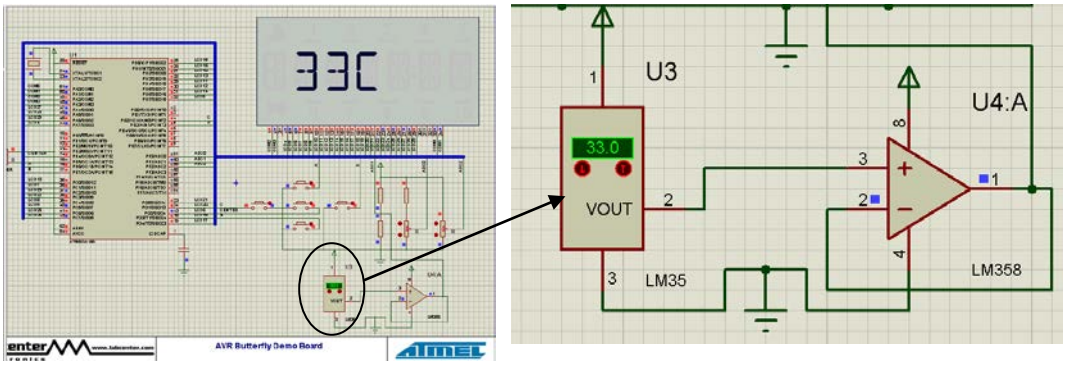


Figura 4.3 Temperatura normal mostrada en AVR Butterfly

Si la temperatura leída por el LM35 es mayor a 40 grados centígrados se mostrara por pantalla el mensaje HOT como se muestra en la figura 4.4.

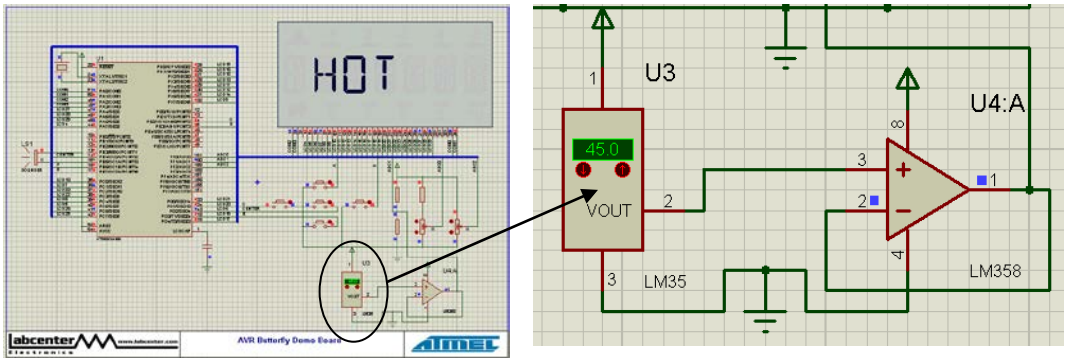


Figura 4.4 Temperatura alta mostrada en AVR Butterfly

Si la temperatura leída por el LM35 es menor a 20 grados centígrados se mostrara por pantalla el mensaje COLD como se muestra en la figura 4.5.

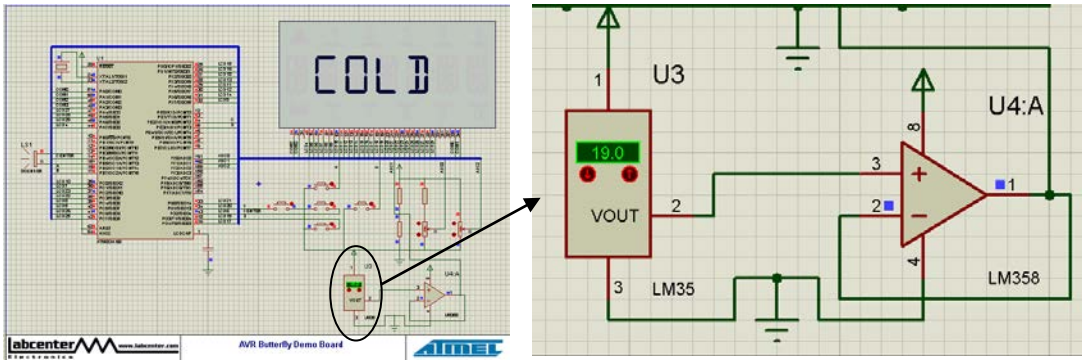


Figura 4.5 Temperatura baja mostrada en AVR Butterfly

4.4 SIMULACIÓN EJERCICIO 3

Este ejercicio muestra como el AVR transmite un dato, en este caso el caracter 'A' por comunicación serial cada vez que la temperatura supera los 20 grados centígrados como se observa en la figura 4.6.

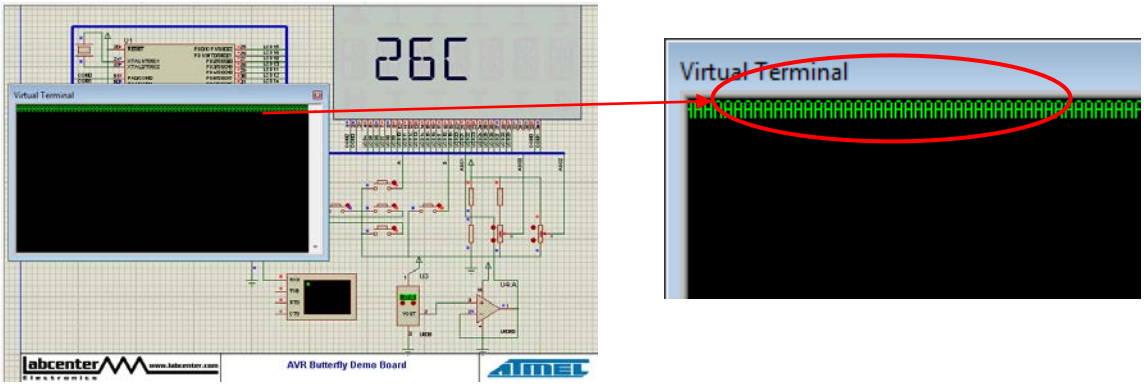


Figura 4.6 Transmisión UART AVR Butterfly

4.5 SIMULACIÓN EJERCICIO 4

Este ejercicio demuestra cómo se accede a los datos guardados en memoria, al presionar el joystick hacia la izquierda se observa en el menú datos guardados como se muestra en las figuras 4.7 y 4.8, si se presiona nuevamente hacia la izquierda se muestran las temperaturas almacenadas una cada vez que se presiona como podemos observar en la figura 4.9, en este caso se almacenan las ultimas 15 medidas.

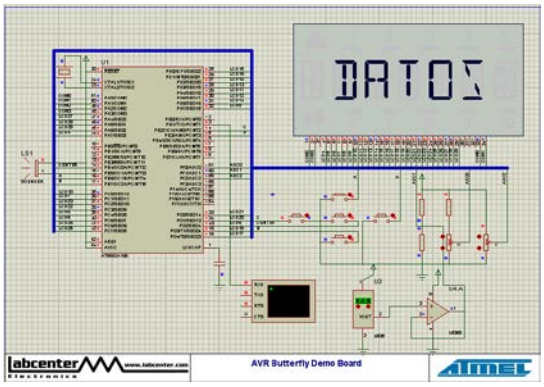


Figura 4.7 Submenú AVR Butterfly

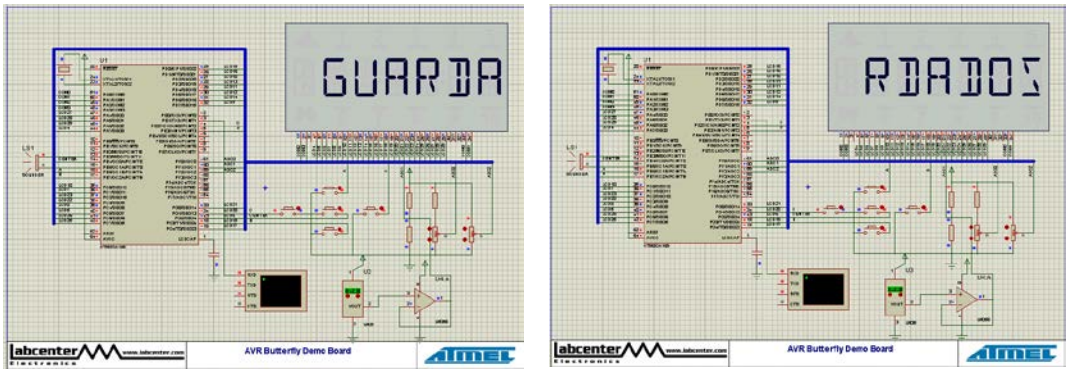


Figura 4.8 Submenú AVR Butterfly

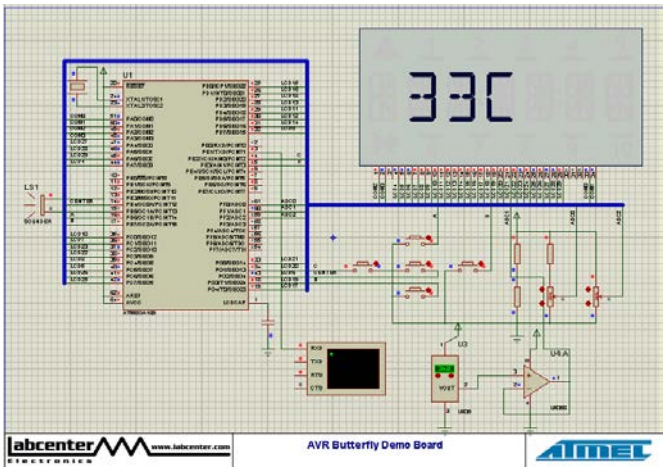


Figura 4.9Temperatura guardada en AVR Butterfly

4.6 SIMULACIÓN EJERCICIO 5

La figura 4.10 muestra un ejemplo básico en el manejo de la tarjeta LPCXpresso, si recibe algún dato por comunicación serial se enciende un LED.

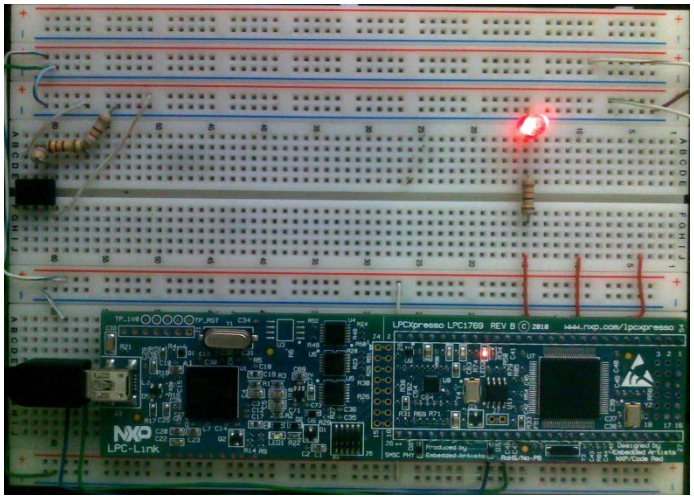


Figura 4.10Manejo LPCXpresso

4.7SIMULACIÓN PROYECTO

La figura 4.11 muestra el funcionamiento de todo el proyecto, el AVR Butterfly muestra datos y envía a la tarjeta LPCXpresso que analizando los datos recibidos enciende un led, el color depende en qué estado de funcionamiento se encuentre.

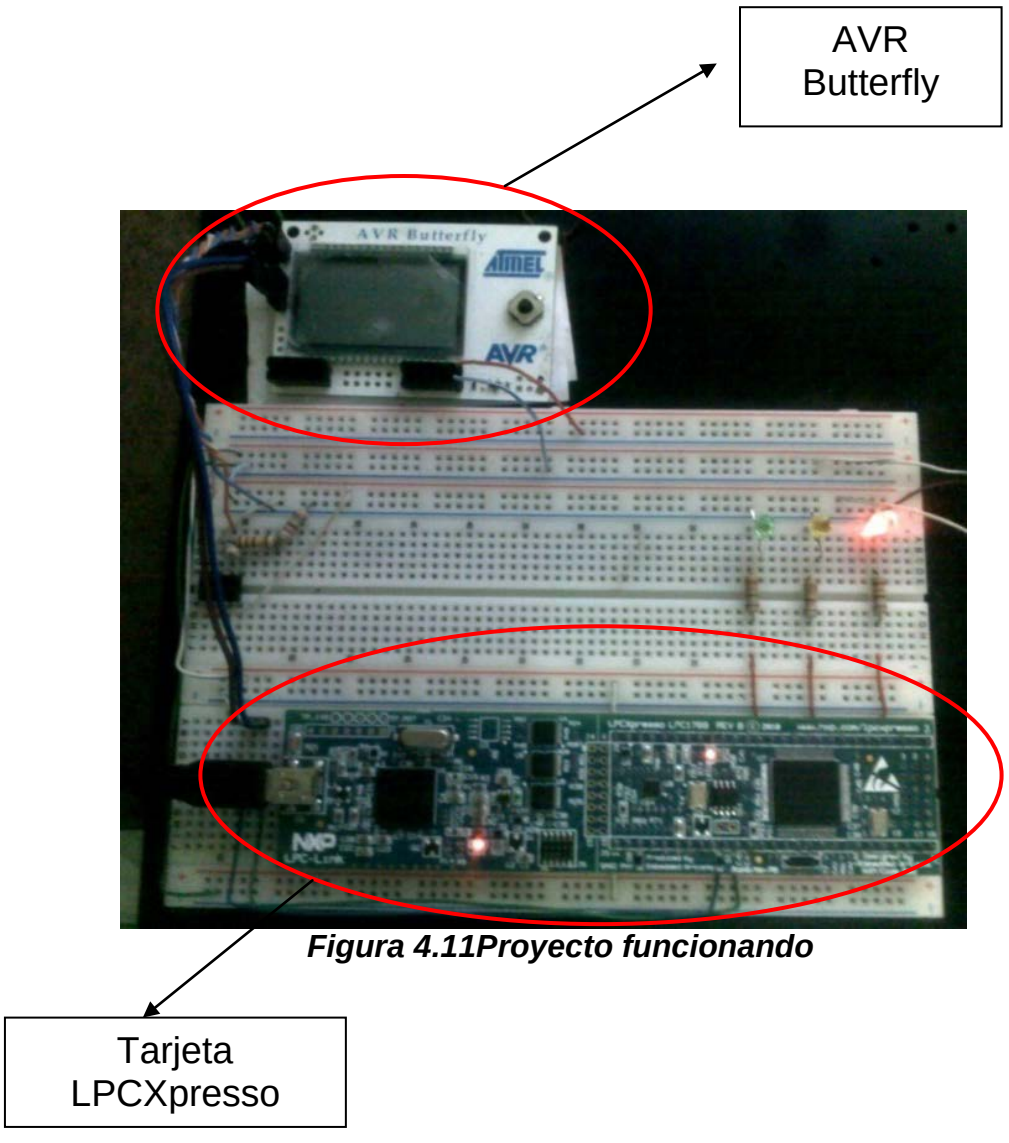


Figura 4.11Proyecto funcionando

CONCLUSIONES

1. En base al proyecto desarrollado podemos concluir que esta tesina es la integración de diferentes tecnologías para el aprovechamiento de recursos y lograr un mismo fin ya que en este proyecto usamos 2 tipos de Microcontroladores diferentes, uno de 32 bits y el otro de 8 bits de diferentes marcas, Atmel y LPCxpresso, creados para diferentes aplicaciones sin embargo con la capacidad de trabajar juntos como lo demostramos en este proyecto.

2. Una de las más importantes acotaciones para la integración de diferentes Microcontroladores es la comunicación entre estos ya que sin importar el tipo de microcontrolador la comunicación entre ellas tienen un estándar y este tipo de comunicación para nuestro caso es la llamada serial RS232, es el enlace que nos permite comunicarnos por eso la importancia de conocer bien el

estándar de comunicación a usar para poder configurar ambas partes y realizar una comunicación con éxito.

3. Al medir temperatura es necesario tener en cuenta varios factores como por ejemplo saber que rangos de temperatura se medirá, que tipo de resolución se desea en la medición y qué tipo de sensor se usará ya que dependiendo del tipo de sensor se debe diseñar el esquema de la implementación, en algunos sensores es necesario un circuito adicional de amplificación, compensación e incluso linealización.

4. La lectura de parámetros analógicos mediante el módulo ADC es una herramienta poderosa y muy versátil, tenerla en el AVR butterfly fue muy ventajoso para nuestro proyecto sin embargo es necesario también hacer ajustes a la lectura obtenida para efectos de escalabilidad es decir poner la lectura en el rango numérico que usaremos para procesar la información y para efectos de calibración.

RECOMENDACIONES

1. Es importante entender primero el funcionamiento individual de cada uno de los microcontroladores que involucra este proyecto antes de combinar sus funciones ya que cada uno de ellos posee su propia estructura y lenguaje.

2. Revisar minuciosamente la sintaxis y tener en cuenta que todas las librerías estén incluidas antes de compilar el código.

3. Documentar correctamente el código para un mejor entendimiento del mismo.

4. Es importante mencionar que en la configuración de la comunicación UART, siempre se tiene que asegurar que tengan la misma velocidad de transferencia tanto en el AvrButterfly como en la tarjeta LPCXpresso, ya que si no se tiene esto no existirá comunicación entre ambos dispositivos.

BIBLIOGRAFÍA

- [1] NXP Semiconductors. LPCXpresso Microcontrollers. Disponible en: http://ics.nxp.com/lpcxpresso/~LPC1769/. (Consultado el19 de Marzo del 2012).

- [2]NXP Semiconductors. LPCXpresso Getting Started. Disponible en: http://ics.nxp.com/support/documents/microcontrollers/pdf/lpcxpresso.getting.started.pdf (Consultado el 20 de Marzo del 2012).

- [3] ATMEL. Avr Butterfly. Disponible en: http://www.atmel.com/Images/doc4271.pdf (Consultado el 22 de Marzo del 2012).

- [4] Pardue, Joe. Quick Start Guide. Disponible en: http://www.smileymicros.com/QuickStartGuide.pdf

- [5]ATMEL. AVR Studio 4. Disponible en: http://www.elec.uow.edu.au/avr/guides/Getting-started-C-programming-Atmel-AVR.pdf(Consultado el 22 de Marzo del 2012).

- [6] Hazael, I. (s.f.). Monografias.com. Transductores y Sensores. Disponible en: http://www.monografias.com/trabajos31/transductores-sensores/transductores-sensores.shtml (Consultado el 23 de Febrero del 2012).

- [7] Microchip, AN857. Brushless DC (BLDC) Motor Fundamentals. Disponible en: http://ww1.microchip.com/downloads/en/AppNotes/00885a.pdf. (Consultado el 13 de Febrero del 2012).

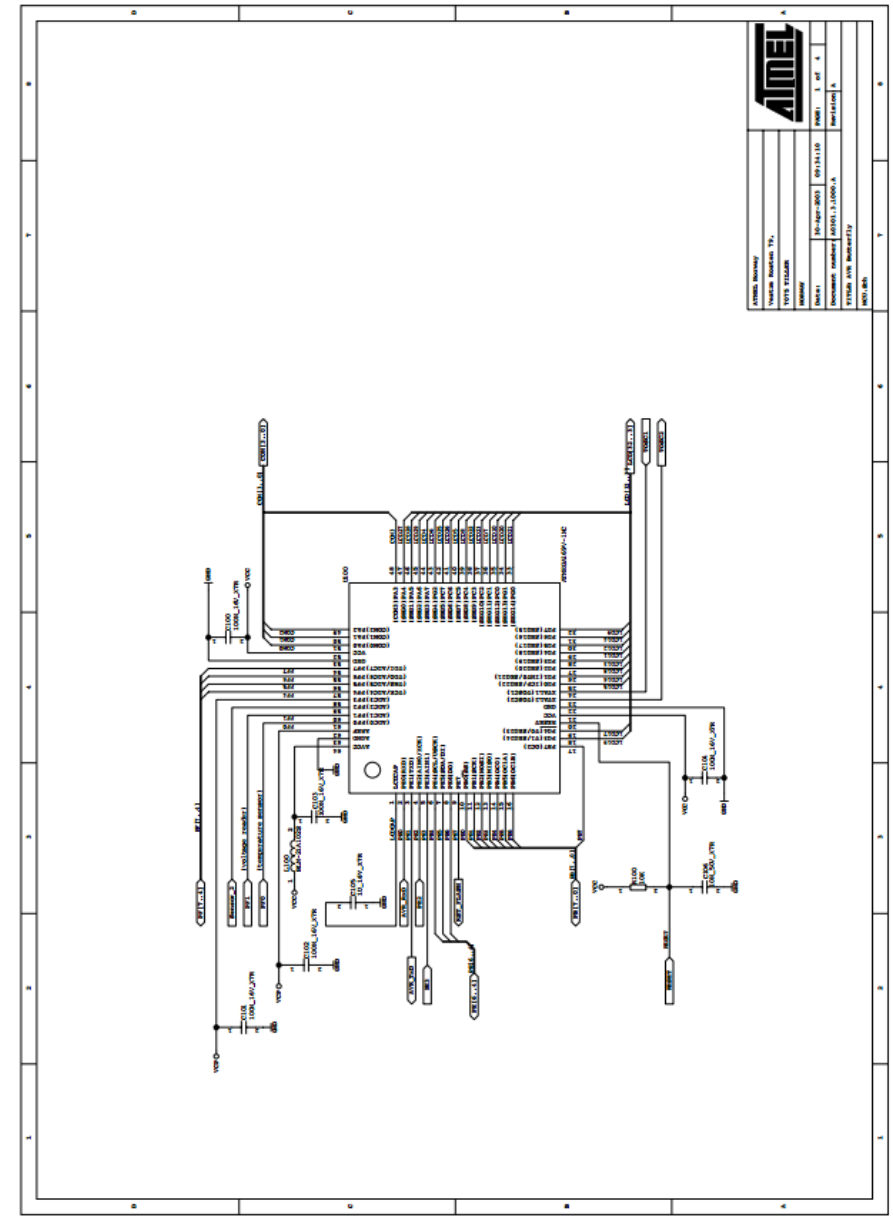
- [8] Burgess, Mark. Tutorial K&R Sobre programación en C http://www.iu.hio.no/~mark/CTutorial/CTutorial.html Abril 15,2012 (Consultado el 19 de Marzo del 2012)

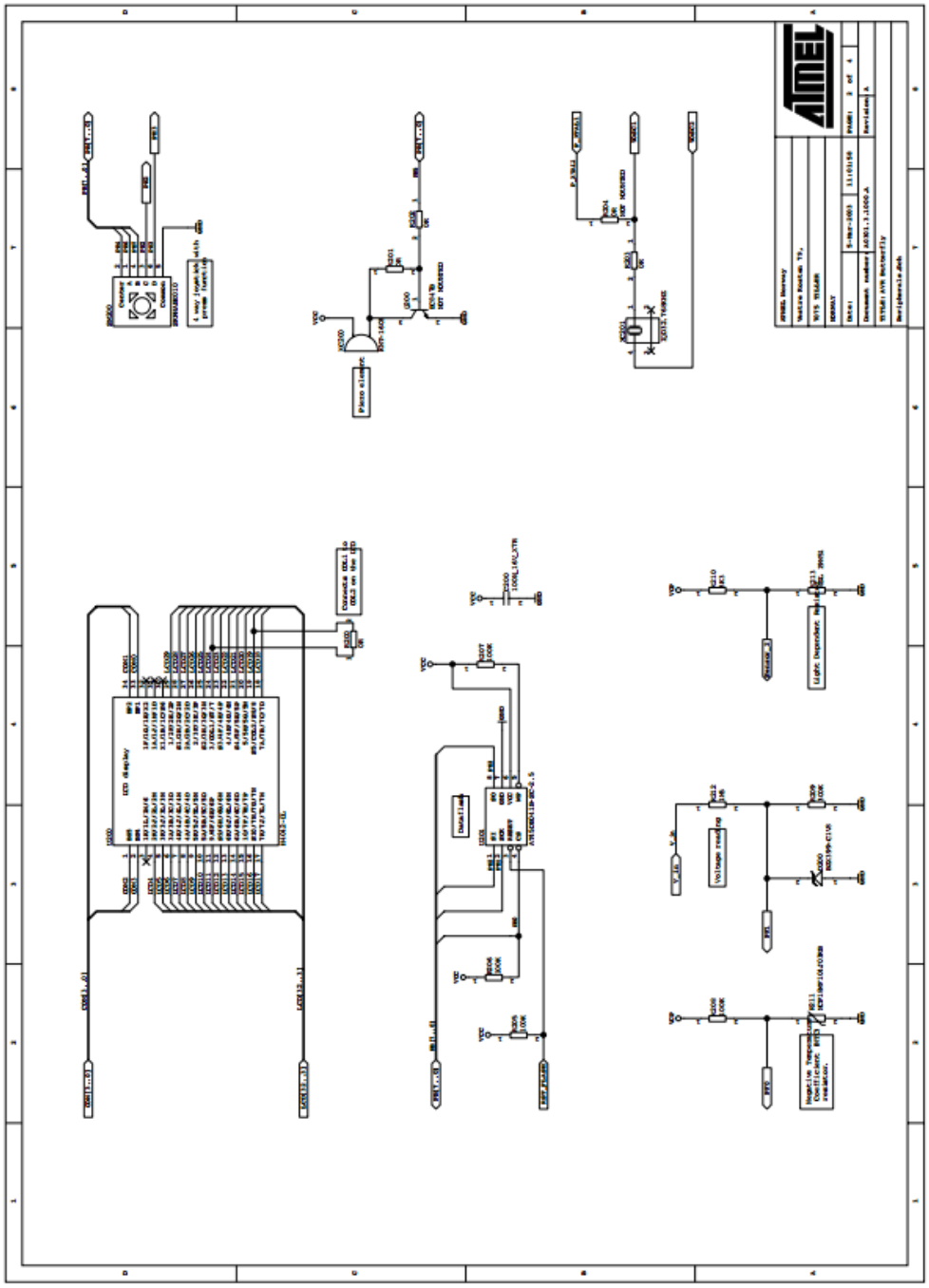
- [9] Microchip Technology Inc.Brushless DC (BLDC) Motor Fundamentals. Disponible en: http://www.ingeniamc.com/Es/-Control-techniques-for-brushless-motors.pdf (Consultado el 19 de Marzo del 2012)

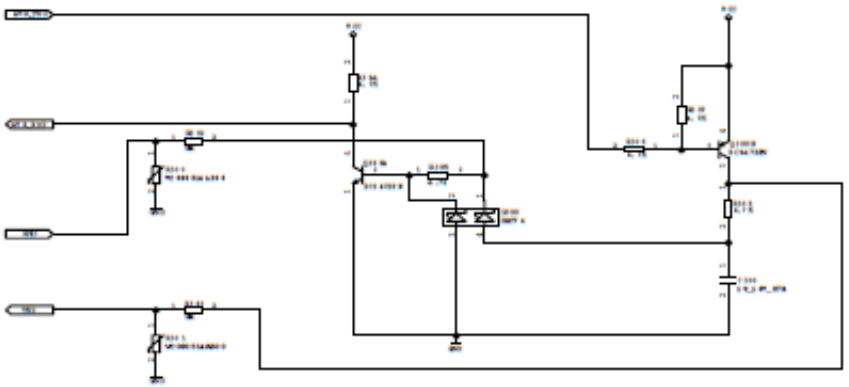
- [10] Universidad de Catarina. Comunicación serial. Disponible en: http://catarina.udlap.mx/u_dl_a/tales/documentos/lem/morales_h_oe/capitulo3.pdf. (Consultado el 22 de Marzo del 2012)

ANEXOS

ESQUEMATICOS AVR BUTTERFLY







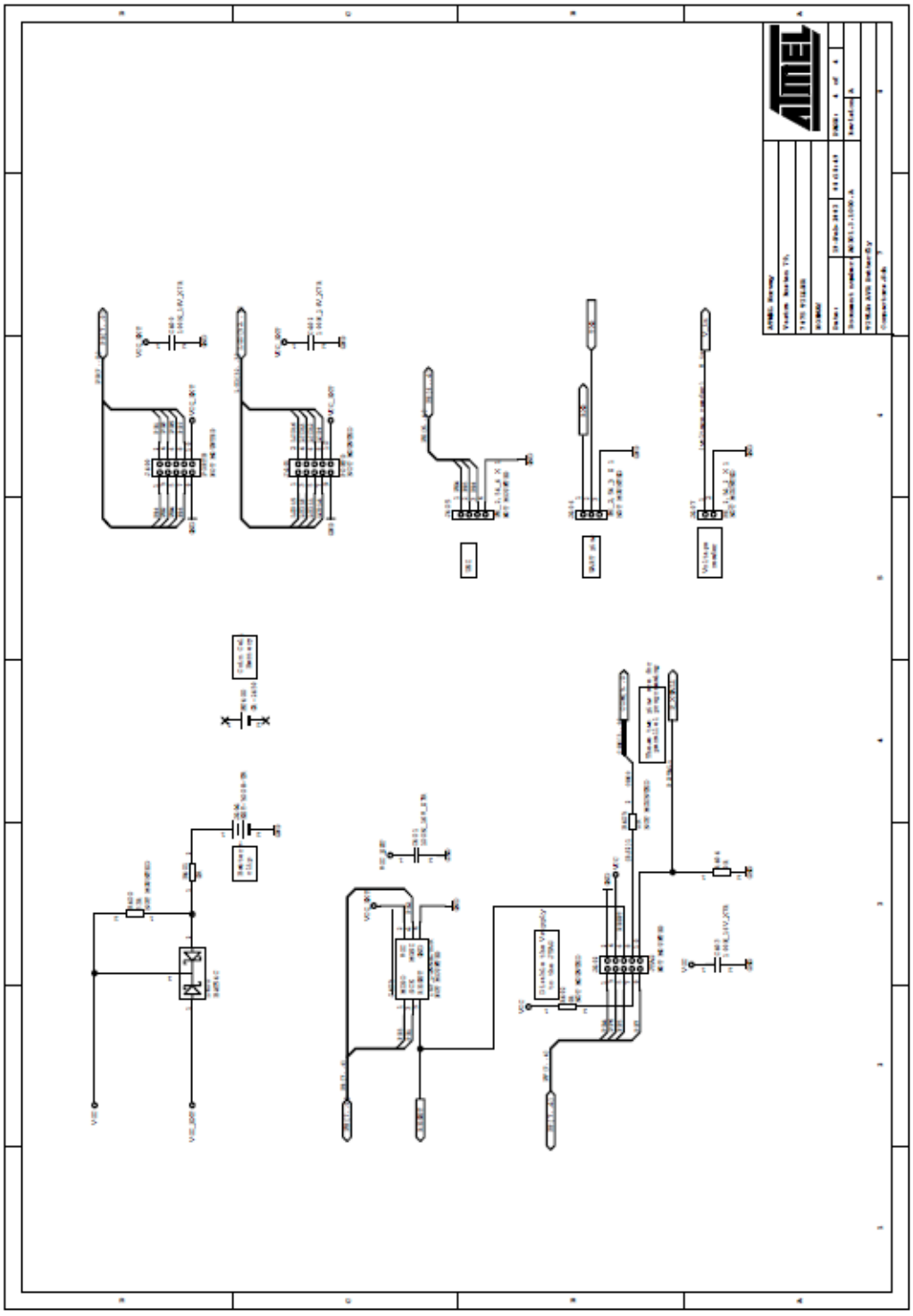
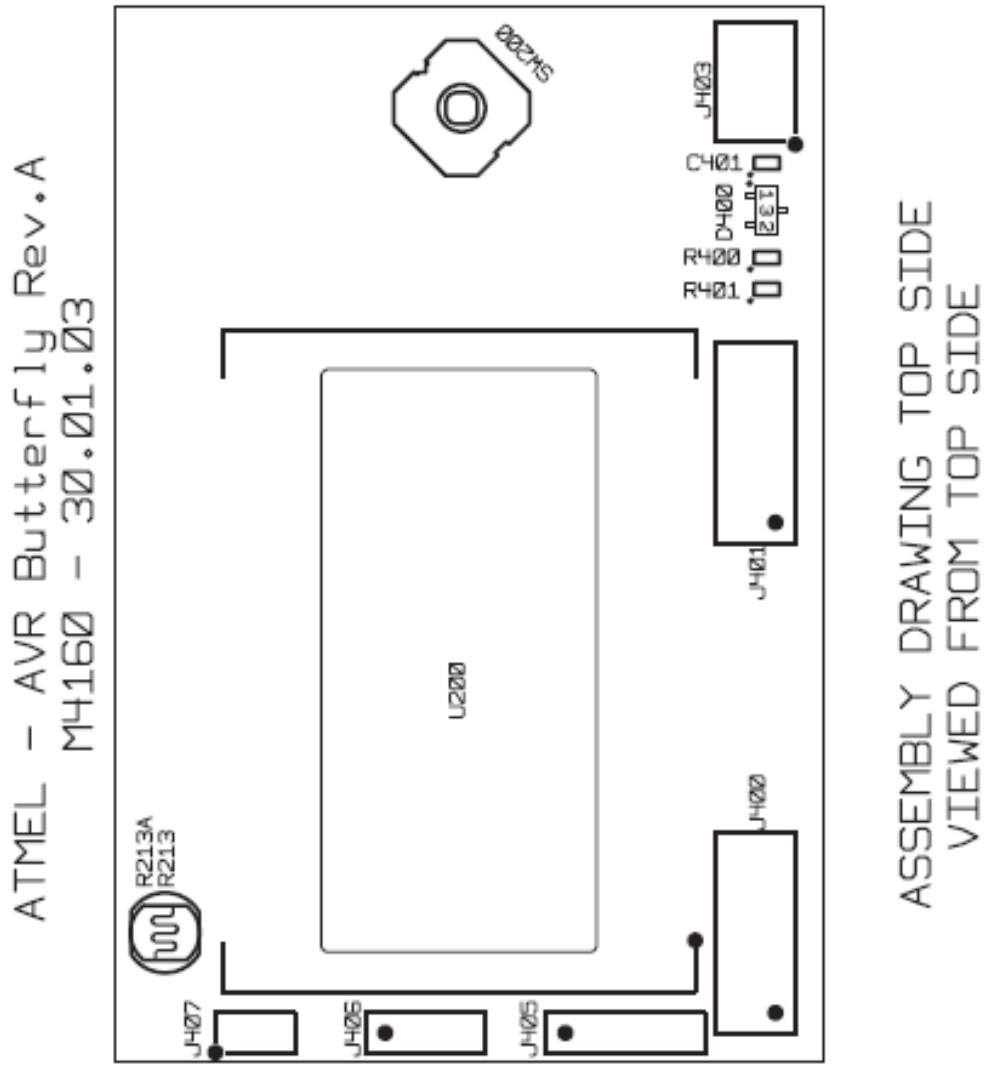


Figure 7-5. Assembly Drawing, Top Side



National Semiconductor

December 1994

LM35/LM35A/LM35C/LM35CA/LM35D Precision Centigrade Temperature Sensors

General Description

The LM35 series are precision integrated-circuit temperature sensors, whose output voltage is linearly proportional to the Celsius (Centigrade) temperature. The LM35 thus has an advantage over linear temperature sensors calibrated in ° Kelvin, as the user is not required to subtract a large constant voltage from its output to obtain convenient Centigrade scaling. The LM35 does not require any external calibration or trimming to provide typical accuracies of ±¼°C at room temperature and ±¾°C over a full −55 to +150°C temperature range. Low cost is assured by trimming and calibration at the wafer level. The LM35's low output impedance, linear output, and precise inherent calibration make interfacing to readout or control circuitry especially easy. It can be used with single power supplies, or with plus and minus supplies. As it draws only 60 µA from its supply, it has very low self-heating, less than 0.1°C in still air. The LM35 is rated to operate over a −55° to +150°C temperature range, while the LM35C is rated for a −40° to +110°C range (−10° with improved accuracy). The LM35 series is available packaged in hermetic TO-46 transistor packages, while the LM35C, LM35CA, and LM35D are also available in the plastic TO-92 transistor package. The LM35D is also available in an 8-lead surface mount small outline package and a plastic TO-202 package.

Features

- Calibrated directly in ° Celsius (Centigrade)
- Linear + 10.0 mV/°C scale factor
- 0.5°C accuracy guaranteeable (at +25°C)
- Rated for full −55° to +150°C range
- Suitable for remote applications
- Low cost due to wafer-level trimming
- Operates from 4 to 30 volts
- Less than 60 µA current drain
- Low self-heating, 0.08°C in still air
- Nonlinearity only ±¼°C typical
- Low impedance output, 0.1 Ω for 1 mA load

Connection Diagrams

TO-46
Metal Can Package*

TL/H/5516–1

*Case is connected to negative pin (GND)

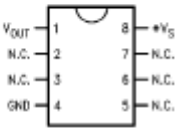
Order Number LM35H, LM35AH, LM35CH, LM35CAH or LM35DH
See NS Package Number H03H

TO-92
Plastic Package

TL/H/5516–2

Order Number LM35CZ, LM35CAZ or LM35DZ
See NS Package Number Z03A

SO-8
Small Outline Molded Package

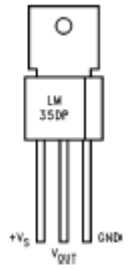


TL/H/5516–21

Top View
N.C. = No Connection

Order Number LM35DM
See NS Package Number M08A

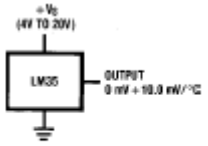
TO-202
Plastic Package



TL/H/5516–24

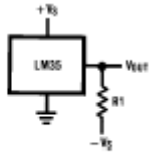
Order Number LM35DP
See NS Package Number P03A

Typical Applications



TL/H/5516–3

FIGURE 1. Basic Centigrade Temperature Sensor (+2°C to +150°C)



TL/H/5516–4

Choose $R_1 = -V_S/50\ \mu A$

$V_{OUT} = +1,500$ mV at +150°C
$= +250$ mV at +25°C
$= -550$ mV at −55°C

FIGURE 2. Full-Range Centigrade Temperature Sensor

TRI-STATE® is a registered trademark of National Semiconductor Corporation.

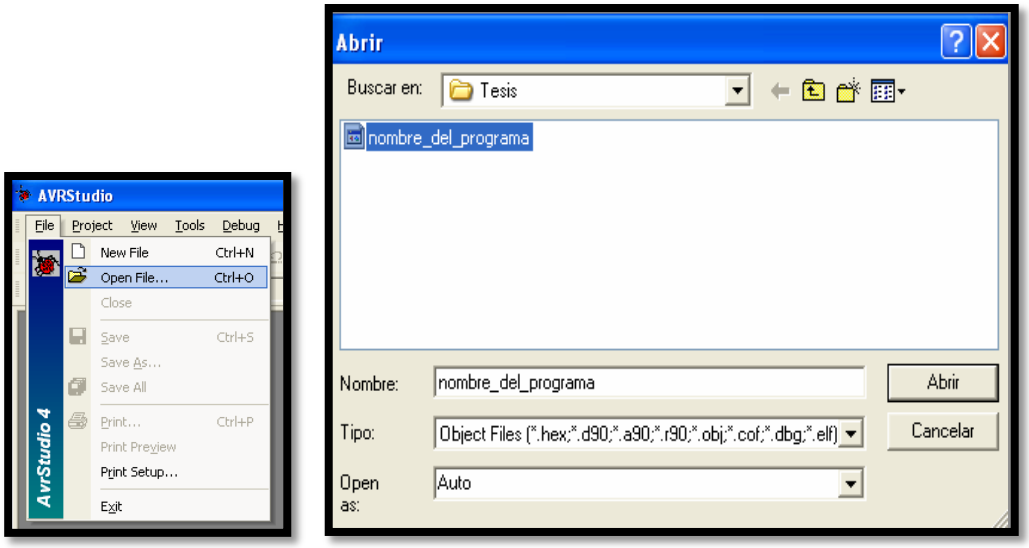
RRD-B30M75/Printed in U.S.A.

PROGRAMACIÓN DEL AVR BUTTERFLY

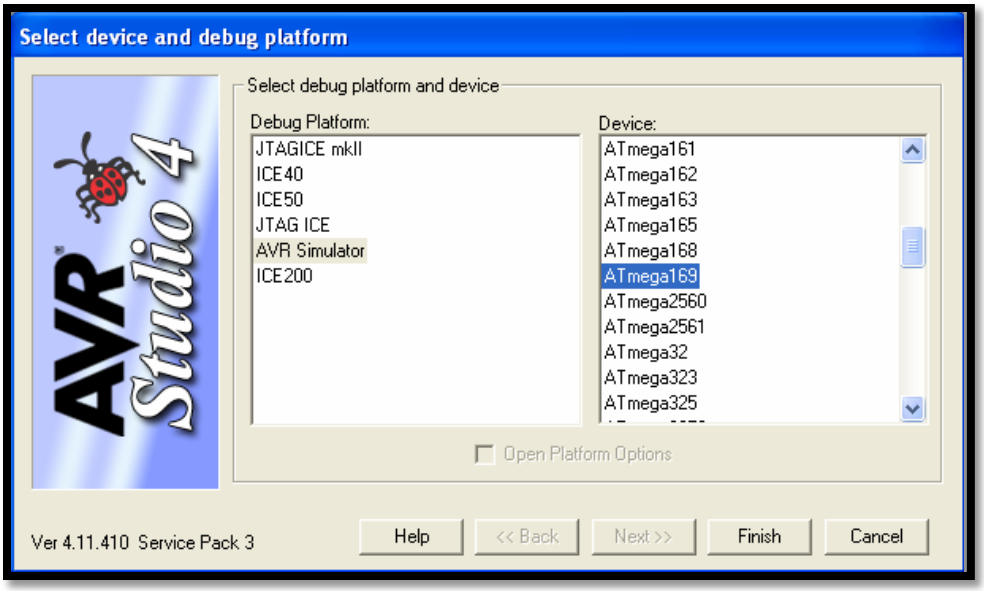
Los pasos necesarios para la Programación del Kit AVR Butterfly son los siguientes:

En la PC, localizar y ejecutar el AVR Studio.

En el menú “File” del AVR Studio seleccionar “Open File“, luego seleccionar el archivo con el que se desea programar al AVR Butterfly, tal como indica la Figura; por ejemplo: …\nombre_del_programa.cof.



Seleccionar el AVR Simulator y luego el ATmega169 como en la Figura A.4



- Presionar “Finish”.
- Conectar el cable serial entre la PC y el AVR Butterfly como se indica en la Figura A.2.
- Resetear el AVR Butterfly cortocircuitando los pines 5 y 6 en el conector J403, conector ISP, o quitando y aplicando nuevamente la fuente de alimentación.
- Luego de un reset, el microcontrolador ATmega169 comenzará desde la sección de Arranque. Nada se desplegará en el LCD mientras esté en la sección de Arranque. Entonces se deberá presionar ENTRAR en el joystick y mantener esa posición; mientras tanto desde la PC en el AVR Studio, iniciar el AVR Prog.

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

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