



**ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL**  
**FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN**

“RELEVANCIA DEL TMR0 DEL MICRO ATMEGA 169 EN EL KIT AVR  
BUTTERFLY”

**TESINA DE SEMINARIO**

Previa la obtención del Título de:

**INGENIERO EN ELECTRICIDAD**

**ESPECIALIZACIÓN EN ELECTRÓNICA Y AUTOMATIZACIÓN  
INDUSTRIAL**

Presentado por:

Patricio Javier Salazar Moreira

Luis Alfredo Ortiz Moran

GUAYAQUIL – ECUADOR

AÑO 2011-2012

# AGRADECIMIENTO

A Dios.

A nuestra familia

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

A todos quienes apuestan por el desarrollo tecnológico en Ecuador.

Al Ing. Carlos Valdivieso.

A las personas que en conjunto forman a la ESPOL por brindarnos la oportunidad de adquirir conocimientos y formarnos profesionalmente.

# DEDICATORIA

A Dios, nuestro creador por su amor y protección.

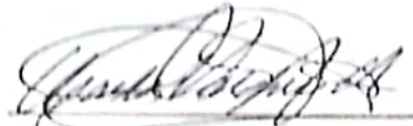
A nuestros padres, por sus sacrificios y desvelos, por los valores que nos inculcaron, por el ejemplo de tenacidad y perseverancia para alcanzar la meta anhelada.

A nuestra familia, por su afecto incondicional, estímulo permanente, apoyo y comprensión.

A nuestros maestros, por su paciencia, bondad y generosidad por iluminar nuestra senda con la luz del conocimiento, por enseñarnos a ser útiles a los demás.

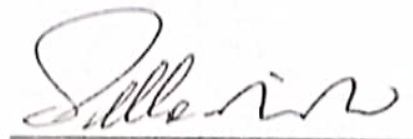
A nuestros amigos y compañeros, por la alegría compartida, por ser espíritu de solidaridad, por animarnos constantemente hacia la culminación de nuestra carrera.

# TRIBUNAL DE SUSTENTACION



Ing. Carlos Valdivieso A.

Profesor del Seminario de Graduación



Ing. Hugo Villavicencio V.

Delegado del Decano

# DECLARACIÓN EXPRESA

“La responsabilidad del contenido de este trabajo, nos corresponde exclusivamente; y el patrimonio intelectual del mismo a la ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL”.

(Reglamento de exámenes y títulos profesionales de la ESPOL)



Patricio Javier Salazar Moreira



Luis Alfredo Ortiz Moran

# RESUMEN

El objetivo de este proyecto de tesis es lograr la implementación del kit Butterfly AVR, para el cual estudiamos todos los registros asociados al TMR0 para implementar este proyecto de grado además entenderemos el funcionamiento de este para el microcontrolador Atmega 169 que se encuentra en el AVR Butterfly.

Así como también podremos aplicar nuestros conocimientos en el software y hardware que constaran en cinco ejercicios, tres de ellos realizados en el lenguaje C y los otros dos restantes en lenguaje de programación assembler.

Mostraremos el uso correcto para poder configurar el TMR0 en sus diversas formas tanto como desbordamiento del TRM0, por modo de comparación y también en sus diferentes formas de poder realizar el PWM.

En cada uno de los ejercicios mostraremos su teoría respectiva, además su implementación en el hardware y presentaremos sus códigos para un fácil entendimiento.

# ÍNDICE GENERAL

CAPÍTULO 1 .....	1
1. – INTRODUCCION AL CAPITULO.....	1
1.1.- DESCRIPCIÓN GENERAL DEL TIMER0 .....	2
1.1.1. TCCR0 – TIMER/COUNTER REGISTRO DE CONTROL.....	2
1.1.2. TCNT0 – TIMER/COUNTER REGISTRO. ....	4
1.1.3. TIMSK– TIMER / COUNTER REGISTRO DE ENMASCARAMIENTO DE INTERRUPCIONES.....	4
1.1.4. TIFR – TIMER/COUNTER REGISTRO DE BANDERAS DE..... INTERRUPCIONES.....	5
1.1.5. TCCR0A-TIMER /COUNTER REGISTRO DE CONTROL .....	5
A .....	5
1.2.-TIMER 0 EN AVR DESCRIPCIÓN COMPLETA. ....	9
1.2.1.-DIAGRAMA DE BLOQUES Y EXPLICACION.....	9
1.2.2.- MODOS DE OPERACION .....	10
1.2.2.1.- MODO NORMAL PWM .....	11
1.2.2.2.- MODO DE COMPARACION DE LIMPIEZA DEL TIMER ....	11
(CTC) .....	11

1.2.2.3.- MODO RAPIDO PWM. ....	12
1.2.2.4.- MODO PWM DE FASE CORRECTA. ....	16
1.2.3.- CARACTERISTICAS DEL TIMER0 EN LA FAMILIA .....	19
MICROCHIP .....	19
CAPÍTULO 2 .....	1
2.1.- EL MICROCONTROLADOR ATMEL.....	1
2.2.- PROTEUS ISIS.....	18
2.2.1.- ISIS (Intelligent Schematic Input System): .....	18
2.2.2.- ARES (Advanced Routing Modelling):.....	18
2.2.3.- VSM (Virtual System Modelling):.....	19
2.3.- AVR BUTTERFLY .....	20
2.4.- ATmega169 .....	23
2.5.- AVR STUDIO .....	25
CAPÍTULO 3 .....	27
3. COMPONENTES.....	28
3.1 IMPLEMENTACION.....	29
3.2 DESCRIPCION DE LOS EJERCICIOS. ....	30
3.2.1. EJERCICIO #1.- Desplazamiento de una palabra en la LCD del AVR Butterfly .....	30
3.2.2. EJERCICIO #2.- Cronómetro presentado en el display del .....	31
AVR butterfly haciendo uso de las interrupciones del .....	31
TIMER 0. ....	31



3.2.3. EJERCICIO #3.- Desplazamiento de una animación en la .....	32
LCD del AVR Butterfly .....	32
3.2.4. EJERCICIO #4.- Desplazamiento de una Frase en la LCD .....	33
del AVR Butterfly haciendo uso de las interrupciones del .....	33
TIMER0. ....	33
3.2.5. EJERCICIO #5.- Como crear un sonido utilizando el speaker del AVR Butterfly haciendo uso de las interrupciones del TIMER0 para simular el control de alarma de un carro. ....	33
CAPÍTULO 4 .....	35
4.1 EJERCICIO .....	36
4.1.1 EJERCICIO #1.- Desplazamiento de una palabra en la LCD del AVR Butterfly .....	36
4.1.2 DIAGRAMA DE BLOQUES DEL EJERCICIO N°1 .....	36
4.1.3 DIAGRAMA DE FLUJO DEL EJERCICIO N° 1 .....	31
4.1.4 REPRESENTACION GRAFICA DEL ISIS EJERCICIO .....	32
N° 1.....	32
4.1.5 CODIGO DE SIMULACION EN EL AVR STUDIO DEL .....	33
EJERCICIO #1.....	33
4.2 EJERCICIO .....	42
4.2.1 EJERCICIO #2.- Cronometro presentado en el display del .....	42
AVR butterfly haciendo uso de las interrupciones del .....	42
TIMER0 .....	42

4.2.2 DIAGRAMA DE BLOQUES DEL DIAGRAMA DE .....	42
BLOQUES DEL N° 2 .....	42
4.2.3DIAGRAMA DE FLUJO DEL EJERCICIO N° 2 .....	43
4.2.4 REPRESENTACION GRAFICA DEL ISIS EJERCICIO .....	44
N°2 .....	44
4.2.5 CODIGO DE SIMULACION EN EL AVR STUDIO DEL .....	45
EJERCICIO #2.....	45
4.3 EJERCICIO .....	55
4.3.1 EJERCICIO #3.- Desplazamiento de una animación en la.....	55
LCD del AVR Butterfly haciendo uso de las interrupciones .....	55
del TIMER0.....	55
4.3.2 DIAGRAMA DE BLOQUES DEL EJERCICIO N°3 .....	55
4.3.3 DIAGRAMA DE FLUJO DEL EJERCICIO N° 3 .....	56
4.3.4 REPRESENTACION GRAFICA DEL ISIS EJERCICIO .....	57
N° 3.....	57
4.3.5 CODIGO DE SIMULACION EN EL AVR STUDIO DEL EJERCICIO	
N° 3.....	58
4.4 EJERCICIO .....	67
4.4.1 EJERCICIO #4.- Desplazamiento de una Frase en la LCD del.....	67
AVR Butterfly haciendo uso de las interrupciones del .....	67
TIMER0 usando lenguaje assembler.....	67

4.4.2 DIAGRAMA DE BLOQUES DEL EJERCICIO N°4 .....	67
4.4.3 DIAGRAMA DE FLUJO DEL EJERCICIO N° 4 .....	68
4.4.4 REPRESENTACION GRAFICA DEL ISIS EJERCICIO N° 4.....	69
4.4.5 CODIGO DE SIMULACION EN EL AVR STUDIO DEL EJERCICIO N° 4.....	70
4.5 EJERCICIO .....	78
4.5.1 EJERCICIO #5.- Como crear un sonidos utilizando el .....	78
speaker del AVR Butterfly haciendo uso de las .....	78
interrupciones del TIMER0 para simular el control de .....	78
alarma de un carro. ....	78
4.5.2 DIAGRAMA DE BLOQUES EL EJERCICIO N°5 .....	79
4.5.3 DIAGRAMA DE FLUJO DEL EJERCICIO N° 5 .....	79
4.5.4 REPRESENTACION GRAFICA DEL ISIS EJERCICIO N°5.....	80
4.5.5 CODIGO DE SIMULACION EN EL AVR STUDIO DEL .....	81
EJERCICIO N° 5 .....	81
CONCLUSIONES .....	107
RECOMENDACIONES.....	108
BIBLIOGRAFÍA.....	109

# ÍNDICE DE FIGURAS

Fig. 1-1 TCCR0 Registro .....	2
Fig. 1-2 TCNT0 Registro .....	3
Fig. 1-3 TIMSK Registro .....	4
Fig. 1-4 TIFR Registro .....	4
Figura 1-5. Registro TCCR0A .....	5
Figura 1-6. Diagrama de bloques del TIMER/CONTADOR 0 de 8 bits.....	9
Figura 1-7. El diagrama de tiempo para el modo PWM rápido. ....	14
Figura 1-8. El diagrama de tiempos para el modo de fase correcta PWM....	17
Figura 2.1.- Componentes del programa ISIS .....	19
Figura 2.2.- Avr Butterfly .....	20
Figura 2.3.- Diferentes periféricos que podemos usar en el AVR BUTTERFLY.....	22
Figura 2.4.- conexión del Joystick.....	22
Figura3.1.- implementación de la Butterfly AVR .....	29
Figura3.2.- Ejercicio #1 .....	30
Figura3.3.- Ejercicio #2 .....	31
Figura3.4.- Ejercicio #3.....	32
Figura3.5.- Ejercicio #4 .....	33

Figura3.6.- Ejercicio #5 ..... 34

# ÍNDICE DE TABLAS

TABLA. 1.1 Reloj Descripción de Bit Escogida.....	3
TABLA 1.2. DESCRIPCIÓN DE BIT DE MODO DE .....	6
GENERACIÓN DE FORMA DE ONDA.....	6
TABLA 1.3. Modo de salida del comparador, sin Modo PWM .....	7
TABLA 1.4. Modo de Comparación, Modo PWM Rápido .....	8
TABLA 1.5. Modo de Comparación, Modo PWM de Fase Correcta .....	8

# INTRODUCCIÓN

Se tiene como objetivo de este proyecto, el diseño e implementación de los usos del TMR0 en sus diferentes formas de uso.

El sistema es controlado por el PIC AVR Atmega169 propio del AVR BUTTERFLY, que se encarga de almacenar y ejecutar los diferentes ejercicios que mostraremos a continuación.

El primer capítulo contiene una descripción general del proyecto, las partes que lo componen y las funciones que es capaz de realizar, además contiene las aplicaciones más comunes en las cuales es utilizado el proyecto y sistemas de similares características.

El segundo capítulo contiene una descripción general del software y el hardware utilizado para la elaboración del proyecto; entre las herramientas de software de programación, tenemos el compilador AVR STUDIO 4, con sus respectivas características, el simulador de circuitos Proteus con sus funciones principales.

El tercer capítulo contiene todo lo referente a la descripción e implementación del proyecto, comenzando con el prototipo inicial, las pruebas realizadas y los cambios realizados para tratar de obtener un mejor resultado, de acuerdo a las especificaciones planteadas.

El cuarto capítulo, se muestra los esquemas utilizados para el correcto funcionamiento del proyecto. Además se adjuntan los diagramas y esquemas electrónicos

# CAPÍTULO 1

## RELEVANCIA DEL TIMER/COUNTER0 PARA EL DESARROLLO DE ESTE PROYECTO

### 1. – INTRODUCCION AL CAPITULO.

En este capítulo vamos a destacar el uso del Tmr0, describiendo en esta sección como utilizar y configurar Timer/Contador 0, así como sus modos de operación, dando una descripción general del mismo y sus características. Hemos visto como los temporizadores están compuesto de registros, cuyo valor automáticamente aumenta/disminuye. En AVR, hay tres tipos de temporizadores - TIMER0, TIMER1 y TIMER2. De estos, TIMER1 es un temporizador 16 bites mientras que los otros son temporizadores de 8 bit. Hemos visto como prescalador es usado para establecer la duración de tiempo con la resolución



## 1.1.- DESCRIPCIÓN GENERAL DEL TIMER0

Timer/Counter0 es un módulo interno del microcontrolador que sirve para realizar Temporizaciones internas en el micro o bien, conteos de pulsos externos.

El Timer0 tiene las siguientes características:

- Temporizador/contador de 8-BIT
- Contador con una unidad de comparación.
- Modo de borrado de contador en la igualdad de la comparación CTC.
- Modulador de ancho de pulso (PWM) de fase correcta y libre de glitches (Pulsos cortos).
- Generador de frecuencia.
- Pre-escalador de 10 bits (permite dividir la frecuencia).
- Dos fuentes de interrupción: desbordamiento e igualdad en la comparación (TOV0 y OCF0).

Los registros que intervienen para la configuración del timer0 son:

### 1.1.1. TCCR0 – TIMER/COUNTER REGISTRO DE CONTROL.

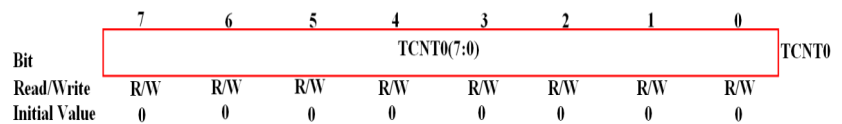
Bit	7	6	5	4	3	2	1	0	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Fig. 1-1 TCCR0 Registro

Ahora mismo, nos concentraremos en los bit destacados. Para seleccionar estos tres **Bit de Reloj Escogidos**, CS02, CS01, CS00, establecemos el temporizador escogiendo el prescalador apropiado. Se Muestran las combinaciones posibles.

<b>CS02</b>	<b>CS01</b>	<b>CS00</b>	<b>DESCRIPCION</b>
<b>0</b>	<b>0</b>	<b>0</b>	<b>Ninguna fuente de reloj</b> (Timer/contador detenido.)
<b>0</b>	<b>0</b>	<b>1</b>	<b>CLK<sub>0/1</sub></b> (Reloj de sistema)
<b>0</b>	<b>1</b>	<b>0</b>	<b>CLK<sub>0/1/8</sub></b> ( Reloj/8 de sistema)
<b>0</b>	<b>1</b>	<b>1</b>	<b>CLK<sub>0/1/64</sub></b> (Reloj/64 de sistema)
<b>1</b>	<b>0</b>	<b>0</b>	<b>CLK<sub>0/1/256</sub></b> (Reloj/256 de sistema)
<b>1</b>	<b>0</b>	<b>1</b>	<b>CLK<sub>0/1/1024</sub></b> (Reloj/1024 de sistema)
<b>1</b>	<b>1</b>	<b>0</b>	<b>Fuente de reloj Externo en el pin. En flanco descendente</b>
<b>1</b>	<b>1</b>	<b>1</b>	<b>Fuente de reloj Externo en el pin. En el flanco ascendente</b>

**TABLA. 1.1 Reloj Descripción de Bit Escogida**



**Fig. 1-2 TCNT0 Registro**

### 1.1.2. TCNT0 – TIMER/COUNTER REGISTRO.

Esto es donde el contador de 8 bit del temporizador reside. El valor del contador es almacenado aquí y aumenta/disminuye automáticamente. Los datos pueden ser ambos leído/escritos de este registro. Ahora conocemos en donde está el valor, pero este registro no será activado a no ser que nosotros activemos el temporizador. Así tenemos que establecer el temporizador.

### 1.1.3. TIMSK– TIMER / COUNTER REGISTRO DE ENMASCARAMIENTO DE INTERRUPCIONES.

Bit	7	6	5	4	3	2	1	0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Fig. 1-3 TIMSK Registro

Registro de enmascaramiento de interrupciones del Timer/Contador, este registro es común para los tres temporizadores. Para TIMER0, OCIE0 es la habilitación de la interrupción por igualdad en la comparación de salida y TOIE0 es la habilitación de la interrupción por desbordamiento.

Bit	7	6	5	4	3	2	1	0	TIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Fig. 1-4 TIFR Registro

#### 1.1.4. TIFR – TIMER/COUNTER REGISTRO DE BANDERAS DE INTERRUPTACIONES.

Nos enfocaremos en el **TOV0** bit y en el **OCF0** bit. TOV0 es “1” siempre que TIMER0 se desborde y OCF0 es “1” cuando se habilita la igualdad de comparación de salida.

#### 1.1.5. TCCR0A-TIMER /COUNTER REGISTRO DE CONTROL

A

Bit	7	6	5	4	3	2	1	0	
	FOC0A	WGM00	COM0A1	COM0A0	WGM01	CS02	CS01	CS00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 1-5. Registro TCCR0A

##### Bit 7 – FOC0A: Forzar a Output Compare A.

Este bit está activo únicamente cuando el bit WGM especifica un modo sin-PWM. Al escribir un uno en este bit, se fuerza a una coincidencia de comparación inmediata en la Unidad Generadora de Onda. La salida OC0A cambia según la configuración de sus bits COM0A1:0.

**BIT 6, 3 – WGM01:0: Modo de Generación de Onda.**

Los modos de operación soportados por la unidad Timer/Counter0 son: modo Normal, modo Clear Timer on Compare Match (CTC) y dos tipos de modo PWM.

Mode	WGM1	WGM0	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on <sup>(1)(2)</sup>
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCRA	Immediate	MAX
3	1	1	Fast PWM	0xFF	TOP	MAX

**TABLA 1.2. DESCRIPCIÓN DE BIT DE MODO DE GENERACIÓN DE FORMA DE ONDA**

Nota: 1. MAX = 0xFF

2. BOTTOM = 0x00

Bit 7, FOC0A (Force Output Compare A): sólo está activo cuando tenemos seleccionado un modo distinto a los de PWM (modo normal o CTC). Poniendo el bit a uno la salida OC0A es cambiada conforme su configuración en los bits COM0A1:0, no genera ninguna interrupción ni limpia el registro que lleva la cuenta.

El comparador de 8 bits continuamente compara el registro TCNT0 con el Registro de Comparación de Salida (OCR0). Cuando TCNT0 iguala a OCR0 las señales del comparador se igualan. Una

igualdad pondrá en 1 en la bandera de comparación de salida (OCF0) en el próximo ciclo de reloj del timer, si es habilitada OCIE0 = 1 y la Bandera de Interrupciones Globales SREG = 1 la Bandera de Salida de Comparación genera una interrupción de comparación a la salida de la bandera OCF0 es automáticamente limpiada cuando las interrupciones se ejecuta alternativamente la bandera OCF0 puede ser limpiada con un 1 lógico de su bit específico de I/O. El generador de forma de onda usara la señal igualada para generar una salida acorde al modo de operación colocado por los bits WGM01:0 y los bits del Modo de Comparación de Salida COM01:0 las señales de Max y Botton son usadas por el generador de forma de onda para el manejo especial de los valores extremos en algunos modos de operación.

#### **Bit 5:4 – COM0A1:0: Modo Compare Match Output.**

Estos bits controlan el comportamiento del pin Output Compare (OC0A). El bit del Registro de Dirección de Datos correspondiente al pin OC0A debe ser escrito con uno lógico “1” para habilitarlo como salida.

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	Toggle OC0A on Compare Match
1	0	Clear OC0A on Compare Match
1	1	Set OC0A on Compare Match

**TABLA 1.3. Modo de salida del comparador, sin Modo PWM**

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	Reserved
1	0	Clear OC0A on compare match, set OC0A at TOP
1	1	Set OC0A on compare match, clear OC0A at TOP

**TABLA 1.4. Modo de Comparación, Modo PWM Rápido**

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	Reserved
1	0	Clear OC0A on compare match when up-counting. Set OC0A on compare match when downcounting.
1	1	Set OC0A on compare match when up-counting. Clear OC0A on compare match when downcounting.

**TABLA 1.5. Modo de Comparación, Modo PWM de Fase Correcta**

## 1.2.-TIMER 0 EN AVR DESCRIPCIÓN COMPLETA.

### 1.2.1.-DIAGRAMA DE BLOQUES Y EXPLICACION.

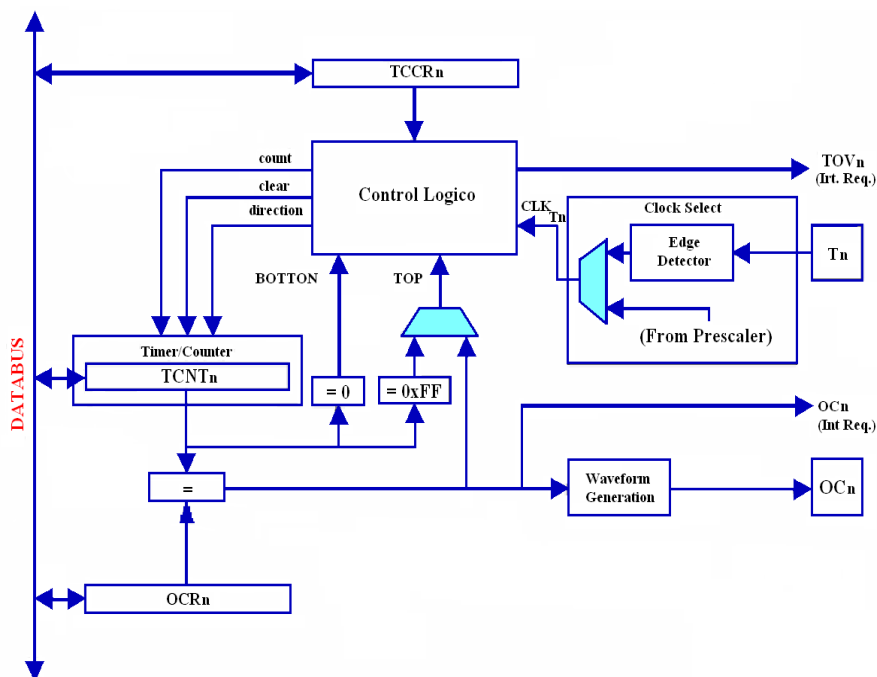


Figura 1-6. Diagrama de bloques del TIMER/CONTADOR 0 de 8 bits.

Entre los bloques podemos destacar a TCCRn que es el registro que configura al timer en su modo de funcionamiento, Tn es por donde se inyecta la señal de reloj para el modo de contador de eventos externos, OCn es el pin por el cual se genera la señal digital.

Los registros del TIMER/CONTADOR 0 (TCNT0) y el de comparación de salida (OCR0) son de 8 bits. Las señales de petición de interrupción (Interrupt Request) son visibles en el Registro de Banderas de la Interrupción del Timer (TIFR).



TCNT0 es el registro de 8 bits donde se lleva la cuenta de los pulsos del reloj (interno o externo)

**Señales:**

**Count:** incrementa o decrementa la cuenta de TCNT0 con cada pulso de reloj.

**Direction:** selecciona entre incremento o decremento de TCNT0.

**Clear:** pone todo los bits de TCNT0 a 0.

**CLKtn:** la señal de reloj.

**Top:** indica que TCNT0 ha alcanzado su valor máximo, el valor máximo lo podemos asignar en el registro OCROA.

**Bottom:** TCNT0 ha alcanzado el valor mínimo 0x00.

El hardware del timer compara de manera continua la cuenta de TCNT0 con el valor de los registro OCR0A y OCR0B.

### 1.2.2.- MODOS DE OPERACION

El modo de operación, por ejemplo, la conducta del Timer/Contador y los pines de Comparación de Salida son definidos por la combinación de los bits en el modo de Generación de Forma de Onda (WGM01:0) y los bits del modo de Comparación de Salida (COM01:0).

### 1.2.2.1.- MODO NORMAL PWM

El modo más simple es el modo normal (WGM01: 0 = 0). En este modo la dirección del conteo es siempre ascendente y no se limpia el contador. El contador simplemente se sobrescribe cuando pasa de su máximo valor de 8 bits (TOP = 0xFF) y entonces se reinicia desde su valor más bajo (0x00). En operación normal la Bandera de Sobre flujo del Timer/Contador 0 (TOV0) será puesta a uno en el mismo ciclo de reloj del timer como el TCNT0 llega a ser cero. La bandera TOV 0 en este caso se comporta como el noveno bit, excepto si esta puesto en uno, y no es limpiado. Sin embargo, combinado con la interrupción de sobre flujo del timer que automáticamente limpia la bandera TOV0, la resolución del timer puede ser incrementada por software.

### 1.2.2.2.- MODO DE COMPARACION DE LIMPIEZA DEL TIMER

(CTC).

En el modo de comparación limpieza del timer (WGM01:0=2), el registro OCR0 se utiliza para manipular la resolución del contador. En el modo CTC el contador se limpia a cero cuando el valor del contador (TCNT0) iguala a OCR0. El valor del contador (TCNT0) se incrementa hasta que una igualdad en la comparación entre TCNT0y OCR0, y entonces el contador TCNT0 se limpia.

Una interrupción puede generarse cada vez que el valor del contador alcanza el valor TOP usando la bandera OCF0. Si la interrupción está habilitada, la rutina de manejo de la interrupción puede ser usada para actualizar el valor TOP. Sin embargo, cambiando TOP a un valor cercano a BOTTOM cuando el contador está corriendo sin ningún valor bajo de prescalador, debe realizarse con cuidado ya que en el modo CTC no se tiene la característica de doble buffer. Si el valor nuevo escrito a OCR0 es tan más pequeño que el valor actual de TCNT0, el contador perderá la igualdad de comparación. El contador entonces contará a su máximo valor (0xFF) y volverá a contar desde 0x00 antes de que la igualdad de comparación ocurra.

$$f_{ocn} = \frac{f_{clk\_I/O}}{2.N.(1+OCRn)}$$

**La variable N representa el factor pre-escalar (1, 8, 64, 256 o 1024).**

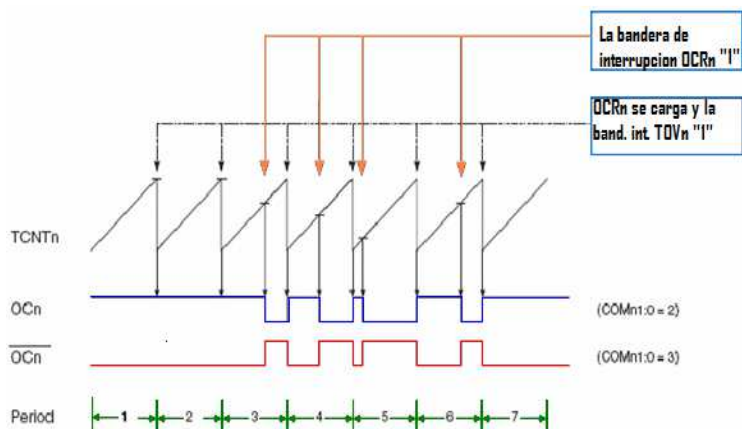
En el modo normal de operación, la bandera TOV0 se coloca a uno en el mismo ciclo de reloj cuando contador pasa de MAX a 0x00.

### **1.2.2.3.- MODO RAPIDO PWM.**

La modulación rápida PWM o el modo rápido PWM (WGM01:0=3) provee una alta generación en frecuencia de forma de onda PWM. El PWM rápido difiere de la opción

anterior de PWM por su operación de una sola pendiente. El contador se incrementa de BOTTOM a MAX, entonces se reinicia desde BOTOM. En el modo no invertido de comparación de salida, el bit de comparación de salida (OC0) se limpia cuando se iguala la comparación entre TCNT0 y OCR0, y se ajusta a BOTTOM. En el modo invertido de comparación de salida, la salida es puesta a uno cuando se iguala la comparación y se limpia a BOTTOM. Debido a la operación de una sola pendiente, la frecuencia de operación del modo PWM rápido puede ser doblada tan alta como el modo de fase correcto de PWM que usa la operación dual de pendientes. Esta frecuencia alta hace que el modo rápido PWM se ajuste en aplicaciones de regulación de potencia, rectificación y DAC.

En el modo rápido PWM, el contador se incrementa hasta el valor del contador iguala el valor MAX. El contador es entonces limpiado en el siguiente ciclo de reloj. El valor de TCNT0 está en el diagrama de tiempo mostrado como un histograma para ilustrar la operación de una sola pendiente. El diagrama incluye salidas de PWM no invertida e invertida. La línea pequeña horizontal marca en donde la pendiente llega a TCNT0 y se iguala a OCR0.



**Figura 1-7. El diagrama de tiempo para el modo PWM rápido.**

La bandera de sobre flujo Timer/Contador (TOV0) se coloca a uno cada vez que el contador alcanza MAX. Si la interrupción está habilitada, la rutina que maneja la interrupción puede ser usada para actualizar el valor a comparar. En el modo PWM rápido, la unidad de comparación permite la generación de formas de onda de PWM en el pin OC0. Ajustando los bits  $COM01:0 = 2$  producirá un PWM no invertido y un PWM invertido se logra ajustando los bits  $COM01:0 = 3$  se genera un PWM invertido. El valor de OC0 actual será solamente visible en el pin del puerto si la dirección de los datos para el pin de puerto se coloca como salida. La forma de onda de PWM se genera poniendo a uno (o a cero) el registro OC0 en la comparación de igualdad entre OCR0 y TCNT0, y poniendo a cero (o a uno) el registro OC0 en el ciclo del timer de reloj del contador es limpiado (cambiando de MAX a BOTTOM).

La frecuencia PWM para la salida puede ser calculada por la siguiente ecuación:

$$f_{OCnPWM} = \frac{f_{clk\ I/O}}{N \cdot 256}$$

**La variable N representa el factor prescalar (1, 8, 64, 256 o 1024).**

Los valores extremos para el registro OCR0 representan casos especiales cuando se genera una forma de onda PWM de salida en el modo rápido PWM. Si el OCR0 es igual BOTTOM, la salida será un pequeño salto por cada ciclo de reloj MAX+1. Ajustando de OCR0 igual a MAX resultara en una salida constante alta o baja (dependiendo de la polaridad del ajuste de salida por los bits COM01:0).

Una forma de onda con frecuencia de salida de un 50% a razón de ciclo en el modo rápido PWM puede ser alcanzado ajustando OC0 a su nivel lógico invertido en cada igualdad de comparación (COM01:0=1). La forma de onda generada tendrá una frecuencia máxima de  $f_{OC0} = f_{clk\_I/O}/2$  cuando OCR0 este en cero. Esta característica es similar al modo CTC de OC0 invertido, excepto por la característica del doble buffer en la unidad de comparación de salida este habilitada en el modo rápido PWM.

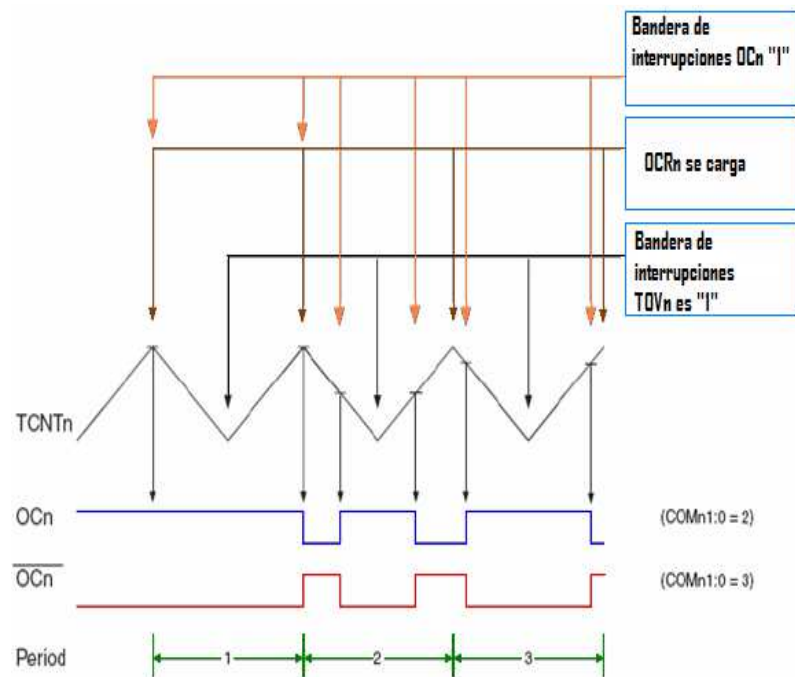
#### 1.2.2.4.- MODO PWM DE FASE CORRECTA.

El modo de fase correcta PWM (WGM01:0=1) provee una alta resolución de fase correcta de forma de onda en la generación de PWM.

El modo de fase correcta de PWM se basa en la operación de doble pendiente. El contador se incrementa repetidamente desde BOTTOM a MAX y se decrementa desde MAX a BOTTOM. En el modo de comparación de salida no invertido, la comparación de salida (OC0) se limpia cuando se igualan los registros entre TCNT0 y OCR0 mientras se cuenta ascendientemente y se pone a uno en la comparación igualada mientras se cuenta descendientemente. En el modo de comparación de salida invertida, la operación se invierte. La operación de pendiente doble tiene una frecuencia de operación máxima más baja que la operación de una sola pendiente. Sin embargo, debido a la característica simétrica de los modos de doble pendiente PWM, estos modos se prefieren para aplicaciones de control de motores.

La resolución PWM para el modo de fase correcta PWM es fija a 8 bits. En el modo de fase correcta PWM el contador se incrementa hasta que el valor del contador iguala a MAX. Cuando el contador alcanza el MAX, cambia la dirección de la cuenta. El valor de TCNT0 será igual a MAX en un ciclo de reloj del timer. El valor de TCNT0 se muestra como un histograma para ilustrar la operación de doble pendiente. El diagrama incluye salidas de PWM invertidas y no invertidas. La línea horizontal en

la pendiente TCNT0 representa la comparación igualada entre OCR0 y TCNT0.



**Figura 1-8. El diagrama de tiempos para el modo de fase correcta PWM.**

La bandera de sobre flujo del Timer/Contador 0 (TOV0) se pone a uno cada vez que el contador alcanza el BOTTOM. La bandera de interrupción se usa para generar una interrupción cada vez el contador alcanza el valor de BOTTOM.

En el modo de fase correcta PWM, la unidad de comparación permite la generación de formas de onda PWM en el pin OC0.



Ajustando los bits COM01:0=2 producirá una salida PWM no invertida. Una salida PWM invertida se genera colocando los bits COM01:0=3. El valor actual OC0 solamente será visible en el pin del puerto si la dirección de datos para el pin es puesto como salida. La forma de onda PWM es generada limpiando (o poniendo a 1) el registro OC0 en la comparación igualada entre OCR0y TCNT0 cuando el contador se incrementa, y poniendo a uno (o limpiando) el registro OC0 en la comparación igualada entre OCR0 yTCNT0 cuando el contador se decrementa. La frecuencia del PWM para la salida cuando se usa el modo de fase correcta PWM puede ser calculado por la siguiente ecuación:

$$f_{OCnPCPWM} = \frac{f_{clk} / O}{N \cdot 510}$$

**La variable N representa el factor pre-escalar (1, 8, 64, 256 o 1024)**

Los valores extremos para el registro OCR0 representan casos especiales cuando se genera una forma de onda de salida PWM en el modo de fase correcta PWM. El registro OCR0 es puesto igual a BOTTOM, la salida será continuamente en bajo y cuando se pone igual a MAX la salida será continuamente alta para el modo PWM no invertido. Para el PWM invertido la salida tendrá valores lógicos opuestos.

### 1.2.3.- CARACTERISTICAS DEL TIMER0 EN LA FAMILIA

#### MICROCHIP

Aquí se les presentara algunas características del TIMER0 en microchip para que ustedes puedan apreciar algunas diferencias tanto a nivel de operación como en el diagrama de bloques.

- Se basa en un contador ascendente de 8 bits al que se accede mediante un registro en RAM denominado TMR0 (posiciones 01h-101h).
- Puede utilizar un prescaler o divisor de frecuencia previo de 8 bits cuyo valor de división es configurable por software.
- Se puede seleccionar como fuente de reloj: un reloj interno ( $f_{osc}/4$ ) como temporizador o uno externo que entre a través del pin RA4/T0CKI como contador
- Permite solicitar interrupciones cuando se produce un desbordamiento (overflow) del registro TMR0. Es decir cuando pasa del valor 0xFF al 0x00.

# CAPÍTULO 2

## BASE TEORICA

En este capítulo les hablaremos sobre el software y hardware a utilizar, además daremos una breve introducción sobre:

- Características Microcontralador Atmel
- Proteus Isis
- AVR BUTTERFLY
- ATmega169
- AVR Studio (plataforma en donde vamos a programar) características

### 2.1.- EL MICROCONTROLADOR ATMEL

ATMEL fabrica los microcontroladores de la familia AVR, esta nueva tecnología proporciona todos los beneficios habituales de arquitectura RISC y memoria flash reprogramable eléctricamente. La característica que los identifica a estos microcontroladores de ATMEL es la memoria flash y EEPROM que incorpora. AVR

compite con varias familias de microcontroladores bien establecidas en el mercado, tales como 8051 de Intel, 68HC11 de motorola y la familia PIC de Microchip. La compañía ATMEL también produce y vende varios subproductos de la popular familia 8051 con la diferencia de que están basados en la memoria flash.

El diseño AVR de ATMEL difiere de los demás microcontroladores de 8 bits por tener mayor cantidad de registros (32) y un conjunto ortogonal de instrucciones. AVR es mucho más moderna que su competencia. Por ejemplo, los 8051, 6805 y los PIC, se los arreglan con un único acumulador, los 658HC11 y 68HC12 tienen simplemente 2. Esto hace que la arquitectura AVR sea más fácil de programar a nivel de lenguaje ensamblador y que sea fácil de optimizar con un compilador. El gran conjunto de registros disminuye la dependencia respecto a la memoria, lo cual mejora la velocidad y disminuye las necesidades de almacenamiento de datos. Además casi todas las instrucciones se ejecutan en 1 ó 2 ciclos de reloj versus 5-10 ciclos de reloj para los chips 8051, 6805, 68HC11 y PIC.

Adicionalmente, ATMEL también proporciona en línea el entorno software (AVR estudio) que permite editar, ensamblar y simular el código fuente, (la explicación del Avr Studio 4.0, se explicará más adelante). Una vez ensamblado y depurado el código fuente del programa, se transferirá el código máquina a la memoria flash del microcontrolador para esto se debe disponer de otro entorno de desarrollo para programar en forma serial o paralelo la memoria flash.

Las familias AVR rápidamente han crecido en el mercado y se dispone de las siguientes categorías:

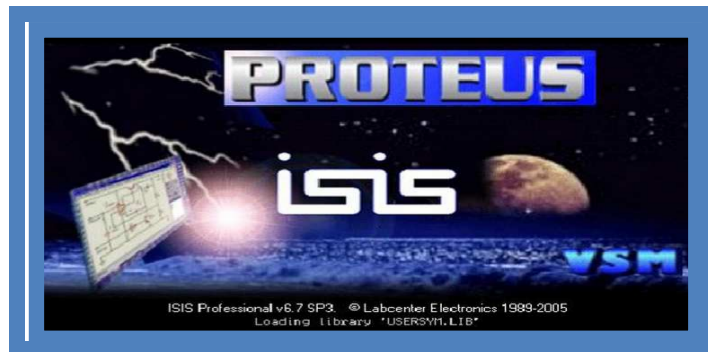
- TINY AVR: son microcontroladores de propósito general con memoria flash hasta 2 Kbytes y 128 bytes de memorias SRAM y EEPROM.

- AVR: Microcontroladores de propósito general con 8 Kbytes de memoria flash y 512 bytes de memoria SRAM y EEPROM.
- Mega AVR.- Memoria flash hasta 256 Kbytes, 4 Kbytes de memoria EEPROM y SRAM.

Los tipos de encapsulado del microcontrolador del ATmega presenta desde 28 pines hasta 100 pines en la forma de DIP, TQFP y MLF y su voltaje de alimentación está en el rango de 1.8 a 5.5 voltios.

Existen herramientas que son comunes a todas las familias de los AVR, entre ellas, se puede mencionar el ambiente de Desarrollo denominado AVR STUDIO, que cuenta con la posibilidad de, no solo programar en ASSEMBLER, sino de integrar un compilador para C (GNU/GCC). Ambos son gratuitos y se pueden obtener accediendo al website de ATMEL. La característica principal del AVR STUDIO es su fácil manejo, permite visualizar rápidamente que ocurre con los registros, como también así se pueden modificar.

## 2.2.- PROTEUS ISIS



El software de diseño y simulación Proteus VSM es una herramienta útil para estudiantes y profesionales que deseen mejorar habilidades para el desarrollo de aplicaciones analógicas y digitales. El programa Proteus es una aplicación CAD (aplicación de diseño) que se compone de tres módulos:

### 2.2.1.- ISIS (Intelligent Schematic Input System):

Es el modulo de captura de esquemas. Permite el diseño de circuitos empleando un entorno gráfico, en el cual es posible colocar los símbolos de los componentes y realizar simulaciones de su funcionamiento, sin el riesgo de ocasionar daños a los circuitos.

### 2.2.2.- ARES (Advanced Routing Modelling):

Es el modulo para realización de circuitos impresos. Proteus VSM tiene la capacidad de pasar el diseño a un programa integrado llamado ARES en el cual se puede llevar a cabo el desarrollo de placas de circuitos impresos.

### 2.2.3.- VSM (Virtual System Modelling):

Es el modulo de simulación. Simular el circuito para comprobar el funcionamiento correcto sin que se generen errores o warnings.

Una vez comprendidos los objetivos del programa ISIS, se presenta a continuación el entorno laboral del programa en donde se pueden observar sus componentes.

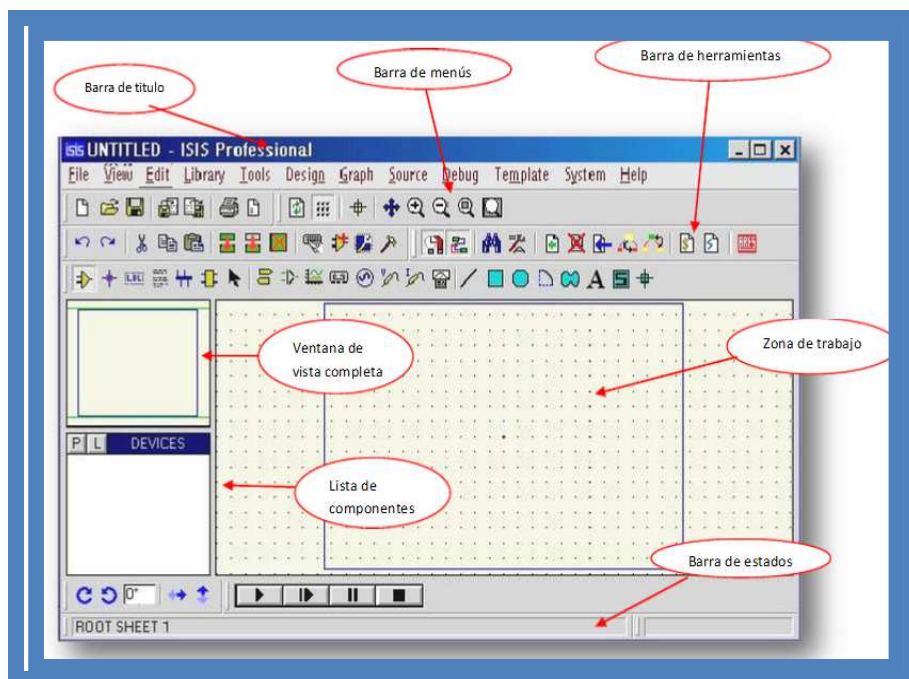


Figura 2.1.- Componentes del programa ISIS

### 2.3.- AVR BUTTERFLY

La mariposa AVR es un sistema autónomo que funciona con pila plana de demostración que ejecuta el microcontrolador Atmel AVR ATMEGA169PV. El tablero incluye una pantalla de cristal líquido (LCD), joystick, altavoz, puerto serie, reloj en tiempo real (RTC), la memoria flash interna, y los sensores de temperatura y voltaje. Programando este elemento puede ser usado para que el usuario introduzca su nombre en la pantalla LCD.

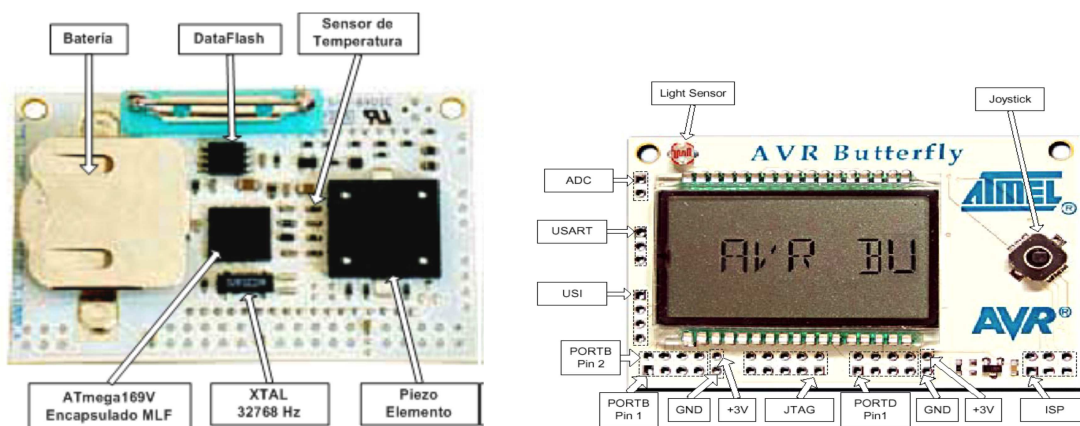


Figura 2.2.- Avr Butterfly



La mariposa AVR viene precargado con el software que muestra muchas de las capacidades de los microcontroladores AVR. Visualización de las lecturas del sensor, y mostrar la hora. La mariposa AVR también tiene un altavoz que pueden reproducir sonidos y la música.

Utilizando los periféricos:

- Controlador de LCD.
- Memoria
  - Flash, EEPROM, SRAM, los datos externos de Flash
- Interfaces de comunicación
  - # UART, SPI, USI
- Los métodos de programación
  - Selfprogramming / gestor de arranque, SPI, en paralelo, JTAG
- Conversor analógico digital (ADC)
- Temporizadores / contadores
  - ❖ Contador en tiempo real (RTC)
  - ❖ Pulse Width Modulation (PWM)

... Etc

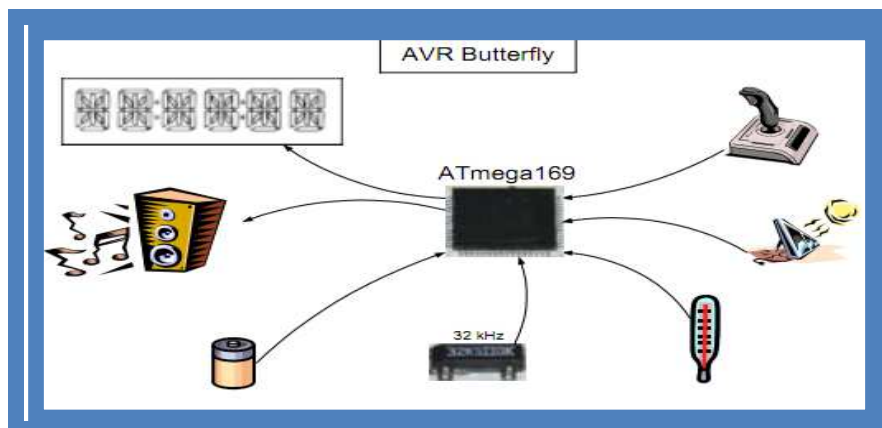


Figura 2.3.- Diferentes periféricos que podemos usar en el AVR BUTTERFLY

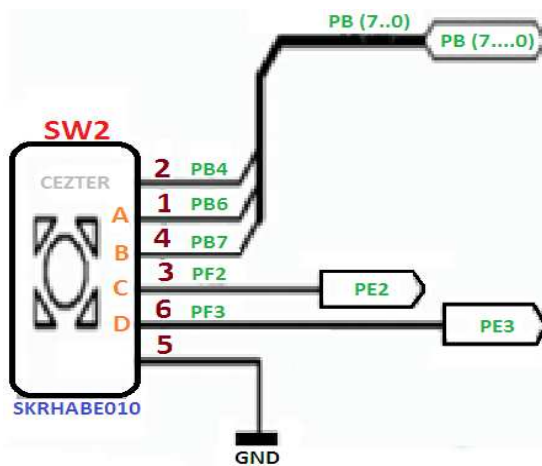


Figura 2.4.- conexión del Joystick

- Joystick
  - ❖ 4 direcciones
  - ❖ Un centro de empuje
  
- UART

- ❖ Disponible en las cabeceras de pin J406.
- ❖ Con los convertidores de nivel
  - Sólo tienes que conectar TxD, RxD y GND.
  - Vcc 2.0V min.
- USI
  - ❖ Disponible en las cabeceras de pin J405
  - ❖ Usos 3 pines: PE4, PE5, PE6
  - ❖ Si USI no son necesario entonces los pines se puede utilizar con normalidad I/O
- Reset
  - ❖ Breve corte el pin 5 y 6 de la ISP cabecera (J403)
- Temperatura
  - ❖ NTC para montaje en superficie (Coeficiente negativo de temperatura)
  - ❖ Termistor
    - Probado -10 +60 grados
    - Conectado a ADC0 (PF0)

## 2.4.- ATmega169

- 16 KB de Flash.
- 512B EEPROM.
- 1 KB de SRAM interna.
- Interfaz JTAG.
- 4 x 25 segmentos LCD del controlador.
- Dos de 8 bits temporizadores / contadores.
- Uno de 16 bits temporizador / contador.
- Contador en tiempo real.

- 4 canales de PWM.
- De 8 canales, 10-bit ADC
- USART.
- SPI.
- Interfaz Universal Serial.
- Watchdog Timer.
- Comparador analógico.
- Poder en la detección de Reset y Brown fuera.
- Oscilador interno calibrado.
- Cinco modos de suspensión:
  - ❖ Reducción de marcha mínima, ruido ADC, de ahorro de energía, al apagar.
- 53 programables líneas I / O y 1 entrada de línea.
- 64 derivaciones TQFP y del Fondo Multilateral 64-pad.
- Tensión de funcionamiento:
  - ❖ 1,8 - 3,6 para ATmega169V.
  - ❖ 2,7 - 3,6 para ATmega169L.
- Rango de temperatura:
  - ❖ 0 ° C a 70 ° C.
- Velocidad de Grado:
  - ❖ 0 - 1 MHz para ATmega169V.
  - ❖ 0 - 4 MHz para ATmega169L.
- Ultra-bajo consumo de energía
  - ❖ Modo activo:
    - 1 MHz, 1,8 V: 300 $\mu$ A.
    - 32 kHz, 1,8 V: 20 $\mu$ A (incluido el oscilador).
    - 32 kHz, 1,8 V: TBD (incluido el oscilador y LCD).
- Power-Down Mode:
  - ❖ 0.5 $\mu$ A a 1.8V.

## 2.5.- AVR STUDIO



Nosotros utilizaremos el entorno de desarrollo AVR-STUDIO para programar el microcontrolador ATMEGA169. La programación se realiza mediante la plataforma de depuración/programación AVR STUDIO también del fabricante ATMEL.

AVR Studio es el software soportado por la propia empresa Atmel para el trabajo con sus microcontroladores de 8 bits.

Este software nos ayuda a programar en tanto en Assembler como en "C" facilitándonos con la visualización de registros, entradas y salidas que utilicemos. El AVR STUDIO es fácil de manejar y rápido en la simulación de cualquier proyecto, también podemos incorporar cualquier librería para una rápida programación.

# CAPÍTULO 3

## PLATAFORMA DE IMPLEMENTACION

El ensamble de nuestro prototipo electrónico se hace sobre un elemento llamado protoboard o “tablero de prototipos”, en este aparato se pueden montar y modificar, fácil y rápidamente, circuitos electrónicos sin necesidad de soldadura y muchas veces, sin herramientas.

En este capítulo describiremos la plataforma en la vamos a implementar nuestros proyectos, haciendo un análisis de la de las conexiones y una descripción de los componentes utilizados entre el KIT AVR butterfly y todos sus periféricos.

Describiremos como trabaja la plataforma en cada uno de los proyectos que se va a implementar y su forma de uso.

### **3. COMPONENTES.**

- Kit de desarrollo AVR butterfly
- Protoboard (BREAD BOARD)
- porta pilas AA
- Pilas AA(1.5V)
- Cable UTP
- Cable conector de USB a DB9 macho
- Conector DB9 hembra
- Carrito plástico.
- Leds y resistencias.
- Botoneras.
- Headers.
- Espadines.

### 3.1 IMPLEMENTACION.

Vamos a describir las conexiones entre el Kit de desarrollo AVR butterfly, protoboard y la comunicación serial con el computador.

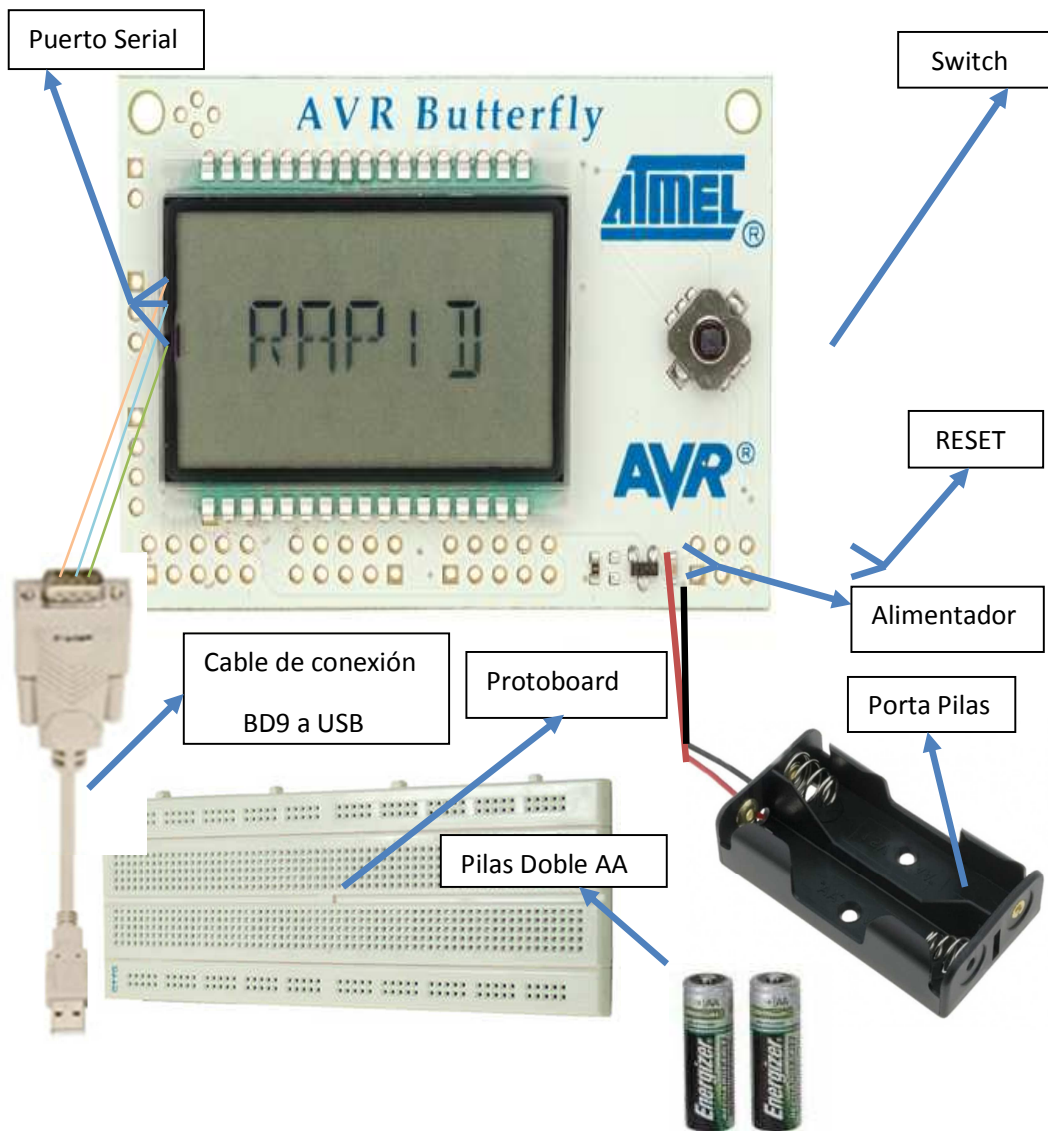


Figura3.1.- implementación de la Butterfly AVR



## 3.2 DESCRIPCION DE LOS EJERCICIOS.

### 3.2.1. EJERCICIO #1.- Desplazamiento de una palabra en la LCD del AVR Butterfly

Este programa desplaza un mensaje por medio del timer0 cada vez q se genera una igualdad de comparación se incrementa un contador con el fin de presentar una porción del mensaje, cada letra ingresada se va desplazando por el delay, y además se hace uso de la butterfly en la cual mostramos en un display liquido el mensaje guardado en nuestro código, mediante el uso de la memoria interna de la LCD.

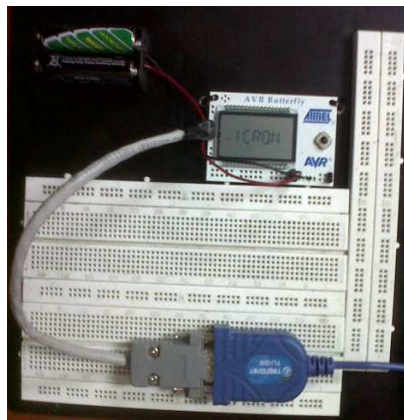
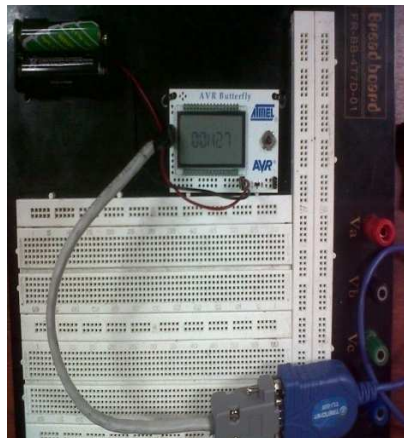


Figura3.2.- Ejercicio #1

**3.2.2. EJERCICIO #2.- Cronómetro presentado en el display del AVR butterfly haciendo uso de las interrupciones del TIMER 0.**

En este programa cada vez q se genera una interrupción por el TIMER0 se incrementa el contador para crear el cronómetro, este se presentara en el display del AVR BUTTERFLY.Cada interrupción incrementa un contador la cual nos ayudara a separar las variables de segundos y minutos, estas están separadas por una letra “H” cual la creamos con un grupo de nibbles para una mejor presentación.



**Figura3.3.- Ejercicio #2**

### 3.2.3. EJERCICIO #3.- Desplazamiento de una animación en la LCD del AVR Butterfly

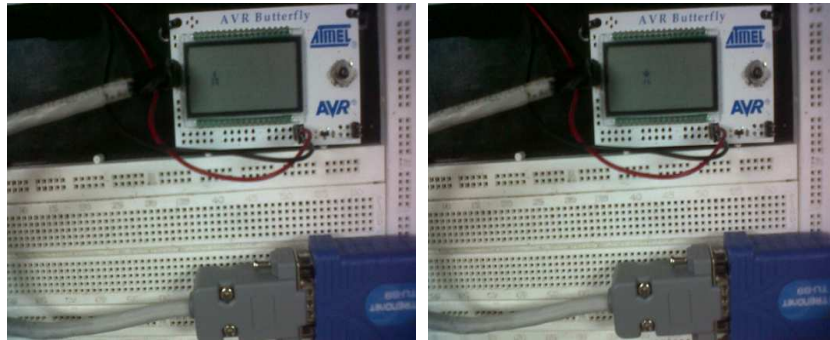
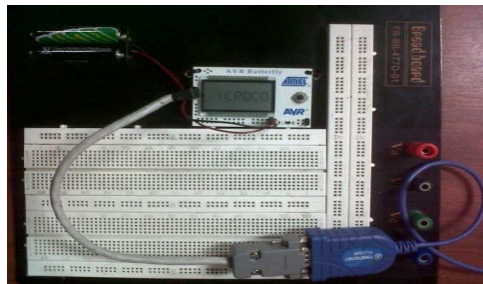


Figura3.4.- Ejercicio #3

En este programa cada vez q se genera una interrupción por el TIMER0 se incrementa un contador la cual nos sirve para presentar una animación diferente, cada animación es graficada en un segmento de la LCD estas animaciones son creadas por los nibbles del display que se va desplazando por el delay que le damos por el vector de interrupción del TIMER0

**3.2.4. EJERCICIO #4.- Desplazamiento de una Frase en la LCD del AVR Butterfly haciendo uso de las interrupciones del TIMER0.**



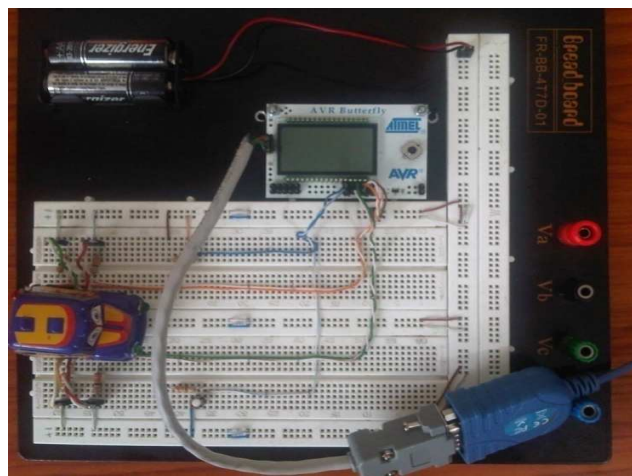
**Figura3.5.- Ejercicio #4**

En este programa cada vez q se genera una interrupción por el TIMER0 se incrementa un contador la cual nos sirve para visualizar una letra diferente, las letras van a ir presentándose en la LCD de la AVR Butterfly hasta completar la frase que se va desplazando por el delay que le damos por el vector de interrupción del TIMER0.

**3.2.5. EJERCICIO #5.- Como crear un sonido utilizando el speaker del AVR Butterfly haciendo uso de las interrupciones del TIMER0 para simular el control de alarma de un carro.**

Ingresamos un pequeño delay por medio de la interrupción del Timer0 para enviar una señal al speaker del AVR Butterfly con la que podemos

crear diferentes sonidos el cual sonara por un tiempo predeterminado, pero también podemos interrumpir su tiempo de duración con un segundo pulsador que simula el control de alarma de un carro si volvemos a presionar el primer pulsador se encenderá la alarma simulando que el carro ha sido tocado otra vez.



**Figura3.6.- Ejercicio #5**

# CAPÍTULO 4

## IMPLEMENTACION DEL PROYECTO

En este capítulo vamos a tratar sobre como poder utilizar el TIMER0 en el Atmega 169 y para su visualización usaremos el AVR Butterfly.

Presentaremos cinco ejercicios y para una mayor comprensión mostraremos el diagrama de bloques, el diagrama de flujo y el código de simulación con el fin de que el estudiante pueda apreciar con claridad cada ejercicio.

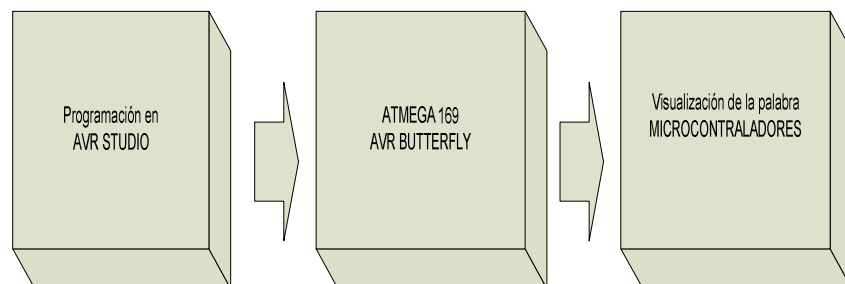
## 4.1 EJERCICIO

### 4.1.1 EJERCICIO #1.- Desplazamiento de una palabra en la LCD del AVR

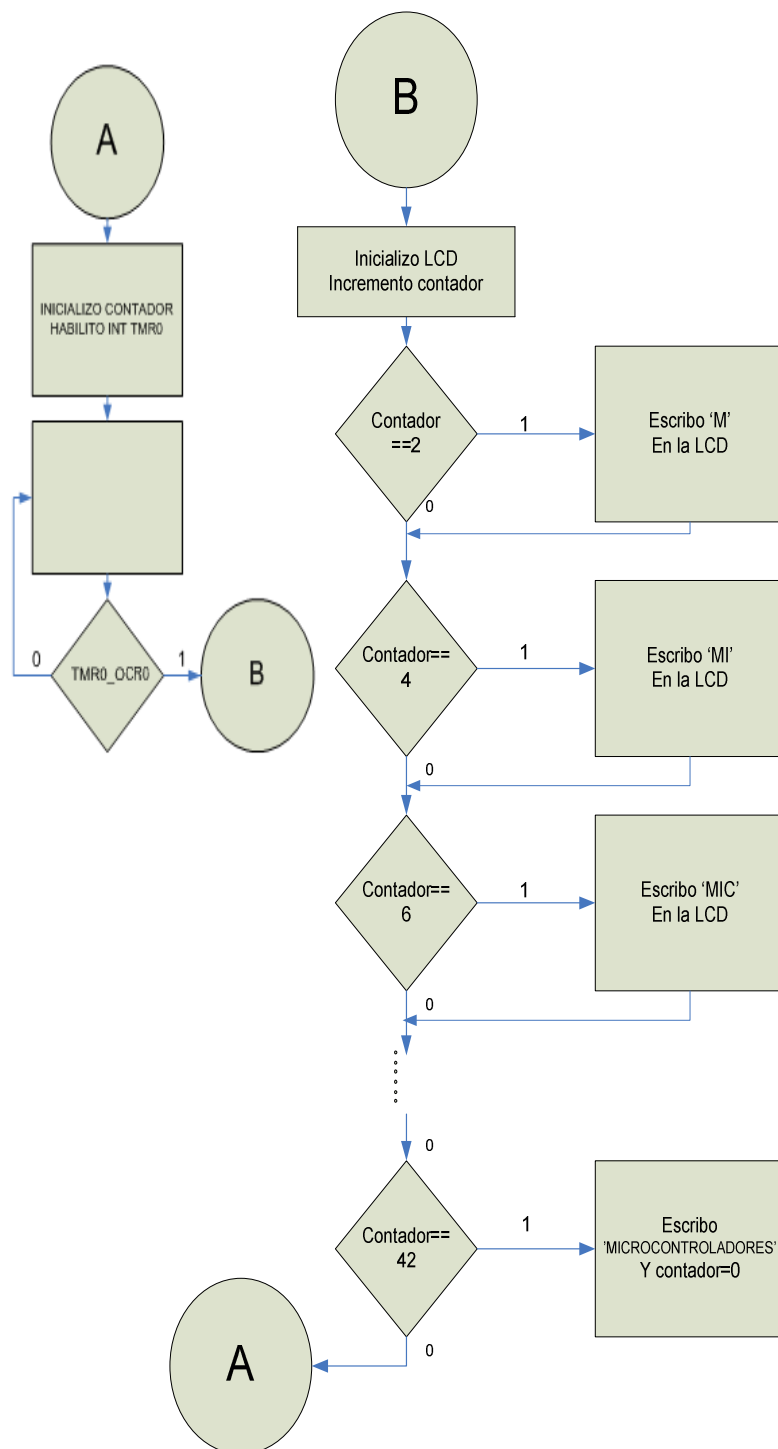
#### Butterfly

Cada vez que se genera una igualdad de comparación por el timer0 se incrementa un contador con el fin de presentar una porción del mensaje, cada letra ingresada se va desplazando por el delay que le damos al TIMER0.

### 4.1.2 DIAGRAMA DE BLOQUES DEL EJERCICIO N°1



## 4.1.3 DIAGRAMA DE FLUJO DEL EJERCICIO N° 1

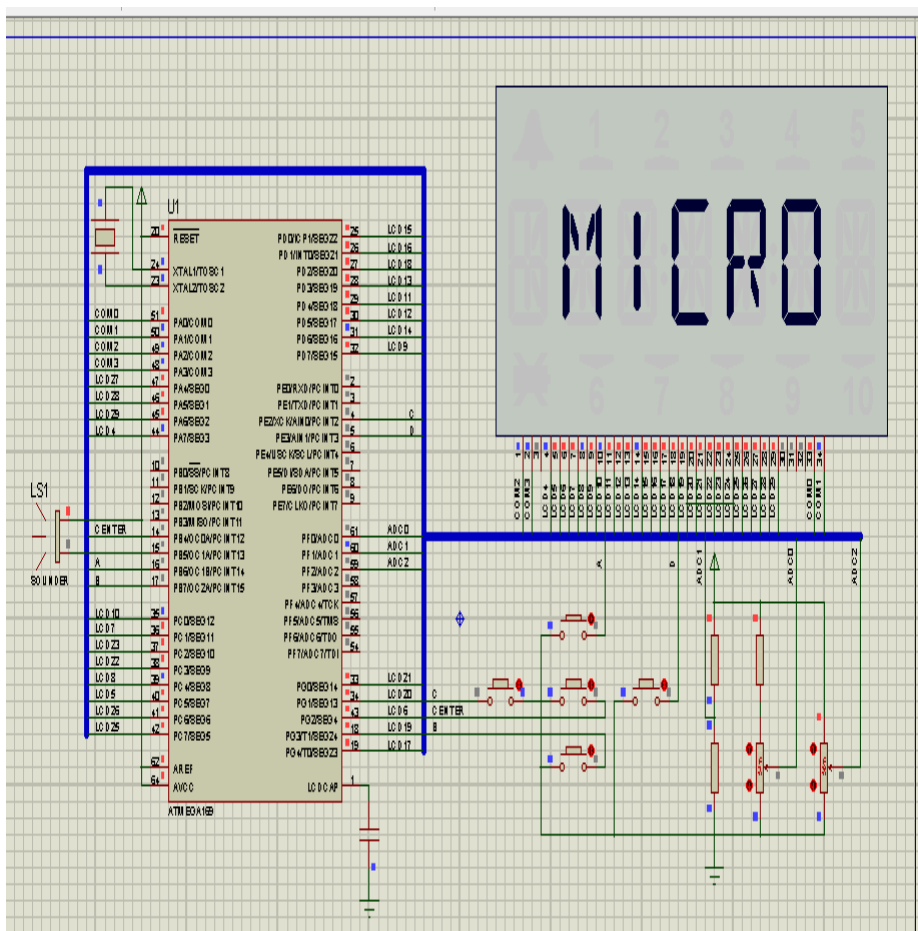




Al principio inicializo el contador y habilito el TMR0, cada vez que existe una igualdad de comparación se incrementara un contador, la cual usaremos para generar delay, a continuación se inicializa la LCD y empezamos a mostrar letra por letra haciendo que cada una se valla mostrando en cada porción de la AVR Butterfly, al terminar completamente la palabra el contador usado toma el valor de cero y regresa al estado inicial para una nueva presentación.

### 4.1.4 REPRESENTACION GRAFICA DEL ISIS EJERCICIO

Nº 1.





```

uint8_tStrEnd;                //Variable para fin de palabra
uint8_tScrollMode;           //Modo desplazamiento
uint8_tScrollCount;          //Contador de desplazamiento
uint8_tUpdateLCD;            //Para que pregunte por la interrupcion
    uint16_tLCD_SegTable[]PROGMEM=//Tabla de simbolos y letras a graficar
{
0xEAA8,// '*'
0x2A80,// '+'
0x4000,// ','
0x0A00,// '-'
0x0A51,// '.' Degree sign
0x4008,// '/'
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)
0x8008,// '<'
0x1A00,// '='
0x4020,// '>'
0x0000,// '?' (Not defined)
0x0000,// '@' (Not defined)
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')
0x1441, // '['
0x8020, // '\'
0x1111, // ']'
0x0000, // '^' (Not defined)
0x1000, // '_'
};
void LCD_Init(void)
{
LCDCCR=0x0F; // Selecciona la fuente de reloj asincrona, habilita todos los
COM pins and habilita todos los segmenteos de los pines.

LCDCRB=(1<<LCDCS)|(3<<LCDMUX0)|(7<<LCDPM0);
// habilita al LCD prescaler y da velocidad de trama de 32,0 Hz

```

```

LCDFRR=(0<<LCDPS0)|(7<<LCDCD0);
// Habilitar la interfaz LCD y configurar la forma de onda de baja potencia

LCDCRA=(1<<LCDEN)|(1<<LCDAB);
//Habilitar interrupción por inicio de trama LCD:
LCDCRA|=(1<<LCDIE);
}
voidLCD_puts_f(constuint8_t*FlashData) // Sirve para escribir un valor
fijo en la lcd.
{
uint8_tStrBuff[LCD_TEXTBUFFER_SIZE]; // creo un arreglo de
tamaño 20.strcpy_P(StrBuff,FlashData); //sobre escribo el contido
de flashdata en strbuff.
LCD_puts(StrBuff);
}

voidLCD_puts(uint8_t*Data) //Funcionpa escribir palabras en el display.
{
uint8_tLoadB;//Declara variable.
for(LoadB=0;LoadB<20;LoadB++)//Se repite 20 veces.
{
uint8_tCByte=*(Data++); //Va ir almacenando el valor q
contenga*(data++)enbyte.
if((CByte>='*')&&(CByte<='z')) //pregunto si sta en el rango.
TextBuffer[LoadB]=((CByte==' ')?0xFF:(CByte-'*')); //si es spacio pongo
ff si pertenece al rango resto este valor con *.
elseif(CByte==0x00) //Caso contrario q sea enter q ia tas al final de la
cadena.
break;
else
TextBuffer[LoadB]=0xFF; //Caso contrario grafica espacio
.
}

```

```

ScrollMode=((LoadB>6)?TRUE:FALSE); //Modo desplazamiento se activa si es
mayor a 6.
ScrollCount=LCD_DELAYCOUNT_DEFAULT;//Contador de desplazamiento es igual a
10.
for(uint8_tNulls=0;Nulls<7;Nulls++) //For para llenar los espacios en
blanco.
TextBuffer[LoadB++]=0xFF; //Llenar los espacios en blanco.

TextBuffer[LoadB]=0x00; //Para saber si es el final de la
cadena.
StrStart=0; //Inicio de la cadena o del estring es cero.
StrEnd=LoadB; //Final de la cadena.
UpdateLCD=TRUE; //Vas a subir un dato a la lcd le envio
un true
}

voidLCD_WriteChar(uint8_tByte,uint8_tDigit) //programa que me ayuda a
imprimir letra a letra en la casilla
{
uint16_tSegData=0x00; //Variable para ir guardando los caracteres
uint8_t*BuffPtr=(&SegBuffer[0]+(Digit>>1)); //Apunta el buffer de datos
de la LCD y guarda en un arreglo SegBuffer para tener la direccion en el
putneroBuffPtr
if(Byte!=0xFF) //Si es diferente de espacio
SegData=pgm_read_word(&LCD_SegTable[Byte]); //Obtiene el SCC (Segment
Control Code) de la tabla y lo guarda en SegData

for(uint8_tBNib=0;BNib<4;BNib++)//Para asegurarme de cojer todo el nibble
{
uint8_tMask=0xF0; //Posiciones 0,2,4 de la LCD
uint8_tMaskedSegData=(SegData&0x0000F); //Aisla el nibble menos signiicativo
y coje un nibble a la vez

if(Digit&0x01) //Si Digit apunta a la primera posicion

```

```

{
Mask=0x0F;           //Aputno a las posiciones 1,3,5 de la LCD
MaskedSegData<<=4;   //Desplazo el nibble 4 posiciones a la
                    izquierda
}
*BuffPtr=((*BuffPtr&Mask)|MaskedSegData); //Escribe el nibble en el buffer
de la LCD
SegData>>=4; //Desplaza el SCC 4 posiciones a la derecha pacojer el
proximonibble
BuffPtr+=5; //Incrementa el buffer 5 posiciones para que el siguiente
nibble sea en LCCDRx+5
} //y la siguiente en la LCCDRx+10 y asi hasta ocupar todos los
espacios
}

ISR(LCD_vect) //interrupcion de tipo LCD
{
if(ScrollMode) //modo desplazamiento
{
if(!(ScrollCount)) //si sea diferente de 10
UpdateLCD=TRUE;
else
ScrollCount--; //como no es verdadero disminuye
}

if(UpdateLCD)
{
for(uint8_tCharacter=0;Character<6;Character++) // crea un lazo donde
caracter es el valor a recorrer en el display
{
uint8_tByte=(StrStart+Character); // byte contendra el valor de la suma de
strstart + carácter
if(Byte>=StrEnd) // si byte es mayor o igual a strendosea hasta fin del
string

```

```

Byte=TextBuffer[Byte-StrEnd];      //si la instruccion pasa va almacenar
spacios en blanco
Else
Byte=TextBuffer[Byte];             // se almacena el arreglo en byte
LCD_WriteChar(Byte,Character);     // es escrito y puesto en la posicion q
indica caracter
}if(StrStart++==StrEnd)           // comprobar si se termina de escribir la
palabra
StrStart=1;

ScrollCount=LCD_SCROLLCOUNT_DEFAULT;//contador de desplazamientos
UpdateLCD=FALSE;                 //si vuelve a ocurrir la interrupcion no ingrese
}
for(uint8_tLCDChar=0;LCDChar<LCD_SEGBUFFER_SIZE;LCDChar++)
*(pLCDREG+LCDChar)=SegBuffer[LCDChar]; // Escribe el caracter y lo
almacena
}
ISR(TIMER0_OVF_vect)             // Vector interrupción
{
LCD_Init();
contador++;
if(contador==2){                // Cuenta hasta 2
LCD_puts("  M");                // Escribo la M
}
if(contador==4){                // Cuenta hasta 4
LCD_puts("  MI");               // Escribo la MI
}
}
if(contador==6){                // Cuenta hasta 6
LCD_puts("  MIC");              // Escribo la MIC
}
}
if(contador==8){                // Cuenta hasta 8
LCD_puts("  MICR");             // Escribo la MICR
}
}
if(contador==10){
LCD_puts("MICRO");
}

```



```
}  
if(contador==12){  
LCD_puts("ICRON");  
}  
if(contador==14){  
LCD_puts("CRONT");  
}  
if(contador==16){  
LCD_puts("RONTR");  
}  
if(contador==18){  
LCD_puts("ONTRA");  
}  
if(contador==20){  
LCD_puts("NTRAL");  
}  
if(contador==22){  
LCD_puts("TRALA");  
}  
if(contador==24){  
LCD_puts("RALAD");  
}  
    if(contador==26){  
LCD_puts("RALAD");  
}  
if(contador==28){  
LCD_puts("ALADO");  
}  
if(contador==30){  
LCD_puts("LADOR");  
}  
if(contador==32){  
LCD_puts("ADORE");  
}  
}
```

```

if(contador==34){
LCD_puts("DORES");
}

if(contador==36){
LCD_puts("ORES");
}
if(contador==38){
LCD_puts("RES");
}
if(contador==40){
LCD_puts("ES");
}
if(contador==42){
LCD_puts("S");
}
if(contador==44){
LCD_puts("");
contador=0;
}
} // PROGRAMA PRINCIPAL
int main(void)
{
contador=0;
DDRD=0B11111111;
OCR0A=255;
sei();// habilito interrupciones
TCCR0A=0B100101;//Normal Port - Normal - Flanco de Bajada
TIMSK0=0B0000001;//Habilito Interrupción por desbordamiento
MCUCR=0B0000010;//INT0 Flanco de Bajada
for(;;){} //Bucle Infinito}

```

## 4.2 EJERCICIO

### 4.2.1 EJERCICIO #2.- Cronometro presentado en el display del

AVR butterfly haciendo uso de las interrupciones del

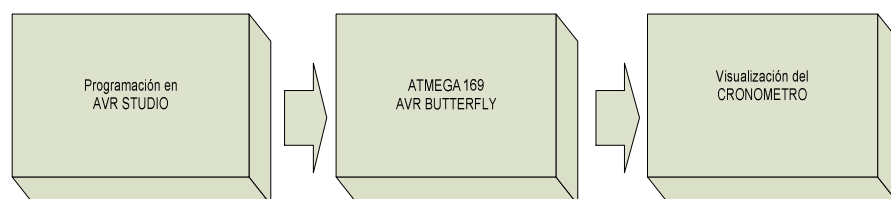
TIMER0

Cada vez q se genera una interrupción por el TIMER0 se incrementa el contador para crear el cronometro, este se presentara en el display del AVR BUTTERFLY.

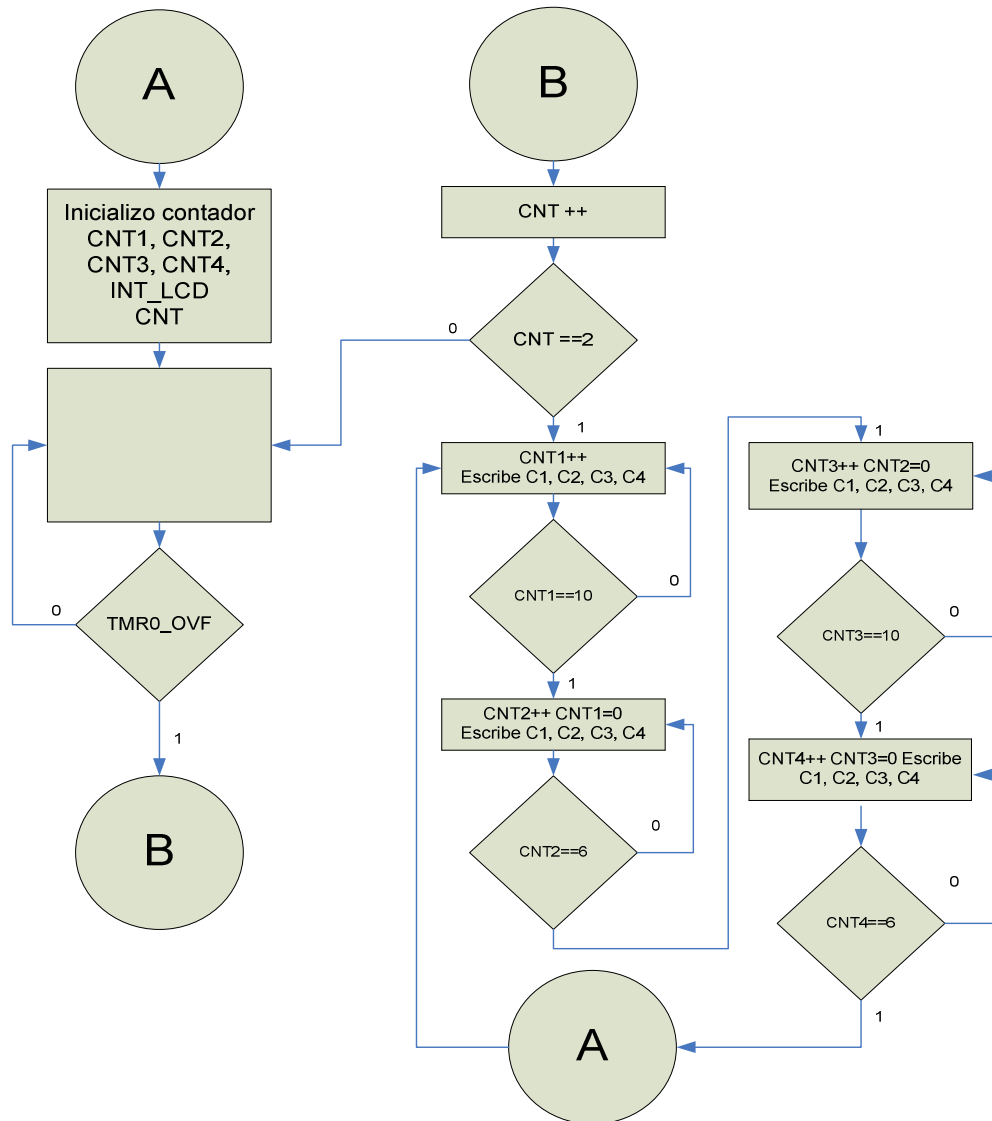
Cada interrupción incrementa un contador la cual nos ayudara a separar las variables de segundos y minutos, estas están separadas por una letra “H” cual la creamos con un grupo de nibbles para una mejor presentación.

### 4.2.2 DIAGRAMA DE BLOQUES DEL DIAGRAMA DE

BLOQUES DEL N° 2



### 4.2.3 DIAGRAMA DE FLUJO DEL EJERCICIO N° 2

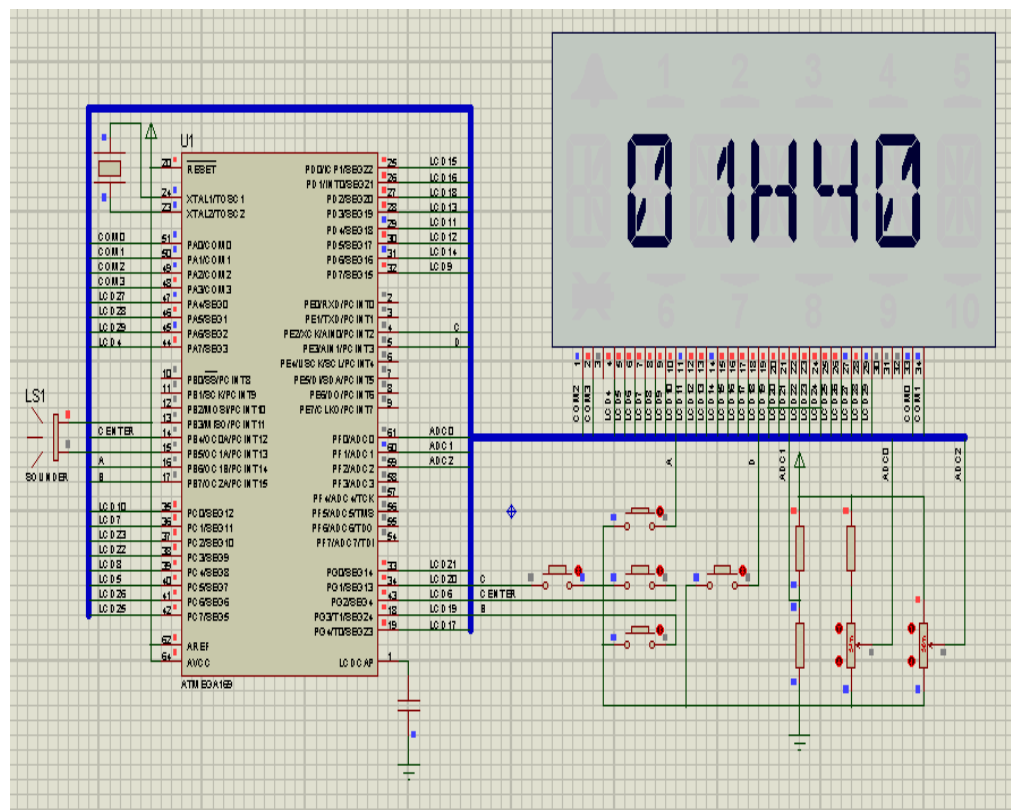


Al principio inicializo cuatro contadores bajo el nombre de CNT1, CNT2, CNT3, CNT4 y habilito el TMRO, una vez que ocurra una interrupción me voy al siguiente estado en donde inicializo el LCD e incremento la variable CNT hasta que su valor se igual a 2 y nos iríamos a otro estado,

caso contrario seguiremos en el mismo estado, en el estado siguiente el CNT se iguala a 0 y nuestro contador CNT1 y se cicla incrementando su valor hasta que llegue a 10, en el siguiente estado, al ocurrir este suceso nuestro contador CNT2 se incrementa y encendamos al contador CNT1 y se cicla hasta que CNT2 llegue al valor de 6, en el siguiente estado incrementa nuestro contador CNT3 y encendamos nuestro contador CNT2 y CNT1 y se cicla hasta que CNT3 llegue al valor de 10, en el siguiente estado se incrementa nuestro contador CNT4 y encendamos nuestro anteriores contadores y se cicla hasta que llegue al valor de 6 y para una mejor presentación creamos con nibbles la letra “H” para una mejor presentación.

### 4.2.4 REPRESENTACION GRAFICA DEL ISIS EJERCICIO

Nº2



## 4.2.5 CODIGO DE SIMULACION EN EL AVR STUDIO DEL

### EJERCICIO #2

```

/*****
*****
***
**      MICROCONTROLADORES AVANZADOS      **
**  Profesor de Catedra:   Ing.CarlosValdiviezo.      **
**  Integrantes de grupo:  Patricio Salazar Moreira.    **
**                          Luis Ortiz Moran.          **
**  Título:   Cronometro presentado en el display del AVR butterfly**
**              haciendo uso de las interrupciones del TIMER0 **
**  Descripción:Cada vez q se genera una interrupcion por el TIMER0 **
**              se incrementa el contador pa crear el cronometro,      **
**              este se presentara en el displays del AVR BUTTERFLY    **
**                                                                    **
*****
*****/
#include<avr/io.h>    // Agrega todas las funciones de las librerias de
ATMEL.
#include<avr/interrupt.h>    // Libreria de interrupciones.
#include<util/delay.h>      // Librería para generar retardos
#include "LCD_Driver.h"    // Libreria para LCD

                // Variables para cronometro
unsignedcharcnt;
unsignedcharcnt1;
unsignedcharcnt2;
unsignedcharcnt3;
unsignedcharcnt4;

                //Variables temporales

uint8_tTextBuffer[LCD_TEXTBUFFER_SIZE+7]; //Arreglo para guardar letras

```

```

uint8_tSegBuffer[LCD_SEGBUFFER_SIZE];    //Arreglo para guardar letras
uint8_tStrStart;                          //Variable para inicio de palabra
uint8_tStrEnd;                            //Variable para fin de palabra
uint8_tScrollMode;                       //Modo desplazamiento
uint8_tScrollCount;                      //Contador de desplazamiento
uint8_tUpdateLCD;                        //Para que pregunte por la interrupcion
uint16_tLCD_SegTable[]PROGMEM=          //Tabla de simbolos y letras a graficar
{
0xEAA8, // '*'
0x2A80, // '+'
0x4000, // ','
0x0A00, // '-'
0x0A51, // '.' Degree sign
0x4008, // '/'
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)
0x8008, // '<'
0x1A00, // '='
0x4020, // '>'
0x0000, // '?' (Not defined)
0x0000, // '@' (Not defined)
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')
0x1441, // '['
0x8020, // '\'
0x1111, // ']'
0x0000, // '^' (Not defined)
0x1000, // '_'
};
void LCD_Init(void)
{
  LCDCCR=0x0F;
  // Selecciona la fuente de reloj asincrona, habilita todos los COM pins and
  // habilita todos los segmenteos de los pines.

```



```

LCDCRB=(1<<LCDCS)|(3<<LCDMUX0)|(7<<LCDPM0);

// habilita al LCD prescaler y da velocidad de trama de 32,0 Hz
LCDFRR=(0<<LCDPS0)|(7<<LCDCD0);

// Habilitar la interfaz LCD y configurar la forma de onda de baja potencia
LCDCRA=(1<<LCDEN)|(1<<LCDAB);

//Habilitar interrupción por inicio de trama LCD
LCDCRA|=(1<<LCDIE);
}
voidLCD_puts_f(constuint8_t*FlashData) // Sirve para escribir un valor
fijo en la lcd
{
uint8_tStrBuff[LCD_TEXTBUFFER_SIZE]; // creo un arreglo de tamaño 20
strcpy_P(StrBuff,FlashData); //sobre escribo el contido de flashdata en
strbuff
LCD_puts(StrBuff);
}

voidLCD_puts(uint8_t*Data) //Funcionpa escribir palabras en el display
{
uint8_tLoadB;//Declara variable

for(LoadB=0;LoadB<20;LoadB++) //Se repite 20 veces
{
uint8_tCByte=*(Data++); //Va ir almacenando el valor q contenga
*(data++) en cbyte

if((CByte>='*')&&(CByte<='z')) //pregunto si sta en el rango
TextBuffer[LoadB]=((CByte==' ')?0xFF:(CByte-'*')); //si es spacio pongo
ff si pertenece al rango resto este valor con *
elseif(CByte==0x00) //Caso contrario q sea enter q ia tas al final de la
cadena

```

```

break;
else
TextBuffer[LoadB]=0xFF;          //Caso contrario grafica espacio
}
ScrollMode=((LoadB>6)?TRUE:FALSE); //Modo desplazamiento se activa si es
mayor a 6
ScrollCount=LCD_DELAYCOUNT_DEFAULT;//Contador de desplazamiento es igual a
10
for(uint8_tNulls=0;Nulls<7;Nulls++)//For para llenar los espacios en blanco

TextBuffer[LoadB++]=0xFF;          //Llenar los espacios en blanco
TextBuffer[LoadB]=0x00;          //Para saber si es el final de la cadena
StrStart=0;                      //Inicio de la cadena o del estring es cero
StrEnd=LoadB;                    //Final de la cadena
UpdateLCD=TRUE;                 //Vas a subir un dato a la lcd le envié un true
}
voidLCD_WriteChar(uint8_tByte,uint8_tDigit) //programa que me ayuda a
imprimir letra a letra en la casilla
{
uint16_tSegData=0x00;          //Variable para ir guardando los caracteres
uint8_t*BufPtr=&SegBuffer[0]+(Digit>>1); //Apunta el buffer de datos
de la LCD y guarda en un arreglo SegBuffer para tener la dirección en el
putneroBufPtr

if(Byte!=0xFF)                 //Si es diferente de espacio
SegData=pgm_read_word(&LCD_SegTable[Byte]); //Obtiene el SCC (Segment
Control Code) de la tabla y lo guarda en SegData

for(uint8_tBNib=0;BNib<4;BNib++)//Para asegurarme de cojer todo el nibble
{
uint8_tMask=0xF0;           //Posiciones 0,2,4 de la LCD
uint8_tMaskedSegData=(SegData&0x0000F); //Aísla el nibble menos
significativo y coge un nibble a la vez

```

```

if(Digit&0x01)          //Si Digit apunta a la primera posición
{
Mask=0x0F;              //Apunto a las posiciones 1,3,5 de la LCD
MaskedSegData<<=4;     //Desplazo el nibble 4 posiciones a la izquierda
}

*BuffPtr=((*BuffPtr&Mask)|MaskedSegData); //Escribe el nibble en el buffer
de la LCD
SegData>>=4;           //Desplaza el SCC 4 posiciones a la derecha para coger
el próximo nibble
BuffPtr+=5;           //Incrementa el buffer 5 posiciones para que el siguiente
nibble sea en LCCDRx+5
} //y la siguiente en la LCCDRx+10 y así hasta ocupar todos los
espacios
}

ISR(LCD_vect)           //interrupción de tipo LCD
{
if(ScrollMode)         //modo desplazamiento
{
if(!(ScrollCount))    //si sea diferente de 10
UpdateLCD=TRUE;
else
ScrollCount--; //como no es verdadero disminuye
}

if(UpdateLCD)
{
for(uint8_tCharacter=0;Character<6;Character++) // crea un lazo donde
caracter es el valor a recorrer en el display{
uint8_tByte=(StrStart+Character); // byte contendrá el valor de la suma de
strstart + carácter
}
}
}

```

```

if(Byte>=StrEnd)    // si byte es mayor o igual a strendosea hasta fin del
string
Byte=TextBuffer[Byte-StrEnd];    //si la instrucción pasa va almacenar
spacios en blanco
else
Byte=TextBuffer[Byte];    // se almacena el arreglo en
byteLCD_WriteChar(Byte,Character); // es escrito y puesto en la posición q
indica carácter
}

if(StrStart++==StrEnd)    // comprobar si se termina de escribir la
palabra
StrStart=1;
ScrollCount=LCD_SCROLLCOUNT_DEFAULT;    //contador de desplazamientos
UpdateLCD=FALSE;    //si vuelve a ocurrir la interrupción ya no ingrese
}

for(uint8_tLCDChar=0;LCDChar<LCD_SEGBUFFER_SIZE;LCDChar++)
*(pLCDREG+LCDChar)=SegBuffer[LCDChar];
}

ISR(TIMER0_OVF_vect)    // Vector interrupción
{
LCD_Init();
cnt++;    //Incrementocnt
if(cnt==2){    //Comparocnt con 2
cnt1++;    //Incremento cnt1
LCD_WriteChar(30,2);    //presento la H
LCD_WriteChar((6+cnt3),1);    //Incremento cnt3 mas 6 para empezar
desde los numeros y pongo en casilla 1
LCD_WriteChar((6+cnt4),0); //Incremento cnt4 mas 6 para empezar desde los
numeros y pongo en casilla 0
LCD_WriteChar((6+cnt1),4);    //Incremento cnt1 mas 6 para empezar
desde los numeros y pongo en casilla 4
}
}

```

```

LCD_WriteChar((6+cnt2),3);          //Incremento cnt2 mas 6 para empezar
desde los numeros y pongo en casilla 3
if(cnt1==10)                        //Si cnt1 es igual a 10
{
cnt1=0;                             //cnt1=0
cnt2++;                             //Incremento cnt2

LCD_WriteChar((6+cnt3),1);
//Incremento cnt3 mas 6 para empezar desde los números y pongo en casilla
1
LCD_WriteChar((6+cnt4),0);
//Incremento cnt4 mas 6 para empezar desde los números y pongo en casilla
0

LCD_WriteChar((6+cnt1),4);
//Incremento cnt1 mas 6 para empezar desde los números y pongo en casilla
4
LCD_WriteChar((6+cnt2),3);
//Incremento cnt2 mas 6 para empezar desde los números y pongo en casilla
3

}
if(cnt2==6)                          //Si cnt2=0
{
cnt2=0;                             //cnt2=0

cnt3++;                             //Incremento cnt3

LCD_WriteChar((6+cnt3),1);
//Incremento cnt3 mas 6 para empezar desde los números y pongo en casilla
1
LCD_WriteChar((6+cnt4),0);
//Incremento cnt4 mas 6 para empezar desde los números y pongo en casilla
0

```

```

LCD_WriteChar((6+cnt1),4);
//Incremento cnt1 mas 6 para empezar desde los números y pongo en casilla
4
LCD_WriteChar((6+cnt2),3);
//Incremento cnt2 mas 6 para empezar desde los números y pongo en casilla
3

}

if(cnt3==10)                //Si cnt3 = 10
{
cnt3=0;                    //cnt3 = 0
cnt4++;                    //Incremento cnt4

LCD_WriteChar((6+cnt3),1);
//Incremento cnt3 mas 6 para empezar desde los números y pongo en casilla
1
LCD_WriteChar((6+cnt4),0);
//Incremento cnt4 mas 6 para empezar desde los números y pongo en casilla
0

LCD_WriteChar((6+cnt1),4);
//Incremento cnt1 mas 6 para empezar desde los números y pongo en casilla
4
LCD_WriteChar((6+cnt2),3);
//Incremento cnt2 mas 6 para empezar desde los números y pongo en casilla
3

}

if(cnt4==6)                //Si cnt4 = 6
{
cnt4=0;                    //cnt4 = 0

```

```

LCD_WriteChar((6+cnt3),1);
//Incremento cnt3 mas 6 para empezar desde los números y pongo en casilla
1
LCD_WriteChar((6+cnt4),0);
//Incremento cnt4 mas 6 para empezar desde los números y pongo en casilla
0

LCD_WriteChar((6+cnt1),4);
//Incremento cnt1 mas 6 para empezar desde los números y pongo en casilla
4
LCD_WriteChar((6+cnt2),3);
//Incremento cnt2 mas 6 para empezar desde los números y pongo en casilla
3
}
cnt=0;           //cnt=0
}

}

// PROGRAMA PRINCIPAL
intmain(void)
{
cnt1=0;
cnt2=0;
DDRD=0B11111111;           //Salida en el PORTD
OCR0A=255;
sei();           //habilitar interrupciones
TCCR0A=0B1000101           //Normal Port - Normal - Flanco de Bajada
TIMSK0=0B00000001;           //Habilito Interrupción por desbordamiento
MCUCR=0B00000010;           //INT0 Flanco de Bajada
for(;;){}           //Bucle Infinito

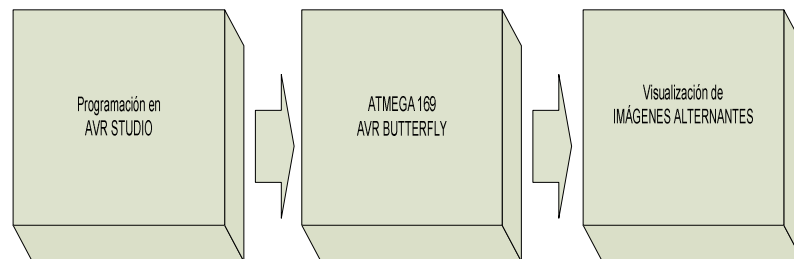
```

### 4.3 EJERCICIO

#### 4.3.1 EJERCICIO #3.- Desplazamiento de una animación en la LCD del AVR Butterfly haciendo uso de las interrupciones del TIMER0.

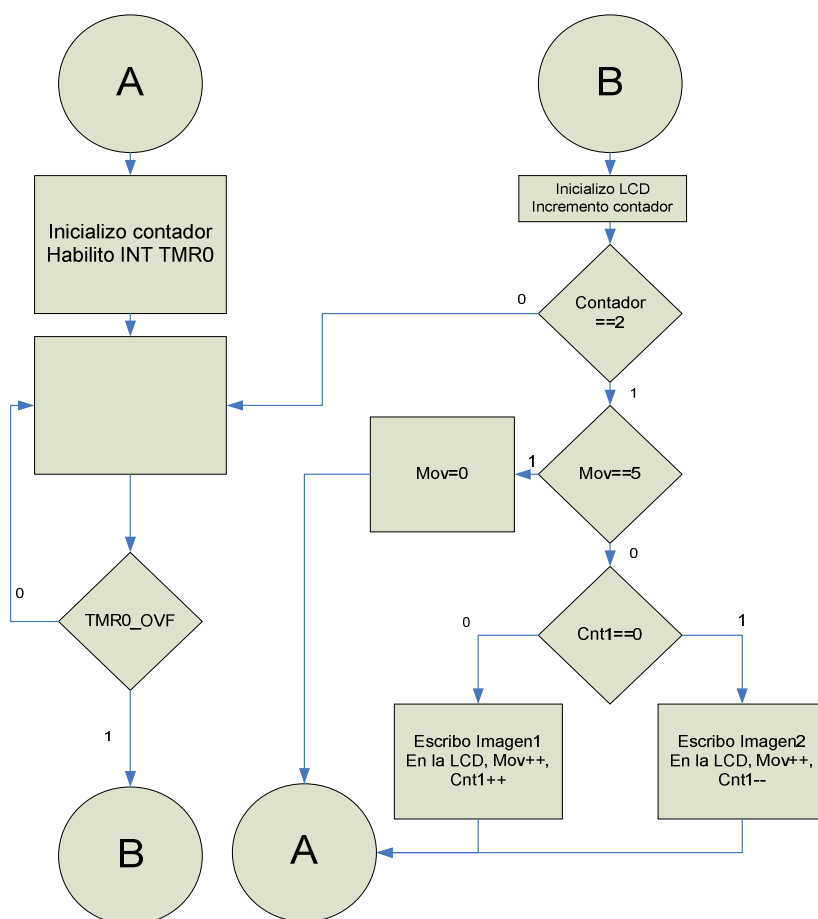
Cada vez q se genera una interrupción por el TIMER0 se incrementa un contador la cual nos sirve para presentar una animación diferente, cada animación es graficada en un segmento de la LCD estas animaciones son creadas por los nibbles del display que se va desplazando por el delay que le damos por el vector de interrupción del TIMER0

#### 4.3.2 DIAGRAMA DE BLOQUES DEL EJERCICIO N°3





### 4.3.3 DIAGRAMA DE FLUJO DEL EJERCICIO N° 3

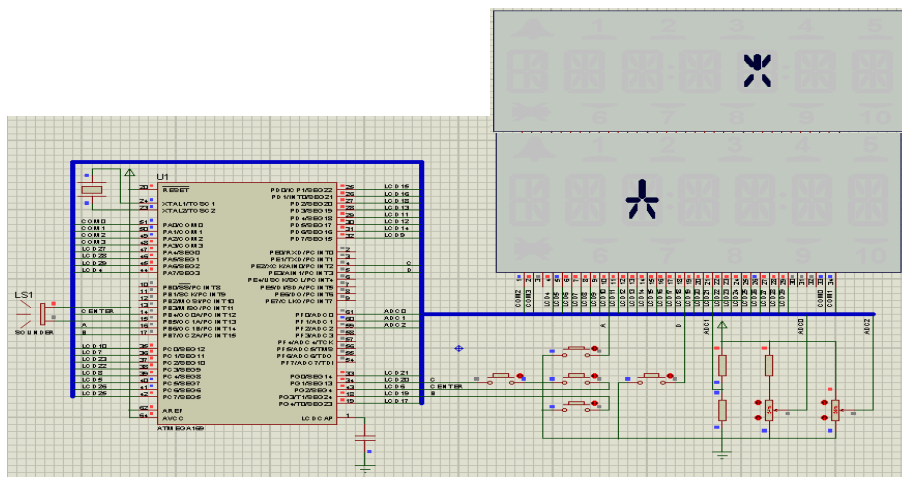


Al principio inicializo el contador y habilito el TMR0, cada vez que existe una interrupción se incrementara un contador, la cual usaremos para generar delay, a continuación se inicializa la LCD y comenzamos a incrementar un contador que al sumar a un numero 2 se va a un estado mov caso contrario regresa al estado de interrupción, sai al contador le sumamos 2 nos vamos a mov, como mov es igual a 0 se va directamente a

un contador CNT1, al principio mov es igual a “0”, entonces se va escribir en imagen1 en la LCD, incrementado el mov y el contador CNT1, y regreso otra vez para inicializar el LCD e incremento al contador dos numero mas, pero ahora mov ya no es 0 sino 1, pero aun mov no es igual a cinco, y nos vamos a al contador CNT1 pero esta vez aya no es igual a 0 sino a 1, por lo que escribimos imagen1 en la LCD, incrementando el mov y decrementando el contador CNT1, y ocurrirá lo mismo hasta que mov sea igual a 5, en ese caso, nos vamos a un estado en donde mov sea igual a cero y inicializamos el LCD e incrementamos el contador

### 4.3.4 REPRESENTACION GRAFICA DEL ISIS EJERCICIO

#### Nº 3



### 4.3.5 CODIGO DE SIMULACION EN EL AVR STUDIO DEL EJERCICIO N° 3

```

/*****
*****
**      MICROCONTROLADORES AVANZADOS      **
** Profesor de Catedra:   Ing.Carlos Valdiviezo.      **
** Integrantes de grupo:  Patricio Salazar Moreira.    **
**                        Luis Ortiz Moran.            **
** Título:Desplazamiento de una animación en la LCD del AVR butterfly**
**      haciendo uso de las interrupciones del TIMER0.      **
**                                                              **
** Descripción:Cada vez q se genera una interrupción por el timer0 se**
**      presenta una animación diferente, cada animación es  **
**      graficada en un segmento de la LCD estas animaciones son**
**      creadas por los nibbles del display que se va desplazando**
**      por el delay que le damos al interrumpir al TIMER0.  **
*****
*****/
#include<avr/io.h>    // Agrega todas las funciones de las librerías de
ATMEL.
#include<avr/interrupt.h>    // Librería de interrupciones.
#include<util/delay.h>      // Librería para generar retardos.
#include "LCD_Driver.h"     // Librería para LCD.

//Variables auxiliares
unsigned char cnt;        //Contador

unsigned char cnt1;      //Contador1
unsigned char cnt2;      //Contador2
unsigned char mov;       //Variable para movimiento

```

```

//Variables temporales

uint8_tTextBuffer[LCD_TEXTBUFFER_SIZE+7]; //Arreglo para guardar letras

uint8_tSegBuffer[LCD_SEGBUFFER_SIZE]; //Arreglo para guardar letras
uint8_tStrStart; //Variable para inicio de palabra
uint8_tStrEnd; //Variable para fin de palabra
uint8_tScrollMode; //Modo desplazamiento
uint8_tScrollCount; //Contador de desplazamiento
uint8_tUpdateLCD; //Para que pregunte por la
interrupción

uint16_tLCD_SegTable[]PROGMEM= //Tabla de símbolos y letras a
graficar
{
0xC0A8, // '*'
0xCA80, // '+'
0x4000, // ','
0x0A00, // '-'
0x0A51, // '.' Degree sign
0x4008, // '/'
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)

```

```
0x8008, // '<'
0x1A00, // '='
0x4020, // '>'
0x0000, // '?' (Not defined)
0x0000, // '@' (Not defined)
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')
0x1441, // '['
0x8020, // '\'
0x1111, // ']'
```

```

0x0000, // '^' (Not defined)
0x1000 // '_'
};

void LCD_Init(void)
{
    LCDCCR=0x0F;
    // Selecciona la fuente de reloj asincrona, habilita todos los COM pins and
    // habilita todos los segmentos de los pines.
    LCDCRB=(1<<LCDCS)|(3<<LCDMUX0)|(7<<LCDPM0);

    // habilita al LCD prescaler y da velocidad de trama de 32,0 Hz
    LCDFRR=(0<<LCDPS0)|(7<<LCDCD0);

    // Habilitar la interfaz LCD y configurar la forma de onda de baja potencia
    LCD CRA=(1<<LCDEN)|(1<<LCDAB);

    //Habilitar interrupción por inicio de trama LCD
    LCD CRA|=(1<<LCDIE);
}

void LCD_puts_f(const uint8_t*FlashData) // Sirve para escribir un valor
fijo en la lcd.
{
    uint8_t StrBuff[LCD_TEXTBUFFER_SIZE]; // creo un arreglo de
tamaño 20.

    strcpy_P(StrBuff,FlashData); //sobre escribo el contenido de flashdata en
strbuff
    LCD_puts(StrBuff);
}

```

```

void LCD_puts(uint8_t*Data) //Funcionpa escribir palabras en el
display
{
uint8_t LoadB; //Declara variable.

for(LoadB=0; LoadB<20; LoadB++) //Se repite 20 veces
{
uint8_t CByte=*(Data++); //Va ir almacenando el valor q contenga
*(data++) en cbyte

if((CByte>='*') && (CByte<='z')) //pregunto si sta en el rango
TextBuffer[LoadB]=((CByte==' ') ? 0xFF : (CByte - '*')); //si es
espacio pongo ff si pertenece al rango resto este valor con *
elseif(CByte==0x00) //Caso contrario q sea enter q ia tas al final de la
cadena
break;
else
TextBuffer[LoadB]=0xFF; //Caso contrario grafica espacio
}

ScrollMode=((LoadB>6)?TRUE:FALSE); //Modo desplazamiento se activa si es
jmayor a 6
ScrollCount=LCD_DELAYCOUNT_DEFAULT; //Contador de desplazamiento es igual a
10

for(uint8_t Nulls=0; Nulls<7; Nulls++) //For para llenar los espacios en blanco
TextBuffer[LoadB++]=0xFF; //Llenar los espacios en blanco

TextBuffer[LoadB]=0x00; //Para saber si es el final de la cadena
StrStart=0; //Inicio de la cadena o del esting es cero
StrEnd=LoadB; //Final de la cadena

```

```

UpdateLCD=TRUE;           //Vas a subir un dato a la lcd le envié un true
}

voidLCD_WriteChar(uint8_tByte,uint8_tDigit) //programa que me ayuda a
imprimir letra a letra en la casilla
{
uint16_tSegData=0x00;     //Variable para ir guardando los caracteres

uint8_t*BuffPtr=&SegBuffer[0]+(Digit>>1)); //Apunta el buffer de datos
de la LCD y guarda en un arreglo SegBuffer para tener la dirección en el
putneroBuffPtr

if(Byte!=0xFF)           //Si es diferente de espacio
SegData=pgm_read_word(&LCD_SegTable[Byte]); //Obtiene el SCC (Segment
Control Code) de la tabla y lo guarda en SegData

for(uint8_tBNib=0;BNib<4;BNib++) //Para asegurarme de coger todo el
nibble
{
uint8_tMask=0xF0;       //Posiciones 0,2,4 de la LCD
uint8_tMaskedSegData=(SegData&0x000F); //Aísla el nibble menos
significativo y coje un nibble a la vez

if(Digit&0x01) //Si Digit apunta a la primera posición
{
Mask=0x0F;             //Aputno a las posiciones 1,3,5 de la LCD
MaskedSegData<<=4;    //Desplazo el nibble 4 posiciones a la izquierda
}

*BuffPtr=((*BuffPtr&Mask)|MaskedSegData); //Escribe el nibble en el buffer
de la LCD
SegData>>=4; //Desplaza el SCC 4 posiciones a la derecha pacojer el
proximonibble
}

```



```

BuffPtr+=5; //Incrementa el buffer 5 posiciones para que el siguiente
nibble sea en LCCDRx+5
} //y la siguiente en la LCCDRx+10 y asi hasta ocupar todos los espacios
}
ISR(LCD_vect) //Interrupcion de tipo LCD
{
if(ScrollMode) //Modo desplazamiento
{
if(!(ScrollCount)) //si sea diferente de 10

UpdateLCD=TRUE; //como no es verdadero disminuye
else
ScrollCount--;
}

if(UpdateLCD)
{
for(uint8_tCharacter=0;Character<6;Character++) // crea un lazo donde
caracter es el valor a recorrer en el display
{
uint8_tByte=(StrStart+Character); // byte contendra el valor de la suma de
strstart + caracter

if(Byte>=StrEnd) // si byte es mayor o igual a strendosea hasta fin del
string
Byte=TextBuffer[Byte-StrEnd]; //si la instruccion pasa va almacenar
spacios en blanco
else
Byte=TextBuffer[Byte]; // se almacena el arreglo en byte

LCD_WriteChar(Byte,Character); // es escrito y puesto en la posicion q
indica caracter
}
}

```

```

if(StrStart++==StrEnd)           // comprobar si se termina de escribir
la palabra
StrStart=1;

ScrollCount=LCD_SCROLLCOUNT_DEFAULT;    //contador de desplazamientos
UpdateLCD=FALSE;    //si vuelve a ocurrir la interrupcion no ingrese
}

for(uint8_tLCDChar=0;LCDChar<LCD_SEGBUFFER_SIZE;LCDChar++)
*(pLCDREG+LCDChar)=SegBuffer[LCDChar];
}

ISR(TIMER0_OVF_vect)           // Vector interrupción
{
LCD_Init();

    cnt++;           //incremento y si es igual a 2
    if(cnt==2){
LCD_WriteChar(16,(mov-1));    //Escribo en la LCD

    if(mov==5){           //Para solo mover 5 espacios
mov=0;
    }
    if(cnt1==0){
LCD_WriteChar(cnt1,mov);           //Pongo el dibujo en la posicion q
indica cnt1 de la tabla en la primera posicion
LCD_WriteChar(16,(mov-1));
    }
    if(cnt1==1){
LCD_WriteChar(cnt1,mov);    //Pongo el dibujo en la posicion q indica cnt2
de la tabla en la segundo posicion
LCD_WriteChar(16,(mov-1));
    }
}

```

```

cnt1++; //Si controlo q solo se escriban en la lcd los contenidos de los
contadores en la tabla
    if(cnt1==2){
        cnt1=0;
    }
    mov++;          //Incremento variable de movimiento
    cnt=0;         //Pongo en cero cnt
    }
}

// PROGRAMA PRINCIPAL
intmain(void)
{
    cnt1=0;
    cnt2=0;
    cnt=0;
    mov=0;
    DDRD=0B11111111;          // Declaro salida al PORTD
    OCR0A=255;
    sei();                    // Habilito interrupciones
    TCCR0A=0B1000101;        //Normal Port - Normal - Flanco de Bajada
    TIMSK0=0B00000001;      //Habilito Interrupción por desbordamiento
    MCUCR=0B00000010;       //INT0 Flanco de Bajada

for(;;){}                    //Bucle Infinito

```

## 4.4 EJERCICIO

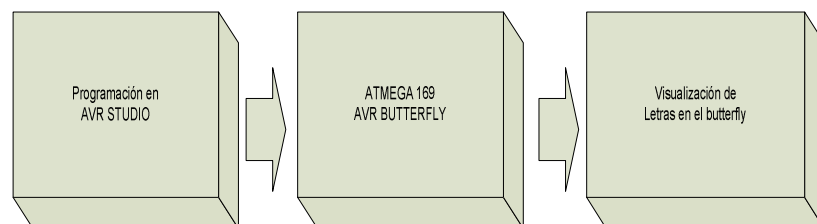
### 4.4.1 EJERCICIO #4.- Desplazamiento de una Frase en la LCD del

**AVR Butterfly haciendo uso de las interrupciones del**

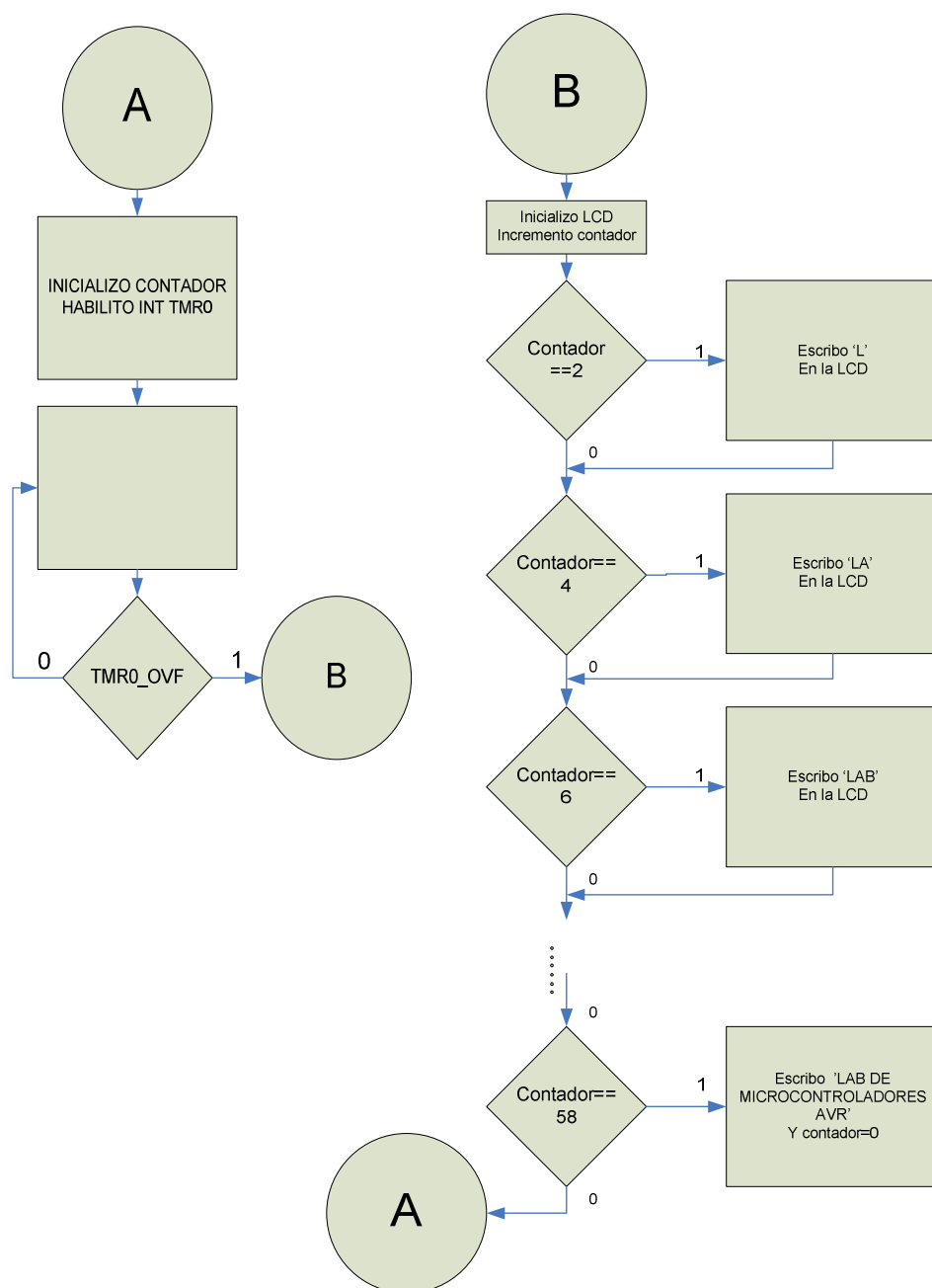
**TIMER0 usando lenguaje assembler.**

Cada vez q se genera una interrupción por el TIMER0 se incrementa un contador la cual nos sirve para visualizar una letra diferente, las letras van a ir presentándose en la LCD de la AVR Butterfly hasta completar la frase que se va desplazando por el delay que le damos por el vector de interrupción del TIMER0.

### 4.4.2 DIAGRAMA DE BLOQUES DEL EJERCICIO N°4

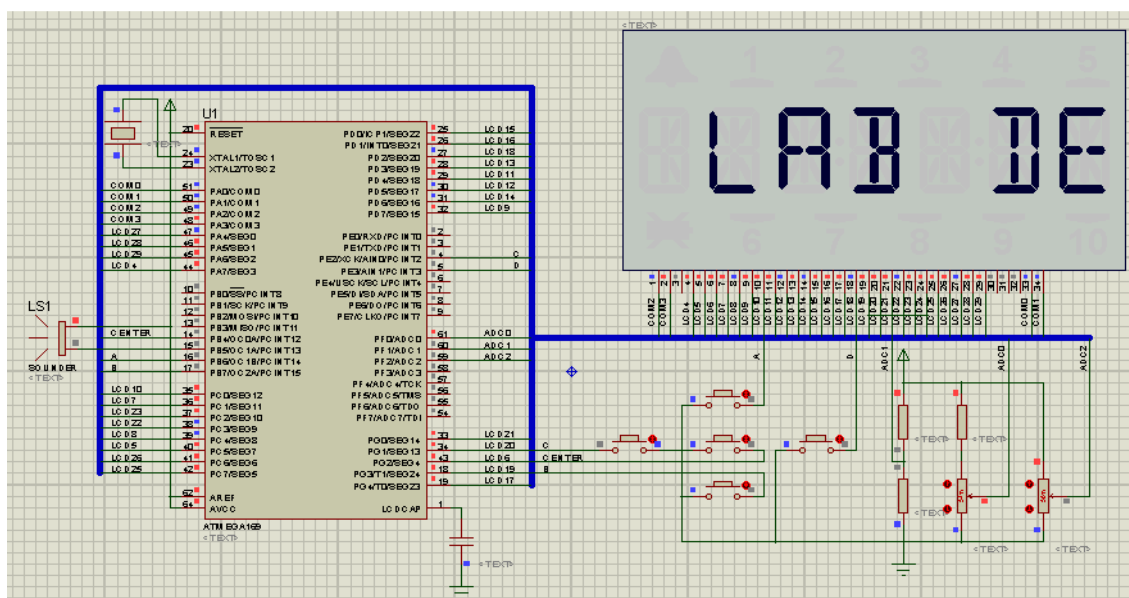


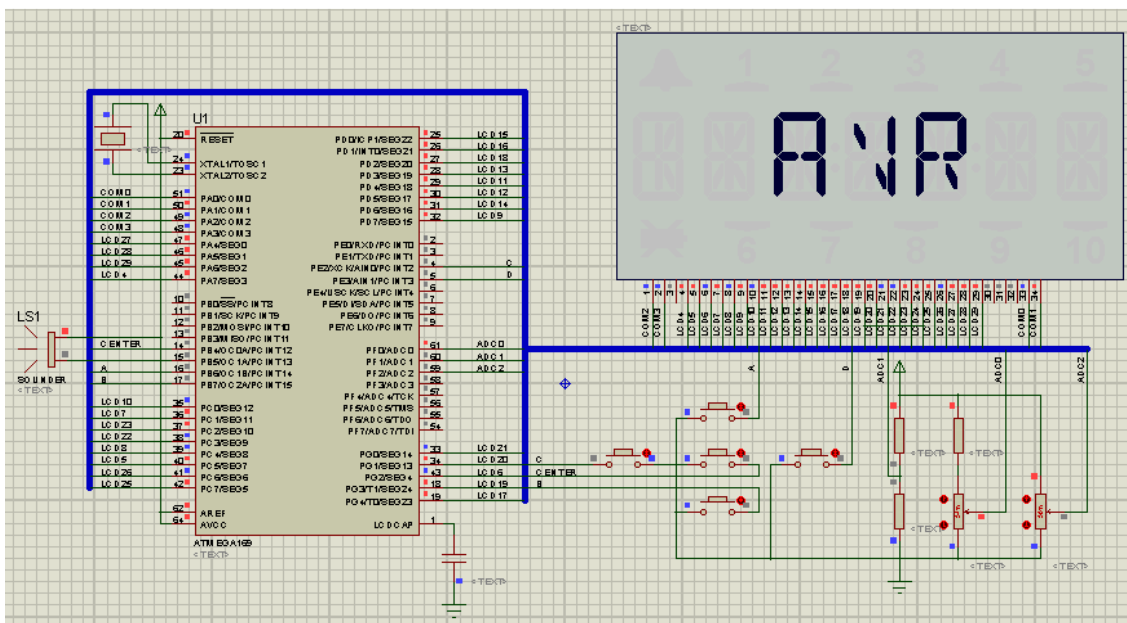
## 4.4.3 DIAGRAMA DE FLUJO DEL EJERCICIO N° 4



Al principio inicializo el contador y habilito el TMR0, cada vez que existe una interrupción se incrementa un contador, la cual usaremos para generar delay, a continuación se inicializa la LCD y empezamos a mostrar letra por letra haciendo que cada una se vaya mostrando en cada porción de la AVR Butterfly, al terminar completamente la palabra el contador usado toma el valor de cero para una nueva presentación.

#### 4.4.4 REPRESENTACION GRAFICA DEL ISIS EJERCICIO N° 4





#### 4.4.5 CODIGO DE SIMULACION EN EL AVR STUDIO DEL EJERCICIO N° 4

```

/*****
*****
**      MICROCONTROLADORES AVANZADOS      **
** Profersor de Catedra:   Ing.CarlosValdiviezo.      **
**                               **
** Integrantes de grupo:   Patricio Salazar Moreira.   **
**                               Luis Ortiz Moran.     **
** Título:Desplazamiento de una animacion en la LCD del AVR butterfly**
** haciendo uso de las interrupciones del TIMER0.      **
** Descripción: Cada vez q se genera una interrupcion por el timer0 se**
** presenta una letra, cada letra esgraficada en un segmento **
** de la LCD que se va desplazando **
** por el delay que le damos al ineterrumpir al TIMER0. **
*****

```

```

*****/

.INCLUDE "M169DEF.INC" ;definimos para usar el BUTTERFLY

;-----:
; Definimos los registros con los que vamos a trabajar ;
;-----;

.SET SPEED = 6 ;habilitamos el scrol ver las pausas de las rutinas
.SET CHR6BUF = 2 ;6 CHAR BUFFER es [R2,R3,R4,R5,R6,R7]

.DEF ZERO = R8
.DEF T1 = R11
.DEF T2 = R12
.DEF A = R16 ;R16 y R31 pueden ser cargados directamente por (LDI)
.DEF AH = R17
.DEF B = R18
.DEF C = R19
.DEF D = R20
.DEF I = R21
.DEF J = R22
.DEF K = R23
.DEF N = R24
.DEF CONT = R25
.ORG $0000
RJMP RESET
.ORG $0016
RJMP OVF_INT_TMR0
;-----;
; Desbordamiento POR TIMER0 ;
;-----;
OVF_INT_TMR0:
DEC CONT

```



```

RETI

;-----;
; Inicializacion ;
;-----;
RESET: CLR ZERO
      LDI CONT,6
LDI A,HIGH(RAMEND) ;habilitamos el puntero de pila para los mas
significativos
OUT SPH,A;guardamos los mas significativos en A
LDI A,LOW(RAMEND) ;habilitamos el puntero de pila para los menos
significativos
OUT SPL,A ;guardamos los menos significativos en A
RCALL LCD_INIT ;Inicializamos LCD
RCALL LCD_CLR ;Limpiamos los segmentos
      LDI A,0b0000_0001 ;Habilitamos las interrupciones del TIMER0 por
desborde
      STS TIMSK0,A;Habilitamos las interrupciones globales
      LDI A,0b0000_0101 ;Habilitamos el prescaler a /1024
      OUT TCCR0A,A;TIMER/COUNTER CONTROL REGISTER "A"
      LDI A,255
      OUT OCR0A,A
      SEI
;-----;
; MAIN PRINCIPAL ;
;-----;
MAIN:
LOOP: LDI YL,LOW(MESSAGE*2) ;Habilitamos el msn en la parte alta
LDI YH,HIGH(MESSAGE*2) ; ;Habilitamos el msn en la parte alta
RCALL SCROLL ;SCROLL MESSAGE
DONE: RJMP LOOP
MESSAGE: .DB " Lab de Microcontroladores AVR ."
;-----;
; SCROLL MESSAGE LOOP ;

```

```

;-----;
SCROLL: MOVW Z,Y;mueve el puntero de la posicion 1 a la 2
PUSH YL                ;guarda estos valores en la parte baja
PUSH YH                ;guarda estos valores en la parte alta
RCALL SHOWBUF
        LDI CONT,2
BUCLE:
        ;   DEC CONT
        SEI
        CPI CONT,0
BRNE BUCLE
POP YH                ;Restar 1ST POINTER
POP YL
ADIW YH:YL,1;Incrementa el punetro
CPI A, '.'            ;detiene el periodo '.'
BRNE SCROLL
RET
;-----;
; Copia el texto al buffer del display ;
;-----;
SHOWBUF:LPM A,Z+
MOV R7,A
LPM A,Z+
MOV R6,A
LPM A,Z+
MOV R5,A
LPM A,Z+
MOV R4,A
LPM A,Z+
MOV R3,A
LPM A,Z+
MOV R2,A
PUSH A
RCALL DISPN

```

```

POP A
RET
;-----;
; DISPN - numeros del DISPLAY en R7:R2      ;
;-----;
DISPN:  LDI  XL,LOW(CHR6BUF) ;POINTS BUFFER-6
LDI  XH,HIGH(CHR6BUF)
;-----;
; ENTER HERE IF XH:XL SET ;
;-----;
LCD_DSP: LDI  N,6;6 CHARS
LDI  B,$F0    ;BITMASK
DSPNXT: LD   A,X+    ;FETCH THE CHAR TO DISP
CPI  A,' '    ;SPACE?
BRNE NOSPC   ;SPACE XLATION
LDI  A,SPACE-LCD_TABLE
NOSPC:
CPIA,'a'     ;CHARACTER XLATION
BRLONOSMLET ;SMALL LETTERS?
SUBI A,$20   ;FOLD#1 a=>A
NOSMLET: CPI A,'A'    ;CAP LETTERS
BRLONOBGLET ;
SUBI A,$37   ;FOLD#2 A=>10
NOBGLET: CPI A,'0'   ;ASCII NUMBERS
BRLO NOANUM ;
SUBI A,$30   ;FOLD#3 "0"=>0
NOANUM: LSL  A;POINT Z INTO TABLE
LDI  ZL,LOW(LCD_TABLE*2)
LDI  ZH,HIGH(LCD_TABLE*2)
ADD  ZL,A;OFFSET INTO
ADC  ZH,ZERO;CHARACTER TABLE
LDI  YL,LOW(LCDDR1)-1 ;(=251)POINTS TO
CLR  YH;LCD SEGMENTS
MOV  A,N;USE COUNTER

```

```

DEC  A
LSR  A;AS OFFSET TO
ADD  YL,A;SEGMENTS
SET
LDI  I,4
DISPLUP: CPI  YL,LOW(LCDDR8) ;PAST CHECK POINT?
BRLO NOZINC ;PAST 2ND READ?
BRTCNOZINC ;SHOULD WE INCZ?
ADIW ZH:ZL,1;INCZ AFTER 2ND READ
CLT;STOP FURTHER INCZ
NOZINC: LPM  A,Z;LOAD SEGMENT DATA
SBRS  I,0;USE BIT0
SWAP  A;SWAP ON EVEN SEGS
SBRC  N,0;USE BIT0
SWAP  A;SWAP ON EVEN DIGITS
POTRIP: AND  A,B;MASK NEEDED INFO
COM   B ;INVERT MASK
LD    C,Y;READ-IN SEGMENT
AND   C,B;CLEAR A SPOT
OR    A,C;SHOVE-IN NEW
ST    Y,A;WRITE-BACK
COM   B ;RE-INVERT MASK
ADIW  YH:YL,5;NEXT SEG
DEC   I
BRNEDISPLUP ;DONE 4 SEGS?
SKPNUM: COM  B;INVERT BIT-MASK
DEC   N;DONE 6 DIGITS?
NOINC: BRNE DSPNXT
RET
;-----;
; Limpiar todos los segmentos de LCD ;
;-----;
LCD_CLR: LDI  YL,LOW(LCDDR0)
CLR   YH

```

```

CLRLUPE: ST  Y+,ZERO
CPI  YL,LCDDR18+1
BRNE CLRLUPE
RET

;-----;
; INITIALIZE LCD DISP REGISTERS ;
;-----;

LCD_INIT: PUSH A
LDI A,0b1011_0111 ;Setea el reloj, DUTY CYCLE AND # PINS
STS LCDCRB, A      ;Habilita todos los segmentos
LDI A,0b0000_0111 ;habilito el FRAME RATE a 32Hz
STS LCDFRR, A      ;Seteo el Prescaler a 32KHz
LDI A,0b0000_1110 ;Seteo el contraste a 3.3 VOLTS
STS LCDCCR, A      ;SET THE CONTRAST
LDI A,0b1100_0000 ;habilito LCD POWER-SAVE WAVE FROM
STS LCDCRA, A      ;habilito THE LCD
POP A
RET

LCD_TABLE:
;  --mpndlegcjfwbk--a <-----> LCD SEGMENTS
.DW 0b0001010101010001 ;ZERO
.DW 0b0010000010000000 ;1
.DW 0b0001111000010001 ;2
.DW 0b0001101100010001 ;3
.DW 0b0000101101010000 ;4
.DW 0b0001101101000001 ;5
.DW 0b0001111101000001 ;6
.DW 0b0000000101010001 ;7
.DW 0b0001111101010001 ;8
.DW 0b0001101101010001 ;9
.DW 0b0000111101010001 ;A
.DW 0b0011100110010001 ;B      -----a-----
.DW 0b0001010001000001 ;C      | \ | / |
.DW 0b0011000110010001 ;D      f h j k b

```

```

.DW 0b0001111001000001 ;E | \ | / |
.DW 0b0000111001000001 ;F --g-- --l--
.DW 0b0001110101000001 ;G | / | \ |
.DW 0b0000111101010000 ;H e p n m c
.DW 0b0010000010000000 ;I | / | \ |
.DW 0b0001010100010000 ;J -----d-----
.DW 0b1000011001001000 ;K
.DW 0b0001010001000000 ;L
.DW 0b0000010101111000 ;M
.DW 0b1000010101110000 ;N
.DW 0b0001010101010001 ;O
.DW 0b0000111001010001 ;P
.DW 0b1001010101010001 ;Q
.DW 0b1000111001010001 ;R
.DW 0b0001101101000001 ;S
.DW 0b0010000010000001 ;T
.DW 0b0001010101010000 ;U
.DW 0b1000000100110000 ;V
.DW 0b1100010101010000 ;W
.DW 0b1100000000101000 ;X
.DW 0b0010000000101000 ;Y
.DW 0b0101000000001001 ;Z
.DW 0b0001010001000001 ;[
.DW 0b1000000000100000 ;\
.DW 0b0001000100010001 ;]
.DW 0b0000000001100000 ;^
.DW 0b0001000000000000 ;_
.DW 0b0000000000001000 ;'
.DW 0b1110101010101000 ;*
.DW 0b0010101010000000 ;+
SPACE:.DW 0 ;(SPACE) ;
.DW 0b0000101000000000 ;-
.DW 0b0100000000000000 ;.
.DW 0b0100000000001000 ;/

```

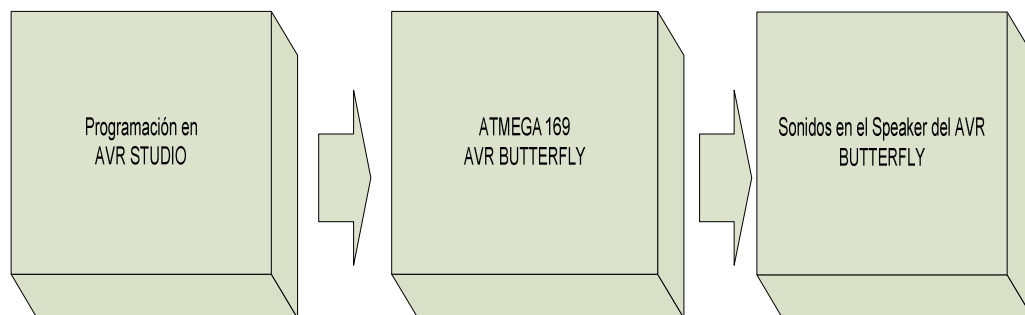
```
.DW 0b1000000000001000 ;<  
.DW 0b0001101000000000 ;=  
.DW 0b0100000000100000 ;>
```

## 4.5 EJERCICIO

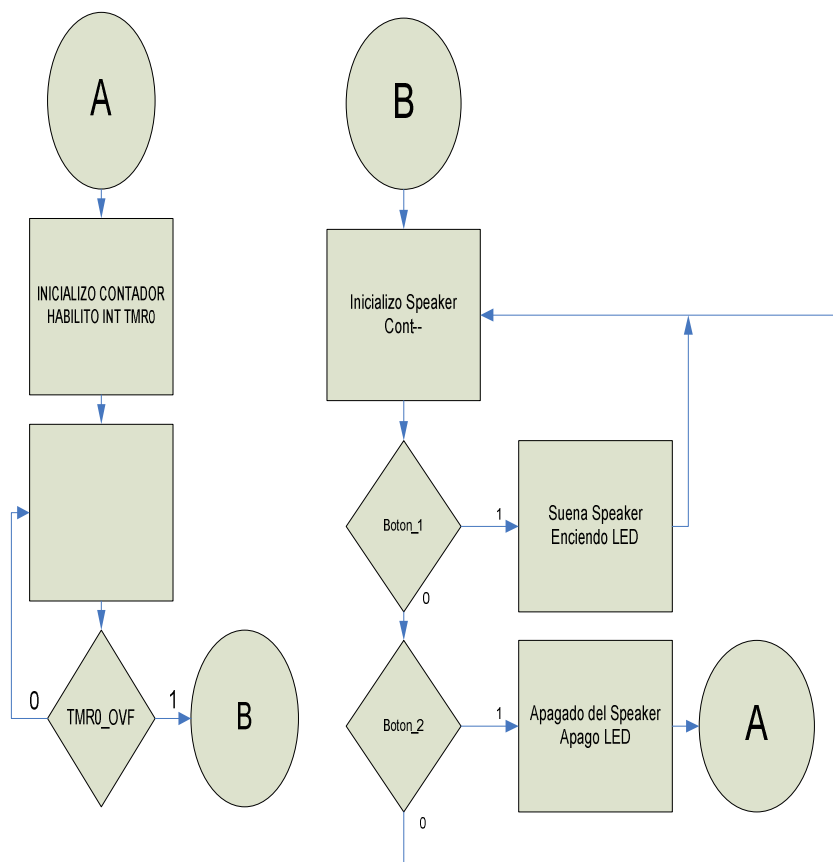
### 4.5.1 EJERCICIO #5.- Como crear un sonidos utilizando el speaker del AVR Butterfly haciendo uso de las interrupciones del TIMER0 para simular el control de alarma de un carro.

Ingresamos un pequeño delay por medio de la interrupción del Timer0 para a continuación mandar una señal al speaker del AVR Butterfly con la cual podemos crear diferentes sonidos el cual sonara por un tiempo predeterminado, pero también podemos interrumpir su tiempo de duración con un segundo pulsador que simula el control de alarma de un carro si volvemos a presionar el primer pulsador se encenderá la alarma simulando que el carro ha sido tocado otra vez.

#### 4.5.2 DIAGRAMA DE BLOQUES EL EJERCICIO N°5



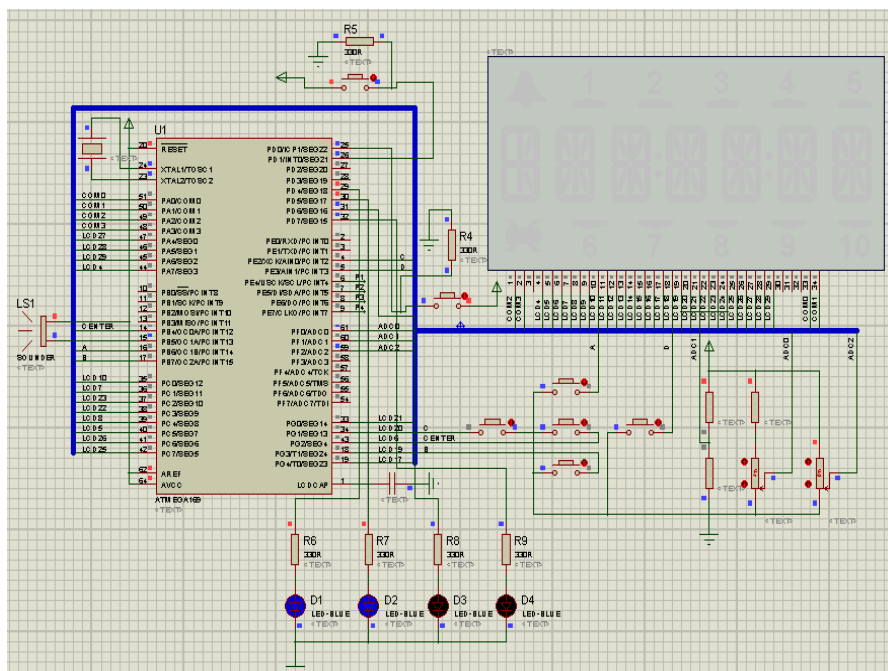
#### 4.5.3 DIAGRAMA DE FLUJO DEL EJERCICIO N° 5





Al principio inicializo el contador y habilito el TMRO, cada vez que existe una interrupción se incrementara un contador, la cual usaremos para generar delay, a continuación se inicializa el SPEAKER si no se presiona nada se mantiene el en mismo estado, pero si lo presionamos el Botton1 suena el SPEAKER(alarma del carro) más las luces del carro se encienden (representado en el ISIS por unos leds) , pero al presiona la segunda botonera, botón-2 se apaga el Speaker(apaga la alarma) y las luces se apagan.

**4.5.4 REPRESENTACION GRAFICA DEL ISIS EJERCICIO N°5.**



## 4.5 CODIGO DE SIMULACION EN EL AVR STUDIO DEL

### EJERCICIO N° 5

```

/*****
*****
**      MICROCONTROLADORES AVANZADOS      **
**  Profesor de Catedra:  Ing.Carlos Valdiviezo.      **
**  Integrantes de grupo:  Patricio Salazar Moreira.      **
**                          Luis Ortiz Moran            **
**  Título:  Desplazamiento de un mensaje en la LCD del AVR butterfly      **
**            haciendo uso de las interrupciones del TIMER0.      **
**  Descripción:  Cada vez q se genera una interrupcion por el timer0 se      **
**                presenta una porcion del mensaje, cada letra ingresada      **
**                se va desplazando por el delay que le damos al ineterrumpir al
**                TIMER0.
*****
*****/

.INCLUDE "M169DEF.INC"
.DEF A      = R16      /*creamos la variable A*/
.DEF CONT   = R17      /* definimos la variable cont */
.DEF PAR    = R19      /*creamos la variable par */

.ORG $0000      /*etapa para habilitar el tmr0*/
    RJMP RESET
.ORG $0016
    RJMP INT_OVF_TMR0

RESET: LDI    A,LOW(RAMEND) /*cargamos la parte menos significativa en el A*/
        OUT    SPL,A
        LDI    A,HIGH(RAMEND)/*cargamos la parte mas significativa en el A*/
        OUT    SPH,A
        SBI    DDRB,5 /*habilitamos el pin 5 del PORTB*/
        SBI    DDRD,7/*habilitamos el pin 7 del PORTD*/
        SBI    DDRD,6 /*habilitamos el pin 6 del PORTD*/
        SBI    DDRD,5 /*habilitamos el pin 5 del PORTD*/
        SBI    DDRD,4 /*habilitamos el pin 4 del PORTD*/
        CBI    DDRD,0 /*habilitamos el pin 0 del PORTD*/
        CBI    DDRD,1 /*habilitamos el pin 1 del PORTD*/

        LDI    A,0b0000_0001/*habilitamos por desbordamiento el registro TIMSK0*/
        STS    TIMSK0,A
        LDI    A,0b0000_0101/*prescalador de 1024 en el registro TCCR0A*/
        OUT    TCCR0A,A
        LDI    A,255      /*cargamos con 255 al timer0*/
        OUT    OCR0A,A
        SEI      /*habilitamos las interrupciones*/
        LDI    CONT,200   /*cargamos con 200 al cont*/

```

```

BUCLE:
    LDI CONT,200          /*cargamos con 200 al cont*/
    LDI A,1              /*cargamos con 1 a A*/
    OUT TCNT0,A         /*cargamos A al timer*/
    IN R18,PIND         /*declaramos como entrada al PIND en R18*/
    SBRS R18,0          /*si es igual a cero salta si es diferente se keda en
bucle*/
    RJMP BUCLE

BUCLE2:
    IN R18,PIND         /*declaramos como entrada al PIND en R18*/
    SBRS R18,0          /*para que salte en bucle 0 o 2*/
    RJMP BUCLE
    RJMP BUCLE2

BUCLE1:
    LDI CONT,200          /*cargamos con 200 al cont*/
    LDI A,1              /*cargamos con 1 a A*/
    OUT TCNT0,A         /*cargamos A al timer*/
    IN R18,PIND         /*declaramos como entrada al PIND en R18*/
    SBRS R18,1          /*si es igual a uno salta si es diferente se keda en
bucle*/
    RJMP BUCLE1

BUCLE3:
    IN R18,PIND         /*declaramos como entrada al PIND en R18*/
    SBRS R18,1          /*para que salte en bucle 1 o 3*/
    RJMP BUCLE1
    RJMP BUCLE3

INT_OVF_TMR0:

CUENTA:
    INC PAR             /*incremento par*/
    SBRS PAR,0         /*pra seleccionar cual de los dos ciclos*/
    RJMP CICLO1
    RJMP CICLO2

CICLO1:
    CBI PORTD,5         /*desabilito PORTD5*/
    CBI PORTD,4         /*desabilito PORTD4*/
    SBI PORTD,7         /*habilito PORTD7*/
    SBI PORTD,6         /*habilito PORTD6*/
    RJMP SAL

CICLO2:
    SBI PORTD,5         /*habilito PORTD5*/
    SBI PORTD,4         /*habilito PORTD4*/
    CBI PORTD,7         /*desabilito PORTD7*/
    CBI PORTD,6         /*desabilito PORTD6*/

SAL:
    IN R18,PIND         /*declaramos como entrada al PIND en R18*/
    SBRS R18,1         /*pra seleccionar cual de los dos lazos*/

```

```

        RJMP SIGUE
        RJMP FIN
SIGUE:
        DEC  CONT          /*decremenmto el cont*/
        BRNE MLUPE        /*salta a mlupe si s igual*/
        RJMP FIN          /*salta a fin*/

HOLD_TONE:                                /*mantiene el tono*/
        RCALL  FREQ
        DEC   R10
        BRNE  HOLD_TONE
        RET

FREQ:                                       /*seteo la salida del speaker*/
        PUSH  A
        SBI   PINB,5
FLUPE: DEC   A
        BRNE  FLUPE
        POP  A
        RET

MLUPE:                                       /*creo diferentes tonos*/
        LDI   A,20
        RCALL HOLD_TONE
        LDI   A,100
        RCALL HOLD_TONE
        RJMP  CUENTA

FIN:                                       /*encero las salidas*/
        CBI   PORTD,5
        CBI   PORTD,4
        CBI   PORTD,7
        CBI   PORTD,6
        RETI

```

## CONCLUSIONES

1. La plataforma de trabajo construida incluyendo el Kit AVR Butterfly es una práctica y poderosa herramienta de aprendizaje, eficaz y sencilla, que con el desarrollo de los proyectos implementados en él, vamos descubriendo progresivamente las características del microcontrolador atmega169 que son fáciles de manipular. Ya que ensamblada en el kit AVR Butterfly lo hace mucho más sencillo y didáctico.
2. El microcontrolador ATmega169 de ATMEL es un dispositivo programable de excelente desempeño y de poca complejidad que combinado con el software de Desarrollo Integrado AVR Studio 4, es una herramienta poderosa que permite simular, emular y depurar un proyecto antes de intentar probarlo con el hardware, permitiéndonos así encontrar rápidamente un posible error, ahorrando tiempo y recursos.
3. El Timer/counter0 opera de dos modos: ya sea como contador o como temporizador. El modo de funcionamiento del mismo está determinado por el tipo de reloj seleccionado en el bloque de clock Select en donde determinamos por los bit CS0-2 ya sea como reloj interno actuando como temporizador y solo dos opciones como reloj externo, tanto en ascendente como en descendente para actuar como contador.

## RECOMENDACIONES

1. Reemplazar la batería que incluye el fabricante en el Kit AVR Butterfly, ya que al principio tuvimos confusión por la batería, y la reemplazamos por una fuente externa de voltaje de 3V. Ya que esto permitirá desarrollar fiablemente las aplicaciones, la cual evitara los errores que por batería descargada se podrían dar y confundir al usuario.
2. No apoyar el Kit AVR Butterfly en superficies conductivas tales como metal, líquidos, etc. puesto que podrían causar corto y por ende daños en el mismo y se debe tener mucha precaución al soldar sus pines de no intentar mucha pasta de soldar, ya que podría estropear la conducción y perder parte de la visualización en el display.
3. Debemos tener mucho precaución cuando programamos tanto en lenguaje C como en lenguaje Assembler ya que utilizamos otras instrucciones y al Simular los dispositivos y depurar el código fuente simultáneamente en AVR Studio, en donde evitaremos dañar el dispositivo por errores de codificación del software.
4. Es necesario revisar las hojas de especificaciones del kit AVR Butterfly para evitar averiarlo o des configurarlo, así como también es importante no conectar cables directamente en los espacios para conexiones externas del Kit, ya que podrían causar cortocircuito en su lugar, colocar Headers fijos o un bus de dato como lo hemos colocado nosotros.

## BIBLIOGRAFÍA

[1] Datasheet Atmega169, Introducción al capítulo y descripción general del TIMER0,

<http://es.scribd.com/doc/7842963/Capitulo6-Timer0-del-AVR-del-ATmega32-espanol>, fecha de consulta: 21/11/12.

[2] Daniels Jhons, Características del TIMER0 en la familia MICROCHIP,

<http://perso.wanadoo.es/pictob/micropic16f84.htm>, fecha de consulta: 24/11/12.

[3] Lab. Centers Electronicis, PROTEUS ISIS,

<http://es.scribd.com/doc/25217993/Manual-basico-PROTEUS-ISIS>, fecha de consulta: 29/11/12.

[4] Atmel Corporation, Avr Butterfly,

[http://gandalf.arubi.uni-kl.de/avr\\_projects/AVR\\_Butterfly\\_Introduction.pdf](http://gandalf.arubi.uni-kl.de/avr_projects/AVR_Butterfly_Introduction.pdf), fecha de consulta: 03/12/12.

[5] Atmel Corporation , AVR STUDIO,

<http://www.microdigitaled.com/AVR/Software/AVRstudioTutorial.pdf>, fecha de consulta: 24/11/12.