



ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería Eléctrica y Computación

Estudio detallado de la conversión analógica digital empleada en los
microcontroladores de Atmel con la implementación
en hardware de ejercicios prácticos

TESINA DE SEMINARIO

Previo a la obtención del Título de:

INGENIERO EN ELECTRICIDAD ESPECIALIZACIÓN ELECTRÓNICA Y AUTOMATIZACIÓN INDUSTRIAL

Presentado por:

Ángel Danilo Riera Márquez

Miguel Paúl Rodríguez Guzmán

Grace Alexandra Yagual Machuca

GUAYAQUIL – ECUADOR

AÑO: 2012

AGRADECIMIENTO

A Dios por llenarnos de sus bendiciones durante nuestra carrera profesional, a nuestros familiares quienes nos han incentivado a seguir adelante en este largo camino, y a nuestro profesor, el Ing. Carlos Valdiviezo, por brindarnos sus conocimientos y motivación, así como a nuestros compañeros del seminario, con quienes hemos compartido esta emocionante experiencia de aprendizaje e investigación.

También estamos agradecidos sinceramente con todas las personas que han aportado ayudándonos a cumplir nuestra meta: amigos, profesores, compañeros y autoridades de este prestigioso establecimiento educativo, ya que gracias a ellos hemos obtenido las bases necesarias para continuar con nuestra vida profesional y laboral.

DEDICATORIA

De todo corazón este trabajo se lo dedico a Dios por permitirme alcanzar ésta primera meta, a mi mamá Nereyda Yagual y a mi abuelita Grace Machuca por sus sabios consejos, además de brindarme su total apoyo y confianza durante toda mi vida.

Grace Yagual

Agradezco a Dios, mi familia y amigos por sus bendiciones y apoyo incondicional durante toda mi vida; gracias a ellos y al compromiso que yo he dado estoy satisfecho por los resultados que tengo aspirando siempre a algo mejor.

Miguel Rodríguez

Agradezco a Dios por bendecirme siempre, a mis padres y mi amor Isabel por apoyarme siempre, a mis compañeros y amigos por estar ahí conmigo. Este triunfo va dedicado a todos los que creyeron en mí y que saben que nunca me rendiré, gracias.

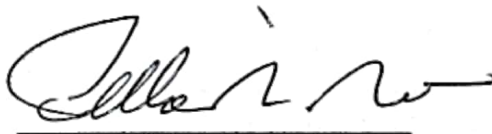
Danilo Riera

TRIBUNAL DE SUSTENTACIÓN



Ing. Carlos Valdivieso A.

PROF. DEL SEMINARIO DE GRADUACIÓN



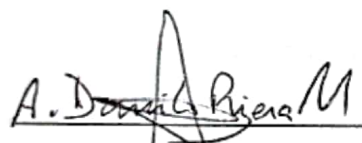
Ing. Hugo Villavicencio V.

DELEGADO DEL DECANO

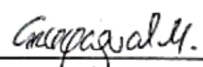
DECLARACIÓN EXPRESA

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

(Reglamento de Graduación de la ESPOL)


Danilo Riera Márquez


Miguel Rodríguez Guzmán


Grace Yagual Machuca

RESUMEN

En el presente proyecto hablamos sobre el Convertidor Analógico Digital (ADC), importancia, funcionamiento y comparación en diferentes microcontroladores para escoger el adecuado al momento de realizar un proyecto, así como también del software y hardware que se uso durante el desarrollo del proyecto en sí, que consta de cinco ejercicios, tres de ellos realizados en lenguaje de programación C y los dos restantes en lenguaje de programación Assembler.

Ejercicios que mediante cierta regulación en la entrada del ADC del microcontrolador nosotros obtenemos una salida ya sea visualizada en leds o en la pantalla del AVR Butterfly, un dispositivo que nos ayuda en el momento de la implementación por las múltiples funciones que contiene.

En cada uno de los ejercicios, tenemos de cada uno de ellos su teoría respectiva, su implementación en software y hardware, su documentación y su toma de fotografía para que quede registrado cada uno de ellos.

ÍNDICE GENERAL

RESUMEN.....	VI
INTRODUCCIÓN.....	XIV
CAPITULO I. CONVERTIDOR ANALÓGICO DIGITAL (ADC).....	1
1.0. GENERALIDADES.....	1
1.1. IMPORTANCIA DEL ADC.....	1
1.2. FUNCIONAMIENTO DEL ADC.....	3
1.3. COMPARACION ENTRE LOS MODULOS ADC DE LOS MICROCONTROLADORES.....	4
CAPITULO II. FUNDAMENTO TEÓRICO.....	13
2.1. Descripción del Software.....	13
2.1.1 AVR STUDIO 4.....	13
2.1.1.1 VENTANAS PRINCIPALES DEL AVR STUDIO.....	15
2.1.1.2 La Ventana Workspace (Área de Trabajo).....	17
2.1.1.3 La ventana Project.....	17
2.1.1.4 La ventana I/O.....	17
2.1.1.5 La ventana Info.....	18
2.1.1.6 La Ventana Memory.....	20
2.1.1.7 La Barra de Estado.....	34
2.1.2 PROTEUS 7.7.....	21
2.1.3 AVR BUTTERFLY.....	22
CAPITULO III. IMPLEMENTACIÓN DEL TRABAJO.....	23
3.0. GENERALIDADES.....	23

CAPITULO IV. DESARROLLO GENERAL USO DEL ADC.....	26
4.1 MEDIDOR DE TEMPERATURA.....	26
4.2 USO DEL SENSOR DE LUZ Y EL SPEAKER.....	42
4.3 ADC - DEMOSTRACION EN LEDS.....	51
4.4 LECTURA DEL VOLTAJE.....	58
4.5 MEDIDOR DE LUZ.....	66
CONCLUSIONES.....	82
RECOMENDACIONES.....	83
BIBLIOGRAFÍA.....	85

INDICE DE TABLAS

Tabla

Pág.

Tabla 1.1. Modulo ADC justificación izquierda.....	5
Tabla 1.2. Modulo ADC justificación derecha.....	5
Tabla 1.3. Tabla de canales del ADC en Attiny2313.....	6
Tabla 1.4. Tabla de canales del ADC en Atmega169.....	7

INDICE DE FIGURAS

Figura

Pág.

Figura 1.1. Formato del resultado Selección de los bits.....	5
Figura 1.2. Diagrama de bloques de un convertidor A/D de aproximaciones sucesivas típico como un circuito integrado.....	8
Figura 1.3. Diagrama del microprocesador con el ADC0804.....	9
Figura 1.4. Entradas analógicas del ADC0804.....	10
Figura 1.5. Señales de CS y WR.....	11
Figura 1.6. Señales de CS y WR.....	11
Figura 1.7. Señales de la Salida y del Reset de INTR.....	12
Figura 2.1. Compilador AVR Studio 4.....	14
Figura 2.2. Código de inicialización del ADC.....	15

Figura 2.3. Ventana Output.....	16
Figura 2.4. Ventana Project.....	17
Figura 2.5. Ventana info.....	18
Figura 2.6. Ventana Watch.....	19
Figura 2.7. Ventana Memory.....	20
Figura 2.8. Barra de estado.....	20
Figura 2.9. Demo del AVR Butterfly en Proteus.....	21
Figura 2.10. AVR Butterfly vista frontal.....	22
Figura 2.11. AVR Butterfly vista trasera.....	22
Figura 2.12. Canal ADC1del AVR Butterfly.....	22
Figura 3.1.Espadines de conexión.....	23
Figura 3.2. AVR Butterfly.....	24
Figura 3.3. Conector Serial BD9.....	24
Figura 3.4.Butterfly midiendo temperatura.....	24

Figura 3.5. Proyecto en Protoboard.....	25
Figura 3.6. Port B y Port D del Butterfly.....	25
Figura 3.7. Batería de alimentación.....	25
Figura 4.1. Sensor de temperatura integrado al AVR Butterfly.....	26
Figura 4.2. Diagrama de bloques de medición de temperatura.....	27
Figura 4.3. Diagrama de flujo de medición de temperatura.....	28
Figura 4.4. Simulación en Proteus de medición de temperatura.....	41
Figura 4.5. Sensor fotoeléctrico LDR.....	42
Figura 4.6. Diagrama de bloques LDR con ADC.....	43
Figura 4.7. Diagrama de flujo del LDR con ADC.....	44
Figura 4.8. Diagrama de flujo de subrutina de pausa.....	45
Figura 4.9. Simulación en Proteus LDR con ADC.....	50
Figura 4.10. Diagrama de bloques del ADC con LEDS.....	51
Figura 4.11. Diagrama de flujo del ADC con LEDS.....	52

Figura 4.12. Simulación en Proteus del ADC con LEDS.....	57
Figura 4.13. Diagrama de bloques de la lectura del voltaje.....	58
Figura 4.14. Diagrama de flujo de la lectura del voltaje.....	59
Figura 4.15. Simulación en Proteus de la medición de voltaje.....	65
Figura 4.16. Diagrama de bloques de la medición de luz.....	66
Figura 4.17. Diagrama de flujo de la medición de luz.....	67
Figura 4.18. Simulación en Proteus de medición de luz (a).....	81
Figura 4.19. Simulación en Proteus de medición de luz (b).....	81

INTRODUCCIÓN

Desde la invención de los semiconductores, el desarrollo de la tecnología digital ha dado lugar a dispositivos cada vez más complejos y rápidos. Entre ellos los microprocesadores y los microcontroladores. Estos dispositivos se encuentran en nuestro trabajo, en nuestra casa, etc. Controlan el funcionamiento de los teclados de las computadoras, están en los teléfonos celulares, en general, en todo aparato electrónico que posea un grado de automatismo.

Se dice que un controlador es un dispositivo que se emplea para manejar uno o varios procesos. Por ejemplo, para ver televisión, un controlador evalúa la señal que ingresa por la antena y la procesa para que a la pantalla y el parlante llegue con el mismo nivel promedio, sin importar el nivel de la señal ingresante, siempre que esté dentro de determinados parámetros. Hasta hace unos 35 años, los controladores se construían con componentes electrónicos de lógica discreta. Desde comienzos de los 90 todos los elementos del controlador se han podido incluir en un solo circuito integrado, el cual recibe el nombre de microcontrolador. Es decir, un microcontrolador es un chip que posee en su interior a un microprocesador, memoria de programa, memoria de datos y puertos para comunicarse con el exterior. Como es muy frecuente el trabajo con señales analógicas, éstas deben ser convertidas a digital y por ello muchos microcontroladores incorporan un conversor analógico-digital, el cual se utiliza para tomar datos de varias entradas diferentes que se seleccionan mediante un multiplexor. Las resoluciones más frecuentes son 8 y 10 bits, que son suficientes para aplicaciones sencillas. Para aplicaciones en control e instrumentación están disponibles resoluciones de 12bit, 16bit y 24bit.

C A P Í T U L O 1

CONVERTIDOR ANALOGICO DIGITAL (ADC)

1.0 GENERALIDADES

En este capítulo hablaremos básicamente sobre el Convertidor Analógico Digital de los Microcontroladores Atmel, Intel y Pic; la importancia, características, funcionamientos, registros, y detalles propios de cada uno de los microcontroladores mencionados anteriormente.

Hablaremos desde cuales son los pasos a tener en cuenta al seleccionar un microcontrolador, todas las ventajas y desventajas en un micro hasta poder tomar por cuenta propia la decisión de cual elegir de acuerdo a nuestras conveniencias de proyecto y de dinero.

Tenemos detalles de cada uno de los tres microcontroladores, comprenderemos su funcionamiento y como paso a paso realizan la conversión analógica digital.

1.1 IMPORTANCIA DEL ADC

Los Convertidores Analógicos Digitales permiten una comunicación eficaz entre los sistemas analógicos y los sistemas digitales, tomando muestras del mundo real para generar datos que puedan ser manipulados por un microcontrolador por ejemplo, logrando así convertir cualquier tipo de señal física en tensiones eléctricas cuyos datos podrán ser procesados por el dispositivo electrónico.

En el mundo real existen muchas variables físicas, las cuales son de naturaleza analógica y pueden tomar cualquier valor dentro de un rango continuo. Las señales

analógicas más comunes son las de temperatura, presión, intensidad luminosa, señales de audio, velocidad entre otras.

Citando un ejemplo tenemos la salida de voltaje de un amplificador de audio hacia los altavoces. Este voltaje es una cantidad analógica porque cada uno de sus posibles valores produce una respuesta diferente en el altavoz, y por lo tanto su valor exacto sí es significativo.

Una cantidad digital tiene un valor que se especifica por una de las posibilidades como un 0 o 1, (bajo o alto). Es decir un rango de valores pequeños serán reflejados por un 0, y un rango de valores altos serán iguales a un 1 lógico.

Pero los valores exactos de los voltajes no son significativos, ya que los circuitos digitales responden de la misma manera para todos los voltajes que se encuentran dentro de un rango dado.

Por otra parte, los circuitos analógicos procesan las variables físicas no muy rápidamente, ya que manejan 10 posibles valores y los circuitos digitales procesan variables físicas o señales más rápidamente ya que tienen únicamente dos posibles valores, 0 y 1.

Es por ello que se crearon los Convertidores Analógicos Digitales, para poder aumentar la velocidad del procesamiento de las señales. Por lo tanto el ADC sirve para acoplar los sistemas analógicos con los sistemas digitales, es decir para que exista compatibilidad.

1.2 FUNCIONAMIENTO DEL ADC

El ADC convierte un voltaje analógico de entrada en un valor digital de 10 bits a través de aproximaciones sucesivas. El valor mínimo representa a GND y el valor máximo representa al voltaje en el pin AREF menos 1 LSB. Opcionalmente, AVCC o un voltaje de referencia interna de 1.1V puede conectarse al pin AREF escribiendo en los bits REFSn en el Registro ADMUX. La referencia de voltaje interna puede ser desacoplada por un condensador externo en el pin AREF para mejorar la inmunidad al ruido.

El ADC genera un resultado de 10 bits que se presenta en los Registros de Datos del ADC (ADC Data Registers), ADCH y ADCL. Por defecto, el resultado se presenta ajustado hacia la derecha, pero opcionalmente puede presentarse ajustado hacia la izquierda configurando el bit ADLAR en el ADMUX.

Si el resultado está ajustado hacia la izquierda y no se requiere más que 8 bits, es suficiente leer el ADCH. De otro modo, ADCL debe leerse primero, luego ADCH, para asegurarse de que el contenido de los Registros de Datos correspondan a la misma conversión. Una vez que el ADCL es leído, se bloquea el acceso del ADC a los Registros de Datos. Esto significa que si se ha leído el ADCL, y una conversión se completa antes de que se lea el ADCH, ni el registro es actualizado ni el resultado de la conversión se pierde. Cuando el ADCH es leído, el acceso del ADC a los Registros ADCH y ADCL se habilita nuevamente.

El ADC tiene su propia interrupción que puede activarse cuando una conversión se ha completado. Cuando se prohíbe el acceso del ADC a los Registros de Datos en medio de la lectura del ADCH y del ADCL, la interrupción se activará aún si el resultado se pierde.

1.3 COMPARACION ENTRE LOS MODULOS ADC DE DIFERENTES TIPOD DE MICROCONTROLADORES

Entre las familias de microcontroladores que conocemos tenemos los Microcotroladores ATMEL, PIC e INTEL

Los PIC son una familia de microcontroladores de 8 bits fabricados por la empresa estadounidense

MICROCHIP, cuentan con un CPU RISC y memoria FLASH para el almacenamiento del Firmware. Por otro lado los AVR son una familia de microcontroladores fabricada por la compañía noruega ATMEL, estos microcontroladores de 8 bits cuentan con una CPU RISC y su memoria de programa viene implementada en FLASH. Ambas familias cuentan con periféricos como Puertos Digitales, ADC, PWM, entre otros. Se podría pensar que los PIC y AVR son iguales, en cierto modo si, desde un punto de vista de estructura general, pero no en: Lenguaje de programación, IDE, interfaces para la programación, reloj interno, voltaje de alimentación, potencia, costo, etc.

Módulos ADC de los MicrocontroladoresAtmel y PICs[ref.- 6]

Ambos módulos ADC poseen 2 registros de datos de 8 bits cada uno, espacio donde guardan el resultado de la conversión. Un registro del bit alto de la conversión y un registro del bit bajo de la conversión. De los 16 bits solo se usan 10.

En AVR: ADCH y ADCL

En PICs: ADRESH y ADRESL

El modulo A/D tiene la flexibilidad de colocar el resultado justificado a la derecha o a la izquierda de esos 16 bits (formato).

El bit que se selecciona el formato:

En AVR es ADLAR:

Con el bit ADLAR=0: justificación izquierda

Bit	15	14	13	12	11	10	9	8	
	—	—	—	—	—	—	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Tabla 1.1. Modulo ADC justificación izquierda

Con el bit ADLAR=1: justificación derecha

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	—	—	—	—	—	—	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Tabla 1.2. Modulo ADC justificación derecha

En PICs es ADFM

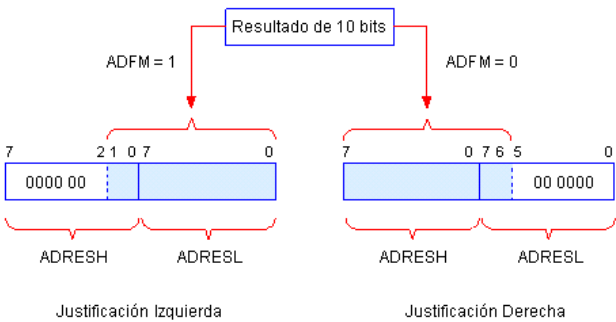


Figura1.1. Formato del resultado Selección de los bits

Para encender el modulo ADC en AVR se setea el bit ADEN mientras que en PICs es el bit ADON

El inicio de la conversión se setea el bit iniciar conversión. En los AVR el bit se llama ADSC y en los PICs es el bit GO/DONE

Ambos micros tienen canales los cuales varían dependiendo de la cantidad de combinaciones que tenga su multiplexer la cual está relacionada al tamaño del puerto que tenga el microcontrolador.

Por ejemplo en AVR:

Attiny2313

MUX1..0	Single Ended Input
00	ADC0 (PB5)
01	ADC1 (PB2)
10	ADC2 (PB4)
11	ADC3 (PB3)

Tabla 1.3. Tabla de canales del ADC en Attiny2313

Atmega169

MUX4..0	Single Ended Input	Positive Differential Input	Negative Differential Input
00000	ADC0	N/A	
00001	ADC1		
00010	ADC2		
00011	ADC3		
00100	ADC4		
00101	ADC5		
00110	ADC6		
00111	ADC7		
01000	N/A		
01001			
01010			
01011			
01100			
01101			
01110			
01111			
10000		ADC0	ADC1
10001		ADC1	ADC1
10010		ADC2	ADC1
10011		ADC3	ADC1
10100		ADC4	ADC1
10101		ADC5	ADC1
10110		ADC6	ADC1
10111		ADC7	ADC1
11000		ADC0	ADC2
11001		ADC1	ADC2
11010		ADC2	ADC2
11011		ADC3	ADC2
11100		ADC4	ADC2
11101		ADC5	ADC2
11110	1.1V (V _{BI2})	N/A	
11111	0V (GND)		

Tabla 1.4. Tabla de canales del ADC

Ambos microcontroladores poseen un bit llamado ADIF el cual se pone en uno lógico cuando se realiza la conversión. La interrupción se la habilita en los AVR

seteando los bits ADIE e I-REG mientras que en los PICs se habilita seteando los bits ADIE, PEIE y GIE. Para el control de los registros los microcontroladores disponen de los siguientes registros:

En AVR el registro de control se llama ADCSR, y si dispone de 2 registros de control pueden llamarse ADCSRA y ADCRSB. En PICs el control de registros se llama ADCON, y si dispone de 2 registros de control pueden llamarse ADCON0 y ADCON1. A diferencia de los PICs, el micro AVR dispone de un registro llamado ADMUX el cual contiene los bits para la selección de canales mientras que en los PICs la selección de canales se encuentra en el registro ADCON0. Ambos registros de control poseen un preescalador que representa la división del reloj para el ADC. En PICs son los bits ADSC2:0 y en AVR son los bits ADPS01:00.

MICROCONTROLADORES INTEL

CONVERTIDOR ANALOGICO/DIGITAL ADC080X

La conversión A/D es un proceso de cuantización en la cual una señal analógica es representada por su equivalente en estados binarios.

El ADC0804 es un ADC de bajo costo, basado en aproximaciones sucesivas, que pertenece a convertidores que son casi iguales excepto en la exactitud. Es ideal para muchas aplicaciones que no requieren un alto grado de exactitud. El AC0804 requiere hasta 100 μ s para convertir un V_{in} analógico a una salida en código digital.

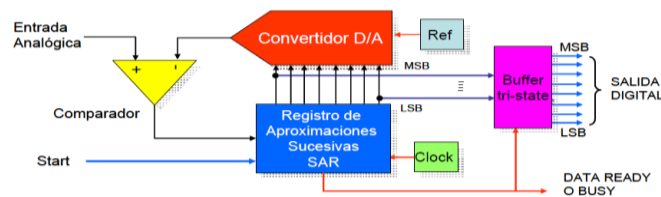


Figura 1.2. Diagrama de bloques de un convertidor A/D de aproximaciones sucesivas típico como un circuito integrado

Un pulso en START inicia el proceso de conversión y deshabilita el buffer tri-state de salida. Al final del periodo de conversión. Se activa la salida DATA READY y la salida digital queda disponible en el buffer de salida.

Para utilizar un ADC con un microprocesador, este debe realizar lo siguiente:

- Enviar un pulso a la terminal START. Esta puede ser derivada de una señal de control tal como la “write” (WR).
- Esperar hasta el final de la conversión. El final del periodo de conversión puede ser verificado ya sea checando el status (polling) o usando interrupciones.
- Leer la señal digital por un puerto de entrada

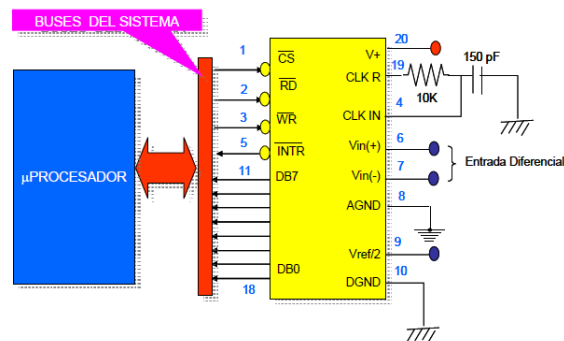


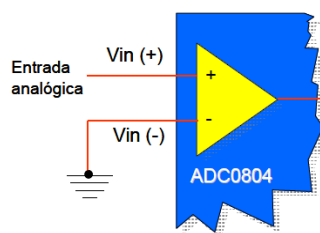
Figura 1.3. Diagrama del microprocesador con el ADC0804

El ADC0804 de National Semiconductor tiene implementadas todas las señales de control necesarias para conectarse a un microprocesador.

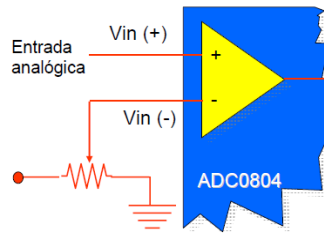
CARACTERISTICAS MÁS IMPORTANTES DEL ADC0804 [ref.- 2]

- Resolución de 8 bits
- Habilidad de conexión directa al bus del microprocesador
- Tiempo de conversión $< 100\mu s$
- Entrada de voltaje diferencial
- Entradas y salidas compatibles con TTL's
- Generador de reloj dentro del chip
- Rango de voltaje de entrada de 0v A 5v (una sola fuente de +5v)
- No requiere ajuste de cero

Entradas analógicas del ADC0804



(a) para detectar una entrada de 0 a +5 volts.



(b) para detectar una entrada con un desvío respecto a tierra

Figura 1.4. Entradas analógicas del ADC0804

GENERADOR DE LA SEÑAL DE RELOJ

El ADC0804 requiere un reloj para funcionar. El reloj puede ser externo, conectado a la terminal CLK IN o puede ser generado por un circuito RC

El rango de frecuencias del reloj permisibles está entre 100KHz y 1460KHz. Para que el tiempo de conversión sea menor es conveniente usar la frecuencia más alta posible.

Si el reloj se genera con un circuito RC, se utilizan las terminales CLK IN y CLK R conectadas con un circuito RC, como se muestra en la figura. La frecuencia del reloj se calcula con:

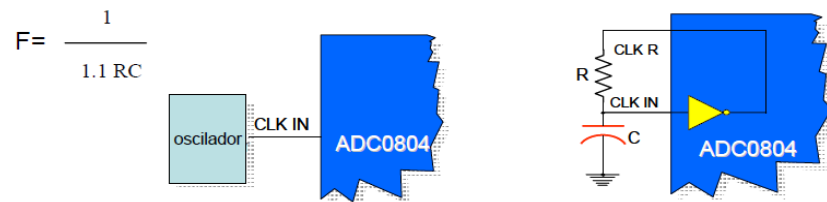


Figura 1.5. Generación de clock del ADC

INICIO DE LA CONVERSION

Una conversión inicia activando las señales CS y WR. Y al final de la conversión, el convertidor genera una señal INTR (similar al DATA READY). Esta señal puede usarse para interrumpir al procesador indicándole que el byte de dato está listo y que ya puede ser leído.

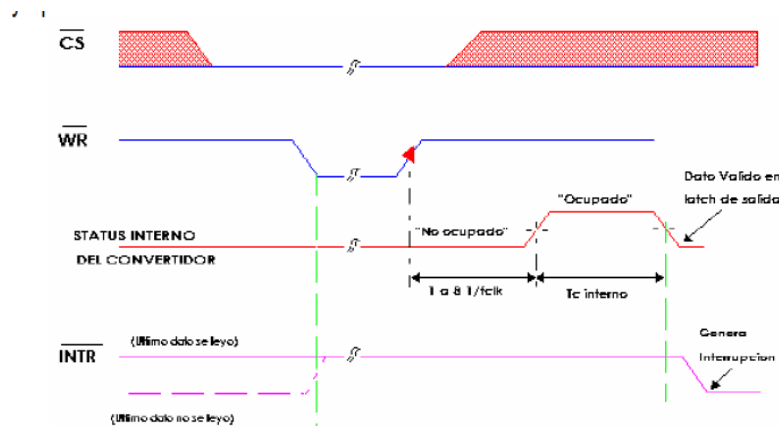


Figura 1.6. Señales de CS y WR

HABILITACION DE LA SALIDA Y RESET DE INTR

El procesador lee el byte activando la señal RD y puede iniciar con la siguiente convertidor si fuera necesaria.

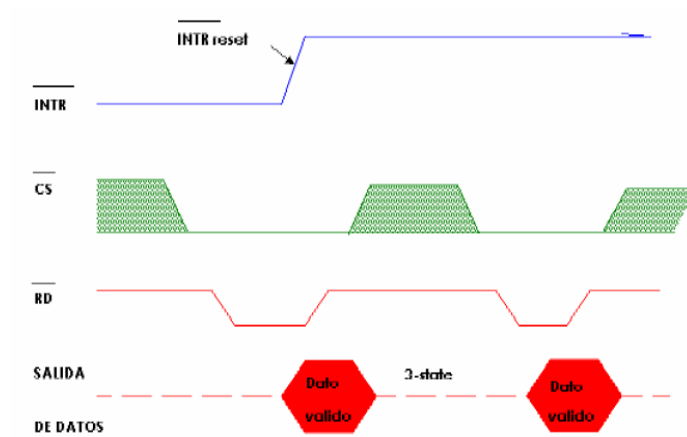


Figura 1.7. Señales de la Salida y del Reset de INTR

CAPÍTULO 2

FUNDAMENTO TEÓRICO

2.1 DESCRIPCION DEL SOFTWARE

Las herramientas necesarias para el desarrollo, simulación, e implementación de los diferentes ejercicios que mostramos en este documento serán descritas en el presente capítulo.

Entre ellas se mostrará el entorno usado para la programación y depuración del microcontrolador ATmega169 que está incluido en el Kit AVR Butterfly; además de la interfaz utilizada para simular los circuitos electrónicos que fueron utilizados para mostrar el funcionamiento de los programas.

2.1.1 AVR STUDIO 4[ref.- 4]

AVR Studio 4 es un entorno maduro de desarrollo integrado (IDE) para el desarrollo de aplicaciones Atmel AVR 8-bits. Este software es actualizado continuamente y está disponible para descargarlo desde www.atmel.com.

AVR Studio apoya al diseñador en el diseño, desarrollo, depuración y parte de la comprobación del proceso.

Provee herramientas de manejo, editor de código fuente, que puede ser en lenguaje ensamblador o lenguaje C. Una de las características principales de este producto tenemos es que se integra con compilador GCC.

El lenguaje ensamblador nos permite manipular con mayor detalle los valores de los registros que posee el microcontrolador que estemos utilizando. De la misma

manera las herramientas que posee nos permiten observar los valores que toman las diferentes variables que hayamos declarado a lo largo del desarrollo del programa. El lenguaje GCC (GNU C Compiler) permite una programación de alto nivel de modo que consigamos desarrollar aplicaciones de mayor grado de desarrollo.

El 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.

Este entorno consiste de varias ventanas y sub-módulos. Cada ventana apoya a las partes del trabajo que se intenta emprender.

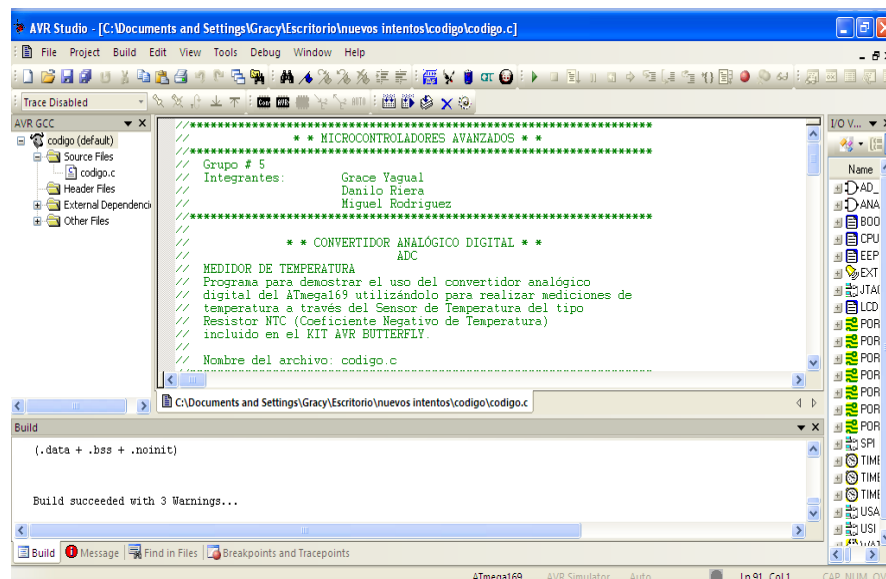
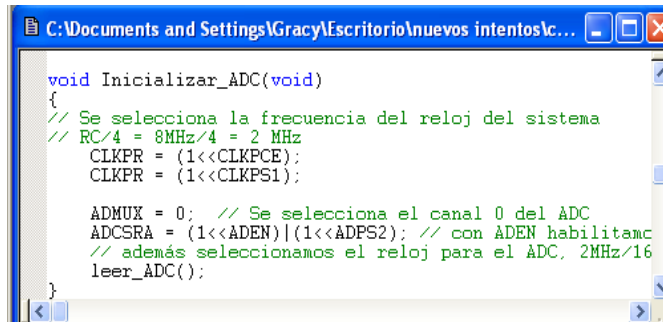


Figura2.1. Compilador AVR Studio 4

Administración de Proyectos.

Toda creación de código dentro del AVR Studio se la realiza como proyectos de programación. Todos los códigos de los proyectos tienen un archivo project que mantiene la información acerca del proyecto. El archivo Project se asegura de que

el proyecto sea el mismo cada vez que regrese a él, que todo esté adecuadamente organizado y que compile.



```
void Inicializar_ADC(void)
{
    // Se selecciona la frecuencia del reloj del sistema
    // RC/4 = 8MHz/4 = 2 MHz
    CLKPR = (1<<CLKPCE);
    CLKPR = (1<<CLKPS1);

    ADMUX = 0; // Se selecciona el canal 0 del ADC
    ADCSRA = (1<<ADEN)|(1<<ADPS2); // con ADEN habilitamc
    // además seleccionamos el reloj para el ADC, 2MHz/16
    leer_ADC();
}
```

Figura 2.2. Código de inicialización del ADC

2.1.1.1 LAS VENTANAS PRINCIPALES DEL IDE AVR STUDIO 4

Ventana Editor

Este es el lugar donde se realiza el trabajo de codificar. El AVR Studio usa el editor Stringray de la corporación Bsquare.

Este es un editor completo para programación con toda la funcionalidad que el usuario empleará, incluso codificación de color para la sintaxis (que puede ser alterada y extendida por el usuario). La ventana del editor también se usa cuando al depurar código.

La Ventana Output (Ventana de Resultados)

Es una colección de varias ventanas (Build, Messages,...) integradas en un marco etiquetado. Con las etiquetas sobre el marco se selecciona la ventana que se desea observar. En la Figura 2.3 podemos observar la ventana de resultados.

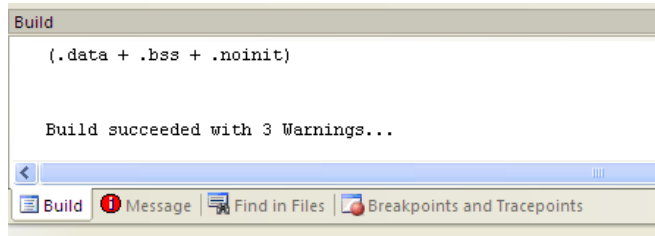


Figura 2.3. Ventana Output

Las opciones que se presentan en esta ventana son las siguientes:

Build. El resultado del compilador se dirige hacia esta ventana. Aquí puede leerse el resultado de la compilación.

Messages. AVR Studio está hecho con muchos objetos SW encapsulados con tecnología Microsoft DCOM. Muchos de los objetos del software no tienen su propia interfaz de usuario. La ventana Messages es la ventana común que todos los módulos de software emplean para presentar mensajes al usuario. Los mensajes están codificados por color. La mayoría de mensajes son mensajes simples sin prioridad significativa que no tienen color. Las advertencias que se presentan en amarillo señalan problemas potenciales. Los errores están en rojo. Todos los mensajes pueden mostrarse acompañados del tiempo, haciendo clic derecho con mouse sobre la ventana y seleccionando la opción mostrar el tiempo (time stampoption).

Find in files. AVR Studio 4 tiene una función avanzada para “buscar en archivos”. El resultado de la búsqueda es dirigido hacia esta ventana.

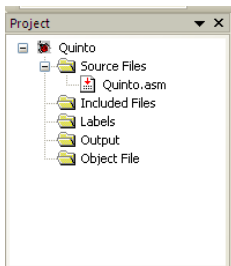
Breakpoints. Enlista todos los breakpoints (puntos de detención) activos en todos los módulos. A los breakpoints se los puede habilitar, deshabilitar y remover desde esta ventana.

2.1.1.2 La Ventana Workspace (Área de Trabajo)

El área de trabajo del AVR Studio 4 consiste de varias ventanas proyectadas para ayudar al desarrollador en la depuración del código que él ha escrito de forma sistemática.

Las ventanas disponibles dentro de la ventana Workspace son las siguientes:

2.1.1.3 La ventana Project



Si se ha creado el código de un proyecto, la ventana Project enlista los archivos que están incluidos en dicho proyecto. Si se tiene abierto un archivo objeto para depuración, la ventana Project muestra el nombre del archivo objeto actualmente cargado así como también los archivos fuente a los cuales el archivo objeto se refiere.

Figura 2.4. Ventana Project

2.1.1.4 La ventana I/O

Esta ventana despliega información de los registros de I/O así como también del procesador y de los registros.

La ventana I/O permite visualizar los siguientes campos:

Registro. Todos los controladores AVR tienen un conjunto de 32 registros de propósito general que pueden usarse libremente por el programador o compilador. Cada vez que una simulación se detiene temporalmente (break), este campo se actualiza con los valores actuales de los registros internos del dispositivo. Si el valor cambió desde la última detención, el registro aparece codificado con color.

Procesador. En comparación con el registro de campo, los registros del procesador se actualizan cuando la ejecución se detiene temporalmente (break). Aquí se puede ver el contador del programa, el contador de la pila (stack) y otros.

Registros de los módulos de I/O. Cada dispositivo AVR diferente se caracteriza por los periféricos que el dispositivo soporta. Un ATmega169 tiene un conjunto de periféricos completamente diferente comprado con un AT90S8515, pero el núcleo y el conjunto de instrucciones de los dos chips son totalmente iguales. Todos los periféricos son controlados a través de registros de 8 o 16 bits que pueden ser leídos ó escritos. Este campo de I/O se configura para el dispositivo que se ha seleccionado e indica todos los registros y bits lógicamente agrupados. Los bits y registros pueden ser escritos cuando la simulación está en modo break. Esta vista ofrece control total sobre el dispositivo sometido a depuración. La actualización de los dispositivos que soporta AVR Studio se hacen a través de la página de Atmel en la web.

2.1.1.5 La ventana Info

Esta ventana es estática y muestra todas las interrupciones, configuración de pin y direcciones disponibles de I/O para el dispositivo seleccionado.

La ventana Info se subdivide en los siguientes campos:

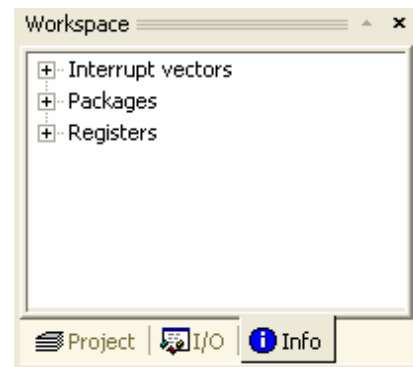


Figura 2.5. Ventana info

Interruptvectors. Los vectores de interrupción para un dispositivo usualmente reflejan sus periféricos. En consecuencia son diferentes para cada dispositivo AVR. Este campo enlista todas las interrupciones y sus correspondientes direcciones de vector de interrupción. Este material es útil si se programa las rutinas para el servicio de interrupción.

Packages. Enlista los encapsulados disponibles para el dispositivo, y la correspondiente distribución de pines para cada encapsulado. La ayuda informativa sobre las herramientas tendrá información sobre el uso de la mayoría de los pines. Así mismo, es una característica útil si se está depurando HW y SW, se tiene la información necesaria de pines en todo momento en este campo.

Registers. Es un complemento para la ventana de I/O. La ventana de I/O muestra todos los registros lógicamente agrupados en funciones periféricas. Este campo enlista a todos los registros desde la dirección más alta hasta la más baja.

La Ventana Watch

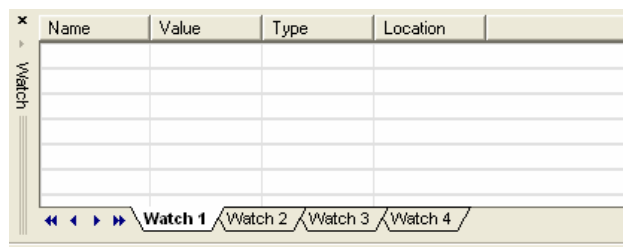


Figura 2.6. Ventana Watch

Al depurar lenguajes de alto nivel como C/C++, se necesita vigilar las variables. Con herramientas como AVR Studio 4 esto es una tarea fácil, se debe hacer clic sobre la variable que se quiera vigilar, arrastrarla y soltarla dentro de la ventana para vigilancia (Watch). Si la variable seleccionada es una estructura o una matriz, un símbolo [+] aparecerá en frente de la variable, indicando que ésta puede ser expandida dentro de la vista. Cuando la ejecución se interrumpe temporalmente

(break), la variable se actualizará automáticamente si está dentro del alcance. Diferentes lenguajes definen su alcance, pero estar dentro del alcance significa que la variable actualmente existe en la memoria en la localidad donde el programa detiene su ejecución.

2.1.1.6 La Ventana Memory

Un microcontrolador como el AVR no puede hacer nada sin memoria. El código ejecutado está en la memoria de programa, las variables se ubican en la SRAM (principalmente), los registros de I/O se distribuyen dentro del área de memoria de I/O y la EEPROM es otra área de memoria. La ventana Memory está disponible para visualizar todos los diferentes tipos de memoria asociados a los dispositivos AVR. Y en relación a la ventana Watch y la I/O, el área expuesta en la pantalla se actualiza automáticamente cuando ocurre una detención temporal (break) de la ejecución.

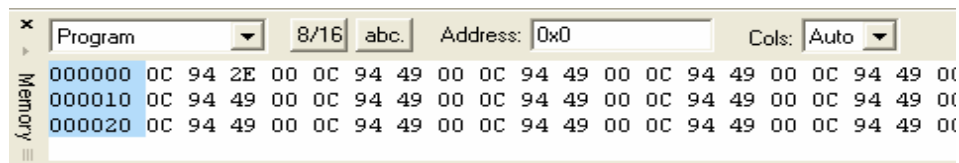


Figura 2.7.Ventana Memory

2.1.1.7 La Barra de Estado

La barra de estado siempre indica el nombre del dispositivo actual, la plataforma de depuración, el estado de ejecución y la ubicación del cursor dentro de la ventana Editor.

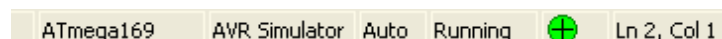


Figura 2.8.Barra de Estado

2.1.2 PROTEUS 7.7

Proteus 7.7 es un software de simulación y diseño de PCB distribuido por LabcenterElectronics, que permite analizar el comportamiento de los circuitos, previo a la implementación física del circuito.

Es la única plataforma que nos ofrece la posibilidad de co-simular el código para microcontroladores en bajo y alto nivel.

Provee de herramientas de edición, a su vez de librerías que contienen componentes análogos y familias de microcontroladores entre ellas Microchip, Motorola, ARM, AVR, y microprocesadores de la familia ATMEL, Motorola entre otros.

Una de las ventajas para el desarrollo del proyecto es que se hizo uso de los ejemplos para ATMEL en el cual encontramos el Kit AVR Butterfly funcionando con su programa demo, con lo cual no fue necesario el diseño del esquemático correspondiente.

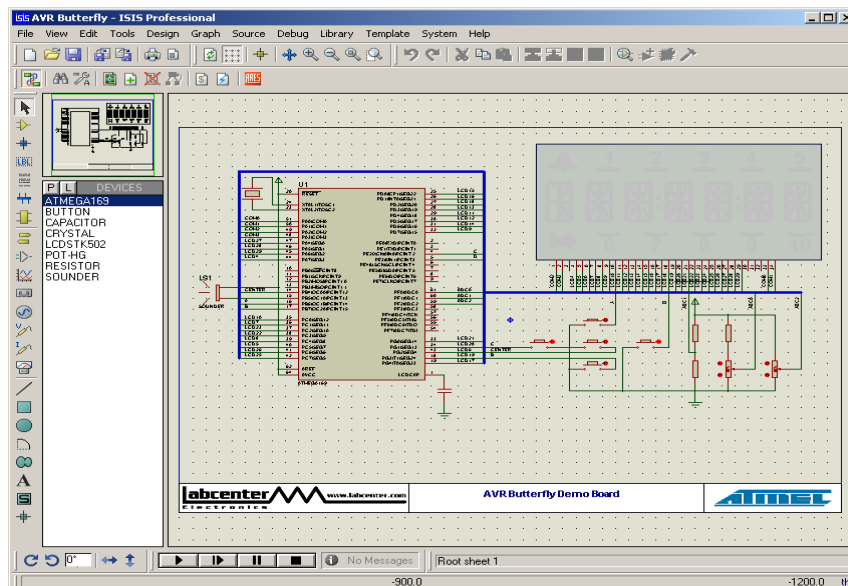


Figura2.9. Demo del AVR Butterfly en Proteus

2.1.3 AVR BUTTERFLY

El kit de evaluación del AVR Butterfly está diseñado para mostrar las características y beneficios más importantes de los microcontroladores ATMEL.

El microcontrolador usado por el AVR Butterfly es el ATmega169, que combina la tecnología Flash con el más avanzado microcontrolador de 8 bits disponible.



Figura 2.10 AVR Butterfly vista frontal

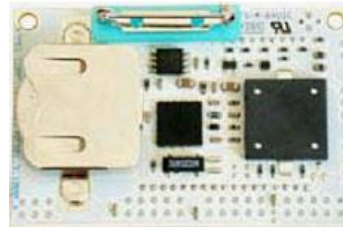


Figura 2.11. AVR Butterfly vista trasera

El kit del AVR butterfly posee un lector de voltaje (ADC, CANAL 1), por lo cual el AVR Butterfly es capaz de leer voltajes en el rango de 0 a 5 V. El voltaje a leerse debe aplicarse a los conectores externos del canal uno del ADC, tal como se indica en la Figura. Valiéndose de un divisor de voltaje y leyendo el voltaje sobre la resistencia, a través del canal 1 del ADC del ATmega169, se podrá medir el voltaje aplicado. La precisión es aproximadamente de 0.1 V.

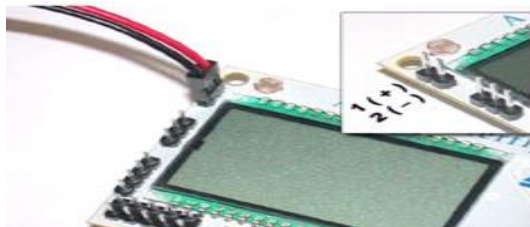


Figura2.12. Canal ADC1del AVR Butterfly

No se debe aplicar voltajes mayores al máximo de 10V

CAPÍTULO 3

IMPLEMENTACIÓN DEL TRABAJO

3.0.- GENERALIDADES[ref.- 5]

En el transcurso del desarrollo del proyecto se realizaron conexiones externas en el KIT AVR Butterfly,

Para facilitar el acceso a PORTD y PORTB, es necesario soldar Headers de 10 pines en los conectores del AVR Butterfly. Para hacerlo se recomienda utilizar un espadín.



Figura3.1. Espadines de conexion

Es necesario dividir el espadín en segmentos de 5 pines para formar Headers de 10 pines que coincidan con los agujeros de los conectores del AVR Butterfly. Entonces, se debe soldar los pines de dichos segmentos de tal modo que queden fijos.

El USART es accesible a conexiones externas mediante conectores de tres pines. Para mejorar el acceso, es necesario soldar un segmento de espadín de 3 pines.

Una vez optimizado el acceso a USART, es necesario un cable para interfaz serial RS-232. Para el un extremo del cable, se recomienda usar un conector de 3 entrada, que sea compatible con la dimensión del conector soldado en el AVR Butterfly. En el otro extremo del cable, se recomienda utilizar un conector hembra DB-9, pues

este a su vez será compatible con el conector macho cuyo otro extremo que es un USB estará conectado a la PC. En lo que respecta al cable, es recomendable usar uno flexible.



Figura 3.2. AVR Butterfly

A continuación se muestra la conexión entre la PC y el AVR Butterfly, utilizando el cable para la interfaz serial.

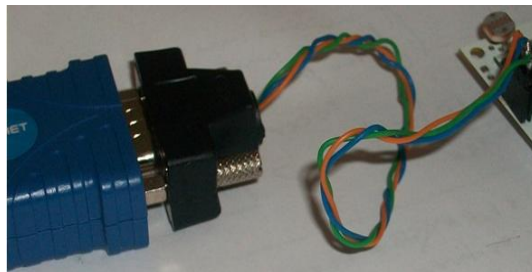


Figura 3.3. Conector serial BD9

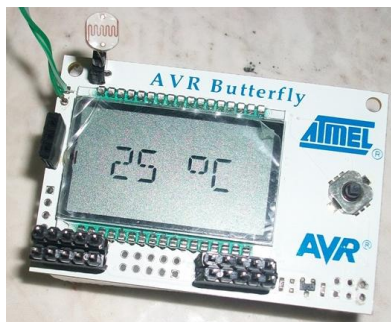


Figura 3.4. Butterfly midiendo temperatura

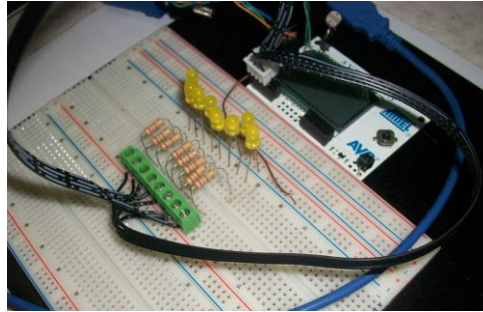


Figura 3.5. Proyecto en protoboard

FUENTE DE ALIMENTACIÓN EXTERNA

El AVR Butterfly puede ser alimentado/energizado por una fuente de voltaje externa de 3 V, a través de los pines VCC_EXT y GND de los conectores externos PORTB y PORTD. Por comodidad, se recomienda utilizar dos baterías tipo AA de 1.5 V y un porta-baterías para las mismas.

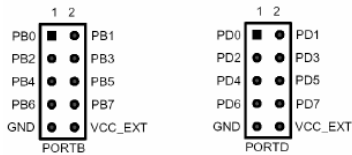


Figura 3.6. PortB y PortD



Figura 3.7. Baterías de alimentación

LISTA DE MATERIALES

1 AVR Butterfly	1 LDR
2 Pilas AA	Espadies
Cable para conexiones	1 Protoboard
1 Conector DB9-USB	8 Diodos leds
8 Resistencias 320 ohmios	1 Pulsador

CAPÍTULO 4

DESARROLLO

4.1.- USO DEL ADC JUNTO – MEDIDOR DE TEMPERATURA[ref.- 1]

1.- Enunciado del proyecto

Especificaciones

En este ejemplo se utilizará el ADC para realizar mediciones de temperatura a través del Sensor de Temperatura del tipo Resistor NTC (Coeficiente Negativo de Temperatura), el cual viene incluido en el Kit AVR Butterfly.

El sensor de temperatura se puede encontrar en la parte de atrás del AVR Butterfly como se muestra en la figura.

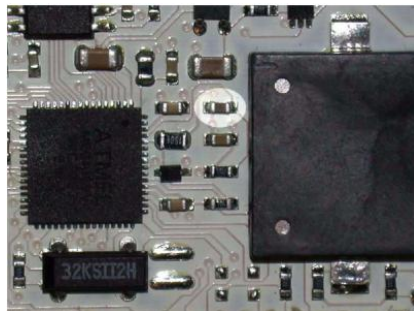


Figura 4.1. Sensor de temperatura integrado al Butterfly

Un termistor NTC se caracteriza por el hecho de que cuando la temperatura disminuye la resistencia aumenta y viceversa. La temperatura puede ser calculada, usando un divisor de voltaje para leer el voltaje sobre el termistor a través de los canales del ADC en el Atmega169. La ecuación para el cálculo de la temperatura en grados Celsius es la siguiente:

$$Temperatura = \frac{\beta}{\left(\ln\left(\frac{ADC}{1024-ADC}\right) + \frac{\beta}{T_{amb}}\right)} - T_o$$

$$\beta = 4250$$

$$ADC = \text{ValordelregistrodedatosdelADCenelATmega169} - ADCLyADCH$$

$$T_o = 273^{\circ}K$$

$$T_{amb} = 293^{\circ}K = 273^{\circ} + 20^{\circ}$$

La temperatura en Fahrenheit:

$$\left(\frac{F - 32}{1.8} \right) = C$$

El AVR Butterfly es capaz de medir temperaturas desde -10°C/+14°F hasta 60°C/140°F con una precisión de $\pm 1^{\circ}C$.

2.- Diagrama de bloques

Los recursos del AVR Butterfly que se serán usados en el medidor de temperatura se muestran en el diagrama de bloques a continuación.

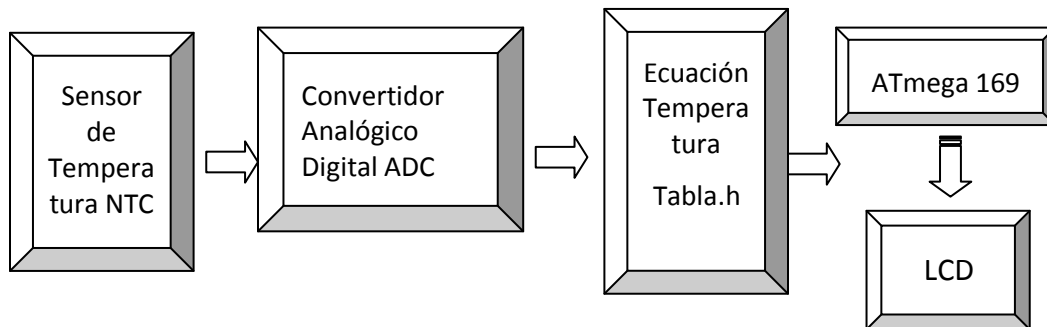


Figura 4.2. Diagrama de bloques de medición de temperatura

3.- Diagrama de flujo funcional del programa

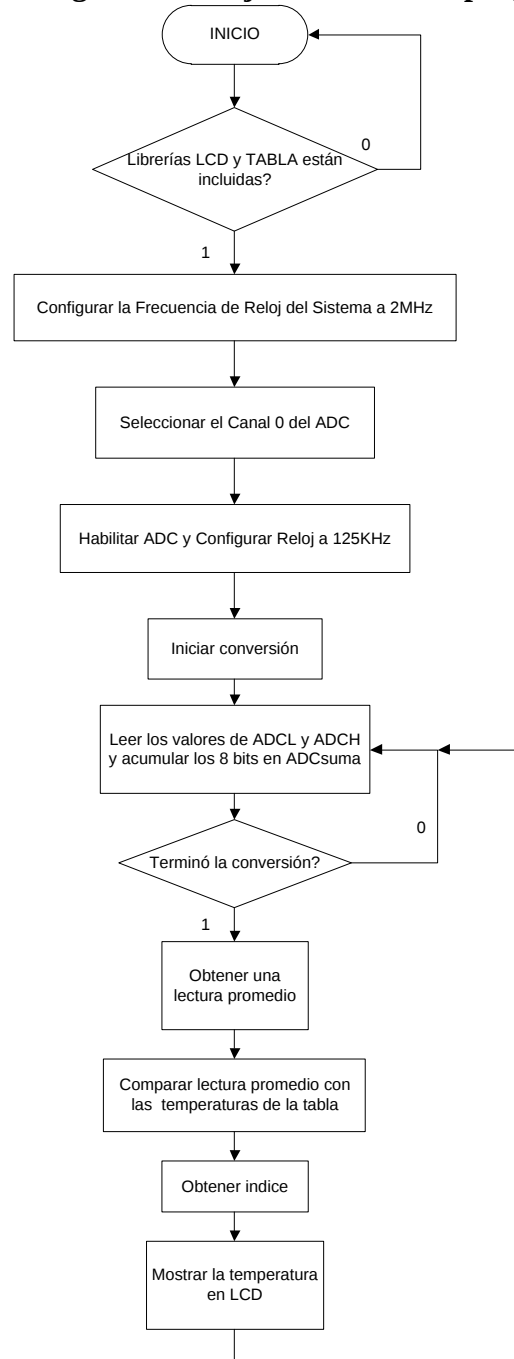


Figura 4.3. Diagrama de flujo de medición de temperatura

4.-Descripción del algoritmo o estrategia utilizado

El microcontrolador utilizado en el siguiente ejercicio es el Atmega169 que viene incluido en el Kit AVR Butterfly.

Al inicio se incluirán los archivos de cabecera (#include):

- avr/io.h, que contiene la definición de los registros y de sus respectivos bits.
- stdlib.h, contiene macros y funciones básicas, esta librería es necesaria para poder utilizar la función itoa(), presente en el código, que transforma un valor entero en una cadena de caracteres.
- LCD.h, es un código desarrollado que contiene algunas de las funciones más cotidianas para el uso del LCD.
- tabla.h, archivo que contiene las lecturas ADC correspondientes a las temperaturas de 0 °C a 60 °C, calculadas mediante la Ecuación de Temperatura.

A continuación se declararán las funciones necesarias para la conversión ADC y para el procesamiento de la información obtenida.

```
voidInicializar_ADC(void);  
intleer_ADC(void);  
void leer_temperatura(void);
```

La función **Inicializar_ADC()**;

```
voidInicializar_ADC(void)
```

```
{  
  Seleccionar la frecuencia del reloj del sistema,  $RC/4 = 8\text{MHz}/4 = 2\text{ MHz}$ :  
  CLKPR = (1<<CLKPCE);  
  CLKPR = (1<<CLKPS1);  
  Seleccionar el Canal 0 del ADC:  
  ADMUX = 0;  
  Habilitar el ADC y seleccionar el reloj para el ADC,  $2\text{MHz}/16 = 125\text{ KHz}$   
  para obtener una velocidad de muestreo promedio.  
  “Para obtener máxima resolución, el fabricante recomienda seleccionar una  
  frecuencia entre 50KHz - 200 KHz”  
  ADCSRA = (1<<ADEN)|(1<<ADPS2);
```

Ejecutar una conversión para inicializar el ADC y evitar errores en la primera lectura de la aplicación:

```
leer_ADC();  
}
```

La función **leer_ADC()**:

```
intleer_ADC(void)  
{  
chari;  
intADC_temp;
```

```
intADCsuma = 0;
```

Activa y energiza los sensores del AVR Butterfly, PORTF3 será la alimentación VCP para el Resistor NTC y para la LDR.

Configurar el pin 3 del Puerto F como salida:

```
PORTF |= (1<<PF3);
```

Configurar pin 3 del Puerto F a nivel alto para activar/energizar a los sensores:

```
DDRF |= (1<<DDF3);
```

Habilitar el ADC, por si otra aplicación lo deshabilita:

```
ADCSRA |= (1<<ADEN);
```

Activa y ordena la ejecución de una conversión ADC, se ejecutarán 8 conversiones.

```
ADCSRA |= (1<<ADSC);
```

Esperar hasta que el ADC indique que ha culminado una conversión, mediante el sondeo del indicador ADIF:

```
while(!(ADCSRA & 0x10));
```

Repetir ocho veces el proceso dentro del lazo FOR:

```
for(i=0;i<8;i++)
```

```
{
```

Activar/ordenar la ejecución de una conversión ADC:

```
ADCSRA |= (1<<ADSC);
```

Esperar hasta que el ADC indique que ha culminado una conversión, mediante el sondeo del indicador ADIF:

```
while(!(ADCSRA & 0x10));
```

Leer el byte bajo del Registro de Datos de 16 bits del ADC:

```
ADC_temp = ADCL;
```

Leer el byte alto del Registro de Datos de 16 bits del ADC:

```
ADC_temp += (ADCH << 8);
```

Acumular en ADCsuma los resultados de las 8 lecturas/conversiones ADC:

```
ADCsuma += ADC_temp;
```

```

}
Calcular el promedio de las lecturas ADC. Como son 8 lecturas, se debe
dividir ADCsuma para 8, puesto que es lo mismo que desplazar el contenido
de ADCsuma hacia la derecha 3 posiciones:
ADCsuma = ADCsuma>> 3;
Deshabilitar los Sensores quitándoles la alimentación de voltaje desde
PORTF3:
PORTF&=~(1<<PF3);
DDRF&=~(1<<DDF3);
Deshabilitar el ADC:
ADCSRA&=~(1<<ADEN);
Retornar el promedio de las lecturas/conversiones, retorna un valor de 16 bits
(int):
returnADCsuma;
}

```

La función **leer_temperatura()**:

```

voidleer_temperatura(void)
{
intlectura=0;
inti = 0;
Reservar espacio para máximo 3 dígitos para la lectura de temperatura, puesto
que luego del procesamiento de la lectura ADC el resultado será desplegado
en el LCD; para ello, es necesario convertir el resultado final de la lectura, de
un valor entero a una cadena de máximo tres caracteres:
chartemperatura_ASCII[]={ '0','0','0','\0'};
Leer la temperatura utilizando el ADC:

lectura = leer_ADC();
Localizar en la tabla el índice correspondiente a la lectura ADC, en la tabla
constan 61 lecturas, correspondientes a 0 °C hasta 60 °C, esto quiere decir
índices de 0 hasta 61:
for(i=0;i<=61;i++)
{
Incrementar el índice hasta que lectura sea igual o mayor a uno de los valores
indexados en la tabla almacenada en la memoria de programa.
pgm_read_word() lee una palabra en la memoria de programa Flash con
direcciones de 16 bits:
if(lectura>=pgm_read_word(&TEMP_Celcius_pos[i]))
{

```

Cuando llega a ser igual o mayor, romper el lazo FOR que se utiliza para búsqueda, quedando almacenado el índice en la variable i:

```
break;  
}  
}
```

Obtenido el índice, que en este caso representa la temperatura, se realiza una conversión de formato entero a formato ASCII, esta función está contenida en el archivo cabecera stdlib.h. Entonces, almacenar en una matriz, en este caso temperatura_ASCII, el valor del entero contenido en la variable 'i' (el índice) que representa la temperatura en grados Celsius:

```
itoa(i,temperatura_ASCII,10);
```

Enviar a escribir/desplegar en el LCD el valor resultante del proceso de medición de temperatura (conversión ADC-procesamiento):

```
escribir_caracter_en_LCD(temperatura_ASCII[0],0);  
escribir_caracter_en_LCD(temperatura_ASCII[1],1);  
escribir_caracter_en_LCD(temperatura_ASCII[2],2);  
escribir_caracter_en_LCD('.',3);  
escribir_caracter_en_LCD('C',4);  
actualizar_LCD();
```

```
}
```

A continuación se debe crear y guardar el archivo tabla.h que contiene las conversiones ADC equivalentes, o las más cercanas, a las que corresponden a los valores enteros de Temperatura calculados con la Ecuación 6.2 en el rango de 0 °C a 60 °C.

PROGMEM permite el acceso a la memoria de programa FlashROM, entonces la tabla TEMP_Celcius_pos[] de temperaturas será declarada como una matriz que estará almacenada en la Memoria Flash de Programa.

```
constintTEMP_Celcius_pos[] PROGMEM =  
{ // de 0 a 60 grados  
761,750,738,727,715,703,691,678,666,653, //0-9 °C  
640,628,615,602,589,576,563,551,538,525, //10-19 °C  
512,500,487,475,462,450,438,426,415,403, //20-29 °C  
392,381,370,359,349,338,328,319,309,300, //30-39 °C  
291,282,273,265,256,248,240,233,225,218, //40-49 °C  
211,205,198,192,186,180,174,168,163,158, //50-59 °C  
153 //60 °C  
};
```

Para terminar se realizará la función principal. La función **main()** que empieza ejecutando la inicialización del ADC y del LCD, habilita el uso de interrupciones y despliega la palabra “Temperatura” en el LCD; luego entra en el lazo infinito, ordenando al microcontrolador que ejecute una lectura de Temperatura y que entre en modo Sleep luego de atender a cualquier interrupción.

```

intmain(void)
{
    char *palabra="Temperatura\0";
    Inicializar el ADC y el LCD:
        Inicializar_ADC();
        Inicializar_LCD();
    Habilitar globalmente las interrupciones:
        sei();

    Desplegar en el LCD la palabra “Temperatura”:
        escribir_palabra_en_LCD(palabra);
        while(1)
        {
            Leer la temperatura y desplegar su valor en el LCD:
                leer_temperatura();
            Entrar en modo Sleep, en el modo ADC NoiseReduction (Reducción de
            Ruido del ADC):
                SMCR=(1<<SM0)|(1<<SE);
            }
        return 0;
    }

```

5.- Listado del programa fuente en lenguaje c con comentarios en las líneas de código que considere fundamentales

```

#include<avr/io.h>                // contiene la definición de registros y sus
respectivos bits
#include<avr/interrupt.h>
#include<stdlib.h>                // contiene funciones básicas, esta librería es
necesaria para usar la función itoa() más adelante
#include<avr/delay.h>
#include "lcd.h"                  // contiene varias funciones para el manejo del LCD

```

```
#include "tabla.h"// contiene los valores obtenidos gracias a la ecuación // del cálculo de
Temperatura
```

```
// Se declaran varias funciones que son necesarias para la conversión ADC.
```

```
voidInicializar_ADC(void);
```

```
intleer_ADC(void);
```

```
voidobtenerTemperatura(void); // con esta función obtenemos en valor de la temperatura
```

```
int main(void)
```

```
{
```

```
    unsigned char i = 0;
```

```
    char palabra[NUMERO_MAXIMO_DE_CARACTERES]="Temperatura\0";
```

```
    Inicializar_ADC();
```

```
    inicializar_LCD();
```

```
    sei(); // se habilitan las interrupciones globalmente
```

```
    escribir_palabras_en_LCD(palabra);
```

```
    for(i=0;i<5;i++)_delay_loop_2(50000); // retardo
```

```
    while(1) // lazo infinito, se llama a la función para obtener el valor de la
Temperatura
    {
```

```
        obtenerTemperatura();//una vez que se obtuvo el valor se lo muestra en el LCD
```

```
        SMCR=(1<<SM0)|(1<<SE);// entra en modo Sleep, en el modo ADC
NoiseReduction
```

```
        //(Reducción de Ruido del ADC):
```

```
    }
```

```
    return 0;
```

```
}
```

```
voidInicializar_ADC(void)
```

```
{
```

```
// Se selecciona la frecuencia del reloj del sistema
```

```
// RC/4 = 8MHz/4 = 2 MHz
```

```
    CLKPR = (1<<CLKPCE);
```

```
    CLKPR = (1<<CLKPS1);
```

```
    ADMUX = 0;
```

```
    // Se selecciona el canal 0 del ADC
```

```
    ADCSRA = (1<<ADEN)|(1<<ADPS2); // con ADEN habilitamos el ADC
```

```
    // además seleccionamos el reloj para el ADC, 2MHz/16 = 125 KHz
```

```
    leer_ADC();
```

```
}
```

```
intleer_ADC(void)
```

```
{
```

```
    chari;
```

```

    intADC_temp;
    intADCsuma = 0;
    PORTF |= (1<<PF3); //Seleccionamos el pin 3 del Puerto F como salida
    DDRF |= (1<<DDF3); //Configuramos el pin 3 del Puerto F a nivel alto para
activar los sensores
    ADCSRA |= (1<<ADEN);
    ADCSRA |= (1<<ADSC); // Se ordena el inicio de la conversión
// El bit AFIF es el que indica que se realizó la conversión, por lo tanto
// realizamos el siguiente lazo while para preguntar si la conversión ha finalizado
    while(!(ADCSRA & 0x10));
    for(i=0;i<8;i++)
    {
        ADCSRA |= (1<<ADSC);
        while(!(ADCSRA & 0x10));
// Leer el byte bajo del Registro de Datos de 16 bits del ADC
        ADC_temp = ADCL;
// Leer el byte alto del Registro de Datos de 16 bits del ADC
        ADC_temp += (ADCH << 8);
        ADCsuma += ADC_temp; // en ADCsuma son acumulados los 8 valores
de las lecturas
    }
    ADCsuma = (ADCsuma>> 3);
    PORTF&=~(1<<PF3); //Desabilitamos los sensores y el ADC
    DDRF&=~(1<<DDF3);
    ADCSRA&=~(1<<ADEN);
    returnADCsuma;}

voidobtenerTemperatura(void)
{
    intlectura=0;
    inti = 0;
    unsigned char j = 0;
    chartemperatura_ASCII[]={ '0','0','0','0','0' };
    lectura = leer_ADC();
    for(i=0;i<=61;i++)
    {
        if(lectura>pgm_read_word(&TEMP_Celcius_pos[i]))
        {
            break;
        }
    }
    itoa(i,temperatura_ASCII,10);
    escribir_caracter_en_LCD(temperatura_ASCII[0],0);
    escribir_caracter_en_LCD(temperatura_ASCII[1],1);

```

```

        escribir_caracter_en_LCD(temperatura_ASCII[2],2);
        escribir_caracter_en_LCD('.',3);
        escribir_caracter_en_LCD('C',4);
        actualizar_LCD();
        for(j=0;j<5;j++)_delay_loop_2(50000);
    }
}
Lcd.h

```

```

//*****
//
//          * * MICROCONTROLADORES AVANZADOS * *
//*****
//
//      Grupo # 5
//      Integrantes:      Grace Yagual
//                        Danilo Riera
//                        Miguel Rodriguez
//*****
//
//
//          * * LCD * *
//
//
// Este archivo contiene varias funciones que nos ayudarán
// con el manejo del LCD del AVR Butterfly.
//
// Nombre del archivo: lcd.h
//*****
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <avr/signal.h>
#include <avr/interrupt.h>
#include <avr/delay.h>

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

charLCD_Data[TAMANIO_DEL_REGISTRO_LCD];
charmemo_temp_texto[NUMERO_MAXIMO_DE_CARACTERES];
unsignedchar ESCRITURA_DE_CADENA_HABILITADO = 0;
unsigned char LCD_INT_contador = 0;
voidinicializar_LCD(void);
voidescribir_caracter_en_LCD(char , char );
voidescribir_palabras_en_LCD(char *);
voidborrar_LCD(void);
voidactualizar_LCD(void);
unsignedinttabla_de_caracteres_LCD[] PROGMEM =

```

```

{
0x0A51, // '*' (?)
0x2A80, // '+'
0x0000, // ',' (Sin definir)
0x0A00, // '-'
0x0A51, // '.' Signo de grados
0x0000, // '/' (Sin definir)
0x5559, // '0'
0x0118, // '1'
0x1e11, // '2'
0x1b11, // '3'
0x0b50, // '4'
0x1b41, // '5'
0x1f41, // '6'
0x0111, // '7'
0x1f51, // '8'
0x1b51, // '9'
0x0000, // ':' (Sin definir)
0x0000, // ';' (Sin definir)
0x0000, // '<' (Sin definir)
0x0000, // '=' (Sin definir)
0x0000, // '>' (Sin definir)
0x0000, // '?' (Sin definir)
0x0000, // '@' (Sin definir)
0x0f51, // 'A' (+ 'a')
0x3991, // 'B' (+ 'b')
0x1441, // 'C' (+ 'c')
0x3191, // 'D' (+ 'd')
0x1e41, // 'E' (+ 'e')
0x0e41, // 'F' (+ 'f')
0x1d41, // 'G' (+ 'g')
0x0f50, // 'H' (+ 'h')
0x2080, // 'I' (+ 'i')
0x1510, // 'J' (+ 'j')
0x8648, // 'K' (+ 'k')
0x1440, // 'L' (+ 'l')
0x0578, // 'M' (+ 'm')
0x8570, // 'N' (+ 'n')
0x1551, // 'O' (+ 'o')
0x0e51, // 'P' (+ 'p')
0x9551, // 'Q' (+ 'q')
0x8e51, // 'R' (+ 'r')
0x9021, // 'S' (+ 's')
0x2081, // 'T' (+ 't')

```

```

0x1550, // 'U' (+ 'u')
0x4448, // 'V' (+ 'v')
0xc550, // 'W' (+ 'w')
0xc028, // 'X' (+ 'x')
0x2028, // 'Y' (+ 'y')
0x5009, // 'Z' (+ 'z')
0x0000, // '[' (Sin definir)
0x0000, // '\' (Sin definir)
0x0000, // ']' (Sin definir)
0x0000, // '^' (Sin definir)
0x0000 // '_'
};
void inicializar_LCD(void)
{
    borrar_LCD();
    LCDCRA = (1<<LCDEN) | (1<<LCDAB);
    LCDCCR = (1<<LCDDC2)|(1<<LCDDC1)|(1<<LCDDC0)|(1<<LCDCC3)|(1<<LCDCC2)
    |(1<<LCDCC1)|(1<<LCDCC0);
    ASSR = (1<<AS2);
    LCDFRR = (0<<LCDPS0) | (1<<LCDCD1)|(1<<LCDCD0);
    LCDCRB
    (1<<LCDCS)|(1<<LCDMUX1)|(1<<LCDMUX0)|(1<<LCDPM2)|(1<<LCDPM1)
    |(1<<LCDPM0);
    LCDCRA |= (1<<LCDIE);
}
void describir_caracter_en_LCD(char c, char posicion)
{
    unsigned int seg = 0x0000;
    char mascara, nibble;
    char *ptr;
    char i;
    if (posicion > 5) return;
    if ((c >= '*') && (c <= 'z'))
    {
        if (c >= 'a') c &= ~0x20;
        c -= '*';
        seg = (unsigned int) pgm_read_word(&tabla_de_caracteres_LCD[(uint8_t)c]);
    }
    if (posicion & 0x01) mascara = 0x0F;
    else mascara = 0xF0;
    ptr = LCD_Data + (posicion >> 1);
    for (i = 0; i < 4; i++)
    {
        nibble = seg & 0x000F;

```

```

seg>= 4;
if (posicion& 0x01) nibble<= 4;
*ptr = (*ptr& mascara) | nibble;
ptr += 5;
}
}
void describir_palabras_en_LCD(char *palabra)
{
    unsigned char i=0;
    for( i=0;i<NUMERO_MAXIMO_DE_CARACTERES;i++)
        memo_temp_texto[i]='\0';
    LCD_INT_contador = 0;
    ESCRITURA_DE_CADENA_HABILITADO = 1;
    for(
        i=0;(i<NUMERO_MAXIMO_DE_CARACTERES)&&(*palabra!='\0');i++,palabra++)
        memo_temp_texto[i]=*palabra;
    }
    void borrar_LCD(void)
    {
        unsigned char i=0;
        for( i=0;i<NUMERO_MAXIMO_DE_CARACTERES;i++)
            memo_temp_texto[i]='\0';
        for (i = 0; i< TAMANIO_DEL_REGISTRO_LCD; i++)
        {
            *(pLCDREG + i) = 0x00;
            *(LCD_Data+i) = 0x00;
        }
        actualizar_LCD();
    }
    void actualizar_LCD(void)
    {
        ESCRITURA_DE_CADENA_HABILITADO = 0;
        for (char i = 0; i< TAMANIO_DEL_REGISTRO_LCD; i++)
            *(pLCDREG + i) = *(LCD_Data+i);
        }
        SIGNAL(SIG_LCD)
        {
            unsigned char letra=0;
            unsigned char i=0;
            if (ESCRITURA_DE_CADENA_HABILITADO==1)
            {
                for(i=0;(i<6);i++)
                {
                    if(!(memo_temp_texto[i+LCD_INT_contador]=='\0'))

```

```

{
letra = memo_temp_texto[i+LCD_INT_contador];
escribir_caracter_en_LCD(letra,i);
}
else
{
escribir_caracter_en_LCD(' ',i);
}
_delay_loop_2(20000);
}
if(LCD_INT_contador<NUMERO_MAXIMO_DE_CARACTERES)
LCD_INT_contador++;
else
{
LCD_INT_contador=0;
ESCRITURA_DE_CADENA_HABILITADO = 0;
}
}
for (char i = 0; i< TAMANIO_DEL_REGISTRO_LCD; i++)
*(pLCDREG + i) = *(LCD_Data+i);
}

```

Tabla.h

```

const int TEMP_Celcius_pos[] PROGMEM = // temperaturas - ADC
{ // de 0 a 60 grados
761,750,738,727,715,703,691,678,666,653, //0-9 °C
640,628,615,602,589,576,563,551,538,525, //10-19 °C
512,500,487,475,462,450,438,426,415,403, //20-29 °C
392,381,370,359,349,338,328,319,309,300, //30-39 °C
291,282,273,265,256,248,240,233,225,218, //40-49 °C
211,205,198,192,186,180,174,168,163,158, //50-59 °C
153 //60 °C
};

```

6.- Copia impresa del circuito armado en PROTEUS para la simulación en el momento de su ejecución

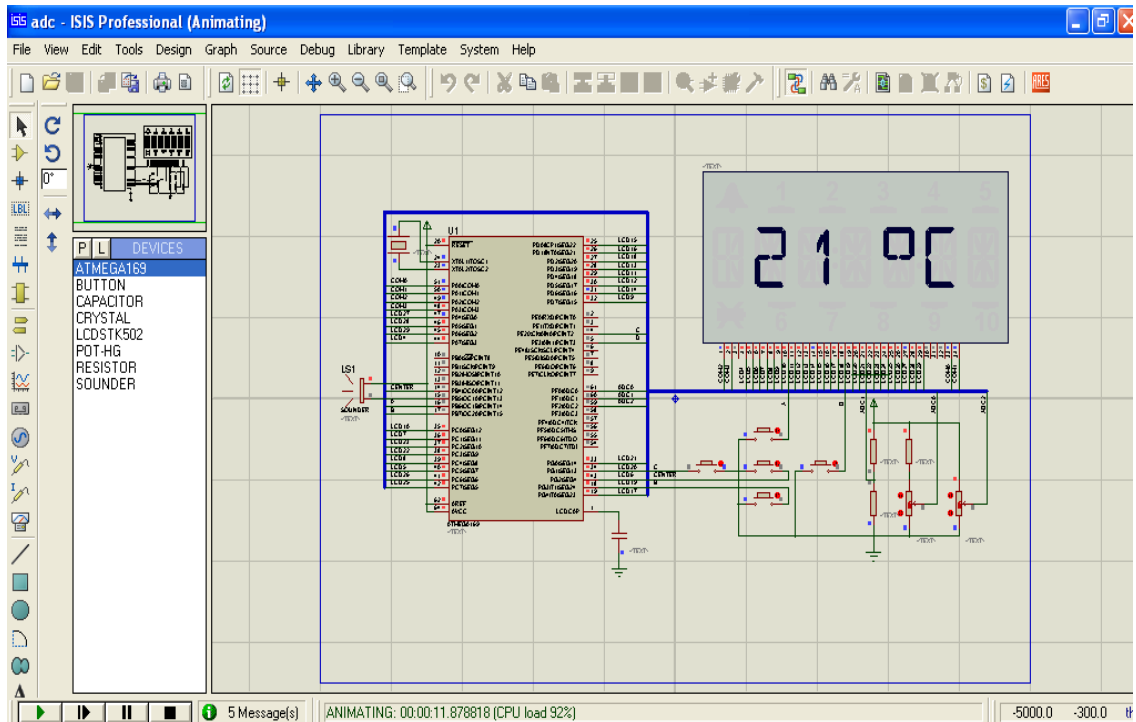


Figura 4.4. Simulación en Proteus de medición de temperatura

4.2.- APLICACIÓN DEL ADC USANDO LDR

1.- Enunciado del proyecto

En este ejemplo aprenderemos sobre el convertidor analógico digital usando una resistencia que depende de la luz (LDR). Con el movimiento de la mano sobre la LDR, el altavoz (speaker) producirá un sonido parecido al de las antiguas películas de ciencia ficción de la década de 1950.

Luego de producir el sonido en el auricular, habrá una pausa por un período de tiempo que es determinado por el valor de la tensión / resistencia que se mide.

El resultado es un cambio de frecuencia del sonido sobre la base de nuestra aportación de la LDR.

El sensor de luz será ubicado en la parte delantera del AVR Butterfly, sobre la pantalla del LCD.

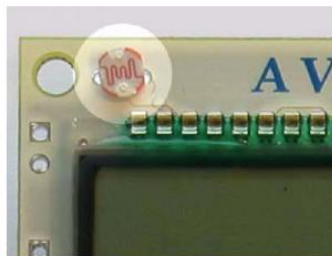


Figura 4.5. Sensor fotoeléctrico LDR

Un LDR se utiliza para medir la luz y se caracteriza por el hecho de que cuando la luz disminuye la resistencia aumenta.

La luz puede ser calculada, usando un divisor de voltaje y leyendo el voltaje sobre el LDR a través de los canales del convertidor analógico digital ADC en el Atmega 169, existente en el AVR Butterfly.

2.- Diagrama de bloques

Los recursos del AVR Butterfly que se serán usados en el presente ejemplo se muestran en el diagrama de bloques a continuación.

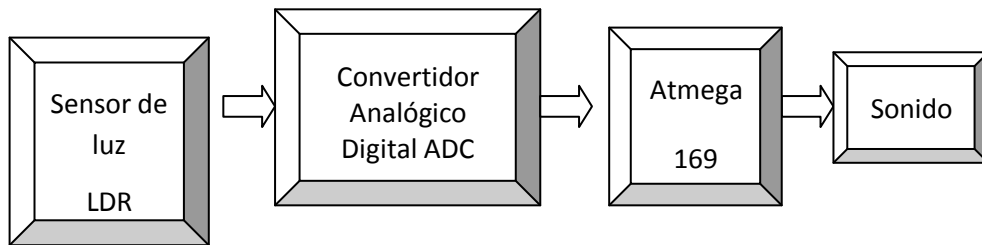


Figura 4.6. Diagrama de bloques LDR con ADC

3.- Diagrama de flujo funcional del programa

3.1.- Diagrama de flujo principal.

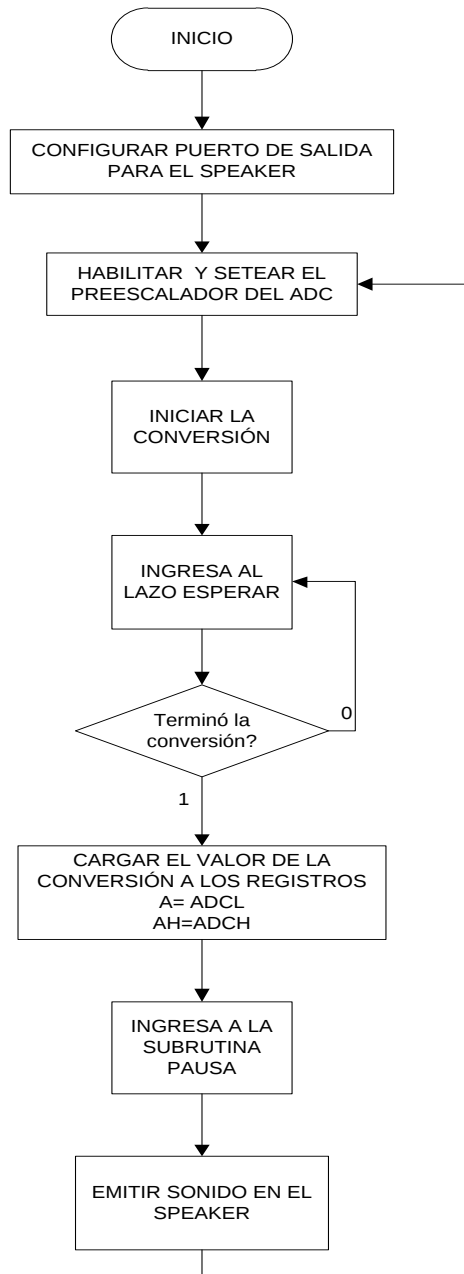


Figura 4.7. Diagrama de flujo del LDR con ADC

3.2.- Subrutina PAUSA

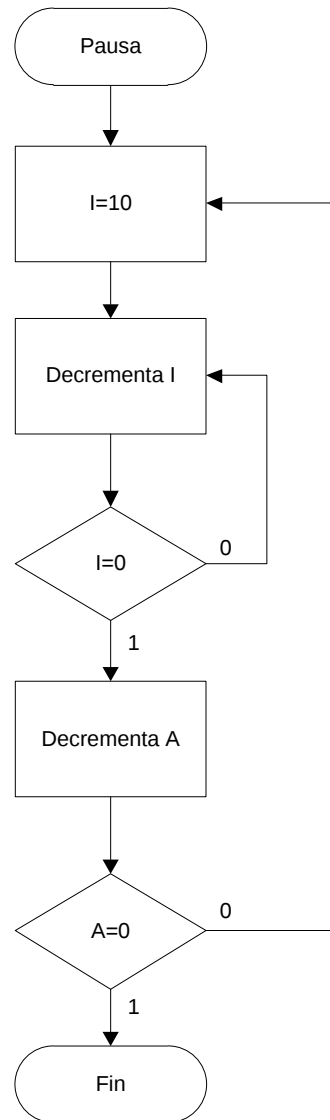


Figura 4.8. Diagrama de flujo de subrutina de pausa

4.-Descripción del algoritmo o estrategia utilizado

El microcontrolador utilizado en el siguiente ejercicio es el Atmega169 que viene incluido en el Kit AVR Butterfly.

El comando `.INCLUDE` le dice el ensamblador que vamos a utilizar, en este caso el Atmega169, en el programa y la directiva `.def` define un nombre para cada registro a ser utilizado.

```
.INCLUDE "M169DEF.INC"
```

```
.DEF A = R16
```

```
.DEF AH = R17
```

```
.DEF I = R21
```

El comando `ORG.` le informa al ensamblador cual es el lugar en la memoria donde deseamos colocar nuestro programar. Debido a que no estamos usando las interrupciones, éste puede comenzar en la parte inferior de la memoria:

```
. ORG $ 0000
```

A continuación configuramos la pila en la parte superior de la memoria. [ref.- 3]

```
RESET: LDI A,HIGH(RAMEND) ;Empieza el programa principal
```

```
OUT SPH,A ;establece el puntero de la pila a la  
;parte superior de la memoria RAM
```

```
LDI A,LOW(RAMEND) ;
```

```
OUT SPL,A
```

Antes de que usemos el speaker en el Puerto B5, tenemos que configurarlo como salida escribiendo un uno en su dirección de registro de datos (DDRB):

```
SBI DDRB,5 ;configura al Puerto como salida para el speaker
```

Se le dirá al Multiplexor ADC que se desea leer la LDR. La ficha técnica nos dice que para seleccionar ADC2 tenemos que establecer el MUX a dos:

```
LDI A,0b0000_0010 ;
```

STS ADMUX,A

Aquí activar la resistencia interna escribiendo un uno en el PORTF3.

SBI PORTF,PORTF3 ;Puerto F3 a uno

El datasheet nos dice que el ADC funciona mejor a una frecuencia entre 50 kHz y 200 kHz. Por lo tanto, se debe seleccionar un pre-escalador con un factor de división de dieciséis bits, ya que 2Mhz dividido por dieciséis nos da una frecuencia de 125kHz.

Así mismo se habilitará el ADC (ADEN=1) y también se iniciará el proceso de la conversión (ADSC=1):

MLUPE: ;lazo principal

LDI A,0b1100_0100 ;Se habilita, se inicia la conversion

STS ADCSRA,A ;y se ajusta el preescalador a 16

Cuando se completa la conversión, la bandera ADIF se establece en uno, lo siguiente es para representar la espera de que termine dicha conversión:

WAIT4:

LDS A,ADCSRA ;espera que la conversión ADC se complete

ANDI A,0b001_0000 ;ADIF = 1

BREQ WAIT4

Ahora que el proceso se completó, leeremos el resultado de la conversión en los registros ADCL y ADCH. El resultado es un número de diez bits, ocho bits en ADCL y dos bits en ADCH, que vamos a ignorar. ADCL debe ser leído primero, seguido por el ADCH para que funcione correctamente.

LDS A, ADCL ;se debe leer ADCL antes que ADCH

LDS AH, ADCH

Ahora se llama a una rutina de pausa, entonces esto cambia la salida en el Puerto B5 que está conectado con el altavoz de la mariposa.

MLUPE:

```
RCALL BPAUSE    ;Espera
SBI  PINB,5     ; speaker intermitente
RJMP MLUPE      ;hacerlo de Nuevo
```

La rutina de pausa es un proceso dentro de un proceso que hace más lenta la rutina, lo suficiente como para que una frecuencia pueda ser escuchada por el altavoz:

```
BPAUSE:         ;rutina pausa
BLOOP: LDI I,10  ;tiempo depende de "A"
BPLUPE: DEC I
BRNE BPLUPE
DEC  A
BRNE BLOOP
```

RET

Como cambia la cantidad de luz que llega a la LDR, la resistencia de la LDR va a cambiar, haciendo que la tensión en nuestra entrada a la ADC cambie. Este cambio de voltaje se convierte en un valor digital por el ADC. Leemos este valor en el registro "A" y se producirá una frecuencia variable a nuestra salida de los altavoces, variando la longitud de tiempo que pasamos en nuestra rutina PAUSA. El resultado es un instrumento musical que controlamos moviendo la mano sobre el circuito.

5.- Listado del programa fuente en lenguaje c con comentarios en las líneas de código que considere fundamentales

```
.INCLUDE "M169DEF.INC" ; encabezado del programa

.DEF A    = R16  ; se define el nombre de los registros
.DEF AH   = R17
.DEF I    = R21
.ORG $0000          ; la dirección es cero.
```

```

RESET:
    LDI A,HIGH(RAMEND)    ;Empieza el programa principal
    OUT SPH,A            ;establece el puntero de la pila a la parte superior de la
memoria RAM
    LDI A,LOW(RAMEND)
    OUT SPL,A
    SBI DDRB,5           ;configura al Puerto como salida para el speaker
    LDI A,0b0000_0010
    STS ADMUX,A
    SBI PORTF,PORTF3      ;Puerto F3 a uno

MLUPE:
    LDI A,0b1100_0100     ;Se habilita, se inicia la conversion
    STS ADCSRA,A          ;y se ajusta el preescalador a 16

WAIT4:
    LDS A,ADCSRA          ;espera que la conversión ADC se complete
    ANDI A,0b001_0000     ;ADIF = 1
    BREQ WAIT4
    LDS A, ADCL            ;se debe leer ADCL antes que ADCH
    LDS AH, ADCH
    RCALL BPAUSE           ;Espera
    SBI PINB,5             ;speaker intermitente
    RJMP MLUPE             ;hacerlo de Nuevo

BPAUSE:                    ;rutina pausa
BLOOP: LDI I,10            ;tiempo depende de "A"
BPLUPE:    DEC I
           BRNE BPLUPE
           DEC A
           BRNE BLOOP
           RET

```

6.- Copia impresa del circuito armado en PROTEUS para la simulación en el momento de su ejecución

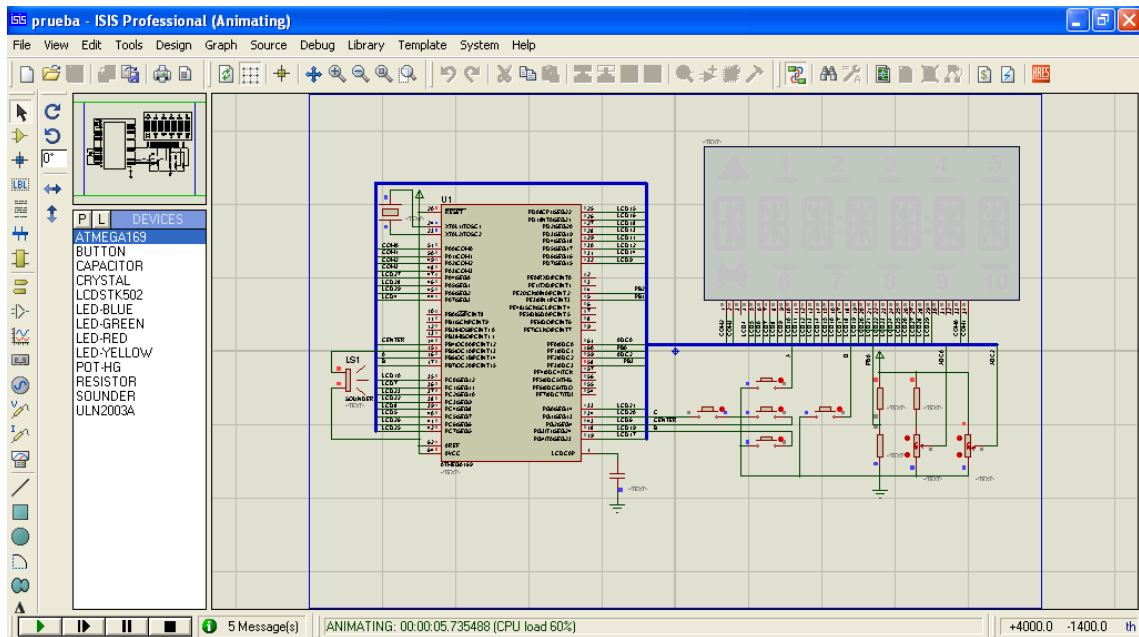


Figura 4.9. Simulación en Proteus LDR con ADC

En la imagen no se muestra nada en la pantalla del LCD por ser la salida el sonido del speaker

4.3.- USO DEL ADC – DEMOSTRACION EN LEDS

1.- Enunciado del proyecto

En este ejemplo usamos un potenciómetro de 5kohmios que lo regulamos en la entrada del microcontrolador por ADC0 y mediante programación nos mostrará en sus salidas por el puerto B un número de forma binaria; cada vez que se varíe el potenciómetro lo relacionará con un voltaje que está en el rango de 0 a 5 voltios.

2.- Diagrama de bloques

Los recursos del AVR Butterfly que se serán usados en el presente ejemplo se muestran en el diagrama de bloques a continuación.

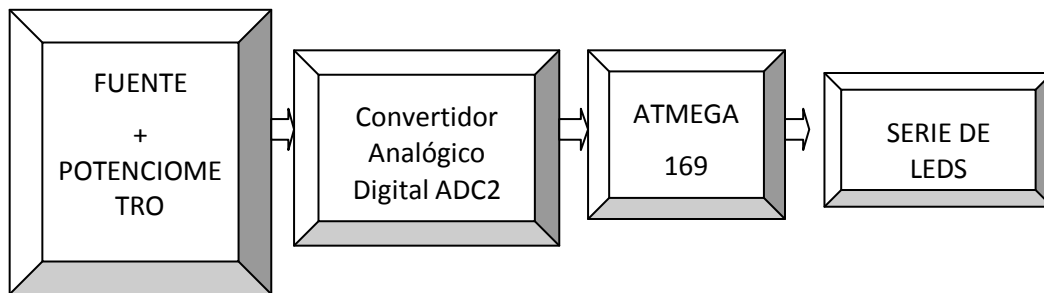


Figura 4.10. Diagrama de bloques del ADC con LEDS

3.- Diagrama de flujo funcional del programa

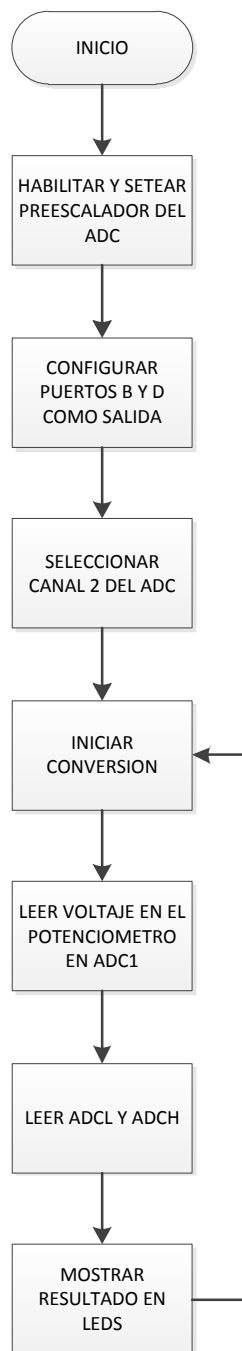


Figura 4.11. Diagrama de flujo del ADC con LEDS

4.-Descripción del algoritmo o estrategia utilizado

El microcontrolador utilizado es el Atmega169 que viene incluido en el Kit AVR Butterfly.

Primero se declaran las librerías:

- avr/io.h, que contiene la definición de los registros y de sus respectivos bits.
- #include <util/delay.h> es la que da los intervalos de tiempo en la compilación

Inicialización del programa, del adc y de su conversión.

```
void adc_init(void);
```

```
void adc_start_conversion(uint8_t);
```

```
void init(void);
```

```
//*****  
*****
```

```
// Inicialización del módulo ADC
```

```
//*****  
*****
```

```
void adc_init(void)
```

```
{  
//Selecciona un voltaje de referencia  
//AVCC with external capacitor at AREF pin  
ADMUX|=(0<<REFS1)|(1<<REFS0);  
//set prescaller and enable ADC  
ADCSRA|=(1<<ADEN)|(1<<ADIE);  
//enable ADC with dummy conversion  
}
```

```
//*****  
*****
```

```
// ADC Rutina de conversion simple
```

```
//*****  
*****
```

```
void adc_start_conversion(uint8_t channel)
```

```

{
//establecer canal ADC
ADMUX=(ADMUX&0xF0)|channel;
//Empezar conversion
ADCSRA |= (1<<ADSC)|(1<<ADIE);
}

//*****

// init AVR
//*****

void init(void)
{
    //Init ADC
    adc_init();
    DDRB=0xff; //puertos del atmega169 seran configurados como salidas
    DDRD=0xff; //puertos del atmega169 seran configurados como salidas
    PORTB=0X00;
    PORTD=0X00;
}

//*****

// run analog digital converter, timer.
//*****

int main(void)
{
    init();
    while(1)
    {
        adc_start_conversion(0);
    }
}

```

```

        _delay_ms(1);
        PORTB = ADCL;
        PORTD = ADCH;
    }
    return 0;
}

```

5.- Listado del programa fuente en lenguaje c con comentarios en las líneas de código que considere fundamentales

```

//*****
//          * * MICROCONTROLADORES AVANZADOS * *
//*****
//      Grupo # 5
//      Integrantes:          Grace Yagual
//                              Danilo Riera
//                              Miguel Rodriguez
//*****
//
//                      Programa # 1
//Programa que contiene una conversión sencilla y salidas en Puerto C y D
//Nombre del archivo: adc.c
//*****
*****

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

#define F_CPU 20000000
void adc_init(void);
void adc_start_conversion(uint8_t);
void init(void);

```

```

void adc_init(void)
{
    //Selecciona un voltaje de referencia
    //AVCC with external capacitor at AREF pin
    ADMUX|=(0<<REFS1)|(1<<REFS0);
    //set prescaller and enable ADC
    ADCSRA|=(1<<ADEN)|(1<<ADIE);
    //enable ADC with dummy conversion
}

void adc_start_conversion(uint8_t channel)
{
    //establecer canal ADC
    ADMUX=(ADMUX&0xF0)|channel;
    //Empezar conversion
    ADCSRA |= (1<<ADSC)|(1<<ADIE);
}

void init(void)
{
    adc_init();
    DDRB=0xff; //puertos del atmega169 seran configurados como salidas
    DDRE=0xff; //puertos del atmega169 seran configurados como salidas
    PORTB=0X00;
    PORTE=0X00;

}

int main(void)
{
    init();
    while(1)
    {

```

```

adc_start_conversion(2);

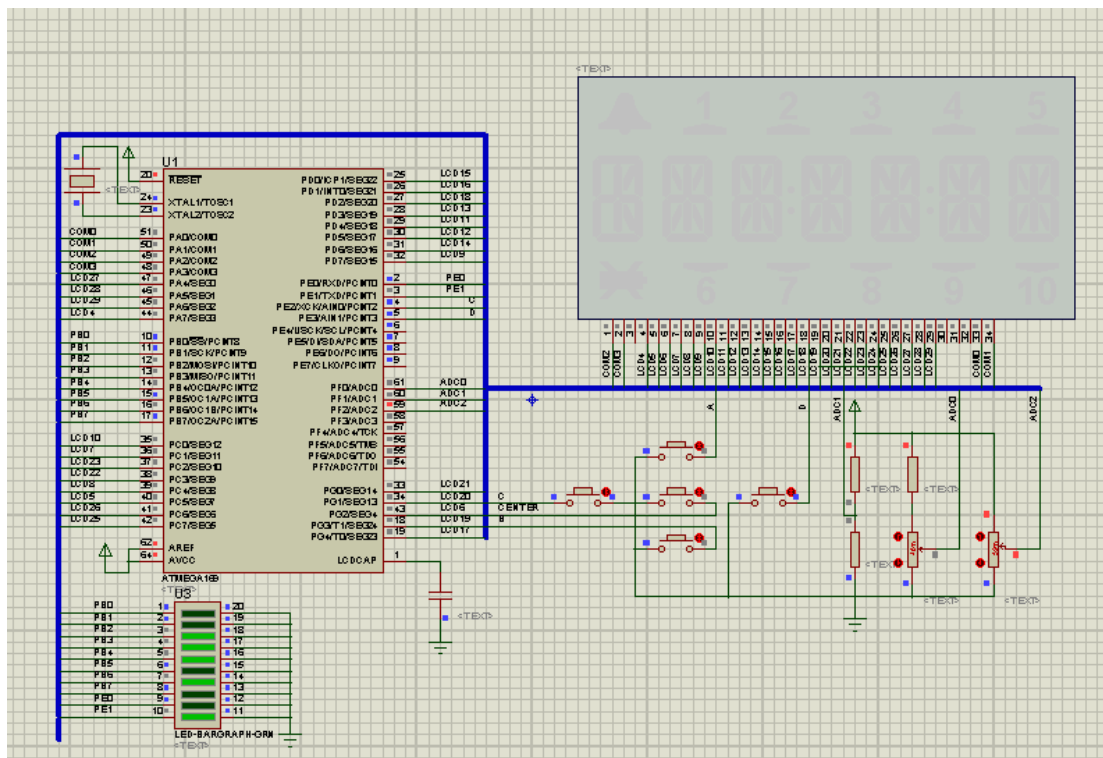
_delay_ms(1);

PORTB = ADCL;
PORTE = ADCH;
}

return 0;
}

```

6.- Copia impresa del circuito armado en PROTEUS para la simulación en el momento de su ejecución



4.4 LECTURA DE VOLTAJE

1.- Enunciado del proyecto

En el display del Avr Butterfly se mostrara un valor comprendido entre el 1 y el 9 y la letra V que representa el valor del voltaje el cual es regulado y fijado mediante un potenciómetro que esta ubicado en el ADC del microcontrolador.

2.- Diagrama de bloques

Los recursos del AVR Butterfly que se serán usados en el presente ejemplo se muestran en el diagrama de bloques a continuación.

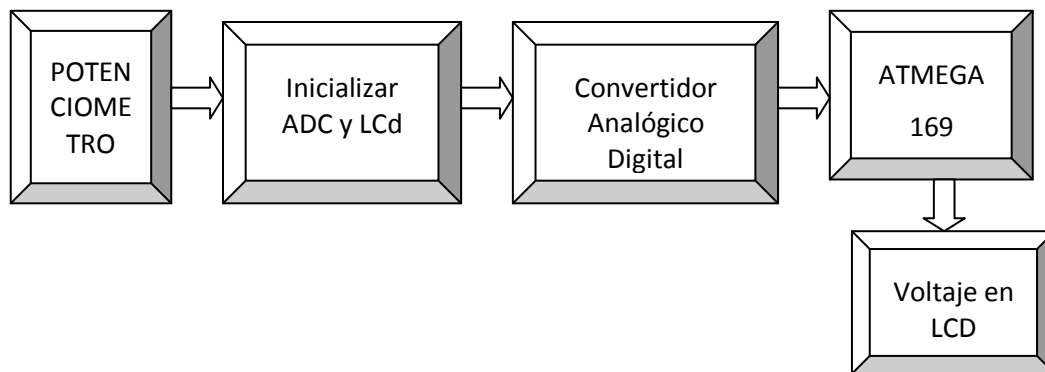


Figura 4.13. Diagrama de bloques de la lectura del voltaje

3.- Diagrama de flujo funcional del programa

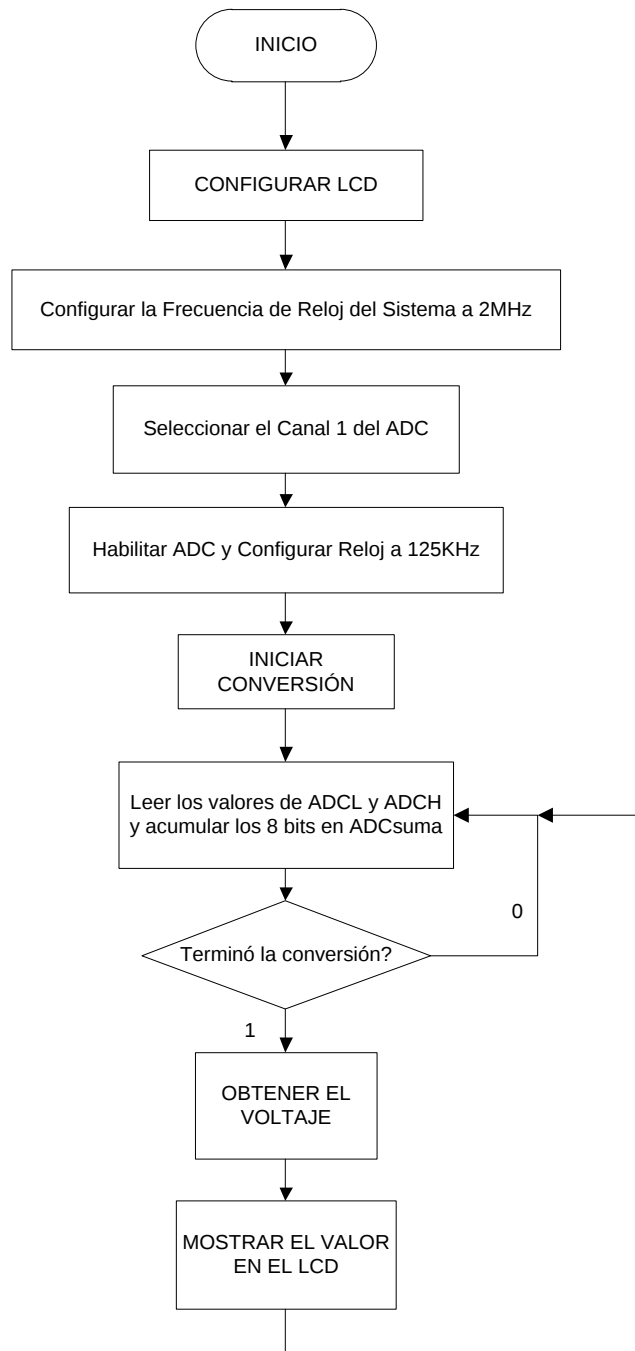


Figura 4.14. Diagrama de flujo de la lectura del voltaje

4.-Descripción del algoritmo o estrategia utilizado

Al inicio se incluirán los archivos de cabecera (#include):

- **avr/io.h**, que contiene la definición de los registros y de sus respectivos bits.
- **stdlib.h**, contiene macros y funciones básicas, esta librería es necesaria para poder utilizar la función itoa(), presente en el código, que transforma un valor entero en una cadena de caracteres.
- **avr/interrupt.h**, habilita las interrupciones en el programa
- **lcd.h**, permite que trabaje la pantalla LCD del AvrButterfly

A continuación se declararán las funciones necesarias para la conversión ADC y para el procesamiento de la información obtenida.

```
voidInicializar_ADC(void);
```

```
intleer_ADC(void);
```

```
voidgetVolt(void);
```

```
int main(void); inicialización del programa principal
```

```
    unsigned char i = 0;
```

```
    char *palabra="La lectura en voltios\0"; Texto que se ve de entrada en la  
pantalla LCD
```

```
        Inicializacion del ADC y LCD
```

```
        Inicializar_ADC();
```

```
        inicializar_LCD();
```

```
        sei();
```

```
        escribir_palabras_en_LCD(palabra);
```

```
        for(i=0;i<5;i++)_delay_loop_2(50000); en incrementos de 1 del 0 al 5 para las  
letras
```

```
        while(1); mientras el while este en 1
```

```
        SMCR=1; entrar en Sleep modo Idle
```

```
La función leer_ADC( ):
```

```
    intleer_ADC(void)
```

```
    {
```

```
    chari;
```

```
    intADC_temp;
```

```
    intADCsuma = 0;
```

Activa y energiza los sensores del AVR Butterfly, PORTF3 será la alimentación VCP para el Resistor NTC y para la LDR.

Configurar el pin 3 del Puerto F como salida:

```
PORTF |= (1<<PF3);
```

Configurar pin 3 del Puerto F a nivel alto para activar/energizar a los sensores:

```
DDRF |= (1<<DDF3);
```

Habilitar el ADC, por si otra aplicación lo deshabilita:

```
ADCSRA |= (1<<ADEN);
```

Activa y ordena la ejecución de una conversión ADC, se ejecutarán 8 conversiones.

```
ADCSRA |= (1<<ADSC);
```

Esperar hasta que el ADC indique que ha culminado una conversión, mediante el sondeo del indicador ADIF:

```
while(!(ADCSRA & 0x10));
```

Repetir ocho veces el proceso dentro del lazo FOR:

```
for(i=0;i<8;i++)
```

```
{
```

Activar/ordenar la ejecución de una conversión ADC:

```
ADCSRA |= (1<<ADSC);
```

Esperar hasta que el ADC indique que ha culminado una conversión, mediante el sondeo del indicador ADIF:

```
while(!(ADCSRA & 0x10));
```

Leer el byte bajo del Registro de Datos de 16 bits del ADC:

```
ADC_temp = ADCL;
```

Leer el byte alto del Registro de Datos de 16 bits del ADC:

```
ADC_temp += (ADCH << 8);
```

Acumular en ADCsuma los resultados de las 8 lecturas/conversiones ADC:

```
ADCsuma += ADC_temp;
```

```
}
```

Calcular el promedio de las lecturas ADC. Como son 8 lecturas, se debe dividir ADCsuma para 8, puesto que es lo mismo que desplazar el contenido de ADCsuma hacia la derecha 3 posiciones:

```
ADCsuma = ADCsuma >> 3;
```

Deshabilitar los Sensores quitándoles la alimentación de voltaje desde PORTF3:

```
PORTF &= ~(1<<PF3);
```

```
DDRF &= ~(1<<DDF3);
```

Deshabilitar el ADC:

```
ADCSRA &= ~(1<<ADEN);
```

Retornar el promedio de las lecturas/conversiones, retorna un valor de 16 bits (int):

```
return ADCsuma;
```

```
}
```

5.- Listado del programa fuente en lenguaje c con comentarios en las líneas de código que considere fundamentales

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdlib.h>
#include "lcd.h"

void Inicializar_ADC(void);
int leer_ADC(void);
void getVolt(void);
int main(void)
{
    unsigned char i = 0;
    char *palabra="La lectura en voltios\0";
    Inicializar_ADC();
    inicializar_LCD();
    sei();
    escribir_palabras_en_LCD(palabra);
    for(i=0;i<5;i++)_delay_loop_2(50000);
    while(1)
    {
        getVolt();
        SMCR=1;    // entrar en Sleep modo Idle
    }
    return 0;
}

void Inicializar_ADC(void)
{
    CLKPR = (1<<CLKPCE);
```

```

        CLKPR = (1<<CLKPS1);
        ADMUX = 1;
        ADCSRA = (1<<ADEN)|(1<<ADPS2);
        leer_ADC();
    }
    intleer_ADC(void)
    {
        chari;
        intADC_temp;
        intADCsuma = 0;
        PORTF |= (1<<PF3);
        DDRF |= (1<<DDF3);
        ADCSRA |= (1<<ADEN);
        ADCSRA |= (1<<ADSC);
        while(!(ADCSRA & 0x10));
        for(i=0;i<8;i++)
        {
            ADCSRA |= (1<<ADSC);
            while(!(ADCSRA & 0x10));
            ADC_temp = ADCL;
            ADC_temp += (ADCH << 8);
            ADCsuma += ADC_temp;
        }
        ADCsuma = (ADCsuma>> 3);
        PORTF&=~(1<<PF3);
        DDRF&=~(1<<DDF3);
        ADCSRA&=~(1<<ADEN);
        returnADCsuma;
    }

```

```

void getVolt()
{
    unsigned char j = 0;
    char voltintpart[] = {'0','0','0','\0'};
    char voltfractpart[] = {'0','0','0','\0'};
    intintpart = 0;
    intfractpart = 0;
    intADCresult = 0;

    ADCresult = leer_ADC();
    intpart = ADCresult/62;
    fractpart = ADCresult%62;

    itoa(intpart, voltintpart, 10);
    itoa(fractpart, voltfractpart, 10);

    escribir_caracter_en_LCD(voltintpart[0],0);
    escribir_caracter_en_LCD(',',1);
    escribir_caracter_en_LCD(voltfractpart[0],2);
    escribir_caracter_en_LCD(voltfractpart[1],3);
    escribir_caracter_en_LCD('V',4);
    actualizar_LCD();
    for(j=0;j<5;j++)_delay_loop_2(50000);

}

```

6.- Copia impresa del circuito armado en PROTEUS para la simulación en el momento de su ejecución

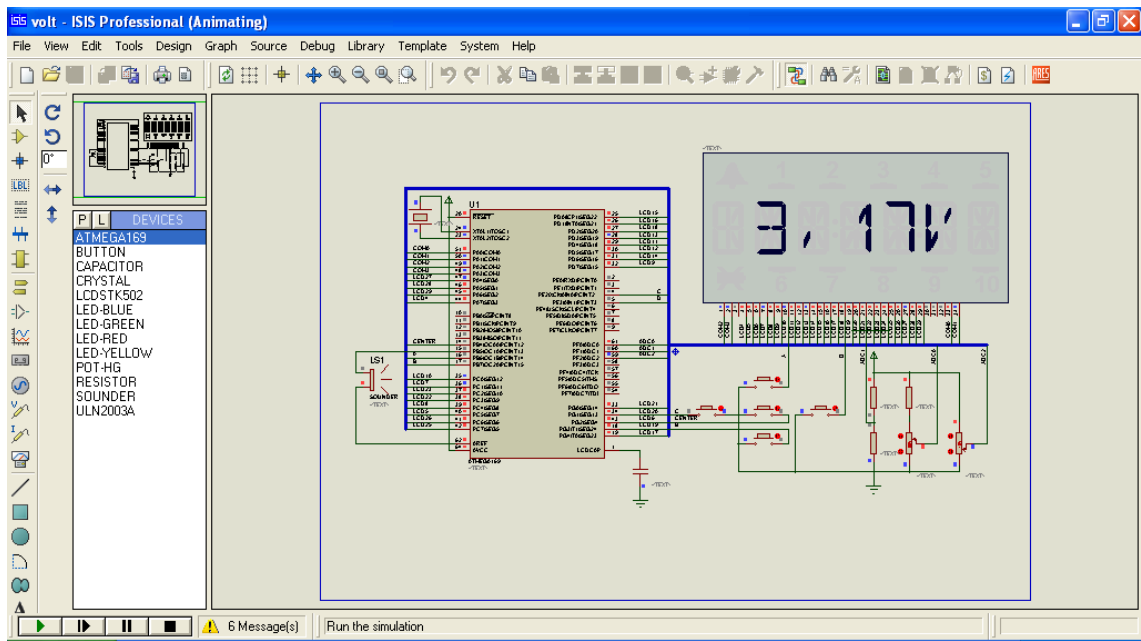


Figura 4.15. Simulación en Proteus de la medición de voltaje

4.5 MEDIDOR DE LUZ

1.- Enunciado del proyecto

Variando la resistencia en el LDR, aumentando y disminuyendo su valor cambiara en la pantalla del LCD el valor de un numero hexadecimal; esto por medio de la entrada del ADC0 donde normalmente estamos programando nuestros programas.

2.- Diagrama de bloques

Los recursos del AVR Butterfly que se serán usados en el presente ejemplo se muestran en el diagrama de bloques a continuación.

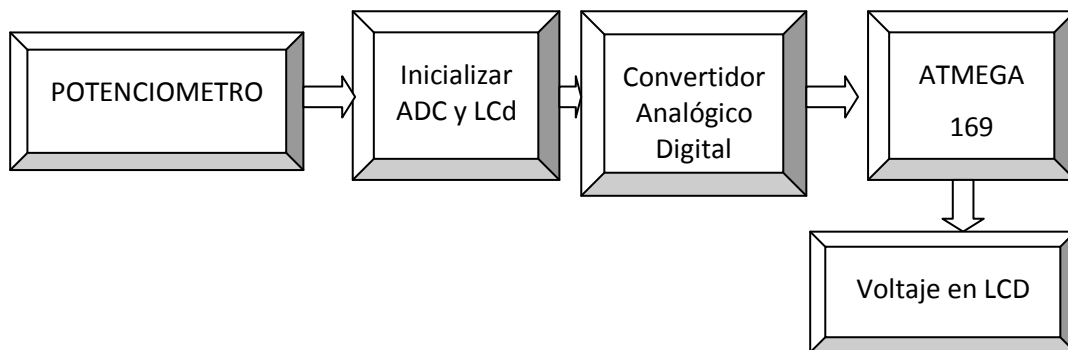


Figura 4.16. Diagrama de bloques de la medición de luz

3.- Diagrama de flujo funcional del programa

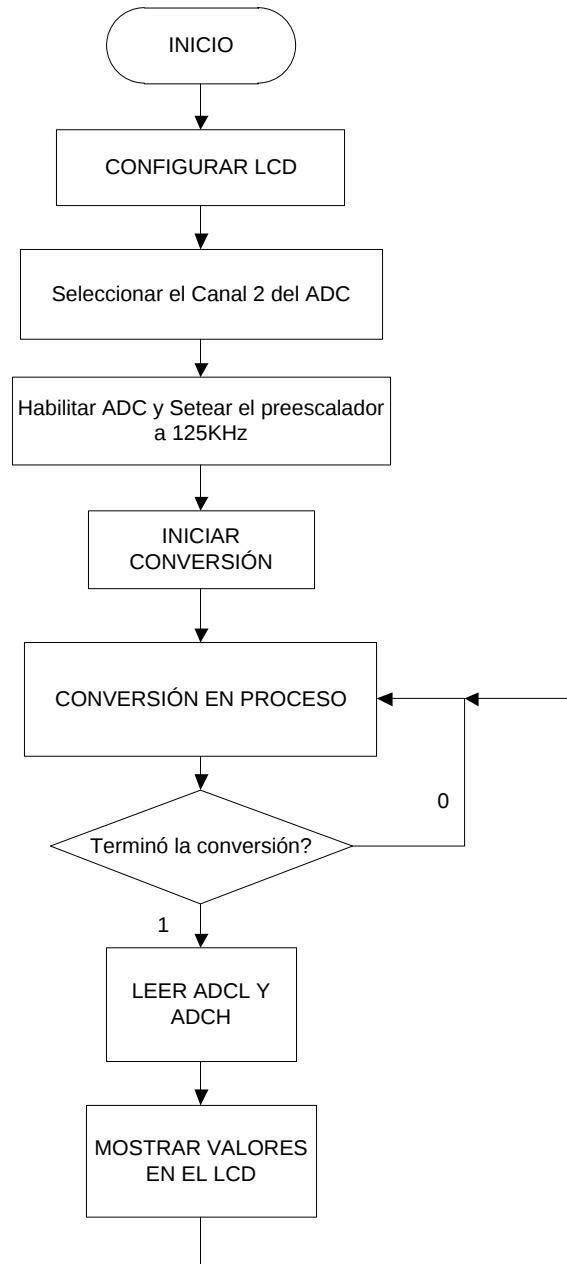


Figura 4.17. Diagrama de flujo de la medición de luz

4.-Descripción del algoritmo o estrategia utilizado

.include "m169def.inc"; **librería del avr atmel 169 para que compile el programa respectivo**

.org 0x0000

rjmp INICIO; **salta a la función INICIO que empieza el programa principal**

INICIO:

ldi r16, (1<<LCDCS) | (1<<LCDMUX1) | (1<<LCDMUX0) | (1<<LCDPM2) |
(1<<LCDPM1) | (1<<LCDPM0)

sts LCDCRB, r16

ldi r16, (1<<LCDPS0); **cargar en r16**

sts LCDFRR, r16

ldi r16, 0x0F

sts LCDCCR, r16

ldi r16, (1<<LCDEN)

sts LCDCRA, r16

; 0x5559, // '0' ; **reconoce el numero respectivo**

; 0x0118, // '1'

; 0x0f51, // 'A' (+ 'a'); **reconoce la letra codigo respectiva**

; 0x3991, // 'B' (+ 'b')

5.- Listado del programa fuente en lenguaje c con comentarios en las líneas de código que considere fundamentales

```
//*****  
*****
```

```
//                                * * MICROCONTROLADORES AVANZADOS * *
```

```
//*****  
*****
```

```
//      Grupo # 5
```

```
//      Integrantes:                      Grace Yagual
```

```

//                                     Danilo Riera
//                                     Miguel Rodriguez
//*****
//*****

//
//                                     * * CONVERTIDOR ANALÓGICO DIGITAL * *
//                                     ADC
//
// Nombre del archivo: light.c
//*****

.include "m169def.inc"

.org 0x0000
rjmp INICIO
INICIO:
    ldi r16, (1<<LCDCS) | (1<<LCDMUX1) | (1<<LCDMUX0) | (1<<LCDPM2) |
(1<<LCDPM1) | (1<<LCDPM0)
    sts LCDCRB, r16
    ldi r16, (1<<LCDPS0)
    sts LCDFRR, r16
    ldi r16, 0x0F
    sts LCDCCR, r16
    ldi r16, (1<<LCDEN)
    sts LCDCRA, r16
; 0x5559, // '0'
; 0x0118, // '1'
; 0x1e11, // '2'
; 0x1b11, // '3'
; 0x0b50, // '4'
; 0x1b41, // '5'
; 0x1f41, // '6'

```

```

; 0x0111, // '7
; 0x1f51, // '8
; 0x1b51, // '9'
; 0x0000, // ':' (Not defined)
; 0x0000, // ';' (Not defined)
; 0x0000, // '<' (Not defined)
; 0x0000, // '=' (Not defined)
; 0x0000, // '>' (Not defined)
; 0x0000, // '?' (Not defined)
; 0x0000, // '@' (Not defined)
; 0x0f51, // 'A' (+ 'a')
; 0x3991, // 'B' (+ 'b')
; 0x1441, // 'C' (+ 'c')
; 0x3191, // 'D' (+ 'd')
; 0x1e41, // 'E' (+ 'e')
; 0x0e41, // 'F' (+ 'f')
ldi r16, 0x02 ;input 2, light
    sts ADMUX, r16
; delay for a bit
    ldi r16, 50
dela:
    dec r16
    cpi r16, 0x0
brne dela
    ldi r16, (1<<ADEN) | (1<<ADPS1) | (1<<ADPS0) ;set ADC prescaler to , 1MHz / 8 =
125kHz
    sts ADCSRA, r16
    sbi PORTF, PORTF3
    sbi DDRF, PORTF3

```

IniciarConversion:

```
ldi r16, (1<<ADSC) | (1<<ADEN) | (1<<ADPS1) | (1<<ADPS0) ;set ADC prescaler to  
, 1MHz / 8 = 125kHz
```

```
sts ADCSRA, r16
```

Conversion:

```
lds r16, ADCSRA
```

```
andi r16, (1<<ADIF)
```

```
breq Conversion
```

```
lds r19, ADCL ;Se leerá el ADCL antes del ADCH
```

```
lds r18, ADCH
```

```
mov ZL, r18
```

```
ldi ZH, 0x04
```

```
lpm r20,Z
```

```
ldi ZH, 0x06
```

```
lpm r21,Z
```

```
ldi ZH, 0x08
```

```
lpm r22,Z
```

```
ldi ZH, 0x0A
```

```
lpm r23,Z
```

```
sts LCDDR0,r20
```

```
sts LCDDR5,r21
```

```
sts LCDDR10,r22
```

```
sts LCDDR15,r23
```

```
mov ZL, r19
```

```
ldi ZH, 0x04
```

```
lpm r20,Z
```

```
ldi ZH, 0x06
```

```
lpm r21,Z
```

```
ldi ZH, 0x08
```

```

    lpm r22,Z
    ldi ZH, 0x0A
    lpm r23,Z
    sts LCDDR1,r20
    sts LCDDR6,r21
    sts LCDDR11,r22
    sts LCDDR16,r23
    rjmp IniciarConversion
cicl:
    rjmp cicl
;a continuación un metodo forzado para la conversión a segmentos del LCD
.org 0x0200
.db 0x99, 0x89, 0x19, 0x19
.db 0x09, 0x19, 0x19, 0x19
.db 0x19, 0x19, 0x19, 0x19
.db 0x19, 0x19, 0x19, 0x19
.db 0x98, 0x88, 0x18, 0x18
.db 0x08, 0x18, 0x18, 0x18
.db 0x18, 0x18, 0x18, 0x18
.db 0x18, 0x18, 0x18, 0x18
.db 0x91, 0x81, 0x11, 0x11
.db 0x01, 0x11, 0x11, 0x11
.db 0x11, 0x11, 0x11, 0x11
.db 0x11, 0x11, 0x11, 0x11
.db 0x91, 0x81, 0x11, 0x11
.db 0x01, 0x11, 0x11, 0x11
.db 0x11, 0x11, 0x11, 0x11
.db 0x11, 0x11, 0x11, 0x11
.db 0x90, 0x80, 0x10, 0x10
.db 0x00, 0x10, 0x10, 0x10

```

.db 0x10, 0x10, 0x10, 0x10
.db 0x10, 0x10, 0x10, 0x10
.db 0x91, 0x81, 0x11, 0x11
.db 0x01, 0x11, 0x11, 0x11
.db 0x11, 0x11, 0x11, 0x11
.db 0x11, 0x11, 0x11, 0x11
.db 0x91, 0x81, 0x11, 0x11
.db 0x01, 0x11, 0x11, 0x11
.db 0x11, 0x11, 0x11, 0x11
.db 0x11, 0x11, 0x11, 0x11
.db 0x91, 0x81, 0x11, 0x11
.db 0x01, 0x11, 0x11, 0x11
.db 0x11, 0x11, 0x11, 0x11
.db 0x11, 0x11, 0x11, 0x11
.db 0x91, 0x81, 0x11, 0x11
.db 0x01, 0x11, 0x11, 0x11
.db 0x11, 0x11, 0x11, 0x11
.db 0x11, 0x11, 0x11, 0x11
.db 0x91, 0x81, 0x11, 0x11
.db 0x01, 0x11, 0x11, 0x11
.db 0x11, 0x11, 0x11, 0x11
.db 0x11, 0x11, 0x11, 0x11
.db 0x91, 0x81, 0x11, 0x11
.db 0x01, 0x11, 0x11, 0x11
.db 0x11, 0x11, 0x11, 0x11
.db 0x11, 0x11, 0x11, 0x11
.db 0x91, 0x81, 0x11, 0x11
.db 0x01, 0x11, 0x11, 0x11
.db 0x11, 0x11, 0x11, 0x11
.db 0x11, 0x11, 0x11, 0x11

```
.db 0x91, 0x81, 0x11, 0x11
.db 0x01, 0x11, 0x11, 0x11
.db 0x11, 0x11, 0x11, 0x11
.db 0x11, 0x11, 0x11, 0x11
.db 0x91, 0x81, 0x11, 0x11
.db 0x01, 0x11, 0x11, 0x11
.db 0x11, 0x11, 0x11, 0x11
.db 0x11, 0x11, 0x11, 0x11
.db 0x91, 0x81, 0x11, 0x11
.db 0x01, 0x11, 0x11, 0x11
.db 0x11, 0x11, 0x11, 0x11
.db 0x11, 0x11, 0x11, 0x11
.db 0x91, 0x81, 0x11, 0x11
.db 0x01, 0x11, 0x11, 0x11
.db 0x11, 0x11, 0x11, 0x11
.db 0x11, 0x11, 0x11, 0x11
```

```
.org 0x0300
.db 0x55, 0x15, 0x15, 0x15
.db 0x55, 0x45, 0x45, 0x15
.db 0x55, 0x55, 0x55, 0x95
.db 0x45, 0x95, 0x45, 0x45
.db 0x51, 0x11, 0x11, 0x11
.db 0x51, 0x41, 0x41, 0x11
.db 0x51, 0x51, 0x51, 0x91
.db 0x41, 0x91, 0x41, 0x41
.db 0x51, 0x11, 0x11, 0x11
.db 0x51, 0x41, 0x41, 0x11
.db 0x51, 0x51, 0x51, 0x91
.db 0x41, 0x91, 0x41, 0x41
```

.db 0x51, 0x11, 0x11, 0x11
.db 0x51, 0x41, 0x41, 0x11
.db 0x51, 0x51, 0x51, 0x91
.db 0x41, 0x91, 0x41, 0x41
.db 0x55, 0x15, 0x15, 0x15
.db 0x55, 0x45, 0x45, 0x15
.db 0x55, 0x55, 0x55, 0x95
.db 0x45, 0x95, 0x45, 0x45
.db 0x54, 0x14, 0x14, 0x14
.db 0x54, 0x44, 0x44, 0x14
.db 0x54, 0x54, 0x54, 0x94
.db 0x44, 0x94, 0x44, 0x44
.db 0x54, 0x14, 0x14, 0x14
.db 0x54, 0x44, 0x44, 0x14
.db 0x54, 0x54, 0x54, 0x94
.db 0x44, 0x94, 0x44, 0x44
.db 0x51, 0x11, 0x11, 0x11
.db 0x51, 0x41, 0x41, 0x11
.db 0x51, 0x51, 0x51, 0x91
.db 0x41, 0x91, 0x41, 0x41
.db 0x55, 0x15, 0x15, 0x15
.db 0x55, 0x45, 0x45, 0x15
.db 0x55, 0x55, 0x55, 0x95
.db 0x45, 0x95, 0x45, 0x45
.db 0x55, 0x15, 0x15, 0x15
.db 0x55, 0x45, 0x45, 0x15
.db 0x55, 0x55, 0x55, 0x95
.db 0x45, 0x95, 0x45, 0x45
.db 0x55, 0x15, 0x15, 0x15
.db 0x55, 0x45, 0x45, 0x15

```
.db 0x55, 0x55, 0x55, 0x95
.db 0x45, 0x95, 0x45, 0x45
.db 0x59, 0x19, 0x19, 0x19
.db 0x59, 0x49, 0x49, 0x19
.db 0x59, 0x59, 0x59, 0x99
.db 0x49, 0x99, 0x49, 0x49
.db 0x54, 0x14, 0x14, 0x14
.db 0x54, 0x44, 0x44, 0x14
.db 0x54, 0x54, 0x54, 0x94
.db 0x44, 0x94, 0x44, 0x44
.db 0x59, 0x19, 0x19, 0x19
.db 0x59, 0x49, 0x49, 0x19
.db 0x59, 0x59, 0x59, 0x99
.db 0x49, 0x99, 0x49, 0x49
.db 0x54, 0x14, 0x14, 0x14
.db 0x54, 0x44, 0x44, 0x14
.db 0x54, 0x54, 0x54, 0x94
.db 0x44, 0x94, 0x44, 0x44
.db 0x54, 0x14, 0x14, 0x14
.db 0x54, 0x44, 0x44, 0x14
.db 0x54, 0x54, 0x54, 0x94
.db 0x44, 0x94, 0x44, 0x44
```

```
.org 0x0400
```

```
.db 0x55, 0x15, 0xE5, 0xB5
.db 0xB5, 0xB5, 0xF5, 0x15
.db 0xF5, 0xB5, 0xF5, 0x95
.db 0x45, 0x15, 0xE5, 0xE5
.db 0x51, 0x11, 0xE1, 0xB1
.db 0xB1, 0xB1, 0xF1, 0x11
```

.db 0xF1, 0xB1, 0xF1, 0x91
.db 0x41, 0x11, 0xE1, 0xE1
.db 0x5E, 0x1E, 0xEE, 0xBE
.db 0xBE, 0xBE, 0xFE, 0x1E
.db 0xFE, 0xBE, 0xFE, 0x9E
.db 0x4E, 0x1E, 0xEE, 0xEE
.db 0x5B, 0x1B, 0xEB, 0xBB
.db 0xBB, 0xBB, 0xFB, 0x1B
.db 0xFB, 0xBB, 0xFB, 0x9B
.db 0x4B, 0x1B, 0xEB, 0xEB
.db 0x5B, 0x1B, 0xEB, 0xBB
.db 0xBB, 0xBB, 0xFB, 0x1B
.db 0xFB, 0xBB, 0xFB, 0x9B
.db 0x4B, 0x1B, 0xEB, 0xEB
.db 0x5B, 0x1B, 0xEB, 0xBB
.db 0xBB, 0xBB, 0xFB, 0x1B
.db 0xFB, 0xBB, 0xFB, 0x9B
.db 0x4B, 0x1B, 0xEB, 0xEB
.db 0x5F, 0x1F, 0xEF, 0xBF
.db 0xBF, 0xBF, 0xFF, 0x1F
.db 0xFF, 0xBF, 0xFF, 0x9F
.db 0x4F, 0x1F, 0xEF, 0xEF
.db 0x51, 0x11, 0xE1, 0xB1
.db 0xB1, 0xB1, 0xF1, 0x11
.db 0xF1, 0xB1, 0xF1, 0x91
.db 0x41, 0x11, 0xE1, 0xE1
.db 0x5F, 0x1F, 0xEF, 0xBF
.db 0xBF, 0xBF, 0xFF, 0x1F
.db 0xFF, 0xBF, 0xFF, 0x9F
.db 0x4F, 0x1F, 0xEF, 0xEF

```

.db 0x5B, 0x1B, 0xEB, 0xBB
.db 0xBB, 0xBB, 0xFB, 0x1B
.db 0xFB, 0xBB, 0xFB, 0x9B
.db 0x4B, 0x1B, 0xEB, 0xEB
.db 0x5F, 0x1F, 0xEF, 0xBF
.db 0xBF, 0xBF, 0xFF, 0x1F
.db 0xFF, 0xBF, 0xFF, 0x9F
.db 0x4F, 0x1F, 0xEF, 0xEF
.db 0x59, 0x19, 0xE9, 0xB9
.db 0xB9, 0xB9, 0xF9, 0x19
.db 0xF9, 0xB9, 0xF9, 0x99
.db 0x49, 0x19, 0xE9, 0xE9
.db 0x54, 0x14, 0xE4, 0xB4
.db 0xB4, 0xB4, 0xF4, 0x14
.db 0xF4, 0xB4, 0xF4, 0x94
.db 0x44, 0x14, 0xE4, 0xE4
.db 0x51, 0x11, 0xE1, 0xB1
.db 0xB1, 0xB1, 0xF1, 0x11
.db 0xF1, 0xB1, 0xF1, 0x91
.db 0x41, 0x11, 0xE1, 0xE1
.db 0x5E, 0x1E, 0xEE, 0xBE
.db 0xBE, 0xBE, 0xFE, 0x1E
.db 0xFE, 0xBE, 0xFE, 0x9E
.db 0x4E, 0x1E, 0xEE, 0xEE
.db 0x5E, 0x1E, 0xEE, 0xBE
.db 0xBE, 0xBE, 0xFE, 0x1E
.db 0xFE, 0xBE, 0xFE, 0x9E
.db 0x4E, 0x1E, 0xEE, 0xEE

.org 0x0500

```

.db 0x55, 0x05, 0x15, 0x15
.db 0x05, 0x15, 0x15, 0x05
.db 0x15, 0x15, 0x05, 0x35
.db 0x15, 0x35, 0x15, 0x05
.db 0x50, 0x00, 0x10, 0x10
.db 0x00, 0x10, 0x10, 0x00
.db 0x10, 0x10, 0x00, 0x30
.db 0x10, 0x30, 0x10, 0x00
.db 0x51, 0x01, 0x11, 0x11
.db 0x01, 0x11, 0x11, 0x01
.db 0x11, 0x11, 0x01, 0x31
.db 0x11, 0x31, 0x11, 0x01
.db 0x51, 0x01, 0x11, 0x11
.db 0x01, 0x11, 0x11, 0x01
.db 0x11, 0x11, 0x01, 0x31
.db 0x11, 0x31, 0x11, 0x01
.db 0x50, 0x00, 0x10, 0x10
.db 0x00, 0x10, 0x10, 0x00
.db 0x10, 0x10, 0x00, 0x30
.db 0x10, 0x30, 0x10, 0x00
.db 0x51, 0x01, 0x11, 0x11
.db 0x01, 0x11, 0x11, 0x01
.db 0x11, 0x11, 0x01, 0x31
.db 0x11, 0x31, 0x11, 0x01
.db 0x51, 0x01, 0x11, 0x11
.db 0x01, 0x11, 0x11, 0x01
.db 0x11, 0x11, 0x01, 0x31
.db 0x11, 0x31, 0x11, 0x01
.db 0x50, 0x00, 0x10, 0x10
.db 0x00, 0x10, 0x10, 0x00

.db 0x10, 0x10, 0x00, 0x30
.db 0x10, 0x30, 0x10, 0x00
.db 0x51, 0x01, 0x11, 0x11
.db 0x01, 0x11, 0x11, 0x01
.db 0x11, 0x11, 0x01, 0x31
.db 0x11, 0x31, 0x11, 0x01
.db 0x51, 0x01, 0x11, 0x11
.db 0x01, 0x11, 0x11, 0x01
.db 0x11, 0x11, 0x01, 0x31
.db 0x11, 0x31, 0x11, 0x01
.db 0x50, 0x00, 0x10, 0x10
.db 0x00, 0x10, 0x10, 0x00
.db 0x10, 0x10, 0x00, 0x30
.db 0x10, 0x30, 0x10, 0x00
.db 0x53, 0x03, 0x13, 0x13
.db 0x03, 0x13, 0x13, 0x03
.db 0x13, 0x13, 0x03, 0x33
.db 0x13, 0x33, 0x13, 0x03
.db 0x51, 0x01, 0x11, 0x11
.db 0x01, 0x11, 0x11, 0x01
.db 0x11, 0x11, 0x01, 0x31
.db 0x11, 0x31, 0x11, 0x01
.db 0x53, 0x03, 0x13, 0x13
.db 0x03, 0x13, 0x13, 0x03
.db 0x13, 0x13, 0x03, 0x33
.db 0x13, 0x33, 0x13, 0x03
.db 0x51, 0x01, 0x11, 0x11
.db 0x01, 0x11, 0x11, 0x01
.db 0x11, 0x11, 0x01, 0x31
.db 0x11, 0x31, 0x11, 0x01

.db 0x50, 0x00, 0x10, 0x10

.db 0x00, 0x10, 0x10, 0x00

.db 0x10, 0x10, 0x00, 0x30

.db 0x10, 0x30, 0x10, 0x00

6.- Copia impresa del circuito armado en PROTEUS

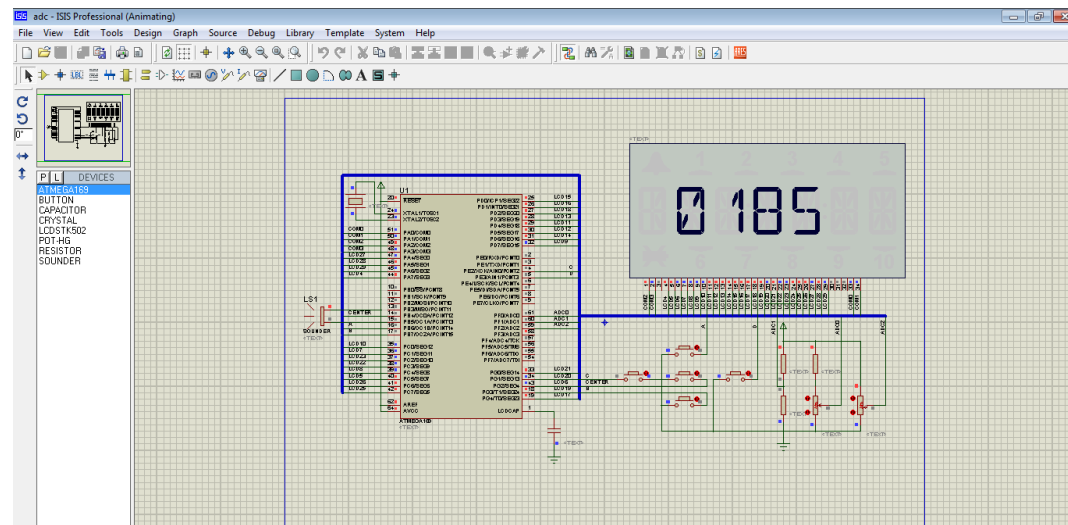


Figura 4.18. Simulación en Proteus de medición de luz (a)

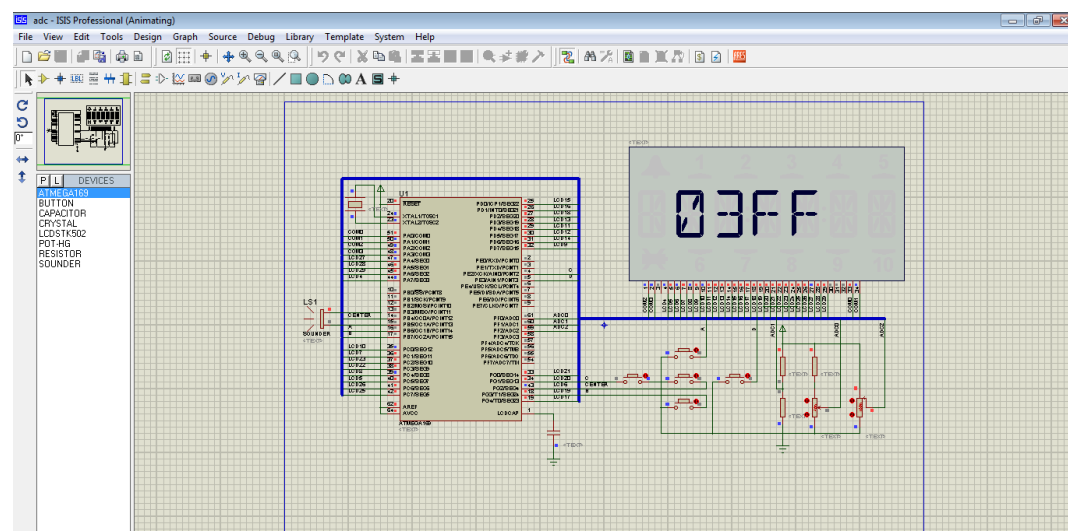


Figura 4.19. Simulación en Proteus de medición de luz (b)

CONCLUSIONES

1.- La plataforma de trabajo propuesta incluyendo el Kit AVR Butterfly es una poderosa herramienta de aprendizaje, que dispone de componentes importantes para los ejemplos propuestos, entre ellos contiene en especial los sensores de temperatura (NTC) y de luz (LDR), que nos permitieron obtener resultados satisfactorios en el desarrollo de los diferentes ejercicios que mencionamos, con los cuales aprendimos el funcionamiento del ADC y descubrimos progresivamente las características del microcontrolador ATmega169.

2.-Se consiguió un ahorro de tiempo y recursos gracias al entorno de programación AVR Studio y a la interfaz de simulación PROTEUS, que nos permitieron tener un enfoque de lo real antes de realizar la implementación física y las respectivas pruebas con el hardware, además de realizar los cambios que fueran necesarios para mejorar las aplicaciones.

3.-El lenguaje C es óptimo en el desarrollo de software para microcontroladores, y nos ha facilitado la programación del módulo AVR Butterfly, ya que podemos segmentar el código fuente en varias funciones especializadas, permitiéndonos volver a usar las mismas funciones en otras aplicaciones. Además, este lenguaje permite codificar más ordenadamente y al ser un lenguaje de medio nivel, es ideal para desarrollar aplicaciones de ingeniería.

RECOMENDACIONES

1.-No apoyar el Kit AVR Butterfly en superficies conductivas tales como metal, líquidos, etc., puesto que podrían causar daños en el mismo.

Colocar Headers fijos tal como se muestra el Capítulo 3 pues conectar cables directamente en los espacios para conexiones externas del Kit podría provocar un cortocircuito.

2.-Reemplazar la batería que incluye el fabricante en el Kit AVR Butterfly, por una fuente externa de voltaje de 3 V. Esto permitirá desarrollar con confianza las aplicaciones, y nos evita posibles errores que por batería descargada confunden al usuario.

3.-En el proyecto del sensor de temperatura, tener en cuenta que el rango de temperatura es de -10°C a 60°C, para no tener inconvenientes con el ambiente del sensor y que este no sufra problema alguno. En el proyecto con la LDR si queremos la máxima resistencia posible, el ambiente de trabajo debe estar completamente oscuro, y viceversa.

4.-Al momento de programar en el Avr Studio 4, seguir todos los pasos recomendados para no perder continuidad de la compilación y el programa se cargue sin ningún problema

BIBLIOGRAFÍA

[1] Joe Pardue, C Programming for Microcontrollers, Featuring ATMEL's AVR Butterfly and the Free WinAVR Compiler , Edición 2005

Fecha de consulta: 20/10/11.

[2] Barry B. Brey, Los microprocesadores de Intel: arquitectura, programación e interfaces, Editorial Prentice Hall, 7ma edición, fecha de consulta Octubre del 2011.

[3] Gerhard Schmidt, Beginners Introduction to the Assembly Language of ATMEL AVR Microprocessors, Web: <http://www.avr-asm-tutorial.net>, Abril 2009, fecha de consulta Octubre del 2011.

[4] AVR Instruction Set,
http://atmel.com/dyn/resources/prod_documents/doc0856.pdf, fecha de consulta octubre del 2011.

[5] AVR Butterfly User Guide,
http://www.atmel.com/dyn/resources/prod_documents/doc4271.pdf , fecha de consulta octubre del 2011.

[6] Hoja de especificaciones del microcontrolador ATMEGA169,
http://www.atmel.com/dyn/resources/prod_documents/doc2514.pdf?&MMN_position=43:43 , fecha de consulta octubre del 2011